bilgin_sherifov__customer_value_challenge

August 8, 2019

0.0.1 Submition for

0.1 *****: Data Challenge

by Bilgin Sherifov Date: ****--

Packages to install before running the notebook (with these you could run the cells till the "clustering2 session. For more advanced sessions you will need more packages):

1. Python 3.x
2. Pandas
3. Numpy
4. Matplotlib
5. Seaborn
6. Contextlib
7. xlrd

(python -m pip install --user numpy matplotlib pandas seaborn contextlib xlrd)

0.2 Note on running the notebook:

First, the notebook should be ran in the ~/code folder. Then, one should extract the tables in the data folder.

```
In [1]: # import numpy and pandas
        import numpy as np
        import pandas as pd

        # import plotting packages
        import matplotlib.pyplot as plt
        import matplotlib.dates as mdates
        import seaborn as sns; sns.set();
        %matplotlib inline

        # time-related libraries
        import time as tm
        import datetime as dt
        from datetime import timedelta
```

```
# system and OS libraries
import os
from os.path import join as pj
import copy
import pickle
import gc; gc.collect()

#MSCL
from contextlib import contextmanager
import warnings
import itertools

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:95% !important; }</style>"))

sns.set_context("talk") # paper, notebook, talk, poster

USE_DARK_THEME = False
```

```
<IPython.core.display.HTML object>
```

1 SOME CORE FUNCTIONS / CLASSES

```
In [2]: @contextmanager
        def timer(title, n_blanks = 1):
            print(n_blanks * ' ' + 'STARTING: {}'.format(title))
            t0 = tm.time()
            yield
            print(n_blanks * ' ' + "FINISHED in {:.2f} min. \n".format((tm.time() - t0)/60))
```

```
In [3]: def to_datetime(df, date_columns):
            for date_column in date_columns:
                with timer(date_column, n_blanks=4):
                    df[date_column] = pd.to_datetime(df[date_column], dayfirst=True, format='%
            
            return df
```

2 WORKSPACE

2.1 Set paths and parameters

```
In [4]: if False:
            PATHS = {'root' : os.getcwd()}
            
            for new_path in ['data', 'results', 'WORKSPACE', 'info']:
                os.mkdir(pj(PATHS['root'], new_path))
```

```
            PATHS[new_path] = pj(PATHS['root'], new_path)
    else:
        #os.chdir("..")
        os.chdir(os.path.dirname(os.getcwd()))
        PATHS = {'root' : os.getcwd()}
        for new_path in ['data', 'results', 'WORKSPACE', 'info', 'code']:
            PATHS[new_path] = pj(PATHS['root'], new_path)
    [print(key,':', val) for key, val in PATHS.items()];
```

```
data : E:\JupyterNB\Job Applications\Shop Direct\data
code : E:\JupyterNB\Job Applications\Shop Direct\code
results : E:\JupyterNB\Job Applications\Shop Direct\results
info : E:\JupyterNB\Job Applications\Shop Direct\info
root : E:\JupyterNB\Job Applications\Shop Direct
WORKSPACE : E:\JupyterNB\Job Applications\Shop Direct\WORKSPACE
```

2.2 PLAN:

I'll define a customer as valuable if he/she is unlikely to default and also generates a lot of revenue. Therefore, a complete system should focus on both of these. However, due to time limitations, for this submission I'll focus only on the first part, i.e. wheteher a customer is likely to default or not.

Assessing the robability of deafault is basically a classification problem, while estimating the revenue that will be generated is a regression problem. Thus, in this submission I'll be focusing on classification. While there are many classifiers one can use, I'll use LightGBM, since it is fast, accurate, agnostic to data types (categorical or numeric), and has proven to be quite popular recently.

Before doing classification, though, I'll do basic data exploration with the purpose of selecting most relevant features. For this I'll use the fact that according to Bayes' theorem, the probability of defaulting or not given a set of features is proportional to the joint likelyhood of observing these features given that a customer has defaulted or not, multiplied by the prior probability of defaulting or not (and normalised by the joint probility of observing the features).

It is difficult to calculate and visualize joint probabilities of features (conditional on default or not) of multidimensional data. Therefore, I'll take one feature at a time, akin Naive Bayes. If two features taken independently exhibit significantly different probbilty distributions for defaulted and non-defaulted visitor, one might safely assume that the joint probability of the two features will be even more distinct for the two classes of users.

I'll also use cluster analysis to asses whether I could seperate customers into two distinct subsets on the basis of the features selected and I'll compare how the clusters overlap with whether a customer has defaulted or not. This, ths will be one way of reassuring myself that I have selected a good set of features.

Thus, here is the plan:

1. Load the data and look at some basic things, like row and columns counts and data types etc
2. Compare visually probability distributions of various fetures, conditioned on whether the user defaulted or not.
3. Asses the quality of the features using some cluster analysis

4. Build a two-stage classification system, whereby first I build a classifier to detect customers that have very low probability of defaulting and filtering them out. Then, on the remaining set of data, which now should be much more class-balanced, I build a second classifier tht is tuned to better detect customers that are likely to deault.
5. I also plot the features that are most important in the two stages of classification.

2.3 Quick exploration of the provided data

2.3.1 Read the tables and their legends and collect them into a dictionary

```
In [6]: with timer('Reading original data files'):
            dataset = {}
            for root, directories, files in os.walk(PATHS['data']):
                for file in files:
                    if '.csv' in file:
                        try:
                            fileType = file.split('.')[0]
                            dataset[file.split('.')[0]] = {
                                'data': pd.read_csv(pj(PATHS['data'], file)),
                                'legend': pd.read_excel(pj(PATHS['info'], 'Data_Dictionary.xls
                            }
                            print(4*' ' + 'Read:', file)
                        except:
                            print(4*' ' + "Couldn't read:", file)

        print(dataset.keys())
```

```
 STARTING: Reading original data files
    Read: Calendar.csv
    Read: Customer.csv
    Read: Order.csv
 FINISHED in 0.06 min.

dict_keys(['Customer', 'Calendar', 'Order'])
```

2.3.2 Print table stats etc

```
In [7]: def table_stats(df, tableName):
            columns = list(df.columns)
            print('TABLE', tableName.upper())
            print(100 * '-')
            print('{0:<40s}: {1:,d}'.format('Number of rows', len(df)))
            print('{0:<40s}: {1:,d}'.format('Number of columns', len(columns)))
            print('\n{0:40s}  {1:<30s}\n'.format('Column name', 'Number of unique values (data
            for column in columns:
                print('{0:>40s}: {1:,d} ({2:})'.format(column, len(df[column].unique()), df[co
            print(100 * '-')
```

Customers

Show table stats and a few rows

```
In [8]: tableName = 'Customer'
        table_stats(dataset[tableName]['data'], tableName)
        dataset[tableName]['data'].head(5)
```

```
TABLE CUSTOMER
------------------------------------------------------------------------------------------------
Number of rows                          : 509,693
Number of columns                       : 15

Column name                               Number of unique values (data type)

                             identifier: 509,693 (int64)
                             title_desc: 11 (object)
                        principal_brand: 2 (object)
                             birth_year: 96 (float64)
                            credit_band: 25 (object)
                            status_code: 3 (object)
                       postcode_outward: 2,813 (object)
                             date_start: 940 (object)
                           date_default: 194 (object)
                         date_completed: 1,002 (object)
                      date_last_payment: 1,158 (object)
                         open_to_buy_amt: 4,274 (float64)
                    LTIME_NET_SALES_AMT: 101,243 (float64)
                        LTIME_NO_ORDERS: 167 (int64)
                     LTIME_RETURNED_AMT: 23,665 (float64)
------------------------------------------------------------------------------------------------
```

```
Out[8]:    identifier title_desc principal_brand  birth_year credit_band status_code  \
        0       37295       MISS             LAI      1972.0          Y1      active
        1      315441        MRS             LEX      1966.0          Y1   completed
        2      489894       MISS             LEX      1996.0          Y3      active
        3      419603         MR             LEX      1946.0          Y1      active
        4      444889       MISS             LEX      1968.0          U1      active

          postcode_outward date_start date_default date_completed date_last_payment  \
        0             NR30  19OCT2016          NaN            NaN         09MAR2017
        1              MK5  08APR2016          NaN      21JAN2017         22APR2016
        2             B 71  27AUG2016          NaN            NaN         13MAR2017
        3             DE73  13DEC2016          NaN            NaN         14DEC2016
        4             EX10  20JUN2014          NaN            NaN         02FEB2017

           open_to_buy_amt  LTIME_NET_SALES_AMT  LTIME_NO_ORDERS  LTIME_RETURNED_AMT
```

```
0         78.0         604.42          5          0.0
1       3000.0          34.39          1          0.0
2          4.0         512.94          5          0.0
3       1500.0          25.98          1          0.0
4          0.0         424.94          2          0.0
```

Show table legend

```
In [9]: dataset[tableName]['legend']
```

```
Out[9]:              FIELD NAME     DATA TYPE              EXAMPLE  \
        0            identifier       integer                37295
        1            title_desc     character                 MISS
        2       principal_brand     character                  LAI
        3            birth_year       integer                 1972
        4           credit_band     character                   Y1
        5           status_code     character               active
        6      postcode_outward     character                 NR30
        7            date_start          date  2016-10-19 00:00:00
        8          date_default          date                  NaN
        9        date_completed          date                  NaN
        10    date_last_payment          date  2017-03-09 00:00:00
        11      open_to_buy_amt  decimal(8,2)                   78
        12  LTIME_NET_SALES_AMT  decimal(9,2)               604.42
        13       LTIME_NO_ORDERS       integer                    5
        14   LTIME_RETURNED_AMT  decimal(9,2)                    0

                                                  DESCRIPTION
        0                               Individual identifier
        1        Customer's title, such as Mr, Ms, Mrs, Dr etc.
        2   Brand of customers account - usually matches t...
        3                                Year customer was born
        4   Code that identifies the level of credit risk ...
        5   status of account which can be - active - cust...
        6                       Outward for customers address
        7                       Date customer started trading
        8   Date customer's account defaulted - blank if n...
        9   Date customer's account was completed - blank ...
        10  Date customer last made a payment on their acc...
        11  Amount of credit still available on customer's...
        12                     Total Net Sales on the account
        13          Total number of orders made on the account
        14         Total Value of returned items on the account
```

Orders

Show table stats and a few rows

```
In [10]: tableName = 'Order'
         table_stats(dataset[tableName]['data'], tableName)
         dataset[tableName]['data'].head(5)
```

```
TABLE ORDER
---------------------------------------------------------------------------------------------------
Number of rows                          : 3,613,185
Number of columns                       : 10

Column name                               Number of unique values (data type)

                              identifier: 376,746 (int64)
                                   Brand: 2 (object)
                       Account_Year_Week: 157 (int64)
                             Week_Ending: 157 (object)
                                 Channel: 2 (object)
                online_device_type_detail: 4 (object)
                           Account_Type2: 2 (object)
                        Gross_Demand_Pre: 9,305 (float64)
                                New_Cust: 2 (object)
                            Product_dept: 11 (object)
---------------------------------------------------------------------------------------------------
```

```
Out[10]:     identifier Brand  Account_Year_Week Week_Ending Channel  \
         0       37295   LAI             201652   23DEC2016  Online
         1      315441   LEX             201615   08APR2016  Online
         2      489894   LEX             201649   02DEC2016  Online
         3      419603   LEX             201651   16DEC2016  Online
         4      431206   LAI             201547   20NOV2015  Online

           online_device_type_detail Account_Type2  Gross_Demand_Pre New_Cust  \
         0                    TABLET        Credit             27.00        Y
         1                   DESKTOP        Credit             30.40        Y
         2                    MOBILE        Credit            129.00        Y
         3                   DESKTOP        Credit             21.99        Y
         4                    MOBILE          Cash             29.75        Y

           Product_dept
         0       Dept G
         1       Dept A
         2       Dept B
         3       Dept C
         4       Dept D
```

Show table legend

```
In [11]: dataset[tableName]['legend']
```

```
Out[11]:                  FIELD NAME     DATA TYPE  EXAMPLE  \
         0                identifier       integer    37295
         1                     Brand     character      LAI
         2         Account_Year_Week       integer   201652
         3               Week_Ending       integer  1161223
         4                   Channel     character   Online
         5  online_device_type_detail     character   TABLET
         6             Account_Type2     character   Credit
         7    Gross_Demand_Pre_Credit  decimal(9.2)       27
         8                  New_Cust     character        Y
         9              Product_dept     character  DEPT A

                                                  DESCRIPTION
         0                              Individual identifier
         1  Identifies the Brand, LEX = VERY. LAI = Little...
         2                                      Week and Year
         3      Date on last day of the week, i.e. the Friday
         4         Either online purchase or offline purchase
         5                            Device used for purchase
         6  Whether the account a cash account or a credit...
         7                          Price customer pays for order
         8             Y = new customer, N = existing customer
         9                                     Department name
```

Calendar

Show table stats and a few rows

```
In [12]: tableName = 'Calendar'
         table_stats(dataset[tableName]['data'], tableName)
         dataset[tableName]['data'].head(5)
```

```
TABLE CALENDAR
----------------------------------------------------------------------------------------------
Number of rows                          : 1,099
Number of columns                       : 20

Column name                               Number of unique values (data type)

                         ACCOUNT_PERIOD: 12 (int64)
                    ACCOUNT_PERIOD_WEEK: 6 (int64)
                           ACCOUNT_WEEK: 53 (int64)
                    ACCOUNT_YEAR_PERIOD: 36 (object)
             ACCOUNT_YEAR_PERIOD_REL_NO: 36 (int64)
                      ACCOUNT_YEAR_WEEK: 157 (object)
               ACCOUNT_YEAR_WEEK_REL_NO: 157 (int64)
                           ACCOUNT_YEAR: 3 (object)
                    ACCOUNT_YEAR_REL_NO: 3 (int64)
```

```
            CAL_DATE: 1,099 (object)
     CAL_DATE_REL_NO: 1,099 (object)
          CAL_DAY_ID: 7 (int64)
            CAL_WEEK: 53 (int64)
          CAL_PERIOD: 13 (int64)
     CAL_PERIOD_WEEK: 5 (int64)
      CAL_YEAR_MONTH: 37 (object)
              DAY_ID: 6 (int64)
              SEASON: 6 (object)
       SEASON_REL_NO: 6 (int64)
         SEASON_WEEK: 27 (int64)
--------------------------------------------------------------------------------------------
```

Out[12]:

```
   ACCOUNT_PERIOD  ACCOUNT_PERIOD_WEEK  ACCOUNT_WEEK ACCOUNT_YEAR_PERIOD  \
0               2                    4             8             201,502
1              11                    2            46             201,611
2               5                    2            19             201,605
3               8                    4            34             201,408
4               1                    1             1             201,501

   ACCOUNT_YEAR_PERIOD_REL_NO ACCOUNT_YEAR_WEEK  ACCOUNT_YEAR_WEEK_REL_NO  \
0                         -25           201,508                      -108
1                          -4           201,646                       -18
2                         -10           201,619                       -45
3                         -31           201,434                      -134
4                         -26           201,501                      -115

  ACCOUNT_YEAR  ACCOUNT_YEAR_REL_NO    CAL_DATE CAL_DATE_REL_NO  CAL_DAY_ID  \
0        2,015                   -2  18/02/2015            -758           4
1        2,016                   -1  10/11/2016            -127           5
2        2,016                   -1  30/04/2016            -321           0
3        2,014                   -3  21/08/2014            -939           5
4        2,015                   -2  31/12/2014            -807           4

   CAL_WEEK  CAL_PERIOD  CAL_PERIOD_WEEK CAL_YEAR_MONTH  DAY_ID  SEASON  \
0         8           2                4        201,502       4  SS2015
1        45          12                1        201,611       5  AW2016
2        17           5                1        201,604       1  SS2016
3        34           9                2        201,408       5  AW2014
4         1           1                1        201,412       4  SS2015

   SEASON_REL_NO  SEASON_WEEK
0             -4            8
1             -1           19
2             -2           19
3             -5            8
4             -4            1
```

Show table legend

```
In [13]: dataset[tableName]['legend']
```

```
Out[13]:                     FIELD NAME     DATA TYPE              EXAMPLE  \
         0               ACCOUNT_PERIOD       byteint                    2
         1          ACCOUNT_PERIOD_WEEK       byteint                    4
         2                 ACCOUNT_WEEK       byteint                    8
         3          ACCOUNT_YEAR_PERIOD       integer               201502
         4   ACCOUNT_YEAR_PERIOD_REL_NO       integer                  -25
         5            ACCOUNT_YEAR_WEEK       integer               201508
         6     ACCOUNT_YEAR_WEEK_REL_NO       integer                 -108
         7                 ACCOUNT_YEAR      smallint                 2015
         8          ACCOUNT_YEAR_REL_NO       integer                   -2
         9                     CAL_DATE          date  2015-02-18 00:00:00
         10             CAL_DATE_REL_NO       integer                 -758
         11                  CAL_DAY_ID       byteint                    4
         12                    CAL_WEEK       byteint                    8
         13                  CAL_PERIOD       byteint                    2
         14             CAL_PERIOD_WEEK       byteint                    4
         15              CAL_YEAR_MONTH       integer               201502
         16                      DAY_ID       byteint                    4
         17                      SEASON  character(6)               SS2015
         18               SEASON_REL_NO       integer                   -4
         19                 SEASON_WEEK       byteint                    8

                                                   DESCRIPTION
         0           Accounting period. Values from 1 to 12.
         1         Accounting period week. Values from 1 to 6.
         2              Accounting week. Values from 1 to 53.
         3   Accounting year period. Format yyy,ypp where y...
         4   Accounting year period relative number.  It re...
         5   Accounting year week. Format yyy,yww where yyy...
         6   Accounting year week relative number. It repre...
         7                                    Accounting year.
         8   Accounting year relative number. It represents...
         9                                      Calendar date.
         10  Calendar date relative number. Number that rep...
         11  Calendar day ID. From 0 to 6: (Sat 0, Sun 1, M...
         12  Calendar week. One calendar year has up to 53 ...
         13             Calendar period. Values from 1 to 53.
         14              Calendar period week. From 1 up to 5
         15             Calendar year & month. Format yyy,ypp
         16  Calendar day ID. From 1 to 6: (Sat 1, Sun 1, M...
         17  Season name. Format: 'SSyyyy' or 'AWyyyy' wher...
         18  Season relative number. It represents the numb...
         19                 Week number within each season.
```

2.4 A little more indepth exploration of the provided data

2.4.1 Customers table alone

Data augmenttion and transformation At this stage, I'll look at the probability distributions of certain fields as a function of whether the customer has defaulted or not and see if they are different for each customer type.

1. First, I'll append a column, dafaulted, which will be a flag for whether customer deafaulted or not (True if defaulted, False otherwise)
2. Next, I'll convert all date-related columns to datetime data format
3. Next, I'll add a few more date-related columns

In [14]:
```
with timer('Appending a column that is a flag for whether customer deafaulted or not')
    # make a copy of the table
    dfCustomer = copy.deepcopy(dataset['Customer']['data'])
    # ad a columns that indictes defaulted customer or not and set all to False at fi
    dfCustomer['defaulted'] = False
    # get row index of defaulted customers
    idxDefaulted = dfCustomer[dfCustomer['date_default'].notnull()].index
    # set the defaulted flag to True for those rows
    dfCustomer.loc[idxDefaulted, 'defaulted'] = True

with timer('Converting date columns to datetime'):
    dateColumns = ['date_start', 'date_default', 'date_completed', 'date_last_payment
    dfCustomer = to_datetime(dfCustomer, dateColumns)

with timer('Appending date-related columns'):
    # get year and month of start date
    dfCustomer['year_start'] = dfCustomer['date_start'].dt.year
    dfCustomer['month_start'] = dfCustomer['date_start'].dt.month
    # get year and month of last payment date
    dfCustomer['year_last_payment'] = dfCustomer['date_last_payment'].dt.year
    dfCustomer['month_last_payment'] = dfCustomer['date_last_payment'].dt.month
    # calculate time in days between last payment date and account creation date
    dfCustomer['days_between_last_payment_and_start'] = (dfCustomer['date_last_paymen
    # set identifier as a row index
    dfCustomer = dfCustomer.set_index('identifier', drop=False)


dfCustomer.head(5)
```

```
STARTING: Appending a column that is a flag for whether customer deafaulted or not
FINISHED in 0.00 min.

STARTING: Converting date columns to datetime
   STARTING: date_start
   FINISHED in 0.02 min.
```

```
    STARTING: date_default
    FINISHED in 0.00 min.

    STARTING: date_completed
    FINISHED in 0.01 min.

    STARTING: date_last_payment
    FINISHED in 0.01 min.

 FINISHED in 0.04 min.

 STARTING: Appending date-related columns
 FINISHED in 0.00 min.
```

Out[14]:

```
            identifier title_desc principal_brand  birth_year credit_band  \
identifier
37295            37295       MISS             LAI      1972.0          Y1
315441          315441        MRS             LEX      1966.0          Y1
489894          489894       MISS             LEX      1996.0          Y3
419603          419603         MR             LEX      1946.0          Y1
444889          444889       MISS             LEX      1968.0          U1

           status_code postcode_outward date_start date_default  \
identifier
37295           active             NR30 2016-10-19          NaT
315441       completed              MK5 2016-04-08          NaT
489894          active             B 71 2016-08-27          NaT
419603          active             DE73 2016-12-13          NaT
444889          active             EX10 2014-06-20          NaT

           date_completed                 ...                     \
identifier                                ...
37295                 NaT                 ...
315441         2017-01-21                 ...
489894                NaT                 ...
419603                NaT                 ...
444889                NaT                 ...

           open_to_buy_amt  LTIME_NET_SALES_AMT  LTIME_NO_ORDERS  \
identifier
37295                 78.0               604.42                5
315441              3000.0                34.39                1
489894                 4.0               512.94                5
419603              1500.0                25.98                1
444889                 0.0               424.94                2
```

```
           LTIME_RETURNED_AMT  defaulted  year_start  month_start  \
identifier
37295                     0.0      False        2016           10
315441                    0.0      False        2016            4
489894                    0.0      False        2016            8
419603                    0.0      False        2016           12
444889                    0.0      False        2014            6

           year_last_payment  month_last_payment  \
identifier
37295                 2017.0                 3.0
315441                2016.0                 4.0
489894                2017.0                 3.0
419603                2016.0                12.0
444889                2017.0                 2.0

           days_between_last_payment_and_start
identifier
37295                                    141.0
315441                                    14.0
489894                                   198.0
419603                                     1.0
444889                                   958.0

[5 rows x 21 columns]
```

Plot distributions for numeric features

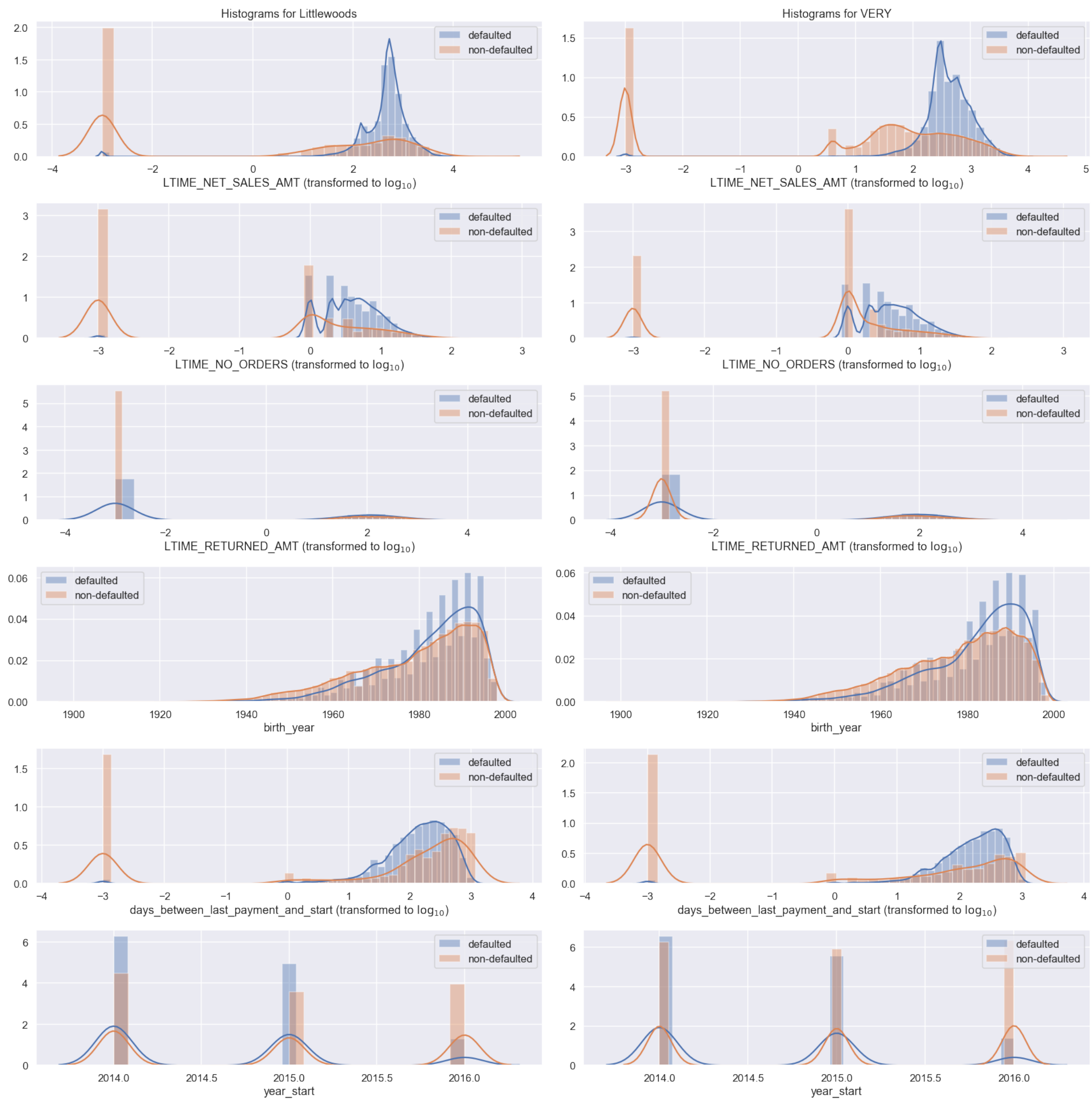
```
In [15]: def plot_prob_numeric_feature_given_defaulted(data, features, log_transformed_fields =
             n_rows = len(features)
             plt.close('all')
             plt.figure(figsize=(24, n_rows * 4))
             for j, feature in enumerate(features):
                 plt.subplot(n_rows, 2, j*2+1)
                 df = copy.deepcopy(data[data['principal_brand'] == 'LAI'])
                 if feature in log_transformed_fields:
                     # get rwo index of non-negaive values only, because I'll be taking the lo
                     idx_non_negative = df[df[feature] >= 0].index
                     df = df.loc[idx_non_negative]
                     series_defaulted = (df[df['defaulted']][feature]+.001).apply(np.log10)
                     series_non_defaulted = (df[~df['defaulted']][feature]+.001).apply(np.log1
                 else:
                     idx_non_null = df[df[feature].notnull()].index
                     df = df.loc[idx_non_null]
                     series_defaulted = df[df['defaulted']][feature]
                     series_non_defaulted = df[~df['defaulted']][feature]

                 # plot normed histogram of log_value for defaulted customers
```

```
    sns.distplot(
        series_defaulted,
        hist = True,
        label='defaulted')
    # plot normed histogram of log_value for non-defaulted customers
    sns.distplot(
        series_non_defaulted,
        hist = True,
        label='non-defaulted')

    if feature in log_transformed_fields:
        plt.xlabel(feature + ' (transformed to $\log_{10}$)')
    plt.legend()
    if j == 0:
        plt.title('Histograms for Littlewoods')


    plt.subplot(n_rows, 2, j*2+2)
    df = copy.deepcopy(data[data['principal_brand'] == 'LEX'])
    if feature in log_transformed_fields:
        # get rwo index of non-negaive values only, because I'll be taking the lo
        idx_non_negative = df[df[feature] >= 0].index
        df = df.loc[idx_non_negative]
        series_defaulted = (df[df['defaulted']][feature]+.001).apply(np.log10)
        series_non_defaulted = (df[~df['defaulted']][feature]+.001).apply(np.log1
    else:
        idx_non_null = df[df[feature].notnull()].index
        df = df.loc[idx_non_null]
        series_defaulted = df[df['defaulted']][feature]
        series_non_defaulted = df[~df['defaulted']][feature]

    # plot normed histogram of log_value for defaulted customers
    sns.distplot(
        series_defaulted,
        hist = True,
        label='defaulted')
    # plot normed histogram of log_value for non-defaulted customers
    sns.distplot(
        series_non_defaulted,
        hist = True,
        label='non-defaulted')

    if feature in log_transformed_fields:
        plt.xlabel(feature + ' (transformed to $\log_{10}$)')
    plt.legend()
    if j == 0:
        plt.title('Histograms for VERY')
```

```
        plt.tight_layout()
```

```
In [16]: plot_prob_numeric_feature_given_defaulted(
             dfCustomer,
             [
                 'LTIME_NET_SALES_AMT',
                 'LTIME_NO_ORDERS',
                 'LTIME_RETURNED_AMT',
                 'birth_year',
                 'days_between_last_payment_and_start',
                 'year_start'
             ],
             log_transformed_fields = [
                 'LTIME_NET_SALES_AMT',
                 'LTIME_NO_ORDERS',
                 'LTIME_RETURNED_AMT',
                 'days_between_last_payment_and_start'
             ]
         )
```

```
C:\Users\bilgin\Anaconda3\envs\ml_adv_py35\lib\site-packages\scipy\stats\stats.py:1713: FutureW
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Observations:

- Total Net Sales: In both brand accounts, there is a difference in the distribution of Total Net Sales for defaulted and non-defaulted users. For defaulted users this distribution peaks at higher values and is more concentrated.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
  The difference in distributions between defaulted and non-defaulted users seems to be higher for "VERY"
- Total number of orders: In both brand accounts, there is a difference in the distribution of Total number of orders for defaulted and non-defaulted users. For defaulted users this distribution is skewed towards higher values.
  Thus, this feature should be included in any model that tries to predict the chance of a user

defaulting.
There seems to be no visible difference in distributions between defaulted and non-defaulted users accross brands.

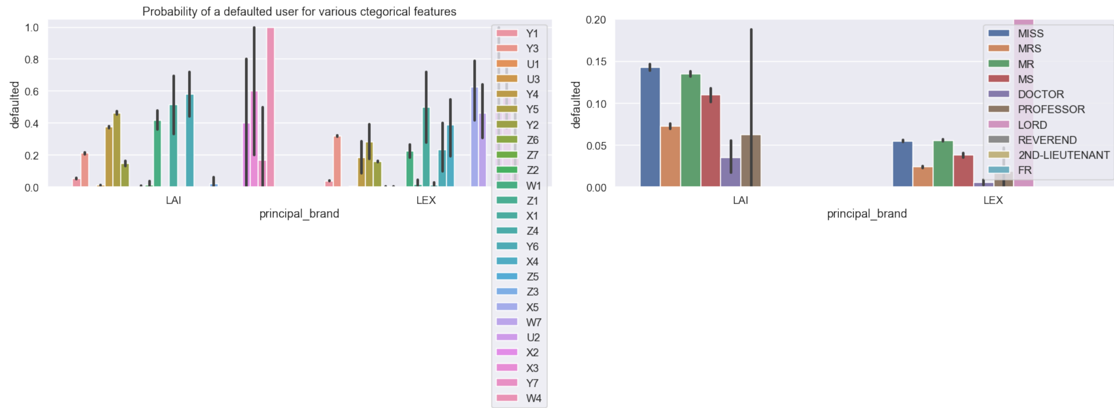
- Total Value of returned items: In both brand accounts, there is NO visible difference in the distribution of Total Value of returned items for defaulted and non-defaulted users.
  Thus, this feature could be excluded in any model that tries to predict the chance of a user defaulting.
- Year customer was born: In both brand accounts, the distribution Year customer was born is skewed towards higher values, implying younger customers predominate. However, it looks like the distribution is more concentrated around young customers for defaulted users. So, there is a difference in the distribution of Year customer was born for defaulted and non-defaulted users.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
  There seems to be no difference in distributions between defaulted and non-defaulted users accross brands.
- Days between last payment and start date: In both brand accounts, there is a difference in the distribution of Days between last payment and start date for defaulted and non-defaulted users. For defaulted users this distribution is centered and peaked towards lower values.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
  There seems to be no visible difference in distributions between defaulted and non-defaulted users accross brands.
- Year account started: In both brand accounts, there is a difference in the distribution of Year account started for defaulted and non-defaulted users. For defaulted users this distribution is skewed towards earlier years. This is probably expected,as the longet the account has been acive, probably the more chances to default.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
  The difference in distributions between defaulted and non-defaulted users seems to be higher for "Littlewoods"

Plot distributions for categorical features

```
In [17]: def plot_prob_categorical_feature_given_defaulted(data, features):
             n_rows = len(features)
             plt.close('all')
             plt.figure(figsize=(24, n_rows * 6))
             for j, feature in enumerate(features):
                 plt.subplot(n_rows, 2, j+1)
                 sns.barplot(x="principal_brand", y="defaulted", hue=feature, estimator = np.me
                 if j == 0:
                     plt.title('Probability of a defaulted user for various ctegorical feature
                 if feature == 'title_desc':
                     plt.ylim([0, .2])
                 plt.legend(loc='upper right')
             plt.tight_layout()
```

```
In [18]: plot_prob_categorical_feature_given_defaulted(dfCustomer, ['credit_band', 'title_desc
```

```
C:\Users\bilgin\Anaconda3\envs\ml_adv_py35\lib\site-packages\scipy\stats\stats.py:1713: FutureW
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Observations:

- Level of credit risk: In both brand accounts, there is a difference in the distribution of Level of credit risk for defaulted and non-defaulted users. This is not surprising, since these brobably were already based on prior credit risk assesment. Nevertheless, this feature will be included in later stages.
  Also, there seems to be some difference in the distributions between defaulted and non-defaulted users accross brands.
- Customer's title: In both brand accounts, there seems to have a statistically significant difference in the distribution of Customer's title for defaulted and non-defaulted users. For defaulted users this distribution peaks at higher values and is more concentrated.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
  There seems to be no difference in distributions between defaulted and non-defaulted users accross brands (excluding the title LORD, which has one only sample per group).

Select only relevant features and one-hot-encode the selected categorical features

```
In [19]: with timer('Removing un-informative columns:'):
             selected_columns_customer = ['identifier', 'defaulted']

             selected_numeric_columns_customer = [
                 'LTIME_NET_SALES_AMT',
                 'LTIME_NO_ORDERS',
                 'birth_year',
                 'days_between_last_payment_and_start',
                 'year_start'
             ]
             selected_columns_customer.extend(selected_numeric_columns_customer)
```

```
            dfCustomer_1 = copy.deepcopy(dfCustomer[selected_columns_customer])

        with timer('One-hot-encoding credit_band', n_blanks=4):
            # one-hot-encoding by credit_band and append to previous
            dfTemp = dfCustomer.groupby(['identifier','credit_band']).size().unstack(1).fillna
            dfTemp.columns = ['credit_band.'+ clmn for clmn in list(dfTemp)]
            dfCustomer_1 = dfCustomer_1.merge(dfTemp, how = 'inner', left_index=True, right_in

        with timer('One-hot-encoding title_desc', n_blanks=4):
            # one-hot-encoding by title_description and append to previous
            dfTemp = dfCustomer.groupby(['identifier','title_desc']).size().unstack(1).fillna
            dfTemp.columns = ['title.'+ clmn for clmn in list(dfTemp)]
            dfCustomer_1 = dfCustomer_1.merge(dfTemp, how = 'inner', left_index=True, right_in

        # free memory by deleting the dfCustomer table
        del dfCustomer
        gc.collect();

        dfCustomer_1.head(5)
```

```
 STARTING: Removing un-informative columns:
 FINISHED in 0.00 min.

    STARTING: One-hot-encoding credit_band


C:\Users\bilgin\Anaconda3\envs\ml_adv_py35\lib\site-packages\ipykernel\__main__.py:16: FutureWa
Defaulting to column, but this will raise an ambiguity error in a future version


    FINISHED in 0.02 min.

    STARTING: One-hot-encoding title_desc


C:\Users\bilgin\Anaconda3\envs\ml_adv_py35\lib\site-packages\ipykernel\__main__.py:22: FutureWa
Defaulting to column, but this will raise an ambiguity error in a future version


    FINISHED in 0.02 min.
```

Out[19]:

```
                  identifier  defaulted  LTIME_NET_SALES_AMT  LTIME_NO_ORDERS  \
identifier
37295                  37295      False               604.42                5
315441                315441      False                34.39                1
489894                489894      False               512.94                5
419603                419603      False                25.98                1
```

```
444889          444889      False              424.94                2

             birth_year  days_between_last_payment_and_start  year_start  \
identifier
37295            1972.0                                 141.0        2016
315441           1966.0                                  14.0        2016
489894           1996.0                                 198.0        2016
419603           1946.0                                   1.0        2016
444889           1968.0                                 958.0        2014

             credit_band.U1  credit_band.U2  credit_band.U3      ...        \
identifier                                                       ...
37295                     0               0               0      ...
315441                    0               0               0      ...
489894                    0               0               0      ...
419603                    0               0               0      ...
444889                    1               0               0      ...

             title.2ND-LIEUTENANT  title.DOCTOR  title.FR  title.LORD  \
identifier
37295                           0             0         0           0
315441                          0             0         0           0
489894                          0             0         0           0
419603                          0             0         0           0
444889                          0             0         0           0

             title.MISS  title.MR  title.MRS  title.MS  title.PROFESSOR  \
identifier
37295                 1         0          0         0                0
315441                0         0          1         0                0
489894                1         0          0         0                0
419603                0         1          0         0                0
444889                1         0          0         0                0

             title.REVEREND
identifier
37295                     0
315441                    0
489894                    0
419603                    0
444889                    0

[5 rows x 42 columns]
```

2.4.2 Incude Orders table, as well

Data augmenttion and transformation

1. First, append the 'identifier' abnd 'defaulted' columns from the Cutomer table into the

Order table. Merge on 'identifier'.

2. Then convert the data type of the 'Week_Ending' column int datetime.
3. Then, append more columns related to purchase date and counts
4. Finally, transform the orers table into a table indexed by row and columns are either one-hot-encoded version of the original categorical features or are includng stats of the numeric features. I'll call it dfCustomers_2

```
In [20]: with timer('Appending deafulted column from Customers table'):
             # make a copy of the table
             dfOrder = copy.deepcopy(dataset['Order']['data'])
             # ad a columns that indictes defaulted customer or not and set all to False at fi

             dfOrder = dfOrder.merge(dfCustomer_1[['identifier', 'defaulted']], how = 'left', 

         with timer('Converting date columns to datetime'):
             dateColumns = ['Week_Ending']
             dfOrder = to_datetime(dfOrder, dateColumns)

         with timer('Appending columns related to purchase date and counts'):
             #Create a grouped object (by identiier)
             grouped = dfOrder.groupby('identifier')
             # attach a column indicating days between last and first order
             dfOrder = dfOrder.merge(
                 pd.DataFrame({'days_between_last_and_first_order':
                                   (grouped['Week_Ending'].max()-grouped['Week_Ending'].min())
                 left_on='identifier',
                 right_on='identifier')
             # attach a column indicating number of orders
             dfOrder = dfOrder.merge(
                 pd.DataFrame({'number_of_orders':grouped.size()}),
                 left_on='identifier',
                 right_on='identifier')

         dfOrder.head(5)
```

```
 STARTING: Appending deafulted column from Customers table


C:\Users\bilgin\Anaconda3\envs\ml_adv_py35\lib\site-packages\ipykernel\__main__.py:6: FutureWa
Defaulting to column, but this will raise an ambiguity error in a future version


 FINISHED in 0.03 min.

 STARTING: Converting date columns to datetime
    STARTING: Week_Ending
    FINISHED in 0.13 min.
```

```
FINISHED in 0.13 min.

STARTING: Appending columns related to purchase date and counts
FINISHED in 0.05 min.
```

Out[20]:

```
   identifier Brand  Account_Year_Week Week_Ending Channel  \
0       37295   LAI             201652  2016-12-23  Online
1       37295   LAI             201643  2016-10-21  Online
2       37295   LAI             201651  2016-12-16  Online
3      315441   LEX             201615  2016-04-08  Online
4      489894   LEX             201649  2016-12-02  Online

  online_device_type_detail Account_Type2  Gross_Demand_Pre New_Cust  \
0                    TABLET        Credit             27.00        Y
1                   DESKTOP        Credit            249.99        Y
2                    TABLET        Credit             24.00        Y
3                   DESKTOP        Credit             30.40        Y
4                    MOBILE        Credit            129.00        Y

  Product_dept defaulted  days_between_last_and_first_order  number_of_orders
0       Dept G     False                                 63                 3
1       Dept C     False                                 63                 3
2       Dept B     False                                 63                 3
3       Dept A     False                                  0                 1
4       Dept B     False                                 91                 4
```

In [21]:

```python
with timer('Transformimg Orders table into Customer vs Features table'):
    with timer('One-hot-encoding Brand', n_blanks=4):
        # one-hot-encoding by Brand, row index by identifier
        dfCustomer_2 = dfOrder.groupby(['identifier','Brand']).size().unstack(1).filln
        dfCustomer_2.columns = ['Brand.'+ clmn for clmn in list(dfCustomer_2)]

    with timer('One-hot-encoding Channel', n_blanks=4):
        # one-hot-encoding by Channel and append to previous
        dfTemp = dfOrder.groupby(['identifier','Channel']).size().unstack(1).fillna(0
        dfTemp.columns = ['Channel.'+ clmn for clmn in list(dfTemp)]
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rigl

    with timer('One-hot-encoding Device Type', n_blanks=4):
        # one-hot-encoding by online_device_type_detail and append to previous
        dfTemp = dfOrder.groupby(['identifier','online_device_type_detail']).size().ur
        dfTemp.columns = ['device.'+ clmn for clmn in list(dfTemp)]
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rigl

    with timer('One-hot-encoding Account Type', n_blanks=4):
        # one-hot-encoding by online_device_type_detail and append to previous
```

```
        dfTemp = dfOrder.groupby(['identifier','Account_Type2']).size().unstack(1).fil
        dfTemp.columns = ['account_type.'+ clmn for clmn in list(dfTemp)]
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rig

    with timer('One-hot-encoding Customer Type', n_blanks=4):
        # rename Y to New and N to Returning
        idx = dfOrder[dfOrder['New_Cust'] == 'Y'].index
        dfOrder.loc[idx,'New_Cust'] = 'new'
        idx = dfOrder[dfOrder['New_Cust'] == 'N'].index
        dfOrder.loc[idx,'New_Cust'] = 'returning'
        # one-hot-encoding by online_device_type_detail and append to previous
        dfTemp = dfOrder.groupby(['identifier','New_Cust']).size().unstack(1).fillna(
        dfTemp.columns = ['customer_type.'+ clmn for clmn in list(dfTemp)]
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rig

    with timer('One-hot-encoding Product Department', n_blanks=4):
        # one-hot-encoding by online_device_type_detail and append to previous
        dfTemp = dfOrder.groupby(['identifier','Product_dept']).size().unstack(1).fil
        dfTemp.columns = ['department.'+ clmn for clmn in list(dfTemp)]
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rig

    with timer('Adding stats columns for Gross Demand Pre Credit', n_blanks=4):
        # one-hot-encoding by online_device_type_detail and append to previous
        grouped = dfOrder.groupby('identifier')
        dfTemp = grouped['Gross_Demand_Pre'].agg(['min', 'max', 'mean', 'median', 'st
        dfTemp.columns = ['Gross_Demand_Pre_Credit.'+ clmn for clmn in list(dfTemp)]
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rig
        dfCustomer_2['Gross_Demand_Pre_Credit.std'].fillna(0,inplace=True),

    with timer('Adding stats columns for number_of_orders', n_blanks=4):
        # one-hot-encoding by online_device_type_detail and append to previous
        dfTemp = grouped[['number_of_orders']].mean()
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rig

    with timer('Adding stats columns for Days Between Last_and First Order', n_blanks
        # one-hot-encoding by online_device_type_detail and append to previous
        dfTemp = grouped[['days_between_last_and_first_order']].mean()
        dfCustomer_2 = dfCustomer_2.merge(dfTemp, how = 'inner', left_index=True, rig
        dfCustomer_2['avg_days_between_orders'] = (
            dfCustomer_2['days_between_last_and_first_order']
            / dfCustomer_2['number_of_orders'])

    with timer('Appending the defaulted flag column', n_blanks=4):
        dfCustomer_2 = dfCustomer_2.merge(dfCustomer_1[['defaulted']], how = 'inner',
```

```
dfCustomer_2.head(5)
```

```
STARTING: Transformimg Orders table into Customer vs Features table
   STARTING: One-hot-encoding Brand
   FINISHED in 0.02 min.

   STARTING: One-hot-encoding Channel
   FINISHED in 0.01 min.

   STARTING: One-hot-encoding Device Type
   FINISHED in 0.01 min.

   STARTING: One-hot-encoding Account Type
   FINISHED in 0.01 min.

   STARTING: One-hot-encoding Customer Type
   FINISHED in 0.04 min.

   STARTING: One-hot-encoding Product Department
   FINISHED in 0.02 min.

   STARTING: Adding stats columns for Gross Demand Pre Credit
   FINISHED in 0.01 min.

   STARTING: Adding stats columns for number_of_orders
   FINISHED in 0.00 min.

   STARTING: Adding stats columns for Days Between Last_and First Order
   FINISHED in 0.00 min.

   STARTING: Appending the defaulted flag column
   FINISHED in 0.00 min.

FINISHED in 0.13 min.
```

Out[21]:

```
            Brand.LAI  Brand.LEX  Channel.Offline  Channel.Online  \
identifier
1                   0         31                0              31
2                   0          1                0               1
3                   0          2                0               2
4                   0          1                0               1
5                   0          9                0               9

            device.DESKTOP  device.MOBILE  device.TABLET  account_type.Cash  \
identifier
1                        0             27              4                  0
2                        0              1              0                  1
3                        0              2              0                  0
```

```
4                        1              0              0                    0
5                        1              3              5                    0

            account_type.Credit  customer_type.new    ...      \
identifier                                            ...
1                            31                  2    ...
2                             0                  1    ...
3                             2                  1    ...
4                             1                  1    ...
5                             9                  7    ...

            department.Dept X  Gross_Demand_Pre_Credit.min  \
identifier
1                           0                         7.00
2                           0                        79.99
3                           0                        59.99
4                           0                        79.99
5                           0                         9.90

            Gross_Demand_Pre_Credit.max  Gross_Demand_Pre_Credit.mean  \
identifier
1                                110.00                     38.137097
2                                 79.99                     79.990000
3                                199.00                    129.495000
4                                 79.99                     79.990000
5                                409.00                    210.100000

            Gross_Demand_Pre_Credit.median  Gross_Demand_Pre_Credit.std  \
identifier
1                                   35.000                    21.695798
2                                   79.990                     0.000000
3                                  129.495                    98.294914
4                                   79.990                     0.000000
5                                  215.000                   194.573707

            number_of_orders  days_between_last_and_first_order  \
identifier
1                         31                                532
2                          1                                  0
3                          2                                 70
4                          1                                  0
5                          9                                175

            avg_days_between_orders  defaulted
identifier
1                         17.161290      False
2                          0.000000      False
3                         35.000000      False
```

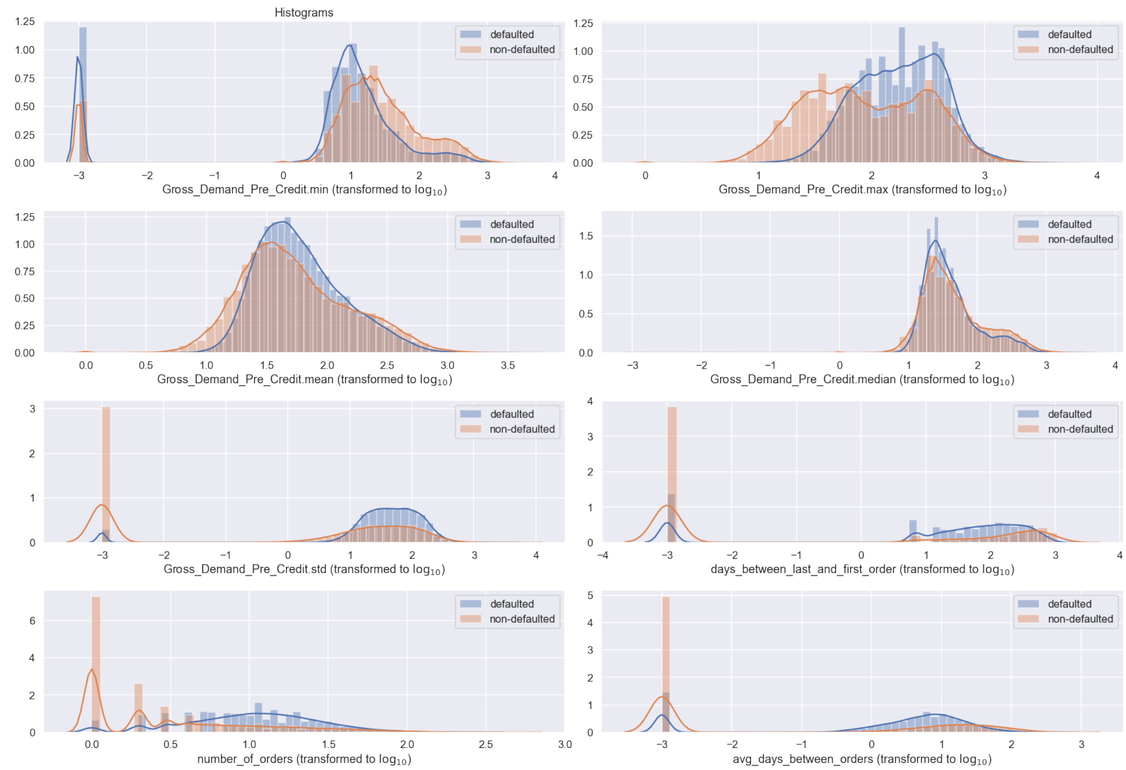
```
4                            0.000000      False
5                           19.444444      False

[5 rows x 31 columns]
```

Plot distributions for numeric features

```
In [22]: def plot_prob_numeric_feature_given_defaulted_orders(data, features, log_transformed_f
             n_rows = len(features)
             plt.close('all')
             plt.figure(figsize=(24, n_rows * 4))
             for j, feature in enumerate(features):
                 plt.subplot(n_rows, 2, j+1)
                 df = copy.deepcopy(data)
                 if feature in log_transformed_fields:
                     series_defaulted = (df[df['defaulted']][feature]+.001).apply(np.log10)
                     series_non_defaulted = (df[~df['defaulted']][feature]+.001).apply(np.log10
                 else:
                     series_defaulted = df[df['defaulted']][feature]
                     series_non_defaulted = df[~df['defaulted']][feature]

                 # plot normed histogram of log_value for defaulted customers
                 sns.distplot(
                     series_defaulted,
                     hist = True,
                     label='defaulted')
                 # plot normed histogram of log_value for non-defaulted customers
                 sns.distplot(
                     series_non_defaulted,
                     hist = True,
                     label='non-defaulted')

                 if feature in log_transformed_fields:
                     plt.xlabel(feature + ' (transformed to $\log_{10}$)')
                 plt.legend()
                 if j == 0:
                     plt.title('Histograms')


             plt.tight_layout()

In [23]: plot_prob_numeric_feature_given_defaulted_orders(
             dfCustomer_2,
             [
                 'Gross_Demand_Pre_Credit.min',
                 'Gross_Demand_Pre_Credit.max',
                 'Gross_Demand_Pre_Credit.mean',
                 'Gross_Demand_Pre_Credit.median',
```

```
                'Gross_Demand_Pre_Credit.std',
                'days_between_last_and_first_order',
                'number_of_orders',
                'avg_days_between_orders'
            ],
            log_transformed_fields = [
                'Gross_Demand_Pre_Credit.min',
                'Gross_Demand_Pre_Credit.max',
                'Gross_Demand_Pre_Credit.mean',
                'Gross_Demand_Pre_Credit.median',
                'Gross_Demand_Pre_Credit.std',
                'days_between_last_and_first_order',
                'number_of_orders',
                'avg_days_between_orders'
            ]
        )
```

```
C:\Users\bilgin\Anaconda3\envs\ml_adv_py35\lib\site-packages\scipy\stats\stats.py:1713: FutureW
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Observations:

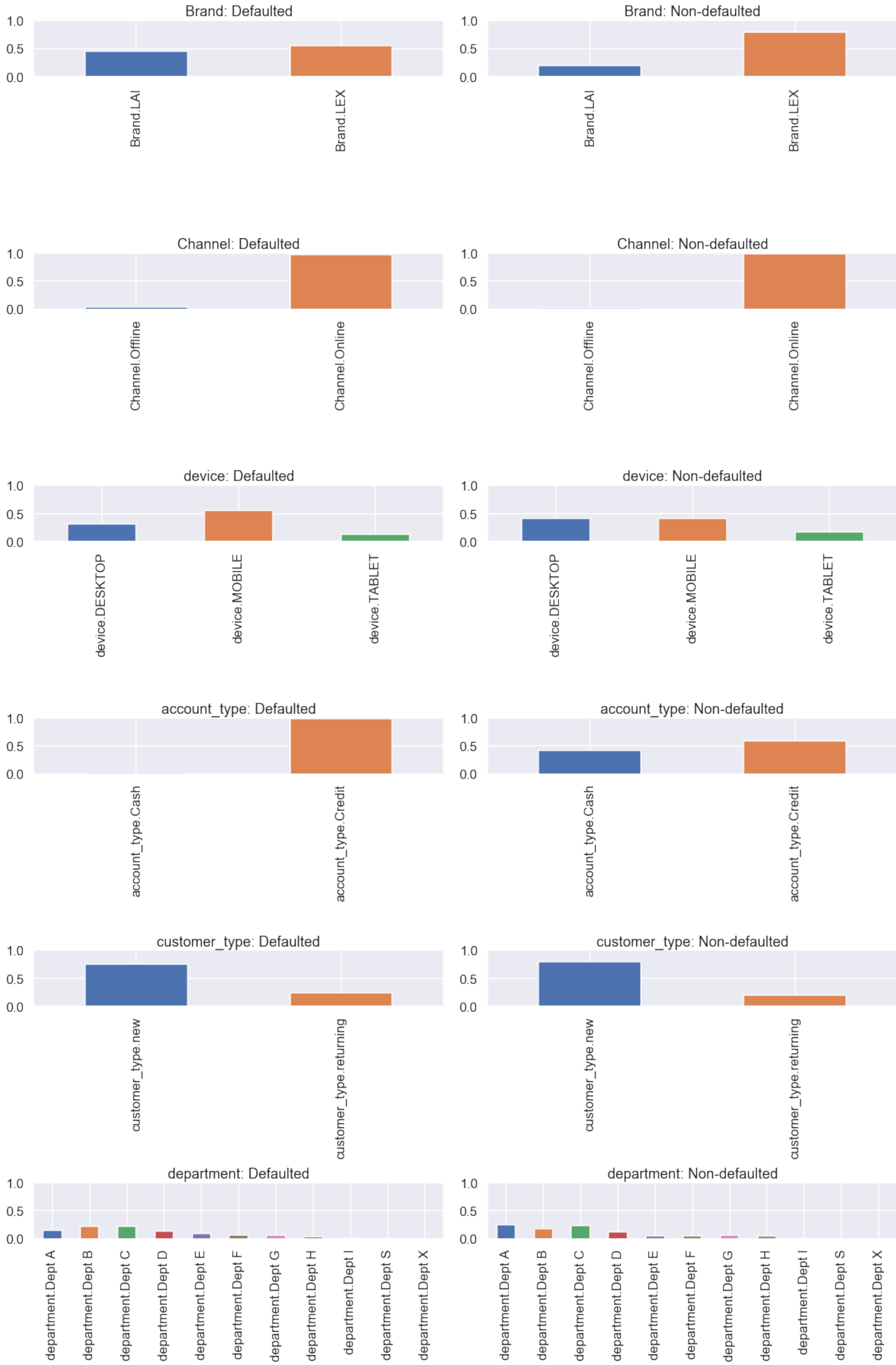
- Minimum value of Gross_Demand_Pre_Credit: There is a difference in the distribution of this feature for defaulted and non-defaulted users. For defaulted users the distribution is skewed towards lower values.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Maximum value of Gross_Demand_Pre_Credit: There is a difference in the distribution of this feature for defaulted and non-defaulted users. For defaulted users the distribution is skewed towards higher values.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Mean value Gross_Demand_Pre_Credit: There is a slight difference in the distribution of this feature for defaulted and non-defaulted users. For defaulted users the distribution is peaked at higher values.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Median value of Gross_Demand_Pre_Credit: There is not a visible difference in the distribution of this feature for defaulted and non-defaulted users.
  Thus, this feature could be be excluded from any model that tries to predict the chance of a user defaulting.
- Standard Deviation in Gross_Demand_Pre_Credit: There is not a visible difference in the distribution of this feature for defaulted and non-defaulted users, except for the very high peak at zero for non-deafaulted users, due to customers with single purchase. Thus, this feature could be be excluded from any model that tries to predict the chance of a user defaulting.
- Days between last payment and first purchase: There is a difference in the distribution of this feature for defaulted and non-defaulted users. For defaulted users the distribution is skewed towards lower values.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Number of Orders: There is a difference in the distribution of this feature for defaulted and non-defaulted users. For defaulted users the distribution is skewed towards higher values. Also, there are many non-dfulted customers with a single purchase.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Average Days Between Orders: There is a difference in the distribution of this feature for defaulted and non-defaulted users. For defaulted users the distribution is skewed towards higher values. Also, there are many non-dfulted customers with a single purchase.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.

Plot distributions for categorical features

```
In [24]: def plot_prob_categorical_unstacked_features_given_defaulted(df, features):
             idx_defaulted = df[df['defaulted']].index
             idx_non_defaulted = df[~df['defaulted']].index

             all_columns = list(df.columns)
             selected_columns = []
```

```
    cnt_rows = len(features)
    for feature in features:
        feature_columns = [column for column in all_columns if column.split('.')[0] ==
        selected_columns.append(feature_columns)

    plt.close('all')
    plt.figure(figsize=(16, 4 * cnt_rows))
    for j, columns in enumerate(selected_columns):
        df_temp = df[columns].divide(df['number_of_orders'],axis=0).merge(df[['defaul
        plt.subplot(cnt_rows,2,j*2 + 1)
        df_probs = df_temp.loc[idx_defaulted][columns].mean()
        df_probs /=  df_probs.sum()
        df_probs.plot.bar()
        plt.ylim([0,1])
        plt.title(features[j] + ': Defaulted')

        plt.subplot(cnt_rows,2,j*2 + 2)
        df_probs = df_temp.loc[idx_non_defaulted][columns].mean()
        df_probs /=  df_probs.sum()
        df_probs.plot.bar()
        plt.ylim([0,1])
        plt.title(features[j] + ': Non-defaulted')

    plt.tight_layout()
```

```
In [25]: features = ['Brand', 'Channel', 'device', 'account_type', 'customer_type', 'departmen
         plot_prob_categorical_unstacked_features_given_defaulted(dfCustomer_2, features)
```


Brand: Defaulted
Brand: Non-defaulted
Brand.LAI
Brand.LEX
Channel: Defaulted
Channel: Non-defaulted
Channel.Offline
Channel.Online
device: Defaulted
device: Non-defaulted
device.DESKTOP
device.MOBILE
device.TABLET
account_type: Defaulted
account_type: Non-defaulted
account_type.Cash
account_type.Credit
customer_type: Defaulted
customer_type: Non-defaulted
customer_type.new
customer_type.returning
department: Defaulted
department: Non-defaulted
department.Dept A
department.Dept B
department.Dept C
department.Dept D
department.Dept E
department.Dept F
department.Dept G
department.Dept H
department.Dept I
department.Dept S
department.Dept X
1.0
0.5
0.0

Observations:

- Brand: There is a difference in the distribution of this feature for defaulted and non-defaulted users. There is higher probability of observing Littlewoods for deafulted than for non-defaulted customers.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Channel: There is no visible difference in the distribution of this feature for defaulted and non-defaulted users.
  Thus, this feature could be excluded from any model that tries to predict the chance of a user defaulting.
- Device: There is a difference in the distribution of this feature for defaulted and non-defaulted users. There is higher probability of observing Mobile for deafulted than for non-defaulted customers.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Account Type: There is a difference in the distribution of this feature for defaulted and non-defaulted users. There is lowe probability of observing Cash for deafulted than for non-defaulted customers.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.
- Customer Type: There is no visible difference in the distribution of this feature for defaulted and non-defaulted users.
  Thus, this feature could be excluded from any model that tries to predict the chance of a user defaulting.
- Department: There is a slight difference in the distribution of this feature for defaulted and non-defaulted users. There is higher probability of observing DepartmentB (and lower probability for Department A) for deafulted than for non-defaulted customers.
  Thus, this feature should be included in any model that tries to predict the chance of a user defaulting.

Select only relevant features and one-hot-encode the selected categorical features

```
In [26]: with timer('Removing un-informative columns:'):
             selected_columns_orders = []

             selected_numeric_columns_orders = [
                 'Gross_Demand_Pre_Credit.min',
                 'Gross_Demand_Pre_Credit.max',
                 'Gross_Demand_Pre_Credit.mean',
                 'days_between_last_and_first_order',
                 'number_of_orders',
                 'avg_days_between_orders'
             ]
             selected_columns_orders.extend(selected_numeric_columns_orders)
```

```python
    selected_categorical_columns_orders = [
       'Brand.LAI',
        'Brand.LEX',
        'device.DESKTOP',
        'device.MOBILE',
        'device.TABLET',
        'account_type.Cash',
        'account_type.Credit',
        'department.Dept A',
        'department.Dept B',
        'department.Dept C',
        'department.Dept D',
        'department.Dept E',
        'department.Dept F',
        'department.Dept G',
        'department.Dept H',
        'department.Dept I',
        'department.Dept S',
        'department.Dept X',
    ]
    selected_columns_orders.extend(selected_categorical_columns_orders)

    dfCustomer_2 = copy.deepcopy(dfCustomer_2[selected_columns_orders])

dfCustomer_2.head(5)
```

```
STARTING: Removing un-informative columns:
FINISHED in 0.00 min.
```

Out[26]:

identifier	Gross_Demand_Pre_Credit.min	Gross_Demand_Pre_Credit.max
1	7.00	110.00
2	79.99	79.99
3	59.99	199.00
4	79.99	79.99
5	9.90	409.00

identifier	Gross_Demand_Pre_Credit.mean	days_between_last_and_first_order
1	38.137097	532
2	79.990000	0
3	129.495000	70
4	79.990000	0
5	210.100000	175

```
            number_of_orders  avg_days_between_orders  Brand.LAI  Brand.LEX  \
identifier
1                         31                17.161290          0         31
2                          1                 0.000000          0          1
3                          2                35.000000          0          2
4                          1                 0.000000          0          1
5                          9                19.444444          0          9

            device.DESKTOP  device.MOBILE        ...          \
identifier                                       ...
1                        0             27        ...
2                        0              1        ...
3                        0              2        ...
4                        1              0        ...
5                        1              3        ...

            department.Dept B  department.Dept C  department.Dept D  \
identifier
1                           1                  0                  6
2                           0                  1                  0
3                           0                  1                  0
4                           0                  1                  0
5                           1                  4                  0

            department.Dept E  department.Dept F  department.Dept G  \
identifier
1                          18                  5                  1
2                           0                  0                  0
3                           0                  0                  0
4                           0                  0                  0
5                           4                  0                  0

            department.Dept H  department.Dept I  department.Dept S  \
identifier
1                           0                  0                  0
2                           0                  0                  0
3                           1                  0                  0
4                           0                  0                  0
5                           0                  0                  0

            department.Dept X
identifier
1                           0
2                           0
3                           0
4                           0
5                           0
```

[5 rows x 24 columns]

2.4.3 Join the two tables into a single data table and log-transform some columns:

This table will be the main table for clustering and classification studies. Each row will be indexed by customer id and each column will represent a feature.

```
In [27]: with timer('Merging the dfCustomer_1 and dfCustomer_2 tables'):
             # merge teables
             dfCustomerFeatures = dfCustomer_1.merge(dfCustomer_2, how='inner', left_index=True
             # print some stats
             table_stats(dfCustomer_1, 'dfCustomer_1')
             print()
             table_stats(dfCustomer_2, 'dfCustomer_2')
             print()
             table_stats(dfCustomerFeatures, 'dfCustomerFeatures')


         with timer('Log-transforming some numeric features'):
             columns_to_log_transform = [
                 'LTIME_NET_SALES_AMT',
                 'LTIME_NO_ORDERS',
                 'days_between_last_payment_and_start',
                 'Gross_Demand_Pre_Credit.min',
                 'Gross_Demand_Pre_Credit.max',
                 'Gross_Demand_Pre_Credit.mean',
                 'days_between_last_and_first_order',
                 'number_of_orders',
                 'avg_days_between_orders'
             ]
             for clmn in columns_to_log_transform:
                 # find rows with value equal to zero or less
                 idxZero = dfCustomerFeatures[dfCustomerFeatures[clmn] <= 0].index
                 # set them to a small value
                 dfCustomerFeatures.loc[idxZero, clmn] = 1e-6
                 # log transform
                 dfCustomerFeatures[clmn] = dfCustomerFeatures[clmn].apply(np.log10)

         #free memory
         del dfCustomer_1, dfCustomer_2
         gc.collect();

         dfCustomerFeatures.head(5)
```

```
 STARTING: Merging the dfCustomer_1 and dfCustomer_2 tables
TABLE DFCUSTOMER_1
-------------------------------------------------------------------------------------------
Number of rows                          : 509,684
```

```
Number of columns                         : 42

Column name                                 Number of unique values (data type)

                          identifier: 509,684 (int64)
                           defaulted: 2 (bool)
                  LTIME_NET_SALES_AMT: 101,239 (float64)
                      LTIME_NO_ORDERS: 167 (int64)
                          birth_year: 96 (float64)
     days_between_last_payment_and_start: 1,173 (float64)
                          year_start: 3 (int64)
                      credit_band.U1: 2 (int32)
                      credit_band.U2: 2 (int32)
                      credit_band.U3: 2 (int32)
                      credit_band.W1: 2 (int32)
                      credit_band.W4: 2 (int32)
                      credit_band.W7: 2 (int32)
                      credit_band.X1: 2 (int32)
                      credit_band.X2: 2 (int32)
                      credit_band.X3: 2 (int32)
                      credit_band.X4: 2 (int32)
                      credit_band.X5: 2 (int32)
                      credit_band.Y1: 2 (int32)
                      credit_band.Y2: 2 (int32)
                      credit_band.Y3: 2 (int32)
                      credit_band.Y4: 2 (int32)
                      credit_band.Y5: 2 (int32)
                      credit_band.Y6: 2 (int32)
                      credit_band.Y7: 2 (int32)
                      credit_band.Z1: 2 (int32)
                      credit_band.Z2: 2 (int32)
                      credit_band.Z3: 2 (int32)
                      credit_band.Z4: 2 (int32)
                      credit_band.Z5: 2 (int32)
                      credit_band.Z6: 2 (int32)
                      credit_band.Z7: 2 (int32)
                 title.2ND-LIEUTENANT: 2 (int32)
                        title.DOCTOR: 2 (int32)
                            title.FR: 2 (int32)
                          title.LORD: 2 (int32)
                          title.MISS: 2 (int32)
                            title.MR: 2 (int32)
                           title.MRS: 2 (int32)
                            title.MS: 2 (int32)
                     title.PROFESSOR: 2 (int32)
                      title.REVEREND: 2 (int32)
----------------------------------------------------------------------------------------------
```

```
TABLE DFCUSTOMER_2
--------------------------------------------------------------------------------
Number of rows                          : 367,669
Number of columns                       : 24

Column name                               Number of unique values (data type)

             Gross_Demand_Pre_Credit.min: 4,158 (float64)
             Gross_Demand_Pre_Credit.max: 5,444 (float64)
            Gross_Demand_Pre_Credit.mean: 127,629 (float64)
       days_between_last_and_first_order: 157 (int64)
                        number_of_orders: 330 (int64)
                  avg_days_between_orders: 9,320 (float64)
                               Brand.LAI: 243 (int32)
                               Brand.LEX: 320 (int32)
                          device.DESKTOP: 222 (int32)
                           device.MOBILE: 256 (int32)
                           device.TABLET: 190 (int32)
                       account_type.Cash: 102 (int32)
                     account_type.Credit: 329 (int32)
                       department.Dept A: 184 (int32)
                       department.Dept B: 124 (int32)
                       department.Dept C: 63 (int32)
                       department.Dept D: 85 (int32)
                       department.Dept E: 83 (int32)
                       department.Dept F: 85 (int32)
                       department.Dept G: 67 (int32)
                       department.Dept H: 25 (int32)
                       department.Dept I: 15 (int32)
                       department.Dept S: 10 (int32)
                       department.Dept X: 6 (int32)
--------------------------------------------------------------------------------

TABLE DFCUSTOMERFEATURES
--------------------------------------------------------------------------------
Number of rows                          : 367,669
Number of columns                       : 66

Column name                               Number of unique values (data type)

                              identifier: 367,669 (int64)
                               defaulted: 2 (bool)
                      LTIME_NET_SALES_AMT: 99,632 (float64)
                          LTIME_NO_ORDERS: 166 (int64)
                              birth_year: 95 (float64)
     days_between_last_payment_and_start: 1,173 (float64)
                              year_start: 3 (int64)
                          credit_band.U1: 2 (int32)
```

```
                 credit_band.U2: 2 (int32)
                 credit_band.U3: 2 (int32)
                 credit_band.W1: 2 (int32)
                 credit_band.W4: 1 (int32)
                 credit_band.W7: 2 (int32)
                 credit_band.X1: 2 (int32)
                 credit_band.X2: 2 (int32)
                 credit_band.X3: 2 (int32)
                 credit_band.X4: 2 (int32)
                 credit_band.X5: 2 (int32)
                 credit_band.Y1: 2 (int32)
                 credit_band.Y2: 2 (int32)
                 credit_band.Y3: 2 (int32)
                 credit_band.Y4: 2 (int32)
                 credit_band.Y5: 2 (int32)
                 credit_band.Y6: 2 (int32)
                 credit_band.Y7: 2 (int32)
                 credit_band.Z1: 2 (int32)
                 credit_band.Z2: 2 (int32)
                 credit_band.Z3: 2 (int32)
                 credit_band.Z4: 2 (int32)
                 credit_band.Z5: 2 (int32)
                 credit_band.Z6: 2 (int32)
                 credit_band.Z7: 2 (int32)
           title.2ND-LIEUTENANT: 1 (int32)
                   title.DOCTOR: 2 (int32)
                       title.FR: 2 (int32)
                     title.LORD: 2 (int32)
                     title.MISS: 2 (int32)
                       title.MR: 2 (int32)
                      title.MRS: 2 (int32)
                       title.MS: 2 (int32)
                title.PROFESSOR: 2 (int32)
                 title.REVEREND: 1 (int32)
    Gross_Demand_Pre_Credit.min: 4,158 (float64)
    Gross_Demand_Pre_Credit.max: 5,444 (float64)
   Gross_Demand_Pre_Credit.mean: 127,629 (float64)
days_between_last_and_first_order: 157 (int64)
               number_of_orders: 330 (int64)
         avg_days_between_orders: 9,320 (float64)
                      Brand.LAI: 243 (int32)
                      Brand.LEX: 320 (int32)
                 device.DESKTOP: 222 (int32)
                  device.MOBILE: 256 (int32)
                  device.TABLET: 190 (int32)
              account_type.Cash: 102 (int32)
            account_type.Credit: 329 (int32)
              department.Dept A: 184 (int32)
```

```
                    department.Dept B: 124 (int32)
                    department.Dept C: 63 (int32)
                    department.Dept D: 85 (int32)
                    department.Dept E: 83 (int32)
                    department.Dept F: 85 (int32)
                    department.Dept G: 67 (int32)
                    department.Dept H: 25 (int32)
                    department.Dept I: 15 (int32)
                    department.Dept S: 10 (int32)
                    department.Dept X: 6 (int32)
-------------------------------------------------------------------------------------------------
FINISHED in 0.01 min.

STARTING: Log-transforming some numeric features
FINISHED in 0.01 min.
```

Out[27]:

```
            identifier  defaulted  LTIME_NET_SALES_AMT  LTIME_NO_ORDERS  \
identifier
37295            37295      False             2.781339         0.698970
315441          315441      False             1.536432         0.000000
489894          489894      False             2.710067         0.698970
419603          419603      False             1.414639         0.000000
399884          399884      False             2.472917         0.778151

            birth_year  days_between_last_payment_and_start  year_start  \
identifier
37295           1972.0                             2.149219        2016
315441          1966.0                             1.146128        2016
489894          1996.0                             2.296665        2016
419603          1946.0                             0.000000        2016
399884          1996.0                                  NaN        2016

            credit_band.U1  credit_band.U2  credit_band.U3       ...          \
identifier                                                       ...
37295                    0               0               0       ...
315441                   0               0               0       ...
489894                   0               0               0       ...
419603                   0               0               0       ...
399884                   0               0               0       ...

            department.Dept B  department.Dept C  department.Dept D  \
identifier
37295                       1                  1                  0
315441                      0                  0                  0
489894                      2                  2                  0
419603                      0                  1                  0
```

```
399884                        0                  2                  0

            department.Dept E  department.Dept F  department.Dept G  \
identifier
37295                         0                  0                  1
315441                        0                  0                  0
489894                        0                  0                  0
419603                        0                  0                  0
399884                        2                  3                  0

            department.Dept H  department.Dept I  department.Dept S  \
identifier
37295                         0                  0                  0
315441                        0                  0                  0
489894                        0                  0                  0
419603                        0                  0                  0
399884                        0                  0                  0

            department.Dept X
identifier
37295                         0
315441                        0
489894                        0
419603                        0
399884                        0

[5 rows x 66 columns]
```

2.5 Customer clastering using unsupervised methods

```
In [ ]: import scipy as sp
        import sklearn as sk
        from sklearn.preprocessing import StandardScaler, MaxAbsScaler, MinMaxScaler, RobustSca
        from sklearn import cluster, mixture, linear_model
        from sklearn.cluster import AgglomerativeClustering, DBSCAN
        from sklearn.metrics import pairwise_distances
        from sklearn.metrics.pairwise import euclidean_distances, cosine_distances
        from sklearn import (manifold, datasets, decomposition, ensemble,
                     discriminant_analysis, random_projection, metrics)
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neighbors import kneighbors_graph
        #from sklearn.feature_selection import f_regression, mutual_info_regression
        from sklearn.model_selection import train_test_split, cross_val_score
        from scipy.spatial.distance import pdist, squareform
        #from scipy.optimize import curve_fit
        #from scipy.stats import power_divergence
        #from scipy.special import xlogy
        import umap
```

2.5.1 Visual inspecton of possible clusters using embedings to lower dimensions

1. I'll embed the a sample of the original samples with 64 features into 3D space using three different embeding methos: PCA, Umap, and t-SNE. I'll use different metrics, as well.
2. Then I'll look at the projections of the samples in these lower dimensional spaces to get an idea of how the data (customers) cluster.
3. Then I'll run a few clustering methods in the original data space to cluster the sampled set of data
4. I'll plot the cluster results in various embeded sub-spaces

In [28]:
```
class ClusterAnalysis():
    def __init__(self, df_data):
        self.arr_data = df_data.fillna(0).values
        self.dat_column_labels = list(df_data.columns)
        self.embeding_models = {}
        self.embeded_data = {}

    def standardise_data(self, method):
        if method.lower() == 'standard':
            return StandardScaler().fit_transform(self.arr_data)
        elif method.lower() == 'minmax':
            return MinMaxScaler().fit_transform(self.arr_data)
        elif method.lower() == 'maxabs':
            return MaxAbsScaler().fit_transform(self.arr_data)
        elif method.lower() == 'robust':
            return RobustScaler().fit_transform(self.arr_data)
        else:
            return self.arr_data

    def embed_pca(self, n_components=3, algorithm = 'arpack', rescaling_method=None):
        self.embeding_models['pca'] = decomposition.TruncatedSVD(
            n_components = n_components,
            algorithm = algorithm
        )
        self.embeded_data['pca'] = self.embeding_models['pca'].fit_transform(
            self.standardise_data(rescaling_method))

    def embed_umap(self, n_components=3, n_neighbors=5, min_dist=.2, metrics=None, res
        if metrics is None:
            self.embeding_models['umap'] = umap.UMAP(
                n_components = n_components,
                n_neighbors = n_neighbors,
                min_dist = min_dist)
            self.embeded_data['umap'] = self.embeding_models['umap'].fit_transform(
                self.standardise_data(rescaling_method))
        else:
            self.embeding_models['umap'] = {}
            self.embeded_data['umap'] = {}
            for metric in metrics:
```

```python
                self.embeding_models['umap'][metric] = umap.UMAP(
                    n_components = n_components,
                    n_neighbors = n_neighbors,
                    min_dist = min_dist,
                    metric = metric)
                self.embeded_data['umap'][metric] = self.embeding_models['umap'][metri
                    self.standardise_data(rescaling_method))

    def embed_tsne(self, n_components=3, perplexity=30, learning_rate=4, early_exagge
                   metrics=None, rescaling_method=None):
        if metrics is None:
            self.embeding_models['tsne'] = manifold.TSNE(
                n_components = n_components,
                perplexity = perplexity,
                learning_rate = learning_rate,
                early_exaggeration = early_exaggeration,
                n_iter = n_iter,
                init=init)
            self.embeded_data['tsne'] = self.embeding_models['tsne'].fit_transform(
                self.standardise_data(self.arr_data, rescaling_method))
        else:
            self.embeding_models['tsne'] = {}
            self.embeded_data['tsne'] = {}
            for metric in metrics:
                self.embeding_models['tsne'][metric] = manifold.TSNE(
                    n_components = n_components,
                    perplexity = perplexity,
                    learning_rate = learning_rate,
                    early_exaggeration = early_exaggeration,
                    n_iter = n_iter,
                    init=init,
                    metric = metric
                )
                self.embeded_data['tsne'][metric] = self.embeding_models['tsne'][metri
                    self.standardise_data(rescaling_method))


    def plot_embeddings(self, alpha=0.25, color_labels=None):
        if color_labels is not None:
            n_colors = len(np.unique(color_labels))
        cnt_rows = 0
        for model, item in self.embeded_data.items():
            if model == 'pca':
                cnt_rows += 1
            else:
                #cnt_rows += len(item.keys())
                for metric, item_2 in item.items():
                    cnt_rows += 1
```

```
        plt.close('all')
        plt.figure(figsize=(24, 5 * cnt_rows))
        j = 0
        for model, item in self.embeded_data.items():
            if model == 'pca':
                for k in range(item.shape[1]):
                    for l in range(k+1,item.shape[1]):
                        j+=1
                        plt.subplot(cnt_rows, 3, j)
                        if color_labels is None:
                            plt.plot(item[:, k], item[:, l], '.', alpha = alpha)
                        else:
                            for color_ in range(n_colors):
                                idx_color = np.where(color_labels == color_)[0]
                                plt.plot(item[idx_color, k], item[idx_color, l], '.',
                            plt.legend()
                        plt.xticks([]), plt.yticks([])
                        plt.title(model + ': P_' + str(k)+str(l))
            else:
                #cnt_rows += len(item.keys())
                for metric, item_2 in item.items():
                    for k in range(item_2.shape[1]):
                        for l in range(k+1,item_2.shape[1]):
                            j+=1
                            plt.subplot(cnt_rows, 3, j)
                            if color_labels is None:
                                plt.plot(item_2[:, k], item_2[:, l], '.', alpha = alp
                            else:
                                for color_ in range(n_colors):
                                    idx_color = np.where(color_labels == color_)[0]
                                    plt.plot(item_2[idx_color, k], item_2[idx_color, 
                                #plt.legend()
                            plt.xticks([]), plt.yticks([])
                            plt.title(model + ': ' + metric)


    def cluster_data(self, params, rescaling_method=None):

        # ============
        # Create cluster objects
        # ============

        mini_batch_kmeans = cluster.MiniBatchKMeans(n_clusters=params['n_clusters'])

        affinity_propagation = cluster.AffinityPropagation(
            damping=params['affinityPropagation_damping'], preference=params['affinit
```

```
# estimate bandwidth for mean shift
bandwidth = cluster.estimate_bandwidth(self.standardise_data(rescaling_method)
mean_shift = cluster.MeanShift(bandwidth=bandwidth, bin_seeding=False)

spectral = cluster.SpectralClustering(
    n_clusters=params['n_clusters'], eigen_solver='arpack',
    affinity="nearest_neighbors")

# connectivity matrix for structured Ward
connectivity = kneighbors_graph(
    self.standardise_data(rescaling_method), n_neighbors=params['ward_n_neighb
# make connectivity symmetric
connectivity = 0.5 * (connectivity + connectivity.T)
ward = cluster.AgglomerativeClustering(
    n_clusters=params['n_clusters'], linkage='ward',
    connectivity=connectivity)

average_linkage = cluster.AgglomerativeClustering(
    linkage="average", affinity="cosine",
    n_clusters=params['n_clusters'], connectivity=connectivity)

dbscan = cluster.DBSCAN(eps=params['dbscan_eps'], metric=params['dbscan_metric

birch = cluster.Birch(threshold=params['birch_thrsld'], n_clusters=params['n_c

gmm = mixture.GaussianMixture(
    n_components=params['n_clusters'], covariance_type='full')

self.clustering_algorithms = [
    ('MiniBatchKMeans', mini_batch_kmeans),
    #('AffinityPropagation', affinity_propagation),
    #('MeanShift', mean_shift),
    ('SpectralClustering', spectral),
    ('Ward', ward),
    ('AgglomerativeClustering', average_linkage),
    #('DBSCAN', dbscan),
    ('Birch', birch),
    ('GaussianMixture', gmm)
]

self.cluster_labels=np.zeros((self.standardise_data(rescaling_method).shape[0]

for j, (name, algorithm) in enumerate(self.clustering_algorithms):
    t0 = tm.time()

    # catch warnings related to kneighbors_graph
    with warnings.catch_warnings():
        warnings.filterwarnings(
```

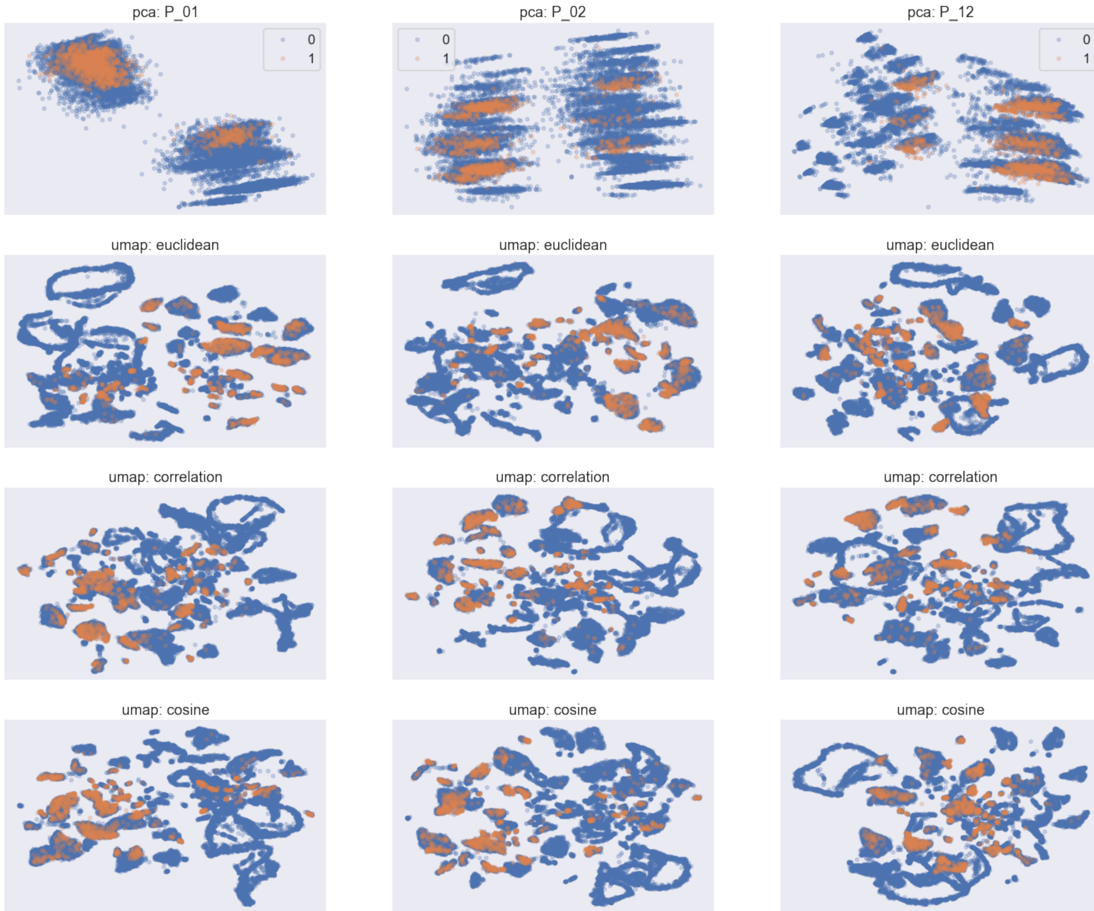
```
                "ignore",
                message="the number of connected components of the " +
                "connectivity matrix is [0-9]{1,2}" +
                " > 1. Completing it to avoid stopping the tree early.",
                category=UserWarning)
            warnings.filterwarnings(
                "ignore",
                message="Graph is not fully connected, spectral embedding" +
                " may not work as expected.",
                category=UserWarning)
            algorithm.fit(self.standardise_data(rescaling_method))


        if hasattr(algorithm, 'labels_'):
            self.cluster_labels[:,j] = algorithm.labels_.flatten()
            numberClusters = len(np.unique(algorithm.labels_))
        else:
            self.cluster_labels[:,j] =algorithm.predict(self.standardise_data(resc
            numberClusters = len(algorithm.weights_)

        t1 = tm.time()
        print('{0:2d}. {1:30s}: time {2:3.3f}; number of clusters = {3:}'.format(j


def plot_clusters(self, data, clustering_algorithms, labels, alpha = 0.01, type_=

    #palette = ["#2ecc71", "#3498db", "#e74c3c", "#9b59b6", "#34495e", ]
    palette = itertools.cycle(sns.color_palette())

    plt.close('all')

    plt.figure(figsize = (24, 32))
    for n, (name, algorithm) in enumerate(clustering_algorithms):
        temp = copy.deepcopy(data)
        #print('Clustering method is',name)
        #temp['labels'] = labels[:,n] #[str(c) for c in labels[:,n]]
        uniqueLabels = np.unique(labels[:,n])
        #print('unique labels', uniqueLabels)

        # Number of clusters in labels, ignoring noise if present.
        n_clusters_ = len(set(labels[:,n])) - (1 if -1 in labels[:,n] else 0)
        #print('Estimated number of clusters: %d' % n_clusters_)
        #print('')


        plt.subplot(5,2,n+1)
        plt.title(name + '; Number clusters = ' + str(n_clusters_))
        for j, label in enumerate(uniqueLabels):
```

```
            #print(temp[np.where(labels[:,n] == label)[0],:].shape)
            color_ = next(palette)
            if type_ == 'features':
                '''
                plt.plot(
                    np.transpose(temp[np.where(labels[:,n] == label)[0],:]),
                    color=color_,
                    alpha=alpha
                )
                '''

                centroid = np.mean(temp[np.where(labels[:,n] == label)[0],:], axis
                plt.plot(
                    centroid,
                    color=color_,
                    linewidth=3
                )

                if xticks_ is None:
                    a=1
                else:
                    plt.xticks(range(len(centroid)), xticks_, rotation='vertical')

            elif type_ == 'scatter':
                plt.plot(
                    temp[np.where(labels[:,n] == label)[0],0],
                    temp[np.where(labels[:,n] == label)[0],1],
                    '.',
                    markersize=30,
                    color=color_,
                    alpha=alpha
                )

                centroid = np.mean(temp[np.where(labels[:,n] == label)[0],:], axis
                plt.plot(
                    centroid[0],
                    centroid[1],
                    color=color_,
                    markersize=200,
                )

        plt.tight_layout()
        plt.show()
```

Note: here, in rder to save time, I use a rando subset of the whole data table

In [29]:
```python
with timer('Embeding data'):
    features = [clmn for clmn in dfCustomerFeatures.columns if clmn not in ['identifie
    metrics = [
        'euclidean',
        'cosine',
        'correlation'
    ]
    n_sample = 20000
    selected_rows = np.random.permutation(len(dfCustomerFeatures))[:n_sample]
    objClusterAnalysis = ClusterAnalysis(dfCustomerFeatures.iloc[selected_rows][featu
    with timer('embeding PCA', n_blanks=4):
        objClusterAnalysis.embed_pca(rescaling_method = 'maxabs')
    with timer('embeding UMAP', n_blanks=4):
        objClusterAnalysis.embed_umap(n_neighbors=100, min_dist=.7, rescaling_method =
    #with timer('embeding TSNE', n_blanks=4):
    #    objClusterAnalysis.embed_tsne(n_components=3, perplexity=30, learning_rate=4
    with timer('plotting embeding projections', n_blanks=4):
        objClusterAnalysis.plot_embeddings(color_labels = dfCustomerFeatures.iloc[sele
```

```
STARTING: Embeding data
   STARTING: embeding PCA
   FINISHED in 0.00 min.

   STARTING: embeding UMAP
   FINISHED in 2.28 min.

   STARTING: plotting embeding projections
   FINISHED in 0.00 min.

FINISHED in 2.28 min.
```



Note: In the above figure, in each panel one sees the projections of the data on two of the directions of the embeding subspace. First column is for directions 1 and 2, second is for 1 and 3, and last is for 2 and 3.

Observations:

- PCA projections: There are two big clusters, which further break down into smaller less distinct ones.
  These two big clusters roughly seem to align with whether a customer defaulted or not (orange for defaulted). There are more defaulted customers in one of the clusters.
  Thus, we could run a clustering algorithm with two clusters only, then we'd run similar embedings and clustering for the cluster with more defaulted customers.
- UMAP projections with Euclidean metric: Again, clearly distinguishible two big clusters, which this time break daown into more distinct smaller ones. There seem to be four bigger sub-clusters, two under ech main cluster These two big clusters, too, roughly seem to align with whether a customer defaulted or not (orange for defaulted). There are more defaulted customers in one of the clusters.
  Also, the defaulted users are seen only in among distinct sub-clusters.
  Furthermore, we can see that some of the smaller clusters for a loop, indicating a continuous

transformations in the original feature space.
Again, we could run a clustering algorithm with two clusters only, then we'd run similar embedings and clustering for the cluster with more defaulted customers.

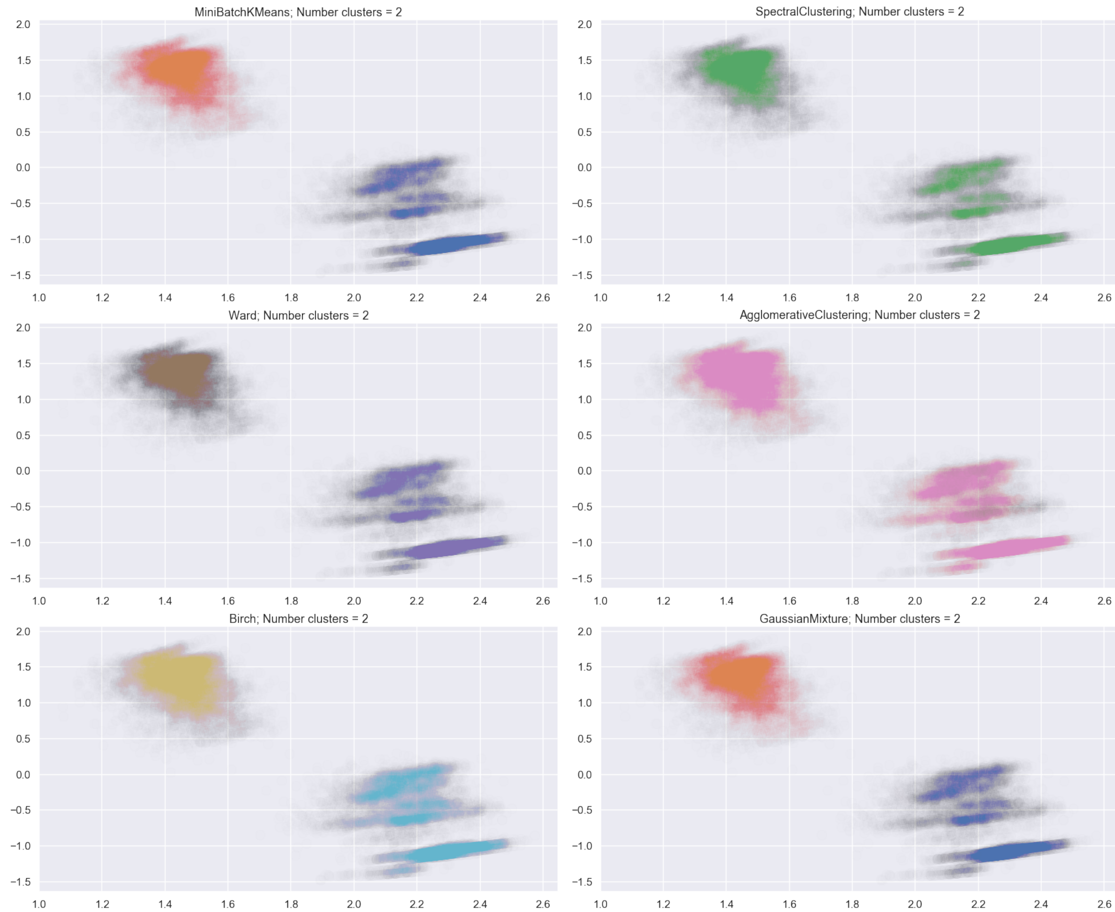
- UMAP projections with Cosine metric: Less clearly distinguishible two big clusters, which again break daown into distinct smaller ones.
Otherwise, similar to the previous case
- UMAP projections with Correlation metric: Similar to UMAP projections with Correlation metric

2.5.2 Run clustering algorithms

... and colour-code the previous projection by the labels generated by each algorithm.

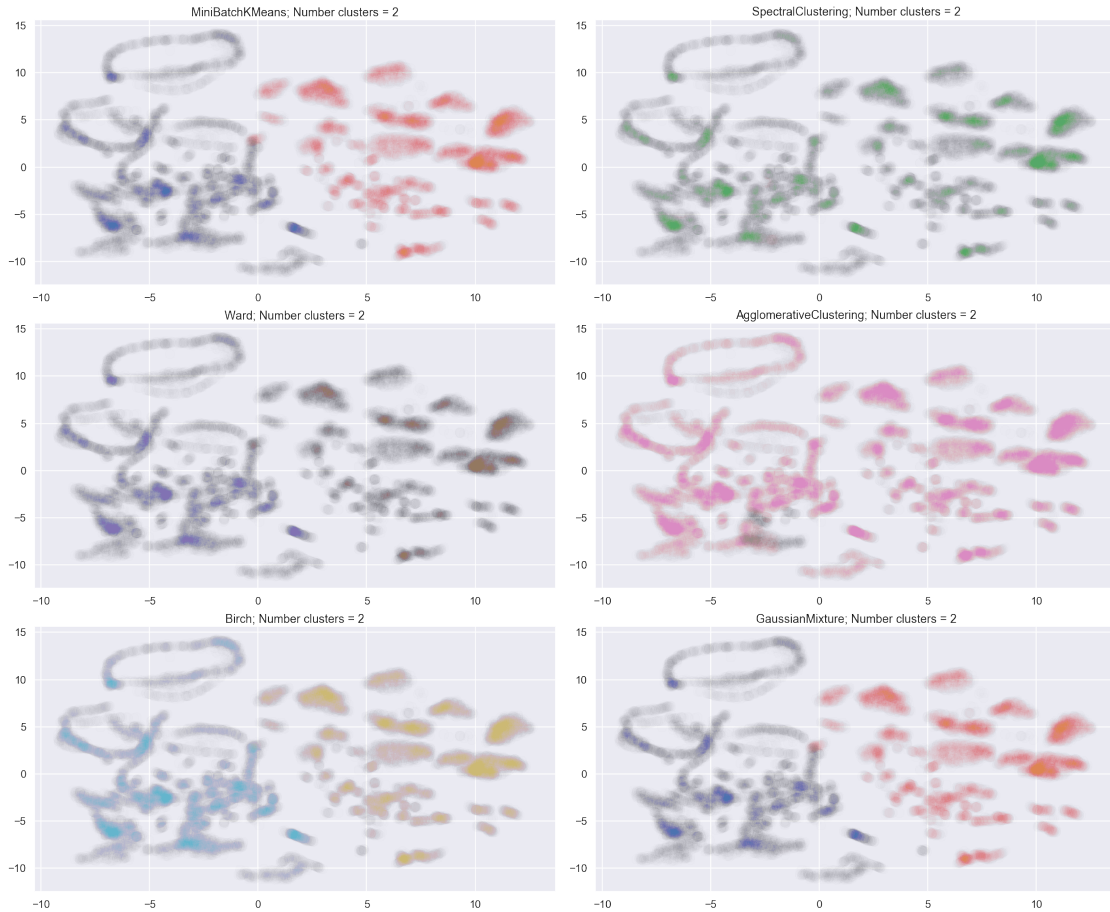
```
In [30]: with timer('Clustering customers'):
             clusteringParams = {
                 'menShift_quantile': .55,
                 'affinityPropagation_damping': .999,
                 'affinityPropagation_preference': None,
                 'affinityPropagation_max_iter': 500,
                 'ward_n_neighbors': 10,
                 'n_clusters': 2,
                 'birch_thrsld':.1,
                 'dbscan_eps': 10.01,
                 'dbscan_metric': 'euclidean'
             }
             objClusterAnalysis.cluster_data(clusteringParams, rescaling_method = 'maxabs')

         with timer('Plotting cluster results in embeded spaces'):
             with timer('onto PCA projections', n_blanks=4):
                 objClusterAnalysis.plot_clusters(objClusterAnalysis.embeded_data['pca'], objCl

             with timer('onto UMAP Euclidean projections', n_blanks=4):
                 objClusterAnalysis.plot_clusters(objClusterAnalysis.embeded_data['umap']['eucl

             with timer('onto UMAP Cosine projections', n_blanks=4):
                 objClusterAnalysis.plot_clusters(objClusterAnalysis.embeded_data['umap']['cos
```

```
STARTING: Clustering customers
0. MiniBatchKMeans                : time 0.122; number of clusters = 2
1. SpectralClustering             : time 57.233; number of clusters = 2
2. Ward                           : time 4.382; number of clusters = 2
3. AgglomerativeClustering        : time 4.543; number of clusters = 2
4. Birch                          : time 4.551; number of clusters = 2
5. GaussianMixture                : time 0.432; number of clusters = 2
FINISHED in 1.84 min.

STARTING: Plotting cluster results in embeded spaces
   STARTING: onto PCA projections
```

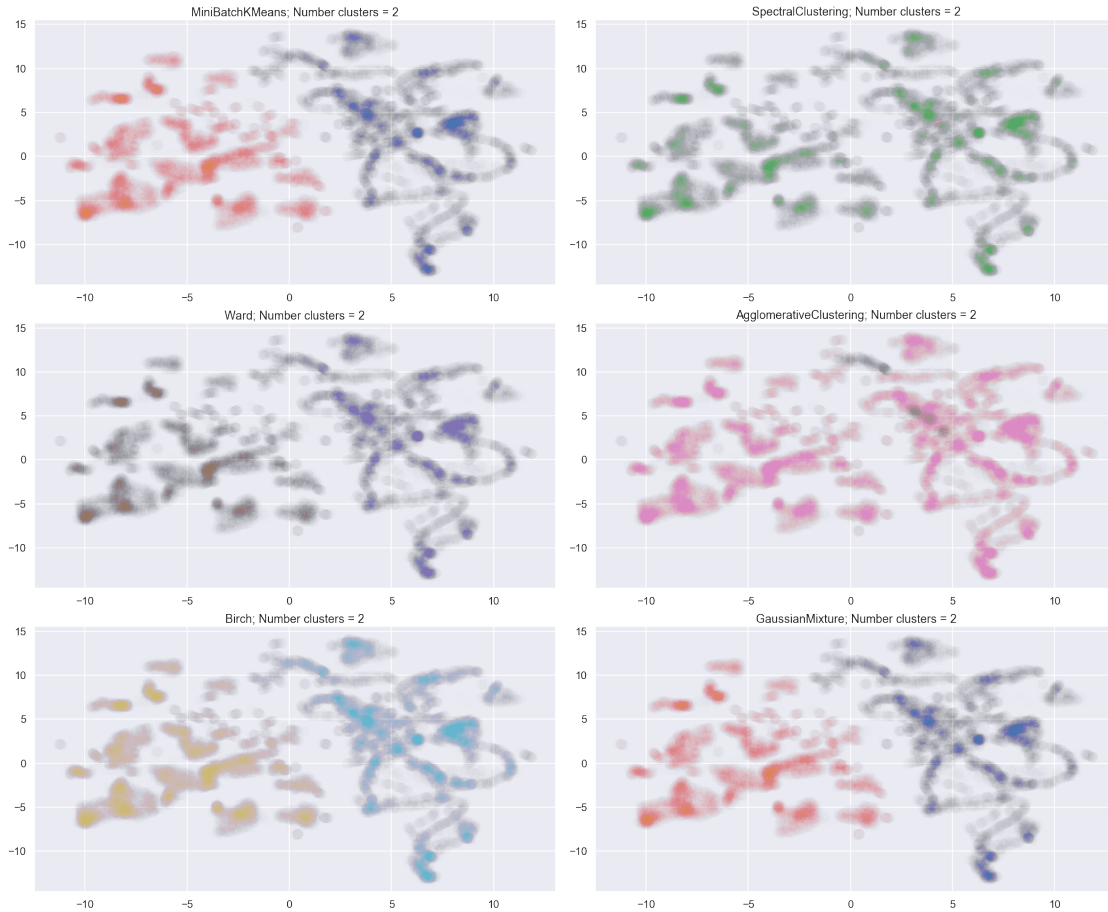


FINISHED in 0.03 min.

STARTING: onto UMAP Euclidean projections




```
FINISHED in 0.03 min.

STARTING: onto UMAP Cosine projections
```




```
FINISHED in 0.03 min.
```

```
FINISHED in 0.09 min.
```

Observations:

- Best algorithms: All algorithms but SpectralClusterring seem to produce acceptible results
- Embedings most consitent with cluster results: All embedings seem to be doing a resonable job, with PC clearly standing out and UMP with Euclidean Metric giving a nice balance of separating both bigger clustters and smaller clusters within them

2.5.3 Next steps I'd do, if I had time to do it till the end:

1. Use either MiniBatchKMeans or Ward or GaussianMixture methods results and compare wich features of the two clusters differ the most and identify these as important features for identifying customers with high risk of defaults.
2. Select the cluster with more defaulted users.

3. Run sub-space embeding and clustering algorithms on this selected data set.
4. Compare on which features samples in different clusters differ and how.
5. Go to 2.
6. Keep iterating this loop until desired level of granularity has been reached.

2.6 Customer clustering using supervised methods (classification)

```
In [31]: from sklearn import linear_model
         from sklearn import preprocessing
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.model_selection import train_test_split, cross_val_score
         from lightgbm import LGBMRegressor, LGBMClassifier
         from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, accuracy_score
         from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, GridSea
```

2.6.1 Split data into training and validations sets

Note: here, in rder to save time, I use a rando subset of the whole data table

```
In [32]: with timer('Spliting data to train and validation'):
             n_sample = 50000
             selected_rows = np.random.permutation(len(dfCustomerFeatures))[:n_sample]

             features_classifier = [clmn for clmn in dfCustomerFeatures.columns if clmn not in

             train_x, valid_x, train_y, valid_y = train_test_split(
                 dfCustomerFeatures.iloc[selected_rows][features_classifier],
                 dfCustomerFeatures.iloc[selected_rows]['defaulted'].astype(int),
                 test_size = 0.40,
                 random_state= 1984
             )
             print(4*' ' + 'train_x.shape =', train_x.shape)
             print(4*' ' + 'train_y.shape =', train_y.shape)
             print(4*' ' + 'valid_x.shape =', valid_x.shape)
             print(4*' ' + 'valid_y.shape =', valid_y.shape)
```

```
STARTING: Spliting data to train and validation
    train_x.shape = (30000, 64)
    train_y.shape = (30000,)
    valid_x.shape = (20000, 64)
    valid_y.shape = (20000,)
FINISHED in 0.00 min.
```

2.6.2 Set initial prameters for LightGBM classifier

```
In [33]: lgbmClassifierParams = {
             'objective': 'xentropy', #  'binary', 'xentlambda
```

```
        'metric': 'binary_logloss', #'binary_error', 'xentropy' , 'xentlambda'

        'bagging_fraction': 0.75,
        'bagging_freq': 10,
        'boosting_type': 'gbdt',
        #'categorical_feature': [0, 23, 24, 25, 26],
        #'early_stopping_rounds': 10,
        'feature_fraction': 0.75,
        'importance_type': 'gain', # 'split'
        'learning_rate': 0.1,
        'n_estimators': 512, # 512, 1024,
        'num_leaves': 15, #31,
        'lambda_l1': 0.01,
        'lambda_l2': 5.0,
        'verbose': 0,

        'min_child_samples': 5,
        'min_child_weight': 5.0,
        'min_split_gain': 0.5,
        'min_data_in_bin': 10,
        'min_sum_hessian_in_leaf': 10,
        'min_data_in_leaf': 20,
        'max_bin': 11,
        'xgboost_dart_mode': True,
        #'max_depth' : -1,
        #'nthread': 4,

        #'max_bin': 512,
        #'subsample_for_bin': 200,
        #'subsample': .25,
        #'subsample_freq': 1,
        #'colsample_bytree': 0.8,
        #'lambda_l1': 0.,
        #'lambda_l2': 10.,
        #'min_split_gain': 1.0,
        #'min_child_weight': 1,
        #'min_child_samples': 5,
        #'scale_pos_weight': 1,

        #'num_boost_round': 20,
        'is_unbalance': True,
        #'categorical_feature': categorical_features
    }
```

2.6.3 Do grid search for some of the prameters for LightGBM classifier

```
In [34]: gridParamsClassifier = {
        'learning_rate': [.05, .1, 1.],
```

```
        #'min_child_samples': [5, 20, 50],
        #'min_child_weight': [1., 5., 10.],
        #'min_split_gain': [.1, .5, 1.],
        'metric': ['binary_logloss', 'binary_error', 'xentropy' , 'xentlambda'],
        'importance_type': ['gain', 'split'],
        'n_estimators': [256, 512, 1024],
        #'num_leaves': [7, 9, 15],
        #'min_split_gain': [0.5, 1.0],
        #'max_depth': [1, 5, 10],
        #'feature_fraction': [0.7, 0.8, 0.9],
        #'bagging_fraction': [0.7, 0.8, 0.9],
        #'colsample_bytree' : [0.64, 0.65],
        #'subsample' : [0.25, 0.50, 0.75],
        #'bagging_freq' : [1, 5, 10, 50],
        #'lambda_l1': [0.001, 0.01, .05],
        #'lambda_l2': [0.5, 5.0, 25.],
        #'max_bin': [11, 31, 51],
        #'min_sum_hessian_in_leaf': [0.1, 1.0, 5., 50.],
        #'min_data_in_leaf': [5, 10, 50],
        #'boosting_type': ['gbdt', 'dart']
        }

    clf = LGBMClassifier(**lgbmClassifierParams)

    num_folds= 5
    stratified = False

    if stratified:
        folds = StratifiedKFold(n_splits= num_folds, shuffle=True, random_state=47)
    else:
        folds = KFold(n_splits= num_folds, shuffle=True, random_state=47)

    grid = GridSearchCV(clf, gridParamsClassifier, verbose=2, cv=folds, n_jobs=6)
    print('grid', grid)

    # Run the grid
    grid.fit(
        train_x, train_y,
        eval_set=[(train_x, train_y), (valid_x, valid_y)],
        verbose= 4,
        feature_name = features_classifier,
        early_stopping_rounds= 20,
        #categorical_feature= categorical_features
    )
```

```
grid GridSearchCV(cv=KFold(n_splits=5, random_state=47, shuffle=True),
       error_score='raise-deprecating',
       estimator=LGBMClassifier(bagging_fraction=0.75, bagging_freq=10, boosting_type='gbdt',
```

```
        class_weight=None, colsample_bytree=1.0, feature_fraction=0.75,
        importance_type='gain', is_unbalance=True, lambda_l1=0.01,
        lambda_l2=5.0, learning_rate=0.1, max_bin=11, max_depth=-1,
      ...=1.0,
        subsample_for_bin=200000, subsample_freq=0, verbose=0,
        xgboost_dart_mode=True),
       fit_params=None, iid='warn', n_jobs=6,
       param_grid={'learning_rate': [0.05, 0.1, 1.0], 'metric': ['binary_logloss', 'binary_erro
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=2)
Fitting 5 folds for each of 72 candidates, totalling 360 fits


[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  29 tasks      | elapsed:   14.5s
[Parallel(n_jobs=6)]: Done 150 tasks      | elapsed:   53.7s
[Parallel(n_jobs=6)]: Done 360 out of 360 | elapsed:  1.9min finished


Training until validation scores don't improve for 20 rounds.
[4]        training's binary_logloss: 0.221476       valid_1's binary_logloss: 0.219317
[8]        training's binary_logloss: 0.189428       valid_1's binary_logloss: 0.187547
[12]        training's binary_logloss: 0.165132       valid_1's binary_logloss: 0.163345
[16]        training's binary_logloss: 0.146988       valid_1's binary_logloss: 0.144964
[20]        training's binary_logloss: 0.133848       valid_1's binary_logloss: 0.13195
[24]        training's binary_logloss: 0.125182       valid_1's binary_logloss: 0.123539
[28]        training's binary_logloss: 0.11693       valid_1's binary_logloss: 0.115171
[32]        training's binary_logloss: 0.109081       valid_1's binary_logloss: 0.107495
[36]        training's binary_logloss: 0.103531       valid_1's binary_logloss: 0.102456
[40]        training's binary_logloss: 0.0986961       valid_1's binary_logloss: 0.09783
[44]        training's binary_logloss: 0.0947491       valid_1's binary_logloss: 0.0941831
[48]        training's binary_logloss: 0.0916795       valid_1's binary_logloss: 0.0913966
[52]        training's binary_logloss: 0.0886457       valid_1's binary_logloss: 0.0884472
[56]        training's binary_logloss: 0.0861759       valid_1's binary_logloss: 0.0863032
[60]        training's binary_logloss: 0.0842436       valid_1's binary_logloss: 0.0845307
[64]        training's binary_logloss: 0.0820139       valid_1's binary_logloss: 0.0823364
[68]        training's binary_logloss: 0.0805659       valid_1's binary_logloss: 0.0813648
[72]        training's binary_logloss: 0.078911       valid_1's binary_logloss: 0.0798436
[76]        training's binary_logloss: 0.0777085       valid_1's binary_logloss: 0.0788264
[80]        training's binary_logloss: 0.0765763       valid_1's binary_logloss: 0.077914
[84]        training's binary_logloss: 0.0755249       valid_1's binary_logloss: 0.0771205
[88]        training's binary_logloss: 0.0745966       valid_1's binary_logloss: 0.0765632
[92]        training's binary_logloss: 0.0737024       valid_1's binary_logloss: 0.0758952
[96]        training's binary_logloss: 0.0728745       valid_1's binary_logloss: 0.0752475
[100]        training's binary_logloss: 0.0721177       valid_1's binary_logloss: 0.0747582
[104]        training's binary_logloss: 0.0714687       valid_1's binary_logloss: 0.0743927
[108]        training's binary_logloss: 0.0708465       valid_1's binary_logloss: 0.0739629
[112]        training's binary_logloss: 0.0702505       valid_1's binary_logloss: 0.0735391
```

```
[116]       training's binary_logloss: 0.0697876      valid_1's binary_logloss: 0.0733292
[120]       training's binary_logloss: 0.0692635      valid_1's binary_logloss: 0.0729811
[124]       training's binary_logloss: 0.0687147      valid_1's binary_logloss: 0.072688
[128]       training's binary_logloss: 0.0682666      valid_1's binary_logloss: 0.0725064
[132]       training's binary_logloss: 0.0678309      valid_1's binary_logloss: 0.0723409
[136]       training's binary_logloss: 0.0674335      valid_1's binary_logloss: 0.0722154
[140]       training's binary_logloss: 0.0670037      valid_1's binary_logloss: 0.0720368
[144]       training's binary_logloss: 0.0666055      valid_1's binary_logloss: 0.0718031
[148]       training's binary_logloss: 0.0662053      valid_1's binary_logloss: 0.0716191
[152]       training's binary_logloss: 0.0658261      valid_1's binary_logloss: 0.0714225
[156]       training's binary_logloss: 0.0654859      valid_1's binary_logloss: 0.0712934
[160]       training's binary_logloss: 0.0652007      valid_1's binary_logloss: 0.0711845
[164]       training's binary_logloss: 0.0648966      valid_1's binary_logloss: 0.071202
[168]       training's binary_logloss: 0.0645507      valid_1's binary_logloss: 0.0711241
[172]       training's binary_logloss: 0.0642628      valid_1's binary_logloss: 0.0710779
[176]       training's binary_logloss: 0.0639812      valid_1's binary_logloss: 0.0709004
[180]       training's binary_logloss: 0.0636887      valid_1's binary_logloss: 0.0707854
[184]       training's binary_logloss: 0.063409      valid_1's binary_logloss: 0.0707319
[188]       training's binary_logloss: 0.0631403      valid_1's binary_logloss: 0.0707272
[192]       training's binary_logloss: 0.0628833      valid_1's binary_logloss: 0.0707078
[196]       training's binary_logloss: 0.0626216      valid_1's binary_logloss: 0.0706989
[200]       training's binary_logloss: 0.0623916      valid_1's binary_logloss: 0.0707027
[204]       training's binary_logloss: 0.062192      valid_1's binary_logloss: 0.0707155
[208]       training's binary_logloss: 0.0619751      valid_1's binary_logloss: 0.0706382
[212]       training's binary_logloss: 0.0617194      valid_1's binary_logloss: 0.0705728
[216]       training's binary_logloss: 0.0614779      valid_1's binary_logloss: 0.070546
[220]       training's binary_logloss: 0.0612675      valid_1's binary_logloss: 0.0704969
[224]       training's binary_logloss: 0.061036      valid_1's binary_logloss: 0.0704879
[228]       training's binary_logloss: 0.0607817      valid_1's binary_logloss: 0.070482
[232]       training's binary_logloss: 0.0605832      valid_1's binary_logloss: 0.0704828
[236]       training's binary_logloss: 0.0603811      valid_1's binary_logloss: 0.0704251
[240]       training's binary_logloss: 0.0601705      valid_1's binary_logloss: 0.0703963
[244]       training's binary_logloss: 0.0599377      valid_1's binary_logloss: 0.0704188
[248]       training's binary_logloss: 0.0597796      valid_1's binary_logloss: 0.0704207
[252]       training's binary_logloss: 0.0595481      valid_1's binary_logloss: 0.0703991
[256]       training's binary_logloss: 0.0592917      valid_1's binary_logloss: 0.0703368
Did not meet early stopping. Best iteration is:
[256]       training's binary_logloss: 0.0592917      valid_1's binary_logloss: 0.0703368
```

```
Out[34]: GridSearchCV(cv=KFold(n_splits=5, random_state=47, shuffle=True),
                error_score='raise-deprecating',
                estimator=LGBMClassifier(bagging_fraction=0.75, bagging_freq=10, boosting_type=
                 class_weight=None, colsample_bytree=1.0, feature_fraction=0.75,
                 importance_type='gain', is_unbalance=True, lambda_l1=0.01,
                 lambda_l2=5.0, learning_rate=0.1, max_bin=11, max_depth=-1,
               ...=1.0,
                 subsample_for_bin=200000, subsample_freq=0, verbose=0,
```

```
                 xgboost_dart_mode=True),
                fit_params=None, iid='warn', n_jobs=6,
                param_grid={'learning_rate': [0.05, 0.1, 1.0], 'metric': ['binary_logloss', 'bi
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=2)
```

2.6.4 Fit a new classifier with the best parameters from the grid search

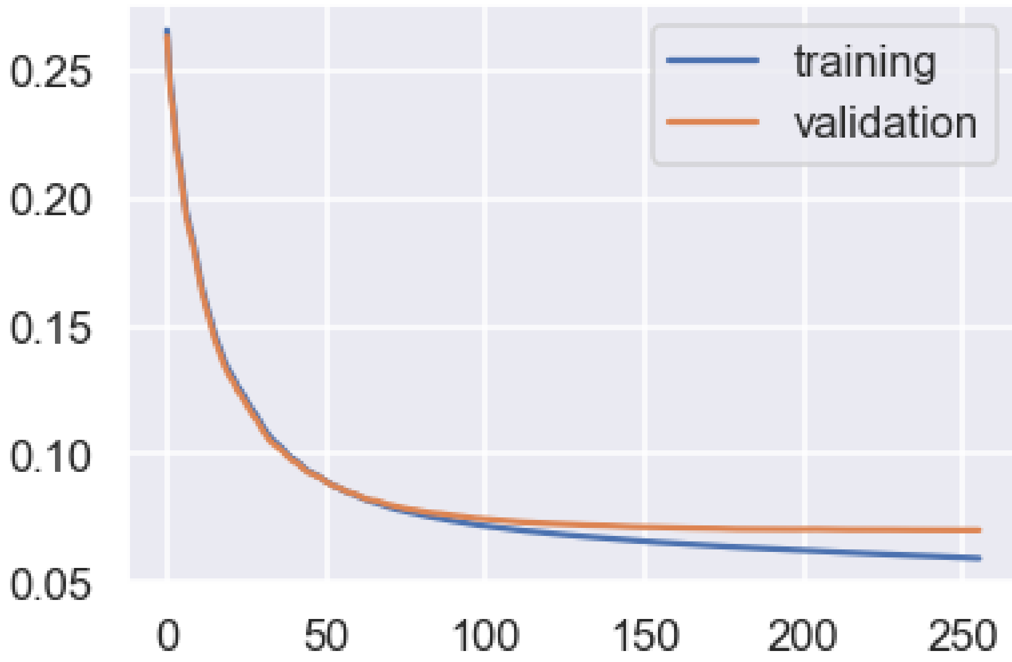
```
In [35]: for param,val in grid.best_params_.items():
                 lgbmClassifierParams[param] = val
         print('BEST MODEL PARAMETERS:')
         [print(key,':',val) for key,val in lgbmClassifierParams.items()]
         print()
         clf = LGBMClassifier(**lgbmClassifierParams)
         clf.fit(
             train_x, train_y,
             eval_set=[(train_x, train_y), (valid_x, valid_y)],
             verbose= 4,
             feature_name = features_classifier,
             early_stopping_rounds= 20,
             #categorical_feature= categorical_features
         )

         with timer('Saving model', n_blanks=4):
             # save model to file
             clf.booster_.save_model(
                 os.path.join(
                     PATHS['WORKSPACE'],
                     'clf_predict_01.txt'))

             # dump model with pickle
             with open(
                 os.path.join(
                     PATHS['WORKSPACE'],
                     'clf_predict_01.pkl'), 'wb') as fout:
                 pickle.dump(clf, fout)

         plt.close('all')
         #plt.plot(clf.evals_result_['training'][lgbmParams['metric']], label='training')
         plt.plot(clf.evals_result_['training'][lgbmClassifierParams['metric']], label='traini
         plt.plot(clf.evals_result_['valid_1'][lgbmClassifierParams['metric']], label='validat
         plt.legend()
```

```
BEST MODEL PARAMETERS:
lambda_l1 : 0.01
metric : binary_logloss
xgboost_dart_mode : True
min_child_weight : 5.0
```

```
objective : xentropy
importance_type : gain
min_split_gain : 0.5
verbose : 0
min_data_in_bin : 10
bagging_freq : 10
bagging_fraction : 0.75
is_unbalance : True
min_data_in_leaf : 20
min_sum_hessian_in_leaf : 10
learning_rate : 0.05
max_bin : 11
boosting_type : gbdt
n_estimators : 256
feature_fraction : 0.75
min_child_samples : 5
num_leaves : 15
lambda_l2 : 5.0

Training until validation scores don't improve for 20 rounds.
[4]	training's binary_logloss: 0.221476	valid_1's binary_logloss: 0.219317
[8]	training's binary_logloss: 0.189428	valid_1's binary_logloss: 0.187547
[12]	training's binary_logloss: 0.165132	valid_1's binary_logloss: 0.163345
[16]	training's binary_logloss: 0.146988	valid_1's binary_logloss: 0.144964
[20]	training's binary_logloss: 0.133848	valid_1's binary_logloss: 0.13195
[24]	training's binary_logloss: 0.125182	valid_1's binary_logloss: 0.123539
[28]	training's binary_logloss: 0.11693	valid_1's binary_logloss: 0.115171
[32]	training's binary_logloss: 0.109081	valid_1's binary_logloss: 0.107495
[36]	training's binary_logloss: 0.103531	valid_1's binary_logloss: 0.102456
[40]	training's binary_logloss: 0.0986961	valid_1's binary_logloss: 0.09783
[44]	training's binary_logloss: 0.0947491	valid_1's binary_logloss: 0.0941831
[48]	training's binary_logloss: 0.0916795	valid_1's binary_logloss: 0.0913966
[52]	training's binary_logloss: 0.0886457	valid_1's binary_logloss: 0.0884472
[56]	training's binary_logloss: 0.0861759	valid_1's binary_logloss: 0.0863032
[60]	training's binary_logloss: 0.0842436	valid_1's binary_logloss: 0.0845307
[64]	training's binary_logloss: 0.0820139	valid_1's binary_logloss: 0.0823364
[68]	training's binary_logloss: 0.0805659	valid_1's binary_logloss: 0.0813648
[72]	training's binary_logloss: 0.078911	valid_1's binary_logloss: 0.0798436
[76]	training's binary_logloss: 0.0777085	valid_1's binary_logloss: 0.0788264
[80]	training's binary_logloss: 0.0765763	valid_1's binary_logloss: 0.077914
[84]	training's binary_logloss: 0.0755249	valid_1's binary_logloss: 0.0771205
[88]	training's binary_logloss: 0.0745966	valid_1's binary_logloss: 0.0765632
[92]	training's binary_logloss: 0.0737024	valid_1's binary_logloss: 0.0758952
[96]	training's binary_logloss: 0.0728745	valid_1's binary_logloss: 0.0752475
[100]	training's binary_logloss: 0.0721177	valid_1's binary_logloss: 0.0747582
[104]	training's binary_logloss: 0.0714687	valid_1's binary_logloss: 0.0743927
[108]	training's binary_logloss: 0.0708465	valid_1's binary_logloss: 0.0739629
[112]	training's binary_logloss: 0.0702505	valid_1's binary_logloss: 0.0735391
```

```
[116]	training's binary_logloss: 0.0697876	valid_1's binary_logloss: 0.0733292
[120]	training's binary_logloss: 0.0692635	valid_1's binary_logloss: 0.0729811
[124]	training's binary_logloss: 0.0687147	valid_1's binary_logloss: 0.072688
[128]	training's binary_logloss: 0.0682666	valid_1's binary_logloss: 0.0725064
[132]	training's binary_logloss: 0.0678309	valid_1's binary_logloss: 0.0723409
[136]	training's binary_logloss: 0.0674335	valid_1's binary_logloss: 0.0722154
[140]	training's binary_logloss: 0.0670037	valid_1's binary_logloss: 0.0720368
[144]	training's binary_logloss: 0.0666055	valid_1's binary_logloss: 0.0718031
[148]	training's binary_logloss: 0.0662053	valid_1's binary_logloss: 0.0716191
[152]	training's binary_logloss: 0.0658261	valid_1's binary_logloss: 0.0714225
[156]	training's binary_logloss: 0.0654859	valid_1's binary_logloss: 0.0712934
[160]	training's binary_logloss: 0.0652007	valid_1's binary_logloss: 0.0711845
[164]	training's binary_logloss: 0.0648966	valid_1's binary_logloss: 0.071202
[168]	training's binary_logloss: 0.0645507	valid_1's binary_logloss: 0.0711241
[172]	training's binary_logloss: 0.0642628	valid_1's binary_logloss: 0.0710779
[176]	training's binary_logloss: 0.0639812	valid_1's binary_logloss: 0.0709004
[180]	training's binary_logloss: 0.0636887	valid_1's binary_logloss: 0.0707854
[184]	training's binary_logloss: 0.063409	valid_1's binary_logloss: 0.0707319
[188]	training's binary_logloss: 0.0631403	valid_1's binary_logloss: 0.0707272
[192]	training's binary_logloss: 0.0628833	valid_1's binary_logloss: 0.0707078
[196]	training's binary_logloss: 0.0626216	valid_1's binary_logloss: 0.0706989
[200]	training's binary_logloss: 0.0623916	valid_1's binary_logloss: 0.0707027
[204]	training's binary_logloss: 0.062192	valid_1's binary_logloss: 0.0707155
[208]	training's binary_logloss: 0.0619751	valid_1's binary_logloss: 0.0706382
[212]	training's binary_logloss: 0.0617194	valid_1's binary_logloss: 0.0705728
[216]	training's binary_logloss: 0.0614779	valid_1's binary_logloss: 0.070546
[220]	training's binary_logloss: 0.0612675	valid_1's binary_logloss: 0.0704969
[224]	training's binary_logloss: 0.061036	valid_1's binary_logloss: 0.0704879
[228]	training's binary_logloss: 0.0607817	valid_1's binary_logloss: 0.070482
[232]	training's binary_logloss: 0.0605832	valid_1's binary_logloss: 0.0704828
[236]	training's binary_logloss: 0.0603811	valid_1's binary_logloss: 0.0704251
[240]	training's binary_logloss: 0.0601705	valid_1's binary_logloss: 0.0703963
[244]	training's binary_logloss: 0.0599377	valid_1's binary_logloss: 0.0704188
[248]	training's binary_logloss: 0.0597796	valid_1's binary_logloss: 0.0704207
[252]	training's binary_logloss: 0.0595481	valid_1's binary_logloss: 0.0703991
[256]	training's binary_logloss: 0.0592917	valid_1's binary_logloss: 0.0703368
Did not meet early stopping. Best iteration is:
[256]	training's binary_logloss: 0.0592917	valid_1's binary_logloss: 0.0703368
    STARTING: Saving model
    FINISHED in 0.00 min.
```

```
Out[35]: <matplotlib.legend.Legend at 0x29e822d8eb8>
```

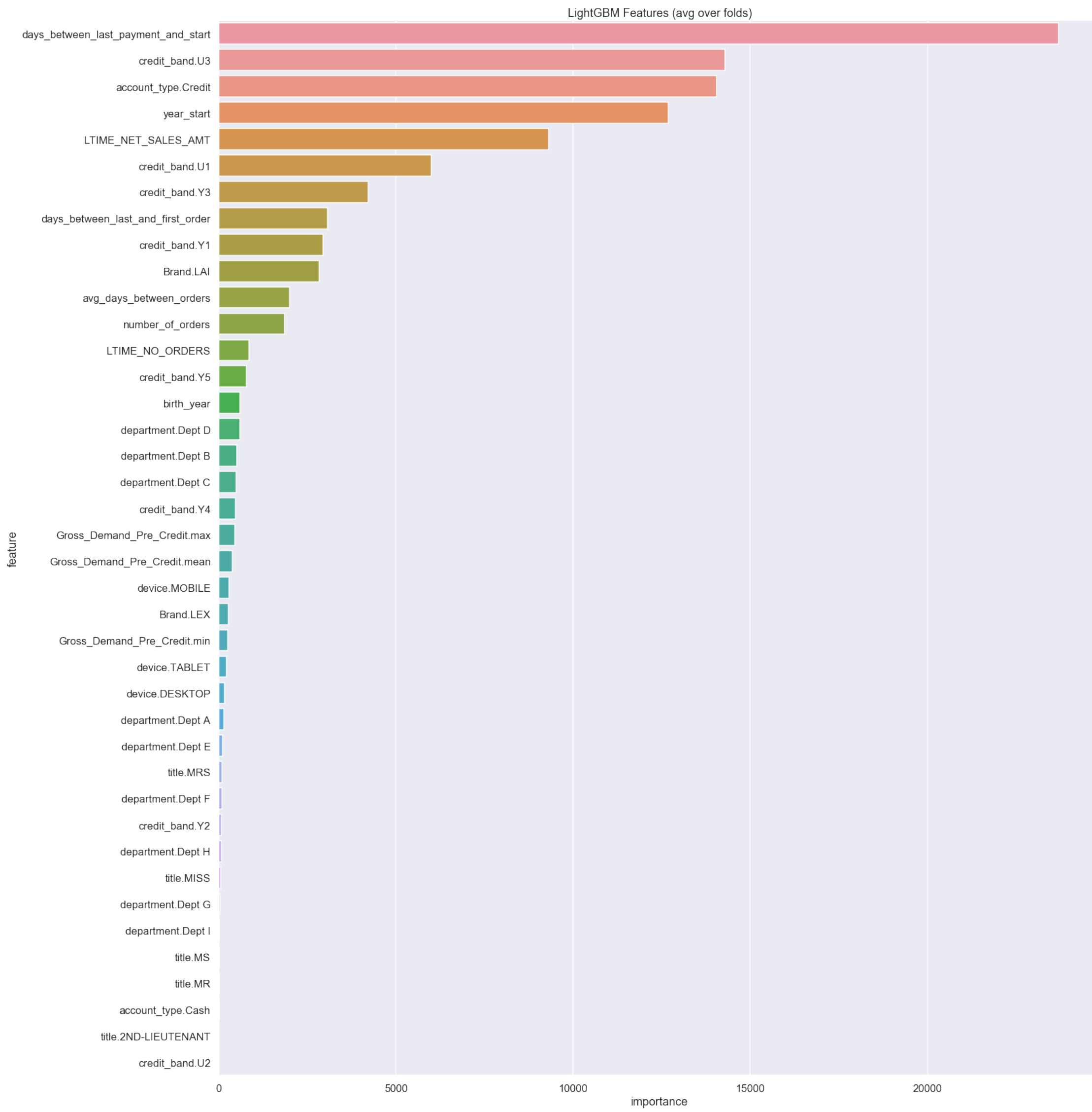


2.6.5 Display feature importances, based on the trained model

```
In [36]: def display_importances(feature_importance_df_):
             cols = feature_importance_df_[["feature", "importance"]].groupby("feature").mean(
             best_features = feature_importance_df_.loc[feature_importance_df_.feature.isin(co
             plt.close('all')
             plt.figure(figsize=(24, 24))
             sns.barplot(x="importance", y="feature", data=best_features.sort_values(by="impor
             plt.title('LightGBM Features (avg over folds)')
             plt.tight_layout()
             plt.show()
```

```
In [37]: if False:
             feature_imp = pd.DataFrame(sorted(zip(clf.feature_importances_, train_x.columns))

             plt.figure(figsize=(20, 10))
             sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascen
             plt.title('LightGBM Features (avg over folds)')
             plt.tight_layout()
             plt.show()
         else:
             feature_importance_df = pd.DataFrame()
             feature_importance_df["feature"] = features_classifier
             feature_importance_df["importance"] = clf.feature_importances_
```

```
    display_importances(feature_importance_df)

sumFI = feature_importance_df['importance'].sum()
feature_importance_df['importance'] /= sumFI
feature_importance_df.sort_values(by='importance', ascending=False)[:10]
```




```
Out[37]:                                  feature  importance
        3   days_between_last_payment_and_start    0.227785
        7                        credit_band.U3    0.137253
        52                  account_type.Credit    0.134956
        4                            year_start    0.121871
        0                   LTIME_NET_SALES_AMT    0.089386
        5                        credit_band.U1    0.057595
```

```
18                        credit_band.Y3    0.040513
43    days_between_last_and_first_order    0.029602
16                        credit_band.Y1    0.028241
46                             Brand.LAI    0.027193
```

2.6.6 Check model performance

Since this is highly imbalanced data, aka. there are many more customers who do not default than defaulted (roughly 9 to 1), I'll check how well the trained model performs against the null model of predicting only non-default

In [38]:
```
y_pred = clf.predict(valid_x)
print('The accuracy of prediction is:', accuracy_score(valid_y, y_pred))
print('The roc_auc_score of prediction is:', roc_auc_score(valid_y, y_pred))
print('The acccuracy of always predicting one class is:', max(valid_y.mean(), 1 - vali
```

```
The accuracy of prediction is: 0.9737
The roc_auc_score of prediction is: 0.8836433340559903
The acccuracy of always predicting one class is: 0.92035
```

Observation: So, the model performes better than the null model, but it still might need class-balancing the data.

2.6.7 Balance the data set:

I'll check the probabilityies of deaulting that the model produces for each sample, then play with the threshold in such a way that when the model predicts non-defaulted it's accurate for almost all of the cases (high precision for non-defaulted) and is only about 5% of the times wrong when it says defaulted (high recall for defaulted).

Thus in the set of predicted defaults, there will be almost all actual defaulted customers, plus only a fraction of all non-deaulted ones.

Then I'll train a second model on this second set alone.

The final system will be omposed of two stages:

1. First, filter out the customers for which we are fairly confident they will not default.
2. Next, focuse on the remaining ones.

Run the classifier for various thresholds and monitor recal, precision and f1 score

In [39]:
```
#====================
#  Confusion Matrix
#    rows: actual
#    cols: predicted
#====================


tblPredictions_Valid = pd.DataFrame({'actual': valid_y})
tblPredictions_Valid['probability'] = clf.predict_proba(valid_x, num_iteration=clf.bes
```

```
grid_thrslds = np.linspace(0.001,0.1, 10)
grid_thrslds
for thrshld in grid_thrslds:
    tblPredictions_Valid['predicted'] = (tblPredictions_Valid['probability'] >= thrshl
    cm = confusion_matrix(tblPredictions_Valid['actual'], tblPredictions_Valid['predic
    recall = np.array([cm[0,0]/np.sum(cm[0,:]), cm[1,1]/np.sum(cm[1,:])])
    precission = np.array([cm[0,0]/np.sum(cm[:,0]), cm[1,1]/np.sum(cm[:,1])])
    f1 = 2 * (precission * recall) / (precission + recall)
    print('\nThreshold =',thrshld)
    print('recall:',recall, np.sqrt(recall[0]*recall[1]))
    print('precission:', precission, np.sqrt(precission[0]*precission[1]))
    print('F1:', f1, np.sqrt(f1[0]*f1[1]))
    print('cm[:,0]/np.sum(cm,axis=1)', cm[:,0]/np.sum(cm,axis=1))
    print('cm[:,0]',cm[:,0])
```

```
Threshold = 0.001
recall: [0.41842777 1.        ] 0.6468599323217078
precission: [1.         0.12953326] 0.35990729006264127
F1: [0.58998813 0.22935714] 0.3678559337916082
cm[:,0]/np.sum(cm,axis=1) [0.41842777 0.        ]
cm[:,0] [7702    0]

Threshold = 0.012
recall: [0.78774379 0.98618958] 0.8813992965857973
precission: [0.99848506 0.2867835 ] 0.5351159098914741
F1: [0.88068268 0.44435016] 0.6255649386733703
cm[:,0]/np.sum(cm,axis=1) [0.78774379 0.01381042]
cm[:,0] [14500    22]

Threshold = 0.023000000000000003
recall: [0.85880372 0.9742624 ] 0.9147131614512369
precission: [0.99741309 0.37388581] 0.6106706151516869
F1: [0.92293321 0.54038997] 0.7062179911222087
cm[:,0]/np.sum(cm,axis=1) [0.85880372 0.0257376 ]
cm[:,0] [15808    41]

Threshold = 0.034
recall: [0.89335579 0.96421846] 0.9281110595580282
precission: [0.99654566 0.43898257] 0.6614122565545911
F1: [0.94213361 0.60329929] 0.7539154726044324
cm[:,0]/np.sum(cm,axis=1) [0.89335579 0.03578154]
cm[:,0] [16444    57]

Threshold = 0.045000000000000005
recall: [0.91302222 0.95480226] 0.9336785735998363
precission: [0.99573409 0.4871877 ] 0.6964979555939649
```

```
F1: [0.95258608 0.64517497] 0.7839545279970394
cm[:,0]/np.sum(cm,axis=1) [0.91302222 0.04519774]
cm[:,0] [16806    72]

Threshold = 0.05600000000000001
recall: [0.92660401 0.94978029] 0.9381205805035248
precission: [0.99533147 0.52828212] 0.7251315879274832
F1: [0.95973891 0.67893202] 0.8072158772701432
cm[:,0]/np.sum(cm,axis=1) [0.92660401 0.05021971]
cm[:,0] [17056    80]

Threshold = 0.067
recall: [0.93681752 0.94224733] 0.939528501017373
precission: [0.99469312 0.56343844] 0.7486309775770266
F1: [0.96488823 0.70519145] 0.824882373011152
cm[:,0]/np.sum(cm,axis=1) [0.93681752 0.05775267]
cm[:,0] [17244    92]

Threshold = 0.07800000000000001
recall: [0.94474928 0.94161959] 0.9431831347980971
precission: [0.99468055 0.59594756] 0.7699204124375624
F1: [0.96907216 0.72992701] 0.8410421780254836
cm[:,0]/np.sum(cm,axis=1) [0.94474928 0.05838041]
cm[:,0] [17390    93]

Threshold = 0.08900000000000001
recall: [0.95121421 0.93659761] 0.9438776201838109
precission: [0.99426462 0.62426778] 0.7878371474832838
F1: [0.9722631  0.74918403] 0.8534658681185733
cm[:,0]/np.sum(cm,axis=1) [0.95121421 0.06340239]
cm[:,0] [17509   101]

Threshold = 0.1
recall: [0.95528875 0.93157564] 0.94335768978518
precission: [0.99383937 0.64325964] 0.7995603547342092
F1: [0.97418283 0.76102564] 0.8610331638448737
cm[:,0]/np.sum(cm,axis=1) [0.95528875 0.06842436]
cm[:,0] [17584   109]
```

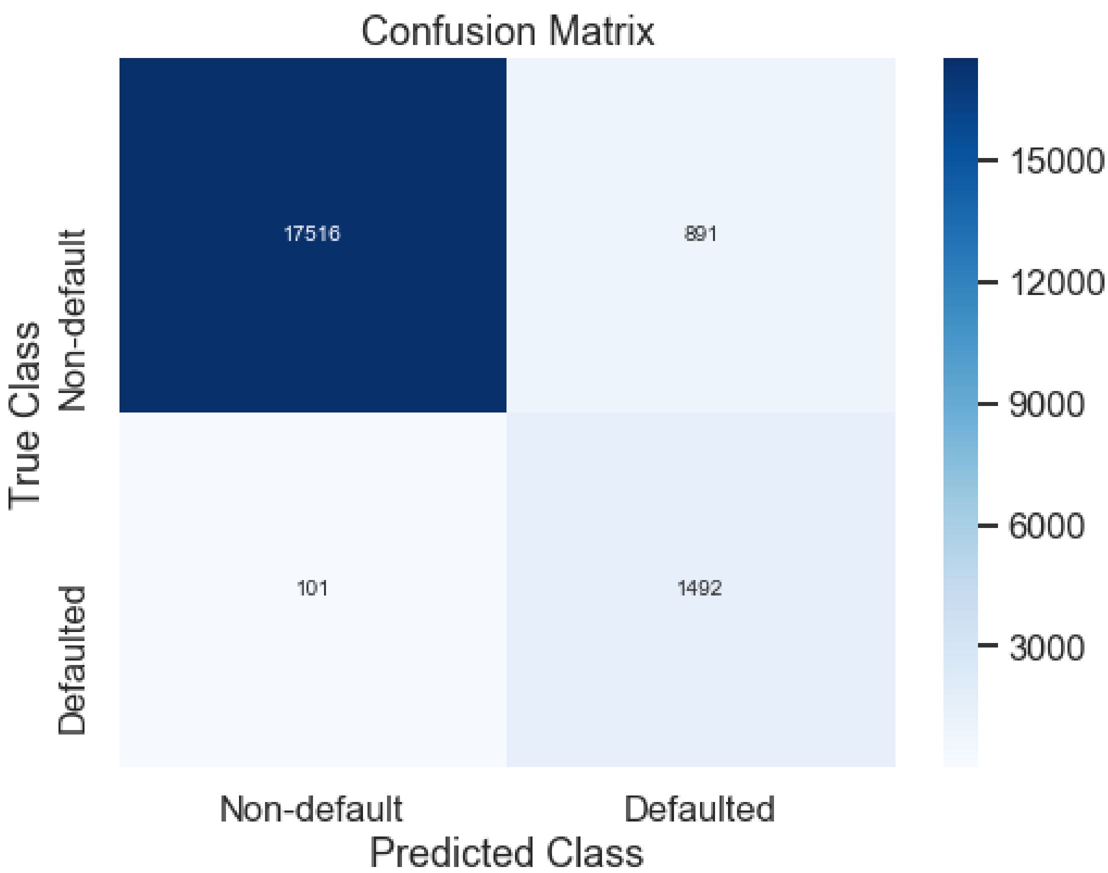
Select athreshold for the clasifier that will wilter out the customers for which we are confident they will not default

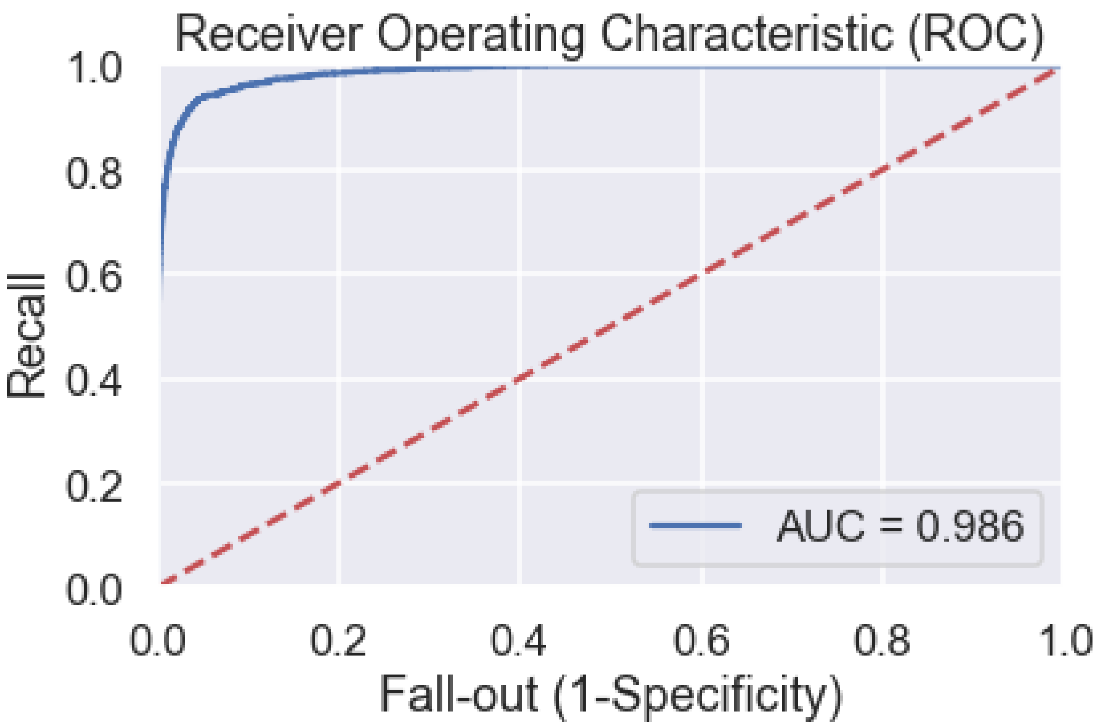
In [43]:
```
THRSHLD = 0.090

#tblVisitorFeatures = copy.deepcopy(tblVisitorFeatures_orgnl)
tblPredictions_Train = pd.DataFrame({'actual': train_y})
tblPredictions_Train['probability'] = clf.predict_proba(train_x, num_iteration=clf.bes
```

```
    tblPredictions_Train['predicted'] = (tblPredictions_Train['probability'] >= THRSHLD).a

    tblPredictions_Valid = pd.DataFrame({'actual': valid_y})
    tblPredictions_Valid['probability'] = clf.predict_proba(valid_x, num_iteration=clf.bes
    tblPredictions_Valid['predicted'] = (tblPredictions_Valid['probability'] >= THRSHLD).a

    cm = confusion_matrix(tblPredictions_Valid['actual'], tblPredictions_Valid['predicted
    recall = np.array([cm[0,0]/np.sum(cm[0,:]), cm[1,1]/np.sum(cm[1,:])])
    precission = np.array([cm[0,0]/np.sum(cm[:,0]), cm[1,1]/np.sum(cm[:,1])])
    f1 = 2 * (precission * recall) / (precission + recall)
    print('\n\nThreshold =',THRSHLD)
    print('recall:',recall, np.sqrt(recall[0]*recall[1]))
    print('precission:', precission, np.sqrt(precission[0]*precission[1]))
    print('F1:', f1, np.sqrt(f1[0]*f1[1]))
    print('cm[:,0]/np.sum(cm,axis=1)', cm[:,0]/np.sum(cm,axis=1))
    print('cm[:,0]',cm[:,0])

    #Print Confusion Matrix
    plt.figure()
    labels = ['Non-default', 'Defaulted']
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, xticklabels = labels, yticklabels = labels, annot = True, fmt='d', cma
    plt.title('Confusion Matrix')
    plt.ylabel('True Class')
    plt.xlabel('Predicted Class')
    plt.show()

    plt.figure()
    false_positive_rate, recall_, thresholds = roc_curve(tblPredictions_Valid['actual'], t
    roc_auc = auc(false_positive_rate, recall_)
    plt.title('Receiver Operating Characteristic (ROC)')
    plt.plot(false_positive_rate, recall_, 'b', label = 'AUC = %0.3f' %roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0,1], [0,1], 'r--')
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.0])
    plt.ylabel('Recall')
    plt.xlabel('Fall-out (1-Specificity)')

    plt.tight_layout()
    plt.show()
```

```
Threshold = 0.09
recall: [0.9515945  0.93659761] 0.9440662798188143
precission: [0.9942669  0.62610155] 0.7889943286614143
F1: [0.9724628  0.75050302] 0.8543045524470365
```

```
cm[:,0]/np.sum(cm,axis=1) [0.9515945  0.06340239]
cm[:,0] [17516   101]


<Figure size 432x288 with 0 Axes>
```





Observation: The set of predicted defaults seems much more balanced now.

2.6.8 Select a new set of training and testing

```
In [44]: train_x_2 = train_x.loc[tblPredictions_Train[tblPredictions_Train['predicted'] == 1].i
         train_y_2 = train_y[tblPredictions_Train[tblPredictions_Train['predicted'] == 1].inde
         tblPredictions_Train_2 = tblPredictions_Train[tblPredictions_Train['predicted'] == 1]

         valid_x_2 = valid_x.loc[tblPredictions_Valid[tblPredictions_Valid['predicted'] == 1].i
         valid_y_2 = valid_y[tblPredictions_Valid[tblPredictions_Valid['predicted'] == 1].inde
         tblPredictions_Valid_2 = tblPredictions_Valid[tblPredictions_Valid['predicted'] == 1]

         # sanity check
         print(len(train_y_2)/len(train_y),
               len(valid_y_2)/len(valid_y))
         print(len(train_y_2[tblPredictions_Train_2['actual'] == 1])/len(train_y_2),
               len(valid_y_2[tblPredictions_Valid_2['actual'] == 1])/len(valid_y_2))
```

```
0.1197 0.11915
0.6427179058758006 0.6261015526647083
```

2.6.9 New grid search and a second model:

Starting with the parameters for the latest model, do grid serch for a new model on the new subset of data

In [45]:
```
gridParamsClassifier = {
    'learning_rate': [.05, .1, 1.],
    #'min_child_samples': [5, 20, 50],
    #'min_child_weight': [1., 5., 10.],
    #'min_split_gain': [.1, .5, 1.],
    'metric': ['binary_logloss', 'binary_error', 'xentropy' , 'xentlambda'],
    'importance_type': ['gain', 'split'],
    'n_estimators': [256, 512, 1024],
    #'num_leaves': [7, 9, 15],
    #'min_split_gain': [0.5, 1.0],
    #'max_depth': [1, 5, 10],
    #'feature_fraction': [0.7, 0.8, 0.9],
    #'bagging_fraction': [0.7, 0.8, 0.9],
    #'colsample_bytree' : [0.64, 0.65],
    #'subsample' : [0.25, 0.50, 0.75],
    #'bagging_freq' : [1, 5, 10, 50],
    #'lambda_l1': [0.001, 0.01, .05],
    #'lambda_l2': [0.5, 5.0, 25.],
    #'max_bin': [11, 31, 51],
    #'min_sum_hessian_in_leaf': [0.1, 1.0, 5., 50.],
    #'min_data_in_leaf': [5, 10, 50],
    #'boosting_type': ['gbdt', 'dart']
    }

clf_2 = LGBMClassifier(**lgbmClassifierParams)

num_folds= 5
stratified = False

if stratified:
    folds = StratifiedKFold(n_splits= num_folds, shuffle=True, random_state=47)
else:
    folds = KFold(n_splits= num_folds, shuffle=True, random_state=47)

grid = GridSearchCV(clf_2, gridParamsClassifier, verbose=2, cv=folds, n_jobs=6)
print('grid', grid)

# Run the grid
grid.fit(
    train_x_2, train_y_2,
    eval_set=[(train_x_2, train_y_2), (valid_x_2, valid_y_2)],
    verbose= 4,
    feature_name = features_classifier,
    early_stopping_rounds= 20,
```

```
            #categorical_feature= categorical_features
        )

        # get the best params
        for param,val in grid.best_params_.items():
                lgbmClassifierParams[param] = val
        print('BEST MODEL PARAMETERS:')
        [print(key,':',val) for key,val in lgbmClassifierParams.items()]
        print()

        #fit a new model with best params
        clf_2 = LGBMClassifier(**lgbmClassifierParams)
        clf_2.fit(
            train_x_2, train_y_2,
            eval_set=[(train_x_2, train_y_2), (valid_x_2, valid_y_2)],
            verbose= 4,
            feature_name = features_classifier,
            early_stopping_rounds= 20,
            #categorical_feature= categorical_features
        )

        with timer('Saving model', n_blanks=4):
            # save model to file
            clf_2.booster_.save_model(
                os.path.join(
                    PATHS['WORKSPACE'],
                    'clf_predict_02.txt'))

            # dump model with pickle
            with open(
                os.path.join(
                    PATHS['WORKSPACE'],
                    'clf_predict_02.pkl'), 'wb') as fout:
                pickle.dump(clf_2, fout)

        plt.close('all')
        #plt.plot(clf.evals_result_['training'][lgbmParams['metric']], label='training')
        plt.plot(clf.evals_result_['training'][lgbmClassifierParams['metric']], label='trainin
        plt.plot(clf.evals_result_['valid_1'][lgbmClassifierParams['metric']], label='validat
        plt.legend()
```

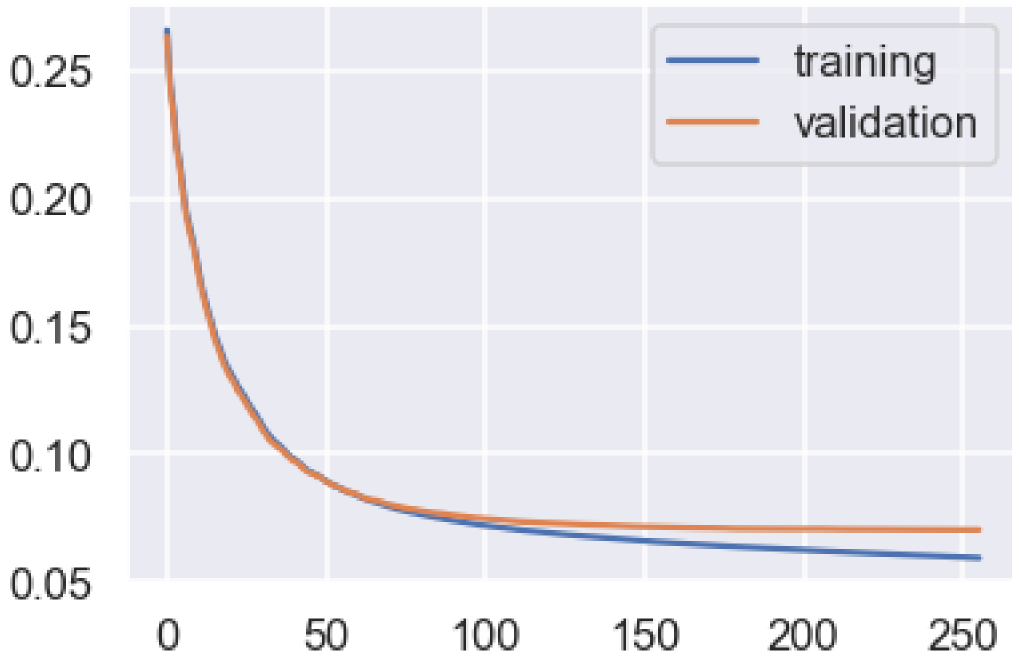
```
grid GridSearchCV(cv=KFold(n_splits=5, random_state=47, shuffle=True),
       error_score='raise-deprecating',
       estimator=LGBMClassifier(bagging_fraction=0.75, bagging_freq=10, boosting_type='gbdt',
        class_weight=None, colsample_bytree=1.0, feature_fraction=0.75,
        importance_type='gain', is_unbalance=True, lambda_l1=0.01,
        lambda_l2=5.0, learning_rate=0.05, max_bin=11, max_depth=-1,
     ...=1.0,
```

```
        subsample_for_bin=200000, subsample_freq=0, verbose=0,
        xgboost_dart_mode=True),
       fit_params=None, iid='warn', n_jobs=6,
       param_grid={'learning_rate': [0.05, 0.1, 1.0], 'metric': ['binary_logloss', 'binary_erro
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=2)
Fitting 5 folds for each of 72 candidates, totalling 360 fits


[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  29 tasks      | elapsed:    2.1s
[Parallel(n_jobs=6)]: Done 150 tasks      | elapsed:    9.1s
[Parallel(n_jobs=6)]: Done 360 out of 360 | elapsed:   19.7s finished


Training until validation scores don't improve for 20 rounds.
[4]	training's binary_logloss: 0.609124	valid_1's binary_logloss: 0.622025
[8]	training's binary_logloss: 0.571005	valid_1's binary_logloss: 0.586194
[12]	training's binary_logloss: 0.548262	valid_1's binary_logloss: 0.565728
[16]	training's binary_logloss: 0.522544	valid_1's binary_logloss: 0.540234
[20]	training's binary_logloss: 0.502779	valid_1's binary_logloss: 0.520384
[24]	training's binary_logloss: 0.486471	valid_1's binary_logloss: 0.505532
[28]	training's binary_logloss: 0.474739	valid_1's binary_logloss: 0.495437
[32]	training's binary_logloss: 0.465547	valid_1's binary_logloss: 0.48806
[36]	training's binary_logloss: 0.457019	valid_1's binary_logloss: 0.480825
[40]	training's binary_logloss: 0.447358	valid_1's binary_logloss: 0.472064
[44]	training's binary_logloss: 0.440314	valid_1's binary_logloss: 0.466459
[48]	training's binary_logloss: 0.433494	valid_1's binary_logloss: 0.461038
[52]	training's binary_logloss: 0.426912	valid_1's binary_logloss: 0.455134
[56]	training's binary_logloss: 0.421354	valid_1's binary_logloss: 0.450184
[60]	training's binary_logloss: 0.416655	valid_1's binary_logloss: 0.447033
[64]	training's binary_logloss: 0.411562	valid_1's binary_logloss: 0.441849
[68]	training's binary_logloss: 0.407763	valid_1's binary_logloss: 0.439055
[72]	training's binary_logloss: 0.403477	valid_1's binary_logloss: 0.435599
[76]	training's binary_logloss: 0.399585	valid_1's binary_logloss: 0.433311
[80]	training's binary_logloss: 0.395983	valid_1's binary_logloss: 0.430908
[84]	training's binary_logloss: 0.392614	valid_1's binary_logloss: 0.428403
[88]	training's binary_logloss: 0.389415	valid_1's binary_logloss: 0.426887
[92]	training's binary_logloss: 0.385736	valid_1's binary_logloss: 0.424258
[96]	training's binary_logloss: 0.383164	valid_1's binary_logloss: 0.422798
[100]	training's binary_logloss: 0.380511	valid_1's binary_logloss: 0.420911
[104]	training's binary_logloss: 0.377965	valid_1's binary_logloss: 0.419965
[108]	training's binary_logloss: 0.375412	valid_1's binary_logloss: 0.419165
[112]	training's binary_logloss: 0.372933	valid_1's binary_logloss: 0.418482
[116]	training's binary_logloss: 0.370387	valid_1's binary_logloss: 0.418143
[120]	training's binary_logloss: 0.367902	valid_1's binary_logloss: 0.417114
[124]	training's binary_logloss: 0.36565	valid_1's binary_logloss: 0.416619
[128]	training's binary_logloss: 0.363587	valid_1's binary_logloss: 0.416487
```

```
[132]        training's binary_logloss: 0.361586       valid_1's binary_logloss: 0.415255
[136]        training's binary_logloss: 0.359654       valid_1's binary_logloss: 0.414588
[140]        training's binary_logloss: 0.35782       valid_1's binary_logloss: 0.414268
[144]        training's binary_logloss: 0.356173       valid_1's binary_logloss: 0.414285
[148]        training's binary_logloss: 0.354341       valid_1's binary_logloss: 0.414508
[152]        training's binary_logloss: 0.352514       valid_1's binary_logloss: 0.414358
[156]        training's binary_logloss: 0.350901       valid_1's binary_logloss: 0.41492
[160]        training's binary_logloss: 0.349565       valid_1's binary_logloss: 0.415574
[164]        training's binary_logloss: 0.347905       valid_1's binary_logloss: 0.415367
Early stopping, best iteration is:
[146]        training's binary_logloss: 0.355175       valid_1's binary_logloss: 0.414061
BEST MODEL PARAMETERS:
lambda_l1 : 0.01
metric : binary_logloss
xgboost_dart_mode : True
min_child_weight : 5.0
objective : xentropy
importance_type : gain
min_split_gain : 0.5
verbose : 0
min_data_in_bin : 10
bagging_freq : 10
bagging_fraction : 0.75
is_unbalance : True
min_data_in_leaf : 20
min_sum_hessian_in_leaf : 10
learning_rate : 0.05
max_bin : 11
boosting_type : gbdt
n_estimators : 256
feature_fraction : 0.75
min_child_samples : 5
num_leaves : 15
lambda_l2 : 5.0

Training until validation scores don't improve for 20 rounds.
[4]        training's binary_logloss: 0.609124       valid_1's binary_logloss: 0.622025
[8]        training's binary_logloss: 0.571005       valid_1's binary_logloss: 0.586194
[12]        training's binary_logloss: 0.548262       valid_1's binary_logloss: 0.565728
[16]        training's binary_logloss: 0.522544       valid_1's binary_logloss: 0.540234
[20]        training's binary_logloss: 0.502779       valid_1's binary_logloss: 0.520384
[24]        training's binary_logloss: 0.486471       valid_1's binary_logloss: 0.505532
[28]        training's binary_logloss: 0.474739       valid_1's binary_logloss: 0.495437
[32]        training's binary_logloss: 0.465547       valid_1's binary_logloss: 0.48806
[36]        training's binary_logloss: 0.457019       valid_1's binary_logloss: 0.480825
[40]        training's binary_logloss: 0.447358       valid_1's binary_logloss: 0.472064
[44]        training's binary_logloss: 0.440314       valid_1's binary_logloss: 0.466459
[48]        training's binary_logloss: 0.433494       valid_1's binary_logloss: 0.461038
```

```
[52]	training's binary_logloss: 0.426912	valid_1's binary_logloss: 0.455134
[56]	training's binary_logloss: 0.421354	valid_1's binary_logloss: 0.450184
[60]	training's binary_logloss: 0.416655	valid_1's binary_logloss: 0.447033
[64]	training's binary_logloss: 0.411562	valid_1's binary_logloss: 0.441849
[68]	training's binary_logloss: 0.407763	valid_1's binary_logloss: 0.439055
[72]	training's binary_logloss: 0.403477	valid_1's binary_logloss: 0.435599
[76]	training's binary_logloss: 0.399585	valid_1's binary_logloss: 0.433311
[80]	training's binary_logloss: 0.395983	valid_1's binary_logloss: 0.430908
[84]	training's binary_logloss: 0.392614	valid_1's binary_logloss: 0.428403
[88]	training's binary_logloss: 0.389415	valid_1's binary_logloss: 0.426887
[92]	training's binary_logloss: 0.385736	valid_1's binary_logloss: 0.424258
[96]	training's binary_logloss: 0.383164	valid_1's binary_logloss: 0.422798
[100]	training's binary_logloss: 0.380511	valid_1's binary_logloss: 0.420911
[104]	training's binary_logloss: 0.377965	valid_1's binary_logloss: 0.419965
[108]	training's binary_logloss: 0.375412	valid_1's binary_logloss: 0.419165
[112]	training's binary_logloss: 0.372933	valid_1's binary_logloss: 0.418482
[116]	training's binary_logloss: 0.370387	valid_1's binary_logloss: 0.418143
[120]	training's binary_logloss: 0.367902	valid_1's binary_logloss: 0.417114
[124]	training's binary_logloss: 0.36565	valid_1's binary_logloss: 0.416619
[128]	training's binary_logloss: 0.363587	valid_1's binary_logloss: 0.416487
[132]	training's binary_logloss: 0.361586	valid_1's binary_logloss: 0.415255
[136]	training's binary_logloss: 0.359654	valid_1's binary_logloss: 0.414588
[140]	training's binary_logloss: 0.35782	valid_1's binary_logloss: 0.414268
[144]	training's binary_logloss: 0.356173	valid_1's binary_logloss: 0.414285
[148]	training's binary_logloss: 0.354341	valid_1's binary_logloss: 0.414508
[152]	training's binary_logloss: 0.352514	valid_1's binary_logloss: 0.414358
[156]	training's binary_logloss: 0.350901	valid_1's binary_logloss: 0.41492
[160]	training's binary_logloss: 0.349565	valid_1's binary_logloss: 0.415574
[164]	training's binary_logloss: 0.347905	valid_1's binary_logloss: 0.415367
Early stopping, best iteration is:
[146]	training's binary_logloss: 0.355175	valid_1's binary_logloss: 0.414061
    STARTING: Saving model
    FINISHED in 0.00 min.
```

Out[45]: <matplotlib.legend.Legend at 0x29e9d79c438>



2.6.10 Check model performance

```
In [46]: y_pred_2 = clf_2.predict(valid_x_2)
         print('The accuracy of prediction is:', accuracy_score(valid_y_2, y_pred_2))
         print('The roc_auc_score of prediction is:', roc_auc_score(valid_y_2, y_pred_2))
         print('The acccuracy of always predicting one class is:',
               max(valid_y_2.mean(), 1 - valid_y_2.mean()))
```

```
The accuracy of prediction is: 0.8107427612253462
The roc_auc_score of prediction is: 0.793479176633779
The acccuracy of always predicting one class is: 0.6261015526647083
```

Observation: The fitted model vastly outperforms rndom guess or guessing only one class.

2.6.11 Select best threshold:

Use f1 score to select best threshold for the new classifier

```
In [47]: tblPredictions_Valid = pd.DataFrame({'actual': valid_y_2})
         tblPredictions_Valid['probability'] = clf_2.predict_proba(valid_x_2, num_iteration=cl

         grid_thrslds = np.linspace(0.3,0.6, 10)
         grid_thrslds
```

```
for thrshld in grid_thrslds:
    tblPredictions_Valid['predicted'] = (tblPredictions_Valid['probability'] >= thrshl
    cm = confusion_matrix(tblPredictions_Valid['actual'], tblPredictions_Valid['predic
    recall = np.array([cm[0,0]/np.sum(cm[0,:]), cm[1,1]/np.sum(cm[1,:])])
    precission = np.array([cm[0,0]/np.sum(cm[:,0]), cm[1,1]/np.sum(cm[:,1])])
    f1 = 2 * (precission * recall) / (precission + recall)
    print('\nThreshold =',thrshld)
    print('recall:',recall, np.sqrt(recall[0]*recall[1]))
    print('precission:', precission, np.sqrt(precission[0]*precission[1]))
    print('F1:', f1, np.sqrt(f1[0]*f1[1]))
    print('cm[:,0]/np.sum(cm,axis=1)', cm[:,0]/np.sum(cm,axis=1))
    print('cm[:,0]',cm[:,0])
```

```
Threshold = 0.3
recall: [0.41301908 0.95174263] 0.6269671953492263
precission: [0.83636364 0.73082862] 0.7818174201526366
F1: [0.55296769 0.82678311] 0.6761540890153248
cm[:,0]/np.sum(cm,axis=1) [0.41301908 0.04825737]
cm[:,0] [368  72]

Threshold = 0.3333333333333333
recall: [0.47923681 0.93632708] 0.6698674527569357
precission: [0.81800766 0.75067168] 0.7836167354393573
F1: [0.60438783 0.83328363] 0.7096664573037669
cm[:,0]/np.sum(cm,axis=1) [0.47923681 0.06367292]
cm[:,0] [427  95]

Threshold = 0.36666666666666664
recall: [0.5375982  0.92359249] 0.7046429350169472
precission: [0.80775717 0.7698324 ] 0.7885668267519315
F1: [0.64555256 0.83973187] 0.7362683338691935
cm[:,0]/np.sum(cm,axis=1) [0.5375982  0.07640751]
cm[:,0] [479 114]

Threshold = 0.4
recall: [0.60381594 0.91353887] 0.7427040671245592
precission: [0.8065967  0.79428904] 0.8004192172103435
F1: [0.69062901 0.84975062] 0.7660694701749375
cm[:,0]/np.sum(cm,axis=1) [0.60381594 0.08646113]
cm[:,0] [538 129]

Threshold = 0.43333333333333335
recall: [0.65544332 0.89477212] 0.7658148663539445
precission: [0.78812416 0.81303289] 0.8004816413213243
F1: [0.71568627 0.85194639] 0.7808497557653901
cm[:,0]/np.sum(cm,axis=1) [0.65544332 0.10522788]
cm[:,0] [584 157]
```

```
Threshold = 0.4666666666666667
recall: [0.69584736 0.87734584] 0.7813442211412118
precission: [0.77210461 0.82848101] 0.7997962286000203
F1: [0.73199528 0.85221354] 0.78982041495443
cm[:,0]/np.sum(cm,axis=1) [0.69584736 0.12265416]
cm[:,0] [620 183]

Threshold = 0.5
recall: [0.72502806 0.86193029] 0.7905211244227947
precission: [0.75821596 0.83997387] 0.7980486192950116
F1: [0.74125072 0.85081045] 0.7941434748149198
cm[:,0]/np.sum(cm,axis=1) [0.72502806 0.13806971]
cm[:,0] [646 206]

Threshold = 0.5333333333333333
recall: [0.7687991  0.84316354] 0.8051232027690756
precission: [0.74537541 0.85928962] 0.8003082838937287
F1: [0.75690608 0.8511502 ] 0.8026460997035546
cm[:,0]/np.sum(cm,axis=1) [0.7687991  0.15683646]
cm[:,0] [685 234]

Threshold = 0.5666666666666667
recall: [0.79349046 0.82372654] 0.8084671622905186
precission: [0.72886598 0.86978061] 0.7962119662245034
F1: [0.75980656 0.84612737] 0.8018061611357316
cm[:,0]/np.sum(cm,axis=1) [0.79349046 0.17627346]
cm[:,0] [707 263]

Threshold = 0.6
recall: [0.81144781 0.80764075] 0.8095420431129163
precission: [0.71584158 0.8776402 ] 0.7926230843878502
F1: [0.76065229 0.84118674] 0.7999066295270267
cm[:,0]/np.sum(cm,axis=1) [0.81144781 0.19235925]
cm[:,0] [723 287]
```

2.6.12 Observation:

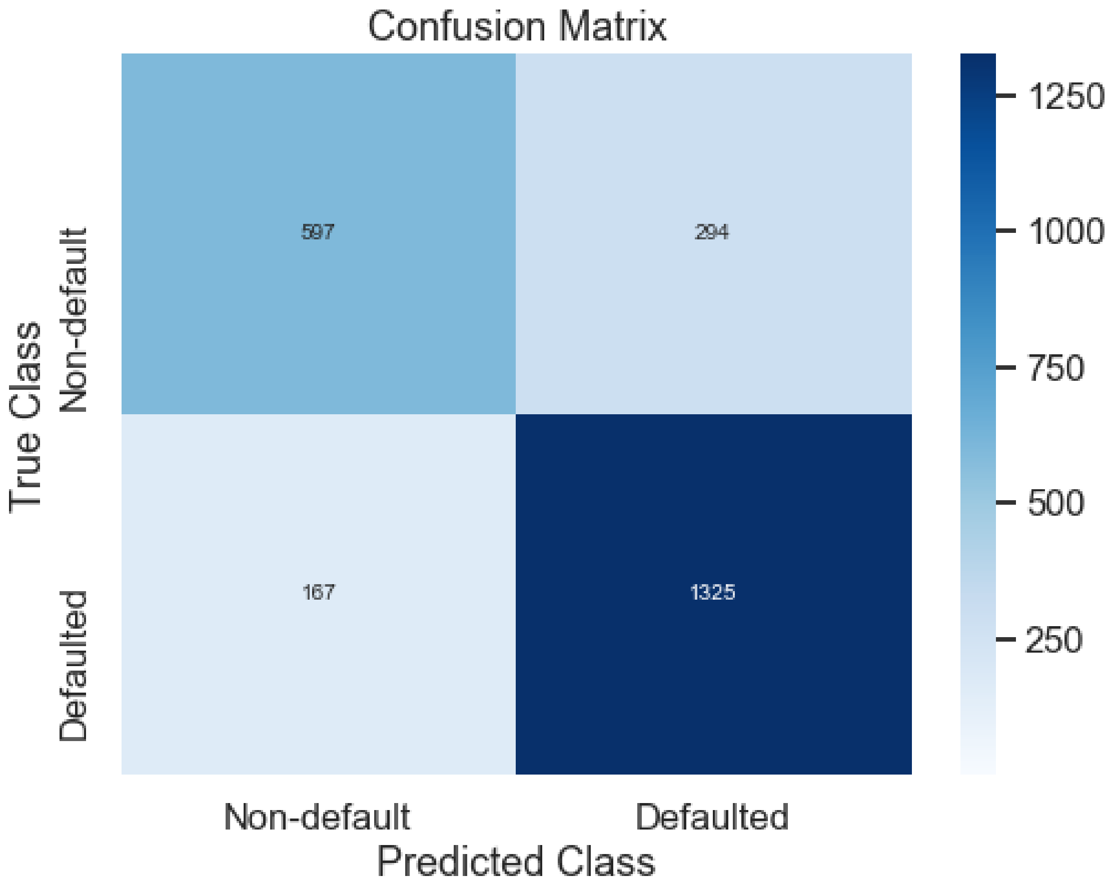
Best threshold seems to be about 0.45

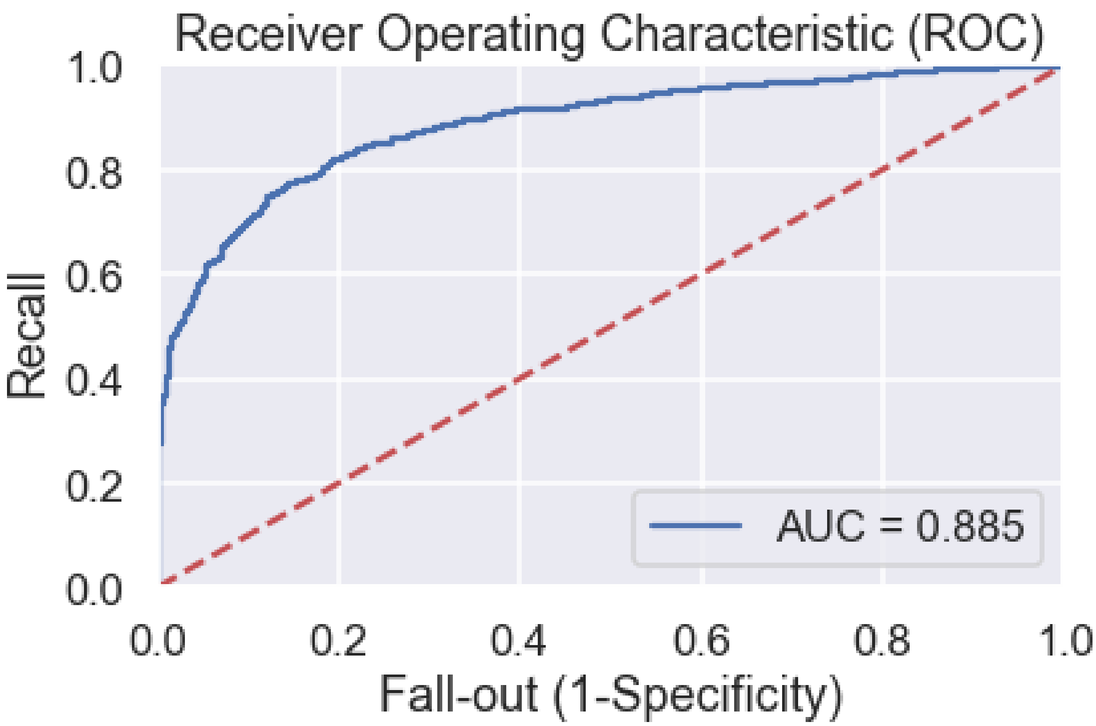
In [48]:
```
THRSHLD = 0.45

tblPredictions_Valid = pd.DataFrame({'actual': valid_y_2})
tblPredictions_Valid['probability'] = clf_2.predict_proba(valid_x_2, num_iteration=cl
tblPredictions_Valid['predicted'] = (tblPredictions_Valid['probability'] >= THRSHLD).a
```

```
cm = confusion_matrix(tblPredictions_Valid['actual'], tblPredictions_Valid['predicted
recall = np.array([cm[0,0]/np.sum(cm[0,:]), cm[1,1]/np.sum(cm[1,:])])
precission = np.array([cm[0,0]/np.sum(cm[:,0]), cm[1,1]/np.sum(cm[:,1])])
f1 = 2 * (precission * recall) / (precission + recall)
print('\n\nThreshold =',THRSHLD)
print('recall:',recall, np.sqrt(recall[0]*recall[1]))
print('precission:', precission, np.sqrt(precission[0]*precission[1]))
print('F1:', f1, np.sqrt(f1[0]*f1[1]))
print('cm[:,0]/np.sum(cm,axis=1)', cm[:,0]/np.sum(cm,axis=1))
print('cm[:,0]',cm[:,0])

#Print Confusion Matrix
plt.figure()
labels = ['Non-default', 'Defaulted']
plt.figure(figsize=(8,6))
sns.heatmap(cm, xticklabels = labels, yticklabels = labels, annot = True, fmt='d', cma
plt.title('Confusion Matrix')
plt.ylabel('True Class')
plt.xlabel('Predicted Class')
plt.show()

plt.figure()
false_positive_rate, recall_, thresholds = roc_curve(tblPredictions_Valid['actual'], t
roc_auc = auc(false_positive_rate, recall_)
plt.title('Receiver Operating Characteristic (ROC)')
plt.plot(false_positive_rate, recall_, 'b', label = 'AUC = %0.3f' %roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1], [0,1], 'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall')
plt.xlabel('Fall-out (1-Specificity)')

plt.tight_layout()
plt.show()
```

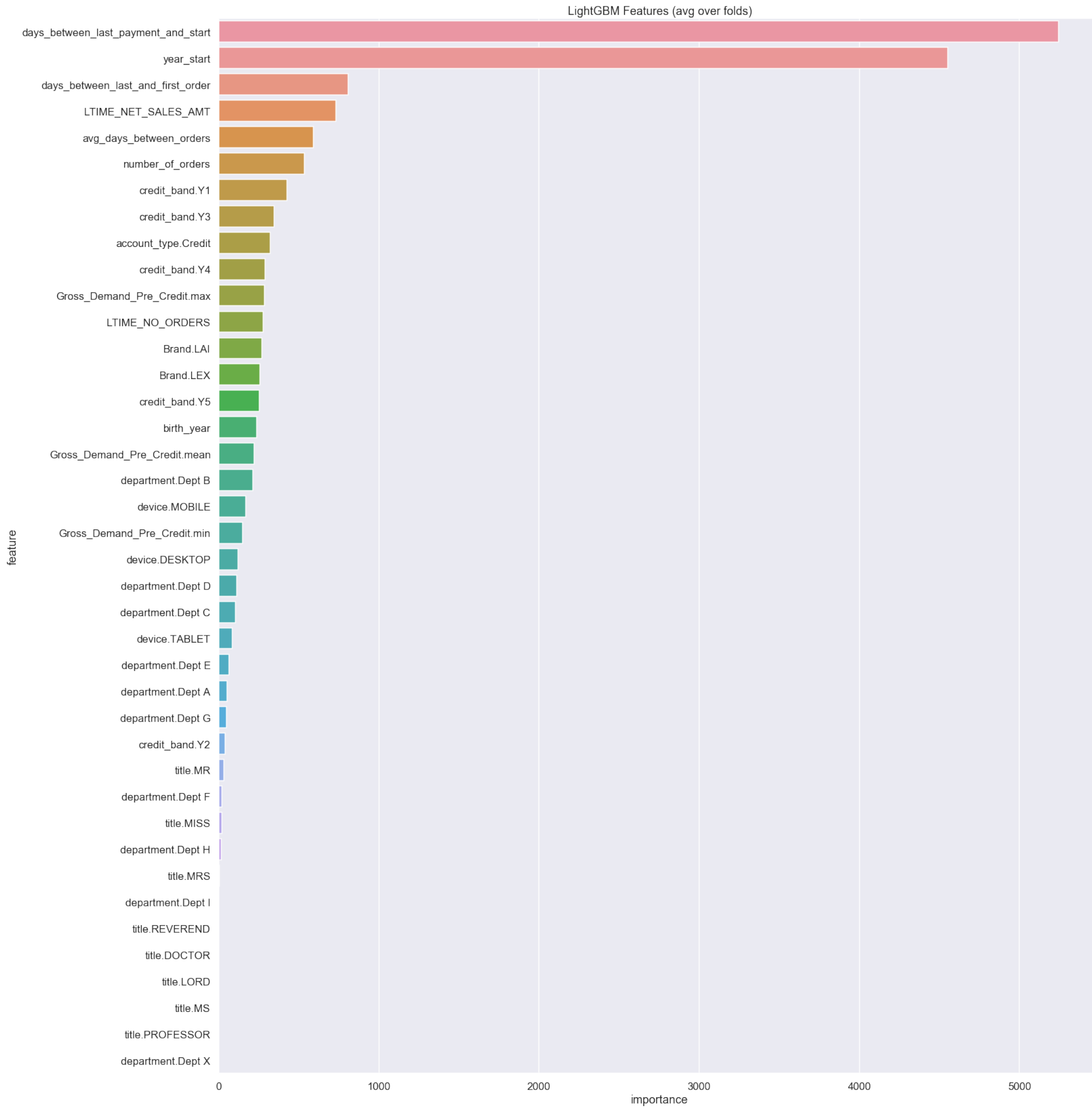
```
Threshold = 0.45
recall: [0.67003367 0.88806971] 0.7713861573490546
precission: [0.78141361 0.81840642] 0.7996961423594078
F1: [0.72145015 0.85181614] 0.7839278539508419
cm[:,0]/np.sum(cm,axis=1) [0.67003367 0.11193029]
cm[:,0] [597 167]
```

```
<Figure size 432x288 with 0 Axes>
```


Confusion Matrix
True Class
Non-default
Defaulted
597
294
167
1325
Non-default
Defaulted
Predicted Class
1250
1000
750
500
250




```
In [49]: if False:
             feature_imp = pd.DataFrame(sorted(zip(clf_2.feature_importances_, train_x_2.colum

             plt.figure(figsize=(20, 10))
             sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascen
             plt.title('LightGBM Features (avg over folds)')
             plt.tight_layout()
             plt.show()
         else:
             feature_importance_df = pd.DataFrame()
             feature_importance_df["feature"] = features_classifier
             feature_importance_df["importance"] = clf_2.feature_importances_
             display_importances(feature_importance_df)

         sumFI = feature_importance_df['importance'].sum()
         feature_importance_df['importance'] /= sumFI
         feature_importance_df.sort_values(by='importance', ascending=False)[:10]
```



Out[49]:

	feature	importance
3	days_between_last_payment_and_start	0.310259
4	year_start	0.269377
43	days_between_last_and_first_order	0.047922
0	LTIME_NET_SALES_AMT	0.043315
45	avg_days_between_orders	0.034964
44	number_of_orders	0.031725
16	credit_band.Y1	0.025332
18	credit_band.Y3	0.020360
52	account_type.Credit	0.019094
19	credit_band.Y4	0.017047

2.6.13 Observation:

While before balancing the dataset days_between_last_payment_and_start and days_between_last_payment_and_start where among the most important features, now they have become even more prominent in detecting customers that might default.

2.7 Best and worst customers:

Best customers are those that are least likely to default and generate the most revenue, which I'll measure by the total sales a customer generates (LTIME_NET_SALES_AMT).
Customer value will be the product of the two measures: $(1 - Prob(deault|features) \times TotalRevenue$.

I'll deine worst customers as those that are most likely to default and generate the least revenue

2.7.1 First, get predictions for all customers with the first model (catcher of non-defaulting customers):

```
In [50]: tblPredictions = pd.DataFrame({'actual': dfCustomerFeatures['defaulted'].astype(int)}
         tblPredictions['probability'] = clf.predict_proba(dfCustomerFeatures[features], num_i
         tblPredictions['total_revenue'] = dfCustomerFeatures['LTIME_NET_SALES_AMT']
         tblPredictions['customer_value'] = (1 - tblPredictions['probability']) * tblPredictio
```

2.7.2 Top 20 customers

```
In [51]: tblPredictions.sort_values(by='customer_value', ascending=False).head(20)
```

Out[51]:

identifier	actual	probability	total_revenue	customer_value
328199	0	0.012648	4.346431	4.291459
465063	0	0.038217	4.452349	4.282196
283271	0	0.011285	4.183307	4.136099
104343	0	0.008523	4.135119	4.099875
88199	0	0.025435	4.203445	4.096532
485939	0	0.016994	4.140219	4.069859
83159	0	0.012525	4.116779	4.065217
300620	0	0.007279	4.094956	4.065148
372080	0	0.020329	4.145769	4.061491
236829	0	0.007644	4.091630	4.060355
170510	0	0.012361	4.108292	4.057512
455546	0	0.009072	4.091845	4.054725
308941	0	0.026766	4.162677	4.051257
121609	0	0.007867	4.072497	4.040457
430161	0	0.009522	4.075609	4.036801
153083	0	0.028240	4.143778	4.026758
153568	0	0.014764	4.076085	4.015907
36042	0	0.007238	4.043469	4.014203
455060	0	0.009251	4.050987	4.013512
327394	0	0.006695	4.037221	4.010190

2.7.3 Worst 20 customers

In [52]:
```
# get predictions with the second model (catcher of defaulting customers)
tblPredictions_2 = pd.DataFrame({'actual': dfCustomerFeatures['defaulted'].astype(int)
tblPredictions_2['probability'] = clf_2.predict_proba(dfCustomerFeatures[features], nu
tblPredictions_2['total_revenue'] = dfCustomerFeatures['LTIME_NET_SALES_AMT']
tblPredictions_2['customer_value'] = (1 - tblPredictions_2['probability']) * tblPredic

tblPredictions[tblPredictions['total_revenue'] > 0].sort_values(by='customer_value', a
```

Out[52]:

identifier	actual	probability	total_revenue	customer_value
485117	1	0.999124	2.355068	0.002064
493358	1	0.999050	2.173186	0.002064
275914	1	0.998958	2.250420	0.002346
253145	1	0.998695	2.146128	0.002800
357838	1	0.998620	2.159868	0.002980
329070	1	0.998947	2.836324	0.002986
460496	1	0.998622	2.191311	0.003020
180837	1	0.998594	2.158362	0.003036
471896	1	0.998537	2.079181	0.003041
503873	1	0.998531	2.110590	0.003101
236365	1	0.998705	2.395588	0.003103
432087	1	0.998705	2.403721	0.003112
298055	1	0.998559	2.170262	0.003127
168883	1	0.998703	2.420038	0.003140
364392	1	0.998699	2.438701	0.003174
487431	1	0.998771	2.597146	0.003191
378973	1	0.998686	2.443106	0.003210
375378	1	0.998484	2.154424	0.003267
272550	1	0.998679	2.492411	0.003292
380258	1	0.998733	2.618100	0.003316