Web-based DAS ML System

(WebML DAS)

Software Requirement Analysis

|  |  |  |
| --- | --- | --- |
| Author | Date | Log |
| Rupali Roy | Aug 24, 2020 | Created GUI document |

**DAS File Processing & Anomaly Detection**

1. **Problem Statement:**

The aim of this summer project was to develop a web portal that is used to process DAS: Distributed Acoustic Sensing data and identify anomalies in the data for earthquake detection.

The DAS data is mostly used from PoroTomo and Forge files, for your reference you can refer the following link that has distributed acoustic sensing data acquired from University of Wisconsin Porotomo Ftp server.

[*ftp://roftp.ssec.wisc.edu/porotomo/PoroTomo2/DATA/DASH/20160321/*](ftp://roftp.ssec.wisc.edu/porotomo/PoroTomo2/DATA/DASH/20160321/)

Specifically, we are interested in detecting the earthquake event located in this dataset as below

[*ftp://roftp.ssec.wisc.edu/porotomo/PoroTomo2/DATA/DASH/20160321/PoroTomo\_iDAS16043\_160321000721.sgy*](ftp://roftp.ssec.wisc.edu/porotomo/PoroTomo2/DATA/DASH/20160321/PoroTomo_iDAS16043_160321000721.sgy)

Each dataset includes data recorded in 30 sec

1. **Overall System Design for DAS Data**

The WebML DAS system is an unsupervised anomaly detection portal for DAS data. There are several virtual sensors that are installed at different locations (Channels) that continuously record the ground vibrations along with timestamp and channel number (this indicate the location details). The virtual sensor data is stored in seg-y files recorded data is stored in segy files. The DAS data is stored as seg-y files ( [Society of Exploration Geophysicists](https://en.wikipedia.org/wiki/Society_of_Exploration_Geophysicists) (SEG) for storing [geophysical](https://en.wikipedia.org/wiki/Geophysics) data) on different FTP servers.

To detect anomalies in the seg-y files, we process the DAS data to generate different kinds of plots that can be used to train a deep learning model.

The current webML system uses deep unsupervised clustering to detect anomalies in the data.

Virtual Sensor

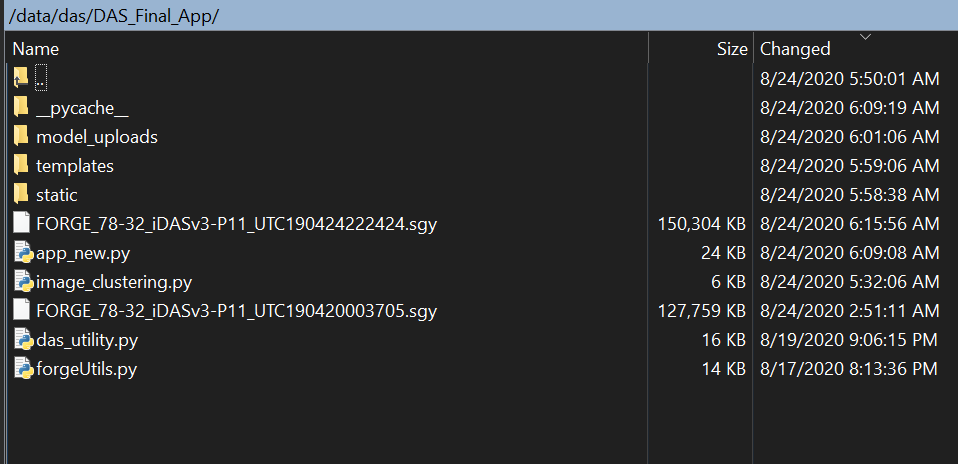
FTP Server for DAS Files

Flask Server

Front End

1. **File Structure for the project**

* The Static folder includes all the CSS styling scripts, JavaScript files and the “Obspy\_Plots” folder which has all the images that needs to be displayed on UI.
* All the HTML files for the front-end needs to be there in the templates folder.
* If we want to display any particular image we need to keep it in the static folder. For this particular project I have included all the images that are shown on the web page in the “static/Obspy\_plots/diff\_plots”. This folder needs to be specified in the app\_new.py file.
* All the JavaScript files that are used in the project are stored in “Static/js” folder.



1. **Steps to run the web application on server**
2. Open Command prompt and enter the command **“ ssh –i .ssh/key** [**cc@129.114.24.214**](mailto:cc@129.114.24.214)**”**

Key is your public key that is used to authenticate you to login to server.

1. Login to server and change your directory to /data/das using the command **“cd /data/das”**
2. The DAS Web App is into **“cd /data/das/DAS\_Final\_App”.**
3. We have a requirements\_new.txt file in this directory, this will help you install all the dependencies for this project. Open the command prompt to install all the python libraries using “**pip install -r requirements.txt** “
4. Run the application by using the following command**: “python app\_new.py”. This will run the web application on port 5000.**
5. **DAS Web Application Flow**

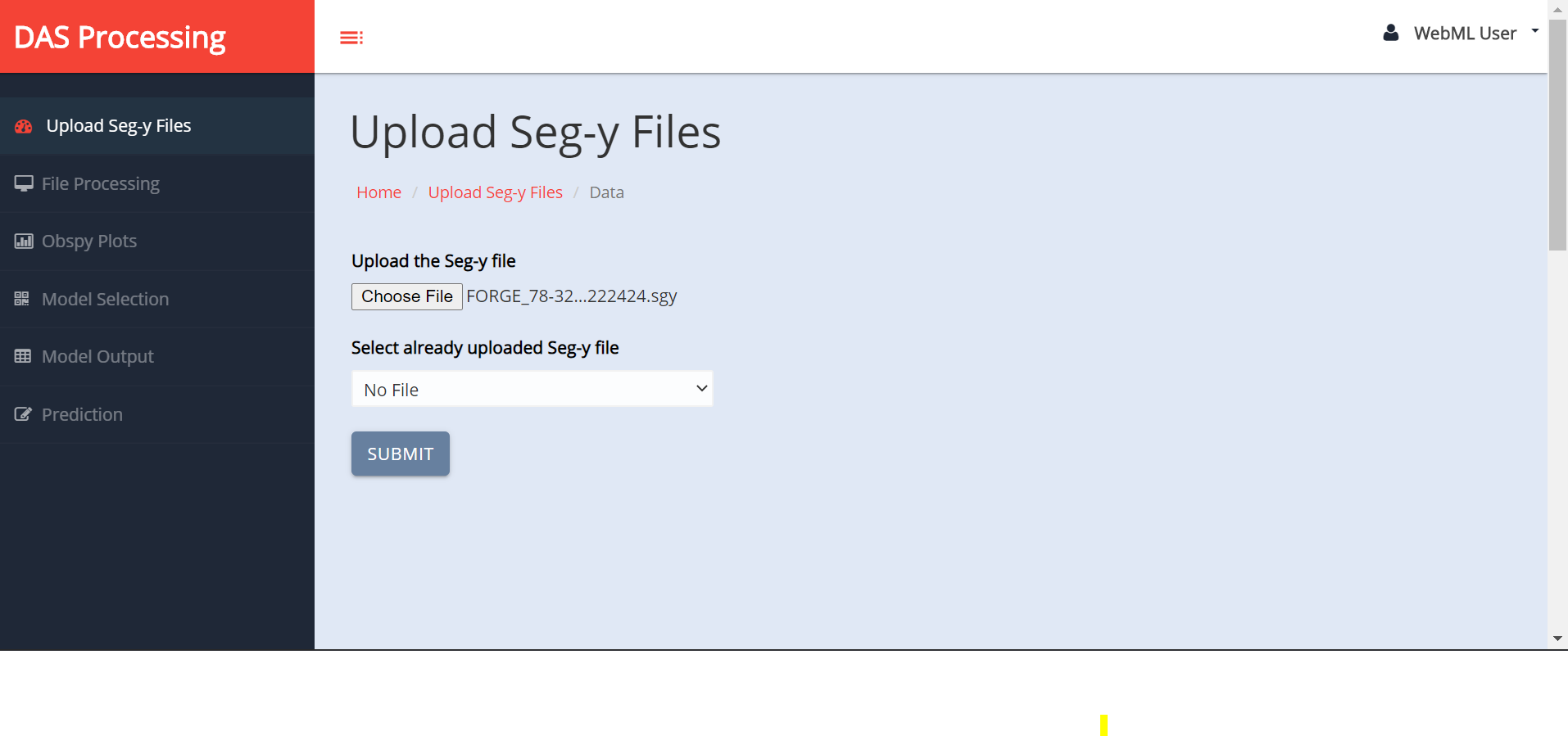
The WebML DAS front end portal is designed to serve the following two purposes:

1. Process the DAS data
2. Determine anomalies in DAS data

**Process the DAS data**

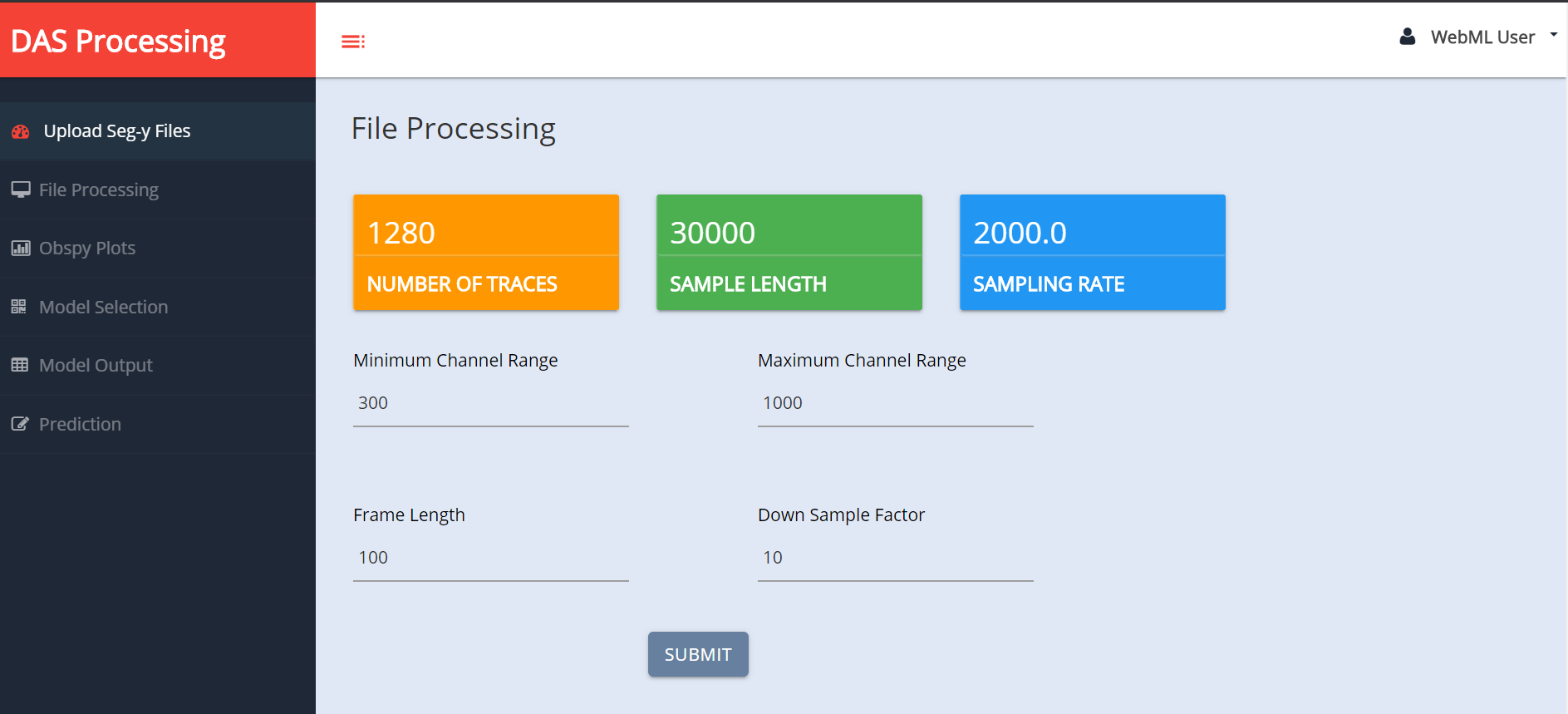
**Step 1. Upload Seg-y files to server**

Use the “Choose File” option on the screen to upload the seg-y DAS data file that needs to be processed. Since file uploading takes time, the UI also has an option to select the already uploaded seg-y files. When we click the submit button on this page, an API endpoint “/process” is called that executes the function “processdata()”. This function reads the uploaded .sgy file and returns the number of traces, sample length of each trace and sampling rate for each trace. The below image is for “DASindex.html” from the templates folder.



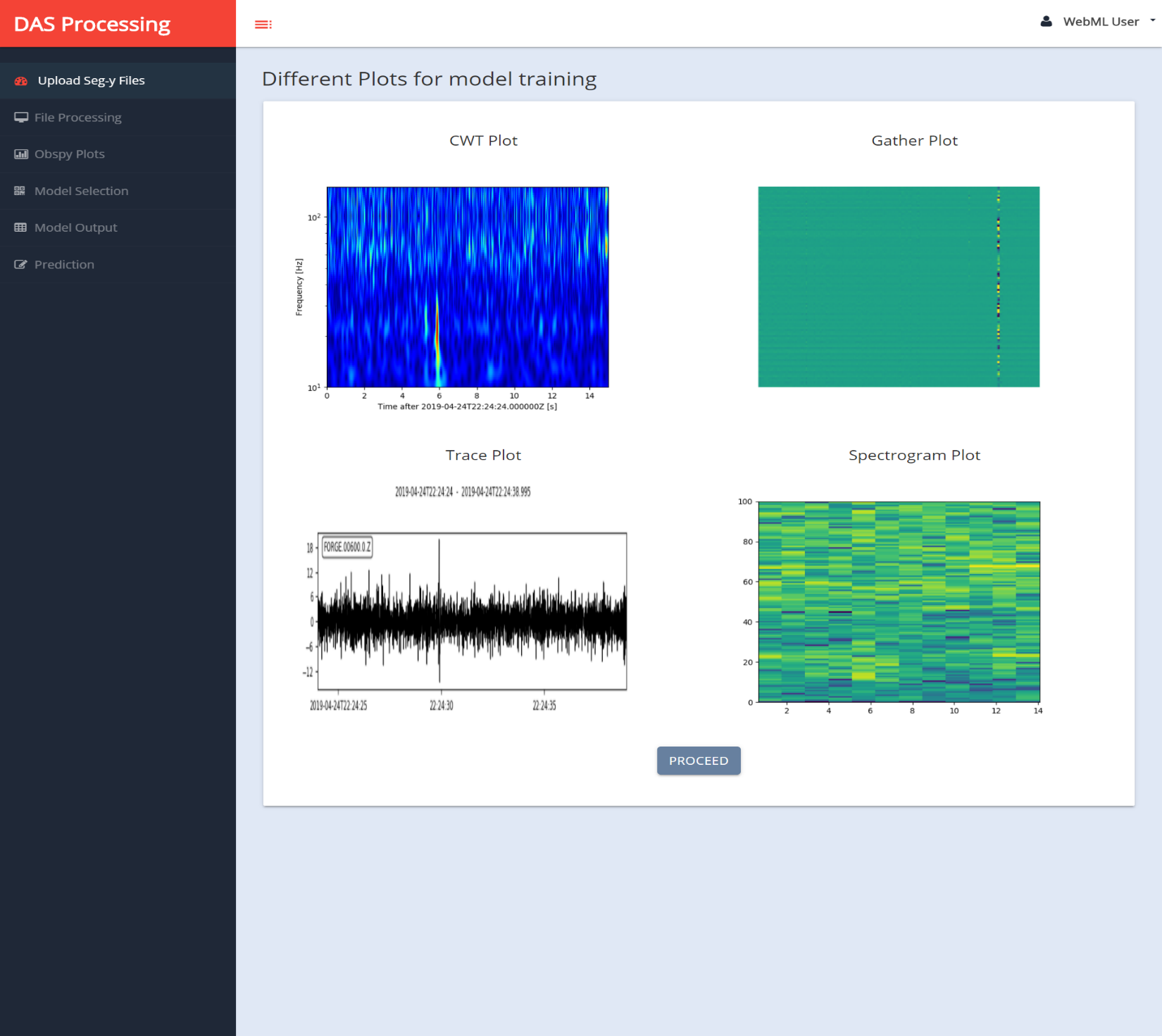
**Step 2. DAS File Processing for Generating different kinds of plots using Obsby Module**

The following screen shows the details about the uploaded seg-y files. It has an input form that allows users to enter the minimum and maximum channel range, the required frame length and down sample factor. These details are used to generate different kinds of plots like Scalogram, Spectrogram, Trace and Gather plots for the input channel range. The Obspy python module is used to generate all these plots. Once the user clicks the “Submit” button all the input details are given as input to the API endpoint “/model” which calls the function display\_plots().



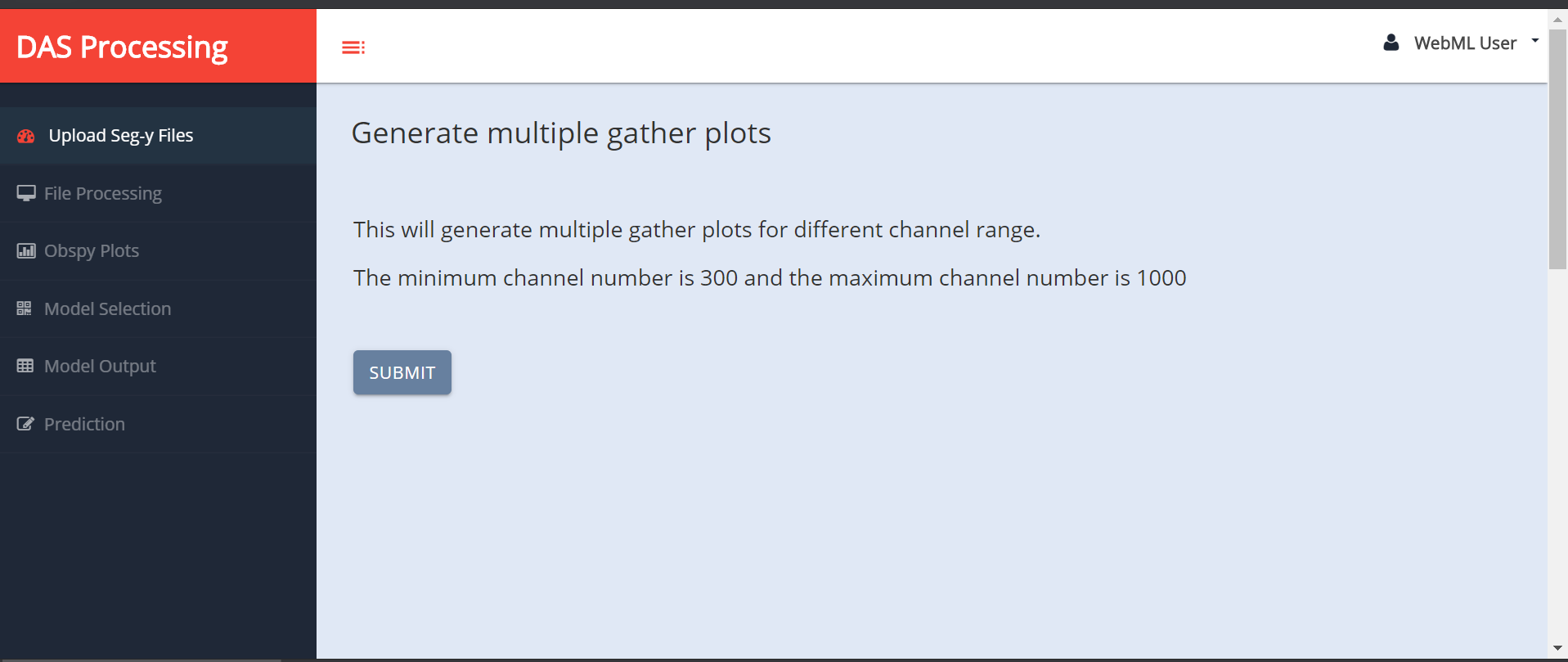
**Step 3. Displaying the generated Obspy Plots**

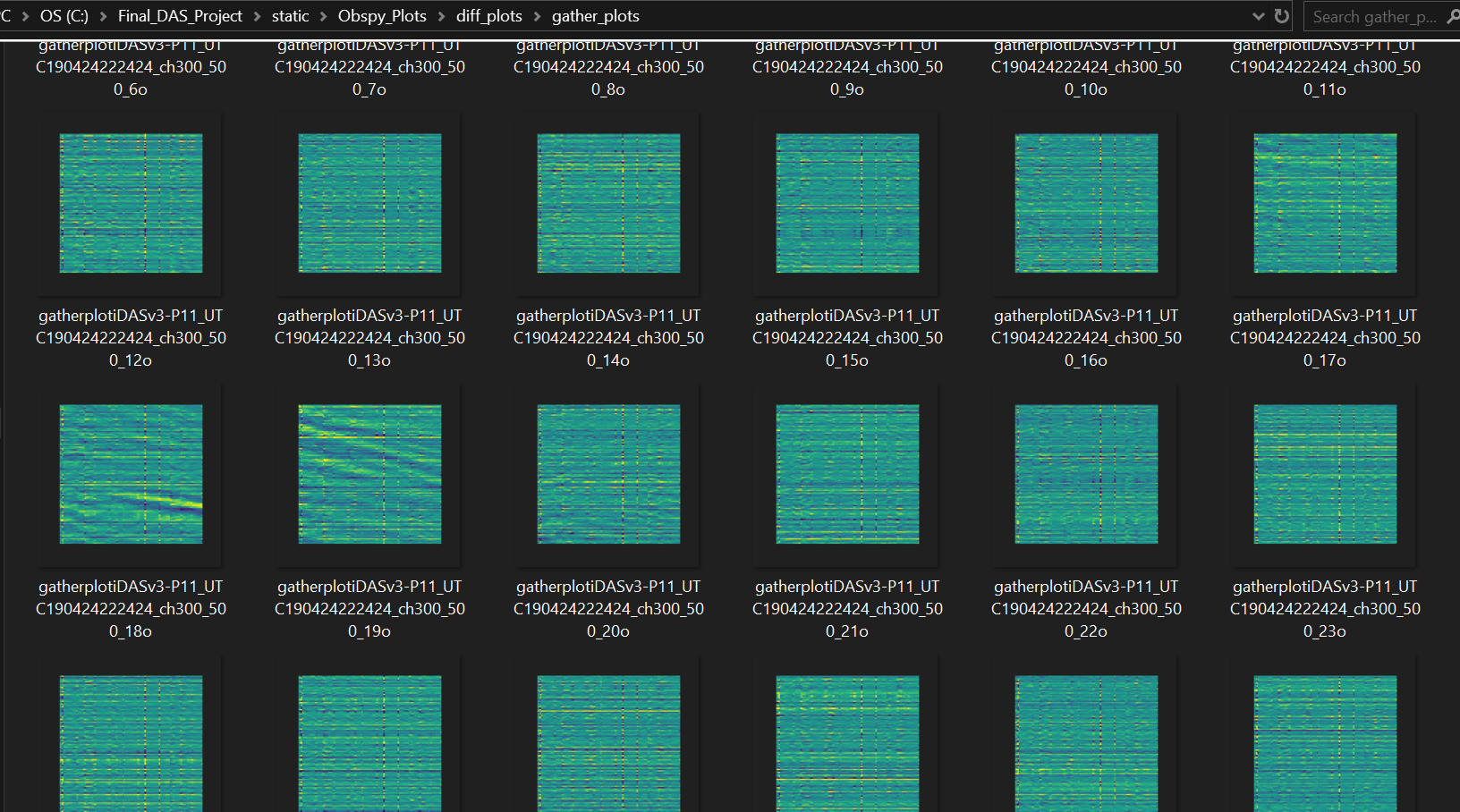
This screen shows the different types of plots that are generated using the inputs from the user related to the file processing. These plots can be used for training our model for a anomaly detection.



**Step 4: Generating multiple gather plots from the uploaded seg-y files**

In this step we are generating gather plots using random channel range. The channels that are used for generating these plots are in the randomly selected, which are in the range of min and max channel that was entered during the DAS Processing stage (Step 2). These gather plots are stored at the following location: *‘static/Obspy\_Plots/diff\_plots/gather\_plots’.* These plots are then used to generate clusters. This step takes a bit time to generate the plots arnd 50-65 secs.

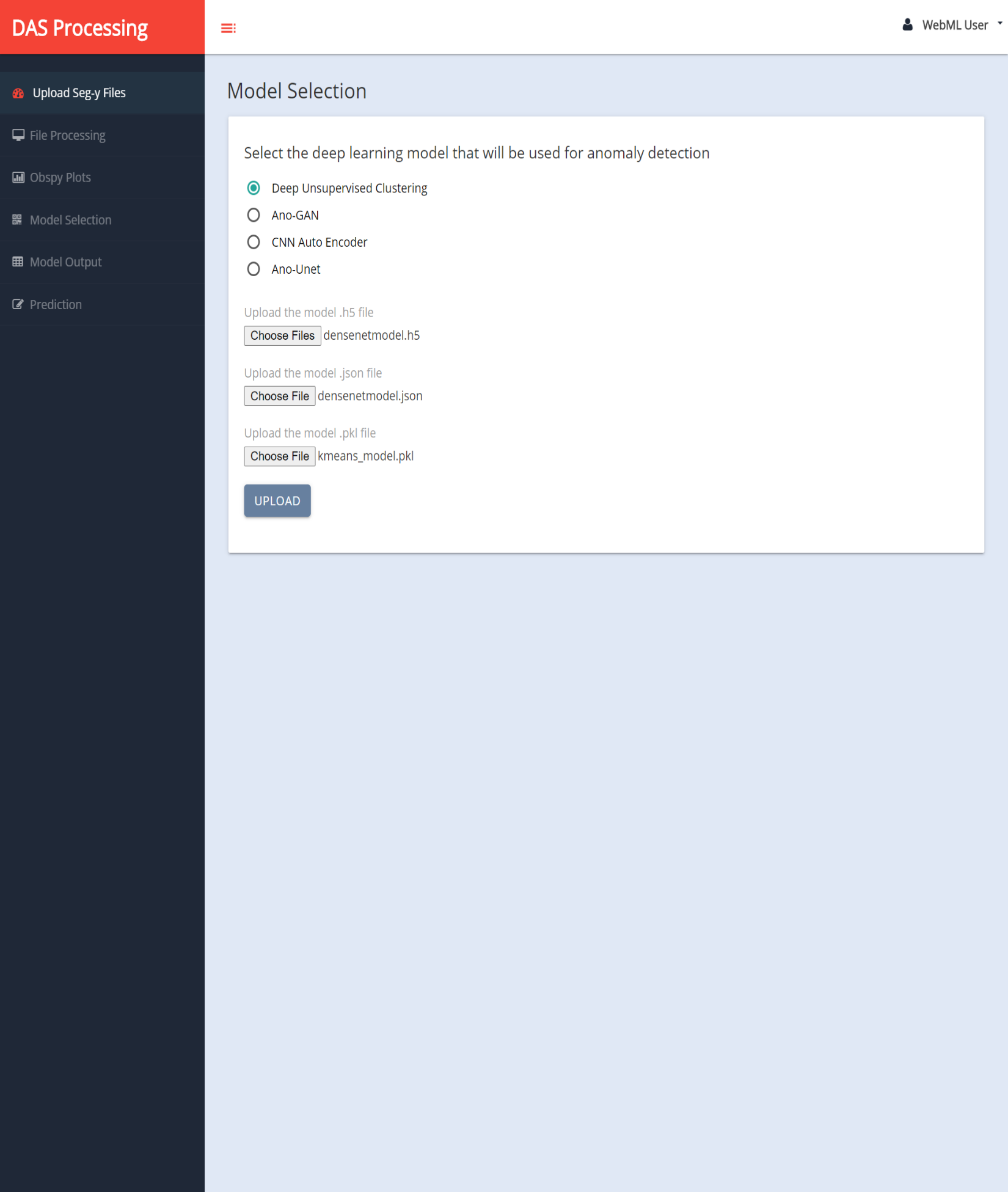




**Step 5: Model Selection**

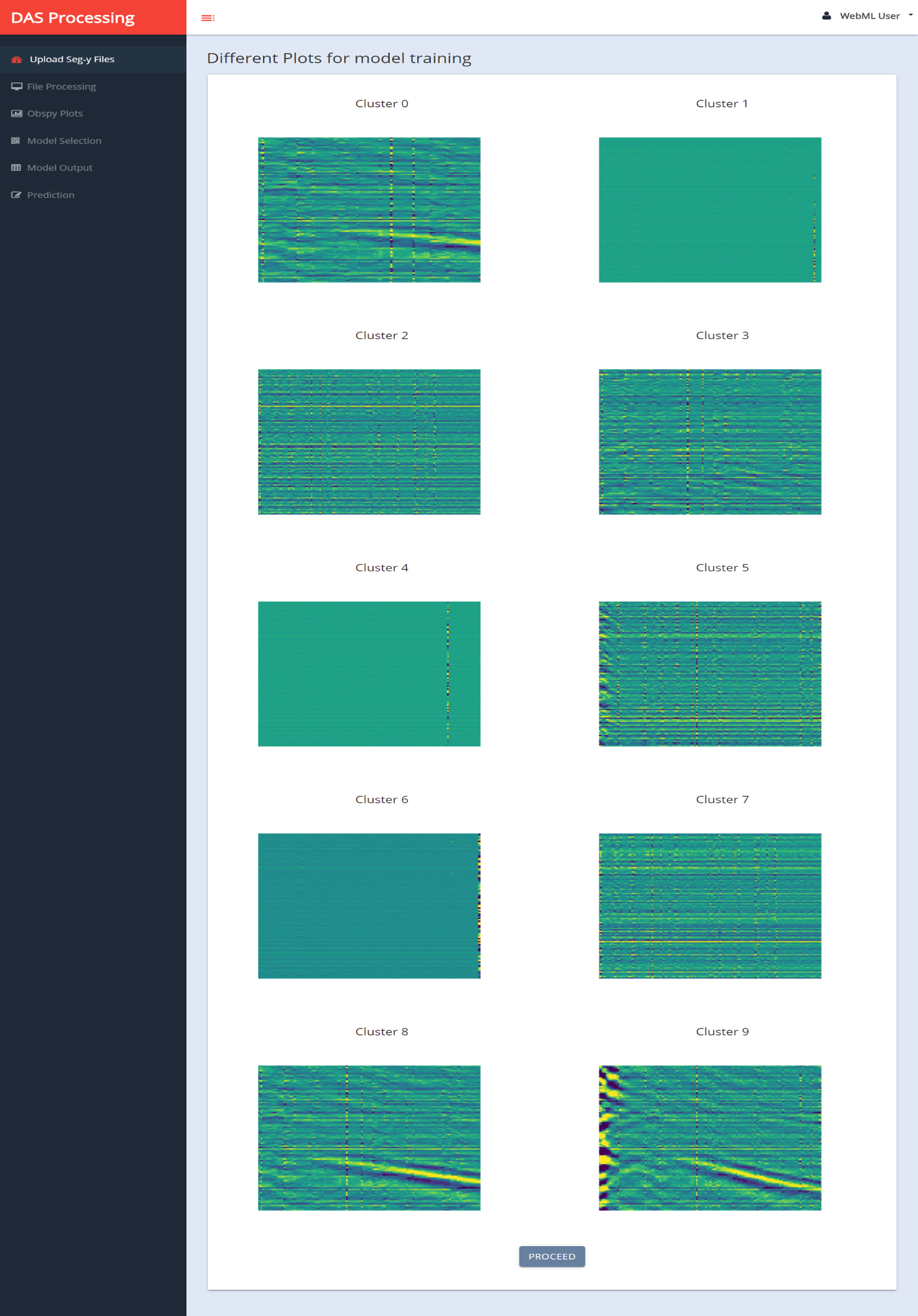
In this step, we can select which deep learning model we want to use for detecting the anomalies in the DAS data for earthquake detection. Currently we have used “Deep Unsupervised Clustering” for anomaly detection, in this we use the image vectors that are generated using the “imagenet” weights and the densenet model.

The GUI allows user to upload the trained model, for eg. In our case we can upload the .h5 and json file for the densenet model and the .pkl file for the Kmeans model.



**Step 6: Image clusters using gather plots**

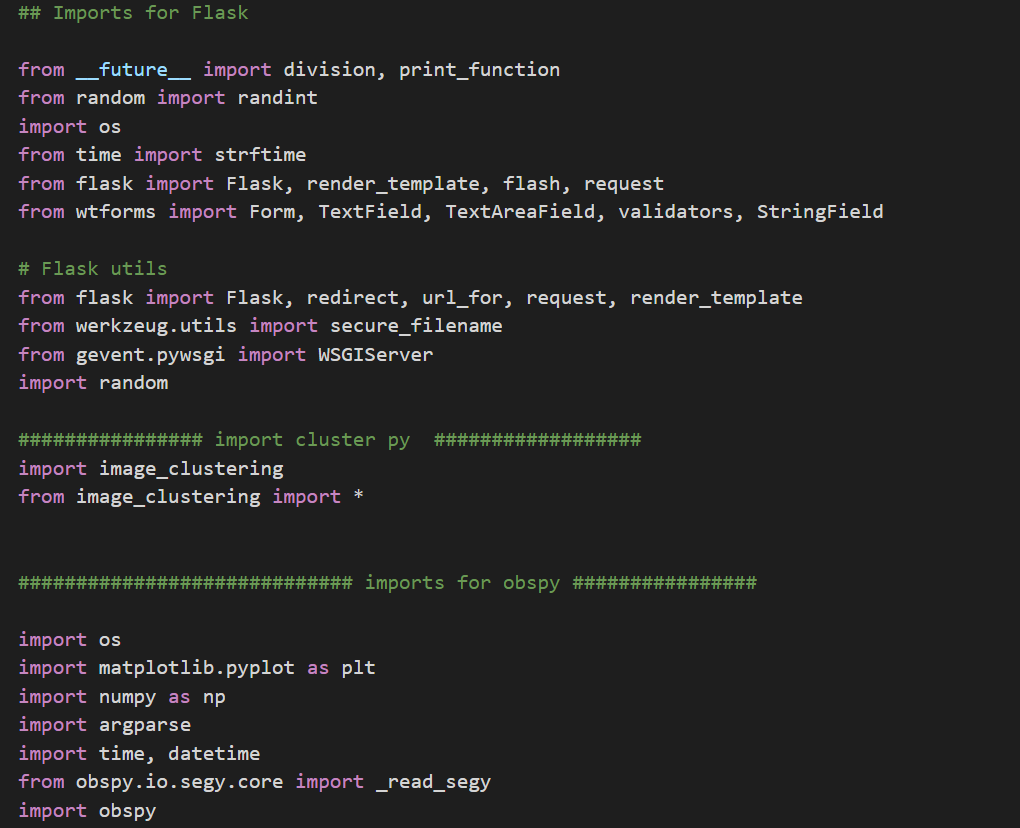
The image below shows a sample image from each cluster. After visual inspection cluster 0,3,8 and 9 are most likely to have gather plots that correspond to anomalies.



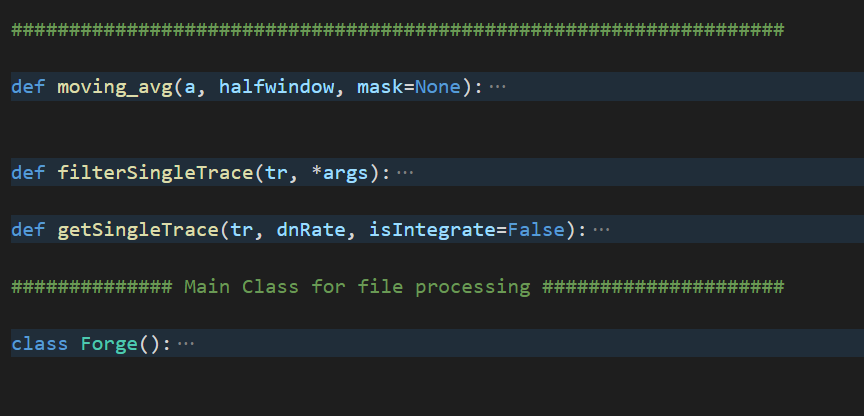
**Step 7: Predict the cluster assignment for a sample gather plot**

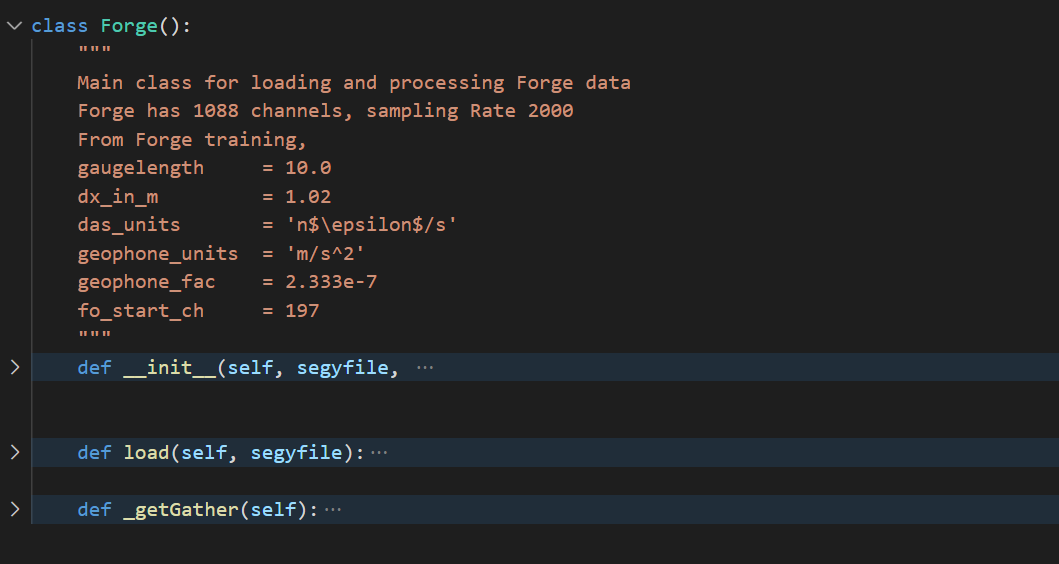


1. **Code Walkthrough for app\_new.py**

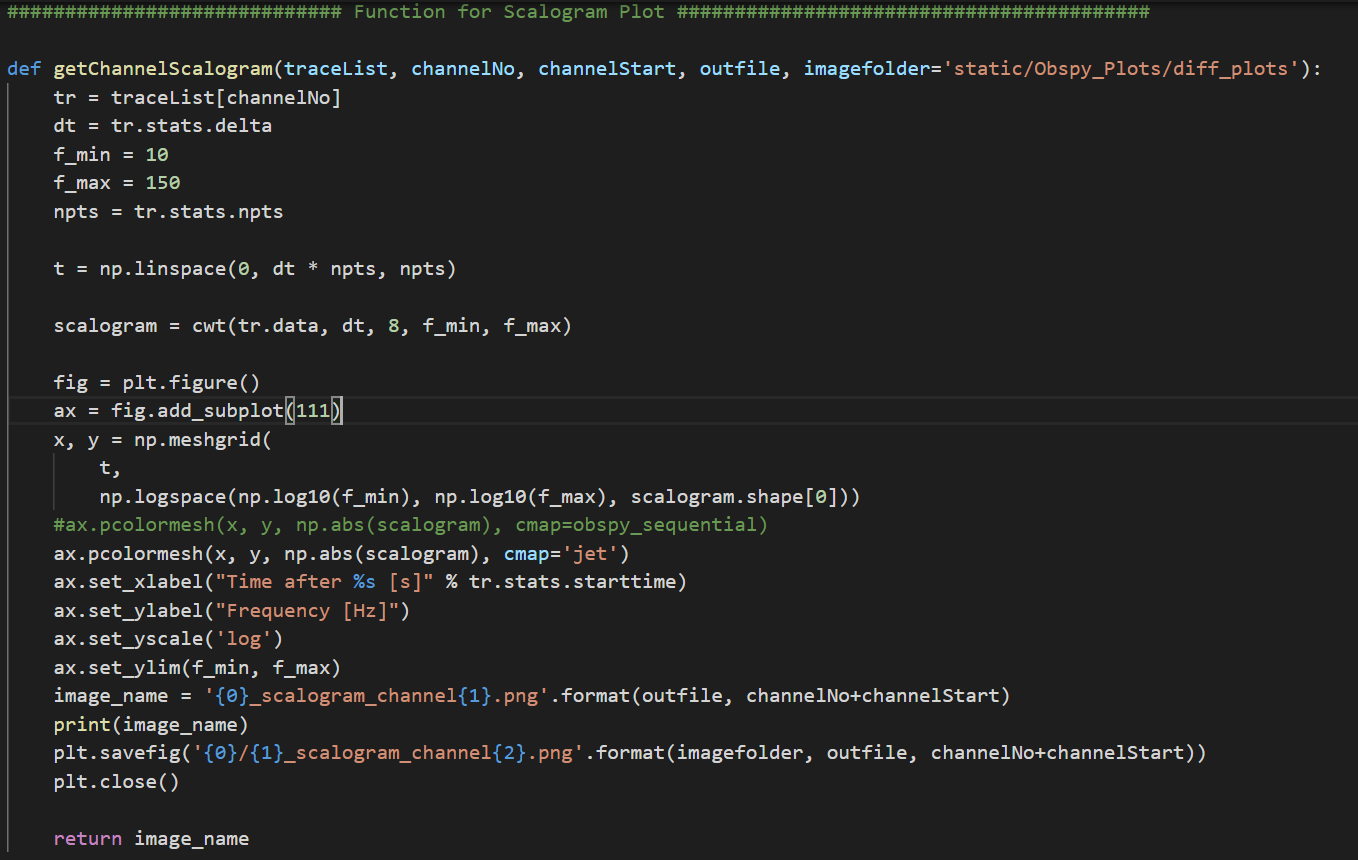


1. At the start of the app\_new.py file we include all the major python libraries that are used for DAS file processing and also those that are required for running the FLASK application. The above image shows some of the imports that are included in the file.
2. The image below shows the functions that are used for processing the DAS data and getting details for each trace List. We use the object of the class “Forge”, to generate different kinds of plots.





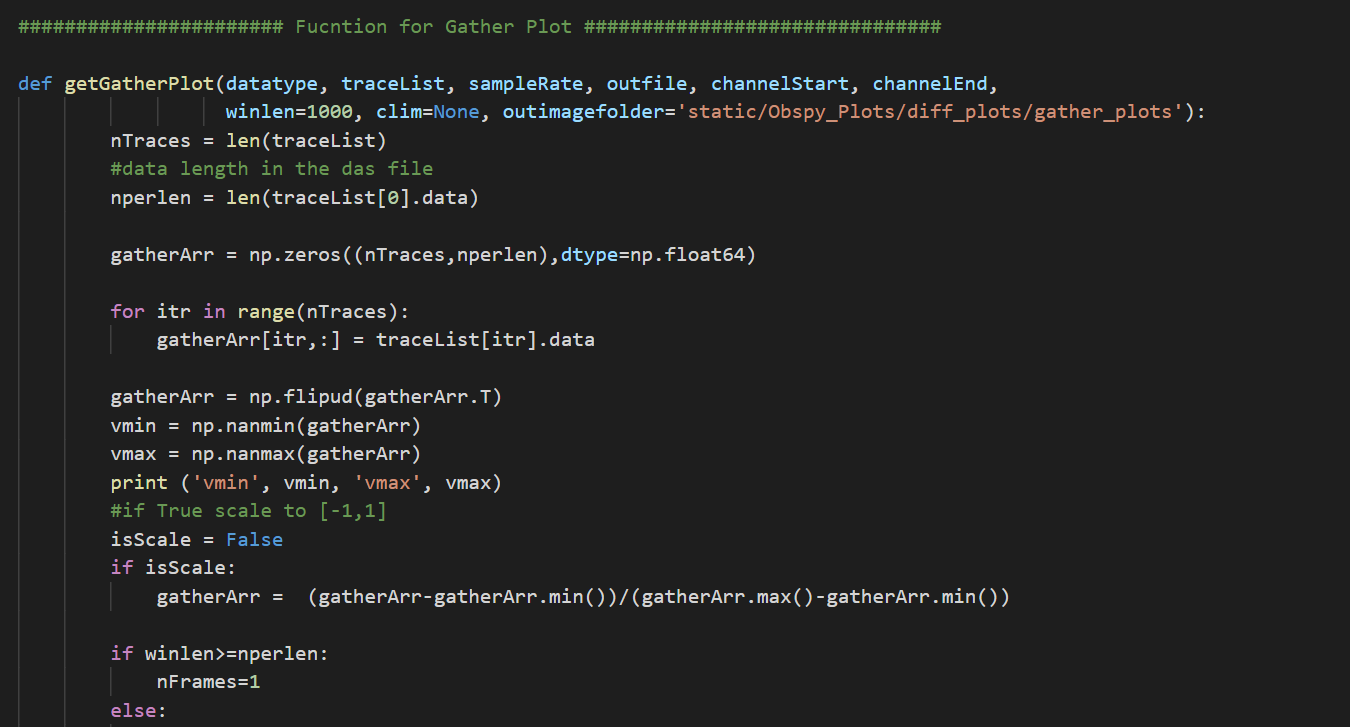
1. Below function is used to generate scalogram plots. We include the folder location where we need to save the image that is generated.



1. getChannelSpecgram() is used to generate spectrogram plots that we display on the UI. It takes the input from user such as channel start and outfile



1. Function to generate gather\_plots

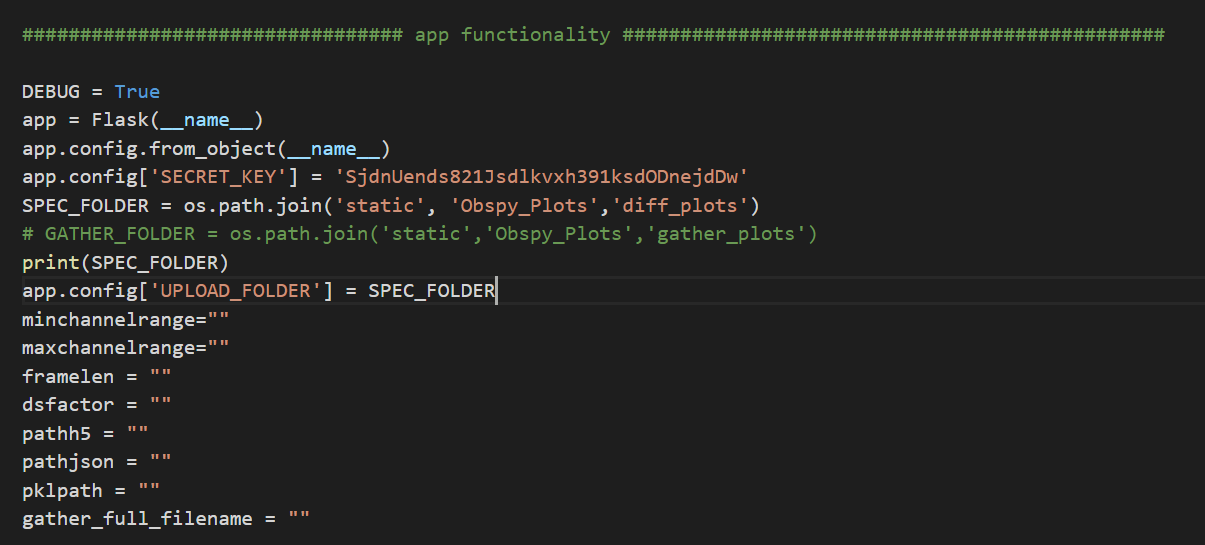




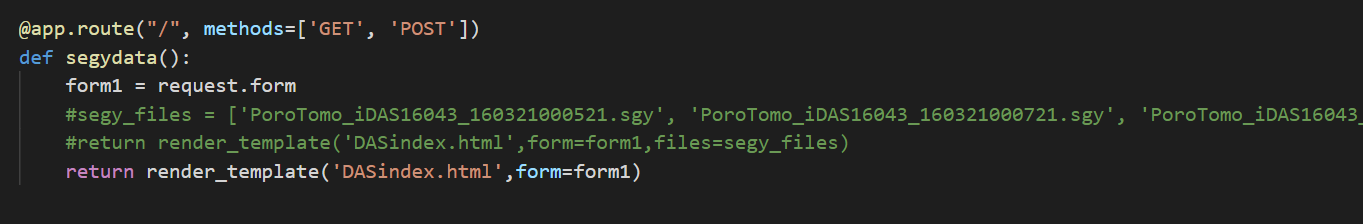


1. App configuration settings

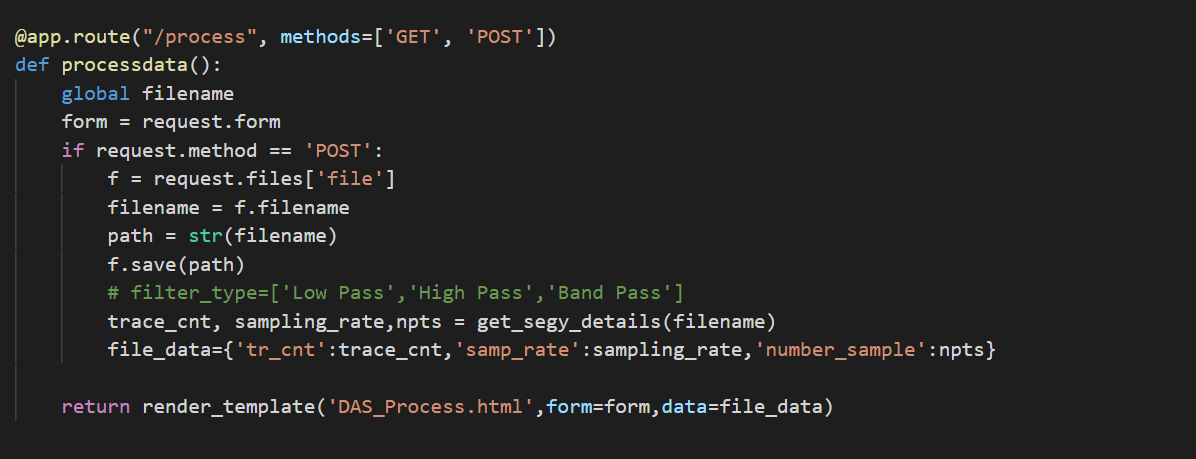
* Include “SECRET\_KEY”: it can be any random string input. It is mostly required for encryption.
* App.config[‘UPLOAD\_FOLDER’] has path to the folder where you need to store the images that are generated. It may also contain any files that are uploaded via the web app.
* The variables like framelen, dsfactor, minchannelrange etc are the global variables that are used later on for different purposes.



1. App route for index page, the segy\_files variable has the list of segy files that are shown in the dropdown in UI. You can change these file names to include other segy files for processing. Currently we just have the upload functionality.



1. Once we click the submit button for uploading the seg-y file, this function is called. This retrieves the filename and saves it on the server. It processes the files and gets the trace count, sampling rate and the number of samples per trace. These details are shown on the next HTML page “DAS\_Process.html”

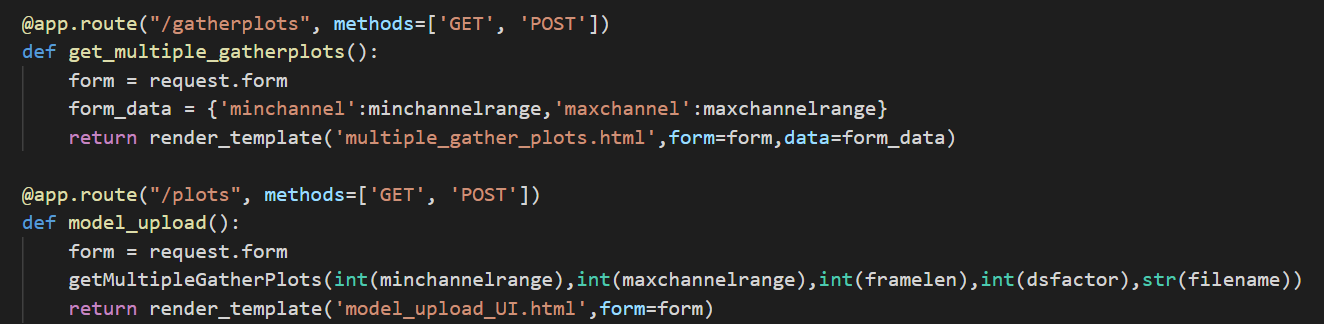


1. After entering the input details for processing the DAS data like minchannelrange, maxchannelrange, framelen and downsampling factor we read it on the following function. This then calls the function

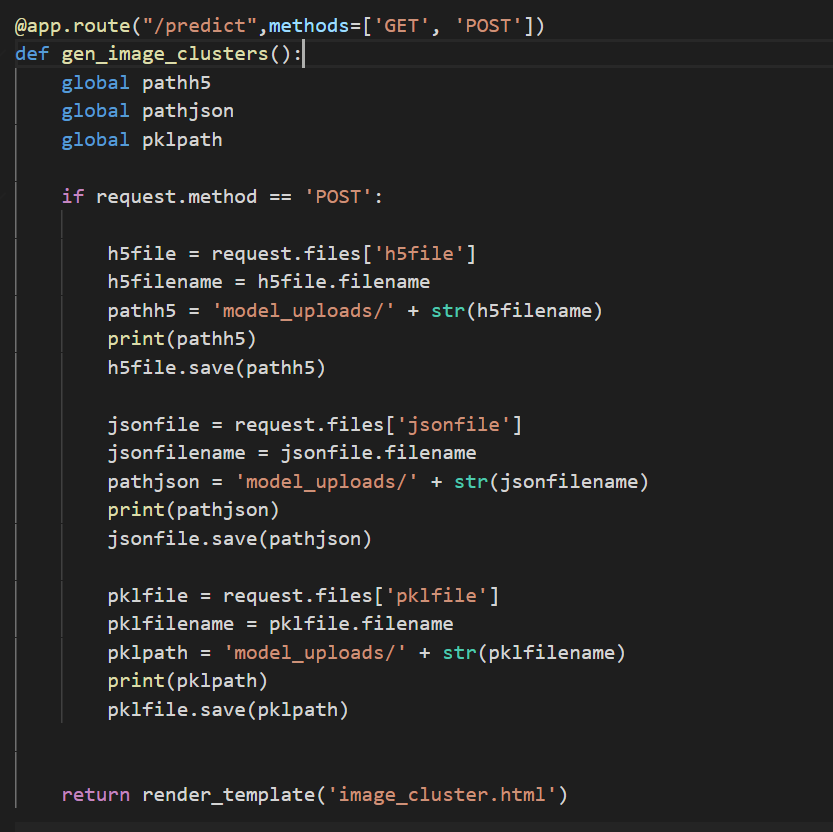
“get\_obspy\_plots” , which returns the image name for all the different kinds of plots. We then get the file location for these plots and send them to the UI via dictionary (test\_data).



1. The following app route are for generating multiple gather plots for different channel ranges within the same uploaded file. These plots may be used for deep clustering. Once the plots are generated, we render the HTML that allows user to upload different models for deep learning.



1. In this step we read the files that were uploaded during the model upload step and save it in the model\_uploads folder on server. The variables pathh5, pathjson, pklpath are global variables which can be used later on while using the uploaded files for prediction.



1. In the following code we read the uploaded models and pass it to a function used for predicting the cluster assignment. Currently, I have commented off this code and hardcoded the cluster assignment as I am getting unpickling error.



**HTML Code Changes**

In the form tag, we enter the action that needs to be performed when we click the submit button for that particular form. In this case, we enter all the input details like min and max channelrange etc. Once we click the submit button “display\_plots()” function is called from the app.py file.

