Adstock is a term used in advertising to describe the gradual decline in the effectiveness of an advertisement over time.

We have used Bass Diffusion Model to calculate advertising adstock.

Bass Diffusion Model

The Bass Diffusion Model is a mathematical model used to predict the adoption of new products or ideas in a population. It was first introduced in 1969 by Frank Bass, a marketing professor.

The model assumes that there are **three types of individuals in a population**:

- **Innovators**- they are the first to adopt a new product or idea
- **Early adopters** - theyare the next group to adopt.
- **The majority**- of the population then adopts the product or idea over time.

The Bass Diffusion Model uses two parameters to describe the rate at which new products or ideas are adopted:

- The **innovation coefficient (p)**- represents the rate at which innovators adopt the new product or idea
- **Imitation coefficient (q)**- represents the rate at which the early adopters and the majority adopt the new product or idea.

This a model for analyzing the spread of new products or ideas in a population and can be used to calculate the adstock effect by assuming that the rate of adoption is influenced by past and current advertising spend, as well as the overall size of the population

It can be used to estimate the adstock rate for advertising by modeling the adoption of the advertised product or idea in a population.

1. The first step is to gather **historical data** on the performance of similar campaigns, such as the **number of individuals who have adopted the product or idea over time**. This data can be used **to estimate the values of the innovation coefficient (p) and the imitation coefficient (q)** which describe the rate at which individuals adopt the product or idea.
2. Next, we can use the estimated values **of p and q to predict the adoption rate of the advertised product or idea over time**. This prediction can be used to estimate the adstock rate, which is the gradual decline in the effectiveness of the advertisement over time.

```python
def estimate_adstock_rate(data, p, q):
    # data is a list of the number of individuals who have adopted the product or idea over time
    # p is the innovation coefficient
    # q is the imitation coefficient

    # Initialize an empty list to store the adstock values
    adstock_values = []

    # Loop through the data and calculate the adstock value at each time step
    for i in range(len(data)):
        # Initialize the adstock value to 0
        adstock_value = 0

        # Loop through previous time steps to calculate the adstock value
        for j in range(i):
            adstock_value += data[j] * (p + (1-p) * q) ** (i-j)

        # Append the adstock value to the list
        adstock_values.append(adstock_value)

    return adstock_values


adstock_values = estimate_adstock_rate(data, p, q)
print(adstock_values)
```

One of the ways to estimate the innovation coefficient (p) and the imitation coefficient (q) in the Bass Diffusion Model is Bayesian approach.

Bayesian Approach

We have used a Bayesian approach, which involves specifying prior distributions for p and q and using the observed data to update the distributions. This is using Markov Chain Monte Carlo (MCMC) methods, such as the NUTS, to sample from the posterior distribution of p and q.

We used the PyMC3 library to implement this Bayesian approach.

- It starts by specifying prior distributions for p and q using the **pm.Uniform** function.
- Then it defines the likelihood function using the Poisson distribution and the observed data, and connects it with the model.
- Finally, it performs Markov Chain Monte Carlo (MCMC) sampling using the **pm.sample** function to sample from the posterior distribution of p and q.

The implementation of the Bayesian approach to estimate the parameters may require more adjustments, such as the choice of the sampler, the number of draws, the tuning of the parameters, and the validation of the model.

```python
import pymc3 as pm
import numpy as np

# Define the model
with pm.Model() as model:
    # Prior distributions for p and q
    p = pm.Uniform("p", 0, 1)
    q = pm.Uniform("q", 0, 1)

    # Define the likelihood function
    likelihood = pm.Poisson("likelihood", mu=p + (1-p) * q ** np.arange(1, len(data) + 1), observed=data)

    # Perform Markov Chain Monte Carlo sampling
    trace = pm.sample(draws=5000, tune=2000, return_inferencedata=False)

# Extract the samples of p and q
p_samples = trace["p"]
q_samples = trace["q"]

# Summary statistics of the samples
p_mean = np.mean(p_samples)
p_std = np.std(p_samples)
q_mean = np.mean(q_samples)
q_std = np.std(q_samples)
```

Estimatinng Prior and Posterior distributions.

Uniform distribution

The **pm.Uniform** function is a probability distribution function that defines a uniform distribution over a given range of values. In this case, it specifies that the prior distribution of both p and q are uniform over the range [0, 1], meaning that all values between 0 and 1 have equal probability of being the true value of p and q.

This is one way to specify prior distributions for the parameters, and it's a non-informative prior, meaning that it doesn't impose any prior knowledge about the values of p and q. However, in some cases, we may have some prior knowledge about the parameters, and we may want to specify different prior distributions based on wer specific use case.

The prior distributions are used to express our uncertainty about the parameters before observing the data, and the posterior distributions are the updated distributions after observing the data.

Poisson distribution

The **pm.Poisson** function is a probability distribution function that defines a Poisson distribution, which is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space if these events happen with a known average rate and independently of the time since the last event.

Poisson distribution is a statistical concept that helps to describe how many events might occur in a fixed interval of time or space. For example, it can be used to calculate the average number of calls

received at a call center in a given hour, or the average number of customers arriving at a store in a given day. It tells us the likelihood of a certain number of events happening given the average rate at which they occur. The result is always a whole number, as it describes a count of events, like the number of calls or customers.

In this case, the likelihood function uses the Poisson distribution with the mean parameter **mu** equal to **p + (1-p) * q ** np.arange(1, len(data) + 1)**. This expression is the probability of adoption at each time step, where **p** is the probability of innovation, **q** is the probability of imitation, and **np.arange(1, len(data) + 1)** is a sequence of integers representing the time steps.

The likelihood function is conditioned on the observed data, meaning that the likelihood function uses the observed data as the actual number of events.

Markov Chain Monte Carlo (MCMC)

This likelihood function is used to connect the prior distributions of the parameters (p,q) and the observed data. The prior distributions, together with the likelihood function, define the posterior distributions of the parameters, which are the updated distributions after observing the data.

trace = pm.sample(draws=5000, tune=1000) is performing Markov Chain Monte Carlo (MCMC) sampling using the PyMC3 library.

pm.sample is a function that generates samples from the posterior distribution of the parameters, given the prior distributions and the likelihood function defined in the model. The **draws** argument specifies the number of samples to generate, and the **tune** argument specifies the number of initial samples to discard as "burn-in" samples.

The function uses an MCMC algorithm, such as the NUTS algorithm, to generate the samples. The NUTS algorithm is an adaptive algorithm that uses gradient information to automatically tune the proposal distribution, making it more efficient for high-dimensional models.

The posterior distribution of the parameters is the distribution of the parameters after taking into account the information from the observed data. In Bayesian statistics, the prior distribution of the parameters represents our beliefs about the parameters before observing the data, while the posterior distribution represents our updated beliefs after observing the data. The posterior distribution is calculated by applying Bayes' theorem, which states that the posterior is proportional to the product of the prior and the likelihood. In other words, the posterior is a reflection of the prior and the information in the data, and it gives us a more accurate estimate of the parameters.

In practice, the posterior distribution is often estimated using Markov Chain Monte Carlo (MCMC) sampling. The MCMC sample is a representation of the posterior distribution, and it can be used to make inferences about the parameters, such as calculating the mean and standard deviation, or

calculating credible intervals, which are intervals that contain the true values with a certain level of confidence.

Markov Chain Monte Carlo (MCMC) sampling is a method used to approximate complex probability distributions by generating a sequence of samples that converge to the true distribution. The samples can be used to estimate the properties of the distribution, such as the mean and standard deviation, as well as to perform other tasks such as parameter inference, model evaluation, and posterior predictive checking.

The basic idea behind MCMC sampling is to construct a Markov Chain, which is a sequence of states that satisfy the Markov property, meaning that the probability of being in a certain state depends only on the previous state. The chain is constructed so that it has the same stationary distribution as the target distribution.

For a layman, we can think of MCMC sampling as a way to explore the different possibilities for the values of the parameters in a model, and finding the values that are most likely to be correct. The method is used in many fields, such as economics, physics, and biology, to help make predictions and understand complex systems.

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [q, p]
```


```
Sampling 2 chains for 2_000 tune and 5_000 draw iterations (4_000 + 10_000 draws total) took 767 seconds.
There were 38 divergences after tuning. Increase `target_accept` or reparameterize.
There were 57 divergences after tuning. Increase `target_accept` or reparameterize.
The number of effective samples is smaller than 10% for some parameters.
```

```
p mean: 0.9702207392773027
p std: 0.11939538933364681
q mean: 0.6159126831769743
q std: 0.33079286465581476
```

NUTS (No-U-Turn Sampler)

We have used NUTS as a MCMC sampling methods. NUTS (No-U-Turn Sampler) is a computer algorithm used for understanding the likelihood of different outcomes of a particular event or situation. It is used in fields such as statistics and machine learning to make predictions about future events or trends.

Think of it like a compass that helps navigate through a dense forest. The compass guides us to the most likely paths and prevent us from taking on inefficient detours or dead-ends. In the same way, NUTS helps a computer navigate through a complex set of data to find the most likely outcomes.

This algorithm is useful for making predictions about the future, for example, in stock market analysis or weather forecasting. By understanding the likelihood of different outcomes, we can make more informed decisions based on the data. NUTS is efficient and effective, and it is a widely used method for understanding complex data and making predictions.

The **trace** variable will be a PyMC3 MultiTrace object, which contains the samples of the parameters. The samples can be used to estimate the posterior distributions of the parameters, such as the mean and standard deviation, as well as to perform other tasks such as posterior predictive checking, model evaluation, and parameter inference.

The number of draws and the number of tuning samples are hyperparameters of the sampling process, and the choice of these values depends on the specifics of the model, the complexity of the problem, and the desired level of precision.