CHANNELATTRIBUTION

Davide Altomare and David Loris

2022-02-07

This paper introduces ChannelAttribution, an open-source library for the estimation of Markov models from customer journey data. ChannelAttribution consists on a R package and a Python library that let to estimate Markov models easily and quickly.

channelattribution.io

1 Introduction

Library ChannelAttribution approaches attribution problem in a probabilistic way. It uses a k-order Markov representation to identifying structural correlations in the customer journey data. This would allow advertisers to give a more reliable assessment of the marketing contribution of each channel. The approach is the one presented in F. Anderl, I. Becker, F. v. Wangenheim, J.H. Schumann (2014): Mapping the customer journey: a graph-based framework for attribution modeling. Differently from them, ChannelAttribution uses stochastic simulations for the estimation process. In this way it is also possible to take into account conversion values and their variability in the computation of the channel importance. Moreover the package contains a function that estimates three heuristic models (first-touch, last-touch and linear-touch approach) for the same problem. The following paragraph is a gentle introduction to Markov model. It also contains some considerations on heuristic models.

2 First-order Markov Model

First-order Markov model is a probabilistic model used to model changing system. It assumes that future states depend only on current state. There is a large leterature about Markov models and different fields where this kind of models have been applied. In the following we will show how they can be applied to attribution problem in online marketing. Consider the following example in which we have 4 states: (START), A, B, (CONVERSION) and 3 paths recorded:

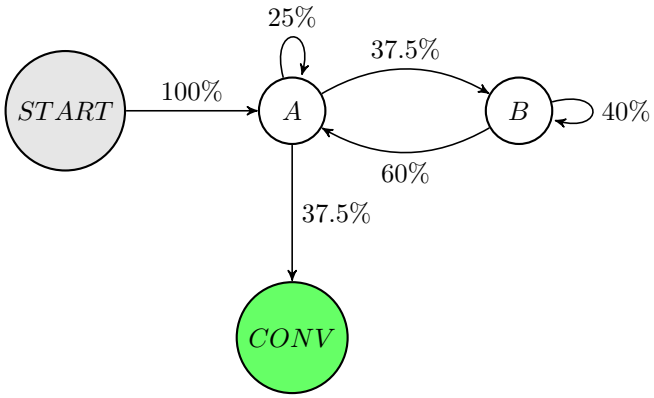
PATH	**CONVERSIONS**
(START) -> A -> B -> A -> B -> B -> A -> (CONV)	1
(START) ->A -> B -> B -> A -> A -> (CONV)	1
(START) -> A -> A -> (CONV)	1
TOTAL	**3**

For every couple of ordered states we count the number of directed edges:

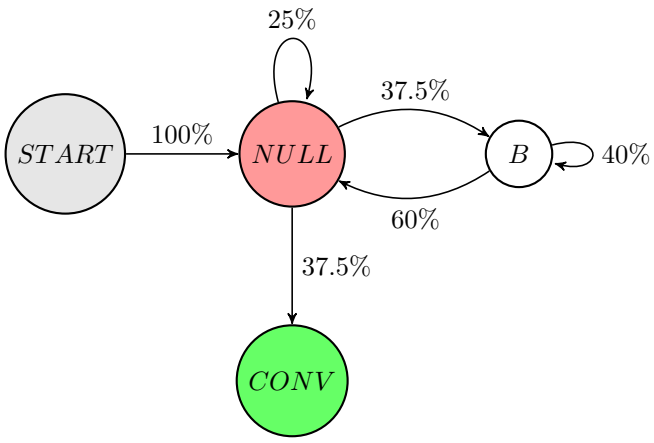
EDGE	**ARROW.COUNT**
(START) -> A	3
(START) -> B	0
A -> A	2
A -> B	3
A -> (CONV)	3
B -> A	3
B -> B	2
B -> (CONV)	0
TOTAL	**16**

From the table we can calculate the transition probabilities between states:
Now we have all the information to plot the Markov Graph:

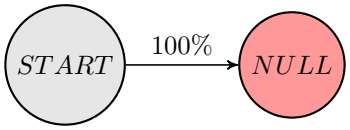
EDGE	ARROW.COUNT	TRANSITION.PROBABILITIES
(START) -> A	3	3/3
(START) -> B	0	0/3
TOT (START)	**3**	
A -> A	2	2/8
A -> B	3	3/8
A -> (CONV)	3	3/8
TOT A	**8**	
B -> A	3	3/5
B -> B	2	2/5
B -> (CONV)	0	0/5
TOT B	**5**	



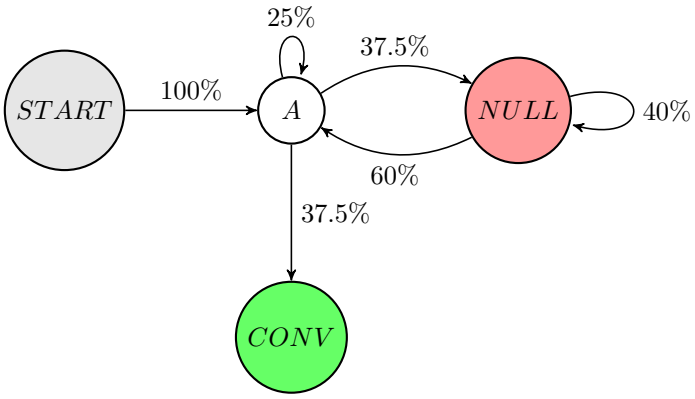
This kind of Markov Graph is called First-Order Markov Graph because the probability of reaching one state depends only on the previous state visited. From the Graph, or more clearly from the original data, we see that every path leads to conversion. Thus the conversion rate of the Graph is 1. Now we want to define a measure of channel importance using the relationship between states described by the Graph. Importance of channel A can be defined as the change in conversion rate if channel A is dropped from the Graph, or in other terms if channel A becomes a NULL state. A NULL state is an absorbing state so if one reaches this STATE can't move on.



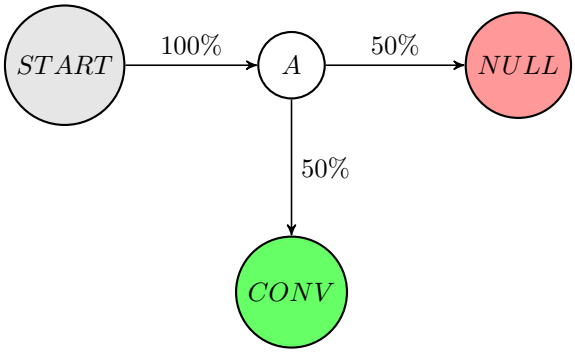
This Graph simplifies to:



In previous Graph it's easy to see that if channel A becomes a NULL state there is no way of reaching conversion from START state. So conversion rate of this Graph is 0. The conversion drops from 1 (conversion of the original Graph) to 0. Thus importance of channel A (defined as the change in conversion rate) is 1. In similar way we define the importance of channel B as the change in conversion rate if channel B is dropped from the Graph, or in other terms if channel B becomes a NULL state.



This Graph simplifies to:



In previous Graph we see the probability of reaching conversion from START state is 0.5. Conversion drops from 1 (conversion rate of the original Graph) to 0.5. Thus the importance of channel B (defined as the change in conversion rate) is **0.5**. Once we have the importance weights of every channel we can do a weighted imputation of total conversions (3 in this case) between channels.

CHANNEL	**CONVERSIONS**
A	2 [=3 x 1/(1+0.5)]
B	1 [= 3 x 0.5/(1+0.5)]
TOTAL	**3**

Now let's go back to the original data:

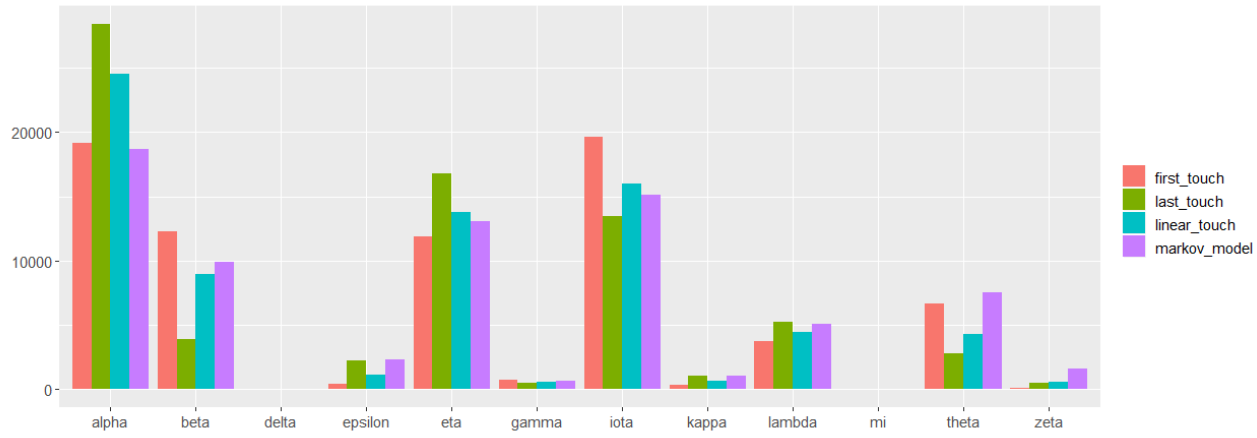
PATH	**CONVERSIONS**
(START) -> A -> B -> A -> B -> B -> A -> (CONV)	1
(START) ->A -> B -> B -> A -> A -> (CONV)	1
(START) -> A -> A -> (CONV)	1
TOTAL	**3**

We see that if we use a first-touch or last-touch approach, all the conversions are assigned to channel A, despite the important work of channel B in "conversion game" is clear from the data. The channel attribution problem can be viewed as a football match, to better understand how different approaches work. So channels can be viewed as players, paths are game actions and conversions are goals. Markov Model analyses relationships between game actions to understand the role of the player in scoring. While heuristic approach analyzes one action (path) at the time. So last-touch approach rewards only players who scored, while first-touch approach rewards only players who started the action. Linear approach rewards with the same credit to every player who took part the action, while time-decay approaches gives subjective weights to every player who took part the action. As we have seen Markov Model require as inputs paths and total conversions and does not require subjective assumptions, differently from heuristic approaches.

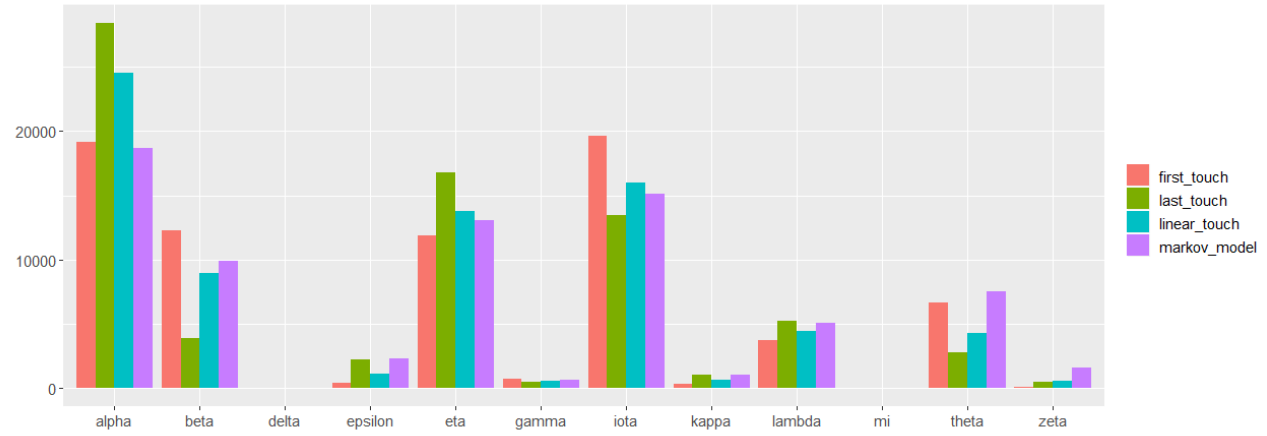
3 R package ChannelAttribution

In the following example we will show how R package ChannelAttribution can be used for multichannel attribution problem.

```r
#LOAD LIBRARIES AND DATA

library(ChannelAttribution)
library(reshape2)
library(ggplot2)
data(PathData)

#ESTIMATE HEURISTIC MODELS

H=heuristic_models(Data,"path","total_conversions",var_value="total_conversion_value")

#ESTIMATE MARKOV MODEL

M=markov_model(Data, "path", "total_conversions", var_value="total_conversion_value")

#PLOT TOTAL CONVERSIONS

R=merge(H,M,by="channel_name")
R1=R[,(colnames(R)%in%c("channel_name","first_touch_conversions","last_touch_conversions",
"linear_touch_conversions","total_conversion"))]
colnames(R1)=c("channel_name","first_touch","last_touch","linear_touch","markov_model")
R1=melt(R1,id="channel_name")

ggplot(R1, aes(channel_name, value, fill = variable)) +
ggtitle("")+
geom_bar(stat="identity", position = "dodge") +
theme(plot.title = element_text(hjust = 0.5))+
theme(text = element_text(size=14)) +
theme(plot.title=element_text(size=18)) +
theme(legend.title = element_blank()) +
ylab("") +
xlab("")
```



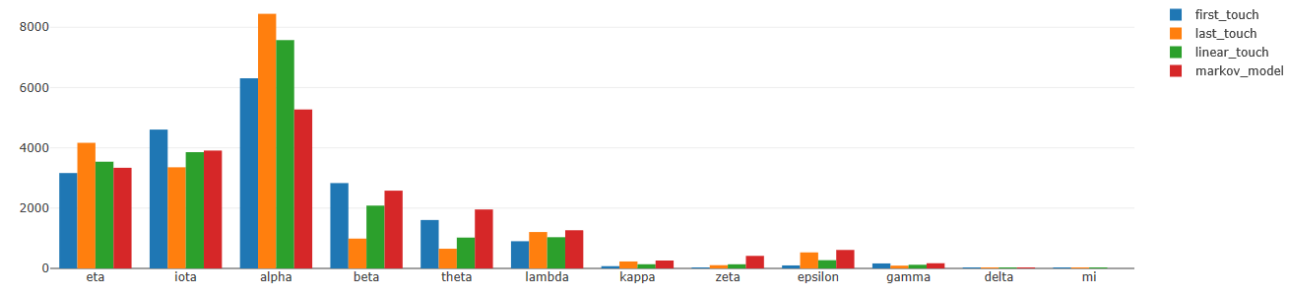

```r
#PLOT REVENUES

R2=R[,(colnames(R)%in%c("channel_name","first_touch_value","last_touch_value","linear_touch_value",
"total_conversion_value"))]
colnames(R2)=c("channel_name","first_touch","last_touch","linear_touch","markov_model")
R2=melt(R2,id="channel_name")

ggplot(R2, aes(channel_name, value, fill = variable)) +
ggtitle("")+
geom_bar(stat="identity", position = "dodge") +
theme(plot.title = element_text(hjust = 0.5))+
theme(text = element_text(size=14)) +
theme(plot.title=element_text(size=18)) +
theme(legend.title = element_blank()) +
ylab("") +
xlab("")
```



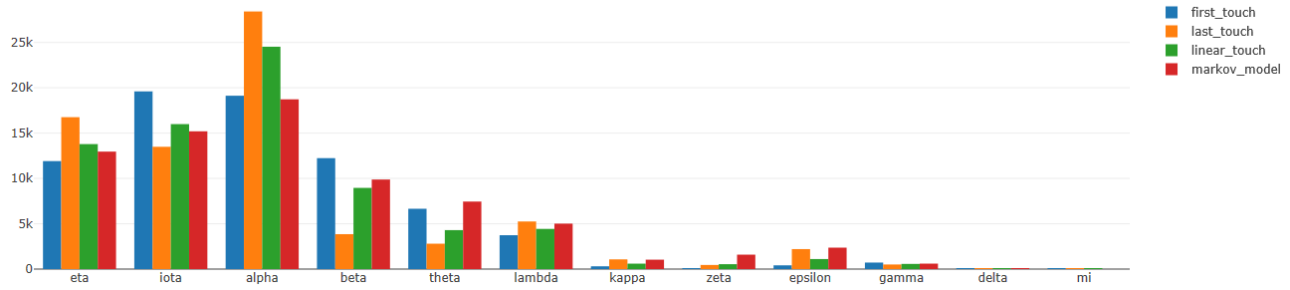
4 Python library ChannelAttribution

In the following example we will show how Python library ChannelAttribution can be used for multichannel attribution problem.

```
#LOAD LIBRARIES AND DATA

import numpy as np
import pandas as pd
from ChannelAttribution import *
import plotly.io as pio

Data = pd.read_csv("https://channelattribution.io/csv/Data.csv",sep=";")

#ESTIMATE HEURISTIC MODELS

H=heuristic_models(Data,"path","total_conversions",var_value="total_conversion_value")

#ESTIMATE MARKOV MODEL

M=markov_model(Data, "path", "total_conversions", var_value="total_conversion_value")

#PLOT TOTAL CONVERSIONS

R=pd.merge(H,M,on="channel_name",how="inner")
R1=R[["channel_name","first_touch_conversions","last_touch_conversions",\
"linear_touch_conversions","total_conversions"]]
R1.columns=["channel_name","first_touch","last_touch","linear_touch","markov_model"]
R1=pd.melt(R1, id_vars="channel_name")

data = [dict(
  type = "histogram",
  histfunc="sum",
  x = R1.channel_name,
  y = R1.value,
  transforms = [dict(
    type = "groupby",
    groups = R1.variable,
  )],
)]

fig = dict({"data":data})
pio.show(fig,validate=False)
```




```
#PLOT REVENUES

R2=R[["channel_name","first_touch_value","last_touch_value",\
"linear_touch_value","total_conversion_value"]]
R2.columns=["channel_name","first_touch","last_touch","linear_touch","markov_model"]

R2=pd.melt(R2, id_vars="channel_name")

data = [dict(
  type = "histogram",
  histfunc="sum",
  x = R2.channel_name,
  y = R2.value,
  transforms = [dict(
    type = "groupby",
    groups = R2.variable,
  )],
)]


fig = dict({"data":data})
pio.show(fig,validate=False)
```



5 Transaction level attribution with Markov Model

5.1 Transaction level attribution problem

Transaction level attribution is problematic using Markov models. Markov model can be considered a "global" approach because it makes attribution considering all the paths together. Instead heuristic models are "local" approaches because they make attribution considering one path at the time and then global attribution for each channel is obtained through aggregation. Using Markov model one can not go from global to local attribution in a unique way. Because Markov model returns the aggregate result and you can not go from this aggregation to single path attribution.

Why Markov model is a global approach? Markov model aggregates real paths to build a Markov graph, a graphical representation of a transition matrix. Markov graph is a mathematical representation of the dynamics between channels. Markov model generates millions of random paths from Markov graph. Random paths are used to calculate importance weights (removal effects) for each channel. At the end of the process importance weights are normalized and multiplied by total conversions (the overall conversions observed for all the paths considered) to make attribution for each channel. Thus Makov model is global because first it aggregates paths to build transition matrix and then it works with simulate paths “forgetting” each single real path.

5.2 Functions for transaction level attribution

There are two ways you can make transaction-level attribution using ChannelAttribution: the APIs included in the open-source Python library or installing our commercial version ChannelAttributionPro which is available both for Python and for R. The main difference between them is that through the APIs your data is encrypted and sent to our server for the elaboration while ChannelAttributionPro can be installed locally and no data is transferred. Moreover, the APIs include a maximum size limit of 10MB for the number of customer journeys that can be elaborated while ChannelAttributionPro includes a trial period where the functions can be used without any limitations. If you want to try ChannelAttributionPro write us at info@channelattribution.io.

There are two functions in ChannelAttribution that are useful for transaction level attribution: markov_model_local and new_paths_attribution.

5.2.1 Function generate_token()

You can use this function to generate a token that enables the use of our APIs to make path-level attribution. An email containing your personal token will be sent to the email address indicated.

generate_token()

Parameters

email	your business/university email at which we will send your personal token
company	name of your company/university
job	your job

5.2.2 Function markov_model_local()

markov_model_local can be used to make transaction level attribution.

markov_model_local()

Parameters

Data	data.frame containing customer journeys data
var_path	name of the column containing paths
var_conv	name of the column containing total conversions
var_value	name of the column containing total conversion value
var_null	name of the column containing total paths that do not lead to conversions
order	Markov model order
sep	separator between the channels
ncore	number of threads used in computation
conv_par_gob	convergence parameter for the global attribution. The estimation process ends when the percentage of variation of the results over different repetitions is less than conv_par_loc (this is equal to conv_par parameter of markov_model function)
conv_par_loc	convergence parameter for the local attribution. The estimation process ends when the percentage difference between global and aggregated local attribution is less than conv_par_loc
verbose	if TRUE, additional information about process convergence will be shown

Output

path_attribution

idpath	numerical id for the path considered
path	Path
channel	channel name
total_conversions_weight	normalized channel weight used to make conversion attribution at path level
total_conversions_attribution	conversions attributed to channel considered at path level
total_conversion_value_weight	normalized channel weight used to make conversion value attribution at path level
total_conversion_value_attribution	conversion value attributed to channel considered at path level

removal_ effects

channel_name	channel name
removal_effects_conversion	removal effects for conversion attribution from global attribution
removal_effects_conversion_value	removal effects for value attribution from global attribution

corrective_factors – total conversions

channel	channel name
perc_corr_j	correction percentage at iteration j from the iterative matching process between global and local attribution

corrective_factors – total conversion value

channel	channel name
perc_corr_j	correction percentage at iteration j from the iterative matching process between global and local attribution

5.2.3 Function new_paths_attribution()

new_paths_attribution uses removal effects and corrective factors from markov_model_local to calculate weights for channels belonging to new paths. This function is useful for weights calculation in **real time attribution**.

new_paths_ attribution()

Parameters

tab_new	data.frame containing new paths
var_path	name of the column containing paths
Tab_re	removal effects from global attribution
D_tab_corr	corrective factors from local attribution
Sep	separator between channels

Output

path	path
Idpath	numerical id for the path considered
channel	channel
weight_total_conversions	normalized channel weight used to make conversion attribution at path level
weight_total_conversion_value	normalized channel weight used to make conversion value attribution at path level

5.3 Examples

In the following it will be shown how these functions can be used to make transaction level attribution using R or Python.

5.3.1 APIs

Our APIs can be used to test path-level attribution on your data. Running the functions, your data will be encrypted and sent to our server where they will be elaborated. Then the output will be encrypted and sent to your local session. We do not share or store your data that will be canceled from our server immediately after the end of the elaboration.

```
#######
#PYTHON
#######
```

```
from ChannelAttribution import *

#Download data

Data = pd.read_csv("https://channelattribution.io/csv/Data.csv",sep=";")

#Generate token

generate_token(email="mario.rossi@data.com", job="data scientist", company="data.com")

#Train

res=markov_model_local_api(token, Data, var_path="path", var_conv="total_conversions",\
var_value="total_conversion_value", var_null="total_null", order=1, sep=">")

#Tab_re and D_tab_corr are output from train, you need to store them if you want to make real time attribution
   on new paths

Tab_re=res["removal_effects"].copy()
D_tab_corr=res["corrective_factors"].copy()

#tab_new is a sample dataset containing new paths to be attributed

tab_new=Data.loc[0:5,["path"]]

#real time attribution on new paths

res_new=new_paths_attribution_api(token, tab_new,var_path="path", Tab_re=Tab_re, D_tab_corr=D_tab_corr, sep=">"
   )
```

5.3.2 ChannelAttributionPro

ChannelAttributionPro is the commercial version of ChannelAttribution. You can try it for free asking to info@channelattribution.io. You will receive the installation instructions and a password. At the beginning of each execution, ChannelAttributionPro will send your password to our server that will enable the computation. No data will be transferred to our server and all the execution will be made locally.

5.3.3 Python

```
from ChannelAttributionPro import *

password="youpassword"

#Download data

Data = pd.read_csv("https://channelattribution.io/csv/Data.csv",sep=";")

#Train

res=markov_model_local(Data, var_path="path", var_conv="total_conversions",\
var_value="total_conversion_value", var_null="total_null", order=1, sep=">", ncore=1, conv_par_glob=0.05,\
conv_par_loc=0.01,verbose=True,server="app.channelattribution.net", password=password)

#Tab_re and D_tab_corr are output from train, you need to store them if you want to make real time attribution
   on new paths

Tab_re=res["removal_effects"].copy()
D_tab_corr=res["corrective_factors"].copy()

#tab_new is a sample dataset containing new paths to be attributed

tab_new=Data.loc[0:5,["path"]]

#real time attribution on new paths

res_new=new_paths_attribution(tab_new,var_path="path",Tab_re=Tab_re,D_tab_corr=D_tab_corr,sep=">",\
server="app.channelattribution.net", password=password)
```

5.3.4 R

```
library(ChannelAttributionPro)

#Load Data

data(PathData)

password="youpassword"

#Train

res=markov_model_local(Data, var_path="path", var_conv="total_conversions",
var_value="total_conversion_value", var_null="total_null", order=1, sep=">", ncore=1, conv_par_glob=0.05, conv_
   par_loc=0.01, verbose=TRUE,
server="app.channelattribution.net", password=password)

#Tab_re and D_tab_corr are output from train, you need to store them if you want to make real time attribution
   on new paths

#Save Tab_re and D_tab_corr

Out_ML=list()
Out_ML[["Tab_re"]]=res[['removal_effects']]
Out_ML[["D_tab_corr"]]=res[['corrective_factors']]

save(Out_ML,file=".../Out_ML.RData")
rm(Out_ML)

#tab_new is a sample dataset containing new paths to be attributed

tab_new=Data[0:5,"path"]

#load Tab_re and D_tab_corr

load(file="../Out_ML.RData")

Tab_re=Out_ML[["Tab_re"]]
D_tab_corr=Out_ML[["D_tab_corr"]]

#real time attribution on new paths

res_new=new_paths_attribution(tab_new,var_path="path",Tab_re=Tab_re,D_tab_corr=D_tab_corr,sep=">",
server="app.channelattribution.net", password=password)
```