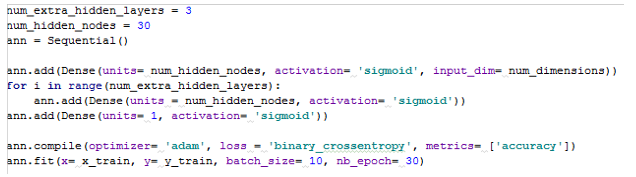
**System Choices**

**Python**

As with most software-related projects, one of the primary choices that must be made is what programming language to implement the components of the system in, along with what development environment it is to be built using. Both of these have a large impact in the time and ease it will take to develop the system, as well as how optimal it will be running in its final variation. For the choice of programming language, we chose to use **Python** **(3.6)** to build all scripts from for the following reasons:

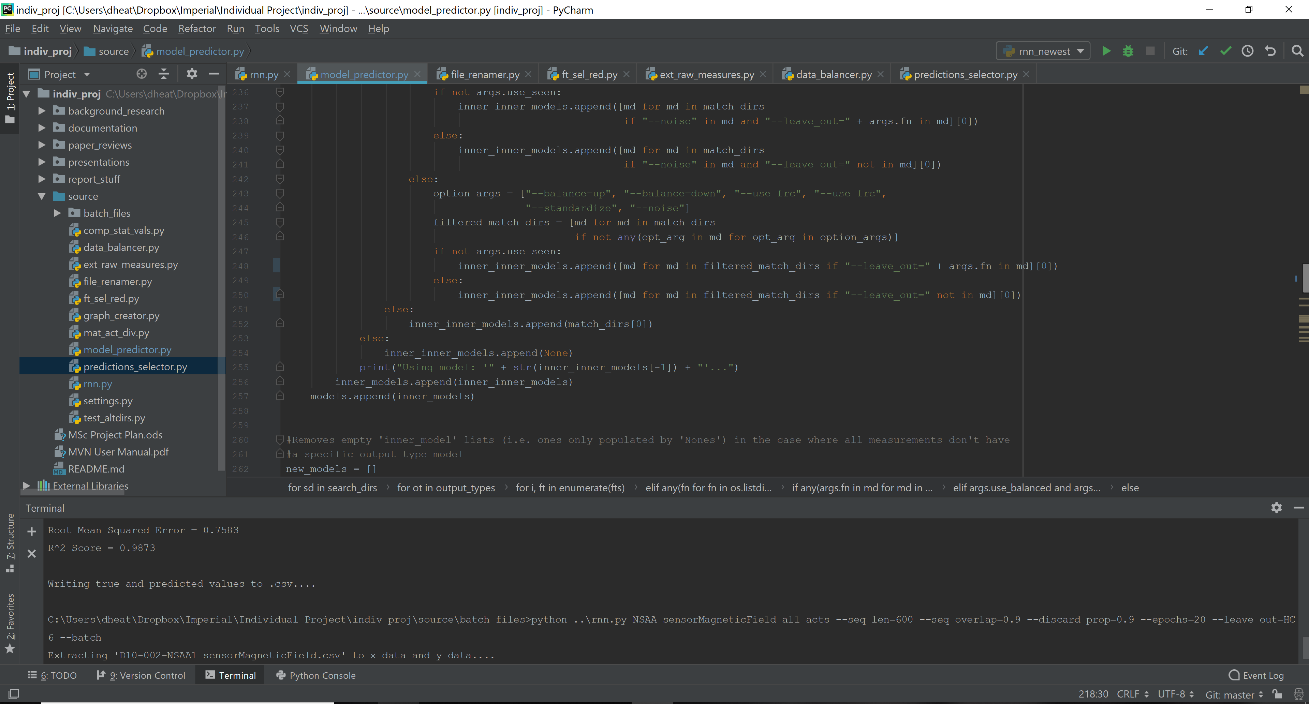
* It takes very few lines to implement many things when compared with other languages like Java; for example, the code below shows how a neural network can be implemented in Python in only 9 lines using the Keras library (a wrapper for TesorFlow):



This enables faster development and easier testing of new ideas and concepts than other languages, as we can afford to care less about problems of tricky syntax (e.g. with using C++) and can instead move towards development ‘at the speed of thought’.

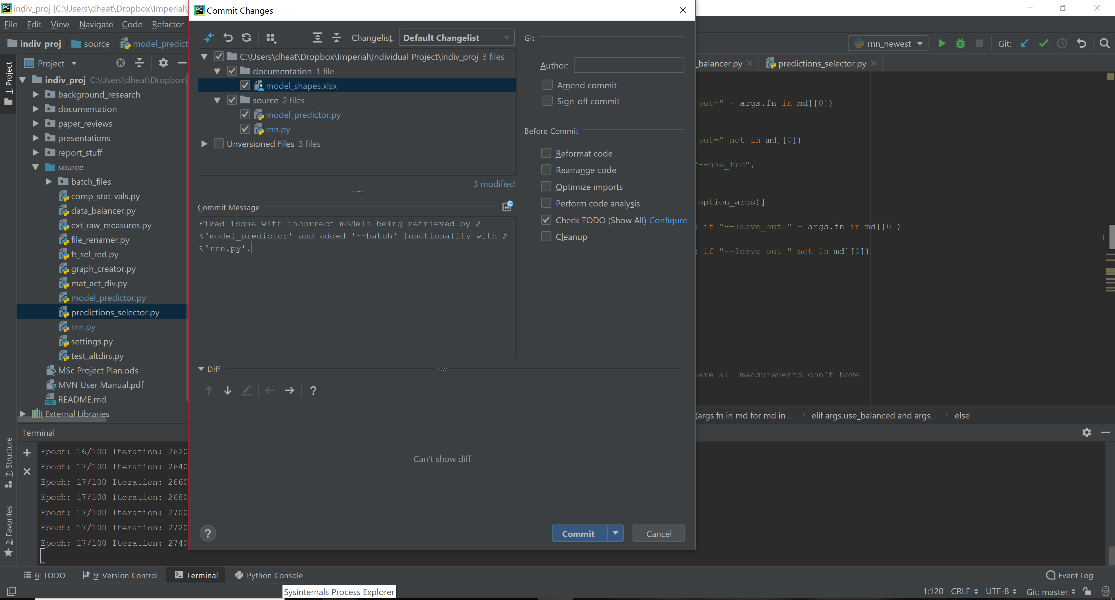
* The open-source nature of Python's community means that there are very often libraries that others have built that fit the profile of what we need, so we don't need to 'reinvent the wheel' by creating our own version of it; this can be seen in the system’s reliance on external functions from ‘scikit-learn’ to compute metrics, along with ‘pandas’ to handle the reading and writing to and from ‘.csv’ files, as opposed to writing our own functions to carry out this functionality.
* Python has seen extensive use for building and testing machine learning and deep learning models by research and business communities; thus, it is easily the most well-developed with regards to open-source libraries such as TensorFlow, with TensorFlow's Python API being most complete of its various language implementations [14].
* Many of the frameworks and libraries that power the system we have built (e.g. NumPy and SciPy) build upon lower-layer Fortran and C implementations for fast and vectorized operations for multidimensional arrays, which helps overcome the inferior speed of a scripting language like Python when compared to these lower-level languages [15].
* Further development of the system is easier, as being written in a language like Python (which is close to English) lends itself to the easier understanding of what is going on within each script. Coupled with variables having intuitive names and commenting where necessary, this helps with anyone undertaking edits or rewrites to one or more of the scripts in the future.

**Integrated Development Environment**



With regards to where we shall develop the Python programs for this project, we have chosen to use the **PyCharm Community Edition 2016** integrated development environment. We can see above a screenshot of how the IDE is setup for this project. This has been chosen for the following reasons over a text editor or another IDE:

* *Multiple program tabs*: This enables the easy transitions between different scripts. This is particularly useful when we are making changes to multiple scripts simultaneously (e.g. adding the same optional argument to both ‘rnn.py’ and ‘model\_predictor.py’ that ensures certain models built by ‘rnn.py’ are then retrieved correctly by ‘model\_predictor.py’).
* *In-built terminal*: This allows us to run programs from command line within the IDE itself without having to open a separate terminal window and navigating to the script directory every time it’s reopened. This is a major quality of life improvement, as most of the scripts are run from the command line with required and optional arguments.
* *Good compatibility with git*: This has two main benefits. The first is that changes made to scripts and their specific lines of change from a previous commit are highlighted in blue, which helps with accounting for modifications made when writing commit messages and keeping track of all recent changes made. The other benefit is the GUI approach to committing to the GitHub repo (as can be seen below) which is a much easier way to make regular commits and also highlights easier more subtle changes made (such as writing lines to output files).



* *Previous experience*: We’ve used it before for many previous Python projects, including in two professional roles, for a final-year undergraduate individual project, and for many pieces of coursework involving the use of several machine learning and deep learning libraries such as ‘scikit-learn’ and ‘TensorFlow’. Hence, this previous experience and the resultant familiarity with the environment helps make development of this project a more expedient and easier experience.
* *Debugging and error handling*: The layout of PyCharm makes for writing and immediate testing and modifying of code very simple, with debug options showing locations of compilation errors very easily and with clarity. This minimizes the time lost in development due to basic syntax errors and other basic software-engineering-related issues.
* *Package implementation*: It’s easy to add additional packages via 'Available Packages' in the 'Project Interpreter', which is useful as we need to add many additional libraries, from TensorFlow to simple libraries like ‘pyexcel’.

**TensorFlow**

For programming in Python, there are numerous options for which library we can use to implement our central RNN models. One option is the ‘Keras’ wrapper that wraps the TensorFlow framework. Although this is a lot simpler to use and has fewer aspects to manually code, there are numerous advantages that TensorFlow has over this that includes the following:

* More extensive and highly detailed documentation and examples for TensorFlow.
* Higher amount of direct control over the RNN models with things such as weights and optimizers.
* Better performance with TensorFlow through threading and queues to speed up the training process.
* Availability of the TensorBoard visualization tool to help understand our models.

Thus, using TensorFlow will hopefully lead to more successful RNN models that can better learn from raw measurements and computed statistical values that can thus better operate on newly-presented subject data. Preparatory work for using this framework includes prior use as the engine for an undergraduate individual project, along with reading Chapter 14 and 16 of [15] where we learned:

* The benefits of using TensorFlow for neural network training performance in utilizing GPU cores, where using a high-end GPU resulting in ~15 times more floating-point calculations per second than using an equivalently-priced CPU.
* Concepts of graphs, sessions, ranks, tensors and operations, which gave clarity to the concepts of the computational graph structure used by TensorFlow.
* The 'placeholder' concept of TensorFlow where a variable is an 'empty' variable that expects data input (in our case, these 'placeholders' will be implemented for 'x\_train' / 'x\_test').
* How aspects specific to an RNN works in TensorFlow, such as implementing layers as LSTM cells and initial/final states for the variables.

With this obtained knowledge from the above textbook, along with examination of other examples found primarily on GitHub or the TensorFlow documentation website, we felt confident enough in our knowledge of the TensorFlow library that, coupled with prepared input data, our RNN models could now be created.