Geist

A multimodal data transformation, query, and reporting language

Meng Li, Timothy McPhillips, Bertram Ludäscher

Table of contents

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, RDFLib, and Clingo.

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

Install Geist with the backend(s) you need:

```
pip install geist-p[duckdb]    # DuckDB module
pip install geist-p[rdflib]    # RDFLib module
pip install geist-p[clingo]    # Clingo module
pip install geist-p[all]       # All backends
```

You can also combine multiple backends:

```
pip install geist-p[duckdb,rdflib]
```

To install the development version:

```
pip install "geist-p[duckdb] @ git+https://github.com/CIRSS/geist-p.git@develop"
pip install "geist-p[rdflib] @ git+https://github.com/CIRSS/geist-p.git@develop"
pip install "geist-p[clingo] @ git+https://github.com/CIRSS/geist-p.git@develop"
pip install "geist-p[all] @ git+https://github.com/CIRSS/geist-p.git@develop"
```

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
- `{{ ... }}` for Expressions to print to the template output
- `{# ... #}` 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
- load
- query
- 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
- **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
- **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?

CLI

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
                                template to expand  [required]
-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
--help                         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
    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" %}

END_TEMPLATE
```

Example 2: expand a report from a file

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

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
    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" %}
```

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:

```
{% 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> "{{ sentence }}" .
    <http://example.com/drewp> <http://example.com/feels> "{{ feeling }}" .
{% 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" %}
```

Python API

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

Parameters description for *report()*:

Name	Type	Description	Default
inputfile	string	A report to be expanded	REQUIRED
isinputpath	bool	`True` if the inputfile is the file path, otherwise the inputfile is the content	`False`
outputroot	string	Path of the directory to store the expanded report	current directory, i.e., `./`
suppressoutput	bool	`True` to suppress output	`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	`{}`

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
    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" %}

"""

# 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
    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" %}
```

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:

```
{% 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> "{{ sentence }}" .
    <http://example.com/drewp> <http://example.com/feels> "{{ feeling }}" .
{% 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" %}
```

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. `clingo`, `duckdb`, and `rdflib` are available. (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.
`predicate`	`'isfirstcol'` for using the first column as the predicate name; strings other than `'isfirstcol'` are used as the predicate name directly (by default, `'isfirstcol'`). Available for `clingo` data backend only.
`name`	Name of the program (by default, `'base'`). Available for `clingo` 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" .
{% endcreate %}
```

Example 2: the given string is a file path

Here is the `test.nt` 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:

```
{% create "test", datastore="rdflib", inputformat="nt", isfilepath=True %} test.nt {% 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. `clingo`, `duckdb`, and `rdflib` are available. (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 dataset file (e.g., `.geistdata/rdflib/test.pkl`) 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 the `quiet` 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`)

Example: visualize the `test` dataset

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

3.2.4 Tag graph2

The `graph2` tag returns a DOT string of the visualization of a dataset. 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

Example: visualize the `test` dataset

```
{% graph2 dataset="test" %}
```

3.2.5 Tag load

The `load` tag imports data into a dataset. Here are parameters of the `load` tag:

Name	Description
`dataset`	Name of RDF dataset to be removed (default `kb`)
`datastore`	Data backend. `clingo`, `duckdb`, and `rdflib` are available. (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.
`predicate`	`'isfirstcol'` for using the first column as the predicate name; strings other than `'isfirstcol'` are used as the predicate name directly (default `'isfirstcol'`). Available for `clingo` data backend only.
`programname`	Name of the program (default `'base'`). Available for `clingo` data backend only.

Example 1: 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. `clingo`, `duckdb`, and `rdflib` are available. (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)
`predicate`	Name of the predicate to be queried (default `None`). Available for `clingo` data backend only.
`programname`	Name of the program (default `'base'`). Available for `clingo` data backend only.
`rformat`	Format of the returned data, i.e., `'lp'`, `'df'`, or `'dict'` (default `'lp'`). Available for `clingo` data backend only.

Example 1: the given string is not a file path

```
{% query "test", datastore="rdflib", isfilepath=False %}
    SELECT ?s ?p ?o
    WHERE {
        ?s ?p ?o
    }
    ORDER BY ?s ?p ?o
{% endquery %}
```

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 `isfilepath` field to False. A Pandas DataFrame will be returned. Here are parameters of the `map` 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 `html` tag formats and saves the string as a HTML file. Here is a parameter of the `html` tag:

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

Save the `Hello World!` string as a file

```
{% html %}Hello World!{% endhtml %}
```

Expected content of the `report.html` file:

```
<!DOCTYPE html>
<html>
Hello World!
</html>
```

3.2.10 Tag img

The `img` tag renders the Mermaid string 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. Note: `mermaid`, `mmd`, `dot` or `gv` will show code directly. (Known issue: the image is rendered by mermaid.ink for now. Due to the limitation of URL length, you may get the `"414 Request-URI Too Large" errors` message. We recommend select `mermaid` or `mmd` instead.
...	Attributes of the HTML or the HTML tag

Example 1: render as svg

```
{% img src="test.svg", width="50%" %}
    graph TB
        node1 --> node2
{% endimg %}
```

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

```
<img src="test.svg" width="50%" >
```

Example 2: using the mermaid string directly

```
{% img src="test.mermaid" %}
    graph TB
        node1 --> node2
{% endimg %}
```

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

```
<pre class="mermaid">
    graph TB
        node1 --> node2
</pre>
<script type="module">
    import mermaid from "https://cdn.jsdelivr.net/npm/mermaid@11/dist/mermaid.esm.min.mjs";
    mermaid.initialize({{ startOnLoad: true }});
</script>
```

Example 3: show gv directly

```
{% img src="test.gv" %}digraph test_graph { node1 -> node2 }{% endimg %}
```

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

```
<pre><code>digraph test_graph { node1 -> node2 }</code></pre>
```

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`)

Example: embed query results as a table

```
{% table %}
    {%- query isfilepath=False as query_results %}
        SELECT ?s ?p ?o
        WHERE {
            ?s ?p ?o
        }
        ORDER BY ?s ?p ?o
    {% endquery %}
    {{ query_results | df2json }}
{% endtable %}
```

It can also be done with the *df2htmltable* filter:

```
{%- query isfilepath=False as query_results %}
    SELECT ?s ?p ?o
    WHERE {
        ?s ?p ?o
    }
    ORDER BY ?s ?p ?o
{% endquery %}
{{ query_results | df2htmltable }}
```

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. Note that "templates.geist" is interpreted as a path relative to the input file's directory when `isinputpath=True`.
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`)

```
{% use "templates.geist" %}

{%- create inputformat="nt", isfilepath=False %}
    <http://example.com/test1> <http://example.com/p1> "Hello World".
    <http://example.com/test2> <http://example.com/p2> "What a Nice Day".
{% endcreate %}

{%- query "kb1", isfilepath=False as res %}
    SELECT ?s ?o
    WHERE {
        ?s ?p ?o
    }
    ORDER BY ?s ?o
{% endquery %}
{% set all_triples = res | json2df %}

{% for _, row in all_triples.iterrows() %}
    {% format_output row["s"], row["o"] %}.
{%- endfor %}

{%- destroy %}
```

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   Path of the file containing the report
                                template to expand  [required]
-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
--help                         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
    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" %}

END_TEMPLATE
```

Example 2: expand a report from a file

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

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
    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" %}
```

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:

```
{% 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> "{{ sentence }}" .
    <http://example.com/drewp> <http://example.com/feels> "{{ feeling }}" .
{% 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" %}
```

4.2 Command create

The *create* command has three subcommands, all 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.

Commands:
clingo  Create a new ASP dataset using Clingo
duckdb  Create a new SQL dataset using DuckDB
rdflib  Create a new RDF dataset using RDFLib
```

geist create clingo [OPTIONS]

```
Usage: geist create clingo [OPTIONS]

Create a new ASP dataset using Clingo

Options:
-d, --dataset TEXT              Name of ASP dataset to create (default "kb")
-ifile, --inputfile FILENAME    Path of the file to be loaded as facts,
                                rules, and contraints  [required]
-iformat, --inputformat [lp|csv|json]
                                Format of the file to be loaded as facts,
                                rules, and constraints. Note that "csv" only
                                supports facts (default "lp"). If multiple
                                possibilities are provided (as a list), only
                                the first one will be considered.
-pred, --predicate TEXT         "isfirstcol" for using the first column as
                                the predicate name; strings other than
                                "isfirstcol" are used as the predicate name
                                directly (default: "isfirstcol")
-prog, --programname TEXT       Name of the program (default: "base")
--help                          Show this message and exit.
```

Example 1: create a `test` ASP dataset from stdin with LP format

```
geist create clingo --dataset test --inputformat lp << __END_INPUT__
friends(a, b).
friends(a, c).
__END_INPUT__
```

Example 2: create a `test` ASP dataset from a file with LP format

Here is the `friends.lp` file:

```
friends(a, b).
friends(a, c).
```

Code:

```
geist create clingo --dataset test --inputfile friends.lp --inputformat lp
```

Example 3: create a `test` ASP dataset from a CSV file

Here is the `friends.csv` file:

```
arg1,arg2
a,b
a,c
```

Code:

```
geist create clingo --dataset test --inputfile friends.csv --inputformat csv --predicate friends
```

geist create duckdb [OPTIONS]

```
Usage: geist create duckdb [OPTIONS]

Create a new SQL dataset using DuckDB

Options:
-d, --dataset TEXT              Name of SQL dataset to create (default "kb")
-ifile, --inputfile FILENAME    Path of the file to be loaded as a Pandas
                                DataFrame  [required]
-iformat, --inputformat [csv|json]
                                Format of the file to be loaded as a Pandas
                                DataFrame (default csv)
-t, --table TEXT                Name of the table to be created (default
                                "df")
--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)
--colnames TEXT                 Column names of triples with the format of
                                [[subject1, predicate1, object1], [subject2,
                                predicate2, object2], ...] when the input
                                format is csv
--infer [none|rdfs|owl|rdfs_owl]
                                Inference to perform on update [none, rdfs,
                                owl, rdfs_owl] (default "none")
--help                          Show this message and exit.
```

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

```
geist create rdflib --dataset test --inputformat nt --infer none << __END_INPUT__

<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:

```
<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:

```
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 three subcommands for *destroy*:

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

  Delete a dataset

Options:
  --help  Show this message and exit.

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

geist destroy clingo [OPTIONS]

```
Usage: geist destroy clingo [OPTIONS]

Delete an ASP dataset

Options:
-d, --dataset TEXT  Name of an ASP 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` ASP dataset

```
geist destroy clingo --dataset test
```

The `.geistdata/clingo/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 clingo --dataset test --quiet
```

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 three subcommands for *export*:

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

  Export a dataset

Options:
  --help  Show this message and exit.

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

geist export clingo [OPTIONS]

```
Usage: geist export clingo [OPTIONS]

Export an ASP dataset

Options:
-d, --dataset TEXT              Name of an ASP dataset to be exported
                                (default "kb")
-oroot, --outputroot TEXT       Path of the directory to store the exported
                                data (default: current directory). If the
                                given path (i.e., --outputfile) is None or a
                                relative path, it will be ignored.
-ofile, --outputfile TEXT       Path of the file to store the exported data
                                (default: None). This file can be reused to
                                create a dataset by setting
                                inputformat=json.
-rformat, --returnformat [lp|df|dict]
                                Format of the returned data in memory
                                (default lp)
-pred, --predicate TEXT         Name of the predicate to be exported
                                (default "predicate")
--help                          Show this message and exit.
```

Example: export the `test` ASP dataset

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

```
geist export clingo --dataset test
```

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")
-oroot, --outputroot TEXT       Path of the directory to store the exported
                                table (default: current directory). If the
                                given path (i.e., --outputfile) is None or a
                                relative path, it will be ignored.
-ofile, --outputfile TEXT       Path of the file to store the exported table
                                (default: None)
-oformat, --outputformat [csv|json]
                                Format of the exported table (default csv)
-t, --table TEXT                Name of the table to be exported (default
                                "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]

```
Usage: geist export rdflib [OPTIONS]

Export an RDF dataset

Options:
-d, --dataset TEXT              Name of RDF dataset to be exported (default
                                "kb")
-oroot, --outputroot TEXT       Path of the directory to store these
                                exported triples (default: current
                                directory). If the given path (i.e.,
                                --outputfile) is None or a relative path, it
                                will be ignored.
-ofile, --outputfile TEXT       Path of the file to store these exported
                                triples (default: None)
-oformat, --outputformat [json-ld|n3|nquads|nt|hext|pretty-xml|trig|trix|turtle|longturtle|xml]
                                Format of the exported triples (default nt)
--help                          Show this message and exit.
```

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.
```

geist graph rdflib [OPTIONS]

```
Usage: geist graph rdflib [OPTIONS]

Visualize an RDF dataset

Options:
-d, --dataset TEXT              Name of RDF dataset to be visualized
                                (default "kb")
-r, --rankdir [TB|BT|LR|RL]     Direction of the graph (default TB): TB or
                                BT or LR or RL
-m, --mappings TEXT             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, --on TEXT                  Column(s) to be mapped.
-sc, --samecolor                Use the same color for same edges.
-oroot, --outputroot TEXT       Path of the directory to store the graph
                                (default: current directory). If the given
                                path (i.e., --outputfile) is a relative
                                path, it will be ignored.
-ofile, --outputfile TEXT       Path of the file without extension to store
                                the graph (default: res)
-oformat, --outputformat [none|svg|png|gv]
                                Format of the graph (default: none): none or
                                svg or png or gv
--help                          Show this message and exit.
```

Example: visualize the `test` dataset

```
geist graph rdflib --dataset test --outputformat svg
```

4.6 Command load

load command can import data into an existing dataset.

There are three subcommands for *load*:

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

Import data into a dataset

Options:
  --help  Show this message and exit.

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

geist load clingo [OPTIONS]

```
Usage: geist load clingo [OPTIONS]

Import data into an ASP dataset

Options:
-d, --dataset TEXT              Name of an ASP dataset to load a file
                                (default "kb")
-ifile, --inputfile FILENAME    Path of the file to be loaded  [required]
-iformat, --inputformat [lp|csv|json]
                                Format of the file to be loaded (default
                                "lp"). If multiple possibilities are
                                provided (as a list), only the first one
                                will be considered.
-pred, --predicate TEXT         Name of the predicate to be loaded (default:
                                "isfirstcol")
-prog, --programname TEXT       Name of the program (default: "base")
--help                          Show this message and exit.
```

Example: load a file into the `test` ASP dataset

```
geist load clingo --dataset test --inputfile new_friends.lp --inputformat lp
```

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    Path of the file to be loaded as a table
                                [required]
-iformat, --inputformat [csv|json]
                                Format of the file to be loaded as a table
                                (default csv)
-t, --table TEXT                Name of the table to be created  [required]
--help                          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
```

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    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)
--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 three subcommands for *query*:

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

  Perform a query on a dataset

Options:
  --help  Show this message and exit.

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

geist query clingo [OPTIONS]

```
Usage: geist query clingo [OPTIONS]

Perform an ASP query on a dataset

Options:
-d, --dataset TEXT              Name of ASP dataset to be queried (default
                                "kb")
-ifile, --inputfile FILENAME    Specify either the path of the file
                                containing the ASP query to execute or
                                provide the ASP 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       Path of the file to store the query results
                                (default: None). This file can be reused to
                                create a dataset by setting
                                inputformat=json.
-rformat, --returnformat [lp|df|dict]
                                Format of the returned data in memory
                                (default lp)
-pred, --predicate TEXT         Name of the predicate to be queried
-prog, --programname TEXT       Name of the program
--help                          Show this message and exit.
```

Example 1: query the `test` ASP dataset from stdin

```
geist query clingo --dataset test << __END_QUERY__

friends(X, Z) :- friends(X, Y), friends(Y, Z), X < Z.
friends(Y, Z) :- friends(X, Y), friends(X, Z), Y < Z.

__END_QUERY__
```

Example 2: query the `test` ASP dataset from a query file

```
geist query clingo --dataset test --inputfile query
```

Here is the query file's content:

```
friends(X, Z) :- friends(X, Y), friends(Y, Z), X < Z.
friends(Y, Z) :- friends(X, Y), friends(X, Z), Y < Z.
```

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  Specify either the path of the file containing
                                the SQL query to execute or provide the SQL
                                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     Path of the file to store the query results
                                (default: None)
--help                        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
```

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 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     Path of the file to store the query results
                                (default: None)
--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:

```
SELECT ?s ?p ?o
WHERE {
    ?s ?p ?o
}
ORDER BY ?s ?p ?o
```

5. Python API

5.1 Class Connection

Connection class can interact with a dataset with `create`, `close`, `destroy`, `export`, `graph`, `load`, and `query` methods.

5.1.1 What is a *Connection* class?

A *Connection* class has three attributes:

Name	Type	Description	Default
datastore	string	A backend datastore, i.e., `'duckdb'`, `'rdflib'`, or `'clingo'`	REQUIRED
dataset	string	Name of the dataset. Note that `':memory:'` is a reserved value for datasets that exist only in memory	REQUIRED
conn	object	A `DuckPyConnection` object, a `GeistGraph` object, or a `Control` object	`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 DuckDB dataset named test
connection = geist.Connection.connect(datastore='duckdb', dataset='test')
```

```
# create a Connection object to an existing Clingo dataset named test
connection = geist.Connection.connect(datastore='clingo', 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

create function

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

Parameters description for *create()*:

Name	Type	Description	Default
datastore	string	A backend datastore, i.e., `'clingo'`, `'duckdb'`, or `'rdflib'`	REQUIRED
dataset	string	Name of the dataset to be created. Note that `':memory:'` is a reserved value for datasets that exist only in memory	REQUIRED
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

Description for the *config* parameter:

datastore: clingo

Name	Type	Description	Default
predicate	string	`'isfirstcol'` for using the first column as the predicate name; strings other than `'isfirstcol'` are used as the predicate name directly	`'isfirstcol'`
programname	string	Name of the program	`'base'`

Example 1: create a `test` ASP dataset on disk from a string

The `.geistdata/clingo/test.pkl` file is created and a `Control` object is returned.

```
import geist

lp_str = """
friends(a, b).
friends(a, c).
"""

# Create a Control object
conn = geist.create(datastore='clingo', dataset='test', inputfile=lp_str, inputformat="lp", isinputpath=False)
```

Example 2: create a `test` ASP dataset on disk from a file

The `.geistdata/clingo/test.pkl` file is created and a `Control` object is returned.

Here is the `friends.lp` file:

```
friends(a, b).
friends(a, c).
```

Code:

```
import geist

# Create a Control object
conn = geist.create(datastore='clingo', dataset='test', inputfile="friends.lp", inputformat="lp", isinputpath=True)
```

Example 3: create an ASP dataset in memory from a string

A `Control` object is returned.

```
import geist

lp_str = """
friends(a, b).
friends(a, c).
"""

# Create a Control object
conn = geist.create(datastore='clingo', dataset=':memory:', inputfile=lp_str, inputformat="lp", isinputpath=False)
```

datastore: duckdb

Name	Type	Description	Default
table	string	Name of the table to be created	`'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.

```
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 `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"})
```

datastore: rdflib

Name	Type	Description	Default
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'`	`'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://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='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.

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

Approach 2: use the create method of the Connection class

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 initialzing the *Connection* class.

Parameters description for *create* method of the *Connection* class:

Name	Type	Description	Default
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

Description for the *config* parameter:

datastore: duckdb

Key	Type	Description	Default
table	string	Name of the table to be created	`'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"})
```

datastore: rdflib

Key	Type	Description	Default
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'`	`'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://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
<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=':memory:')
connection.create(inputfile=csv_str, inputformat="csv", isinputpath=False, config={"colnames": "[['subject', 'predicate', 'object']]"})
```

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

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://example.com/drewp>,<http://example.com/says>,"Hello World"
```

Code:

```
import geist

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

datastore: clingo

Key	Type	Description	Default
predicate	string	`'isfirstcol'` for using the first column as the predicate name; strings other than `'isfirstcol'` are used as the predicate name directly	`'isfirstcol'`
programname	string	Name of the program	`'base'`

Example 1: create a `test` ASP dataset from a string

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

```
import geist

lp_str = """
friends(a, b).
friends(a, c).
"""

# Create a Connection instance
connection = geist.Connection(datastore='clingo', dataset='test')
connection.create(inputfile=lp_str, inputformat="lp", isinputpath=False)
```

Example 2: create a `test` ASP dataset from a file

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

Here is the `friends.lp` file:

```
friends(a, b).
friends(a, c).
```

Code:

```
import geist

# Create a Connection instance
connection = geist.Connection(datastore='clingo', dataset='test')
connection.create(inputfile="friends.lp", inputformat="lp", isinputpath=True)
```

Example 3: create an ASP dataset in memory from a string

A `Connection` instance is returned.

```
import geist

lp_str = """
friends(a, b).
friends(a, c).
"""

# Create a Connection instance
connection = geist.Connection(datastore='clingo', dataset=':memory:')
connection.create(inputfile=lp_str, inputformat="lp", isinputpath=False)
```

```
import geist

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

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

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

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* class.

Parameters description for *export* method of the *Connection* class:

Name	Type	Description	Default
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

Description for the *config* parameter:

datastore: duckdb

Key	Type	Description	Default
outputroot	string	Path of the directory to store the exported table	`'./'`
outputfile	string	Path of the file to store the exported table	`None`
outputformat	string	Format of the exported table, i.e., `'csv'` or `'json'`	`'csv'`
table	string	Name of the table to be exported	`'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`.

```
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 geist

# 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'})
```

datastore: rdflib

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

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

There exist a file with the path of `.geistdata/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)
```

datastore: clingo

Key	Type	Description	Default
returnformat	string	Format of the returned data, i.e., `'lp'`, `'df'`, or `'dict'`	`'lp'`
predicate	string	Name of the predicate to be exported	`None`

Example 1: export all facts of the `test` ASP dataset on disk

There exist a file with the path of `.geistdata/clingo/test.pkl`. The following code returns data.

```
import geist

# Create a Connection instance
connection = geist.Connection.connect(datastore='clingo', dataset='test')
# Export all facts of the test ASP dataset
data = connection.export(hasoutput=False)
```

Example 2: export all facts of the `test` ASP dataset in memory

Suppose `conn` is a `Control` object pointing to an ASP dataset in memory. The following code returns data.

```
import geist

# Create a Connection instance
connection = geist.Connection(datastore='clingo', dataset=':memory:', conn=conn)
# Export all facts of the test ASP 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	Type	Description	Default
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

Description for the *config* parameter:

datastore: rdflib

Name	Type	Description	Default
rankdir	string	Direction of the graph: `'TB'` or `'BT'` or `'LR'` or `'RL'`	`'TB'`
mappings	string	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	`None`
on	string	Column(s) to be mapped	`None`
samecolor	bool	`True` to use the same color for same edges	`True`
outputroot	string	Path of the directory to store the graph	`'./'`
outputfile	string	Path of the file without extension to store the graph	`'res'`
outputformats	list	Format of the graph: `'none'` or `'svg'` or `'png'` or `'gv'`	`['none']`

Example 1: visualize the `test` dataset on disk

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

```
import geist

# Create a Connection instance
connection = geist.Connection.connect(datastore='duckdb', dataset='test')
# 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
connection = geist.Connection(datastore='duckdb', dataset=':memory:', conn=conn)
# 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 `inmemory` 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	Type	Description	Default
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

Description for the *config* parameter:

datastore: duckdb

Name	Type	Description	Default
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 `.geistdata/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.

```
import geist

csv_str = """
v1,v2,v3
1,1,1
2,2,2
3,3,3
"""

# Create a Connection instance
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'})
```

datastore: rdflib

Name	Type	Description	Default
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 `.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']]"})
```

datastore: clingo

Name	Type	Description	Default
predicate	string	`'isfirstcol'` for using the first column as the predicate name; strings other than `'isfirstcol'` are used as the predicate name directly	`'isfirstcol'`
programname	string	Name of the program	`'base'`

Example: load facts into the `test` ASP dataset

There exist a file with the path of `.geistdata/clingo/test.pkl`. The `lp_str` will be imported into the `test` ASP dataset.

```
import geist

lp_str = """
friends(b, d).
"""

# Create a Connection instance
connection = geist.Connection.connect(datastore='clingo', dataset='test')
# Load lp_str into the test ASP dataset
connection.load(inputfile=lp_str, inputformat='lp', isinputpath=False)
```

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	Type	Description	Default
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

Description for the *config* parameter:

Name	Type	Description	Default
outputroot	string	Path of the directory to store the query results	`'./'`
outputfile	string	Path of the file to store the query results	`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 `.geistdata/duckdb/test.duckdb`. The following code returns a Pandas data frame named `res` with query results.

```
import geist

# Create a Connection instance
connection = 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
connection = geist.Connection.connect(datastore='duckdb', dataset='test')
# 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
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
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)
```

5.2 Functions

5.2.1 Function report

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

Parameters description for *report()*:

Name	Type	Description	Default
inputfile	string	A report to be expanded	REQUIRED
isinputpath	bool	`True` if the inputfile is the file path, otherwise the inputfile is the content	`False`
outputroot	string	Path of the directory to store the expanded report	current directory, i.e., `./`
suppressoutput	bool	`True` to suppress output	`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	`{}`

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
    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" %}

"""

# 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
    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" %}
```

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:

```
{% 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> "{{ sentence }}" .
    <http://example.com/drewp> <http://example.com/feels> "{{ feeling }}" .
{% 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	Type	Description	Default
datastore	string	A backend datastore, i.e., `'clingo'`, `'duckdb'`, or `'rdflib'`	REQUIRED
dataset	string	Name of the dataset to be created. Note that `':memory:'` is a reserved value for datasets that exist only in memory	REQUIRED
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

Description for the *config* parameter:

datastore: clingo

Name	Type	Description	Default
predicate	string	`'isfirstcol'` for using the first column as the predicate name; strings other than `'isfirstcol'` are used as the predicate name directly	`'isfirstcol'`
programname	string	Name of the program	`'base'`

Example 1: create a `test` ASP dataset on disk from a string

The `.geistdata/clingo/test.pkl` file is created and a `Control` object is returned.

```
import geist

lp_str = """
friends(a, b).
friends(a, c).
"""

# Create a Control object
conn = geist.create(datastore='clingo', dataset='test', inputfile=lp_str, inputformat="lp", isinputpath=False)
```

Example 2: create a `test` ASP dataset on disk from a file

The `.geistdata/clingo/test.pkl` file is created and a `Control` object is returned.

Here is the `friends.lp` file:

```
friends(a, b).
friends(a, c).
```

Code:

```
import geist

# Create a Control object
conn = geist.create(datastore='clingo', dataset='test', inputfile="friends.lp", inputformat="lp", isinputpath=True)
```

Example 3: create an ASP dataset in memory from a string

A `Control` object is returned.

```
import geist

lp_str = """
friends(a, b).
friends(a, c).
"""

# Create a Control object
conn = geist.create(datastore='clingo', dataset=':memory:', inputfile=lp_str, inputformat="lp", isinputpath=False)
```

datastore: duckdb

Name	Type	Description	Default
table	string	Name of the table to be created	`'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.

```
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 `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"})
```

datastore: rdflib

Name	Type	Description	Default
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'`	`'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://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='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.

```
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=':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
<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
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	Type	Description	Default
datastore	string	A backend datastore, i.e., `'clingo'`, `'duckdb'`, or `'rdflib'`	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	`False`

Example: delete the `test` dataset

```
import geist
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	Type	Description	Default
datastore	string	A backend datastore, i.e., `'clingo'`, `'duckdb'`, or `'rdflib'`	REQUIRED
dataset	string OR `Control` object 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 `Control` object, 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

Description for the *config* parameter:

datastore: clingo

Name	Type	Description	Default
returnformat	string	Format of the returned data, i.e., `'lp'`, `'df'`, or `'dict'`	`'lp'`
predicate	string	Name of the predicate to be exported	`None`

Example 1: export all facts of the `test` ASP dataset on disk

There exist a file with the path of `.geistdata/clingo/test.pkl`. The following code returns data and a `Control` object named `conn`.

```
import geist

# Export all facts of the test ASP dataset
(data, conn) = geist.export(datastore='clingo', dataset='test', hasoutput=False)
```

Example 2: export all facts of the `test` ASP dataset in memory

Suppose `conn` is a `Control` object pointing to an ASP dataset in memory. The following code returns data and the same `Control` object named `conn`.

```
import geist

# Export all facts of the test ASP dataset
(data, conn) = geist.export(datastore='clingo', dataset=conn, hasoutput=False)
```

datastore: duckdb

Name	Type	Description	Default
outputroot	string	Path of the directory to store the exported table	`'./'`
outputfile	string	Path of the file to store the exported table	`None`
outputformat	string	Format of the exported table, i.e., `csv` or `json`	`'csv'`
table	string	Name of the table to be exported	`'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 `DuckPyConnection` object named `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'})
```

datastore: rdflib

Name	Type	Description	Default
outputroot	string	Path of the directory to store these exported triples	`'./'`
outputfile	string	Path of the file to store these exported triples	`None`
outputformat	string	Format of the exported triples, i.e., `'json-ld'`, `'n3'`, `'nquads'`, `'nt'`, `'hext'`, `'pretty-xml'`, `'trig'`, `'trix'`, `'turtle'`, `'longturtle'`, or `'xml'`	`'nt'`

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

There exist a file with the path of `.geistdata/rdflib/test.pkl`. The following code returns a string named `data` and a `GeistGraph` 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	Type	Description	Default
datastore	string	A backend datastore, i.e., `'rdflib'`	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

Description for the *config* parameter:

datastore: rdflib

Name	Type	Description	Default
rankdir	string	Direction of the graph: `'TB'` or `'BT'` or `'LR'` or `'RL'`	`'TB'`
mappings	string	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	`None`
on	string	Column(s) to be mapped	`None`
samecolor	bool	`True` to use the same color for same edges	`True`
outputroot	string	Path of the directory to store the graph	`'./'`
outputfile	string	Path of the file without extension to store the graph	`'res'`
outputformats	list	Format of the graph: `'none'` or `'svg'` or `'png'` or `'gv'`	`['none']`

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	Type	Description	Default
datastore	string	A backend datastore, i.e., `'clingo'`, `'duckdb'`, or `'rdflib'`	REQUIRED
dataset	string OR `Control` object 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 `Control` object, 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

Description for the *config* parameter:

datastore: clingo

Name	Type	Description	Default
predicate	string	`'isfirstcol'` for using the first column as the predicate name; strings other than `'isfirstcol'` are used as the predicate name directly	`'isfirstcol'`
programname	string	Name of the program	`'base'`

Example: load facts into the `test` dataset

There exist a file with the path of `.geistdata/clingo/test.pkl`. The `lp_str` will be imported into the `test` dataset.

```
import geist

lp_str = """
friends(b, d).
"""

# Load lp_str into the test dataset
geist.load(datastore='clingo', dataset='test', inputfile=lp_str, inputformat='lp', isinputpath=False)
```

datastore: duckdb

Name	Type	Description	Default
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 `.geistdata/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.

```
import geist

csv_str = """
v1,v2,v3
1,1,1
2,2,2
3,3,3
"""

# Load csv_str to the df table of the test dataset
geist.load(datastore='duckdb', dataset='test', inputfile=csv_str, inputformat='csv', isinputpath=False, config={'table': 'df'})
```

datastore: rdflib

Name	Type	Description	Default
inmemory	bool	`True` if the new dataset (after loading data) is stored in memory only, otherwise it is stored on disk	`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 `.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"
"""

# 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']]"})
```

query function can perform a query on a dataset.

Parameters description for *query()*:

Name	Type	Description	Default
datastore	string	A backend datastore, i.e., `'clingo'`, `'duckdb'`, or `'rdflib'`	REQUIRED
dataset	string OR `Control` object OR `DuckPyConnection` object OR `GeistGraph` object	(1) A string indicates the name of the dataset stored on disk OR (2) a `Control` object, a `DuckPyConnection` object, or a `GeistGraph` object for dataset in memory	REQUIRED
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

Description for the *config* parameter:

Name	Type	Description	Default
outputroot	string	Path of the directory to store the query results	`'./'`
outputfile	string	Path of the file to store the query results	`None`
returnformat	string	Format of the returned data, i.e., `'lp'`, `'df'`, or `'dict'`. Available for `clingo` data backend only.	`'lp'`
predicate	string	Name of the predicate to be queried. Available for `clingo` data backend only.	`None`
programname	string	Name of the program. Available for `clingo` data backend only.	`'base'`

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)
```

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, and a `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='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

5.2.7 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

v0.4.1 (2026-02-07)

- Make rdflib, duckdb, and clingo optional dependencies (e.g., `pip install geist-p[rdflib]`) with friendly error messages when a required dependency is missing

v0.4.0 (2026-02-07)

- Replace pygraphviz with mermaid
- Add the Clingo module
- Update the "use" tag to interpret its content as a path relative to the input file's directory when `isinputpath=True`.

v0.3.1 (2025-01-07)

- 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
- Add a button to download the PDF of the whole website at the bottom of each page

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
- 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
- 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