

[◀ Return to Classroom](#)[DISCUSS ON STUDENT HUB](#)

Disaster Response Pipeline

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations 🙌🌟🌟🌟

Dear Learner,

Congratulations on making it in this project implementation. 🎯
This is an amazing piece of work you have done here.

I went through all your modules and it clearly demonstrates all the steps. Great job.
I really appreciate your hard work and dedication to this project.

Keep Learning and keep doing the good work.
Good luck !!!

Github & Code Quality

All project code is stored in a GitHub repository and a link to the repository has been provided for reviewers. The student made at least 3 commits to this repository.

Awesome, All project code is stored in a GitHub repository.

The README file includes a summary of the project, how to run the Python scripts and web app, and an explanation of the files in the repository. Comments are used effectively and each function has a docstring.

Perfect, your README file is properly documented mentioning about every step very clearly.

Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.

Nice!,

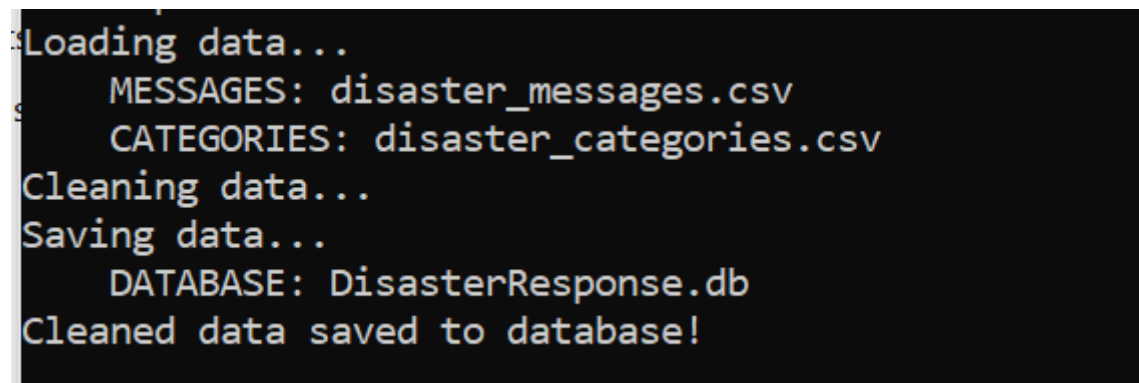
Scripts have an intuitive, easy-to-follow structure with code separated into logical functions.

It's nice to see that you have used proper "docstrings" into your functions.

ETL

The ETL script, `process_data.py`, runs in the terminal without errors. The script takes the file paths of the two datasets and database, cleans the datasets, and stores the clean data into a SQLite database in the specified database file path.

Yes, The ETL script, `process_data.py`, runs in the terminal without errors accepting required arguments.



```
Loading data...
  MESSAGES: disaster_messages.csv
  CATEGORIES: disaster_categories.csv
Cleaning data...
Saving data...
  DATABASE: DisasterResponse.db
Cleaned data saved to database!
```

The script successfully follows steps to clean the dataset. It merges the messages and categories datasets, splits the categories column into separate, clearly named columns, converts values to binary, and drops duplicates.

Perfect, all the steps such as loading, cleaning and merging the databases have been completed successfully.

- [✓] Merges the messages and categories datasets.
- [✓] Splits the categories column into separate, clearly named columns
- [✓] Converts values to binary
- [✓] Drops duplicates

Machine Learning

The machine learning script, `train_classifier.py`, runs in the terminal without errors. The script takes the database file path and model file path, creates and trains a classifier, and stores the classifier into a pickle file to the specified model file path.

Awesome, the script takes the database file path and model file path, creates and trains a classifier, and stores the classifier into a pickle file to the specified model file path.

```
Loading data...
  DATABASE: ../data/DisasterResponse.db
Building model...
Training model...
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5; 1/1] START clf__n_estimators=50.....
[CV 1/5; 1/1] END .....clf__n_estimators=50; total time= 4.1min
[CV 2/5; 1/1] START clf__n_estimators=50.....
[CV 2/5; 1/1] END .....clf__n_estimators=50; total time= 4.1min
[CV 3/5; 1/1] START clf__n_estimators=50.....
[CV 3/5; 1/1] END .....clf__n_estimators=50; total time= 6.8min
[CV 4/5; 1/1] START clf__n_estimators=50.....
[CV 4/5; 1/1] END .....clf__n_estimators=50; total time= 6.3min
[CV 5/5; 1/1] START clf__n_estimators=50.....
[CV 5/5; 1/1] END .....clf__n_estimators=50; total time= 6.9min
```

The script uses a custom tokenize function using `nltk` to case normalize, lemmatize, and tokenize text. This function is used in the machine learning pipeline to vectorize and then apply TF-IDF to the text.

The script builds a pipeline that processes text and then performs multi-output classification on the 36 categories in the dataset. `GridSearchCV` is used to find the best parameters for the model.

Awesome, the script builds a pipeline that processes text and then performs multi-output classification. Also, you have performed hyperparameter tuning using `GridSearchCV`.

The TF-IDF pipeline is only trained with the training data. The f1 score, precision and recall for the test set is outputted for each category.


	precision	recall	f1-score	support
related	0.85	0.93	0.89	4011
request	0.81	0.50	0.62	877
offer	0.00	0.00	0.00	19
aid_related	0.81	0.56	0.67	2168
medical_help	0.55	0.01	0.03	421
medical_products	0.47	0.03	0.06	251
search_and_rescue	0.27	0.02	0.04	145
security	0.00	0.00	0.00	100
military	0.50	0.01	0.02	164
child_alone	0.00	0.00	0.00	0
water	0.93	0.31	0.47	330
food	0.87	0.56	0.68	588
shelter	0.83	0.23	0.36	445
clothing	0.75	0.13	0.22	71
money	1.00	0.01	0.02	120
missing_people	1.00	0.03	0.06	65
refugees	0.00	0.00	0.00	179
death	0.73	0.09	0.17	232
other_aid	0.68	0.06	0.12	666
infrastructure_related	0.00	0.00	0.00	376
transport	0.67	0.01	0.02	237
buildings	0.75	0.04	0.07	247
electricity	0.00	0.00	0.00	96
tools	0.00	0.00	0.00	27
hospitals	0.00	0.00	0.00	73
shops	0.00	0.00	0.00	27
aid_centers	0.00	0.00	0.00	79
other_infrastructure	0.00	0.00	0.00	243
weather_related	0.90	0.58	0.71	1454
floods	0.93	0.31	0.47	433
storm	0.80	0.31	0.45	467
fire	0.00	0.00	0.00	55
earthquake	0.90	0.70	0.79	471
cold	0.00	0.00	0.00	96
other_weather	0.47	0.02	0.05	287
direct_report	0.79	0.39	0.52	991

Deployment

The web app, run.py, runs in the terminal without errors. The main page includes at least two visualizations using data from the SQLite database.

Perfect, the web app, run.py, runs in the terminal properly.
Also, required plots are available on the home page.

When a user inputs a message into the app, the app returns classification results for all 36 categories.

When a user inputs a message into the app, the app returns classification results for all 36 categories. 

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)