1. **Requirements:** In order to run the source code, Python 3.7 and necessary packages need to be installed. The commands to install packages are in `install.sh`. We may need to download some data packages used for `nltk` and `spacy` if they are not available on the local machine.
2. **Classical models**
   Source code: classical/classical_clf.py
   Explanation:
   - Text processing: convert all text into lowercase, tokenize the tweet sentence into words. Remove stopwords and non-alpha words. For each word, find its post of speech using `nltk` and we map it to a root work by using a `WordNetLemmatizer`. Some data needs to be downloaded for use with `nltk`.
   - Vectorization: The text in tweets need to be encoded as numbers for use as input of models. Each tweet is encoded by a vector using `TfidfVectorizer`. `LabelEncoder` is used for encoding intensity class.
   - After fitting X and Y vectors from training data for each algorithm, we use those models to predict new X in testing data.

   Run: cd to `classical` folder, run the command: `python3 classical_clf.py`. The result is printed to standard output. A sample result is available in `output.txt` in the same folder.

3. **Deep learning models**
   Source code: classical/lstm.py, classical/cnn.py
   Explanation:
   - Preprocessing text is similar to the process of building classical models.
   - Vectorization: Each tweet is converted into a vector of size 250 using keras's Tokenizer and `pad_sequences` function from `keras`. For encoding intensity class, we use `get_dummies` from `pandas`.
   - Word embeddings: Each word is represented by an array of D numbers representing its semantic. If V is the vocabulary, then we need a matrix of size |V|xD to store word embeddings. In these models, D = 100.
   - In `lstm.py`, we don't set value for the embedding matrix. In `cnn.py`, the embedding matrix is set with values based on information from `spacy` and its `en_core_web_lg` model.
   - Define the model: In `cnn.py`, the embedding matrix is passed as `weights` to the `Embedding` layer, while it is not used in `lstm.py`.
   - I don't understand the theory of added layers, I just follow some articles from the Internet.
   - Training: Use the `train_X_sequences` and `train_Y` from the conversion step.
   - Check result: `test_X_sequences` and `test_Y` are used for computing accuracy. Sample results are available in `lstm.output.txt` and `cnn.output.txt` in the same folder.

   Run: cd to `deep_learning` folder, run the command: `python3 lstm.py` or `python3 cnn.py`.