Netflix Insights and Metrics

Week 1&2:

Netflix Insights

1. **Content Distribution**
   - The dataset comprises **~8,800 titles**, a combination of both **Movies (~70%)** and **TV Shows (~30%)**.
   - Movies dominate Netflix's catalog, but TV Shows have been increasing in recent years, signaling Netflix's shift towards episodic content.
2. **Temporal Trends**
   - Titles span multiple decades, with older classics alongside recent Netflix Originals.
   - A sharp rise in content is observed post-2015, aligning with Netflix's global expansion strategy.
3. **Genre Representation**
   - A wide variety of genres exist.
   - **Top genres:** Dramas, Documentaries, Comedies.
   - **Emerging genres:** International TV, Stand-up Comedy, and Romantic TV Shows — reflecting user demand.
4. **Geographical Spread**
   - Content originates from over 100 countries, showcasing Netflix's global production and licensing reach.
   - Major contributors**:** United States, India, United Kingdom, Japan, South Korea.
5. **Rating Distribution**
   - Titles are spread across maturity ratings (TV-MA, R, PG-13, TV-14, etc.).
   - A strong presence of mature-rated content (TV-MA, R) indicates a focus on adult audiences, but family-friendly segments (TV-Y, PG) are also well represented.
6. **Missing Data Observations**
   - Director and cast columns had significant missing values, likely due to incomplete metadata.
   - Rating and Duration had gaps, which were filled systematically for consistency.

Netflix Metrics/Scope

1. **Trend Analysis**
   - Evaluate the evolution of Movies vs. TV Shows, genres, and ratings over years.
   - Guide Netflix in shaping its content acquisition and production strategies.
2. **Genre Popularity & Recommendations**
   - Identify top genres globally and regionally.
   - Enable personalized recommendations based on user preferences.

3. **Geographical Expansion Strategy**
    - Assess country-level contributions to Netflix's catalog.
    - Support regional expansion and localized content production.
4. **Content Duration Insights**
    - Distinguish average movie length vs. average TV Show seasons.
    - Inform viewer engagement and content planning.
5. **Data Quality Improvement**
    - Enhance metadata completeness for directors and casts.
    - Support enriched recommendation systems and talent-based content analysis.

Dataset Loading

The Netflix dataset is sourced from Kaggle and loaded into the workspace for preprocessing and analysis.

- **Dataset Source:** Kaggle — Netflix Movies and TV Shows Dataset.
- **Dataset Size:** ~8,800 titles across multiple years and genres.
- **Key Columns:** type, title, director, cast, country, release_year, rating, duration, listed_in, date_added.

Loading the dataset using pandas:

```
import pandas as pd

df_read = pd.read_csv("/Volumes/workspace/default/netflix/netflix_titles.csv")

display(df_read.head())
```

output:

show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
s1	Movie	Example Movie Title	John Doe	Actor A, Actor B	United States	2020-01-01	2019	PG-13	90 min	Dramas, Comedies	A short description of the movie.
s2	TV Show	Example Show Title	Jane Smith	Actor C, Actor D	India	2019-06-10	2018	TV-MA	2 Seasons	TV Dramas, International TV Shows	A short description of the show.

The dataset provides a rich set of features that allow for multi-dimensional analysis of Netflix's content strategy, such as genre diversity, rating distributions, and country-wise availability.

Data Cleaning Steps Using Pandas

Step 1 — Null Handling/Handling missing Values

Purpose:
Missing values in a dataset can distort analysis and predictions. We handle missing data to make the dataset complete and reliable.

Actions Taken:

- Dropped rows missing date_added values (important for temporal analysis).
- Filled missing director and cast values with "Not Available" to keep information consistent.
- Filled missing country with "Unknown" to represent missing geographical data.
- Filled missing duration with "0" to keep numeric processing consistent.
- Filled missing rating values with the **most common rating** (mode) to maintain category balance.

Code:

```
df_read['director'] = df_read['director'].fillna("Not Available")

df_read['cast'] = df_read['cast'].fillna("Not Available")

df_read['country'] = df_read['country'].fillna("Unknown")

df_read['duration'] = df_read['duration'].fillna("0")

mode_rating = df_read['rating'].mode()[0]

df_read['rating'] = df_read['rating'].fillna(mode_rating)
```

Example Output — Missing Values Check:

Missing values per column:

```
show_id      0

type         0

title        0

director     0

cast         0
```

Step 2 — Remove Duplicates

Purpose:
Duplicate rows can bias analysis. Removing them ensures the dataset represents unique entries only.

ActionTaken:
Removed exact duplicate rows from the dataset.

Code:

```
df_cleaned = df_read.drop_duplicates()

print(df_cleaned.shape)
```

Example Output:

```
(7787, 12)
```

(Original dataset size: 8000 rows → After cleaning: 7787 rows)

Step 3 — Whitespace Cleaning

Purpose:
Extra spaces in categorical columns can cause incorrect grouping and encoding. Cleaning spaces ensures consistency.

ActionsTaken:
Trimmed spaces from type and rating columns.

Code:

```
df_cleaned['type'] = df_cleaned['type'].str.strip()

df_cleaned['rating'] = df_cleaned['rating'].str.strip()
```

Example-Output:
Before cleaning: " Movie ", " PG-13 " → After cleaning: "Movie", "PG-13".

Step 4 — Duration Extraction

Purpose:
The duration column contains both numeric and text values (e.g., “90 min”, “2 Seasons”). Splitting them allows quantitative analysis of durations.

Actions Taken:

- Extracted numeric part into duration_num.
- Extracted duration type (min, Season, Seasons) into duration_type.

Code:

```
df_read['duration_num'] = df_read['duration'].str.extract(r'(\d+)').astype(float)

df_cleaned['duration_type'] = df_cleaned['duration'].str.extract(r'(min|Season|Seasons)')
```

Example Output:

duration	duration_num	duration_type
90 min	90.0	min
2 Seasons	2.0	Seasons
1 Season	1.0	Season

Step 5 — Date Conversion

Purpose:
The date_added column must be in datetime format for time-series analysis.

ActionTaken:
Converted date_added to datetime, handling errors.

Code:

```
df_read['date_added'] = pd.to_datetime(df_read['date_added'], errors='coerce')
```

Example Output:

```
Earliest date added: 2008-01-01

Latest date added: 2023-07-15
```

Normalization Process

Purpose:
Normalization is a critical preprocessing step that transforms raw data into a consistent and machine-readable format. It ensures features are scaled, encoded, and structured for analysis and modeling, improving algorithm performance and enabling meaningful comparisons.

In this project, normalization involved scaling numeric values, encoding categorical features, and extracting date-based features for deeper analysis of Netflix content.

Step 1 — Min-Max Scaling for Duration

Code:

```
df_norm['duration_num'] = pd.to_numeric(

    df_norm['duration'].str.extract(r'(\d+)')[0], errors='coerce'

).fillna(0)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df_norm['duration_norm'] = scaler.fit_transform(df_norm[['duration_num']])
```

Example Output:

duration	**duration_num**	**duration_norm**
90 min	90.0	0.237
2 Seasons	2.0	0.005
1 Season	1.0	0.002

Explanation:
The numeric portion of the duration column was extracted and scaled between 0 and 1 using Min-Max scaling. This normalization ensures duration is comparable across content types.

Step 2 — One-Hot Encoding for Type

Code:

```
df_type_onehot = pd.get_dummies(df_norm['type'], prefix='type').astype('int32')

df_norm = pd.concat([df_norm, df_type_onehot], axis=1)
```

Example Output:

type_Movie	type_TV Show
1	0
0	1
1	0

Explanation:
One-hot encoding converts the type feature into binary columns representing Movies or TV Shows, allowing models to process them without implied ordering.

Step 3 — Frequency Encoding for Country

Code:

```
country_freq = df_norm['country'].value_counts().to_dict()

df_norm['country_freq'] = df_norm['country'].map(country_freq)
```

Example Output:

country	country_freq
United States	3000
India	500
United Kingdom	200

Explanation:
Frequency encoding assigns a numeric value based on the occurrence frequency of each country, capturing its importance in the dataset.

Step 4 — Ordinal Encoding for Rating

Code:

```
valid_ratings = ['G','PG','PG-13','R','NC-17','TV-Y','TV-Y7','TV-G','TV-PG','TV-14','TV-MA','Not Rated']

df_norm = df_norm[df_norm['rating'].isin(valid_ratings)]

df_norm['rating'] = df_norm['rating'].replace(['NR','UR','Not Rated','TV-Y7-FV'],'Not Rated')

rating_order = [['G','PG','PG-13','R','NC-17','TV-Y','TV-Y7','TV-G','TV-PG','TV-14','TV-MA','Not Rated']]

encoder = OrdinalEncoder(categories=rating_order)

df_norm['rating_ord'] = encoder.fit_transform(df_norm[['rating']])
```

Example Output:

rating	rating_ord
PG	1.0
TV-MA	10.0
G	0.0

Explanation:
Ordinal encoding assigns numerical values to ratings based on a defined order, allowing models to interpret relative content suitability levels.

Step 5 — Label Encoding for Categorical Features

Code:

```
label_cols = ['rating','country','director','cast']

from sklearn.preprocessing import LabelEncoder

for col in label_cols:

    le = LabelEncoder()

    df_norm[col + '_label'] = le.fit_transform(df_norm[col].astype(str))
```

Example Output:

rating_label	country_label	director_label	cast_label
1	0	1050	2340
10	3	2045	1001

Explanation:
Label encoding transforms categorical variables into numerical codes, allowing machine learning algorithms to process them efficiently.

Step 6 — Genre Encoding

Code:

```
df_norm['primary_genre'] = df_norm['listed_in'].str.split(',').str[0]

df_norm['primary_genre_label'] = LabelEncoder().fit_transform(df_norm['primary_genre'].astype(str))

df_norm['genre_1'] = df_norm['listed_in'].str.split(',').str[0]

df_norm['genre_2'] = df_norm['listed_in'].str.split(',').str[1]

df_genre_onehot = pd.get_dummies(df_norm[['genre_1','genre_2']].fillna(''), prefix=['genre1','genre2']).astype('int32')

df_norm = pd.concat([df_norm, df_genre_onehot], axis=1)
```

Example Output:

genre1_Action	genre2_Comedies
1	0
0	1

Explanation:
Genres are extracted and encoded to reduce dimensionality while retaining important categorical information.

Step 7 — Date Feature Extraction

Code:

```
df_norm['date_added'] = pd.to_datetime(df_norm['date_added'], errors='coerce')

df_norm['year_added'] = df_norm['date_added'].dt.year

df_norm['month_added'] = df_norm['date_added'].dt.month

df_norm['day_added'] = df_norm['date_added'].dt.day

df_norm['dayofweek_added'] = df_norm['date_added'].dt.dayofweek
```

Example Output:

date_added	year_added	month_added	day_added	dayofweek_added
2020-01-15	2020	1	15	2

Explanation:
Date features are extracted to enable time-based analysis of content additions.

Step 8 — Save Normalized Dataset

Code:

```
df_norm.to_csv("/Volumes/workspace/default/netflix/netflix_normalized.csv", index=False)

print("\n Normalized dataset saved as 'netflix_normalized.csv'")

print(f"Shape of dataset: {df_norm.shape}")
```

Example Output:

```
 Normalized dataset saved as 'netflix_normalized.csv'

Shape of dataset: (7787, 120+ columns)
```