**COMP30770 Programming for Big Data**

**Extracting Trends in Movie Metadata**

**Project Report**

**Group Members**

|  |  |
| --- | --- |
| Daniel Fallon | 22393263 |
| Colin Wang | 22343536 |
| Harry McCormack | 22513539 |

**Code Link:** <https://github.com/Daniel7Fallon/BigDataProject>

**Section 1**

**The Movies Dataset**

This dataset integrates 45,000 movies from TMDB (metadata, cast/crew, keywords) with 26 million user ratings from GroupLens. It includes structured CSVs (e.g., movies\_metadata.csv, ratings.csv) and semi-structured JSON-embedded files (e.g., credits.csv, keywords.csv), enabling cross-domain analysis like recommendation systems, box office prediction, and genre/director impact studies.

Source: <https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset?resource=download>

**Volume:**

Size: 943.76 MB

Columns: 43

System Specs:

Some execution times from section 3.

**Variety:**

Dataset contains 7 files, ordered by descending size:

|  |  |
| --- | --- |
| ratings.csv | 692,921 KB |
| credits.csv | 185,467 KB |
| movies\_metadata.csv | 33,638 KB |
| keywords.csv | 6,086 KB |
| ratings\_small.csv | 2,382 KB |
| links.csv | 966 KB |
| links\_small.csv | 180 KB |

The dataset’s variety stems from structural and functional differences across files. Structured data like *ratings.csv* (flat numerical columns) contrasts sharply with semi-structured files like *credits.csv*, where nested JSON fields describe actors’ roles and crew departments.

These files can be combined to extract correlations between themes, cast, revenue and public reception. This is invaluable information to the film industry.

**Section 2**

**Objective:**

This project aims to analyse the resource requirements to extract trends and key insights into the film industry and film marketplace.

**Value:**

Companies within the film industry can use these insights to consult and guide them in their decisions of what projects to pursue.

Specifically, the application of Big Data processing is essential to this application as:

* Markets can be volatile, i.e. subject to sudden change.
* Movies are released to the global market at a staggering rate, multiple per day.
* Reviews of movies are written in their hundreds everyday. Websites like “Letterboxd”, a rapidly growing social media for reviewing films, is an example of how this practice is only becoming more common.

These three factors combined means that for the insights garnered to be relevant, this massive and growing dataset must be continually reprocessed. The quicker the turn over of the processing, the more relevant the insights.

**Section 3. Traditional Solution (2.5 pages)**

Language Used: Python 3.13.2

Steps:

1. Read in movies\_metadata.csv file to pandas dataframe for convenient manipulation and perform necessary cleaning:

metadataDF = pd.read\_csv('TheMoviesDataset/movies\_metadata.csv', usecols=["id", "title", "budget", "genres", "popularity", "release\_date", "revenue", "runtime", "vote\_average", "vote\_count"])

metadataDF['id'] = metadataDF['id'].astype('int64')

**0.7s**

2. Convert JSON entries to lists of elements and skip empty entries:

df\_copy['genres'] = df\_copy['genres'].apply(lambda x: ast.literal\_eval(x) if pd.notna(x) else [])

**0.6s**

3. Count genre occurrences:

genre\_counts = Counter(all\_genres)

genre\_counts\_df = pd.DataFrame(genre\_counts.items(), columns=['Genre', 'Count'])

**0.0s**

|  |  |  |
| --- | --- | --- |
|  | Genre | Count |
| 6 | Drama | 20265 |
| 1 | Comedy | 13182 |
| 9 | Thriller | 7624 |
| 5 | Romance | 6735 |
| 7 | Action | 6596 |
| 10 | Horror | 4673 |

4. Show mean vote\_average and mean popularity per genre:

genre\_stats = df\_exploded\_genres.groupby('genre\_name').agg({

    'vote\_average': 'mean', 'popularity': 'mean'}).reset\_index()

**0.6s**

|  |  |  |
| --- | --- | --- |
|  | genre\_name | vote\_average |
| 2 | Animation | 6.275556 |
| 10 | History | 6.154220 |
| 18 | War | 6.041119 |
| 6 | Drama | 5.905226 |
| 12 | Music | 5.879599 |
|  | genre\_name | popularity |
| 1 | Adventure | 5.998335 |
| 8 | Fantasy | 5.363656 |
| 15 | Science Fiction | 4.997888 |
| 0 | Action | 4.770506 |
| 7 | Family | 4.729328 |

5. Read in credits.csv file to pandas dataframe for cast information:

creditsDF = pd.read\_csv('TheMoviesDataset/credits.csv')

**1.0s**

6. Join credits and metadata information:

metadata\_join\_credits\_DF = metadataDF.merge(creditsDF, on='id', how='inner')

**0.0s**

7. Show the correlation between an actor and how well their movies are received:

cast\_stats = df\_exploded\_cast.groupby('cast\_name').agg({

    'vote\_average': 'mean',

    'popularity': 'mean',

    'id': 'count' #Count of films they have been in

}).reset\_index()

**9.3s**

|  |  |  |  |
| --- | --- | --- | --- |
|  | cast\_name | num\_films | vote\_average |
| 76731 | Idris Ali | 1 | 10.0 |
| 52524 | Elif Baysal | 1 | 10.0 |
| 2899 | Al Mackenzie | 1 | 10.0 |
| 34789 | Chuck Blackwell | 1 | 10.0 |
| 147557 | Pamela Craig | 1 | 10.0 |
| … | … | … | … |
| 113019 | Leo Bruckmann | 1 | 0.0 |
| 12586 | Anni Timm | 1 | 0.0 |
| 181395 | Sven Jerring | 1 | 0.0 |
| 146781 | Oscar Tengström | 1 | 0.0 |
| 24 | Rosenda Schaschmidt | 1 | 0.0 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | cast\_name | num\_films | popularity |
| 4726 | Alex Dowding | 1 | 547.488298 |
| 173537 | Shaun Newnham | 1 | 294.337037 |
| 60652 | Frank Allen Forbes | 1 | 294.337037 |
| 57188 | Eva Dabrowski | 1 | 294.337037 |
| 179026 | Steve Doyle | 1 | 294.337037 |

\*The extremes are populated with actors who have only been in one film, that being incredibly successful or entirely unseen.

8. Show correlation between how many movies an actor has been in and the reception of those movies:

numFilmsCorrelation = cast\_stats.groupby('num\_films').agg({

    'vote\_average': 'mean',

    'popularity': 'mean'

}).reset\_index()

**0.0s**

|  |  |  |  |
| --- | --- | --- | --- |
|  | num\_films | vote\_average | popularity |
| 0 | 1 | 5.880034 | 5.169291 |
| 1 | 2 | 5.873058 | 5.313597 |
| 2 | 3 | 5.885797 | 5.195781 |
| 3 | 4 | 5.859514 | 4.925731 |
| 4 | 5 | 5.843277 | 4.742350 |
| … | … | … | … |
| 103 | 110 | 6.102273 | 5.975370 |
| 104 | 123 | 6.211382 | 11.706544 |
| 105 | 125 | 5.666400 | 3.092939 |
| 106 | 148 | 5.870270 | 4.749606 |
| 107 | 241 | 5.539004 | 2.028024 |

**Section 4. MapReduce Optimisation (2 pages)**

Please identify 1 or 2 most time-consuming steps in your Section 3 that can be optimised by big data programming paradigms: MapReduce. You are free to use either Hadoop MapReduce or Spark MapReduce (Spark Core API, NOT Spark SQL or Dataframe etc.)

* Explain why they can be optimised using MapReduce and present your expectations (e.g., reduce execution time by 2). (3’)
* Present MapReduce solution (3’) - Present MapReduce results. (3’) - Explain why the results match or deviate from your expectations. (3’ )