



ARL-MR-1070 • JAN 2023



Methods for Using Anaconda on Personally Identifiable Information (PII)–Restricted Computers

by Dale Shires, Michael Frauenhoffer, and John Vines

Approved for public release: distribution unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.



Methods for Using Anaconda on Personally Identifiable Information (PII)–Restricted Computers

Dale Shires, Michael Fraunhoffer, and John Vines
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) January 2023		2. REPORT TYPE Memorandum Report		3. DATES COVERED (From - To) 11 August–11 October 2022	
4. TITLE AND SUBTITLE Methods for Using Anaconda on Personally Identifiable Information (PII)–Restricted Computers				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Dale Shires, Michael Frauenhoffer, and John Vines				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEVCOM Army Research Laboratory ATTN: FCDD-RLA-DB Aberdeen Proving Ground, MD 21005				8. PERFORMING ORGANIZATION REPORT NUMBER ARL-MR-1070	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release: distribution unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The data science, deep learning, and machine learning communities commonly utilize prebuilt and pretested libraries to perform complex operations. Quite often software is written in Python, and libraries are managed with the Anaconda system that allows specific versions of libraries to be integrated with Python easily and quickly. This paradigm, however, gets a bit complicated when the computer system to be used for either model development or inferencing or both is placed behind Internet firewalls and restrictions due to the demands of Personally Identifiable Information processing. All is not lost, however, when using Anaconda and Python in such a trying environment. We discuss two different approaches that solve this problem and give examples of these approaches in action: Singularity containers and Conda-Pack.					
15. SUBJECT TERMS Network, Cyber, and Computational Sciences; Anaconda; Python; machine learning; PII; Singularity; containers					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 33	19a. NAME OF RESPONSIBLE PERSON Dale Shires
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (include area code) (410) 278-5006

Contents

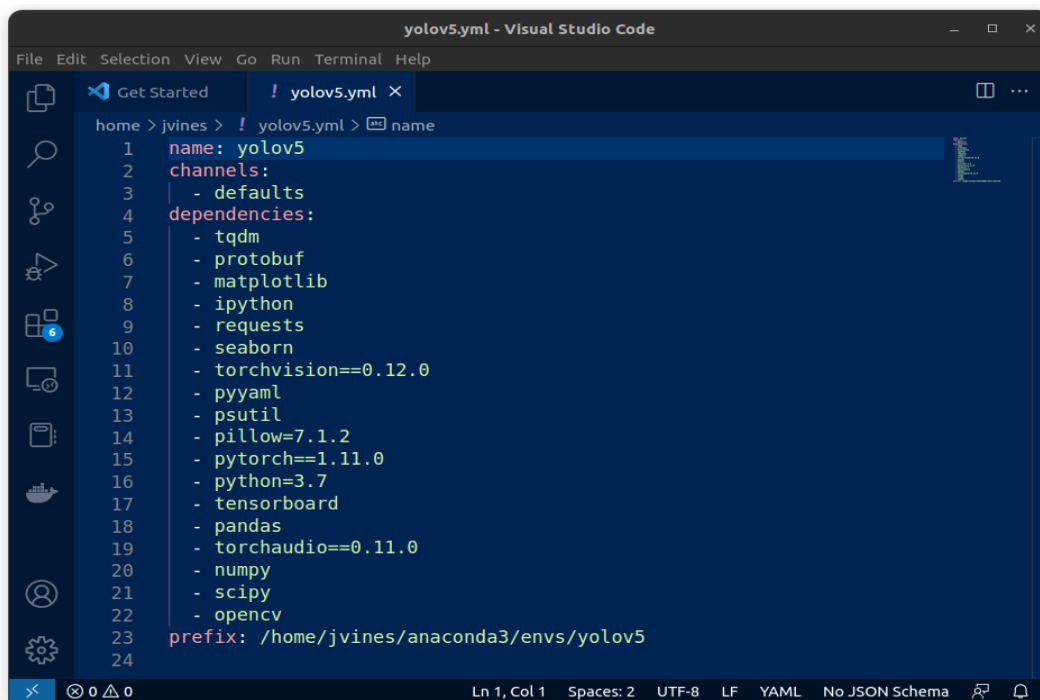
List of Figures	iv
1. Introduction	1
2. Anaconda and Containers	2
2.1 Initial Required Configurations	3
2.2 Downloading and Installing Anaconda	7
2.3 Iris Classification Test Case	7
2.4 Creating, Testing, and Porting a Singularity Container	8
2.5 Possible Issues and Remedies with PSF	9
3. Anaconda and Conda-Pack	10
4. Conclusion	11
5. References	12
Appendix A. Output of “conda list”	13
Appendix B. Python Script: iris.py	17
Appendix C. Output of iris.py on Source Computer	20
Appendix D. Output of iris.py on Destination Computer	22
Appendix E. Output of iris.py when Using Conda-Pack	24
List of Symbols, Abbreviations, and Acronyms	26
Distribution List	27

List of Figures

Fig. 1	An example YAML file	1
Fig. 2	PuTTY configuration for source and destination tunnel ports.....	4
Fig. 3	Connection settings dialog box in Firefox	5
Fig. 4	oVirt login screen.....	5
Fig. 5	Summary page for the VM “test”	6
Fig. 6	Cockpit interface for the VM.....	7

1. Introduction

Anaconda is a Python/R data science distribution and collection of over 7,500 open-source packages that include a package and environment manager (Anaconda 2022). Anaconda can be downloaded and installed in the user's directory space on a computer system and does not necessarily require system administrator or "root" privileges to install. Anaconda can take a fair amount of disk space, so another related option is Miniconda, which only includes the Python interpreter and the package manager and thus reduces overall space requirements. Either software is platform-agnostic; you can install it on numerous operating systems to include Windows, Mac OS, and Linux. Anaconda environments can be created by way of command-line arguments or through the use of a YAML (YAML Ain't Markup Language) configuration file (often seen with a .yaml extension) used to define all of the packages to install in the environment. An example YAML file is shown in Fig. 1.



```
1 name: yolov5
2 channels:
3   - defaults
4 dependencies:
5   - tqdm
6   - protobuf
7   - matplotlib
8   - ipython
9   - requests
10  - seaborn
11  - torchvision==0.12.0
12  - pyyaml
13  - psutil
14  - pillow=7.1.2
15  - pytorch==1.11.0
16  - python=3.7
17  - tensorboard
18  - pandas
19  - torchaudio==0.11.0
20  - numpy
21  - scipy
22  - opencv
23 prefix: /home/jvines/anaconda3/envs/yolov5
24
```

Fig. 1 An example YAML file

Once an Anaconda environment is created and activated, software developers can use the system to install specific versions of many machine learning (ML) packages, such as Pandas, NumPy, and PyTorch to name a few. Thereafter, these ML packages can easily be imported into Python software (code) to greatly simplify and speed the integration of approaches into customized software. Besides ML,

Anaconda supports a myriad of capabilities including neural networks, predictive analysis, and data visualization (anaconda.com 2022 [Oct 11]).

The software that Anaconda manages can be sourced from the Internet, hosted locally, or be from a file that exists on the local host. However, most often software is pulled from active Internet connections, and lacking this connection greatly complicates this process for obvious reasons. The authors recently encountered such a situation. One of the computer systems housed at the US Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory (ARL) was recently cleared to store and process personally identifiable information (PII), but as part of the process, Internet access was greatly curtailed. However, there are approaches to dealing with this problem, and these mitigation efforts are the focus of this report.

We discuss two ways to use Anaconda on systems with limited Internet connectivity. The first uses a concept of containers. Containers are useful for packaging all required software within itself to allow for installation and execution on varying system architectures. If the container software is supported on the development and target systems, the container can execute even if the development system differs from the target and deployed system.

The second method uses a concept of packing all the required software housed in an Anaconda environment into a library and then porting that library to a system with limited connectivity. This method only works when the development and target systems are based on similar architectures and operating systems.

For each of these approaches, we will assume that the Anaconda system has been installed in the user's directory. The methods to install "conda" (short for Anaconda) vary based on the underlying operating system and are covered in more detail on the Anaconda website (Anaconda 2022 [Oct 3]). Commands and prompts are usually listed in the `Consolas` font with the "\$" character as shell prompt.

2. Anaconda and Containers

There are numerous container runtimes (or engines) available today, to include Singularity and Podman among others. Determining which container system to use depends on a number of factors including functionality and ease-of-use for large-scale software deployment. We were looking mainly for a methodology to package and distribute Anaconda, so our criteria really came down to what was "easiest" and currently supported at the DOD Supercomputing Resource Center (DSRC). For these simple reasons, we chose to use Singularity. Like many container systems, Singularity requires a user to have root, or superuser, privileges to be most effective

during the build. To get around this problem, we utilized the Persistent Services Framework (PSF) running at the ARL DSRC (2021).

PSF is currently running on the system “Centennial” at the ARL DSRC. Once a user applies for access to PSF through the standard user-support system and is approved, an account is created on PSF, and the user can enjoy the ability to act as root while in the PSF environment. There are numerous configurations to address to get the overall system to function properly and there are many steps. We enumerate these in the following. In all cases we used Firefox web browser v. 102.3.0.

2.1 Initial Required Configurations

- 1) Acquire krb5 tickets. Use the High Performance Computing (HPC) Kerberos application to acquire login credentials and tickets.
- 2) Use putty to log into centennial15, which is the node where the PSF system is running. Also, putty must be configured properly for this approach to work. The first step is to configure the source and destination ports in the Secure Shell (SSH) Tunnels configuration. In the PuTTY configuration area, we chose to enter the value 5678 in the source port, and the destination we chose was localhost:7890. Once you enter those values in the appropriate fields, click the “Add” button. The window should appear as shown in Fig. 2. Proceed to log into the system on login node centennial15.

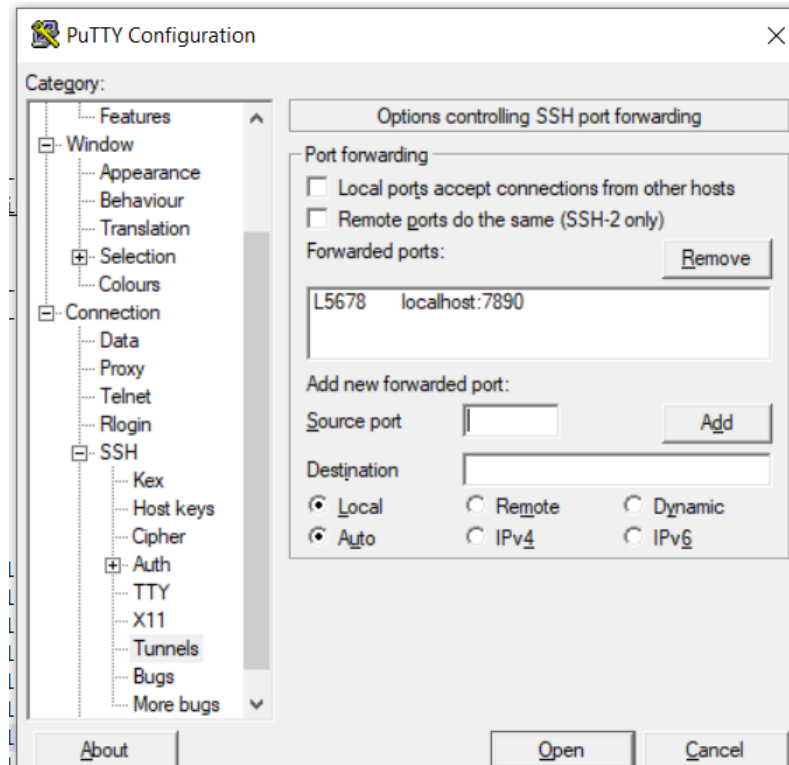


Fig. 2 PuTTY configuration for source and destination tunnel ports

- 3) Start a web browser. We chose to use the Firefox web browser for this test case. Update the connection settings by selecting the hamburger menu (or icon), select “Settings,” then select “Network Settings.” To match the settings we used in PuTTY, select the “Manual proxy configuration” option, and for SOCKS Host enter “localhost” with port number 5678. The radio button SOCKS v5 should be selected, and the option “Proxy DNS when using SOCKS v5” should also be enabled. The window should appear as shown in Fig. 3.

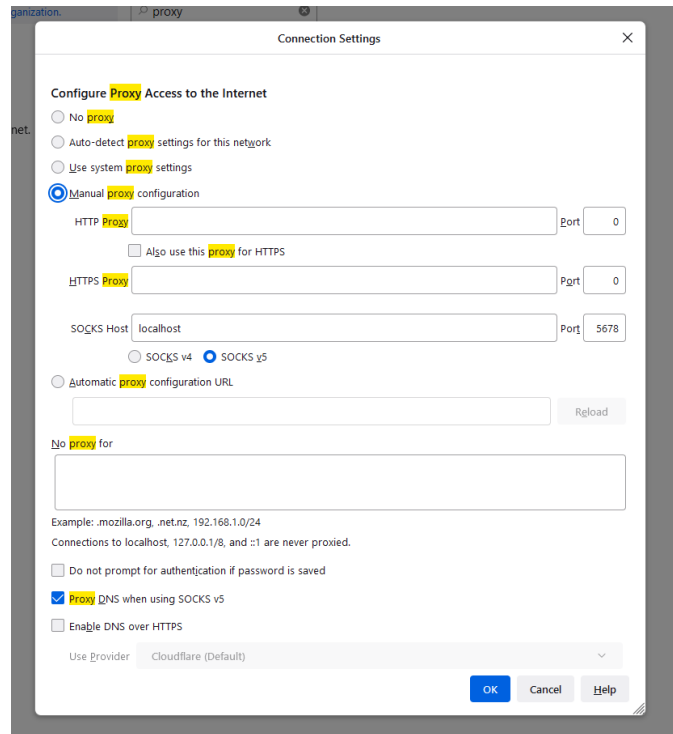


Fig. 3 Connection settings dialog box in Firefox

- 4) Log into the oVirt virtual machine (VM) manager at <https://r6i4n0.ib0.arlu.arl.hpc.mil:8443/>. You will need to supply your username and password created at the time of your PSF account creation. The oVirt interface will let you create, suspend, connect, and more to VMs in the PSF system. The login screen is shown in Fig. 4.

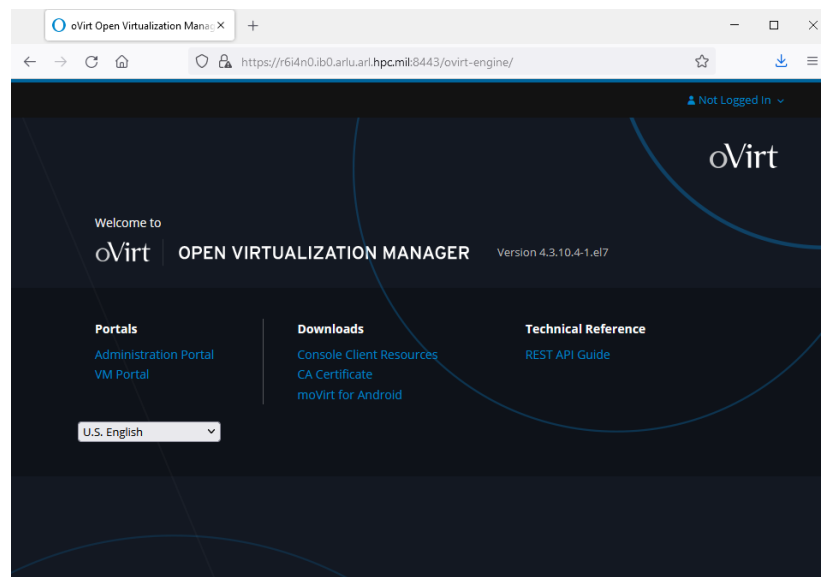


Fig. 4 oVirt login screen

- 5) Create a VM with standard default options. Here we created a VM called “test.” When you click on the VM, the web page will display the summary state of the VM as shown in Fig. 5.

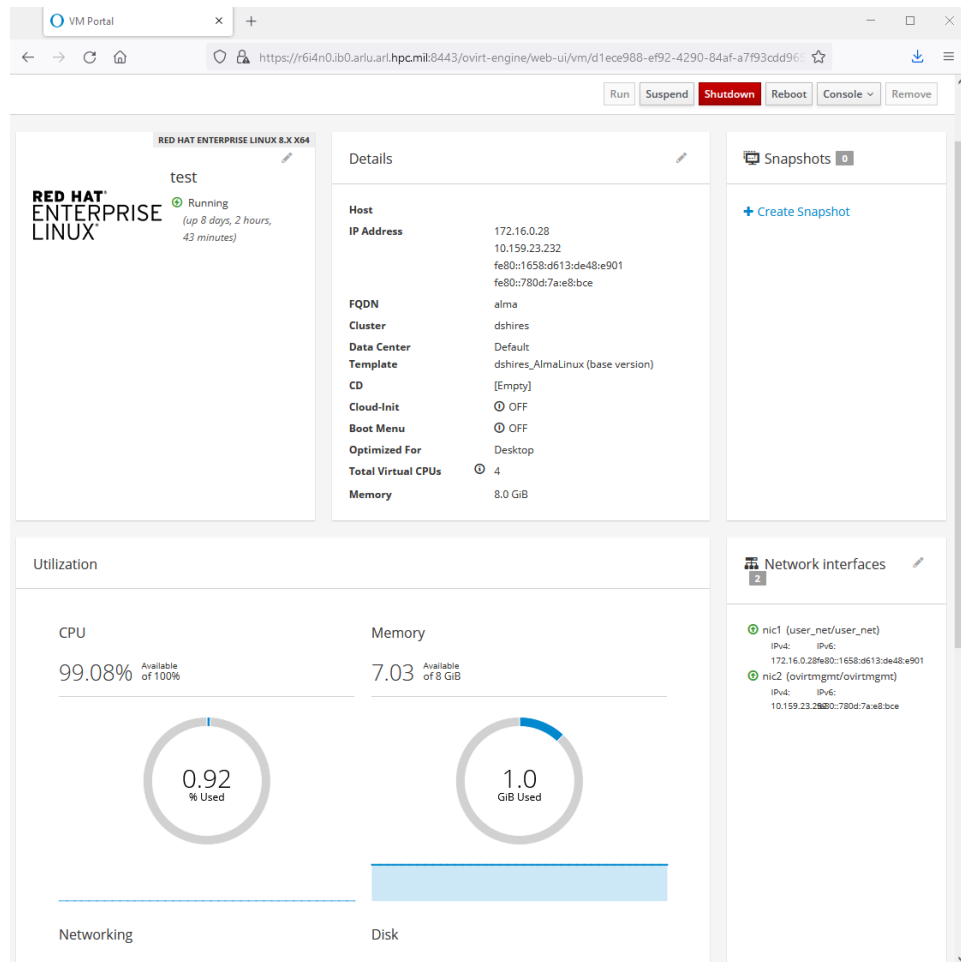


Fig. 5 Summary page for the VM “test”

- 6) Connect to the VM through a web browser. It is possible for the user to enter a terminal window via the console from the oVirt status page, but there is a Cockpit (<https://cockpit-project.org/>) version that has an enhanced look and feel that we chose to use for the remainder of this example. To use this, first determine the IP address for the VM from the Details section of the VM status page. From Fig. 6 we can see that the IP address is 10.159.23.232. To connect to the Cockpit interface, go to port 9090 by typing <https://10.159.23.232:9090> in the address bar and hitting return. This will bring up the Cockpit interface as shown in Fig. 6.

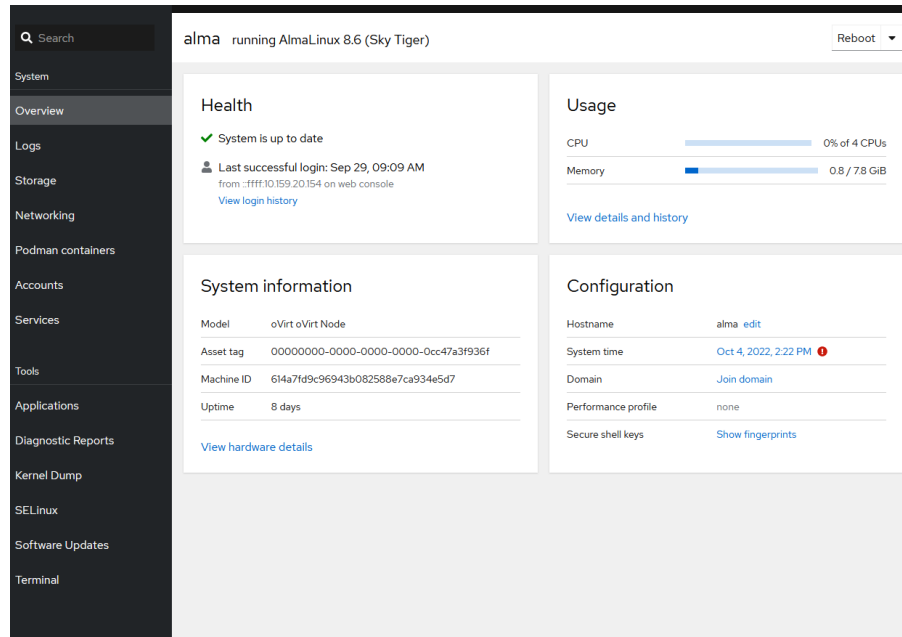


Fig. 6 Cockpit interface for the VM

- 7) Go into the terminal for this VM by selecting the “Terminal” tab in the Cockpit interface. Installing Anaconda and creating the Singularity container will be done through this terminal window.

2.2 Downloading and Installing Anaconda

Anaconda, or conda, can be installed for a variety of operating systems from the download page (Anaconda 2022 [Oct 3]). We are using systems at the ARL DSRC that use a mix of operating systems. Once conda is installed, the user proceeds by creating an environment, activating it, and then installing any required libraries into the environment. For example, suppose someone wants to create a new environment called “my_env” with the most recent version of Pandas (a data analysis library). From a system with Internet access, the following three commands would be entered:

- 1) `$ conda create --name my_env`
- 2) `$ conda activate my_env`
- 3) `$ conda install pandas`

2.3 Iris Classification Test Case

For a test case, we wanted a relatively robust example that uses several Anaconda packages. The iris flower classification dataset provides this requirement, and there is a step-by-step tutorial online on how to build and test several models (machinelearningmastery.com 2020). We created an Anaconda environment called

“my_env.” The libraries installed in the environment are listed using the “conda list” command and are shown in Appendix A – Output of “conda list.” The Python source code is listed in Appendix B – iris.py, and the output of the software when executed natively on centennial is shown in Appendix C – Output of iris.py.

2.4 Creating, Testing, and Porting a Singularity Container

Using Anaconda inside of a container can be tricky. The first time an Anaconda environment is used, it must be initialized for the operating system in use, which in turn must be restarted for Anaconda to take effect. This rebooting step causes problems for Singularity containers. We discovered a good solution to this problem after quite a bit of searching online and by experimenting (csc-training.github.io 2022).

The first step is to export the working Anaconda environment to a YAML file. From the operating system prompt, this is done by “conda env export > environment.yml”. Next, create a Singularity definition file (in this case called iris.def), the contents of which appear as follows:

```
Bootstrap: docker
From: continuumio/miniconda3

%help

    This is a container with conda environments and all materials required to run the
    Iris deep learning models.

%files
    environment.yml
    iris.csv
    iris.py

%environment

%post
ENV_NAME=$(head -1 environment.yml | cut -d ' ' -f2)
echo ". /opt/conda/etc/profile.d/conda.sh" >> $SINGULARITY_ENVIRONMENT
echo "conda activate $ENV_NAME" >> $SINGULARITY_ENVIRONMENT

. /opt/conda/etc/profile.d/conda.sh
conda env create -f environment.yml -p /opt/conda/envs/$ENV_NAME
conda clean --all
chmod 775 iris.csv
chmod 775 iris.py

%runscript
echo "Starting container runscript..."
exec python /iris.py
```

Now this definition file, along with all the other files that are required to run the iris models, are placed in the same directory. (In this example we are using Singularity v. 3.8.7-1.el8.) It is not uncommon to run into issues regarding file and directory size limits when building containers, as they can become quite large. For instance, just using default path settings when building containers, we would often start to get errors that the build failed. Tracing back the reason revealed issues with size limits in our directory structure.

This can be corrected by using two Singularity environment variables: `SINGULARITY_TMPDIR` and `SINGULARITY_CACHEDIR`. Since the `/home` directory had plenty of available room, this directory can be specified for the build. Assuming a user named “joesmith” (for example), issue the following command to build the singularity container:

```
$ sudo SINGULARITY_TMPDIR=/home/joesmith/build/tmp  
SINGULARITY_CACHEDIR=/home/joesmith/build/cache singularity build iris.sif iris.def
```

This will build the file `iris.sif`, where `sif` stands for Singularity Image Format. Note that building a container with several conda packages can take some time to complete.

We transferred this file to the system with PII storage that has limited Internet connectivity with an identical distribution of Singularity (v. 3.8.7-1.el8). Since there is a runscript defined in the `iris.def` build file, it is possible to change the file to executable (by issuing a `chmod +x iris.sif` command) and run it directly from the shell. We did this, and the output is listed in Appendix D on the destination computer. There is no randomization in the script, so the outputs of the two files do match, and the versions of the software modules loaded also match since they are self-contained within the container.

Note that is possible within the destination computer to now also enter a Singularity shell environment that will allow a user to write Python code and import those modules within the container, such as Pandas and NumPy. To do this, one needs to execute the command

```
$ singularity shell iris.sif.
```

If other Anaconda modules are required, they will need to be installed on the source computer with a rebuilt container once again moved to the destination machine.

2.5 Possible Issues and Remedies with PSF

There is a lot of flexibility in the PSF environment; however, there are also many parameters and settings that might need to be adjusted due to things like the Defense Information Systems Agency Security Technical Implementation Guides, for instance. We list some of the possible items to look out for here:

- It is best to use the `dnf` command when installing software in PSF. Using `dnf` works with various security filters to allow the software to install properly. For example, to install Singularity the user would issue the command `dnf install singularity`. This command will update the whitelist of allowed commands.

- The file `/etc/fstab` in PSF might need to be modified to allow for execution privileges on various file systems. For example, if “noexec” is set on a file system, this will prevent code from executing directly from the media.
- If SELinux is active, it might interfere with some software installs or executions.

In those situations where there appears to be strange or hard-to-follow errors, it is best to be cautious and discuss the issue with a system administrator.

3. Anaconda and Conda-Pack

Conda-Pack is a utility that can be used when the development and target computers are of the same architecture type (Conda-Pack 2022). The most common use case is when the development computer has Python and Anaconda installed but the destination computer does not. This method requires a lot less work than the Singularity version and should be used whenever possible to simplify things.

The user must be connected to a terminal on the development computer where Anaconda is installed. This terminal could be the same as that described in Section 2 or simply a Linux terminal on the development computer. Downloading and installing Anaconda proceeds as described in Section 2.2. Conda-Pack should be installed in the root conda environment; that is, prior to issuing any conda activate commands. If an environment is currently active, `conda deactivate` should be done before Conda-Pack is installed.

Following the previous example, assume we have an environment called “my_env” that has all of the required Anaconda software to execute the `iris.py` file. The user issues the command `conda pack -n my_env` from the base environment as follows:

```
$ (base) conda pack -n my_env
Collecting packages...
Packing environment at '/home/joesmith/anaconda3/envs/my_env' to 'my_env.tar.gz'
[#####] | 100% Completed | 1min 10.0s
```

This creates the `my_env.tar.gz` file that is then ported by the user to the destination/target computer. The environment is unpacked into a new directory by issuing the following commands:

```
$ mkdir -p my_env
$ tar -xzf my_env.tar.gz -C my_env
```

Since we are only packing the environment, any other files that are required by the software, for instance in this case the `iris.csv` and `iris.py` files, also need to be moved

to the target computer. To mimic the Anaconda environment, we entered the following command:

```
$ source my_env/bin/activate
```

Then we ran the iris.py Python code with the output given in Appendix E when using Conda-Pack. Note that the code generates the same results as other versions listed. To deactivate the environment and remove it from your path, enter the following command:

```
$ source my_env/bin/deactivate
```

4. Conclusion

The ability to build on prior work and bring in already-built software libraries and routines greatly simplifies the life of a data scientist. Processing datasets that store PII data can be difficult, however, since computer systems may be behind firewalls and other protections that limit the ability to actively download required libraries that data scientists use often. There are at least two ways to deal with this problem, as discussed in this report. We hope that this discussion fast-tracks the use of Anaconda and its data science and ML tools in difficult processing situations such as those encountered when working with PII data.

5. References

- anaconda.com. Anaconda [accessed 2022 Oct 3]. www.anaconda.com.
- anaconda.com. Anaconda use-cases [accessed 2022 Oct 11]. <https://www.anaconda.com/use-cases>.
- [ARL DSRC]. ARL DOD Supercomputing Resource Center. New capability: persistent services framework. In: ARL DSRC spring 2021 newsletter. https://arl.hpc.mil/news/newsletter/ARL_DSRC_Newsletter_Spring_21.pdf.
- conda.github.io. Conda-Pack [accessed 2022 Oct 5]. <https://conda.github.io/conda-pack/>.
- csc-training.github.io. Extra exercise: replicating a conda environment in a container [accessed 2022 Oct 4]. https://csc-training.github.io/csc-env-eff/hands-on/singularity/singularity_extra_replicating-conda.html.
- machinelearningmastery.com. Your first machine learning project in Python step-by-step [accessed 2020 Aug 19]. <https://machinelearningmastery.com/machine-learning-in-python-step-by-step/>.

Appendix A. Output of “conda list”

#	Name	Version	Build	Channel
	_libgcc_mutex	0.1	main	defaults
	_openmp_mutex	5.1	1_gnu	defaults
	_tflow_select	2.3.0	mkl	defaults
	abseil-cpp	20211102.0	hd4dd3e8_0	defaults
	abs1-py	0.15.0	pyhd3eb1b0_0	defaults
	aiohttp	3.8.1	py310h7f8727e_1	defaults
	aiosignal	1.2.0	pyhd3eb1b0_0	defaults
	astunparse	1.6.3	py_0	defaults
	async-timeout	4.0.1	pyhd3eb1b0_0	defaults
	attrs	21.4.0	pyhd3eb1b0_0	defaults
	blas	1.0	mkl	defaults
	blinker	1.4	py310h06a4308_0	defaults
	bottleneck	1.3.5	py310ha9d4c09_0	defaults
	brotli	1.0.9	h5eee18b_7	defaults
	brotli-bin	1.0.9	h5eee18b_7	defaults
	brotlipy	0.7.0	py310h7f8727e_1002	defaults
	bzip2	1.0.8	h7b6447c_0	defaults
	c-ares	1.18.1	h7f8727e_0	defaults
	ca-certificates	2022.07.19	h06a4308_0	defaults
	cachetools	4.2.2	pyhd3eb1b0_0	defaults
	certifi	2022.6.15	py310h06a4308_0	defaults
	cffi	1.15.1	py310h74dc2b5_0	defaults
	charset-normalizer	2.0.4	pyhd3eb1b0_0	defaults
	clang-14	14.0.6	default_hc1a23ef_0	defaults
	click	8.0.4	py310h06a4308_0	defaults
	cryptography	37.0.1	py310h9ce1e76_0	defaults
	cycler	0.11.0	pyhd3eb1b0_0	defaults
	dataclasses	0.8	pyh6d0b6a4_7	defaults
	dbus	1.13.18	hb2f20db_0	defaults
	expat	2.4.4	h295c915_0	defaults
	fftw	3.3.9	h27cfd23_1	defaults
	flatbuffers	2.0.0	h2531618_0	defaults
	fontconfig	2.13.1	h6c09931_0	defaults
	fonttools	4.25.0	pyhd3eb1b0_0	defaults
	freetype	2.11.0	h70c0345_0	defaults
	frozenlist	1.2.0	py310h7f8727e_1	defaults
	gast	0.5.3	pyhd3eb1b0_0	defaults
	giflib	5.2.1	h7b6447c_0	defaults
	glib	2.69.1	h4ff587b_1	defaults
	google-auth	2.6.0	pyhd3eb1b0_0	defaults
	google-auth-oauthlib	0.4.4	pyhd3eb1b0_0	defaults
	google-pasta	0.2.0	pyhd3eb1b0_0	defaults
	grpc-cpp	1.46.1	h33aed49_0	defaults
	grpcio	1.42.0	py310hce63b2e_0	defaults
	gst-plugins-base	1.14.0	h8213a91_2	defaults
	gststreamer	1.14.0	h28cd5cc_2	defaults
	h5py	3.7.0	py310he06866b_0	defaults
	hdf5	1.10.6	h3ffc7dd_1	defaults
	icu	58.2	he6710b0_3	defaults
	idna	3.3	pyhd3eb1b0_0	defaults
	importlib-metadata	4.11.3	py310h06a4308_0	defaults
	intel-openmp	2021.4.0	h06a4308_3561	defaults
	joblib	1.1.0	pyhd3eb1b0_0	defaults
	jpeg	9e	h7f8727e_0	defaults
	keras	2.8.0	py310h06a4308_0	defaults
	keras-preprocessing	1.1.2	pyhd3eb1b0_0	defaults
	kiwisolver	1.4.2	py310h295c915_0	defaults
	krb5	1.19.2	hac12032_0	defaults
	lcms2	2.12	h3be6417_0	defaults
	ld_impl_linux-64	2.38	h1181459_1	defaults
	lerc	3.0	h295c915_0	defaults
	libbrotlicommon	1.0.9	h5eee18b_7	defaults
	libbrotlidec	1.0.9	h5eee18b_7	defaults
	libbrotlienc	1.0.9	h5eee18b_7	defaults
	libclang	10.0.1	default_hb85057a_2	defaults
	libclang-cpp14	14.0.6	default_hc1a23ef_0	defaults
	libcurl	7.84.0	h91b91d3_0	defaults
	libdeflate	1.8	h7f8727e_5	defaults
	libedit	3.1.20210910	h7f8727e_0	defaults
	libev	4.33	h7f8727e_1	defaults

libevent	2.1.12	h8f2d780_0	defaults
libffi	3.3	he6710b0_2	defaults
libgcc-ng	11.2.0	h1234567_1	defaults
libgfortran-ng	11.2.0	h00389a5_1	defaults
libgfortran5	11.2.0	h1234567_1	defaults
libgomp	11.2.0	h1234567_1	defaults
libl1vm10	10.0.1	hbc73fb_5	defaults
libl1vm14	14.0.6	hef93074_0	defaults
libnghttp2	1.46.0	hce63b2e_0	defaults
libpng	1.6.37	hbc83047_0	defaults
libpq	12.9	h16c4e8d_3	defaults
libprotobuf	3.20.1	h4ff587b_0	defaults
libssh2	1.10.0	h8f2d780_0	defaults
libstdcxx-ng	11.2.0	h1234567_1	defaults
libtiff	4.4.0	hecac30_0	defaults
libuuid	1.0.3	h7f8727e_2	defaults
libwebp	1.2.2	h55f646e_0	defaults
libwebp-base	1.2.2	h7f8727e_0	defaults
libxcb	1.15	h7f8727e_0	defaults
libxkbcommon	1.0.1	hfa300c1_0	defaults
libxml2	2.9.14	h74e7548_0	defaults
libxslt	1.1.35	h4e12654_0	defaults
lz4-c	1.9.3	h295c915_1	defaults
markdown	3.3.4	py310h06a4308_0	defaults
matplotlib	3.5.2	py310h06a4308_0	defaults
matplotlib-base	3.5.2	py310hf590b9c_0	defaults
mkl	2021.4.0	h06a4308_640	defaults
mkl-service	2.4.0	py310h7f8727e_0	defaults
mkl_fft	1.3.1	py310hd6ae3a3_0	defaults
mkl_random	1.2.2	py310h00e6091_0	defaults
multidict	5.2.0	py310h5eee18b_3	defaults
munkres	1.1.4	py_0	defaults
ncurses	6.3	h5eee18b_3	defaults
nlTK	3.7	pyhd3eb1b0_0	defaults
nspr	4.33	h295c915_0	defaults
nss	3.74	h0370c37_0	defaults
numexpr	2.8.3	py310hcea2de6_0	defaults
numpy	1.22.3	py310hfa59a62_0	defaults
numpy-base	1.22.3	py310h9585f30_0	defaults
oauthlib	3.2.0	pyhd3eb1b0_1	defaults
openssl	1.1.1q	h7f8727e_0	defaults
opt_einsum	3.3.0	pyhd3eb1b0_1	defaults
packaging	21.3	pyhd3eb1b0_0	defaults
pandas	1.4.3	py310h6a678d5_0	defaults
pcre	8.45	h295c915_0	defaults
pillow	9.2.0	py310hace64e9_1	defaults
pip	22.1.2	py310h06a4308_0	defaults
ply	3.11	py310h06a4308_0	defaults
protobuf	3.20.1	py310h295c915_0	defaults
pyasn1	0.4.8	pyhd3eb1b0_0	defaults
pyasn1-modules	0.2.8	py_0	defaults
pycparser	2.21	pyhd3eb1b0_0	defaults
pyjwt	2.4.0	py310h06a4308_0	defaults
pyopenssl	22.0.0	pyhd3eb1b0_0	defaults
pyarsing	3.0.9	py310h06a4308_0	defaults
pyqt	5.15.7	py310h6a678d5_1	defaults
pyqt5-sip	12.11.0	pypi_0	pypi
pysocks	1.7.1	py310h06a4308_0	defaults
python	3.10.4	h12debd9_0	defaults
python-dateutil	2.8.2	pyhd3eb1b0_0	defaults
python-flatbuffers	2.0	pyhd3eb1b0_0	defaults
pytz	2022.1	py310h06a4308_0	defaults
pyyaml	6.0	pypi_0	pypi
qt-main	5.15.2	h327a75a_7	defaults
qt-webengine	5.15.9	hd2b0992_4	defaults
qtwebkit	5.212	h4eab89a_4	defaults
re2	2022.04.01	h295c915_0	defaults
readline	8.1.2	h7f8727e_1	defaults
regex	2022.7.9	py310h5eee18b_0	defaults
requests	2.28.1	py310h06a4308_0	defaults
requests-oauthlib	1.3.0	py_0	defaults

rsa	4.7.2	pyhd3eb1b0_1	defaults
scikit-learn	1.1.1	py310h6a678d5_0	defaults
scipy	1.7.3	py310h1794996_2	defaults
setuptools	63.4.1	py310h06a4308_0	defaults
sip	6.6.2	py310h6a678d5_0	defaults
six	1.16.0	pyhd3eb1b0_1	defaults
snappy	1.1.9	h295c915_0	defaults
sqlite	3.39.2	h5082296_0	defaults
tensorboard	2.8.0	py310h06a4308_0	defaults
tensorboard-data-server	0.6.0	py310hca6d32c_0	defaults
tensorboard-plugin-wit	1.8.1	py310h06a4308_0	defaults
tensorflow	2.8.2	mkl_py310hd2b8f8c_0	defaults
tensorflow-base	2.8.2	mkl_py310hf890080_0	defaults
tensorflow-estimator	2.8.0	py310h2f386ee_0	defaults
termcolor	1.1.0	py310h06a4308_1	defaults
threadpoolctl	2.2.0	pyh0d69192_0	defaults
tk	8.6.12	h1ccaba5_0	defaults
toml	0.10.2	pyhd3eb1b0_0	defaults
tornado	6.2	py310h5eee18b_0	defaults
tqdm	4.64.0	py310h06a4308_0	defaults
typing-extensions	4.3.0	py310h06a4308_0	defaults
typing_extensions	4.3.0	py310h06a4308_0	defaults
tzdata	2022a	hda174b7_0	defaults
urllib3	1.26.11	py310h06a4308_0	defaults
werkzeug	2.0.3	pyhd3eb1b0_0	defaults
wheel	0.37.1	pyhd3eb1b0_0	defaults
wrapit	1.14.1	py310h5eee18b_0	defaults
xz	5.2.5	h7f8727e_1	defaults
yaml	0.2.5	h7b6447c_0	defaults
yaml	1.8.1	py310h5eee18b_0	defaults
zipp	3.8.0	py310h06a4308_0	defaults
zlib	1.2.12	h7f8727e_2	defaults
zstd	1.5.2	ha4553b6_0	defaults

Appendix B. Python Script: iris.py

```

# This is a test of using conda and ML libraries in general in PSF.
# See https://machinelearningmastery.com/machine-learning-in-python-step-by-step/

import sys
print('Python: {}'.format(sys.version))
import scipy
print('scipy: {}'.format(scipy.__version__))
import numpy
print('numpy: {}'.format(numpy.__version__))
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
import pandas
print('pandas: {}'.format(pandas.__version__))
import sklearn
print('sklearn: {}'.format(sklearn.__version__))

# Load required libraries

from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Load dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv('/iris.csv', names=names)

# Summarize the dataset
# There should be 150 instances and 5 attributes
print(dataset.shape)
print(dataset.head(20))
print(dataset.describe())

# Class distribution
print(dataset.groupby('class').size())

# Split out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1)

models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# Evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='accuracy')
    results.append(cv_results)
    names.append(name)

```

```
print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Make predictions on validation dataset
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# Evaluate predictions
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

Appendix C. Output of iris.py on Source Computer

Python: 3.10.4 (main, Mar 31 2022, 08:41:55) [GCC 7.5.0]

scipy: 1.7.3

numpy: 1.22.3

matplotlib: 3.5.2

pandas: 1.4.3

sklearn: 1.1.1

(150, 5)

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

class

Iris-setosa 50

Iris-versicolor 50

Iris-virginica 50

dtype: int64

LR: 0.941667 (0.065085)

LDA: 0.975000 (0.038188)

KNN: 0.958333 (0.041667)

CART: 0.950000 (0.040825)

NB: 0.950000 (0.055277)

SVM: 0.983333 (0.033333)

0.9666666666666667

[[11 0 0]

[0 12 1]

[0 0 6]]

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

Appendix D. Output of iris.py on Destination Computer

```

Starting container runscript...
Python: 3.10.4 (main, Mar 31 2022, 08:41:55) [GCC 7.5.0]
scipy: 1.7.3
numpy: 1.22.3
matplotlib: 3.5.2
pandas: 1.4.3
sklearn: 1.1.1
(150, 5)

```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

```

count    150.000000    150.000000    150.000000    150.000000
mean      5.843333     3.054000     3.758667     1.198667
std       0.828066     0.433594     1.764420     0.763161
min       4.300000     2.000000     1.000000     0.100000
25%       5.100000     2.800000     1.600000     0.300000
50%       5.800000     3.000000     4.350000     1.300000
75%       6.400000     3.300000     5.100000     1.800000
max       7.900000     4.400000     6.900000     2.500000
class
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
dtype: int64
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.950000 (0.055277)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)
0.9666666666666667
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

Appendix E. Output of iris.py when Using Conda-Pack

```

Python: 3.8.13 (default, Mar 28 2022, 11:38:47)
[GCC 7.5.0]
scipy: 1.7.3
numpy: 1.21.5
matplotlib: 3.5.2
pandas: 1.4.4
sklearn: 1.1.1
(150, 5)

```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

```

count    150.000000    150.000000    150.000000    150.000000
mean      5.843333     3.054000     3.758667     1.198667
std       0.828066     0.433594     1.764420     0.763161
min       4.300000     2.000000     1.000000     0.100000
25%      5.100000     2.800000     1.600000     0.300000
50%      5.800000     3.000000     4.350000     1.300000
75%      6.400000     3.300000     5.100000     1.800000
max       7.900000     4.400000     6.900000     2.500000
class
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
dtype: int64
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.941667 (0.038188)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)
0.9666666666666667
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

List of Symbols, Abbreviations, and Acronyms

ARL	Army Research Laboratory
conda	Anaconda
DEVCOM	US Army Combat Capabilities Development Command
DNS	Domain Name System
DOD	Department of Defense
DSRC	DOD Supercomputing Resource Center
HPC	High Performance Computing
IP	Internet Protocol
ML	machine learning
OS	operating system
PII	personally identifiable information
PSF	Persistent Services Framework
sif	Singularity Image Format
SSH	Secure Shell
VM	virtual machine
YAML	YAML Ain't Markup Language
YML	YAML file

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

1 DEVCOM ARL
(PDF) FCDD RLB CI
TECH LIB

3 DEVCOM ARL
(PDF) FCDD RLA DB
D SHIRES
M FRAUENHOFFER
J VINES