Assignment 3 - Part 2
Attribution Modelling

Attribution Modelling

Multi-touch attribution is a method of marketing measurement that evaluates the impact of each touchpoint in driving a conversion, thereby determining the value of that specific touchpoint.

The major goal is to understand where to devote spend, allocating budget to campaigns that are bringing in more revenue and minimizing funding on those that were ineffective using the user level data or user interactions such as clicks, number of clicks, click positions, etc. It helps marketers to achieve higher ROI for their marketing investments, illuminating where spend is most and least effective and also to identify audiences across channels and determine those users' specific marketing desires.

First Touch Attribution

This model gives full sales credit to the first marketing touchpoint interacted or first impression before conversion.

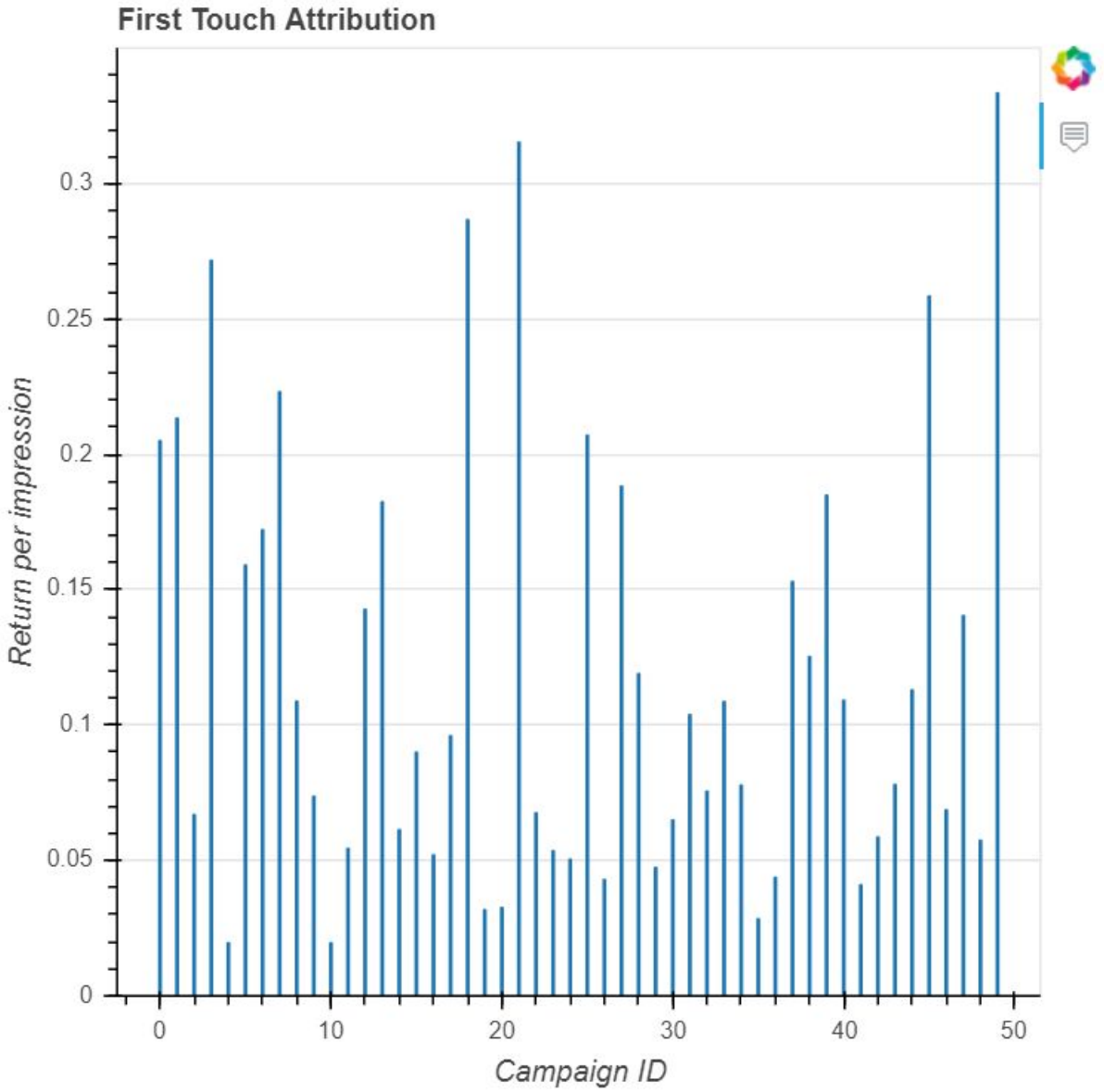
Considering the first touch point impression for 50 campaigns, hence minimum timestamp or first impression for every journey id is given full credits.

```
def first_touch_attribution(df):

    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)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    campaign_impressions = count_by_campaign(df)

    df_converted = df[df['conversion'] == 1]
    idx = df_converted.groupby(['jid'])['timestamp_norm'].transform(min) == df_converted['timestamp_norm']
    campaign_conversions = count_by_campaign(df_converted[idx])

    return campaign_conversions / campaign_impressions

fta = first_touch_attribution(df6)
```

The ROI is as follows -

```
fta
```

```
array([0.22807018, 0.11648848, 0.27781479, 0.15649867, 0.16569767,
       0.20869565, 0.16684842, 0.18136439, 0.34825871, 0.14583333,
       0.29846154, 0.06163708, 0.14974182, 0.12059369, 0.19382022,
       0.2       , 0.08158508, 0.03426124, 0.2406639 , 0.12899263,
       0.19029496, 0.08333333, 0.06904955, 0.11856823, 0.09561753,
       0.0475382 , 0.04424779, 0.23003195, 0.12093023, 0.12878788,
       0.12333333, 0.25793651, 0.07720588, 0.09773124, 0.13274336,
       0.36363636, 0.18536585, 0.05555556, 0.16587678, 0.18009479,
       0.10869565, 0.29824561, 0.21366025, 0.13592233, 0.06774194,
       0.10144928, 0.17610063, 0.17021277, 0.19230769, 0.33333333])
```



Linear Attribution

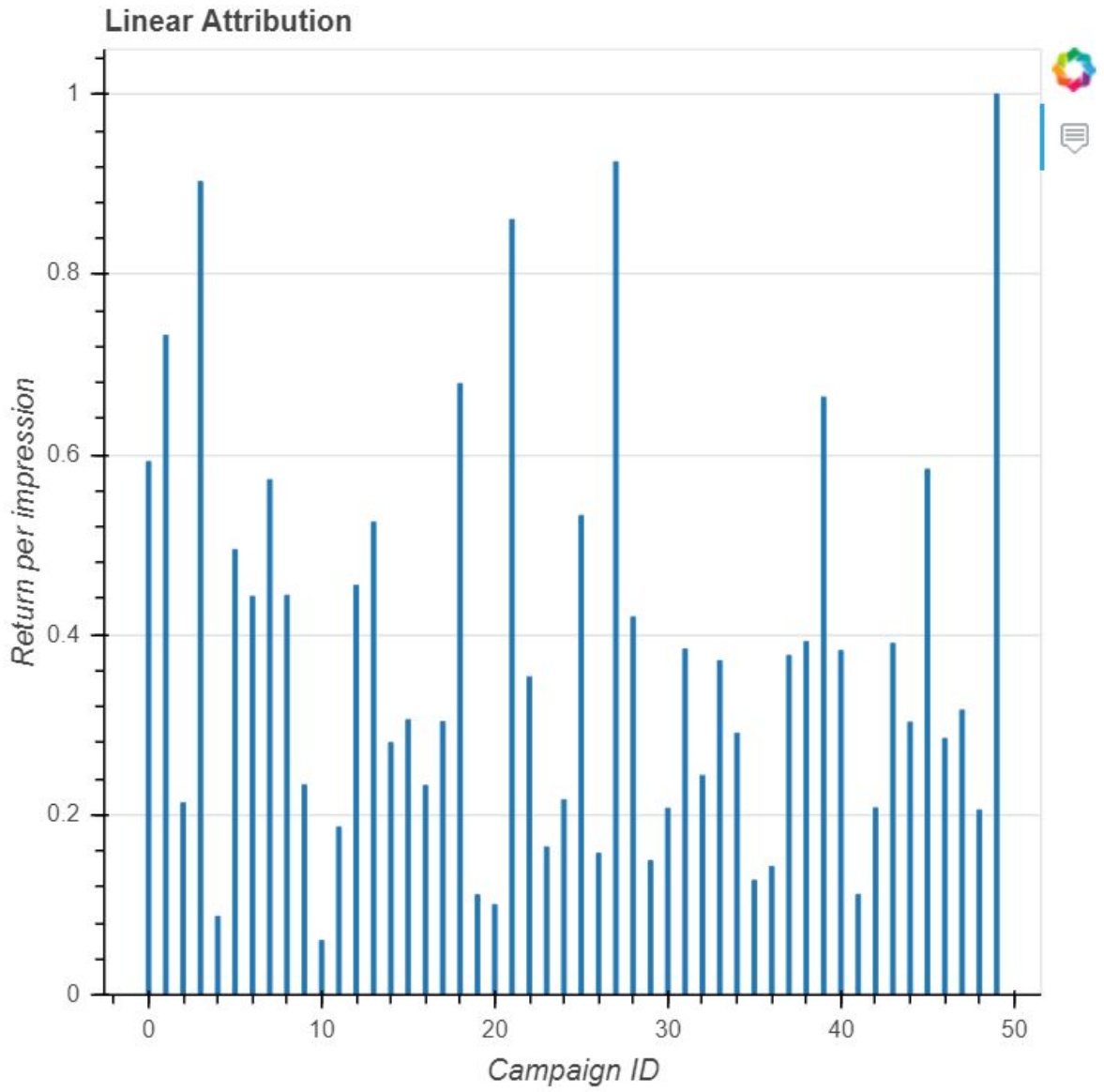
This model gives equal or same credit to all the touchpoints that led to conversion.

```
def linear_touch_attribution(df):

    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)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    campaign_impressions = count_by_campaign(df)

    df_converted = df[df['conversion'] == 1]
#    idx = df_converted.groupby(['jid'])['timestamp_norm'].transform(min) == df_converted['timestamp_norm']
    campaign_conversions = count_by_campaign(df_converted)

    return campaign_conversions / campaign_impressions

linear = linear_touch_attribution(df6)
```

All the touchpoints or impressions are given equal or same credits in the journey till it gets converted hence, the ROI is as follows:

```
linear
```

```
array([0.61403509, 0.39813281, 0.80229847, 0.58650162, 0.52325581,
       0.57391304, 0.70992366, 0.60981697, 0.82089552, 0.55456349,
       0.86461538, 0.53451677, 0.68330465, 0.52226345, 0.59831461,
       0.635     , 0.26806527, 0.0856531 , 0.96348548, 0.58476658,
       0.61370124, 0.25      , 0.26888708, 0.46308725, 0.41035857,
       0.21561969, 0.10324484, 0.79233227, 0.44186047, 0.64393939,
       0.4       , 0.75396825, 0.57352941, 0.39267016, 0.33628319,
       0.99300699, 0.6097561 , 0.16666667, 0.81516588, 0.54976303,
       0.35869565, 0.85964912, 0.99299475, 0.44660194, 0.15806452,
       0.27536232, 0.59748428, 0.5106383 , 0.46153846, 1.        ])
```



U-Shaped Attribution

U-shaped attribution gives maximum credits say (40% each) to the first and last touch points and remaining credits (say 20%) are equally assigned to all the intermediate touch points that led to conversion.

```
In [424]:
def U_touch_attribution(df):

    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)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    campaign_impressions = count_by_campaign(df)

    df_converted = df[df['conversion'] == 1]
    temp_jid=0
    campaign_conversions_nrml=0
    campaign_conversions_first=0
    campaign_conversions_last=0
    campaign_conversions_intermediate=0
    for jid in df_converted.jid.unique():
        if jid != temp_jid:
            temp_jid = jid
            if df_converted[df_converted['jid'] == temp_jid]['timestamp_norm'].max() == df_converted[df_converted['jid'] == t
                idx_nrml = df_converted.groupby(['jid'])['timestamp_norm'].transform(min) == df_converted['timestamp_norm']

                campaign_conversions_nrml = count_by_campaign(df_converted[idx_nrml])
            if df_converted[df_converted['jid'] == temp_jid]['timestamp_norm'].max() != df_converted[df_converted['jid'] == t
                idx_min = df_converted.groupby(['jid'])['timestamp_norm'].transform(min) == df_converted['timestamp_norm']
                idx_max = df_converted.groupby(['jid'])['timestamp_norm'].transform(max) == df_converted['timestamp_norm']

                campaign_conversions_first = count_by_campaign(df_converted[idx_min])
                campaign_conversions_last = count_by_campaign(df_converted[idx_max])
                campaign_conversions_intermediate = (campaign_impressions - (campaign_conversions_first + campaign_conversior
    return (((campaign_conversions_nrml + campaign_conversions_first + campaign_conversions_last ) * .7) + (campaign_conversi

uta = U_touch_attribution(df6)
```

Assigning 70% weights or credits to the first and last touch points and 30% credits to the intermediate touch points -

return (((campaign_conversions_nrml + campaign_conversions_first + campaign_conversions_last) * .7) + (campaign_conversions_intermediate * .3)) / campaign_impressions

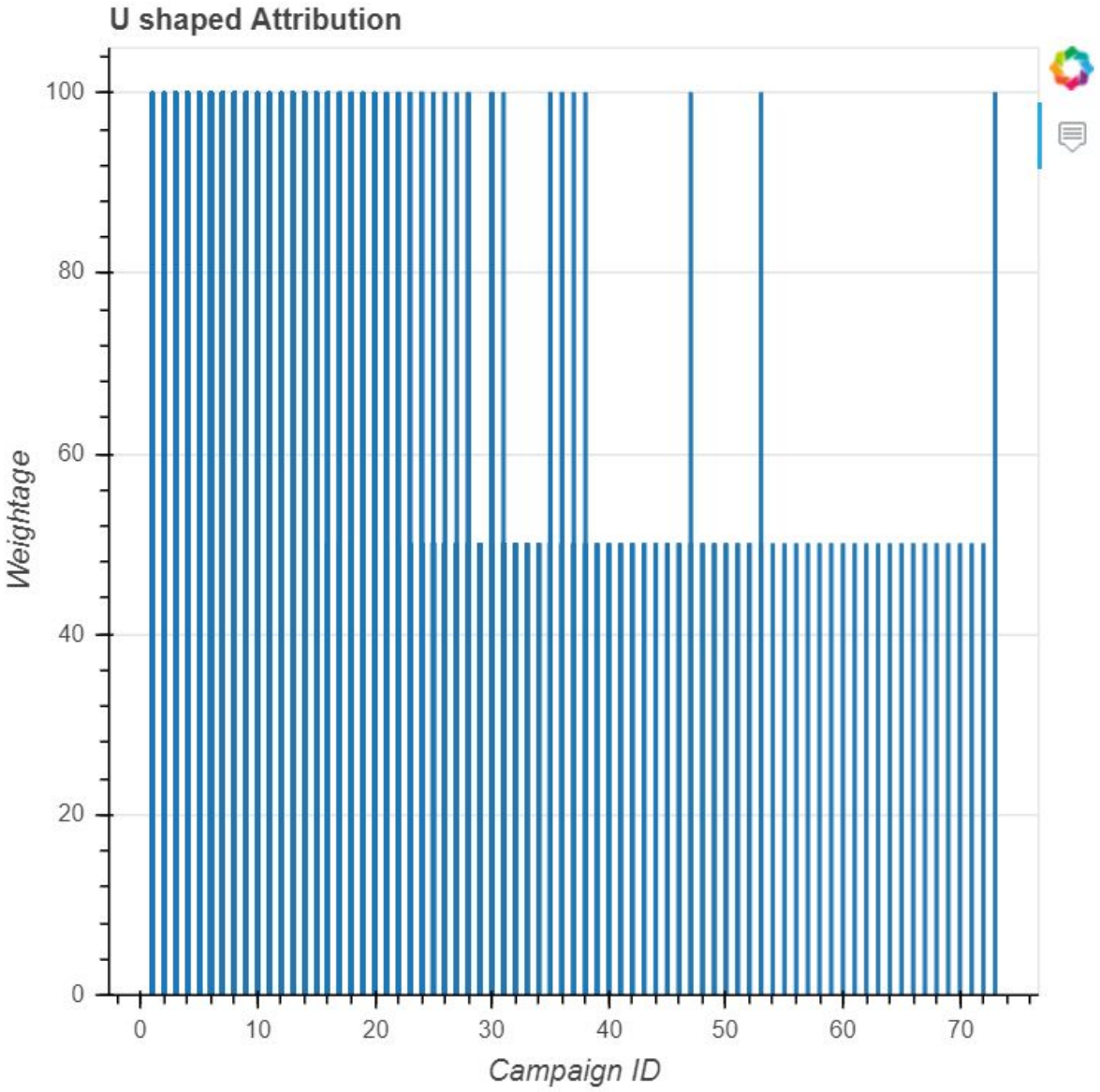
```
dummy = df6
dummy_mod=dummy[dummy['conversion']==1]
dummy_mod.sort_values(by=['campaign','timestamp'], inplace=True)
temp_jid = 0
dummy_array=[]
campaign_array=[]
timestamp_array=[]
ctr=0
for campaign in dummy_mod.campaign.unique():
             ctr = ctr + 1
             print("Campaign No:", ctr,campaign )
             for jid in  dummy_mod[dummy_mod['campaign']==campaign].jid.unique():
                if jid != temp_jid:
                    count = 1
                    temp_jid = jid
                    for idx,timestamp in enumerate (dummy_mod[dummy_mod['jid']==jid]['timestamp']):
                        timestamp_array.append(idx+1)
                        campaign_array.append(campaign)
                        if timestamp == dummy_mod[dummy_mod['jid']==jid]['timestamp'].max():
                            dummy_array.append(100)
                        elif timestamp == dummy_mod[dummy_mod['jid']==jid]['timestamp'].min():
                            dummy_array.append(100)
                        elif timestamp < dummy_mod[dummy_mod['jid']==jid]['timestamp'].max():
                            dummy_array.append(50)
                else:
                     count = count + 1
```

Another approach wherein, 100% credits is given to the first and last touch point for all the conversions in each and every journeys in 50 campaigns & 50% credits is being assigned to the intermediate touchpoints. Building the leaderboard for the campaigns and its weightage as given below,

```
leaderboard = pd.DataFrame({'Timestamp':timestamp_array,'Campaign':campaign_array,'Wightage':dummy_array})
leaderboard
```

	Timestamp	Campaign	Wightage
0	1	787943	100
1	2	787943	50
2	3	787943	100
3	1	787943	100
4	2	787943	100
...	...	...	...
34242	3	32321347	100
34243	1	32321347	100
34244	2	32321347	100
34245	1	32321347	100
34246	2	32321347	100

34247 rows × 3 columns


U shaped Attribution
Weightage
100
80
60
40
20
0
0
10
20
30
40
50
60
70
Campaign ID

Time-Decay Attribution

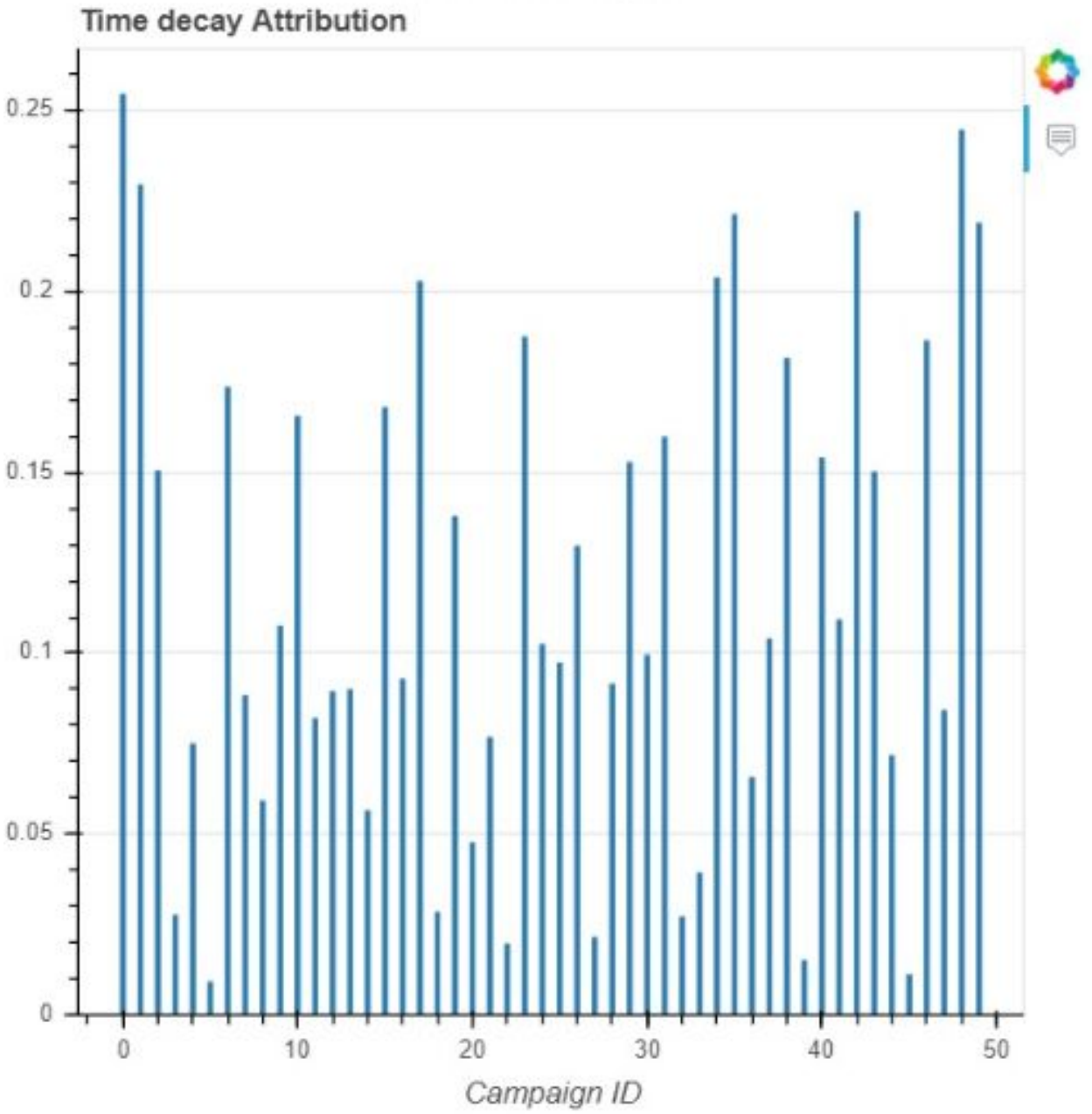
Time Decay attribution model assigns more credits to the touch points close to the conversion that the customer would have interacted with meaning the last touchpoint is given maximum credits and the one before that with a little lesser weightage compared to the last touchpoint and so on in the decreasing order of weightage assigned till the first touch point. Closer the touchpoint is to the date of conversion, higher the weightage is given to that touchpoint

```
from itertools import cycle
n_campaigns = 50
def time_decay_attribution(df):
    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)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters
    campaign_impressions = count_by_campaign(df)
    df_converted = df[df['conversion'] == 1]
    temp_jid=0
    campaign_conversions_nrml=0
    campaign_conversions_first=0
    campaign_conversions_last=0
    campaign_conversions_intermediate=0
    arr = [.75,.7,.6,.5,.4,.3,.25]
    for jid in df_converted.jid.unique():
        if jid != temp_jid:
            temp_jid = jid
            print(temp_jid)
            if df_converted[df_converted['jid'] == temp_jid]['timestamp_norm'].max() == df_converted[df_converted['jid'] == t
                idx_nrml = df_converted.groupby(['jid'])['timestamp_norm'].transform(min) == df_converted['timestamp_norm']
                print("Only 1 Record ")
                campaign_conversions_nrml = count_by_campaign(df_converted[idx_nrml])
            if df_converted[df_converted['jid'] == temp_jid]['timestamp_norm'].max() != df_converted[df_converted['jid'] == t
                idx_min = df_converted.groupby(['jid'])['timestamp_norm'].transform(min) == df_converted['timestamp_norm']
                idx_max = df_converted.groupby(['jid'])['timestamp_norm'].transform(max) == df_converted['timestamp_norm']
                print("Multiple Records ")
            campaign_conversions_first = count_by_campaign(df_converted[idx_min])
            campaign_conversions_last = count_by_campaign(df_converted[idx_max])
            campaign_conversions_btwn=count_by_campaign(df_converted)
            campaign_conversions_intermediate = (campaign_conversions_btwn - (campaign_conversions_first + campaign_conversic

    if len(campaign_conversions_intermediate)>len(arr):
        for i, (value1,value2) in enumerate(zip(campaign_conversions_intermediate, cycle(arr))):
            print('Inside zip')
            campaign_conversions_intermediate_v1 = value1*value2
        print(campaign_conversions_intermediate_v1)

    elif len(campaign_conversions_intermediate)<len(arr):
        for i, (value1,value2) in enumerate(zip(cycle(campaign_conversions_intermediate), arr)):
            print('Inside zip')
            campaign_conversions_intermediate_v1 = value1*value2
        print(campaign_conversions_intermediate_v1)

    return (((campaign_conversions_last) * .8) + (campaign_conversions_first * .1) + (campaign_conversions_intermediate_v1))
tda = time_decay_attribution(df6)
```

All touchpoints are given credit based on their difference from the date of conversion, hence ROI for each campaign after time decay att

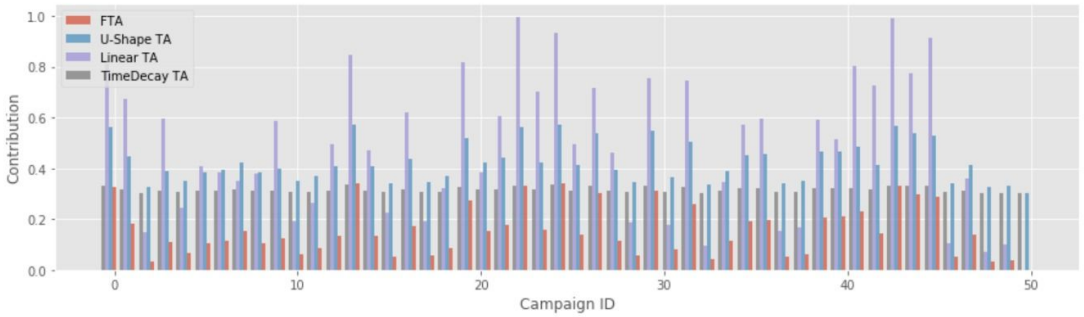
```
print(tdta)
```

```
[0.25435011 0.22924842 0.15026485 0.02747253 0.0748156  0.00899901
 0.17333333 0.08820404 0.05899281 0.10733696 0.16540881 0.08183882
 0.08924303 0.08979228 0.05631692 0.16771379 0.09270073 0.20265487
 0.02825485 0.13768116 0.04742991 0.07648725 0.01951557 0.18729282
 0.10232558 0.09719008 0.12943201 0.02126789 0.09130435 0.15259027
 0.0993228  0.15963402 0.0269331  0.03910615 0.20363636 0.22115385
 0.06551724 0.10381232 0.18139535 0.01486486 0.15384615 0.10909091
 0.221875   0.15       0.07155963 0.0109589  0.1862069  0.08403361
 0.24444444 0.21875   ]
```

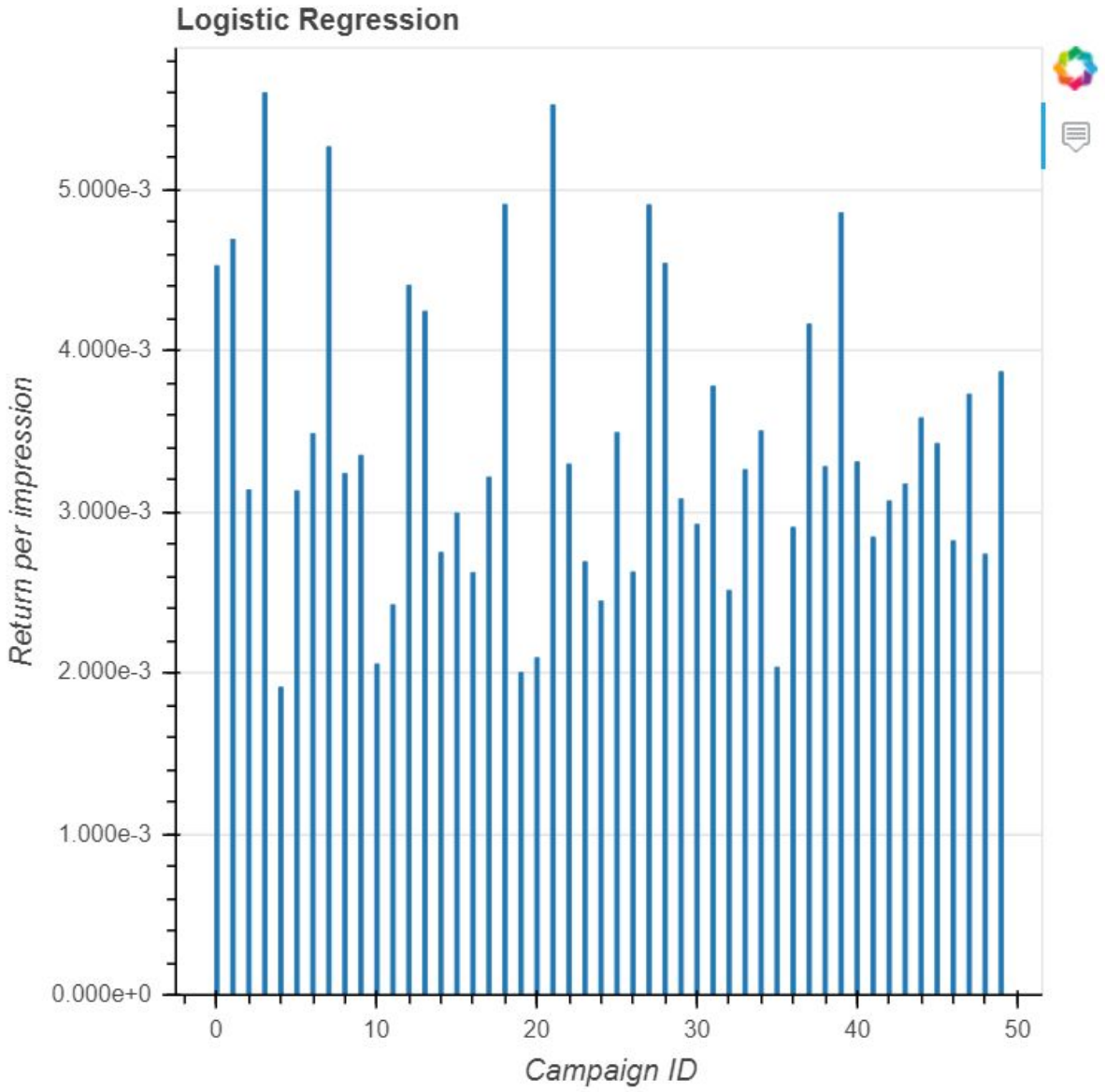

Time decay Attribution
0.25
0.2
0.15
0.1
0.05
0
0
10
20
30
40
50
Campaign ID

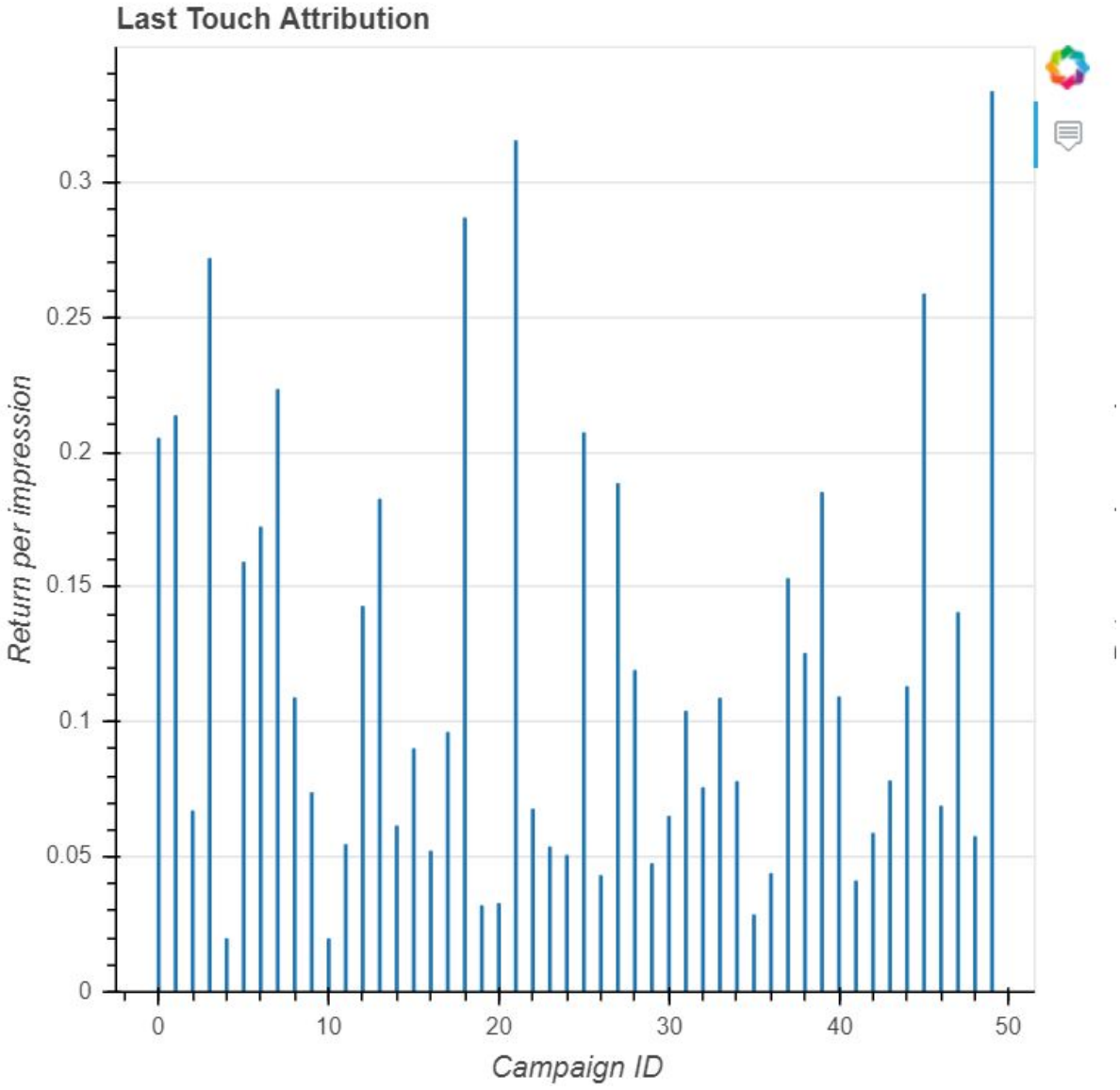
In [61]:

```
fig = plt.figure(figsize=(15, 4))
ax = fig.add_subplot(111)
#ratio = max(fta[idx]) / max(uta[idx])
plt.bar(np.linspace(0, len(campaign_idx), len(campaign_idx)), fta[campaign_idx], width=0.2, alpha=0.7, label='FTA' )
plt.bar(np.linspace(0, len(campaign_idx), len(campaign_idx)) - 0.2, uta[campaign_idx], width=0.2, alpha=0.7, label='U-Shape T
plt.bar(np.linspace(0, len(campaign_idx), len(campaign_idx)) - 0.4, linear[campaign_idx], width=0.2, alpha=0.7, label='Linear
plt.bar(np.linspace(0, len(campaign_idx), len(campaign_idx)) - 0.6, tdta[campaign_idx], width=0.2, alpha=0.7, label='TimeDeca
plt.xlabel('Campaign ID')
plt.ylabel('Contribution')
plt.legend(loc='upper left')
plt.show()
```



Logistic Regression & Last Touch Attribution


Logistic Regression
5.000e-3
4.000e-3
3.000e-3
2.000e-3
1.000e-3
0.000e+0
Return per impression
0
10
20
30
40
50
Campaign ID



Simulation

Simulation algorithm is used to evaluate the performance of the original attribution weights produced by the models, as well as various transformations of these weights and decide the best model among the models generated.

```
def simulate_budget_roi(df, budget_total, attribution, verbose=False):
    budgets = np.ceil(attribution * (budget_total / np.sum(attribution)))

    if(verbose):
        print(budgets)

    blacklist = set()
    conversions = set()
    for i in range(df.shape[0]):
        campaign_id = get_campaign_id(df.loc[i]['campaigns'])
        jid = df.loc[i]['jid']
        if jid not in blacklist:
            if budgets[campaign_id] >= 1:
                budgets[campaign_id] = budgets[campaign_id] - 1
                if(df.loc[i]['conversion'] == 1):
                    conversions.add(jid)
            else:
                blacklist.add(jid)

        if(verbose):
            if(i % 10000 == 0):
                print('{:.2%} : {:.2%} budget spent'.format(i/df.shape[0], 1.0 - np.sum(budgets)/budget_total ))

        if(np.sum(budgets) < budget_total * 0.02):
            break

    return len(conversions.difference(blacklist))
```

```
pitches = [0.5, 1.0, 1.5, 2.0, 2.5]
attributions = [lta, fta, uta, tdta]

for i, pitch in enumerate(pitches):
    for j, attribution in enumerate(attributions):
        reward = simulate_budget_roi(df6, 10000, attribution**pitch)
        print('{} {} : {}'.format(pitch, j, reward))
```

The value are as follows -

```
0.5 0 : 713
0.5 1 : 713
0.5 2 : 632
0.5 3 : 605
1.0 0 : 828
1.0 1 : 828
1.0 2 : 651
1.0 3 : 608
2.0 0 : 977
2.0 1 : 977
2.0 2 : 719
2.0 3 : 618
3.0 0 : 1200
3.0 1 : 1200
3.0 2 : 800
3.0 3 : 632
```

References

https://blog.griddynamics.com/cross-channel-marketing-spend-optimization-deep-learning/

https://github.com/ikatsov/tensor-house/blob/master/promotions/channel-attribution-lstm.ipynb

https://www.marketingevolution.com/marketing-essentials/multi-touch-attribution

https://docs.bokeh.org/en/latest/docs/user_guide/notebook.html

https://medium.com/@mortenhegewald/marketing-channel-attribution-using-markov-chains-101-in-python-78fb181ebf1e

https://github.com/bokeh/bokeh-notebooks/blob/master/tutorial/00%20-%20Introduction%20and%20Setup.ipynb