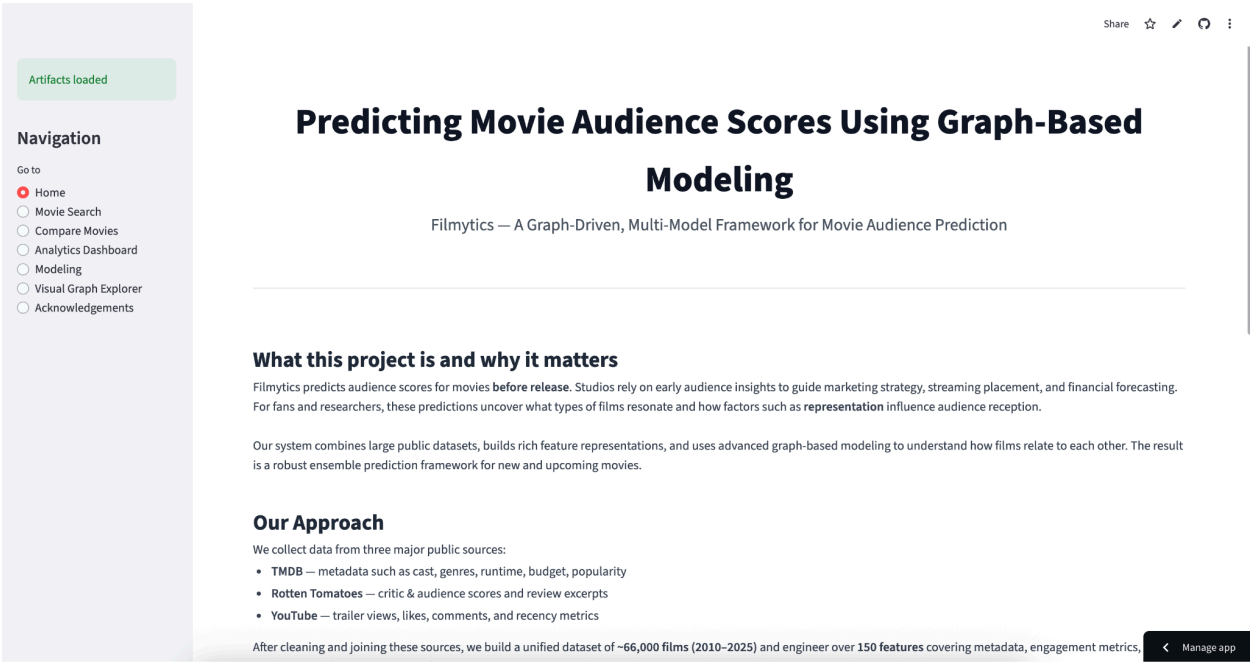
Step by Step Reproducible Deployment Document

STA 160 Team 15: Filmlytics

1. Front-end Setup Guide (with screenshots)

Step-by-step instructions on installing dependencies and running the website locally.

1. Set up local server on Python 3
2. Clone github repository
    - git clone https://github.com/claragwei/filmlytics.git
3. Start local web server
    - Navigate to repository in local machine
    - In terminal run: python3 streamlit run “streamlit_app_clara.py”
    - The home page will open locally as shown below:



2. Backend Model Deployment Instructions

2.1 Overview of Backend Architecture

The FilmLytics backend is a lightweight, integrated system built directly into the Streamlit application. There is no external server or API—every backend operation runs inside Streamlit at runtime.

The backend consists of the following major components:

1. Model Inference Layer

- Pre-trained machine learning models stored as `.joblib` files in the GitHub repository are loaded when the app starts.
- Streamlit's caching mechanisms are used to keep inference fast and avoid reloading models on every interaction.
- If a model ever exceeds GitHub's file-size limit, the app supports downloading it dynamically from external storage (e.g., MongoDB) at runtime.

Purpose:
Provides fast, cached predictions for user inputs such as movie features or metadata.

2. Data Retrieval Layer

FilmLytics retrieves data from two possible sources:

Local Repository Files

- CSV and JSON datasets stored inside the GitHub repo
- Loaded using pandas when users navigate to analytics or exploration pages

MongoDB Atlas (when enabled)

- Connected using a secure connection string stored in Streamlit Secrets
- Used for retrieving additional movie metadata or collections

Streamlit Secrets hides all sensitive credentials so they never appear in the public repository.

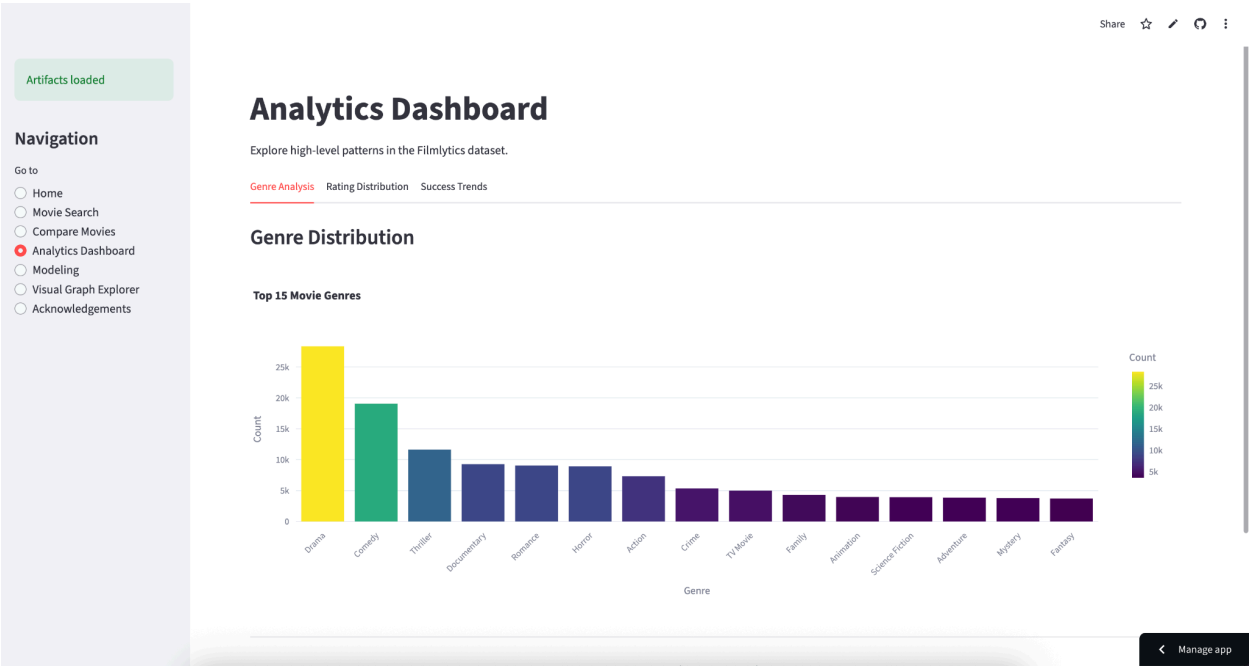
Purpose:
Provides structured data inputs for analysis, model inference, and visualizations.

3. Visualization & Analytics Engine

The backend generates dynamic visuals and analytical outputs using:

- **Plotly** for interactive charts

- **PyVis** for network graphs
- **Pandas / NumPy** for data wrangling
- **Scikit-learn preprocessing** for model-ready feature engineering



These visualizations are created on demand and rendered directly in the Streamlit frontend.

Purpose:
Transforms model results + raw data into interpretable visual insights.

4. Session State & Navigation Logic

Although the app appears to be "just a frontend," Streamlit session state acts as part of the backend:

- Tracks user interactions
- Stores outputs from models
- Maintains page routing across the multi-page layout

- Prevents reloading data or models unnecessarily

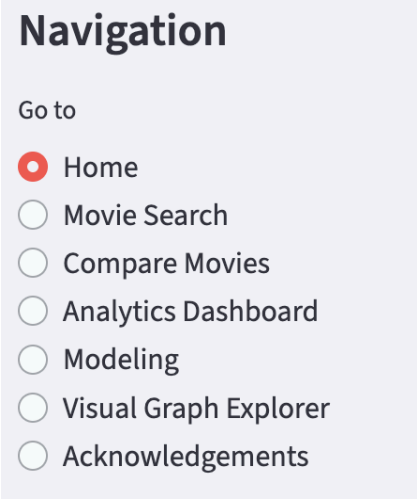
This creates a persistent, stateful user experience while running entirely in the browser.

Purpose:
Ensures smooth navigation, stable performance, and consistent data across all pages.

5. Multipage Application Structure

FilmLytics uses Streamlit's native multi-page framework:

- Each page (Intro, Visualization Tools, Predictive Models, etc.) has its own script
- All pages share the same cached models, data, and session state
- Backend logic runs on whichever page the user opens



Purpose:
Organizes the backend logic into clean modules without requiring a separate backend server.

2.2 Preparing the Backend Environment

Dependencies

Our backend requires an environment with the following libraries:

- streamlit
- pandas

- numpy
- plotly
- pymongo
- certifi
- joblib
- pyvis
- xgboost
- Scikit-learn

These are listed in a requirements.txt file that can be installed using the following command:

- pip install -r requirements.txt

Model Storage Strategy (Large File Constraint)

All trained model artifacts are stored locally in the model artifacts directory within the GitHub repository. All model files are under GitHub's 100MB file size limit, allowing them to be pushed directly into the repository without requiring Git LFS.

2.3 Front-End Integration (Website)

The FilmLytics front-end is built entirely using **Streamlit**, which functions as both the UI framework and the presentation layer for all backend outputs. Because Streamlit automatically links UI components to backend logic, no separate front-end framework or API integration is required.

How the Front-End Connects to the Backend

- User inputs (dropdowns, sliders, text boxes, buttons) are created in `streamlit_app_clara.py`.
- When users interact with the UI, Streamlit triggers backend functions that:
    - load model files
    - process data
    - run predictions
    - generate Plotly charts and PyVis network graphs

- The results are rendered instantly in the same interface.

Local File + Model Integration

- Visualizations and predictions update live based on:
    - the local datasets stored in the GitHub repository
    - the `.joblib` models loaded at app startup
- No separate API calls or AJAX requests are needed—the Streamlit script handles both UI and backend computation.

Multi-Page Front-End Structure

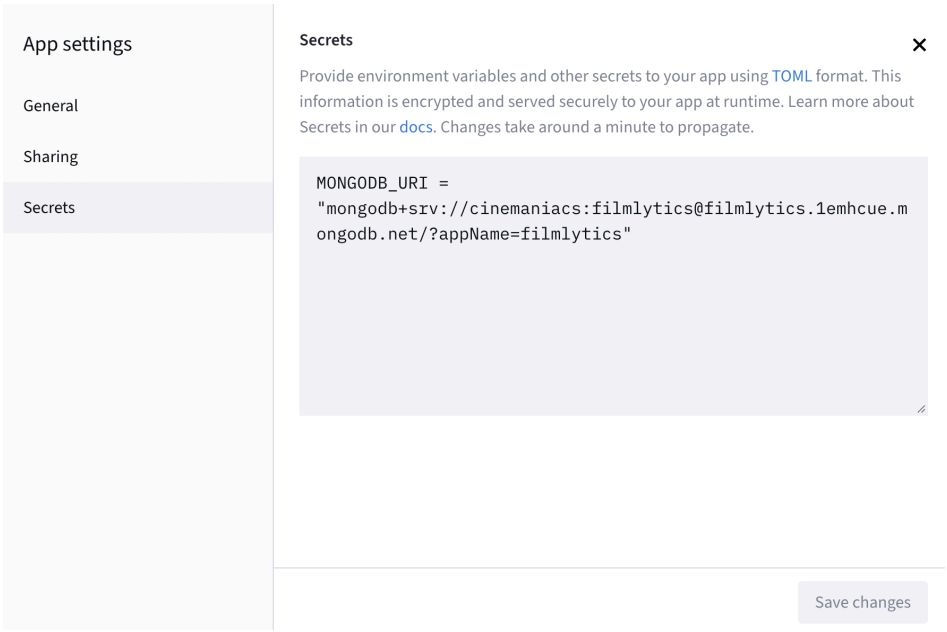
FilmLytics uses Streamlit's native multi-page system:

- Each page (e.g., Home, Movie Explorer, Prediction Tools, etc.) is a separate `.py` file.
- All pages share the same backend data, models, and cached resources.
- Navigation links between pages act like a front-end "router".

Deployment Integration

Once the front-end code and visual elements are pushed to GitHub on the branch clara-streamlit:

- Streamlit Cloud automatically rebuilds the site
    - https://share.streamlit.io/
    - Updated secrets section under settings to have mongodb connection url



- The front-end connects to the backend seamlessly without additional configuration.

This creates a simple, user-friendly interface where analytics, predictions, and visual tools all run inside one unified application.

3. Cost summary of model serving

A short summary of any costs associated with your model serving or hosting.

Cost Summary

Component	Provider	Cost	Notes
Static Website Hosting	Streamlit	$0	All of our front-end files are hosted as static pages, with no additional cost. All components are within the free usage limits.
Total cost		$0	