SVS DATA ANALYTICS

MOVIE DATA ANALYSIS AND INSIGHTS VISUALIZATION

DS5003A Data Engineering Course Project

Team Details

1) Sanjiv Kumar (142402009@smail.iitpkd.ac.in)

2) Y Siva Kumar (142402014@smail.iitpkd.ac.in)

3) Y Vishnu Vardhan (142402016@smail.iitpkd.ac.in)

TABLE OF CONTENTS

PROBLEM DESCRIPTION

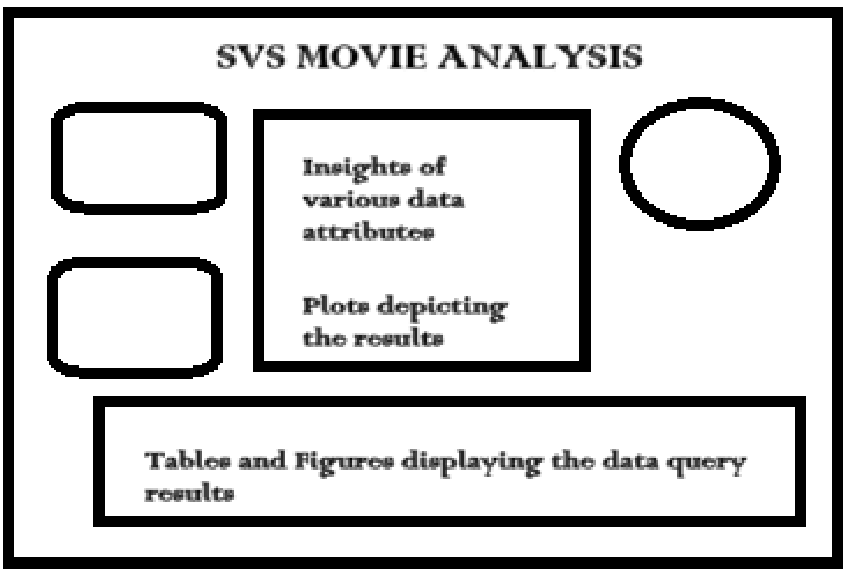
1) You are provided with a CSV file containing detailed information about movies.
2) Perform the necessary data pre-processing and design a relational database schema to store this movie data.
3) Ensure optimization of queries.
4) Develop a dashboard that can connect to the designed database and visualize various aspects of the data.

You are required to analyse the data and answer the following questions as a baseline. Additional insights and custom analysis are encouraged to provide a deeper understanding of the dataset: Who are the top 5 directors by the number of movies they have directed? Which are the top 5 movies with the highest profit margin (revenue-to-budget ratio)? Which directors have the most movies in the top 100 grossing films? Which actors have the highest average vote in Sci-Fi movies? For each director, which actor have they collaborated with the most, and what is the highest-grossing movie from their collaboration? Is there a correlation between higher popularity scores and higher box office revenue across different genres? Which actors have the highest difference between their average ratings in Drama vs. Comedy movies? Which actors have the most appearances in high-rated movies across different genres?

PROJECT PLAN

Based on the problem statement, we came up with below features and functionalities to be implemented:

1) Our dashboard gives end users lot of meaningful insights and inferences about the movie dataset.
2) Data Preprocessing should involve efficient transformations of certain columns for effective database querying.
3) Normalization of the parent table ensuring lossless decomposition of data.
4) SQL Querying time reduction by utilizing JOIN functionality
5) Visual Plots in the Dashboard to be responsive and gives a complete analysis of the dataset.
6) Good Abstraction of Code and Efficient UI/UX Design
7) Dynamic Retrieval of Query results in order to plot the insights and draw conclusions.
8) Rough Plan of the Website



INDIVIDUAL CONTRIBUTIONS

1) Dashboard Design, Insight Visualizations and Query Optimization - All

2) Data Pre-Processing and Data Base Creation & Normalization - Siva

3) Data Base Querying and Establishing the connection between backend SQL Queries. - Sanjiv

4) Dashboard Development – Vishnu

5) Technical Documentation and Git Version Control - Vishnu

6) User Documentation and Report – Vishnu

TECH STACK USED

1. Data Pre-Processing: Jupyter Notebook, Pandas, Numpy
2. DataBase Management: PostgreSQL, pgAdmin 4, SQLAlchemy
3. Visualization and Dashboard: plotly, dash

OUTSTANDING FEATURES

1) Depicted more than 15 meaningful insights with efficient Dashboard Design. Plots displayed conclusions and inferences & were also holding metadata info

2) Dashboard developed is Responsive with the screen dimensions. Plots and Figures were too responsive with the change in screen UI.

3) Optimization in Data Insights Retrievals by efficient Normalization

DATA UNDERSTANDING

Given CSV file contains detailed information about movies, including metadata such as titles, genres, cast, directors, budget, revenue, release dates, and more.

File Name: movies.csv File Size: 6.56 MB

Dataset Dimensions: 10866 rows X 21 columns

Attributes Definition (Dataset Columns):

1. **'id'**: Unique identification number for each movie.
2. **'imdb_id'**: Movie's unique ID on the IMDb website.
3. **'popularity'**: Typically based on user activity, scores given to each movie indicating how popular the movie is.
4. **'budget'**: Amount of money spent to make the movie (in dollars).
5. **'revenue'**: The total amount of money earned by the movie (in dollars).
6. **'original_title'**: Movie's original title.
7. **'cast'**: Main actors names separated by the "|" symbol.
8. **'homepage'**: Official website of the movie.
9. **'director'**: Name of the movie's director.
10. **'tagline'**: A short phrase or slogan used in the movie's title.
11. **'keywords'**: Important terms or concepts related to the movie.
12. **'overview'**: Brief summary of the movie's plot.
13. **'runtime'**: The length of the movie in minutes.
14. **'genres'**: Categories or Types of the movies (like Action, Comedy etc).
15. **'production_companies'**: The companies responsible for the movie.
16. **'release_date'**: The date the movie was first released.
17. **'vote_count'**: The total number of votes the movie has received on IMDb.
18. **'vote_average'**: The average rating of the movie (on a scale of 1 to 10).
19. **'release_year'**: The year the movie was released.
20. **'budget_adj'**: Movie's budget adjusted for inflation.
21. **'revenue_adj'**: Movie's revenue adjusted for inflation.

DATA PREPROCESSING

Data Cleaning

Looking at the dataset, we figured out what rows are duplicated, and which columns are having missing/null values.

1) There is 1 row duplicated with id 42194

```
data[data['id'].duplicated() == True]
```

	id	imdb_id	popularity	budget	revenue	original_title	cast	homepage	director	tagline	...	overview	runtime	genr
2090	42194	tt0411951	0.59643	30000000	967000	TEKKEN	Jon Foo|Kelly Overton|Cary-Hiroyuki Tagawa|Ian...	NaN	Dwight H. Little	Survival is no game	...	In the year of 2039, after World Wars destroy ...	92	Crime|Drama|Action|Thriller|Scien Fictic

1 rows × 21 columns

Hence, we removed that redundant row, so that 'id' column would now satisfy the unique constraint property and will distinguish each and every movie uniquely.

```
df_cleaned = df_cleaned.drop_duplicates(subset=['id'])
```

After performing the check for null values in each column, we found that 9 out of 21 columns were having missing values.

```
data.isnull().sum().sort_values(ascending = False)
```

```
homepage              7930
tagline               2824
keywords              1493
production_companies  1030
cast                    76
director                44
genres                  23
imdb_id                 10
overview                 4
```

Hence, we decided to drop the columns which have more than 50% null values and those which are irrelevant for the data analysis. Also, special attention taken to the essential columns like 'title', 'budget', 'revenue' and we dropped those rows which have null values in these columns.

```
[34]: df_cleaned = df.dropna(thresh=len(df) * 0.5, axis=1)
```

```
[35]: df_cleaned = df_cleaned.dropna(subset=['original_title', 'budget', 'revenue'])
```

Results showed 5 columns were insignificant and we decided to drop them.

```
columns_to_drop = [ 'homepage', 'tagline', 'overview', 'production_companies', 'keywords']
df_cleaned = df_cleaned.drop(columns=columns_to_drop)
```

Finally, after data cleaning we got (10865, 16) dimensions of data.

```
[31]: df_cleaned.shape
```

```
[31]: (10865, 16)
```

```
id                 int64    runtime          int64
imdb_id           object    genres          object
popularity       float64    release_date    object
budget             int64    vote_count       int64
revenue            int64    vote_average   float64
original_title    object    release_year     int64
cast              object    budget_adj     float64
director          object    revenue_adj    float64
```

Data Transformations

'release_date' column having multiple date formats

We figured out 'release_date' column was having multiple date formats such as 02-04-2015 & 1/21/15 due to which, the DataFrame given object type for that column. Hence, we performed data transformation and made all rows into single date format.

```
df_cleaned['release_date'] = pd.to_datetime(df_cleaned['release_date'], errors='coerce')

df_cleaned[df_cleaned['release_date'].isnull()]

df_cleaned = df_cleaned.dropna(subset=['release_date'])
```

Replacing null values with appropriate terms

```
[32]: df_cleaned.isnull().sum().sort_values(ascending = False)

[32]: cast          76
      director      44
      genres        23
      imdb_id       10
```

Existing null values are less in number and present in 'cast', 'director', 'genres' and 'imdb_id' columns.

```
df_cleaned['imdb_id'].fillna("No_imdb_id", inplace = True)
df_cleaned['cast'].fillna("No Cast Info", inplace = True)
df_cleaned['director'].fillna("No Info about Director", inplace = True)
df_cleaned['genres'].fillna("No Info about Genres", inplace = True)
```

Conversion of Composite attributes to Single attributes

Splitting the contents of 'cast' & 'director' columns to have single valued records for efficient data analysis.

```
for column in ['cast', 'director']:
    df_cleaned = df_cleaned.drop(column, axis=1).join(df_cleaned[column].str.
                        split('|', expand=True).stack().
                        reset_index(level=1, drop=True).rename(column)
        )
```

By the above logic, initial 10865 rows of data got multiplicated to 57612

```
[33]: df_cleaned.shape

[33]: (10865, 16)
```

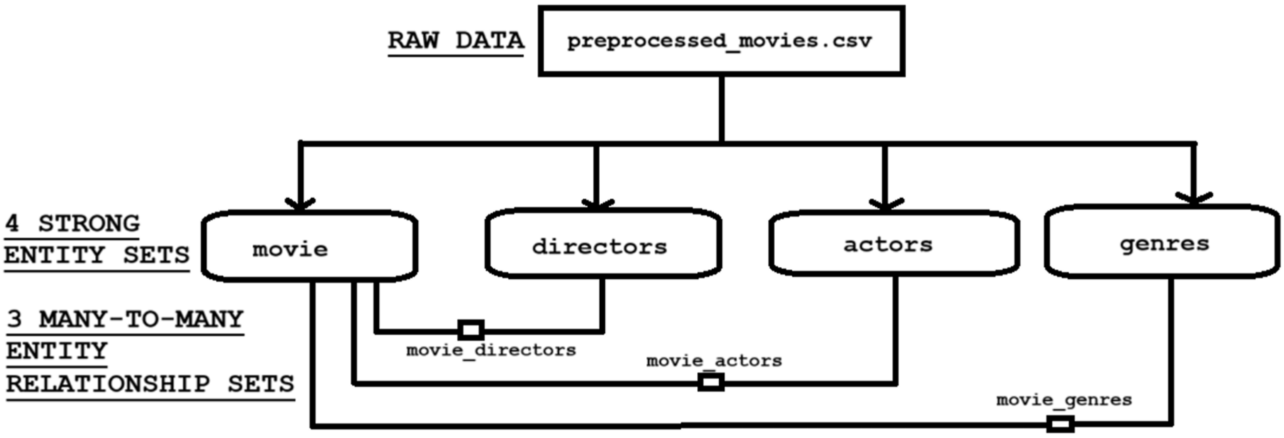
```
      df_final.drop_duplicates()

[49]: dff.shape

[49]: (57612, 16)
```

End of Data Preprocessing

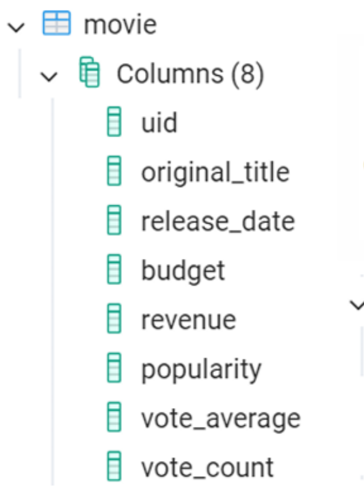
movies.csv	6.6 MB
preprocessed_movies.csv	8.5 MB

Note: Even though the dataset size got increased, by Data Normalization we will segregate the data into multiple tables by which querying complexity gets reduced.

DATABASE SCHEMA DIAGRAM



Details Of Entity Sets

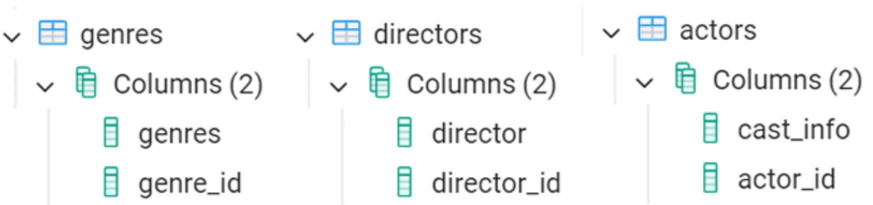


actors
- actor_id (Primary Key)

genres
- genre_id (Primary Key)

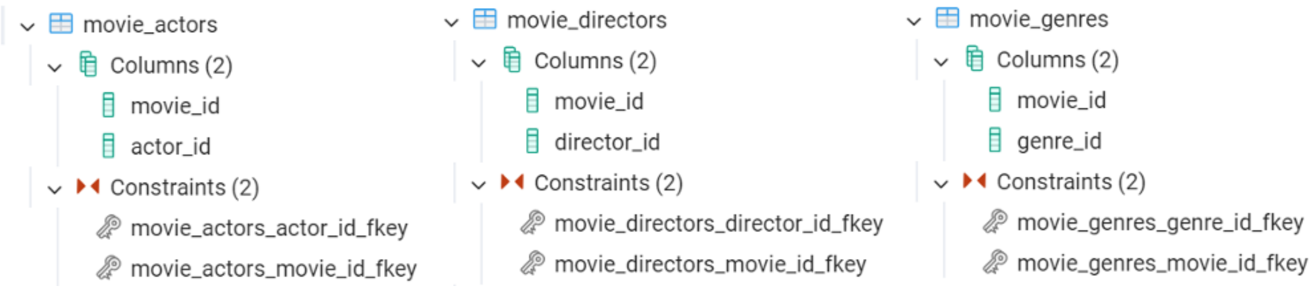
directors
- director_id (Primary Key)

movie
- uid (Primary Key)



In order to build connectivity among the 4 entity tables, extra 3 tables created.

Details Of Entity Relationship Sets



For the above 3 Tables, we ensure

movie_id of movie_actors, movie_directors, movie_genres are foreign keys referencing to the movie_id of movie table

actor_id of movie_actors is foreign key referencing to the actor_id of actors table

director_id of movie_directors is foreign key referencing to the director_id of directors table

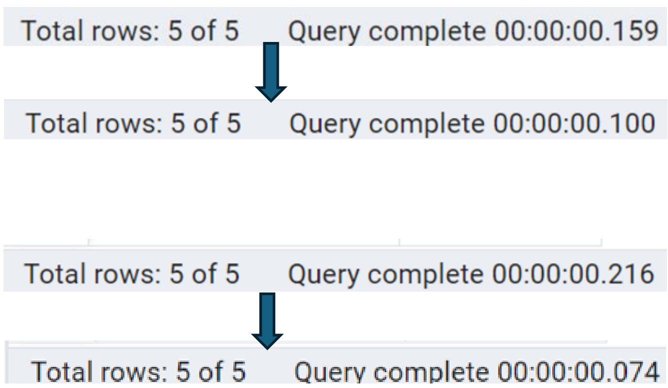
genre_id of movie_genres is foreign key referencing to the genre_id of genres table

DATABASE QUERYING

Therefore, via database normalization we tend to achieve efficient query computing. For a given analytical question:

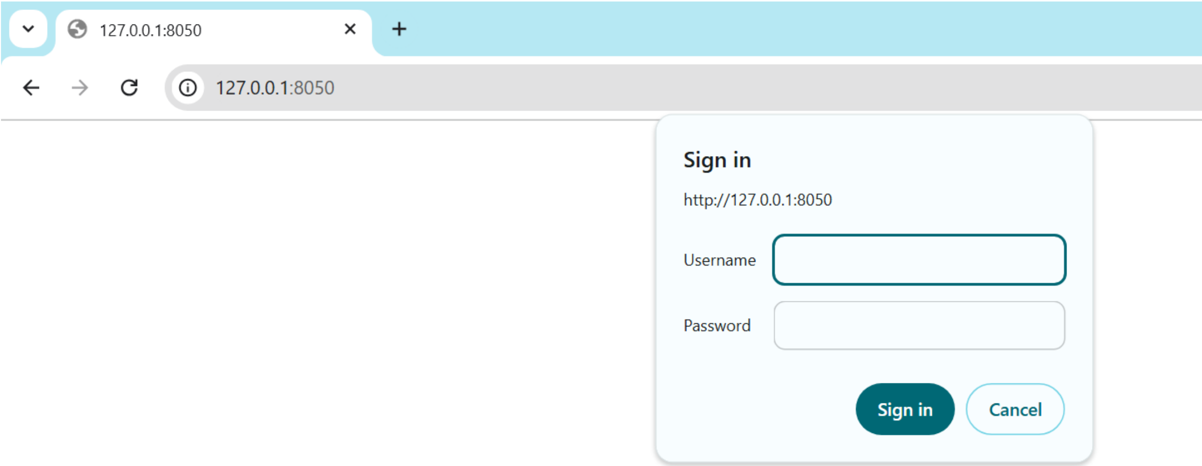
1. We figure out which Entities to be chosen, and the relevant attribute names
2. After knowing the entities, with the help of JOINS we will match the Entity Relationship Sets

Optimization in Query Runtime due to Normalization

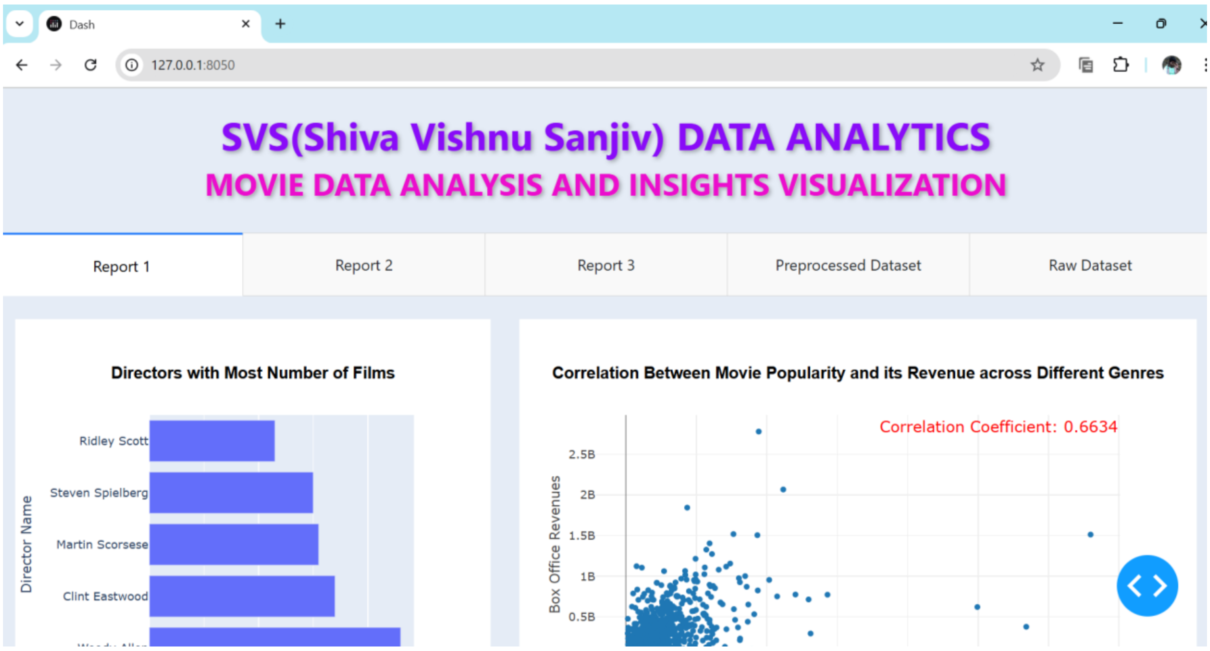


PROJECT DEMONSTRATION

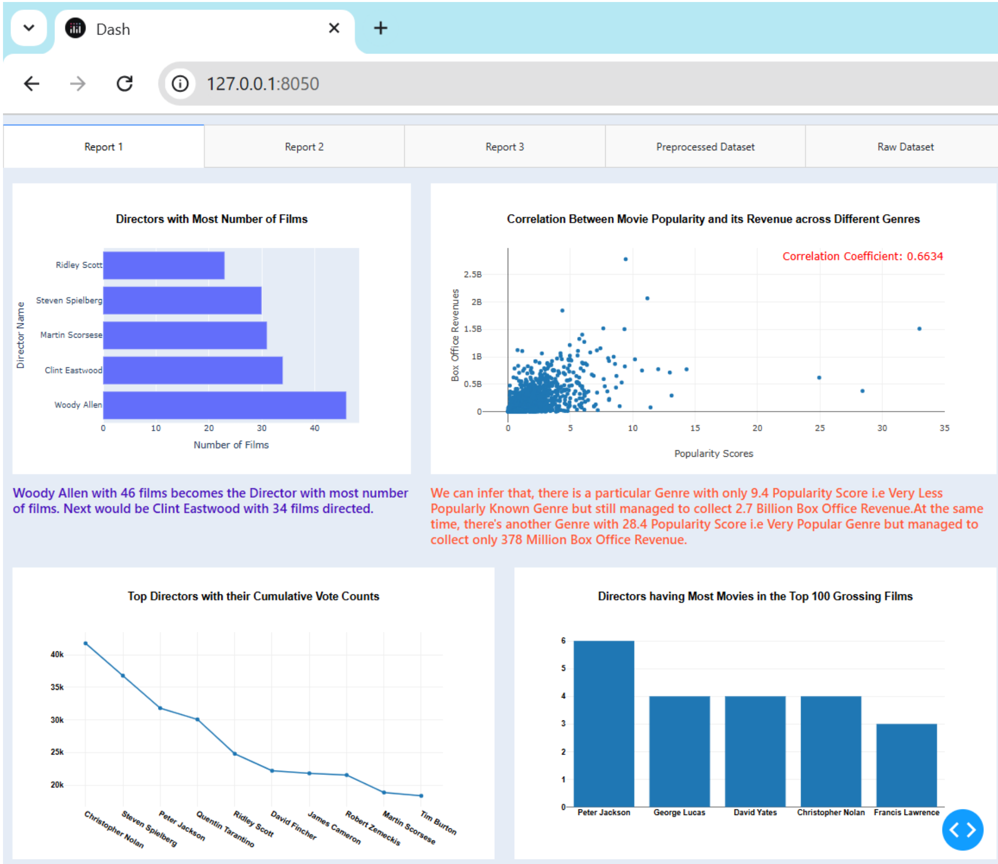
[Source Code] available in Git Repository with clear instructions about local execution.



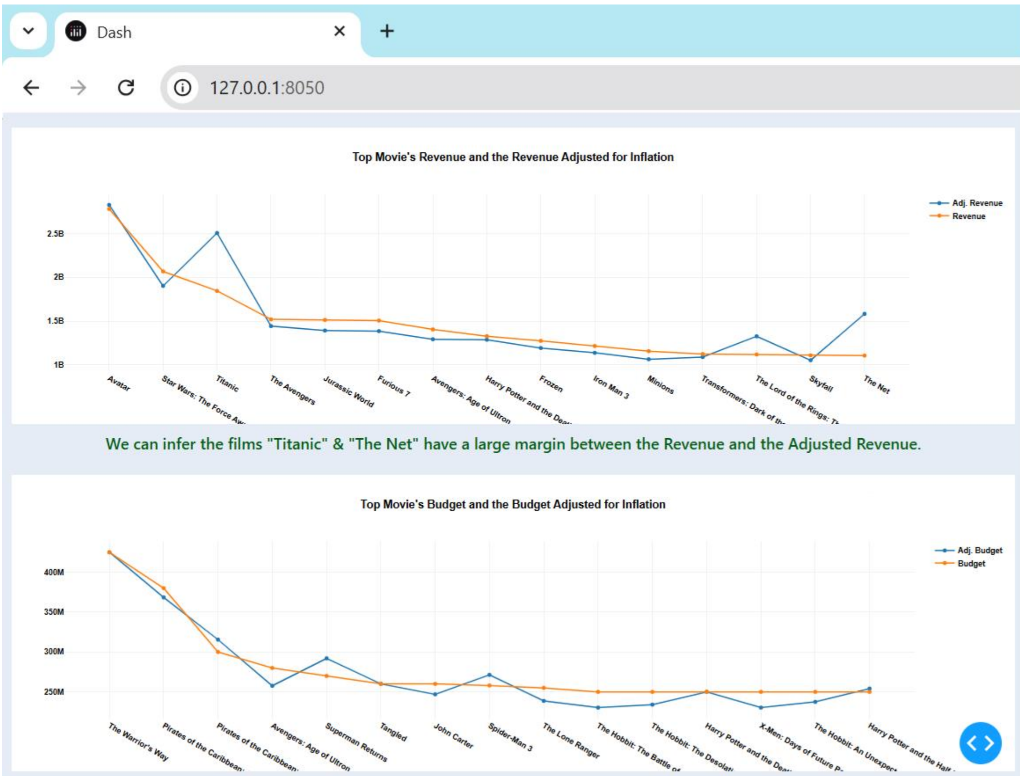
Login Page to ensure User Authentication before viewing the Dashboard



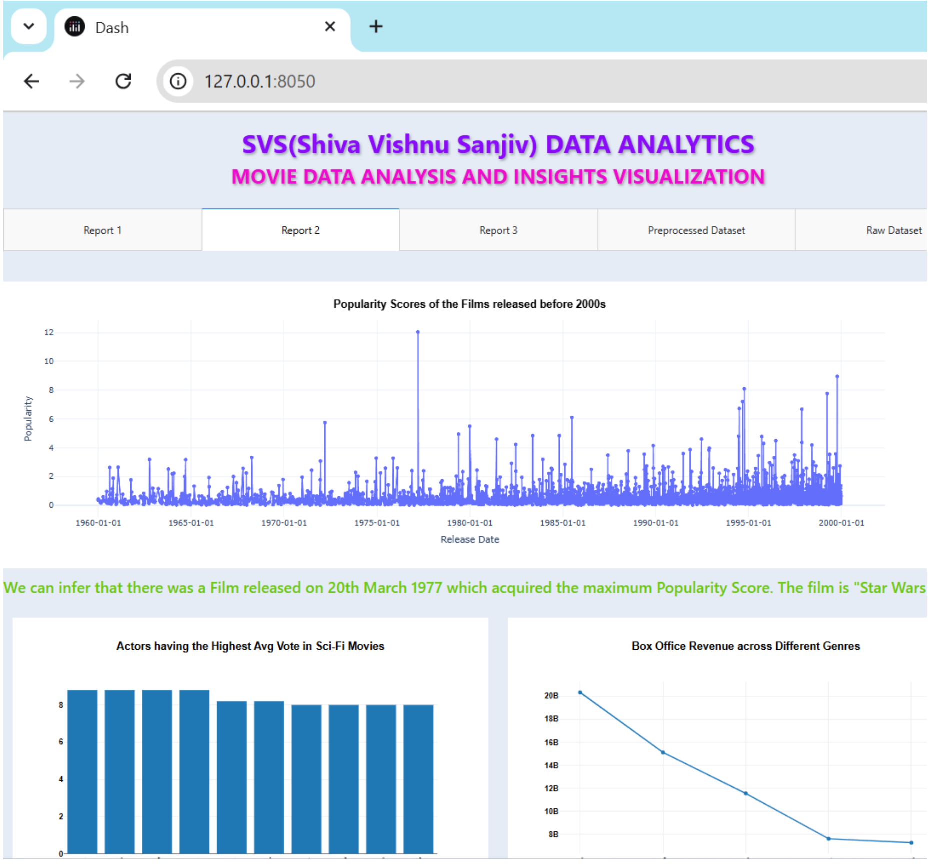
Home Page



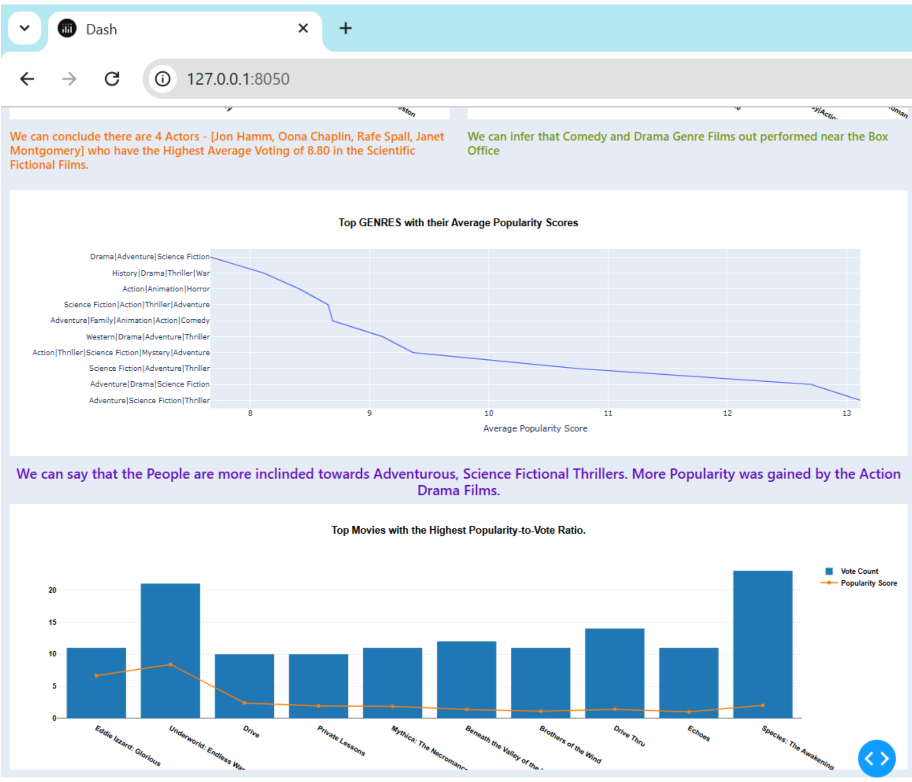
Report 1 depicting multiple plots regarding Directors



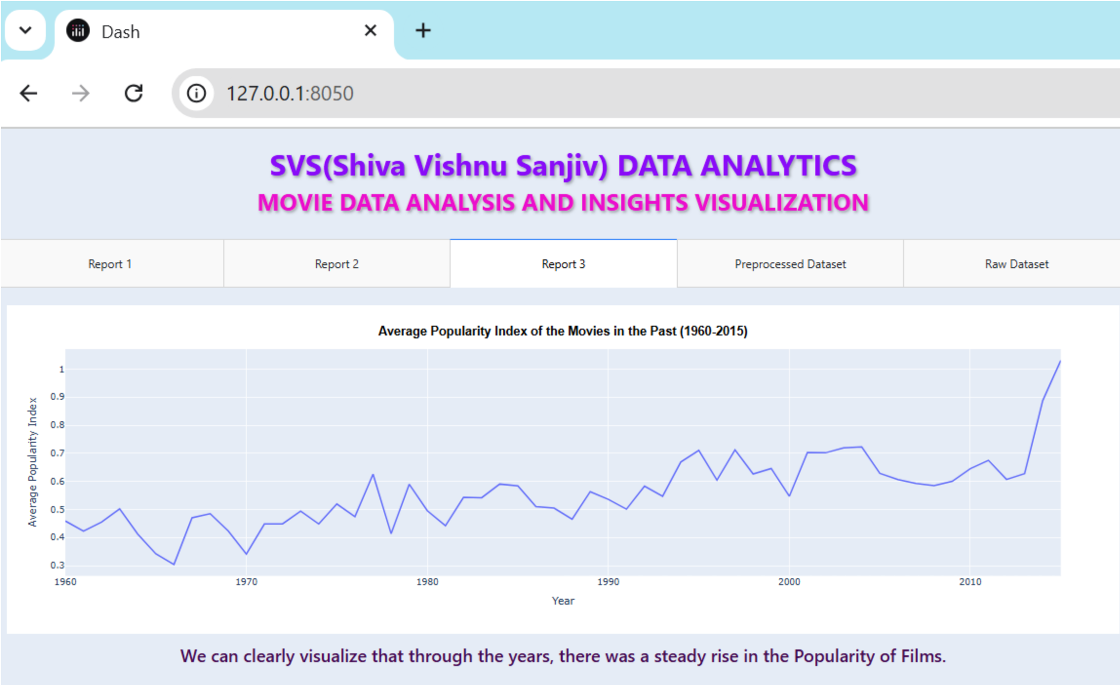
Report 1 depicting Adjusted Financial Plot Insights



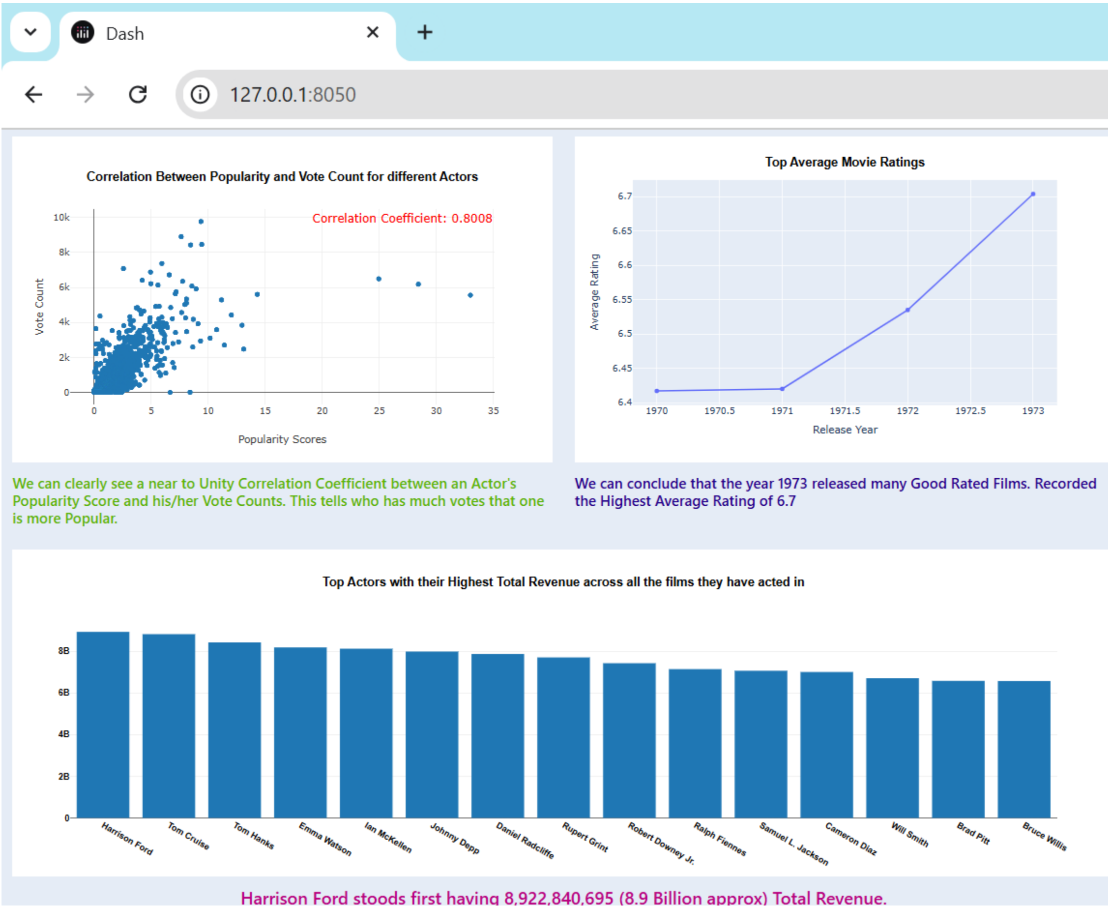
Report 2 depicting other Plot Insights and Inferences



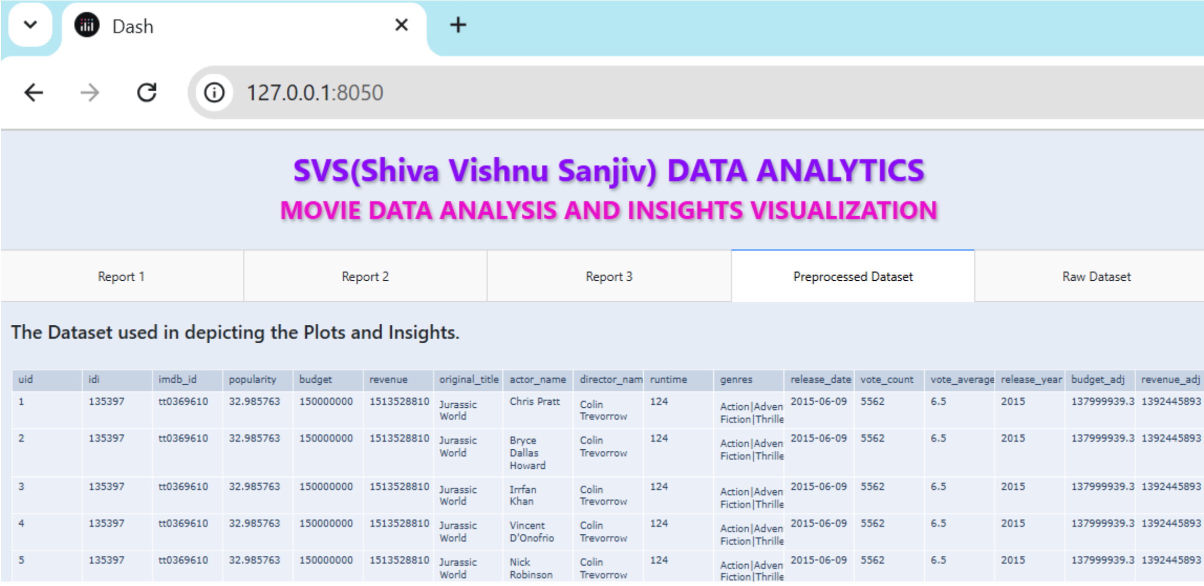
Report 2 depicting the top genres and movies based on Popularity Scores



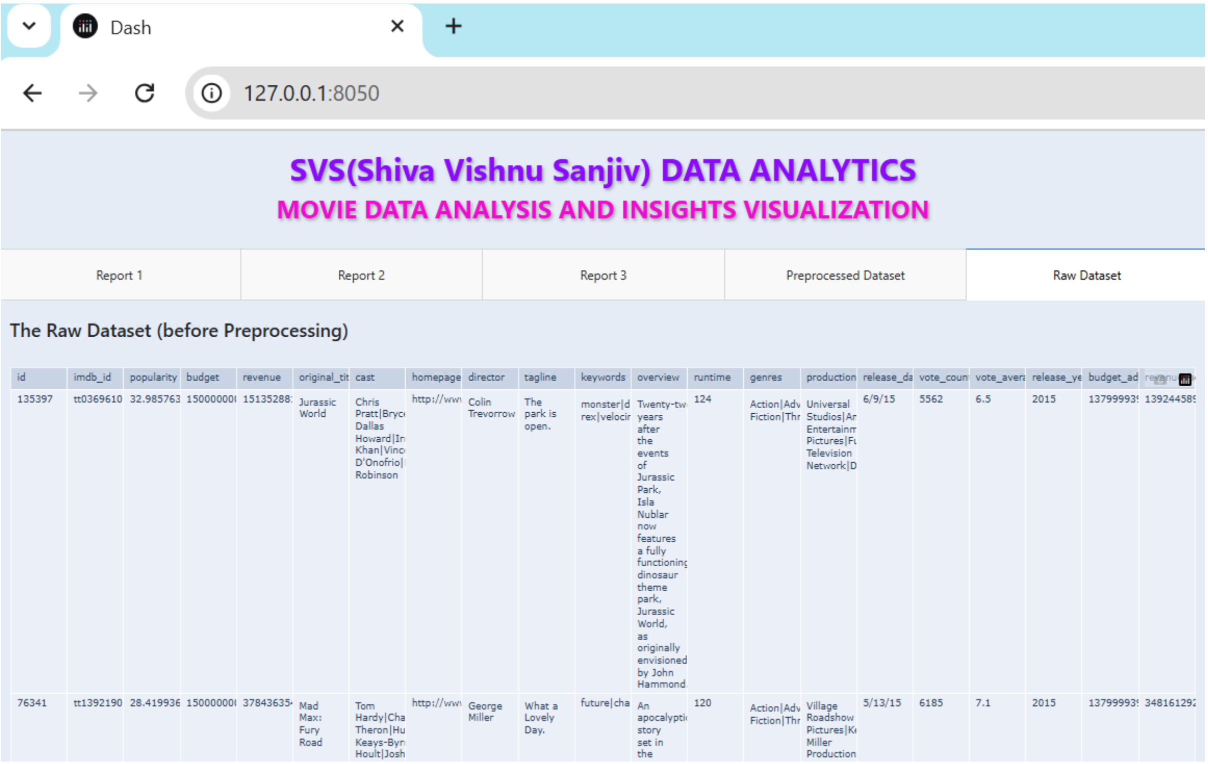
Report 3 depicting the Avg Popularity Index over past decades



Dashboard depicting various plots such as Scatterplot, Bar Chart, Hist

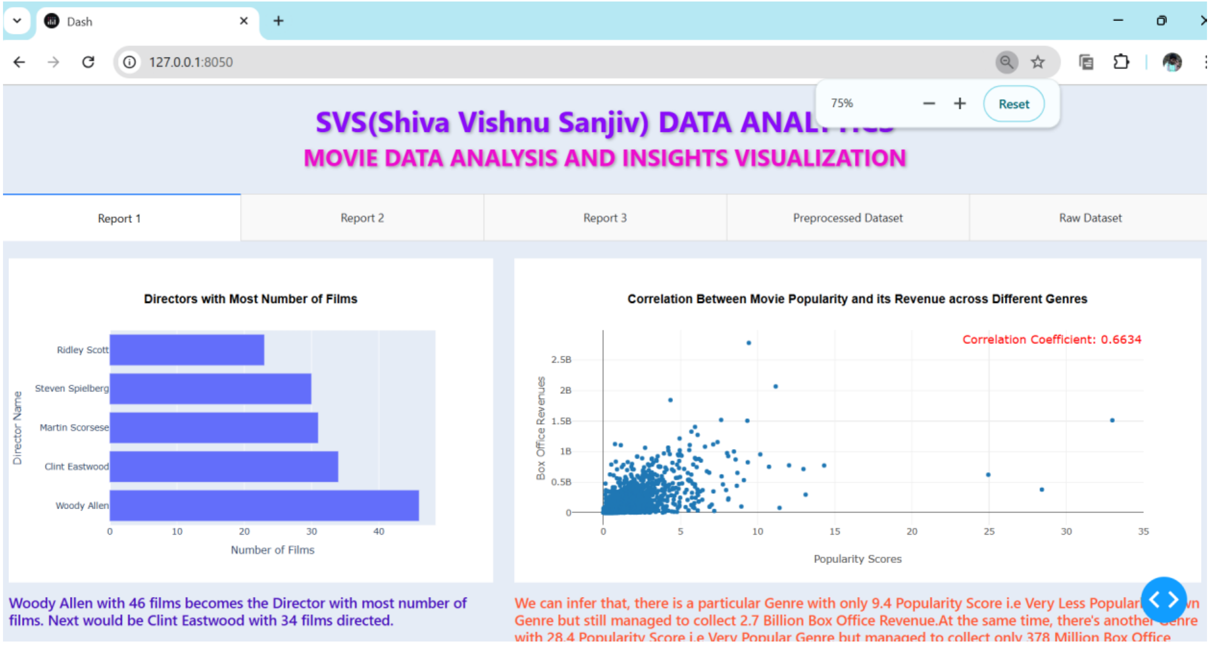


Dashboard depicting what was the Dataset used for querying the insights and plotting them visually

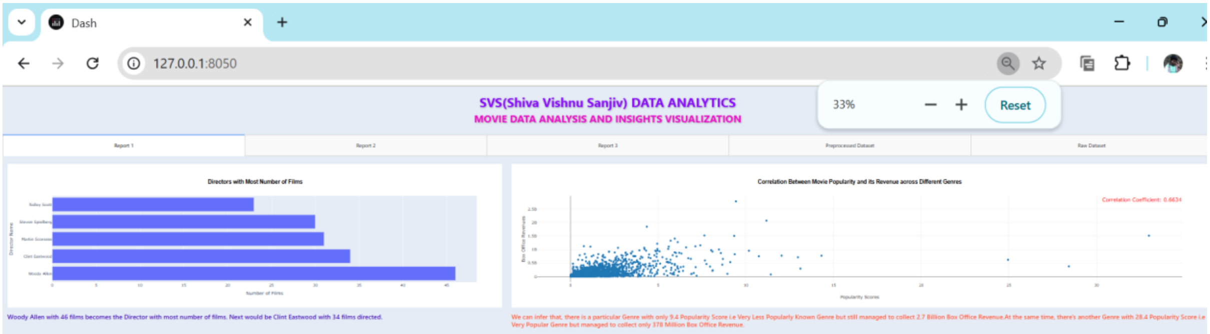


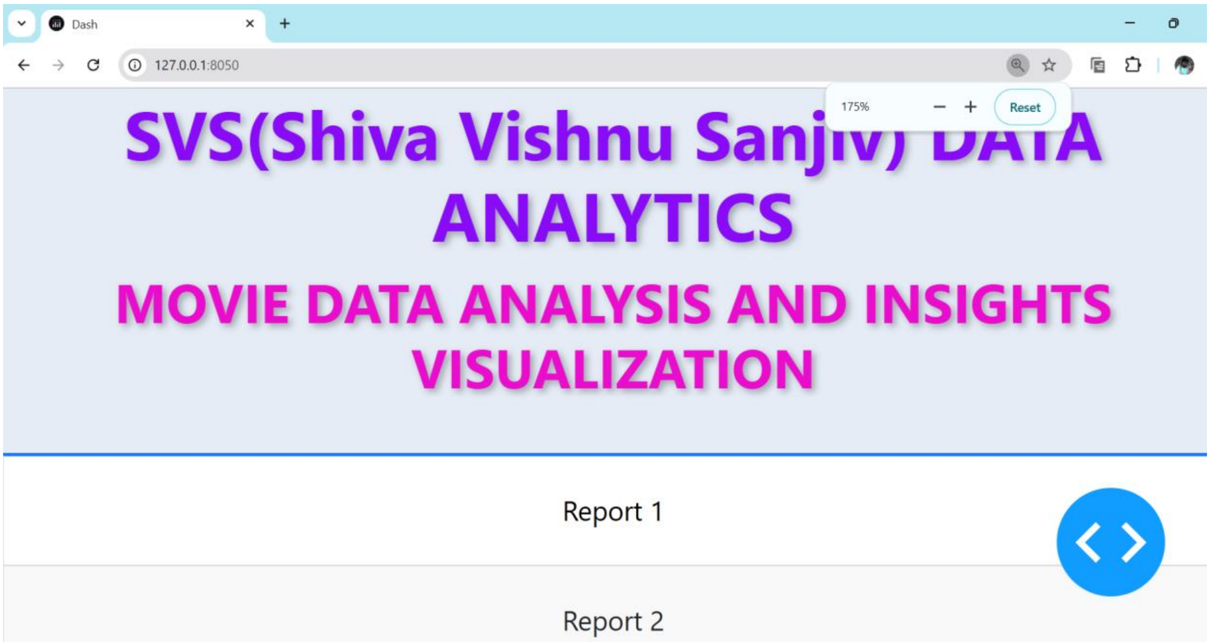
Dashboard also depicts what was the Dataset given as a Problem Statement

Dashboard Responsive UI to the Screen Dimensions



Plots and Inferences aligning dynamically based on screen dimensions





BUSINESS LOGIC

```
[10]: from dash import dcc, html, dash, Input, Output
      import plotly.express as px
      import plotly.graph_objects as go
      import pandas as pd
      import psycopg2
      import sqlalchemy
      from sqlalchemy.engine import create_engine
      from sqlalchemy.sql import text

      #Defining DB Credentials
      USER_NAME = 'postgres'
      PASSWORD = 'padder'
      PORT = 5432
      DATABASE_NAME = 'movies'
      HOST = 'localhost'
```

```
[11]: class PostgresqlDB:
          def __init__(self,user_name,password,host,port,db_name):
              self.user_name = user_name
              self.password = password
              self.host = host
              self.port = port
              self.db_name = db_name
              self.engine = self.create_db_engine()

          def create_db_engine(self):
```

Necessary libraries to be imported initially and use SQLAlchemy to connect the backend PostgreSQL database with the frontend Dash.

```
[31]: q111 = text("select corr(popularity,revenue) from movie;")
      res111 = db.execute_dql_commands(q111)
      for i in res111:
          r111 = i.corr

[32]: r111

[32]: 0.6633602814125884
```

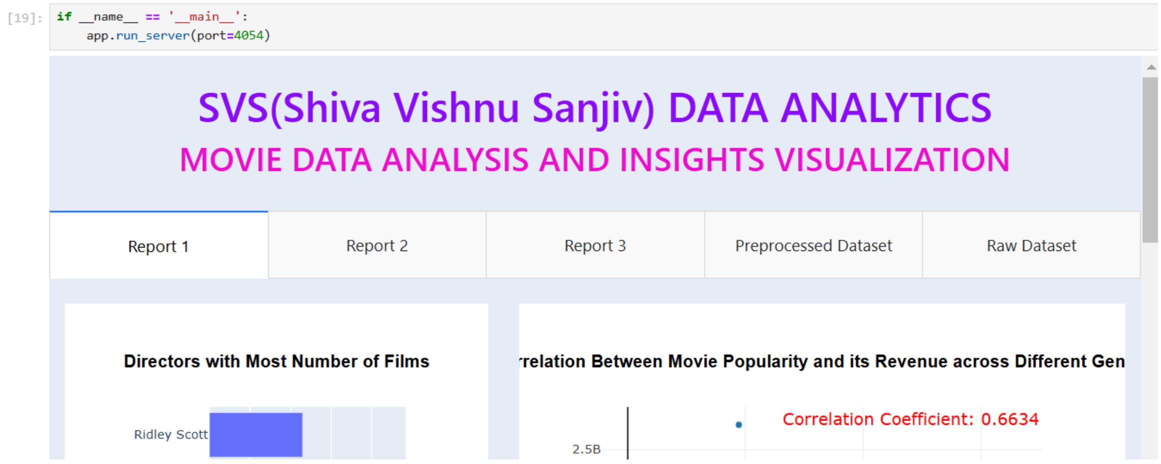
Dynamic Retrieval of Data Insights using SQL Queries.

[18]:
```python
############### DB QURIES FOR TAB 1 #####################
q5 = text('''SELECT DISTINCT(g.genres),m.popularity, m.revenue
FROM movie m
JOIN movie_genres mg ON m.uid = mg.movie_id
JOIN genres g ON mg.genre_id = g.genre_id
WHERE m.revenue > 870000 AND m.popularity IS NOT NULL
ORDER BY m.popularity DESC;''')
res5 = db.execute_dql_commands(q5)
d5 = []
for i in res5:
    d5.append([i.genres,i.popularity,i.revenue])
df5 = pd.DataFrame(d5,columns=['genre_name','popularity_score','box_office_revenue'])
################################
q1 = text('''SELECT d.director_name, COUNT(m.uid) AS movie_count
FROM directors d
INNER JOIN movie_directors md ON d.director_id = md.director_id
INNER JOIN movie m ON md.movie_id = m.uid
WHERE d.director_name NOT LIKE 'No Info about Director'
GROUP BY d.director_name ORDER BY movie_count DESC LIMIT 5;''')
res1 = db.execute_dql_commands(q1)
r1 = []
v1 = []
for i in res1:
    r1.append(i.director_name)
    v1.append(i.movie_count)
```

[24]:
```python
############### PLOTLY FIGURES FOR TAB 1 #####################
def qq1():
    fig1 = go.Figure(go.Bar(
    x=v1,
    y=r1,
    orientation='h'))
    fig1.update_layout(
    title={
        "text": "Directors with Most Number of Films",
        "x": 0.5,
        "xanchor": "center",
        "yanchor": "top",
        "font": {
            "color": "black",
            "family": "Arial",
            "weight": "bold"
        }
    },
    xaxis_title="Number of Films",
    yaxis_title="Director Name",
    autosize=True)
```

```
[33]: ############### PLOTLY FIGURES FOR TAB 2 ####################
t2p1 = dcc.Graph(
        id = 't2p2',
        config={'responsive': True},
        figure = {
            'data' : [
                {'x': v4, 'y':r4,'type':'bar'},
            ],
            'layout':{
                'title': 'Actors having the Highest Avg Vote in Sci-Fi Movies',
                'autosize':True,
                 "font": {
            "color": "black",
            "family": "Arial",
            "weight": "bold"
        }
            }})
t2p2 = dcc.Graph(
        id = 't2p2',
        config={'responsive': True},
        figure = {
            'data' : [
                {'x': r6, 'y':v6,'type':'hist'},
            ],
            'layout':{
                'title': 'Box Office Revenue across Different Genres',
                'autosize':True,
                "font": {
            "color": "black",
            "family": "Arial",
            "weight": "bold"
        }
            }})
```

```
[34]: # DASHBOARD
app.layout = html.Div([
    html.Br(), html.Div([
    html.H1("SVS(Shiva Vishnu Sanjiv) DATA ANALYTICS",style={'color':"#890bf8",
                        "text-align": "center",'font-weight': "bold",'text-shadow': "2px 2px 4px rgba(0, 0, 0, 0.3)"}),
    html.H2("MOVIE DATA ANALYSIS AND INSIGHTS VISUALIZATION",style={'color':"#eb0dcd",
                        "text-align": "center",'font-weight': "bold",'text-shadow': "2px 2px 4px rgba(0, 0, 0, 0.3)"}),
    ]),html.Br(),
    dcc.Tabs(id="tabs", value='tab-1', children=[
        dcc.Tab(label='Report 1', value='tab-1'),
        dcc.Tab(label='Report 2', value='tab-2'),
        dcc.Tab(label='Report 3', value='tab-3'),
        dcc.Tab(label='Preprocessed Dataset', value='tab-4'),
        dcc.Tab(label='Raw Dataset', value='tab-5'),
    ]),
    html.Div(id='tabs-content')],style={"background-color": "#e5ecf6"})
# Callback to render content based on tab selection
@app.callback(Output('tabs-content', 'children'), Input('tabs', 'value'))
def render_content(tab):
    if tab in ['tab-1']:
        return create_tab1_content()
    if tab in ['tab-2']:
        return create_tab2_content()
    if tab in ['tab-3']:
        return create_tab3_content()
    if tab in ['tab-4']:
        return create_tab4_content()
    if tab in ['tab-5']:
        return create_tab5_content()
```



FUTURE WORKS

1) Deploy the Dashboard and run it on a live website.
2) Using Python Scraping Libraries, add few more metadata about the plots and figures.
3) Addition of some more Report Tabs

ACKNOWLEDGMENT

We the team SVS sincerely thank our Course Instructor, Dr Mrinal Kanti Das for giving us the opportunity to implement the learning outcomes of the course in a practical manner. Accomplishing the project, enhanced our skills and expertise on Data Engineering. Learnt various efficient code techniques and mechanisms. By this project, gained practical knowledge about Data Analysis and Database Querying.

****