# Netflix Data Analysis — Brief Report with Code

## 1. Executive Summary

A compact analysis of Netflix titles (≈8.8k rows, 10 columns; 2008–2021). We clean and explore the data to understand type mix, ratings, genres, and time trends. We then build a lightweight classifier to predict content type (Movie vs TV Show) using engineered features. Outputs: clear visuals, key insights, and reproducible Python code.

## 2. Introduction

Netflix's catalog reveals strategic shifts (e.g., rise of TV shows post-2016, dominance of mature ratings). This report provides quick, actionable insights and a minimal ML pipeline suitable for extension into recommendations.

## 3. Dataset Overview & Loading

File: netflix_titles.csv. Columns: show_id, type, title, director, country, date_added, release_year, rating, duration, listed_in. Target for ML demo: type.

```
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv("netflix_titles.csv")

# Basic checks
print(df.shape) # (~8790, 10)
print(df.isna().sum()) # missingness overview
print(df.head(3))
```

## 4. EDA & Key Insights

• Type Mix: Movies ≈70%, TV Shows ≈30%. • Ratings: TV-MA and TV-14 dominate. • Time: Additions peaked around 2018–2020. • Genres: Dramas, Comedies, Documentaries lead for movies; International TV Dramas for shows. • Geography: USA, India, UK contribute most.

```
# Content type distribution
type_counts = df['type'].value_counts()
print(type_counts)

# Convert 'date_added' to datetime and derive year/month
df['date_added'] = pd.to_datetime(df['date_added'], errors='coerce')
df['year_added'] = df['date_added'].dt.year
df['month_added'] = df['date_added'].dt.month

# Split genres for simple counting
df['genres'] = df['listed_in'].fillna("").apply(lambda x: [g.strip() for g in
x.split(',')])
from collections import Counter
top_genres = Counter([g for gs in df['genres'] for g in gs if g]).most_common(10)
print(top_genres)
```

## 5. Preprocessing

Steps: drop duplicates; minimal NA handling (e.g., drop rows missing director/country if needed); parse dates; engineer numeric duration; one-hot encode top genres.

```
# Drop duplicates
df = df.drop_duplicates()
```

```
# Handle critical NAs (example policy)
df = df.dropna(subset=['type', 'title'])
df['country'] = df['country'].fillna("Not Given")

# Duration to minutes (for Movies) and seasons (for TV Shows)
def parse_duration(s):
if pd.isna(s): return np.nan
s = str(s).lower()
if "min" in s:
return int(s.split()[0]) # minutes
if "season" in s:
# encode seasons as 60 * seasons for comparability (simple proxy)
return int(s.split()[0]) * 60
return np.nan

df['duration_num'] = df['duration'].apply(parse_duration)

# Top-N genres for one-hot (keep compact)
N = 8
topN = [g for g, _ in Counter([g for gs in df['genres'] for g in gs if g]).most_common(N)]
for g in topN:
df[f"genre_{g.replace(' ','_')[:15]}"] = df['genres'].apply(lambda gs: int(g in gs))

# Minimal feature set
features = ['release_year', 'duration_num', 'year_added'] + [c for c in df.columns if
c.startswith("genre_")]
df_ml = df.dropna(subset=features + ['type']).copy()
```

## 6. Model Building (Movie vs TV Show)

Goal: predict 'type' from compact features. We use a fast baseline (Logistic Regression) and a non-linear model (Random Forest).

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report

# Prepare X/y
X = df_ml[features].fillna(0)
y = (df_ml['type'] == 'TV Show').astype(int) # TV Show=1, Movie=0

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Logistic Regression
scaler = StandardScaler(with_mean=False) # sparse-safe, though X is dense; keeps it simple
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)

lr = LogisticRegression(max_iter=200, n_jobs=None)
lr.fit(X_train_sc, y_train)
y_pred_lr = lr.predict(X_test_sc)

# Random Forest
rf = RandomForestClassifier(n_estimators=300, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

def metrics(y_true, y_pred, label):
print(f"\n[{label}]")
```

```
print("Accuracy :", round(accuracy_score(y_true, y_pred), 4))
print("Precision:", round(precision_score(y_true, y_pred), 4))
print("Recall :", round(recall_score(y_true, y_pred), 4))
print("F1-Score :", round(f1_score(y_true, y_pred), 4))

metrics(y_test, y_pred_lr, "Logistic Regression")
metrics(y_test, y_pred_rf, "Random Forest")
```

## 7. Evaluation (Quick View)

Both models provide strong baseline performance on this dataset. Random Forest usually edges out Logistic Regression due to non-linear patterns in genres and duration. Use stratified split and multiple seeds for robust estimates.

## 8. Key Insights

• Movies dominate overall count; TV Shows surged post-2016. • TV-MA and TV-14 are most frequent ratings. • Duration and genre indicators are strong signals of type. • Top contributing countries: USA, India, UK.

## 9. Minimal Reproducible Script

```
# Run end-to-end in one go (assumes netflix_titles.csv present)
import pandas as pd, numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

df = pd.read_csv("netflix_titles.csv").drop_duplicates()
df['date_added'] = pd.to_datetime(df['date_added'], errors='coerce')
df['year_added'] = df['date_added'].dt.year

df['genres'] = df['listed_in'].fillna("").apply(lambda x: [g.strip() for g in
x.split(',')])
def parse_duration(s):
if pd.isna(s): return np.nan
s = str(s).lower()
if "min" in s: return int(s.split()[0])
if "season" in s: return int(s.split()[0]) * 60
return np.nan
df['duration_num'] = df['duration'].apply(parse_duration)

N = 8
topN = [g for g,_ in Counter([g for gs in df['genres'] for g in gs if g]).most_common(N)]
for g in topN:
df[f"genre_{g.replace(' ','_')[:15]}"] = df['genres'].apply(lambda gs: int(g in gs))

features = ['release_year','duration_num','year_added'] + [c for c in df.columns if
c.startswith("genre_")]
df_ml = df.dropna(subset=features + ['type']).copy()
X = df_ml[features].fillna(0)
y = (df_ml['type'] == 'TV Show').astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

scaler = StandardScaler(with_mean=False)
X_train_sc, X_test_sc = scaler.fit_transform(X_train), scaler.transform(X_test)

lr = LogisticRegression(max_iter=200).fit(X_train_sc, y_train)
rf = RandomForestClassifier(n_estimators=300, random_state=42).fit(X_train, y_train)
```

```
def show_metrics(m, X_tr, X_te, y_te, name):
import numpy as np
y_pred = m.predict(X_te)
print(f"[{name}] acc={accuracy_score(y_te,y_pred):.3f}, "
f"prec={precision_score(y_te,y_pred):.3f}, "
f"rec={recall_score(y_te,y_pred):.3f}, "
f"f1={f1_score(y_te,y_pred):.3f}")
show_metrics(lr, X_train_sc, X_test_sc, y_test, "LogReg")
show_metrics(rf, X_train, X_test, y_test, "RandForest")
```