Movies Dataset

IUM • Group: Andrei Luian Fistos

Introduction

This report documents the data engineering and visualization work carried out on a multi-country movies dataset. We focus on (i) cleaning movies and genres datasets, (ii) aggregating genres on movies for ratings calculation, (iii) ratings over time, (iv) genre specific ratings over time, (v) director based ratings, (vi) building an interactive globe visualization, (vii) customizing hover tooltips and selective filtering (e.g., excluding the USA), and (viii) computing average ratings per country by merging movie metadata. The goal is to analyze the basis of rating values, their change over time and the reasons that influence them.

Technical Task 1 — Country normalization

Solution

We standardized free-text country names into ISO Alpha-3 codes using `pycountry` with a manual fallback dictionary for hard cases (e.g., "UK"→"GBR", "USA"→"USA", "South Korea"→"KOR", "Hong Kong"→"HKG"). This yielded a new column `iso_alpha` for reliable joins and plotting.

Key code sketch

```
def get_iso_alpha3(country_name):
    manual_map = {
        "UK": "GBR",
        "USA": "USA",
        "South Korea": "KOR",
        "Hong Kong": "HKG"
    }
    if country_name in manual_map:
        return manual_map[country_name]
    try:
        #automatic standardizing
        return pycountry.countries.lookup(country_name).alpha_3
    except:
        return None
```

Design & Motivations

- **Why ISO-3?** Plot libraries and GIS assets align more robustly on ISO codes than on free-text names (which vary by spelling/aliases).
- **Hybrid resolver:** Automatic lookup + small manual map minimizes manual curation while keeping control for edge cases.

Issues

- Historical entities (e.g., *USSR*, *Czechoslovakia*) or territories may not resolve directly.
- Some names in the raw data can be ambiguous or non-sovereign (e.g., *Hong Kong* vs *China*).

Requirements

- Produces a deterministic `iso_alpha` for each recognized `country` value.
- Leaves non-resolvable cases as `None` so they can be audited.

Limitations

- Requires maintaining a small manual dictionary for persistent edge cases.
- Historical codes (e.g., **UN M.49**, custom shapes) would be needed for time-aware maps.

Technical Task 2 — Count Movies per Country & Build Interactive Globe

Solution

1. **Aggregate counts**: `country_counts = df.groupby("country").size().reset_index(name="movie_count")`.

2. **Attach ISO codes**: map `country_counts.country → iso_alpha`.

3. **Plot**: Interactive globe choropleth with Plotly (`projection="orthographic"`).

Key code sketch

```
filtered_counts = country_counts[country_counts["country"] != "USA"]

fig = px.choropleth(
   filtered_counts,
   locations="Country",
   color="Movie count",
   hover_name="country",
   color_continuous_scale="Viridis",
   projection="orthographic",
   title="Number of Movies Produced per Country"
)

fig.show()
import plotly.io as pio
pio.renderers.default = "notebook"
```

Design & Motivations

- **Orthographic projection** communicates a globe; users can rotate to explore regions.
- **Hover customization** hides codes and surfaces human-readable names and counts.

Issues

- Renderer configuration in Jupyter may block display; we set `plotly.io.renderers.default` to a notebook-friendly backend.

Requirements

- The map must reflect **all countries present** in data and color by their **movie frequencies**.

Limitations

- Choropleths with very skewed distributions can hide small countries; consider log scales or tooltips.

Conclusions

We established a reproducible pipeline to standardize countries, aggregate movie counts, build interactive globe visualizations, customize tooltips, filter subsets (e.g., excluding the

USA), and compute per-country average ratings through a join of movies and countries. The pattern—**normalize** → **aggregate** → **visualize** → **iterate**—proved effective and extensible.

Division of Work *(MANDATORY)*

50/50, the work has been split in half and sometimes we have been working together even if commits were pushed from a certain account.

Extra Information *(MANDATORY)*

Environment & Dependencies

- Python ≥ 3.9
- `pandas`, `pycountry`, `plotly` (for Jupyter: ensure the proper Plotly renderer — e.g., `notebook` or `notebook_connected`).

Running the notebook

1. Install dependencies: `pip install pandas pycountry plotly`.

Configuration

- No additional configuration required beyond installing the packages above.

Bibliography

[1] Plotly Express Choropleth Maps — Official Documentation.
[2] `pycountry` — ISO country definitions for Python.
[3] Pandas User Guide — GroupBy and Aggregations.