Omni Channel Marketing

By Prof.Vishal Chugh

What is Omni Channel ?

It helps businesses understand which marketing channels influenced the customer journey and how much each contributed

Process

Assign Credit

A method to assign Credit for conversions across multiple marketing touchpoints

Handling Budgets

Enables smarter budget allocation and growth decisions

Channel Performance

It becomes easy to understand which channel influence conversion and revenue

Proper Attribution Models

First Touch

All credit goes to the first interaction

Last Touch

All credit goes to the final interaction before conversion

Popular Attribution Models

Linear

It gives equal credit to all touchpoints

Time Decay

More credit to recent interactions

When to recommend first touch attribution ?

It is ideal for campaigns focused on brand awareness and initial customer engagement. It helps understand which marketing efforts are most effective at attracting new customers and getting them into the sales funnel.

When to recommend last touch attribution ?

Suitable when the final interaction before a conversion is crucial, and the sales cycle is relatively short. It helps pinpoint the marketing efforts that directly lead to sales or conversions.

When to recommend Linear attribution ?

Appropriate when all touchpoints in the customer journey are considered equally important. It provides a balanced view of marketing performance across all interactions.

When to recommend Time Decay attribution ?

Best for longer sales cycles where recent interactions are likely to have a greater impact on the final decision. It emphasizes the importance of recent touchpoints in the conversion process.

Complete Process to Tap Customers Touchpoints

Common Parameters

1. Email ID
2. Phone Number
3. Loyalty Card
4. Discounted Coupons
5. Premium Memberships

Concepts of Lists, Tuples and Dictionary

By Prof.Vishal Chugh

What is List ?

In Python, a list is an collection of items (elements) that can store multiple values in a single variable.

Lists are mutable (you can change, add, or remove elements).

Lists can contain different data types (numbers, strings, floats, even other lists).

They are written with square brackets [ ].

Marketing Campaigns List

```
# A list of Google marketing campaigns
google_campaigns = ["Search Ads", "Display Ads", "YouTube Ads", "Shopping Ads", "App Campaigns"]

print(google_campaigns)
```

```
['Search Ads', 'Display Ads', 'YouTube Ads', 'Shopping Ads', 'App Campaigns']
```

Why are lists useful in Marketing ?

```
# Adding the data in the list
google_campaigns.append("Performance Max")
```

```
print(google_campaigns)
```

```
['Search Ads', 'Display Ads', 'YouTube Ads', 'Shopping Ads', 'App Campaigns', 'Performance Max']
```

```
# remove shopping Ads
google_campaigns.remove("Shopping Ads")
print(google_campaigns)
```

```
['Search Ads', 'Display Ads', 'YouTube Ads', 'App Campaigns', 'Performance Max']
```

Why are lists useful in Marketing ?

```
# Analyze Marketing Campaigns
print("First Campaign:", google_campaigns[0])  # Search Ads
print("Last Campaign:", google_campaigns[-1])  # Performance Max
```

```
First Campaign: Search Ads
Last Campaign: Performance Max
```

What is Tuple?

A tuple in Python is an collection of items like a list, but it is immutable – meaning once you create it, you cannot change, add, or remove elements.

Written with parentheses ().

Good for data that should not be modified.

Marketing Campaigns Tuple

```
# Campaign name, Clicks, Impressions, Conversions
campaign_performance = ("Search Ads", 5000, 200000, 450)

print(campaign_performance)
```

```
('Search Ads', 5000, 200000, 450)
```

Marketing Campaigns Tuple

```
# Analyse campaign performance
print("Campaign:", campaign_performance[0])       # Search Ads
print("Total Clicks:", campaign_performance[1])   # 5000
```

```
Campaign: Search Ads
Total Clicks: 5000
```

What is Dictionary ?

A dictionary in Python is a Semi - Structured data stored in key–value pairs.

Keys act like labels (unique identifiers).

Values are the data associated with those keys.

Written with curly brackets {key1 : value1 , key2: value2 }

Marketing Campaigns Dictionary

```
# Campaign spend dictionary
campaign_spend = {
    "Search Ads": 12000,
    "Display Ads": 8000,
    "YouTube Ads": 15000,
    "Shopping Ads": 6000}

print(campaign_spend)
```

```
{'Search Ads': 12000, 'Display Ads': 8000, 'YouTube Ads': 15000, 'Shopping Ads': 6000}
```

Why Dictionary is useful in Marketing ?

```
# check the youtube spends
print("YouTube Ads Spend:", campaign_spend["YouTube Ads"])
```

```
YouTube Ads Spend: 15000
```

```
# Add a campaign
campaign_spend["Performance Max"] = 10000
```

```
print(campaign_spend)
```

```
{'Search Ads': 12000, 'Display Ads': 8000, 'YouTube Ads': 15000, 'Shopping Ads': 6000, 'Performance Max': 10000}
```

Why Dictionary is useful in Marketing ?

```
# Update the spend
campaign_spend["Display Ads"] = 9000
print(campaign_spend)
```

```
{'Search Ads': 12000, 'Display Ads': 9000, 'YouTube Ads': 15000, 'Shopping Ads': 6000, 'Performance Max': 10000}
```

```
# Calculate total spend
total_spend = sum(campaign_spend.values())
print("Total Spend:", total_spend)
```

```
Total Spend: 52000
```

Explore the Data in Python

By Prof.Vishal Chugh

The Omni Channel Data

	journey_id	customer_id	path	touch_dates	touch_count	converted	revenue
0	1	C1001	Social Media > Physical Store > Social Media Ads	2023-04-23 | 2023-04-24 | 2023-04-29	3	0	0
1	2	C1002	Physical Store > Social Media > Social Media >...	2024-02-13 | 2024-02-16 | 2024-02-19 | 2024-02...	5	1	9398
2	3	C1003	Website > Social Media Ads > Website > Affilia...	2024-07-07 | 2024-07-09 | 2024-07-13 | 2024-07...	5	0	0
3	4	C1004	Social Media > Social Media > Website > Email ...	2023-03-08 | 2023-03-10 | 2023-03-12 | 2023-03...	5	1	7096
4	5	C1005	Website > Physical Store > Affiliates > Email	2024-01-19 | 2024-01-20 | 2024-01-21 | 2024-01-22	4	1	1838

First Touch Attribution

```
# First Touch = First Channel in the path
data['first_touch'] = data['path'].str.split('>').str[0]
```

```
# First Touch
data['first_touch'].head()
```

	first_touch
0	Social Media
1	Physical Store
2	Website
3	Social Media
4	Website

Group the first touch attribution with revenue

```
# group the column by first_touch and see the revenue
first_touch_group = data.groupby('first_touch')['revenue'].sum()
```

```
# first touch sort in descending order
first_touch_group.sort_values(ascending= False)
```

first_touch	revenue
Social Media Ads	342613
Social Media	304188
Physical Store	285730
Affiliates	275749
Website	208410
Email	191722

Last Touch Attribution

```
# Last touch Attribution
data['last_touch'] = data['path'].str.split('>').str[-1]
```

```
# Last Touch
data['last_touch'].head()
```

	last_touch
0	Social Media Ads
1	Email
2	Affiliates
3	Affiliates
4	Email

Group the last touch attribution with revenue

```
# group the column by last_touch and see the revenue
last_touch_group = data.groupby('last_touch')['revenue'].sum()
```

```
# Sort in descending order
last_touch_group.sort_values(ascending= False)
```

last_touch	revenue
Affiliates	325231
Physical Store	281943
Email	280547
Website	266746
Social Media	236963
Social Media Ads	216982

For Linear Attribution We Split the Data

```
# Step 1: Split the customer journey into steps
touch_points_wide = data["path"].str.split(" > ", expand = True)
```

```
touch_points_wide.head()
```

	0	1	2	3	4
0	Social Media	Physical Store	Social Media Ads	None	None
1	Physical Store	Social Media	Social Media	Physical Store	Email
2	Website	Social Media Ads	Website	Affiliates	Affiliates
3	Social Media	Social Media	Website	Email	Affiliates
4	Website	Physical Store	Affiliates	Email	None

Count touchpoints

```
# Count how many touches each customer had
touch_count_calc = touch_points_wide.notna().sum(axis =1).rename('touch_count_calc')
```

```
touch_count_calc.head()
```

	touch_count_calc
0	3
1	5
2	5
3	5
4	4

Convert Wide to long format

```
# Convert from "wide" to "long" format
long = touch_points_wide.reset_index().melt(id_vars = 'index' , var_name = 'position', value_name = 'channel' ).dropna(subset = ['channel'])
```

```
long
```

	index	position	channel
0	0	0	Social Media
1	1	0	Physical Store
2	2	0	Website
3	3	0	Social Media
4	4	0	Website

Add Revenue and Touch Point Info

```
# Add revenue & touch count info
concatenation = pd.concat([data['revenue'], touch_count_calc], axis =1)
```

```
long = long.merge(concatenation, left_on = 'index', right_index = True)
```

```
long.head()
```

	index	position	channel	revenue	touch_count_calc
0	0	0	Social Media	0	3
1	1	0	Physical Store	9398	5
2	2	0	Website	0	5
3	3	0	Social Media	7096	5
4	4	0	Website	1838	4

Adjust Position Numbers instead of 0 to 1

```
# Adjust position numbers
long['positions'] = long['position'].astype(int) + 1
```

```
long.head()
```

	index	position	channel	revenue	touch_count_calc	positions
0	0	0	Social Media	0	3	1
1	1	0	Physical Store	9398	5	1
2	2	0	Website	0	5	1
3	3	0	Social Media	7096	5	1
4	4	0	Website	1838	4	1

Linear Attribution

```
# Linear Attribution (equal split across touches)
long["linear_share"] = long["revenue"] / long["touch_count_calc"]
```

```
# linear attribution
linear_att = long.groupby('channel')['linear_share'].sum()
```

```
# sort in descending order
linear_attribution = linear_att.sort_values(ascending= False)
round(linear_attribution,2)
```

channel	linear_share
Social Media	293721.35
Physical Store	291351.37
Affiliates	289056.35
Email	248163.15
Social Media Ads	243335.27
Website	242784.52

Time Decay Attribution -> n(n+1)/2

```
# Time-Decay Attribution (position-based, later touches get more weight)
denom = (touch_count_calc * (touch_count_calc +1) / 2). rename('denominator') # n(n+1)/2
```

```
# merge the data with the long variable
long = long.merge(denom, left_on = 'index', right_index = True)
```

```
long.head()
```

	index	position	channel	revenue	touch_count_calc	positions	linear_share	denominator
0	0	0	Social Media	0	3	1	0.0	6.0
1	1	0	Physical Store	9398	5	1	1879.6	15.0
2	2	0	Website	0	5	1	0.0	15.0
3	3	0	Social Media	7096	5	1	1419.2	15.0
4	4	0	Website	1838	4	1	459.5	10.0

Calculate Weights and Shares

```
long['weights'] = long['positions'] / long['denominator']
long['shares'] = long['revenue'] * long['weights']
```

```
long.head()
```

	index	position	channel	revenue	touch_count_calc	positions	linear_share	denominator	weights	shares
0	0	0	Social Media	0	3	1	0.0	6.0	0.166667	0.000000
1	1	0	Physical Store	9398	5	1	1879.6	15.0	0.066667	626.533333
2	2	0	Website	0	5	1	0.0	15.0	0.066667	0.000000
3	3	0	Social Media	7096	5	1	1419.2	15.0	0.066667	473.066667
4	4	0	Website	1838	4	1	459.5	10.0	0.100000	183.800000

Calculate Weights and Shares

```
long['weights'] = long['positions'] / long['denominator']
long['shares'] = long['revenue'] * long['weights']
```

```
long.head()
```

	index	position	channel	revenue	touch_count_calc	positions	linear_share	denominator	weights	shares
0	0	0	Social Media	0	3	1	0.0	6.0	0.166667	0.000000
1	1	0	Physical Store	9398	5	1	1879.6	15.0	0.066667	626.533333
2	2	0	Website	0	5	1	0.0	15.0	0.066667	0.000000
3	3	0	Social Media	7096	5	1	1419.2	15.0	0.066667	473.066667
4	4	0	Website	1838	4	1	459.5	10.0	0.100000	183.800000

Time Decay Attribution

```
# time decay attribution
time_att = long.groupby('channel')['shares'].sum()
```

```
time_attribution = time_att.sort_values(ascending= False)
round(time_attribution,2)
```

channel	shares
Affiliates	297361.40
Physical Store	290921.07
Social Media	280769.43
Email	262814.57
Website	252472.33
Social Media Ads	224073.20