

# Libraries and Environment Setup

All the code examples in this book are in Python and these use various python packages/libraries such as PyTorch, TensorFlow, Gymnasium RL environments, Stable Baselines 3, and a few other ones. All the accompanying code has been setup as Jupyter notebooks to make the code execution interactive with detailed explanations accompanying the code. Jupyter notebook is a great way to mix the code and explanation with rich format. There are many ways to set up the environment and we will talk about a few common ways with step-by-step instructions. Please do note that the code is self-contained and can be run in any type of Python environment, local or cloud.

As we will be covering quite a few variations, readers should be able to use these instructions as guidance to setup code execution environments on other platforms of their choice. However, unless there is a reason, we suggest that readers choose one of the two recommended ways to go about setting it up – either a local environment or cloud-based Google Colab. Most of the notebooks have been designed to run on a local computer with just CPU. To leverage GPUs, you may need to do some platform specific installs and some minor code changes in relevant notebooks requiring GPU processing.

## Local Install (recommended for local option)

Let us first walk through the steps required for local installation. This is the most preferred approach for running the code locally. Please follow the steps given below:

1. We recommend Python version 3.9 as code has been tested on this specific version of Python. Having said that the code should work on most of the recent Python versions including 3.10 or 3.11.

On a windows machine, we recommend using the WSL2 based ubuntu distribution. On a Mac or Linux (especially the latest Ubuntu distributions) you should already have everything ready. In case you are new to WSL2, please refer to the link <https://learn.microsoft.com/en-us/windows/wsl/install> to get it ready. All the code in the book has been tested using WSL2 with Ubuntu 22.04. It should also work flawlessly on most other versions of Linux.

2. Having identified the platform on which you will set up the local environment, the next step is to install a specific version of Python. We will talk about two approaches. One is that of creating a virtual environment based on `venv`, the approach we took in the book. The other one is to set up a `miniconda` based

virtual environment. Step 3 walks you through the `venv` setup and Step 4 details the `miniconda` based setup.

3. VENV based virtual environment: On Windows WSL and Ubuntu, you can use the `apt-get` command on your WSL2/Ubuntu shell to first install the specific version of Python, i.e. 3.9.x in our case. Run the following set of commands to do so:

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install python3.9
```

Next, we create and activate a virtual environment based on Python 3.9 using the commands as follows:

```
# Install venv package for python 3.9
sudo apt install python3.9-venv

# Make a folder for venv virtual environments
mkdir ~/.venvs

# Create a new virtual environment
python3.9 -m venv ~/.venvs/drl

# Activate the new venv
source ~/.venvs/my-venv-name/bin/activate
```

At this point, the prompt in the shell window should show `(drl)` as a prefix on command line. You can check the version of the python installed by using the following command on the shell prompt: `python -V`. It should print something like: Python 3.9.18 with “18” in the end replaced with the latest sub version 3.9 has at the point in time you do the installation. Please check the pip version also with the command: `pip3 --version` which should print the version number of whatever pip is linked in your virtual environment that you just installed.

To deactivate the virtual environment, you can use the following command: `deactivate` which will deactivate the virtual environment and return you back to your original system default version of python. Please do not do this right now as

you need to be in the activate Python3.9.x version to follow the remaining steps of the installation.

At this point, you can move to Step 5. Step 4 is to use `conda` environment management for installation of a specific Python version and creating an isolated environment for the execution of code in this book.

Make sure that all the commands that we ask you to run in upcoming steps are carried out in the same terminal where you have activated the new `venv` environment.

4. Conda based environment: Conda provides package dependency, and environment management for any language. You can visit the link <https://conda.io/projects/conda/en/latest/index.html> for more details on what problem does Conda solve, its advantages and disadvantages. We will work through the steps for installing `conda` on WSL2/Unix. The links also have specific instructions for installing the same on macOS both in native way or with the help of homebrew, a popular package manager for macOS.

- a. Visit <https://docs.conda.io/en/latest/miniconda.html> and download the Miniconda install for your platform. Please choose the latest Python3.x version. If you already have Anaconda or Miniconda installed, you can skip this step. Run the downloaded program to install miniconda on your local machine.
- b. Next, we create a specific environment with python3.9 version. Open a shell window use the following command:

```
conda create -n drl python=3.9
```

where `drl` is the name of the environment and we are using Python 3.9.x. Answer yes to all the prompts.

- c. Switch to the new environment you created using the following:

```
conda activate drl
```

Make sure that all the commands that we ask you to run in upcoming steps are carried out in the same terminal where you have activated the new `conda` environment.

Having installed a specific Python version, you can continue with Step 5.

5. With `venv` (step 3) or `conda` (step 4) activated, please follow along to complete rest of the steps. Please make a local folder and download or clone the source code provided in the links to the book. A mirror repository maintained by the author with latest updates can be found at <https://github.com/nsanghi/drl-2ed>

Change directory (`cd`) to the newly created local folder and run the following command to make a local copy of the source code accompanying the book.

```
git clone https://github.com/nsanghi/drl-2ed.git
```

This command will create another subfolder named “`drl-2ed`” in the current folder where you ran the command. You can also visit the publisher’s source code link to clone the repository or download and unzip the code.

6. Setting up the dependencies: We are now ready to install all the required libraries with the help of two commands as follows:

```
# First install some required packages
apt-get install swig cmake ffmpeg freeglut3-dev xvfb \
git-lfs

git lfs install
```

The above commands install the required system packages which are required by some of the python packages we will be installing next. On macOS we recommend the use of `brew` command from homebrew to install these packages. You will need to replace `apt-get` with `brew` in above command.

**NOTE: macOS Users - The `xvfb` package is not available for macOS. Accordingly, please use the following commands:**

```
# First install some required packages
brew install swig cmake ffmpeg freeglut3-dev git-lfs
```

```
git lfs install
```

Next, we install the required python packages. These have been provided as a `requirements.txt` file which is a preferred and reproducible way to share environments in python eco system. Please make sure that you are currently in the sub-folder `drl-2ed` which was created with the `git clone` command or the `copy` and `unzip` command in Step 5. Once you have confirmed the same, you can run the following command to install all the required dependencies. Please note that it may take 5-10 mins for the installation to finish. Time for you to grab a cup of coffee which the installation finishes.

```
# install python packages from requirements.txt file
pip install -r requirements.txt
```

**NOTE: macOS Users – You may face some build errors while running the above command line. In that case, please run the following sequence of commands to resolve the issue:**

```
# install python packages from requirements.txt file
pip install --use-pep517 pymunk
pip install -r requirements.txt
```

After the above command is executed and completed, you have everything required to explore and execute the code.

The steps 1 to 6 need to be run only once. After the initial installation, you do not need to run these again unless you want to do a reinstall.

7. As part of the python installation, we have also installed JupyterLab package. JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. You can use JupyterLab interface to open a notebook or even a command shell.

Please make sure that your `venv` or `conda` environment is **active** and you are currently **inside the drl-2ed directory** where you cloned/copied the code. You can now start the Jupyter Lab session with the command:

## Jupyter lab

It will start the Jupyter session and will print following instructions somewhere in the middle of the output created by the above command:

To access the server, open this file in a browser:

`file:///home/nsanghi/.local/share/jupyter/runtime/jpserver-1608-open.html`

Or copy and paste one of these URLs:

`http://localhost:8888/lab?token=<random string>`

`http://127.0.0.1:8888/lab?token=<random string>`

Click on any of the links to open your system default browser. You should see a page similar to the one shown in Figure 1-2 open. You may navigate to any of the chapter specific folders, double click on specific notebooks (files with “.ipynb” extension) to open. You are ready to follow along the notebook instructions and code execution.

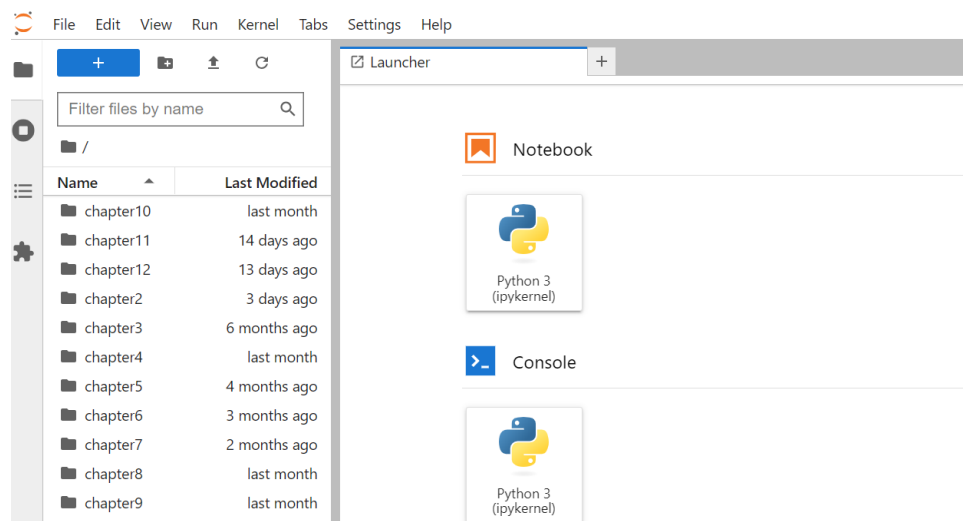


Figure 1-2 Jupyter Lab landing page

# Local Install with VSCode

If you want to use VSCode, you can still follow up all the steps for “local Install” and then use VS Code to open, run and execute the notebooks. It gives a similar experience to a local install but with some additional capabilities. You may also use this if you prefer the richer GUI experience VS Code offers and if VS Code is already a part of your developer experience. High level steps to do are as follows:

1. Install VS Code on your local machine. Readers may refer to the link <https://code.visualstudio.com/> for detailed setup instructions.
2. Next, we need to enable a few plugins for Python: Python, Python Debugger, PyLance, Jupyter, Jupyter Notebook Renderers and Jupyter Keymap, Jupyter Cell Tags and Jupyter Slide Show. Some of these are optional but installing all of them will give you a richer experience. You can read through each individual plugin and decide which ones you want to skip.
3. Navigate to the `drl-2ed` directory where you cloned/downloaded the source code. Run the following command:

```
code .
```

Do not forget the period (.) at the end of the command. This command instructs the system to open VS Code in the current folder.

4. You may see a pop-up at the bottom right corner with the message “*Folder contains Dev Container configuration file. Reopen folder to develop in a container*”. Please do not click on the button “*Reopen in Container*”. Instead click on the cross at the top right of the pop-up to close it.

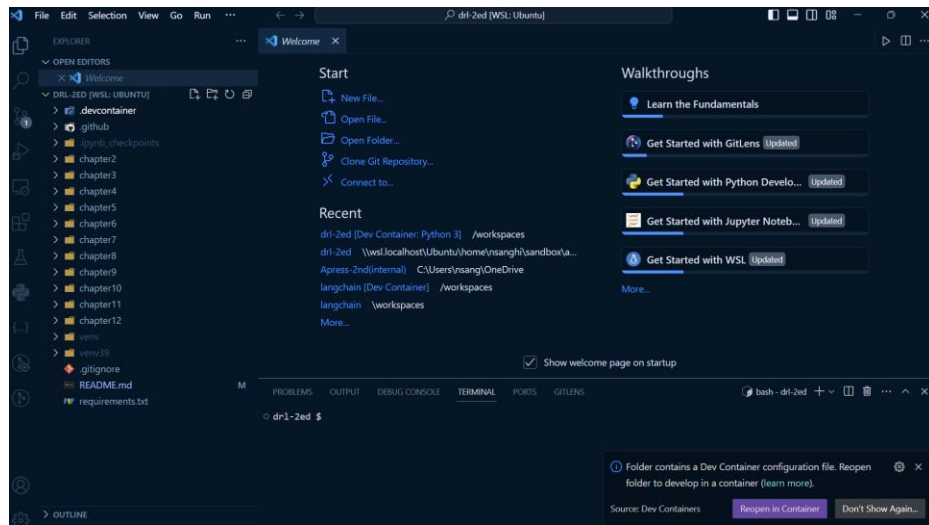


Figure 1-3 Opening in VS Code

- Next, Press “Control+Shift+P” or “command+Shift+P” to open the command palette in VS Code and type “Python” to see a list of commands applicable for Python in VS Code. Choose the option “*Python: Select Interpreter*” as shown in Figure 1-4.

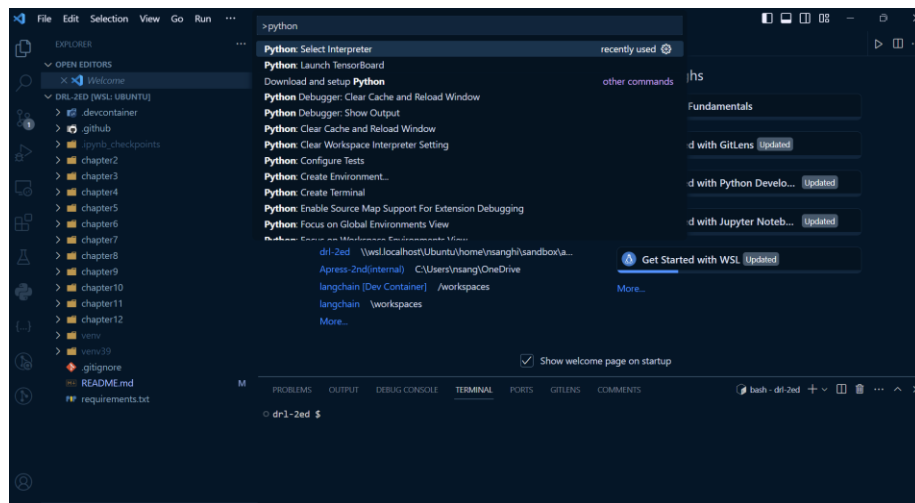


Figure 1-4 Selceting Python Enviroment in VSCode

It will show all the python environments found in your system. Please choose the `venv` or `conda` environment you created for local install. Either you will find this



option listed in the command palette or you can add it using the option “*Enter interpreter path...*”

6. You may now open any notebook. At times, even after you have chosen the Python environment in the previous step, you again may need to select the same environment once more as the notebook kernel before you can execute the code in the notebook. You will find that option on the top right side of the notebook and it will read as “*Select Kernel*”. Please click on this and again select the same Python environment that you created for local install and the one you selected as Python Interpreter in previous Step 5.

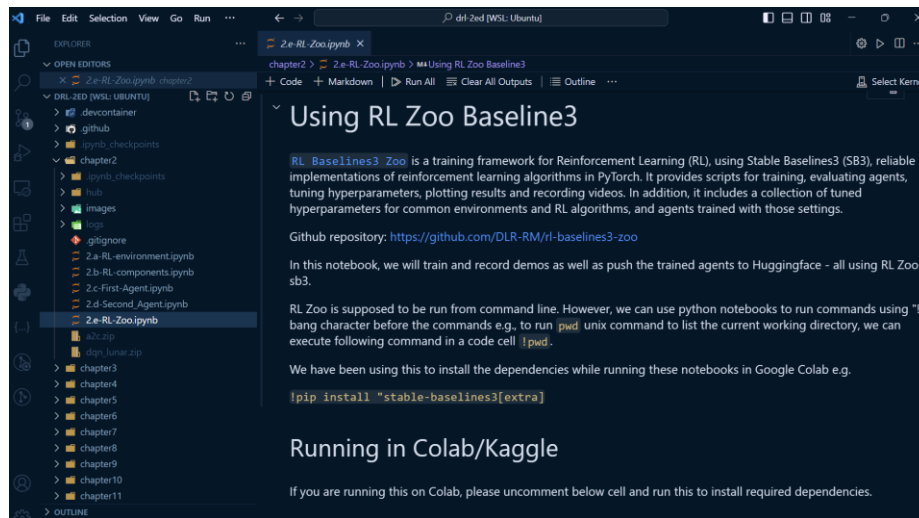


Figure 1-5 Selecting Notebook Kernel in VSCode

You are ready to explore the notebooks and execute the code cells contained in the Jupyter notebook.

## Running on Google Colab (recommended for cloud option)

Google Colab is a free online platform that allows you to write and run Python code in your browser. It is based on Jupyter Notebook. Google Colab provides some advantages over Jupyter Notebook, such as:

- You can access Google Colab from any device with an internet connection, without installing anything on your local machine.
- You can use the computing power of Google's servers, including GPUs and TPUs, to speed up your code execution and save resources on your own device.
- You can easily share your notebooks with others and collaborate on them in real time.
- You can integrate your notebooks with Google Drive and other Google services, such as Google Sheets, Google Forms, and Google Maps.

To use Google Colab, you need a Google account and a web browser. You can create a new notebook by visiting <https://colab.research.google.com/> or opening an existing notebook from Google Drive. You can also upload a notebook file from your local machine or import one from GitHub or other sources. Once you have a notebook open, you can write and execute code cells, add text and images, and use various tools and features provided by Google Colab. You can also customize your notebook settings, such as the runtime type, the theme, and the keyboard shortcuts.

We will use GitHub option to directly open notebooks from the github repository. It is just a matter of personal preference. And if you have cloned the source code repository in github to your github account, you will also be able to make changes and issue commits to your repo giving you a seamless developer experience without any local install on your computer. Let us walk through the steps.

1. Navigate to <https://colab.research.google.com/> which will open the webpage with a dialog box giving you various options to open a notebook. In our current workflow, we will choose the github option.
2. If this is the first time you are using github from inside the Colaboratory (Google Colab), clicking on github option on the dialog box will take you through an OAuth workflow to grant Google Colab access to your github account.
3. After the authentication is done and Google Colab is connected to your github account, you can use the dialog box to open a specific notebook of interest or you can directly type the url of github account housing the repository, e.g. <https://github.com/nsanghi/>.
4. Click on the repository dropdown to navigate to the repository. Keep the branch selection to "main" or whatever else is applicable for the repository you are trying to open. A sample of the dialog at this point is shown in Figure 1-6.

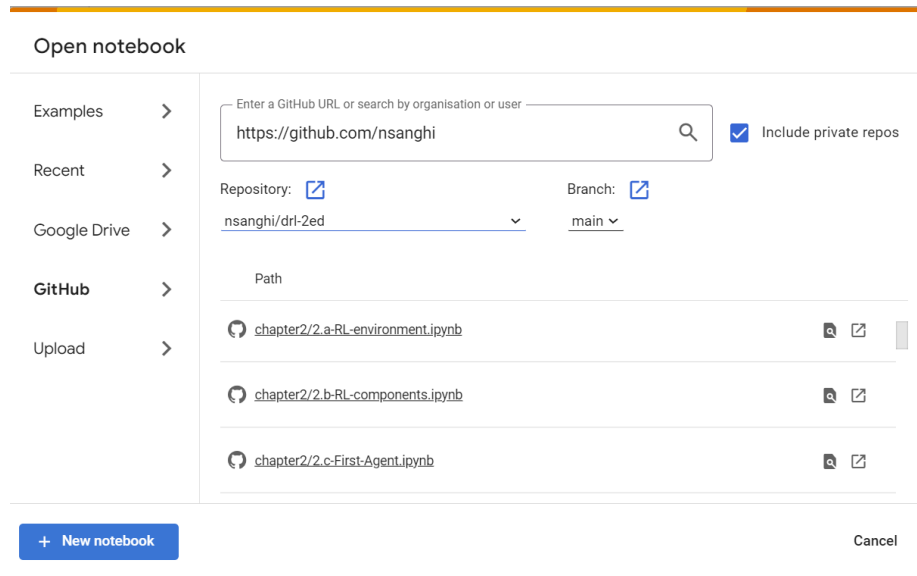
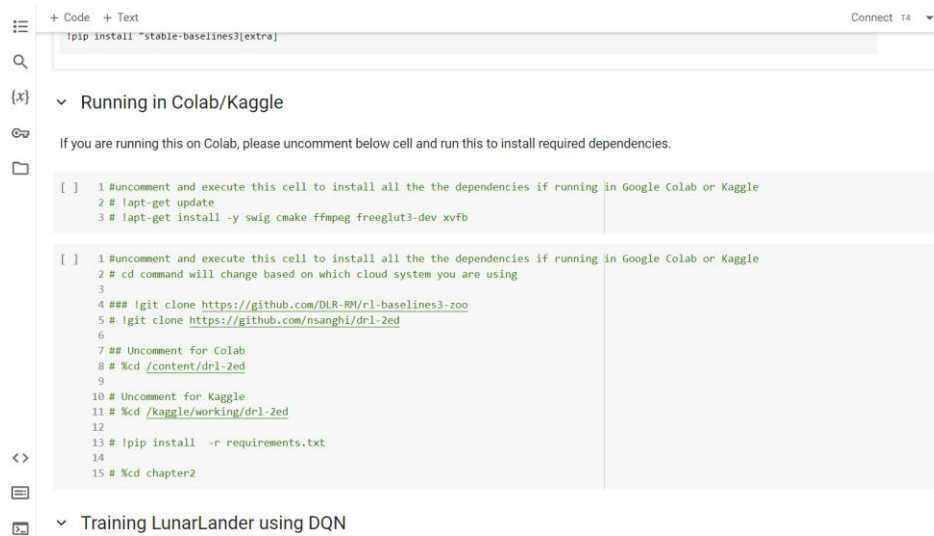


Figure 1-6 Selecting the repository and notebook in Google Colab

5. Select the notebook e.g. `chapter2/2.a-RL-environment.ipynb`. It will close the dialog and open the notebook in the browser with an interface very similar to Jupyter lab interface. There are some Google Colab specific enhancements, but most are not very relevant for our case. The only thing to note is the run time type that you can choose from the drop down labelled “Connect” on the top right of the Colab web page.

We will be mostly using CPU environment as free tier of Google Colab gives limited access to GPU/TPU environment. Most of the notebooks can be run on a CPU environment. However, Chapter 6 onwards, once we enter the realm of Deep Reinforcement Learning and start training agents on either large text data (such as RLHF) or images (such as Atari games), we may run those notebooks with GPU enabled.

6. Google Colab comes with most of the libraries pre-installed. However, we have designed the notebooks such that you may run some additional commands from inside the notebook to install system level packages as well as required python libraries. An example of the same is the notebook “`2.e-RL-Zoo.ipynb`”. Let us open this notebook in Google Colab following instructions from Step 4. Once it is opened, you will see two code execution cells which have code commented out as shown in Figure 1-7.



The screenshot shows a Google Colab notebook with two code cells. The first cell contains the command `!pip install "stable-baselines3[extra]"`. The second cell contains a series of commands for cloning a GitHub repository and installing dependencies, with comments indicating which lines to uncomment for Colab or Kaggle. The notebook interface includes a left sidebar with icons for file explorer, search, and other functions, and a top bar with a 'Connect' button.

```
+ Code + Text
!pip install "stable-baselines3[extra]"

Connect T4

Running in Colab/Kaggle

If you are running this on Colab, please uncomment below cell and run this to install required dependencies.

[ ] 1 #uncomment and execute this cell to install all the the dependencies if running in Google Colab or Kaggle
    2 # !apt-get update
    3 # !apt-get install -y swig cmake ffmpeg freeglut3-dev xvfb

[ ] 1 #uncomment and execute this cell to install all the the dependencies if running in Google Colab or Kaggle
    2 # cd command will change based on which cloud system you are using
    3
    4 ### !git clone https://github.com/DLR-RM/r1-baselines3-zoo
    5 # !git clone https://github.com/nsanghi/dr1-2ed
    6
    7 ## Uncomment for Colab
    8 # %cd /content/dr1-2ed
    9
    10 # Uncomment for Kaggle
    11 # %cd /kaggle/working/dr1-2ed
    12
    13 # !pip install -r requirements.txt
    14
    15 # %cd chapter2

Training LunarLander using DQN
```

Figure 1-7 Running code cells for package installation in Google Colab

The first cell installs the Ubuntu system level packages. The second cell clones the github repository and runs the requirements.txt file from cloned repository to install all the dependencies.

Please uncomment and run these cells before proceeding further with the notebook.

Please note that you do not need to run the commented code cells for any kind of local installation-based execution.

## Running on Kaggle

Just like Google Colab, you may also use a similar functionality offered by Kaggle. Kaggle is a platform that hosts data science and machine learning competitions, where you can find and explore various datasets, learn new skills, and share your solutions with other users. Kaggle also provides a cloud-based service called Kaggle Kernel, which allows you to run notebooks online without installing anything on your local machine. You can use Kaggle Kernel to access the datasets from the competitions, run code in Python or R, and publish your results as interactive web pages. Kaggle Kernel is a convenient way to experiment with different models and techniques, collaborate with other users, and showcase your work. You can also fork and edit existing notebooks from other users and use them as a starting point for your own projects.

Running the notebook from this book on Kaggle is pretty similar to the way you do on Google Colab. You can read more about it in the Kaggle documentation and try this out.

## Using devcontainer based environments

Another way to run notebooks is to use devcontainer, which stands for development container. A “.devcontainer” is a folder in a repository that contains a Docker file and a configuration file that specify the tools and settings for a development environment. You can use “.devcontainer” to create a consistent and reproducible workspace that can run on any machine that supports Docker and Visual Studio Code (VS Code).

To use devcontainer, you need to have Docker and VS Code installed on your local machine or cloud service. You also need to install the Remote Development extension pack for VS Code, which enables you to work with remote environments. Then, you can find a repository that has a “.devcontainer” folder, which indicates that it supports devcontainer.

Once you have a devcontainer repository, you can clone it to your local machine or cloud service and open it with VS Code. VS Code will detect the .devcontainer folder and prompt you to reopen the folder in a container. This will build the Docker image and launch the container with the specified tools and settings. You can then open the notebook file and run it inside the container. You can also modify the code and push the changes back to the repository.

Using devcontainer is a convenient way to work with notebooks that require specific dependencies or configurations, without affecting your local machine or cloud service. You can also share your devcontainer setup with other users and ensure that they have the same development environment as you. Devcontainer is a useful feature for collaborating and deploying code. You can read more about it in the VS Code documentation and try it out.

The source code repository for the book has this capability enabled. We will look at two ways to use this. One will be using a local system, and another will be using github’s Codespace which provides a cloud environment to run code.

## Running devcontainer Locally

As discussed previously, you need to have Docker and VS Code installed locally on your machine. To install Docker, you may head over to the link <https://www.docker.com/products/docker-desktop/> and install the Docker Desktop version for your system. Download and run the binary for your platform and follow all the prompts

choosing the recommended/default options. VS Code can be installed from <https://code.visualstudio.com/> . Once VS code is installed, please also install the Remote Development extension pack for VS Code. With these steps your local machine is now ready for local deployment of a docker container. The advantage of this approach is that your development environment is housed in a docker container, completely isolating your development environment from your local machine and irrespective of the operating system of your local machine, the code will run the same docker configuration which in our case is a ubuntu image with all the required dependencies. Let us now walk through the steps of downloading the source code and running it locally.

1. Navigate to a folder of your choice on the local machine. If you are on a windows machine, you can either choose the WSL2 subsystem or the windows subsystem.
2. Clone the source code repository locally with the command:

```
git clone https://github.com/nsanghi/drl-2ed.git
```

This command will create another subfolder named “drl-2ed” in the current folder where you ran the above command. You can also visit the publisher’s source code link to clone the repository or download and unzip the code.

3. Navigate to the drl-2ed folder created with the above git command and run VS Code from there with the command:

```
code .
```

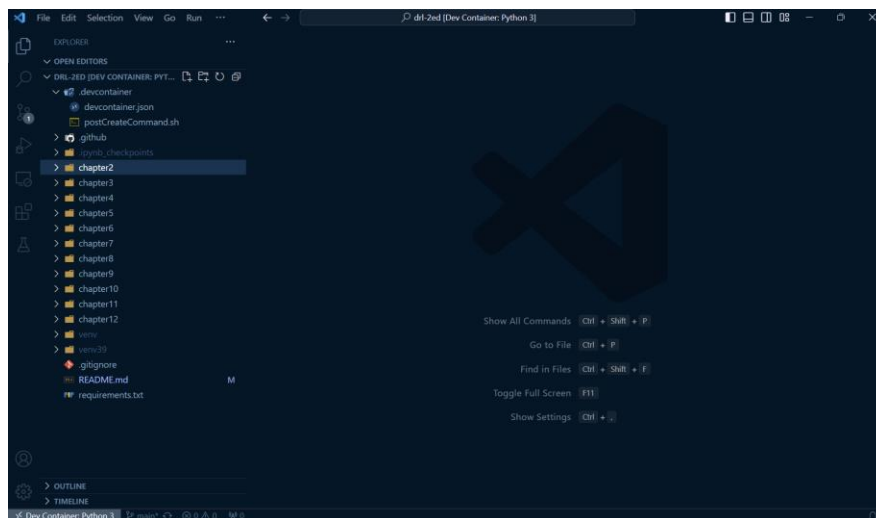
Do not forget the period (.) at the end of the command. This command instructs the system to open VS Code in the current folder.

4. You may see a pop-up at the right corner with the message “*Folder contains Dev Container configuration file. Reopen folder to develop in a container*”. **Please click on the button “Reopen in Container”** to kick in the docker build. It will use the development environment image specified inside the “.devcontainer” folder of the repository to create and start the docker container based on the specifications inside this folder. This is opposite to the instructions we had for running on VS Code locally wherein you did not click the button “Reopen in Container”.

First time you do this; it has to download the Docker image as well as install the required ubuntu packages and run the `"pip install requirements.txt"` command to setup the python environment. It may take a while, say about 10 mins or so for the first time to run the whole setup process. It will save the final docker container locally and make it visible in your docker desktop.

The subsequent runs will be very fast as it will use the saved docker container. In case you need to do a rebuild of docker environment, VS Code command palette has options to do so.

When the code is opened in a .devcontainer specified docker image, you will see a status like "Dev Container: Python 3" at the bottom left of the VS Code as shown in Figure 1-8. This confirms that you are now working inside a docker based dev container.



*Figure 1-8 Running in Docker Container*

5. Next, Press "Control+Shift+P" or "command+Shift+P" to open the command palette in VS Code and type "Python" see a list of commands application for Python in VS Code. Choose the option "Python: Select Interpreter" as shown in Figure 1-9. This time the Python interpreter options will be different. Please choose the Python 3.9.18 version as highlighted in Figure 1-9.

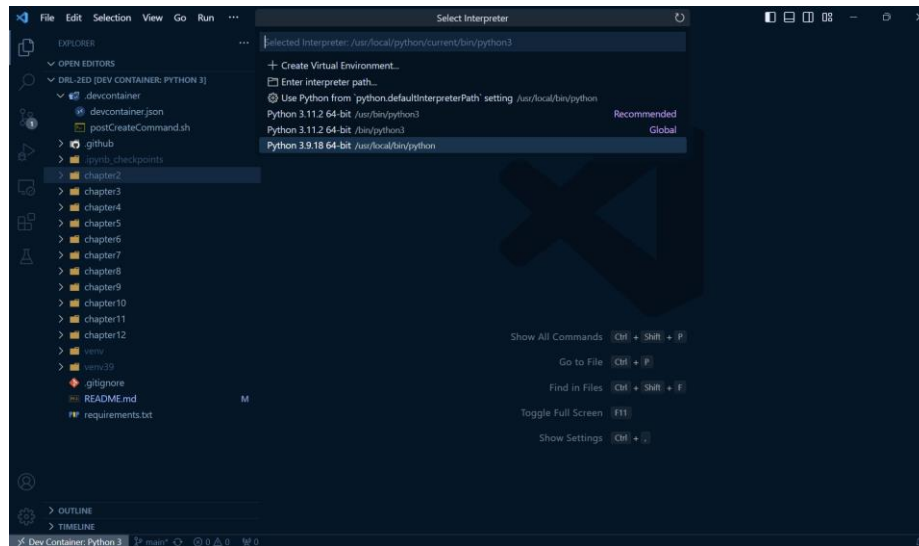


Figure 1-9 Selecting Python Environment in VSCode

6. You can open any notebook. You again may need to select the same environment as the notebook kernel before you can execute the code in the notebook. You will find that option on the top right side of the notebook and it will read as “*Select Kernel*”. Please click on this and again select the same Python environment with Python 3.9.18.

You can now execute the code just like the way you would do in a Jupyter Lab environment.

## Running on Github Codespaces

GitHub Codespaces is a cloud-based development environment that allows you to create, edit, and run code from any device. You can access your codespaces from Visual Studio Code, a browser-based editor, or GitHub.com. GitHub Codespaces integrates seamlessly with GitHub features, such as pull requests, issues, actions, and packages. You can also customize your codespaces with your own dotfiles, extensions, and settings. The key advantages of using something like Codespaces are:

- You can start coding quickly without setting up a local environment or installing dependencies.



- You can work on multiple projects and switch between them easily without cluttering your machine.
- You can collaborate with others in real time and share your codespaces with teammates or contributors.
- You can use the same tools and workflows that you are familiar with, such as VS Code, Git, and terminal.
- You can leverage the power and scalability of the cloud and run your code on fast and secure servers.

You can start with codespaces on the free tier which will get you 60 hours per month of 2-core machines. Let us go through the steps involved in using Codespaces:

1. Navigate to the repository, e.g. <https://github.com/nsanghi/drl-2ed/>
2. Make sure you are logged into your github account and click on the dropdown arrow next to the blue button with text “Code”. Next, you click on the button with text “Create codespace on main” as shown in Figure 1-10. It will open a new tab and start setting up the codespace using the docker container specifications provided in the “.devcontainer” folder of the repository.

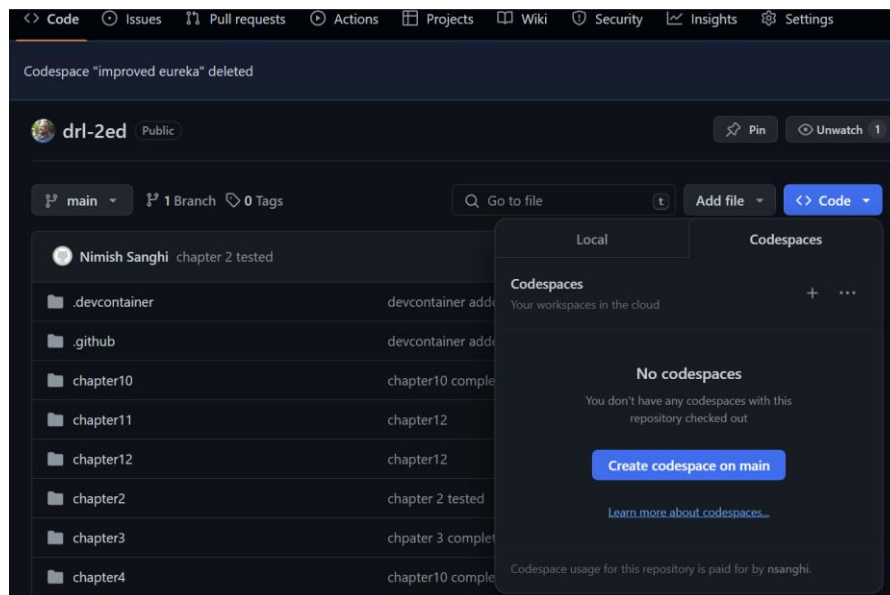


Figure 1-10 Setting up Github Codespace

3. Once the container is set up, it will open the repository in a cloud version of VS Code and run the pip install to install all the python packages. Like the local

docker container setup, first time setup on codespaces may also take about 5-10 mins. However, subsequent runs will be faster as it will reuse the cloud saved docker container it created in the first run.

4. Similar to the option of running on local docker, you need to select the right Python interpreter with Python 3.9.18 as shown in Figure 1-9. You also need to select the same Python version as the kernel while running the notebooks.
5. You are now ready to run the code on Github codespace. This option gives you full developer experience without any local install.
6. As the docker image is stored in codespace and is tied to your github account, next time you visit the github repository and click on the dropdown next to code, you will see the previously created codespace listed as shown in Figure 1-11. You can use that to run code in subsequent visits. It will use the saved docker container and in a matter of seconds get your environment running.

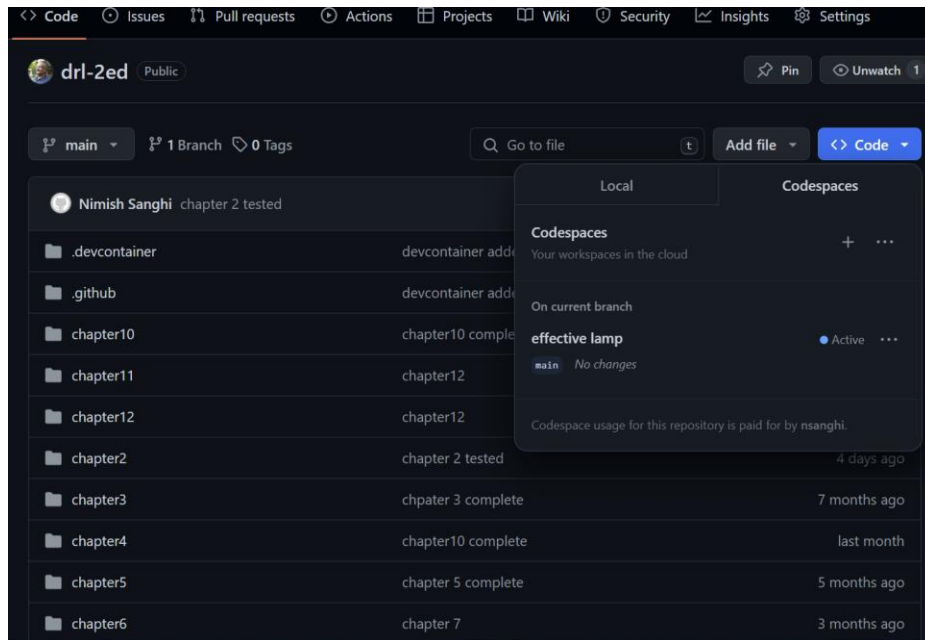


Figure 1-11 Setting up Github Codespace

7. You can click on the three dots on the right side of the container name and open the codespace container. You can also use the menu to open the cloud hosted codespace inside your local VS Code.

# Running on AWS Studio Lab

AWS Studio Lab is a free tier version from AWS which can be used as an alternative to Google Colab. It provides you with certain hours of free CPU and GPU access. You need to create an account by navigating to the link <https://studiolab.sagemaker.aws/>.

Once you have created the account and logged in, you can clone the github repository, create a `conda` environment and run the setup to install the ubuntu packages as well as python libraries specified in the `requirements.txt` file. The steps involved are as follows:

1. Once you are logged in, click on the button “Start runtime” to start a Jupyter instance.
2. After the runtime is running, click on the circle button “Open Project”. This will open a new tab with Jupyter Lab interface.
3. Next you need to clone the repository. Either you can open a terminal from the Jupyter Lab interface or click on the menu “Git”-> “Clone Git Repository”. Follow the instructions given in the dialog box to complete this step.
4. The next step is to create the `conda` environment with Python 3.9, install the ubuntu packages and also run the pip install. First open a terminal from the Jupyter Lab interface and run commands as follows:

```
# create conda environment and activate it
conda create -n drl python=3.9.18
conda activate drl

# install required Linux packages
conda install -c conda-forge swig cmake ffmpeg
conda install conda-forge::xvfbwrapper
conda install conda-forge::freeglut

# Navigate to the drl-2ed folder and
# install the required Python libraries
pip install requirements.txt
```

5. You can now open any of the notebooks from the explorer on the left side. When presented with the option to choose the kernel, choose the option of “drl”, the

conda environment we created in previous step. In case you do not see the option, you may need to click on “Amazon SageMaker Studio Lab”-> “Restart JupyterLab” option.

6. Having selected the right kernel, you may start executing the notebook code cells. The conda environment you created persists between sessions making subsequent runs very fast.

Use of SageMaker Studio Lab allows you to run the code on cloud, just like a few other options that we have talked about in earlier sections. It requires no local install at all. Figure 1-12 shows a screenshot of AWS Studio Lab in action.

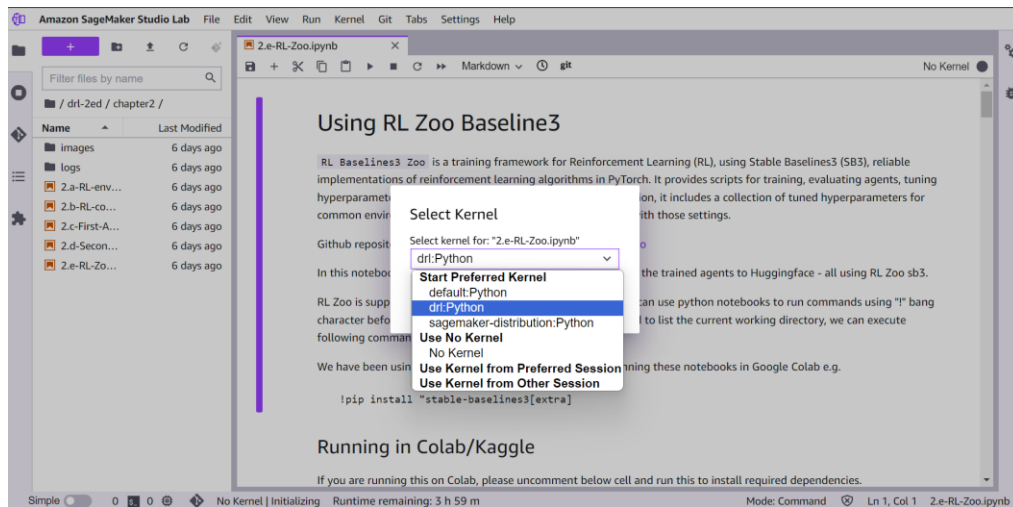


Figure 1-12 AWS Studio Lab

## Running using Lightning.ai

Lightning AI Studio (<https://lightning.ai/>) is a cloud-based AI development platform (similar to Google Colab or Amazon Sagemaker Studio Lab) that aims to eliminate the hassle of setting up local environments for machine learning projects. Some key features of Lightning AI Studio are:

- It integrates popular machine learning tools into a single interface. This allows for building scalable AI apps and endpoints more easily.

- There is no environment setup required. You can code in the browser or connect your local IDE (VSCode or PyCharm). You can also easily switch between CPU and GPU with no environment changes.
- It allows hosting and sharing AI apps built with Streamlit, Gradio, React JS, etc. It also enables multi-user collaboration by coding together.
- It provides unlimited storage and the ability to upload and share files as well as connect S3 buckets.
- It enables training models at a massive scale using thousands of GPUs under paid plans.
- There is a growing list of community templates (Studios) ready for different use cases and different models which can act as your starting point.

Let us look at high level steps to run code on Lightning.ai studio.

1. First go the url <https://lightning.ai/> and signup.
2. Next you click on “New Studio” button on the top right, choose defaults on the prompt and then click “Start”.
3. Once a studio starts, you can click on the terminal on right side of the studio and use the terminal to a) install apt-get packages b) clone the source code repository and c) do “`pip install -r requirements.txt`” to install required Python packages and libraries.
4. At this point you are ready to start executing the Jupyter notebooks. At the time of writing, the free tier version offers a 4 CPU configuration making it a powerful experimentation platform in free tier offering. You also get access to GPU resources.

## Other Options to run code

There are many more choices. In case you want to run the code on GPU machines for graphic games or 3-d environments, you can use many paid choices. A few of them are listed below. Depending on your need, you should refer to the documentation of the relevant platform to set it up for code execution:

1. Google Vertex AI notebooks
2. AWS Sagemaker
3. MS Azure ML Studio

4. Paperspace
5. Lambda labs
6. Jarvis Labs

There is a growing list of cloud GPU and CPU providers for Deep Learning workloads. For our purpose of running the source code of this book, any entry level configuration is good enough.

## Summary

In this chapter, we started by introducing the field of reinforcement learning and the history of how it has evolved from a rigid rule-based decision-making system to a flexible optimal behavior learning system, learning on its own from prior experiences.

We talked about the three sub-branches of machine learning, i.e., supervised learning, unsupervised learning, and reinforcement learning. In the context of Supervised and Unsupervised methods, we also discussed emerging hybrid approaches such as Semi-Supervised, Self-Supervised learning as Generative AI.

We compared the three core approaches of supervised, unsupervised and reinforcement learning to elaborate on the context where each of these make sense. We also talked about the subcomponents and terms that comprise a reinforcement learning setup. These are *agent*, *behavior*, *state*, *action*, *policy*, *reward*, and *environment*. We used the example of a car and a robot to show how these subcomponents interact and what each of these terms means.

We talked about the concept of reward and value functions. We discussed that rewards are short-term feedback and that value functions are long-term feedback of the agent's behavior. Finally, we introduced model-based and model-free learning approaches. Next, we talked about the influence of deep learning in the field of reinforcement learning and how DQN started the trend of combining deep learning with reinforcement learning. We also discussed how the combined approach has resulted in scalable learning including from unstructured inputs such as images, text, and voice.

We moved on to talk about the various use cases of reinforcement learning citing examples from the fields of autonomous vehicles, intelligent robots, recommender systems, Large Language Models and stock trading, healthcare, and video/board games.

Finally, we walked through various ways to set up a Python environment and be able to run the Jupiter notebooks accompanying this book.