Build a Movie Recommendation System using Python and Azure

A Step-by-Step Guide to building a Movie Recommendation System

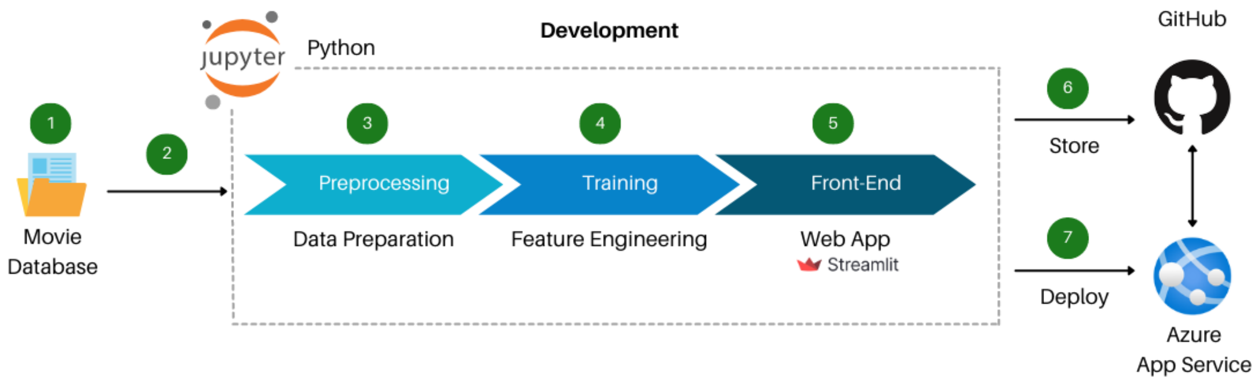
If you're a movie lover, you know how difficult it can be to find a new film to watch. With so many options across multiple platforms, it can be overwhelming to find what to watch. That's where a movie recommendation system comes in handy. By analyzing your past movie ratings and preferences, a recommendation system can suggest new films and shows, you're likely to enjoy.

Recommendation systems are essential AI-based tools that help predict the rating or preference a user would give to an item. These systems have become ubiquitous and can be commonly seen in online stores, and streaming services. There are several types of recommendation systems and the one being discussed in this article is a content-based recommendation system.

A content-based recommendation system recommends items based on the characteristics of the item and the user's past preferences. These types of systems, represent each item (e.g., a movie) as a vector of its characteristics (e.g., genre, cast, director, plot overview, keywords, etc.)

Solution Architecture

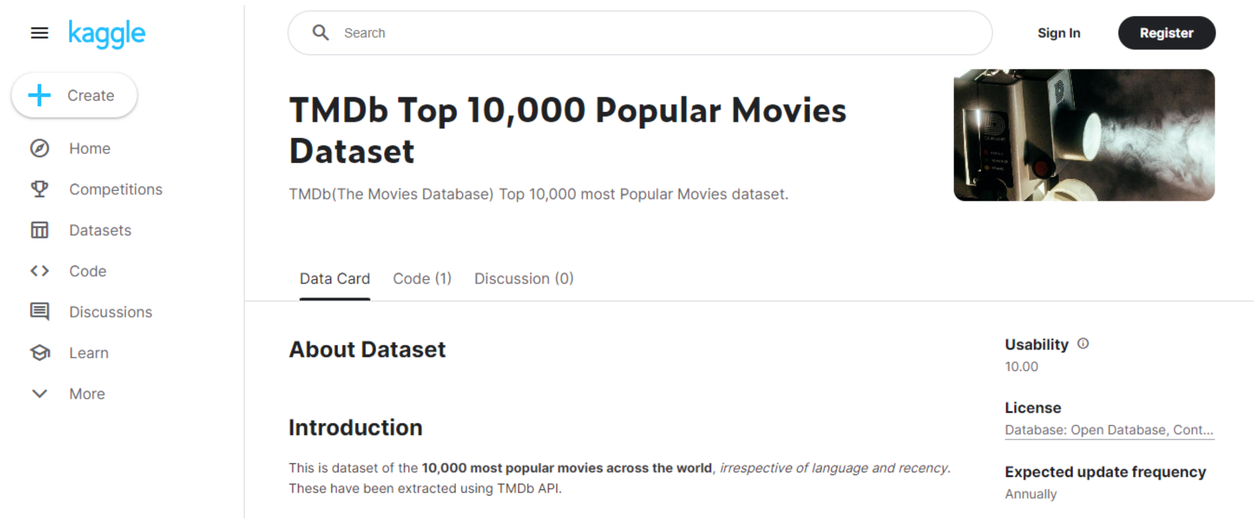
This article describes the process of building a movie recommendation system using python and the open-sourced libraries Scikit-Learn and NLTK. Streamlit is used to create a simple web app for interacting with the recommendation system and azure app service is used to deploy the app



Technical Implementation of the Solution

Step 1—Gather the Data

For building the recommendation system, this tutorial will be using the TMDb dataset, which contains data from over 10,000+ popular movies around the world. To download the dataset, you will need to sign up for an account on the Kaggle website and agree to their terms of use.



TMDb Top 10,000 Popular Movies Dataset

TMDb(The Movies Database) Top 10,000 most Popular Movies dataset.

About Dataset

Introduction

This is dataset of the **10,000 most popular movies across the world**, *irrespective of language and recency*. These have been extracted using TMDb API.

Usability
10.00

License
Database: Open Database, Cont...

Expected update frequency
Annually

Image from Author— Download the TMDb Movies Dataset

```
#import python packages
import numpy  as np
import pandas as pd

#load dataset to pandas dataframe
df = pd.read_csv("tmdb_movies_data.csv")
```

Next, create a new Azure notebook and import all the necessary python packages. Extract the downloaded zip file and read the movie dataset into a python data frame using pandas package.

Step 2—Pre-Process the Data

Filter the data to remove any missing values. This is important because missing values can induce unwanted bias in the data. You can alternatively try using one of the imputers given here.

```
#select the key columns that'll be used while building the model
movies =
df[['id','cast','director','genres','overview','original_title','keywords']]
```

```
]

#drop any null values from the database
movies.isnull().sum()
movies.dropna(inplace = True)
```

Remove any unwanted characters, duplicate data, or sparse values and transform the dataset into a standard format. Next, stem the data to reduce the words in the dataset to their base form

```
#function to transform the data to a standard format
def convert(obj):
    s = list(obj)
    for i in range(len(s)):
        if s[i] == '|':
            s[i] = " "

    temp_str = "".join(s)
    temp_list = temp_str.split()

    return temp_list

#apply the convert function to all the columns
movies['genres'] = movies['genres'].apply(convert)
movies['keywords'] = movies['keywords'].apply(convert)
movies['overview'] = movies['overview'].apply(lambda x:x.split())
movies['cast'] = movies['cast'].apply(convert)
movies['director'] = movies['director'].apply(convert)

#stem the data to reduce words to their base form
import nltk
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()

def stem(text):
    y = []
    for i in text.split():
        y.append(ps.stem(i))

    return " ".join(y)
```

Step 3—Build the Model

Now, perform text vectorization to transform each movie into a vector. The approach described in this article is called the Bag of Words. There also exist several other techniques for performing text vectorization such as Binary Term Frequency, normalized TF/IDF and Word2Vec

```
#create a new column that combines all characteristics of the movie
movies['tags'] = movies['cast'] + movies['genres'] + movies['director'] +
movies['overview'] + movies['keywords']

#create a new dataframe consisting of id, original title, and tags
new_df = movies[['id','original_title','tags']]
new_df['tags'] = new_df['tags'].apply(lambda x:" ".join(x))

#apply the stem function to the tags column of your dataframe
new_df['tags'] = new_df['tags'].apply(stem)
```

Next, fit the model to the dataset after removing stop words. Stop words are small, common words (such as “the”, “a”, and “and”) that are frequently used in the text but carry little meaning.

```
#perform text vectorization after disregarding stop words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=10000,stop_words='english')

#transform the SciPy sparse matrix to NumPy array form
vectors = cv.fit_transform(new_df['tags']).toarray()
```

Mathematically, for any given vector the recommendation model outputs the five closest vectors. Here, cosine distance is used in place of euclidean distance as in higher dimensionality, the former gives a better measure of distance. This is referred to as the “Curse of Dimensionality”.

```
#calculate the cosine similarity between the vectors
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vectors)
```

After processing the results, make a pickle dump to save the objects for later use in the program

```
#take the pickle dump of the results for later use
import pickle

pickle.dump(new_df,open('movies.pkl','wb'))
pickle.dump(new_df.to_dict(),open('movie_dict.pkl','wb'))
pickle.dump(similarity,open('similarity.pkl','wb'))
```

Step 4—Develop the Front-End

Install Streamlit in your virtual python environment, and create a new Python file called app.py. This file will contain the code for designing the user interface of the web application and for recommending movies to the user by calculating the cosine similarity between different movies.

To develop the front end for your web app, install Streamlit and create a new python file called app.py. This file would contain the program for designing the user interface of your Streamlit app

```
import streamlit as st
import pickle
import pandas as pd
import requests

#Recommend movies based on content
def recommend(movie):
    movie_index = movies[movies['original_title'] == movie].index[0]
    distances = similarity[movie_index]
    movies_list = sorted(list(enumerate(distances)), reverse=True,
key=lambda x: x[1])[1:6]

    recommended_movies = []
    recommended_movies_poster = []

    for  i in movies_list:
        movie_id = movies.iloc[i[0]].id
        recommended_movies.append(movies.iloc[i[0]].original_title)
        recommended_movies_poster.append(fetch_poster(movie_id))
       return recommended_movies,recommended_movies_poster

movies_dict = pickle.load(open('pickle/movie_dict.pkl','rb'))
movies = pd.DataFrame(movies_dict)

similarity = pickle.load(open('pickle/similarity.pkl','rb'))
```

Next, fetch the posters of the movies recommended by your recommendation system using the TMDb API. You can create your very own API by logging into the TMDb developers API v3 site.

```
#Fetch posters from TMDb Database
```

```python
def fetch_poster(movie_id):
    response =
requests.get('https://api.themoviedb.org/3/movie/{}?api_key=ENTER_API_KEY_H
ERE&language=en-US'.format(movie_id))
    data = response.json()
    return "https://image.tmdb.org/t/p/w500/" + data['poster_path']

#Frontend Hero Section
st.title("Movie Recommender System")

selected_movie_name = st.selectbox(
'Select a movie to recommend',
movies['original_title'].values)

#Output Recommendations with Posters
if st.button('Recommend'):
    name, posters = recommend(selected_movie_name)

    col1, col2, col3, col4,  col5 = st.columns(5)
    with col1:
        st.text(name[0])
        st.image(posters[0])
    with col2:
        st.text(name[1])
        st.image(posters[1])
    with col3:
        st.text(name[2])
        st.image(posters[2])
    with col4:
        st.text(name[3])
        st.image(posters[3])
    with col5:
        st.text(name[4])
        st.image(posters[4])
```

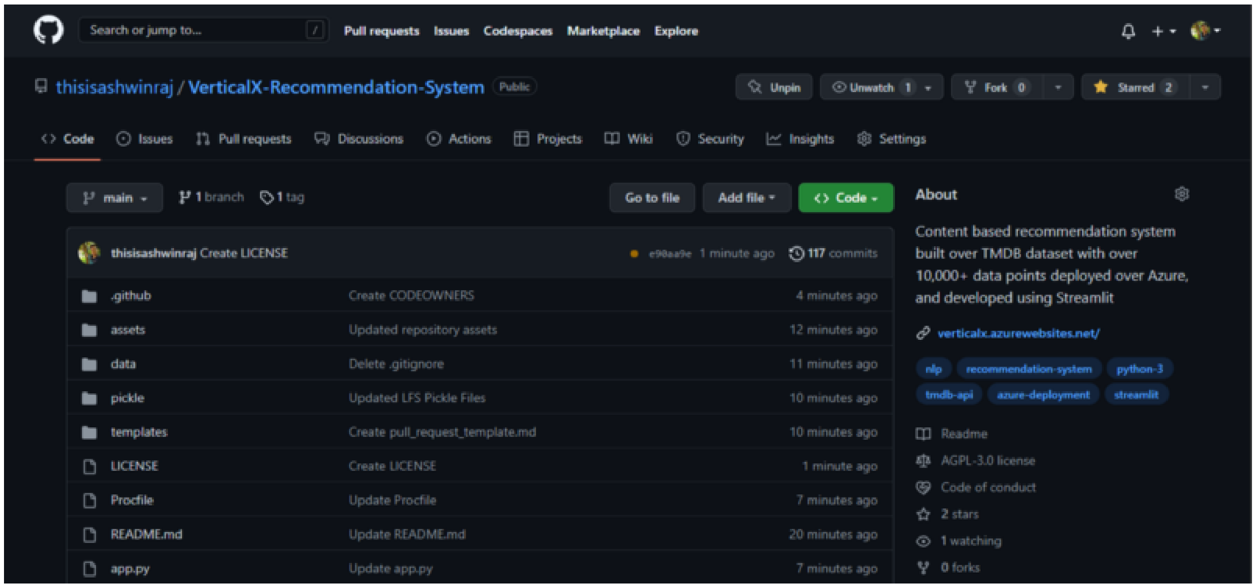
Remember to replace the API key in line 3 of this program with your own API key. Once your frontend is ready, run the app locally by opening the terminal and typing the following command:

```
streamlit run app.py
```

This command may take a few minutes before launching the web app on http://localhost:8501.

Step 5—Deploying the WebApp on Azure

Next, you need to deploy the web app to Azure. If you don't already have an Azure account, you can sign up for a free trial here. Now, upload your files to a GitHub repository. Once your repository is ready, go to the Azure dashboard, click on App Service and select the create option



thisisashwinraj / VerticalX-Recommendation-System Public

thisisashwinraj Create LICENSE · e90aa9e 1 minute ago · 117 commits

.github	Create CODEOWNERS	4 minutes ago
assets	Updated repository assets	12 minutes ago
data	Delete .gitignore	11 minutes ago
pickle	Updated LFS Pickle Files	10 minutes ago
templates	Create pull_request_template.md	10 minutes ago
LICENSE	Create LICENSE	1 minute ago
Procfile	Update Procfile	7 minutes ago
README.md	Update README.md	20 minutes ago
app.py	Update app.py	7 minutes ago

About

Content based recommendation system built over TMDB dataset with over 10,000+ data points deployed over Azure, and developed using Streamlit

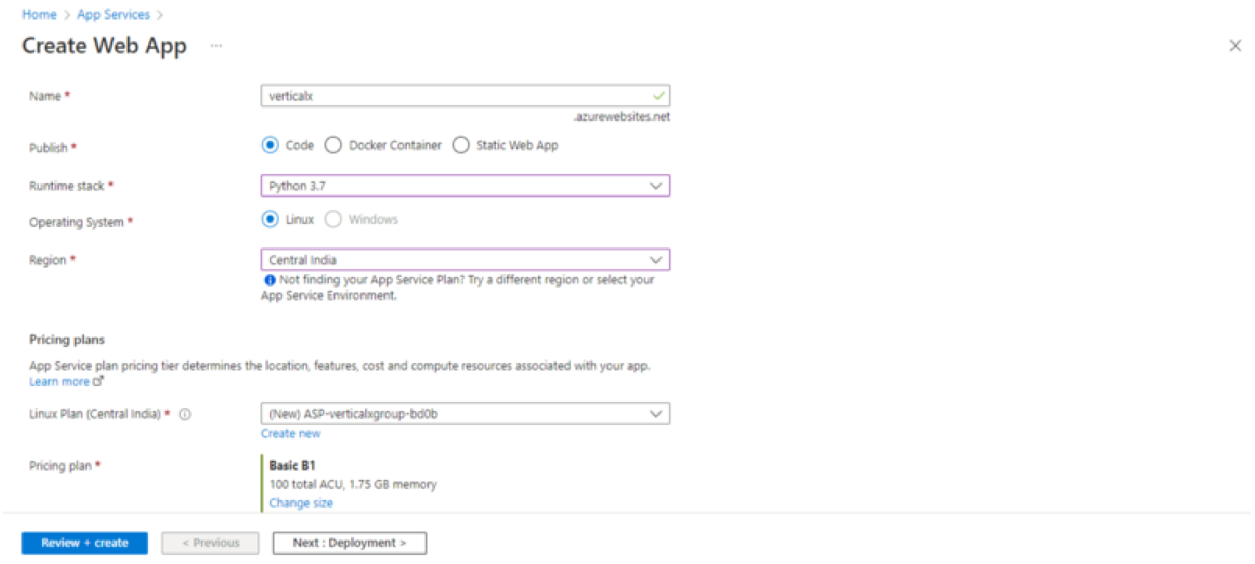
verticalx.azurewebsites.net/

nlp · recommendation-system · python-3 · tmdb-api · azure-deployment · streamlit

Readme
AGPL-3.0 license
Code of conduct
2 stars
1 watching
0 forks

Image from Author— Store your files in GitHub

In the resource creation window, enter any unique name for your app, select the runtime stack as 'Python 3.7', and set the region field to any location close to you. Now, change the pricing plan by clicking on change size and select the B1 tier. Keep all other fields to their default values



Create Web App

Name *	verticalx .azurewebsites.net
Publish *	[✓ Code] Docker Container Static Web App
Runtime stack *	Python 3.7
Operating System *	[✓ Linux] Windows
Region *	Central India

Not finding your App Service Plan? Try a different region or select your App Service Environment.

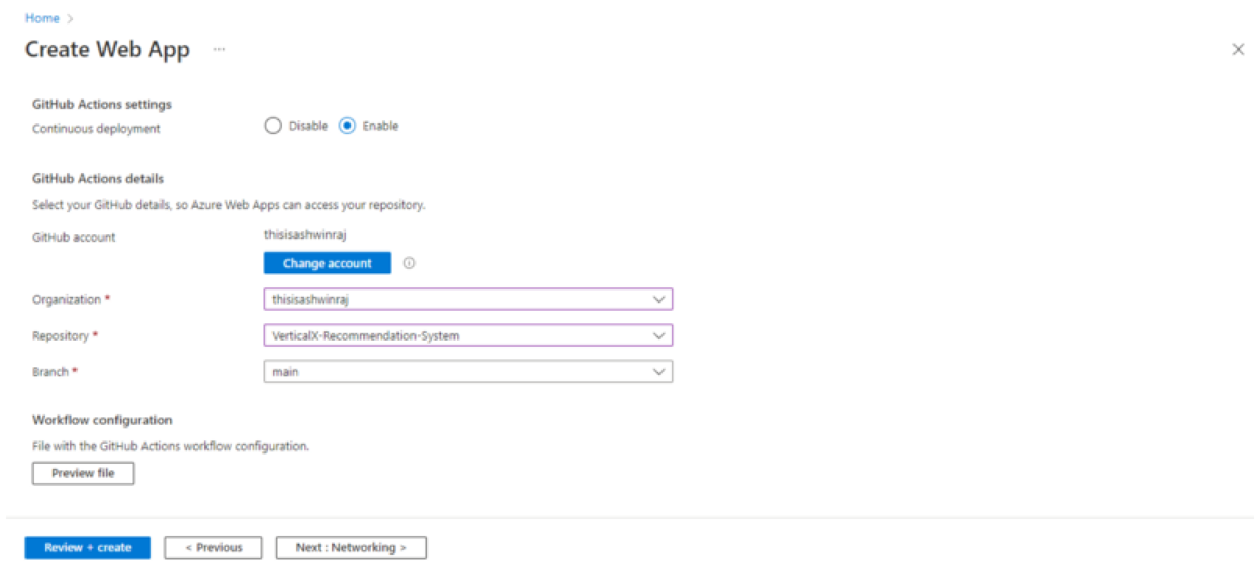
Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. Learn more

Linux Plan (Central India) *	(New) ASP-verticalxgroup-bd0b Create new
Pricing plan *	Basic B1 100 total ACU, 1.75 GB memory Change size

Image from Author— Create a new App Service

In the deployments section, enable the continuous deployment option under GitHub actions settings. Next, connect your GitHub account to Azure, select your username in the organization field and choose your repository. If your repository has multiple branches select the main branch



Home >

Create Web App

GitHub Actions settings

Continuous deployment: Disable [✓ Enable]

GitHub Actions details

Select your GitHub details, so Azure Web Apps can access your repository.

GitHub account: thisisashwinraj

Organization *: thisisashwinraj

Repository *: VerticalX-Recommendation-System

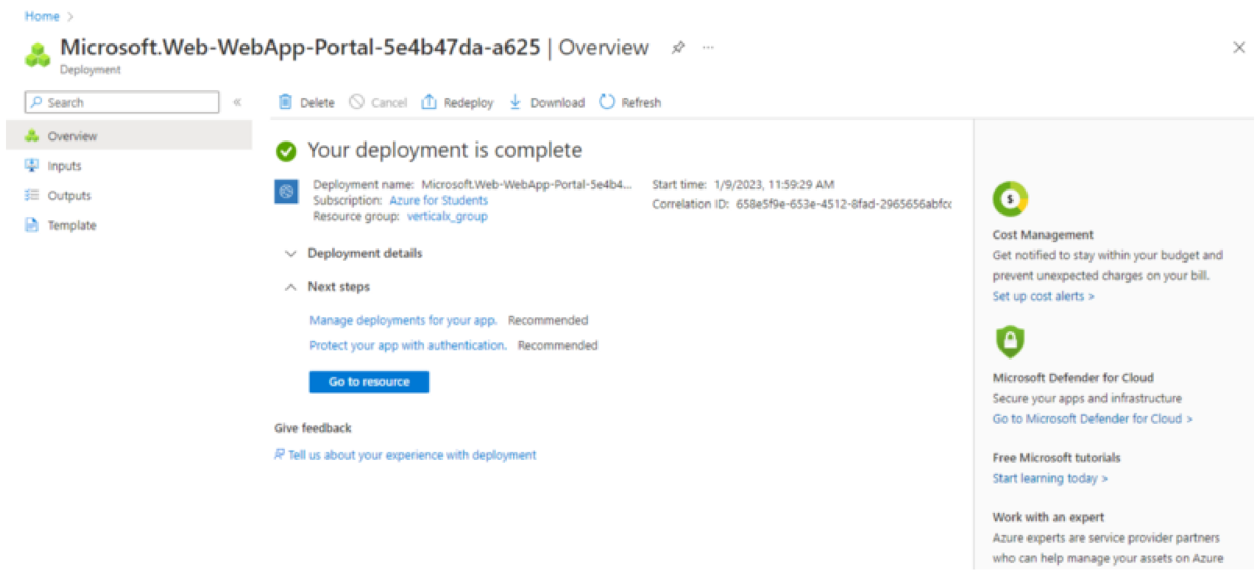
Branch *: main

Workflow configuration

File with the GitHub Actions workflow configuration.

Image from Author— Connect the App Service to your GitHub Repository

You don't have to make any changes in the Networking and Monitoring sections, so simply click on 'Review + Create'. After reviewing the changes you made, click on create and wait for your deployment to complete. Now in your repository, a new GitHub action workflow starts executing.



Home >

Microsoft.Web-WebApp-Portal-5e4b47da-a625 | Overview

Deployment

Your deployment is complete

Deployment name: Microsoft.Web-WebApp-Portal-5e4b4...
Subscription: Azure for Students
Resource group: verticalx_group

Start time: 1/9/2023, 11:59:29 AM
Correlation ID: 658e5f9e-653e-4512-8fad-2965656abfcc

Deployment details

Next steps

Manage deployments for your app. Recommended

Protect your app with authentication. Recommended

Give feedback

Tell us about your experience with deployment

Cost Management

Get notified to stay within your budget and prevent unexpected charges on your bill.

Set up cost alerts >

Microsoft Defender for Cloud

Secure your apps and infrastructure

Go to Microsoft Defender for Cloud >

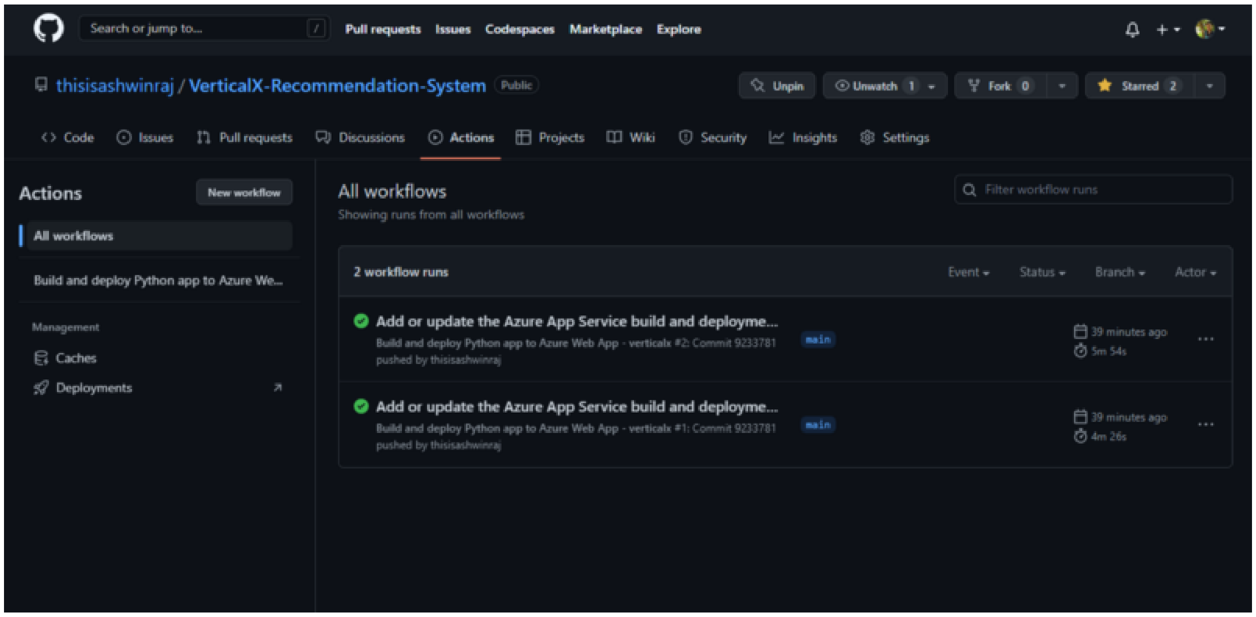
Free Microsoft tutorials

Start learning today >

Work with an expert

Azure experts are service provider partners who can help manage your assets on Azure

Image from Author— Wait for the Deployment to complete



thisisashwinraj / VerticalX-Recommendation-System Public

All workflows
Showing runs from all workflows

2 workflow runs

Add or update the Azure App Service build and deployme...
Build and deploy Python app to Azure Web App - verticalx #2: Commit 9233781 pushed by thisisashwinraj — main — 39 minutes ago — 5m 54s

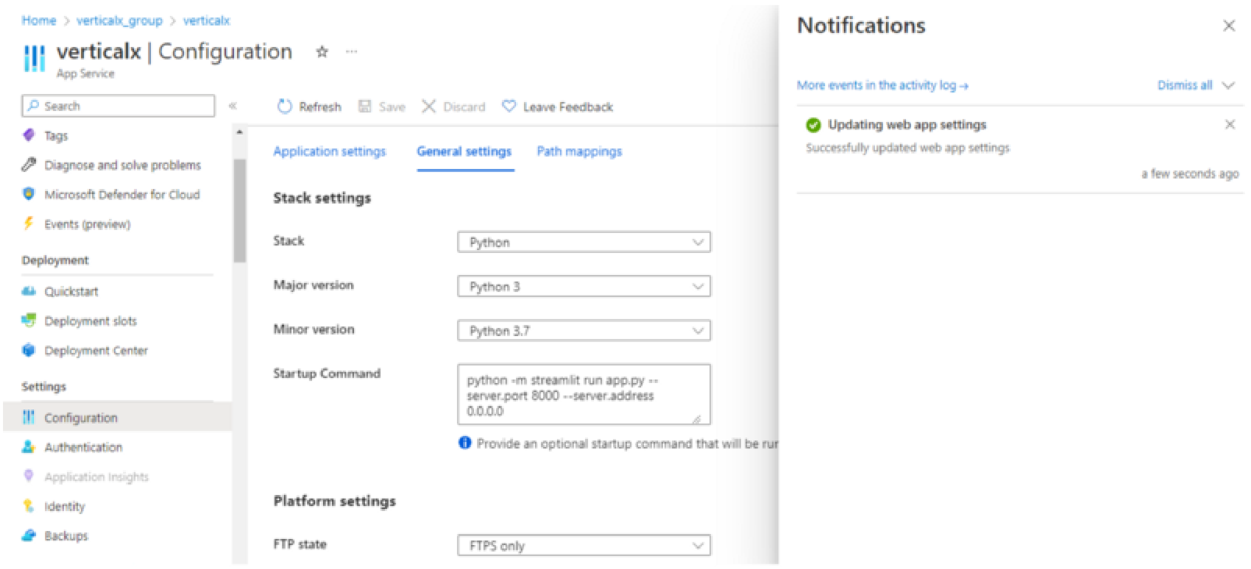
Add or update the Azure App Service build and deployme...
Build and deploy Python app to Azure Web App - verticalx #1: Commit 9233781 pushed by thisisashwinraj — main — 39 minutes ago — 4m 26s

Image from Author— Wait for the workflows to finish running

Wait for the workflow to finish executing. Now back in the Azure portal, go to the configurations option, and select general settings. In the startup command field, enter the following command:

```
python -m streamlit run app.py -- server.port 8000 -- server.address
0.0.0.0
```

Once done, click on the save button and wait for the web app to be updated. Navigate to the URL address of your web application and voila, your recommendation system is up and running.



Home > verticalx_group > verticalx
verticalx | Configuration
App Service

Stack settings

Stack: Python
Major version: Python 3
Minor version: Python 3.7
Startup Command: python -m streamlit run app.py --server.port 8000 --server.address 0.0.0.0

Provide an optional startup command that will be ru

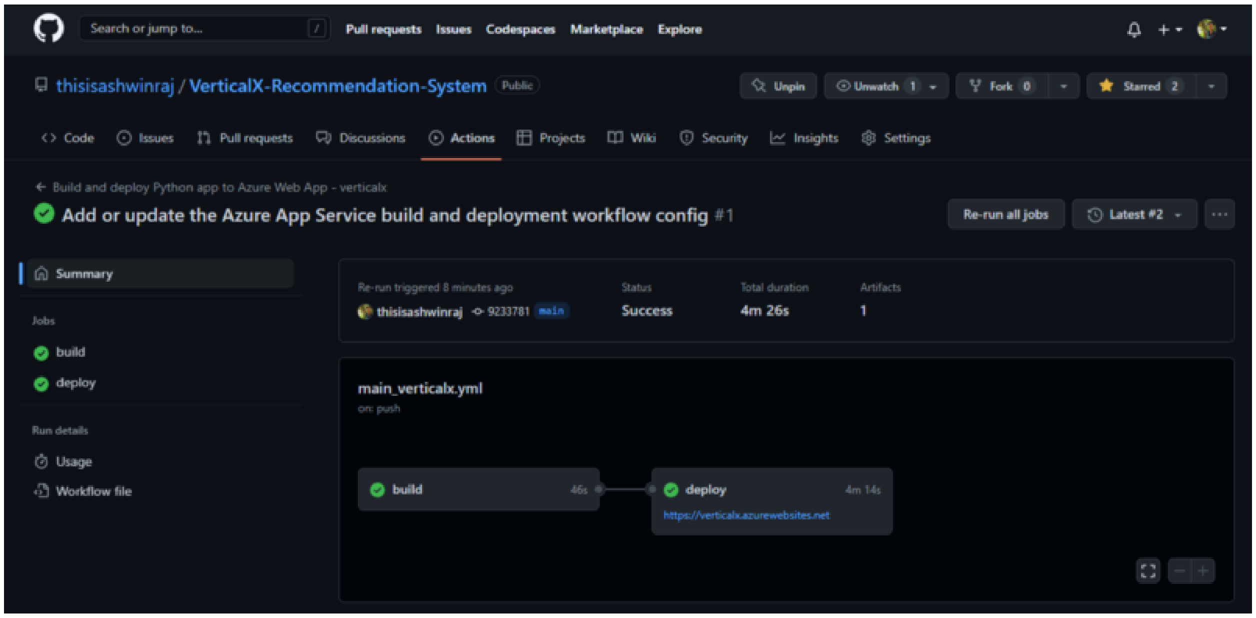
Platform settings

FTP state: FTPS only

Notifications

Updating web app settings
Successfully updated web app settings
a few seconds ago

Image from Author—Configure the startup command


thisisashwinraj / VerticalX-Recommendation-System Public

← Build and deploy Python app to Azure Web App - verticalx

Add or update the Azure App Service build and deployment workflow config #1

Re-run triggered 8 minutes ago: thisisashwinraj 9233781 main
Status: Success
Total duration: 4m 26s
Artifacts: 1

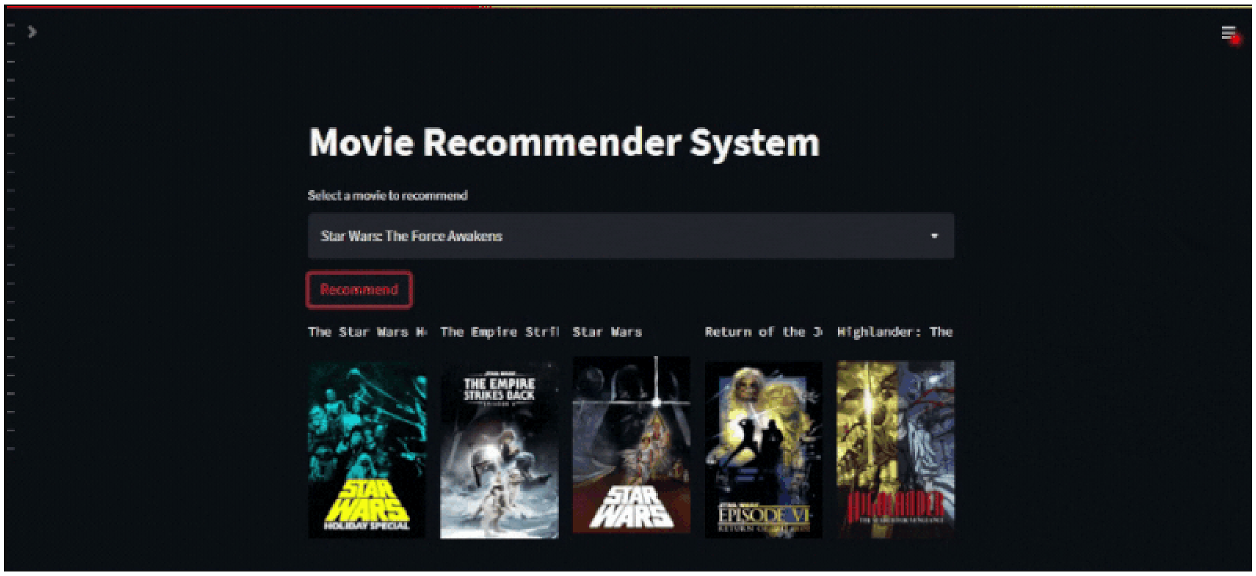
Jobs: build, deploy

main_verticalx.yml
on: push

build 46s → deploy 4m 14s https://verticalx.azurewebsites.net

Image from Author— Wait for the code to be deployed

To generate recommendations, simply select a movie from the dropdown list, and hit the recommend button. The recommendation system will now show you the top 5 recommendations


Movie Recommender System

Select a movie to recommend

Star Wars: The Force Awakens

Recommend

The Star Wars H | The Empire Strik | Star Wars | Return of the J | Highlander: The

Image from Author—Demo of the Movie Recommendation System

Knowledge Sharing and Best Practices

1. **Curse of Dimensionality:** While working with high-dimensional data, cosine distance is used in place of euclidean distance as the former gives a better measure of distance.

2. **API Key Secrecy:** While working with API keys, make sure to keep them confidential and protected from unauthorized access. This can be achieved by means of best practices such as storing them in secure location and by using multi-factor authentication

Challenges in Implementing the Solution

The pickle files used in building the recommendation system exceed the file size limit of GitHub. To upload such files, you'd need to use Git LFS. Every account receives up to 1 GB of free storage and bandwidth. If your bandwidth quota exceeds this limit, you may need to purchase an additional quota for Git LFS otherwise your application will not be able to fetch the pickle files

Business Benefits

Movie recommendation systems can provide a number of benefits for businesses in the entertainment industry. One of the main benefits is the ability to increase customer engagement and satisfaction by providing personalized recommendations to users. This can lead to increased repeat business, positive word-of-mouth, and customer loyalty. Additionally, these systems can help businesses to better understand their customers and their viewing habits, allowing them to make more informed decisions about content creation and distribution strategy. This results in increased revenue through targeted advertising and product placement.

Conclusion and Final Thoughts

You can further optimize the performance of your recommendation system by fine-tuning the model parameters or by switching to some more dynamic algorithms. You can also deploy your recommendation system after containerizing the app using Docker and Azure container registry.

References

1. TMDb Movies Dataset:
   https://www.kaggle.com/datasets/balaka18/tmdb-top-10000-popular-movies-dataset
2. TMDb API Site:
   https://developers.themoviedb.org/3/getting-started/introduction
3. Streamlit Documentation:
   https://docs.streamlit.io/
4. GitHub Repository:
   https://github.com/thisisashwinraj/VerticalX-Recommendation-System
5. Scikit-Learn Documentation:
   https://scikit-learn.org/stable/user_guide.html
6. Azure App Service:
   https://learn.microsoft.com/en-us/azure/app-service/quickstart-python?tabs=flask

Written by Ashwin Raj (linkedin.com/in/thisisashwinraj/)