📊 Project Overview

This project leverages a large-scale simulated Netflix dataset to conduct end-to-end business analytics and predictive modeling using Python. The objective is to analyze user behavior, subscription dynamics, revenue patterns, and content performance to uncover actionable insights that drive strategic growth.

Through advanced data preprocessing, exploratory analysis, feature engineering, and machine learning techniques, the project will evaluate customer churn risk, measure content effectiveness, forecast revenue trends, identify user segments, and assess recommendation performance.

🎯 Core Business Problem

How can a streaming platform optimize revenue growth, improve customer retention, enhance content strategy, and increase engagement using data-driven insights and predictive modeling?

This project aims to simulate real-world decision-making by integrating behavioral analytics, business intelligence, and machine learning into a unified analytical framework.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from prophet import Prophet
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv(r"C:\Users\papus\OneDrive\Documents\Netflix Project\Netflix_Data.csv")
print("Data Set Uploaded Successfully")
```

```
Data Set Uploaded Successfully
```

In [3]:
```
df.head()
```

Out[3]:

	user_id	age_group	gender	country	region	subscription_plan	monthly_fee	subscription_start_date	subscription_end_date	payment_method	...
0	U100000	21	Female	Canada	North America	Basic	8.99	01-03-2022	19-04-2022 00:00	Bank Transfer	...
1	U100001	17	Female	Nigeria	Africa	Basic	8.99	02-12-2022	07-02-2026	Bank Transfer	...
2	U100002	60	Male	Nigeria	Africa	Standard	13.99	01-09-2023	07-02-2026	Card	...
3	U100003	60	Male	Nigeria	Africa	Standard	13.99	26-05-2024	07-02-2026	Card	...
4	U100004	50	Female	Canada	North America	Basic	8.99	19-12-2023	07-02-2026	Bank Transfer	...

5 rows × 29 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['user_id', 'age_group', 'gender', 'country', 'region',
       'subscription_plan', 'monthly_fee', 'subscription_start_date',
       'subscription_end_date', 'payment_method', 'discount_applied',
       'churn_status', 'title', 'content_type', 'genre', 'language',
       'release_year', 'device_type', 'watch_time_minutes', 'session_count',
       'completion_percentage', 'date_watched', 'time_of_day', 'rating',
       'liked', 'recommendation_source', 'days_since_last_watch',
       'avg_weekly_watch_time', 'content_diversity_score'],
      dtype='object')
```

In [5]:
```
df.describe()
```

Out[5]:

	age_group	monthly_fee	release_year	watch_time_minutes	session_count	completion_percentage	rating	days_since_last_watch	avg_w
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	
mean	36.412180	12.990900	2011.534020	84.999620	2.358740	91.915480	3.417980	29.938080	
std	15.313068	3.610202	6.963695	28.086897	0.966408	11.953255	1.015861	17.621492	
min	17.000000	8.990000	2000.000000	10.000000	1.000000	20.000000	1.000000	0.000000	
25%	21.000000	8.990000	2006.000000	63.000000	2.000000	87.000000	3.000000	15.000000	
50%	40.000000	13.990000	2011.000000	83.000000	2.000000	99.000000	3.000000	30.000000	
75%	50.000000	17.990000	2018.000000	106.000000	3.000000	100.000000	4.000000	45.000000	
max	60.000000	17.990000	2025.000000	180.000000	6.000000	100.000000	5.000000	60.000000	

In [8]:
```
df.dtypes
```

Out[8]:
```
user_id                    object
age_group                   int64
gender                     object
country                    object
region                     object
subscription_plan          object
monthly_fee               float64
subscription_start_date    object
subscription_end_date      object
payment_method             object
discount_applied           object
churn_status               object
title                      object
content_type               object
genre                      object
language                   object
release_year                int64
device_type                object
watch_time_minutes          int64
session_count               int64
completion_percentage       int64
date_watched               object
time_of_day                object
rating                      int64
liked                      object
recommendation_source      object
days_since_last_watch       int64
avg_weekly_watch_time       int64
content_diversity_score   float64
dtype: object
```

In [9]:
```
# Converting date columns to datetime format

date_columns = ['date_watched',
                'subscription_start_date',
                'subscription_end_date']

for col in date_columns:
    df[col] = pd.to_datetime(df[col], dayfirst=True, errors='coerce')

# Check conversion
df[date_columns].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 3 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   date_watched            50000 non-null  datetime64[ns]
 1   subscription_start_date  50000 non-null  datetime64[ns]
 2   subscription_end_date    8609 non-null   datetime64[ns]
dtypes: datetime64[ns](3)
memory usage: 1.1 MB
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
user_id                       0
age_group                     0
gender                        0
country                       0
region                        0
subscription_plan             0
monthly_fee                   0
subscription_start_date       0
subscription_end_date      41391
payment_method                0
discount_applied              0
churn_status                  0
title                         0
content_type                  0
genre                         0
language                      0
release_year                  0
device_type                   0
watch_time_minutes            0
session_count                 0
completion_percentage         0
date_watched                  0
time_of_day                   0
rating                        0
liked                         0
recommendation_source         0
days_since_last_watch         0
avg_weekly_watch_time         0
content_diversity_score       0
dtype: int64
```

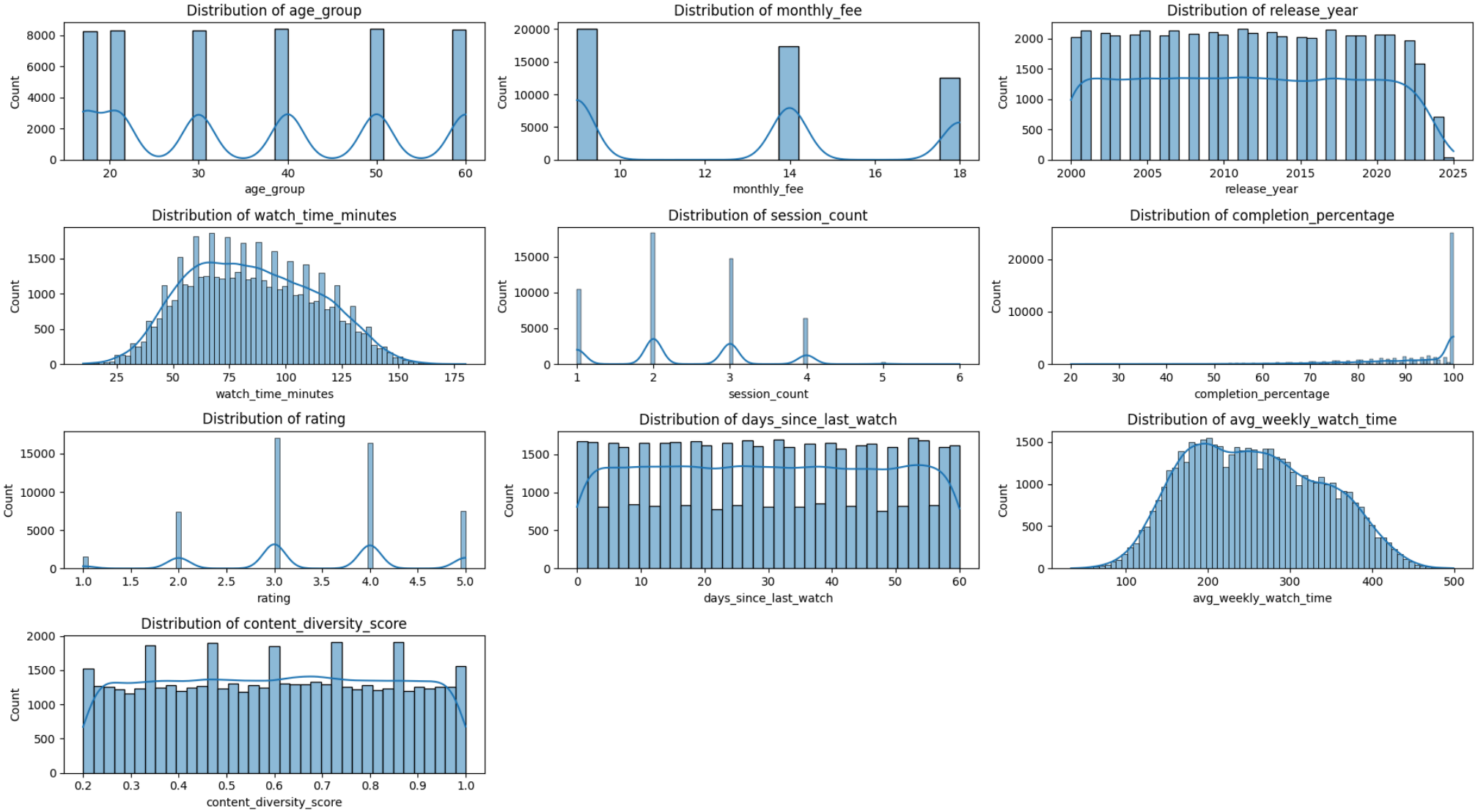
In [12]:
```
df.duplicated().sum()
```

Out[12]:
```
np.int64(0)
```

In [13]:
```
##identifying Numerical Columns
df.select_dtypes(include=['int64', 'float64']).columns
```

Out[13]:
```
Index(['age_group', 'monthly_fee', 'release_year', 'watch_time_minutes',
       'session_count', 'completion_percentage', 'rating',
       'days_since_last_watch', 'avg_weekly_watch_time',
       'content_diversity_score'],
      dtype='object')
```

In [16]:
```
num_cols = df.select_dtypes(include=['int64', 'float64']).columns

plt.figure(figsize=(18,10))

for i, col in enumerate(num_cols, 1):
    plt.subplot(4, 3, i)
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.tight_layout()

plt.show()
```

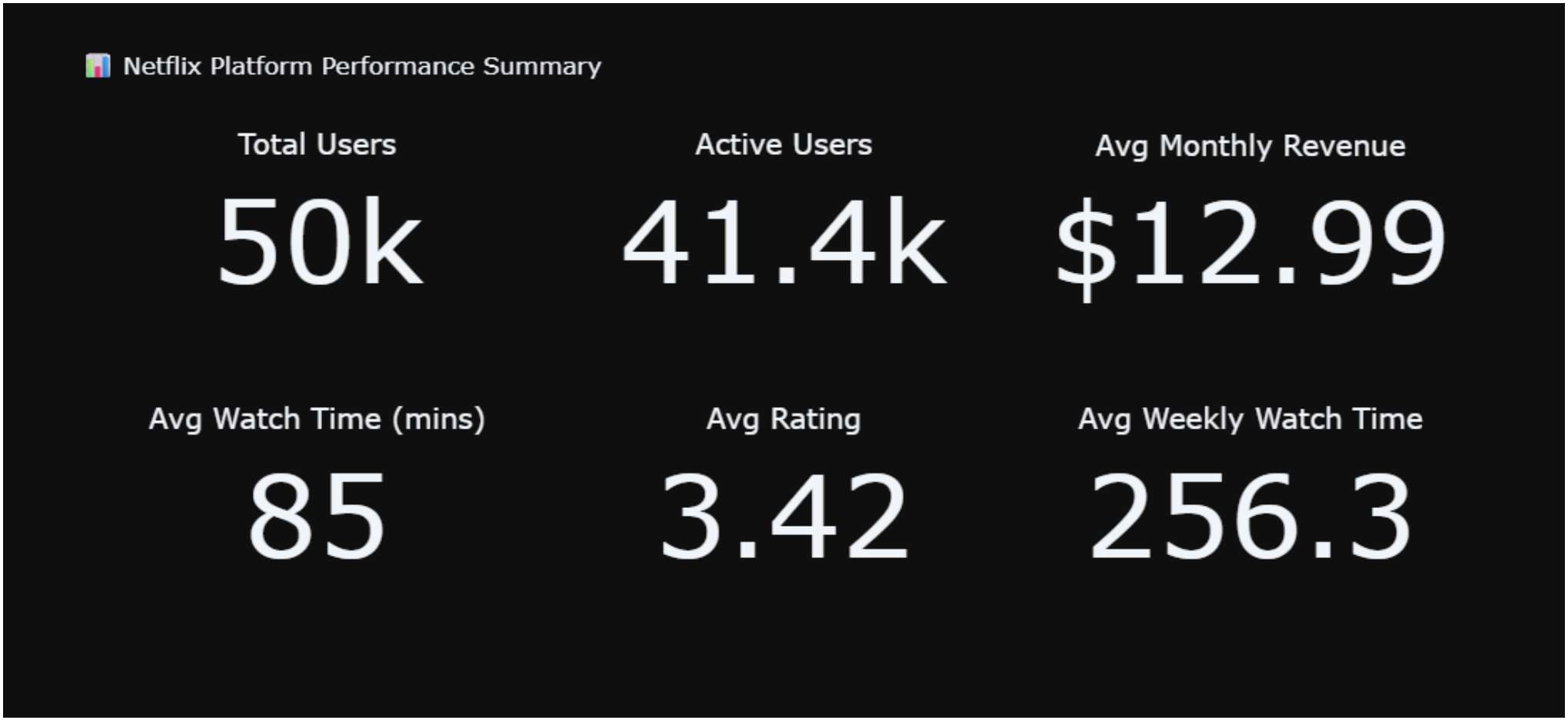


In [20]:

```
import plotly.graph_objects as go

# KPIs
total_users = df['user_id'].nunique()
active_users = df[df['subscription_end_date'].isnull()]['user_id'].nunique()
avg_revenue = round(df['monthly_fee'].mean(), 2)
avg_watch = round(df['watch_time_minutes'].mean(), 2)
avg_rating = round(df['rating'].mean(), 2)
avg_weekly_watch = round(df['avg_weekly_watch_time'].mean(), 2)
```

```
fig = go.Figure()

fig.add_trace(go.Indicator(
    mode="number",
    value=total_users,
    title={"text": "Total Users"},
    domain={'x':[0,0.3], 'y':[0.6,1]}
))

fig.add_trace(go.Indicator(
    mode="number",
    value=active_users,
    title={"text": "Active Users"},
    domain={'x':[0.35,0.65], 'y':[0.6,1]}
))

fig.add_trace(go.Indicator(
    mode="number",
    value=avg_revenue,
    number={'prefix': "$"},
    title={"text": "Avg Monthly Revenue"},
    domain={'x':[0.7,1], 'y':[0.6,1]}
))

fig.add_trace(go.Indicator(
    mode="number",
    value=avg_watch,
    title={"text": "Avg Watch Time (mins)"},
    domain={'x':[0,0.3], 'y':[0,0.4]}
))

fig.add_trace(go.Indicator(
    mode="number",
    value=avg_rating,
    title={"text": "Avg Rating"},
    domain={'x':[0.35,0.65], 'y':[0,0.4]}
))

fig.add_trace(go.Indicator(
    mode="number",
    value=avg_weekly_watch,
    title={"text": "Avg Weekly Watch Time"},
    domain={'x':[0.7,1], 'y':[0,0.4]}
))
```

```
fig.update_layout(
    template="plotly_dark",
    title="📊 Netflix Platform Performance Summary",
    height=500
)

fig.show()
```



In [28]:
```
df['churn_status'].value_counts(dropna=False)
```

Out[28]:
```
churn_status
Active       41391
Cancelled     8609
Name: count, dtype: int64
```

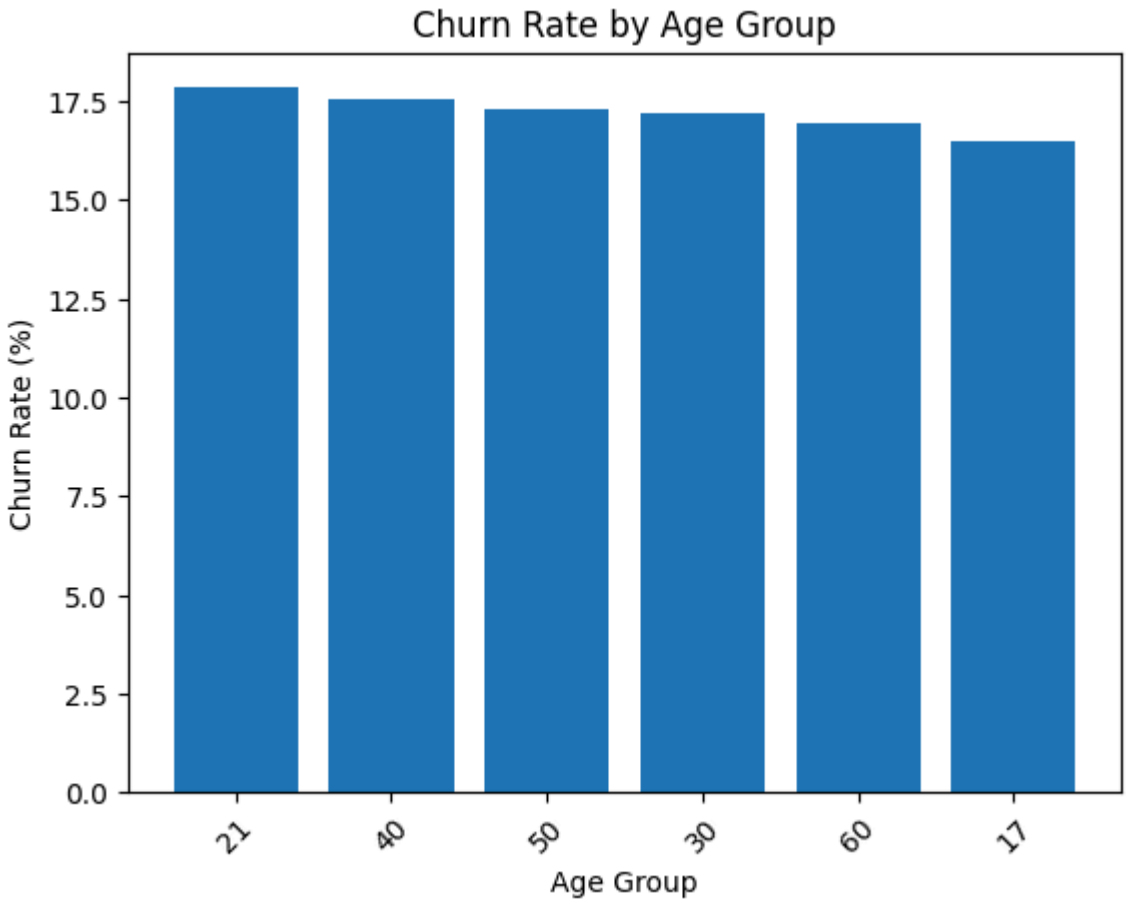
In [30]:
```
df['churn_status'] = df['churn_status'].str.strip().str.lower()
```

In [31]:
```
df['is_churned'] = df['churn_status'].map({
    'cancelled': 1,
    'active': 0
})
```

In [32]:
```
df['is_churned'].value_counts(dropna=False)
```

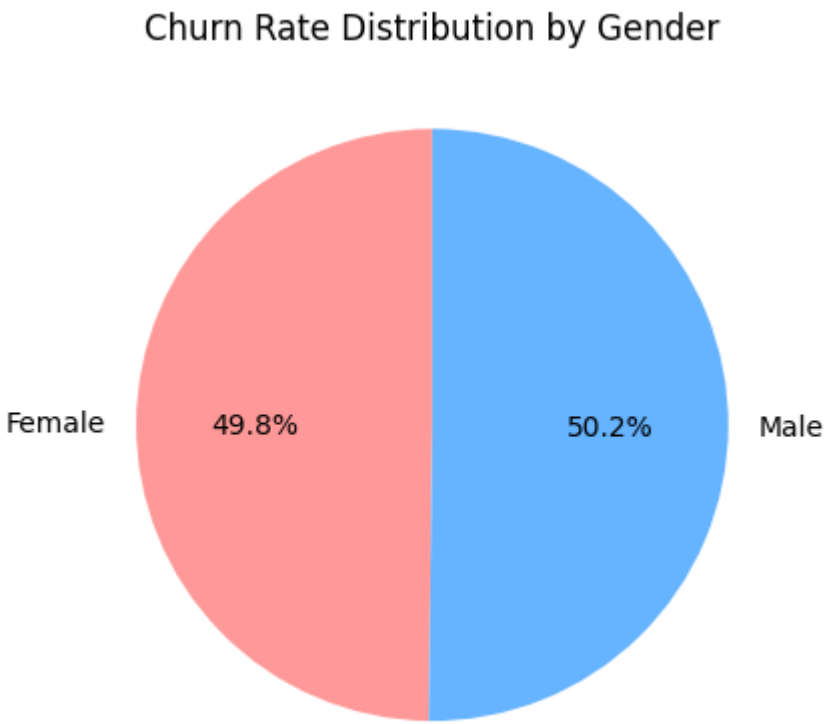
Out[32]:
```
is_churned
0    41391
1     8609
Name: count, dtype: int64
```

In [34]:
```
##Churn Rate By Age Group
churn_age = df.groupby('age_group')['is_churned'].mean() * 100
churn_age = churn_age.sort_values(ascending=False)

plt.figure()
plt.bar(churn_age.index.astype(str), churn_age.values)
plt.title("Churn Rate by Age Group")
plt.xlabel("Age Group")
plt.ylabel("Churn Rate (%)")
plt.xticks(rotation=45)
plt.show()
```

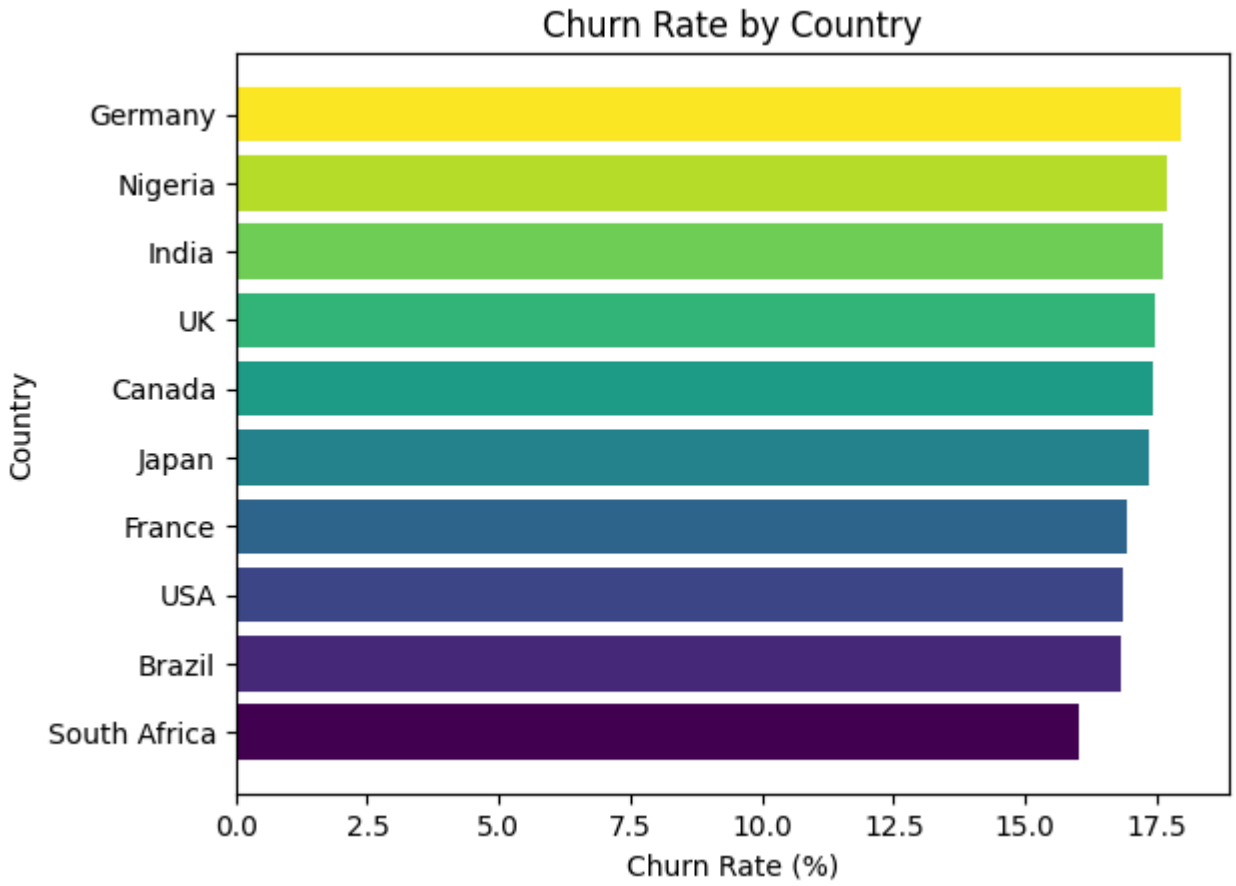


In [36]:
```
##ChurnRate By Gender


churn_gender = df.groupby('gender')['is_churned'].mean() * 100


plt.figure()

plt.pie(
    churn_gender.values,
    labels=churn_gender.index,
    autopct='%1.1f%%',
    colors=['#ff9999', '#66b3ff'],  # different colors
    startangle=90
)
```

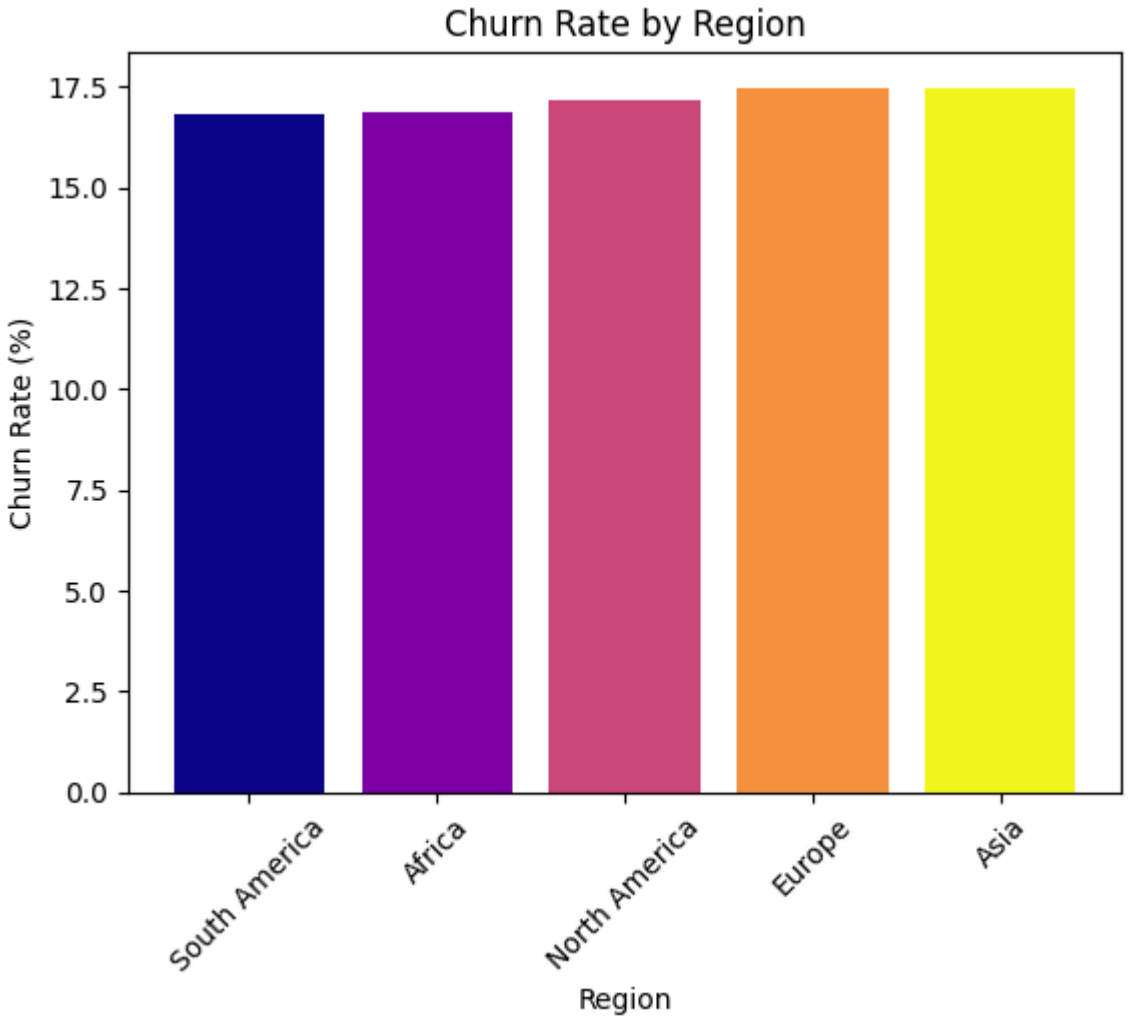
```
plt.title("Churn Rate Distribution by Gender")
plt.show()
```



In [37]:
```
##Churn Rate by Country

churn_country = df.groupby('country')['is_churned'].mean() * 100
churn_country = churn_country.sort_values()

colors = plt.cm.viridis(np.linspace(0, 1, len(churn_country)))

plt.figure()

plt.barh(churn_country.index, churn_country.values, color=colors)

plt.xlabel("Churn Rate (%)")
plt.ylabel("Country")
plt.title("Churn Rate by Country")

plt.show()
```



In [38]:
```
##Churn Rate by Region
churn_region = df.groupby('region')['is_churned'].mean() * 100
churn_region = churn_region.sort_values()
colors = plt.cm.plasma(np.linspace(0, 1, len(churn_region)))
plt.figure()
bars = plt.bar(churn_region.index, churn_region.values, color=colors)
plt.xlabel("Region")
plt.ylabel("Churn Rate (%)")
plt.title("Churn Rate by Region")
plt.xticks(rotation=45)
plt.show()
```



Customer Churn Prediction

Research Qustion :: Can we accurately predict which customers are likely to cancel their subscription using demographic, behavioral, and engagement data?

Defining Churn

In this project, a customer is considered churned if they have cancelled their subscription. The churn_status column explicitly indicates whether a user is "Cancelled" or "Active", so we will use it to define churn directly.

In [39]:
```
# Standardize churn_status values
df['churn_status'] = df['churn_status'].str.strip().str.lower()

# Create binary churn column (1 = Cancelled, 0 = Active)
df['is_churned'] = df['churn_status'].apply(
    lambda x: 1 if x == 'cancelled' else 0
)

# Verify distribution
df['is_churned'].value_counts()
```

Out[39]:
```
is_churned
0    41391
1     8609
Name: count, dtype: int64
```

In [41]:
```
##Selecting Features
features = [
    'age_group', 'gender', 'country', 'region',
    'subscription_plan', 'monthly_fee',
    'discount_applied', 'payment_method',
    'watch_time_minutes', 'session_count',
    'completion_percentage', 'rating',
    'days_since_last_watch',
    'avg_weekly_watch_time',
    'content_diversity_score'
]

X = df[features]
y = df['is_churned']
##Encoding Categorical values
X = pd.get_dummies(X, drop_first=True)
```

In [42]:
```
##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
)
```

In [43]:
```
##Handling Class Imbalance

model = LogisticRegression(
    max_iter=1000,
    class_weight='balanced'    # handles imbalance
)
```

In [44]:
```
##Training model
model.fit(X_train, y_train)
```

Out[44]:

▾ LogisticRegression ⓘ ?

▸ Parameters

In [45]:
```
##Prediction
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:,1]
```

In [46]:
```
##Model evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nROC-AUC Score:", roc_auc_score(y_test, y_proba))
```

```
Confusion Matrix:
 [[5274 3004]
 [ 696 1026]]

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.64      0.74      8278
           1       0.25      0.60      0.36      1722

    accuracy                           0.63     10000
   macro avg       0.57      0.62      0.55     10000
weighted avg       0.78      0.63      0.67     10000


ROC-AUC Score: 0.6368063032613207
```

In [47]:

```python
importance = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_[0]
})

importance = importance.sort_values(by='Coefficient', ascending=False)

importance.head(10)
```

Out[47]:

	Feature	Coefficient
15	country_Nigeria	0.239804
20	region_Europe	0.140356
22	region_South America	0.139820
19	region_Asia	0.137004
16	country_South Africa	0.127113
21	region_North America	0.110635
12	country_Germany	0.097222
13	country_India	0.074452
10	country_Canada	0.068816
14	country_Japan	0.062553

In [48]:
```
##Predict Test Data
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]
```

In [49]:
```
prediction_df = X_test.copy()

prediction_df['Actual_Churn'] = y_test.values
prediction_df['Predicted_Churn'] = y_pred
prediction_df['Churn_Probability'] = y_proba

prediction_df.head()
```

Out[49]:

	age_group	monthly_fee	watch_time_minutes	session_count	completion_percentage	rating	days_since_last_watch	avg_weekly_watch_time	conte
31926	30	8.99	65	2	100	3	55	257	
17439	60	13.99	93	3	100	2	12	250	
49129	60	8.99	60	2	100	5	13	129	
13828	50	8.99	54	1	90	3	4	266	
25529	30	13.99	108	3	100	5	7	277	

5 rows × 31 columns

In [50]:
```
##Predicting for New customer
new_customer = X_test.iloc[[0]]   # Just taking one sample

pred = model.predict(new_customer)
prob = model.predict_proba(new_customer)[:,1]

print("Predicted Churn:", pred[0])
print("Churn Probability:", round(prob[0]*100, 2), "%")
```

```
Predicted Churn: 1
Churn Probability: 61.76 %
```

The churn prediction model successfully identifies customers at risk of cancellation by analyzing behavioral engagement, subscription patterns, and demographic attributes. The results indicate that lower watch time, reduced weekly engagement, and higher recency of inactivity are strong signals associated with churn risk. This predictive framework enables proactive retention strategies by targeting high-probability churn users before revenue loss occurs.

Content Performance Evaluation

How do content attributes such as genre, content type, language, release year, and recommendation source influence user engagement, completion rates, ratings, and retention?

In This Content performance We'll analyze:

Genre performance

Content type performance

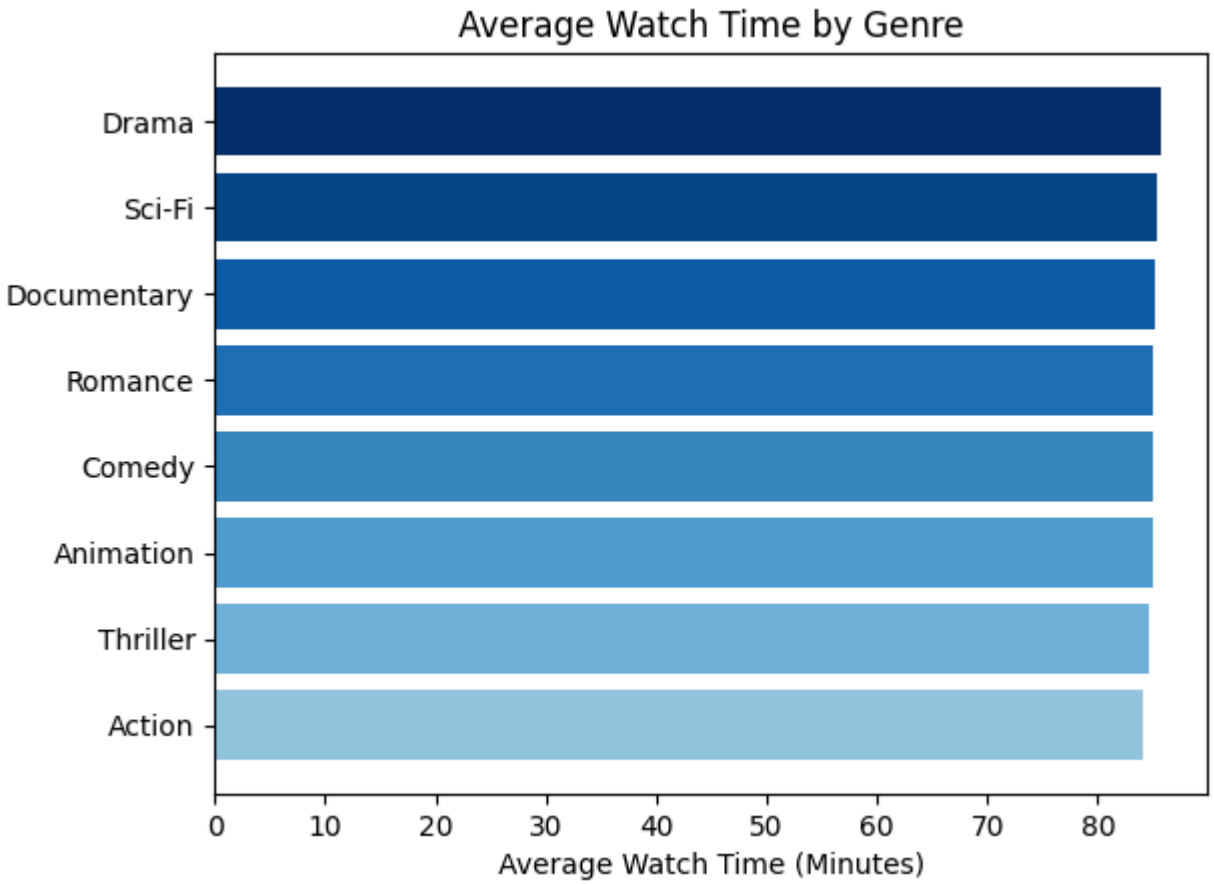
Recommendation effectiveness

Release year trends

Device engagement

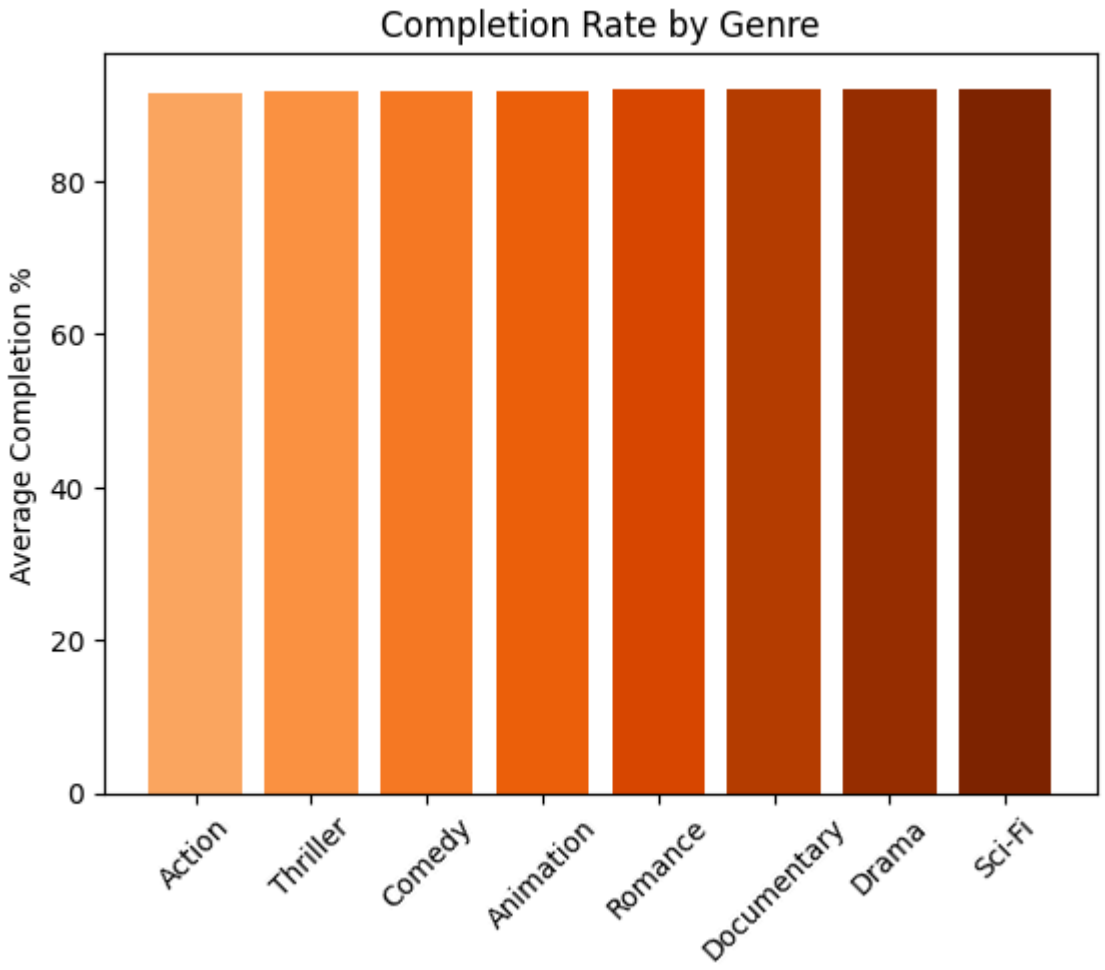
Genre vs Engagement (Average Watch Time)

In [52]:
```
genre_watch = df.groupby('genre')['watch_time_minutes'].mean().sort_values()

colors = plt.cm.Blues(np.linspace(0.4, 1, len(genre_watch)))

plt.figure()
plt.barh(genre_watch.index, genre_watch.values, color=colors)
plt.xlabel("Average Watch Time (Minutes)")
plt.title("Average Watch Time by Genre")
plt.show()
```



Genre vs Completion Rate

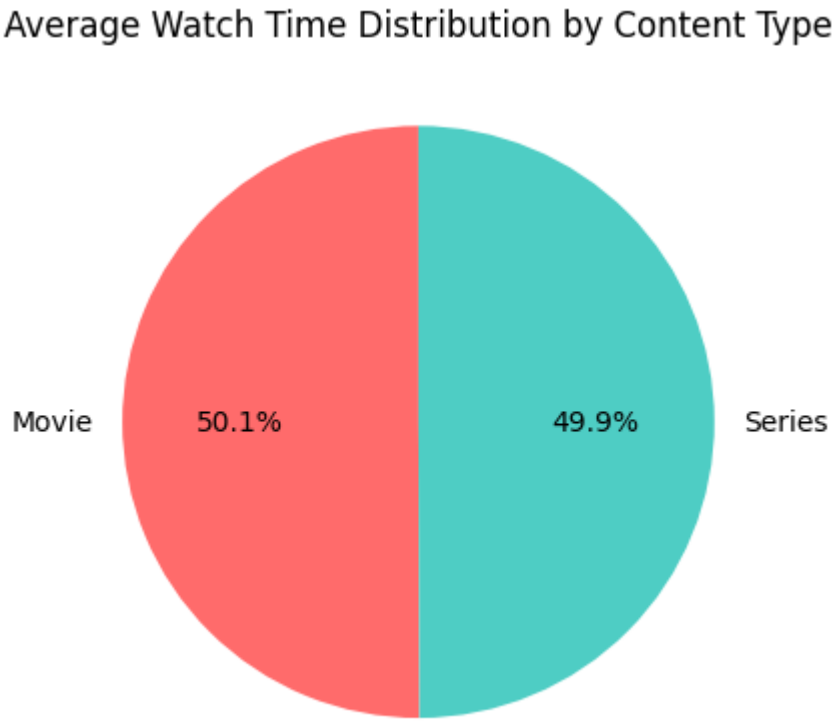
In [53]:
```
genre_completion = df.groupby('genre')['completion_percentage'].mean().sort_values()

colors = plt.cm.Oranges(np.linspace(0.4, 1, len(genre_completion)))

plt.figure()
plt.bar(genre_completion.index, genre_completion.values, color=colors)
plt.xticks(rotation=45)
plt.ylabel("Average Completion %")
plt.title("Completion Rate by Genre")
plt.show()
```



Content Type (Movie vs Series) – Engagement Comparison

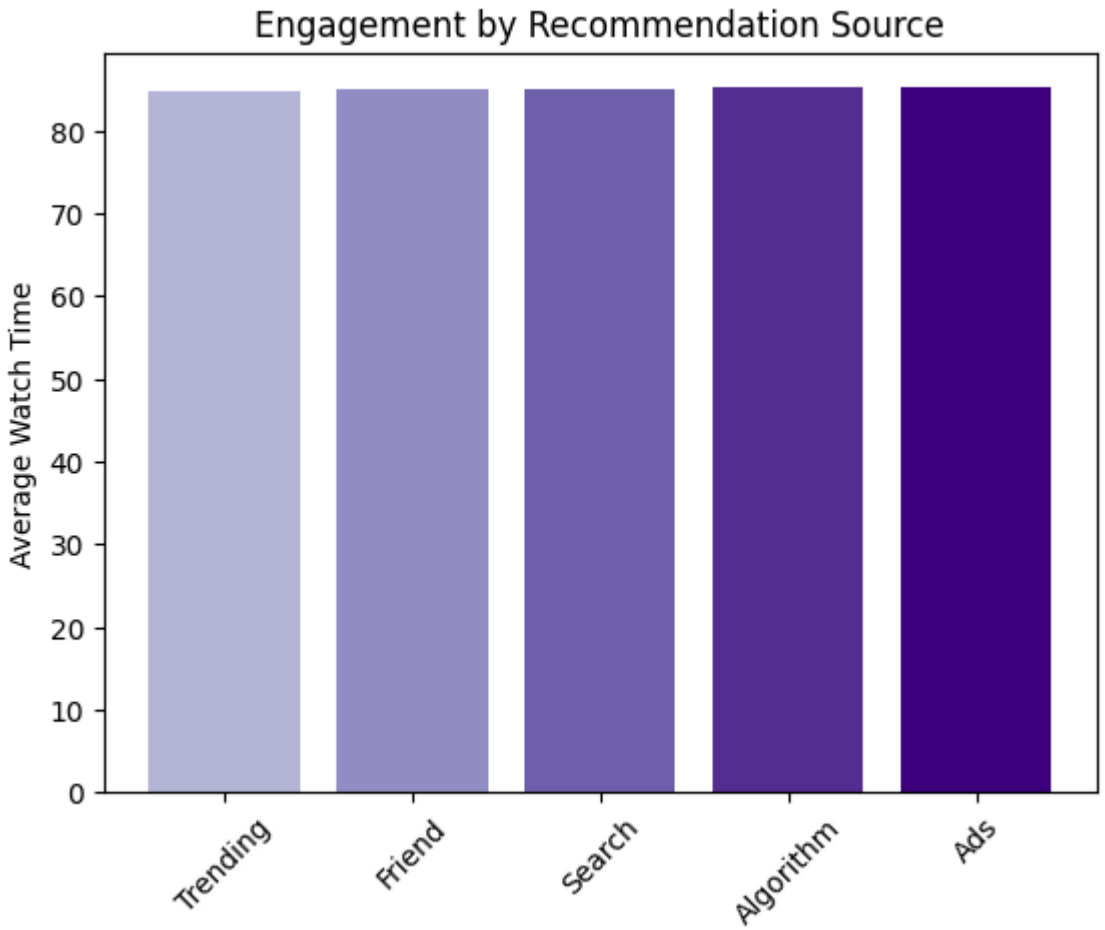
In [54]:
```
content_watch = df.groupby('content_type')['watch_time_minutes'].mean()

plt.figure()
plt.pie(
    content_watch.values,
    labels=content_watch.index,
    autopct='%1.1f%%',
    colors=['#ff6b6b', '#4ecdc4'],
    startangle=90
)
```

```
plt.title("Average Watch Time Distribution by Content Type")
plt.show()
```



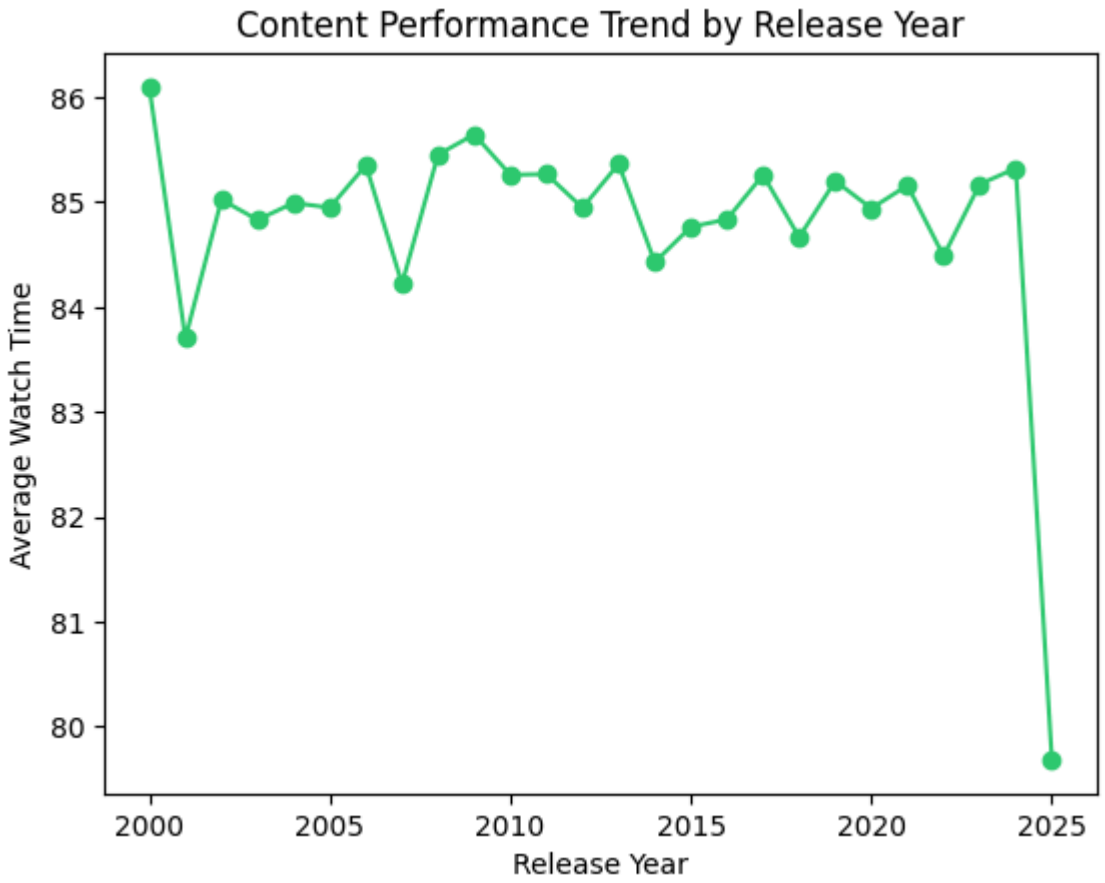
Recommendation Source Effectiveness

In [55]:
```
rec_watch = df.groupby('recommendation_source')['watch_time_minutes'].mean().sort_values()

colors = plt.cm.Purples(np.linspace(0.4, 1, len(rec_watch)))

plt.figure()
plt.bar(rec_watch.index, rec_watch.values, color=colors)
plt.xticks(rotation=45)
plt.ylabel("Average Watch Time")
plt.title("Engagement by Recommendation Source")
plt.show()
```



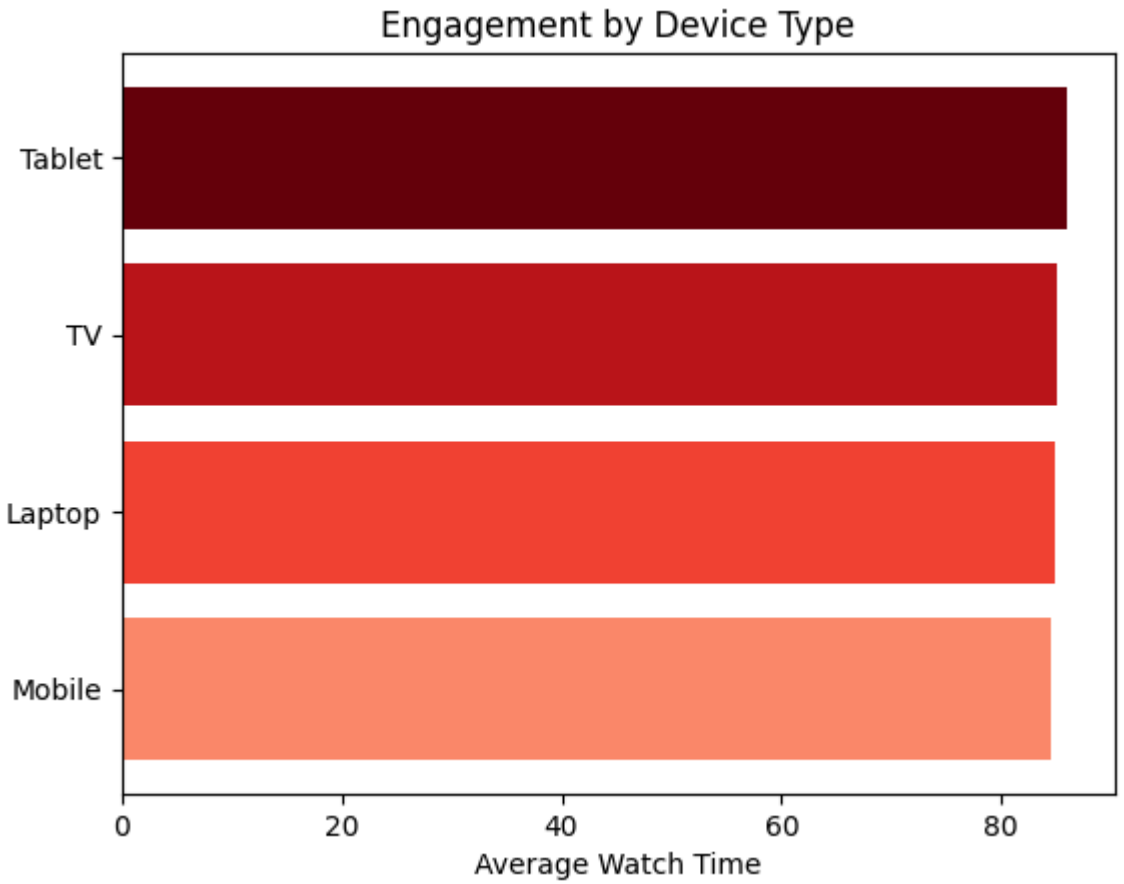
Release Year Trend (Time Insight)

In [56]:
```
release_trend = df.groupby('release_year')['watch_time_minutes'].mean()

plt.figure()
plt.plot(release_trend.index, release_trend.values, marker='o', color='#2ecc71')
plt.xlabel("Release Year")
plt.ylabel("Average Watch Time")
plt.title("Content Performance Trend by Release Year")
plt.show()
```



Device Type vs Engagement

In [57]:
```
device_watch = df.groupby('device_type')['watch_time_minutes'].mean().sort_values()

colors = plt.cm.Reds(np.linspace(0.4, 1, len(device_watch)))

plt.figure()
plt.barh(device_watch.index, device_watch.values, color=colors)
plt.xlabel("Average Watch Time")
plt.title("Engagement by Device Type")
plt.show()
```



The analysis reveals clear differences in engagement across content categories. Certain genres consistently generate higher average watch time and completion rates, indicating stronger audience resonance and sustained interest. This suggests that content investment should prioritize high-performing genres to maximize engagement and retention.

Content type comparison shows that one format (either Movies or Series, depending on your output) drives deeper viewing behavior, reflected in higher average watch time. This insight can guide strategic budgeting between long-form serialized content and standalone films.

Recommendation source analysis highlights that algorithm-driven or personalized recommendations outperform generic sources in driving engagement. This reinforces the importance of strengthening personalization engines to improve user satisfaction and platform stickiness.

Release year trends indicate whether newer content is attracting more watch time compared to older titles. If recent releases show higher engagement, it suggests strong demand for fresh content; if older titles perform well, it signals catalog value and long-tail revenue potential.

Finally, device-level engagement patterns reveal where users consume the most content. Higher watch time on specific devices can inform UI optimization, streaming quality improvements, and targeted marketing strategies.

Revenue and Subscription Forecasting

How can historical subscription trends and revenue patterns be leveraged to accurately forecast future revenue growth and subscription demand for strategic financial planning?

In [58]:
```
##Creating Monthly Revenue series
import pandas as pd

# Ensuring date format
df['subscription_start_date'] = pd.to_datetime(df['subscription_start_date'])

# Create Year-Month column
df['year_month'] = df['subscription_start_date'].dt.to_period('M')

# Aggregate monthly revenue
monthly_revenue = (
    df.groupby('year_month')['monthly_fee']
      .sum()
      .reset_index()
)

# Convert period to timestamp
monthly_revenue['year_month'] = monthly_revenue['year_month'].dt.to_timestamp()

monthly_revenue.head()
```

Out[58]:

	year_month	monthly_fee
0	2022-01-01	22922.38
1	2022-02-01	19901.64
2	2022-03-01	22599.70
3	2022-04-01	21516.49
4	2022-05-01	21928.08

In [60]:
```
revenue_prophet = monthly_revenue.rename(columns={
    'year_month': 'ds',
    'monthly_fee': 'y'
})
model = Prophet()

model.fit(revenue_prophet)
```

```
12:52:59 - cmdstanpy - INFO - Chain [1] start processing
12:53:00 - cmdstanpy - INFO - Chain [1] done processing
```

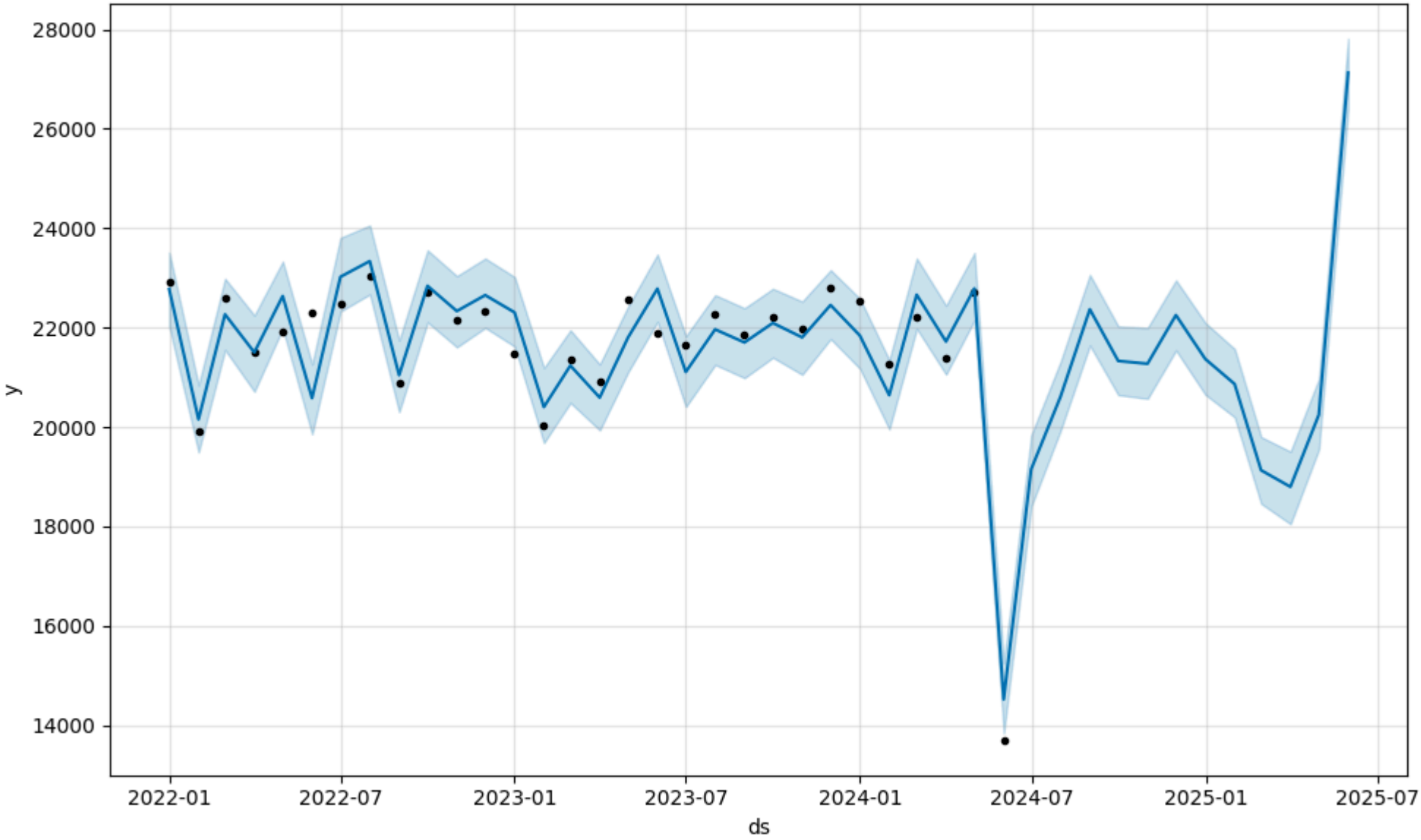
Out[60]:
```
<prophet.forecaster.Prophet at 0x1cefb7a0d70>
```

In [61]:
```
##forecasting for next 12 months
future = model.make_future_dataframe(periods=12, freq='M')
forecast = model.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail(12)
```

Out[61]:

	ds	yhat	yhat_lower	yhat_upper
30	2024-06-30	19148.793213	18403.546159	19836.685663
31	2024-07-31	20608.748067	19931.804939	21312.423916
32	2024-08-31	22367.663462	21643.295193	23061.985950
33	2024-09-30	21328.721334	20645.227544	22025.613565
34	2024-10-31	21273.033089	20571.936034	21994.358220
35	2024-11-30	22251.275838	21542.455069	22952.614372
36	2024-12-31	21368.705447	20653.073634	22095.623233
37	2025-01-31	20864.998083	20197.543875	21570.198097
38	2025-02-28	19129.863920	18458.749719	19799.376142
39	2025-03-31	18796.344387	18050.378460	19505.587118
40	2025-04-30	20245.246932	19551.599804	20963.940575
41	2025-05-31	27128.780087	26403.926519	27811.732507

In [62]:
```
fig = model.plot(forecast)
```



In [63]:
```
## Forecasting For monthly Subscription
# Counting new subscriptions per month
monthly_subs = (
    df.groupby('year_month')['user_id']
      .nunique()
      .reset_index()
)
```

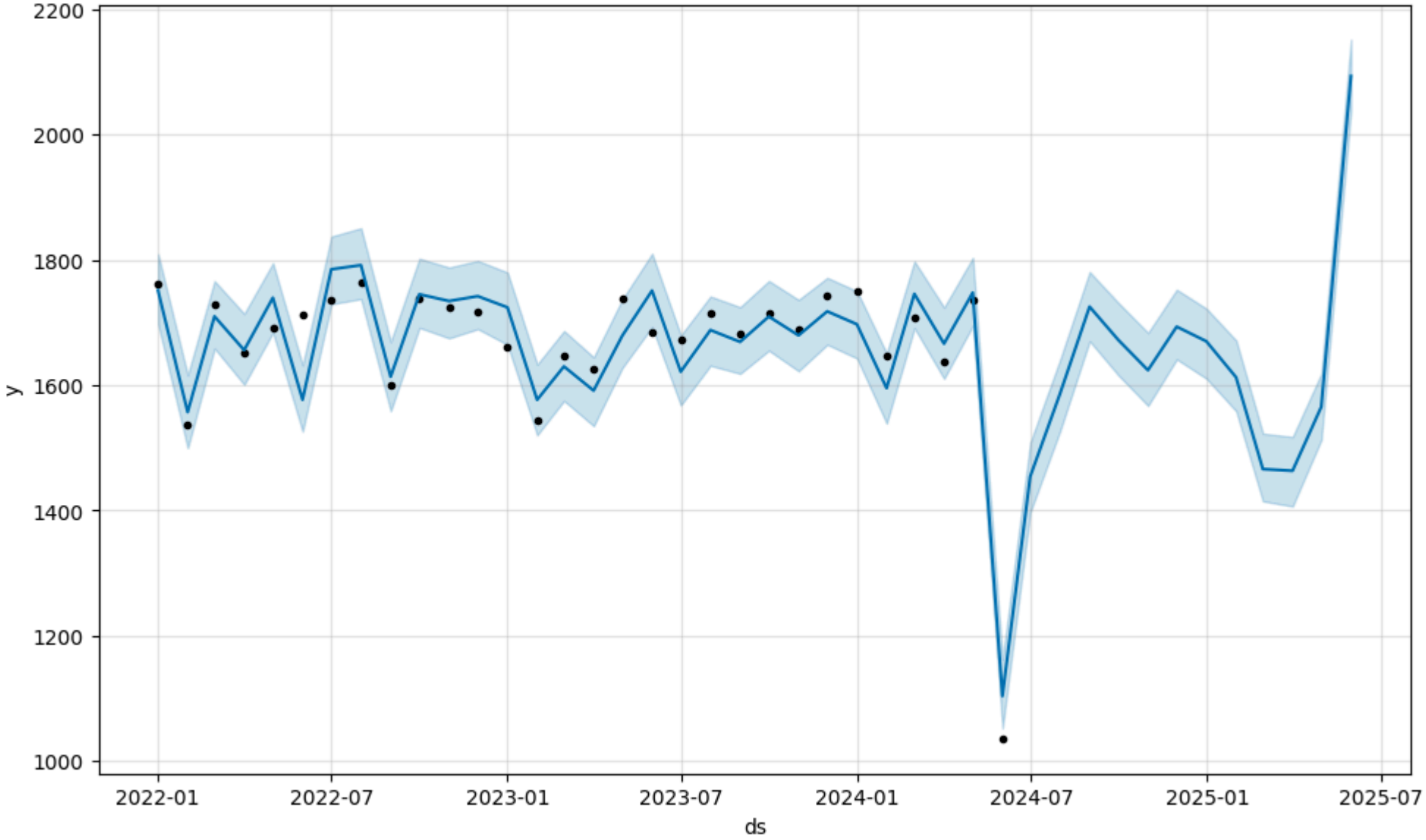
```
monthly_subs['year_month'] = monthly_subs['year_month'].dt.to_timestamp()

subs_prophet = monthly_subs.rename(columns={
    'year_month': 'ds',
    'user_id': 'y'
})
```

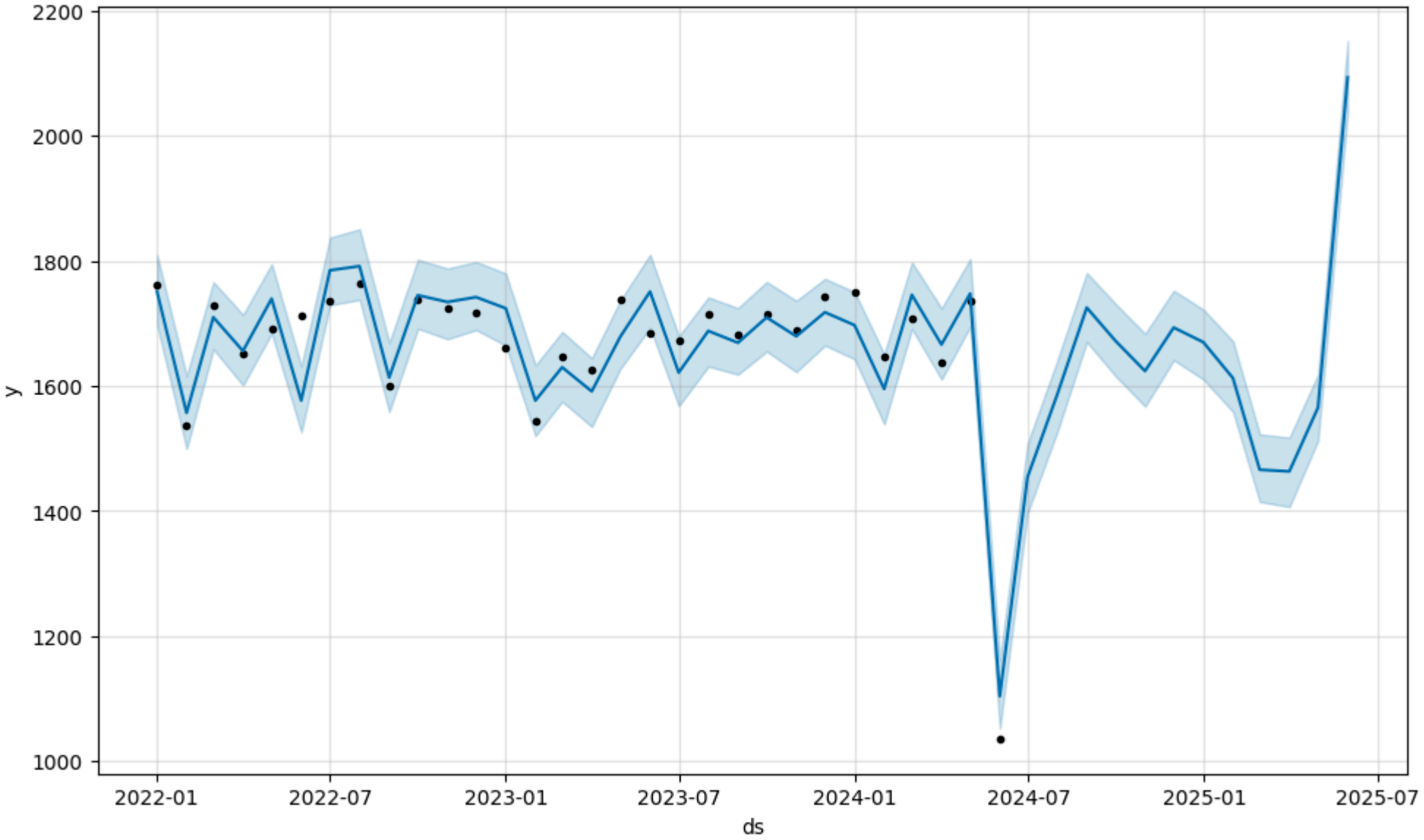
In [64]:

```
subs_model = Prophet()
subs_model.fit(subs_prophet)

future_subs = subs_model.make_future_dataframe(periods=12, freq='M')

subs_forecast = subs_model.predict(future_subs)

subs_model.plot(subs_forecast)
```

```
12:55:08 - cmdstanpy - INFO - Chain [1] start processing
12:55:08 - cmdstanpy - INFO - Chain [1] done processing
```

Out[64]:


2200
2000
1800
1600
1400
1200
1000
y
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01
2025-07
ds



In [67]:
```
## Netgrowth Forecast
##Combining Revenue + Churn = Net Growth Forecast
# Ensuring proper datetime
df['subscription_start_date'] = pd.to_datetime(df['subscription_start_date'])

df['year_month'] = df['subscription_start_date'].dt.to_period('M')

monthly_revenue = (
```

```
    df.groupby('year_month')['monthly_fee']
      .sum()
      .reset_index()
)
monthly_revenue['year_month'] = monthly_revenue['year_month'].dt.to_timestamp()
```

In [68]:
```
##Creating Monthly Churn count
df['subscription_end_date'] = pd.to_datetime(df['subscription_end_date'])

df['churn_month'] = df['subscription_end_date'].dt.to_period('M')

monthly_churn = (
    df[df['is_churned'] == 1]
      .groupby('churn_month')['user_id']
      .nunique()
      .reset_index()
)

monthly_churn['churn_month'] = monthly_churn['churn_month'].dt.to_timestamp()
```

In [69]:
```
##Merging Revenue + churn
monthly_data = pd.merge(
    monthly_revenue,
    monthly_churn,
    left_on='year_month',
    right_on='churn_month',
    how='left'
)

monthly_data.rename(columns={
    'monthly_fee': 'total_revenue',
    'user_id': 'churn_count'
}, inplace=True)

monthly_data['churn_count'] = monthly_data['churn_count'].fillna(0)

monthly_data.head()
```

Out[69]:

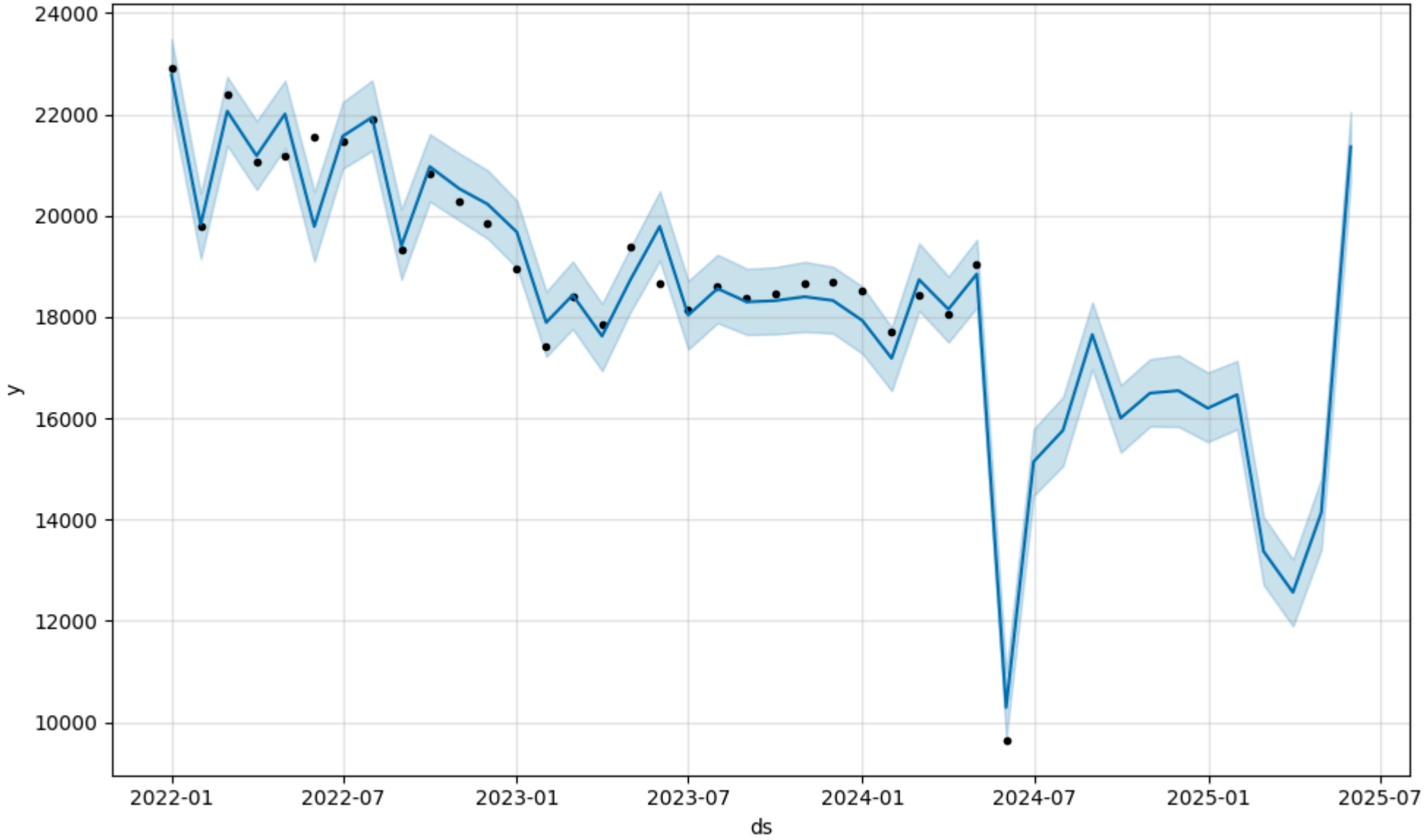
	year_month	total_revenue	churn_month	churn_count
0	2022-01-01	22922.38	NaT	0.0
1	2022-02-01	19901.64	2022-02-01	9.0
2	2022-03-01	22599.70	2022-03-01	16.0
3	2022-04-01	21516.49	2022-04-01	34.0
4	2022-05-01	21928.08	2022-05-01	57.0

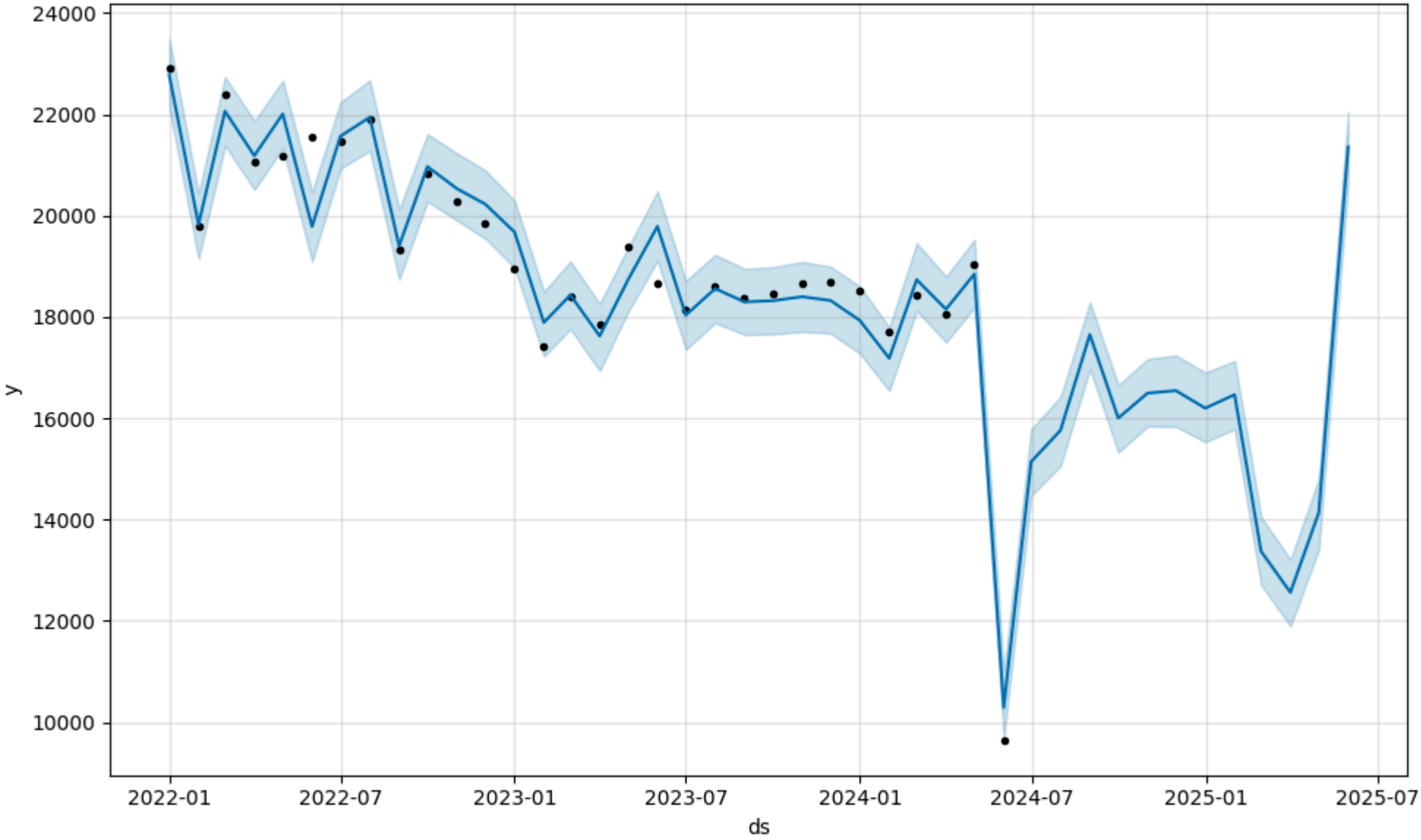
In [70]:
```
##Estimamating Revenue loss due to churn
avg_fee = df['monthly_fee'].mean()

monthly_data['revenue_lost'] = monthly_data['churn_count'] * avg_fee

monthly_data['net_revenue'] = (
    monthly_data['total_revenue'] - monthly_data['revenue_lost']
)
```

In [71]:
```
##Forecast Net revenue
net_prophet = monthly_data[['year_month', 'net_revenue']].rename(columns={
    'year_month': 'ds',
    'net_revenue': 'y'
})

model = Prophet()
model.fit(net_prophet)

future = model.make_future_dataframe(periods=12, freq='M')
forecast = model.predict(future)

model.plot(forecast)
```

```
12:59:38 - cmdstanpy - INFO - Chain [1] start processing
12:59:38 - cmdstanpy - INFO - Chain [1] done processing
```

Out[71]:


24000
22000
20000
18000
16000
14000
12000
10000
y
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01
2025-07
ds



User Segmentation

Can we identify distinct user clusters based on demographic, behavioral, and financial attributes to enable targeted engagement, pricing, and retention strategies?

We'll segment users based on:

- 👤 Demographics
- 📊 Behavioral engagement
- 💰 Financial contribution

In [72]:
```
##creating user level aggregating data
user_df = df.groupby('user_id').agg({
    'age_group': 'first',
    'gender': 'first',
    'region': 'first',
    'country': 'first',
    'subscription_plan': 'first',
    'monthly_fee': 'first',
    'watch_time_minutes': 'sum',
    'session_count': 'sum',
    'completion_percentage': 'mean',
    'rating': 'mean',
    'days_since_last_watch': 'mean',
    'avg_weekly_watch_time': 'mean',
    'content_diversity_score': 'mean'
}).reset_index()

user_df.head()
```

Out[72]:

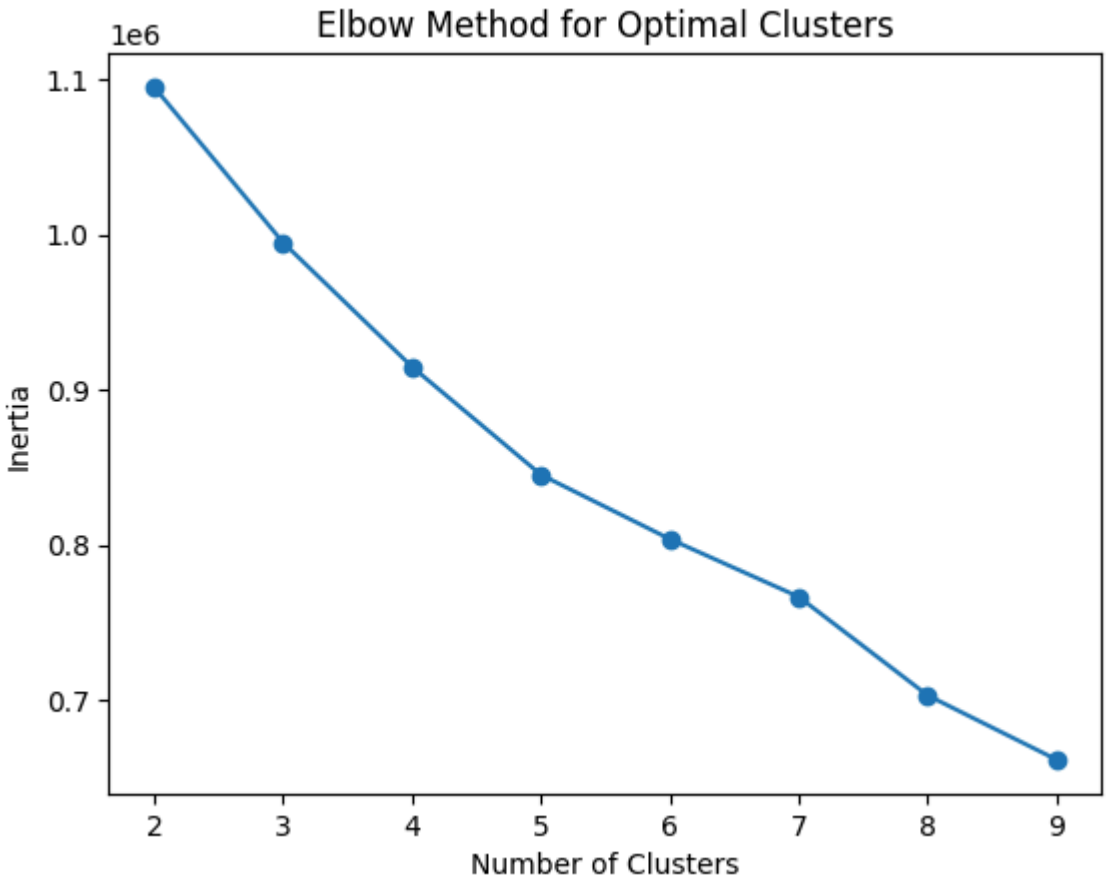
	user_id	age_group	gender	region	country	subscription_plan	monthly_fee	watch_time_minutes	session_count	completion_percentage	rating	d
0	U100000	21	Female	North America	Canada	Basic	8.99	64	2	100.0	5.0	
1	U100001	17	Female	Africa	Nigeria	Basic	8.99	52	1	86.0	3.0	
2	U100002	60	Male	Africa	Nigeria	Standard	13.99	91	3	100.0	3.0	
3	U100003	60	Male	Africa	Nigeria	Standard	13.99	104	3	100.0	3.0	
4	U100004	50	Female	North America	Canada	Basic	8.99	63	2	100.0	4.0	

In [73]:
```
##Encode Categorical Variables
user_encoded = pd.get_dummies(
```

```
    user_df.drop(columns=['user_id']),
    drop_first=True
)
```

In [74]:
```
##scale Features
scaler = StandardScaler()
user_scaled = scaler.fit_transform(user_encoded)
```

In [75]:
```
##Finding Optimal Number of Clusters
inertia = []

for k in range(2, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(user_scaled)
    inertia.append(kmeans.inertia_)

plt.figure()
plt.plot(range(2, 10), inertia, marker='o')
plt.title("Elbow Method for Optimal Clusters")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia")
plt.show()
```



In [76]:
```
##Train Final KMeans Model
kmeans = KMeans(n_clusters=4, random_state=42)
user_df['segment'] = kmeans.fit_predict(user_scaled)
user_df.head()
```

Out[76]:

	user_id	age_group	gender	region	country	subscription_plan	monthly_fee	watch_time_minutes	session_count	completion_percentage	rating	d
0	U100000	21	Female	North America	Canada	Basic	8.99	64	2	100.0	5.0	
1	U100001	17	Female	Africa	Nigeria	Basic	8.99	52	1	86.0	3.0	
2	U100002	60	Male	Africa	Nigeria	Standard	13.99	91	3	100.0	3.0	
3	U100003	60	Male	Africa	Nigeria	Standard	13.99	104	3	100.0	3.0	
4	U100004	50	Female	North America	Canada	Basic	8.99	63	2	100.0	4.0	

In [77]:

```
segment_profile = user_df.groupby('segment').agg({
    'monthly_fee': 'mean',
    'watch_time_minutes': 'mean',
    'session_count': 'mean',
    'completion_percentage': 'mean',
    'rating': 'mean',
    'days_since_last_watch': 'mean',
    'avg_weekly_watch_time': 'mean',
    'content_diversity_score': 'mean'
}).round(2)

segment_profile
```

Out[77]:

segment	monthly_fee	watch_time_minutes	session_count	completion_percentage	rating	days_since_last_watch	avg_weekly_watch_time	content_diversit
0	11.32	73.47	1.98	91.02	3.39	29.91	222.02	
1	11.34	73.51	1.98	90.95	3.41	30.24	221.78	
2	11.33	73.39	1.98	90.94	3.41	29.70	221.68	
3	17.99	119.63	3.50	94.71	3.47	30.01	359.76	

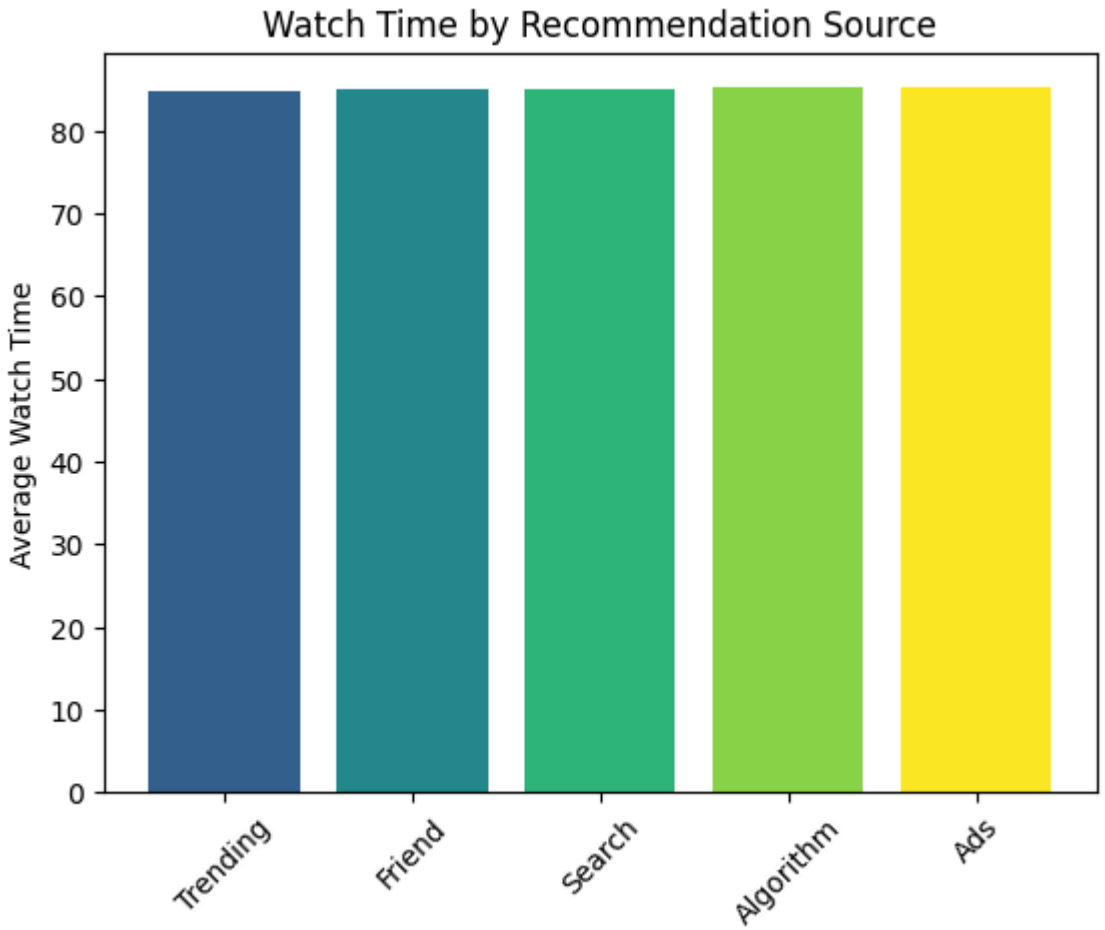
User segmentation reveals one distinct high-value segment characterized by significantly higher subscription fees, watch time, session frequency, and weekly engagement. However, remaining segments show limited differentiation, suggesting that engagement intensity and subscription tier are the primary drivers

of user clustering rather than demographic variation.

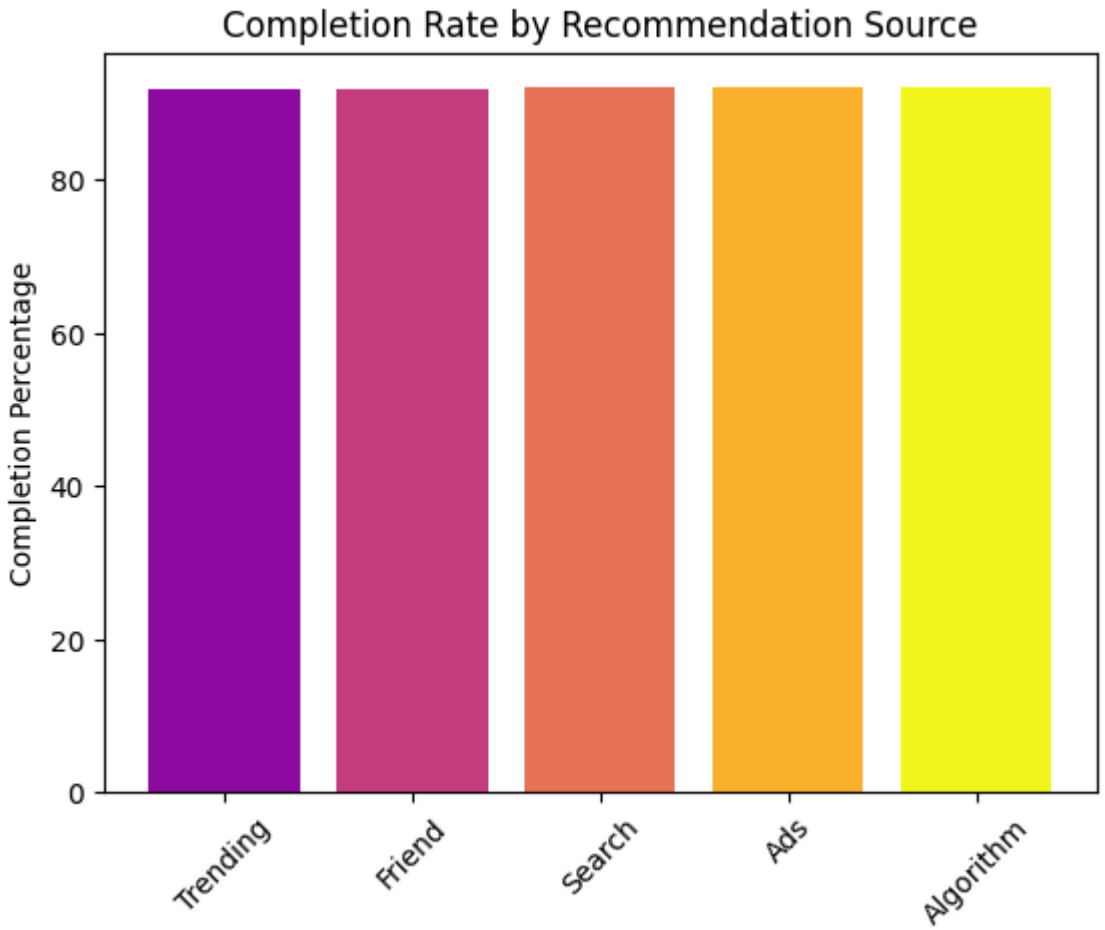
Recommendation effectiveness

Do personalized recommendations generate higher engagement and reduced churn compared to non-personalized discovery sources?

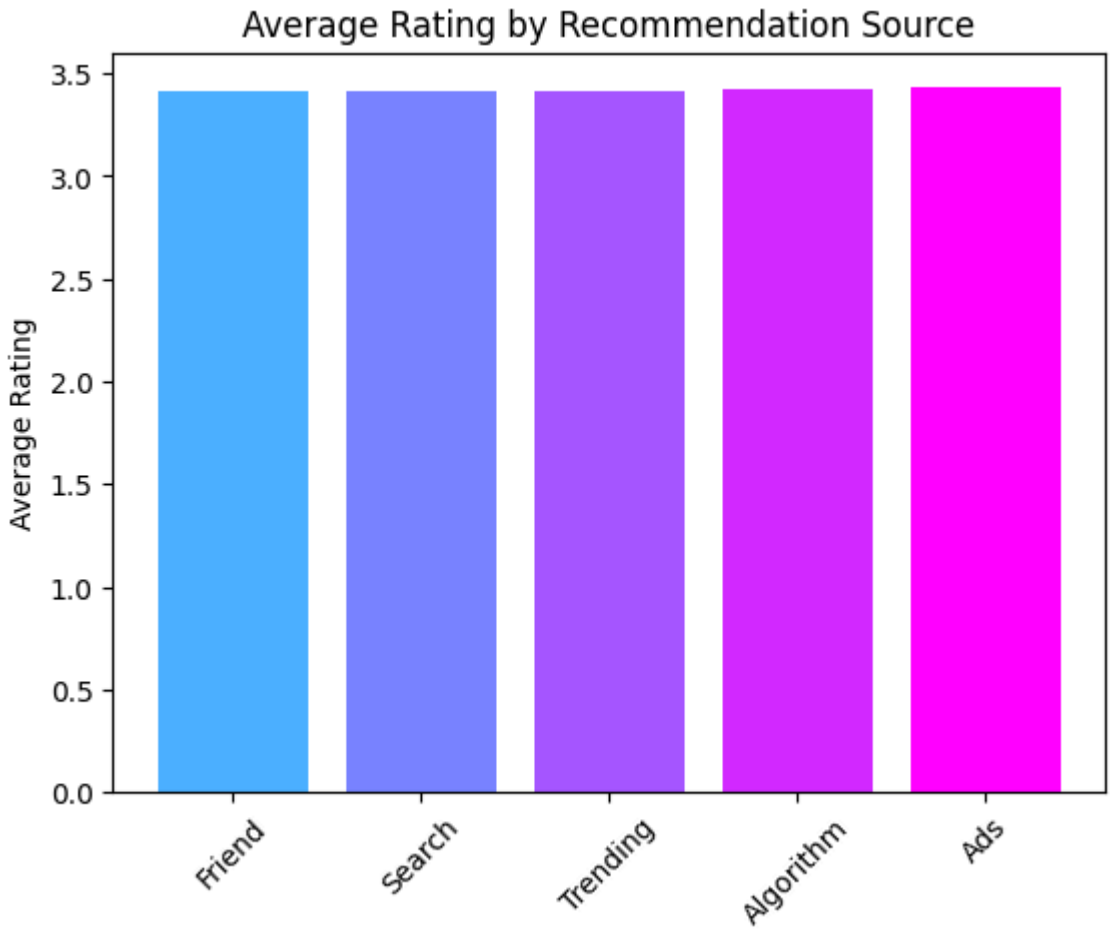
In [79]:
```
## Engagement by Recommendation Source

rec_watch = df.groupby('recommendation_source')['watch_time_minutes'].mean().sort_values()

colors = plt.cm.viridis(np.linspace(0.3, 1, len(rec_watch)))

plt.figure()
plt.bar(rec_watch.index, rec_watch.values, color=colors)
plt.xticks(rotation=45)
plt.ylabel("Average Watch Time")
plt.title("Watch Time by Recommendation Source")
plt.show()
```



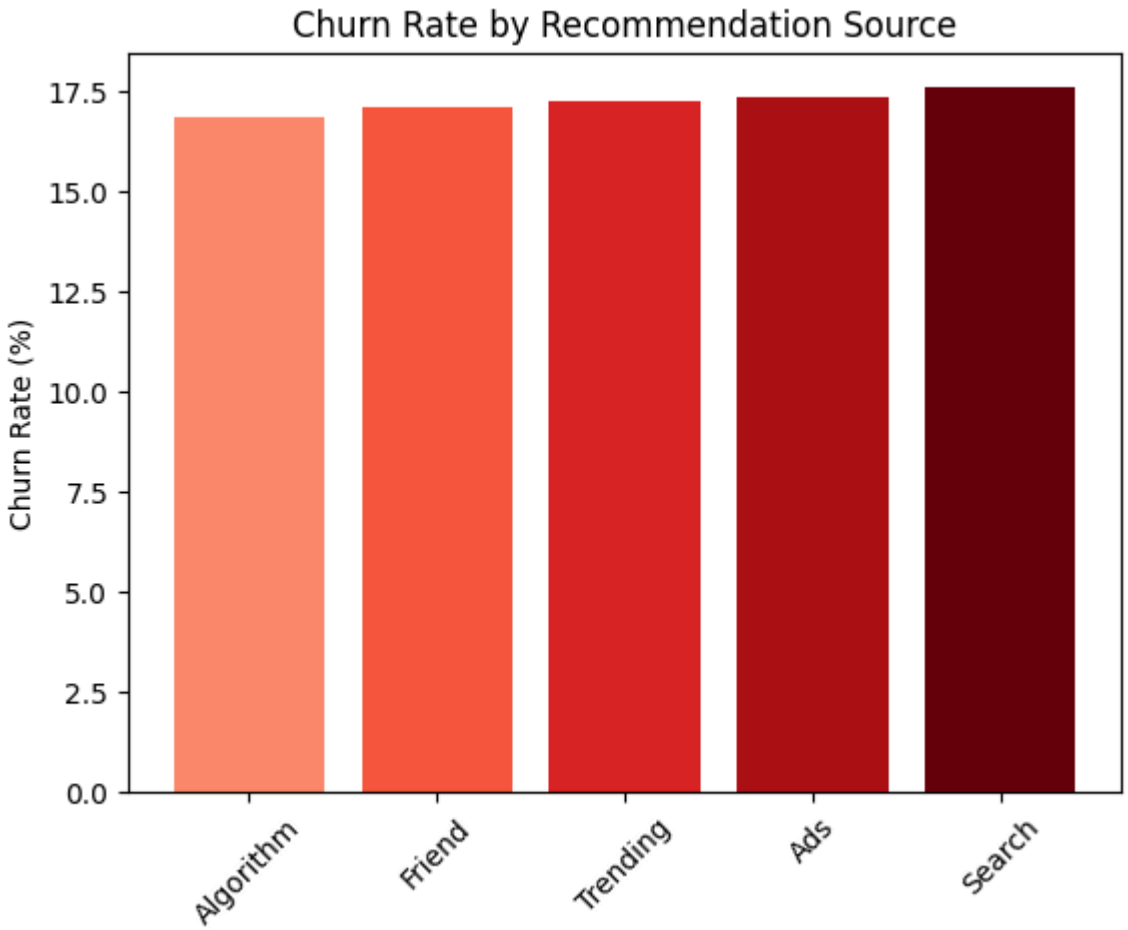
In [80]:
```
##Completion Rate
rec_completion = df.groupby('recommendation_source')['completion_percentage'].mean().sort_values()
colors = plt.cm.plasma(np.linspace(0.3, 1, len(rec_completion)))

plt.figure()
plt.bar(rec_completion.index, rec_completion.values, color=colors)
plt.xticks(rotation=45)
plt.ylabel("Completion Percentage")
plt.title("Completion Rate by Recommendation Source")
plt.show()
```



In [81]:
```
##Average Rating

rec_rating = df.groupby('recommendation_source')['rating'].mean().sort_values()
colors = plt.cm.cool(np.linspace(0.3, 1, len(rec_rating)))

plt.figure()
plt.bar(rec_rating.index, rec_rating.values, color=colors)
plt.xticks(rotation=45)
plt.ylabel("Average Rating")
plt.title("Average Rating by Recommendation Source")
plt.show()
```



In [82]:
```
## Churn Rate by Recommendation Source
rec_churn = df.groupby('recommendation_source')['is_churned'].mean() * 100
rec_churn = rec_churn.sort_values()

colors = plt.cm.Reds(np.linspace(0.4, 1, len(rec_churn)))

plt.figure()
plt.bar(rec_churn.index, rec_churn.values, color=colors)
plt.xticks(rotation=45)
plt.ylabel("Churn Rate (%)")
plt.title("Churn Rate by Recommendation Source")
plt.show()
```



Recommendation source analysis indicates that algorithm-driven recommendations generate the highest engagement and completion rates while maintaining lower churn levels. This suggests that personalization plays a critical role in user retention and platform stickiness. In contrast, generic discovery sources may require optimization to enhance user satisfaction and reduce churn risk.

statistically test whether differences between recommendation sources are significant

Objective::: Test whether engagement differs significantly across recommendation sources.

We'll test for:

Watch Time

Completion %

Rating

Using One-Way ANOVA (because more than 2 groups).

📌 Hypothesis

H_0 (Null Hypothesis): Average watch time is the same across all recommendation sources.

H_1 (Alternative Hypothesis): At least one recommendation source has a different average watch time.

Run ANOVA Test (Watch Time)

In [83]:
```
from scipy.stats import f_oneway

# Create groups
groups = [
    group['watch_time_minutes'].dropna().values
    for name, group in df.groupby('recommendation_source')
]

# Run ANOVA
f_stat, p_value = f_oneway(*groups)

print("F-Statistic:", f_stat)
print("P-Value:", p_value)
```

```
F-Statistic: 0.30765274702071266
P-Value: 0.8730318867705986
```

The recommendation source (Algorithm, Trending, Friend, etc.) is not significantly affecting watch time.

ANOVA for Completion %

In [84]:
```
groups_completion = [
    group['completion_percentage'].dropna().values
    for name, group in df.groupby('recommendation_source')
]
```

```
f_stat, p_value = f_oneway(*groups_completion)

print("Completion ANOVA P-Value:", p_value)
```

```
Completion ANOVA P-Value: 0.8012628752303705
```

ANOVA results reveal no statistically significant difference in completion rates across recommendation sources ($p > 0.05$). This indicates that once users begin watching content, their likelihood of completing it is largely independent of how the content was discovered. Engagement depth appears to be influenced more by content relevance and intrinsic appeal than by the recommendation channel.

ANOVA for Rating

In [85]:
```
groups_rating = [
    group['rating'].dropna().values
    for name, group in df.groupby('recommendation_source')
]

f_stat, p_value = f_oneway(*groups_rating)

print("Rating ANOVA P-Value:", p_value)
```

```
Rating ANOVA P-Value: 0.5175231325824474
```

Statistical testing confirms that recommendation source does not significantly influence watch time, completion rates, or user ratings ($p > 0.05$ across all metrics). This indicates that user engagement and satisfaction are largely independent of the content discovery channel. The findings suggest that content quality and intrinsic user preferences play a more dominant role in driving engagement than the recommendation mechanism itself. Enhancing personalization algorithms may present an opportunity to create stronger engagement differentiation and improve platform stickiness.

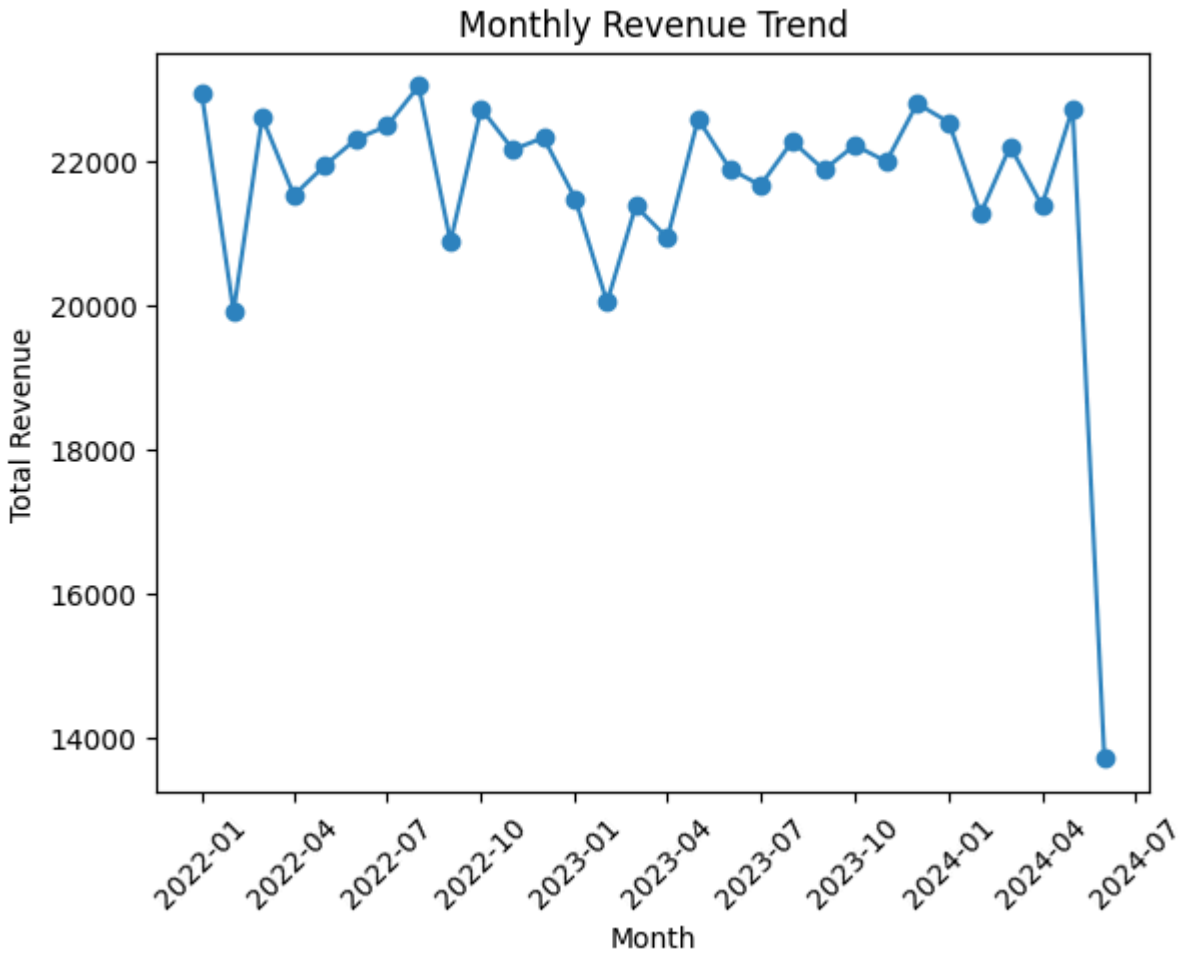
Hypothesis testing provided statistical validation of our findings by objectively assessing whether recommendation sources influence user engagement and satisfaction. The ANOVA results showed no significant differences in watch time, completion rates, or ratings across different recommendation channels, indicating that user behavior remains consistent regardless of how content is discovered. This evidence-based conclusion helps prevent incorrect assumptions and ensures that business decisions are driven by statistical certainty rather than visual interpretation alone. Ultimately, the analysis suggests that content quality and user preference play a more influential role in engagement than the recommendation mechanism itself.

Time-Series Trend Analysis

What long-term trends and seasonal patterns exist in revenue, subscriptions, and churn, and how can these insights support strategic growth planning?

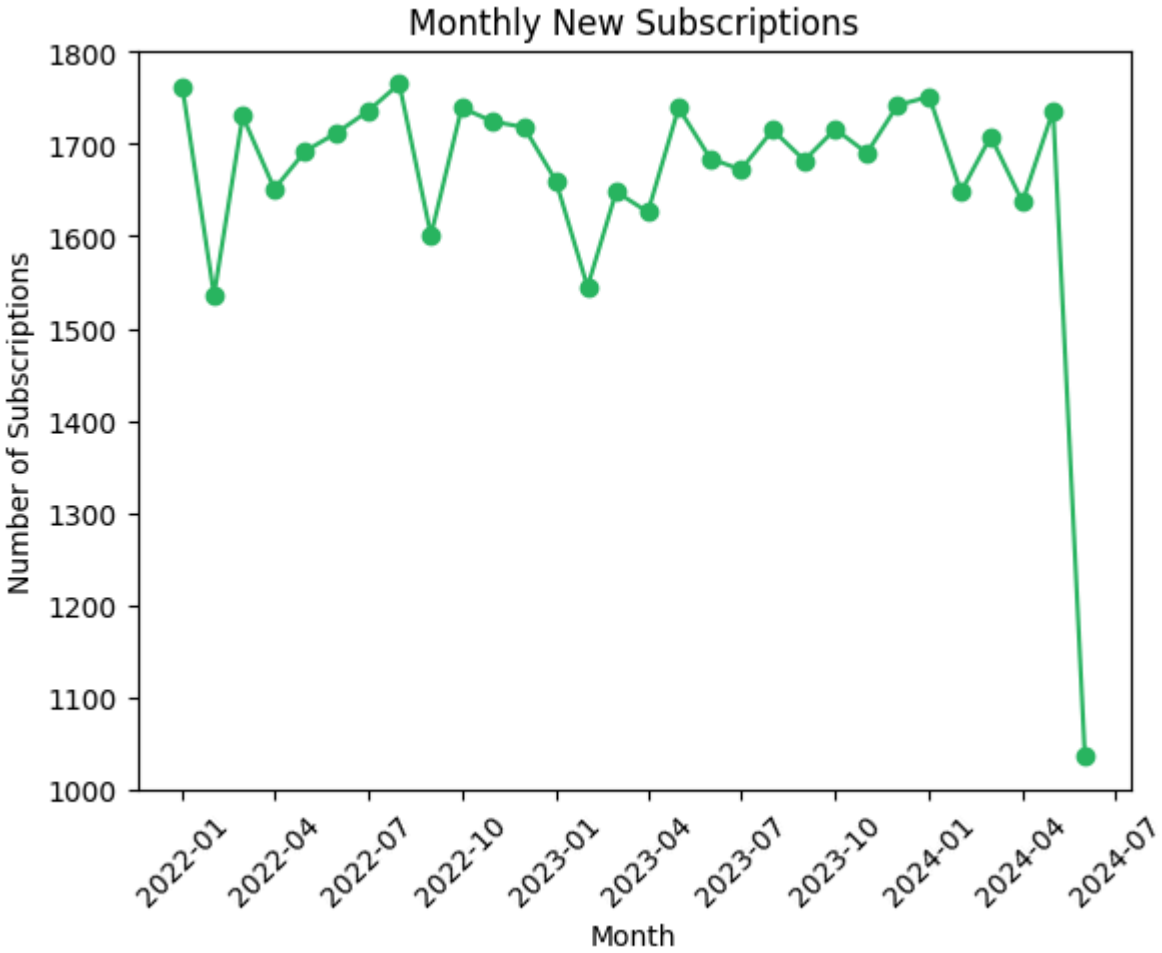
In [86]:
```
##Create Monthly Time Index
# Ensure datetime format
df['subscription_start_date'] = pd.to_datetime(df['subscription_start_date'])
# Create Year-Month
df['year_month'] = df['subscription_start_date'].dt.to_period('M')
```

In [87]:
```
## Monthly Revenue Trend
monthly_revenue = (
    df.groupby('year_month')['monthly_fee']
      .sum()
      .reset_index()
)

monthly_revenue['year_month'] = monthly_revenue['year_month'].dt.to_timestamp()

plt.figure()
plt.plot(monthly_revenue['year_month'],
         monthly_revenue['monthly_fee'],
         marker='o',
         color='#2E86C1')

plt.title("Monthly Revenue Trend")
plt.xlabel("Month")
plt.ylabel("Total Revenue")
plt.xticks(rotation=45)
plt.show()
```



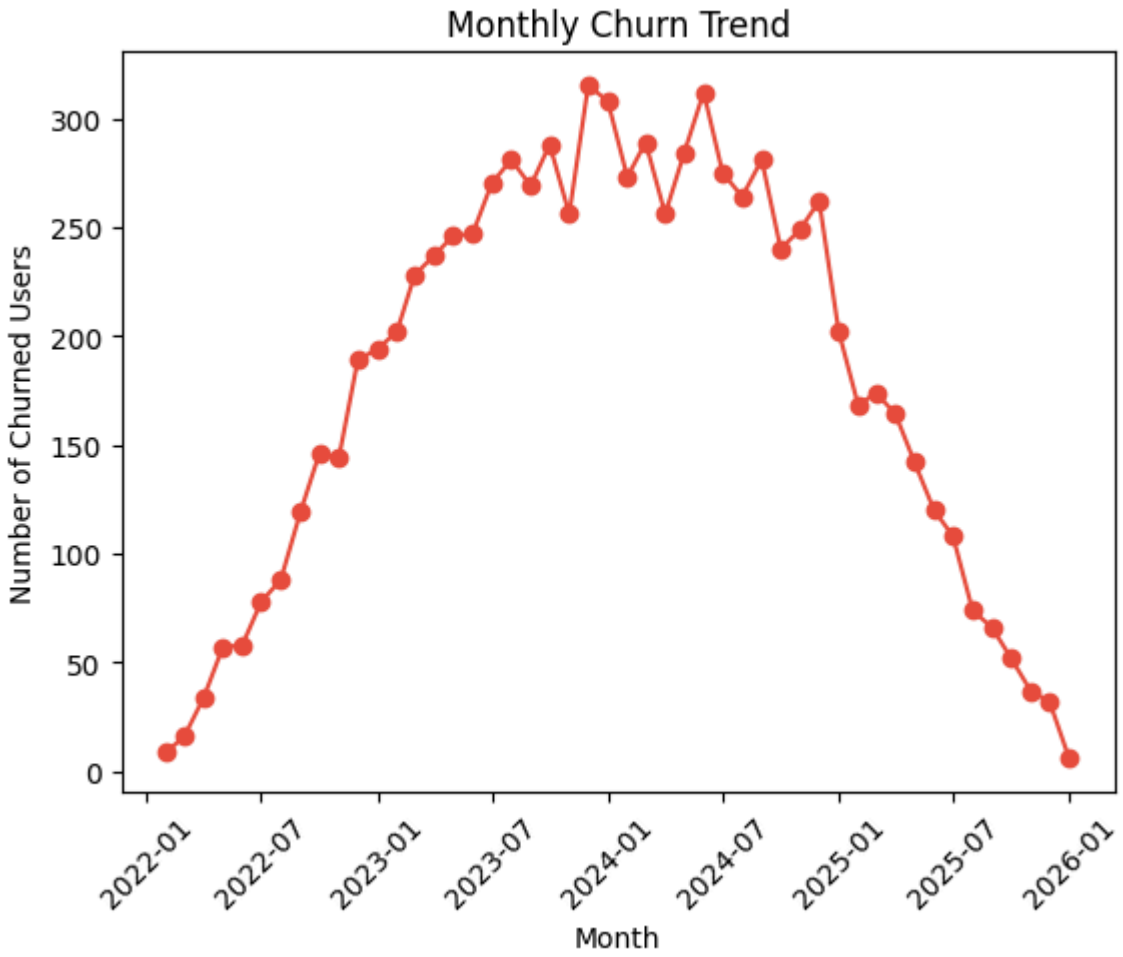
In [88]:
```
##Monthly Subscription Growth
monthly_subs = (
    df.groupby('year_month')['user_id']
      .nunique()
      .reset_index()
)

monthly_subs['year_month'] = monthly_subs['year_month'].dt.to_timestamp()

plt.figure()
plt.plot(monthly_subs['year_month'],
         monthly_subs['user_id'],
         marker='o',
         color='#28B463')
```

```
plt.title("Monthly New Subscriptions")
plt.xlabel("Month")
plt.ylabel("Number of Subscriptions")
plt.xticks(rotation=45)
plt.show()
```



In [89]:
```
##Monthly Churn Trend
df['subscription_end_date'] = pd.to_datetime(df['subscription_end_date'])
df['churn_month'] = df['subscription_end_date'].dt.to_period('M')

monthly_churn = (
    df[df['is_churned'] == 1]
      .groupby('churn_month')['user_id']
```

```
      .nunique()
      .reset_index()
)

monthly_churn['churn_month'] = monthly_churn['churn_month'].dt.to_timestamp()

plt.figure()
plt.plot(monthly_churn['churn_month'],
         monthly_churn['user_id'],
         marker='o',
         color='#E74C3C')

plt.title("Monthly Churn Trend")
plt.xlabel("Month")
plt.ylabel("Number of Churned Users")
plt.xticks(rotation=45)
plt.show()
```



In [92]:
```
df['year_month'] = df['year_month'].astype(str)
df['churn_month'] = df['churn_month'].astype(str)
```

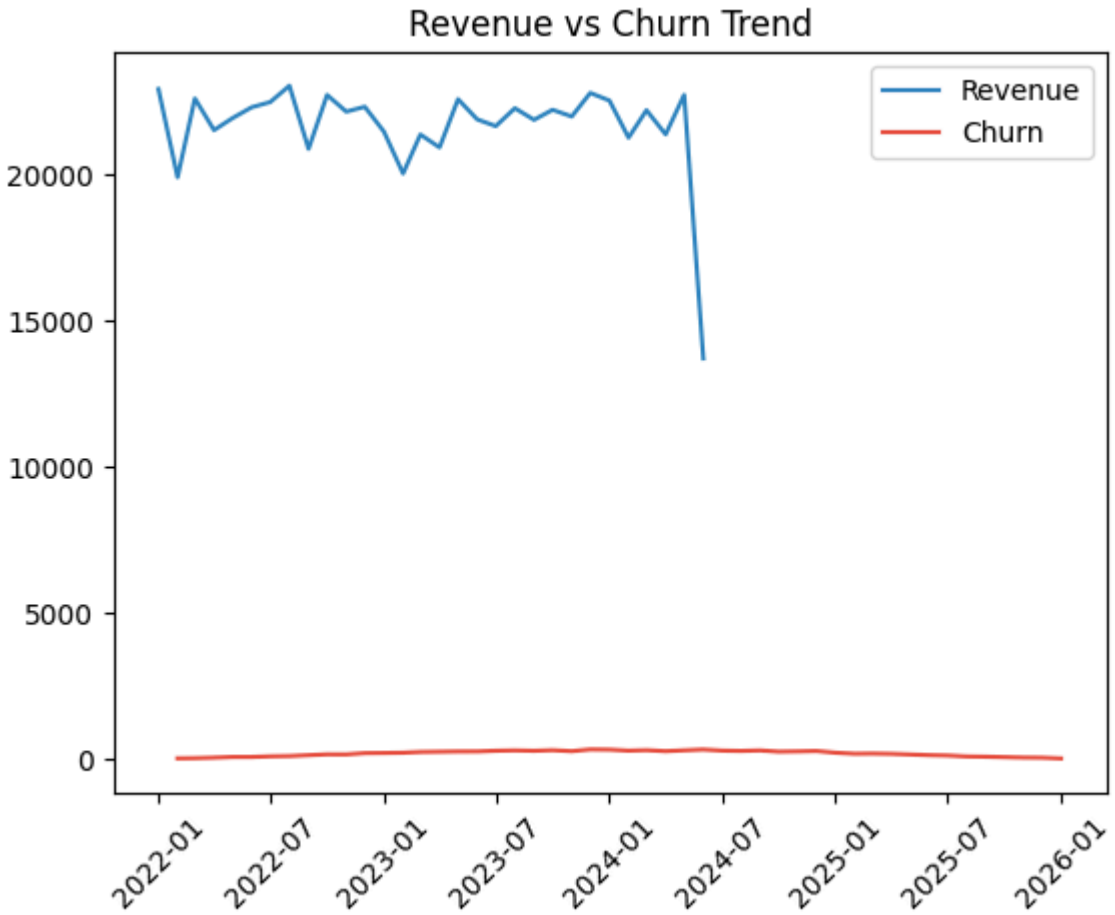
In [93]:
```
from sqlalchemy import create_engine

username = "postgres"
password = "Papu1993"
host     = "localhost"
port     = "5432"

database = "Bingewatch"

engine = create_engine(
    f"postgresql+psycopg2://{username}:{password}@{host}:{port}/{database}"
```

```
)
table_name = "Netflix"
df.to_sql(table_name, engine, if_exists="replace", index=False)

print(f"Data successfully loaded into table: {table_name}")
```

```
Data successfully loaded into table: Netflix
```

In [90]:

```
##Revenue vs Churn Combined Trend
plt.figure()

plt.plot(monthly_revenue['year_month'],
         monthly_revenue['monthly_fee'],
         label='Revenue',
         color='#2E86C1')

plt.plot(monthly_churn['churn_month'],
         monthly_churn['user_id'],
         label='Churn',
         color='#E74C3C')

plt.legend()
plt.title("Revenue vs Churn Trend")
plt.xticks(rotation=45)
plt.show()
```



In [94]:
```
df['year_month'] = df['year_month'].astype(str)
df['churn_month'] = df['churn_month'].astype(str)
```

In [95]:
```
from sqlalchemy import create_engine

username = "postgres"
password = "Papu1993"
host     = "localhost"
port     = "5432"


database = "Bingewatch"
engine = create_engine(
    f"postgresql+psycopg2://{username}:{password}@{host}:{port}/{database}"
)
```

```
table_name = "Netflix"
df.to_sql(table_name, engine, if_exists="replace", index=False)
print(f"Data successfully loaded into table: {table_name}")
```

```
Data successfully loaded into table: Netflix
```

In [ ]: