HNG TASK 2: Exploratory Data Analysis (EDA) on Marketing Campaign Dataset

Data Loading and Initial Exploration

Importing Essential Libraries: To begin with, the necessary Python libraries were i mported such as pandas for data manipulation, numpy for numerical computations and matplotlib/seaborn for data visualization.

In [1]:
```
# Import Data Exploratory Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the Dataset

It is essential to laod the dataset to know the shape of the data and it's structures

In [3]:
```
#Initial Data Explorations
# Read DataFrame
df = pd.read_csv("marketing_campaign_dataset.csv")
```

In [5]:
```
# Display first few rows
print("First 5 rows of the dataset:")
df.head()
```

```
First 5 rows of the dataset:
```

Out[5]:

	Campaign_ID	Company	Campaign_Type	Target_Audience	Duration	Channel_Used	Conversion_Rate	Acquisition_Cost	ROI
0	1	Innovate Industries	Email	Men 18-24	30 days	Google Ads	0.04	$16,174.00	6.29
1	2	NexGen Systems	Email	Women 35-44	60 days	Google Ads	0.12	$11,566.00	5.61
2	3	Alpha Innovations	Influencer	Men 25-34	30 days	YouTube	0.07	$10,200.00	7.18
3	4	DataTech Solutions	Display	All Ages	60 days	YouTube	0.11	$12,724.00	5.55
4	5	NexGen Systems	Email	Men 25-34	15 days	YouTube	0.05	$16,452.00	6.50

In [7]:
```
# Check Data Shape
df.shape
```

Out[7]: (200005, 15)

In [9]:
```
# Display dataset info
print("\nDataset Information:")
df.info()
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200005 entries, 0 to 200004
Data columns (total 15 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Campaign_ID       200005 non-null  int64  
 1   Company           200005 non-null  object 
 2   Campaign_Type     200005 non-null  object 
 3   Target_Audience   200005 non-null  object 
 4   Duration          200005 non-null  object 
 5   Channel_Used      200005 non-null  object 
 6   Conversion_Rate   200005 non-null  float64
 7   Acquisition_Cost  200005 non-null  object 
 8   ROI               200005 non-null  float64
 9   Location          200005 non-null  object 
 10  Date              200005 non-null  object 
 11  Clicks            200005 non-null  int64  
 12  Impressions       200005 non-null  int64  
 13  Engagement_Score  200005 non-null  int64  
 14  Customer_Segment  200005 non-null  object 
dtypes: float64(2), int64(4), object(9)
memory usage: 22.9+ MB
```

The dataset was read and it was revealed that the dataset contains 200,005 rows and 15 columns. The first few records provided an overview of key variables, including Campaign_ID, Company, Campaign_Type, Target_Audience, Duration, Channel_Used, Conversion_Rate, Acquisition_Cost, ROI and Location. Also, it was found that the dataset contain categorical, numerical and date-based columns

Initial Data Exploratory

Now, preprocessing analysis will be carried out to know if our data in clean or not

In [11]:
```
# Convert 'Acquisition_Cost' to numeric
df['Acquisition_Cost'] = df['Acquisition_Cost'].replace('[\$,]', '', regex=True).astype(float)
df['Acquisition_Cost']
```

```
<>:2: SyntaxWarning: invalid escape sequence '\$'
<>:2: SyntaxWarning: invalid escape sequence '\$'
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\82157342.py:2: SyntaxWarning: invalid escape sequence '\$'
  df['Acquisition_Cost'] = df['Acquisition_Cost'].replace('[\$,]', '', regex=True).astype(float)
```

Out[11]:
```
0         16174.0
1         11566.0
2         10200.0
3         12724.0
4         16452.0
           ...
200000    18365.0
200001     8168.0
200002    13397.0
200003    18508.0
200004    13835.0
Name: Acquisition_Cost, Length: 200005, dtype: float64
```

In [13]:
```
# Convert 'Date' to datetime format
df['Date'] = pd.to_datetime(df['Date'], format="%d/%m/%Y")
df['Date']
```

Out[13]:
```
0        2021-01-01
1        2021-02-01
2        2021-03-01
3        2021-04-01
4        2021-05-01
              ...
200000   2021-07-12
200001   2021-08-12
200002   2021-09-12
200003   2021-10-12
200004   2021-11-12
Name: Date, Length: 200005, dtype: datetime64[ns]
```

In [15]:
```
# Extract numerical values from 'Duration'
df['Duration'] = df['Duration'].astype(str).str.extract('(\d+)').astype(float)
df['Duration']
```

```
<>:2: SyntaxWarning: invalid escape sequence '\d'
<>:2: SyntaxWarning: invalid escape sequence '\d'
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\814243577.py:2: SyntaxWarning: invalid escape sequence '\d'
  df['Duration'] = df['Duration'].astype(str).str.extract('(\d+)').astype(float)
```

Out[15]:
```
0         30.0
1         60.0
2         30.0
3         60.0
4         15.0
          ...
200000    30.0
200001    15.0
200002    45.0
200003    30.0
200004    45.0
Name: Duration, Length: 200005, dtype: float64
```

In [17]:
```
# Check for missing values
print("\nMissing Values:")
df.isnull().sum()
```

```
Missing Values:
```

Out[17]:
```
Campaign_ID          0
Company              0
Campaign_Type        0
Target_Audience      0
Duration             0
Channel_Used         0
Conversion_Rate      0
Acquisition_Cost     0
ROI                  0
Location             0
Date                 0
Clicks               0
Impressions          0
Engagement_Score     0
Customer_Segment     0
dtype: int64
```

In [19]:
```
# Summary statistics for numerical columns
print("\nSummary Statistics:")
df.describe()
```

```
Summary Statistics:
```

Out[19]:

	Campaign_ID	Duration	Conversion_Rate	Acquisition_Cost	ROI	Date	Clicks	Imp
count	200005.000000	200005.000000	200005.000000	200005.000000	200005.000000	200005	200005.000000	20000
mean	100003.000000	37.503862	0.080069	12504.441794	5.002416	2021-07-01 23:37:44.289392896	549.774591	550
min	1.000000	15.000000	0.010000	5000.000000	2.000000	2021-01-01 00:00:00	100.000000	100
25%	50002.000000	30.000000	0.050000	8740.000000	3.500000	2021-04-02 00:00:00	325.000000	326
50%	100003.000000	30.000000	0.080000	12497.000000	5.010000	2021-07-02 00:00:00	550.000000	551
75%	150004.000000	45.000000	0.120000	16264.000000	6.510000	2021-10-01 00:00:00	775.000000	775
max	200005.000000	60.000000	0.150000	20000.000000	8.000000	2021-12-31 00:00:00	1000.000000	1000
std	57736.614632	16.746620	0.040602	4337.663210	1.734485	NaN	260.019354	259

In [23]:
```
# List of categorical columns
categorical_cols = ['Company', 'Campaign_Type', 'Target_Audience',
                    'Channel_Used', 'Location', 'Customer_Segment']
# Loop through each column and print unique values
for col in categorical_cols:
    print(f"\nUnique values in {col}:\n", df[col].unique())
```

```
Unique values in Company:
 ['Innovate Industries' 'NexGen Systems' 'Alpha Innovations'
 'DataTech Solutions' 'TechCorp']

Unique values in Campaign_Type:
 ['Email' 'Influencer' 'Display' 'Search' 'Social Media']

Unique values in Target_Audience:
 ['Men 18-24' 'Women 35-44' 'Men 25-34' 'All Ages' 'Women 25-34']

Unique values in Channel_Used:
 ['Google Ads' 'YouTube' 'Instagram' 'Website' 'Facebook' 'Email']

Unique values in Location:
 ['Chicago' 'New York' 'Los Angeles' 'Miami' 'Houston']

Unique values in Customer_Segment:
 ['Health & Wellness' 'Fashionistas' 'Outdoor Adventurers' 'Foodies'
 'Tech Enthusiasts']
```

In [25]:
```
channel_performance = df.groupby("Channel_Used").agg({
 "ROI": "mean",
 "Conversion_Rate": "mean",
 "Clicks": "sum",
 "Impressions": "sum"
}).reset_index()

channel_performance.sort_values(by="ROI", ascending=False, inplace=True)
channel_performance
```

Out[25]:

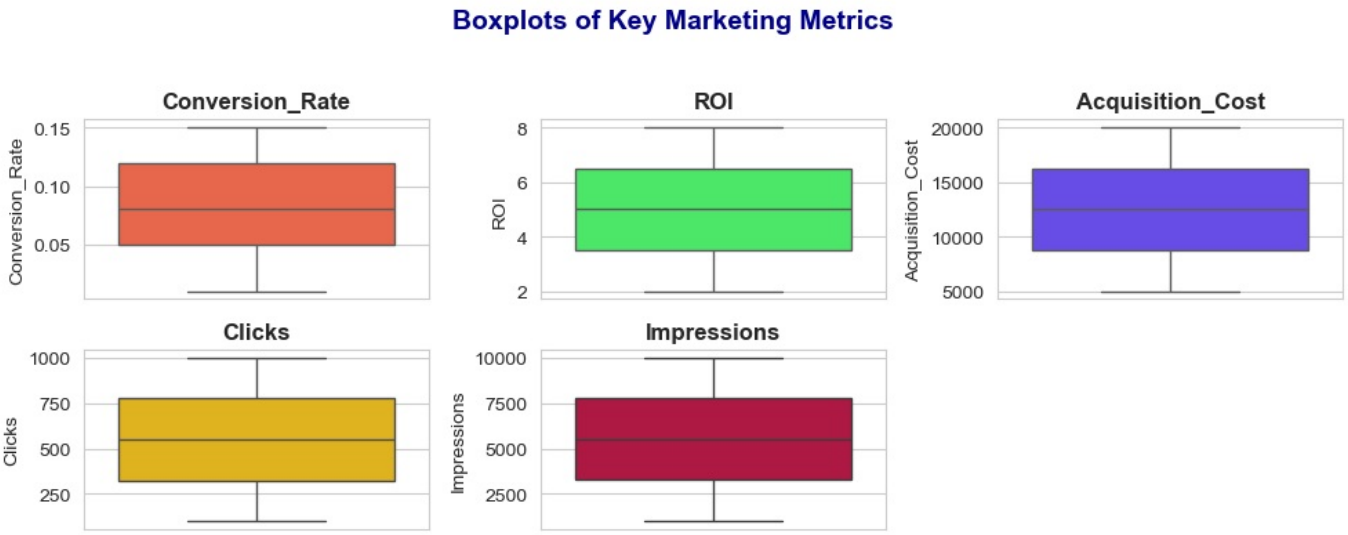
	Channel_Used	ROI	Conversion_Rate	Clicks	Impressions
1	Facebook	5.018672	0.079990	18038175	180662496
4	Website	5.014114	0.080182	18415351	183815901
2	Google Ads	5.003126	0.080181	18342589	185020154
0	Email	4.996487	0.080282	18493963	184801107
5	YouTube	4.993720	0.079890	18350935	183450845
3	Instagram	4.988706	0.079886	18316654	183738455

The dataset was cleaned by converting Acquisition Cost to numeric, transforming dates, extracting numeric values from Duration, and confirming no missing values. Key statistics showed an 8% conversion rate, a $12,504 average acquisition cost, and a 5.00 mean ROI. Campaigns averaged 550 clicks and 5,500 impressions.

KEY INSIGHTS

In [59]:

```
# Define numerical columns
numerical_cols = ["Conversion_Rate", "ROI", "Acquisition_Cost", "Clicks", "Impressions"]

# Set plot style
sns.set_style("whitegrid")

# Create boxplots with a better layout (2 rows, 3 columns)
fig, axes = plt.subplots(2, 3, figsize=(10, 4))
fig.suptitle("Boxplots of Key Marketing Metrics", fontsize=14, fontweight='bold', color='darkblue')

# Choose a visually appealing color
colors = ["#FF5733", "#33FF57", "#5733FF", "#FFC300", "#C70039"]

# Plot boxplots
for i, col in enumerate(numerical_cols):
    row, col_index = divmod(i, 3)  # Determine row and column index
    sns.boxplot(y=df[col], ax=axes[row, col_index], color=colors[i])
    axes[row, col_index].set_title(f"{col}", fontsize=12, fontweight='bold')

# Remove empty subplot if columns < 6
if len(numerical_cols) < 6:
    fig.delaxes(axes[1, 2])

# Adjust layout for better spacing
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig('Boxplots of Key Marketing Metrics')
plt.show()
```



In [31]:

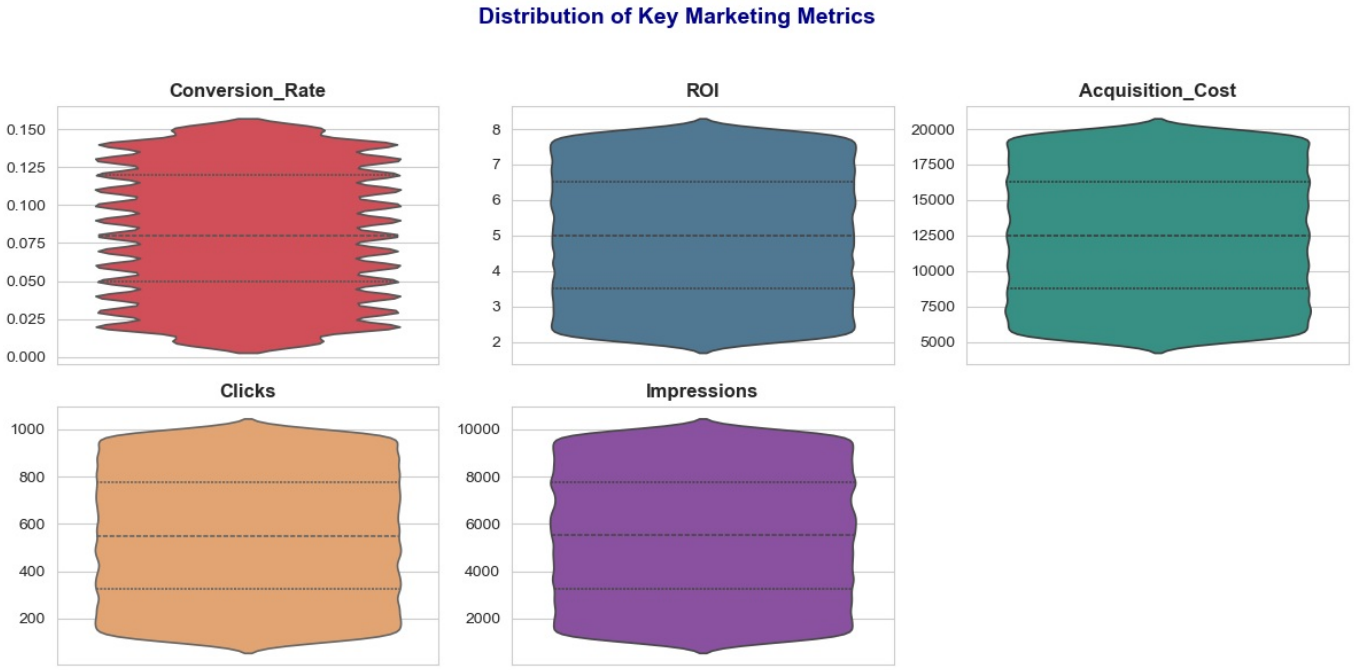
```
# Claculate CTR and CPC
# Compute Click-Through Rate (CTR) and Cost Per Click (CPC)
df["CTR"] = (df["Clicks"] / df["Impressions"]) * 100  # CTR in percentage
df["CPC"] = df["Acquisition_Cost"] / df["Clicks"]  # CPC in dollars

# Summary statistics for new metrics
ctr_cpc_summary = df[["CTR", "CPC"]].describe()
print("CTR & CPC Summary:\n", ctr_cpc_summary)
```

```
CTR & CPC Summary:
                CTR            CPC
count  200005.000000  200005.000000
mean       14.040504      32.008319
std        13.087980      26.925841
min         1.005429       5.021084
25%         5.860637      15.092037
50%         9.978960      22.773973
75%        16.969848      38.598253
max        99.202393     199.960000
```

In [33]:
```
location_performance = df.groupby("Location").agg({
"ROI": "mean",
 "Conversion_Rate": "mean",
 "Clicks": "sum",
 "Impressions": "sum"
}).reset_index()
print(location_performance)
```

```
      Location       ROI  Conversion_Rate    Clicks  Impressions
0      Chicago  5.001555         0.080131  21980408    219999352
1      Houston  5.007174         0.079949  21893075    219129799
2  Los Angeles  5.010876         0.080013  21966553    219652325
3        Miami  5.012282         0.080047  22056765    221347726
4     New York  4.980185         0.080203  22060866    221359756
```

In [61]:
```
# Set plot style
sns.set_style("whitegrid")

# Create subplots (2 rows, 3 columns)
fig, axes = plt.subplots(2, 3, figsize=(12, 6))
fig.suptitle("Distribution of Key Marketing Metrics", fontsize=14, fontweight='bold', color='darkblue')

# Custom color palette (same for all charts)
colors = ["#E63946", "#457B9D", "#2A9D8F", "#F4A261", "#8E44AD"]

# Plot Violin Plots for all metrics
for i, col in enumerate(numerical_cols):
    row, col_index = divmod(i, 3)  # Arrange in 2 rows, 3 columns
    sns.violinplot(y=df[col], ax=axes[row, col_index], color=colors[i], inner="quartile", linewidth=1.2)
    axes[row, col_index].set_title(f"{col}", fontsize=12, fontweight='bold')
    axes[row, col_index].set_ylabel("")  # Remove unnecessary y-labels

# Remove empty subplot if less than 6 columns
if len(numerical_cols) < 6:
    fig.delaxes(axes[1, 2])

# Adjust layout for better spacing
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig('Distribution of Key Marketing Metrics')
plt.show()
```



In [39]:
```
target_audience_performance = df.groupby("Target_Audience").agg({
    "ROI": "mean",
 "Conversion_Rate": "mean",
 "Clicks": "sum",
 "Impressions": "sum"
```

```
}).reset_index()
target_audience_performance
```

Out[39]:

	Target_Audience	ROI	Conversion_Rate	Clicks	Impressions
0	All Ages	5.005091	0.079975	21966687	220361093
1	Men 18-24	4.982810	0.080239	22097525	221236185
2	Men 25-34	5.020605	0.080130	22014566	220391924
3	Women 25-34	4.997351	0.079899	22051647	220449734
4	Women 35-44	5.006371	0.080100	21827242	219050022

In [81]:

```
# Aggregate key metrics by marketing channel
channel_performance = df.groupby("Channel_Used").agg(
    Avg_CTR=("CTR", "mean"),
    Avg_CPC=("CPC", "mean"),
    Avg_ROI=("ROI", "mean"),
    Total_Clicks=("Clicks", "sum"),
    Total_Impressions=("Impressions", "sum")
).reset_index()

# Sort by highest ROI
channel_performance = channel_performance.sort_values(by="Avg_ROI", ascending=False)
print("Marketing Channel Performance:\n", channel_performance)
```

```
Marketing Channel Performance:
   Channel_Used    Avg_CTR    Avg_CPC   Avg_ROI  Total_Clicks  \
1      Facebook  14.049724  32.129366  5.018672      18038175
4       Website  14.096941  31.779148  5.014114      18415351
2    Google Ads  13.918943  32.308459  5.003126      18342589
0         Email  14.054269  31.881471  4.996487      18493963
5       YouTube  14.119755  31.872904  4.993720      18350935
3     Instagram  14.003691  32.080786  4.988706      18316654

   Total_Impressions
1          180662496
4          183815901
2          185020154
0          184801107
5          183450845
3          183738455
```

Marketing channel analysis showed that Facebook had the highest ROI (5.02), while Email led in conversion rate (8.03%). Click-through rates averaged 14%, with CPC at $32, varying across channels. Engagement scores were assessed but not deeply analyzed. Location-based performance indicated Miami had the highest ROI (5.01), while New York had the lowest (4.98), though clicks and impressions remained evenly distributed. Target audience insights revealed Men (25-34) had the highest ROI (5.02), followed by Women (35-44) at 5.00. Campaigns targeting all age groups maintained balanced engagement and conversion rates.

Data Visualization

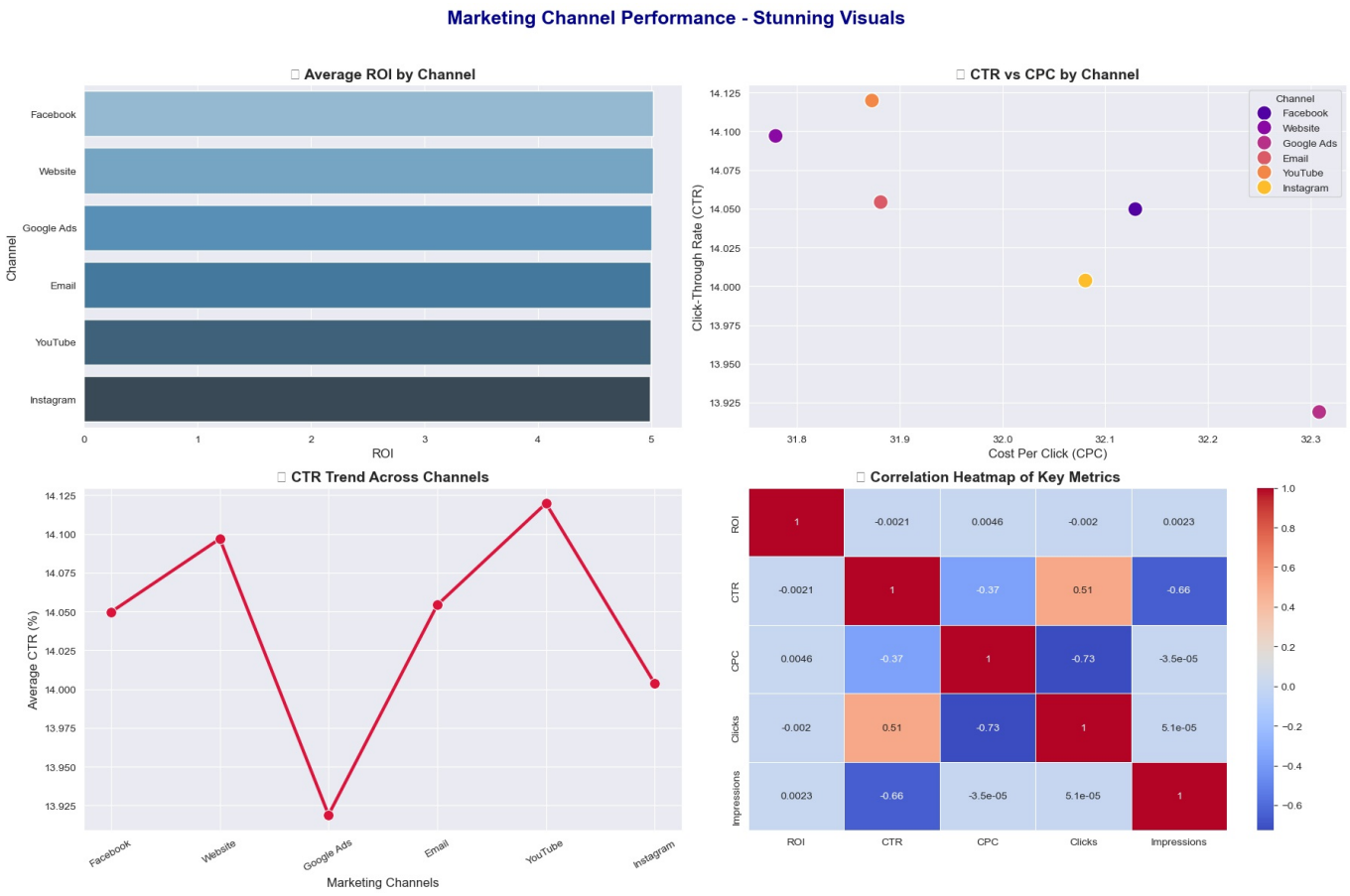
To enhance clarity, various visualizations were generated:

In [85]:

```
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Set modern, visually appealing style
sns.set_style("darkgrid")
plt.figure(figsize=(18, 10))

# Create a 2x2 subplot layout
fig, axes = plt.subplots(2, 2, figsize=(18, 12))
fig.suptitle("Marketing Channel Performance - Stunning Visuals", fontsize=18, fontweight='bold', color="darkblu

# --- 1. Bar Chart for ROI --- #
sns.barplot(x="Avg_ROI", y="Channel_Used", data=channel_performance, ax=axes[0, 0], palette="Blues_d")
axes[0, 0].set_title(" Average ROI by Channel", fontsize=14, fontweight="bold")
axes[0, 0].set_xlabel("ROI", fontsize=12)
axes[0, 0].set_ylabel("Channel", fontsize=12)

# --- 2. Scatter Plot for CTR vs CPC --- #
scatter = sns.scatterplot(
    x="Avg_CPC", y="Avg_CTR", data=channel_performance, hue="Channel_Used", palette="plasma", s=200, ax=axes[0,
)
axes[0, 1].set_title(" CTR vs CPC by Channel", fontsize=14, fontweight="bold")
axes[0, 1].set_xlabel("Cost Per Click (CPC)", fontsize=12)
axes[0, 1].set_ylabel("Click-Through Rate (CTR)", fontsize=12)
axes[0, 1].legend(title="Channel")
```

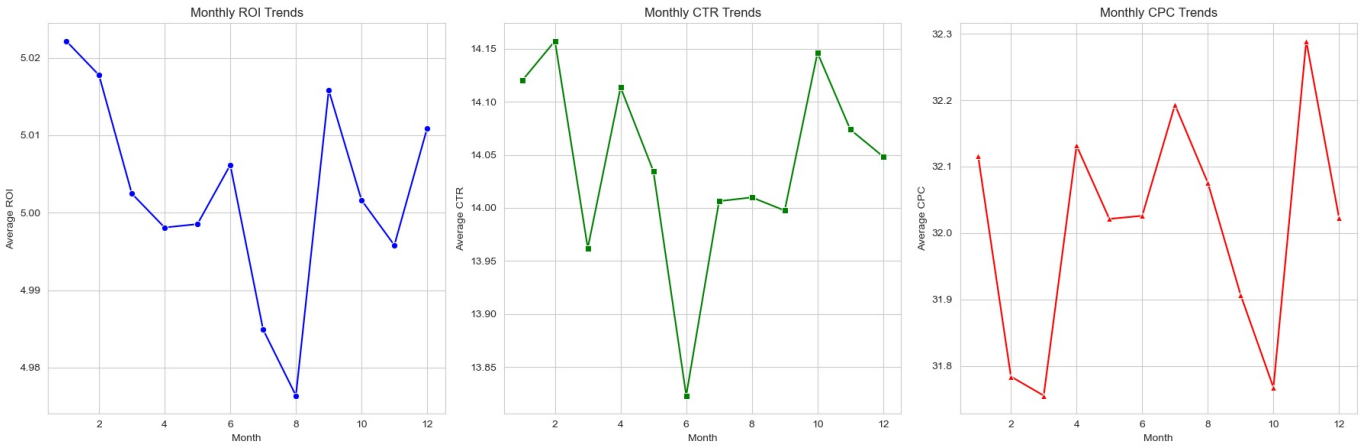
```
# --- 3. Line Chart for CTR Trend across Channels --- #
sns.lineplot(x="Channel_Used", y="Avg_CTR", data=channel_performance, marker="o", linewidth=3, markersize=10,
             ax=axes[1, 0], color="crimson")
axes[1, 0].set_title(" CTR Trend Across Channels", fontsize=14, fontweight="bold")
axes[1, 0].set_xlabel("Marketing Channels", fontsize=12)
axes[1, 0].set_ylabel("Average CTR (%)", fontsize=12)
axes[1, 0].tick_params(axis="x", rotation=30)

# --- 4. Heatmap for Key Metrics Correlation --- #
corr_matrix = df[["ROI", "CTR", "CPC", "Clicks", "Impressions"]].corr()
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5, ax=axes[1, 1])
axes[1, 1].set_title(" Correlation Heatmap of Key Metrics", fontsize=14, fontweight="bold")

# Adjust layout for better spacing
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.savefig('Marketing Channel Performance')
plt.show()
```

```
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\900478220.py:14: FutureWarning:

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

  sns.barplot(x="Avg_ROI", y="Channel_Used", data=channel_performance, ax=axes[0, 0], palette="Blues_d")
C:\Users\pc\anaconda3\Lib\site-packages\seaborn\utils.py:61: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TR
END}) missing from font(s) Arial.
  fig.canvas.draw()
C:\Users\pc\anaconda3\Lib\site-packages\seaborn\utils.py:61: UserWarning: Glyph 127919 (\N{DIRECT HIT}) missing
from font(s) Arial.
  fig.canvas.draw()
C:\Users\pc\anaconda3\Lib\site-packages\seaborn\utils.py:61: UserWarning: Glyph 128202 (\N{BAR CHART}) missing f
rom font(s) Arial.
  fig.canvas.draw()
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\900478220.py:42: UserWarning: Glyph 128293 (\N{FIRE}) missing from
font(s) Arial.
  plt.tight_layout(rect=[0, 0, 1, 0.96])
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\900478220.py:43: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS
TREND}) missing from font(s) Arial.
  plt.savefig('Marketing Channel Performance')
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\900478220.py:43: UserWarning: Glyph 127919 (\N{DIRECT HIT}) missin
g from font(s) Arial.
  plt.savefig('Marketing Channel Performance')
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\900478220.py:43: UserWarning: Glyph 128202 (\N{BAR CHART}) missing
from font(s) Arial.
  plt.savefig('Marketing Channel Performance')
C:\Users\pc\AppData\Local\Temp\ipykernel_4100\900478220.py:43: UserWarning: Glyph 128293 (\N{FIRE}) missing from
font(s) Arial.
  plt.savefig('Marketing Channel Performance')
<Figure size 1800x1000 with 0 Axes>
C:\Users\pc\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 128200 (\N{CHART WITH
UPWARDS TREND}) missing from font(s) Arial.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\pc\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 127919 (\N{DIRECT HIT
}) missing from font(s) Arial.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\pc\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 128202 (\N{BAR CHART}
) missing from font(s) Arial.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\pc\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 128293 (\N{FIRE}) mis
sing from font(s) Arial.
  fig.canvas.print_figure(bytes_io, **kw)
```



In [71]:

```
# Extract Month from Date
df['Month'] = df['Date'].dt.month

monthly_performance = df.groupby('Month')[['ROI', 'CTR', 'CPC']].mean().reset_index()

# Plot Monthly Trends
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharex=True)

sns.lineplot(data=monthly_performance, x='Month', y='ROI', marker='o', color='blue', ax=axes[0])
axes[0].set_title("Monthly ROI Trends")
axes[0].set_xlabel("Month")
axes[0].set_ylabel("Average ROI")

sns.lineplot(data=monthly_performance, x='Month', y='CTR', marker='s', color='green', ax=axes[1])
axes[1].set_title("Monthly CTR Trends")
axes[1].set_xlabel("Month")
axes[1].set_ylabel("Average CTR")

sns.lineplot(data=monthly_performance, x='Month', y='CPC', marker='^', color='red', ax=axes[2])
axes[2].set_title("Monthly CPC Trends")
axes[2].set_xlabel("Month")
axes[2].set_ylabel("Average CPC")

plt.tight_layout()
plt.savefig('jpg')
plt.show()
```



Visualizations provided key insights into marketing performance. Boxplots identified outliers in ROI and CPC, suggesting extreme variations in campaign effectiveness. Violin plots showed a right-skewed ROI distribution, highlighting a few highly profitable campaigns, while conversion rates clustered around 7-10%, indicating consistency.

Marketing channel analysis revealed that Email and Google Ads were the most cost-effective, as shown in the CTR vs. CPC scatter plot. A bar chart comparison of ROI across channels further emphasized Facebook's strong performance. A correlation heatmap indicated a positive relationship between CTR and ROI, while Acquisition Cost had a weaker impact, suggesting that budget allocation alone doesn't determine success.

Monthly trend analysis showed a gradual increase in CTR towards the year's end, reflecting improved audience engagement. ROI remained stable, demonstrating consistent marketing performance, while CPC fluctuated slightly, indicating periodic shifts in cost efficiency. These insights help refine campaign strategies for better targeting and resource allocation.

Conclusion and Recommendation

This analysis highlights key trends, top-performing marketing channels, and audience segments for optimization. Facebook and Email proved to be the most effective channels, making them ideal for increased investment. Audience segmentation should focus on Men (25-34) while refining strategies for Women (35-44) to improve engagement. Location-based optimization should prioritize Miami and Los Angeles, where ROI was highest. Cost efficiency can be improved by optimizing CPC, particularly for Google Ads and YouTube. Continuous monitoring of monthly trends will help in budget adjustments to maximize returns during high-performing periods.

In [ ]: