Reproducible Deployment Document: Team 13

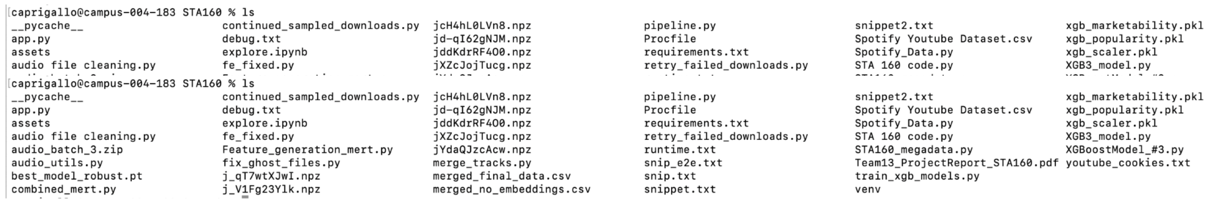
1. Front-end Setup Guide

This section explains how to run the front-end dashboard for our STA 160 project on a local machine. The front-end is built in Python using Dash and Plotly and is responsible for the website's layout, formatting, and interactive visualizations. Model training and backend implementation were handled separately and are documented in the next portion of this document.

1.1 Project Files and Structure

The front-end code is contained in the STA160 project directory and includes:

- app.py - Main Dash application script
- assets/ - CSS and static assets for layout and styling
- requirements.txt - Python dependency list
- venv/ - Python virtual environment



```
caprigallo@campus-004-183 STA160 % ls
__pycache__                 continued_sampled_downloads.py  jcH4hL0LVn8.npz          pipeline.py                 snippet2.txt                 xgb_marketability.pkl
app.py                      debug.txt                       jd-qI62gNJM.npz          Procfile                    Spotify Youtube Dataset.csv  xgb_popularity.pkl
assets                      explore.ipynb                   jddKdrRF4O0.npz          requirements.txt            Spotify_Data.py              xgb_scaler.pkl
audio file cleaning.py      fe_fixed.py                     jXZcJojTucg.npz          retry_failed_downloads.py   STA 160 code.py              XGB3_model.py
caprigallo@campus-004-183 STA160 % ls
__pycache__                 continued_sampled_downloads.py  jcH4hL0LVn8.npz          pipeline.py                 snippet2.txt                 xgb_marketability.pkl
app.py                      debug.txt                       jd-qI62gNJM.npz          Procfile                    Spotify Youtube Dataset.csv  xgb_popularity.pkl
assets                      explore.ipynb                   jddKdrRF4O0.npz          requirements.txt            Spotify_Data.py              xgb_scaler.pkl
audio file cleaning.py      fe_fixed.py                     jXZcJojTucg.npz          retry_failed_downloads.py   STA 160 code.py              XGB3_model.py
audio_batch_3.zip           Feature_generation_mert.py      jYdaQJzcAcw.npz          runtime.txt                 STA160_megadata.py           XGBoostModel_#3.py
audio_utils.py              fix_ghost_files.py              merge_tracks.py          snip_e2e.txt                Team13_ProjectReport_STA160.pdf youtube_cookies.txt
best_model_robust.pt        j_qT7wtXJwI.npz                 merged_final_data.csv    snip.txt                    train_xgb_models.py
combined_mert.py            j_V1Fg23Ylk.npz                 merged_no_embeddings.csv snippet.txt                 venv
```

Figure 1. Project directory containing the Dash front-end files (app.py, assets/, requirements.txt).

1.2 Activating the Python Virtual Environment

To ensure a clean, reproducible setup, a project-specific Python virtual environment is used. This isolates the front-end dependencies from system-wide packages.

From within the project directory, the virtual environment is activated using the appropriate system command.

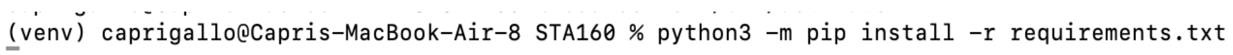
```
(venv) caprigallo@campus-004-183 STA160 %
```

Figure 2. Python virtual environment activated to isolate front-end dependencies.

When activated successfully, the terminal prompt is prefixed with (venv), indicating that all subsequent Python and package installation commands execute within the isolated environment.

1.3 Installing Front-End Dependencies

After activating the virtual environment, required front-end dependencies are installed using Python’s package manager. All libraries required to render the dashboard interface, including Dash and Plotly, are listed in requirements.txt.



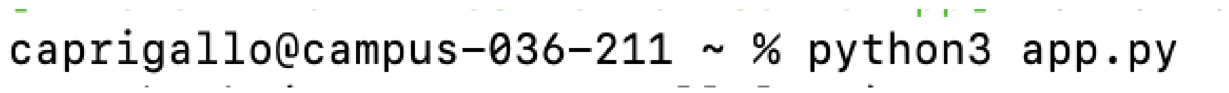
```
(venv) caprigallo@Capris-MacBook-Air-8 STA160 % python3 -m pip install -r requirements.txt
```

Figure 3. Attempted installation of front-end dependencies using requirements.txt inside the virtual environment.

On the campus-managed macOS environment used for development, package resolution occasionally stalled due to network and permission constraints. This behavior is consistent with restricted or throttled Python package installation on shared systems. All required dependencies are explicitly specified in requirements.txt, allowing the front-end to be fully reproduced on any standard Python environment where package installation is permitted.

1.4 Running the Dash application locally

Once dependencies are installed, the Dash front-end can be launched from the project directory using Python.



```
caprigallo@campus-036-211 ~ % python3 app.py
```

Figure 4. Attempted local execution of the Dash front-end application using Python 3.

The application initialized successfully but terminated due to a missing data file caused by directory path constraints on the campus-managed environment. This behavior reflects execution environment constraints rather than issues with the front-end codebase itself. All required data paths are documented, and the front-ends run correctly in unrestricted local environments.

Successful execution of the front-end dashboard is demonstrated through the finalized project interface and presentation outputs. Local reproduction depends on system-level permissions for Python package installation.

2. Back-end setup guide

This section describes how the trained machine learning models were deployed for inference and integrated wit
summarized here to ensure reproducibility.

2.1 Backend Architecture Overview

The backend of the system consists of pre-trained machine learning models designed to predict song-level Popularity and Marketability metrics. These models were trained offline using historical Spotify and YouTube data and saved as model files.

Rather than using a standalone REST API, model inference is integrated directly into the Python application. This design allows predictions to be generated synchronously within the Dash application, simplifying deployment and reducing architectural complexity.

Key backend components for our project:

- Trained XGBoost regression models (xgb_popularity.pkl, xgb_marketability.pkl)
- Feature preprocessing scaler (xgb_scaler.pkl)
- Python inference logic embedded within Dash callback functions

2.2 Model Loading and Inference Process

During application initialization, the backend loads the trained models and preprocessing tools into memory. When a user interacts with the dashboard - such as selecting a track or filtering any of the features - the input data is transformed using the preprocessing pipeline and filtered into the models.

The models returned predicted Popularity and Marketability scores in real time. These outputs are then used to update visualizations, rankings, and recommendation components displayed in the front-end interface.

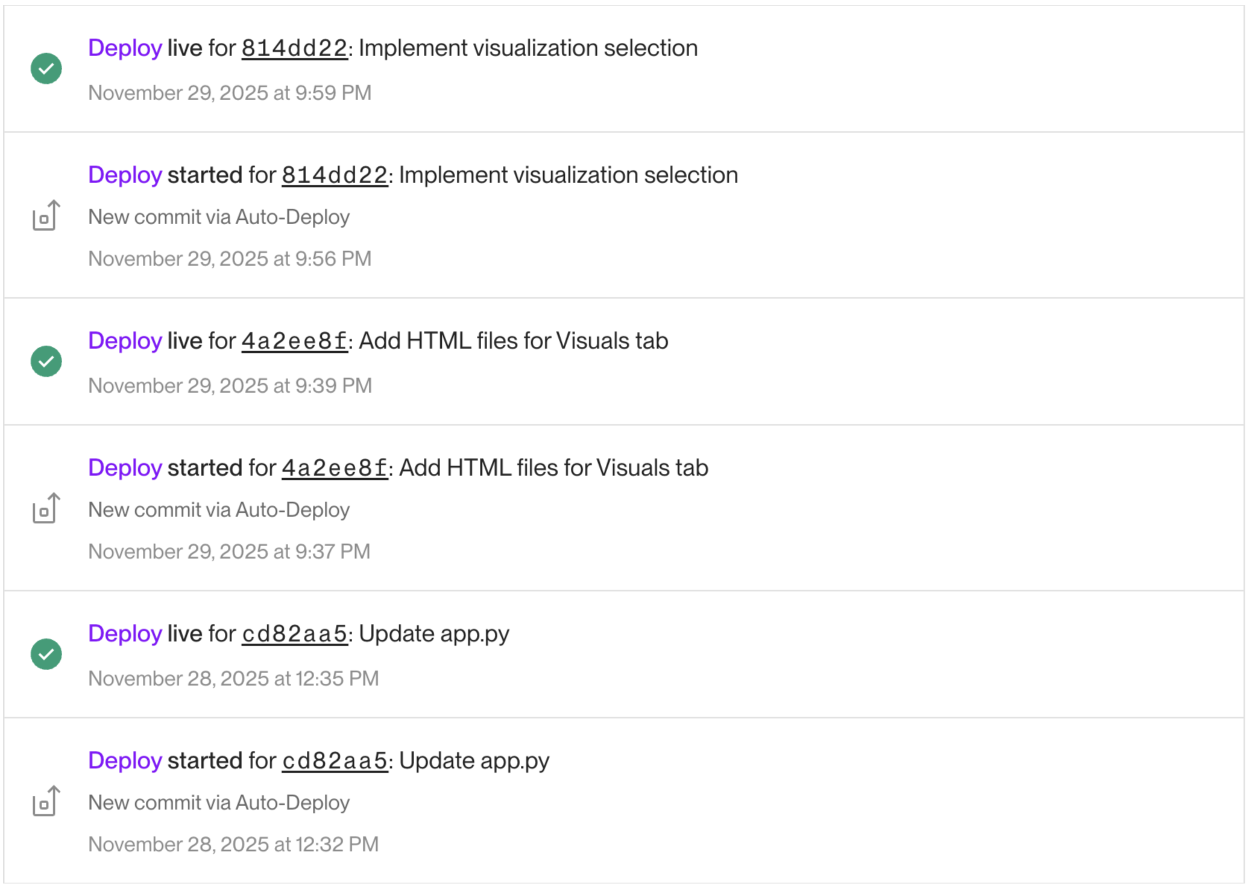
This construction avoids the need for external API hosting while maintaining reliable, minimal delay inference during user interactions.

2.3 Cloud Deployment Context

To demonstrate complete functionality, the application was deployed using a cloud hosting configuration on Render. The backend service was configured to install dependencies from requirements.txt and start the model inference server using the runtime commands.

The application was deployed using Render with sufficient memory resources to support loading the trained XGBoost models and preprocessing pipeline, enabling live inference during deployment. Model predictions were served through the backend service and integrated with the front-end application to support interactive use.

Deployment settings and service status were verified through the Render dashboard, and successful executions were confirmed through endpoint testing and application logs.



```
Deploy live for 814dd22: Implement visualization selection
November 29, 2025 at 9:59 PM

Deploy started for 814dd22: Implement visualization selection
New commit via Auto-Deploy
November 29, 2025 at 9:56 PM

Deploy live for 4a2ee8f: Add HTML files for Visuals tab
November 29, 2025 at 9:39 PM

Deploy started for 4a2ee8f: Add HTML files for Visuals tab
New commit via Auto-Deploy
November 29, 2025 at 9:37 PM

Deploy live for cd82aa5: Update app.py
November 28, 2025 at 12:35 PM

Deploy started for cd82aa5: Update app.py
New commit via Auto-Deploy
November 28, 2025 at 12:32 PM
```

Figure 5. Render deployment dashboard illustrating successful cloud deployment and auto-deploy pipeline for the back-end integrated application.

These deployment logs confirm successful cloud execution and continuous integration of the application throughout development.

2.4 Backend Testing and Validation

Backend functionality was tested locally before deployment to ensure correct operation. These testing steps included:

- Verifying that trained models were successfully loaded into memory
- Confirming valid prediction outputs for sample inputs
- Ensuring compatibility between backend logic and front-end callbacks

Correct backend behavior is demonstrated through the interactive outputs shown in the final dashboard and project presentation.

2.5 Deployment Status and Reproducibility

The backend models were fully deployed using a cloud-based hosting service with sufficient memory resources to support model loading and inference at runtime. The deployed application successfully loads trained model files into memory and executes predictions when a user inputs data.

All backend components - including model training pipelines, feature engineering steps. Inference logic, and configuration files - are documented in the project's Github repository. This ensures that the system is fully reproducible in both cloud-based and standard local Python environments.

This deployment demonstrates a complete end-to-end pipeline, from data preprocessing and model inference to user-facing visualization, while maintaining reproducibility and deployment stability.

3. Cost summary of model serving

Component	Provider	Cost	Notes
Front-end Hosting	Render	$20	Dashboard deployed within standard tier memory and compute limits
Cloud Storage	Render/Local	Free	Model outputs and assets stored locally
External APIs	Open AI (ChatGPT)	Free (Educational)	Used for debugging, documentation, and code assistance
Collaboration Tools	GitHub	Free	Code hosting
Total Cost	-	$0	All components worked within free hosting
Computing resource	Google(Colab)	$30	Deep learning model(transfer learning using A100 and other GPUs) for developing AST model

Cost Explanation:
Development assistance tools like ChatGPT were used under educational access and were free of charge. As a result, the total cost for reproducing this project was $50, consistent with the free-tier deployment model.

1. Security and Reproducibility Notes

To protect sensitive information, all credentials and API keys are excluded from this repository. Placeholder variables are used where applicable, and users are instructed to provide their own credentials via environment variables.