

```

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

import gc
from tqdm import tqdm
from sklearn.utils import resample
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
import keras
from google.colab import files
import io
import os
import urllib.request
import zipfile
from functools import lru_cache

```

### Downloading the dataset from the link

```

def download_dataset():
    urllib.request.urlretrieve(
        "https://go.criteois.com/criteo-research-attribution-dataset.zip",
        "criteo_attribution_dataset.zip"
    )

    with zipfile.ZipFile("criteo_attribution_dataset.zip", "r") as zip_ref:
        zip_ref.extractall("criteo_attribution_dataset")

dataset_path = 'criteo_attribution_dataset/criteo_attribution_dataset.tsv.gz'

if not os.path.exists(dataset_path):
    download_dataset()

```

### ✎ Using a subset sample of the dataset.

Reason: The dataset size exceeds available memory, causing RAM overflow and application crashes

```
DEBUG_SAMPLE = 0.7
```


### We are selecting a subset sample, comprising 70% of the total dataset

```

def get_sampled_dataset(debug_sample=0.7):
    """Get a dataset in which we sample on user ids to ensure we keep the entire histories
    """
    df = pd.read_csv(dataset_path, sep='\t', compression="gzip")
    uid_and_salt = df['uid'].astype(str) + 'hash_salt_for_sampling'
    hashed_uid_and_salt = pd.util.hash_pandas_object(uid_and_salt, index=False)
    random_column_based_on_uid = hashed_uid_and_salt / np.iinfo(np.uint64).max
    return df[random_column_based_on_uid < debug_sample]

df_criteo = get_sampled_dataset(DEBUG_SAMPLE)
df_criteo

```




	timestamp	uid	campaign	conversion	conversion_timestamp	conversion_id	attribution	click	click_pos	click_r
0	0	20073966	22589171	0	-1	-1	0	0	-1	...
2	2	28474333	18975823	0	-1	-1	0	0	-1	...
3	3	7306395	29427842	1	1449193	3063962	0	1	0	...
4	3	25357769	13365547	0	-1	-1	0	0	-1	...
5	4	93907	17686799	0	-1	-1	0	1	-1	...
...	...	...	...	...	...	...	...	...	...	...
16468022	2671199	5767906	4869923	0	-1	-1	0	1	-1	...
16468023	2671199	6852682	10002432	0	-1	-1	0	0	-1	...
16468024	2671199	16638720	7061828	0	-1	-1	0	0	-1	...
16468025	2671199	3032300	5061834	0	-1	-1	0	1	-1	...
16468026	2671199	23160808	15398570	0	-1	-1	0	0	-1	...

11516130 rows x 22 columns

```
df_criteo = pd.read_csv(dataset_path, sep='\\t', compression="gzip")
```


```
df_criteo.head()
```




	timestamp	uid	campaign	conversion	conversion_timestamp	conversion_id	attribution	click	click_pos	click_nb	...
0	0	20073966	22589171	0	-1	-1	0	0	-1	-1	...
2	2	28474333	18975823	0	-1	-1	0	0	-1	-1	...
3	3	7306395	29427842	1	1449193	3063962	0	1	0	7	...
4	3	25357769	13365547	0	-1	-1	0	0	-1	-1	...
5	4	93907	17686799	0	-1	-1	0	1	-1	-1	...

5 rows x 22 columns

```
df_criteo.shape
```

 (11516130, 22)

```
num_distinct_campaigns = df_criteo['campaign'].nunique()
print("Number of distinct campaigns:", num_distinct_campaigns)
```

 Number of distinct campaigns: 675

```
n_campaigns = 500
```

```
df_criteo['day'] = np.floor (df_criteo['timestamp']/86400).astype(int)
```

```
#Creating a JourneyID column from UserID and ConversionID
```

```
def add_derived_columns(df):
    df_ext = df.copy()
    df_ext ['jid'] = df_ext['uid'].map(str) + '_' + df_ext['conversion_id'].map(str)

    min_max_scaler = MinMaxScaler()

    for cname in ('timestamp','time_since_last_click'):
        x = df_ext[cname].values.reshape(-1,1)
        df_ext[cname + ' _norm'] = min_max_scaler.fit_transform(x)

    return df_ext
```

```
df_criteo= add_derived_columns(df_criteo)
```

```
df_criteo.head()
```

```
↗
```

	timestamp	uid	campaign	conversion	conversion_timestamp	conversion_id	attribution	click	click_pos	click_nb	...
0	0	20073966	22589171	0	-1	-1	0	0	-1	-1	...
2	2	28474333	18975823	0	-1	-1	0	0	-1	-1	...
3	3	7306395	29427842	1	1449193	3063962	0	1	0	7	...
4	3	25357769	13365547	0	-1	-1	0	0	-1	-1	...
5	4	93907	17686799	0	-1	-1	0	1	-1	-1	...

5 rows × 26 columns

```
np.random.seed(42)
def sample_campaigns(df,n_campaigns):
    campaigns = np.random.choice(df_criteo['campaign'].unique(),n_campaigns,replace=False)
    return df[df['campaign'].isin(campaigns)]
```

```
#Calling the sample_campaigns function with n_campaigns=400
df_criteo = sample_campaigns(df_criteo,n_campaigns)
```

```
df_criteo.shape
```

```
↗ (8761116, 26)
```

```
def filter_journeys_by_length(df,min_touchpoints) :
    if min_touchpoints <= 1:
        return df
    else :
        grouped = df.groupby(['jid'])['uid'].count().reset_index(name="count")
        return df[df['jid'].isin(grouped[grouped['count'] >= min_touchpoints] ['jid'].values)]
```

```
#Calling the filter_journeys_by_length function
df_criteo = filter_journeys_by_length(df_criteo, 2)
#Displaying the structure of the datasetd
```

```
def balance_conversions(df):
    df_minority = df [df.conversion == 1]
    df_majority = df [df.conversion == 0]

    df_majority_jids = np.array_split(df_majority['jid'].unique(), 100 * df_majority.shape[0]/df_minority.shape[0])

    df_majority_sampled = pd.DataFrame(data=None, columns=df. columns)

    for jid_chunk in df_majority_jids:
        df_majority_sampled = pd. concat([df_majority_sampled, df_majority[df_majority.jid.isin(jid_chunk)]])
        if df_majority_sampled.shape[0] > df_minority.shape [0]:
            break
    return pd.concat([df_majority_sampled,df_minority]).sample(frac=1).reset_index(drop=True)
```

```
df_criteo = balance_conversions(df_criteo)
```

```
↗ <ipython-input-20-e4b9469fd8d2>:10: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is d
df_majority_sampled = pd. concat([df_majority_sampled, df_majority[df_majority.jid.isin(jid_chunk)]])
```

```
df_criteo.shape
```

```
↗ (303671, 26)
```

```
def map_one_hot(df, column_names, result_column_name):
    mapper = {}
    for i, col_name in enumerate(column_names):
        for val in df[col_name].unique():
            mapper[str(val) + str(i)] = len(mapper)

    df_ext = df.copy()
```

```
def one_hot (values):
    v = np.zeros(len(mapper))
    for i, val in enumerate(values):
        v[mapper[str(val) + str(i)]] = 1
    return v

df_ext[result_column_name] = df_ext[column_names].values.tolist()
df_ext[result_column_name] = df_ext[result_column_name].map(one_hot)

return df_ext
```

```
df_criteo = map_one_hot(df_criteo,['cat1','cat2','cat3','cat4','cat5','cat6','cat8'], 'cats')
```

```
df_criteo.cats[0]
```

```
→ array([1., 0., 0., ..., 0., 0., 0.])
```

```
df_criteo = map_one_hot(df_criteo,['campaign'],'campaigns').sort_values(by=['timestamp_norm'])
```

```
df_criteo.shape
```

```
→ (303671, 28)
```

```
[df_criteo[df_criteo.conversion == 0].shape[0], df_criteo[df_criteo.conversion == 1].shape[0]]
```

```
→ [5027, 298644]
```

```
df_criteo.head()
```



	timestamp	uid	campaign	conversion	conversion_timestamp	conversion_id	attribution	click	click_pos	click_nb
13804	3	25357769	13365547	0	-1	-1	0	0	-1	-1
12591	3	7306395	29427842	1	1449193	3063962	0	1	0	7
137673	4	19923387	31772643	0	-1	-1	0	0	-1	-1
191520	7	23074162	16184517	0	-1	-1	0	0	-1	-1
258702	7	5588915	27491436	0	-1	-1	0	0	-1	-1

5 rows x 28 columns

df\_criteo.campaign.unique()

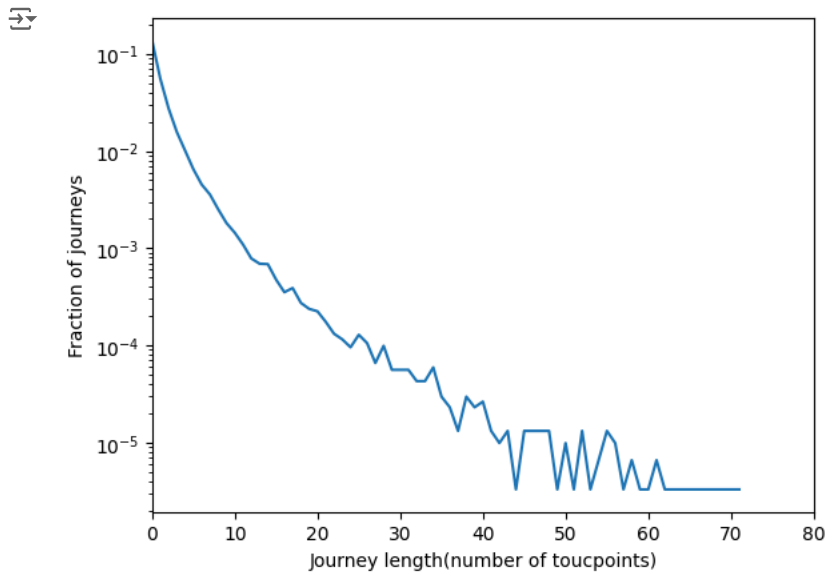


```
array([13365547, 29427842, 31772643, 16184517, 27491436, 21640043,
       24267774, 18443077, 13462155, 1415896, 10341182, 27684173, 9100689,
       32135670, 13422843, 2946551, 4869922, 25064375, 15184511, 13289562,
       2144729, 24045148, 5061834, 32452111, 30491418, 16922675, 25419531,
       9106407, 13661605, 2953716, 497590, 3291800, 497593, 5544859,
       27888153, 17661711, 14661605, 24378636, 13661597, 1586481,
       22589171, 31736975, 17288254, 12947794, 1341195, 24126625, 8877963,
       30427825, 32452108, 8892339, 32368244, 9100693, 15398570, 28739284,
       20309767, 3884873, 12351511, 30801593, 30405203, 12700454,
       27031690, 22431924, 21830168, 22891668, 29614012, 28729624,
       26852339, 11803618, 14289571, 2077112, 13947793, 14121532,
       11500306, 7877971, 13947795, 9100690, 30881002, 2694554, 31330666,
       25259856, 7289590, 28328731, 24389111, 14102485, 289466, 27891651,
       29534051, 7686710, 27118781, 15568932, 23817046, 19129839,
       23852330, 32385772, 11843320, 13462158, 8710016, 15574227,
       12289574, 31129938, 4814007, 20938751, 8407229, 26389107, 8980571,
       17661709, 6686704, 1871873, 3788051, 26321366, 828346, 9100692,
       16321090, 28071513, 23330020, 5044698, 11289564, 5281156, 14445192,
       26998428, 7061828, 24491616, 10013589, 10491643, 9097339, 28351001,
       17710664, 15661602, 25172774, 8980585, 14006998, 10474106,
       29513914, 3422224, 15959625, 13661608, 21016759, 2869134, 10462151,
```

```
df_criteo.campaigns[0]
```

```
plt.plot(range(len(hist_x)),hist_y,label = 'all journeys')
plt.yscale('log')
plt.xlim(0, 80)
plt.xlabel('Journey length(number of toucpoints)')
```

```
plt.ylabel('Fraction of journeys')
plt.show()
```



## ✓ **Linear Attribution Model**

```
# Defining the linear attribution function
def linear_attribution(df):
    # Counting Impressions by Campaign
    def count_by_campaign(df):
        counters = np.zeros(n_campaigns)

        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot) # Find the index of the campaign
            counters[campaign_id] += 1
        return counters

    # Calculate Campaign Impressions
    campaign_impressions = count_by_campaign(df)

    # Filter for Conversions and Calculate Conversion Rate per Click
    df_converted = df[df['conversion'] == 1].copy() # Filter rows where conversion == 1
    df_converted['linear'] = df_converted['conversion'] / df_converted['click_nb'].astype(float)

    # Counting Conversions by Campaign
    def linear_attribution_by_campaign(df_conv):
        counters = np.zeros(n_campaigns)
        for idx in range(len(df_conv)):
            campaign_id = np.argmax(df_conv.iloc[idx]['campaigns'])
            counters[campaign_id] += df_conv.iloc[idx]['linear']
        return counters

    # Calculate Conversion Attribution
    campaign_conversions = linear_attribution_by_campaign(df_converted)

    return campaign_conversions / campaign_impressions # Conversion rate per impression

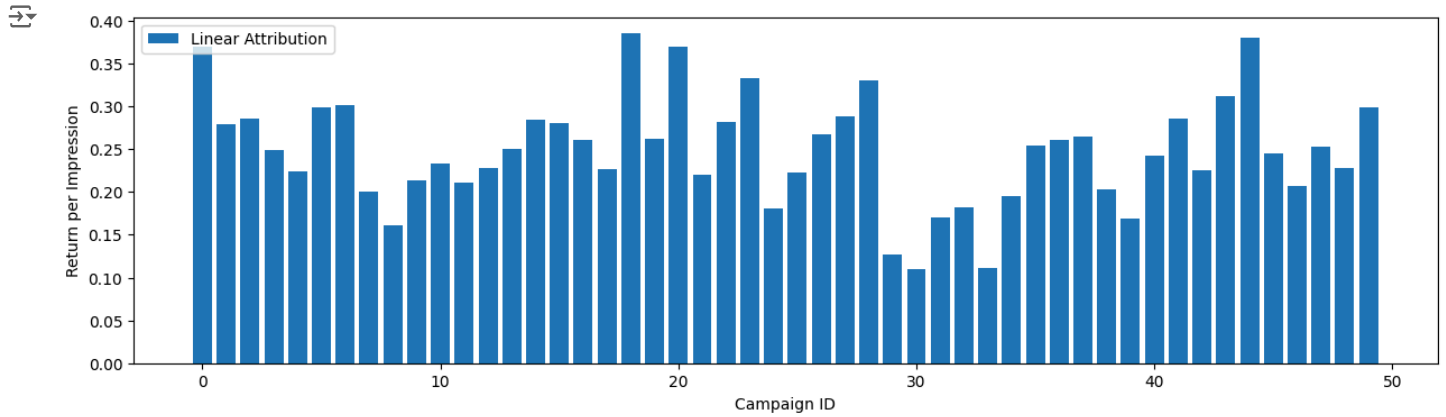
# Example Usage
# Ensure df_criteo is defined and that n_campaigns = 400 is defined outside
linear_attr = linear_attribution(df_criteo)
```

```
<ipython-input-38-58692e882100>:30: RuntimeWarning: invalid value encountered in divide
    return campaign_conversions / campaign_impressions # Conversion rate per impression
```

```
campaign_idx = range(150, 200)
```

```
fig = plt.figure(figsize=(15, 4))
plt.bar(range(len(linear_attr[campaign_idx])), linear_attr[campaign_idx], label='Linear Attribution')
plt.ylabel('Return per Impression')
plt.xlabel('Campaign ID')
```

```
plt.legend(loc='upper left')
plt.show()
```



linear\_attr

```
array([0.23340841, 0.22983208, 0.29486157, 0.3075, 0.2178867,
0.20951803, 0.16876965, 0.27150698, 0.2192458, 0.22622844,
0.21686362, 0.13581444, 0.26589145, 0.18161339, 0.25130459,
0.24649799, 0.21021983, 0.1948788, 0.26680418, 0.27242651,
0.24122533, 0.20369879, 0.31638142, 0.19890074, 0.2378169,
0.16195288, 0.18256641, 0.33827331, 0.25353594, 0.19843847,
0.19398011, 0.18983075, 0.32618243, 0.27650473, 0.24832442,
0.2819578, 0.25761515, 0.25887586, 0.21530661, 0.12106814,
0.26975309, 0.29639724, 0.25988846, 0.22815515, 0.1898937,
0.18886827, 0.28745179, 0.25792877, 0.18893514, 0.23624225,
0.21927821, 0.22246052, 0.16182725, 0.22879608, 0.19810268,
0.37167987, 0.31578406, 0.2989202, 0.36508196, 0.23681439,
0.2932392, 0.37137214, 0.20541, 0.17752564, 0.2351666,
0.25257775, 0.36474608, 0.37795326, 0.14026171, 0.18106128,
0.13572236, 0.35388331, 0.19038236, 0.17467978, 0.26875,
0.25193622, 0.21539282, 0.12777249, 0.25353836, 0.4020256,
0.36146768, 0.39256308, 0.21052718, 0.2453147, 0.15393663,
0.22954935, 0.16040494, 0.27969179, 0.37531202, 0.2,
0.30821078, 0.29179799, 0.23057047, 0.22915088, 0.25391487,
0.27408009, 0.24542249, 0.26284612, 0.12135439, 0.22984683,
0.25994508, 0.19395845, 0.19143033, 0.38936252, 0.2814987,
0.27745864, 0.13562425, 0.37089691, 0.27035876, 0.26269863,
0.27172464, 0.21224436, 0.23863636, 0.2413084, 0.26985899,
0.33104606, 0.18785866, 0.16334728, 0.26600369, 0.23458707,
0.23679919, 0.26960329, 0.2716178, 0.12869801, 0.30971597,
0.21249875, 0.14470284, 0.2708618, 0.44444444, 0.15273575,
0.08810885, 0.17739709, 0.2516941, 0.18454906, 0.33680063,
0.24940215, 0.31746032, 0.33862434, 0.17557764, 0.25831227,
0.3266595, 0.30783856, 0.3002008, 0.32824229, 0.16705815,
0.11292878, 0.22359182, 0.25626413, 0.28532204, 0.21608661,
0.36998144, 0.27968045, 0.28530021, 0.24890553, 0.2236791,
0.29892628, 0.30175439, 0.20071516, 0.16057658, 0.21394665,
0.23326779, 0.21106061, 0.22805115, 0.25011171, 0.28411823,
0.28044396, 0.26117338, 0.22629717, 0.38502604, 0.26197732,
0.36923077, 0.21991126, 0.2814256, 0.33318828, 0.18129579,
0.22283523, 0.26757491, 0.28858484, 0.33023679, 0.12682289,
0.10990142, 0.17011183, 0.18237613, 0.11057711, 0.19582861,
0.25438546, 0.26086957, 0.2649469, 0.20333555, 0.16869994,
0.24266278, 0.28551913, 0.22530021, 0.31234973, 0.38063358,
0.24467262, 0.20641349, 0.25305861, 0.22864364, 0.29905437,
0.35326087, 0.3, 0.39118774, 0.27313321, 0.27747836,
0.27584193, 0.37543426, 0.30361946, 0.40887097, 0.32836879,
0.29255319, 0.32061275, 0.31057569, 0.16032222, 0.21083639,
0.15588488, 0.19537839, 0.25774673, 0.26677728, 0.2178623,
0.19572298, 0.33703961, 0.22094904, 0.21976008, 0.21842169,
0.20156099, 0.24383596, 0.17215103, 0.37447257, 0.4,
0.22726107, 0.22458177, 0.27452621, 0.27142142, 0.38107154,
0.33670548, 0.4047619, 0.21678137, 0.40909091, 0.31212121,
0.17223804, 0.2373997, 0.33987426, 0.25393239, 0.22546136,
0.25409836, 0.31237545, 0.19164337, 0.20078125, 0.15135408,
0.27496357, 0.23922718, 0.18143075, 0.26710501, 0.17360928,
0.298099, 0.22803605, 0.1748365, 0.18954248, 0.32130952,
0.29377637, 0.03876249, 0.37748335, 0.26571429, 0.38530316,
0.26708333, 0.14535441, 0.25949047, 0.30806983, 0.26731179,
```



```
0.14575617, 0.15264858, 0.34984985, 0.23519159, 0.29858575,
0.23648057, 0.19472789, 0.4133587 , 0.13240577, 0.21794872,
0.34196913, 0.31606061, 0.4 , 0.39172297, 0.31960784,
0 74846825 0 75040035 0 78874006 0 10876311 0 70346003
```

## ✓ Time Decay Attribution Model

Time-decay attribution gives more credit to the touchpoints that are closer in time to the conversion.

The time decay formula used here is  $2^{-(X/\text{half life})}$

where X = Conversion day - Click day and half life = 7

### Creating the conversion day column

```
df_criteo['conversion_day'] = np.floor(df_criteo.conversion_timestamp/86400).astype(int)
```

```
df_criteo.columns
```

```
Index(['timestamp', 'uid', 'campaign', 'conversion', 'conversion_timestamp',
      'conversion_id', 'attribution', 'click', 'click_pos', 'click_nb',
      'cost', 'cpo', 'time_since_last_click', 'cat1', 'cat2', 'cat3', 'cat4',
      'cat5', 'cat6', 'cat7', 'cat8', 'cat9', 'day', 'jid', 'timestamp_norm',
      'time_since_last_click_norm', 'cats', 'campaigns', 'conversion_day'],
      dtype='object')
```

```
df_criteo.shape
```

```
(303671, 29)
```

### Creating a function for time decay attribution model

```
def time_decay_attribution(df):
    def count_by_campaign(df):
        counters = np.zeros(n_campaigns) # Ensure n_campaigns is defined
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    campaign_impressions = count_by_campaign(df)

    df_converted = df[df['conversion'] == 1]

    def calculate_attribution(conversion_day, click_day):
        rel_pos = conversion_day - click_day
        attribution = pow(2, -(rel_pos / 7))
        return attribution

    def time_decay_attribution_value(df_converted):
        df_converted['time_decay'] = df_converted.apply(
            lambda val: calculate_attribution(val.conversion_day, val.day),
            axis=1 # Specify axis for apply
        )
        return df_converted

    def time_decay_by_campaign(df_converted):
        counters = np.zeros(n_campaigns)
        for idx in range(len(df_converted)):
            campaign_id = np.argmax(df_converted.iloc[idx, 27])
            counters[campaign_id] += df_converted.iloc[idx, 29]
        return counters

    df_converted = time_decay_attribution_value(df_converted)

    campaign_conversions = time_decay_by_campaign(df_converted)
    return campaign_conversions / campaign_impressions # Return the final result

time_decay_attr = time_decay_attribution(df_criteo)
```

```
<ipython-input-46-a0f33238b760>:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view)

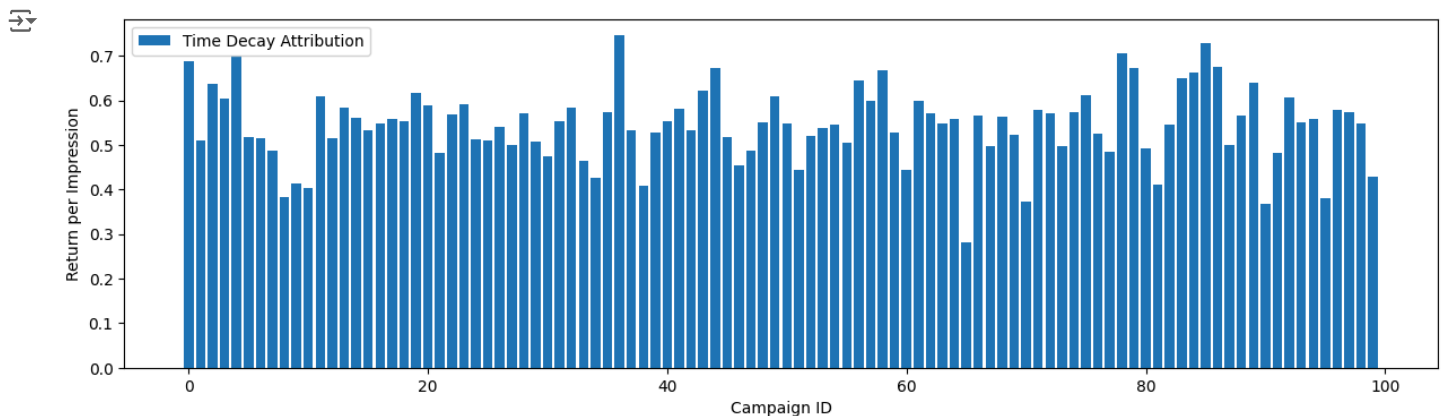
```
df_converted['time_decay'] = df_converted.apply(
<ipython-input-46-a0f33238b760>:35: RuntimeWarning: invalid value encountered in divide
return campaign_conversions / campaign_impressions # Return the final result
```

### Plotting a graph to visualize the attribution scores

```
campaign_idx = range(150, 250)

fig = plt.figure(figsize=(15, 4))
plt.bar(range(len(time_decay_attr[campaign_idx])), time_decay_attr[campaign_idx], label='Time Decay Attribution')
plt.ylabel('Return per Impression') # Fixed quotation mark
plt.xlabel('Campaign ID') # Fixed quotation mark
plt.legend(loc='upper left')

# Display the plot
plt.show()
```



```
time_decay_attr
```

```
array([0.56875054, 0.49205425, 0.54794929, 0.51388499, 0.53952492,
0.47340834, 0.60794914, 0.57067825, 0.46586916, 0.53138059,
0.51211189, 0.5751001 , 0.60018226, 0.50685773, 0.41985446,
0.51725949, 0.50835472, 0.51930287, 0.69201532, 0.52505834,
0.56402795, 0.5226361 , 0.55606818, 0.50748454, 0.5109376 ,
0.4796756 , 0.65403502, 0.6449022 , 0.50211706, 0.49294094,
0.50581242, 0.60084091, 0.58893691, 0.57939261, 0.48308042,
0.50858761, 0.57843195, 0.49652716, 0.51679704, 0.47838975,
0.5251264 , 0.4935041 , 0.47336475, 0.48641537, 0.49301184,
0.50729907, 0.51273811, 0.62699696, 0.53421598, 0.49335436,
0.51128105, 0.47589634, 0.58531341, 0.6145256 , 0.48406957,
0.7742282 , 0.53094698, 0.54058818, 0.65353496, 0.524965 ,
0.4740638 , 0.79139847, 0.48352138, 0.54059723, 0.50671778,
0.50436241, 0.63872278, 0.84459437, 0.51198009, 0.52051868,
0.48656952, 0.56495883, 0.45560871, 0.50054729, 0.68317821,
0.57503806, 0.45602049, 0.54380496, 0.53297911, 0.70841513,
0.66587456, 0.69580629, 0.49258618, 0.46419763, 0.57422002,
0.51088028, 0.58333728, 0.64434798, 0.85604098, 0.37059256,
0.64912515, 0.52433068, 0.53306435, 0.52046177, 0.48552226,
0.51482284, 0.52564633, 0.50258845, 0.5286698 , 0.52244421,
0.59033899, 0.52213288, 0.52884978, 0.70569819, 0.60665827,
0.55384977, 0.40875058, 0.69281429, 0.46273488, 0.54900528,
0.52353752, 0.39831519, 0.49626855, 0.4979543 , 0.5330837 ,
0.55195635, 0.55575148, 0.62551654, 0.49859239, 0.53684991,
0.5165015 , 0.66063885, 0.63284332, 0.44393373, 0.53912973,
```