databricks Group Project

BDM group project - Targeting Model - Group D

Group members:

- Matej Federic: M20210118
- Victor Almeida: M20210270
- Andrea Verbaro: M20210053

Summary

Business request

After analysing the Home AB test, you've found out that the main difference between the two homes is the List 3 component. Based on the A/B testing we want to boost the performance knowing what home version we should show to each customer, individually. Hence we would need to develop a Recommendation / targeting model to pin point what are the customers with higher probability to convert given the home list 3 to be seen.

Contents

1 - Importing Dataframes

```
# imports
import pyspark.pandas as ps
from pyspark.sql.window import Window
from pyspark.sql import functions as f
from pyspark.sql.types import StringType, ArrayType, LongType, DateType,
BooleanType, StructType, StructField
```

```
spark.sparkContext.setLogLevel("WARN")
```

```
%sh
wget https://www.dropbox.com/s/0prlh78825xy4tk/BDM_DATA.zip --quiet
unzip -d ./bdm_data/ BDM_DATA.zip
```

```
Archive:  BDM_DATA.zip
   creating: ./bdm_data/cust_df/
  inflating: ./bdm_data/cust_df/_SUCCESS
  inflating: ./bdm_data/cust_df/_committed_7878371389005906564
  inflating: ./bdm_data/cust_df/_committed_5076822134895256271
  inflating: ./bdm_data/cust_df/_committed_3520508812338357534
  inflating: ./bdm_data/cust_df/_started_5076822134895256271
  inflating: ./bdm_data/cust_df/part-00000-tid-5076822134895256271-fa5ebda2-
8174-4bce-b5ef-fec6a0376257-23668-1-c000.csv
   creating: ./bdm_data/orders_df/
  inflating: ./bdm_data/orders_df/_committed_1749678841354862380
  inflating: ./bdm_data/orders_df/_committed_4173638812266093034
  inflating: ./bdm_data/orders_df/_committed_4196442019492113282
  inflating: ./bdm_data/orders_df/_committed_1102646042990821830
  inflating: ./bdm_data/orders_df/_started_568900438245366187
  inflating: ./bdm_data/orders_df/_committed_568900438245366187
  inflating: ./bdm_data/orders_df/part-00004-tid-568900438245366187-3cd9d01f
-3962-4d69-b460-dccc76db8c33-33460-1-c000.snappy.parquet
  inflating: ./bdm_data/orders_df/part-00000-tid-568900438245366187-3cd9d01f
-3962-4d69-b460-dccc76db8c33-33466-1-c000.snappy.parquet
  inflating: ./bdm_data/orders_df/part-00003-tid-568900438245366187-3cd9d01f
```

```
# The folder "/FileStore" on DBFS is the default folder for imported data.
# dbutils.fs.mv("file:/databricks/driver/bdm_data/",
"dbfs:/FileStore/bdm_data/", True)
```

```
Out[9]: True
```

1.1 - Import of data

```
sessions_df_schema = StructType([
    StructField("customer_id",StringType(),True),
    StructField(
        "session_events",
        ArrayType(
            StructType([
                StructField("datetime", StringType(),True),
                StructField("event", StringType(),True)
            ]), True)
    ),
    StructField("session_id", StringType() ,True),
    StructField("session_rank", LongType(), True)
])

sessions_df = (
    spark.read.format("json")
    .schema(sessions_df_schema)
    .load("dbfs:/FileStore/bdm_data/sessions_df")
    .select('customer_id', 'session_events', 'session_id')
    .toDF('customer_id', 'session_events', 'session_id')
    .withColumn("session_events", f.explode('session_events'))
    .select(f.col("customer_id"), f.col("session_events.*"),
f.col("session_id"))
    .withColumn('session_timestamp', f.col('datetime').cast('timestamp'))
    .withColumn('session_date', f.to_date(f.col('datetime')))
    .withColumn('session_date_month', f.date_trunc('month',
f.col('datetime')).cast('date'))
    .withColumn('session_hour', f.hour(f.col('datetime')))
)

w_lag   =
Window.partitionBy(f.col("customer_id")).orderBy(f.col("order_timestamp"))
w_month = Window.partitionBy(f.col("customer_id"),
f.col('month')).orderBy(f.col("order_timestamp"))

orders_df =(
    spark.read
    .format("parquet").option("inferSchema", "true")
    # Reading the orders parquet file from DBFS
    .load("dbfs:/FileStore/bdm_data/orders_df/")
    # Renaming columns
    .toDF('order_id', 'session_id', 'order_timestamp', 'customer_id',
'total_value', 'discount_value', 'order_category')
    # Creating a column with paid value info
    .withColumn('paid_value', f.col('total_value') -
f.col('discount_value'))
    # Forcing the time of the purchase to timestamp data type
    .withColumn('order_timestamp',
f.col('order_timestamp').astype('timestamp'))
    # Extracting the hour of the timestamp
```

```
    .withColumn('hour', f.hour(f.col('order_timestamp')))
    # Extracting the month of the timestamp
    .withColumn('month', f.date_trunc('month',
f.col('order_timestamp')).astype('date') )
    # Creating the shift
    .withColumn('shift', f.when(f.col('hour') <= 10,
'breakfast').when(f.col('hour')<=17, 'lunch' ).otherwise('dinner'))
    # Creating the discount range category
    .withColumn('discount_percentage', f.round(f.col('discount_value') /
f.col('total_value'), 2) )
    .withColumn('discount_range',
                f.when(f.col('discount_percentage') <= 0.10, '0-10%')
                .when(f.col('discount_percentage') <= 0.20, '10-20%')
                .otherwise('30%+'))
    .withColumn("order_date_lag", f.lag("order_timestamp", offset=1,
default=None).over(w_lag))
    .withColumn("days_since_last_order",
f.datediff(f.col('order_timestamp'), f.col('order_date_lag')))
    .withColumn("last_order", f.max("order_timestamp").over(w_lag))
    .withColumn("recency", f.datediff(f.current_date(),
f.col('last_order')))
)

cust_df = (
    spark.read
    .format("csv")
    .option("inferSchema", "true")
    .option("header", "true")
    .load("dbfs:/FileStore/bdm_data/cust_df/")
    .select('customer_id', f.col('is_referee').astype('float'),
'device_type', 'install_origin')
)



# prepare the kpis based on orders
# 0000be59-44b8-44a7-85dc-b5c417ba2858

orders_df\
    .where(f.col('customer_id') == '0000be59-44b8-44a7-85dc-b5c417ba2858')\
    .limit(15).display()
```

	order_id	session_id
1	88c02423-ccb6-480a-8f24-743493de6a3b	111c756f-cc33-42f0-8e8f-729a4a5
2	1d485b18-d6f5-4c6e-ac48-886034ef126e	581dfc0f-0abd-47cd-8c59-4a0defa
3	f22c3eac-847c-4a5f-964f-0d293f07a5e3	10ddd31f-8f41-4e4b-b99f-2b6133f
4	f76d1e63-66fc-4a89-a619-61bdc9cdb3ff	74ef009f-a3fe-43ae-8cff-bf43bc29
5	6f043389-f5e8-4b9d-bfdb-cb99623a7833	5756e15a-7798-40a9-873f-ef1d02

6	a19cf547-6c8c-4d37-b0d0-ff1dc0ea554c	e7ac6487-6f1d-473e-9472-af3735
7	fa43e8dc-f34e-4d5e-9cad-f250e6e5c24a	def0d910-eabf-4b32-bf5c-07fd7ae
8	e991eac0-406e-4454-acf7-701e5766c336	061389bc-4a6e-4e02-827d-2660f
9	ab89fc43-d77e-4431-aae9-531cb4e8ca5f	9627927f-064a-4773-b3f6-1bef3f
10	3a3461b5-62ca-4d72-ada5-2b88dde2e89c	c366591f-4d0c-480d-b023-06983
11	c9fb05d6-c7c5-4aa0-8ae3-bdb689bd0c0f	b15c8949-d937-464e-9e5b-97415
12	da35603d-ad2b-451a-9084-dbf9de1e9372	bd1edd22-8e5d-4690-adf9-e2910
13	c268c04b-c84b-4152-9326-b360754b3c13	d83a2f13-e71a-425d-b58a-2f0313
14	733fc42c-5b2b-4d82-9da7-3b3156d2e021	bb395613-bd45-4c12-8d77-689e6

2- Data Preparation

Creation of the variables

From Session_df

- Session_date_month = month of session --> needed for the final dataset
- Session_hour = hour of session
- Open_tmsp = timestamp of beginning of session
- Close_tmsp = timestamp of end of session
- Converted = conversions given by "CallbackPurcahse Event in sessions"
- Cvr = Converstion rate for sessions
- Drop_rate = 1 - CVR
- conversion = absolute value of conversions

From order_df

- paid_value = (total_value) minus (Discount_value)
- Hour = hour of the order
- Month = month of the order
- Shift – (variables breakfast, lunch and dinner)

- Discount percentage = (discount_value) / (total_value)
- Days_since_last_order = time in days from last order
- Avg_elapsed_time = time elapsed between each orders
- Avg_hour = average hour of session
- Frequency = frequency of purchase
- Tot_discount_percentage = tot discount percentage given all expenses
- Tot_gross_value = tot gross values paid
- Avg_expenses = average gross expenses
- Avg_discount = average discount in total values
- Tot_net_paid = total net paid purchase
- Avg_paid_value = averge net paid values
- Avg_discount_percentage = avg disocunt percentage
- Avg_shift = most common shift netween lunch, dinner and
- Last3_months_exp = tot expense for last 3 months
- Last3_month_avg_exp = average expense for last 3 months
- Purchase by category = Pizza, alcoholics, Vegetarian, Japanese, Burger expresseed in percentages

```
# create the expenses by month
w_month = Window.partitionBy(f.col("customer_id"),
f.col('month')).orderBy(f.col("month"))
w_cust =
Window.partitionBy(f.col("customer_id")).orderBy(f.col("customer_id"))
w_cust_cumul =
(Window.partitionBy('customer_id').orderBy('month').rangeBetween(Window.unbo
undedPreceding, 0))

orders_input_1 = (
    orders_df\
        .select(f.col('customer_id'), f.col('total_value'), f.col('month'),
f.col('hour'),
                f.col('discount_value'), f.col('days_since_last_order'),
f.col('recency') )\
        .where(f.col('month') >= '2022-02-01')\
        .groupBy(f.col('customer_id'), f.col('month'))\
        .agg(
            f.avg('days_since_last_order').alias('avg_elapsed_time'),
            f.round(f.avg('hour'),0).alias('avg_hour'),
            f.count(f.col('total_value')).alias('frequency'),
            f.sum('discount_value').alias('tot_discount_percentage'),
            f.sum('total_value').alias('tot_gross_value'),
            f.avg('total_value').alias('avg_expenses'),
            f.avg('discount_value').alias('avg_discount')
        )
        .withColumn('tot_net_paid', f.col('tot_gross_value') -
f.col('tot_discount_percentage'))\
        .withColumn('avg_paid_value', f.col('avg_expenses') -
f.col('avg_discount'))\
        .withColumn('avg_discount_percentage',
f.col('tot_discount_percentage') / f.col('tot_gross_value'))\
        .withColumn('avg_shift', f.when(f.col('avg_hour') <= 10,
'breakfast').when(f.col('avg_hour')<=17, 'lunch' ).otherwise('dinner'))\
        .withColumn('last3_months_exp',
f.sum('tot_net_paid').over(w_cust_cumul))\
        .withColumn('last3_months_avg_exp',
f.avg('avg_paid_value').over(w_cust_cumul))
)
```

```
w_session = Window.partitionBy(f.col("customer_id"),
f.col('session_id')).orderBy(f.col("session_id"))

session_duration = (
        sessions_df
            .filter(f.col('session_date_month')>='2022-02-01')
            .select(f.col('session_id'), f.col('customer_id'),
f.col('session_timestamp'), f.col('session_date_month'))

.withColumn('open_tmsp',f.min(f.col('session_timestamp')).over(w_session))

.withColumn('close_tmsp',f.max(f.col('session_timestamp')).over(w_session))
            .withColumn('duration_sec',f.col("close_tmsp").cast("long") -
f.col('open_tmsp').cast("long"))
            .groupBy('customer_id','session_date_month')
            .agg(f.avg('duration_sec').alias('avg_navig_seconds'))
            .select(f.col('customer_id'),
f.col('session_date_month').alias('month'),
f.col('avg_navig_seconds').alias('duration_sec'))

       )


session_duration.limit(5).display()
```

	customer_id	session_date_month	avg_nav
1	0039e757-e52d-4e39-90e4-b01a11b623b9	2022-04-01	107.1363
2	009a901a-4079-4bde-9f58-833de2edce51	2022-03-01	116.4189
3	009bb52d-62f9-4742-ac4a-21fdf300328b	2022-04-01	114.11111
4	009f5c31-722e-474e-a7e7-6c341ad0ec4f	2022-04-01	118.2868
5	00acd007-767b-427a-a09a-202a27e450d6	2022-03-01	117.6

Showing all 5 rows.

```
session_duration_example = (
        sessions_df
            .filter(f.col('session_date_month')>='2022-02-01')
            .select(f.col('session_id'), f.col('customer_id'),
f.col('session_timestamp'), f.col('session_date_month'))
            .groupBy('customer_id','session_date_month', 'session_id')
            .agg(
                f.min(f.col('session_timestamp')).alias('open_tmsp'),
                f.max(f.col('session_timestamp')).alias('close_tmsp')
            )
            .withColumn('duration_sec',f.col("close_tmsp").cast("long") -
f.col('open_tmsp').cast("long"))
            .groupBy('customer_id','session_date_month')
            .agg(
                    f.avg('duration_sec').alias('avg_navig_seconds')
            )

       )
```

```
session_duration_example.limit(10).display()
```

	customer_id	session_date_month	avg_navi
1	6851891a-c7a0-474e-9155-9f01e7d704a7	2022-04-01	110.625
2	fc4316c6-34b3-44a7-9ed7-b93f5a2a1f44	2022-02-01	116.5384
3	1bd5d942-50b0-448d-abf8-c19d1ed575fc	2022-03-01	110.1379
4	47377728-7024-47f2-8af5-6ff9111b4e74	2022-03-01	115.0555
5	2352ecf9-6193-49d1-9712-54586f40df14	2022-02-01	97.14285
6	22c8d25e-056e-4311-a87f-5872dd33e970	2022-04-01	110

Showing all 10 rows.

```
sessions_input = (
    sessions_df
        .where(f.col('session_date_month') >='2022-02-01')
        .select(f.col('customer_id'), f.col('session_id'),
f.col('session_date_month'))
        .groupBy('customer_id','session_date_month')
        .agg(
            f.countDistinct('session_id').alias('freq_sessions'))
        .select(f.col('customer_id'),
f.col('session_date_month').alias('month'), f.col('freq_sessions'))
)
```

```
orders_input_2 = (
    orders_df\
        .where(f.col('month') >= '2022-02-01')\
        .select(f.col('order_id'), f.col('customer_id'),
f.col('order_category'), f.col('month'), f.col('total_value'),
               f.col('discount_value'))\
        .withColumn('paid_value', f.col('total_value') -
f.col('discount_value'))\
        .groupby(f.col('customer_id'), f.col('month'))\
        .agg(
            f.sum(f.when( (f.col('order_category')=='Pizza'),
f.lit(1)).otherwise(f.lit(0)) ).cast('float') .alias('Pizza'),
            f.sum(f.when( (f.col('order_category')=='Burger'),
f.lit(1)).otherwise(f.lit(0)) ).cast('float') .alias('Burger'),
            f.sum(f.when( (f.col('order_category')=='Alc Beverages'),
f.lit(1)).otherwise(f.lit(0)) ).cast('float') .alias('Alcohol'),
            f.sum(f.when( (f.col('order_category')=='Japanese'),
f.lit(1)).otherwise(f.lit(0)) ).cast('float') .alias('Japanese'),
            f.sum(f.when( (f.col('order_category')=='Vegetarian'),
f.lit(1)).otherwise(f.lit(0)) ).cast('float') .alias('Veggy'),
        )
        .withColumn('tot_values' , f.col('Pizza').cast('float') +
f.col('Burger').cast('float')
                    + f.col('Alcohol').cast('float')  +
f.col('Japanese').cast('float')  + f.col('Veggy').cast('float') )\
        .withColumn('%Pizza', f.round(f.col('Pizza').cast('float')  /
f.col('tot_values'),2))\
        .withColumn('%Burger', f.round(f.col('Burger').cast('float')  /
f.col('tot_values'),2))\
        .withColumn('%Japanese',f.round( f.col('Japanese').cast('float')  /
f.col('tot_values'),2))\
        .withColumn('%Alcohol', f.round(f.col('Alcohol').cast('float')  /
f.col('tot_values'),2))\
        .withColumn('%Veggy', f.round(f.col('Veggy').cast('float')  /
f.col('tot_values'),2))\
)
```

```
orders_input_2.printSchema()
```

```
root
 |-- customer_id: string (nullable = true)
 |-- month: date (nullable = true)
 |-- Pizza: float (nullable = true)
 |-- Burger: float (nullable = true)
 |-- Alcohol: float (nullable = true)
 |-- Japanese: float (nullable = true)
 |-- Veggy: float (nullable = true)
 |-- tot_values: float (nullable = true)
 |-- %Pizza: double (nullable = true)
```

```
|-- %Burger: double (nullable = true)
|-- %Japanese: double (nullable = true)
|-- %Alcohol: double (nullable = true)
|-- %Veggy: double (nullable = true)
```

```
# this table not used
orders_input_3 = (
    orders_df\
        .where(f.col('month') >= '2022-02-01')\
        .select(f.col('order_id'), f.col('customer_id'),
f.col('order_category'), f.col('month'), f.col('total_value'),
               f.col('discount_value'))\
        .withColumn('paid_value', f.col('total_value') -
f.col('discount_value'))\
        .groupby(f.col('customer_id'), f.col('month'))\
        .agg(
            f.coalesce(
               f.sum(f.when( (f.col('order_category')=='Pizza'),
'paid_value').otherwise(0) ),
                f.lit(0)
            ).cast('float').alias('Pizza€'),
            f.coalesce(
                f.sum(f.when( (f.col('order_category')=='Burger'),
'paid_value').otherwise(0) ),
               f.lit(0)
            ).cast('float').alias('Burger€'),
            f.coalesce(
                f.sum(f.when( (f.col('order_category')=='Alc Beverages'),
'paid_value').otherwise(0) ),
                f.lit(0)
            ).cast('float').alias('Alcohol€'),
            f.coalesce(
                f.sum(f.when( (f.col('order_category')=='Japanese'),
'paid_value').otherwise(0) ),
                f.lit(0)
            ).cast('float').alias('Japanese€'),
            f.coalesce(
                f.sum(f.when( (f.col('order_category')=='Vegetarian'),
'paid_value').otherwise(0 )),
                f.lit(0)
            ).cast('float').alias('Veggy€')
         )
        .withColumn('tot_spent' , f.col('Pizza€') + f.col('Burger€') +
f.col('Alcohol€') + f.col('Japanese€') + f.col('Veggy€'))\
        .withColumn('%Pizza€', f.col('Pizza€') / f.col('tot_spent'))\
        .withColumn('%Burger€', f.col('Burger€') / f.col('tot_spent'))\
        .withColumn('%Japanese€', f.col('Japanese€') / f.col('tot_spent'))\
        .withColumn('%Alcohol€', f.col('Alcohol€') / f.col('tot_spent'))\
        .withColumn('%Veggy€', f.col('Veggy€') / f.col('tot_spent'))\
)
```

```
orders_input_2.printSchema()
```

```
root
 |-- customer_id: string (nullable = true)
 |-- month: date (nullable = true)
 |-- Pizza: float (nullable = true)
 |-- Burger: float (nullable = true)
 |-- Alcohol: float (nullable = true)
 |-- Japanese: float (nullable = true)
 |-- Veggy: float (nullable = true)
 |-- tot_values: float (nullable = true)
 |-- %Pizza: double (nullable = true)
 |-- %Burger: double (nullable = true)
 |-- %Japanese: double (nullable = true)
 |-- %Alcohol: double (nullable = true)
 |-- %Veggy: double (nullable = true)
```

```
conversions_input = (
    sessions_df
        .where(f.col('session_date_month') >='2022-02-01')
        .select(f.col('customer_id'), f.col('session_date_month'),
f.col('session_id'), f.col('event'))
        .withColumn('converted', f.when(f.col('event') ==
'CallbackPurchase', f.lit(1)).otherwise(f.lit(0)))
        .groupBy('customer_id','session_date_month')
        .agg(
            f.countDistinct('session_id').alias('freq_sessions'),
            f.sum('converted').alias('conversions'))
        .withColumn('cvr', f.col('conversions') / f.col('freq_sessions'))
        .withColumn('drop_rate', 1 - f.col('cvr'))
        .select(f.col('customer_id'),
f.col('session_date_month').alias('month')
                , f.col('conversions'), f.col('cvr'), f.col('drop_rate'))
    )
```

```
sessions_df.limit(10).display()
```

	customer_id	datetime	e
1	828d7bdf-96eb-4e61-af45-69dcc8ec73be	2022-04-17 13:37:15.957171	O
2	828d7bdf-96eb-4e61-af45-69dcc8ec73be	2022-04-17 13:37:27.126802	V
3	828d7bdf-96eb-4e61-af45-69dcc8ec73be	2022-04-17 13:38:47.358932	V
4	828d7bdf-96eb-4e61-af45-69dcc8ec73be	2022-04-17 13:38:40.752893	C
5	828d7bdf-96eb-4e61-af45-69dcc8ec73be	2022-03-31 18:32:15.958422	O
6	828d7bdf-96eb-4e61-af45-69dcc8ec73be	2022-03-31 18:32:24.135571	V

Showing all 10 rows.

```
sessions_df.select('event').distinct().display()
```

	event
1	ViewSearch
2	ViewList3
3	ClickAddRecommendedItem
4	ViewCartCheckout
5	CallbackPurchase
6	ViewHomeVariant

Showing all 12 rows.

```
# add the filter only for customers who saw item list 3
customers_viewlist3 = (
    sessions_df
        .where(f.col('session_date_month') >= '2022-02-01')\
        .withColumn('view_list3', f.when(f.col('event') == 'ViewList3',
f.lit(1)).otherwise(f.lit(0)))\
        .withColumn('converted', f.when(f.col('event') ==
'CallbackPurchase', f.lit(1)).otherwise(f.lit(0)))\
        .groupBy( f.col('customer_id'), f.col('session_date_month'))\
        .agg(
            f.sum(f.col('view_list3')).alias('view_list3'),
            f.sum(f.col('converted')).alias('bought')
        )
        .where(f.col('view_list3')>0)\
        .drop('view_list3')\
        .withColumn('response', f.when(f.col('bought') > 0,
f.lit(1)).otherwise(0))\
        .select(f.col('customer_id'),
f.col('session_date_month').alias('month'), f.col('bought'),
f.col('response'))
    )
```

```
customers_viewlist3.count()
```

```
Out[83]: 205384
```

```
customers_viewlist3\
    .where(f.col('bought') > 0)\
    .limit(10).display()
```

	customer_id	session_date_month	bought
1	5155bce8-0c8d-419f-85d2-8ed6f01980de	2022-03-01	10

2	736065e5-207d-4875-9f06-1e58fab8dd0e	2022-02-01	1
3	6851891a-c7a0-474e-9155-9f01e7d704a7	2022-04-01	11
4	01cef08b-acc3-4b3e-a7a1-d3f38b35e685	2022-04-01	6
5	cf15a938-856a-4835-b365-276520297daa	2022-03-01	1
6	fe07e930-2cb6-42fa-8907-d5f202365077	2022-03-01	13

Showing all 10 rows.

```
# check the amount of rows for each dataset to understand the results of the join
for i in [conversions_input, orders_input_1, orders_input_2, sessions_input, orders_input_3, customers_viewlist3]:
    print(f'Rows count str(i) is  {i.count()}')
```

```
Rows count str(i) is  281751
Rows count str(i) is  199451
Rows count str(i) is  199451
Rows count str(i) is  281751
Rows count str(i) is  199451
Rows count str(i) is  205384
```

```
type(conversions_input)
```

```
Out[95]: pyspark.sql.dataframe.DataFrame
```

```
# creation of input data
input_data = (
    conversions_input
    .join(orders_input_1, ['customer_id', 'month'], 'inner')
    .join(orders_input_2, ['customer_id', 'month'], 'inner')
    .join(sessions_input, ['customer_id', 'month'], 'inner')
    .join(customers_viewlist3, ['customer_id', 'month'], 'inner')
    .join(cust_df, ['customer_id'], 'inner')
    .join(session_duration, ['customer_id', 'month'], 'inner')
)
```

```
spark.sql(f"CREATE DATABASE IF NOT EXISTS project_data")
#input_data.write.mode('overwrite').option("header", "true").saveAsTable("project_data.input_data")
input_data.write.mode('overwrite').option("overwriteSchema", "true").option("header", "true").saveAsTable("project_data.input_data_v2")
input_df = spark.table("project_data.input_data_v2")
```

2.1 - Check on missing values

```
def count_missings(spark_df,sort=True):
    """
    Counts number of nulls and nans in each column
    """
    df = spark_df.select([f.count(f.when(f.isnan(c) | f.isnull(c), 
c)).alias(c) for (c,c_type) in spark_df.dtypes if c_type not in 
('timestamp', 'string', 'date')]).toPandas()

    if len(df) == 0:
        print("There are no any missing values!")
        return None

    if sort:
        return df.rename(index={0: 
'count'}).T.sort_values("count",ascending=False)

    return df


count_missings(input_df)
```

	count
avg_elapsed_time	3986
conversions	0
Burger	0
is_referee	0
response	0
bought	0
freq_sessions	0
%Veggy	0
%Alcohol	0
%Japanese	0
%Burger	0
%Pizza	0
tot_values	0
Veggy	0
Japanese	0
Alcohol	0
Pizza	0
cvr	0
last3_months_avg_exp	0
last3_months_exp	0
avg_discount_percentage	0
avg_paid_value	0
tot_net_paid	0
avg_discount	0
avg_expenses	0
tot_gross_value	0
tot_discount_percentage	0
frequency	0
avg_hour	0
drop_rate	0
duration_sec	0

```
input_df.limit(100).display()
```

	customer_id	month	conversions

1	0000be59-44b8-44a7-85dc-b5c417ba2858	2022-03-01	7
2	00089da8-33fb-4971-9ffc-92cf9bd159e7	2022-04-01	6
3	00110e08-8611-4fb4-9773-c27fc617bff6	2022-02-01	12
4	00110e08-8611-4fb4-9773-c27fc617bff6	2022-03-01	21
5	0011b87c-15f9-43f0-8f89-bfd38f2ee308	2022-03-01	23
6	0020e9da-26d2-4c6c-afd7-6b43a318eb2a	2022-02-01	3

Showing all 100 rows.

```
input_df.columns
```

```
Out[164]: ['customer_id',
 'month',
 'conversions',
 'cvr',
 'drop_rate',
 'avg_elapsed_time',
 'avg_hour',
 'frequency',
 'tot_discount_percentage',
 'tot_gross_value',
 'avg_expenses',
 'avg_discount',
 'tot_net_paid',
 'avg_paid_value',
 'avg_discount_percentage',
 'avg_shift',
 'last3_months_exp',
 'last3_months_avg_exp',
 'Pizza',
 'Burger',
 'Alcohol',
```

```
input_df\
    .select(f.col('response'), f.col('month'))\
    .groupBy(f.col('month'), f.col('response'))\
    .agg(f.count(f.col('response'))).display()
```

	month	response	count(response)
1	2022-02-01	0	5501
2	2022-02-01	1	38178
3	2022-03-01	0	5714
4	2022-03-01	1	51294
5	2022-04-01	0	5569
6	2022-04-01	1	47915

Showing all 6 rows.

2.2 - Correlation check

```
# create copy of input_data to create a correlation matrix.

corr_df = input_df.alias('corr_df')

corr_df = (
    corr_df
    .withColumn('lunch_shift', f.when(f.col('avg_shift') == 'lunch',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('dinner_shift', f.when(f.col('avg_shift') == 'dinner',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('breakfast_shift', f.when(f.col('avg_shift') == 'breakfast',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('low_end_type', f.when(f.col('device_type') == 'Low-End',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('high_end_type', f.when(f.col('device_type') == 'High-End',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('org_origin', f.when(f.col('install_origin') == 'Organic',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('email_origin', f.when(f.col('install_origin') == 'Email',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('sms_origin', f.when(f.col('install_origin') == 'SMS',
f.lit(1)).otherwise(f.lit(0)))
    .withColumn('meta_origin', f.when(f.col('install_origin') == 'Meta',
f.lit(1)).otherwise(f.lit(0)))
)

corr_df.printSchema()
```

```
root
 |-- customer_id: string (nullable = true)
 |-- month: date (nullable = true)
 |-- conversions: long (nullable = true)
 |-- cvr: double (nullable = true)
 |-- drop_rate: double (nullable = true)
 |-- avg_elapsed_time: double (nullable = true)
 |-- avg_hour: double (nullable = true)
 |-- frequency: long (nullable = true)
 |-- tot_discount_percentage: double (nullable = true)
 |-- tot_gross_value: double (nullable = true)
 |-- avg_expenses: double (nullable = true)
 |-- avg_discount: double (nullable = true)
 |-- tot_net_paid: double (nullable = true)
```

```
 |-- avg_paid_value: double (nullable = true)
 |-- avg_discount_percentage: double (nullable = true)
 |-- avg_shift: string (nullable = true)
 |-- last3_months_exp: double (nullable = true)
 |-- last3_months_avg_exp: double (nullable = true)
 |-- Pizza: float (nullable = true)
```

```
list = ['avg_shift', 'device_type', 'install_origin', 'month',
'customer_id', 'high_end_type']

# delete two columns
corr_df = corr_df.drop(*list)


from pyspark.mllib.stat import Statistics
import pandas as pd

# result can be used w/ seaborn's heatmap
def compute_correlation_matrix(corr_df, method='pearson'):
    # wrapper around
    # https://forums.databricks.com/questions/3092/how-to-calculate-
correlation-matrix-with-all-colum.html
    df_rdd = corr_df.rdd.map(lambda row: row[0:])
    corr_mat = Statistics.corr(df_rdd, method=method)
    corr_mat_df = pd.DataFrame(corr_mat,
                    columns=corr_df.columns,
                    index=corr_df.columns)
    return corr_mat_df


from pyspark.sql import DataFrame

new_df = compute_correlation_matrix(corr_df, method='spearman')
new_df

# add markdowns on correlated (over 70/80, -70/80)
```

	conversions	cvr	drop_rate	avg_elapsed_time	avg_hour	freq
conversions	1.000000	0.711041	-0.711041	-0.762653	-0.205896	0.7
cvr	0.711041	1.000000	-1.000000	-0.518023	-0.222043	0.5
drop_rate	-0.711041	-1.000000	1.000000	0.518023	0.222043	-0.5
avg_elapsed_time	-0.762653	-0.518023	0.518023	1.000000	0.190608	-0.9
avg_hour	-0.205896	-0.222043	0.222043	0.190608	1.000000	-0.1
frequency	0.789520	0.527747	-0.527747	-0.903888	-0.195183	1.0
tot_discount_percentage	0.102541	0.121077	-0.121077	-0.129100	-0.036593	0.1
tot_gross_value	0.755379	0.395497	-0.395497	-0.827404	-0.182374	0.9
avg_expenses	0.420950	-0.001923	0.001923	-0.387261	-0.084452	0.4

avg_discount	0.090701	0.116998	-0.116998	-0.116408	-0.034479	0.1
tot_net_paid	0.755152	0.393771	-0.393771	-0.826898	-0.181832	0.9
avg_paid_value	0.423596	-0.001498	0.001498	-0.390097	-0.084892	0.4
avg_discount_percentage	0.087147	0.118865	-0.118865	-0.113397	-0.034576	0.1
last3_months_exp	0.773158	0.417506	-0.417506	-0.867264	-0.186158	0.8
last3_months_avg_exp	0.421634	0.000718	-0.000718	-0.406393	-0.064441	0.4
Pizza	0.397996	0.159205	-0.159205	-0.434179	-0.066644	0.4
Burger	0.381430	0.552745	-0.552745	-0.435096	-0.159450	0.4
Alcohol	0.133009	-0.072370	0.072370	-0.229210	0.113995	0.2
Japanese	0.372672	0.115902	-0.115902	-0.383924	-0.133965	0.4
Veggy	0.293597	0.119347	-0.119347	-0.354528	0.002164	0.3
tot_values	0.789520	0.527747	-0.527747	-0.903888	-0.195183	1.0
%Pizza	0.320423	0.103718	-0.103718	-0.344644	-0.053219	0.3
%Burger	0.081174	0.397171	-0.397171	-0.081429	-0.099930	0.0
%Japanese	0.324466	0.085267	-0.085267	-0.327955	-0.124920	0.3
%Alcohol	-0.206647	-0.316325	0.316325	0.155115	0.189225	-0.1
%Veggy	0.158094	0.033303	-0.033303	-0.204974	0.033637	0.2
freq_sessions	0.535704	-0.131186	0.131186	-0.448472	-0.051828	0.4
bought	1.000000	0.711041	-0.711041	-0.762653	-0.205896	0.7
response	0.541318	0.539923	-0.539923	-0.394902	-0.116794	0.3
is_referee	-0.139959	-0.198052	0.198052	0.123229	0.100701	-0.1
duration_sec	0.328404	0.483473	-0.483473	-0.284030	-0.117869	0.2
lunch_shift	0.267140	0.233447	-0.233447	-0.263138	-0.853799	0.2
dinner_shift	-0.265812	-0.232619	0.232619	0.261540	0.857686	-0.2
breakfast_shift	-0.022268	-0.014281	0.014281	0.026433	-0.055196	-0.0
low_end_type	-0.237766	-0.407503	0.407503	0.206589	0.190761	-0.2
org_origin	0.154155	0.030152	-0.030152	-0.149625	-0.042965	0.1
email_origin	-0.201295	-0.203271	0.203271	0.186174	0.095040	-0.1
sms_origin	0.108053	0.184843	-0.184843	-0.093911	-0.099683	0.0
meta_origin	-0.076883	-0.040005	0.040005	0.071495	0.057264	-0.0

39 rows × 39 columns

Correlation pairs 90% plus / -90% minus

- CVR vs Drop Rate (1.00) since it is the same information but from the opposite sides
- Conversions vs Bought
- Avg elapsed time (days since last order) vs Frequency (0.92)
- Avg elapsed time (days since last order) vs tot paid net (0.9)
- Frequency vs total gross / total net paid (0.90+)
- Total discount vs avg discount (0.99)
- avg expenses vs avg value paid / last 3 monhs avg expenses (0.99)
- Lunch shift vs dinner shift (0.99) - logically as almost noone except few customers have had breakfast orders the most

Correlation pairs 80-90% / -80 -90%

- avg elapsed time (days since last order) vs total gross / total net paid (0.82)
- avg elapsed time vs last 3 months expenses (0.86)
- Avg hour vs shift (lunch / dinner) (0.86)
- last 3 month expenses vs frequency (0.85)
- last 3 month expenses vs total gross value (0.89)

Correlation pairs 70-80% / -70 -80%

- conversions vs cvr / drop rate (0.71), vs frequency, vs tot gross value, vs tot net paid, vs tot values (0.79)
- cvr vs bought (0.71)
- avg elapsed time vs conversions, vs bought
- frequency vs bought
- total gross values vs avg expenses, avg paid value (0.71)

Final Correlation Comments

As expected, many created variables are derived from others and therefore have a strong linear relationship. If keeping them, we would risk high multicolinearity, hurting the model, computational power and eventually even prediction results. Variables very close to each other, or redundant ones - such as drop out rate (w/ formula as 1 - conversion rate), should be extracted.

Instead of manually chosing all the highly correlated variables to extract from the final dataframe, in the next cells we go with the principal component analysis instead. This should ensure the best features' selection while keeping the most information needed for the modeling. We hope to go with around 10 to 14 PCA components (features).

3 - Pipeline

```
# split train test and prediction

train_ds      = input_df.filter(f.col('month').between('2022-02-01', '2022-
03-31'))
prediction_ds = input_df.filter(f.col('month') >= '2022-04-01');

train_data, test_data = train_ds.randomSplit([0.7, 0.3], 2022)
```

```
from pyspark.ml.feature import Imputer, VectorAssembler, StringIndexer,
OneHotEncoder, MinMaxScaler, PCA
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline


numerical = ['conversions','cvr', 'drop_rate', 'avg_elapsed_time',
'avg_hour', 'frequency',
             'tot_discount_percentage', 'tot_gross_value', 'avg_expenses',
'avg_discount', 'tot_net_paid',
             'avg_paid_value', 'avg_discount_percentage',
'last3_months_exp','last3_months_avg_exp', 'Pizza', 'Burger',
             'Alcohol', 'Japanese', 'Veggy', 'tot_values', '%Pizza',
'%Burger', '%Japanese',
             '%Alcohol', '%Veggy', 'freq_sessions', 'duration_sec','bought']
categorical = ['avg_shift', 'device_type', 'install_origin']

impute = Imputer(inputCols=['avg_elapsed_time', 'duration_sec'], outputCols=
['avg_elapsed_time', 'duration_sec'])
assemble = VectorAssembler(inputCols = numerical,
outputCol='continuous_features')
index = StringIndexer(inputCols=categorical, outputCols=['device_type_idx',
'avg_shift_idx', 'install_origin_idx'])
one_hot = OneHotEncoder(inputCols=['device_type_idx', 'avg_shift_idx',
'install_origin_idx'],
                        outputCols=['device_type_vector','avg_shift_vector',
'install_origin_vector' ])
scale = MinMaxScaler(inputCol='continuous_features',
outputCol='scaled_continuous_features')
final_assemble = VectorAssembler(inputCols=['scaled_continuous_features',
'device_type_vector',
                                            'avg_shift_vector',
'install_origin_vector', 'is_referee'], outputCol='features')
PCA   = PCA(k=7, inputCol= 'features', outputCol= 'pcaFeatures')
lr    = LogisticRegression(featuresCol="pcaFeatures",
                           labelCol="response",
                           predictionCol="prediction")


pipe = Pipeline()
pipe.setStages(
    [
        impute,
        assemble,
        index,
        one_hot,
        scale,
        final_assemble,
        PCA,
        lr
```

```
    ]
)
```

```
Out[28]: Pipeline_a14c3ba95442
```

4 - Modelling

```
pipe_model = pipe.fit(train_ds)
```

```
fitted_data = pipe_model.transform(train_ds)
fitted_data.display()
```

	customer_id	month	conversions
1	0000006a-f50f-4153-aa42-e708216f7d66	2022-02-01	4
2	0010cba0-318f-43e9-94e7-c0638c33c80b	2022-03-01	35
3	00123376-6d20-41c8-a640-1688b4ba74ef	2022-02-01	11
4	001405f6-e3c6-4670-91c5-8d88f37bbd7b	2022-02-01	12

Truncated results, showing first 508 rows.

5 - Model Evaluation and optimization

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(
    labelCol="response",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC",
)

metric = evaluator.evaluate(fitted_data)
print(f"Area under ROC = {metric} ")
```

```
Area under ROC = 0.8760004913354661
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

paramGrid = (
    ParamGridBuilder()
    .addGrid(lr.regParam, [0.5, 0.2, 0.1, 0.01])
    .addGrid(lr.elasticNetParam, [0.0,0.2, 0.25 , 0.5,0.8, 1.0])
    .build()
)
```

```
print(lr.explainParams())
```

```
aggregationDepth: suggested depth for treeAggregate (>= 2). (default: 2)
elasticNetParam: the ElasticNet mixing parameter, in range [0, 1]. For alpha
= 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty. (defa
ult: 0.0)
family: The name of family which is a description of the label distribution
to be used in the model. Supported options: auto, binomial, multinomial (def
ault: auto)
featuresCol: features column name. (default: features, current: pcaFeatures)
fitIntercept: whether to fit an intercept term. (default: True)
labelCol: label column name. (default: label, current: response)
lowerBoundsOnCoefficients: The lower bounds on coefficients if fitting under
bound constrained optimization. The bound matrix must be compatible with the
shape (1, number of features) for binomial regression, or (number of classe
s, number of features) for multinomial regression. (undefined)
lowerBoundsOnIntercepts: The lower bounds on intercepts if fitting under bou
nd constrained optimization. The bounds vector size must beequal with 1 for
binomial regression, or the number oflasses for multinomial regression. (und
efined)
maxBlockSizeInMB: maximum memory in MB for stacking input data into blocks.
Data is stacked within partitions. If more than remaining data size in a par
tition then it is adjusted to the data size. Default 0.0 represents choosing
```

```
!pip install mlflow --quiet
```

```
WARNING: You are using pip version 21.0.1; however, version 22.1.2 is availa
ble.
You should consider upgrading via the '/databricks/python3/bin/python -m pip
install --upgrade pip' command.
```

```
# If you get the error "YOU HAVENT CONFIGURED YOUR CLI", run this:
token =
dbutils.notebook.entry_point.getDbutils().notebook().getContext().apiToken()
.get()
dbutils.fs.put("file:///root/.databrickscfg","
[DEFAULT]\nhost=https://community.cloud.databricks.com\ntoken =
"+token,overwrite=True)
```

```
Wrote 98 bytes.
Out[35]: True
```

```
from pyspark.ml.tuning import CrossValidator
import mlflow
from mlflow import spark

mlflow.pyspark.ml.autolog()

mlflow.start_run()
cv = CrossValidator(
    estimator=pipe,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator,
    numFolds=2
)

cv_model = cv.fit(train_ds)
```

```
2022/06/21 21:11:17 WARNING mlflow.utils: Truncated the value of the key `Ve
ctorAssembler_1.inputCols`. Truncated value: `['conversions', 'cvr', 'drop_r
ate', 'avg_elapsed_time', 'avg_hour', 'frequency', 'tot_discount_percentag
e', 'tot_gross_value', 'avg_expenses', 'avg_discount', 'tot_net_paid', 'avg_
paid_value', 'avg_discount_percentage', 'last3_months_exp', 'last3_...`
2022/06/21 21:11:18 WARNING mlflow.utils: Truncated the value of the key `Ve
ctorAssembler_1.inputCols`. Truncated value: `['conversions', 'cvr', 'drop_r
ate', 'avg_elapsed_time', 'avg_hour', 'frequency', 'tot_discount_percentag
e', 'tot_gross_value', 'avg_expenses', 'avg_discount', 'tot_net_paid', 'avg_
paid_value', 'avg_discount_percentage', 'last3_months_exp', 'last3_...`
2022/06/21 21:11:19 WARNING mlflow.utils: Truncated the value of the key `Ve
ctorAssembler_1.inputCols`. Truncated value: `['conversions', 'cvr', 'drop_r
ate', 'avg_elapsed_time', 'avg_hour', 'frequency', 'tot_discount_percentag
e', 'tot_gross_value', 'avg_expenses', 'avg_discount', 'tot_net_paid', 'avg_
paid_value', 'avg_discount_percentage', 'last3_months_exp', 'last3_...`
```

```
2022/06/21 21:11:19 WARNING mlflow.utils: Truncated the value of the key `Ve
ctorAssembler_1.inputCols`. Truncated value: `['conversions', 'cvr', 'drop_r
ate', 'avg_elapsed_time', 'avg_hour', 'frequency', 'tot_discount_percentag
e', 'tot_gross_value', 'avg_expenses', 'avg_discount', 'tot_net_paid', 'avg_
paid_value', 'avg_discount_percentage', 'last3_months_exp', 'last3_...`
```

```
mlflow.spark.log_model(cv_model.bestModel, "model-file")# logs model as
artifacts
mlflow.end_run()
```

```
print(cv_model.avgMetrics)
```

```
[0.8743949119663307, 0.5, 0.5, 0.5, 0.5, 0.5, 0.8748873233763264, 0.83423854
04436317, 0.813335367113174, 0.5, 0.5, 0.5, 0.8757202788306094, 0.8606856021
326458, 0.8562261458647044, 0.8132700513264455, 0.7524681727164235, 0.5, 0.8
802088323193349, 0.8796882334541194, 0.8795441171846625, 0.8786058298598327,
0.8770023415088206, 0.8755978395362136]
```

```
best_model = cv_model.bestModel
```

```
fitted_test_data = best_model.transform(test_data)
```

```
train_metric = evaluator.evaluate(fitted_data)
test_metric = evaluator.evaluate(fitted_test_data)

print(f"Area under ROC on TRAIN= {train_metric}")
print(f"Area under ROC on TEST= {test_metric}")
```

```
Area under ROC on TRAIN= 0.8760015462188199
Area under ROC on TEST= 0.875340577791595
```

6 - Evaluation of model on April dataset and result extraction

```
# get the predictions for the test dataset with customers in april

fitted_prediction_data = best_model.transform(prediction_ds)
predict_metric = evaluator.evaluate(fitted_prediction_data)
```

```
print(f"Area under ROC on TRAIN= {predict_metric}")
```

```
Area under ROC on TRAIN= 0.8682853627998809
```

```
fitted_prediction_data.limit(100).display()
```

	customer_id	month	conversions
1	0007ae88-2bc4-4a19-a8d5-88f7c438508f	2022-04-01	1
2	001b3652-478d-4e35-aca8-69dfbe4e877e	2022-04-01	3
3	001eb828-0a75-40e6-84a4-ee6de0f5435a	2022-04-01	0
4	002018a8-f8ca-4749-9bc3-fe6697658a3b	2022-04-01	13

Showing all 100 rows.

```
fitted_prediction_data\
    .groupBy(f.col('prediction'))\
    .agg(
        f.sum(f.col('prediction')).alias('converted'),
        f.count(f.col('prediction')).alias('total')
        ).display()
```

	prediction	converted	total
1	0	0	1437
2	1	52047	52047

Showing all 2 rows.

```
fitted_prediction_data\
    .groupBy(f.col('response'))\
    .agg(
        f.sum(f.col('response')).alias('converted'),
        f.count(f.col('response')).alias('total')
        ).display()
```

	response	converted	total
1	1	47915	47915

2	0	0	5569

Showing all 2 rows.

```
from pyspark.ml.functions import vector_to_array

results_df = (
    fitted_prediction_data\
        .select(f.col('customer_id'), f.col('probability'),
f.col('prediction'))\
        .withColumn("xs", vector_to_array("probability"))\
        .withColumn('probability_0' , f.col('xs')[0])\
        .withColumn('probability_1' , f.col('xs')[1])\
        .select(f.col('customer_id'), f.col('probability_1'),
f.col('prediction').alias('is_target'))
)
```

7 - Model performance evaluation

By comparing the outcome of the A/B test we see that in 47.8K cases we have conversions and on 5.6K we don't have.
While with the model application, we can see that actually 52K customer could convert with probability over 50% and only 1.4K would not convert.

```
print(f'We could say then the the overall uplift from the model application
is {52047 / 47915}')
```

```
We could say then the the overall uplift from the model application is 1.086
2360429927997
```

8 - Final comment

Overall the model improved the outcome of the conversions given the data that we have. COmpared to the random split of baseline, it performs better. We still think that further improvement can be done using other features, which would make the model more accurate. Another check could be done on F1 score and Recall, for a second iteration, as well as trying with other models, like Random Forest and Neural Network.