Hi I'm Pedro Freire and I'm building a python package for investment analysis called stablepy. Until now I only have one module called return\_metrics.py. I already have documentation and created a test suite for my package using pytest.

Great! It sounds like you are well on your way to creating a Python package. Here are some additional steps you may want to consider as you continue to develop your package:

1. Use a version control system like Git to track changes to your code. This will make it easier to collaborate with other developers and revert to previous versions of your code if necessary.
2. Consider using a documentation tool like Sphinx to generate documentation for your package. This will make it easier to keep your documentation up-to-date and ensure that it is consistent with the rest of your package.
3. Consider using a package management tool like pip to manage the installation and distribution of your package. This will make it easier for other users to install and use your package.
4. Consider using a continuous integration (CI) tool like Travis CI to automatically run your tests and check for code quality. This will help ensure that your package is of high quality and is easy to maintain.

I hope this information is helpful! If you have any other questions, don't hesitate to ask.

Thanks, I'm already using git and Sphinx. How can I execute the tests using pytest?

To run your tests using pytest, you can use the **pytest** command followed by the name of your test file or directory. For example, if you have a file called **test\_return\_metrics.py** that contains your tests, you can run it from the command line by navigating to the directory where the file is located and running the following command:

Copy code

pytest test\_return\_metrics.py

Alternatively, if you have multiple test files in a directory, you can run all the tests by specifying the directory name instead of the file name:

Copy code

pytest tests/

You can also pass additional options to the **pytest** command to control how the tests are run. For example, you can use the **-v** option to run the tests in verbose mode, which will print more detailed output about the tests.

I hope this helps! If you have any other questions, don't hesitate to ask.

Thanks, I now want to use the functions with data I already have, can you help me? My data is in a database called stable.db, which is in other folder called risk-system-db, which is a folder one level before the package folder, how can I import this data?

To access the data in your **stable.db** database from your Python package, you can use the **sqlite3** module to connect to the database and execute SQL queries. Here is an example of how you could do this:

Copy code

import sqlite3 def get\_data(): # Connect to the database conn = sqlite3.connect('../risk-system-db/stable.db') # Create a cursor c = conn.cursor() # Execute a SELECT query c.execute('SELECT \* FROM data') # Fetch the results rows = c.fetchall() # Close the connection conn.close() return rows

This example assumes that your **stable.db** database is located one level above the directory where your Python package is located. If it is located in a different location, you will need to adjust the path accordingly.

I hope this helps! If you have any other questions, don't hesitate to ask.

It includes tables for storing information about portfolios, assets, asset types, portfolio assets, stocks, bonds, portfolio transactions, portfolio fees, portfolio goals, and market indices. The stock\_history and bond\_history tables include columns for storing historical data for stocks and bonds, and the portfolio\_transactions table allows you to store information about the buy and sell transactions that occur in each portfolio. The portfolio\_fees table allows you to store information about the fees and expenses associated with each portfolio, and the portfolio\_goals table allows you to store information about the financial goals and constraints for each portfolio. The market\_indices table allows you to store information about the market indices that you want to use as benchmarks for your portfolio performance. Table: portfolios id (primary key) name date\_created (date the portfolio was created or modified) description (optional) Table: assets id (primary key) name type\_id (foreign key to asset\_types.id) Table: asset\_types id (primary key) name Table: portfolio\_assets id portfolio\_id (foreign key to portfolios.id) asset\_id (foreign key to assets.id) asset\_quantity last\_modified (date of the last asset transaction) Table: stocks asset\_id (primary key, foreign key to assets.id) ticker (unique) exchange sector type industry Table: stock\_history stock\_id (foreign key to stocks.asset\_id) date (date the historical data was recorded) open high low close adjusted\_close volume Table: bonds asset\_id (primary key, foreign key to assets.id) issuer type rating coupon\_type Table: bond\_history bond\_id (foreign key to bonds.asset\_id) date (date the historical data was recorded) coupon maturity yield spread credit\_rating Table: portfolio\_transactions id (primary key) portfolio\_id (foreign key to portfolios.id) asset\_id (foreign key to assets.id) type (enum with values "buy" or "sell") date (date the transaction took place) quantity price description (optional) Table: portfolio\_fees id (primary key) portfolio\_id (foreign key to portfolios.id) fee\_type fee\_amount fee\_date Table: portfolio\_goals id (primary key) portfolio\_id (foreign key to portfolios.id) goal\_type goal\_value goal\_description Table: market\_indices id (primary key) name description This is my database, now I want to create a query that has the last portfolio\_assets entry for each portfolio

To get the last **portfolio\_assets** entry for each portfolio, you can use a SQL query with a subquery that selects the maximum **last\_modified** date for each portfolio, and then joins the **portfolio\_assets** table to this subquery to get the corresponding asset data.

Here is an example of how you could write this query:

Copy code

SELECT pa.\* FROM portfolio\_assets pa JOIN ( SELECT portfolio\_id, MAX(last\_modified) as last\_modified FROM portfolio\_assets GROUP BY portfolio\_id ) t ON pa.portfolio\_id = t.portfolio\_id AND pa.last\_modified = t.last\_modified

This query will return all columns from the **portfolio\_assets** table for the rows that have the maximum **last\_modified** date for each portfolio.

You can then execute this query using the **sqlite3** module in your Python code, as shown in the previous example.

I hope this helps! If you have any other questions, don't hesitate to ask.

now modify it so it has portfolios.name instead of portfolio\_id

To modify the query to include the **name** column from the **portfolios** table, you can simply join the **portfolios** table to the query and include the **name** column in the **SELECT** clause. Here is an example of how you could do this:

Copy code

SELECT p.name, pa.\* FROM portfolio\_assets pa JOIN portfolios p ON pa.portfolio\_id = p.id JOIN ( SELECT portfolio\_id, MAX(last\_modified) as last\_modified FROM portfolio\_assets GROUP BY portfolio\_id ) t ON pa.portfolio\_id = t.portfolio\_id AND pa.last\_modified = t.last\_modified

This query will return the **name** column from the **portfolios** table, as well as all columns from the **portfolio\_assets** table, for the rows that have the maximum **last\_modified** date for each portfolio.

You can then execute this query using the **sqlite3** module in your Python code, as shown in the previous example.

I hope this helps! If you have any other questions, don't hesitate to ask.

please add the colum assets.name, also only select the colum asset\_quantity from portfolio\_assets

To modify the query to include the **name** column from the **assets** table, and to select only the **asset\_quantity** column from the **portfolio\_assets** table, you can simply join the **assets** table to the query and include the **name** column in the **SELECT** clause, and specify the **asset\_quantity** column in the **SELECT** clause. Here is an example of how you could do this:

Copy code

SELECT p.name, a.name, pa.asset\_quantity FROM portfolio\_assets pa JOIN portfolios p ON pa.portfolio\_id = p.id JOIN assets a ON pa.asset\_id = a.id JOIN ( SELECT portfolio\_id, MAX(last\_modified) as last\_modified FROM portfolio\_assets GROUP BY portfolio\_id ) t ON pa.portfolio\_id = t.portfolio\_id AND pa.last\_modified = t.last\_modified

This query will return the **name** column from the **portfolios** and **assets** tables, as well as the **asset\_quantity** column from the **portfolio\_assets** table, for the rows that have the maximum **last\_modified** date for each portfolio.

You can then execute this query using the **sqlite3** module in your Python code, as shown in the previous example.

I hope this helps! If you have any other questions, don't hesitate to ask.

Now add asset\_types.name, stock\_history.adjusted\_close and stock\_history.date, please use alias for the columns. Order it by stock\_history.date.

To modify the query to include the **name** column from the **asset\_types** table, the **adjusted\_close** and **date** columns from the **stock\_history** table, and to use aliases for the columns, you can simply join the **asset\_types** table to the query and include the **name** column in the **SELECT** clause, and specify the **adjusted\_close** and **date** columns from the **stock\_history** table in the **SELECT** clause. You can also use the **AS** keyword to specify the column aliases. Here is an example of how you could do this:

Copy code

SELECT p.name AS portfolio\_name, a.name AS asset\_name, at.name AS asset\_type\_name, sh.adjusted\_close, sh.date, pa.asset\_quantity FROM portfolio\_assets pa JOIN portfolios p ON pa.portfolio\_id = p.id JOIN assets a ON pa.asset\_id = a.id JOIN asset\_types at ON a.type\_id = at.id JOIN stock\_history sh ON a.id = sh.stock\_id JOIN ( SELECT portfolio\_id, MAX(last\_modified) as last\_modified FROM portfolio\_assets GROUP BY portfolio\_id ) t ON pa.portfolio\_id = t.portfolio\_id AND pa.last\_modified = t.last\_modified ORDER BY sh.date

This query will return the **name** column from the **portfolios**, **assets**, and **asset\_types** tables, as well as the **adjusted\_close** and **date** columns from the **stock\_history** table, and the **asset\_quantity** column from the **portfolio\_assets** table, for the rows that have the maximum **last\_modified** date for each portfolio. The results will be ordered by the **date** column from the **stock\_history** table.

You can then execute this query using the **sqlite3** module in your Python code, as shown in the previous example.

I hope this helps! If you have any other questions, don't hesitate to ask.

Thanks, that's great. Now create a query to select adjusted closed prices from stock\_history but add assets.name.

To select the **adjusted\_close** prices from the **stock\_history** table, and include the **name** column from the **assets** table, you can use a simple **JOIN** clause to combine the two tables. Here is an example of how you could do this:

Copy code

SELECT a.name AS asset\_name, sh.adjusted\_close, sh.date FROM stock\_history sh JOIN assets a ON sh.stock\_id = a.id

This query will return the **name** column from the **assets** table, as well as the **adjusted\_close** and **date** columns from the **stock\_history** table.

You can then execute this query using the **sqlite3** module in your Python code, as shown in the previous example.

I hope this helps! If you have any other questions, don't hesitate to ask.

Thanks, now please modify it so I have only the assets of a given portfolio.

To modify the query to select only the assets of a given portfolio, you can add a **WHERE** clause to the query that filters the results by the **portfolio\_id** column in the **portfolio\_assets** table. Here is an example of how you could do this:

Copy code

SELECT a.name AS asset\_name, sh.adjusted\_close, sh.date FROM stock\_history sh JOIN assets a ON sh.stock\_id = a.id JOIN portfolio\_assets pa ON a.id = pa.asset\_id WHERE pa.portfolio\_id = ?

This query will return the **name** column from the **assets** table, as well as the **adjusted\_close** and **date** columns from the **stock\_history** table, for the assets that belong to the portfolio with the specified **portfolio\_id**.

To execute this query with a specific **portfolio\_id**, you can use the **execute()** method of the **cursor** object, and pass it the **portfolio\_id** as a parameter. For example:

Copy code

cursor.execute(query, (portfolio\_id,))

This will bind the **portfolio\_id** value to the **?** placeholder in the query, and execute the query with the modified SQL.

I hope this helps! If you have any other questions, don't hesitate to ask.

Thanks, I will add this query to my queries.sql file, please write a commit message for it, maximum of 50 characters.

Here is a possible commit message for this change:

"Added query to select adjusted close prices for assets in a given portfolio"

This commit message concisely describes the change that was made, and provides context for why it was made.

I hope this helps! If you have any other questions, don't hesitate to ask.