Extending SAS OnDemand for Academics with Python Integration via SASPy

Zhongshan Cheng

Senior Bioinformatics Research Scientist, Center for Applied Bioinformatics, St. Jude Children's Research Hospital, 262 Danny Thomas Hospital, MS 1122, Memphis, TN 38105, US

Cheng.zhong.shan@gmail.com

ABSTRACT

SAS OnDemand for Academics (ODA) enables free access to SAS software for academic users via a web browser. Despite its convenience, the platform limits advanced users by disabling X commands, which are commonly used for tasks like file I/O and local scripting. SASPy, a Python package developed by SAS Institute, enables local Python environments to interface with remote SAS servers, including ODA. However, it is difficult for users to integrate SASPy with SAS ODA across multiple different computer systems, including macOS, Linux and Windows systems. This severely affects the popularities of both SAS ODA and SASPy. To address this issue, this paper provides platform-specific installation and configuration instructions for Windows, macOS, and Linux systems, along with customization tips to enhance HTML output formatting and incorporate user-defined SAS macros hosted on GitHub that are specifically written to handle compressed large data sets. By combining the flexibility of Python, running system commands via Python, the interactivity of JupyterLab as well as its integration with VS Code and GitHub Coplilot for generating codes using artificial intelligence, and the useful SAS macros provided by COVID19_GWAS_Analyzer, this setup empowers advanced SAS ODA workflows for research and instruction.

INTRODUCTION

SAS OnDemand for Academics (ODA) enables free access to SAS software for academic users via a web browser. Despite its convenience, the platform limits advanced users by disabling X commands, which are commonly used for tasks like file I/O and local scripting. SASPy, a Python package developed by SAS Institute, enables local Python environments to interface with remote SAS servers, including ODA. By leveraging SASPy within a JupyterLab environment, users can execute SAS code, transfer data files, and load custom macros seamlessly. This paper details cross-platform installation procedures and provides techniques to customize SASPy behavior for improved output formatting and macro integration with COVID19_GWAS_Analyzer[1,2], as well as using artificial intelligence to write SAS codes with Visual Studio Code (VS Code) and GitHub Copilot.

PART I: SASPy Integration with SAS ODA on Windows

Step 1. Install Anaconda and JupyterLab

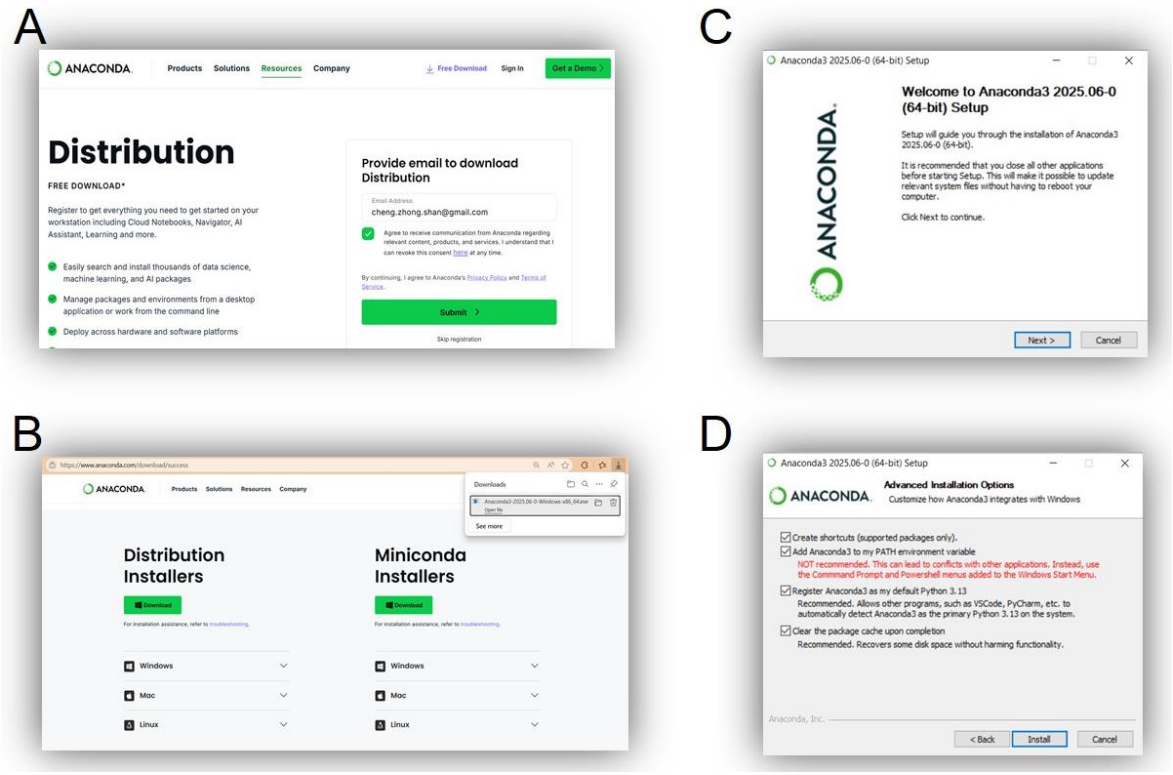


Figure 1. Download and install Anaconda on a Windows system

(A) Register an Anaconda account using your email address. (B) Download the full Anaconda distribution for Windows. (C) Launch the installer and proceed with the default settings. (D) In the "Advanced Installation Options" step, enable all available options, including adding Anaconda to the system PATH, registering Anaconda as the default Python interpreter, enabling desktop shortcuts, and clearing the package cache upon completion.

As illustrated in Figure 1, the installation of Anaconda on a Windows system involves a few straightforward steps. First, navigate to the official Anaconda website and log in using your email address. If you do not already have an account, register by providing your email address before proceeding (Figure 1A).

After logging in Anaconda official website, the download page will present options for both Anaconda and miniconda distributions (Figure 1B). It is recommended to choose the full Anaconda distribution, which includes essential packages such as Jupyter Notebook and JupyterLab. Click the left "Download" button to download the Windows installer (e.g., Anaconda3-2025.06-0-Windows-x86_64.exe). The installer will be saved to your browser's default download directory, typically named "Downloads." If your browser uses a custom directory, locate the installer there.

To begin installation, double-click the downloaded installer (Figure 1C) and proceed with the default configuration. When prompted with the "Advanced Installation Options" (Figure 1D), it is strongly recommended to check all available options:

- Add Anaconda to the system PATH
- Register Anaconda as the default Python interpreter

- Create desktop shortcuts
- Clear the package cache after installation

Enabling these options ensures proper integration with other development tools such as VS Code (https://code.visualstudio.com/download). Notably, JupyterLab—included in the full Anaconda distribution—is required to run SASPy and to view HTML outputs from SAS sessions.

Step 2. Confirm JupyterLab Installation

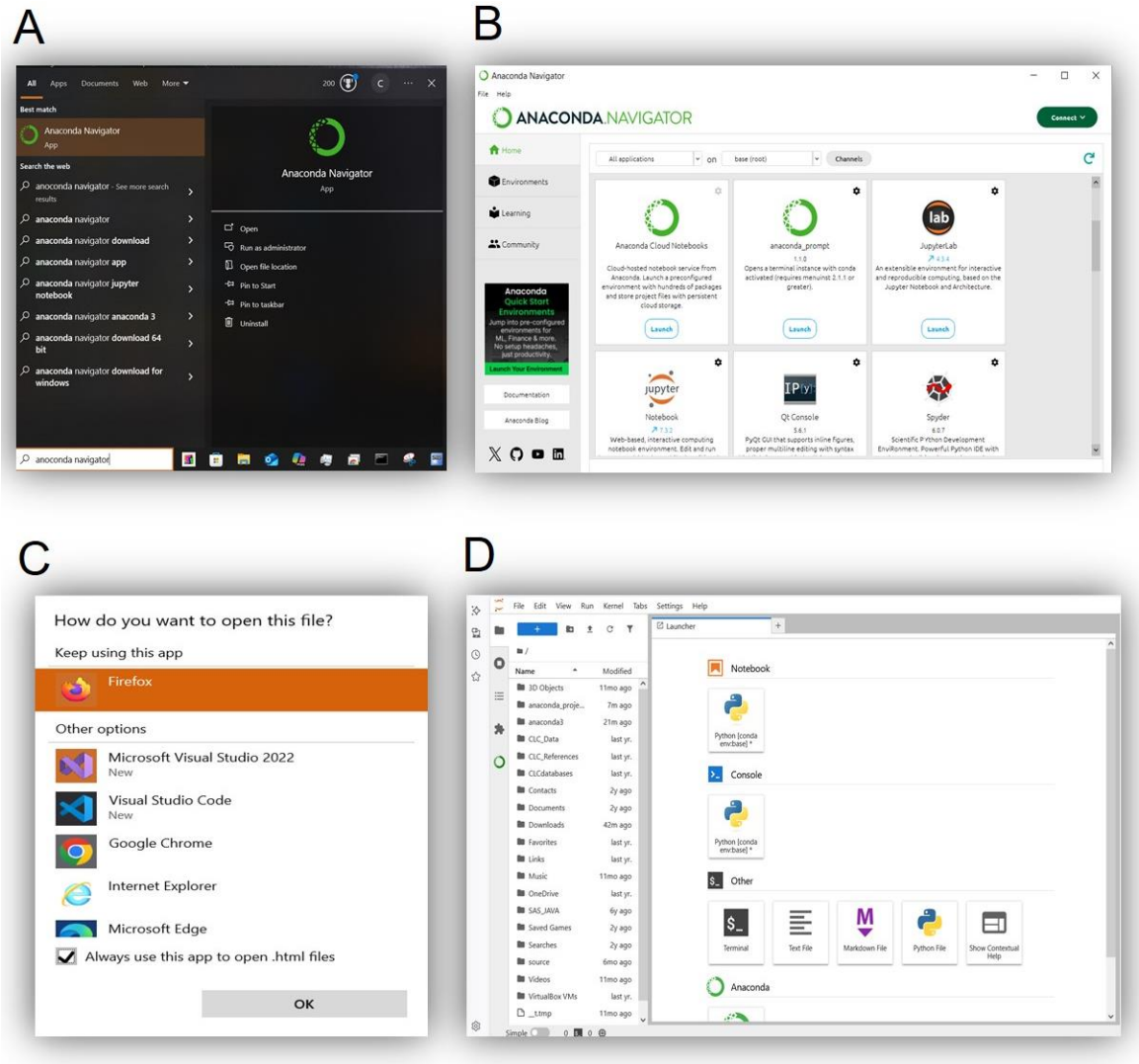


Figure 2. Verify the installation of JupyterLab via Anaconda Navigator
(A) Search for and open Anaconda Navigator using the Windows search bar. (B) Locate the JupyterLab application in the Navigator interface and click the "Launch" button. (C) Upon launching JupyterLab, a prompt appears requesting the selection of a default web browser. (D) Choose a web browser—such as Firefox—as the default to open JupyterLab sessions in a browser window.

To confirm that JupyterLab has been successfully installed with the standard Anaconda distribution, open Anaconda Navigator by typing "Anaconda Navigator" into the Windows search bar (Figure 2A). After launching the Anaconda Navigator, you will find JupyterLab listed among the available applications (Figure 2B). Click the "Launch" button beneath the JupyterLab icon to open the environment.

When launching JupyterLab for the first time, a system prompt may appear requesting you to choose a default web browser (Figure 2C). It is recommended to install Firefox in advance and select it as the default browser, as JupyterLab runs in a browser-based interface. As displayed in Figure 2D, once launched, JupyterLab will open in the selected browser, presenting its interactive user interface where Python notebooks and other tools can be accessed.

Step 3. Install SASPy via Conda

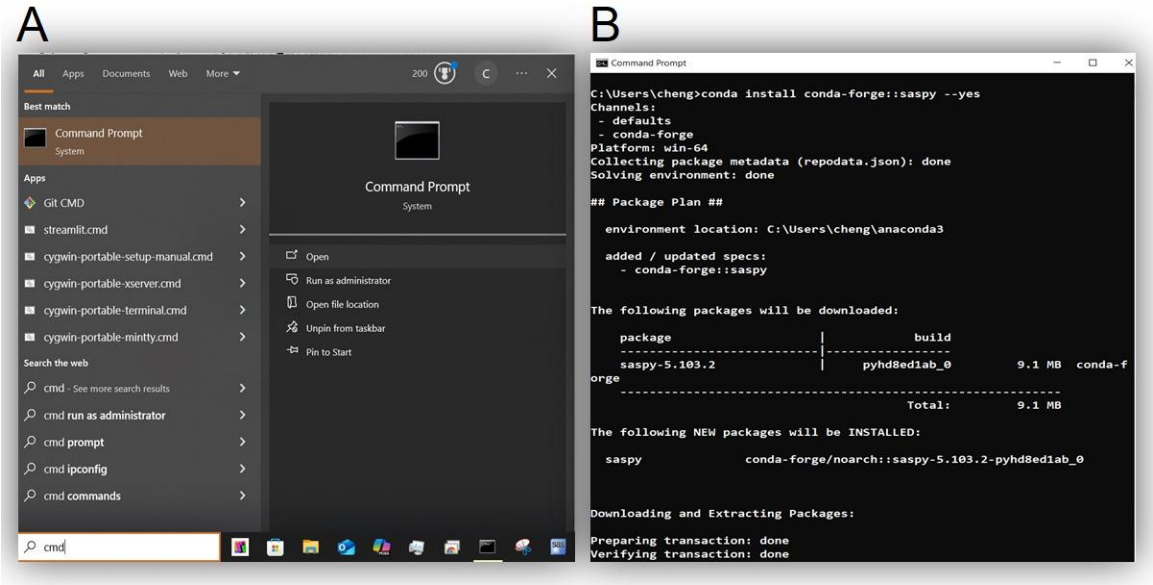


Figure 3. Install SASPy on Windows using Conda
(A) Open the Command Prompt by typing “cmd” in the Windows search bar. (B) Install SASPy by entering the command “conda install conda-forge::saspy --yes” in the terminal window.

To install SASPy on a Windows system using Conda, first open the Command Prompt by typing “cmd” into the Windows search bar and selecting the Command Prompt application (Figure 3A). In the terminal window, execute the command to install SASPy, “conda install conda-forge::saspy --yes” as illustrated in Figure 3B. Note that the “--yes" option automatically confirms all prompts during the installation, allowing the process to proceed without requiring manual confirmation.

If the installation process appears unusually slow using the conda-forge channel, you may alternatively try typing the command “conda install saspy --yes" to speed up the installation. This command typically completes more quickly by using the default Anaconda channel, though it may provide slightly older versions of the package. Both methods install SASPy properly for use in JupyterLab or other Python environments.

Step 4. Configure SASPy Connection to ODA

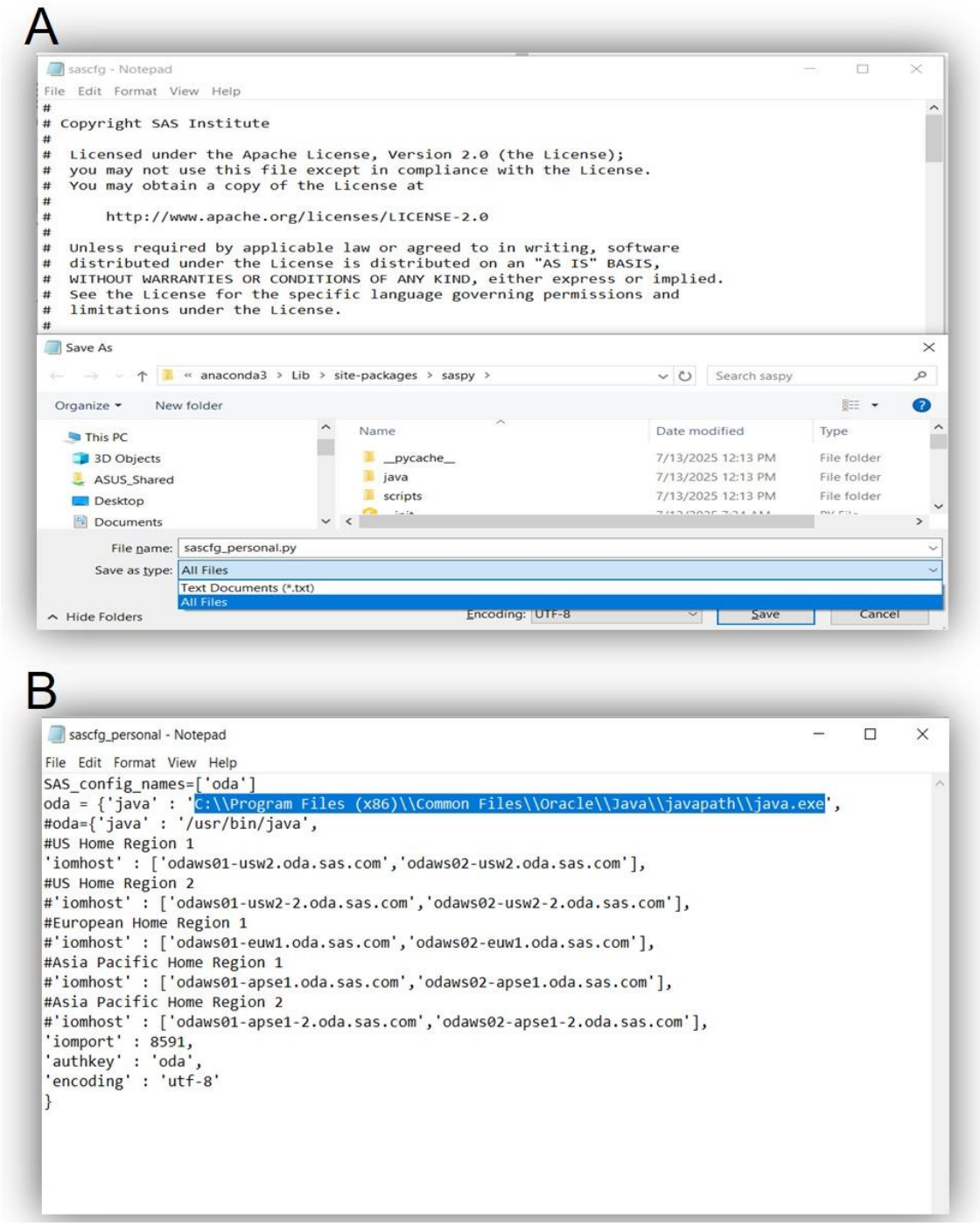


Figure 4. Modify the SASPy configuration file for connecting to SAS ODA
(A) Navigate to the SASPy installation directory and open the default configuration file “sascfg.py”. (B) Save it as “sascfg_personal.py” and edit the configuration for SAS ODA access. The path to the locally installed Java Runtime Environment (JRE) is highlighted and must match your system’s Java installation.

To configure SASPy for use with SAS ODA, begin by navigating to the SASPy installation directory, typically located at: C:\Users\<YourUsername>\Anaconda3\Lib\site-packages\saspy. Replace

<YourUsername> with your actual Windows account name. Locate the file named “sascfg.py”, and open it using a text editor such as Notepad (Figure 4A) if VS code and other editors are not available. Next, save this file as “sascfg_personal.py” in the same directory. To ensure the file is saved correctly with a “.py” extension (and not as a “.txt” file), set the “Save as type” option to “All Files” when saving. If the file is accidentally saved as “sascfg_personal.py.txt”, manually rename it to remove the “.txt” extension. Open “sascfg_personal.py” again in Notepad, press “Ctrl + A” to select all content, and delete it. Then, type the updated configuration block as shown in Figure 4B. Ensure that the java path specified in the configuration reflects the actual path to your installed Java Runtime Environment (JRE). The Java path may appear as either “javapath” or “java8path” depending on your system configuration. If Java is not already installed, download and install it from the official Java website (https://www.java.com/en/). Java 8 or later is recommended for SASPy.

In addition, you must create an authentication file named “_authinfo” in your Windows home directory (e.g., C:\Users\cheng for user cheng; please replace cheng with your user name accordingly). Open Notepad, enter the following content using your SAS ODA credentials: oda user your_email password your_account_password. Replace your_email and your_account_password with your actual SAS ODA login credentials. Save this file as “_authinfo”, again choosing “All Files” as the save type to avoid a “.txt” extension. If necessary, manually rename the file to remove “.txt”. The “_authinfo” file allows SASPy to authenticate with SAS ODA automatically, without prompting for credentials each time.

Lastly, ensure that port 8591 is open on your network. If your system is behind a firewall, ask your network administrator to allow this port for outbound connections, as it is required by SASPy to connect to SAS ODA.

Step 5. Add SAS ODA Encryption JAR Files

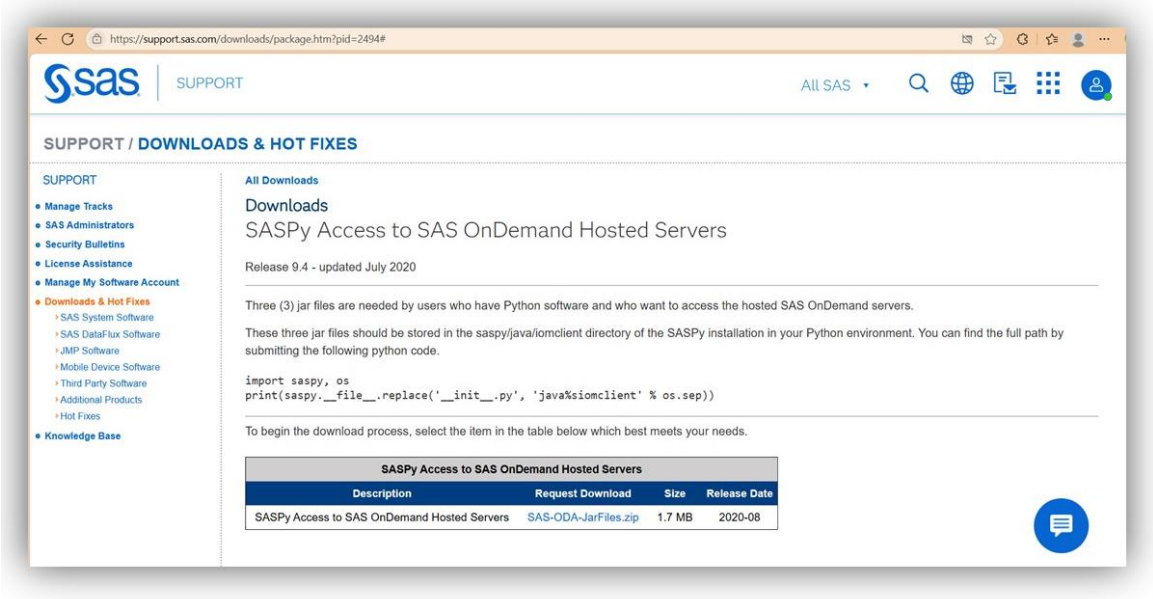


Figure 5. Add SAS-provided encryption Java JAR files to enable SASPy connection with SAS ODA

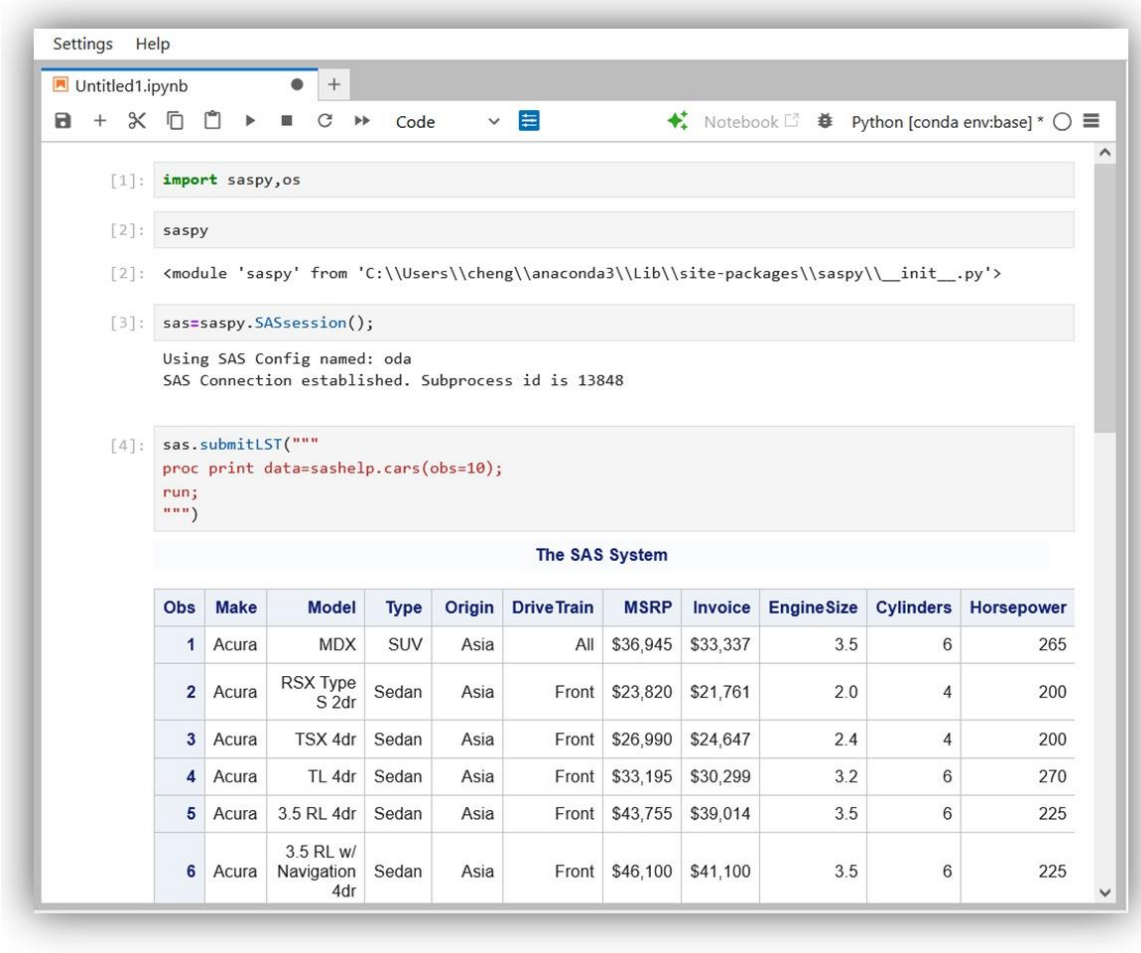
The final step in configuring SASPy to connect to SAS ODA involves installing the encryption Java JAR files provided by SAS Institute. These libraries are essential for enabling secure communication between SASPy and SAS ODA.

First, download the required JAR files from the official SAS support site: https://support.sas.com/downloads/package.htm?pid=2494. Access to this download requires a SAS Profile,

which can be created for free using a valid email address. After downloading, extract or copy the required JAR files—typically including “sas.security.sspi.jar”, “iomclient.jar”, and others—into the following SASPy subdirectory: “C:\Users\<YourUsername>\Anaconda3\Lib\site-packages\saspy\java\iomclient”. Replace “<YourUsername>” with your actual Windows username. Make sure all the provided encryption JAR files are placed in the “iomclient” folder. This ensures that SASPy can locate and use them when initializing a secure connection to SAS ODA.

Step 6. Launch JupyterLab and Test SASPy

Once all configuration steps have been completed—including editing the SASPy configuration file, setting up the “_authoinfo” file, and placing the SAS-provided encryption Java JAR files in the correct directory—you are ready to test the connection between SASPy and SAS ODA.



The SAS System

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4	200
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225
6	Acura	3.5 RL w/ Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225

Figure 6. Test the SASPy connection to SAS ODA within JupyterLab
After completing all configuration steps, re-launch JupyterLab and test the connection by importing SASPy and running SAS codes in a notebook cell, as shown in this figure.

Before testing, completely close and relaunch JupyterLab to ensure that the new session picks up the updated environment settings and has access to the newly added Java JAR files. This step is essential, as changes made to the configuration or file system will not take effect in a running JupyterLab session.

After restarting, open a new notebook in JupyterLab and enter the following commands to test the SASPy connection:

```
import saspy

sas = saspy.SASsession(cfgname='oda')

sas.submitLST("proc print data=sashelp.class; run;")
```

As shown in Figure 6, successful execution of these commands will return the output of the SAS procedure, confirming that SASPy has established a working connection with SAS ODA.

This step verifies that your environment is correctly configured and that you are now ready to run full SAS workflows interactively through JupyterLab using Python and SASPy.

PART II: Improving SASPy Output and Macro Usability

Step 1. Fix HTML Output Alignment

By default, SASPy renders HTML tables with right-aligned headers and cell contents. This alignment is acceptable for short values but becomes problematic when displaying long string variables (Figure 7). A left-aligned format, as illustrated here, is more readable and preferable for textual data.

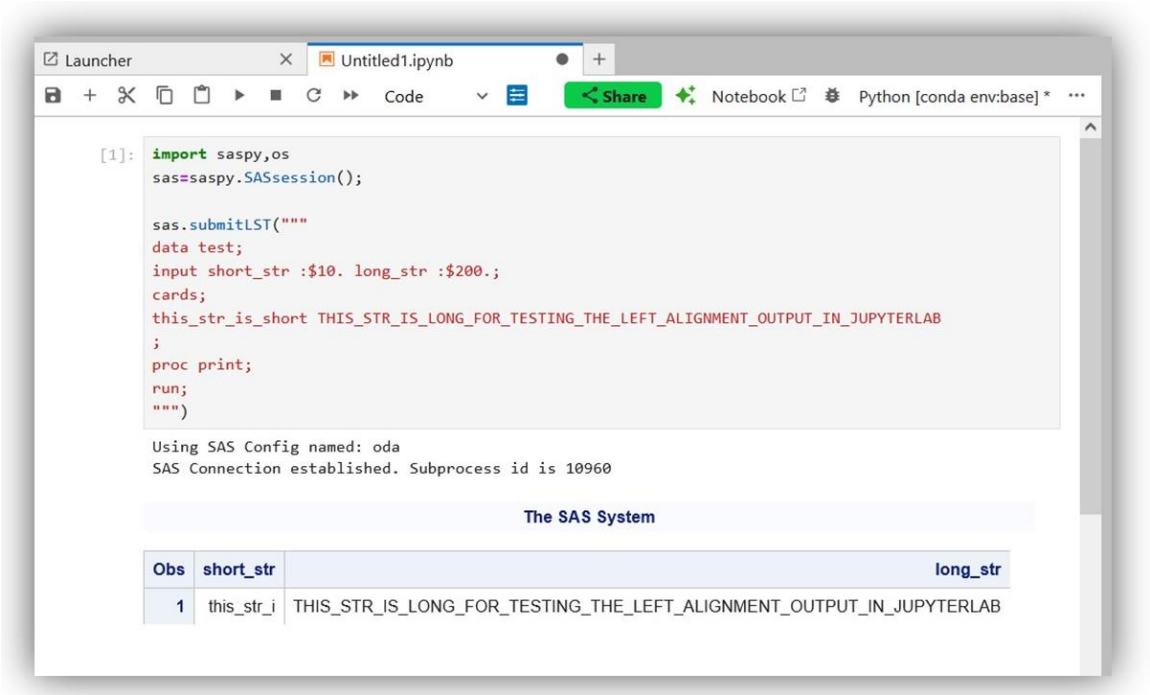


Figure 7. Default SASPy HTML output is right-aligned in JupyterLab, which may reduce readability for long strings

A

```
822         self.sascfg.colorLOG = clog
823         # Inject CSS to force left alignment of all table cells
824         custom_css = """
825         <style>
826         table td, table th {
827             text-align: left !important;
828         }
829         </style>
830         """
831
832         if results.upper() == 'HTML':
833           if   method.lower() == 'listonly':
834             self.DISPLAY(self.HTML(ll['LST']))
835           elif method.lower() == 'listorlog':
836             if len(ll['LST']) > 0:
837               self.DISPLAY(self.HTML(ll['LST']))
838             else:
839               if self.sascfg.colorLOG:
840                 clog = highlight(ll['LOG'], SASLogLexer(), HtmlFormatter(full=True, style=SASLogStyle, lineseparator="<br>"))
841                 self.DISPLAY(self.HTML(clog))
842               else:
843                 print(ll['LOG'])
844           elif method.lower() == 'listandlog':
845             if self.sascfg.colorLOG:
846               clog = highlight(ll['LOG'], SASLogLexer(), HtmlFormatter(full=True, style=SASLogStyle, lineseparator="<br>"))
847               self.DISPLAY(self.HTML(ll['LST']))
848               self.DISPLAY(self.HTML(clog))
849             else:
850               #self.DISPLAY(self.HTML(ll['LST']+"\n<pre>"+ll['LOG']+"</pre>"))
851               self.DISPLAY(self.HTML(ll['LST']))
852               print(ll['LOG'])
853           else:
```

B

```
832         if results.upper() == 'HTML':
833           if   method.lower() == 'listonly':
834             self.DISPLAY(self.HTML(custom_css + ll['LST']))"
835           elif method.lower() == 'listorlog':
836             if len(ll['LST']) > 0:
837               self.DISPLAY(self.HTML(custom_css + ll['LST']))"
838             else:
839               if self.sascfg.colorLOG:
840                 clog = highlight(ll['LOG'], SASLogLexer(), HtmlFormatter(full=True, style=SASLogStyle, lineseparator="<br>"))
841                 self.DISPLAY(self.HTML(clog))
842               else:
843                 print(ll['LOG'])
844           elif method.lower() == 'listandlog':
845             if self.sascfg.colorLOG:
846               clog = highlight(ll['LOG'], SASLogLexer(), HtmlFormatter(full=True, style=SASLogStyle, lineseparator="<br>"))
847               self.DISPLAY(self.HTML(custom_css + ll['LST']))"
848               self.DISPLAY(self.HTML(clog))
849             else:
850               #self.DISPLAY(self.HTML(ll['LST']+"\n<pre>"+ll['LOG']+"</pre>"))
851               self.DISPLAY(self.HTML(custom_css + ll['LST']))"
852               print(ll['LOG'])
853           else:
854             if self.sascfg.colorLOG:
855               clog = highlight(ll['LOG'], SASLogLexer(), HtmlFormatter(full=True, style=SASLogStyle, lineseparator="<br>"))
856               self.DISPLAY(self.HTML(clog))
857               self.DISPLAY(self.HTML(custom_css + ll['LST']))"
858             else:
859               print(ll['LOG'])
860               self.DISPLAY(self.HTML(custom_css + ll['LST']))"
861         else:
```

Figure 8. Modify “sasbase.py” to apply left-aligned formatting to all HTML table outputs from SASPy
(A) Insert a custom CSS block into “sasbase.py” after line 822 to enforce left alignment in all table cells.
(B) Replace every instance of “self.DISPLAY(self.HTML(ll['LST']))” with “self.DISPLAY(self.HTML(custom_css + ll['LST']))” to apply the style globally.

To ensure all SASPy-generated HTML tables in JupyterLab use left-aligned formatting, further updates are required in the SASPy source file “sasbase.py”. These updates involve both injecting a custom CSS style and modifying output rendering function calls throughout the file.

1. Insert the custom CSS block
   Open “sasbase.py” in VS Code and scroll to around line 822, just after the line, insert the following code to define the CSS style that forces left alignment for table headers and cells (shown in Figure 8A, where the inserted code is highlighted in blue).

   ```
   #Inject CSS to force left alignment of all table cells
   custom_css = """
   <style>
   table td, table th {
      text-align: left !important;
   }
   </style>
   """
   ```

2. Update HTML rendering calls
   Throughout “sasbase.py”, locate all instances of “self.DISPLAY(self.HTML(ll['LST']))” and replace them with “self.DISPLAY(self.HTML(custom_css + ll['LST']))”, which prepends the custom CSS to each HTML output block, ensuring that the style is applied consistently.

   To do this efficiently in VS Code:

   - Press Ctrl + H to open the Find and Replace tool.
   - In the “Find” field, enter:
     self.DISPLAY(self.HTML(ll['LST']))
   - In the “Replace” field, enter:
     self.DISPLAY(self.HTML(custom_css + ll['LST']))
   - Click “Replace All” to update every instance in the file.

   As shown in Figure 8B, the original lines are highlighted in light yellow, and the modified lines are shown with changes applied.

3. Final Step
   Save the file and restart JupyterLab to apply the modifications. From now on, all SAS HTML output tables rendered via SASPy will display with left-aligned content, significantly improving the readability of long string variables and overall formatting consistency, as shown in the following Figure 9.

[1]:
```
import saspy,os
sas=saspy.SASsession();

sas.submitLST("""
data test;
input short_str :$10. long_str :$200.;
cards;
this_str_is_short THIS_STR_IS_LONG_FOR_TESTING_THE_LEFT_ALIGNMENT_OUTPUT_IN_JUPYTERLAB
;
proc print;
run;
""")
```

```
Using SAS Config named: oda
SAS Connection established. Subprocess id is 13360
```

The SAS System

Obs	short_str	long_str
1	this_str_i	THIS_STR_IS_LONG_FOR_TESTING_THE_LEFT_ALIGNMENT_OUTPUT_IN_JUPYTERLAB

Figure 9. Improved SASPy HTML output with left-aligned formatting

After modifying the “sasbase.py” script to inject custom CSS and update output rendering, SASPy now produces HTML tables with left-aligned headers and cell content, resulting in improved readability—especially for variables containing long strings.

Step 2. Auto-Load SAS macros from GitHub

A

```
696         self._lastlog = self._io._log
697
698         self._io.submit("""
699             filename M url "https://raw.githubusercontent.com/chengzhongshan/COVID19_GWAS_Analyzer/main/Macros/importallmacros_ue.sas";
700             %include M;
701             Filename M clear;
702             %importallmacros_ue(MacroDir=%sysfunc(pathname(HOME))/Macros,fileRgx=.,verbose=0);
703         """)
704
705
706     def __repr__(self):
707         """
708         Display info about this object
709         :return [str]:
710         """
711         if self._io is None:
712             pyenc = ''
713             if self.sascfg.cfgopts.get('verbose', True):
```

B

```
import saspy,os
sas=saspy.SASsession();
sas.submitLOG("""
%debug_macro;
""")
```

```
Using SAS Config named: oda
SAS Connection established. Subprocess id is 16684

7                                                       The SAS System
Sunday, July 13, 2025 08:54:00 PM

96264      ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inli
ne') device=svg style=HTMLBlue;
96264    ! ods graphics on / outputfmt=png;
96265
96266
96267      %debug_macro;
MPRINT(DEBUG_MACRO):   mlogic;
MLOGIC(DEBUG_MACRO):  Ending execution.
96268
96269
96270
96271      ods html5 (id=saspy_internal) close;ods listing;
96272
8                                                       The SAS System
Sunday, July 13, 2025 08:54:00 PM
```

Figure 10. Enable SASPy to automatically load user-defined SAS macros from GitHub
(A) Modify the "sasbase.py" script by inserting SAS code at the end of the "__init__" method (line 698) to load macros from the COVID19_GWAS_Analyzer GitHub repository. (B) After restarting JupyterLab, test one of the imported macros—%debug_macro—to verify successful integration.

The final enhancement to the "sasbase.py" script enables SASPy to automatically load user-defined SAS macros from a remote GitHub repository, allowing seamless integration of reusable SAS routines within the JupyterLab environment.

To implement this feature, open sasbase.py in VS Code and navigate to the end of the "__init__" method, which begins at line 558 and ends at line 696 in the updated version of the file (which also includes the earlier HTML output alignment improvements).

Insert the following block of code after line 696 (starting at line 698):

```
self._io.submit("""
  filename M url
"https://raw.githubusercontent.com/chengzhongshan/COVID19_GWAS_Analyzer/main/Macros/importall
macros_ue.sas";
  %include M;
  filename M clear;
  %importallmacros_ue(MacroDir=%sysfunc(pathname(HOME))/Macros, fileRgx=., verbose=0);
""")
```

This snippet instructs SAS to download the macro loader script (importallmacros_ue.sas) from the COVID19_GWAS_Analyzer repository on GitHub, execute it, and import all macros into the SAS ODA session automatically. This macro is exclusively designed to download macro files from GitHub for SAS ODA. If users need to access their own macros in GitHub, it is necessary to put this sas macro into their GitHub directory.

As illustrated in Figure 10A, this change allows any SAS macro stored in the linked repository to be accessible from within SASPy as soon as a session is established.

Important: After editing the file, restart JupyterLab to apply the changes.

To verify that the macros have been successfully imported, run a test macro such as %debug_macro in a new Jupyter notebook cell (Figure 10B). If executed without errors and with the expected output, this confirms that the macros have been correctly loaded into the active SAS ODA session through SASPy.

This enhancement improves workflow efficiency by enabling reusable macros to be maintained and updated centrally on GitHub, then dynamically pulled into SASPy sessions as needed.

```
sas.submitLST("""
*%macroparas(macrorgx=list_file);
%list_files4dsd(
dir=.,
file_rgx=sas,
dsdout=x
);
proc print data=x;
run;
""")
```

Obs	fullpath
1	./AddMissingTag4Grps.sas
2	./AddNullData2SeparateDsd.sas
3	./AddRowNumber4Com_Vars.sas
4	./AddSpaces4str.sas
5	./AdjDupsInBim.sas
6	./Adj_missing_cell_value4fisher.sas
7	./AnyOf.sas
8	./App_Dsd_Lbl_Fmt_To_Other_Dsd.sas
9	./Append_All_Data_In_Lib.sas
10	./Append_DataSets_In_Lib.sas
11	./Append_Selected_Data_In_Lib.sas

Figure 11. List all SAS macros loaded from the default "Macros" directory after updating "sasbase.py" to import macros from COVID19_GWAS_Analyzer
Run SAS code to display the macros stored in the default working directory "%sysfunc(pathname(HOME))/Macros", which now contains all macros downloaded from the COVID19_GWAS_Analyzer GitHub repository.

After modifying "sasbase.py" to automatically load SAS macros from the COVID19_GWAS_Analyzer repository, the default working directory for SAS ODA within SASPy becomes:%sysfunc(pathname(HOME))/Macros

This directory houses all imported macros, allowing SASPy to recognize and use them during your SAS sessions.

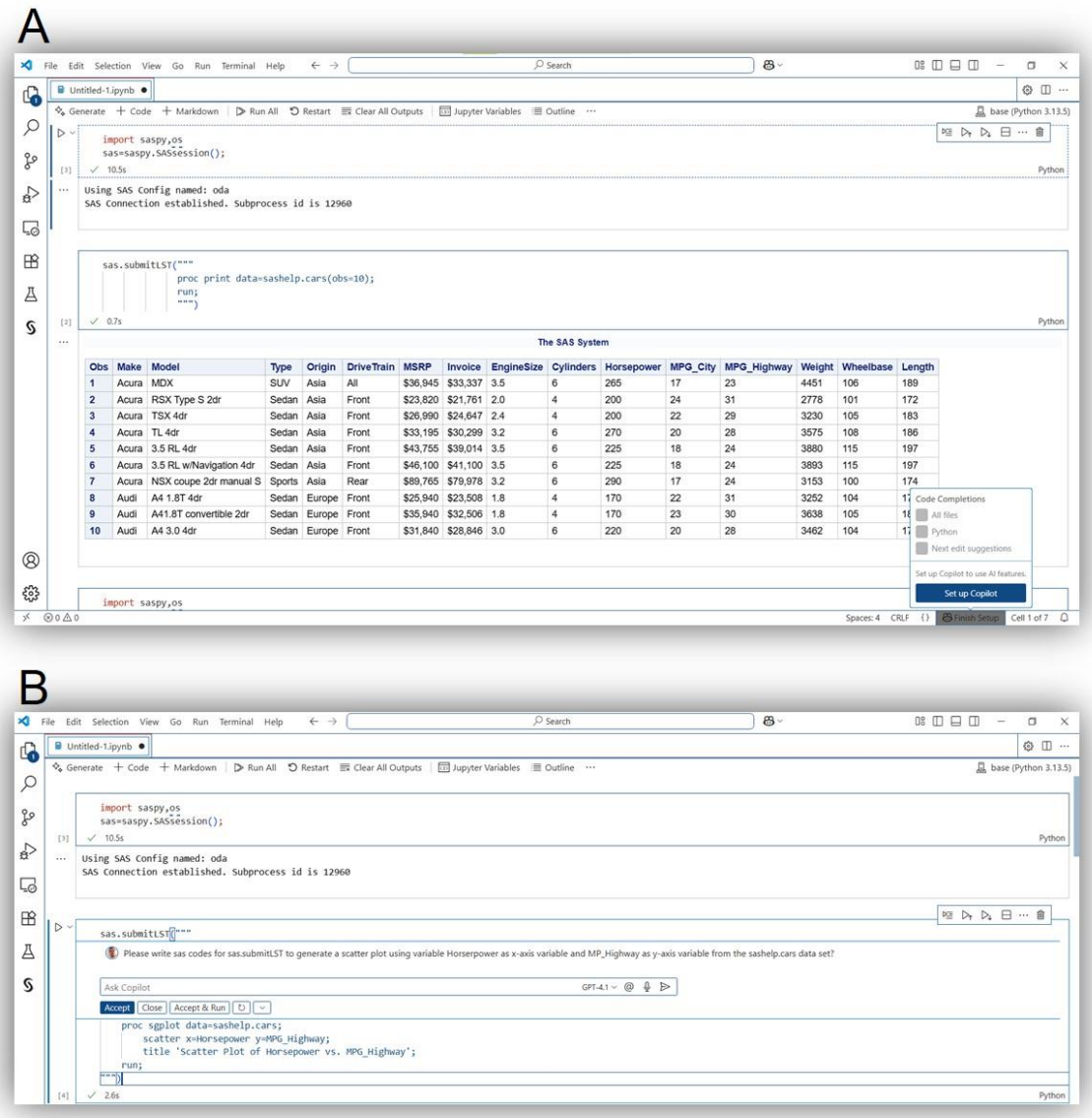
To list all macros available in this directory, you can run the SAS codes shown in Figure 11 within your JupyterLab SASPy session.

PART III: Use VS code to integrate JupyterLab and GitHub Copiolot to write codes with artificial intelligence

To enhance productivity and streamline SASPy coding, integrate VS Code with JupyterLab and GitHub Copilot, an AI-powered code completion tool.

By running SASPy and JupyterLab within VS Code (Figure 12A), you gain a flexible, unified development environment that supports Python, SASPy, and Jupyter notebooks.

GitHub Copilot leverages artificial intelligence to suggest code snippets, complete functions, and assist in writing SAS code interactively for SASPy (Figure 12B). This integration facilitates faster development and reduces manual coding effort, especially for complex SAS procedures and data workflows.



Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	17
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	18
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3.0	6	220	20	28	3462	104	17

Figure 12. Integration of SASPy with JupyterLab and GitHub Copilot via VS Code
(A) Launch SASPy and JupyterLab within VS Code. (B) Use GitHub Copilot's AI-assisted coding features to generate SAS code interactively for SASPy in the Jupyter notebook environment.

To set up this environment, ensure VS Code has the necessary extensions installed:

- Jupyter extension to run notebooks.
- Python extension

Once configured, please select the conda python environment for VS code. You can open a Jupyter notebook running SASPy kernels and receive Copilot suggestions in real time, improving your SAS programming efficiency.

PART IV: Generate and Export High DPI Figures for Publication

This section demonstrates how to use ods graphs to generate high resolution figures for publication. The key parts are to use “dpi=300” along with “ods graphics / height=800 width=600 dpi=300 imagefmt=png”, with the “width”, “height”, and “imagefmt” parameters are customizable for designated figure sizes and figure formats, such as “png” and “jpg.” Please be noted that the output figure is better to be in “png” or “jpg” format, which can be saved into local directory for publication purpose; if supplying “svg” as the figure format, you cannot save or copy the figure in its original resolution, although it is possible to have a screenshot for the figure in much lower resolution that may not be appropriate for publication. The following Figure 12A show the demo codes in VS Code. After generating the figure, you need to click the “…” at the top of the VC Code Window to export the output as a HTML file, and then use a web browser to open the saved HTML file as illustrated in the Figure 12B. Finally, you can hover over the figure and right click your mouse to pop up a panel to save or copy the figure into your designated place in your local computer.

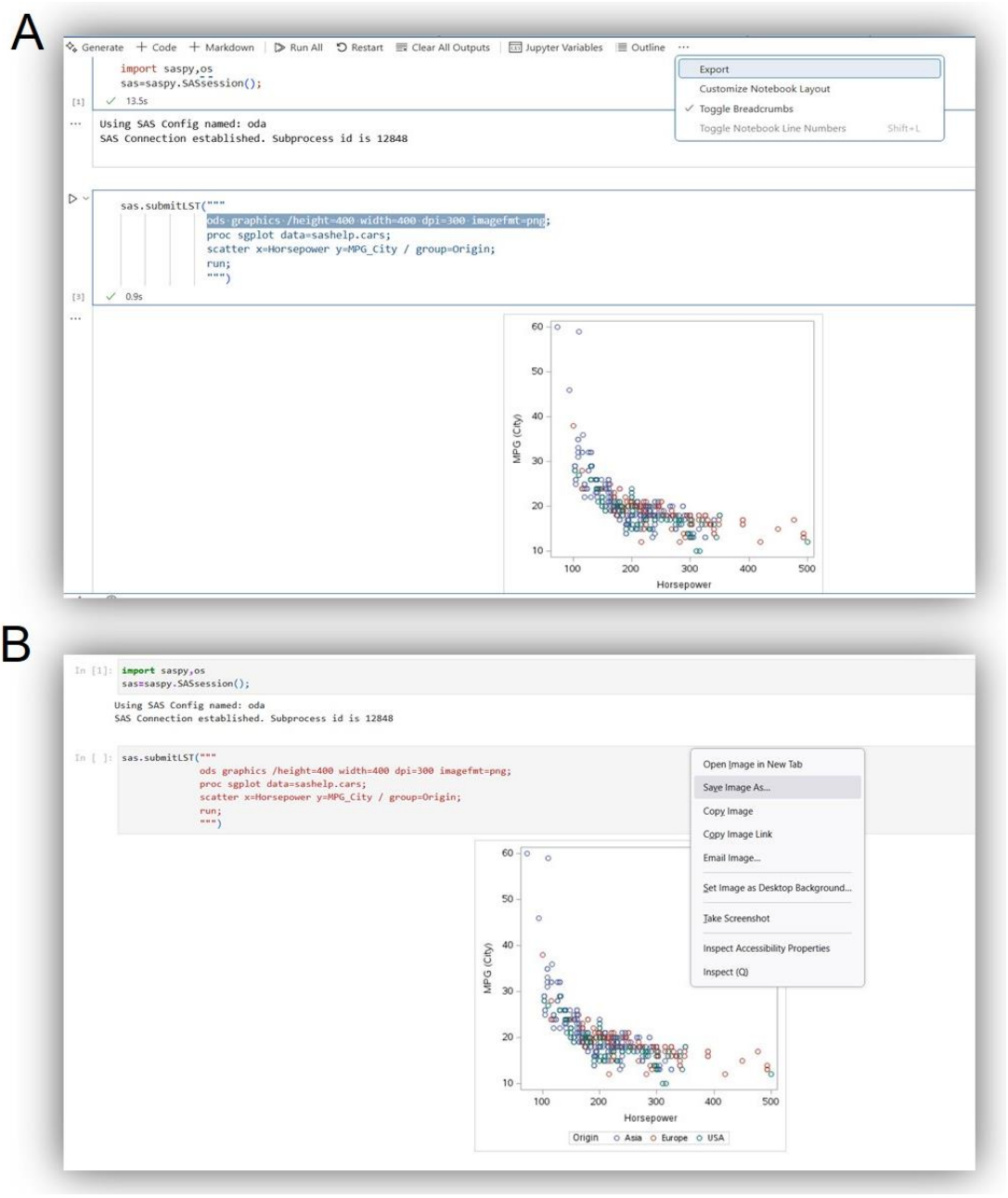


Figure 13. How to create high resolution figure and export it by running SASPy within VS Code

(A) Run SASPy via VS Code to generate a scatter plot and export it into a file with HTML format. (B) Open the exported HTML file and save or copy the figure by hovering over the figure and right clicking your mouse.

PART V: Installation on macOS and Linux (Ubuntu)

This section provides a detailed walkthrough for setting up SASPy and JupyterLab on macOS and Ubuntu Linux, including integration with VS Code and GitHub Copilot for enhanced productivity using AI-assisted coding. Compared to the installation of SASPy and other related software, most of these procedures are completed using commands in Linux terminal.

Step 1. Install Anaconda and JupyterLab

For macOS

1. Open the Terminal app.
2. Install Homebrew if not already installed:

   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

3. Use Homebrew to install Anaconda:

   brew install --cask Anaconda

4. Initialize Conda and restart Terminal:

   conda init

For Ubuntu

1. Download the Anaconda installer from the official website:

   wget https://repo.anaconda.com/archive/Anaconda3-2025.06-0-Linux-x86_64.sh

2. Run the installer:

   bash Anaconda3-2025.06-0-Linux-x86_64.sh

3. Follow the on-screen instructions, and then:

   source ~/.bashrc
   conda init

Step 2. Install SASPy and Java

Install SASPy and dependencies, including nano, JupyterLab and Java with the following command:

conda install -c conda-forge saspy jupyterlab openjdk nano --yes

Step 3. Create the SASPy Configuration File

Navigate to the SASPy directory:

cd ~/Anaconda3/lib/python3*/site-packages/saspy

Copy and edit the configuration file:

```
cp sascfg.py sascfg_personal.py
nano sascfg_personal.py
```

Insert the following configuration:

```
SAS_config_names = ['oda']

oda = {
   'java': '/usr/bin/java',  # Confirm path using `which java`
   'iomhost': 'odaws01-usw2.oda.sas.com',
   'iomport': 8591,
   'authkey': 'oda',
   'encoding': 'utf-8'
}
```

Create the ".authinfo" file in your home directory:

```
nano ~/.authinfo
```

Add the following content and supply your SAS ODA account (email) and its corresponding password:

oda user **your_email** password **your_password**

Save and secure the file:

```
chmod 600 ~/.authinfo
```

Step 4. Install SAS ODA Encryption JAR Files

Download the required Java JAR files from:
https://support.sas.com/downloads/package.htm?pid=2494

Copy them to:

```
cp *.jar ~/Anaconda3/lib/python3*/site-packages/saspy/java/iomclient/
```

Verify that files like "iomclient.jar," "sas.security.sspi.jar" and other jar files are present in the folder.

Step 5. Modify "sasbase.py" for HTML Alignment and Macro Loading

Please download VS Code (https://code.visualstudio.com/download) for MacOS (.zip) or Linux (.deb) and install it by following instructions provided by VS Code website, and then you can use VS Code to modify the file "sasbase.py" based on the following instruction:

Similar to the demonstration for Windows system, edit the "sasbase.py" file in the same directory and insert this block around line 822:

```
custom_css = """
<style>
table td, table th {
```

```
   text-align: left !important;
}
</style>
"""
```

Then replace all instances of "self.DISPLAY(self.HTML(ll['LST']))" with "self.DISPLAY(self.HTML(custom_css + ll['LST']))"

Step 6. Enable Automatic Macro Loading:

At the end of the "__init__" method (after line ~696), add:

```
self._io.submit("""
filename M url
"https://raw.githubusercontent.com/chengzhongshan/COVID19_GWAS_Analyzer/main/Macros/importall
macros_ue.sas";
%include M;
filename M clear;
%importallmacros_ue(MacroDir=%sysfunc(pathname(HOME))/Macros, fileRgx=., verbose=0);
""")
```

Restart JupyterLab to apply the changes.

Step 7. Launch JupyterLab and Test SASPy Connection

Start JupyterLab:

```
jupyter lab &
```

Test the connection:

```
import saspy
sas = saspy.SASsession(cfgname='oda')
sas.submitLST("proc print data=sashelp.class(obs=10); run;")
```

Step 8. Integrate with VS Code and GitHub Copilot

1. If not installed, install VS Code from https://code.visualstudio.com/
2. Open VS Code and install the following extensions:
    - Python
    - Jupyter
3. Set Python interpreter to the Conda environment containing SASPy.
4. Open a ".ipynb" file and begin writing SASPy code.
5. GitHub Copilot will suggest code snippets as you type, improving your efficiency.

PART VI: Advanced Workflows with SAS Macros

Once connected, SASPy enables streamlined workflows included in SASPy and COVID19_GWAS_Analyzer:

- Macro listing: %macroparas

- Uploading files: sas.upload
- Download files: sas.download
- Compressed file import: %ImportFileHeadersFromZIP
- Custom plots: %CaculateMulteQTLs_in_GTEx
- File management: %delete_file_or_dir_with_fullpath
- All other macros included in COVID19_GWAS_Analyzers

These functionalities provided by SASPy and COVID19_GWAS_Analyzer greatly enhance what can be accomplished with SAS ODA.

CONCLUSION

By combining the cloud accessibility of SAS ODA with the flexibility of Python and SASPy, users can perform advanced data operations, incorporate custom macros, and overcome the limitations of a browser-only SAS environment. This paper offers a comprehensive guide to deploying such workflows across Windows, macOS, and Linux, unlocking the full potential of SAS for academic and research purposes.

RESOURCES

SASPy Documentation

https://sassoftware.github.io/saspy/

SAS OnDemand for Academics

https://www.sas.com/en_us/software/on-demand-for-academics.html

COVID19_GWAS_Analyzer

https://sesug.org/proceedings/sesug_2024_SAAG/PresentationSummaries/Papers/151_Final_PDF.pdf

The History and Evolution of SASPy, Including an Overview of What It Can Do and How to Use It, Tom Weber, SAS Institute Inc.

https://support.sas.com/resources/papers/proceedings20/4141-2020.pdf

REFERENCES

1. Cheng, Z., Cai, Y., Zhang, K., Zhang, J., Gui, H., Luo, Y.S., Zhou, J., and DeVeale, B. (2023). MAP3K19 regulatory variation in populations with African ancestry may increase COVID-19 severity. iScience *26*, 107555. 10.1016/j.isci.2023.107555.
2. Zhang, K., Lin, S., Luo, Y.S., and Cheng, Z. (2024). Protocol to search for genetic factors related to severe COVID-19 by analyzing publicly available genome-wide association studies. STAR protocols *5*, 103028. 10.1016/j.xpro.2024.103028.