# Campaign and Adviser Performance Optimization Analysis

## Objective :

The objective of this analysis is to evaluate the effectiveness of marketing campaigns and call center advisers in converting applications to customers. This includes assessing each campaign’s interest and conversion rates, alongside the return on marketing spend, to identify both high-performing and underperforming campaigns. Additionally, adviser and call center conversion rates are analyzed to highlight top performers and opportunities for improvement. Insights gained will guide resource allocation and strategy adjustments, ultimately supporting more effective customer acquisition and better returns on marketing and operational investments.

## 1. Data Exploration and Cleaning

In [1]:

```
# Importing relevant libraries

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

In [2]:

```
# Loading the Campaign dataset

campaigns_pd = pd.read_excel('Campaigns.xlsx')
```

In [3]:

```
# Determining the no. of records in our dataset
campaigns_pd.shape
```

Out[3]:

```
(376, 13)
```

In [4]:

```
campaigns_pd.columns
```

Out[4]:

```
Index(['Campaign', 'Applications generated', 'Applications interested',
       'Customers converted', 'Average value of Customers', 'Marketing Spend',
       'Cost Per Application', 'Cost Per Conversion', 'Interest Rate',
       'Conversion Rate', 'Total Revenue', 'Return on Investment',
       'Net Profit'],
      dtype='object')
```

In [5]:

```
# Checking for nulls
campaigns_pd.isnull().sum()
```

Out[5]:

```
Campaign                       0
Applications generated         0
Applications interested        0
Customers converted            0
Average value of Customers     0
Marketing Spend                0
Cost Per Application           0
Cost Per Conversion            0
Interest Rate                  0
Conversion Rate                0
Total Revenue                  0
Return on Investment           0
Net Profit                     0
dtype: int64
```

In [6]:

```
campaigns_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 376 entries, 0 to 375
Data columns (total 13 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Campaign                    376 non-null    object 
 1   Applications generated      376 non-null    int64  
 2   Applications interested     376 non-null    int64  
 3   Customers converted         376 non-null    float64
 4   Average value of Customers  376 non-null    int64  
 5   Marketing Spend             376 non-null    float64
 6   Cost Per Application        376 non-null    float64
 7   Cost Per Conversion         376 non-null    float64
 8   Interest Rate               376 non-null    float64
 9   Conversion Rate             376 non-null    float64
 10  Total Revenue               376 non-null    float64
 11  Return on Investment        376 non-null    float64
 12  Net Profit                  376 non-null    float64
dtypes: float64(9), int64(3), object(1)
memory usage: 38.3+ KB
```

In [7]:

```
advisers_pd = pd.read_excel('Advisers.xlsx')
```

In [8]:

```
advisers_pd.shape
```

Out[8]:

```
(361, 5)
```

In [9]:

```
advisers_pd.columns
```

Out[9]:

```
Index(['Call Centre', 'Adviser', 'Applications received',
       'Applications converted to customers', 'Advisers' Conversion Rate'],
      dtype='object')
```

In [10]:

```
# Checking for nulls
advisers_pd.isnull().sum()
```

Out[10]:

```
Call Centre                            0
Adviser                                0
Applications received                  0
Applications converted to customers    0
Advisers' Conversion Rate              0
dtype: int64
```

In [11]:

```
advisers_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 361 entries, 0 to 360
Data columns (total 5 columns):
 #   Column                               Non-Null Count  Dtype  
---  ------                               --------------  -----  
 0   Call Centre                          361 non-null    object 
 1   Adviser                              361 non-null    object 
 2   Applications received                361 non-null    int64  
 3   Applications converted to customers  361 non-null    int64  
 4   Advisers' Conversion Rate            361 non-null    float64
dtypes: float64(1), int64(2), object(2)
memory usage: 14.2+ KB
```

# 2. Statistical Analysis

In [12]:

```
# Exploring the descriptive statistics of the variables
campaigns_pd.describe(include='all')
```

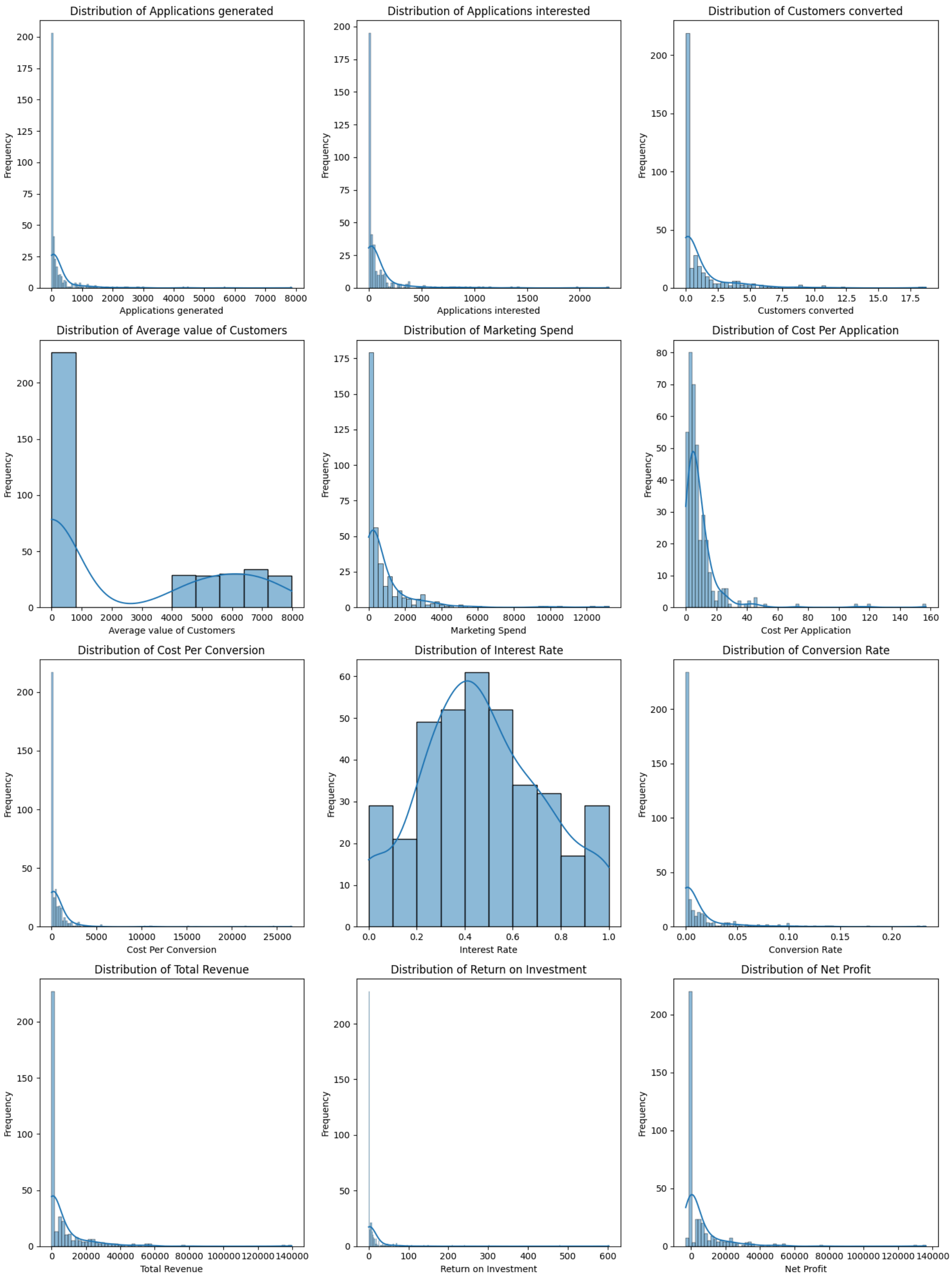
Out[12]:

| | Campaign | Applications generated | Applications interested | Customers converted | Average value of Customers | Marketing Spend | C App |
|---|---|---|---|---|---|---|---|
| **count** | 376 | 376.000000 | 376.000000 | 376.000000 | 376.000000 | 376.000000 | 376 |
| **unique** | 376 | NaN | NaN | NaN | NaN | NaN | |
| **top** | Campaign 376 | NaN | NaN | NaN | NaN | NaN | |
| **freq** | 1 | NaN | NaN | NaN | NaN | NaN | |
| **mean** | NaN | 281.468085 | 102.018617 | 1.163298 | 2374.077128 | 889.594299 | 9 |
| **std** | NaN | 735.377276 | 246.497247 | 2.327203 | 3019.391500 | 1606.414140 | 13 |
| **min** | NaN | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0 |
| **25%** | NaN | 11.000000 | 5.750000 | 0.000000 | 0.000000 | 98.462500 | 3 |
| **50%** | NaN | 43.000000 | 18.500000 | 0.100000 | 0.000000 | 283.355000 | 5 |
| **75%** | NaN | 194.000000 | 80.500000 | 1.100000 | 5563.250000 | 1045.550000 | 11 |
| **max** | NaN | 7865.000000 | 2276.000000 | 18.700000 | 7981.000000 | 13217.920000 | 156 |

In [13]:

```
columns_to_analyze = [col for col in campaigns_pd.columns if col != 'Campaign']
```

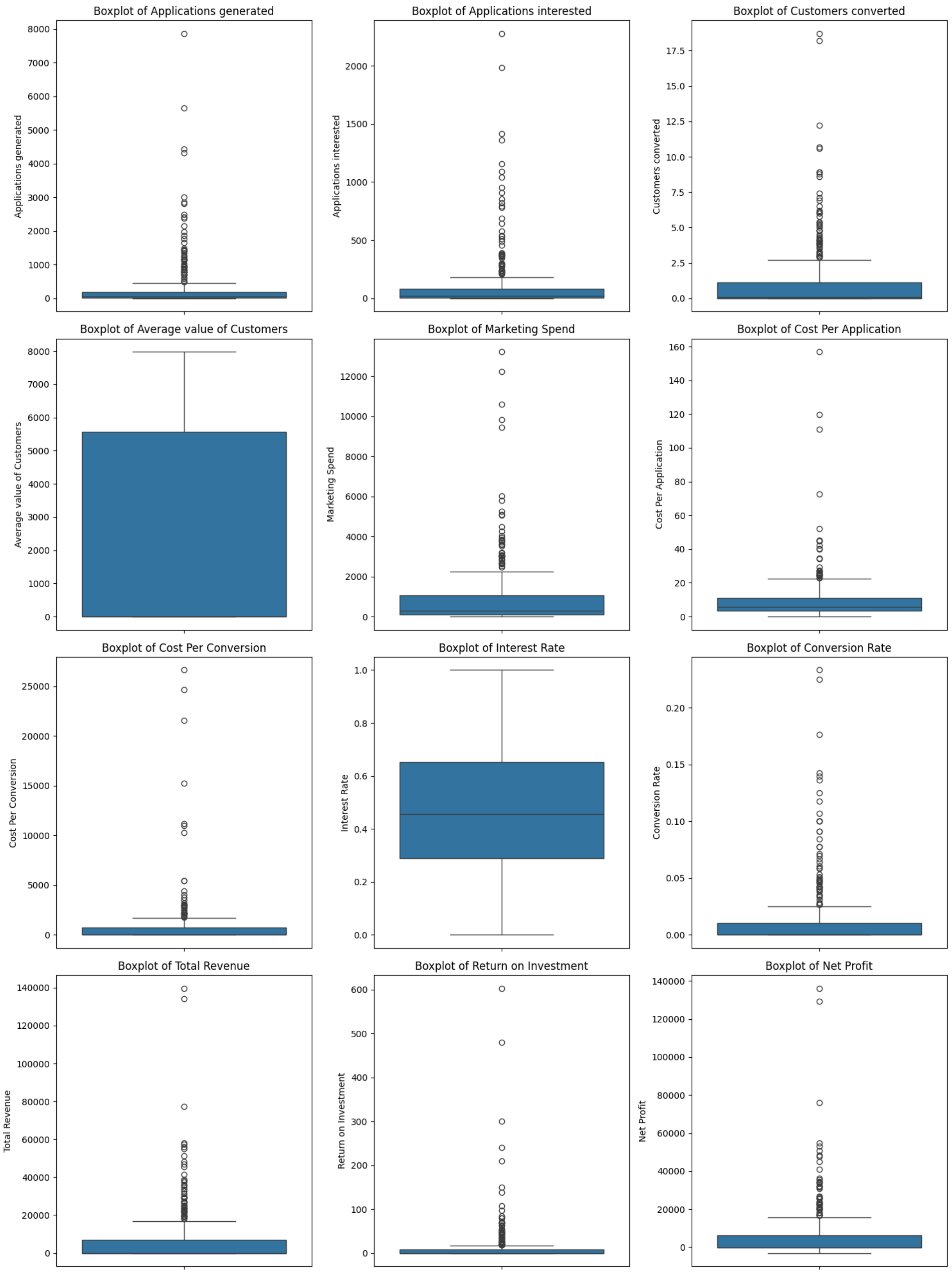
```
# Calculating the number of columns and creating subplots accordingly
num_cols = len(columns_to_analyze)
num_rows = (num_cols + 2) // 3  # Calculate number of rows needed
plt.figure(figsize=(15, 5 * num_rows))

# Iterating through columns in groups of 3
for i in range(0, num_cols, 3):
    # Determining the columns for this subplot
    cols_in_group = columns_to_analyze[i:min(i + 3, num_cols)]

    # Creating a subplot for each group
    for j, col in enumerate(cols_in_group):
        plt.subplot(num_rows, 3, i + j + 1)
        sns.histplot(campaigns_pd[col], kde=True)
        plt.title(f'Distribution of {col}')
        plt.xlabel(col)
        plt.ylabel('Frequency')

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



In [14]:

```
columns_to_analyze = [col for col in campaigns_pd.columns if col != 'Campaign']

# Calculating the number of columns and creating subplots accordingly
num_cols = len(columns_to_analyze)
num_rows = (num_cols + 2) // 3  # Calculate number of rows needed
plt.figure(figsize=(15, 5 * num_rows))
```

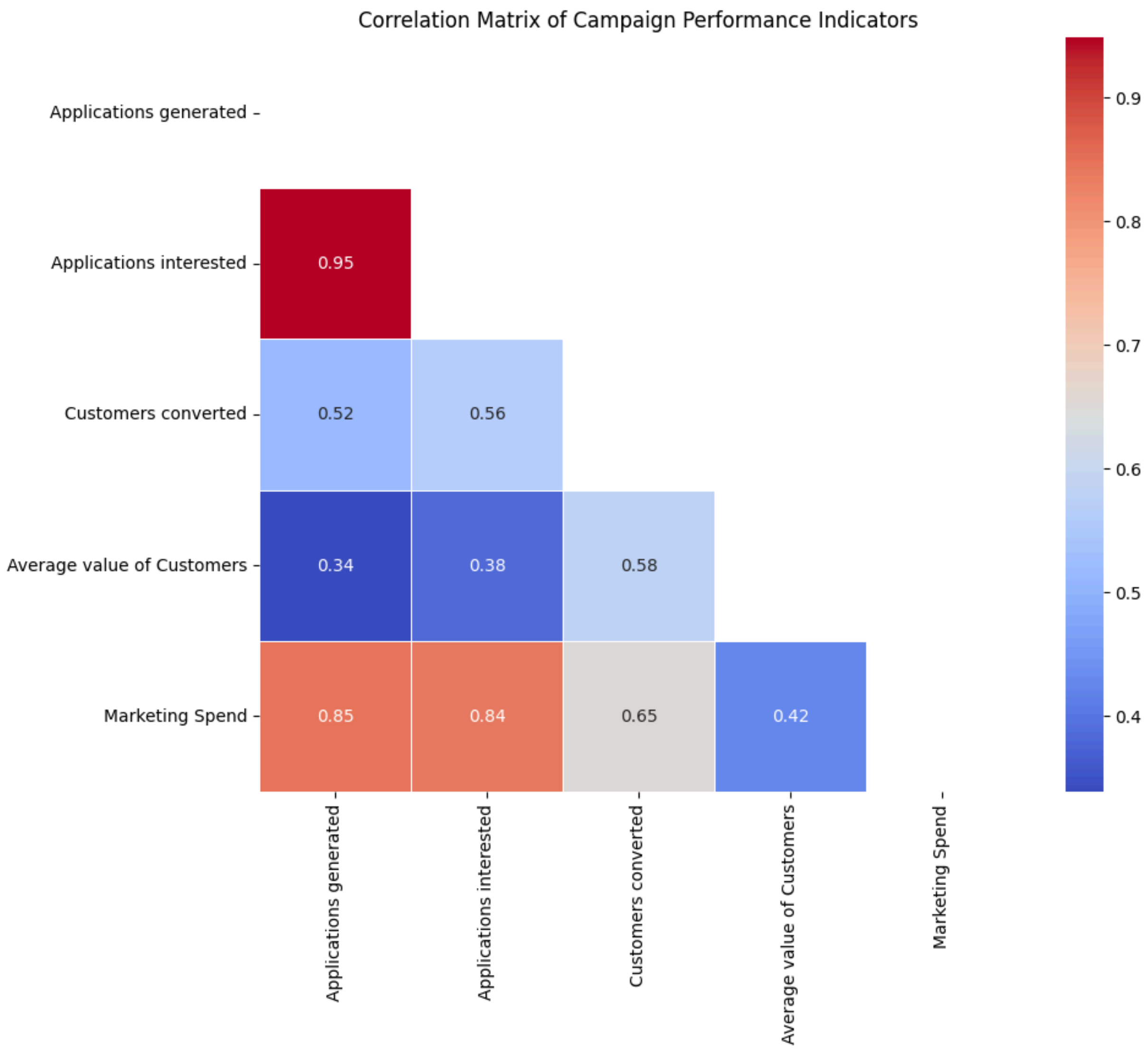
```python
# Iterating through columns in groups of 3
for i in range(0, num_cols, 3):
    # Determining the columns for this subplot
    cols_in_group = columns_to_analyze[i:min(i + 3, num_cols)]

    # Creating a subplot for each group
    for j, col in enumerate(cols_in_group):
        plt.subplot(num_rows, 3, i + j + 1)
        sns.boxplot(y=campaigns_pd[col])
        plt.title(f'Boxplot of {col}')
        plt.ylabel(col)

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



In [15]:

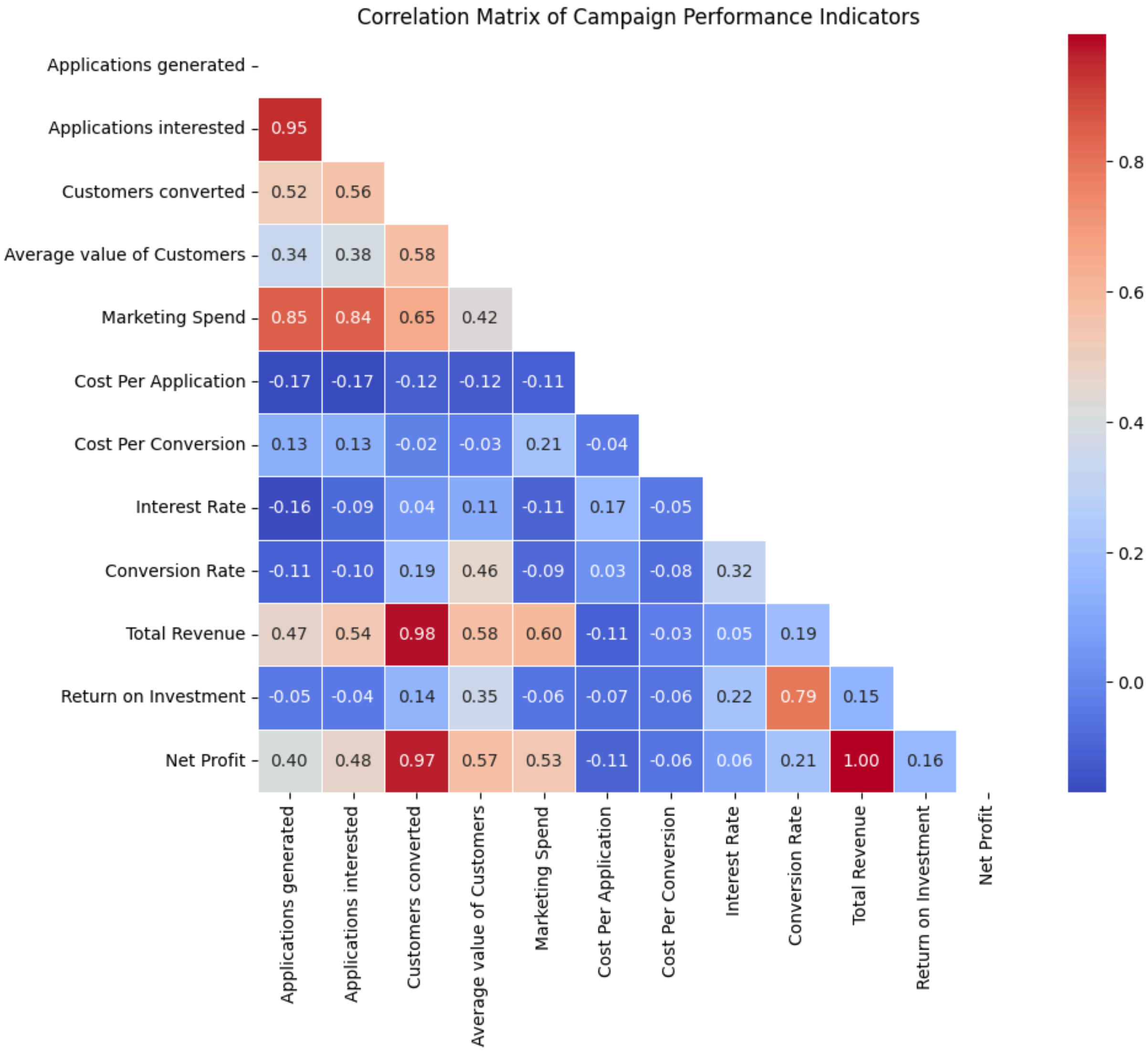
```
columns_for_heatmap = ['Applications generated', 'Applications interested',
                       'Customers converted', 'Average value of Customers', 'Market

# Creating a correlation matrix for the selected columns
correlation_matrix = campaigns_pd[columns_for_heatmap].corr()

# Creating a mask to hide the upper triangle of the heatmap
```

```
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, mask=mask, cmap='coolwarm', fmt=".2f",
plt.title('Correlation Matrix of Campaign Performance Indicators')
plt.show()
```



In [16]:

```
columns_to_analyze = [col for col in campaigns_pd.columns if col != 'Campaign']

# Creating a correlation matrix for the selected columns
correlation_matrix = campaigns_pd[columns_to_analyze].corr()

# Creating a mask to hide the upper triangle of the heatmap
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, mask=mask, cmap='coolwarm', fmt=".2f",
plt.title('Correlation Matrix of Campaign Performance Indicators')
plt.show()
```



In [17]:

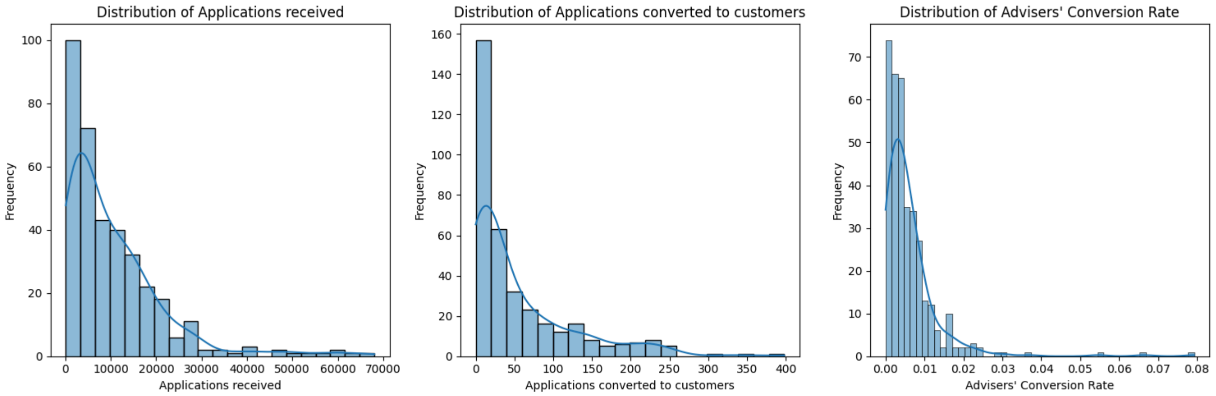
```
advisers_pd.describe(include='all')
```

Out[17]:

|  | Call Centre | Adviser | Applications received | Applications converted to customers | Advisers' Conversion Rate |
|---|---|---|---|---|---|
| **count** | 361 | 361 | 361.000000 | 361.000000 | 361.000000 |
| **unique** | 12 | 361 | NaN | NaN | NaN |
| **top** | A | Adviser H11 | NaN | NaN | NaN |
| **freq** | 78 | 1 | NaN | NaN | NaN |
| **mean** | NaN | NaN | 10369.581717 | 54.296399 | 0.006025 |
| **std** | NaN | NaN | 10998.414653 | 68.771305 | 0.007806 |
| **min** | NaN | NaN | 101.000000 | 0.000000 | 0.000000 |
| **25%** | NaN | NaN | 2893.000000 | 6.000000 | 0.001985 |
| **50%** | NaN | NaN | 7219.000000 | 26.000000 | 0.004193 |
| **75%** | NaN | NaN | 14502.000000 | 77.000000 | 0.007577 |
| **max** | NaN | NaN | 68074.000000 | 398.000000 | 0.079268 |

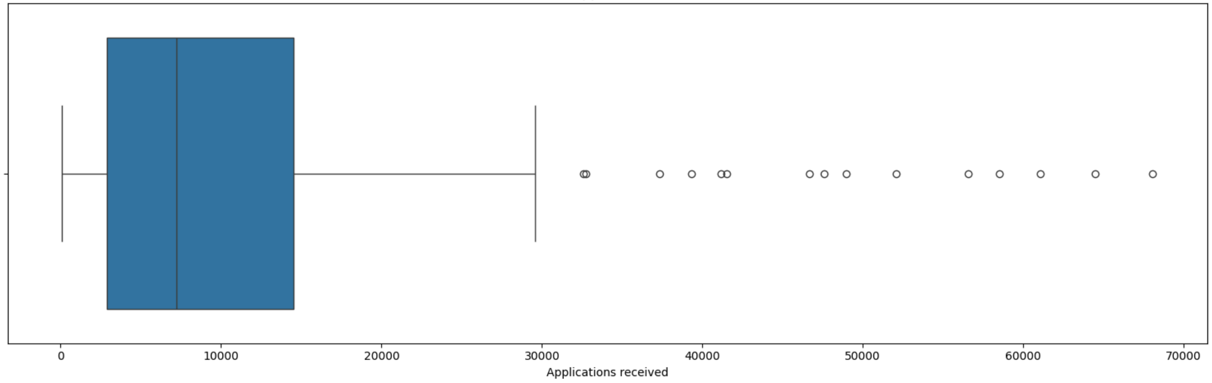
In [18]:

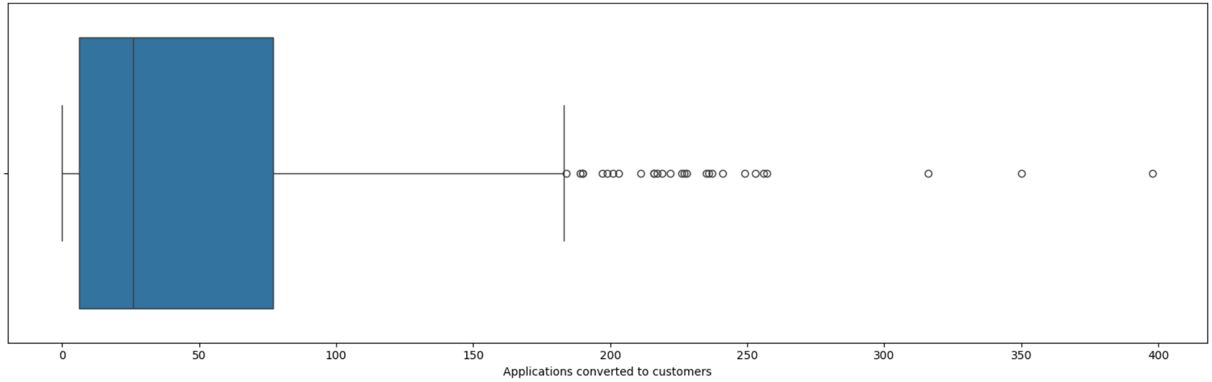
```python
columns_to_analyze = [col for col in advisers_pd.columns if col not in ['Call Centr

# Calculating the number of columns and creating subplots accordingly
num_cols = len(columns_to_analyze)
num_rows = (num_cols + 2) // 3  # Calculate number of rows needed
plt.figure(figsize=(15, 5 * num_rows))

# Iterating through columns in groups of 3
for i in range(0, num_cols, 3):
    # Determining the columns for this subplot
    cols_in_group = columns_to_analyze[i:min(i + 3, num_cols)]

    # Creating a subplot for each group
    for j, col in enumerate(cols_in_group):
        plt.subplot(num_rows, 3, i + j + 1)
        sns.histplot(advisers_pd[col], kde=True)
        plt.title(f'Distribution of {col}')
        plt.xlabel(col)
        plt.ylabel('Frequency')

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

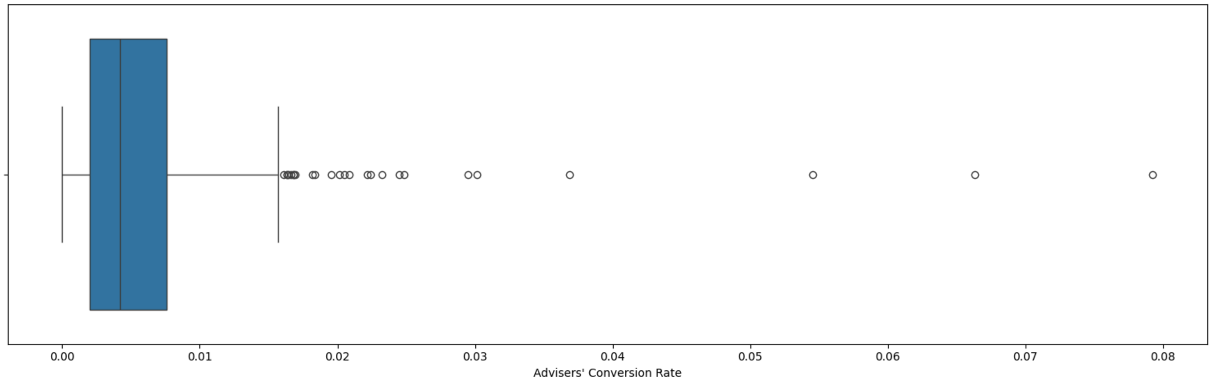


In [19]:

```
columns_to_plot = [col for col in advisers_pd.columns if col not in ['Call Centre',

plt.figure(figsize=(15, 5 * len(columns_to_plot)))

for i, col in enumerate(columns_to_plot):
  plt.subplot(len(columns_to_plot), 1, i + 1)
  sns.boxplot(x=advisers_pd[col])
  plt.title(f'Box Plot of {col}')

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


Box Plot of Applications received


Box Plot of Applications converted to customers


Box Plot of Advisers' Conversion Rate