

```

# Part 1: Data Analysis - Netflix Dataset
# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

# Q1: Data Cleaning and Preprocessing
print("\n--- Missing Values ---")
print(df.isnull().sum())

# Drop rows with missing 'type', 'title', or 'date_added'
df = df.dropna(subset=['type', 'title', 'date_added'])

# Convert 'date_added' to datetime
df['date_added'] = pd.to_datetime(df['date_added'], errors='coerce')

# Extract year and month
df['year_added'] = df['date_added'].dt.year
df['month_added'] = df['date_added'].dt.month_name()

print("\n--- Sample after preprocessing ---")
print("Missing or invalid dates:", df['date_added'].isna().sum())

# Q2: Exploratory Data Analysis (EDA)

# 1. Count of Movies vs TV Shows
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='type', palette='Set2')
plt.title("Count: Movies vs TV Shows")
plt.xlabel("Type")
plt.ylabel("Count")
plt.show()

# 2. Top 5 countries by number of titles
top_countries = df['country'].dropna().str.split(',').explode().str.strip().value_counts().head(5)

plt.figure(figsize=(8,4))
sns.barplot(x=top_countries.index, y=top_countries.values, palette='Blues_d')
plt.title("Top 5 Countries by Netflix Titles")
plt.ylabel("Number of Titles")
plt.xlabel("Country")
plt.show()

# 3. Distribution of content ratings
plt.figure(figsize=(10,4))
sns.countplot(data=df, x='rating', order=df['rating'].value_counts().index, palette='Set3')
plt.title("Distribution of Content Ratings")
plt.xticks(rotation=45)
plt.show()

# Q3: Pattern Discovery and Grouping

# 1. Group by release year
release_counts = df['release_year'].value_counts().sort_index()

plt.figure(figsize=(12,4))
sns.lineplot(x=release_counts.index, y=release_counts.values)
plt.title("Number of Releases Over Years")
plt.xlabel("Release Year")
plt.ylabel("Number of Titles")
plt.grid(True)
plt.show()

# 2. Most popular genres in the U.S.
us_genres = df[df['country'].str.contains("United States", na=False)]
genre_counts = us_genres['listed_in'].str.split(',').explode().str.strip().value_counts().head(10)

plt.figure(figsize=(10,4))
sns.barplot(x=genre_counts.values, y=genre_counts.index, palette='magma')
plt.title("Top 10 Genres in the U.S.")
plt.xlabel("Number of Titles")

```

```
plt.ylabel("Genre")
plt.show()

# 3. Average duration per genre (Movies only)
movie_df = df[df['type'] == 'Movie'].copy()
movie_df['duration'] = movie_df['duration'].str.replace(' min', '', regex=False).astype(float)

genre_duration = movie_df.dropna(subset=['duration']) \
    .assign(genre=movie_df['listed_in'].str.split(',').str[0]) \
    .groupby('genre')['duration'].mean().sort_values(ascending=False).head(10)

plt.figure(figsize=(10,5))
sns.barplot(x=genre_duration.values, y=genre_duration.index, palette='coolwarm')
plt.title("Average Movie Duration by Genre")
plt.xlabel("Average Duration (minutes)")
plt.ylabel("Genre")
plt.show()
```



```

--- Missing Values ---
show_id      0
type         0
title        0
director     2634
cast         825
country      831
date_added   10
release_year  0
rating       4
duration     3
listed_in    0
description  0
dtype: int64

```

```

--- Sample after preprocessing ---

```

```

Missing or invalid dates: 88

```

```

<ipython-input-4-5af6c619b417>:18: SettingWithCopyWarning:

```

```

A value is trying to be set on a copy of a slice from a DataFrame.

```

```

Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a)

```

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

```

```

<ipython-input-4-5af6c619b417>:22: SettingWithCopyWarning:

```

```

A value is trying to be set on a copy of a slice from a DataFrame.

```

```

Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a)

```

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

```

```

<ipython-input-4-5af6c619b417>:23: SettingWithCopyWarning:

```

```

A value is trying to be set on a copy of a slice from a DataFrame.

```

```

Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a)

```

df['month_added'] = df['date_added'].dt.month_name()

```

```

<ipython-input-4-5af6c619b417>:33: FutureWarning:

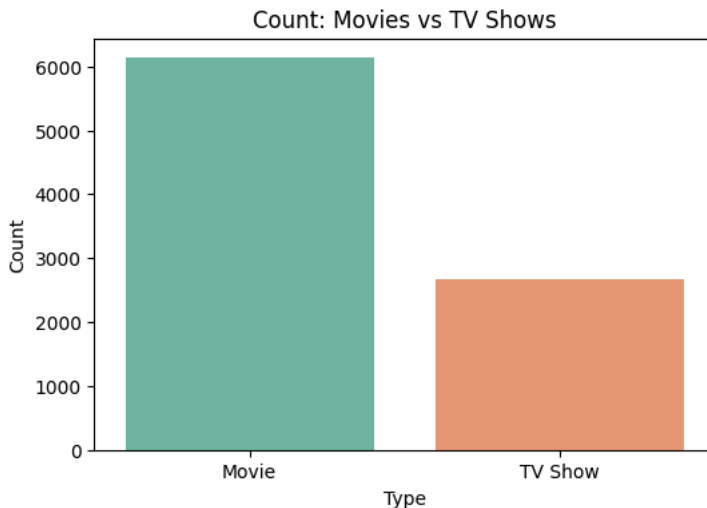
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg`

```

sns.countplot(data=df, x='type', palette='Set2')

```



```

<ipython-input-4-5af6c619b417>:43: FutureWarning:

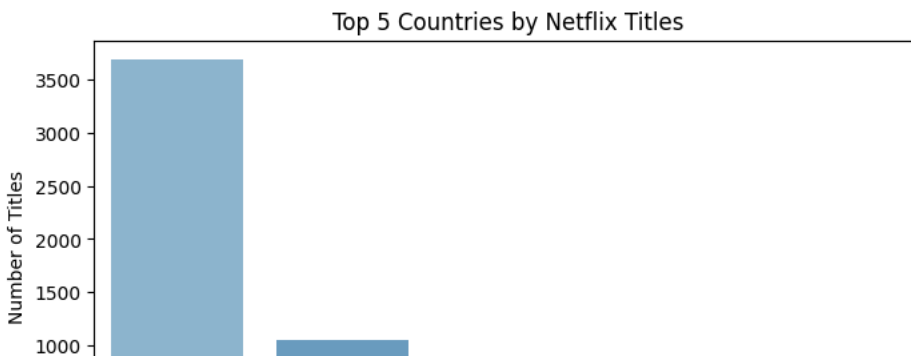
```

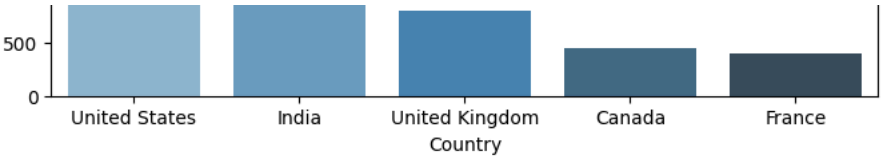
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg`

```

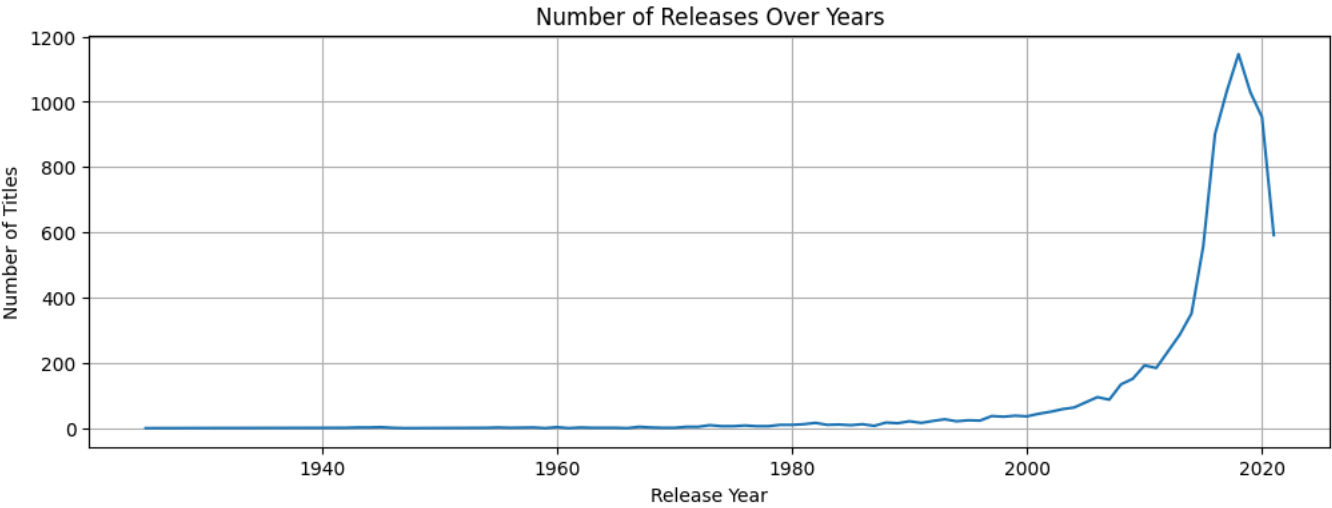
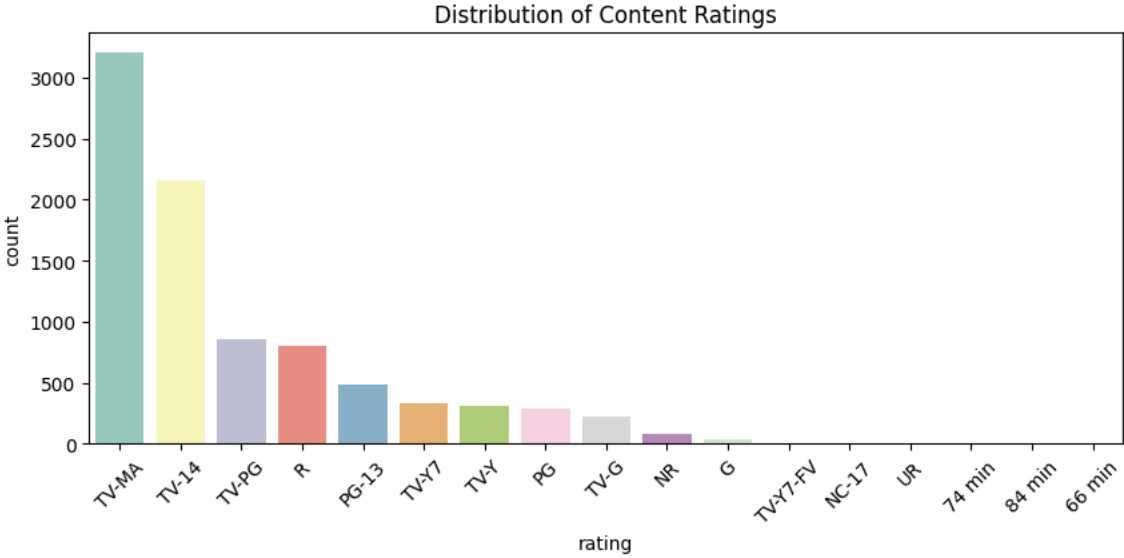
sns.barplot(x=top_countries.index, y=top_countries.values, palette='Blues_d')

```

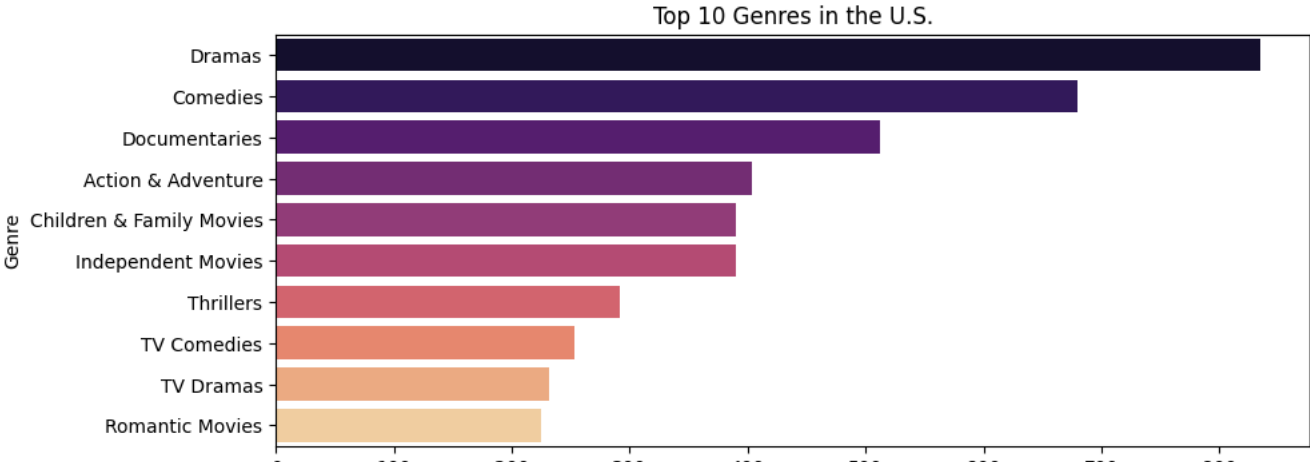




```
<ipython-input-4-5af6c619b417>:51: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg
sns.countplot(data=df, x='rating', order=df['rating'].value_counts().index, palette='Set3')
```

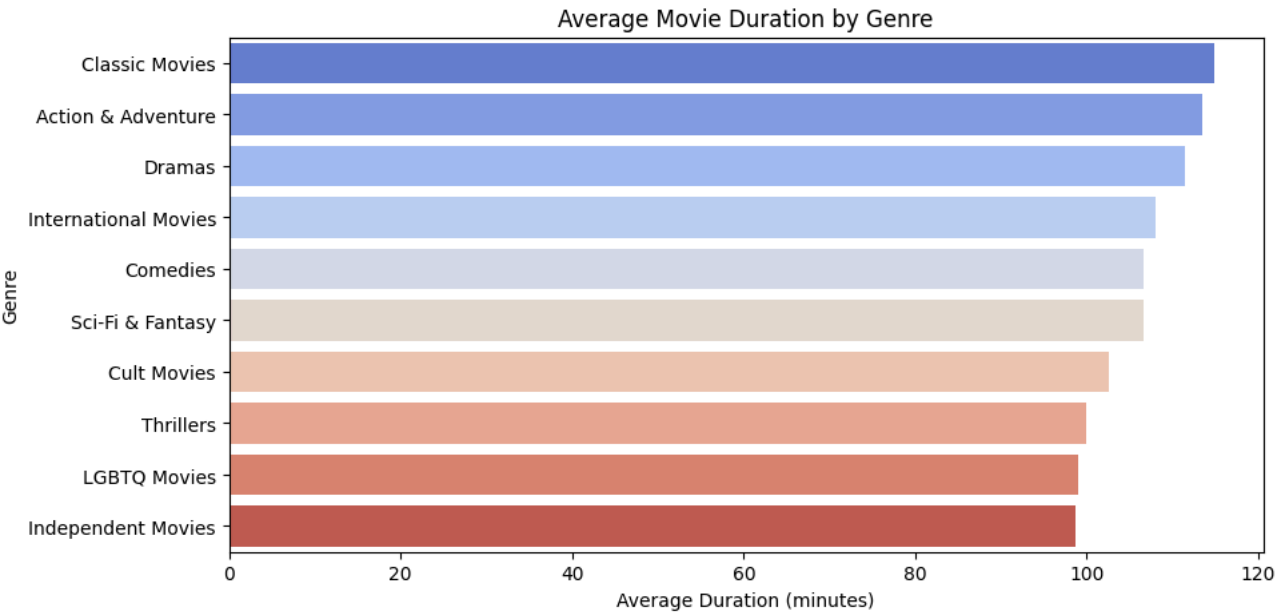


```
<ipython-input-4-5af6c619b417>:74: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `leg
sns.barplot(x=genre_counts.values, y=genre_counts.index, palette='magma')
```



0 100 200 300 400 500 600 700 800  
Number of Titles

```
<ipython-input-4-5af6c619b417>:89: FutureWarning:  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `leg  
sns.barplot(x=genre_duration.values, y=genre_duration.index, palette='coolwarm')
```



```

# Part 2: Linear Regression – Students Performance Dataset
# Q1: Simple Linear Regression
# Task: Predict math score based on hours studied
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

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

# OPTIONAL: Simulate 'hours studied' if not in original dataset
np.random.seed(42)
df['hours_studied'] = np.clip(np.random.normal(loc=5, scale=2, size=len(df)), 1, 10)

# Simple Linear Regression: Predict math score using hours studied
X = df[['hours_studied']]
y = df['math score']


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

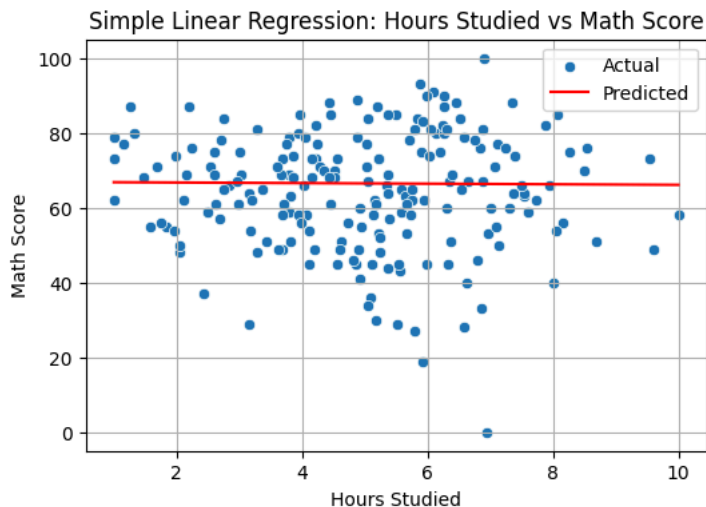
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("R² Score:", r2_score(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))

# Plot regression line
plt.figure(figsize=(6, 4))
sns.scatterplot(x=X_test.squeeze(), y=y_test, label="Actual")
sns.lineplot(x=X_test.squeeze(), y=y_pred, color='red', label="Predicted")
plt.title("Simple Linear Regression: Hours Studied vs Math Score")
plt.xlabel("Hours Studied")
plt.ylabel("Math Score")
plt.legend()
plt.grid(True)
plt.show()

```

 R² Score: -0.016749409907296897  
 Mean Squared Error: 247.41417460778575



```

# Q2: Multiple Linear Regression
# Task: Predict math score using reading score, writing score, and hours studied
# Multiple Linear Regression
features = ['reading score', 'writing score', 'hours_studied']
X = df[features]
y = df['math score']

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

```

```

multi_model = LinearRegression()
multi_model.fit(X_train, y_train)
y_pred_multi = multi_model.predict(X_test)

print("Multiple Linear Regression:")
print("R² Score:", r2_score(y_test, y_pred_multi))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_multi)))

```

```

# Interpret coefficients
for feature, coef in zip(features, multi_model.coef_):
    print(f"{feature}: {coef:.2f}")

```

```

↗ Multiple Linear Regression:
R² Score: 0.682695497612081
RMSE: 8.787056954627777
reading score: 0.57
writing score: 0.28
hours_studied: -0.30

```

```

# Q3: Polynomial Regression
# Task: Predict math score using reading score (polynomial of degree 2 or 3)
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error

# Clean column names in case of extra spaces
df.columns = df.columns.str.strip()

# Define features and target
X = df[['reading score']]
y = df['math score']

# Polynomial transformation (degree 2)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.2, random_state=42)

# Fit polynomial regression model
poly_model = LinearRegression()
poly_model.fit(X_train, y_train)

# Predictions
y_pred_poly = poly_model.predict(X_test)

# Evaluation
print("Polynomial Regression (Degree 2):")
print("R² Score:", r2_score(y_test, y_pred_poly))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_poly)))

# --- Plotting Polynomial Fit ---
# Generate sorted X values for smooth curve
X_sorted = X.sort_values(by='reading score')
X_sorted_poly = poly.transform(X_sorted)
y_sorted_pred = poly_model.predict(X_sorted_poly)

plt.figure(figsize=(8, 5))
sns.scatterplot(x=X['reading score'], y=y, label='Actual', alpha=0.6)
plt.plot(X_sorted, y_sorted_pred, color='red', label='Polynomial Fit')
plt.title("Polynomial Regression: Reading Score vs Math Score")
plt.xlabel("Reading Score")
plt.ylabel("Math Score")
plt.legend()
plt.grid(True)
plt.show()

```