1. **Novelty**: The following attributes of the project make it either novel, unique, interesting or previously unseen in this combination:
    - Prior academic research related to ML-based bug detectors do not attempt to solve more than one or at most two bug types using non-ensemble methods. This tool will attempt to train a neural network to identify a variety of bug types including but not limited to argument swapping, missing function calls, absence of parenthesis, mismatched indentation and incorrect operators.
    - I have not come across a research paper that performs a genuine comparison between ensemble and non-ensemble methods with regards to their accuracy in bug detection. This tool will compare ensemble methods from scikit-learn with various neural network combinations from tensor flow with regards to various classification metrics for bug detection.
    - There does not currently exist a free, easy and readily available dataset of 'bug-free' Python code with corresponding 'buggy' Python code, all appropriately labelled for consumption by a classification algorithm. Therefore, this work involves an input pipeline that consumes bug-free Python code from respective project folders, artificially introduces various bugs of different types wherever possible and then stores them in a csv file for easy consumption by the pandas module in Python.
    - In addition to comparing the performance of a variety of ML methods for the classification of bugs, this work will also compare both the ease and performance of different vectorization tools that process the code for consumption by the ML classifiers.
2. **Value to User Community**: This tool is being written for consumption by Software Developers that work with Python as their primary development language.
    - While there do exist multiple bug detector tools to catch bugs in Python, the vast majority of them do not use any form of ML. Given that the academic community has outlined various reasons why ML is the next step in improving the performance of static analysis powered bug detectors, it is somewhat surprising that the greater part of academic research in this field does not address the fastest growing language currently being used.
    - Moreover, from the results of the survey that I conducted for my midterm paper, it was evident that the most important feature to Software Developers in a bug detector tool is the ability to detect a wide variety of bugs.
    - Therefore, the tool that I am working on will not only address the absence of research-based ML bug detector tools written in Python for Python developers but will also help the user community solve its greatest concern with regards to bug detection - detecting a wider variety of bugs.
    - In addition to the tool itself, the comparisons drawn up by the supporting work, such as those between ensemble methods and non-ensemble methods and those between the various vectorizers will give users a clear understanding of how and why the tool comprises of the components it does thereby empowering users to make informed changes to the tool on their own should they be required.
    - Last of all, the designed input pipeline allows for developers to a) artificially introduce bugs - even of their own design - into code specific to their project/ domain and b) train the detector on their own 'bug-free' and artificially 'buggy' data to provide better classification accuracy for their specific domain.
3. **Datasets**: Most, if not all of the training data being used for this project comes from Github with the following attributes to note:
    - All the bug-free data was taken from popular Python repositories by different authors on Github that contain Python code written for a multitude of purposes.
    - The code extracted therefore did not follow a particular style of coding and contained substantial diversity in terms of functionality and function.

- The codebases were written and hosted to be consumed by members with varying levels of experience in the Python community therefore ensuring that they were well organized and of differing levels of complexity.
- The codebases selected were divided into a train dataset and a test dataset. The train dataset was further subdivided into training (80% of the code) and validation (20%) of the code - ensuring that each of the three categories for training and testing an ML algorithm were covered.
- Each of the codebases were examined to ensure that the artificial bugs being generated by the input pipeline could be created within a substantial portion of the lines of code thereby ensuring that no class was either under or over represented in training. The training dataset covered approximately 80,000 lines of code across the different classes - 'No Bug', 'Arguments Swapped', etc.
- Example of one of the many Github repos scraped for usable code:https://github.com/OmkarPathak/Python-Programs

4. **Comparison Subjects**: Given that there is substantial overlap between the research papers I examined as part of my midterm paper, and the tool that I am working on for my final project, I will be using relevant, similar experiments (and the results thereof) from the aforementioned papers to compare the results of my work/tool to.
5. **Additional Details**: Below are additional details surrounding my project:
    - Once both the database of Python repositories I am using as my input data as well as my own codebase are finalized, I will be uploading all of the above in a well-organized structure to a Github repository which will be accessible to the course staff.
    - I will be using Jupyter notebooks for my own codebase. The notebooks will contain the results from previous runs so that a user need not run the entire training or test suite themselves to see the results generated. In addition, Jupyter notebooks offer Markdown cells that are better suited for periodic explanation of results and code than in-line code comments.
    - If a user would like to run the code themselves, the Github repository will be set up to support any of the Jupyter notebooks being run from start to finish within the repository once it is downloaded to a local machine.
    - The open-source Python modules I will be using are primarily (although not the complete list) Tensorflow, Scikit-learn, Numpy, Matplotlib, Pandas and Keras.