Step-1 Synthetic Dataset using Feature Engineering and storing the excel file

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
# Expanding the dataset with more segments and brands
expanded_data = []

brands = ["Bobby Brown", "Elizabeth Arden", "Aveda", "Kilian", "Freder
categories = ["Face Make-Up", "Skin/Body", "Fragrance", "Hair Dye", "T
geographies = ["North America", "Europe", "South America", "Asia"]
portfolios = ["Fragrance + Color Cosmetics", "Hair/APDO", "Skin/Body"]
segments = ["Lipstick", "Mascara", "Toner", "Bronzer", "Foundation", "
```

In [3]:
```
# Generating 1000 random entries
for _ in range(1000):
    brand = np.random.choice(brands)
    category = np.random.choice(categories)
    geography = np.random.choice(geographies)
    portfolio = np.random.choice(portfolios)
    segment = np.random.choice(segments)

    # Generate random values within a realistic range
    current_revenue = np.random.randint(500000, 8000000)  # Revenue be
    margin = np.random.randint(5, 25)  # Margin between 5% and 25%
    min_trend = np.random.randint(-3, 0)  # Minimum trend between -3%
    max_trend = np.random.randint(3, 10)  # Maximum trend between 3% a
    min_contribution = np.random.randint(3, 10)  # Minimum contributio
    max_contribution = min_contribution + np.random.randint(5, 15)  #

    expanded_data.append([segment, brand, category, geography, portfol
                          min_trend, max_trend, min_contribution, max_
```

In [4]:
```
# Creating the expanded DataFrame
df_expanded = pd.DataFrame(expanded_data, columns=[
    "Segment", "Brand", "Category", "Geography", "Portfolio", "Current
    "Min Trend", "Max Trend", "Min Contribution", "Max Contribution"
])
```

In [5]:
```
# Saving the expanded dataset to a CSV file for download
file_path = "/Users/mohithreddy/Desktop/Python/Expanded_Acme_Data.csv
df_expanded.to_csv(file_path, index=False)
```

In [6]:
```
file_path
```

Out[6]: '/Users/mohithreddy/Desktop/Python/Expanded_Acme_Data.csv '

Step-2 Maximize the Sales

In [8]:
```
# read the file
file_path = "/Users/mohithreddy/Desktop/Python/Expanded_Acme_Data.xlsx
df = pd.read_excel(file_path)
```

In [9]:
```
df.head()
```

Out[9]:

	Segment	Brand	Category	Geography	Portfolio	Current Revenue	Margin	M Tre
0	Toner	Aveda	Face Make-Up	South America	Skin/Body	3944025	7	
1	Foundation	Aveda	Make-Up Brushes	South America	Skin/Body	3476493	9	
2	Toner	Frederic Malle	Face Make-Up	North America	Hair/APDO	6271132	23	
3	Toner	Bobby Brown	Fragrance	North America	Skin/Body	2182713	16	
4	Shampoo	Frederic Malle	Tools	Asia	Fragrance + Color Cosmetics	4612734	12	

In [11]:
```
from scipy.optimize import linprog
```

In [12]:
```
 #Function to generate synthetic constraints
def generate_constraints(df):
    constraints = {
        'global': {
            'max_growth': 0.15,  # 15% max growth globally
            'max_margin': 0.3   # 30% max margin globally
        },
        'branch': df.groupby('Geography').apply(lambda x: {
            'max_growth': np.random.uniform(0.05, 0.2),
            'max_margin': np.random.uniform(0.2, 0.4)
        }).to_dict(),
        'unit': df.apply(lambda x: {
            'max_growth': np.random.uniform(x['Min Trend'] / 100, x['M
            'max_margin': np.random.uniform(x['Min Contribution'] / 10
        }, axis=1)
    }
    return constraints

constraints = generate_constraints(df)
```

```
/var/folders/rc/cy98jnkx7x7ct8mcdjfccvrc0000gn/T/ipykernel_12774/377850
1919.py:8: DeprecationWarning: DataFrameGroupBy.apply operated on the g
rouping columns. This behavior is deprecated, and in a future version o
f pandas the grouping columns will be excluded from the operation. Eith
er pass `include_groups=False` to exclude the groupings or explicitly s
elect the grouping columns after groupby to silence this warning.
  'branch': df.groupby('Geography').apply(lambda x: {
```

In [20]:
```python
# Function to maximize sales
def maximize_sales(df, constraints):
    # Calculate Max Sales based on growth constraints
    df['Max Sales'] = df['Current Revenue'] * (1 + constraints['global
    df['Max Sales'] = df['Max Sales'].clip(upper=df['Current Revenue']

    # Calculate Contribution (Profitability)
    df['Contribution'] = df['Max Sales'] * df['Margin']

    # Compute total max sales
    total_max_sales = df['Max Sales'].sum()

    # Display summary for leadership
    print(f"Maximum possible sales: ${total_max_sales:,.2f}")

    if 'requested_growth' in constraints['global']:
        requested_growth = constraints['global']['requested_growth']
        achievable_growth = total_max_sales - df['Current Revenue'].su
        if achievable_growth < requested_growth:
            print(f"Requested growth of ${requested_growth:,.2f} is no
                  f"Max achievable growth is ${achievable_growth:,.2f}

    # Return detailed breakdown including trends and contribution
    return df[['Segment', 'Brand', 'Category', 'Geography', 'Portfolio
```

Step-3 Maximize Margin

In [21]:
```python
# Function to maximize margin
def maximize_margin(df, constraints):
    # Calculate Max Margin based on growth constraints
    df['Max Margin'] = df['Margin'] * (1 + constraints['global']['max_
    df['Max Margin'] = df['Max Margin'].clip(upper=df['Margin'] * (1 +

    # Compute Contribution (profitability)
    df['Contribution'] = df['Current Revenue'] * df['Max Margin']

    # Compute total max margin
    total_max_margin = df['Contribution'].sum()

    # Display summary for leadership
    print(f"Maximum possible margin: ${total_max_margin:,.2f}")
```

```
    if 'requested_margin' in constraints['global']:
        requested_margin = constraints['global']['requested_margin']
        achievable_margin = total_max_margin - (df['Current Revenue']
        if achievable_margin < requested_margin:
            print(f"Requested margin increase of ${requested_margin:,.
                  f"Max achievable margin increase is ${achievable_mar

    # Return detailed breakdown including trend and contribution
    return df[['Segment', 'Brand', 'Category', 'Geography', 'Portfolio
```

Step-4 Hitting a sales target with Maximizing Margin

In [22]:
```
# Function to hit a sales target while maximizing margin
def hit_sales_target(df, target_sales, constraints):
    # Calculate potential Sales Growth
    df['Sales Growth'] = df['Current Revenue'] * (1 + df['Max Trend']

    # Compute Contribution (profitability)
    df['Contribution'] = df['Sales Growth'] * df['Margin']

    # Sort by Margin (Descending) to maximize profit margin
    df = df.sort_values(by='Margin', ascending=False)

    # Cumulatively add sales until the target is reached
    df['Cumulative Sales'] = df['Sales Growth'].cumsum()

    # Identify the row where the target sales is exceeded
    last_index = df[df['Cumulative Sales'] > target_sales].index.min()

    if pd.notna(last_index):  # If the target is exceeded in a row
        df.loc[last_index, 'Sales Growth'] = target_sales - df.loc[las
        df.loc[last_index, 'Contribution'] = df.loc[last_index, 'Sales
        df = df.loc[:last_index]  # Keep only the required rows

    # Compute final total sales achieved
    total_sales_achieved = df['Sales Growth'].sum()

    # Display leadership summary
    print(f"Target Sales: ${target_sales:,.2f}")
    print(f"Total Sales Achieved: ${total_sales_achieved:,.2f}")

    if total_sales_achieved < target_sales:
        print(f"Warning: Could not fully reach the target. Max achieva

    # Return final dataset including trend and contribution
    return df[['Segment', 'Brand', 'Category', 'Geography', 'Portfolio
```

Step-5 Hitting A Margin Target While Maximizing Sales:

In [23]:
```
# Function to hit a margin target while maximizing sales
def hit_margin_target(df, target_margin, constraints):
    # Calculate potential Margin Growth
    df['Margin Growth'] = df['Margin'] * (1 + df['Max Contribution'] /

    # Compute Contribution (profitability)
    df['Contribution'] = df['Current Revenue'] * df['Margin Growth']

    # Sort by Current Revenue (Descending) to maximize sales impact
    df = df.sort_values(by='Current Revenue', ascending=False)

    # Cumulatively add margins until the target is reached
    df['Cumulative Margin'] = df['Margin Growth'].cumsum()

    # Identify the row where the target margin is exceeded
    last_index = df[df['Cumulative Margin'] > target_margin].index.min

    if pd.notna(last_index):  # If the target is exceeded in a row
        df.loc[last_index, 'Margin Growth'] = target_margin - df.loc[l
        df.loc[last_index, 'Contribution'] = df.loc[last_index, 'Curre
        df = df.loc[:last_index]  # Keep only the required rows

    # Compute final total margin achieved
    total_margin_achieved = df['Margin Growth'].sum()

    # Display leadership summary
    print(f"Target Margin: ${target_margin:,.2f}")
    print(f"Total Margin Achieved: ${total_margin_achieved:,.2f}")

    if total_margin_achieved < target_margin:
        print(f"Warning: Could not fully reach the target. Max achieva

    # Return final dataset including trend and contribution
    return df[['Segment', 'Brand', 'Category', 'Geography', 'Portfolio
```

Step-6 Projections for Each Year Over a 5 Year Period

In [33]:
```
def project_5_years(df, constraints_per_year):
    """
    Projects revenue, margin, and contribution for a 5-year period.

    Args:
    df (DataFrame): Input data containing current revenue, margin, and
```

```
    constraints_per_year (dict): Dictionary with constraints for each

    Returns:
    DataFrame: Updated dataframe with projections over 5 years.
    """

    # Create copies of initial revenue and margin for tracking year-ov
    df['Year 0 Revenue'] = df['Current Revenue']
    df['Year 0 Margin'] = df['Margin']

    # Loop through each year (1 to 5) and apply constraints
    for year in range(1, 6):
        constraints = constraints_per_year.get(year, {})

        # Extract constraints with defaults
        max_growth = constraints.get('max_growth', 0.05)  # Default 5%
        max_margin = constraints.get('max_margin', 0.02)  # Default 2%
        trend_influence = constraints.get('trend_influence', 1.0)  # D

        # Calculate sales growth based on trend and constraints
        df[f'Year {year} Revenue'] = df[f'Year {year-1} Revenue'] * (1

        # Calculate margin growth based on constraints
        df[f'Year {year} Margin'] = df[f'Year {year-1} Margin'] * (1 +

        # Compute contribution (profitability) for each year
        df[f'Year {year} Contribution'] = df[f'Year {year} Revenue'] *

    # Leadership summary
    total_revenue_5_years = df[[f'Year {year} Revenue' for year in ran
    total_contribution_5_years = df[[f'Year {year} Contribution' for y

    print(f"Total projected revenue over 5 years: ${total_revenue_5_ye
    print(f"Total projected contribution over 5 years: ${total_contrib

    return df[['Segment', 'Brand', 'Category', 'Geography', 'Portfolio
              [f'Year {year} Revenue' for year in range(1, 6)] +
              [f'Year {year} Margin' for year in range(1, 6)] +
              [f'Year {year} Contribution' for year in range(1, 6)]]
```

In [32]:
```
# Example usage
sales_maximized = maximize_sales(df, constraints)
margin_maximized = maximize_margin(df, constraints)
sales_target_result = hit_sales_target(df, 50000000, constraints)
margin_target_result = hit_margin_target(df, 10000000, constraints)
```

```
Maximum possible sales: $4,507,308,444.12
Maximum possible margin: $70,197,635,851.03
```

```
---------------------------------------------------------------------
----
KeyError                                  Traceback (most recent call l
ast)
```

```
File ~/miniconda3/envs/av/lib/python3.12/site-packages/pandas/core/indexes/base.py:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:2606, in pandas._libs.hashtable.Int64HashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:2630, in pandas._libs.hashtable.Int64HashTable.get_item()

KeyError: -1

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[32], line 4
      2 sales_maximized = maximize_sales(df, constraints)
      3 margin_maximized = maximize_margin(df, constraints)
----> 4 sales_target_result = hit_sales_target(df, 50000000, constraints)
      5 margin_target_result = hit_margin_target(df, 10000000, constraints)

Cell In[22], line 19, in hit_sales_target(df, target_sales, constraints)
     16 last_index = df[df['Cumulative Sales'] > target_sales].index.min()
     18 if pd.notna(last_index):  # If the target is exceeded in a row
---> 19     df.loc[last_index, 'Sales Growth'] = target_sales - df.loc[last_index - 1, 'Cumulative Sales']
     20     df.loc[last_index, 'Contribution'] = df.loc[last_index, 'Sales Growth'] * df.loc[last_index, 'Margin']
     21     df = df.loc[:last_index]  # Keep only the required rows

File ~/miniconda3/envs/av/lib/python3.12/site-packages/pandas/core/indexing.py:1183, in _LocationIndexer.__getitem__(self, key)
   1181     key = tuple(com.apply_if_callable(x, self.obj) for x in key)
   1182     if self._is_scalar_access(key):
-> 1183         return self.obj._get_value(*key, takeable=self._takeable)
   1184     return self._getitem_tuple(key)
   1185 else:
   1186     # we by definition only have the 0th axis
```

```
File ~/miniconda3/envs/av/lib/python3.12/site-packages/pandas/core/frame.py:4221, in DataFrame._get_value(self, index, col, takeable)
   4215 engine = self.index._engine
   4217 if not isinstance(self.index, MultiIndex):
   4218     # CategoricalIndex: Trying to use the engine fastpath may give incorrect
   4219     #  results if our categories are integers that dont match our codes
   4220     # IntervalIndex: IntervalTree has no get_loc
-> 4221     row = self.index.get_loc(index)
   4222     return series._values[row]
   4224 # For MultiIndex going through engine effectively restricts us to
   4225 #  same-length tuples; see test_get_set_value_no_partial_indexing

File ~/miniconda3/envs/av/lib/python3.12/site-packages/pandas/core/indexes/base.py:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: -1
```

In [29]:
```
print("Maximized Sales:")
print(sales_maximized.head())
print("\nMaximized Margin:")
print(margin_maximized.head())
print("\nSales Target Result:")
print(sales_target_result.head())
print("\nMargin Target Result:")
print(margin_target_result.head())
```

```
Maximized Sales:
      Segment           Brand         Category      Geography  \
0       Toner           Aveda     Face Make-Up  South America
1  Foundation           Aveda  Make-Up Brushes  South America
2       Toner  Frederic Malle     Face Make-Up  North America
3       Toner     Bobby Brown        Fragrance  North America
4     Shampoo  Frederic Malle            Tools           Asia

                    Portfolio   Max Sales  Margin  Max Trend  Contribution
```

```
0                   Skin/Body  4141226.25       7          5  2.898858
e+07
1                   Skin/Body  3719847.51       9          7  3.347863
e+07
2                   Hair/APDO  6459265.96      23          3  1.485631
e+08
3                   Skin/Body  2335502.91      16          7  3.736805
e+07
4  Fragrance + Color Cosmetics  4797243.36      12          4  5.756692
e+07

Maximized Margin:
      Segment           Brand         Category      Geography  \
0       Toner           Aveda     Face Make-Up  South America
1  Foundation           Aveda  Make-Up Brushes  South America
2       Toner  Frederic Malle     Face Make-Up  North America
3       Toner     Bobby Brown        Fragrance  North America
4     Shampoo  Frederic Malle            Tools           Asia

                    Portfolio  Current Revenue  Max Margin  Max Trend
\
0                   Skin/Body          3944025        7.70          5
1                   Skin/Body          3476493       10.98          7
2                   Hair/APDO          6271132       26.22          3
3                   Skin/Body          2182713       19.04          7
4  Fragrance + Color Cosmetics          4612734       13.92          4

   Contribution
0  3.036899e+07
1  3.817189e+07
2  1.644291e+08
3  4.155886e+07
4  6.420926e+07

Sales Target Result:
     Segment            Brand         Category      Geography  Portfoli
o  \
554  Shampoo   Frederic Malle     Face Make-Up  South America  Hair/APD
O
416    Blush      Bobby Brown  Make-Up Brushes           Asia  Hair/APD
O
326    Serum           Kilian            Tools  North America  Hair/APD
O
118  Mascara  Elizabeth Arden     Face Make-Up  North America  Hair/APD
O
672    Serum      Bobby Brown  Make-Up Brushes         Europe  Hair/APD
O

     Sales Growth  Margin
554    5293740.90      24
416    7867252.27      24
326    7425840.82      24
```

```
118    6624647.64      24
672    8027940.51      24

Margin Target Result:
        Segment           Brand      Category      Geography  \
122  Foundation  Frederic Malle         Tools         Europe
331   Concealer           Aveda  Face Make-Up         Europe
956     Bronzer  Frederic Malle         Tools  North America
245       Blush           Aveda     Fragrance  North America
516    Lipstick          Kilian      Hair Dye  South America

                     Portfolio  Current Revenue  Margin Growth
122                  Skin/Body          7996846          24.15
331                  Hair/APDO          7987907          15.34
956                  Hair/APDO          7979832           6.84
245                  Skin/Body          7975078          13.44
516  Fragrance + Color Cosmetics          7974813           5.65
```

Performing Exploratory Data Analysis for the above data

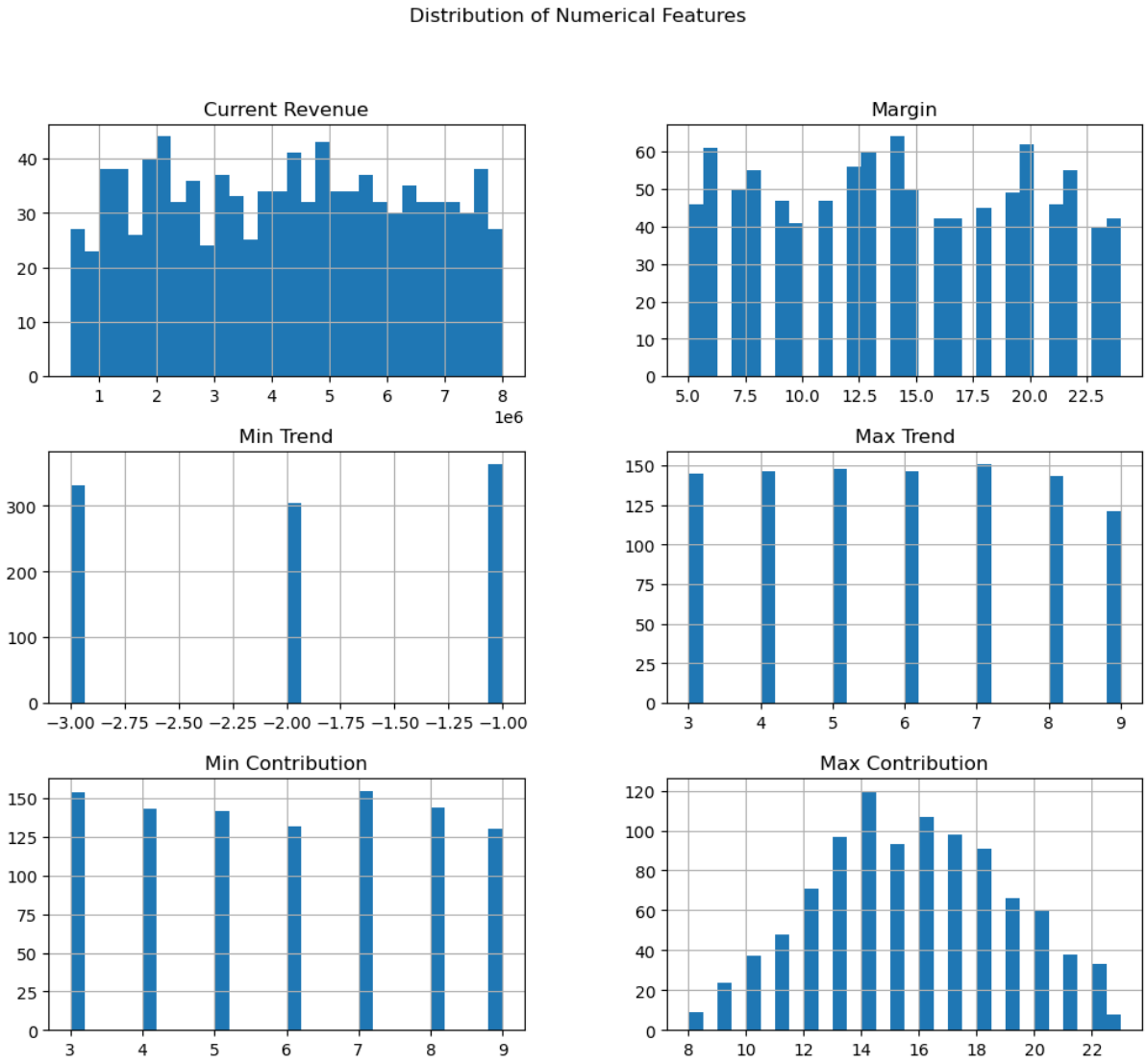
In [35]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [36]:
```
# summary statistics
summary_stats = df.describe()
```

In [37]:
```
# Checking for missing values
missing_values = df.isnull().sum()
```

In [39]:
```
# Visualizing distributions of numerical columns
plt.figure(figsize=(12, 6))
df[['Current Revenue', 'Margin', 'Min Trend', 'Max Trend', 'Min Contri
plt.suptitle('Distribution of Numerical Features')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



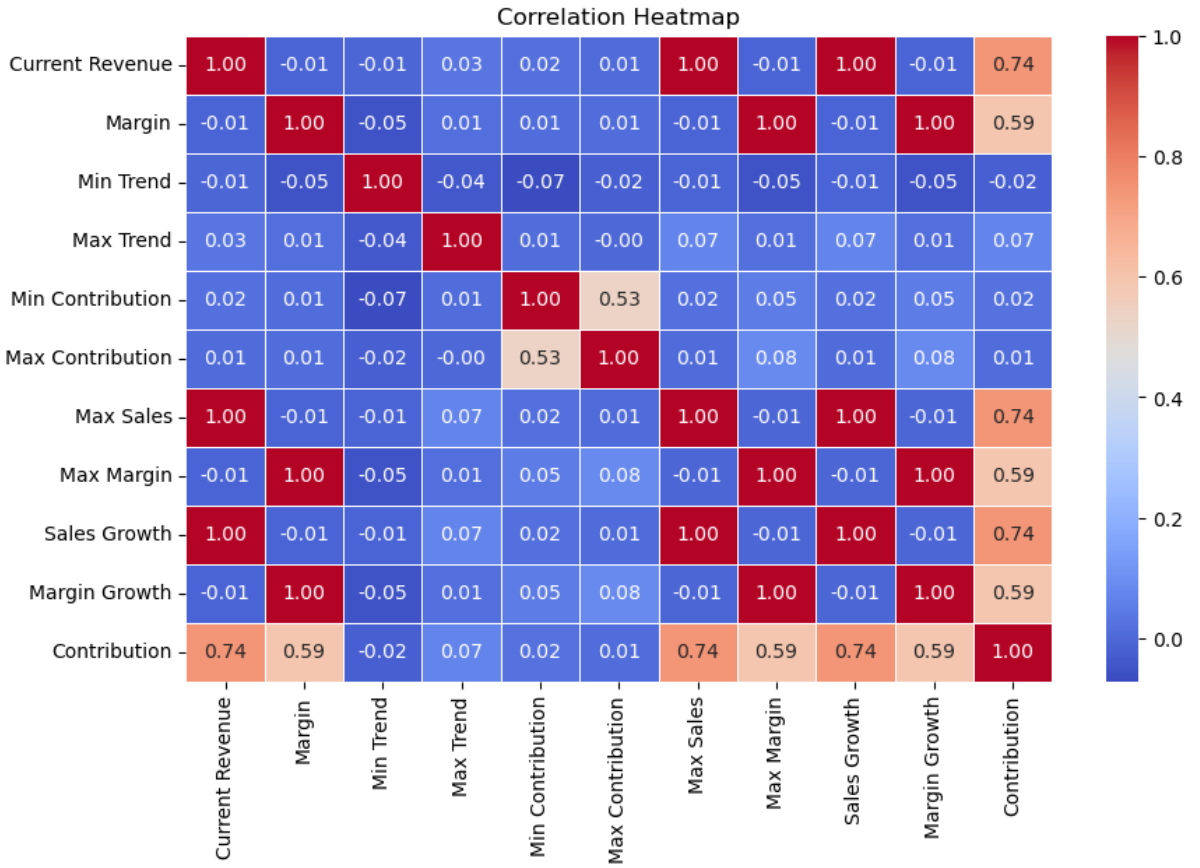
In [42]:
```
# Selecting only numerical columns for correlation calculation
numerical_df = df.select_dtypes(include=['number'])

# Computing correlation matrix
correlation_matrix = numerical_df.corr()
```

In [44]:
```
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                  Current Revenue    Margin  Min Trend  Max Trend  \
Current Revenue          1.000000 -0.009691  -0.006375   0.030406
Margin                  -0.009691  1.000000  -0.050558   0.014486
Min Trend               -0.006375 -0.050558   1.000000  -0.040123
Max Trend                0.030406  0.014486  -0.040123   1.000000
Min Contribution         0.020030  0.009334  -0.070919   0.011155
Max Contribution         0.005066  0.010408  -0.021589  -0.002224
Max Sales                0.999128 -0.008615  -0.008618   0.067540
Max Margin              -0.009149  0.996883  -0.052111   0.013571
Sales Growth             0.999128 -0.008615  -0.008618   0.067540
Margin Growth           -0.009149  0.996883  -0.052111   0.013571
Contribution             0.742758  0.592152  -0.021478   0.067323

                  Min Contribution  Max Contribution  Max Sales  Max Ma
rgin  \
Current Revenue           0.020030          0.005066   0.999128   -0.00
9149
Margin                    0.009334          0.010408  -0.008615    0.99
6883
Min Trend                -0.070919         -0.021589  -0.008618   -0.05
2111
Max Trend                 0.011155         -0.002224   0.067540    0.01
3571
Min Contribution          1.000000          0.533769   0.020288    0.04
9045
Max Contribution          0.533769          1.000000   0.005130    0.08
4014
Max Sales                 0.020288          0.005130   1.000000   -0.00
8079
Max Margin                0.049045          0.084014  -0.008079    1.00
0000
Sales Growth              0.020288          0.005130   1.000000   -0.00
8079
Margin Growth             0.049045          0.084014  -0.008079    1.00
0000
Contribution              0.021458          0.011487   0.744286    0.59
1004

                  Sales Growth  Margin Growth  Contribution
Current Revenue       0.999128      -0.009149      0.742758
Margin               -0.008615       0.996883      0.592152
Min Trend            -0.008618      -0.052111     -0.021478
Max Trend             0.067540       0.013571      0.067323
Min Contribution      0.020288       0.049045      0.021458
Max Contribution      0.005130       0.084014      0.011487
Max Sales             1.000000      -0.008079      0.744286
Max Margin           -0.008079       1.000000      0.591004
Sales Growth          1.000000      -0.008079      0.744286
Margin Growth        -0.008079       1.000000      0.591004
Contribution          0.744286       0.591004      1.000000
```

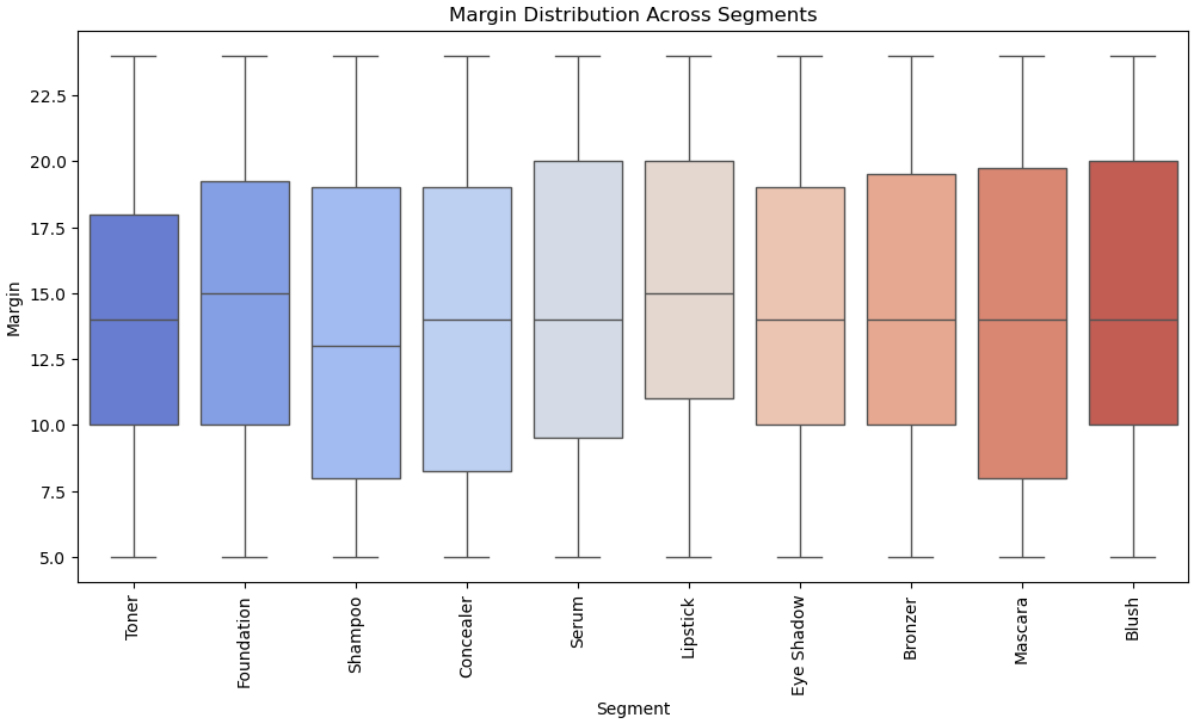
In [46]:
```
# Correlation heatmap
```

```
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f"
plt.title("Correlation Heatmap")
plt.show()
```

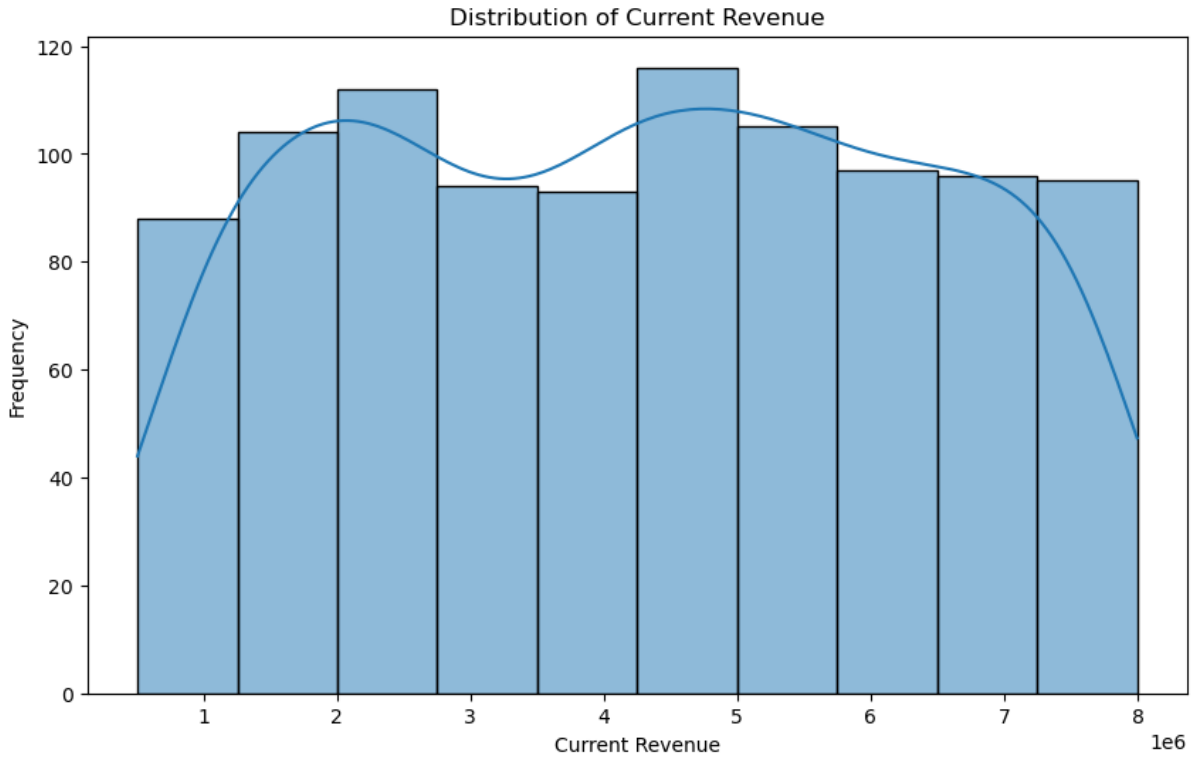


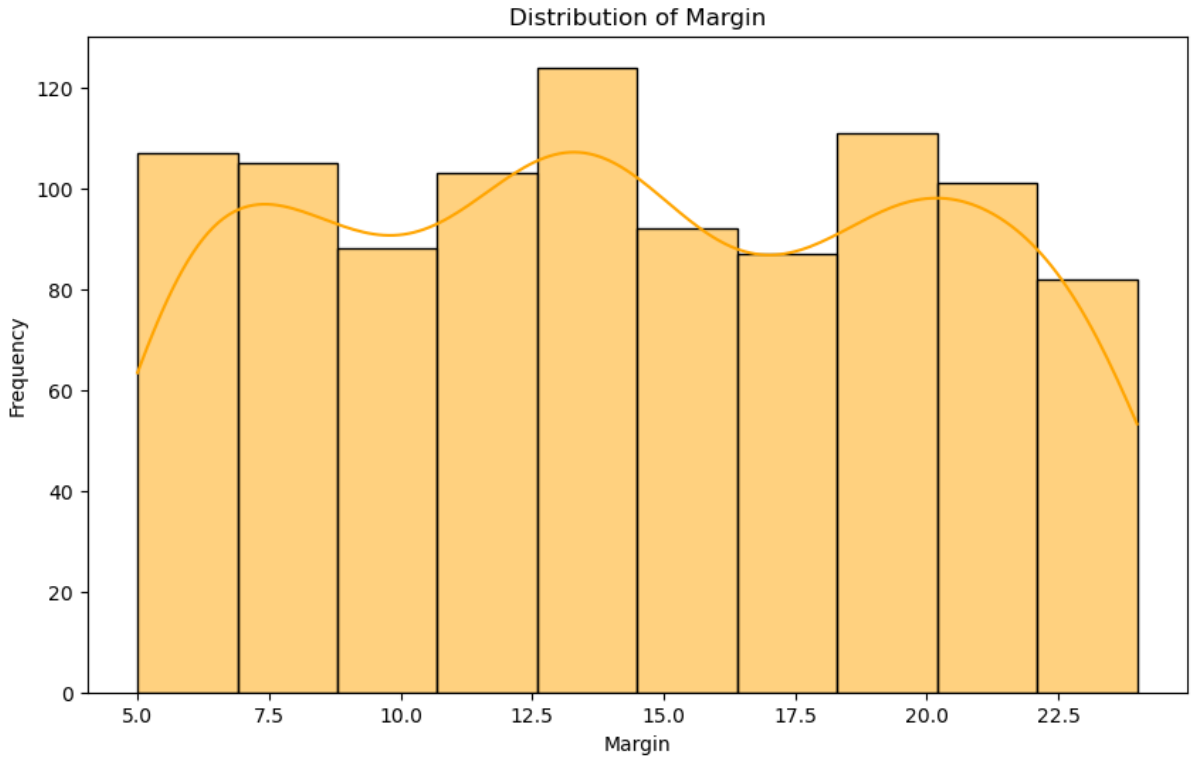
In [48]:
```
# Box plots for margin across different segments
plt.figure(figsize=(12, 6))
sns.boxplot(x='Segment', y='Margin', data=df, palette="coolwarm")  # A
plt.xticks(rotation=90)
plt.title("Margin Distribution Across Segments")
plt.show()
```

```
/var/folders/rc/cy98jnkx7x7ct8mcdjfccvrc0000gn/T/ipykernel_12774/431772
345.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be rem
oved in v0.14.0. Assign the `x` variable to `hue` and set `legend=False
` for the same effect.

  sns.boxplot(x='Segment', y='Margin', data=df, palette="coolwarm")  #
Adding color
```

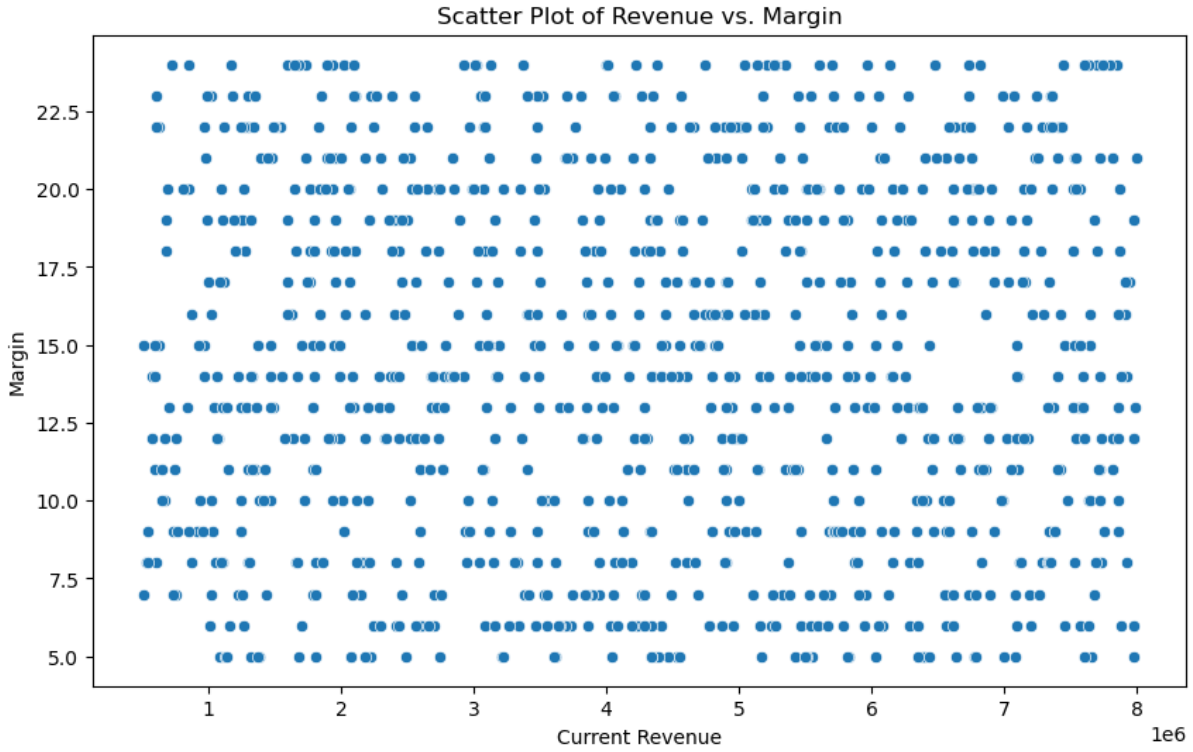


In [49]:
```
# Distribution Plots
plt.figure(figsize=(10, 6))
sns.histplot(df['Current Revenue'], bins=10, kde=True)
plt.title('Distribution of Current Revenue')
plt.xlabel('Current Revenue')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(df['Margin'], bins=10, kde=True, color='orange')
plt.title('Distribution of Margin')
plt.xlabel('Margin')
plt.ylabel('Frequency')
plt.show()
```

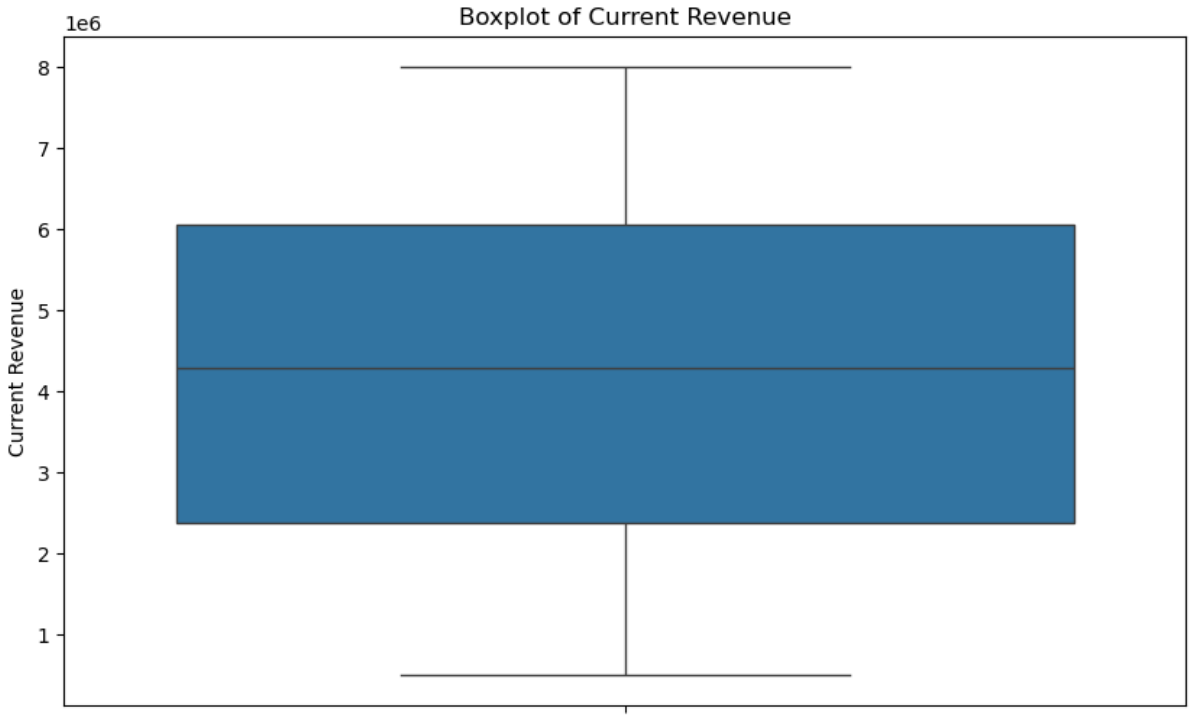


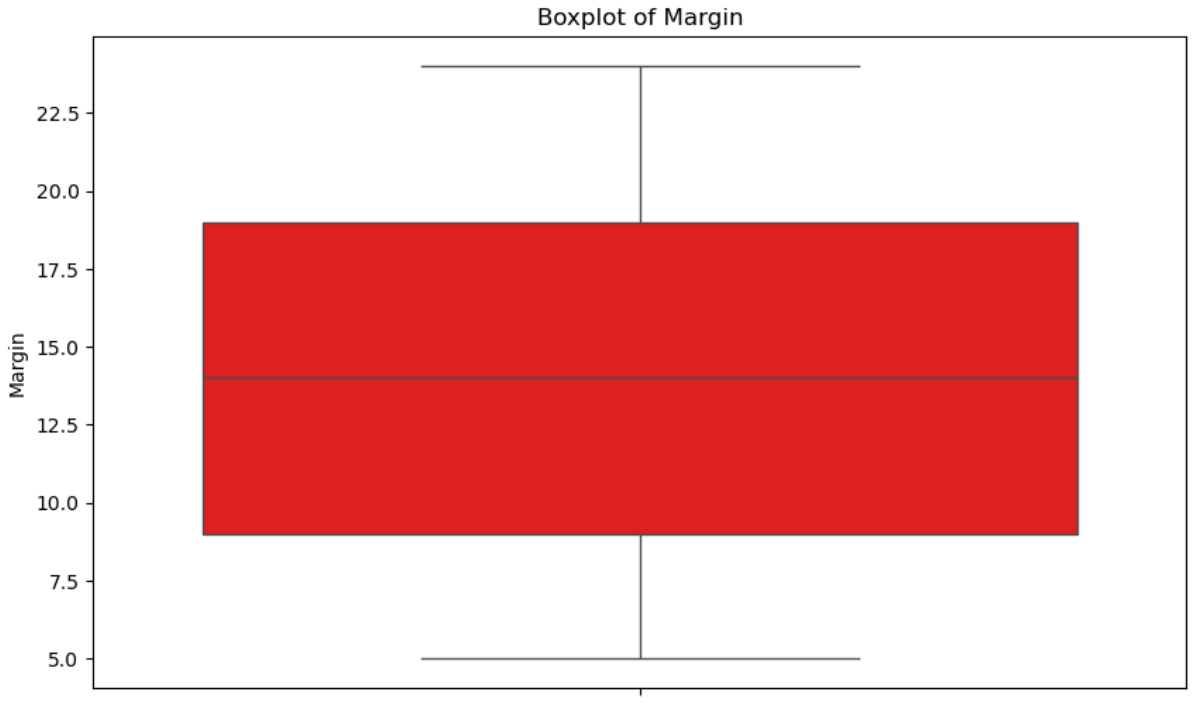


In [50]:
```
# Scatter Plot: Revenue vs. Margin
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['Current Revenue'], y=df['Margin'])
plt.title('Scatter Plot of Revenue vs. Margin')
plt.xlabel('Current Revenue')
plt.ylabel('Margin')
plt.show()
```



In [51]:
```
# Box Plot for Outlier Detection
plt.figure(figsize=(10, 6))
sns.boxplot(y=df['Current Revenue'])
plt.title('Boxplot of Current Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(y=df['Margin'], color='red')
plt.title('Boxplot of Margin')
plt.show()
```



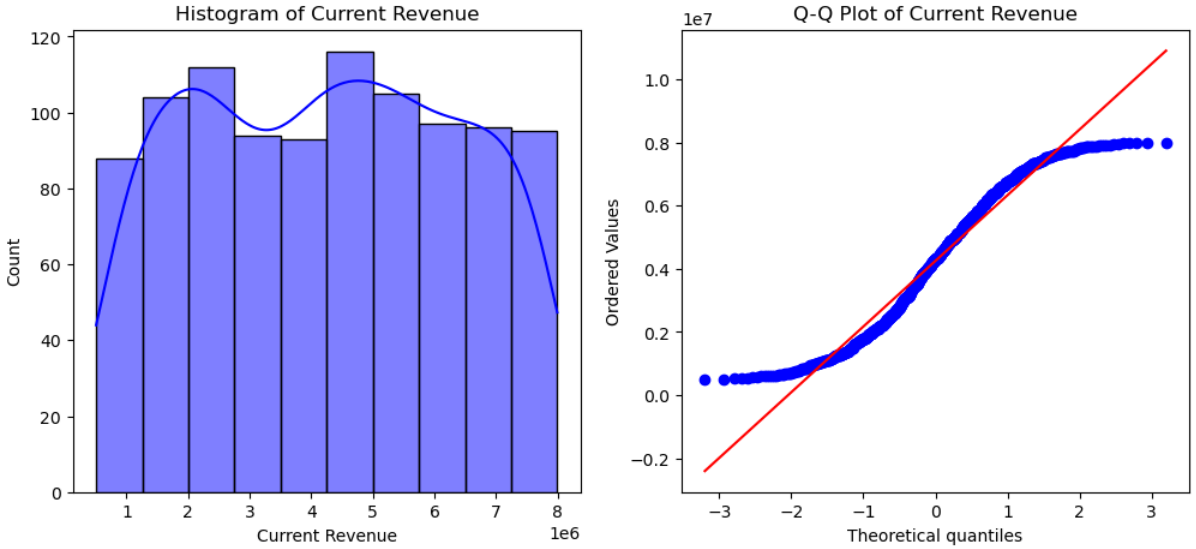


performing Hypothesis testing weather the revenue follows normal distribution

In [76]:
```
import scipy.stats as stats
# Checking for Normality Using Shapiro-Wilk Test
shapiro_stat, shapiro_p_value = stats.shapiro(df['Current Revenue'])

# Checking for Normality Using Kolmogorov-Smirnov Test
```

```
ks_stat, ks_p_value = stats.kstest(df['Current Revenue'], 'norm', args

# Printing the results
print(f"Shapiro-Wilk Test for Normality: Statistic={shapiro_stat:.4f},
if shapiro_p_value > 0.05:
    print("Shapiro-Wilk Test: The revenue data is normally distributed
else:
    print("Shapiro-Wilk Test: The revenue data is not normally distrib

print(f"Kolmogorov-Smirnov Test for Normality: Statistic={ks_stat:.4f}
if ks_p_value > 0.05:
    print("Kolmogorov-Smirnov Test: The revenue data is normally distr
else:
    print("Kolmogorov-Smirnov Test: The revenue data is not normally d

# Plotting Histogram and Q-Q Plot to Visualize Distribution
plt.figure(figsize=(12, 5))

# Histogram with KDE
plt.subplot(1, 2, 1)
sns.histplot(df['Current Revenue'], bins=10, kde=True, color='blue')
plt.title("Histogram of Current Revenue")

# Q-Q Plot
plt.subplot(1, 2, 2)
stats.probplot(df['Current Revenue'], dist="norm", plot=plt)
plt.title("Q-Q Plot of Current Revenue")

plt.show()
```

```
Shapiro-Wilk Test for Normality: Statistic=0.9575, p-value=0.0000
Shapiro-Wilk Test: The revenue data is not normally distributed.
Kolmogorov-Smirnov Test for Normality: Statistic=0.0680, p-value=0.0002
Kolmogorov-Smirnov Test: The revenue data is not normally distributed.
```



In [ ]:
```
# improve the model
```

In [80]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt
import seaborn as sns
```

In [81]:
```
# Load the dataset from your local path
file_path = "/Users/mohithreddy/Desktop/Python/Expanded_Acme_Data.xlsx
df = pd.read_excel(file_path)

# Quick look at the first few rows
print("Data Preview:")
print(df.head())
```

```
Data Preview:
      Segment           Brand         Category      Geography  \
0       Toner           Aveda     Face Make-Up  South America
1  Foundation           Aveda  Make-Up Brushes  South America
2       Toner  Frederic Malle     Face Make-Up  North America
3       Toner     Bobby Brown        Fragrance  North America
4     Shampoo  Frederic Malle            Tools           Asia

                    Portfolio  Current Revenue  Margin  Min Trend  Max
Trend  \
0                   Skin/Body          3944025       7         -3
5
1                   Skin/Body          3476493       9         -2
7
2                   Hair/APDO          6271132      23         -3
3
3                   Skin/Body          2182713      16         -2
7
4  Fragrance + Color Cosmetics          4612734      12         -3
4

   Min Contribution  Max Contribution
0                 5                10
1                 8                22
2                 7                14
3                 6                19
4                 9                16
```

In [82]:
```
# Create a binary target: 1 if above median, 0 if below
df['HighRevenue'] = (df['Current Revenue'] > df['Current Revenue'].med
```

```
# Define feature set (adjust based on your columns)
features = ['Margin', 'Max Trend', 'Max Contribution']  # Example feat
X = df[features]
y = df['HighRevenue']

# Optional: check for missing values or do data cleaning if necessary
print("\nMissing values per column:")
print(df.isnull().sum())
```

```
Missing values per column:
Segment             0
Brand               0
Category            0
Geography           0
Portfolio           0
Current Revenue     0
Margin              0
Min Trend           0
Max Trend           0
Min Contribution    0
Max Contribution    0
HighRevenue         0
dtype: int64
```

In [83]:
```
# Stratify ensures proportional distribution of the target
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
print(f"\nTrain shape: {X_train.shape}, Test shape: {X_test.shape}")
```

```
Train shape: (800, 3), Test shape: (200, 3)
```

In [96]:
```
# 4. Build a Pipeline & Tune KNN with GridSearchCV
# Create a pipeline that scales features and applies the KNN classifie
pipe_knn = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

# Define the parameter grid for tuning KNN hyperparameters.
param_grid_knn = {
    'knn__n_neighbors': [3, 5, 7, 9, 11],
    'knn__weights': ['uniform', 'distance'],
    'knn__metric': ['euclidean', 'manhattan']
}

# Use GridSearchCV to find the best parameters using ROC AUC as scorin
grid_knn = GridSearchCV(pipe_knn, param_grid_knn, cv=5, scoring='roc_a
grid_knn.fit(X_train, y_train)
```

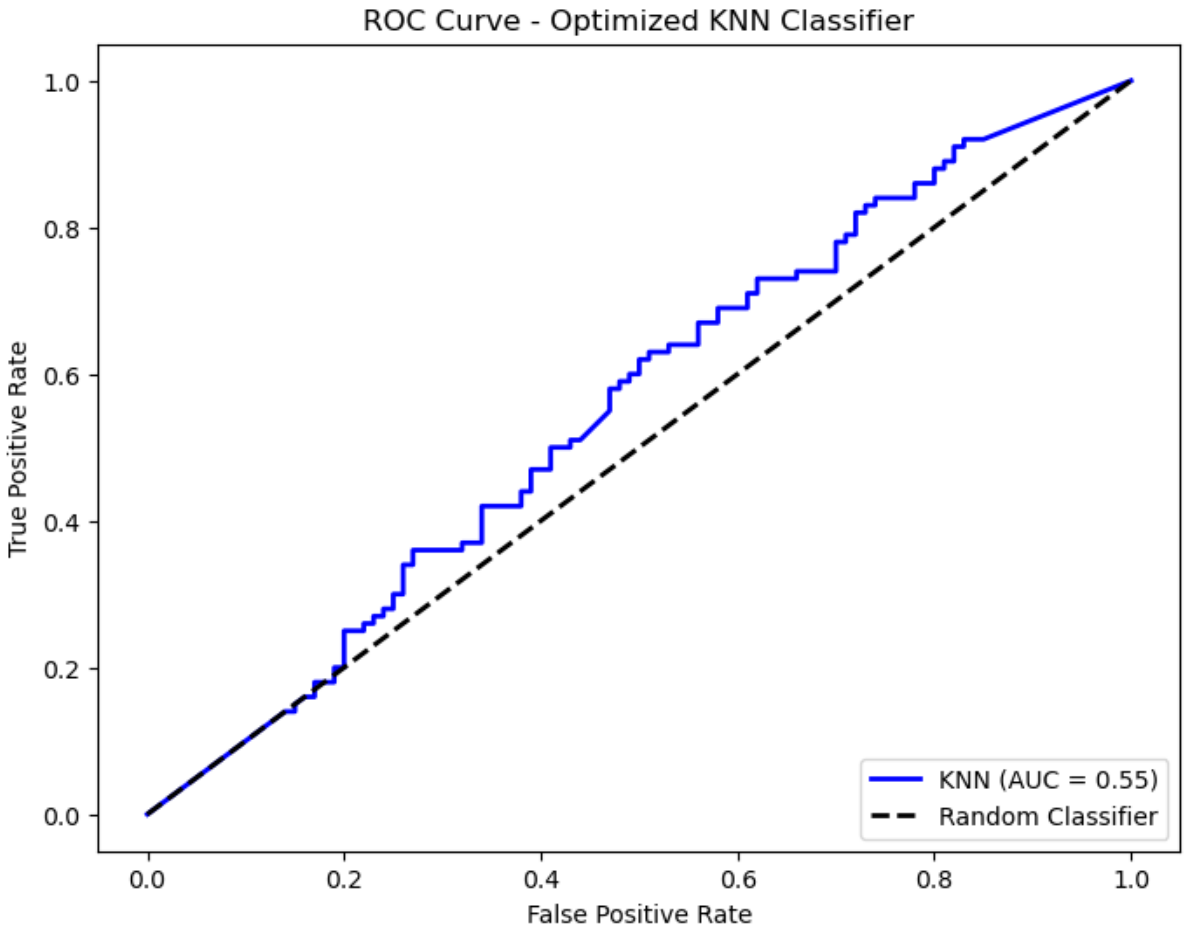
```
# Best estimator from GridSearchCV
best_knn = grid_knn.best_estimator_
print("\nBest KNN Parameters:")
print(grid_knn.best_params_)
```

```
Best KNN Parameters:
{'knn__metric': 'manhattan', 'knn__n_neighbors': 11, 'knn__weights': 'd
istance'}
```

In [94]:
```
# 5. Evaluate the Optimized KNN Classifier
# Make predictions on the test set.
y_pred_knn = best_knn.predict(X_test)
y_prob_knn = best_knn.predict_proba(X_test)[:, 1]

# Calculate accuracy and ROC AUC.
accuracy_knn = accuracy_score(y_test, y_pred_knn)
auc_knn = roc_auc_score(y_test, y_prob_knn)

print("\nEvaluation Metrics for KNN:")
print(f"Accuracy: {accuracy_knn:.2f}")
print(f"AUC: {auc_knn:.2f}")
```

```
Evaluation Metrics for KNN:
Accuracy: 0.54
AUC: 0.55
```

In [95]:
```
# 6. Plot the ROC Curve for KNN

fpr, tpr, _ = roc_curve(y_test, y_prob_knn)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'KNN (AUC = {auc_knn:.2f})', color='blue', l
plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Optimized KNN Classifier')
plt.legend(loc='lower right')
plt.show()
```


ROC Curve - Optimized KNN Classifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
KNN (AUC = 0.55)
Random Classifier

In [ ]: