Notes for the BACPAC Data Portal Workshop

Vincent Toups

April 19, 2022

Contents

1 Introduction

Hi everyone. Thank you for making time to attend this small workshop/hackathon to introduce the BACPAC Data Portal.

Today we'll cover:

1. Getting access to the data portal. (Although you hopefully did this before the workshop)
2. Using the web based Data Portal to browse for assets and upload personal files.
3. Starting and connecting to your virtual machines and doing a little analysis.
4. Installing unusual software or libraries in your virtual machines
5. Using Docker containers in the case that all else fails.

There will be a few points during the workshop where I'll as participants to spend some time actually using the portal so that you get go through some of the more technical parts of the process yourself while I'm here to assist.

To give you all an excuse to do more than a little work on the portal, we'll meet again in one week for people to share some interesting visualizations of the Ashar data set.

2 Gaining Access to the BACPAC Data Portal.

The key to accessing the data portal is the "Data Access and Publication Request Form" which you or someone from your research group will fill out. After you submitted your DAPR requests, you should have been added to a project called "`Workshop_April2022`", which will give you access to the data set we will be using in this project (the data set associated with the Ashar

et al 2021 paper "Effect of Pain Reprocessing Therapy vs Placebo and Usual Care for Patients With Chronic Back Pain A Randomized Clinical Trial").

The form is available via Microsoft Teams and needs to be submitted via Teams.

You should all have filled this form out before this meeting and already have data portal access and access to the `Workshop_April2022` Project.

3 The BACPAC Data Portal

Visit www.bacpacresearch.org and click log in. Here you log in with whatever email address you used when making your DAPR request.

Then we're in.

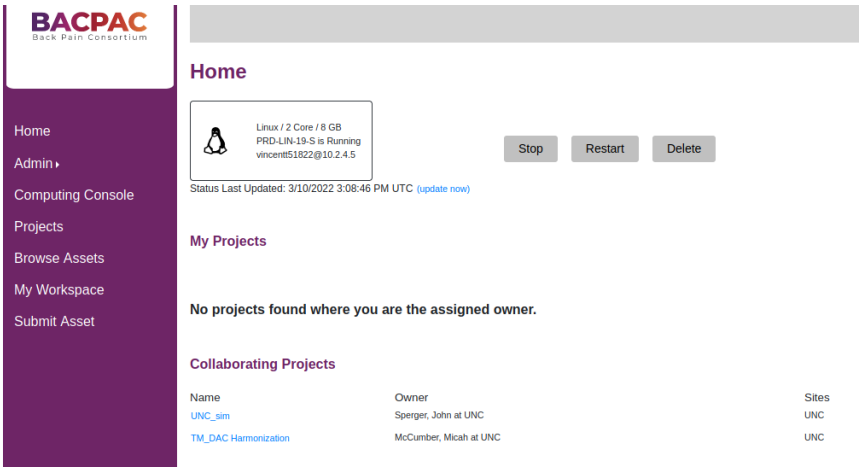


Figure 1: The BACPAC Data Portal Welcome Page

There is a lot going on with the data portal, so let's do a quick run through.

The fundamental reason the data portal exists is to provide researchers with a place to access data which cannot be readily shared for confidentiality and privacy reasons. Instead of bringing the data to your compute resources, the Data Portal lets you create and work on compute resources which are next to the data.

In order to do that, each data portal account is associated with a special profile. Once VMs are created on the Portal's web interface, you can use that profile to log into the machines via your browser and work from there.

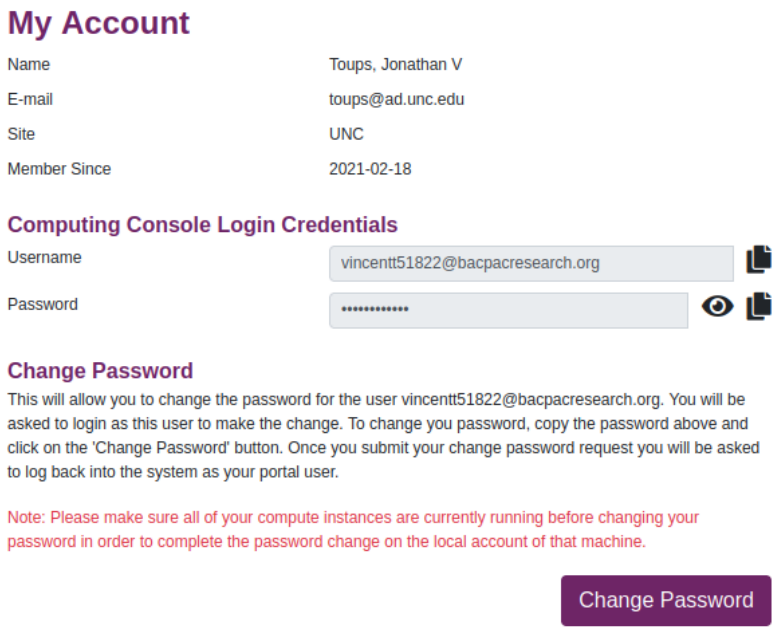


Figure 2: Your profile page contains information necessary to log into your BACPAC Virtual Machines.

4 The Asset Browser

To facilitate asset sharing, the data portal provides an asset browser which allows you to search for data available on the portal in a variety of ways.

A variety of standard meta-data is available for each data set. In addition, many data sets have custom meta-data fields which are controlled by the DAC and allow you to search (for instance) for any data set which contains a particular column or column value (where confidentially allows us to expose that information).

This meta-data search feature can be a little tricky to understand but is pretty useful.

The technically savvy among us might want to think of a data-set's meta-data as a JSON object with a set of Key/value pairs, where most often the value of which is a list.

If you submit data to the Data Portal you may wish to let me know how best to generate meta-data or to send the meta-data fields you'd like to make searchable to me.

5 Project Space

Each Data Access and Publication Request Form results in the creation of a new project (project names are assigned by Cameron Gunn during processing based on the information in the form) under the assumption that each team implied in a new form (as opposed to a request for modification) constitutes a new project.

These project spaces are a great place to host a centralized git repository, particularly because the VM firewall prevents access to sites like github which might otherwise host git repositories.

6 My Workspace

This is personal space. We'll see several uses for this space throughout the workshop, but you can think of it as the "approved" portal between your virtual machines and the outside world.

If you draft a paper inside your VM and want to download a PDF, then you would put it in this space and Download it from the Data Portal. If you want to upload an R library not available on CRAN, or a set of SAS utilities that you carry around with you, you would upload it to this space and it will be available within the VM.

7 The Computing Console

Since this is a hackathon-type event lets just jump right into the computing console.

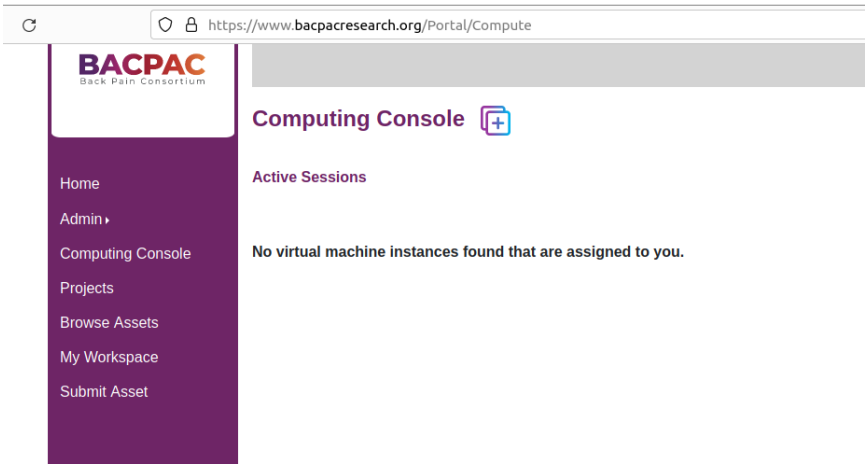


Figure 3: The computing console.

When you first log on you'll see that you have no machines of any type associated with your account. I'm primarily a Linux users but lets begin by starting a Windows VM (a small one). The plan here is for us to start the VM, explore a data set, and then install some custom software.

Click the "+" button next to the "Computing Console" title:

Figure 4: The "+" button allows you to create a new virtual machine.

We have our choice of many virtual machines. Unless you have a specific high performance computing task, its most cost effective for BACPAC if you run a small VM. Let's do that now.

After waiting a little while our VM will be ready to go. Our login process is a bit complicated, but let's go ahead and walk through it.

We click the VM, copy the link from the connect button, paste it into an incognito window, open our profile or VM page, and log in a few times with our *vm* password. Eventually we will find ourselves looking at a Windows Desktop.

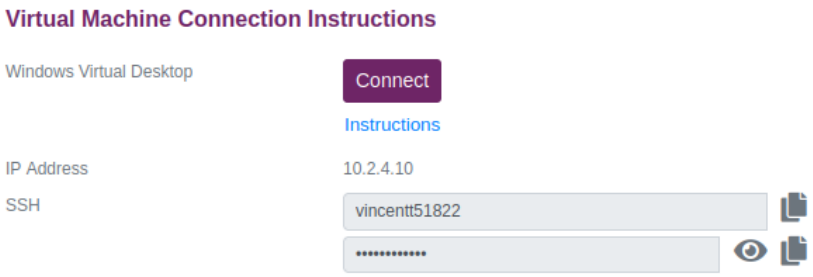


Figure 5: VMs use a different username and password which is presented here.

Figure 6: No one knows.

We click "remote desktop" and then log in:
Click "show options"
The "Open" to open a pre-configured connection setup.

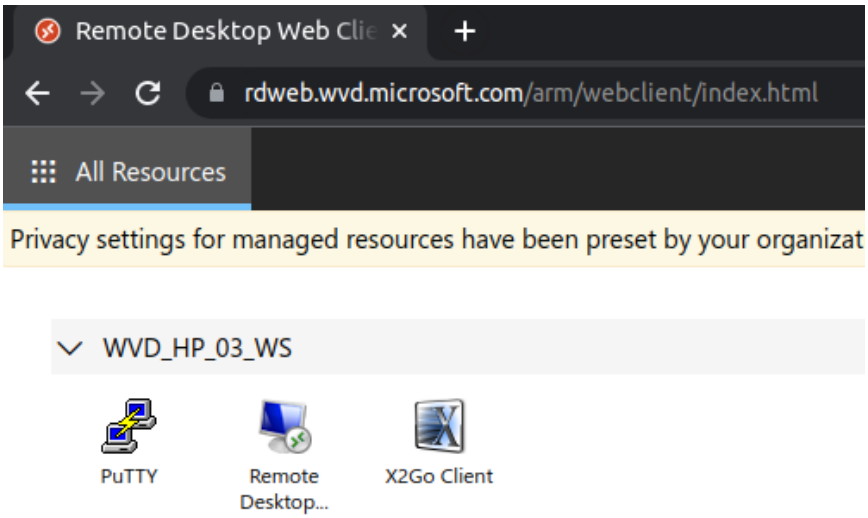


Figure 7: WVD (windows Virtual Desktop), your gateway to the BACPAC Virtual Machines.

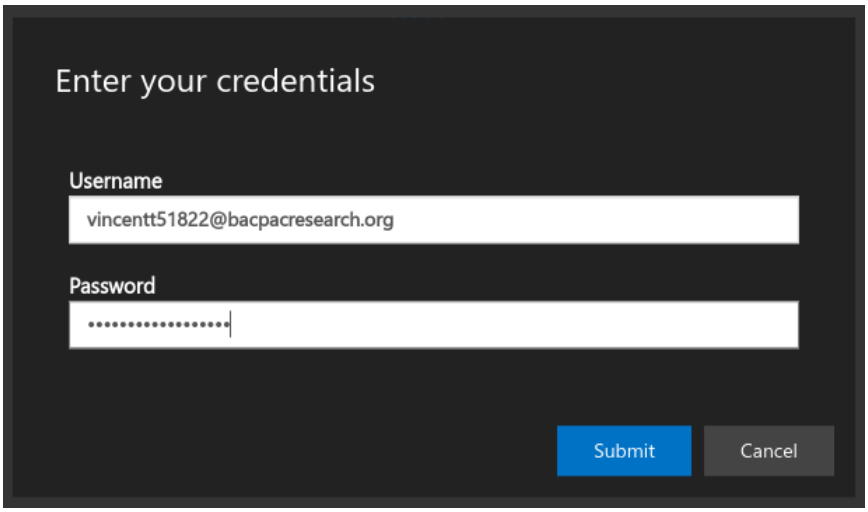


Figure 8: Log in with your special "profile" credentials.

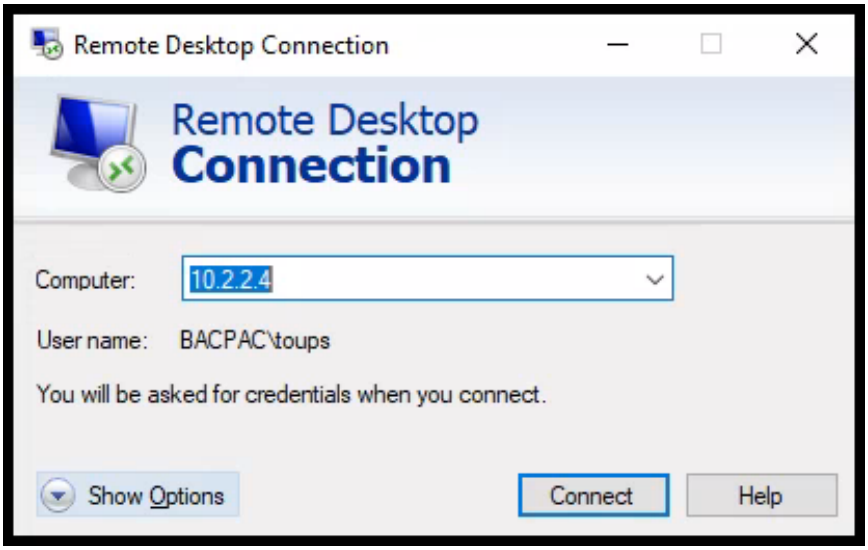


Figure 9: Click "show options."

And then select the right configuration for you virtual machine. Its pretty straightforward: PRD-<OS>-<Number>-<Size>.

After clicking "open" we can now click connect:

And we have a few more hoops to jump through:

Enter your password again:

Accept the certificate:

And you will be connected to your Virtual Machine's desktop.

7.1 Tour of the VM

Regardless of the size of the VM you started, the setup of the Windows VM will be similar. The most important element is the location of the data. Let's navigate to `C:/mnt/containers/` to see our personal, project, and canonical data. For this tutorial we'll be working with the Ashar data in the project space created for this workshop.

Things should more or less work as you expect in this VM. Note, however, that most of the web is blocked to make accidentally leaking confidential data more difficult.

There are exceptions to this rule meant to make life easier: CRAN and PIP repositories are unlocked so you can install material from them (assuming they don't require any other access to the internet).

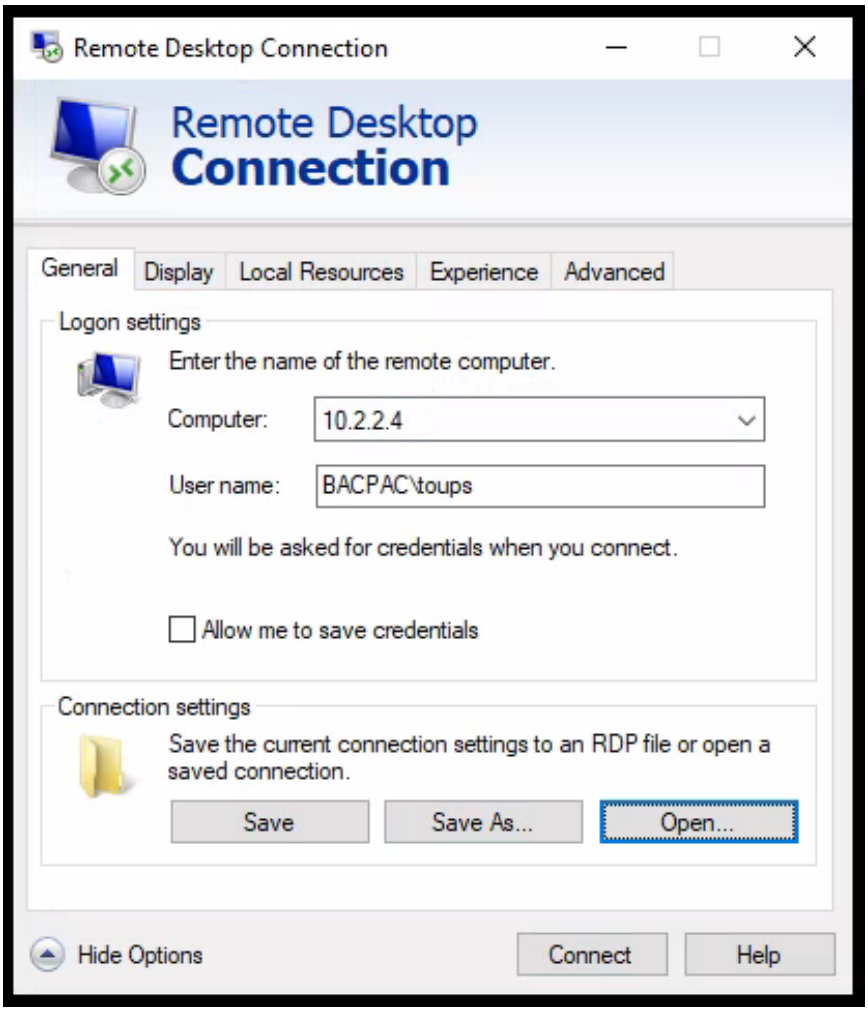


Figure 10: Click "open."

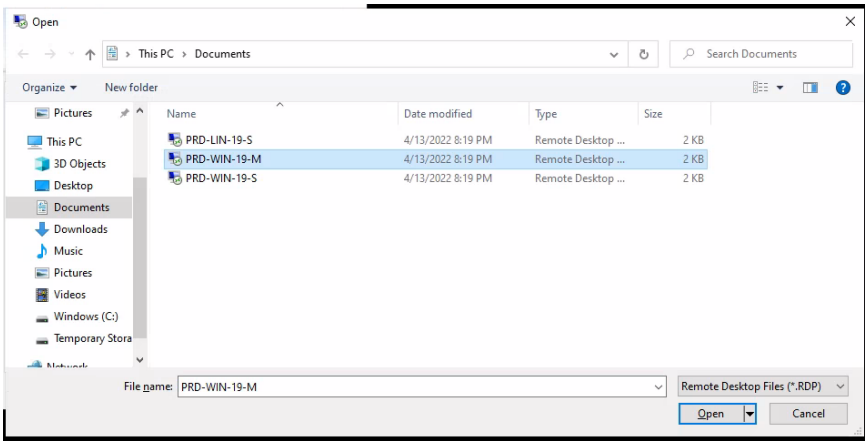


Figure 11: Opening a medium (M) windows (WIN) machine.

For example, it isn't a standard part of the VM load out, but many users enjoy using Jupyter notebooks to organize their work. We can install Jupyter like this:

Or on a Linux machine:

7.2 Data

The main point of the Data Portal is data analysis. The data portal separates its data sets into three categories: Canonical Data which is collected by and available to the entire consortium (this includes publicly available data such as the Ashar data set we'll use in the workshop today), Project Data (which only people who are part of the project may access) and Personal Data which you may use to move things in and out of the data portal.

On both linux and Windows these files are mounted in (more or less) the same location:

`/mnt/containers/`

or

`C:\mnt\containers\`

Within these directories are the personal, project, and canonical spaces.

If you upload data to your personal space, it will appear in the folder under personal. You get the idea.

For today's workshop we'll be looking at the Ashar data set which is available in a shared project folder we created explicitly for the workshop:

`C:\mnt\containters\project\Workshop_April2022/ashar_data`

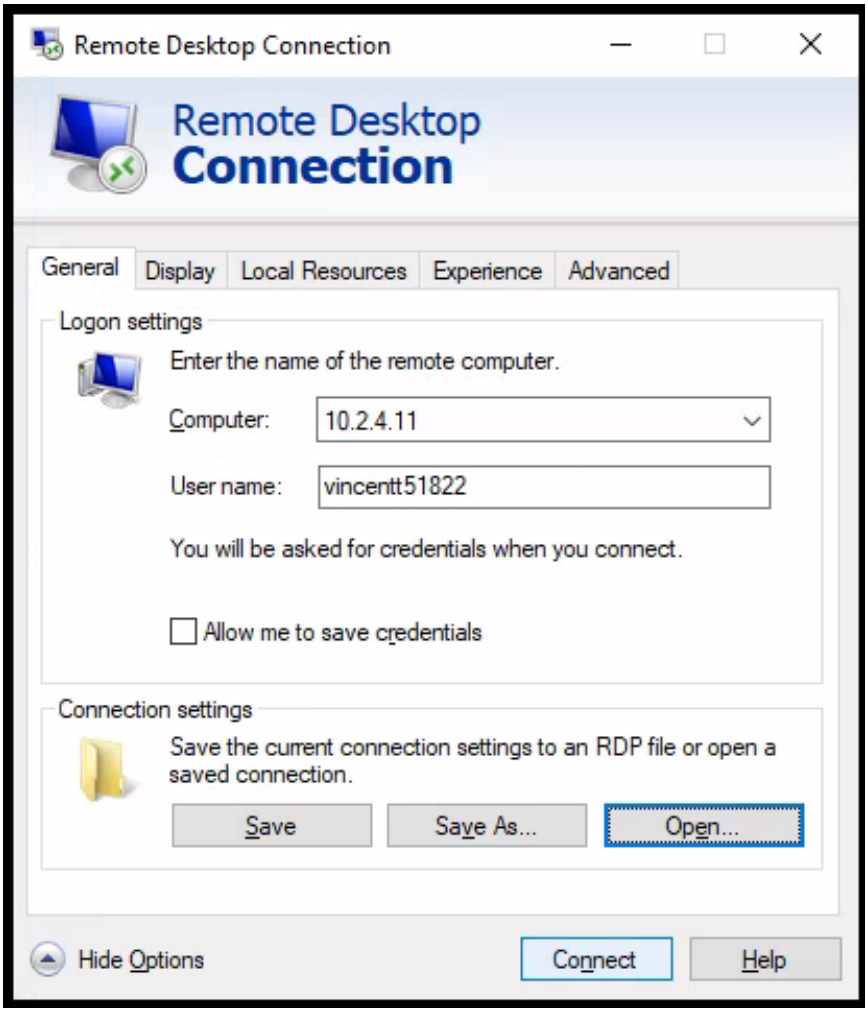


Figure 12: Clicking connect.

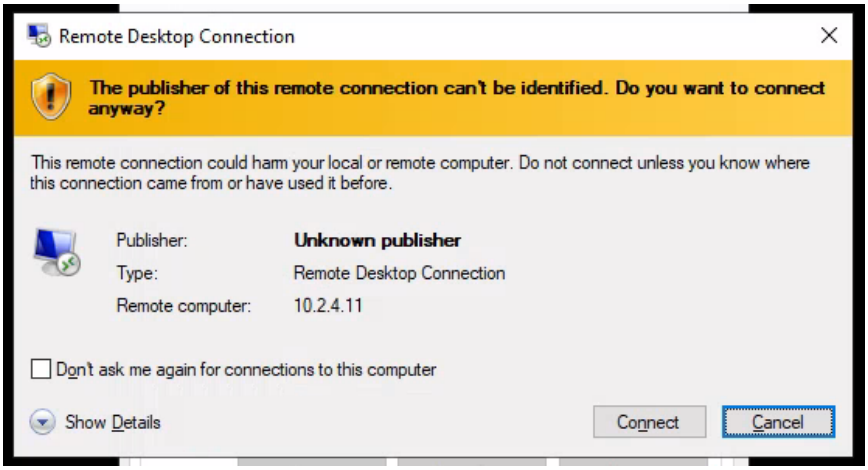


Figure 13: Click "connect."

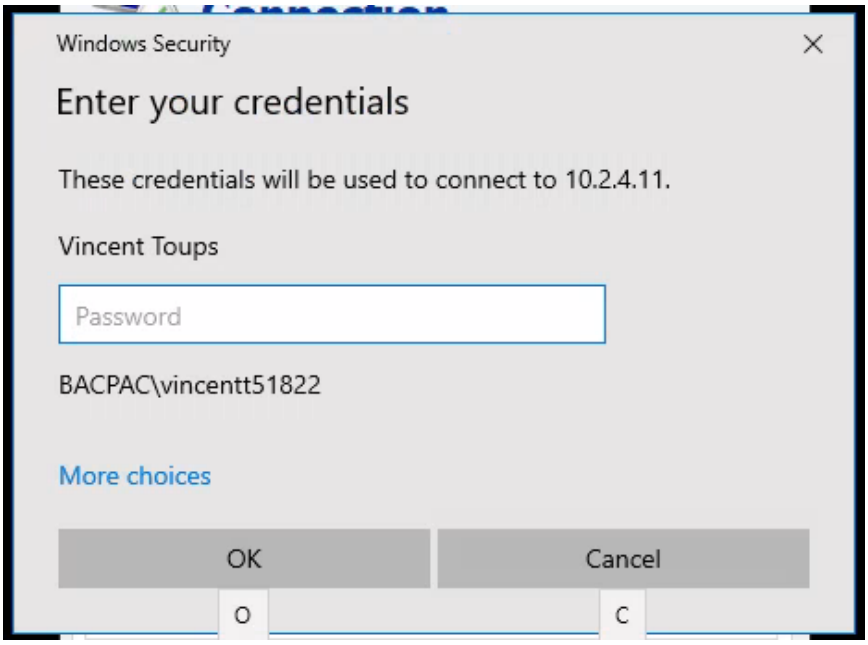


Figure 14: Enter your password again.

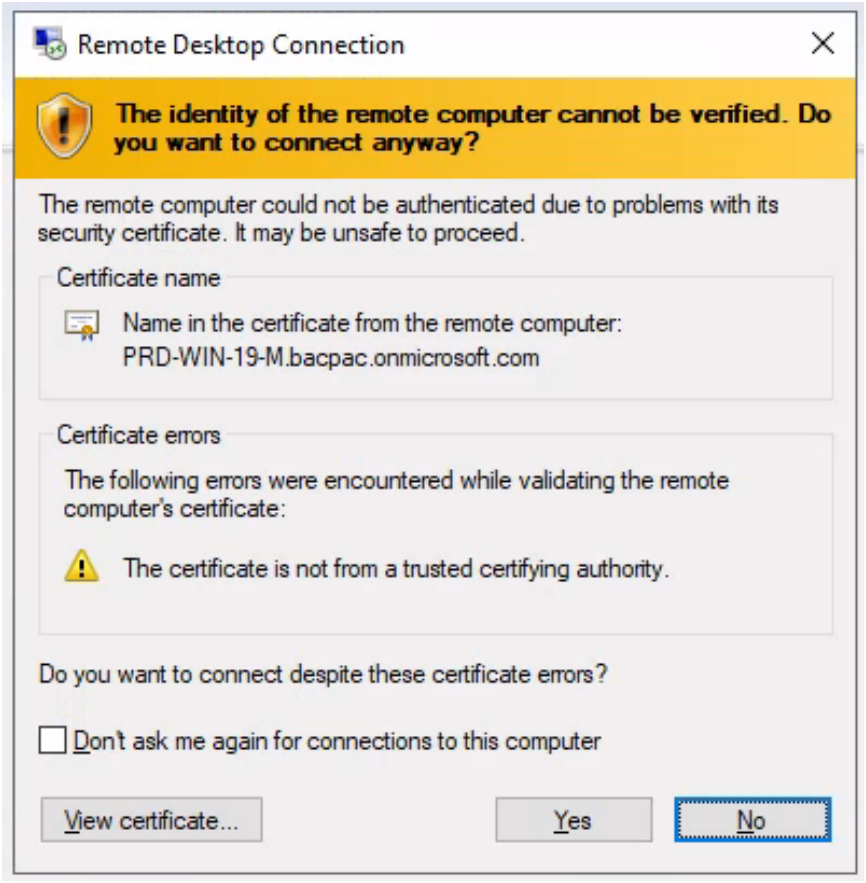


Figure 15: Accept the certificate.

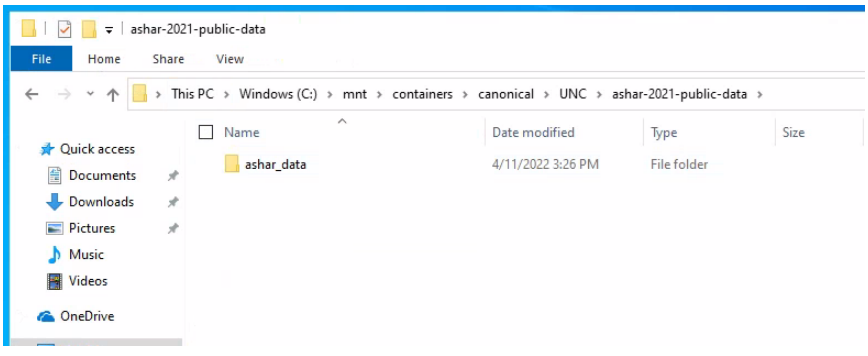


Figure 16: Using pip on Windows to install Jupyterlab.

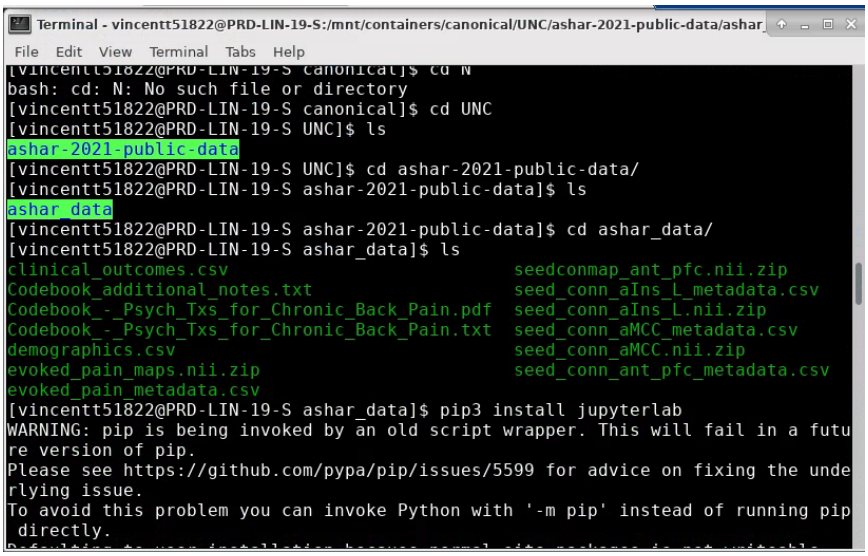


Figure 17: Installing Jupyter Lab on Linux.

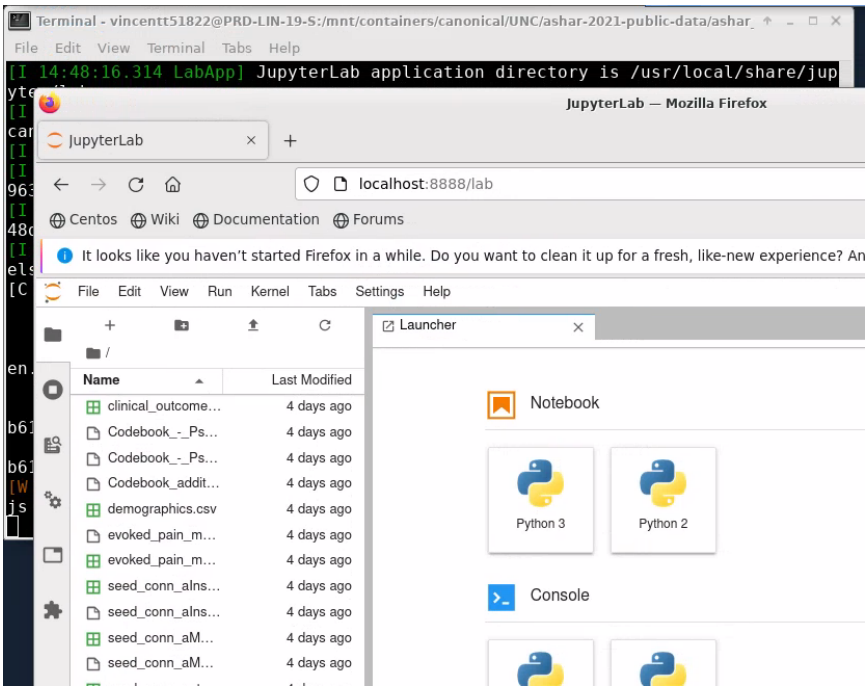


Figure 18: Jupyter Lab up and running.

```
# or
/mnt/containers/project/Workshop_April2022/ashar_data
```

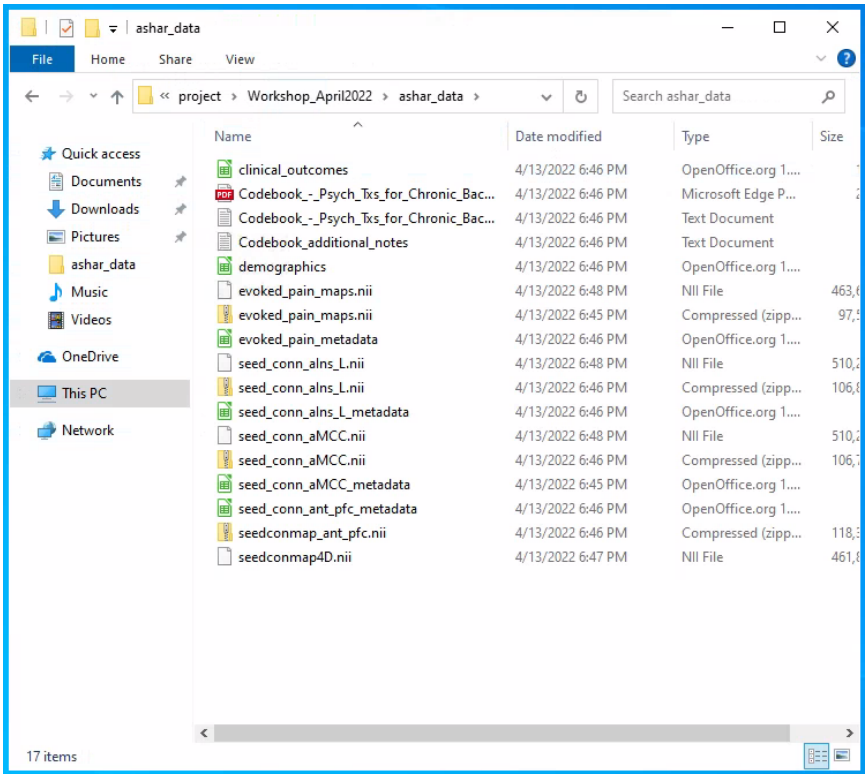


Figure 19: The location of the Ashar et al data set in the project space.

7.3 Doing something w/ the Data

8 "Effect of Pain Reprocessing Therapy vs Placebo and Usual Care for Patients With Chronic Back Pain"

Everyone at the workshop should feel free to explore the data portal with whatever data set they wish, but so that we can all have a common baseline I'll be using data from the paper "Effect of Pain Reprocessing Therapy vs Placebo and Usual Care for Patients With Chronic Back Pain" (Ashar et al, JAMA Psychiatry, 2021).

Very briefly, this paper examines the efficacy of Pain Recontextualization Therapy for chronic lower back pain (compared to saline injections and standard of care).

Patients were split into three groups (Pain Recontextualization Therapy (PRT), Saline Injection and standard of care) and tracked for 12 months. The results are very positive for PRT:

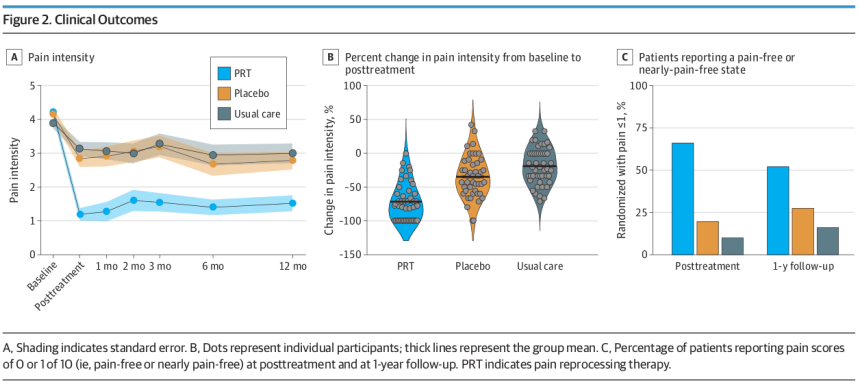


Figure 20: A pretty obvious success story.

Let's take a deeper look at the data.
If we view the folder where our data is located:

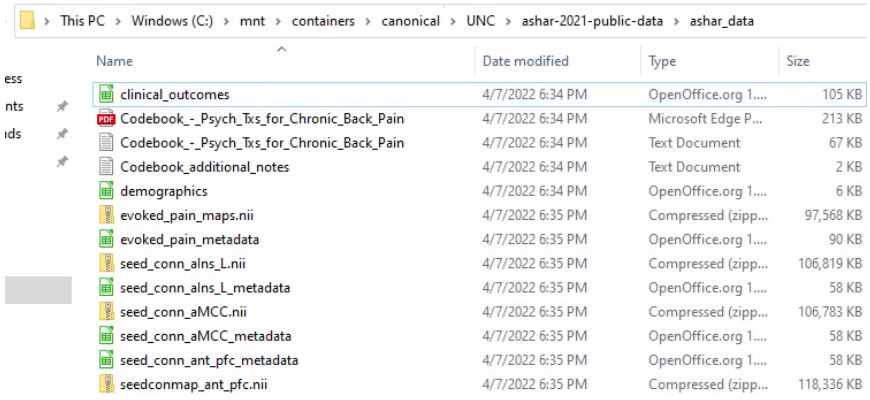


Figure 21: The Ashar data set and associated meta-data.

We can see that the data set comes with a codebook. We'll mostly be working with the "clinical$_{\text{outcomes.csv}}$" data set here, and it turns out that the information about this data set is in "Codebook Additional notes.txt":

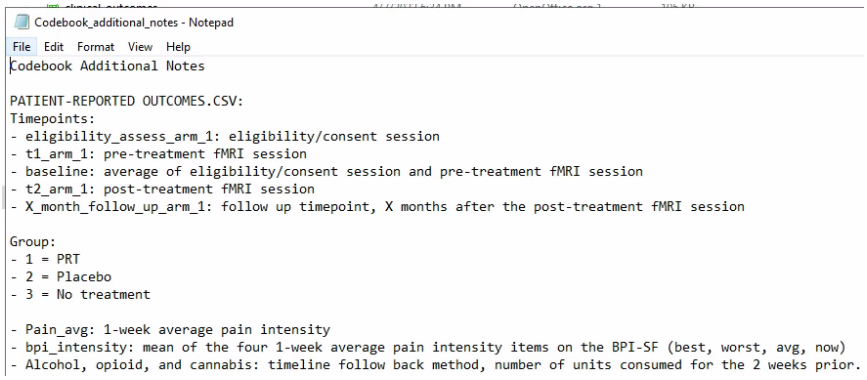
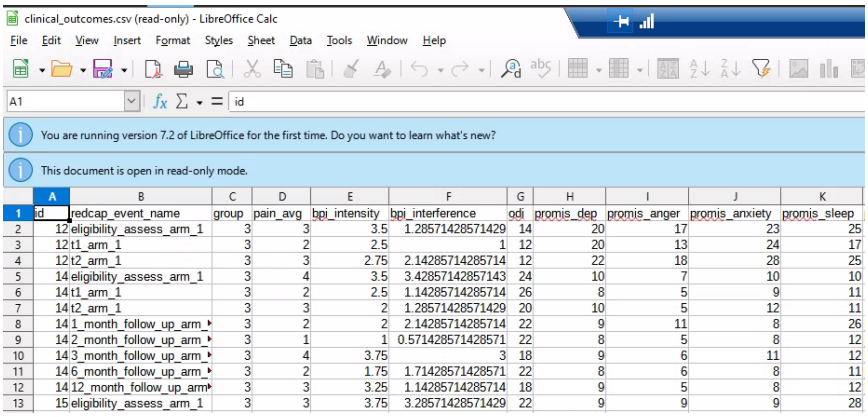


Figure 22: The clinical outcomes codebook.

Let's use Windows and our newly installed Jupyter server to look at our Ashar data set.

We'll be looking at the clinical outcomes data set from that repository.



	A	B	C	D	E	F	G	H	I	J	K
1	id	redcap_event_name	group	pain_avg	bpi_intensity	bpi_interference	odi	promis_dep	promis_anger	promis_anxiety	promis_sleep
2	12	eligibility_assess_arm_1	3	3	3.5	1.28571428571429	14	20	17	23	25
3	12	t1_arm_1	3	2	2.5	1	12	20	13	24	17
4	12	t2_arm_1	3	3	2.75	2.14285714285714	12	22	18	28	25
5	14	eligibility_assess_arm_1	3	4	3.5	3.42857142857143	24	10	7	10	10
6	14	t1_arm_1	3	2	2.5	1.14285714285714	26	8	5	9	11
7	14	t2_arm_1	3	3	2	1.28571428571429	20	10	5	12	11
8	14	1_month_follow_up_arm_1	3	2	2	2.14285714285714	22	9	11	8	26
9	14	2_month_follow_up_arm_1	3	1	1	0.571428571428571	22	8	5	8	12
10	14	3_month_follow_up_arm_1	3	4	3.75	3	18	9	6	11	12
11	14	6_month_follow_up_arm_1	3	2	1.75	1.71428571428571	22	8	6	8	11
12	14	12_month_follow_up_arm_1	3	3	3.25	1.14285714285714	18	9	5	8	12
13	15	eligibility_assess_arm_1	3	3	3.75	3.28571428571429	22	9	9	9	28

Figure 23: The first few rows and columns of the Ashar clinical outcomes data set.

The simplest thing for us to do here is to load the data and make a set of pairplots.

9 Work Time

At this point I'd like to take a half hour to let participants log into the Data Portal, build some VM's, log into them, and at least open the data and

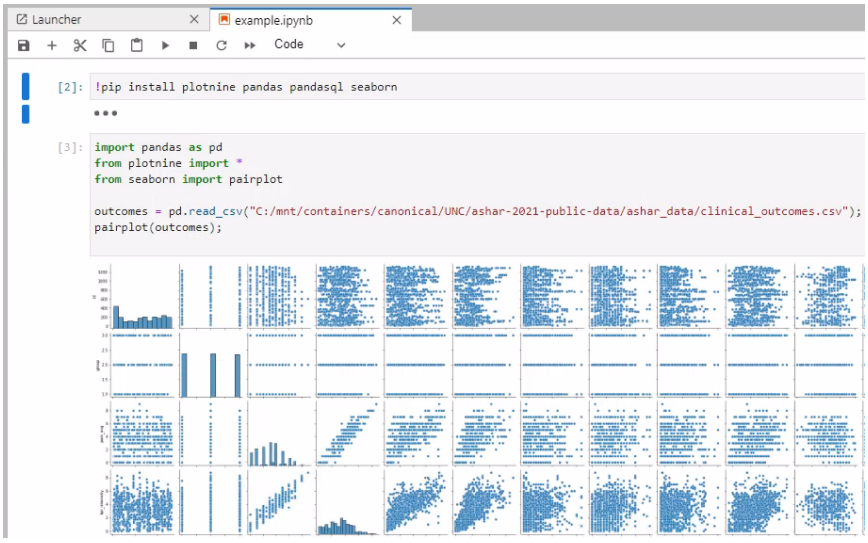


Figure 24: A subset of the pairplots from the Ashar clinical outcomes dataset.

look at it in their preferred environment. I'll be available the entire time for questions. This is also a good time to stretch your legs, make some tea, run a quick mile.

10 A Brief Aside about Ursala

Many years ago I was browsing Reddit (this was before reddit was "cool"/"bad") when I encountered the following post:

You can imagine my confusion when I clicked through and saw someone confidently espousing the value of their own home grown programming language that looked like this:

Ursala, I would later learn, is one of the many descendants of the programming language APL.

Back when people still programmed in Perl, other people used to joke that Perl looked like "line noise" - random characters you'd get if you just send random fluctuations down a teletype terminal. Despite the fact that Ursala looks a lot more like line noise than APL, I was intrigued. In fact, I was already programming in Matlab, which, like R and Numpy, are descendants of APL, and by pulling on the thread that Ursala showed me, I eventually became interested in J.

And now I will pass that brain virus onto you.

Figure 25: Try it out for yourself.

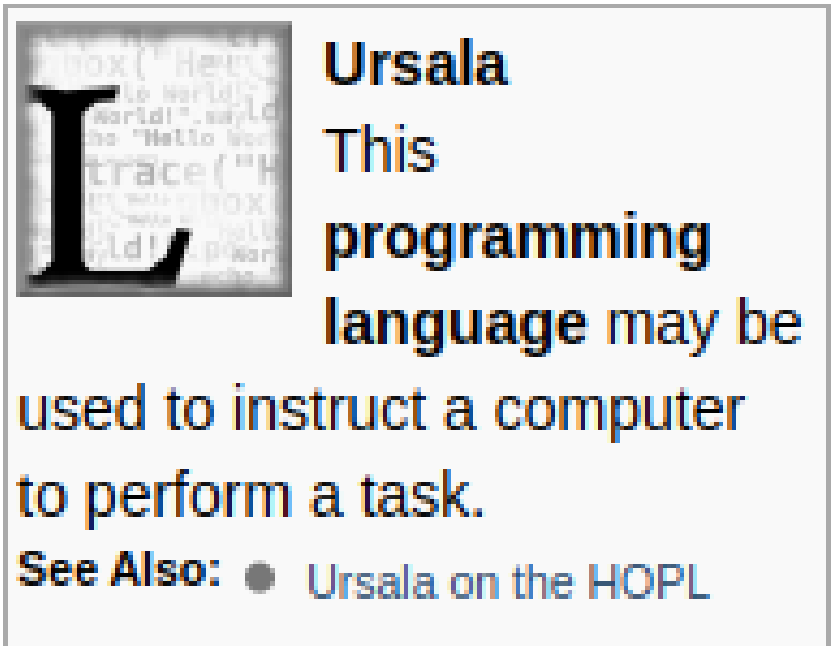


Figure 26: Ursala may be used to instruct a computer to perform a task.

subreddit:
/r/programming

The Ursala Programming Language: More Power, Less Verbiage (basis.uklinux.net)
4 submitted 14 years ago by [deleted]
save

all 24 comments
▸ sorted by: **best**

[–] w00ty 22 points 14 years ago*

8 queens:

```
queens =

%np+~command.options.&h.keyword.&iNC; -+
   ~&iNC+ file$[contents: --<''>+ %nLP*=; * '<'%='[ '+ ','%=', '+ '>'%=' ]']+ ~&rSSs+ nleq-<&l*rFl
   ~&i&& ~&lNrNCXX; ~&rr->rl %tLnLtXLLWXMk+ ^/~&l ~&lrrhrSiF4E?/~&rrlPlCrtPX ~&r; ^|/~& ^|T\~& -+
      -<&l^|*DlrTS/~& ~&iiDlSzyCK9hlPNNXXtCS,
      ^jrX/~& ~&rZK20lrpblPOlrEkPK13lhPK2; ~&i&& nleq$-&lh+-,
   ^/~&NNXS+iota -<&l+ ~&plll2llr2lrPrNCCCCNXS*=irSxPSp+ ^H/block iota; *iiK0 ^/~& sum+-
```

Compare with a sudoku solver:

```
sudoku =

~command.files; <.file$[contents: --<''>]>+ *= ~contents.&F; * -+
   ~&rSL+ (psort (nleq+)* <~&blrl,~&blrr>)+ ~&arg^& -+
      ~&al?\~&ar ~&aa^&~&afahPRPfafatPJPRY+ ~&farlthlriNCSPDPDrlCS2DlrTS2J,
      ^|J/~& ~&rt!=+ ^= ~&s+ ~&H(
         -+.|=&lrr;,|=&lrl;,|=&ll;+-,
         ~&rgg&& ~&irtPFXlrjrXPS; ~&lrK2tkZ2g&& ~&llrSL2rDrlPrrPljXSPTSL)+-,
   //~&p ^|DlrDSLlrlPXrrPDSL(~&,num*+ rep2 block3)*= num block27 ~&iiK0 iota9,
   * `0?=\~&iNC ! ~&t digits+-
```

Thanks, but no thanks.

Figure 27: Ursala - Thanks but no thanks.

Figure 28: An APL Keyboard.

The good ideas in J:

1. tacit programming - If an expression involves only functions (verbs in J) then that function "automagically" composes into a more complex function according to a few rules. This is a little mind bending but its sort of like "data flow" programming - you don't name things you operate on, just think about flow of data.

2. rank - verbs have "rank" (sort of like multidimensional matrices) that you can customize on the fly. This effects the way that verbs operate on the matrices: rank 0 verbs operate on the whole thing, rank 1 on the "1 cells" (eg, in a 1 dimensional array the elements, in a two dimensional array the rows, etc). Using rank you can eliminate almost all looping constructs.

The bad ideas in J:

1. everything else

11 Installing External Software on your VMs

The VM's are totally locked down (with the exception of pip and CRAN and a few other sites). So if you're a weirdo like me and you want to program in J, you have to install this software on your own. The general idea here is:

1. download the installer for your system

2. upload it to your VM

3. install the software there

But in practice making any specific piece of software work the way you want can be a little tricky. Setting up J on the Windows VM is just slightly non-trivial and so serves as a good example case.

The first step is to download J for Windows. If possible, you want to download a "standalone" version of the software you need instead of an installer. Its not uncommon for installers to want to talk to the internet, and this is a non-starter.

If we visit JSoftware we can see what sort of installers we need.

J happens to have a zip "stand alone" type installer which we can grab here:

System/Installation/J903/Zips

< System | Installation | J903

These instructions are for J903 **release** and show how to:

- get base system from a zip or tar.gz release package
- handle security issues - allow downloaded programs to run
- update base, addons, Jqt ide, and JE (jengine)
- create desktop launch icons for user interfaces: jconsole, JHS, and Jqt
- run and play a bit with each of the user interfaces

Following assumes you are installing J in your home folder. Installing in a protected

Platform Specific Steps

Windows [Collapse]

- click and Save File j903_win64.zip
- run Windows Explorer
- navigate to Downloads
- double click downloaded file - be sure to get latest (n) if more than one
- drag j903 folder and drop on C:\Users\Fred
- navigate to C:\Users\Fred\j903\bin
- double click jconsole.exe
- handle any Security Warning

Figure 29: Just a directory of files.

We now just unpack this locally and follow the instructions. J is nice in that you can install literally every package that it comes with in a few minutes:

```
load 'pacman'
'install' jpkg '*'
exit 0
```

So I've done this already on my CSCC Desktop (which runs windows) and then copied it into my VM.

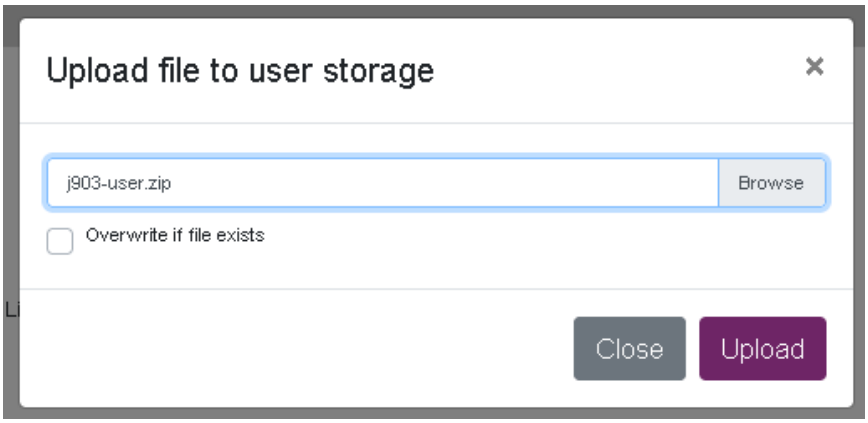


Figure 30: Uploading a zip of my J directories.

Once I've got it into the VM, I can just start J and do some analysis: Believe it or not, this is unusually comprehensible J.

12 git On the Data Portal VMs

12.1 What/Why is Git

Git is a version control system that many people use to organize their software development. It is also very useful for scientific analysis, since it can allow you to maintain a (very granular) history of the analysis code.

The absolute least you need to know to use git are these commands:

```
git init # create a repository
git add <filename> # tell git you would like to record changes to
# this file in the next commmit
```

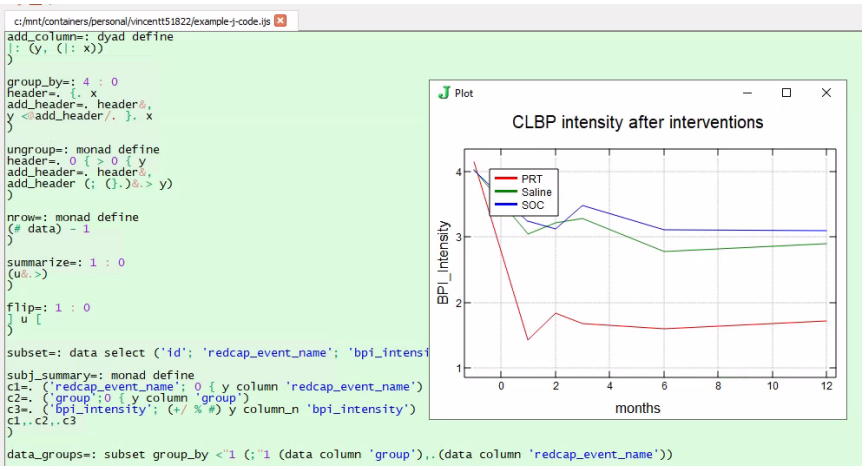


Figure 31: Putting in error bars was more pain than I was willing to endure.

```
# you may make several "git adds" before a ...
git commit -m "I changed a file (ideally more explicit commit message)"
# record the change you "staged" above.
git push # send your commits to a remote repository
git fetch # fetch commits (perhaps made by others) into your local repository
git rebase <remote-name>/main
# try to integrate the fetched commits just fetched into your own code in a "polite" wa
```

I'd encourage everyone to use git if they working with code on the BACPAC Virtual Machines. Most people who want to share code these days use github for sharing repositories, but github is blocked by our firewall.

The most reasonable way for users to share a common git repository is to create a repository on their shared project space. This can be done from either Linux or Windows and only needs to be done once:

```
cd /c/mnt/containers/project/Workshop_April2022/
git init --bare --initial-branch main my_cool_project
```

The "lead" of the project will typically initialize such a repository. That person, as well as everyone else who will contribute to the project, will clone this repository into their own personal space.

That looks like this:

```
cd /mnt/containers/personal/vincentt51822/
git clone /c/mnt/containers/project/Workshop_April2022/my_cool_project
```

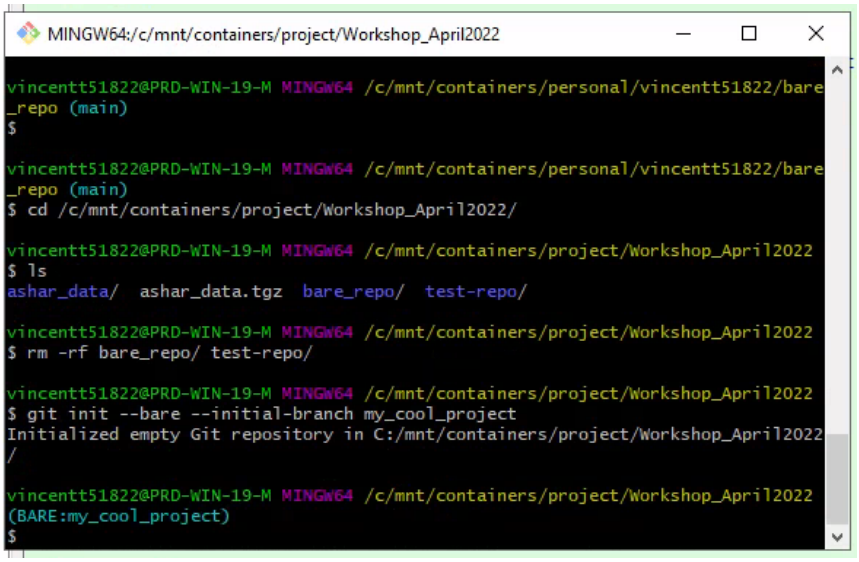


Figure 32: Initializing a bare git repository on Windows.

Once we've cloned the repository we can add some stuff to it and make a commit.

```
cd my_cool_project
touch README.md
git add README.md
git commit -m "I added (an empty) README."
git push
```

Now let's pretend to be someone else working on the project. Anyone else in the project will be able to access the copy of the repository in the project folder. Another user would do this from their VM but we'll just go to another directory to demonstrate.

```
cd /c/Users
git clone /c/mnt/containers/project/Workshop_April2022/my_cool_project
```

```
cd /tmp
git clone /c/mnt/containers/project/Workshop_April2022/my_cool_project
cd my_cool_project
ls
README.md
```

<do some more demo>

13 What (and why) is Docker?

Docker is a tool from the system's administrator's toolbox. It was invented to solve at least one basic problem:

Software doesn't run in a vacuum. Any given program may depend on the context it runs in (the operating system, libraries, environment variables, network configuration) in non-trivial ways. Developers used to solve this problem by either haphazardly, by manually configuring machines for development and production for each user and deployment, or by writing complicated scripts.

When virtualization became a thing, this process was made a little easier: you could snapshot pristine VMs at just the point you needed them and revert or copy snapshots around to smooth out these processes.

But virtual machines are relatively heavy-weight. Docker is a container platform which gives you some of the fleetness of simply running a program and most of what you'd get from maintaining virtual machine images. It also standardizes the process of setting up virtual machines so that VM's can be shared as built images or just files which describe the processes.

14 Docker for BACPAC Machines

The utility of Docker containers for BACPAC researchers working on the virtual machines is this:

if you can set up your workflow on a Linux machine using pretty much arbitrary software, then you can build a Docker container on your local machine, upload it the BACPAC Data Portal, and run that workflow on a VM with access to the BACPAC Data.

We have tried to anticipate your software needs, but if we failed, you can build a Docker container to get you where you need to go. Please let me know if you need any help getting a Docker container for a particular use set up. It is part of what I am here for.

Docker is a large, relatively complex, ecosystem of software. We can only scratch the surface here, but the surface is often all a Data Scientist or Analyst needs to exploit Docker effectively.

15 Brief Introduction to Docker and Dockerfiles

A Dockerfile is just a textual-representation of the steps a user would perform in order to configure a particular operating system for work.

Here is a simple example of a Docker file:

```
FROM rocker/verse
RUN apt update -y && apt install -y scilab
CMD scilab -nw
```

Figure 33: A very simple Dockerfile called scilab.Dockerfile

We can build an image based on this Dockerfile like this:

This Dockerfile installs the scientific computing environment "scilab" and starts a command line interpreter when run. Note that the "RUN" lines indicate something to do on the container as part of a setup process. You will have many RUN lines in general. The CMD line says "when someone runs this container, do this."

```
docker build . -f scilab.Dockerfile -t scilab
```

Figure 34: Building the Docker container.

And then we can run it:

```
docker run -it scilab
```

Figure 35: Building the Docker container.

And this will start an interactive scilab shell "inside" the container.

Its important to realize that the container isolate you from the operating system - by default, this scilab can't see any files outside of the container.

If you have a very mature workflow, you may package source or executables that perform an analysis directly into the container:

```
FROM rocker/verse
RUN apt update -y && apt install -y scilab
COPY hello.sl /
CMD scilab -nw -f /hello.sl
```

Figure 36: A very simple Dockerfile called scilab.Dockerfile

```
docker build . -f scilab2.Dockerfile -t scilab2
docker run -t scilab2
```

Figure 37: Building the second Docker container.

But it is much more common among Data Scientist to simply mount your local directory as well as any data files in the container when you run it.

```
docker run -v $(pwd):/home/rstudio/work -u rstudio -w /home/rstudio/work -t scilab
--> exec("hello.sl") # note that this runs the version of hello.sl in our working direc
```

Figure 38: Mounting our current directory *inside* the container.

If we were running our Docker container on our Virtual Machine, we would also want to mount the container directory, like this:

```
docker run \
       -v /mnt/containers/project/Workshop_April2022/ashar_data:/mnt/containers/project
       -v $(pwd):/home/rstudio/work\
       -u rstudio -w /home/rstudio/work -t scilab
--> exec("hello.sl") # note that this runs the version of hello.sl in our working direc
```

Figure 39: Mounting our current directory *inside* the container.

Speaking of which - how can we get our containers onto the VMs?

15.1 A few technical notes

1. You will need to log out and in again once before your user is able to run Docker.

2. Sometimes volumes mounted inside a container will have the wrong permissions. You'll need to change those permissions in order to modify the contents of those directories.

Both these issues will hopefully resolved in a future update.

16 How to get your Docker container onto the VM

In the near future we will enable a Docker registry accessible from inside and outside of the Virtual Machines. When that happens you will be able to push an image you have built locally to the registry and pull it down inside your Virtual Machine.

In the meantime, however, the easiest solution is to export your container after building it and upload it to your personal space. I'll use a very small image as an example, since otherwise the upload process can be challenging.

```
docker pull hello-world
docker save hello-world:latest | gzip > hello-world.tar.gz
```

Figure 40: exporting and zipping up a Docker image.

Figure 41: Click the upload arrow in My Workspace.

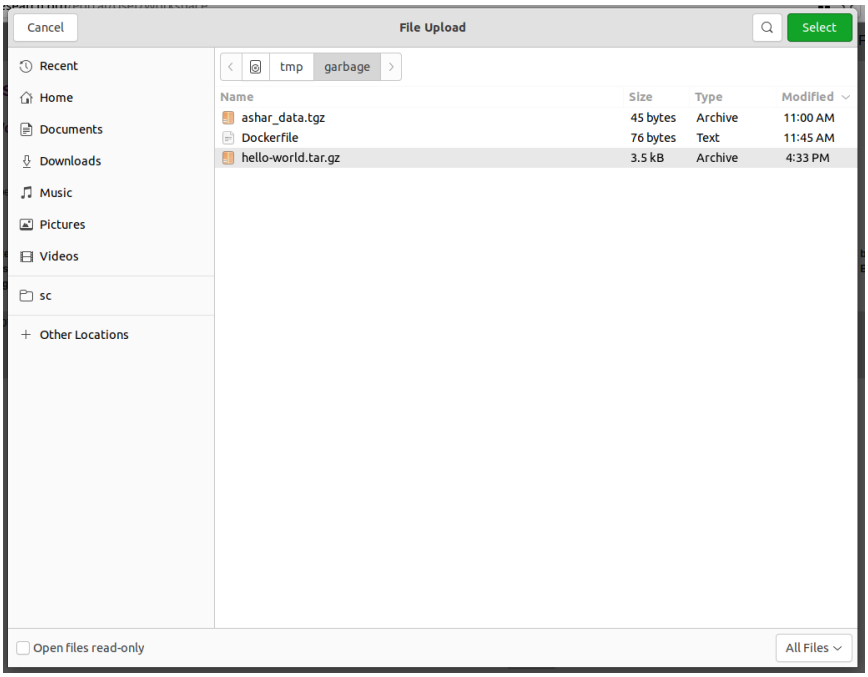


Figure 42: Select the hello-world.tar.gz image.

On the virtual machine we simply write:

```
gunzip hello-world.tar.gz | docker image load
docker run -t hello-world
```