E-Commerce Dataset

Usage Data Tasks (1) Discussions Activity Metadata Version 1 (current) Vote Download

1. Part 1: Marketing Attribution Model: The Shapley value approach

Import Dataset from BigQuery

```
# [ ] import google.colab bigquery auth
auth.authenticate_user()

# [ ] from google.colab import bigquery
project_id = 'bio-storage-lab-service'
# if ip is gcp, use default project
bq = bigquery.Client(project=project_id)
for dataset in client.list_datasets():
    print("Found dataset: {}".
                                                                    ")
```

Import Dataset from BigQuery

```
# [ ] import pandas as pd

project_id = 'bio-storage-lab-service'
# if ip is gcp, use default project
bq = bigquery.Client()

query = """SELECT * FROM `bio-storage-lab-service.olist_ecommerce_dataset.*`"""
df = bq.query_and_wait(query).to_dataframe()
df.shape
```

```
(1,550,000, 12)
```

```
# [ ] df.head()
```

	full_fullvisitorid	date	visitid	visitStartTime	channelGrouping	city_name	city_median	trafficSource.medium	trafficSource.source	trafficSource.campaign
0	9231903503503503503	2016-10-17	1476705607	1476705607	Organic Search	sao	organic	Organic Search	bing	(not set)
1	5239426503503503503	2016-10-17	1476705607	1476705607	Organic Search	sao	organic	Organic Search	bing	(not set)
2	3540967503503503503	2016-10-17	1476705607	1476705607	Organic Search	sao	organic	Organic Search	bing	(not set)
3	9231903503503503503	2016-10-17	1476705607	1476705607	Organic Search	sao	organic	Organic Search	bing	(not set)
4	5239426503503503503	2016-10-17	1476705607	1476705607	Organic Search	sao	organic	Organic Search	bing	(not set)

Keep the data of customers that includes the activity only for 30 days time window

```
# [ ] df_ga_sessions['fullvisitorid'] = df_ga_sessions['fullvisitorid'].astype(str)

# [ ] df_ga_sessions = df_ga_sessions[df_ga_sessions['fullvisitorid'] != 'nan']

# [ ] df_ga_sessions = df_ga_sessions[df_ga_sessions['fullvisitorid'] != 'nan']

# [ ] df_ga_sessions = df_ga_sessions[~df_ga_sessions['fullvisitorid'].
                                                                    ]

# [ ] df_ga_sessions = df_ga_sessions[df_ga_sessions['fullvisitorid'] != '']

# [ ] df_ga_sessions = df_ga_sessions[df_ga_sessions['fullvisitorid'].str.strip() != '']
# [ ] df_ga_sessions.shape
```

The feature is the dataset named 'transactionid' is included or 0 or 1 in order to use it as the indicator of whether there is Conversion or No conversion

```
# [ ] df['transactionid'] = df['transactionid'].fillna(0)

# [ ] df.isin(['transactionid'] ).sum('transactionid') 1 )

# [ ] df['transactionid'] = df.isin(['transactionid'] )
```

Let the channel ordered based on the customers (the coefficients of channels) and the sum of their conversions

```
# [ ] path_df = df.groupby('fullvisitorid').agg({'channelGrouping': lambda(x):tuple(x.
                                                                    ),'transactionid':
# [ ] path_transaction = df.drop_duplicates('fullvisitorid', keep='last')[ ['fullvisitorid',
                                                                    ,'transactionid']]

# [ ] path_df.shape
```

```
(1,000,000, 2)
```

```
# [ ] path_df = path_df.merge(path_transaction, on=['fullvisitorid'])
# [ ] path_df.shape
```

```
(1,000,000, 3)
```

```
# [ ] path_df.sample(5)
```

	fullvisitorid	channelGrouping	transactionid
1	0000010278925465295	(Organic Search)	0
2	0000015405466367167	(Organic Search)	0
3	0000017556317015703	(Organic Search)	0
4	0000018332124353518	(Direct)	0
5	0000028467278085992	(Paid Search)	0

```
# [ ] path_df.sample(5)
```

	fullvisitorid	channelGrouping	transactionid	path
0	0000010278925465295	(Organic Search)	0	Organic Search
1	0000015405466367167	(Organic Search)	0	Organic Search
2	0000017556317015703	(Organic Search)	0	Organic Search
3	0000018332124353518	(Direct)	0	Direct
4	0000028467278085992	(Paid Search)	0	Paid Search
996,996	9999963156278167359	(Organic Search)	0	Organic Search
996,997	9999972101104966	(Organic Search)	0	Organic Search
996,998	9999975350458840	(Organic Search)	0	Organic Search
996,999	9999979458413465	(Organic Search)	0	Organic Search
997,000	9999989602693508948	(Organic Search)	0	Organic Search

996,000 rows × 3 columns

```
# [ ] import numpy as np

# [ ] path_df['path'] = np.where(path_df['transactionid'] == 0,path_df['channelGrouping'].apply(lambda x: str(x).
                                                                    ),path_df['channelGrouping'].apply(lambda x: str(x).
                                                                    ))

# [ ] path_df.sample(5)
```

	fullvisitorid	channelGrouping	transactionid	path
0	0000010278925465295	(Organic Search)	0	Organic Search
1	0000015405466367167	(Organic Search)	0	Organic Search
2	0000017556317015703	(Organic Search)	0	Organic Search
3	0000018332124353518	(Direct)	0	Direct
4	0000028467278085992	(Paid Search)	0	Paid Search

```
# [ ] conversions_df = path_df.groupby('path').agg({'transactionid': sum})
conversions_df
```

	transactionid
path	
(Other)	0
(Other),(Other)	0
(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0
(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other),(Other)	0

```
# [ ] conversions_df.columns = ['channels_subset', 'conversion_sum']
conversions_df
```

	channels_subset	conversion_sum
0	(Other)	0
1	affiliates	4
2	Direct	810
3	Display	1
4	Organic Search	1,172
5	Paid Search	125
6	Referral	1,752
7	Social	1

Create the relevant functions in order to calculate Shapley values

```
# [ ] import itertools
import numpy as np, scipy, sklearn
from sklearn.utils import shuffle, resample, compute_class_weight
from sklearn.metrics import *
```

Function that returns all possible combinations of the channel

```
# [ ] def powerset(iterable):
    "powerset([1,2,3]) --> () () () () () () ()"
    s = list(iterable)
    return itertools.
                                                                    ,
                                                                    ,
                                                                    ,
```

1	+++aKiAiXhTfaqCLC/kyWA==	367	1	40.290	369
2	+++tkVCLHtuRHFq/0FGt9A==	473	2	31.110	475
3	++/+J8M423AmQPekdSCiUw==	700	7	127.365	708
4	++/G67YHZTMKdpvANeYPLw==	424	6	172.425	433
...	...	...	...	...	...
837330	zzyyHmcBdlPu2LT05kFnrQ==	233	1	40.155	235
837331	zzz2aWARN6OJELfdgOZpVw==	36	3	63.510	42
837332	zzzCCNCA1718e2kRiyrHpJQ==	366	2	37.155	368
837333	zzzj3hW4oytUbaZbp6hqiw==	378	1	34.005	380
837334	zzzkp+wC5LHXLSmDg2F9Fg==	234	6	78.840	484

837335 rows × 5 columns

Calculating average monetary values per order

```
[26] rfm_df["Monetary"] = rfm_df["Monetary"] / rfm_df["Frequency"]
     rfm_df.head()
```

	CustomerId	Recency	Frequency	Monetary	T
0	+++SJgx/2lJ+dXq7vF8COg==	110	1	41.9400	112
1	+++aKiAiXhTfaqCLC/kyWA==	367	1	40.2900	369
2	+++tkVCLHtuRHFq/0FGt9A==	473	2	15.5550	475
3	++/+J8M423AmQPekdSCiUw==	700	7	18.1950	708
4	++/G67YHZTMKdpvANeYPLw==	424	6	28.7375	433

Removing one-time purchases from dataset and copying the final dataset

```
rfm_df = rfm_df[(rfm_df['Frequency'] > 1)]
cltv = rfm_df.copy()
rfm_df.head()
```

	CustomerId	Recency	Frequency	Monetary	T
2	+++tkVCLHtuRHFq/0FGt9A==	473	2	15.5550	475
3	++/+J8M423AmQPekdSCiUw==	700	7	18.1950	708
4	++/G67YHZTMKdpvANeYPLw==	424	6	28.7375	433
5	++/GTDXvJzF11ZlUz81SPg==	203	2	35.1525	566
6	++/tX5P6Nzl76EJCfgdbvA==	702	2	25.7485	704

Install and import the relevant libraries to train the BG/NBD model to estimate the expected sales in a given period of time. The trained BG/NBD model will be used in predicting customer lifetime value in the following steps.

```
[ ] pip install Lifetimes==0.2.2.2
```

```
Collecting Lifetimes==0.2.2.2
  Downloading Lifetimes-0.2.2.2.tar.gz (500 kB)
     |████████████████████████████████| 500 kB 5.1 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from Lifetimes==0.2.2.2) (1.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from Lifetimes==0.2.2.2) (1.4.1)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (from Lifetimes==0.2.2.2) (1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.19->Lifetimes==0.2.2.2) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.19->Lifetimes==0.2.2.2) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.19->Lifetimes==0.2.2.2) (1.15.0)
Building wheels for collected packages: Lifetimes
  Building wheel for Lifetimes (setup.py) ... done
  Created wheel for Lifetimes: filename=Lifetimes-0.2.2.2-py3-none-any.whl size=521866 sha256=c8392581e9405fbf33d8a1cff5c4b6402a6e0084b2d98a5e187c2a0540dc8260
  Stored in directory: /root/.cache/pip/wheels/ab/04/7d/368d8eab4dd36c80a3c533c3d6dcc78084c983135193bb0979
Successfully built Lifetimes
Installing collected packages: Lifetimes
Successfully installed Lifetimes-0.2.2.2
```

```
from lifetimes import BetaGeoFitter
from lifetimes import GammaGammaFitter
from lifetimes.plotting import plot_probability_alive_matrix
```

Downgrade Scipy, otherwise the code using the lifetimes libraries do not work

```
[ ] pip install -I scipy==1.1.0
```

```
Collecting scipy==1.1.0
  Downloading scipy-1.1.0-cp37-cp37m-manylinux1_x86_64.whl (31.2 MB)
     |████████████████████████████████| 31.2 MB 1.3 MB/s
Collecting numpy>=1.8.2
  Downloading numpy-1.21.5-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (15.7 MB)
     |████████████████████████████████| 15.7 MB 36.0 MB/s
Installing collected packages: numpy, scipy
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
tensorflow 2.8.0 requires tf-estimator-nightly==2.8.0.dev2021122109, which is not installed.
pymc3 3.11.4 requires scipy>=1.2.0, but you have scipy 1.1.0 which is incompatible.
plotnine 0.6.0 requires scipy>=1.2.0, but you have scipy 1.1.0 which is incompatible.
jax 0.3.4 requires scipy>=1.2.1, but you have scipy 1.1.0 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is incompatible.
Successfully installed numpy-1.21.5 scipy-1.4.1
WARNING: The following packages were previously imported in this runtime:
  [numpy,scipy]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

Checking BG/NBD model assumption and requirements

```
[38] print(cltv[['Monetary', 'Frequency']].corr())  # Correlation between monetary ve recency variables
     cltv["Frequency"] = cltv["Frequency"].astype(int) # Type of frequency variable should be integer for BG-NBD model
```

```
             Monetary  Frequency
Monetary     1.000000  -0.152455
Frequency   -0.152455   1.000000
```

Creating BG-NBD Model

```
[29] bgf = BetaGeoFitter(penalizer_coef=0.001) # model object
     bgf.fit(cltv['Frequency'], cltv['Recency'], cltv['T']) # model fitting
```

```
<lifetimes.BetaGeoFitter: fitted with 718908 subjects, a: 7.98, alpha: 0.42, b: 127.24, r: 0.23>
```

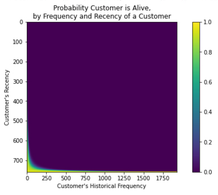
Prediction of expected number of transaction for each customer for one year (365 days)

```
[30] cltv['expctd_num_of_purch'] = bgf.predict(365, cltv['Frequency'], cltv['Recency'], cltv['T'])
     cltv.sort_values("expctd_num_of_purch",ascending=False).head()
```

	CustomerId	Recency	Frequency	Monetary	T	expctd_num_of_purch
337971	NnH34s/qTqamya3uGopDbA==	8	3	31.935000	10	16.573302
621250	jSsdA4z3bPO6mSR5UtaLDw==	6	3	28.960000	8	16.501663
260561	Ht3M9FomY/YyOTWnFQecb0Q==	6	3	71.430000	8	16.501663
805911	xYIzz/m4ad5EPRRBMrEiKg==	10	3	32.495000	12	16.452870
801275	xCcPO/cBc5GT/WXgMk5GLQ==	11	3	59.309333	13	16.353793

```
[31] import matplotlib.pyplot as plt
     %matplotlib inline
     plt.subplots(figsize=(10, 5))
     plot_probability_alive_matrix(bgf)
     plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/lifetimes/estimation.py:636: RuntimeWarning: overflow encountered in power
  return 1. / (1 + (frequency > 0) * (a / (b + frequency - 1)) * ((alpha + T) / (alpha + recency)) ** (r + frequency))
```



Creating Gamma-Gamma Model

```
[32] ggf = GammaGammaFitter(penalizer_coef=0.01) # model object
     ggf.fit(cltv['Frequency'], cltv['Monetary']) # model fitting
```

```
<lifetimes.GammaGammaFitter: fitted with 718908 subjects, p: 3551.95, q: 1.62, v: 0.00>
```

Prediction of expected amount of average profit

```
[33] cltv["expct_avg_spend"] = ggf.conditional_expected_average_profit(cltv['Frequency'], cltv['Monetary'])
     cltv.head()
```

	CustomerId	Recency	Frequency	Monetary	T	expctd_num_of_purch	expct_avg_spend
2	+++tkVCLHtuRHFq/0FGt9A==	473	2	15.5550	475	1.502907	15.556009
3	++/+J8M423AmQPekdSCiUw==	700	7	18.1950	708	3.119596	18.195223
4	++/G67YHZTMKdpvANeYPLw==	424	6	28.7375	433	4.172967	28.737454
5	++/GTDXvJzF11ZlUz81SPg==	203	2	35.1525	566	0.841057	35.151802
6	++/tX5P6Nzl76EJCfgdbvA==	702	2	25.7485	704	1.037281	25.748621

Calculating customer lifetime value by using BG-NBD and GammaGamma models

```
[35] cltv["cltv_one_year"] = ggf.customer_lifetime_value(bgf,
                                                         cltv['Frequency'],
                                                         cltv['Recency'],
                                                         cltv['T'],
                                                         cltv['Monetary'],
                                                         time=12,  # 12 month
                                                         discount_rate=0.01)

     cltv.sort_values("cltv_one_year",ascending=False).head()
```

	CustomerId	Recency	Frequency	Monetary	T	expctd_num_of_purch	expct_avg_spend	cltv_one_year
167575	Aob9BLs2SSN7cETDVJ/prnw==	60	2	1713.150000	62	7.843520	1713.003183	12600.586069
477507	YRQH8jRVa66QQ9VblvlGwg==	743	7	4505.421429	748	2.976359	4505.310003	12449.078970
84679	4Sze1o4Ua+h9LH5nF6swbQ==	48	5	902.046000	71	9.123164	902.015524	7777.199266
311842	LmNg5axPQ9rOhtu8Httkwg==	115	4	941.853750	120	8.093645	941.813922	7139.142099
685811	oNcC3PM+Q3Z2gqGJTcKrzg==	2	2	417.432000	4	15.206619	417.398014	6191.221286