import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
from statsmodels.tsa.arima.model import ARIMA  
import warnings  
warnings.filterwarnings("ignore")  
import seaborn as sns  
import plotly.express as px  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error  
  
from sklearn.preprocessing import LabelEncoder, StandardScaler

df = pd.read\_csv("/content/drive/MyDrive/DataSphere - Case Data - Network 18 (Updated).csv")

data = df.sample(frac=0.2)

# Assuming 'df' is your DataFrame  
  
  
data = df.sample(frac=0.05)  
# Specify the file path where you want to save the Excel file  
excel\_file\_path = "sample\_data.xlsx"  
  
# Save the sampled data to an Excel file  
data.to\_excel(excel\_file\_path, index=False)  
  
print(f"Excel file saved successfully at '{excel\_file\_path}'")

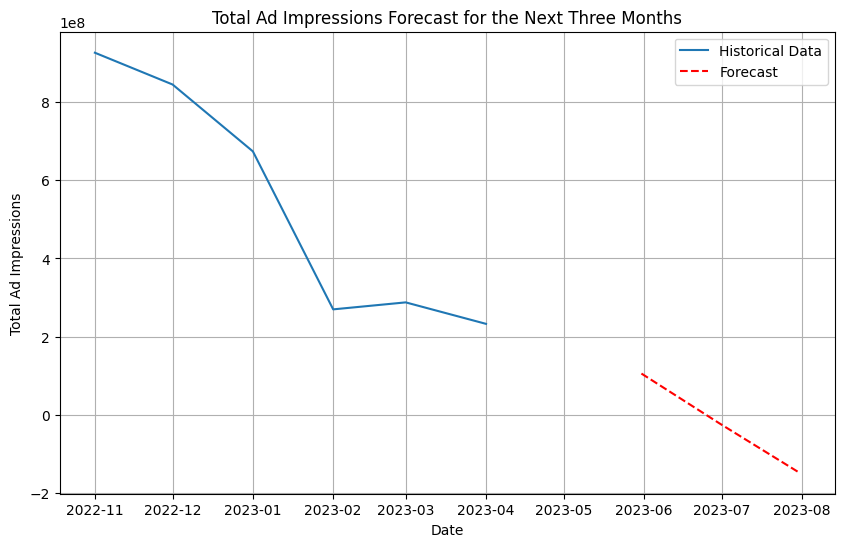
Excel file saved successfully at 'sample\_data.xlsx'

data.isnull().sum()

Unnamed: 0 0  
Dimension.MONTH\_AND\_YEAR 0  
Dimension.REGION\_NAME 0  
Dimension.COUNTRY\_NAME 3479  
Column.TOTAL\_CODE\_SERVED\_COUNT 0  
Column.TOTAL\_INVENTORY\_LEVEL\_UNFILLED\_IMPRESSIONS 0  
Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS 0  
Column.TOTAL\_LINE\_ITEM\_LEVEL\_CLICKS 0  
Column.TOTAL\_LINE\_ITEM\_LEVEL\_CPM\_AND\_CPC\_REVENUE 0  
Column.TOTAL\_LINE\_ITEM\_LEVEL\_WITHOUT\_CPD\_AVERAGE\_ECPM 0  
Column.TOTAL\_LINE\_ITEM\_LEVEL\_CTR 0  
Column.DROPOFF\_RATE 0  
ad\_size 0  
ad\_type 0  
video\_sub\_class 1368038  
website 0  
major\_sites 0  
sections 0  
home\_ros 0  
device 0  
app\_subclass 1288878  
ad\_positions 0  
languages 0  
dtype: int64

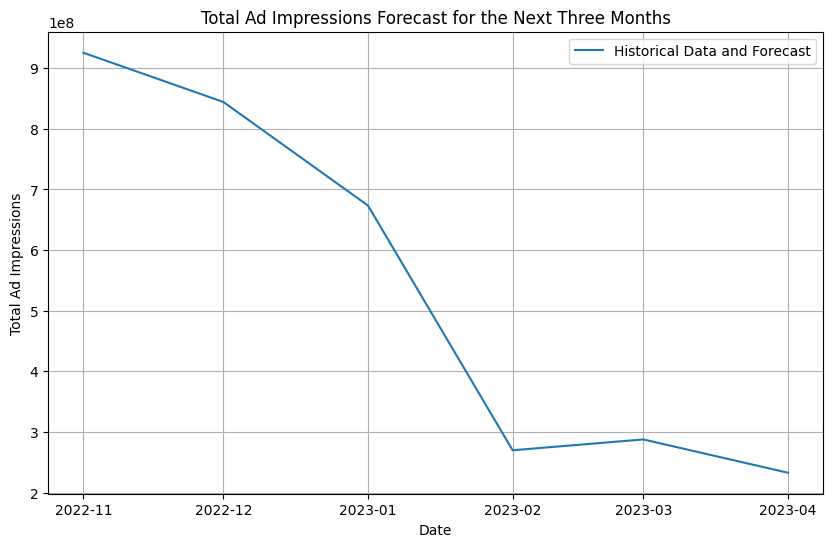
mode\_region = data['Dimension.REGION\_NAME'].mode()[0]  
data['Dimension.REGION\_NAME'].fillna(mode\_region, inplace=True)

# Replace "your\_data.csv" with the path to your dataset  
  
# Convert 'Dimension.MONTH\_AND\_YEAR' to datetime format  
data['Dimension.MONTH\_AND\_YEAR'] = pd.to\_datetime(data['Dimension.MONTH\_AND\_YEAR'])  
  
# Group by 'Dimension.MONTH\_AND\_YEAR' and sum the impressions  
monthly\_impressions = data.groupby('Dimension.MONTH\_AND\_YEAR')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
# Fit ARIMA model  
model = ARIMA(monthly\_impressions, order=(1,1,1))  
model\_fit = model.fit()  
  
# Forecast next three months  
forecast = model\_fit.forecast(steps=3)  
  
# Plot historical data and forecast  
plt.figure(figsize=(10, 6))  
plt.plot(monthly\_impressions.index, monthly\_impressions, label='Historical Data')  
plt.plot(pd.date\_range(start=monthly\_impressions.index[-1], periods=4, freq='M', tz='UTC')[1:], forecast, label='Forecast', linestyle='--', color='red')  
plt.title('Total Ad Impressions Forecast for the Next Three Months')  
plt.xlabel('Date')  
plt.ylabel('Total Ad Impressions')  
plt.legend()  
plt.grid(True)  
plt.show()



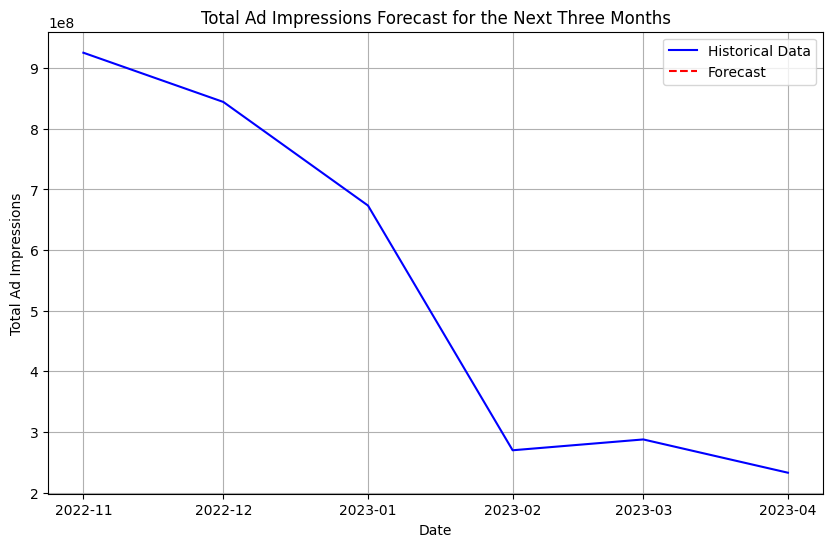
from statsmodels.tsa.arima.model import ARIMA  
  
  
  
# Convert 'Dimension.MONTH\_AND\_YEAR' to datetime format  
data['Dimension.MONTH\_AND\_YEAR'] = pd.to\_datetime(data['Dimension.MONTH\_AND\_YEAR'])  
  
# Group by 'Dimension.MONTH\_AND\_YEAR' and sum the impressions  
monthly\_impressions = data.groupby('Dimension.MONTH\_AND\_YEAR')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
# Fit ARIMA model  
model = ARIMA(monthly\_impressions, order=(1,1,1))  
model\_fit = model.fit()  
  
# Forecast next three months  
forecast = model\_fit.forecast(steps=3)  
  
# Extend the index to include future months  
extended\_index = pd.date\_range(start=monthly\_impressions.index[0], periods=len(monthly\_impressions)+3, freq='M')  
  
# Create a series with historical and forecasted values  
extended\_impressions = monthly\_impressions.append(pd.Series(forecast, index=extended\_index[-3:]))

# Plot historical data and forecast  
plt.figure(figsize=(10, 6))  
plt.plot(extended\_impressions.index, extended\_impressions, label='Historical Data and Forecast', linestyle='-')  
plt.title('Total Ad Impressions Forecast for the Next Three Months')  
plt.xlabel('Date')  
plt.ylabel('Total Ad Impressions')  
plt.legend()  
plt.grid(True)  
plt.show()



from statsmodels.tsa.arima.model import ARIMA  
  
  
  
# Convert 'Dimension.MONTH\_AND\_YEAR' to datetime format  
data['Dimension.MONTH\_AND\_YEAR'] = pd.to\_datetime(data['Dimension.MONTH\_AND\_YEAR'])  
  
# Group by 'Dimension.MONTH\_AND\_YEAR' and sum the impressions  
monthly\_impressions = data.groupby('Dimension.MONTH\_AND\_YEAR')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
# Fit ARIMA model  
model = ARIMA(monthly\_impressions, order=(1,1,1))  
model\_fit = model.fit()

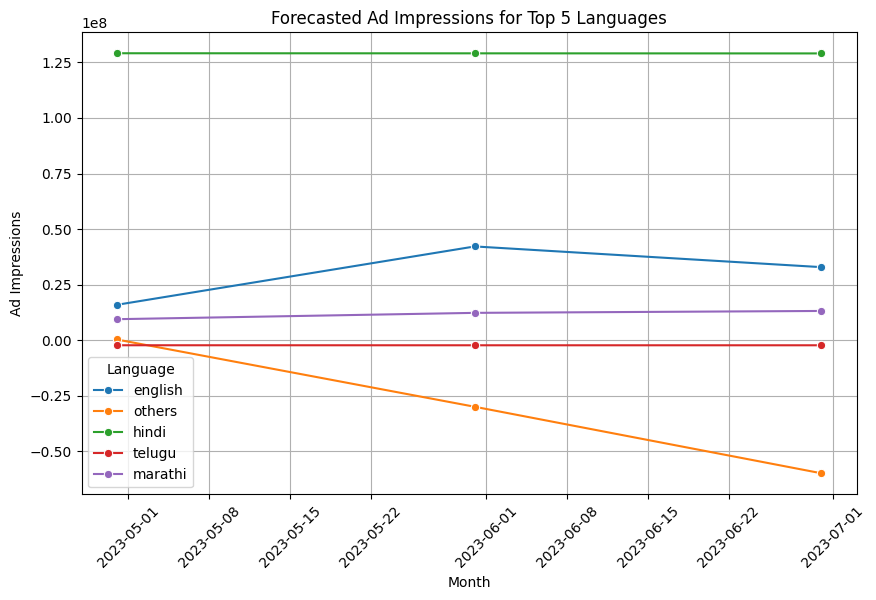
# Forecast next three months  
forecast = model\_fit.forecast(steps=3)  
  
# Extend the index to include future months  
extended\_index = pd.date\_range(start=monthly\_impressions.index[0], periods=len(monthly\_impressions)+3, freq='M')  
  
# Create a series with historical and forecasted values  
extended\_impressions = monthly\_impressions.append(pd.Series(forecast, index=extended\_index[-3:]))  
  
# Plot historical data and forecast  
plt.figure(figsize=(10, 6))  
plt.plot(extended\_impressions.index[:-3], extended\_impressions[:-3], label='Historical Data', linestyle='-', color='blue')  
plt.plot(extended\_impressions.index[-3:], extended\_impressions[-3:], label='Forecast', linestyle='--', color='red')  
plt.title('Total Ad Impressions Forecast for the Next Three Months')  
plt.xlabel('Date')  
plt.ylabel('Total Ad Impressions')  
plt.legend()  
plt.grid(True)  
plt.show()



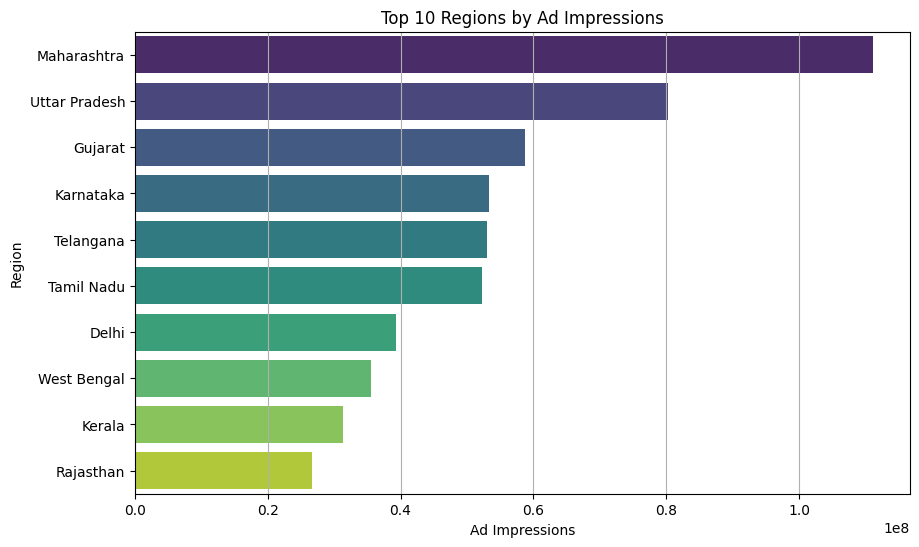
language\_impressions = data.groupby('languages')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
# Sort languages by impressions in descending order  
language\_impressions\_sorted = language\_impressions.sort\_values(ascending=False)  
  
# Create a color sequence using Plotly's default color scale  
color\_sequence = px.colors.qualitative.Plotly  
  
# Create a Plotly bar chart with different colors for each bar  
fig = px.bar(x=language\_impressions\_sorted.index, y=language\_impressions\_sorted.values,  
 labels={'x': 'Language', 'y': 'Ad Impressions'},  
 title='Language-wise Ad Impressions (Descending Order)',  
 color\_discrete\_sequence=color\_sequence,  
 template='plotly\_white')  
  
# Rotate x-axis labels for better readability  
fig.update\_layout(xaxis\_tickangle=-45)  
  
# Show the plot  
fig.show()

# Group by language and sum the impressions  
language\_impressions = data.groupby('languages')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
# Sort languages by impressions in descending order and select top 5 languages  
top\_5\_languages = language\_impressions.sort\_values(ascending=False).head(5)  
  
# Define a function to forecast ad impressions for a given language using ARIMA  
def forecast\_language(language):  
 # Filter data for the specified language  
 language\_data = data[data['languages'] == language]  
  
 # Convert 'Dimension.MONTH\_AND\_YEAR' to datetime format  
 language\_data['Dimension.MONTH\_AND\_YEAR'] = pd.to\_datetime(language\_data['Dimension.MONTH\_AND\_YEAR'])  
  
 # Group by month and sum the impressions  
 monthly\_impressions = language\_data.groupby('Dimension.MONTH\_AND\_YEAR')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
 # Fit ARIMA model  
 model = ARIMA(monthly\_impressions, order=(1, 1, 1))  
 model\_fit = model.fit()  
  
 # Forecast next three months  
 forecast = model\_fit.forecast(steps=3)

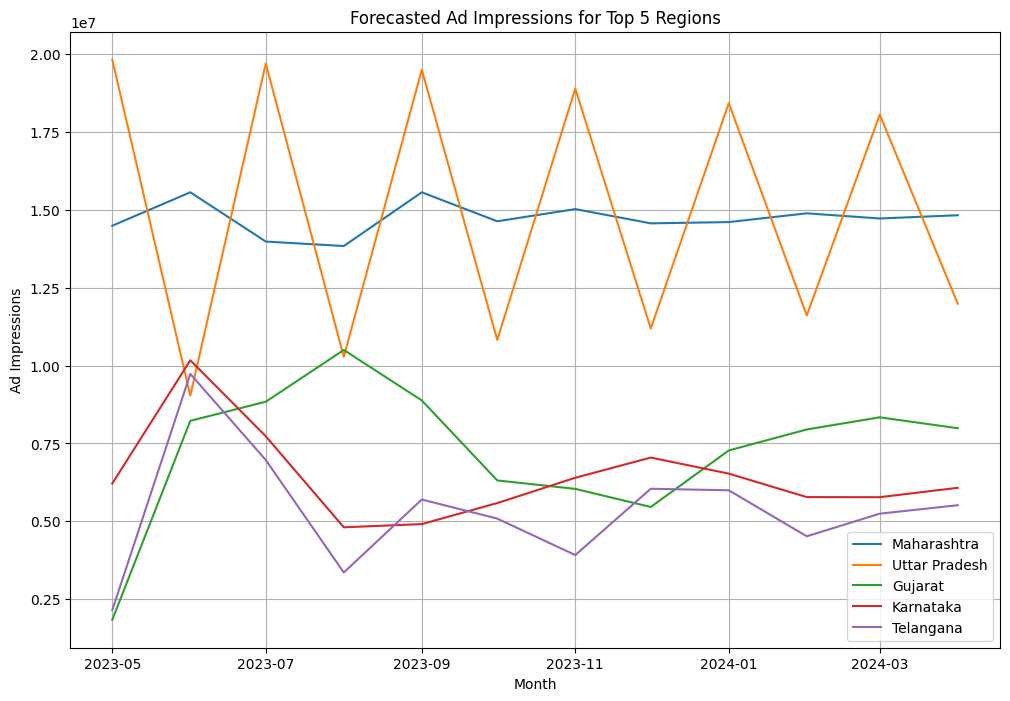
return forecast  
  
# Forecast ad impressions for each of the top 5 languages  
forecast\_data = pd.DataFrame(columns=['Language', 'Month', 'Forecast'])  
  
for language in top\_5\_languages.index:  
 forecast = forecast\_language(language)  
 forecast\_data = forecast\_data.append(pd.DataFrame({'Language': [language]\*3,  
 'Month': pd.date\_range(start=monthly\_impressions.index[-1], periods=3, freq='M'),  
 'Forecast': forecast}), ignore\_index=True)  
  
# Plot the forecasted ad impressions for the top 5 languages  
plt.figure(figsize=(10, 6))  
sns.lineplot(data=forecast\_data, x='Month', y='Forecast', hue='Language', marker='o')  
plt.title('Forecasted Ad Impressions for Top 5 Languages')  
plt.xlabel('Month')  
plt.ylabel('Ad Impressions')  
plt.xticks(rotation=45)  
plt.grid(True)  
plt.legend(title='Language')  
plt.show()



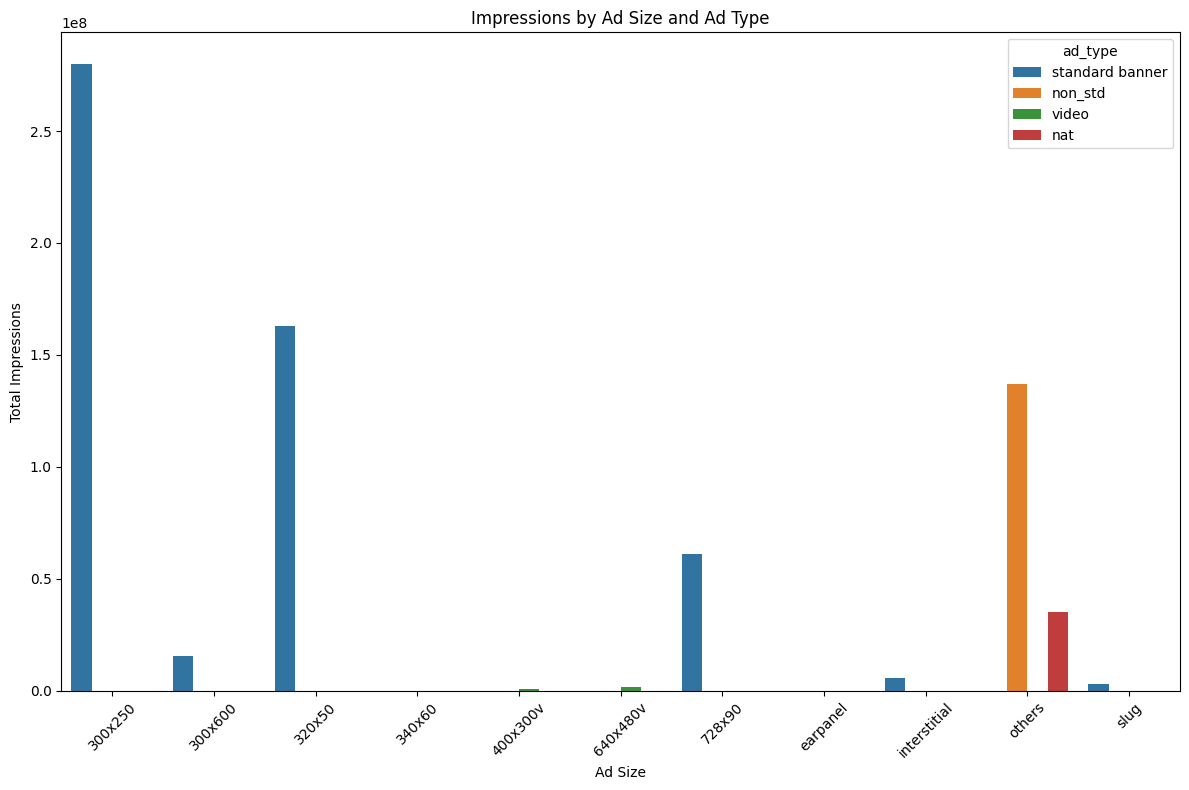
# Assuming 'data' is your DataFrame with region-wise impressions  
  
# Filter for top 10 regions by ad impressions  
top\_10\_regions = data.groupby('Dimension.REGION\_NAME')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum().nlargest(10)  
  
# Plot the bar graph  
plt.figure(figsize=(10, 6))  
sns.barplot(x=top\_10\_regions.values, y=top\_10\_regions.index, palette='viridis')  
plt.title('Top 10 Regions by Ad Impressions')  
plt.xlabel('Ad Impressions')  
plt.ylabel('Region')  
plt.grid(True, axis='x')  
plt.show()



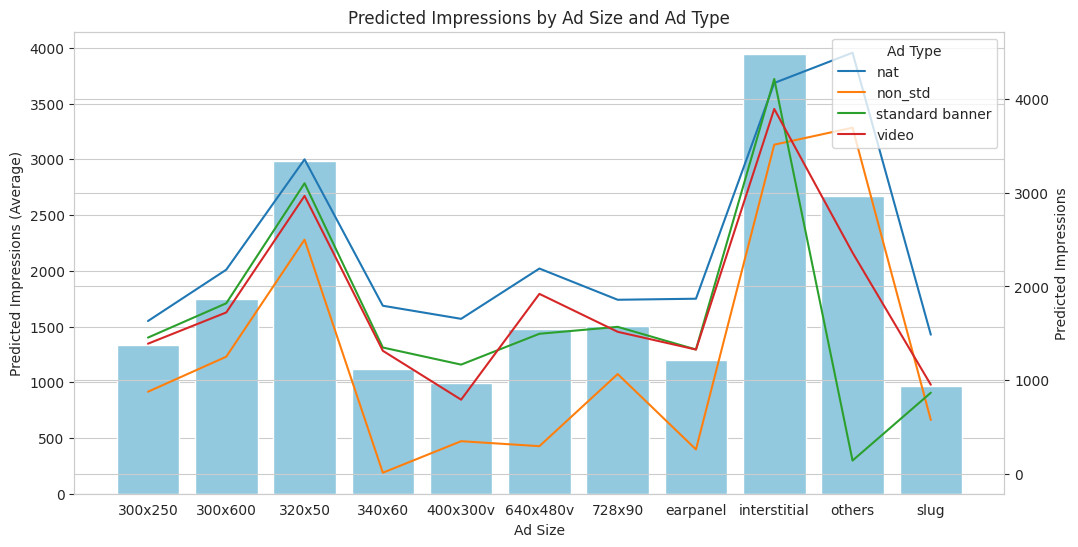
# Group by region and sum the ad impressions for each region  
region\_impressions = data.groupby('Dimension.REGION\_NAME')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
# Select the top 5 regions with the highest ad impressions  
top\_5\_regions = region\_impressions.nlargest(5)  
  
# Initialize a dictionary to store forecasts for each region  
region\_forecasts = {}  
  
# Iterate over the top 5 regions and apply ARIMA model to forecast ad impressions  
for region in top\_5\_regions.index:  
 # Filter data for the current region  
 region\_data = data[data['Dimension.REGION\_NAME'] == region]  
  
 # Group by month and sum ad impressions  
 monthly\_impressions = region\_data.groupby('Dimension.MONTH\_AND\_YEAR')['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum()  
  
 # Define and fit the ARIMA model  
 model = ARIMA(monthly\_impressions, order=(5, 1, 0)) # Example order, you can adjust it  
 model\_fit = model.fit()  
  
 # Forecast ad impressions for the next 12 months  
 forecast = model\_fit.forecast(steps=12)  
  
 # Store the forecast for the current region  
 region\_forecasts[region] = forecast  
  
# Plot the forecasted ad impressions for each region  
plt.figure(figsize=(12, 8))  
for region, forecast in region\_forecasts.items():  
 plt.plot(forecast.index, forecast.values, label=region)  
  
plt.title('Forecasted Ad Impressions for Top 5 Regions')  
plt.xlabel('Month')  
plt.ylabel('Ad Impressions')  
plt.legend()  
plt.grid(True)  
plt.show()



# Filter out rows with missing values in ad\_size and ad\_type columns  
data = data.dropna(subset=['ad\_size', 'ad\_type'])  
  
# Group the data by ad\_size and ad\_type and sum the impressions  
impressions\_by\_ad = data.groupby(['ad\_size', 'ad\_type'])['Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS'].sum().reset\_index()  
  
# Plot the data using seaborn  
plt.figure(figsize=(12, 8))  
sns.barplot(x='ad\_size', y='Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS', hue='ad\_type', data=impressions\_by\_ad)  
plt.title('Impressions by Ad Size and Ad Type')  
plt.xlabel('Ad Size')  
plt.ylabel('Total Impressions')  
plt.xticks(rotation=45)  
plt.tight\_layout()  
plt.show()



# Create a dataframe for visualization  
impression\_barplot = impression\_lineplot.groupby('Ad Size')['Predicted Impressions'].mean().reset\_index()  
  
# Set up the plot  
fig, ax1 = plt.subplots(figsize=(12, 6))  
  
# Plot bar graph for average predicted impressions by ad size  
sns.barplot(x='Ad Size', y='Predicted Impressions', data=impression\_barplot, ax=ax1, color='skyblue')  
ax1.set\_ylabel('Predicted Impressions (Average)')  
  
# Set up the second y-axis for line plot  
ax2 = ax1.twinx()  
  
# Plot line graph for predicted impressions by ad type  
for ad\_type in ad\_types:  
 ad\_type\_data = impression\_lineplot[impression\_lineplot['Ad Type'] == ad\_type]  
 ax2.plot(ad\_type\_data['Ad Size'], ad\_type\_data['Predicted Impressions'], label=ad\_type)  
  
# Add labels and legend  
ax2.set\_ylabel('Predicted Impressions')  
ax2.legend(title='Ad Type', loc='upper right')  
  
# Show plot  
plt.title('Predicted Impressions by Ad Size and Ad Type')  
plt.xlabel('Ad Size')  
  
plt.show()



# Extract relevant columns  
df = data[['ad\_type', 'ad\_size', 'Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS']]  
  
# Encode categorical variables  
df['ad\_type'] = pd.Categorical(df['ad\_type']).codes  
df['ad\_size'] = pd.Categorical(df['ad\_size']).codes  
  
# Calculate correlation coefficients  
correlation\_matrix = df.corr()  
  
# Create heatmap  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)  
plt.title('Correlation Heatmap of Ad Type, Size, and Impressions')  
plt.show()  
sns.pairplot(df)  
plt.show()



# Extract relevant columns  
df = data[['ad\_type', 'ad\_size', 'Column.TOTAL\_LINE\_ITEM\_LEVEL\_IMPRESSIONS']]  
  
# Encode categorical variables  
df['ad\_type'] = pd.Categorical(df['ad\_type']).codes  
df['ad\_size'] = pd.Categorical(df['ad\_size']).codes  
  
# Calculate correlation coefficients  
correlation\_matrix = df.corr()  
correlation\_matrix

{"summary":"{\n \"name\": \"correlation\_matrix\",\n \"rows\": 3,\n \"fields\": [\n {\n \"column\": \"ad\_type\",\n \"properties\":