Name:Krithika Balaji
Unique ID: 67777695

Project code: https://github.com/krithikab99/SI507-FinalProject

Data Sources:

1) **List of Christmas movies and Halloween movies**
   a) **Origin and format:**
      i) **Csv files:**
      christmas.csv:https://www.kaggle.com/datasets/adityak957/imdb-christmas-movies-from-20162022
      halloween.csv:https://www.kaggle.com/datasets/PromptCloudHQ/imdb-horror-movie-dataset

      ii) **API:** https://www.omdbapi.com

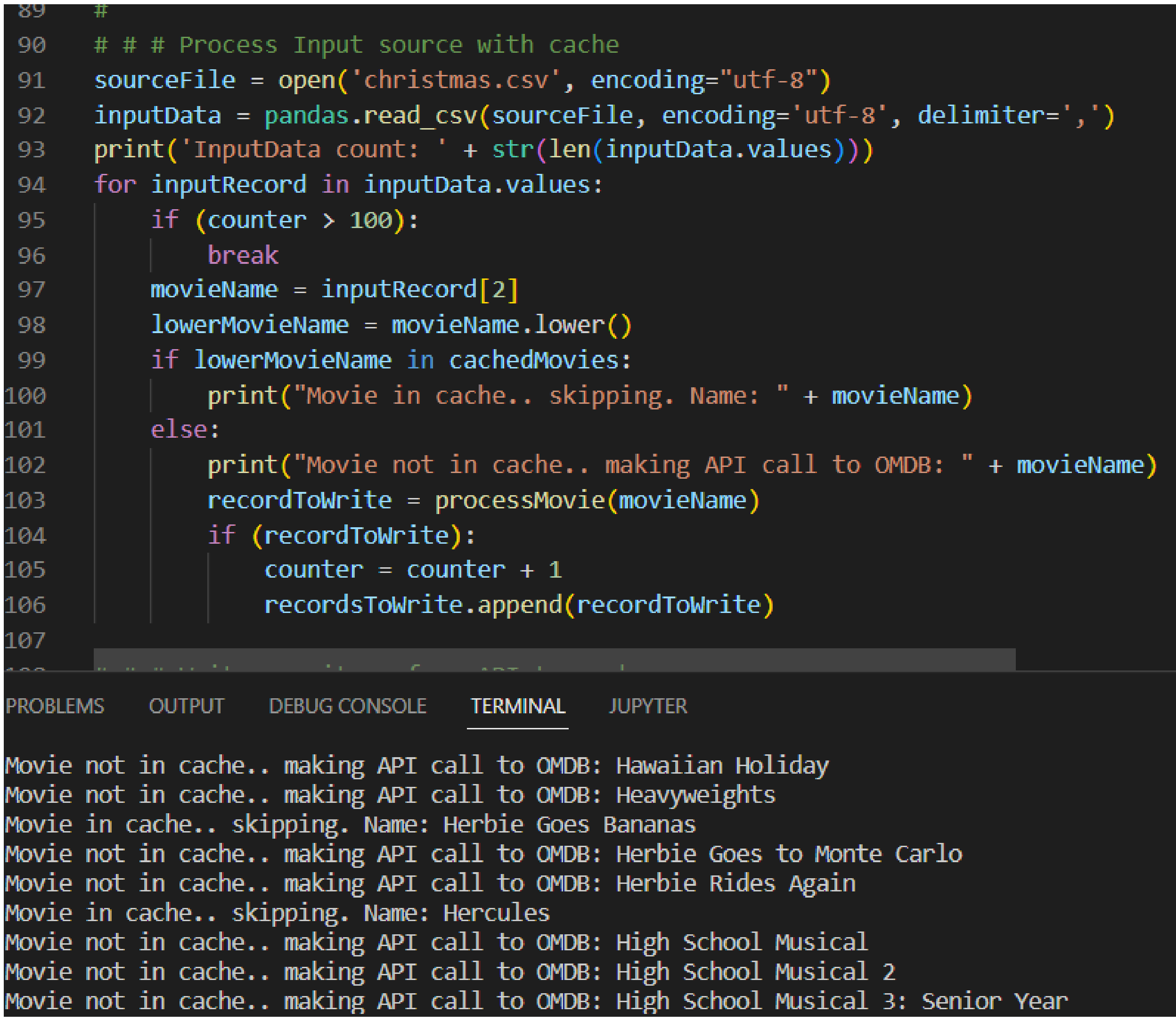
      Originally got a dataset in csv format and used it to get only a list of christmas movies. The names of the movies were then passed onto the API inorder to extract more data like Title,Year,Box Office,Genre.

   b) **How was data accessed? Was caching used?**
   The data was accessed by making the api call. The parameters passed to this api call include the name of the movie (‘t’:moviename) and the apikey. Caching was used to store the results of the api call.

   c) **Summary of data:**
   # of records available -> 278
   # of records retrieved -> 250
   The purpose of extracting the above mentioned records is to ensure that a proper correlation is made between the ratings of the movie against its release date. The title is the value that joins the two datasets. The Year and the Genre have also been taken to ensure that the user can have some interaction with the system while choosing the movie at a later stage.

   d) **Evidence of caching:**

   Sample:
```
Movie not in cache.. making API call to OMDB: Ice Princess
Movie not in cache.. making API call to OMDB: Inner Workings
Movie in cache.. skipping. Name: Inside Out
Movie in cache.. skipping. Name: Inspector Gadget
Movie not in cache.. making API call to OMDB: Inspector Gadget 2
Movie not in cache.. making API call to OMDB: Into the Grand Canyon
Movie not in cache.. making API call to OMDB: Into the Okavango
```



```
    #
    # # # Process Input source with cache
    sourceFile = open('christmas.csv', encoding="utf-8")
    inputData = pandas.read_csv(sourceFile, encoding='utf-8', delimiter=',')
    print('InputData count: ' + str(len(inputData.values)))
    for inputRecord in inputData.values:
        if (counter > 100):
            break
        movieName = inputRecord[2]
        lowerMovieName = movieName.lower()
        if lowerMovieName in cachedMovies:
            print("Movie in cache.. skipping. Name: " + movieName)
        else:
            print("Movie not in cache.. making API call to OMDB: " + movieName)
            recordToWrite = processMovie(movieName)
            if (recordToWrite):
                counter = counter + 1
                recordsToWrite.append(recordToWrite)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
Movie not in cache.. making API call to OMDB: Hawaiian Holiday
Movie not in cache.. making API call to OMDB: Heavyweights
Movie in cache.. skipping. Name: Herbie Goes Bananas
Movie not in cache.. making API call to OMDB: Herbie Goes to Monte Carlo
Movie not in cache.. making API call to OMDB: Herbie Rides Again
Movie in cache.. skipping. Name: Hercules
Movie not in cache.. making API call to OMDB: High School Musical
Movie not in cache.. making API call to OMDB: High School Musical 2
Movie not in cache.. making API call to OMDB: High School Musical 3: Senior Year
```

2) **List of Valentine's Day movies**

   a) **Origin:**
   https://www.imdb.com/search/keyword/?keywords=boyfriend-girlfriend-relationship%2Ckiss%2Clove&mode=detail&page=1&ref_=kw_nxt&sourceid=chrome&ie=UTF-8&sort=moviemeter

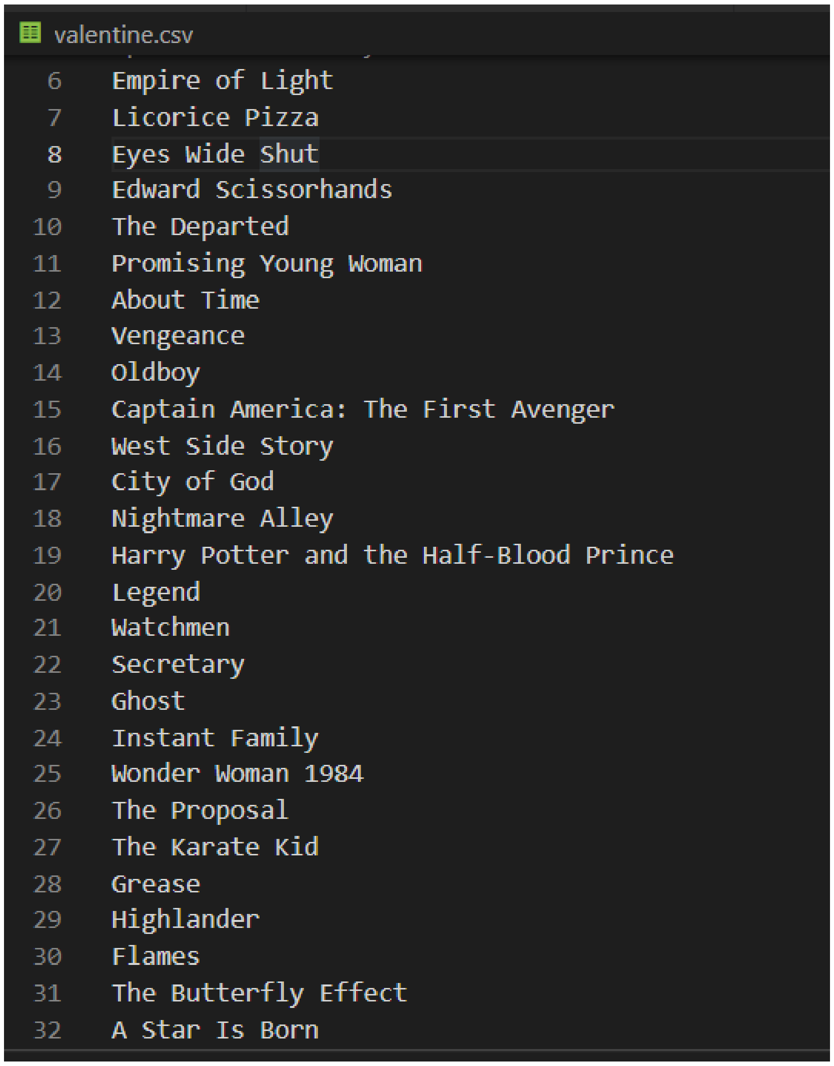
   b) **Format**: It was a URL .The data from the URL is then stored into an Excel sheet.

   c) **How was data accessed? Was caching used**?
   Data was accessed by scraping multiple html pages and storing only the required keyword (The movie name) into the excel sheet. The data in this excel sheet will then be passed into the omdb API.

d) **Summary of data:**
# of records available -> 485
# of records retrieved -> 400

Though I was able to retrieve all the values from the URL by scraping multiple pages, I was only able to retrieve the complete data for around 400 records . This is because OMDB did not have the remaining records stored.

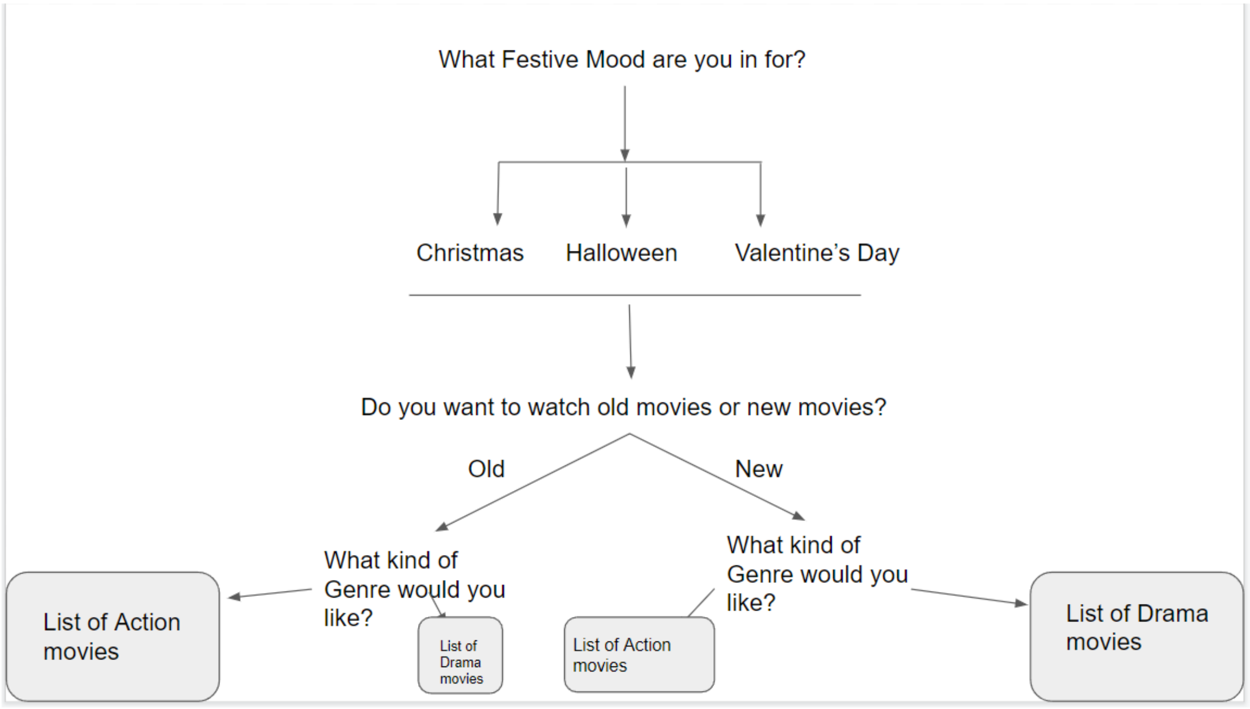
e) **Evidence of caching:**
These records will not be cached, they will be taken from the excel file that its stored in.



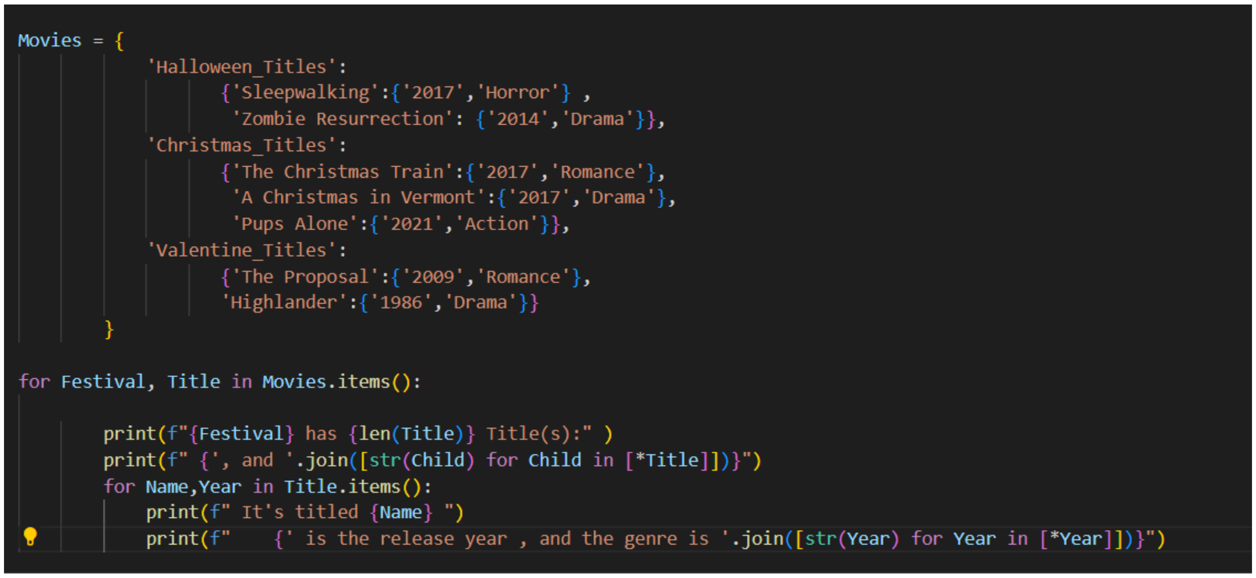
valentine.csv

```
Empire of Light
Licorice Pizza
Eyes Wide Shut
Edward Scissorhands
The Departed
Promising Young Woman
About Time
Vengeance
Oldboy
Captain America: The First Avenger
West Side Story
City of God
Nightmare Alley
Harry Potter and the Half-Blood Prince
Legend
Watchmen
Secretary
Ghost
Instant Family
Wonder Woman 1984
The Proposal
The Karate Kid
Grease
Highlander
Flames
The Butterfly Effect
A Star Is Born
```

Data Structure:
I plan on using the Trees Data structure. There will be a user interaction involved and the tree structure will be used to suggest movies that match closely with the users inputs.


What Festive Mood are you in for?
Christmas
Halloween
Valentine's Day
Do you want to watch old movies or new movies?
Old
New
What kind of Genre would you like?
What kind of Genre would you like?
List of Action movies
List of Drama movies
List of Action movies
List of Drama movies

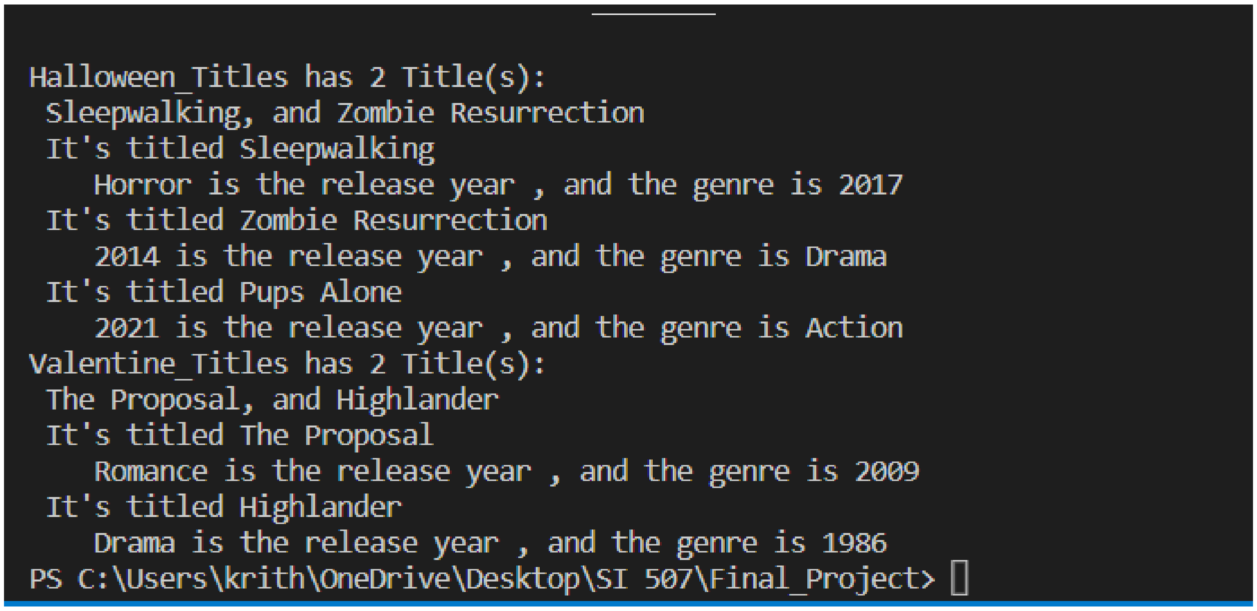
Sample code:


```
Movies = {
            'Halloween_Titles':
                  {'Sleepwalking':{'2017','Horror'} ,
                   'Zombie Resurrection': {'2014','Drama'}},
            'Christmas_Titles':
                  {'The Christmas Train':{'2017','Romance'},
                   'A Christmas in Vermont':{'2017','Drama'},
                   'Pups Alone':{'2021','Action'}},
            'Valentine_Titles':
                  {'The Proposal':{'2009','Romance'},
                  'Highlander':{'1986','Drama'}}
        }

for Festival, Title in Movies.items():

        print(f"{Festival} has {len(Title)} Title(s):" )
        print(f" {', and '.join([str(Child) for Child in [*Title]])}")
        for Name,Year in Title.items():
            print(f" It's titled {Name} ")
            print(f"    {' is the release year , and the genre is '.join([str(Year) for Year in [*Year]])}")
```

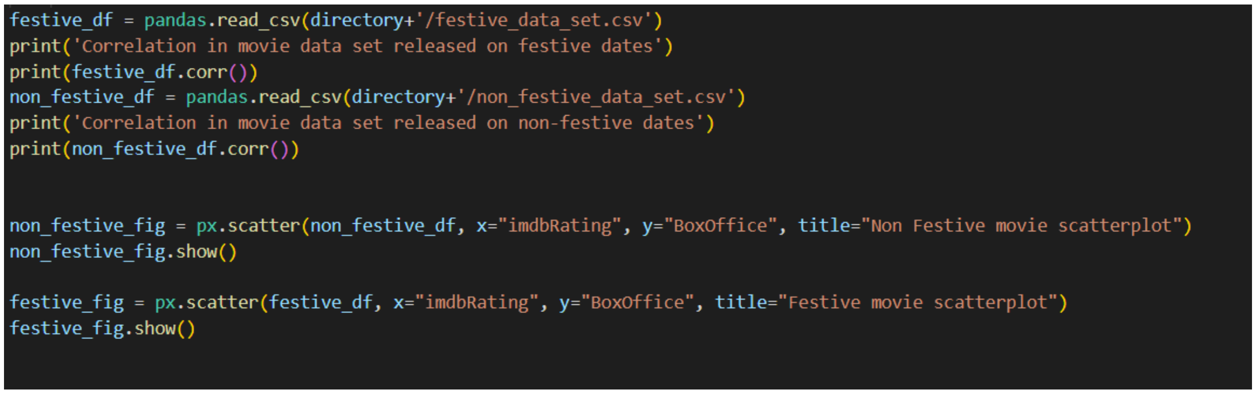
Sample output:



```
Halloween_Titles has 2 Title(s):
 Sleepwalking, and Zombie Resurrection
 It's titled Sleepwalking
    Horror is the release year , and the genre is 2017
 It's titled Zombie Resurrection
    2014 is the release year , and the genre is Drama
 It's titled Pups Alone
    2021 is the release year , and the genre is Action
Valentine_Titles has 2 Title(s):
 The Proposal, and Highlander
 It's titled The Proposal
    Romance is the release year , and the genre is 2009
 It's titled Highlander
    Drama is the release year , and the genre is 1986
PS C:\Users\krith\OneDrive\Desktop\SI 507\Final_Project> 
```

Interaction/Presentation:

The graph has been plotted to show the correlation between the imdb rating and the box office collection. The data set has been split into festive and non festive data.
Plotly has been used.

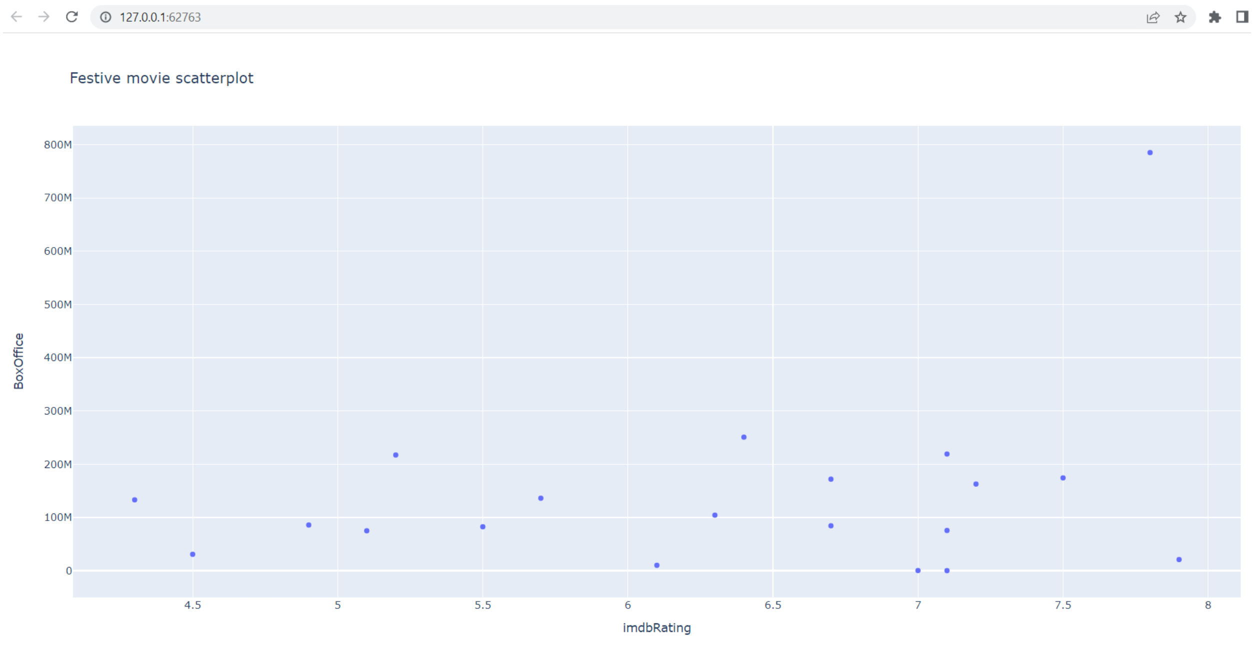


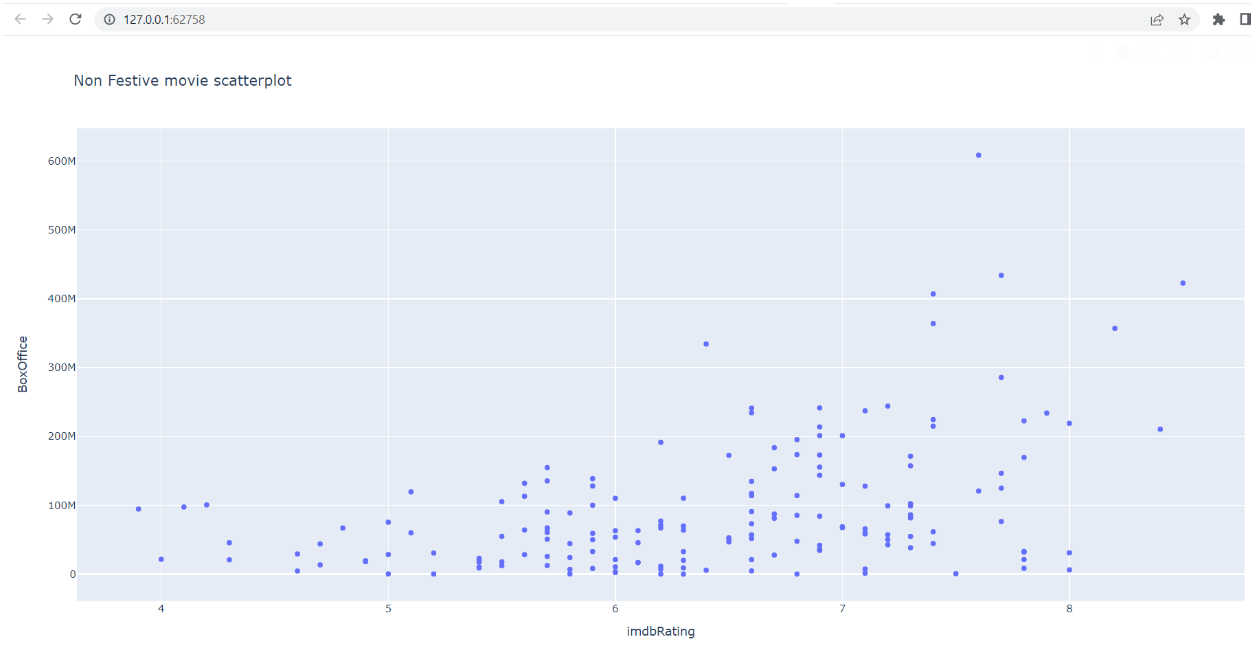
```
festive_df = pandas.read_csv(directory+'/festive_data_set.csv')
print('Correlation in movie data set released on festive dates')
print(festive_df.corr())
non_festive_df = pandas.read_csv(directory+'/non_festive_data_set.csv')
print('Correlation in movie data set released on non-festive dates')
print(non_festive_df.corr())


non_festive_fig = px.scatter(non_festive_df, x="imdbRating", y="BoxOffice", title="Non Festive movie scatterplot")
non_festive_fig.show()

festive_fig = px.scatter(festive_df, x="imdbRating", y="BoxOffice", title="Festive movie scatterplot")
festive_fig.show()
```

Plot:


127.0.0.1:62763
Festive movie scatterplot
BoxOffice
0
100M
200M
300M
400M
500M
600M
700M
800M
imdbRating
4.5
5
5.5
6
6.5
7
7.5
8


127.0.0.1:62758
Non Festive movie scatterplot
BoxOffice
0
100M
200M
300M
400M
500M
600M
imdbRating
4
5
6
7
8