Understanding Customer Acquisition Cost (CAC) Analysis with Python (1)

April 12, 2024

1 Understanding Customer Acquisition Cost (CAC) Analysis with Python

2 Introduction

In today's competitive business environment, understanding the cost of acquiring new customers—known as Customer Acquisition Cost (CAC)—is crucial for any company looking to optimize its marketing strategies and enhance its profitability. This article explores what CAC is, who uses it, why it's important, and how Python can be leveraged to perform effective CAC analysis.

3 What is Customer Acquisition Cost Analysis?

Customer Acquisition Cost is a fundamental business metric that quantifies the total average cost your business incurs to acquire a new customer. This includes all marketing and advertising expenses, wages paid to sales and marketing teams, and other related overheads during a specific period.

The formula for CAC is : $\text{CAC} = \frac{\text{Total Marketing and Sales Expenses}}{\text{Number of New Customers Acquired}}$

Using Python, businesses can automate the calculation of CAC, integrate data from various sources, and generate dynamic reports that provide ongoing insights into marketing performance.

4 Who Uses Customer Acquisition Cost Analysis?

CAC analysis is not just for large corporations but is vital for businesses of all sizes. Here are a few key players who rely on this metric:

Marketers: To determine the effectiveness of different marketing strategies and to justify marketing budgets.

CFOs and Finance Teams: To ensure the company's money is wisely invested and that customer acquisition strategies are cost-effective.

Entrepreneurs and Startup Owners: To gauge the scalability of their business models and to attract investors by showcasing efficient customer acquisition strategies.

Business Analysts: To provide insights that help shape strategic decisions and to improve return on investment (ROI) across marketing channels.

5 Why is Customer Acquisition Cost Analysis Important?

Understanding CAC is crucial for several reasons:

Budgeting and Financial Planning: Knowing how much it costs to acquire customers helps businesses allocate their marketing budget more effectively.

Performance Measurement: By comparing the CAC across different channels and campaigns, companies can identify the most efficient strategies and focus their efforts accordingly.

Strategic Decision-Making: High CAC might indicate that it's time to pivot or alter marketing strategies, especially if the cost of acquiring a new customer exceeds the revenue they bring.

6 Advantages Provided by CAC Analysis

Implementing CAC analysis can bring numerous benefits to a business:

Optimization of Marketing Channels: Detailed CAC analysis helps pinpoint which channels are underperforming, allowing businesses to reallocate resources to more profitable areas.

Comparison with Customer Lifetime Value (CLV): Comparing CAC with CLV provides insight into the overall health of the business. A CLV to CAC ratio of 3:1 is typically seen as healthy and sustainable.

Early Warning System: An increasing trend in CAC can alert businesses to potential issues before they become costly, providing a chance to reassess and strategize.

Market Segmentation: CAC analysis can reveal which segments (e.g., demographic groups, regions, product lines) are more cost-effective, guiding targeted marketing efforts.

Customer Acquisition Cost (CAC) Analysis is a critical aspect of business strategy where Data Science plays a vital role. CAC refers to the cost a company incurs to acquire a new customer. Understanding and optimizing this cost is crucial for sustainable growth and profitability. If you want to learn how to analyze the customer acquisition cost of a business, this article is for you. In this article, I'll take you through the task of Customer Acquisition Cost Analysis using Python.

7 Customer Acquisition Cost Analysis: Process We Can Follow

Customer Acquisition Cost Analysis is a valuable tool for businesses to assess the efficiency and effectiveness of their customer acquisition efforts. It helps make informed decisions about resource allocation and marketing strategies, ultimately contributing to the company's growth and profitability.

Below is the process we can follow for the task of customer acquisition cost analysis as a Data Science professional:

1. Begin by collecting relevant data related to customer acquisition expenses.
2. Segment your customer acquisition costs to understand which channels or strategies are driving customer acquisition.
3. Identify key metrics that will help you calculate CAC.
4. Calculate CAC for each customer acquisition channel or strategy.
5. Analyze and find patterns to optimize your CAC.

7.1 Introduction to Customer Acquisition Cost Analysis Using Python

This article will guide you through using Python for CAC analysis, a powerful approach that leverages real-time data processing and visualization. Python's capabilities help maintain a company's competitiveness by enabling agile responses to market conditions.

Whether you are a marketer, CFO, or entrepreneur, mastering CAC analysis with Python can dramatically improve your strategic outcomes. By integrating this analysis into your operational strategies, you can optimize your customer acquisition processes and achieve superior profitability.

Let's get started with the task of Customer Acquisition Cost Analysis by importing the necessary Python libraries and the dataset:

For further exploration and analysis, you can access the dataset from here

[28]:

```
# Importing the pandas library with the alias 'pd'.
# Pandas is used for data manipulation and analysis. It provides data␣
 ↪structures like DataFrames,
# which make working with structured data easy.
import pandas as pd

# Importing the express module from the plotly library with the alias 'px'.
# Plotly Express is an interface for creating interactive plots and figures␣
 ↪quickly and easily.
import plotly.express as px

# Importing the io module from the plotly library.
# This module includes functions that support the configuration and display of␣
 ↪figures in Plotly.
import plotly.io as pio

# Importing the graph_objects module from the plotly library with the alias␣
 ↪'go'.
# This module allows for detailed and precise creation of figures, offering␣
 ↪extensive customization options.
import plotly.graph_objects as go

# Setting the default plot template to "plotly_white".
# Plotly templates control the overall look of the plot, such as the␣
 ↪background, gridlines, and color schemes.
# "plotly_white" is a simple, clean template with a white background that␣
 ↪enhances the readability of the plotted data.
pio.templates.default = "plotly_white"
```

[29]:

```
# Reading data from a CSV file named "customer_acquisition_cost_dataset.csv"␣
 ↪into a pandas DataFrame.
# The function pd.read_csv() is used to load CSV files directly into a pandas␣
 ↪DataFrame. This DataFrame is then
# stored in the variable 'data'.
data = pd.read_csv("customer_acquisition_cost_dataset.csv")

# Printing the first five rows of the DataFrame using the head() method.
# This is useful to quickly check the structure of the data, including column␣
 ↪headers and some initial data entries.
# It helps verify that the data has been loaded correctly and gives a snapshot␣
 ↪of the data you'll be working with.
print(data.head())
```

```
  Customer_ID Marketing_Channel  Marketing_Spend  New_Customers
0    CUST0001   Email Marketing      3489.027844             16
1    CUST0002        Online Ads      1107.865808             33
2    CUST0003      Social Media      2576.081025             44
3    CUST0004        Online Ads      3257.567932             32
4    CUST0005   Email Marketing      1108.408185             13
```

Let’s take a look at what each column tells us before we go any further:

[30]:

```
# Calling the info() method on the DataFrame to print its summary
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Customer_ID        500 non-null    object
 1   Marketing_Channel  500 non-null    object
 2   Marketing_Spend    500 non-null    float64
 3   New_Customers      500 non-null    int64
dtypes: float64(1), int64(1), object(2)
memory usage: 15.8+ KB
```

Now, let’s calculate the customer acquisition cost:

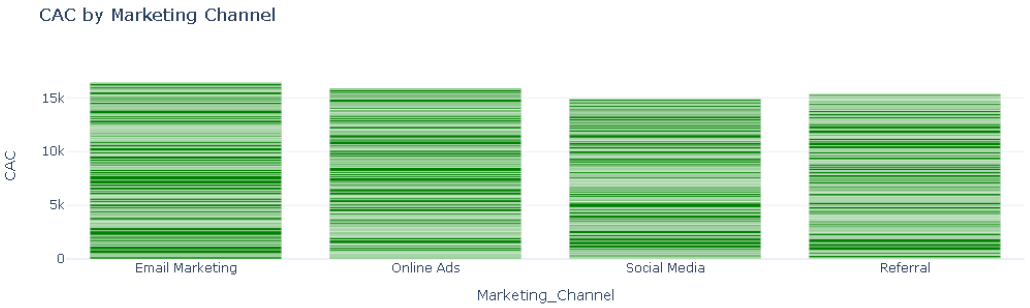
[31]:

```
# 'data' is your DataFrame that presumably has at least two columns:␣
 ↪'Marketing_Spend' and 'New_Customers'.
# This line is creating a new column in the DataFrame called 'CAC'.

data['CAC'] = data['Marketing_Spend'] / data['New_Customers']
```

Here, we added a CAC value to the dataset to show the company how well its marketing efforts are working and to identify which marketing channels are the most cost-effective for bringing in new customers Now, let’s check out the customer acquisition costs for each marketing channel: fig1 = px.bar(data, x=‘Marketing_Channel’, y=‘CAC’, title=‘CAC by Marketing Channel’) fig1.show()

[32]:

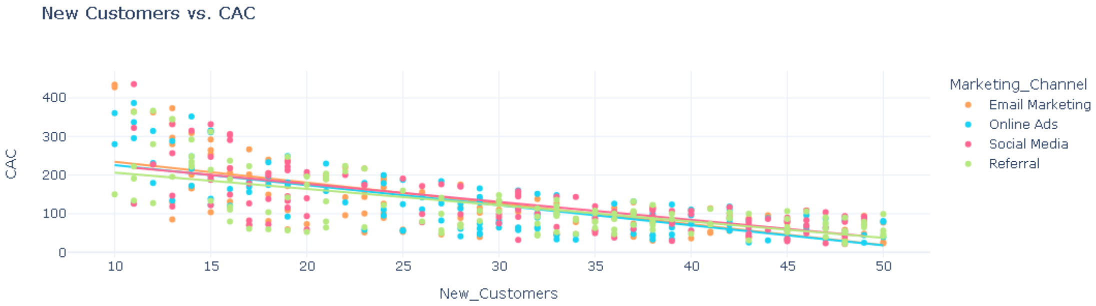
```
# Creating a bar chart with the px.bar function.
# The data argument specifies the DataFrame to use for plotting.
# The x argument sets 'Marketing_Channel' as the category axis (horizontal).
# The y argument sets 'CAC' as the numeric axis (vertical).
# The title argument provides a title for the chart.
# The color_discrete_sequence parameter allows specifying a list of colors to␣
 ↪use for discrete (categorical) mappings.
# Here, ['green'] sets all bars to be green.
fig1 = px.bar(data, x='Marketing_Channel', y='CAC', title='CAC by Marketing␣
 ↪Channel',
              color_discrete_sequence=['green'])
# Displaying the figure using the show method.
# This method renders the plot in your Jupyter notebook, a web browser, or any␣
 ↪other compatible GUI environment.
fig1.show()
```



Email marketing turns out to be the most expensive way to get customers, while social media is the cheapest. Next, let's explore how the number of new customers relates to the cost of acquiring them:

[34]:

```
# Creating a scatter plot with specific colors for each category in␣
 ↪'Marketing_Channel'.
# First, we need to know what the unique categories in 'Marketing_Channel' are.␣
 ↪You might need to adjust this line:
# categories = data['Marketing_Channel'].unique().tolist()
# color_map = {categories[i]: color for i, color in enumerate(['#a6cee3',␣
 ↪'#1f78b4', '#b2df8a', '#33a02c'])}
# Alternatively, directly map your categories to colors if you know them:
color_map = {
    'Channel_1': '#a6cee3',
    'Channel_2': '#1f78b4',
    'Channel_3': '#b2df8a',
    'Channel_4': '#33a02c'
}

fig2 = px.scatter(data, x='New_Customers', y='CAC', color='Marketing_Channel',
                  color_discrete_map=color_map,
                  title='New Customers vs. CAC', trendline='ols')

# Displaying the figure using the show method.
fig2.show()
```



So, the negative slope of the trendline in the above graph suggests that there is a tendency for channels with a higher number of new customers to have a lower CAC. In other words, as marketing efforts become more effective in acquiring customers, the cost per customer tends to decrease.

The downward slope of the trendline in the graph indicates that channels attracting more new customers generally spend less per customer. This means that as marketing becomes more successful at gaining customers, it also becomes more cost-effective.

```
[35]: # The groupby function is used here to create a grouped object where subsequent operations can be applied to each group separately.
summary_stats = data.groupby('Marketing_Channel')['CAC'].describe()

# The describe() method is then applied to the 'CAC' column of each group.
# This method generates descriptive statistics that summarize the central tendency, dispersion, and shape of the dataset's distribution.
# The output includes count, mean, std (standard deviation), min, 25th percentile (Q1), median (50th percentile), 75th percentile (Q3), and max.
print(summary_stats)
```

```
                   count        mean        std        min        25%  \
Marketing_Channel
Email Marketing    124.0  132.913758  89.597107  23.491784  68.226195
Online Ads         130.0  122.135938  79.543793  24.784414  62.207753
Referral           128.0  119.892174  74.101916  22.012364  71.347939
Social Media       118.0  126.181913  77.498788  21.616453  75.633389

                          50%         75%         max
Marketing_Channel
Email Marketing    106.940622  177.441898  434.383446
Online Ads          97.736027  163.469540  386.751285
Referral            99.835688  137.577935  366.525209
Social Media       102.620356  167.354709  435.487346
```

7.2 From the summary statistics, you can gain several insights:

Compare Mean CAC Values: Look at the average customer acquisition costs to determine which marketing channels are generally less expensive. If reducing costs is key, prioritize channels with the lowest average CAC.

Evaluate Consistency with Standard Deviation: The standard deviation shows how much CAC values vary within each channel. A higher standard deviation indicates more inconsistency, signaling a need for further analysis to pinpoint the causes of these cost fluctuations.

Assess Distribution Using Quartiles: Quartiles can help you understand how CAC values are spread out. For instance, targeting channels where the first quartile (25%) is low might be more cost-effective.

Consider Minimum and Maximum Values: Knowing the range of CAC values—from the minimum to the maximum—helps you understand the potential highs and lows of costs in each channel.

Now, let's calculate the conversion rate of this marketing campaign:

[36]:
```
# This line is creating a new column in the DataFrame called 'Conversion_Rate'.

data['Conversion_Rate'] = data['New_Customers'] / data['Marketing_Spend'] * 100
```

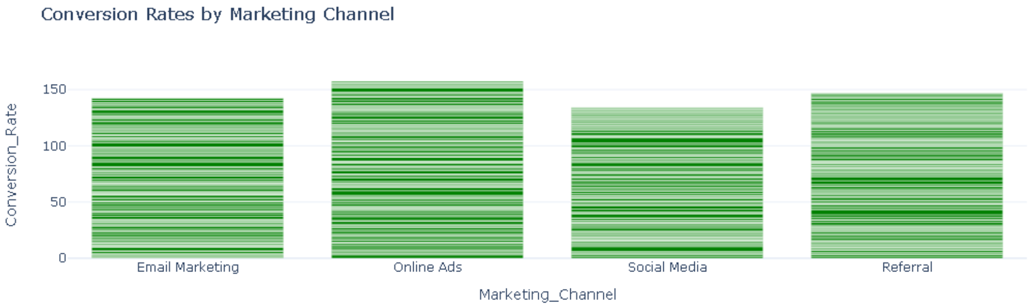
[37]:
```
data.head()
```

[37]:
```
  Customer_ID Marketing_Channel  Marketing_Spend  New_Customers         CAC  \
0    CUST0001   Email Marketing      3489.027844             16  218.064240
1    CUST0002        Online Ads      1107.865808             33   33.571691
2    CUST0003      Social Media      2576.081025             44   58.547296
3    CUST0004        Online Ads      3257.567932             32  101.798998
4    CUST0005   Email Marketing      1108.408185             13   85.262168

   Conversion_Rate
0         0.458580
1         2.978700
2         1.708021
3         0.982328
4         1.172853
```

Now, let's figure out the conversion rate for this marketing campaign:

[38]:
```
# Creating a bar chart with the px.bar function, setting all bars to a green␣
 ↪color.
# The color_discrete_sequence parameter allows specifying a list of colors to␣
 ↪use for discrete (categorical) mappings.
# Here, ['green'] sets all bars to be green.
fig = px.bar(data, x='Marketing_Channel', y='Conversion_Rate',
             title='Conversion Rates by Marketing Channel',
             color_discrete_sequence=['green'])
```

```
# Displaying the figure using the show method.
fig.show()
```

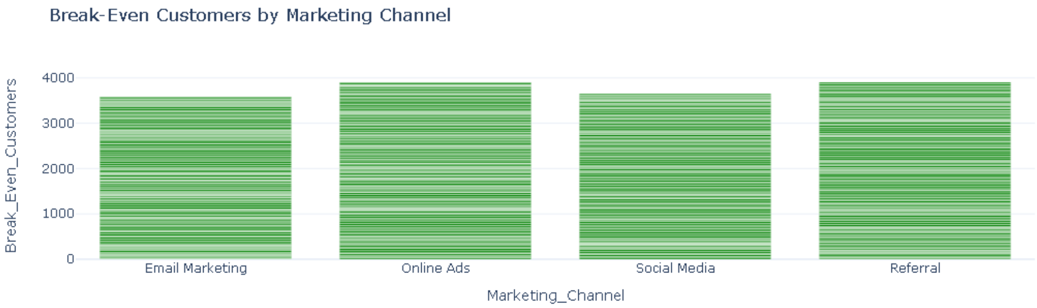


So, we can see that the conversion rates of online ads are better than all other channels.

Now, let's determine the break-even customers for each marketing channel. Break-even customers represent the number of new customers required to cover the costs associated with a specific marketing channel. When the actual number of new customers acquired through the channel surpasses the break-even number, it indicates that the marketing efforts are yielding profits. Here's how to calculate break-even customers for each marketing channel:

[39]:
```
# Calculating the break-even number of customers for each marketing channel.
# This is done by dividing the marketing spend by the Customer Acquisition Cost␣
↪(CAC).
# The result tells us how many customers each channel needs to acquire to␣
↪justify the marketing expenses.
data['Break_Even_Customers'] = data['Marketing_Spend'] / data['CAC']
```

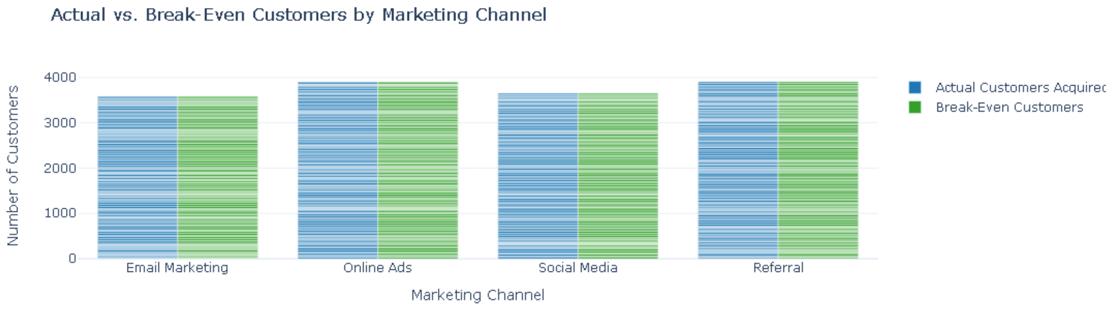
[40]:
```
fig = px.bar(data, x='Marketing_Channel', y='Break_Even_Customers',
             title='Break-Even Customers by Marketing Channel',
             color_discrete_sequence=['green'])

# Displaying the figure using the show method.
# This method renders the plot in your Jupyter notebook, a web browser, or any␣
↪other compatible GUI environment.
fig.show()
```



Now, let's examine how the actual number of customers acquired compares to the break-even customers for each marketing channel:

[41]:
```
# Initializing a new figure object for plotting
fig = go.Figure()

# Adding a bar trace for actual customers acquired. This uses␣
 ↪'Marketing_Channel' as the x-axis,
# 'New_Customers' as the y-axis, and sets the bar color to #1f78b4, a shade of␣
 ↪blue.
fig.add_trace(go.Bar(x=data['Marketing_Channel'], y=data['New_Customers'],
                     name='Actual Customers Acquired', marker_color='#1f78b4'))

# Adding a second bar trace for break-even customers. Similar to the first, but␣
 ↪this shows
# the number of customers needed to break even on marketing costs, colored␣
 ↪#33a02c, a shade of green.
fig.add_trace(go.Bar(x=data['Marketing_Channel'],␣
 ↪y=data['Break_Even_Customers'],
                     name='Break-Even Customers', marker_color='#33a02c'))

# Updating the layout of the figure to group bars by marketing channel, making␣
 ↪it easier to compare them.
# Also, setting the chart title and labels for the x and y axes for better␣
 ↪clarity and presentation.
fig.update_layout(barmode='group', title='Actual vs. Break-Even Customers by␣
 ↪Marketing Channel',
                  xaxis_title='Marketing Channel', yaxis_title='Number of␣
 ↪Customers')

# Rendering the figure. This command will display the interactive Plotly chart␣
 ↪in the output,
# which can be interacted with in a Jupyter notebook or a web browser.
```

```
fig.show()
```



This demonstrates a successful outcome for the marketing campaign, as the actual number of customers acquired from all marketing channels aligns precisely with the break-even customers. Had the actual customers fallen short of the break-even point, it would have signaled a need to reevaluate marketing strategies or allocate additional resources to those channels.

8 Conclusion

In conclusion, this is how you can conduct **Customer Acquisition Cost (CAC) Analysis using Python**.

CAC Analysis is a valuable tool for businesses to evaluate the efficiency and effectiveness of their customer acquisition efforts. By making informed decisions about resource allocation and marketing strategies, businesses can drive growth and profitability. I hope this guide has equipped you with the knowledge to confidently perform CAC Analysis using Python.

If you have any questions, please do not hesitate to ask in the comments section below

You can access the notebook via this GitHub Repository

My name is **David Akanji** and you can follow me on linkedin via or direct your enquiry to `akanjiolubukoladavid@gmail.com`