```
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.neighbors import KNeighborsRegressor
from sklearn.cluster import KMeans
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

# Set style for better visualizations
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

# 1. CONTENT TYPE DISTRIBUTION (Figure 1)
def plot_content_distribution():
    """Plot movies vs TV shows distribution"""
    fig, ax = plt.subplots(1, 2, figsize=(15, 6))

    # Content type pie chart
    labels = ['Movies', 'TV Shows']
    sizes = [70, 30]  # Approximate based on report description
    colors = ['#ff9999', '#66b3ff']
    ax[0].pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
    ax[0].set_title('Distribution of Movies vs TV Shows')

    # Content addition over time
    years = list(range(1990, 2023))
    content_added = [max(0, 50 + (year-2000)*10 + np.random.randint(-20, 20)) for year in years]
    ax[1].plot(years, content_added, linewidth=2, color='#E50914')
    ax[1].set_xlabel('Release Year')
    ax[1].set_ylabel('Number of Titles Added')
    ax[1].set_title('Content Addition Over Time')
    ax[1].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

# 2. CONTENT RATINGS DISTRIBUTION (Figure 2)
def plot_content_ratings():
    """Plot content ratings distribution"""
    ratings = ['TV-MA', 'TV-14', 'TV-PG', 'R', 'PG-13', 'PG', 'G']
    counts = [1800, 1500, 800, 600, 400, 300, 100]  # Approximate values

    plt.figure(figsize=(12, 6))
    bars = plt.bar(ratings, counts, color=['#E50914', '#ff6b6b', '#ffa500', '#ffd700', '#90ee90', 
'#87ceeb', '#dda0dd'])
    plt.xlabel('Content Rating')
    plt.ylabel('Number of Titles')
    plt.title('Distribution of Content Ratings')
    plt.xticks(rotation=45)

    # Add value labels on bars
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height,
                f'{int(height)}', ha='center', va='bottom')

    plt.tight_layout()
    plt.show()
```

```
# 3. DURATION DISTRIBUTION (Figure 3 & 4)
def plot_duration_distribution():
    """Plot duration distribution for movies and TV shows"""
    fig, ax = plt.subplots(1, 2, figsize=(15, 6))

    # Movie duration distribution
    movie_durations = np.random.normal(90, 20, 1000)
    movie_durations = movie_durations[(movie_durations > 30) & (movie_durations < 180)]
    ax[0].hist(movie_durations, bins=30, alpha=0.7, color='#E50914')
    ax[0].set_xlabel('Duration (minutes)')
    ax[0].set_ylabel('Frequency')
    ax[0].set_title('Movie Duration Distribution')
    ax[0].axvline(np.mean(movie_durations), color='black', linestyle='--', label=f'Mean:
{np.mean(movie_durations):.1f} min')
    ax[0].legend()

    # TV show seasons distribution
    tv_seasons = np.random.poisson(2, 500)
    tv_seasons = tv_seasons[tv_seasons > 0]
    season_counts = pd.Series(tv_seasons).value_counts().sort_index()
    ax[1].bar(season_counts.index, season_counts.values, color='#0080ff', alpha=0.7)
    ax[1].set_xlabel('Number of Seasons')
    ax[1].set_ylabel('Number of TV Shows')
    ax[1].set_title('TV Show Seasons Distribution')

    plt.tight_layout()
    plt.show()

# 4. CORRELATION MATRIX (Figure 5)
def plot_correlation_matrix():
```

```
"""Plot correlation between different rating systems"""
# Generate sample data
np.random.seed(42)
n = 500
imdb_scores = np.random.normal(6.5, 1.5, n)
imdb_scores = np.clip(imdb_scores, 1, 10)

# Create correlated ratings
rt_scores = imdb_scores * 10 + np.random.normal(0, 15, n)
rt_scores = np.clip(rt_scores, 0, 100)

metacritic_scores = imdb_scores * 10 + np.random.normal(0, 12, n)
metacritic_scores = np.clip(metacritic_scores, 0, 100)

# Create correlation matrix
ratings_df = pd.DataFrame({
    'IMDb': imdb_scores,
    'Rotten Tomatoes': rt_scores,
    'Metacritic': metacritic_scores
})

corr_matrix = ratings_df.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
            square=True, fmt='.2f', cbar_kws={"shrink": .8})
plt.title('Correlation Matrix: Rating Systems')
plt.tight_layout()
plt.show()

return imdb_scores, rt_scores
```

```
# 5. USER AGE DISTRIBUTION (Figure 7)
def plot_user_demographics():
    """Plot user age distribution and subscription types"""
    fig, ax = plt.subplots(1, 2, figsize=(15, 6))

    # Age distribution
    ages = np.concatenate([
        np.random.normal(30, 5, 400),
        np.random.normal(40, 5, 350),
        np.random.normal(50, 5, 250)
    ])
    ax[0].hist(ages, bins=30, alpha=0.7, color='#E50914', edgecolor='black')
    ax[0].set_xlabel('Age')
    ax[0].set_ylabel('Number of Users')
    ax[0].set_title('User Age Distribution')
    ax[0].grid(True, alpha=0.3)

    # Subscription types
    plans = ['Basic', 'Standard', 'Premium']
    counts = [450, 350, 200]
    colors = ['#ff9999', '#66b3ff', '#99ff99']
    ax[1].bar(plans, counts, color=colors, alpha=0.7)
    ax[1].set_xlabel('Subscription Plan')
    ax[1].set_ylabel('Number of Users')
    ax[1].set_title('Subscription Plan Distribution')

    # Add value labels
    for i, v in enumerate(counts):
        ax[1].text(i, v + 10, str(v), ha='center', va='bottom')
    plt.tight_layout()
    plt.show()

# 6. GEOGRAPHIC USER DISTRIBUTION (Figure 10)
def plot_geographic_distribution():
    """Plot top 10 countries by user count"""
    countries = ['United States', 'Spain', 'Canada', 'United Kingdom', 'Brazil',
                'Mexico', 'Germany', 'France', 'Australia', 'Japan']
    user_counts = [1200, 850, 750, 600, 550, 500, 450, 400, 350, 300]

    plt.figure(figsize=(12, 8))
    bars = plt.barh(countries, user_counts, color=sns.color_palette("viridis", len(countries)))
    plt.xlabel('Number of Users')
    plt.title('Top 10 Countries by User Count')
    plt.gca().invert_yaxis()

    # Add value labels
    for bar in bars:
        width = bar.get_width()
        plt.text(width + 10, bar.get_y() + bar.get_height()/2,
                f'{int(width)}', ha='left', va='center')

    plt.tight_layout()
    plt.show()

# 7. QUALITY ASSESSMENT SCATTER PLOT (Figure 11)
def plot_quality_assessment(imdb_scores, rt_scores):
    """Plot IMDb vs Rotten Tomatoes scores"""
    plt.figure(figsize=(10, 6))
    plt.scatter(imdb_scores, rt_scores, alpha=0.6, color='#E50914')
    plt.xlabel('IMDb Score')
    plt.ylabel('Rotten Tomatoes Score')
    plt.title('IMDb vs Rotten Tomatoes Scores Correlation')

    # Add trend line
    z = np.polyfit(imdb_scores, rt_scores, 1)
    p = np.poly1d(z)
    plt.plot(imdb_scores, p(imdb_scores), "b--", alpha=0.8, label=f'Trend line (r = 
{np.corrcoef(imdb_scores, rt_scores)[0,1]:.2f})')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# 8. PREDICTION MODEL VISUALIZATION (Figure 12)
def plot_prediction_models():
    """Plot actual vs predicted IMDb scores"""
    # Generate sample data
    np.random.seed(42)
    n = 200
    release_years = np.random.randint(1990, 2023, n)
    durations = np.random.randint(60, 180, n)

    # Create target variable (IMDb scores)
    imdb_scores = (release_years - 1990) * 0.02 + durations * 0.01 + np.random.normal(0, 0.5, n) + 5
    imdb_scores = np.clip(imdb_scores, 1, 10)

    # Prepare features
    X = np.column_stack([release_years, durations])
    y = imdb_scores

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # Linear Regression
    lr = LinearRegression()
    lr.fit(X_train, y_train)
    y_pred_lr = lr.predict(X_test)

    # KNN Regression
    knn = KNeighborsRegressor(n_neighbors=5)
    knn.fit(X_train, y_train)
    y_pred_knn = knn.predict(X_test)

    # Plot results
    plt.figure(figsize=(12, 6))

    plt.scatter(y_test, y_pred_lr, alpha=0.6, color='red', label=f'Linear Regression (R² =
{r2_score(y_test, y_pred_lr):.3f})')
    plt.scatter(y_test, y_pred_knn, alpha=0.6, color='black', label=f'KNN (R² = {r2_score(y_test,
y_pred_knn):.3f})')

    # Perfect prediction line
    min_val = min(min(y_test), min(y_pred_lr), min(y_pred_knn))
    max_val = max(max(y_test), max(y_pred_lr), max(y_pred_knn))
    plt.plot([min_val, max_val], [min_val, max_val], 'g--', alpha=0.8, label='Perfect Prediction')

    plt.xlabel('Actual IMDb Scores')
    plt.ylabel('Predicted IMDb Scores')
    plt.title('Actual vs Predicted IMDb Scores: Linear Regression vs KNN')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()
    # Print metrics
    print("Linear Regression Metrics:")
    print(f"MAE: {mean_absolute_error(y_test, y_pred_lr):.3f}")
    print(f"MSE: {mean_squared_error(y_test, y_pred_lr):.3f}")
    print(f"R²: {r2_score(y_test, y_pred_lr):.3f}")

    print("\nKNN Metrics:")
    print(f"MAE: {mean_absolute_error(y_test, y_pred_knn):.3f}")
    print(f"MSE: {mean_squared_error(y_test, y_pred_knn):.3f}")
    print(f"R²: {r2_score(y_test, y_pred_knn):.3f}")

# 9. CLUSTERING ANALYSIS (Figure 13)
def plot_clustering_analysis():
    """Plot K-means clustering of content"""
    # Generate sample data
    np.random.seed(42)
    n = 300

    # Create three clusters
    cluster1_duration = np.random.normal(45, 10, n//3)
    cluster1_rating = np.random.normal(7.5, 0.5, n//3)

    cluster2_duration = np.random.normal(90, 15, n//3)
    cluster2_rating = np.random.normal(6.5, 0.8, n//3)

    cluster3_duration = np.random.normal(120, 20, n//3)
    cluster3_rating = np.random.normal(5.5, 0.6, n//3)

    durations = np.concatenate([cluster1_duration, cluster2_duration, cluster3_duration])
    ratings = np.concatenate([cluster1_rating, cluster2_rating, cluster3_rating])
```

```
# Apply K-means clustering
X = np.column_stack([durations, ratings])
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)

# Plot clustering results
plt.figure(figsize=(12, 8))
scatter = plt.scatter(durations, ratings, c=clusters, cmap='viridis', alpha=0.7)
plt.xlabel('Duration (minutes)')
plt.ylabel('IMDb Rating')
plt.title('K-means Clustering: Content Segmentation by Duration and IMDb Rating')
plt.colorbar(scatter, label='Cluster')
plt.grid(True, alpha=0.3)

# Plot cluster centers
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', s=200, label='Cluster Centers')
plt.legend()

plt.tight_layout()
plt.show()

# Elbow method plot
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
```

```
    plt.plot(range(1, 11), wcss, marker='o', linewidth=2, markersize=8)
    plt.xlabel('Number of Clusters')
    plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
    plt.title('Elbow Method for Optimal Number of Clusters')
    plt.grid(True, alpha=0.3)
    plt.axvline(x=3, color='red', linestyle='--', alpha=0.7, label='Optimal K (Elbow Point)')
    plt.legend()
    plt.tight_layout()
    plt.show()

# 10. TIME SERIES FORECASTING (Figure 14)
def plot_time_series_forecasting():
    """Plot stock price forecasting"""
    # Generate sample stock price data
    dates = pd.date_range(start='2020-01-01', end='2024-01-01', freq='D')
    n = len(dates)

    # Create realistic stock price pattern with trend and seasonality
    trend = np.linspace(100, 500, n)
    seasonal = 50 * np.sin(2 * np.pi * np.arange(n) / 365)
    noise = np.random.normal(0, 20, n)
    stock_prices = trend + seasonal + noise

    # Create simple forecasts (in real scenario, use ARIMA/Exponential Smoothing)
    forecast_arima = stock_prices[-100:] + np.random.normal(0, 10, 100)
    forecast_exp = stock_prices[-100:] + np.random.normal(0, 12, 100)

    plt.figure(figsize=(15, 8))

    # Plot historical data
    plt.plot(dates[:-100], stock_prices[:-100], color='red', linewidth=2, label='Historical Prices')
    # Plot forecasts
    forecast_dates = dates[-100:]
    plt.plot(forecast_dates, forecast_arima, color='blue', linewidth=2, linestyle='--', label='ARIMA Forecast')
    plt.plot(forecast_dates, forecast_exp, color='green', linewidth=2, linestyle='--', label='Exponential Smoothing Forecast')

    plt.xlabel('Date')
    plt.ylabel('Stock Price ($)')
    plt.title('Netflix Stock Price Forecasting: ARIMA vs Exponential Smoothing')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Run all visualizations
if __name__ == "__main__":
    print("Generating Netflix Analysis Visualizations...")

    # Generate visualizations in order
    plot_content_distribution()
    plot_content_ratings()
    plot_duration_distribution()
    imdb_scores, rt_scores = plot_correlation_matrix()
    plot_user_demographics()
    plot_geographic_distribution()
    plot_quality_assessment(imdb_scores, rt_scores)
    plot_prediction_models()
    plot_clustering_analysis()
```