MOVIE ANALYSIS (GENRE AND MONTH)

LOADING LIBRARIES AND DATASET

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#supervised algorithms (random forest)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

#unsupervised algorithms (kmeans)
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, MinMaxScaler

#unsupervised algorithms (association rule mining)
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

#one hot encoding
from sklearn.preprocessing import MultiLabelBinarizer
```

```
movie = pd.read_csv('movie.csv')

print(movie.head()) #first 5 rows
print()
print(movie.info())  #movie info
print()
print(movie.describe)
```

```
0                7.35                7.60                7.60                NaN
1                7.22                7.23                7.23               7.24
2                4.78                4.96                5.31               4.60
3                6.29                 NaN                5.99               6.47
4                6.86                6.80                6.53               6.56
...               ...                 ...                 ...                ...
1368             7.52                7.09                7.20               6.34
1369             6.50                6.50                6.50               6.50
1370             6.38                6.35                6.31               6.22
1371             5.69                5.67                5.67               5.70
1372             5.15                5.74                5.74               5.10

      cinematographer_rating  editor_rating  \
0                       7.13           7.39
1                       7.10           7.06
2                       5.25           6.53
3                       5.90           6.22
4                       6.80           6.80
...                      ...            ...
1368                    7.14           7.09
1369                    6.50           6.50
1370                    6.18           6.32
1371                     NaN           5.67
1372                    5.10           5.74

                                          genre
0                      Comedy, Drama, Thriller
1                                         Drama
2                         Action, Comedy, Crime
3      Adventure, Animation, Comedy, Family, Fantasy
4       Adventure, Animation, Drama, Family, Fantasy
...                                         ...
1368                           Adventure, Drama
1369                                   Thriller
1370                           Action, Thriller
1371  Animation, Comedy, Documentary, Drama, History
1372              Action, Adventure, Comedy, Horror

[1373 rows x 16 columns]>
```

DATA PREPROCESSING

Converting release_year to datetime format and exporting the months

```
from datetime import datetime

# Now convert the entire column to datetime safely
movie['release_date'] = pd.to_datetime(movie['release_date'], errors='coerce')

#adding a column to view the month release for each
movie['release_month'] = movie['release_date'].dt.month_name()

#adding a column to view the year release for each
movie['release_year'] = movie['release_date'].dt.year
```

```
movie.columns
```

```
Index(['title', 'imdb_id', 'release_date', 'budget', 'revenue', 'tmdb_rating',
       'vote_count', 'runtimeMinutes', 'imdb_rating', 'avg_cast_rating',
       'director_rating', 'writer_rating', 'composer_rating',
       'cinematographer_rating', 'editor_rating', 'genre', 'release_month',
       'release_year'],
      dtype='object')
```

```
movie_df = movie[['title', 'release_month', 'runtimeMinutes', 'genre', 'budget',

movie_df.head()
```

	title	release_month	runtimeMinutes	genre	budget	revenue
0	Once Upon a Time... in Hollywood	July	161	Comedy, Drama, Thriller	95000000	392105159
1	Pain and Glory	March	113	Drama	10769016	37359689
2	Taxi 5	January	102	Action, Comedy, Crime	20390000	64497208

CONVERTING GENRE TO ONE HOT ENCODING

```
#split the genres into lists
movie_df['genre'] = movie_df['genre'].str.split(',')

mlb = MultiLabelBinarizer()
genre_encoded = pd.DataFrame(mlb.fit_transform(movie_df['genre']), columns=mlb.c
```

```
/tmp/ipython-input-3359392152.py:2: 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/stabl
  movie_df['genre'] = movie_df['genre'].str.split(',')
```

```
#combine with original dataframe
movie_df = pd.concat([movie_df, genre_encoded], axis=1)
```

CONVERTING MONTHS TO ONE HOT ENCODING

```
# One-hot encode the release month
month_encoded = pd.get_dummies(movie_df['release_month'], prefix='Month')

# Add back to the original dataframe
movie_df = pd.concat([movie_df, month_encoded], axis=1)
```

DATA EXPLORATION USING ALGORITHMS

```
movie_df.columns
```

```
Index(['title', 'release_month', 'runtimeMinutes', 'genre', 'budget',
       'revenue', ' Adventure', ' Animation', ' Biography', ' Comedy',
       ' Crime', ' Documentary', ' Drama', ' Family', ' Fantasy', ' History',
       ' Horror', ' Music', ' Musical', ' Mystery', ' Romance', ' Sci-Fi',
       ' Science Fiction', ' Sport', ' Thriller', ' Tv Movie', ' War',
       ' Western', 'Action', 'Adventure', 'Animation', 'Biography', 'Comedy',
       'Crime', 'Documentary', 'Drama', 'Fantasy', 'Horror', 'Mystery',
       'Sci-Fi', 'Thriller', 'Month_April', 'Month_August', 'Month_December',
       'Month_February', 'Month_January', 'Month_July', 'Month_June',
       'Month_March', 'Month_May', 'Month_November', 'Month_October',
       'Month_September'],
      dtype='object')
```

So to cluster the movies using the features (with emphasis on how block boster, and other movies are released).

```
#setting columns i want to find the clusters
df = movie_df[['revenue','budget', ' Adventure', ' Animation', ' Biography', ' C
       ' Crime', ' Documentary', ' Drama', ' Family', ' Fantasy', ' History',
       ' Horror', ' Music', ' Musical', ' Mystery', ' Romance', ' Sci-Fi',
       ' Science Fiction', ' Sport', ' Thriller', ' Tv Movie', ' War',
       ' Western', 'Action', 'Adventure', 'Animation', 'Biography', 'Comedy',
       'Crime', 'Documentary', 'Drama', 'Fantasy', 'Horror', 'Mystery',
       'Sci-Fi', 'Thriller', 'Month_April', 'Month_August', 'Month_December',
       'Month_February', 'Month_January', 'Month_July', 'Month_June',
       'Month_March', 'Month_May', 'Month_November', 'Month_October',
       'Month_September']]
```

KMEANS ALGORITHM

SCALING THE REVENUE AND BUDGET

```
#scaling numeric features (to expose the outliers clearly)
scaler = StandardScaler()
df_scaled = df.copy()
df_scaled[['revenue', 'budget']] = scaler.fit_transform(df_scaled[['revenue', 'b
```

```
print(df_scaled)
```

```
       revenue    budget  Adventure  Animation  Biography  Comedy  Crime  \
0     1.165293  0.894435          0          0          0       0      0
1    -0.268900 -0.528509          0          0          0       0      0
2    -0.159187 -0.365979          0          0          0       1      1
3     0.063422  0.978902          0          1          0       1      0
4    -0.151229 -0.284721          0          1          0       0      0
...        ...       ...        ...        ...        ...     ...    ...
1368 -0.419941 -0.486212          0          0          0       0      0
1369 -0.419941 -0.709505          0          0          0       0      0
1370 -0.349743 -0.659754          0          0          0       0      0
1371 -0.419941 -0.710181          0          0          0       1      0
1372 -0.419941 -0.709826          1          0          0       1      0

       Documentary  Drama  Family  ...  Month_December  Month_February  \
0                0      1       0  ...           False           False
1                0      0       0  ...           False           False
2                0      0       0  ...           False           False
3                0      0       1  ...           False           False
4                0      1       1  ...           False           False
...            ...    ...     ...  ...             ...             ...
1368             0      1       0  ...           False           False
1369             0      0       0  ...           False           False
1370             0      0       0  ...           False           False
1371             1      1       0  ...           False           False
1372             0      0       0  ...           False           False

      Month_January  Month_July  Month_June  Month_March  Month_May  \
0             False        True       False        False      False
1             False       False       False         True      False
2              True       False       False        False      False
3             False       False       False         True      False
4             False       False       False        False      False
...             ...         ...         ...          ...        ...
1368          False       False       False        False      False
1369          False       False       False        False      False
1370          False       False       False        False      False
1371          False       False       False         True      False
1372          False       False       False        False      False

      Month_November  Month_October  Month_September
0              False          False            False
1              False          False            False
2              False          False            False
3              False          False            False
4              False          False            False
...              ...            ...              ...
1368           False          False             True
1369           False          False             True
1370           False          False            False
1371           False          False            False
```

```
1372          False          False             True

[1373 rows x 49 columns]
```

Scaling instead of minmax to properly visualize the outliers (block buster movies)

APPLYING KMEANS ALGORITHM

```
#applying kmeans algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
movie_df['cluster'] = kmeans.fit_predict(df_scaled)
```

```
print(movie_df[['title', 'revenue', 'cluster']])
```

```
                              title    revenue  cluster
0     Once Upon a Time... in Hollywood  392105159        0
1                       Pain and Glory   37359689        1
2                               Taxi 5   64497208        1
3                          Wonder Park  119559110        1
4                    The King of Kings   66465461        0
...                                ...        ...      ...
1368                       Io Capitano          0        0
1369                         The Dunes          0        1
1370                              Fall   17363261        1
1371         Glossary of Broken Dreams          0        0
1372                    The VelociPastor          0        1

[1373 rows x 3 columns]
```

```
#inspecting cluster by their average
cluster_avg = movie_df.groupby('cluster')['revenue'].mean()
print(cluster_avg)
```

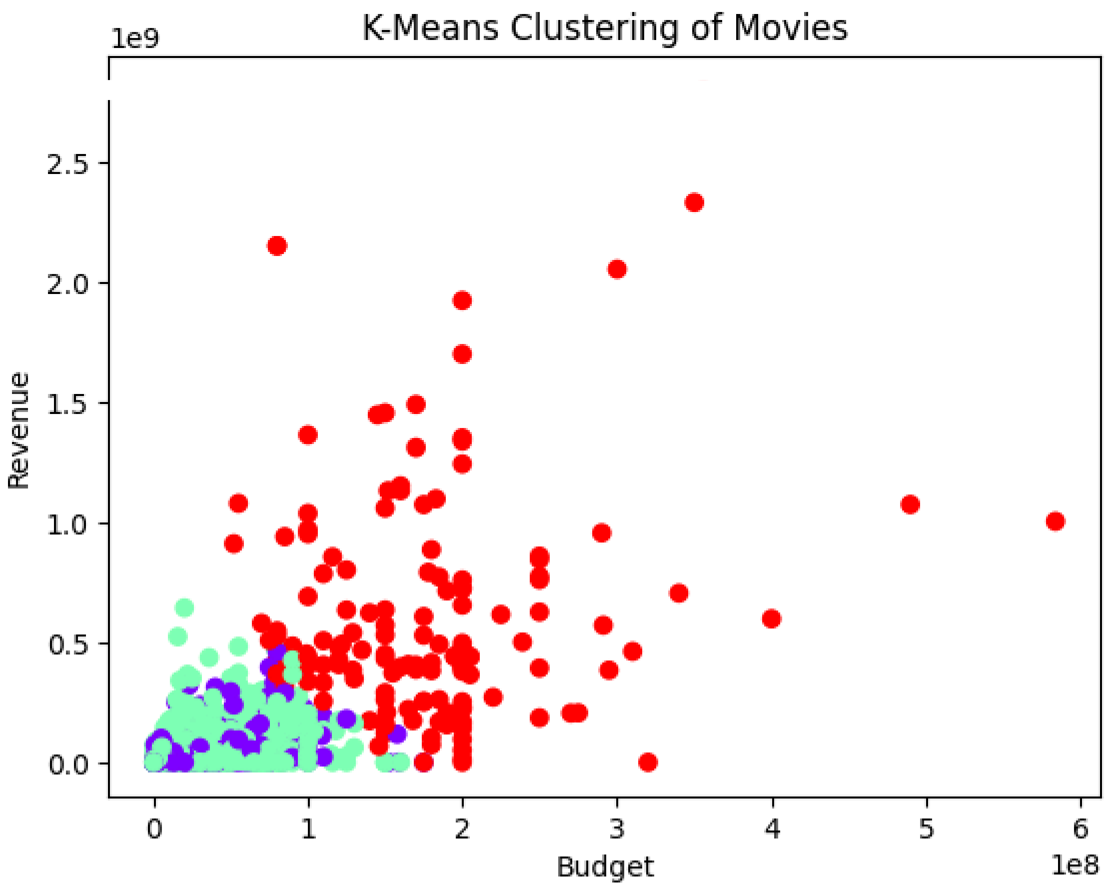
```
cluster
0    3.439339e+07
1    4.862925e+07
2    6.144514e+08
Name: revenue, dtype: float64
```

```
#find cluster with higest average revenue
highest_avg_cluster = cluster_avg.idxmax()

#subset the dataframe
high_rev_movies = movie_df[movie_df['cluster'] == highest_avg_cluster]

print(f'Cluster with the highest average revenue: {highest_avg_cluster}')
```

```
Cluster with the highest average revenue: 2
```

```
#visualizing the clusters
plt.scatter(movie_df['budget'], movie_df['revenue'], c=movie_df['cluster'], cmap
plt.xlabel('Budget')
```

```
plt.ylabel('Revenue')
plt.title('K-Means Clustering of Movies')
plt.show()
```



From the visualization, we can see the outliers very clearly as very large values where grouped into one cluster.

Analyzing only cluster 2 (high earning movies or outliers in this case). Hence we can drop budget and revenue since we already know they have high in both and association rule does not accept partial values(must be 1 Or 0, true or false)

ASSOCIATION RULE MINING

HANDLING OUTLIERS FIRST (HIGH BUDGET AND REVENUE MOVIE)

To determine the studio release for genres according to month

```
#obtaining only features already encoded
rule_feature_1 = df_scaled.loc[high_rev_movies.index]

#removing budget and revenue
rule_feature_1 = rule_feature_1.drop(columns=['budget', 'revenue'])
rule_feature_1. head()
```

	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama	Family	Fa
8	1	0	0	0	0	0	0	0	
10	1	0	0	1	0	0	0	1	
11	1	0	0	1	0	0	0	1	
12	0	0	0	0	0	0	1	0	
14	1	0	0	0	0	0	0	0	

5 rows × 47 columns

ASSOCIATION RULE MINING for outliers

```
#find frequent itemsets
frequent_items = apriori(rule_feature_1, min_support=0.05, use_colnames=True)

#extract rules
rules = association_rules(frequent_items, metric='lift', min_threshold=1)


rules = rules[rules['consequents'].apply(lambda x: all('Month' in item for item
```

```
/usr/local/lib/python3.12/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161
  warnings.warn(
```

```
#sorting by lift
rules = rules.sort_values(by='lift', ascending=False)
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

```
243                          ( Family,  Comedy)      (Month_June)  0.061644
44                                    ( Family)      (Month_June)  0.075342
39                                    ( Comedy)      (Month_June)  0.082192
10                                 ( Adventure)  (Month_February)  0.054795
140                        (Action,  Adventure)  (Month_February)  0.054795
```

```
427    0.421053  2.794258
529    0.421053  2.794258
220    0.400000  2.654545
455    0.400000  2.654545
208    0.384615  2.552448
399    0.380952  2.528139
191    0.380952  2.528139
102    0.300000  2.433333
280    0.300000  2.433333
333    0.300000  2.433333
484    0.347826  2.308300
26     0.333333  2.212121
259    0.320000  2.123636
271    0.307692  2.041958
49     0.240000  1.946667
68     0.275862  1.830721
243    0.272727  1.809917
44     0.268293  1.780488
39     0.218182  1.447934
10     0.078431  1.272331
140    0.078431  1.272331
37     0.145455  1.249198
60     0.073394  1.190622
35     0.145455  1.179798
64     0.110092  1.148100
56     0.166667  1.106061
63     0.128440  1.103076
66     0.128440  1.103076
162    0.127451  1.094579
148    0.127451  1.094579
13     0.127451  1.094579
16     0.127451  1.094579
136    0.127451  1.033769
9      0.127451  1.033769
14     0.098039  1.022409
156    0.098039  1.022409
```

HANDLING OTHER MOVIE (non-outliers)

```
#working on cluster 1 or 2
rule_feature_2 = movie_df[(movie_df['cluster'] == 1) | (movie_df['cluster'] == 0

#removing budget and revenue
rule_feature_2 = rule_feature_2.drop(columns=['budget', 'revenue','title', 'rele
rule_feature_2. head()
```

	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama	Family	Fan
0	0	0	0	0	0	0	1	0	
1	0	0	0	0	0	0	0	0	
2	0	0	0	1	1	0	0	0	
3	0	1	0	1	0	0	0	1	
4	0	1	0	0	0	0	1	1	

5 rows × 47 columns

```
#frequent items
frequent_items_2 = apriori(rule_feature_2, min_support=0.05, use_colnames=True)

#extract rules
rules_2 = association_rules(frequent_items_2, metric='lift', min_threshold=1)

# Keep only rules where the consequent is month
rules_2 = rules_2[rules_2['consequents'].apply(lambda x: all('Month' in item for

rules_2 = rules_2.sort_values(by='lift', ascending=False)
print(rules_2[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

```
/usr/local/lib/python3.12/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161
  warnings.warn(
   antecedents      consequents   support  confidence      lift
19    ( Drama)  (Month_October)  0.051345    0.116022  1.148057
```

COMPARSION OF RULES

FOR OUTLIERS

```
# Convert genre and month sets/tuples into strings
rules['genre_combo'] = rules['antecedents'].apply(lambda x: ', '.join(sorted(lis
rules['month_combo'] = rules['consequents'].apply(lambda x: ', '.join(sorted(lis

# Example: top 20 strongest rules
top_rules = rules.sort_values(by='lift', ascending=False).head(20)
```

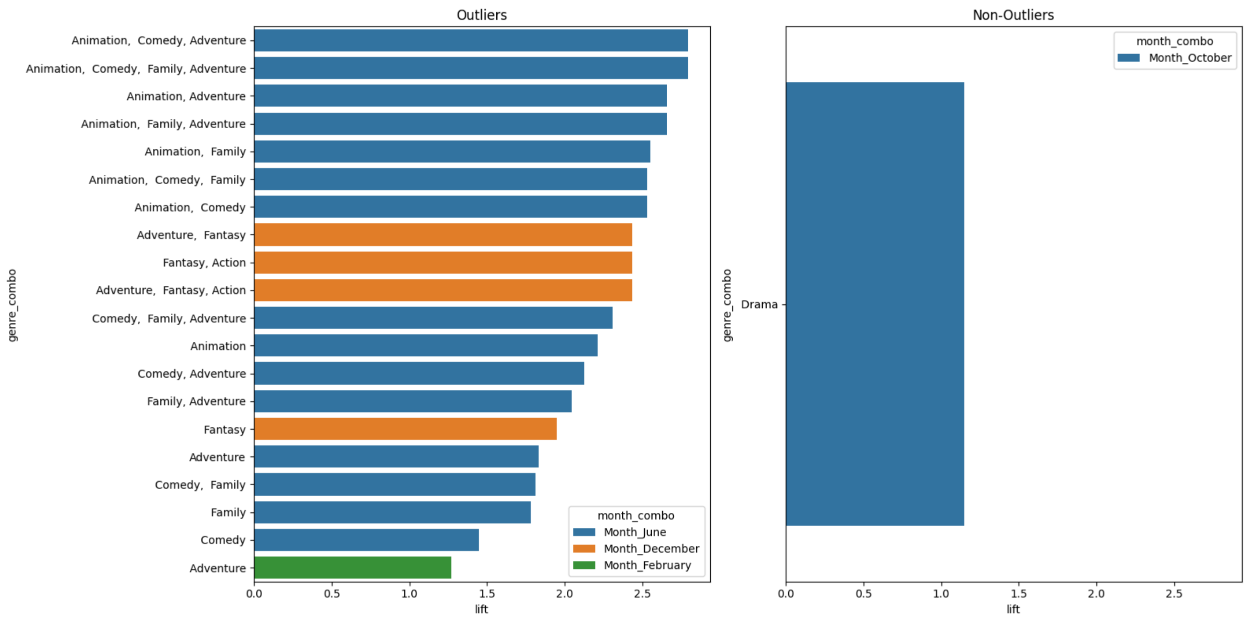
FOR NON-OUTLIERS

```
# Convert genre and month sets/tuples into strings
rules_2['genre_combo'] = rules_2['antecedents'].apply(lambda x: ', '.join(sorted
rules_2['month_combo'] = rules_2['consequents'].apply(lambda x: ', '.join(sorted
```

```
# Example: top 20 strongest rules
top_rules_2 = rules_2.sort_values(by='lift', ascending=False).head(20)
```

BAR PLOT COMPARISONS

```
fig, axes = plt.subplots(1, 2, figsize=(16, 8), sharex=True)

sns.barplot(data=top_rules, x='lift', y='genre_combo', hue='month_combo', ax=axe
axes[0].set_title('Outliers')
sns.barplot(data=top_rules_2, x='lift', y='genre_combo', hue='month_combo', ax=a
axes[1].set_title('Non-Outliers')

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



From the bar plot, we observe a clear pattern in the release timing of outlier (high-grossing) movies, whereas for non-outliers, no discernible pattern emerges. This suggests that most movie studios prioritize strategic release dates for high-budget or potentially high-revenue films, while lower-budget movies receive comparatively less planning in their release strategy.

This finding opens the door for further research: it would be valuable to investigate whether a more strategic release schedule for low-budget movies could positively influence their performance. Such research could focus solely on release timing, independent of marketing expenditure, to isolate the effect of strategic scheduling on movie success. Additionally, integrating social media analysis could provide insights into audience trends and preferences, helping studios identify optimal release windows based not only on historical patterns but also on real-time public interest.

The bar plot reveals a clear pattern in the release timing of high-grossing movies, while