CINEVERSE

WEB APPLICATION

MVP 1

CLIENT: MARKETING LINE OF BUSINESS, XYZ CORPORATION

BDAT1004 FINAL PROJECT

ABOUT THE TEAM

Chanpreet Kaur

Project Role: Developer

Experience: Automation Analyst Co-op, TD Bank (Sept'22 – Present)

Responsibilities:

- Design, develop and test software applications and systems
- Create and maintain the front-end and back-end of the software
- Troubleshoot and debug issues in the software
- Work closely with other team members, such as the project manager, business analyst, and data analyst, to ensure the software meets the project's requirements and specifications
- Stay up to date with new technologies and programming languages that could be beneficial to the project

ABOUT THE TEAM

Dyutita Juneja

Project Role: Developer

Responsibilities:

- Gather and analyze business requirements from stakeholders
- Develop use cases, functional requirements, and other documentation to describe the project's requirements and specifications
- Work with the project manager and development team to ensure the software meets the business requirements
- Conduct user acceptance testing to ensure the software meets the needs of stakeholders
- Serve as a liaison between the business stakeholders and the development team

ABOUT THE TEAM

Sangeeta Gollapalli

Project Role: Project Manager

Responsibilities:

Develop and oversee the project plan, including timelines and budgets

Communicate with stakeholders to ensure the project meets their expectations and requirements

Coordinate with team members to ensure everyone is on track and meeting their deadlines

Manage project risks and resolve issues as they arise

Track and report on project progress and performance

Ensure the project is delivered on time, within budget, and to the satisfaction of stakeholders

ABOUT THE TEAM

Kritika Verma

Project Role: Data Analyst

Responsibilities:

Identify and acquire relevant data for the project

Analyze and clean data to ensure accuracy and completeness

Develop data models and schemas to organize and structure the data

Design and develop databases to store and manage the data

Work with the development team to ensure the software can access and use the data effectively

Continuously monitor and analyze the data to identify insights and trends that could be useful to the project.

TOOLS/ TECHNOLOGIES USED

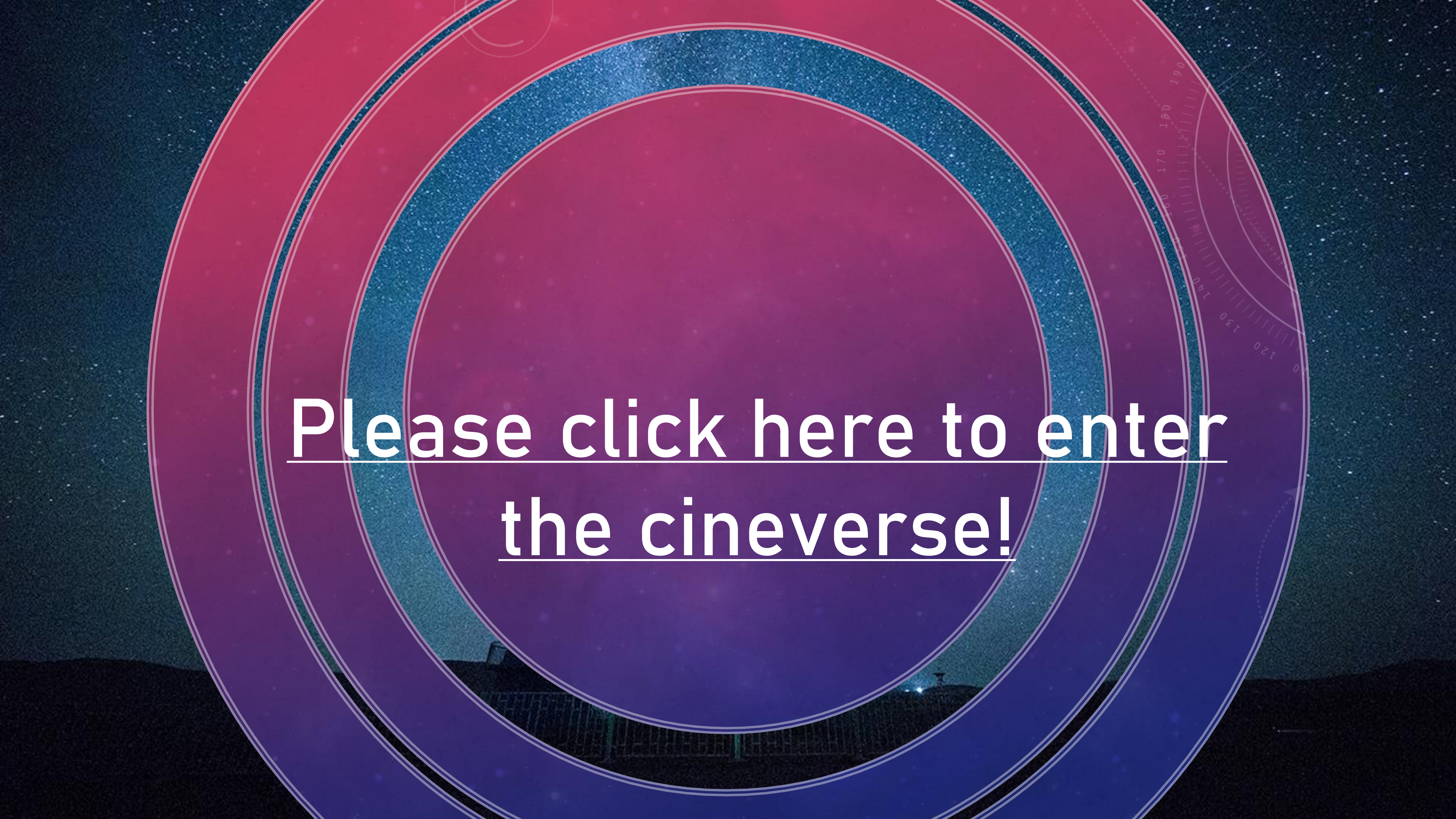
- Python
- Cloud SQL
- Flask
- HTML
- Google Charts

BUSINESS REASONS FOR CINEVERSE

- Welcome to our website, where we've gathered the top 250 movies on IMDb to help you develop your marketing strategies! As a marketing agency, you know that staying on top of the latest trends and insights is crucial for success. That's why we've put together this list of popular movies that can help you stay in tune with the cultural zeitgeist and understand the stories that resonate with audiences.

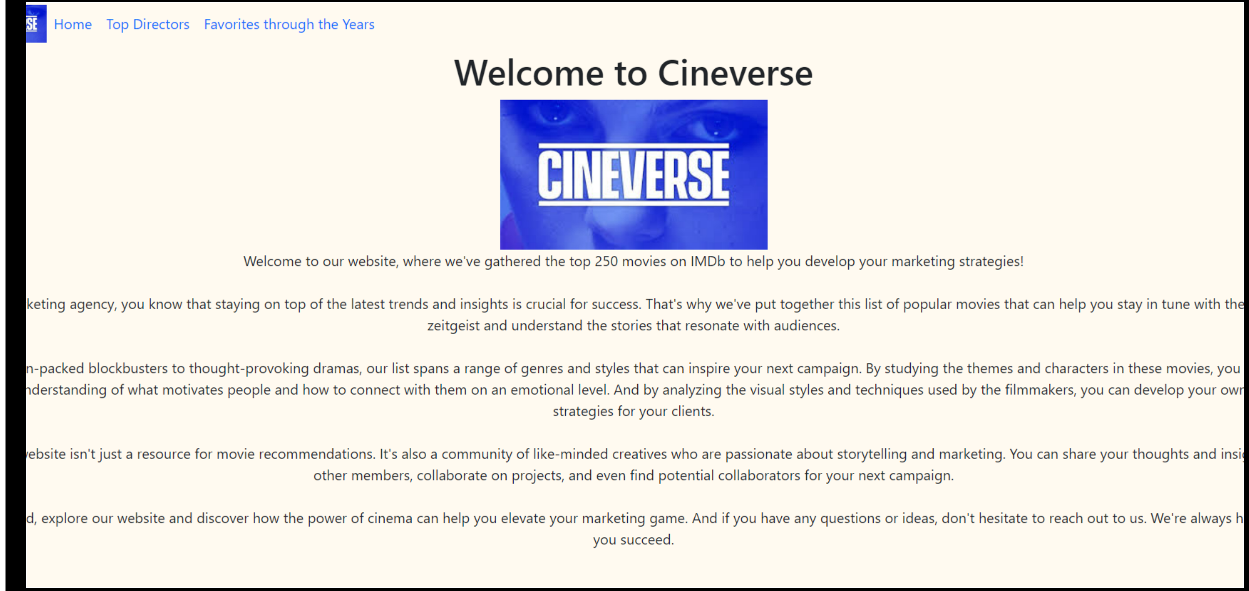
- From action-packed blockbusters to thought-provoking dramas, our list spans a range of genres and styles that can inspire your next campaign. By studying the themes and characters in these movies, you can gain a deeper understanding of what motivates people and how to connect with them on an emotional level. And by analyzing the visual styles and techniques used by the filmmakers, you can develop your own creative strategies for your clients.

- But our website isn't just a resource for movie recommendations. It's also a community of like-minded creatives who are passionate about storytelling and marketing. You can share your thoughts and insights with other members, collaborate on projects, and even find potential collaborators for your next campaign.



Please click here to enter the cineverse!

THE WEB PAGES



Welcome to Cineverse

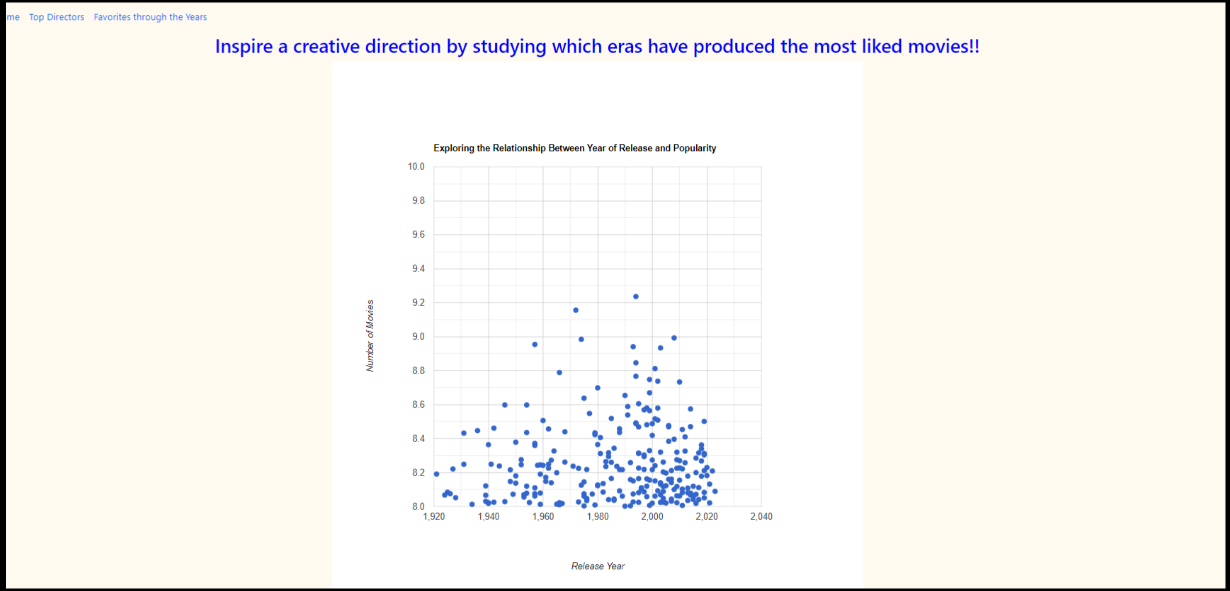
Welcome to our website, where we've gathered the top 250 movies on IMDb to help you develop your marketing strategies!

keting agency, you know that staying on top of the latest trends and insights is crucial for success. That's why we've put together this list of popular movies that can help you stay in tune with the zeitgeist and understand the stories that resonate with audiences.

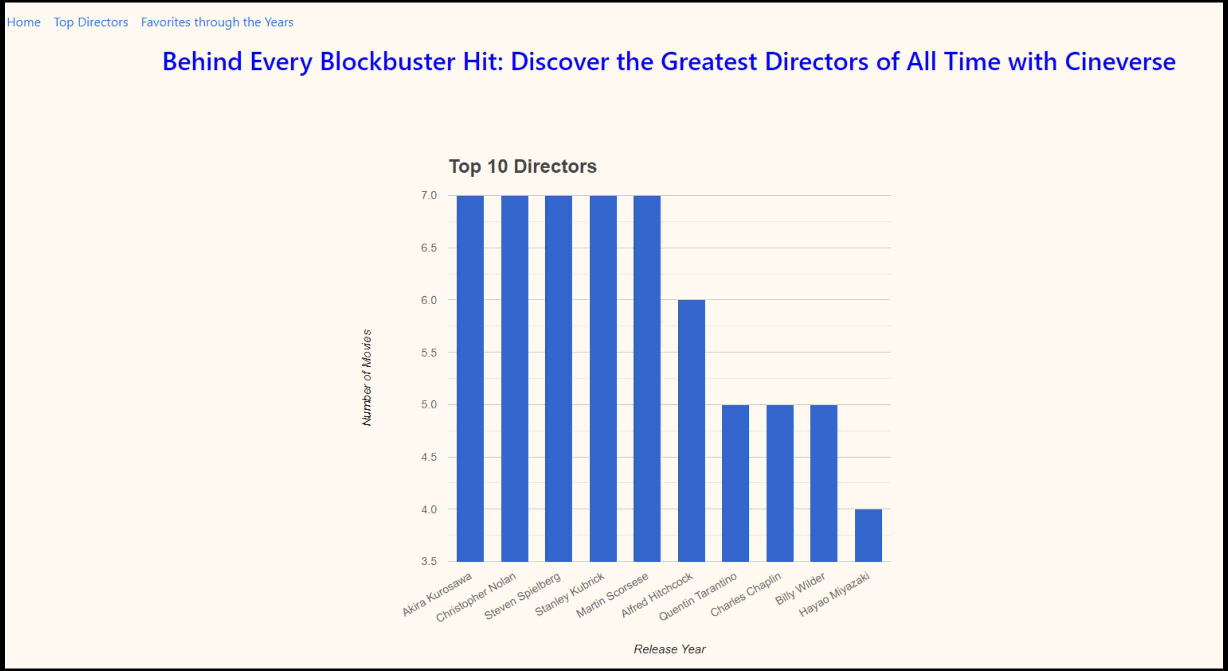
n-packed blockbusters to thought-provoking dramas, our list spans a range of genres and styles that can inspire your next campaign. By studying the themes and characters in these movies, you nderstanding of what motivates people and how to connect with them on an emotional level. And by analyzing the visual styles and techniques used by the filmmakers, you can develop your own strategies for your clients.

ebsite isn't just a resource for movie recommendations. It's also a community of like-minded creatives who are passionate about storytelling and marketing. You can share your thoughts and insi other members, collaborate on projects, and even find potential collaborators for your next campaign.

d, explore our website and discover how the power of cinema can help you elevate your marketing game. And if you have any questions or ideas, don't hesitate to reach out to us. We're always h you succeed.

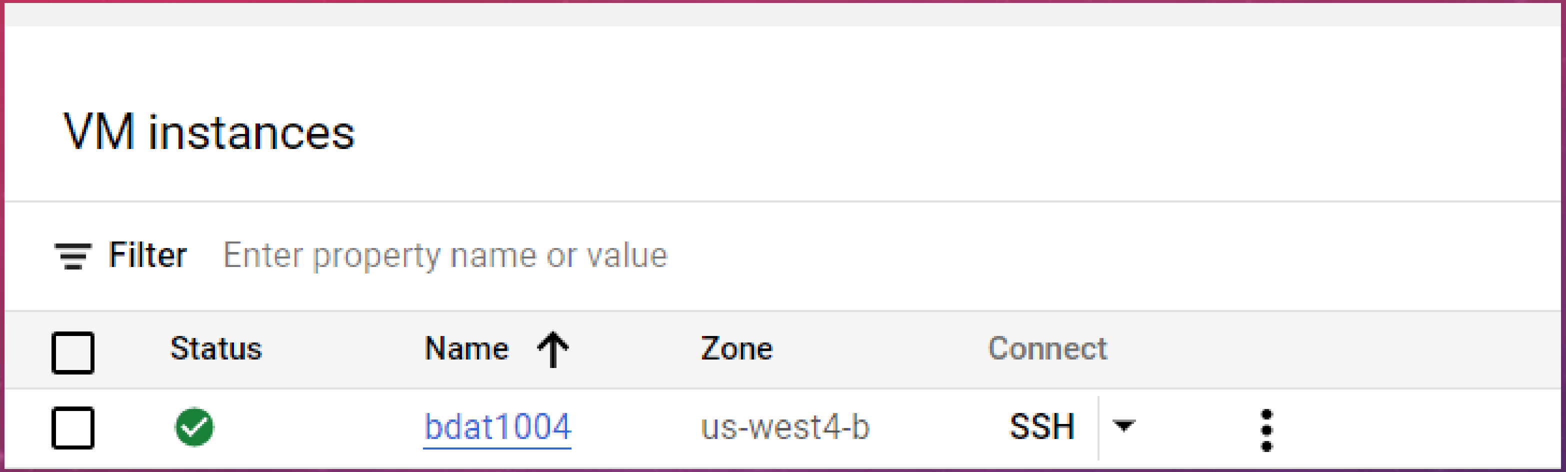


Inspire a creative direction by studying which eras have produced the most liked movies!!



Behind Every Blockbuster Hit: Discover the Greatest Directors of All Time with Cineverse

GOOGLE CLOUD SQL INSTANCE

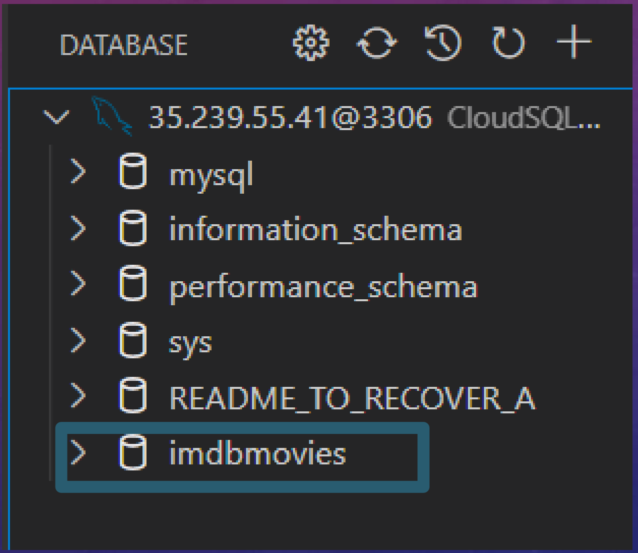


VM instances

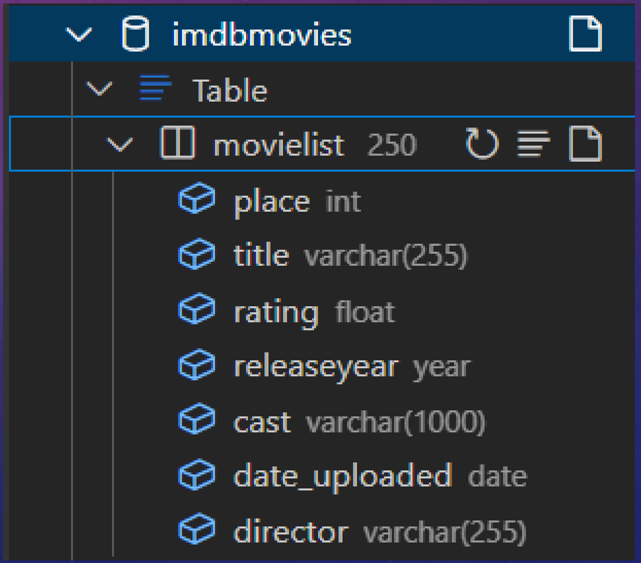
Filter Enter property name or value

	Status	Name ↑	Zone	Connect
☐	✓	bdat1004	us-west4-b	SSH

DATABASE & TABLES (SCREEN CAPTURE FROM VS CODE)

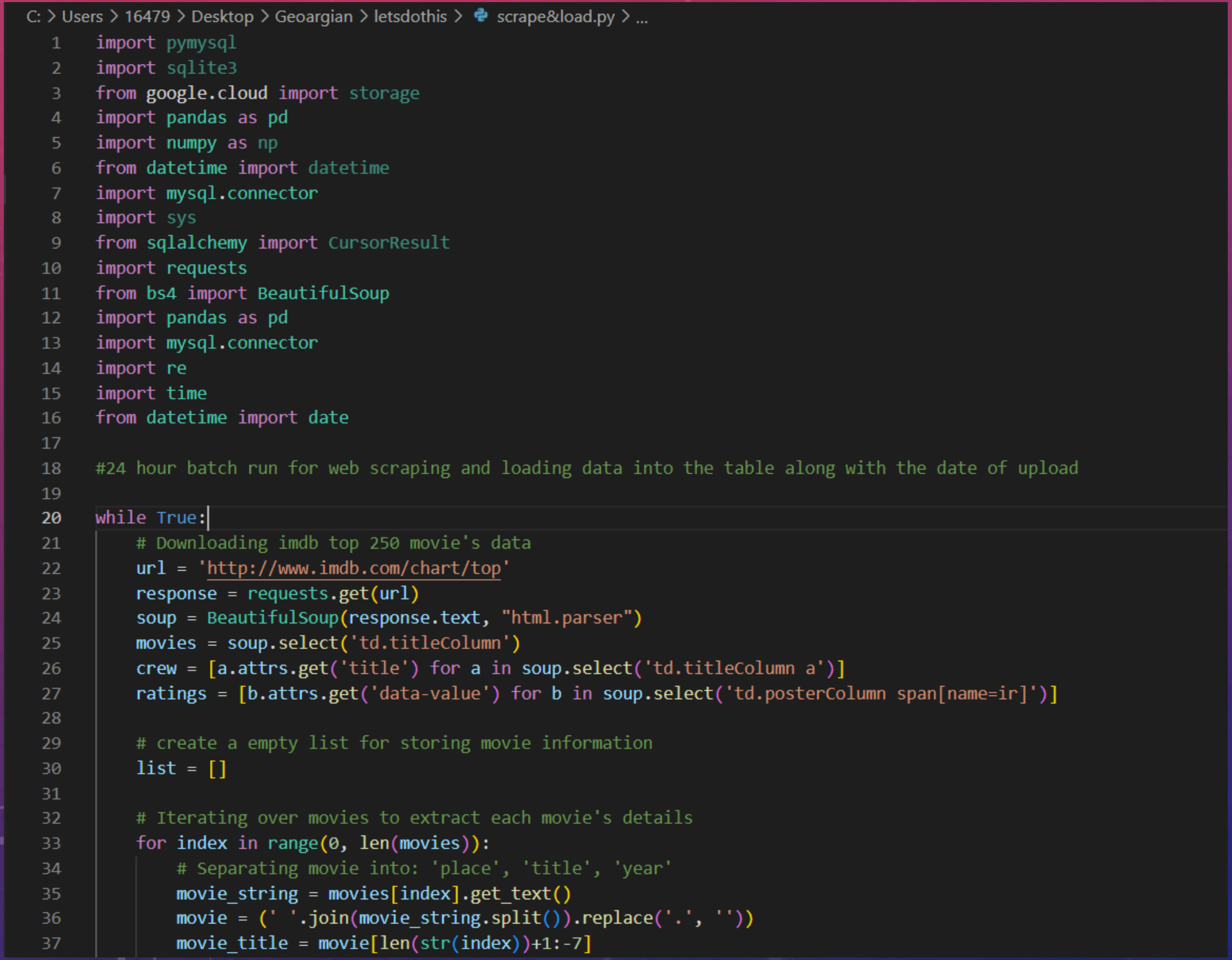


```
DATABASE
35.239.55.41@3306 CloudSQL...
  mysql
  information_schema
  performance_schema
  sys
  README_TO_RECOVER_A
  imdbmovies
```



```
imdbmovies
  Table
    movielist 250
      place int
      title varchar(255)
      rating float
      releaseyear year
      cast varchar(1000)
      date_uploaded date
      director varchar(255)
```

SCRIPT FOR WEB SCRAPING DATA & STORING IT

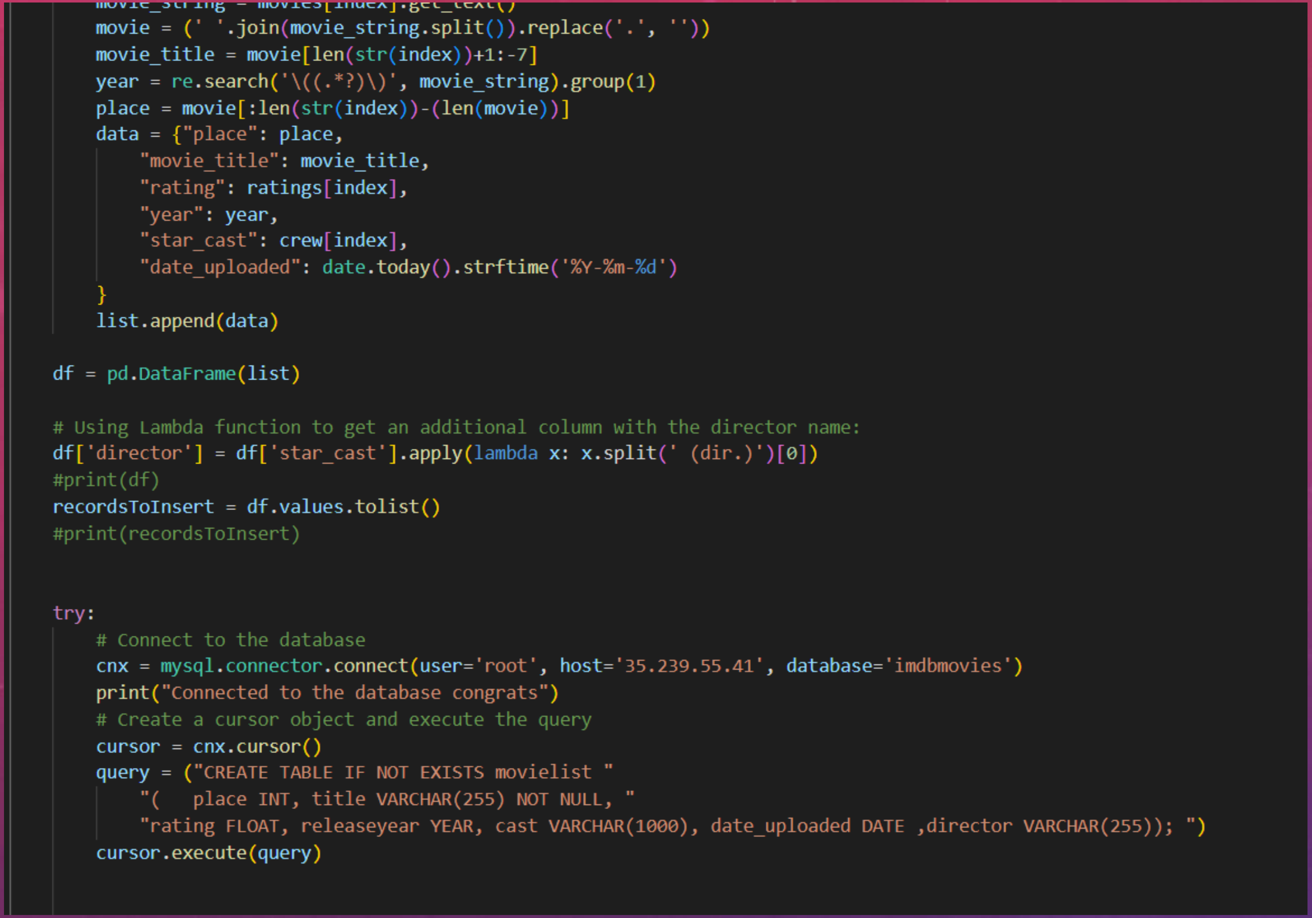


C: > Users > 16479 > Desktop > Geoargian > letsdothis > scrape&load.py > ...

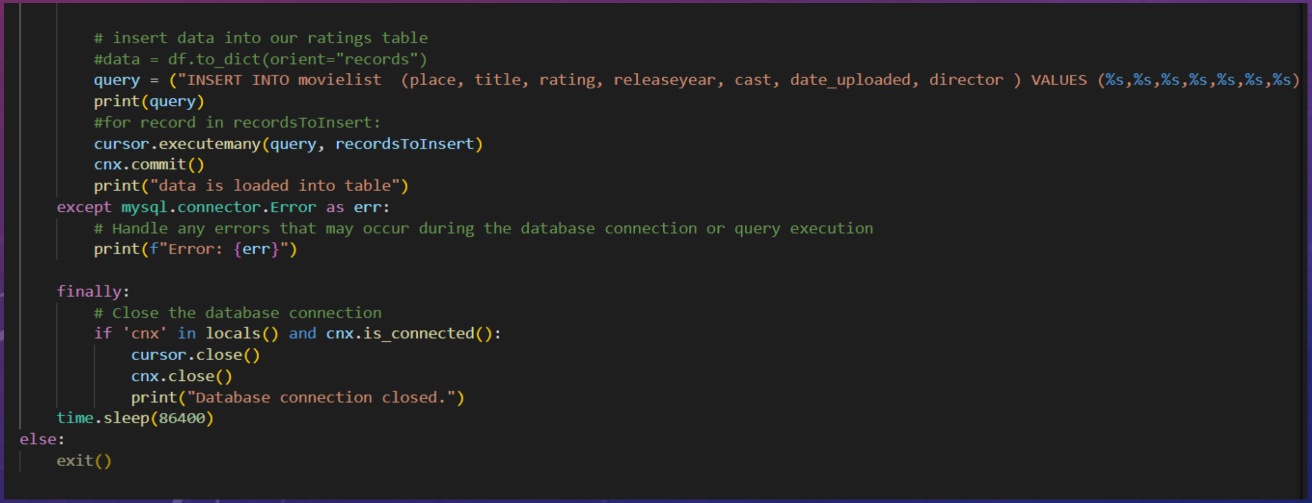
```python
import pymysql
import sqlite3
from google.cloud import storage
import pandas as pd
import numpy as np
from datetime import datetime
import mysql.connector
import sys
from sqlalchemy import CursorResult
import requests
from bs4 import BeautifulSoup
import pandas as pd
import mysql.connector
import re
import time
from datetime import date

#24 hour batch run for web scraping and loading data into the table along with the date of upload

while True:
    # Downloading imdb top 250 movie's data
    url = 'http://www.imdb.com/chart/top'
    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")
    movies = soup.select('td.titleColumn')
    crew = [a.attrs.get('title') for a in soup.select('td.titleColumn a')]
    ratings = [b.attrs.get('data-value') for b in soup.select('td.posterColumn span[name=ir]')]

    # create a empty list for storing movie information
    list = []

    # Iterating over movies to extract each movie's details
    for index in range(0, len(movies)):
        # Separating movie into: 'place', 'title', 'year'
        movie_string = movies[index].get_text()
        movie = (' '.join(movie_string.split()).replace('.', ''))
        movie_title = movie[len(str(index))+1:-7]
```

- The script uses several Python packages, including pymysql, sqlite3, google.cloud, pandas, numpy, datetime, mysql.connector, sys, sqlalchemy, requests, bs4, and re.

- The while loop in the script is set to run every 24 hours (time.sleep(86400)) to periodically scrape the IMDb top 250 movies list and update the database table with the new data.



```
        movie_string = movies[index].get_text()
        movie = (' '.join(movie_string.split()).replace('.', ''))
        movie_title = movie[len(str(index))+1:-7]
        year = re.search('\((.*?)\)', movie_string).group(1)
        place = movie[:len(str(index))-(len(movie))]
        data = {"place": place,
            "movie_title": movie_title,
            "rating": ratings[index],
            "year": year,
            "star_cast": crew[index],
            "date_uploaded": date.today().strftime('%Y-%m-%d')
        }
        list.append(data)

    df = pd.DataFrame(list)

    # Using Lambda function to get an additional column with the director name:
    df['director'] = df['star_cast'].apply(lambda x: x.split(' (dir.)')[0])
    #print(df)
    recordsToInsert = df.values.tolist()
    #print(recordsToInsert)


    try:
        # Connect to the database
        cnx = mysql.connector.connect(user='root', host='35.239.55.41', database='imdbmovies')
        print("Connected to the database congrats")
        # Create a cursor object and execute the query
        cursor = cnx.cursor()
        query = ("CREATE TABLE IF NOT EXISTS movielist "
            "(   place INT, title VARCHAR(255) NOT NULL, "
            "rating FLOAT, releaseyear YEAR, cast VARCHAR(1000), date_uploaded DATE ,director VARCHAR(255)); ")
        cursor.execute(query)
```

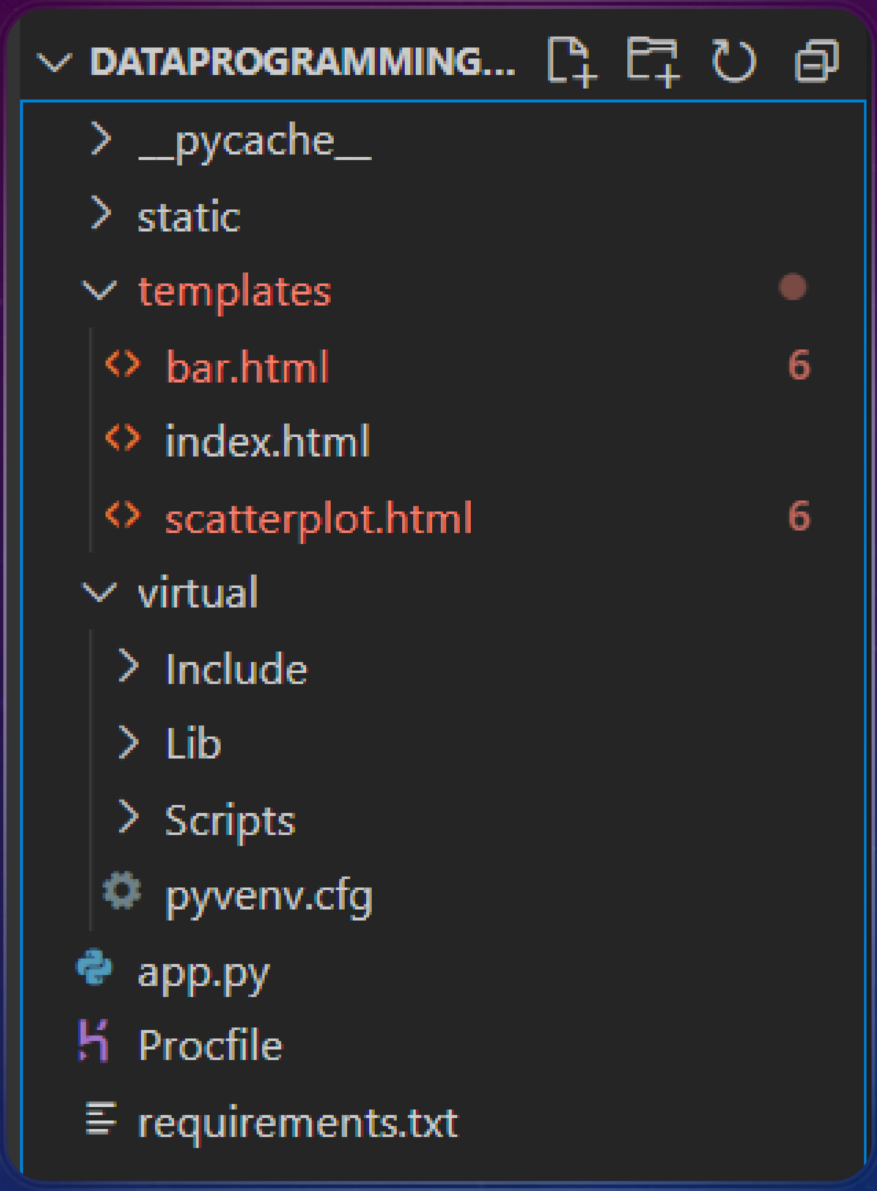


```
        # insert data into our ratings table
        #data = df.to_dict(orient="records")
        query = ("INSERT INTO movielist  (place, title, rating, releaseyear, cast, date_uploaded, director ) VALUES (%s,%s,%s,%s,%s,%s,%s)
        print(query)
        #for record in recordsToInsert:
        cursor.executemany(query, recordsToInsert)
        cnx.commit()
        print("data is loaded into table")
    except mysql.connector.Error as err:
        # Handle any errors that may occur during the database connection or query execution
        print(f"Error: {err}")

    finally:
        # Close the database connection
        if 'cnx' in locals() and cnx.is_connected():
            cursor.close()
            cnx.close()
            print("Database connection closed.")
    time.sleep(86400)
else:
    exit()
```

- The script first uses the requests and BeautifulSoup packages to scrape the IMDb website and extract the movie title, release year, rating, star cast, and director for each movie in the top 250 list. It then converts this data into a pandas DataFrame, adds a new column with the director name using a lambda function, and stores the DataFrame records in a list.

- The try block in the code is used for error handling. In this case, it attempts to connect to the MySQL database using the mysql.connector module, creates a cursor object, and executes a query to create a new table named "movielist" if it does not already exist. It then inserts the movie data into the "movielist" table.

- If there are any errors during the execution of the code within the try block, the except block will be executed, and the error will be printed to the console. The purpose of using the try-except block is to catch any exceptions or errors that may occur during the database connection or query execution, and handle them in a controlled manner, instead of causing the program to crash.

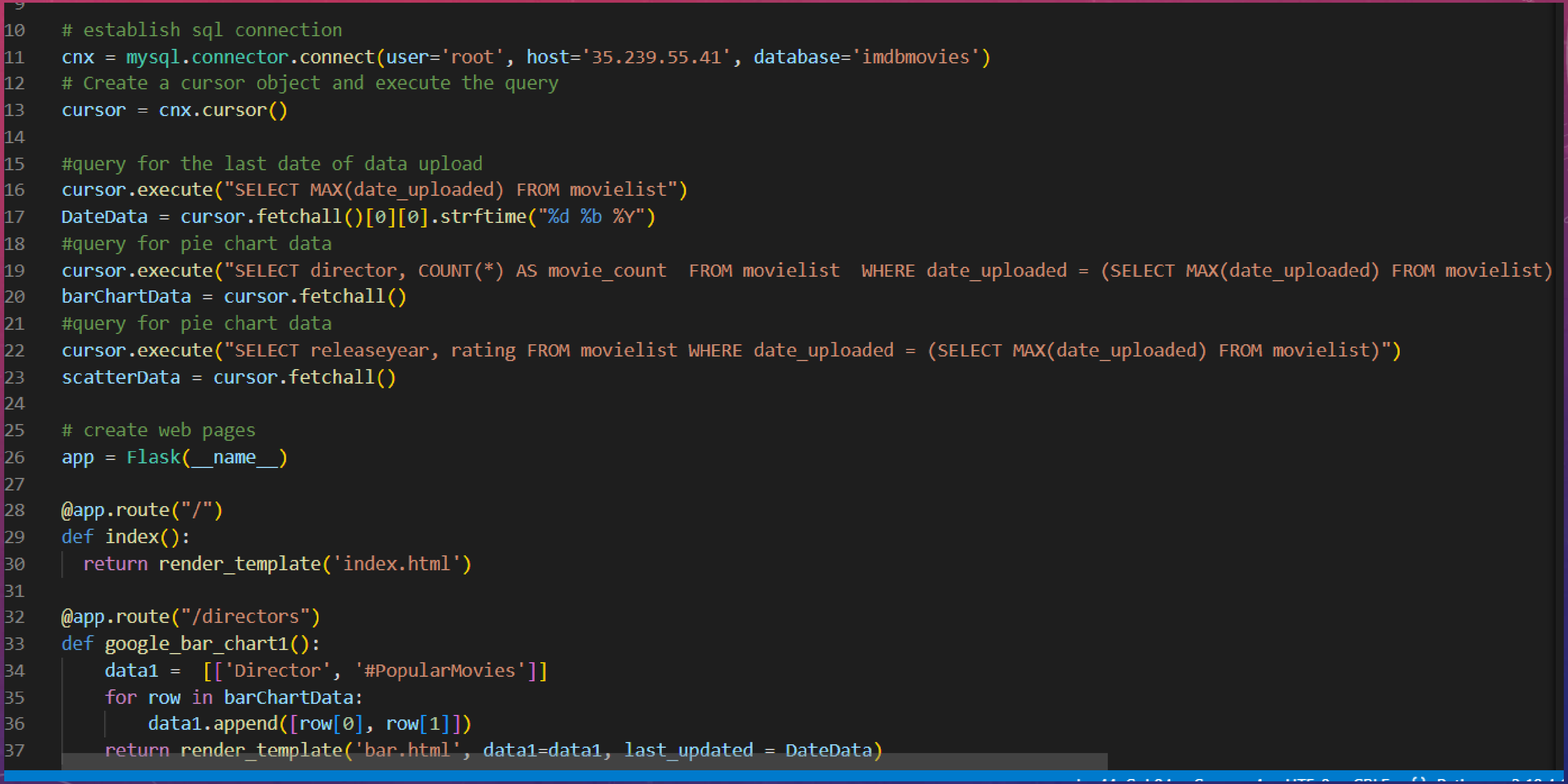


```
DATAPROGRAMMING...
  > __pycache__
  > static
  v templates
      bar.html            6
      index.html
      scatterplot.html    6
  v virtual
      > Include
      > Lib
      > Scripts
      pyvenv.cfg
    app.py
    Procfile
    requirements.txt
```

FOLDER STRUCTURE FOR THE WEB APPLICATION

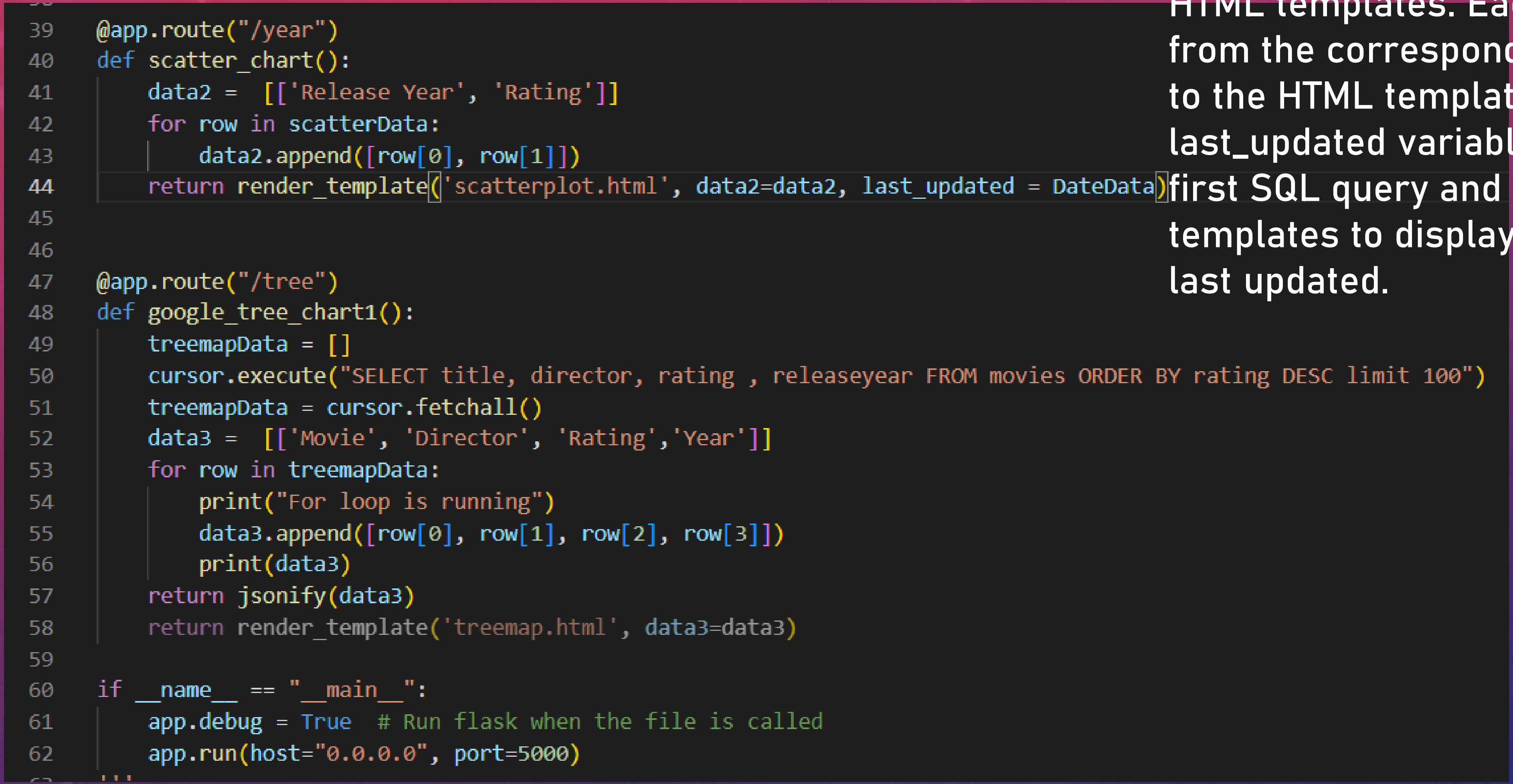
- Framework: Flask
- Server-side code: Python
- Client-side code: JavaScript
- Data Visualizations: Google Chart

PYTHON FLASK WEB APPLICATION



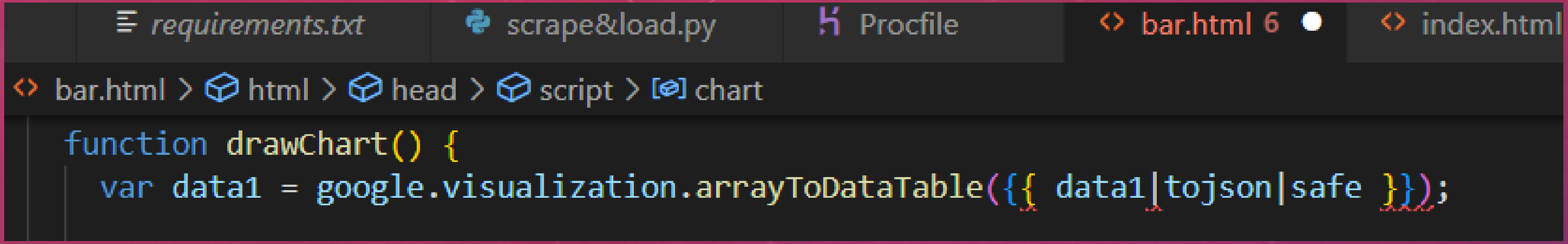
```
# establish sql connection
cnx = mysql.connector.connect(user='root', host='35.239.55.41', database='imdbmovies')
# Create a cursor object and execute the query
cursor = cnx.cursor()

#query for the last date of data upload
cursor.execute("SELECT MAX(date_uploaded) FROM movielist")
DateData = cursor.fetchall()[0][0].strftime("%d %b %Y")
#query for pie chart data
cursor.execute("SELECT director, COUNT(*) AS movie_count  FROM movielist  WHERE date_uploaded = (SELECT MAX(date_uploaded) FROM movielist)
barChartData = cursor.fetchall()
#query for pie chart data
cursor.execute("SELECT releaseyear, rating FROM movielist WHERE date_uploaded = (SELECT MAX(date_uploaded) FROM movielist)")
scatterData = cursor.fetchall()

# create web pages
app = Flask(__name__)

@app.route("/")
def index():
  return render_template('index.html')

@app.route("/directors")
def google_bar_chart1():
    data1 =  [['Director', '#PopularMovies']]
    for row in barChartData:
        data1.append([row[0], row[1]])
    return render_template('bar.html', data1=data1, last_updated = DateData)
```

The first part of the code establishes a connection to the database and executes three SQL queries to retrieve data for two chart types: a bar chart showing the top 10 directors with the most popular movies, a scatter plot showing the relationship between the release year and rating of movies, The second part of the code defines three routes for each chart type and creates web pages using HTML templates. Each route retrieves the data from the corresponding SQL query and passes it to the HTML template as a variable. The last_updated variable is also defined using the first SQL query and passed to all three HTML templates to display the date when the data was last updated.



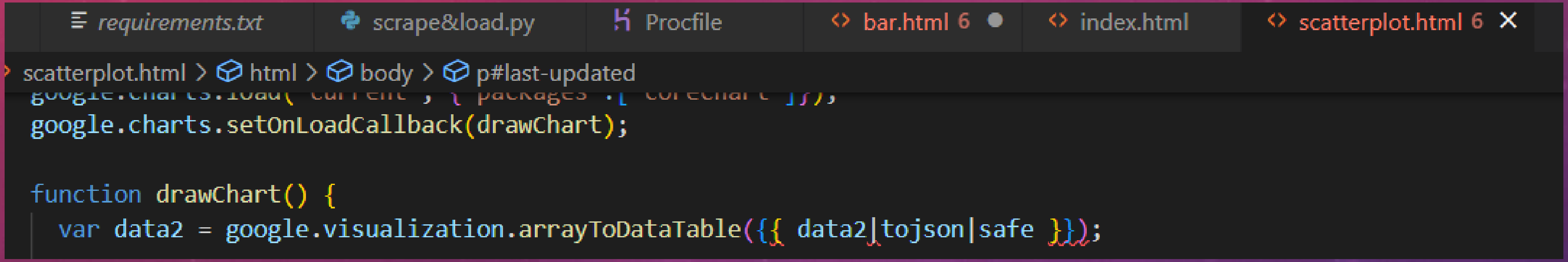
```python
@app.route("/year")
def scatter_chart():
    data2 =  [['Release Year', 'Rating']]
    for row in scatterData:
        data2.append([row[0], row[1]])
    return render_template('scatterplot.html', data2=data2, last_updated = DateData)


@app.route("/tree")
def google_tree_chart1():
    treemapData = []
    cursor.execute("SELECT title, director, rating , releaseyear FROM movies ORDER BY rating DESC limit 100")
    treemapData = cursor.fetchall()
    data3 =  [['Movie', 'Director', 'Rating','Year']]
    for row in treemapData:
        print("For loop is running")
        data3.append([row[0], row[1], row[2], row[3]])
        print(data3)
    return jsonify(data3)
    return render_template('treemap.html', data3=data3)

if __name__ == "__main__":
    app.debug = True  # Run flask when the file is called
    app.run(host="0.0.0.0", port=5000)
```

Finally, the app.run() function is called to start the Flask web application on the local host with port 5000. By accessing each of the three routes in the web browser, the user can see the three different types of charts displaying movie data.


requirements.txt | scrape&load.py | Procfile | bar.html 6 | index.html

bar.html > html > head > script > chart

```
function drawChart() {
  var data1 = google.visualization.arrayToDataTable({{ data1|tojson|safe }});
```


requirements.txt | scrape&load.py | Procfile | bar.html 6 | index.html | scatterplot.html 6

scatterplot.html > html > body > p#last-updated

```
google.charts.load('current', {'packages':['corechart']});
google.charts.setOnLoadCallback(drawChart);

function drawChart() {
  var data2 = google.visualization.arrayToDataTable({{ data2|tojson|safe }});
```

This is a JavaScript function that uses the Google Charts API to draw a bar chart. The function takes the data1 variable as input, which is a two-dimensional array of data representing the values to be plotted in the chart. The google.visualization.arrayToDataTable() function converts this data into a format that can be used by the Google Charts API.

The {{ data1|tojson|safe }} code is a Jinja2 template that inserts the value of the data1 variable into the JavaScript code. The tojson filter converts the data1 variable to a JSON string, which can be parsed by the Google Charts API. The safe filter ensures that the string is not automatically escaped by Jinja2, which could cause errors in the JavaScript code.

So go ahead, explore our website and discover how the power of cinema can help you elevate your marketing game. And if you have any questions or ideas, don't hesitate to reach out to us. We're always here to help you succeed.

THANK YOU!

WE LOOK FORWARD TO COLLABORATING WITH YOU ON FUTURE PROJECTS