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

```
Customers = pd.read_csv("/content/Customers.csv")
```

```
Subscriptions = pd.read_csv("/content/Subscription.csv")
```

```
Transactions = pd.read_csv("/content/Transcation.csv")
```

```
Usage = pd.read_csv("/content/Usage.csv")
```

```
merged_df = pd.merge(Customers, Subscriptions, on="CustomerID", how="left")
```

```
merged_df = merged_df.merge(Transactions, on="CustomerID", how="left")
```

```
merged_df = merged_df.merge(Usage, on="CustomerID", how="left")
```

```
# Check for non-datetime values
print(merged_df[["StartDate", "EndDate"]].dtypes)

# Look for rows with invalid dates
print(merged_df[merged_df["StartDate"].isna() | merged_df["EndDate"].isna()])
```

```
68   15.99           Credit       5.0       Watch Movie  2024-02-08
69   15.99           Credit      30.0      Rate Content  2024-02-25

    usage_amount
0            NaN
2          21.23
7          30.51
9          55.69
11         76.66
12         76.66
16           NaN
18         78.63
19         78.63
21         54.42
22         54.42
24         55.69
25         55.69
32         87.98
33         50.84
35           NaN
38         33.81
41         36.39
42         67.21
45           NaN
46           NaN
47           NaN
49           NaN
51           NaN
55           NaN
59         47.97
60         50.43
61         51.60
62         40.62
63         63.91
65         68.99
66         75.87
68         27.90
69         32.63
```

```
print(merged_df.columns)
```

```
Index(['CustomerID', 'Name', 'Age', 'Gender', 'Income', 'Location',
       'SubscriptionID', 'StartDate', 'EndDate', 'Status', 'transaction_id',
       'transaction_date', 'amount', 'transaction_type', 'usage_id',
       'feature_used', 'usage_date', 'usage_amount'],
      dtype='object')
```

```
merged_df["StartDate"] = pd.to_datetime(merged_df["StartDate"], errors="coerce")
merged_df["EndDate"] = pd.to_datetime(merged_df["EndDate"], errors="coerce")
merged_df["tenure"] = (merged_df["EndDate"] - merged_df["StartDate"]).dt.days
```

```
print(merged_df[["StartDate", "EndDate", "tenure"]].head())
```

```
   StartDate    EndDate  tenure
0 2022-01-31        NaT     NaN
1 2022-02-28 2024-11-21   997.0
2 2022-03-31        NaT     NaN
3 2022-04-30 2024-11-21   936.0
4 2022-04-30 2024-11-21   936.0
```

```
# Avoid division by zero by replacing tenure of 0 with NaN
merged_df["tenure"] = merged_df["tenure"].replace(0, np.nan)

# Calculate average monthly spend
merged_df["average_monthly_spend"] = merged_df["amount"] / (merged_df["tenure"] / 30.0)

# Fill NaN values in 'average_monthly_spend' if any tenure is missing or invalid
merged_df["average_monthly_spend"].fillna(0, inplace=True)
```

```
<ipython-input-78-fca80104520c>:8: FutureWarning: A value is trying to be set on a copy
The behavior will change in pandas 3.0. This inplace method will never work because the

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col


  merged_df["average_monthly_spend"].fillna(0, inplace=True)
```

```
print(merged_df.columns)
```

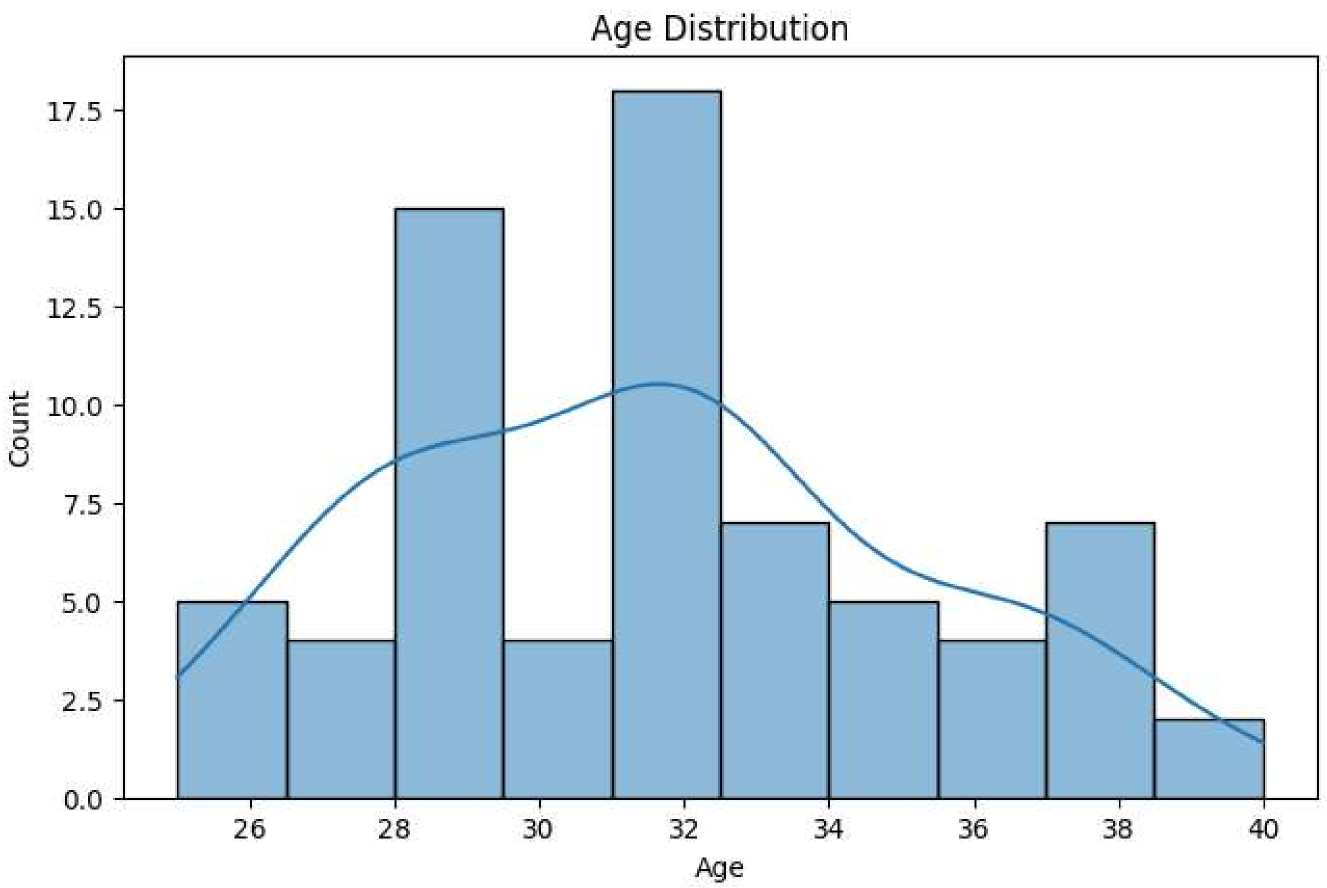
```
Index(['CustomerID', 'Name', 'Age', 'Gender', 'Income', 'Location',
       'SubscriptionID', 'StartDate', 'EndDate', 'Status', 'transaction_id',
       'transaction_date', 'amount', 'transaction_type', 'usage_id',
       'feature_used', 'usage_date', 'usage_amount', 'tenure',
       'average_monthly_spend'],
      dtype='object')
```

```
print(merged_df.head())
```

```
   CustomerID             Name  Age  Gender  Income   Location SubscriptionID  \
0           1      Karan Reddy   29    Male   40000  Hyderabad        SUB0001
1           2      Anaya Joshi   35  Female   55000  Bangalore        SUB0002
2           3         Diya Das   28  Female   60000    Chennai        SUB0003
3           4  Siddharth Reddy   32    Male   45000      Delhi        SUB0004
4           4  Siddharth Reddy   32    Male   45000      Delhi        SUB0004

   StartDate    EndDate Status transaction_id transaction_date  amount  \
0 2022-01-31        NaT    Yes            NaN              NaN     NaN
1 2022-02-28 2024-11-21     No       e183937c       2024-10-01   13.92
2 2022-03-31        NaT    Yes            NaN              NaN     NaN
```

```
   3 2022-04-30 2024-11-21     No       ce657026       2024-06-05   17.95
   4 2022-04-30 2024-11-21     No       ce657026       2024-06-05   17.95

     transaction_type  usage_id      feature_used  usage_date  usage_amount  \
   0              NaN       NaN               NaN         NaN           NaN
   1            Debit       4.0       Watch Movie  2024-08-02         30.94
   2              NaN      11.0      Rate Content  2024-02-21         21.23
   3            Debit      21.0  Download Episode  2024-11-05         82.45
   4            Debit      28.0       Watch Movie  2024-09-04         66.47

      tenure  average_monthly_spend
   0     NaN               0.000000
   1   997.0               0.418857
   2     NaN               0.000000
   3   936.0               0.575321
   4   936.0               0.575321
```

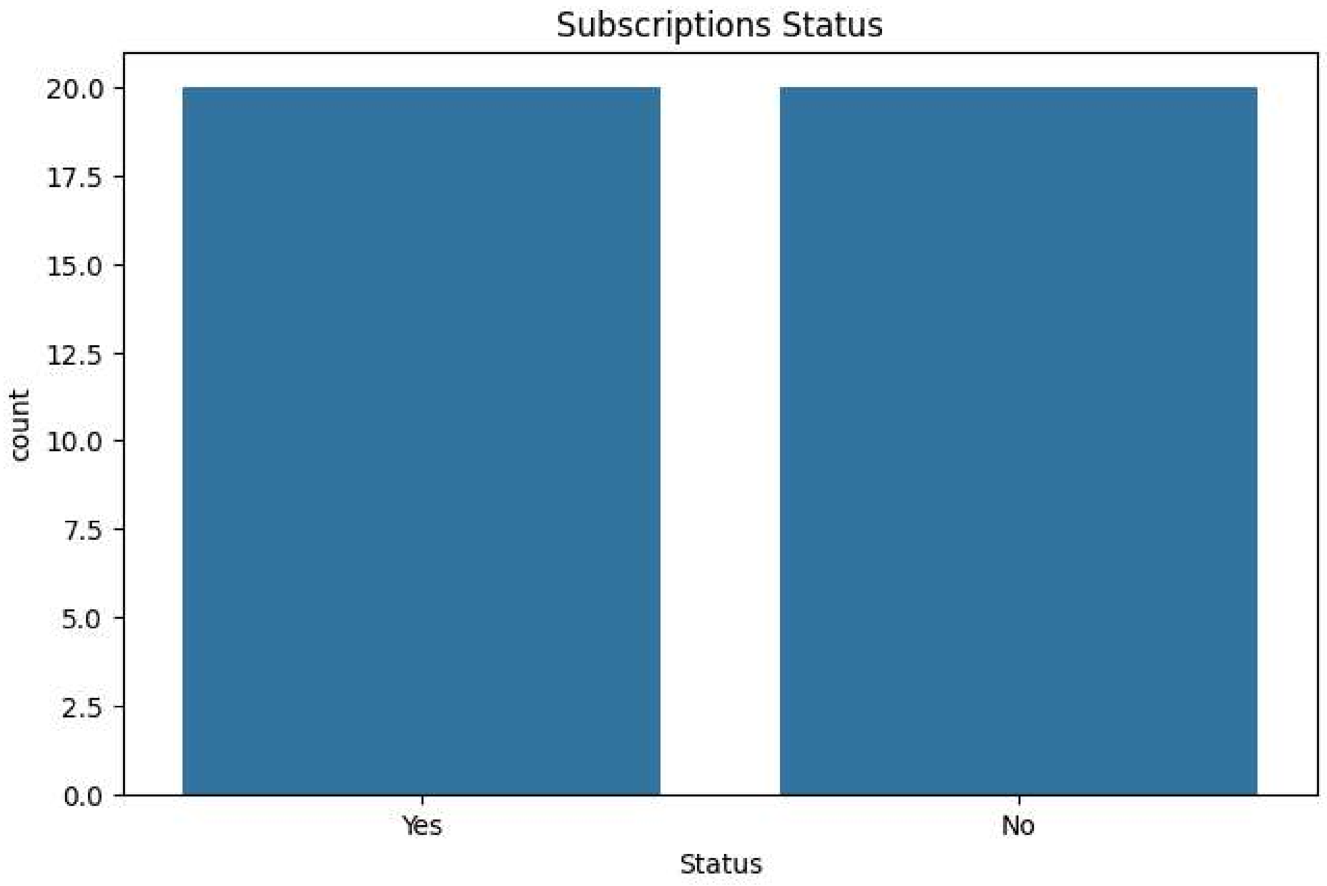
```
plt.figure(figsize=(8, 5))
sns.histplot(merged_df["Age"], bins=10, kde=True)
plt.title("Age Distribution")
plt.show()
```



```
Subscriptions = pd.read_csv("/content/Subscription.csv")
print(Subscriptions.columns)
plt.figure(figsize=(8, 5))
```

```
sns.countplot(x="Status", data=Subscriptions)
plt.title("Subscriptions Status")
plt.show()
```

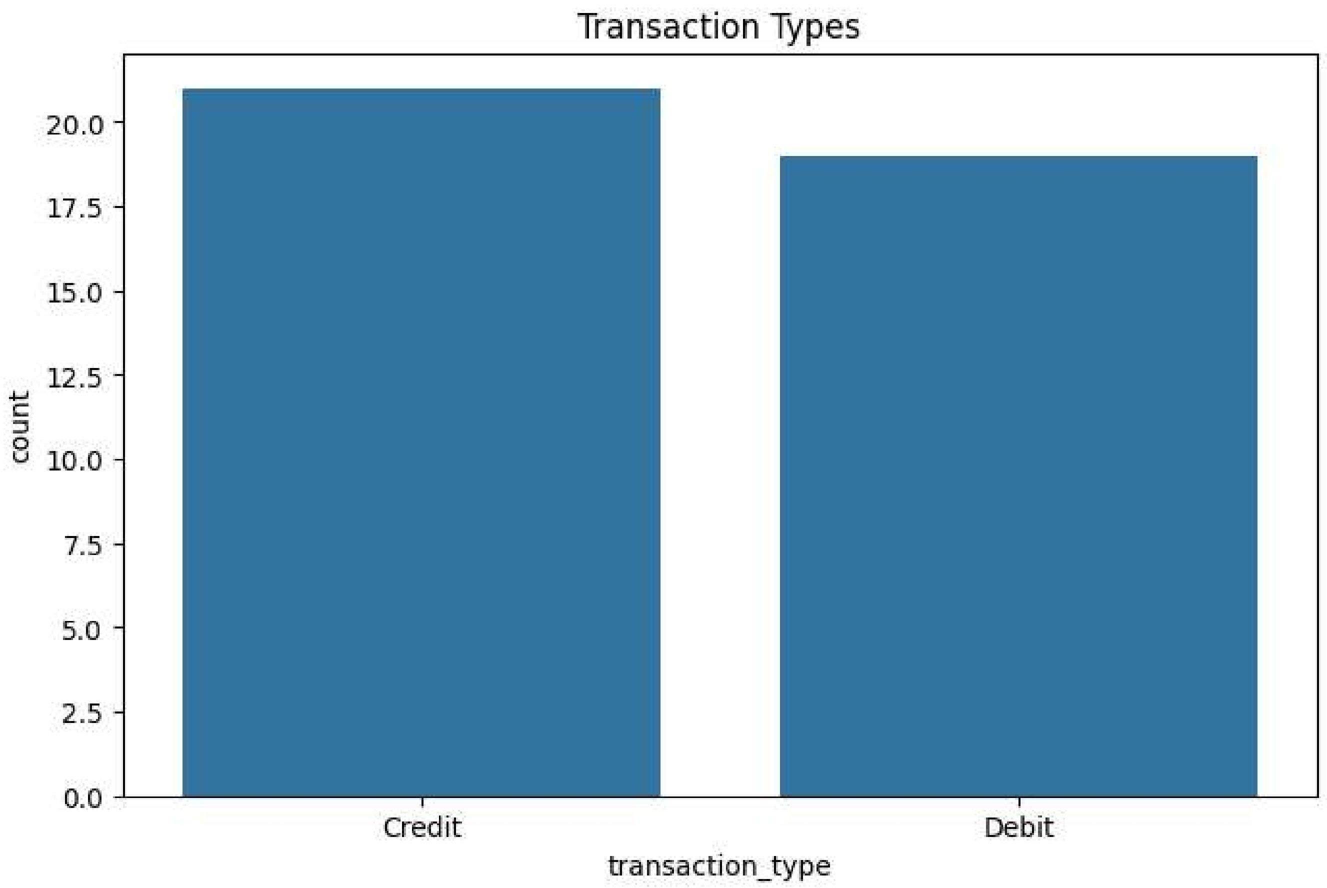
```
Index(['CustomerID', 'SubscriptionID', 'StartDate', 'EndDate', 'Status'], dtype='object'
```



```
print(Subscriptions.columns)
```

```
Index(['CustomerID', 'SubscriptionID', 'StartDate', 'EndDate', 'Status'], dtype='object'
```

```
plt.figure(figsize=(8, 5))
sns.countplot(x="transaction_type", data=Transactions)
plt.title("Transaction Types")
plt.show()
```



```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Print columns to inspect available ones
print(merged_df.columns)

# Apply label encoding with correct column names
merged_df["Gender"] = LabelEncoder().fit_transform(merged_df["Gender"])
merged_df["Location"] = LabelEncoder().fit_transform(merged_df["Location"])

# Use the correct column names based on your DataFrame
merged_df["SubscriptionID"] = LabelEncoder().fit_transform(merged_df["SubscriptionID"])  # I
merged_df["transaction_type"] = LabelEncoder().fit_transform(merged_df["transaction_type"])
```

```
Index(['CustomerID', 'Name', 'Age', 'Gender', 'Income', 'Location',
       'SubscriptionID', 'StartDate', 'EndDate', 'Status', 'transaction_id',
       'transaction_date', 'amount', 'transaction_type', 'usage_id',
       'feature_used', 'usage_date', 'usage_amount', 'tenure',
       'average_monthly_spend'],
      dtype='object')
```

Start coding or generate with AI.

```
# Check column names to make sure 'churned' is present
print(merged_df.columns)

# If 'churned' column doesn't exist, create it
if 'churned' not in merged_df.columns:
    merged_df["churned"] = (merged_df["Status"] == "Churned").astype(int)

# Define features and target variable
features = ["Age", "Gender", "Income", "tenure", "average_monthly_spend"]
X = merged_df[features]
y = merged_df["churned"]
```

```
Index(['CustomerID', 'Name', 'Age', 'Gender', 'Income', 'Location',
       'SubscriptionID', 'StartDate', 'EndDate', 'Status', 'transaction_id',
       'transaction_date', 'amount', 'transaction_type', 'usage_id',
       'feature_used', 'usage_date', 'usage_amount', 'tenure',
       'average_monthly_spend'],
      dtype='object')
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

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

# Train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluate model
predictions = model.predict(X_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15
```

```
from sklearn.impute import SimpleImputer

# Create an imputer to fill missing values with the mean
imputer = SimpleImputer(strategy='mean')

# Impute the missing values in X
```

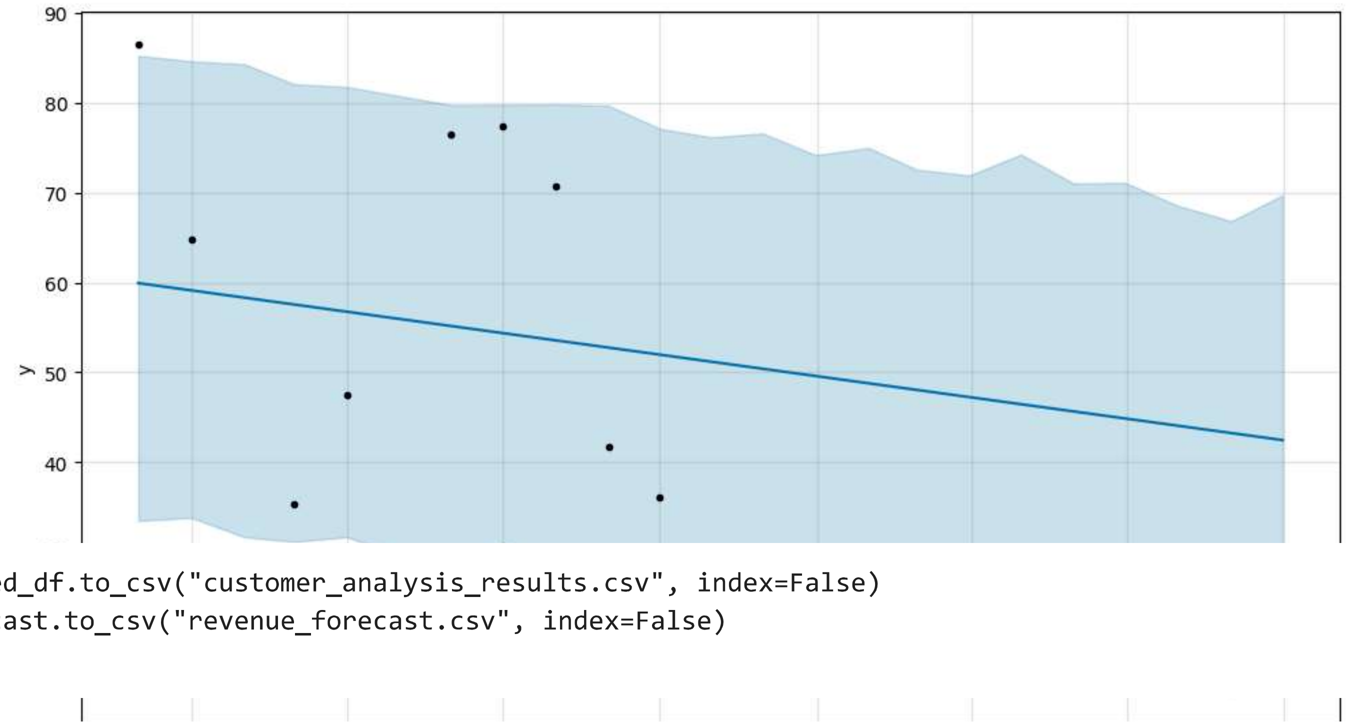
```
X_imputed = imputer.fit_transform(X)

# Perform clustering on the cleaned data
kmeans = KMeans(n_clusters=3, random_state=42)
merged_df["segment"] = kmeans.fit_predict(X_imputed)

# Analyze clusters
for segment in merged_df["segment"].unique():
    print(merged_df[merged_df["segment"] == segment].describe())
```

```
      50%    15.730000          0.000000  22.000000     54.420000  326.000000
      75%    16.307500          1.000000  30.250000     76.660000  326.000000
      max    18.900000          2.000000  36.000000     78.630000  326.000000
      std     3.941125          0.785905  12.802403     21.668911  211.310199

             average_monthly_spend  churned  segment
      count              17.000000     17.0     17.0
      mean                0.081577      0.0      1.0
      min                 0.000000      0.0      1.0
      25%                 0.000000      0.0      1.0
      50%                 0.000000      0.0      1.0
      75%                 0.000000      0.0      1.0
      max                 0.783129      0.0      1.0
      std                 0.232458      0.0      0.0
```

```
Transactions["transaction_date"] = pd.to_datetime(Transactions["transaction_date"])
```

```
# Prepare revenue data
revenue_data = Transactions.groupby(Transactions["transaction_date"].dt.to_period("M"))["amc
revenue_data.columns = ["ds", "y"]
revenue_data["ds"] = revenue_data["ds"].dt.to_timestamp()

# Fit Prophet model
model = Prophet()
model.fit(revenue_data)

# Forecast revenue
future = model.make_future_dataframe(periods=12, freq="M")
forecast = model.predict(future)
model.plot(forecast)
plt.show()
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to c
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to c
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to ove
INFO:prophet:n_changepoints greater than number of observations. Using 7.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpqyozztqq/uraxaqkh.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpqyozztqq/o1wt7bpx.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_moc
12:05:47 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
12:05:47 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.10/dist-packages/prophet/forecaster.py:1854: FutureWarning: 'M' i
  dates = pd.date_range(
```



```
merged_df.to_csv("customer_analysis_results.csv", index=False)
forecast.to_csv("revenue_forecast.csv", index=False)
```

Start coding or generate with AI.