User's Guide

This guide is written for non-experts looking to use the foundational model for transfer learning for their own tasks. We will begin with the basic installation and move into an example of how to use the foundational model for Buchwald-Hartwig yield prediction. The model was trained on Linux OS and we highly recommend that users do the same.

Prerequisites

Installation of Conda:
A good rule of thumb when working with code in Python is to use virtual environments for your projects. This will allow you to install the exact packages you need without worrying about conflicts from other projects / previous installations. We will be using conda virtual environments. Official instructions for installation can be found here: https://docs.conda.io/projects/miniconda/en/latest/. Please follow the instructions according to your operating system.

Installation of Git (Optional):
The program used to interface with GitHub is git. Although not a requirement for getting the model from the GitHub, it is a very useful tool in managing code. Official instructions for installation can be found here: https://github.com/git-guides/install-git. Please follow the instructions according to your operating system.

Getting the Code

We will show you the commands used (highlighted in yellow) and the associated output (the line after the command is run) to assist users in using command line inputs and our model. Commands can be run by pressing the "return" key. Within this text, commands will be indicated with `this font`. These commands are to be implemented in your computer's terminal / console.

Create a New Directory:
A directory can also be referred to as a "folder". For the remainder of the document, we will be using the term directory but you may think of them as folders. To access our code, two methods are possible. The first uses git, the second goes through GitHub.com's web-interface. We will go through both.

Start by making a new directory called "transfer_learning" and move into the directory. This can be done by typing:

```
mkdir transfer_learning
```

There will be no output from this command.

Move into the directory with:

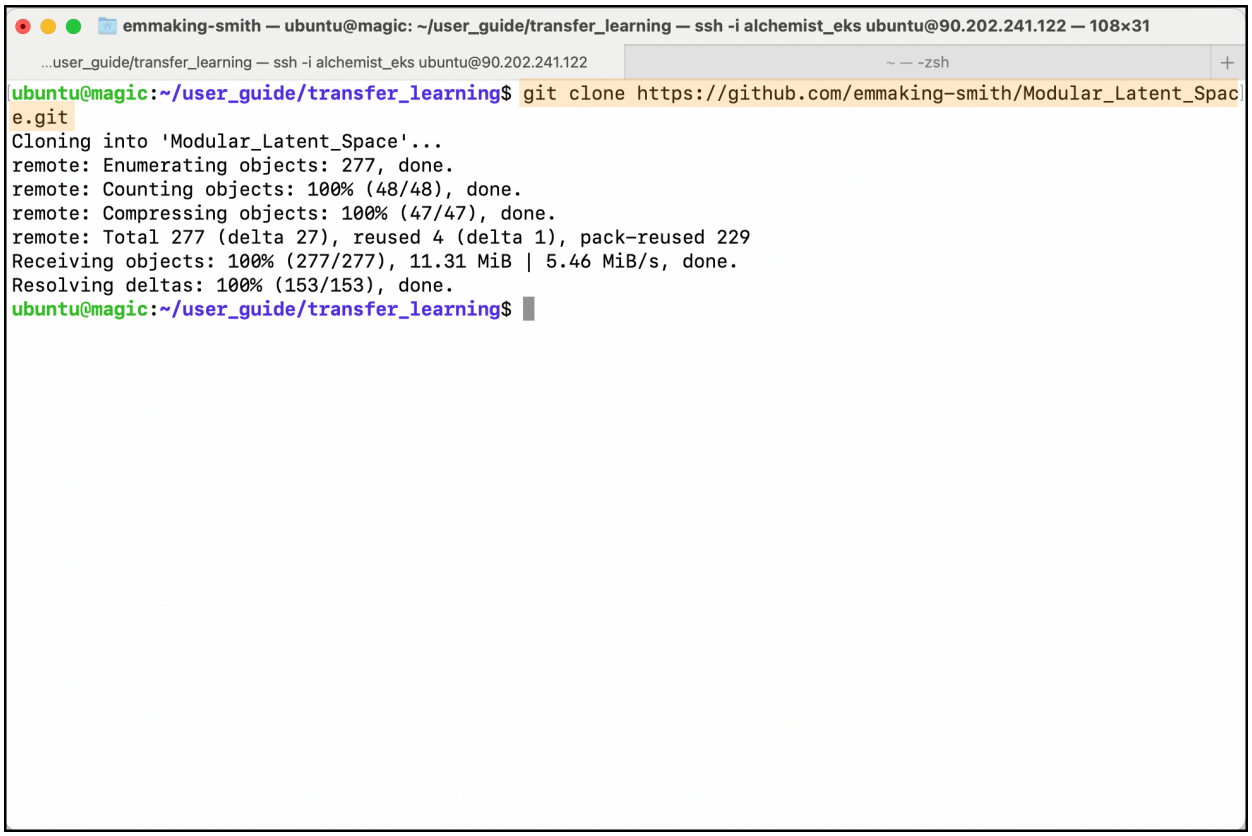
```
cd transfer_learning
```

We are now ready to download the code for Transfer Learning for a Foundational Chemistry Model. You have two options. Option 1 requires the installation of git and Option 2 does not. Please choose **either** Option 1 or Option 2.

Option 1 - Download the GitHub Repository with Git (Git installation required):

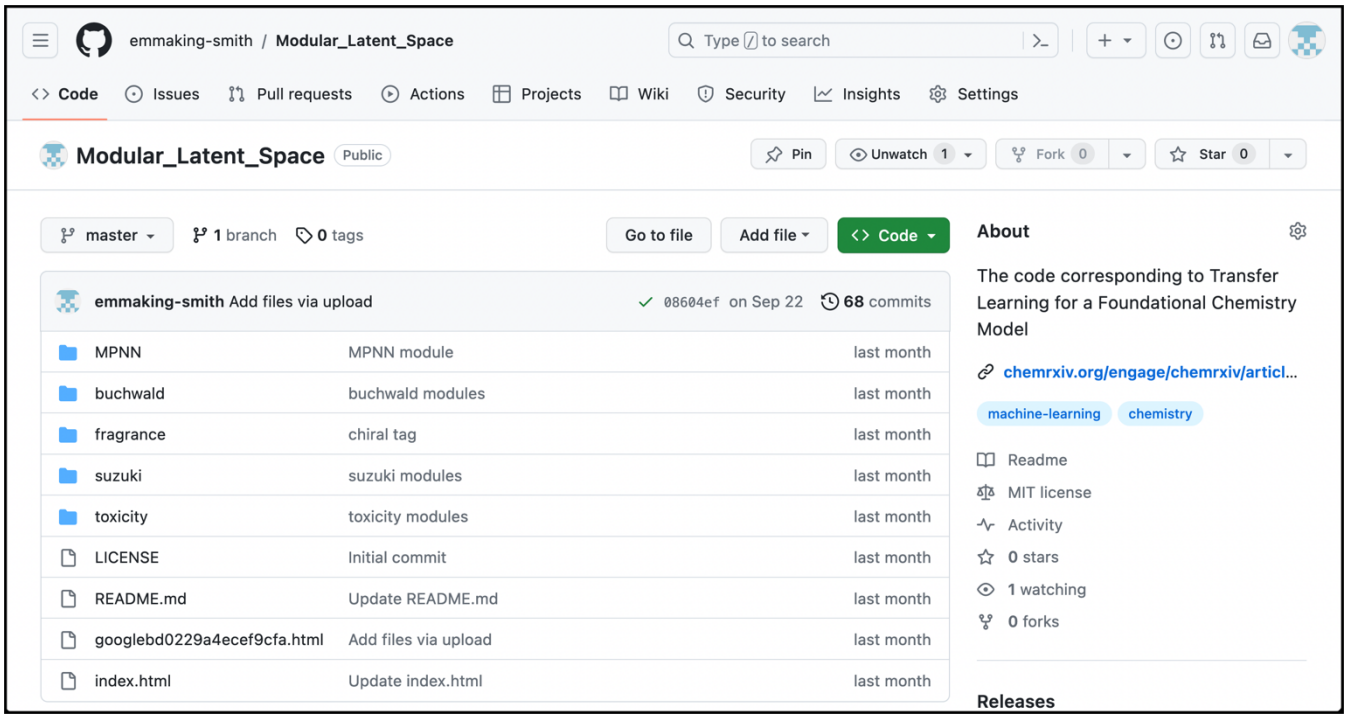
Clone the repository. This allows you to instantly download all the information off of the repository to your new directory. Type:

```
git clone https://github.com/emmaking-smith/Modular_Latent_Space.git
```

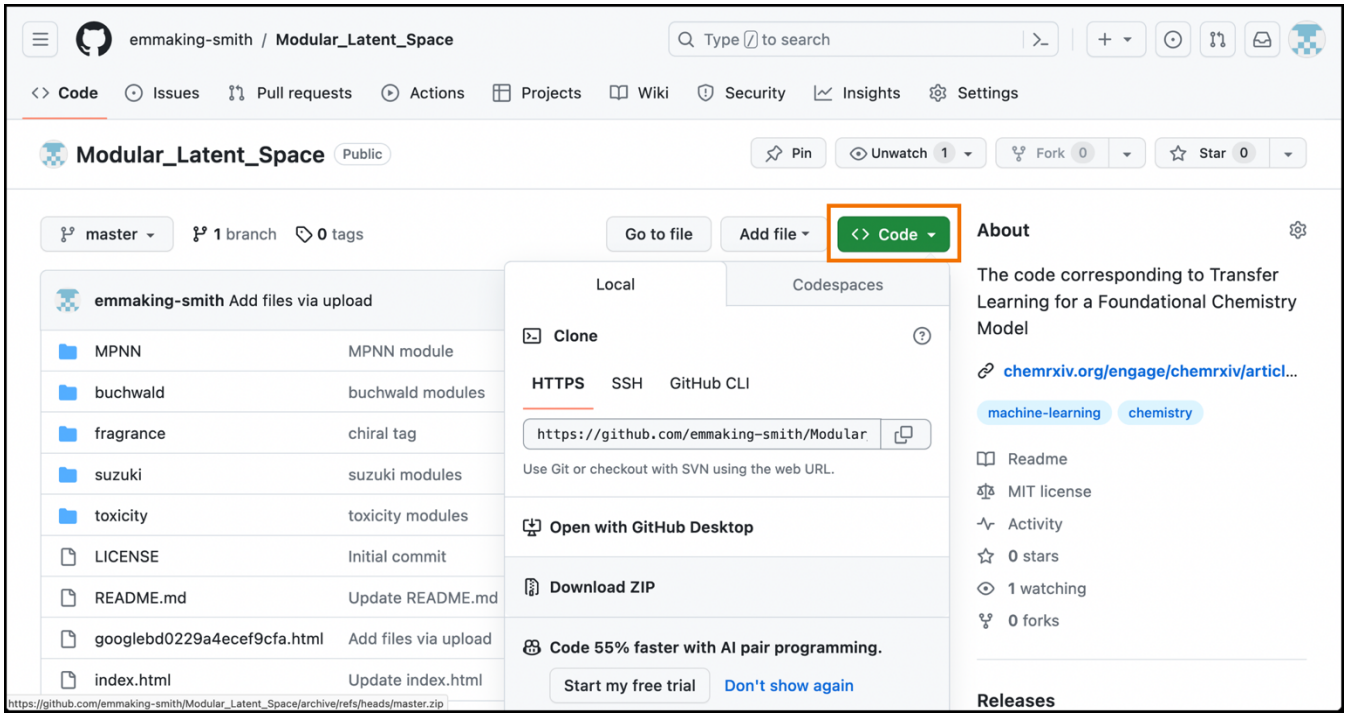


Option 2 - Download the GitHub Repository from GitHub.com:

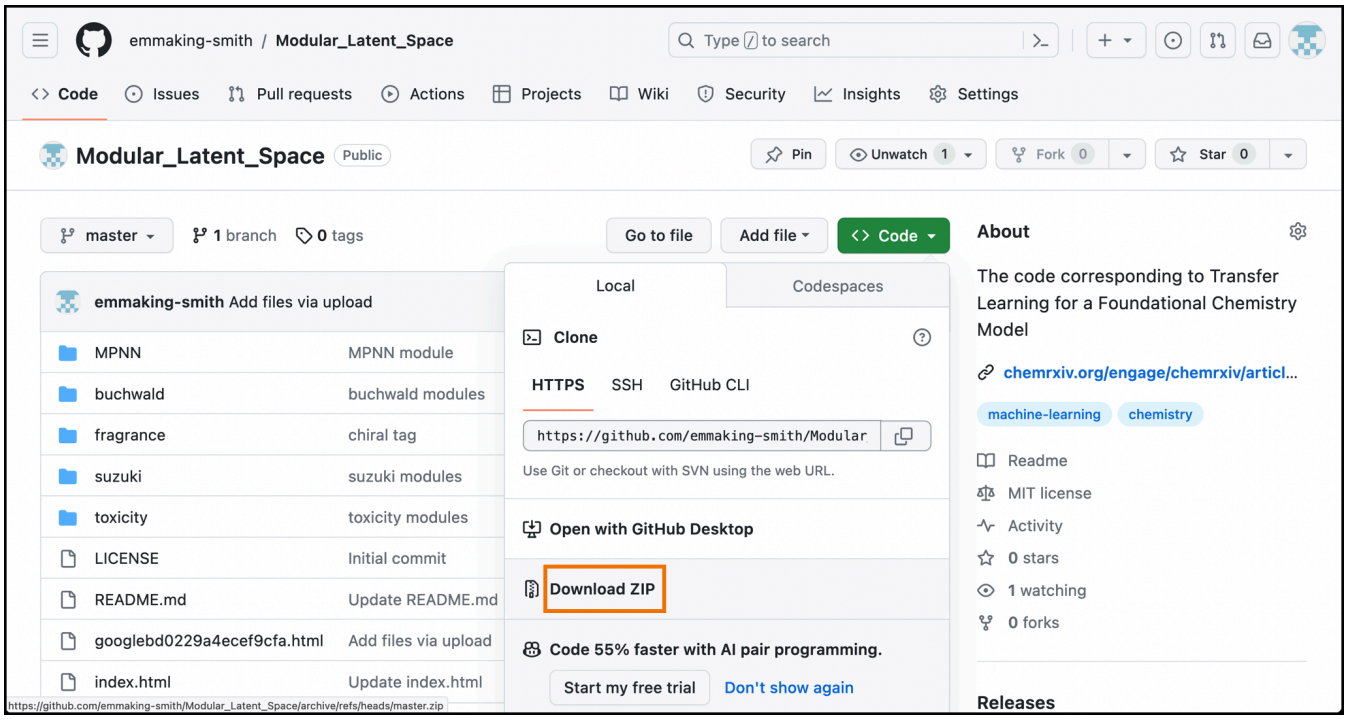
If you do not want to install git, you may download all the code from the GitHub website.
Go to https://github.com/emmaking-smith/Modular_Latent_Space.



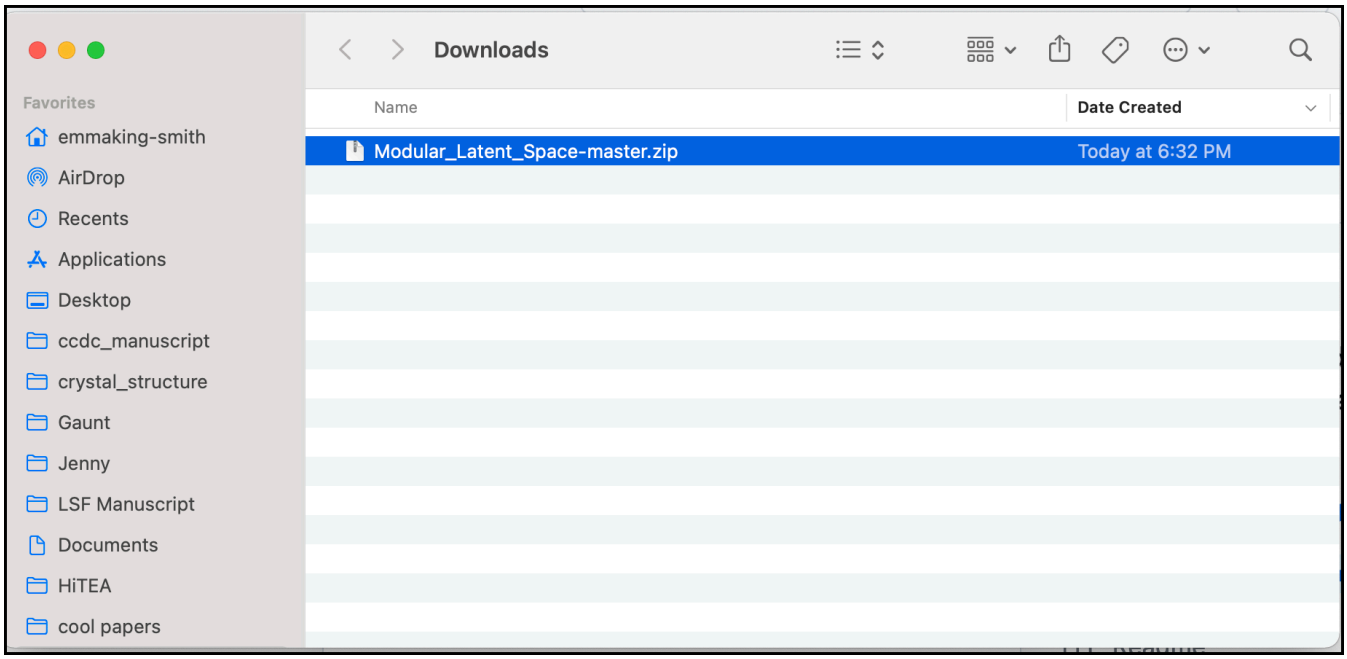
Click on the green "Code" button.



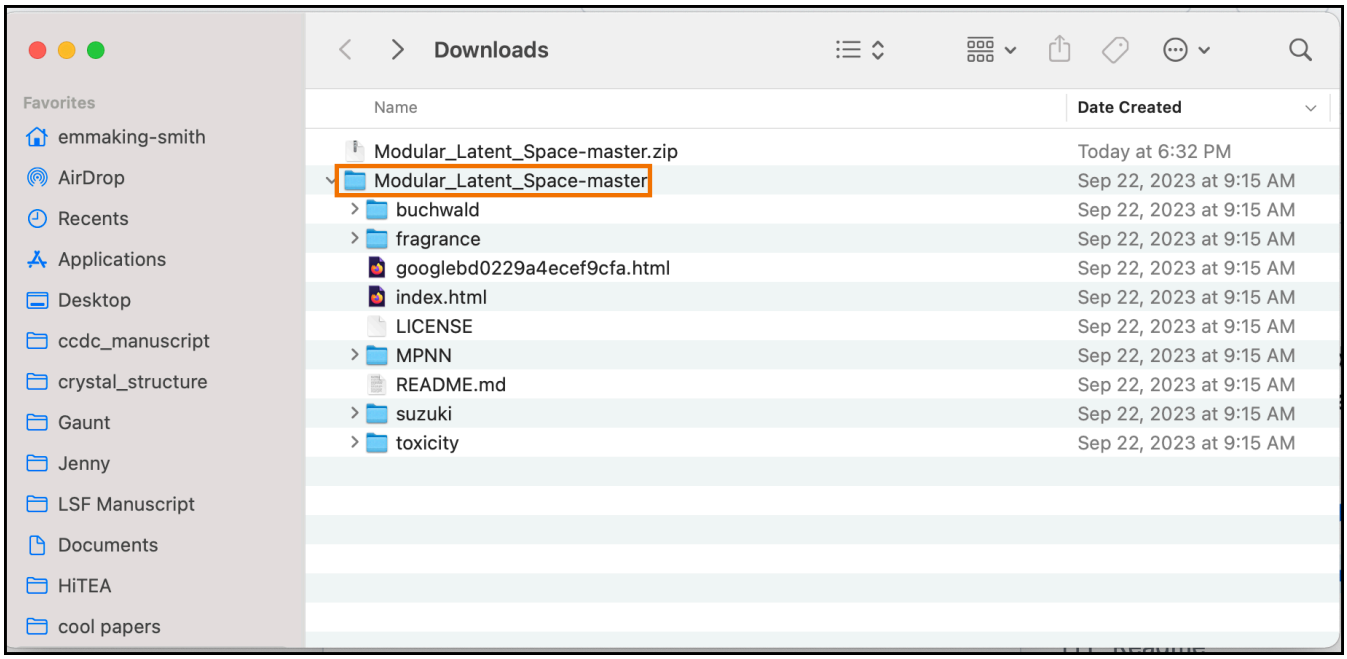
Select "Download ZIP"



Unzip the zip file. This can typically be achieved by double clicking on the file.



Move the unzipped directory, Modular_Latent_Space-master from its current directory to the directory we created at the beginning of this section (we named it transfer_learning). Drag and drop is the easiest way to do this.



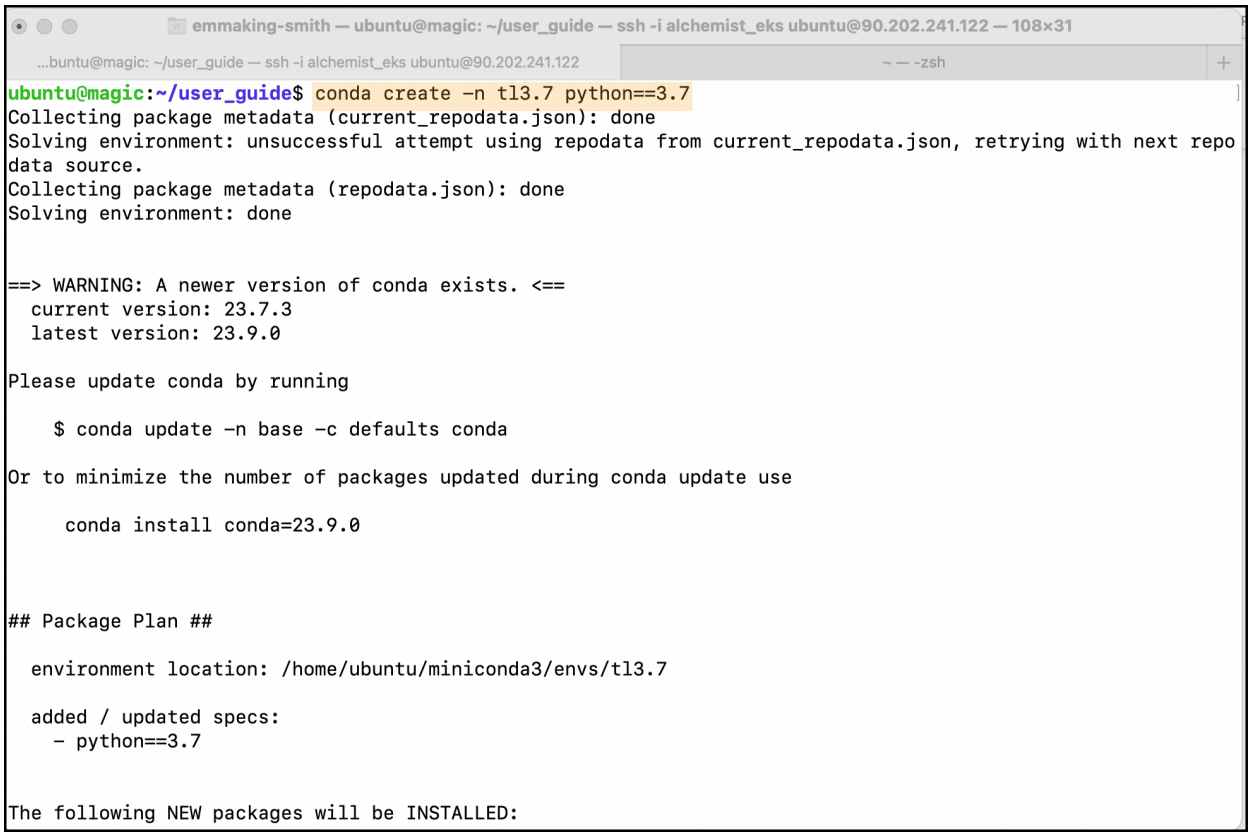
Setting up the Virtual Environment

Here, we install the necessary packages to run the transfer learning. The necessary packages can be found at https://github.com/emmaking-smith/Modular_Latent_Space at the bottom of the page under "Dependencies".

First we create our virtual environment that will be used to run all of our code. Note that you must be within this virtual environment to run the code without errors. To do this, we run the following command:

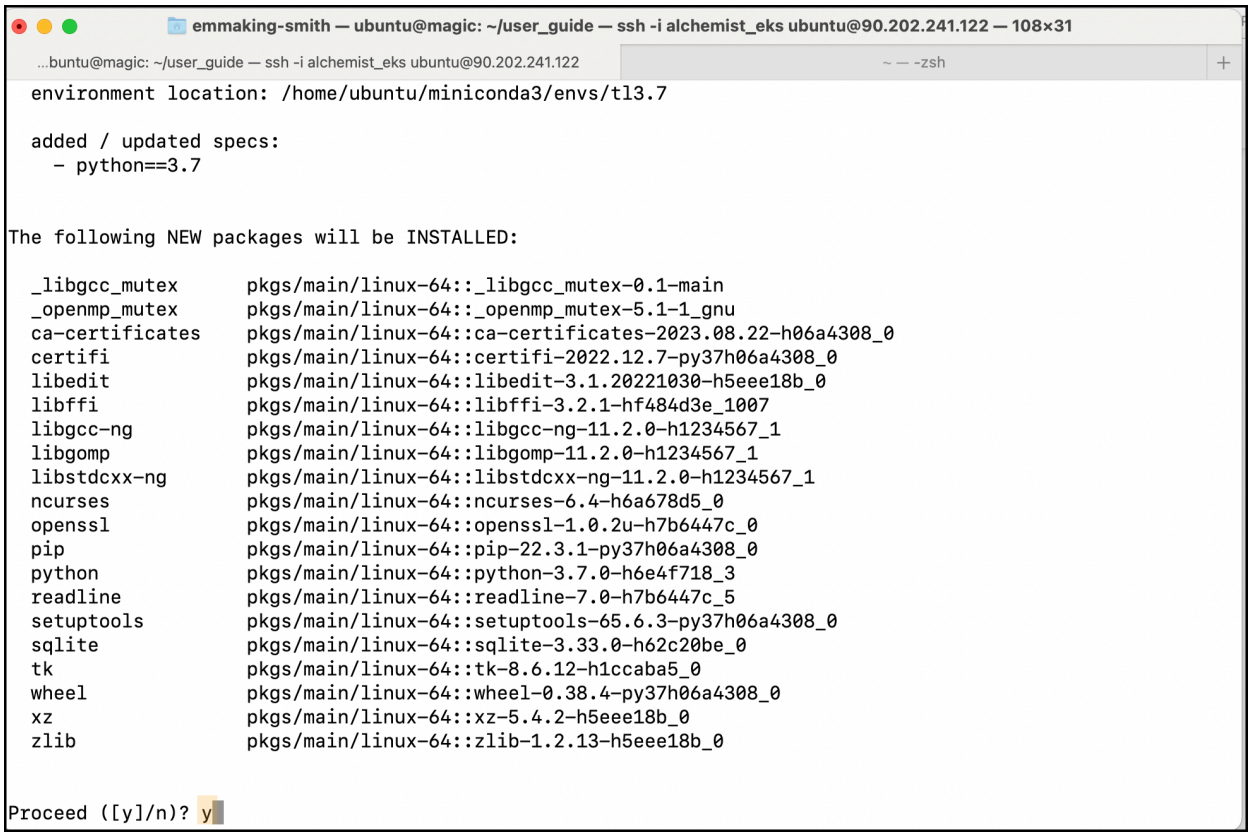
```
conda create -n tl3.7 python==3.7
```

Our virtual environment's name is tl3.7 (transfer learning python version 3.7). We designate the python version with the suffix of "`python==3.7`". **It is critical to use the correct version of python when attempting to use any other programmer's code.**

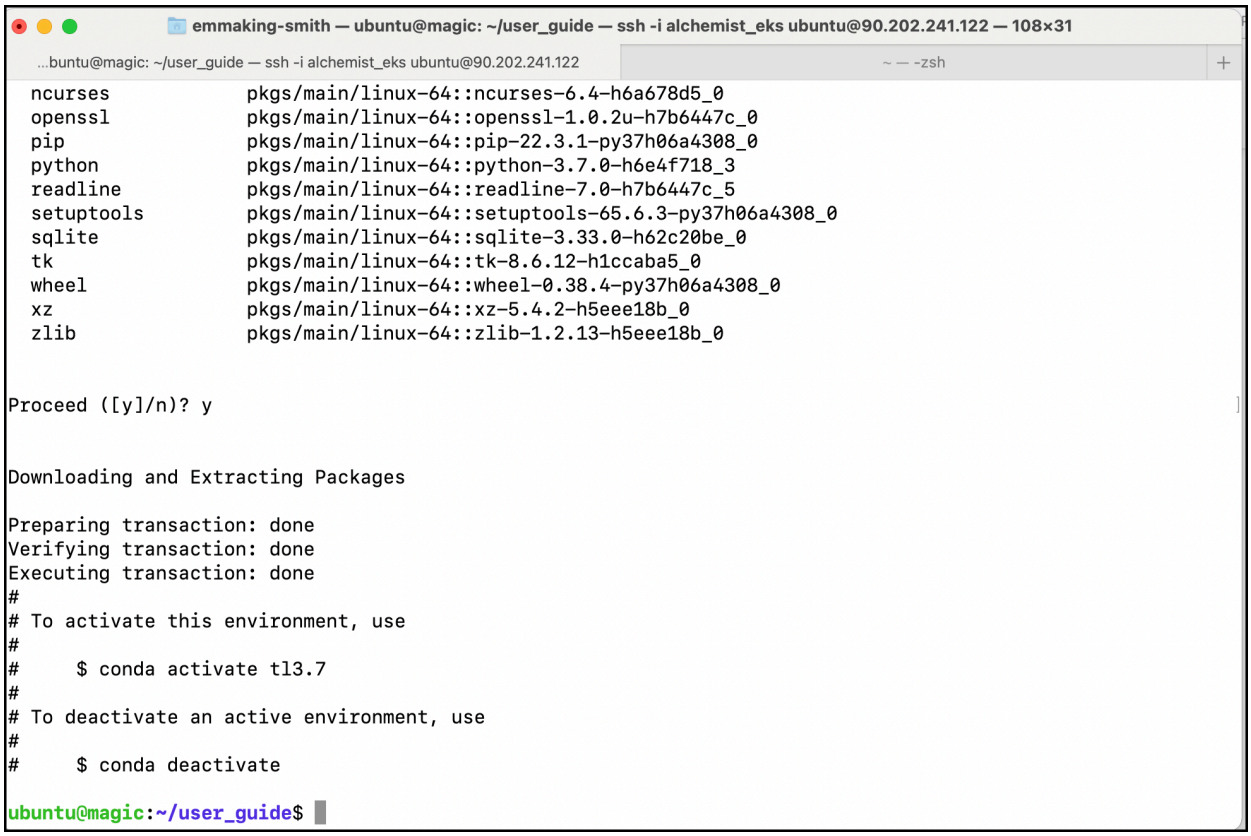


Shortly thereafter, the console will prompt you to accept the installation of new packages. Type:

y



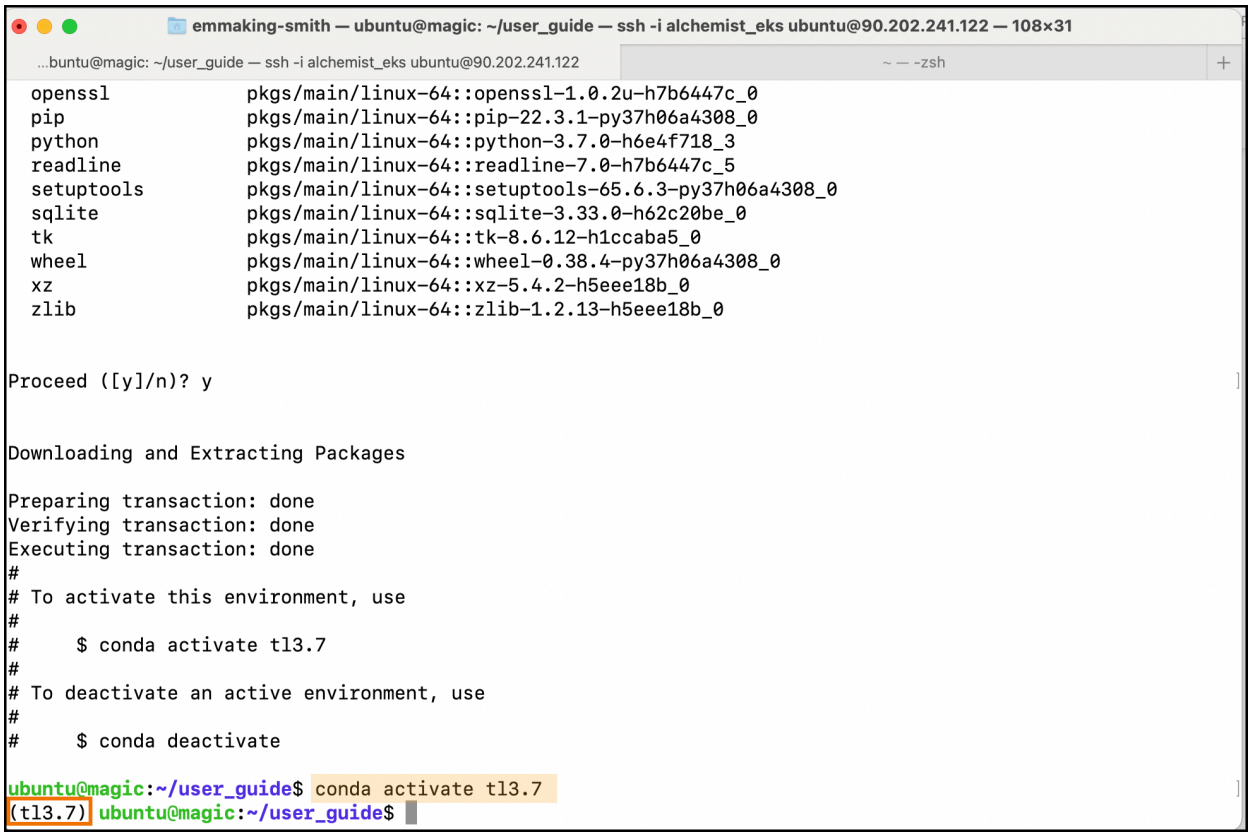
The output of these commands will look something like this:



We then activate our virtual environment.

```
conda activate tl3.7
```

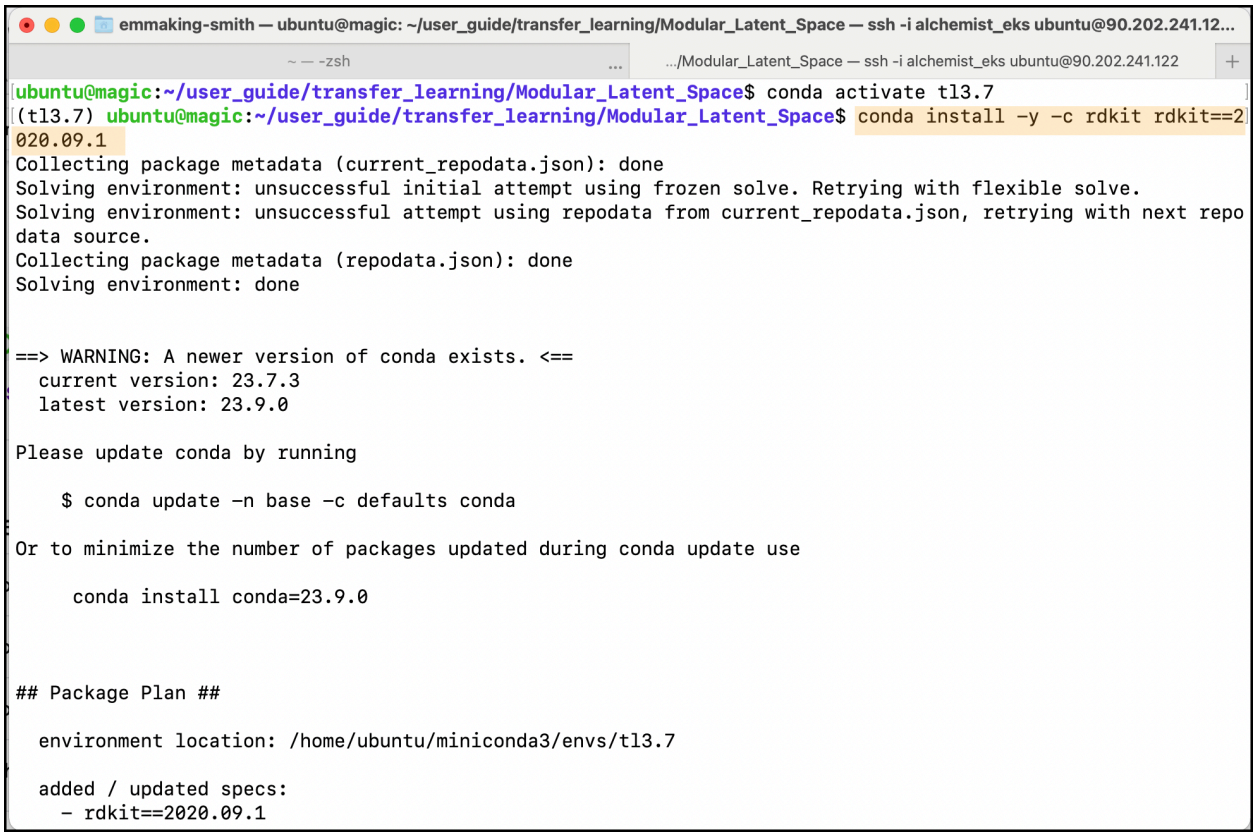
You can tell that you are in a virtual environment by the leftmost text, which now says the environment name (see orange box).



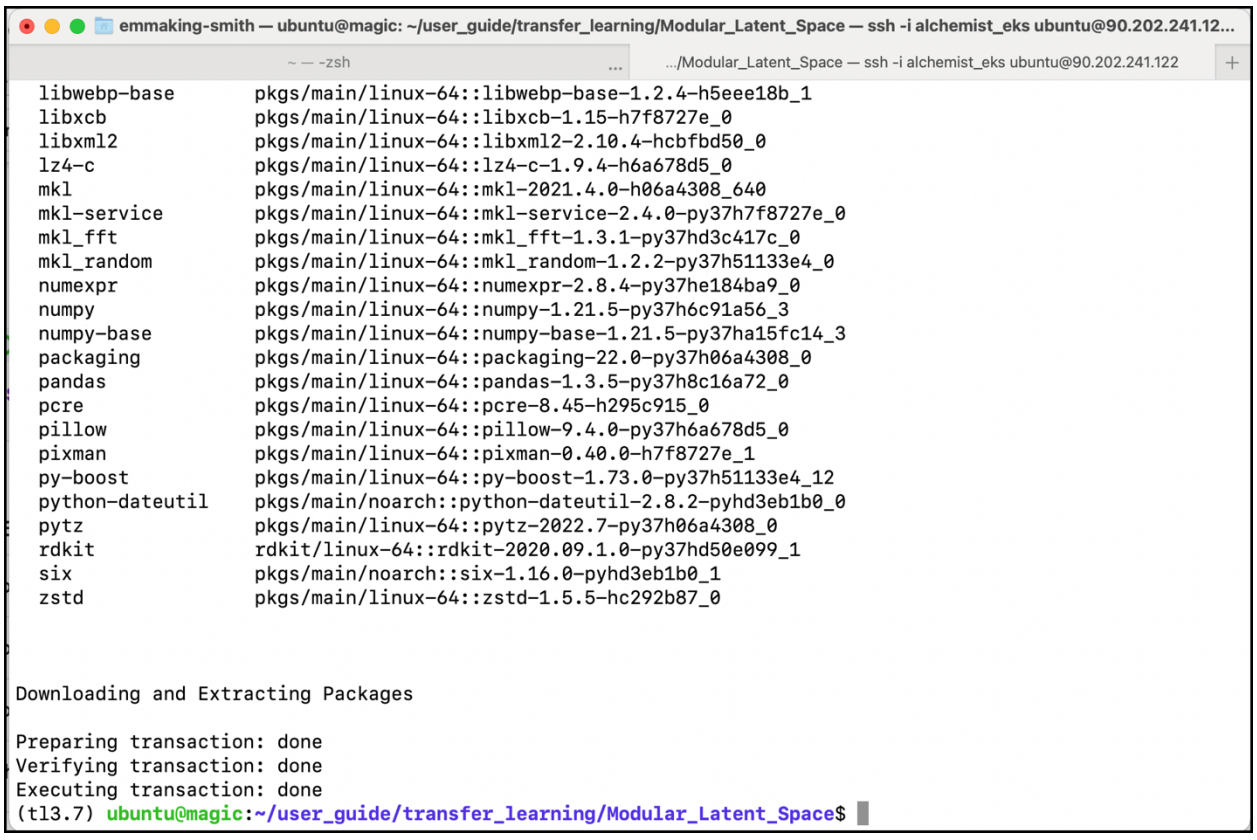
Package Installation

Next, all the relevant packages of specific versions will be installed. We specify this with the "==" sign. RDKit will be installed first.

```
conda install -y -c rdkit rdkit==2020.09.1
```

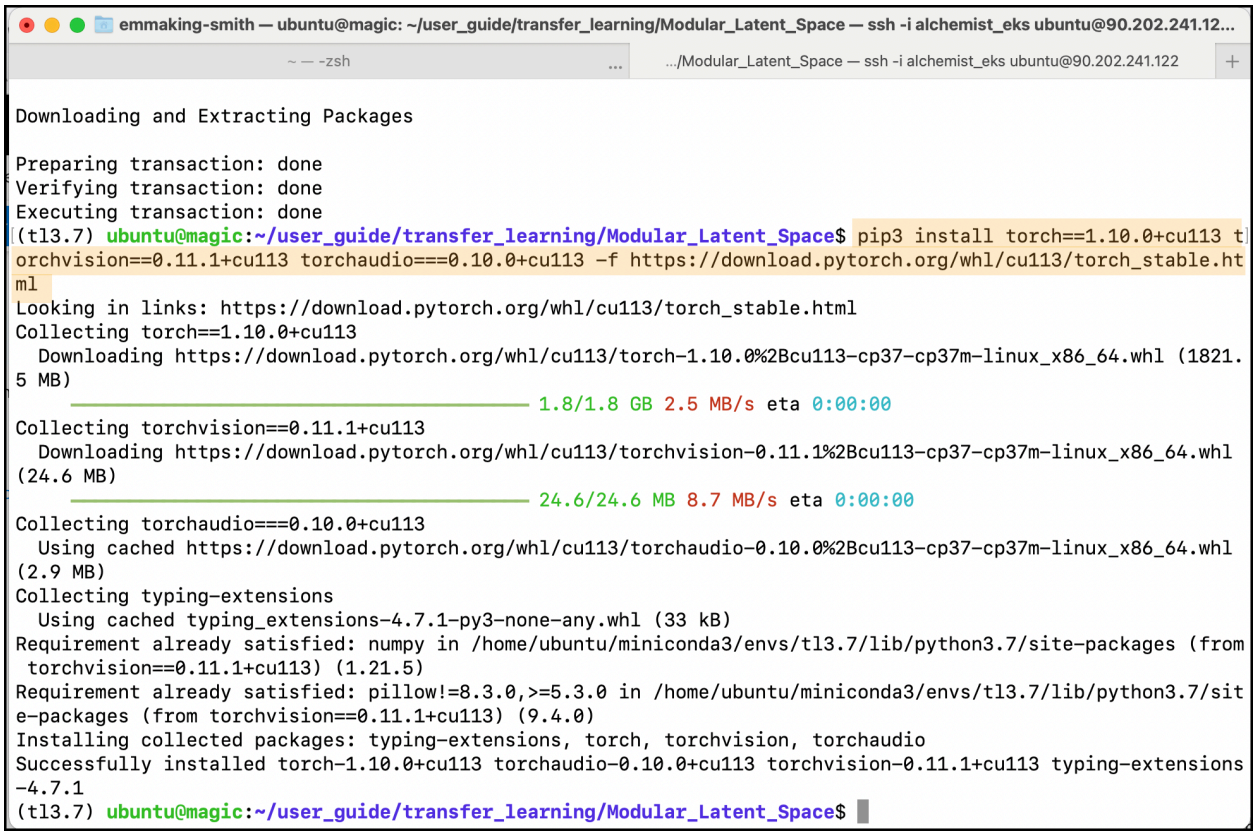


You will see many packages being installed. The final output will look something like this:



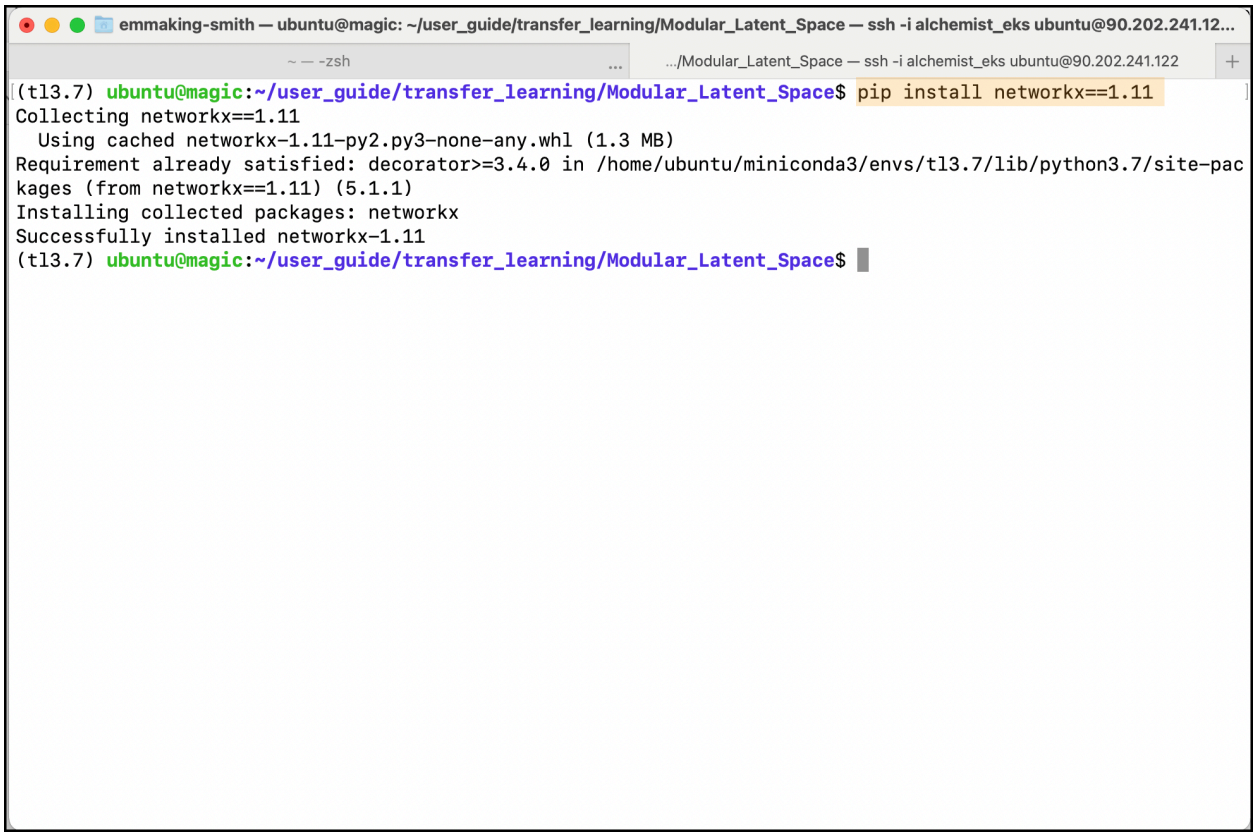
Pytorch and other useful torch packages are installed next with:

```
pip3 install torch==1.10.0+cu113 torchvision==0.11.1+cu113
torchaudio===0.10.0+cu113 -f
https://download.pytorch.org/whl/cu113/torch_stable.html
```



Then networkx is installed with:

```
pip install networkx==1.11
```



This is the most critical package to get the correct version. After version 1.11, the graph nomenclature changed and attempting to run the code with later versions will result in an error. **If you believe you are seeing a networkx error, please double check that the version you are running is 1.11.** This can easily be done using the following commands.

Start python - make sure you are in your virtual environment. Type:

```
python
```

Import networkx with:

```
import networkx
```

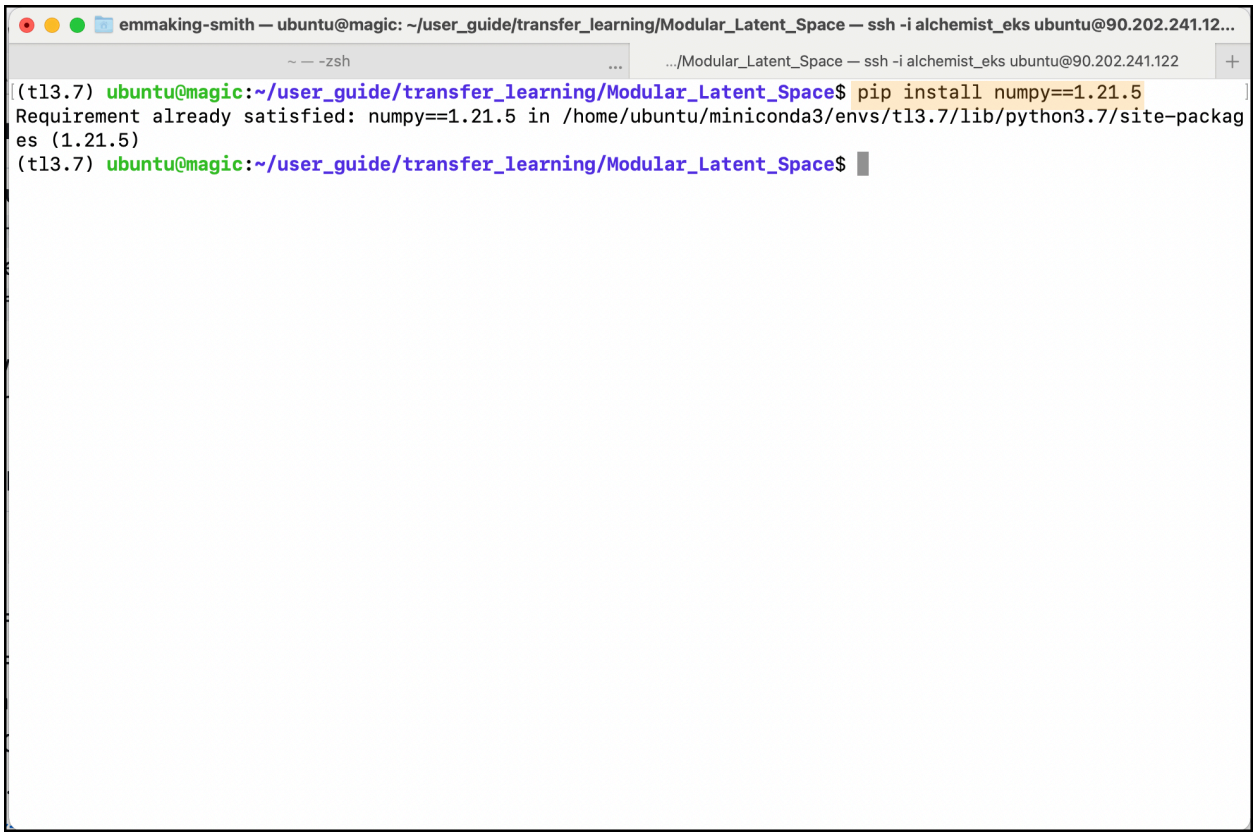
Check version with:

```
networkx.__version__
```

You should have an output of 1.11.

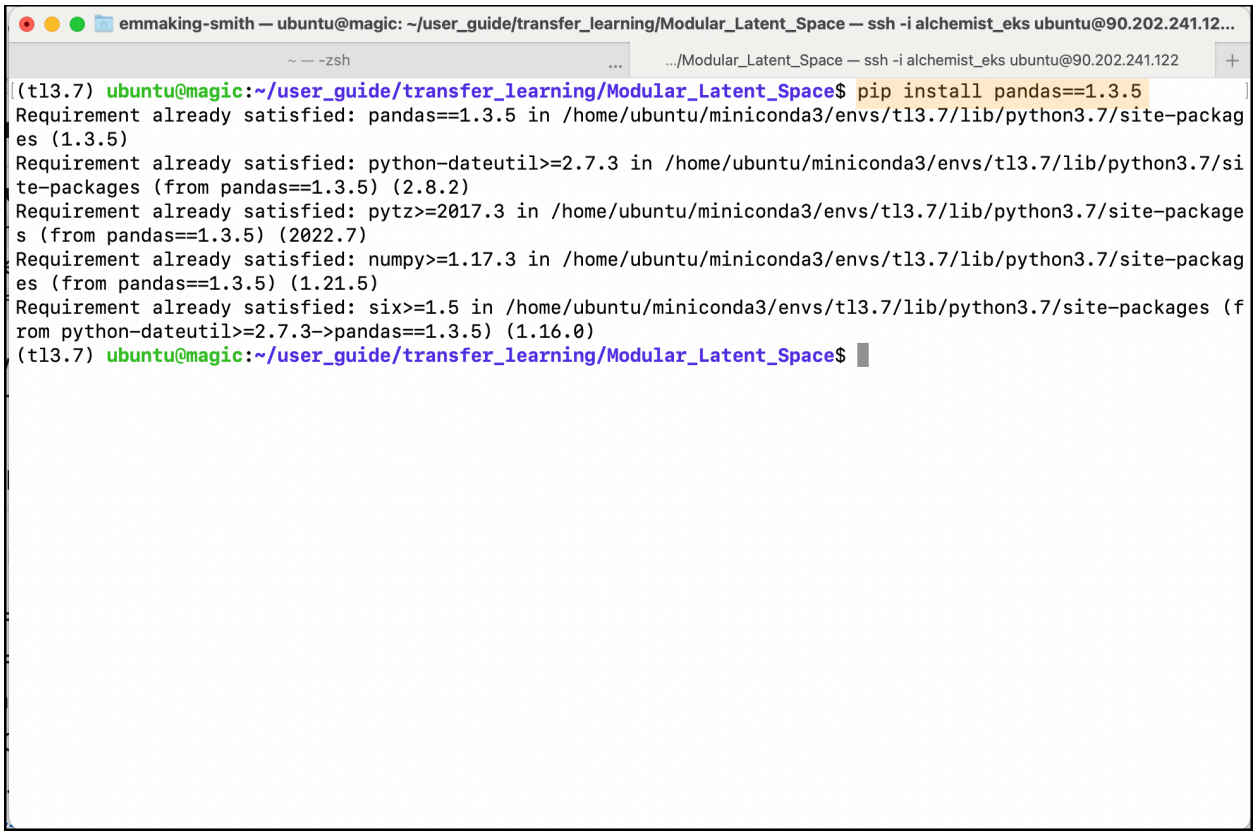
Next install numpy. This may have already been installed with a previous package.

```
pip install numpy==1.21.5
```



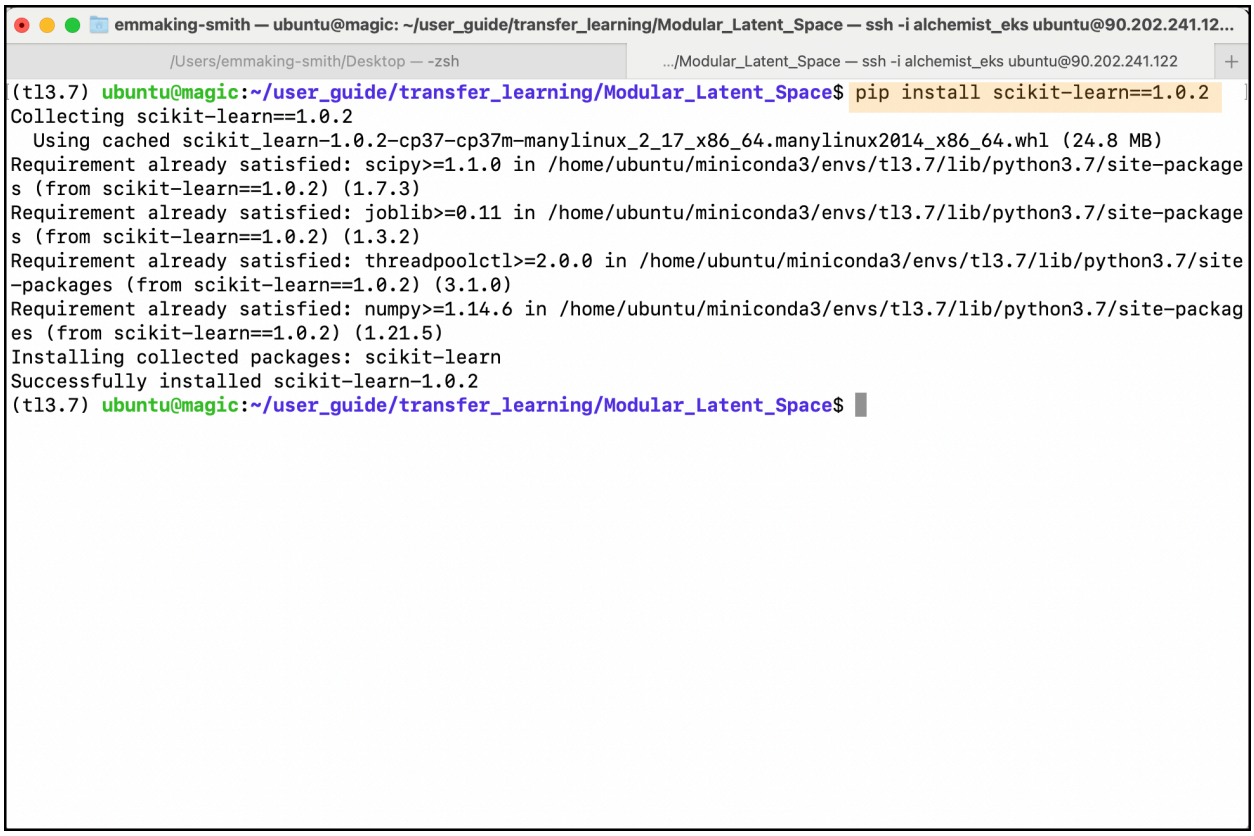
Then install pandas. Similar to numpy, this may already have been accomplished with a previous package installation.

```
pip install pandas==1.3.5
```



Finally, install sklearn.

```
pip install scikit-learn==1.0.2
```



We are now all set up!

Run the Transfer Learning

We will be using the Buchwald-Hartwig dataset as our example transfer learning. Note that the pretrained layers from the crystal structure information will be frozen. This has been done for you and can be observed in Modular_Latent_Space/buchwald/buchwald_yield_mpnn lines 29-30 (see orange box in the figure below).

```
13    device = 'cuda' if torch.cuda.is_available() else 'cpu'
14
15    class Buchwald_MPNN(nn.Module):
16        def __init__(self, message_size, message_passes, atom_list, pretrained_mpnn_path):
17            super(Buchwald_MPNN, self).__init__()
18
19            self.message_size = message_size
20            self.message_passes = message_passes
21            self.atom_list = atom_list
22            self.pretrained_mpnn_path = pretrained_mpnn_path
23
24            self.mpnn = Big_MPNN(self.message_size, self.message_passes, self.atom_list)
25            mpnn_trained_state_dict = self.gen_states(self.pretrained_mpnn_path)
26            self.mpnn.load_state_dict(mpnn_trained_state_dict)
27
28            # Turning off the params.
29            for param in self.mpnn.parameters():
30                param.requires_grad = False
31
32            self.yield_predictor = nn.Sequential(
33                nn.Linear(self.message_size * 4, self.message_size * 4),
34                nn.ReLU(),
35                # nn.Linear(self.message_size * 4, self.message_size * 4),
36                # nn.ReLU(),
37                # nn.Linear(self.message_size * 4, self.message_size * 4),
38                # nn.ReLU(),
39                # nn.Linear(self.message_size * 4, self.message_size * 4),
```

To run the transfer learning, we run a variation of following command:

`python py_file_that_makes_predictions.py --option1 choice --option2 choice ...`

The "`--`" indicates a flag. We are telling the model what arguments we are inputting. The flag lets the program know that a user-defined selection will occur, and the words after it detail the selection.

Understanding the Buchwald-Hartwig Flags:

For the Buchwald-Hartwig transfer learning, we have 3 flags of importance: split, test_mol_idx, and save_path. The split refers to what type of molecule should be left out for model validation. The options are "halide", "base", "ligand", and "additive". They are case sensitive.

The test_mol_idx stands for test molecule index and refers to which halide / base / ligand / additive molecules should be left out for model validation. Each index will yield a different set of molecules.

The final flag, save_path, is the place we wish to save our model and predictions. If we want to save it in a new directory called "predictions" to our Desktop our save_path flag would look like:

`--save_path ~/Desktop/predictions`.

To summarize our flags:

Flag name	Expected Value(s)	Example Flag Value	What to type
split	one of the following: halide,	ligand	`--split ligand`

	base, ligand, additive. **NOTE:** Case sensitive!		
test_mol_idx	An integer between 0 and 3 if your split is **NOT** base. An integer between 0 and 2 if your split **IS** base.	3* *Must not have --split base flag.	`--test_mol_idx 3`
save_path	A path to a directory.	A new directory in the Modular_Latent_Space directory called transfer_learning_test.	`--save_path transfer_learning_test` **NOTE:** Assuming your current location is the Modular_Latent_Space directory.

For more flags, please refer to Modular_Latent_Space/buchwald/buchwald_yield_prediction.py lines 25 - 37. For a basic transfer learning, feel free to use the default options.

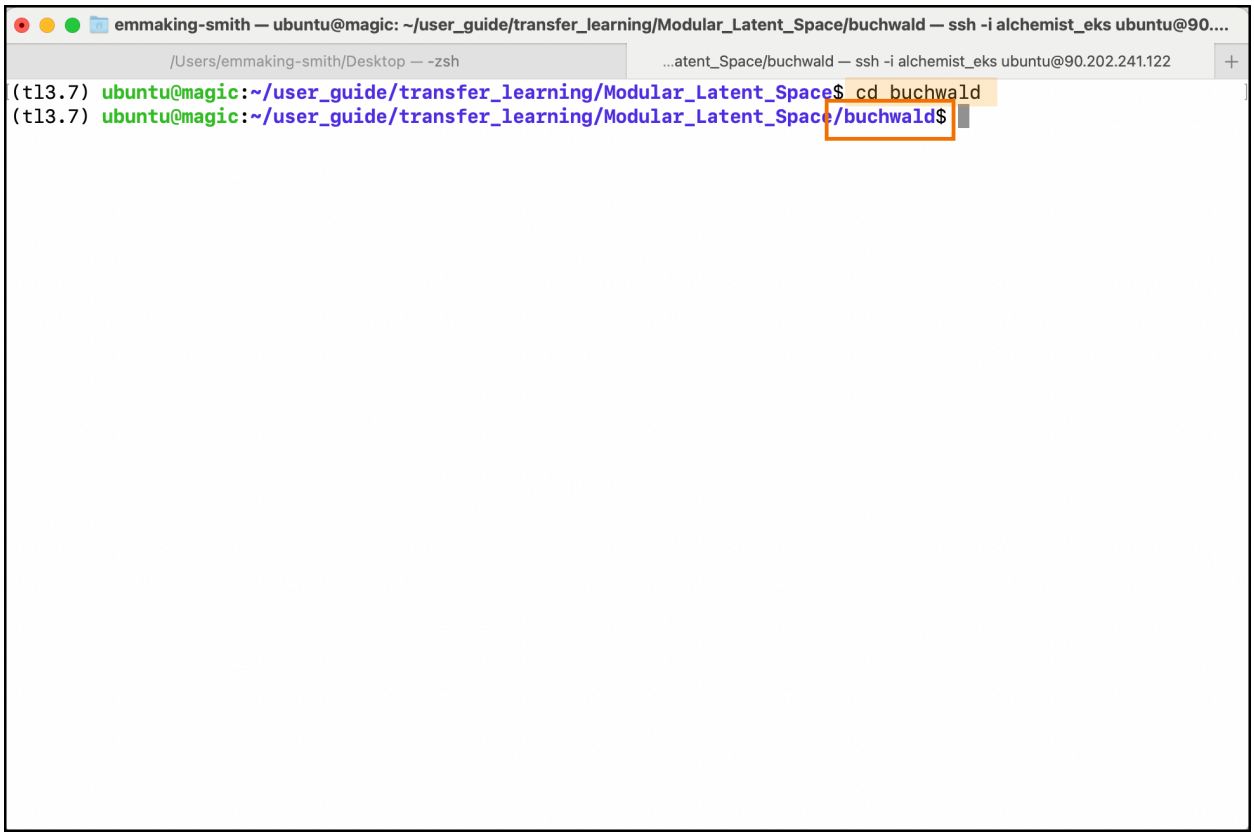
```
19  from buchwald_yield_mpnn import Buchwald_MPNN
20
21  class Initialization:
22      def __init__(self):
23          pass
24
25      def init_args(self):
26          parser = argparse.ArgumentParser()
27          parser.add_argument('--path_to_buchwald_data', type=str, help='The path to the Buchwald csv file.',
28                              default='doyle_buchwald_data.csv')
29          parser.add_argument('--epochs', type=int, default=100)
30          parser.add_argument('--learning_rate', type=float, default=1e-4)
31          parser.add_argument('--batch_size', type=int, default=64)
32          parser.add_argument('--message_size', type=int, default=128)
33          parser.add_argument('--message_passes', type=int, default=3)
34          parser.add_argument('--pretrained_mpnn_path', type=str, default='../MPNN/big_mpnn_no_delocalised_no_unknown_model')
35          parser.add_argument('--save_path', '-s', type=str)
36          parser.add_argument('--split', type=str, help='Options are: halide, ligand, base, additive.')
37          parser.add_argument('--test_mol_idx', type=int, help='What molecule to leave out for testing.')
38          return parser.parse_args()
39
```

Run the Transfer Learning (Finally!):

We will then run the transfer learning on the Buchwald-Hartwig dataset. The module to do so is called buchwald_yield_prediction.py and is located in the buchwald directory. Move yourself into the buchwald directory using the `cd` command. If you are currently in the Modular_Latent_Space directory, this can be achieved with:

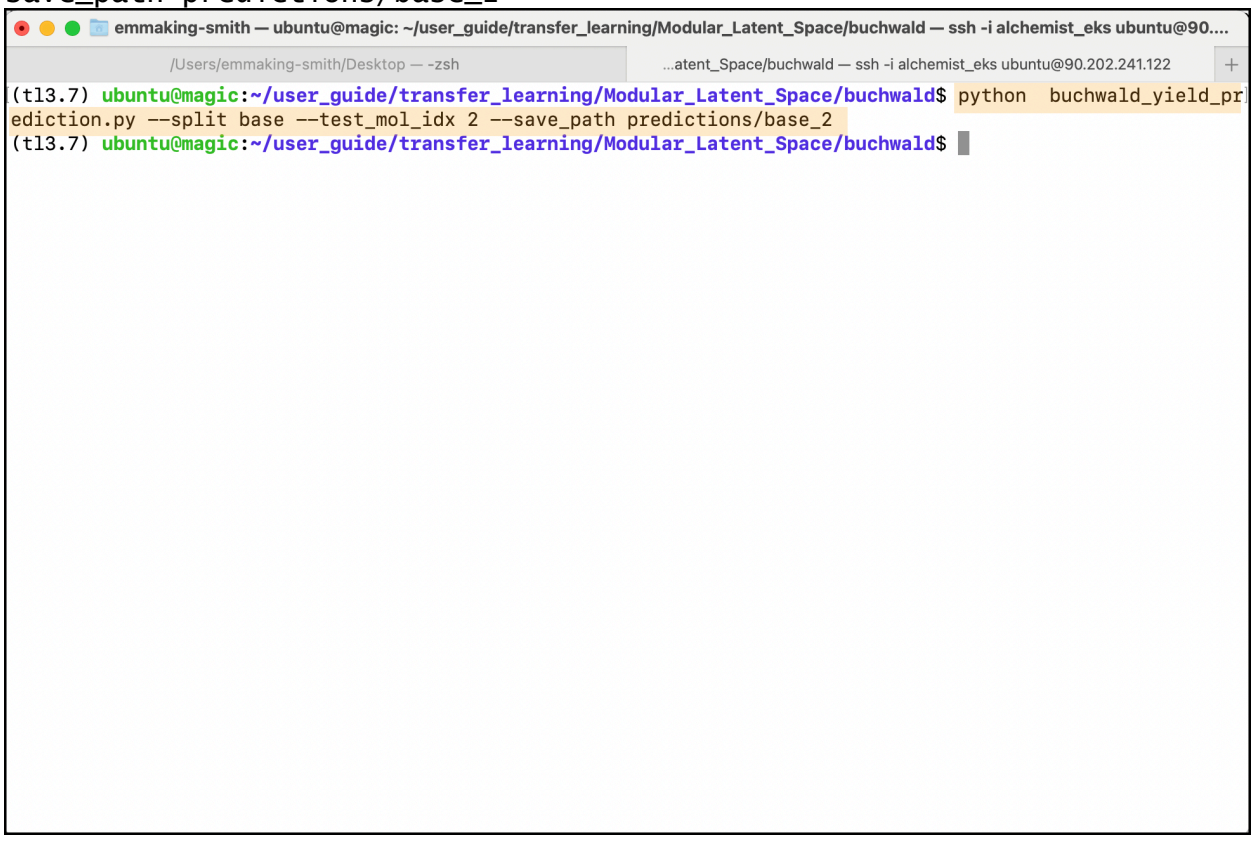
```
cd buchwald
```

You can easily tell you are now in the buchwald directory by looking at the path in blue. The final name will be "buchwald" (see orange box in figure below).



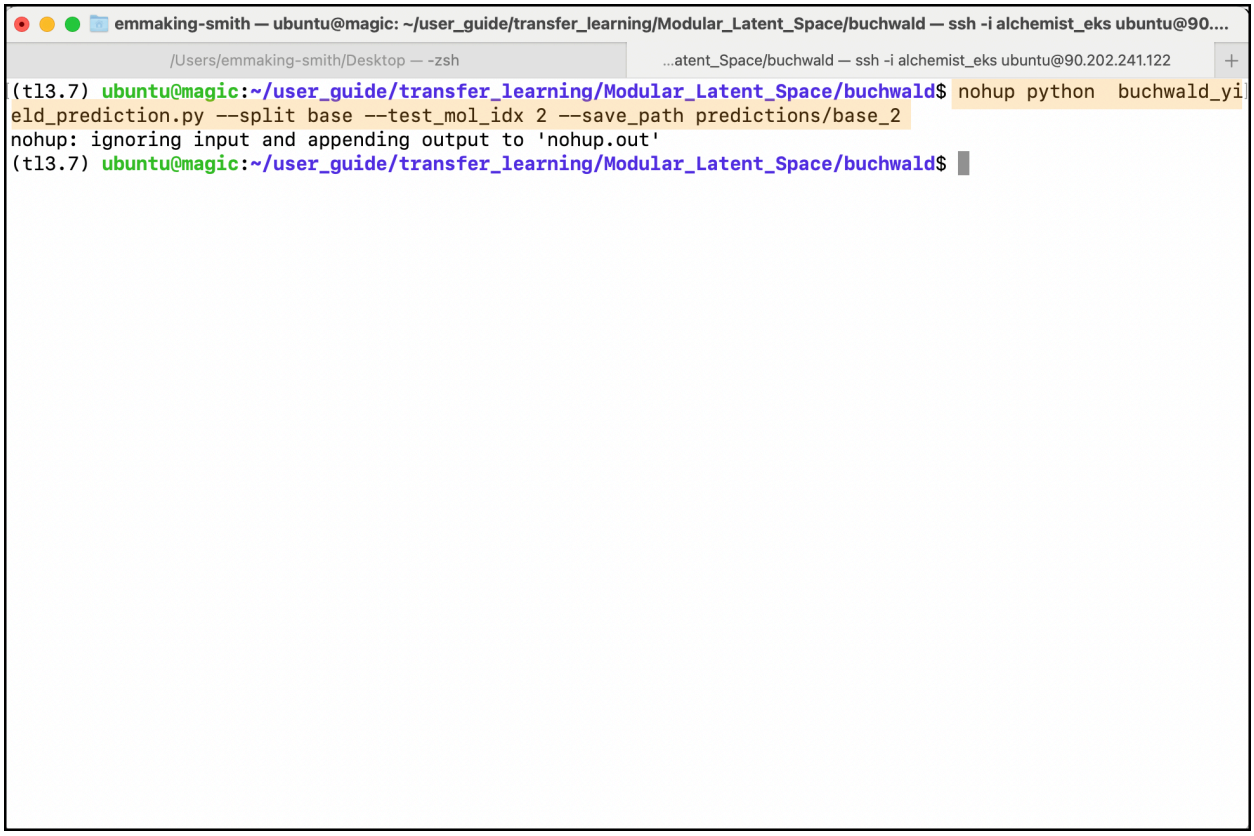
To run the Buchwald-Hartwig predictions, using a base split, with split index 2, and saving the predictions to a new directory called predictions/base_2, type:

```
python  buchwald_yield_prediction.py --split base --test_mol_idx 2 --
save_path predictions/base_2
```



Helpful hint: If you are running these computations through an ssh, adding the nohup command to the beginning will keep your training running even if you lose the ssh connection:

```
nohup python  buchwald_yield_prediction.py --split base --test_mol_idx 2 --
save_path predictions/base_2
```



You can track your progress with the model_log.log (saved to the save_path directory) and the nohup output file (default nohup.out).

Voila! Your predictions will be saved to predictions/base_2.

The preds.pickle are the predicted values, the model file is the final trained model, and the model_log.log is the log file that gives more information regarding the training loss and model parameters.

Viewing the Predictions

Ensure that you are in the correct directory (your save_path directory). If you are in the buchwald directory after completing training and have been following the naming convention of this document, you can move to the save_path directory by typing:

```
cd predictions/base_2/
```

To view pickle files, go into python by typing:

`python`

Import pandas with:

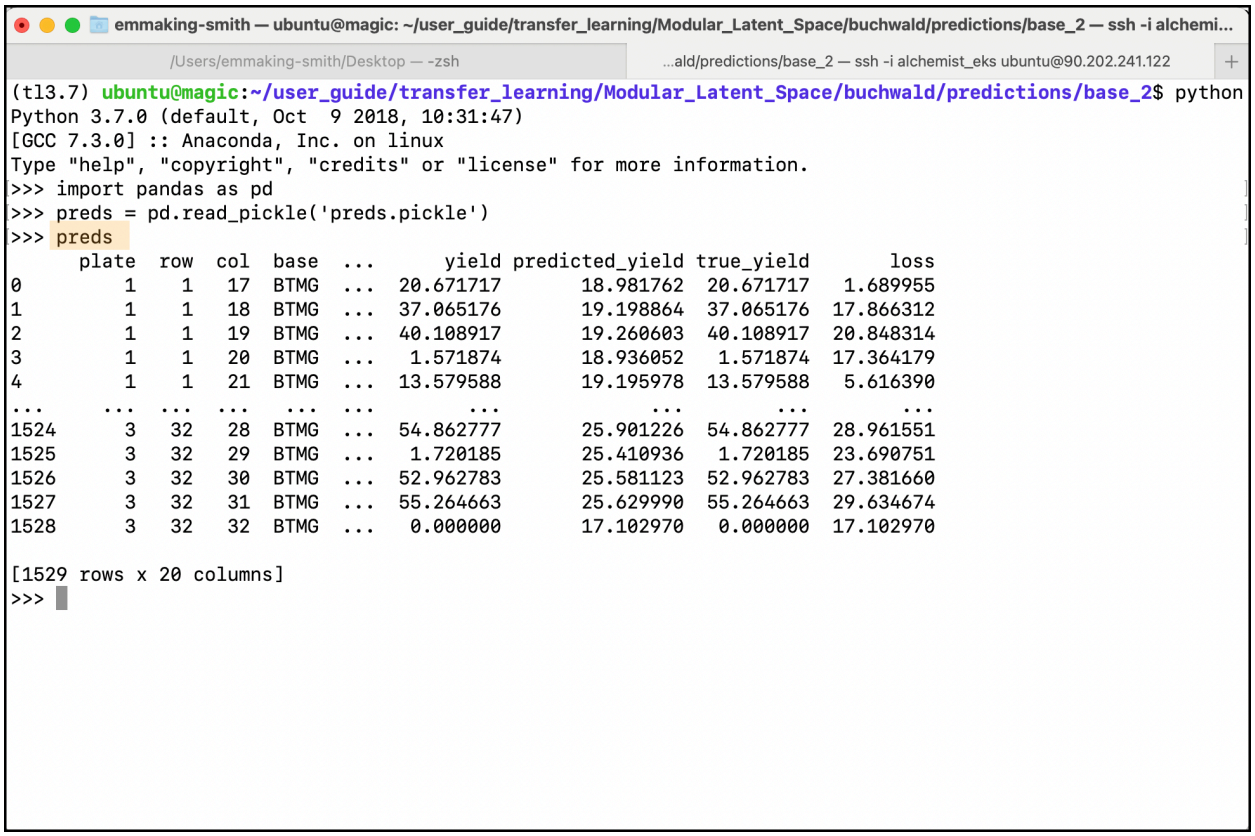
```
import pandas as pd
```

Import your predictions file with:

```
preds = pd.read_pickle('preds.pickle')
```

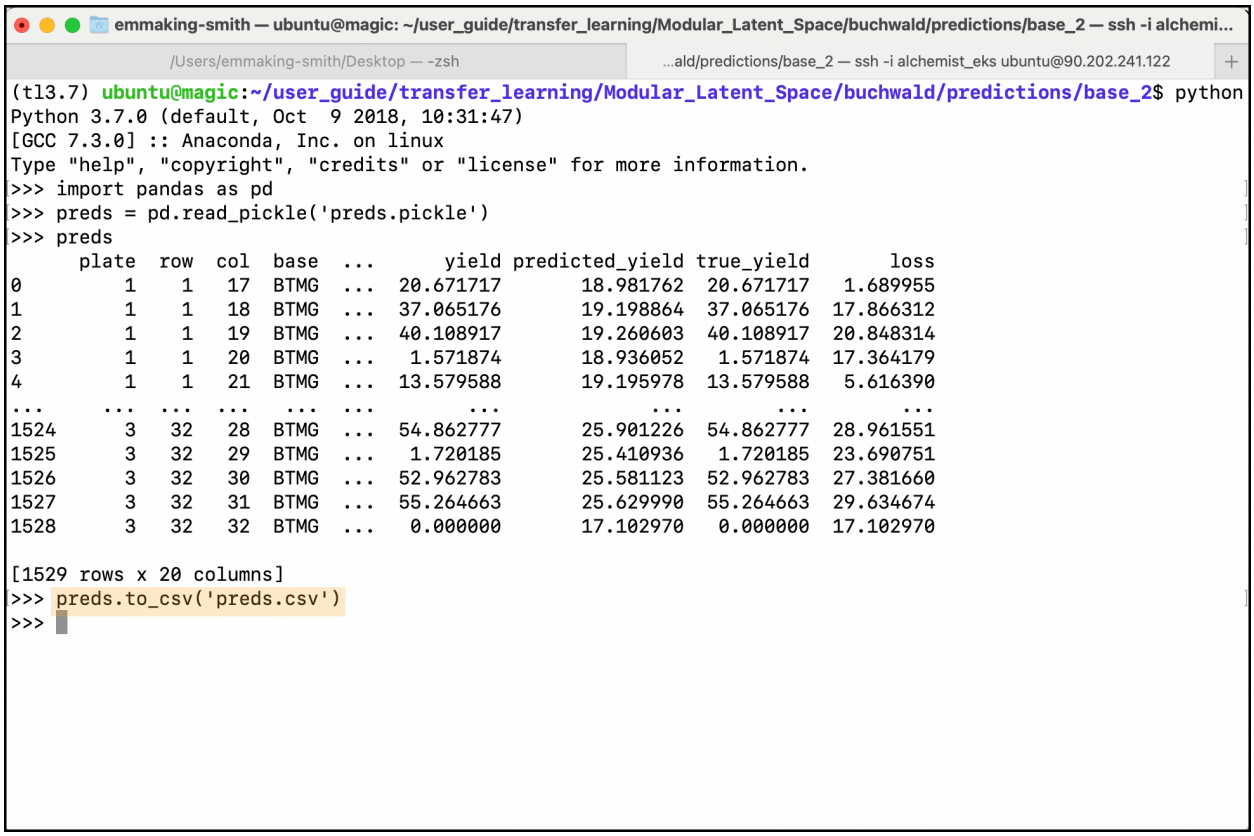
View your predictions file with:

```
preds
```



This will give you a snapshot of the prediction file. You can save it out as a csv file which is openable with Microsoft Excel if that is more convenient for you with:

```
preds.to_csv('preds.csv')
```



This will generate a csv file in your working directory called preds.

Thank you for reading to the end of this guide!

We hope it has been a helpful resource for you.