Task 0 – Python, OpenCV & PyTorch Installation (for Ubuntu)

NOTE: We recommend to use **Ubuntu OS** for Machine Learning, since we have thoroughly tested the software installation on it at our end. If you still wish to use **Windows OS**, kindly refer to this document.

This document contains instructions to install the following software/libraries on **Ubuntu OS**:

- Anaconda for Python 3
- OpenCV
- PyTorch
- Visual Studio Code ***(optional but highly recommended)***

The installation of all software/libraries has been tested on **Ubuntu 16.04** and **18.04**. We recommend you to use one of these versions of **Ubuntu OS**. These software/libraries have to be installed **ONLY ON 64-bit OS**.

1. **Anaconda for Python 3:**

   We will be using Anaconda for Python 3.6. Anaconda is an open source Python distribution that has many of the packages like numpy, scipy, matplotlib, scikit-learn, etc. required for data science and machine learning preinstalled.

   - Download **Anaconda for Python 3** for **64-bit** OS (here).
   - We need to make this downloaded file executable. Right-click on the folder where the downloaded file is present and select **Open in Terminal** option.
   - Type "**sudo chmod u+x Anaconda3-5.3.0-Linux-x86_64.sh**" for **64-bit** OS file.
   - To install Anaconda, type in Terminal: "**./Anaconda3-5.3.0-Linux-x86_64.sh**" for **64-bit** OS.
   - Read and follow the instructions, accept the license terms by typing "**yes**". Confirm the location of installation by pressing Enter. It will install the necessary libraries.
   - Type "**yes**" when asked to initialize **Anaconda3** in **bashrc**.
   - At the end, it will ask whether to proceed with installation of Micorsoft VSCode. Type "**no**"; we will install it manually later.

- Congrats, **Anaconda for Python 3** is successfully installed. Let's verify now.
- In Terminal, type "**conda info**". You will see the list of information related to the Anaconda, the output will be as shown in Figure 1.

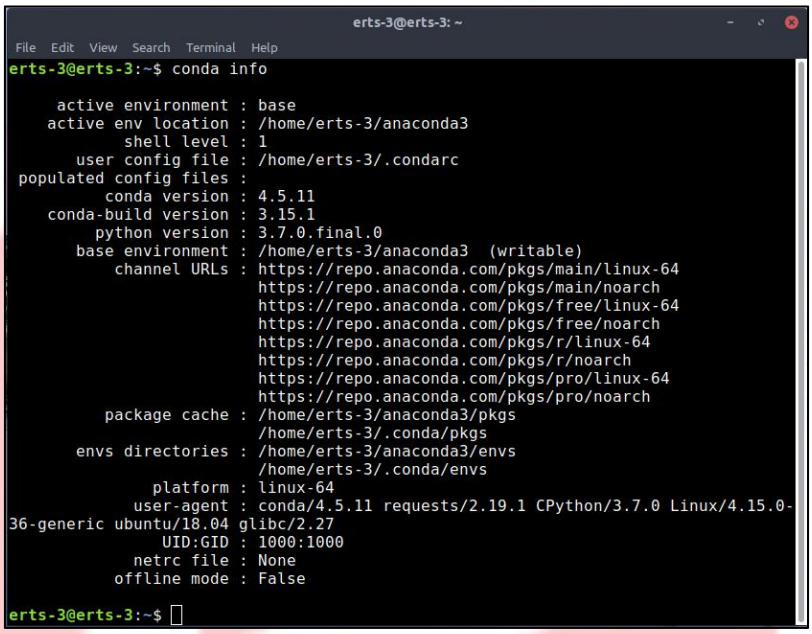

Figure 1: "**conda info**" output

- Check the path and version of default Python that comes with Anaconda installation, by typing "**which python**" and "**python**". You will see similar output as in Figure 2. Your **python** version (***3.7.0***) as shown above might be different but that's okay as long as it is ***3.7.x***.

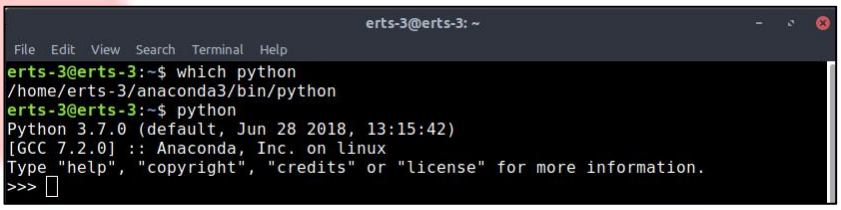

Figure 2: Default output of "**which python**" and "**python**"

- Anytime you are creating a Python project/s it is better to keep packages separate as per the requirements of the project/s. This is required because different packages may be dependent on different version of other packages and there is a chance of conflict because of presence of existing packages of some other version. For example, for web development projects, you can have a separate isolated environment and another one for machine learning projects. We can do this by creating a Python **Virtual Environment**. So, let's create one environment for the Tasks in Stage 1.

- Anaconda creates a default environment named **base** at the installation directory, let's check this out. Type "**conda env list**" to get the output as shown in Figure 3. The * against the environment name indicates the current active environment.

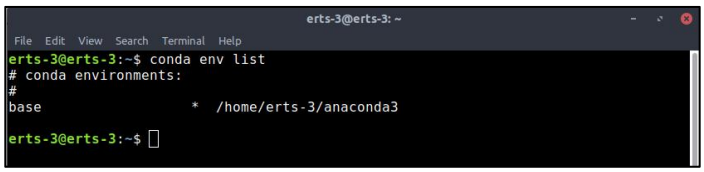

Figure 3: "**conda env list**" output

- Create an environment for Stage 1 with the name "***HC#<Team_ID>_stage1***" by tying the command "**conda create -n HC#9999_stage1 python=3.6 anaconda**" if the ***Team_ID*** is ***9999***. Replace ***<Team_ID>*** with your **Team_ID**. The prefix **-n** means the next argument will be name of the environment. We will be using **Python 3.6**, hence we are creating the environment by specifying the Python version. If we do not specify this, Anaconda will create an environment with the default Python version of **3.7**. After creating the environment, you will see the output on terminal as in Figure 4.

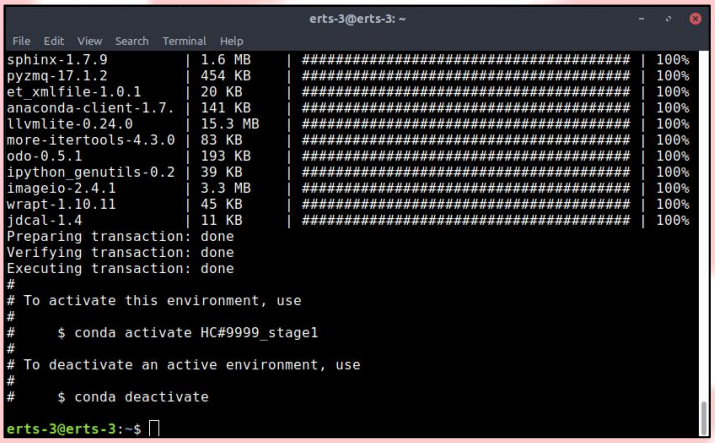

Figure 4: "**conda create -n HC#9999_stage1 python=3.6 anaconda**" output

- Before activating our environment, let's check the environment list by typing "**conda env list**", you will see two environments. First, the **base** default environment with its path and * against name (meaning its currently active) and second, your environment name with its path as in Figure 5.

- Activate your environment by typing "**conda activate HC#9999_stage1**". Replace ***HC#9999_stage1*** with your environment name. Check the environment list again using command "**conda env list**", you will see that * is now against your environment name indicating it is currently active as shown in Figure 5. You shall also see your environment name in parentheses "**(HC#9999_stage1)**" at start of the terminal line as soon as you activate the environment. For deactivating the environment, just type "**conda deactivate**".

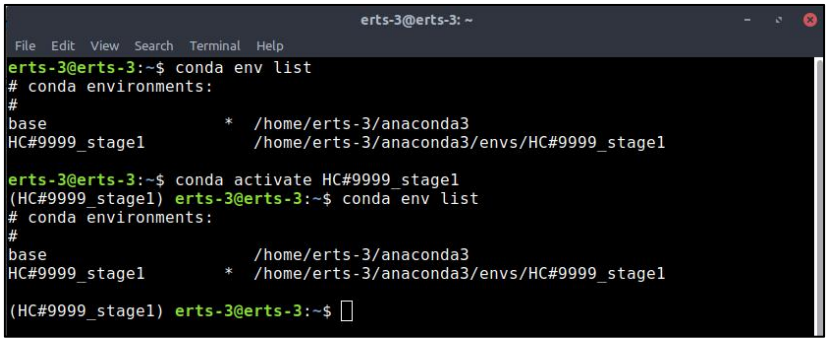

Figure 5: "**conda env list**" output before and after activating your environment

- Now, let's check the path and version of Python (**3.6**) we provided while creating the environment. First, activate your environment if it's not active and then type "**which python**" and "**python**" to see the output as shown in Figure 6.

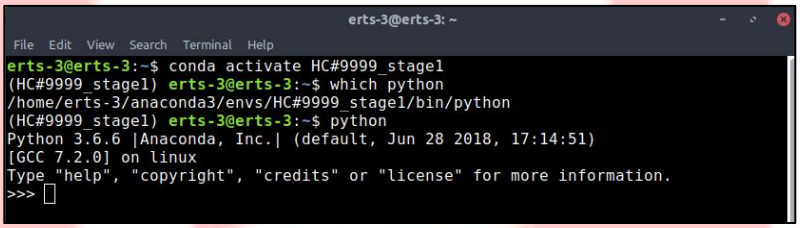

Figure 6: "**which python**" and "**python**" output after activating your environment

- Congrats, you have installed **Anaconda** and setup the **virtual environment** for Stage 1 on **64-bit Ubuntu OS** successfully.
- To run a code for any task in Stage 1, activate the environment first using "**conda activate *your_env_name***" and then proceed for testing your code.
- **NOTE:** If you wish to install any package/library using **conda** or **pip**, first activate your environment and then type "**conda install *package/library_name***" or type "**pip install *package/library_name***" respectively.

2. OpenCV:

We need OpenCV for performing all sorts of image operations and computer vision processes, from loading an image in Python, cropping or re-sizing an image to find contours, and so on.

- Activate your environment using command: "**conda activate HC#9999_stage1**".
- Run command to install OpenCV from **pip**: "**pip install opencv-contrib-python**".

3. PyTorch:

PyTorch is a deep learning framework much like Tensorflow, but it has its own differences. The major difference between Tensorflow and PyTorch is in building of computational graph. In fact, that is what we will be doing mostly in Stage 1. Although this discussion won't make much sense if you haven't used any one of the machine learning frameworks before but you can still read about it here. To conclude, PyTorch is more natural to use and has a very good interfacing with Python than Tensorflow.

PyTorch can also be used as a scientific computation framework (actually its mostly that) like NumPy but the difference will be that PyTorch can also leverage the power of GPUs providing increased computation speed and performance. PyTorch can be installed for **CPU** or **GPU** (needs **NVIDIA GPU** on your system). If you don't have **GPU** on your system, follow steps for **CPU** else go for **GPU**.

For CPU:

- Activate your environment using command: "**conda activate HC#9999_stage1**".
- Run the command: "**conda install pytorch-cpu torchvision-cpu -c pytorch**" to install via Anaconda. If this doesn't work, install using **pip**, you can find the command on this webpage. For **CUDA**, select **None** for **CPU**.

For GPU:

- Activate your environment using command: "**conda activate HC#9999_stage1**".
- Install **NVIDIA Graphics Drivers**, **CUDA 9.0** and **cuDNN** by following the steps and resources provided here.
- Verify that **CUDA 9.0** and **cuDNN** are installed properly. Then install PyTorch for **CUDA 9.0**.
- Run this command: "**conda install pytorch torchvision -c pytorch**" to install via Anaconda. If this doesn't work, install using **pip**, you can find the command on this webpage. For **CUDA**, select appropriate CUDA version for **GPU**.

4. Visual Studio Code *(optional but highly recommended)*:

The below steps will help you improve your programming efficiency. Programmers spend a lot of time looking at documentation, re-factoring code, debugging issues, unit testing and styling code. What if you get a helping hand at this so that your code looks

beautiful and you could debug issues faster ? Read further to know more. Don't worry. Since using tooling may be a new thing to most of you it may look difficult but none of this is difficult. It's actually rather very very easy and you'll become a better programmer the earlier you master these tools.

- Download a Text Editor. We recommend Visual Studio Code. We will write a code in Jupyter Notebook. But if you wish to reuse code written in Jupyter Notebook, you will have to move the code in a **.py** file and then just use the file in Jupyter Notebook. When writing a Python program with IDLE you miss out on a lot of things, like auto-complete suggestions, auto-indentation, inline documentation for module/function (like Jupyter Notebook), aesthetics, etc. For efficacy reasons, we will use a good text editor like **VS Code**.
- Set up **VS Code** for **Python** (link).
- Other than auto-completion, inline documentation, etc. **VS Code** also lints your code, helps you debug your code very easily (so you can avoid print statements).

In order to verify the installation of above libraries, follow the steps given in Test_Setup_Read_Me.pdf document in ***2. Test_Setup*** folder.