Jupyter notebook export, anonymized and output/print partly truncated

Table of Contents

Introduction

In [ ]:

```
See project desc
```

Library Imports

In [227]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('always')  # "error", "ignore", "always", "default", "module" or "once"
import category_encoders as ce
from pprint import pprint
from collections import Counter
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, f1_score, confusion_matrix, recall_score
from sklearn.pipeline import Pipeline, make_pipeline
from imblearn.over_sampling import SMOTE
from sklearn.utils import resample
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
 from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve
```

Data

Demographics File

File contains information on mobile pay users

Data Import and Exploration

In [3]:

```
# ISO encoding for special characters
# Skip columns with obviously irrelevant data (IDs, etc.)
pd.options.display.max_columns = 100

df_demo = pd.read_csv('data/R_demographics.csv', encoding='ISO-8859-1', low_memory=False, skipinitialspace=True, usecols=lambda column : column not in
["Subs_Asset_Id" , "End_Dt", "Subs_Id", "Cust_Owner_Id", "Cust_Bill_Id", "Cust_Used_Id", "Bill_Prof_Id", "Subs_Age_Days",
"Prod_Short_Desc", "Prod_Long_Desc", "Party_Id", "Scs_Customer_Id", "Master_Party_Id", "Top_Party_Id", "Top_Scs_Customer_Id",
"Cust_Name", "Bill_Prof_Ind", "Party_Cust_Id", "Party_Cust_Src_Id", "Rmc_Exvko", "Actual_Rmc_Laufd", "Actual_Rmc_Step",
"Actual_Rmc_Step_Last", "Rmc_Vkont", "First_Rmc_Step_No_Pay", "First_Rmc_Step_Last_No_Pay", "Last_Rmc_Step_No_Pay", "Last_Rmc_Step_Last_No_Pay"])

df_demo.head()
```

Out[3]:

	Co_Nplay_Typ_Id	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Stack_Typ_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Prod_Typ_Id	Cust_Seg_Id	Cust_Class_Id	Party_Typ_Id	Cust_Hier_Typ_Id	Ind_Gender	Ind_Birth_Dt	Ind_Age	Ind_Nationality_Code	Written_Language_Code	Ora g
0	1PMoPost	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1997-03-11	22.0	CH	DE	DE

In [4]:

```
# Remove leading and trailing spaces
df_demo = df_demo.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

In [5]:

```
df_demo.shape
```

Out[5]:

```
(515211, 34)
```

In [6]:

```
df_demo.info()
```

```
RangeIndex: 515211 entries, 0 to 515210
Data columns (total 34 columns):
Main_Phone_Num                515211 non-null int64
Co_Nplay_Typ_Id               515211 non-null object
Start_Dt                      515211 non-null object
Subs_Stat_Id                  515211 non-null object
Subscr_Since_Dt               515211 non-null object
Tac_Id                        515211 non-null object
Stack_Typ_Id                  515211 non-null object
Hectare_Cell_X_Coordinate     515211 non-null int64
Hectare_Cell_Y_Coordinate     515211 non-null int64
List_Recurring_Chrg_Amt       515211 non-null float64
Actual_Recurring_Chrg_Amt     515211 non-null float64
Subs_Age_Months               515211 non-null int64
Prod_Id                       515211 non-null object
```

```
Reg_Relevant_Flag            515211 non-null object
Prod_Item_Typ_Id             515211 non-null object
Price_Typ_Id                 515211 non-null object
Prod_Typ_Id                  515211 non-null object
Cust_Seg_Id                  515211 non-null int64
Cust_Class_Id                515211 non-null object
Party_Typ_Id                 515211 non-null object
Cust_Hier_Typ_Id             515211 non-null object
Ind_Gender                   515211 non-null object
Ind_Birth_Dt                 515211 non-null object
Ind_Age                      515130 non-null float64
Ind_Nationality_Code         515209 non-null object
Written_Language_Code        515211 non-null object
Oral_Language_Code           515211 non-null object
Cust_Lifecycle_Stat_Id       515211 non-null object
Cust_Lifecycle_Typ_Id        515211 non-null object
Cust_Stat_Id                 515211 non-null object
First_No_Pay_Dt              7798 non-null object
Last_no_pay_Dt               7798 non-null object
Bad_Pay_Count                7798 non-null float64
Flag_Last_6_Month            7798 non-null float64
dtypes: float64(5), int64(5), object(24)
memory usage: 133.6+ MB
```

Merchants File

File contains additional information for every merchant

Data Import and Exploration

In [7]:

```
df_mer = pd.read_csv('data/MerchantKey_work.csv', encoding='ISO-8859-1', low_memory=False, skipinitialspace=True)

df_mer.head()
```

Out[7]:

	MERCHANTNAME	MERCHANTLOCALE	MERCHANT_PAYMENT_TYPE
0	NATEL	PPT01	NATEL

In [8]:

```
df_mer.shape
```

Out[8]:

```
(595, 3)
```

In [9]:

```
df_mer.info()
```

```
RangeIndex: 595 entries, 0 to 594
Data columns (total 3 columns):
MERCHANTNAME             595 non-null object
MERCHANTLOCALE           595 non-null object
MERCHANT_PAYMENT_TYPE    595 non-null object
dtypes: object(3)
memory usage: 14.0+ KB
```

In [10]:

```
# Remove leading and trailing spaces
df_mer = df_mer.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

In [11]:

```
# Check columns for NULL value
df_mer.isna().sum()
```

Out[11]:

```
MERCHANTNAME             0
MERCHANTLOCALE           0
MERCHANT_PAYMENT_TYPE    0
dtype: int64
```

In [12]:

```
pprint(df_mer['MERCHANTLOCALE'].value_counts(dropna = False))
```

```
MEC02     2
FQC003    2
CNC005    2
MNC01     2
TPC001    2
MEC03     2
KIT01     2
FQC001    2
SHC001    2
BMC001    2
MEC01     2
YBR001    2
ZVC01     2
VBC01     2
ZZC01     2
BTC001    2
AXC01     2
STC01     2
CH        2
AFC016    1
ECC003    1
SIC610    1
IS        1
SIC16     1
SIC163    1
EGC102    1
AFC011    1
EVC046    1
SIC81     1
AMT01     1
Name: MERCHANTLOCALE, Length: 576, dtype: int64
```

Finding:

- Duplicate rows for same MERCHANTLOCALE -> drop duplicates
- Merchants have more than one MERCHANTLOCALE code, e.g. SIC164, SIC22 -> Create new feature company code only

In [13]:

```
# Check out duplicate MERCHANTLOCALE
df_mer.loc[(df_mer['MERCHANTLOCALE'] == 'AXC01')]
```

Out[13]:

	MERCHANTNAME	MERCHANTLOCALE	MERCHANT_PAYMENT_TYPE
94	E_pay: Ax	AXC01	E_pay
184	E_pay: Ax	AXC01	E_pay

In [14]:

```
# Drop duplicate MERCHANTLOCALE rows
df_mer = df_mer.drop_duplicates(subset='MERCHANTLOCALE', keep='first')
pprint(df_mer['MERCHANTLOCALE'].value_counts(dropna = False))
```

```
SPC09     1
EVC025    1
CAC01     1
SIC81     1
SIC163    1
SIC16     1
IS        1
SIC610    1
ECC003    1
EGC111    1
SIC153    1
ECC611    1
SIC205    1
EGC26     1
MOC010    1
SYT01     1
EVC64     1
VMC001    1
SIC158    1
EVC022    1
AFC015    1
SIC041    1
SIC120    1
SIC04     1
SPC019    1
MAC60     1
MOC013    1
SIC51     1
IDC03     1
GIC001    1
Name: MERCHANTLOCALE, Length: 576, dtype: int64
```

In [15]:

```
# MERCHANT_PAYMENT_TYPE: Replace blanks with _ (for one-hot_encoding later)
df_mer['MERCHANT_PAYMENT_TYPE'] = df_mer['MERCHANT_PAYMENT_TYPE'].str.replace(" ", "_")
```

Transactions File

File contains all mobile pay transactions

Data Import and Exploration

In [16]:

```
# ISO encoding for special characters
# NF_MERCHANT was created in R, it groups Merchant Keys (e.g. AIT01, AIT02 = AIT)
```

```
df_data = pd.read_csv('data/R_data.csv', encoding='ISO-8859-1', low_memory=False, skipinitialspace=True)
df_data.head()
```

Out[16]:

	ODI_MSISDN	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
0	4179	GG	4.9	2017-11-26 00:00:00.0000000	GG

In [17]:

```
df_data.shape
```

Out[17]:

```
(28050941, 5)
```

In [18]:

```
df_data.info()
```

```
RangeIndex: 28050941 entries, 0 to 28050940
Data columns (total 5 columns):
ODI_MSISDN          object
ODI_MERCHANT_KEY    object
GROSS_PRICE_AMT     float64
REV_EFF_TS          object
NF_MERCHANT         object
dtypes: float64(1), object(4)
memory usage: 1.0+ GB
```

In [19]:

```
# Remove leading and trailing spaces
df_data = df_data.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

In [20]:

```
# Check columns for NULL value
df_data.isna().sum()
```

Out[20]:

```
ODI_MSISDN          251530
ODI_MERCHANT_KEY         0
GROSS_PRICE_AMT          0
REV_EFF_TS               0
NF_MERCHANT              0
dtype: int64
```

Finding:

Drop rows without ODI_MSISDN (mobile no.) as they cannot be matched with demographics -> these rows belong to Netflix transactions

In [21]:

```
# Drop rows ODI_MSISDN = NULL
df_data_stats = df_data
df_data = df_data.dropna(subset=['ODI_MSISDN'])

rows_dropped = df_data_stats.shape[0] - df_data.shape[0]
pprint('No. of dropped rows: ' + str(rows_dropped))
```

```
'No. of dropped rows: 251530'
```

In [22]:

```
# Transactions with amount = 0?
df_data.loc[(df_data['GROSS_PRICE_AMT'] == 0.0)]
```

Out[22]:

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
6780	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6791	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6802	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6811	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6821	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6833	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6847	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6859	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6872	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6882	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6896	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6911	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN
6920	UNKNOWN	0.0	2017-06-30 00:00:00.0000000	UNKNOWN

1423321 rows × 5 columns

In [23]:

```
# Transactions with amount < 0 (credit notes)?
df_data.loc[(df_data['GROSS_PRICE_AMT'] < 0.0)]
```

Out[23]:

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
7408	CHT03	-1.00	2017-06-15 00:00:00.0000000	CHT
12023	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12039	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12163	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12197	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12204	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12344	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12348	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12357	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12404	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12512	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12546	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12566	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
12582	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
36576	CHT03	-1.00	2017-06-05 00:00:00.0000000	CHT
41130	CHT03	-2.50	2017-07-09 00:00:00.0000000	CHT

78237 rows × 5 columns

In [24]:

```
# Check out random customer
df_data.loc[(df_data['ODI_MSISDN'] == '417XXXXXXX')]
```

Out[24]:

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
12023	UNKNOWN	-1.77	2017-06-30 00:00:00.0000000	UNKNOWN
289302	UNKNOWN	7.36	2017-06-30 00:00:00.0000000	UNKNOWN
8527237	UNKNOWN	0.80	2017-05-31 00:00:00.0000000	UNKNOWN

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
9082586	UNKNOWN	40.00	2017-05-31 00:00:00.0000000	UNKNOWN
9190511	UNKNOWN	-40.00	2017-05-31 00:00:00.0000000	UNKNOWN
9224074	UNKNOWN	53.23	2017-05-31 00:00:00.0000000	UNKNOWN
9411566	UNKNOWN	-53.23	2017-05-31 00:00:00.0000000	UNKNOWN
9789573	UNKNOWN	3.84	2017-05-31 00:00:00.0000000	UNKNOWN
11231900	UNKNOWN	0.50	2017-06-29 00:00:00.0000000	UNKNOWN
11315517	UNKNOWN	0.10	2017-06-29 00:00:00.0000000	UNKNOWN
11356499	UNKNOWN	0.80	2017-06-29 00:00:00.0000000	UNKNOWN
11359913	UNKNOWN	3.00	2017-06-29 00:00:00.0000000	UNKNOWN
11480484	UNKNOWN	7.36	2017-06-29 00:00:00.0000000	UNKNOWN
19600787	UNKNOWN	0.00	2017-05-31 00:00:00.0000000	UNKNOWN
19600964	UNKNOWN	0.00	2017-05-31 00:00:00.0000000	UNKNOWN
19603670	UNKNOWN	0.00	2017-05-31 00:00:00.0000000	UNKNOWN

Finding:

- GROSS_PRICE_AMT = 0.0 looks like useless data -> drop rows
- GROSS_PRICE_AMT < 0.0 must be kept (credit notes)
- Looks like lots of duplicates!! -> same mobile no., merchant, amount and date. Unfortunately no transaction ID or transaction timestamp is available to clearly identify duplicates. Considering the large amount of transaction data I will drop what looks like duplicates

In [25]:

```
# Drop duplicate transaction rows
df_data_stats = df_data

df_data = df_data.drop_duplicates(subset=['ODI_MSISDN', 'ODI_MERCHANT_KEY', 'GROSS_PRICE_AMT', 'REV_EFF_TS'], keep='first')
rows_dropped = df_data_stats.shape[0] - df_data.shape[0]

pprint('No. of dropped rows: ' + str(rows_dropped))
```

```
'No. of dropped rows: 4003896'
```

In [26]:

```
# Check out random customer
df_data.loc[(df_data['ODI_MSISDN'] == '417XXXXXXX')].sort_values(by='REV_EFF_TS', ascending=False)
```

Out[26]:

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
22131355	GG	1.0	2019-03-20 00:00:00.0000000	GG
22158762	GG	2.0	2019-03-20 00:00:00.0000000	GG
17668078	GG	20.0	2019-03-01 00:00:00.0000000	GG
18182482	GG	1.0	2019-03-01 00:00:00.0000000	GG
17674664	GG	9.9	2019-03-01 00:00:00.0000000	GG
17131414	GG	6.0	2019-02-28 00:00:00.0000000	GG
16811975	GG	2.0	2019-02-28 00:00:00.0000000	GG
15982897	GG	9.9	2019-02-28 00:00:00.0000000	GG
15982625	GG	20.0	2019-02-28 00:00:00.0000000	GG
14991186	GG	9.9	2019-02-27 00:00:00.0000000	GG
14990824	GG	20.0	2019-02-27 00:00:00.0000000	GG
15852877	GG	6.9	2019-02-27 00:00:00.0000000	GG

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
14989206	GG	6.9	2019-02-26 00:00:00.0000000	GG
14118947	GG	9.9	2019-02-26 00:00:00.0000000	GG

226 rows × 5 columns

In [27]:

```
# Drop rows GROSS_PRICE_AMT = 0.0
df_data_stats = df_data

df_data = df_data.drop(df_data[(df_data.GROSS_PRICE_AMT == 0.0)].index)
rows_dropped = df_data_stats.shape[0] - df_data.shape[0]

pprint('No. of dropped rows: ' + str(rows_dropped))
```

```
'No. of dropped rows: 307592'
```

In [28]:

```
# UNKNOWN mobile no.
df_data.loc[(df_data['ODI_MSISDN'] == 'UNKNOWN')]
```

Out[28]:

	ODI_MSISDN	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT
61795	UNKNOWN	UNKNOWN	69.80	2018-08-01 00:00:00.0000000	UNKNOWN
67425	UNKNOWN	UNKNOWN	54.80	2018-08-01 00:00:00.0000000	UNKNOWN
118979	UNKNOWN	UNKNOWN	-5.00	2018-08-01 00:00:00.0000000	UNKNOWN
182694	UNKNOWN	UNKNOWN	69.00	2018-08-01 00:00:00.0000000	UNKNOWN
214879	UNKNOWN	UNKNOWN	9.00	2018-08-01 00:00:00.0000000	UNKNOWN
263241	UNKNOWN	UNKNOWN	29.80	2018-08-01 00:00:00.0000000	UNKNOWN
263254	UNKNOWN	UNKNOWN	0.10	2018-08-01 00:00:00.0000000	UNKNOWN
288022	UNKNOWN	UNKNOWN	0.50	2018-08-01 00:00:00.0000000	UNKNOWN
288033	UNKNOWN	UNKNOWN	0.30	2018-08-01 00:00:00.0000000	UNKNOWN
295711	UNKNOWN	UNKNOWN	0.20	2018-08-01 00:00:00.0000000	UNKNOWN
314687	UNKNOWN	UNKNOWN	0.60	2018-08-01 00:00:00.0000000	UNKNOWN

2613 rows × 5 columns

Finding:

Drop 'UNKNOWN' rows as they cannot be matched to any mobile subscribers

In [29]:

```
# Drop 'UNKNOWN' rows
df_data_stats = df_data

df_data = df_data.drop(df_data[(df_data.ODI_MSISDN =='UNKNOWN')].index)
rows_dropped = df_data_stats.shape[0] - df_data.shape[0]

pprint('No. of dropped rows: ' + str(rows_dropped))
```

```
'No. of dropped rows: 2613'
```

In [30]:

```
# Convert data types
df_data['ODI_MSISDN'] = df_data.ODI_MSISDN.astype(int) # to match data type of demographics file
df_data['REV_EFF_TS'] = pd.to_datetime(df_data['REV_EFF_TS']) # for calculations later
```

Merge Data with Merchants

In [31]:

```
# Add Merchant information
# Join df_data with df_mer

df_data_mer = pd.merge(df_data,
                  df_mer[['MERCHANTLOCALE', 'MERCHANT_PAYMENT_TYPE']],
                  left_on='ODI_MERCHANT_KEY',
                  right_on='MERCHANTLOCALE',
                  how='left')

df_data_mer = df_data_mer.drop('MERCHANTLOCALE', 1)

df_data_mer.head()
```

Out[31]:

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT	MERCHANT_PAYMENT_TYPE
0	GG	4.9	2017-11-26	GG	N_Pay
1	AIT01	3.0	2018-05-14	AIT	N_Pay
2	AIT01	1.0	2018-03-30	AIT	N_Pay
3	AIT01	12.9	2017-09-22	AIT	N_Pay
4	GG	3.0	2017-11-08	GG	N_Pay

In [32]:

```
# Temp Backup
df_data_mer_backup = df_data_mer.copy()
```

In [33]:

```
# Closer look at unknown ODI_MERCHANT_KEY
df_data_mer.loc[(df_data_mer['ODI_MERCHANT_KEY'] == 'UNKNOWN')]
```

Out[33]:

	ODI_MERCHANT_KEY	GROSS_PRICE_AMT	REV_EFF_TS	NF_MERCHANT	MERCHANT_PAYMENT_TYPE
9871	UNKNOWN	4.86	2017-06-30	UNKNOWN	NaN
9880	UNKNOWN	12.00	2017-06-30	UNKNOWN	NaN
9888	UNKNOWN	4.59	2017-06-30	UNKNOWN	NaN
9894	UNKNOWN	2.10	2017-06-30	UNKNOWN	NaN
9898	UNKNOWN	12.00	2017-06-30	UNKNOWN	NaN
9902	UNKNOWN	4.59	2017-06-30	UNKNOWN	NaN
9910	UNKNOWN	4.59	2017-06-30	UNKNOWN	NaN
9914	UNKNOWN	3.80	2017-06-30	UNKNOWN	NaN
9923	UNKNOWN	4.59	2017-06-30	UNKNOWN	NaN
9930	UNKNOWN	4.86	2017-06-30	UNKNOWN	NaN
9935	UNKNOWN	4.59	2017-06-30	UNKNOWN	NaN

722969 rows × 6 columns

Finding:

Keep rows ODI_MERCHANT_KEY = UNKNOWN, change MERCHANT_PAYMENT_TYPE from NaN to UNKNOWN

In [34]:

```
# Change MERCHANT_PAYMENT_TYPE
df_data_mer['MERCHANT_PAYMENT_TYPE'].fillna('UNKNOWN', inplace = True)
```

In [35]:

```
# Check columns for NULL value
df_data_mer.isna().sum()
```

Out[35]:

```
ODI_MSISDN               0
ODI_MERCHANT_KEY         0
GROSS_PRICE_AMT          0
REV_EFF_TS               0
NF_MERCHANT              0
MERCHANT_PAYMENT_TYPE    0
dtype: int64
```

Merge Demographics with Data

In [36]:

```
# Merge Demographics with Data

mrg_demo_data = df_demo.merge(df_data_mer, left_on='Main_Phone_Num', right_on='ODI_MSISDN', how='inner')
mrg_demo_data = mrg_demo_data.drop('ODI_MSISDN', 1)

mrg_demo_data.head()
```

Out[36]:

	Co_Nplay_Typ_Id	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Stack_Typ_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Prod_Typ_Id	Cust_Seg_Id	Cust_Class_Id	Party_Typ_Id	Cust_Hier_Typ_Id	Ind_Gender	Ind_Birth_Dt	Ind_Age	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Cust_Lifecycle_Stat_Id	Cust_Lifecycle_Typ_Id	Cust_Stat_Id
0	1PMoPost	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1997-03-11	22.0	CH	DE	DE	Current Customer	Current	Active
1	1PMoPost	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1997-03-11	22.0	CH	DE	DE	Current Customer	Current	Active
2	1PMoPost	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1997-03-11	22.0	CH	DE	DE	Current Customer	Current	Active
3	1PMoPost	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1997-03-11	22.0	CH	DE	DE	Current Customer	Current	Active
4	1PMoPost	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1997-03-11	22.0	CH	DE	DE	Current Customer	Current	Active

Aggregate Transaction Data

In [37]:

```
# Create new features based on aggregation of transaction data

mrg_agg = mrg_demo_data.groupby('Main_Phone_Num').agg({'Main_Phone_Num':'count',
                                   'GROSS_PRICE_AMT':['mean', 'sum', 'max', 'min'],
                                   'REV_EFF_TS':['max', 'min']})
```

```
mrg_agg.head()
```

Out[37]:

Main_Phone_Num	GROSS_PRICE_AMT				REV_EFF_TS	
count	mean	sum	max	min	max	min
22	5.159091	113.50	11.7	2.0	2019-02-06	2017-07-08
187	2.892246	540.85	8.0	1.0	2019-03-22	2017-04-12
3	8.333333	25.00	10.0	5.0	2018-09-18	2018-02-23
1	5.200000	5.20	5.2	5.2	2019-03-25	2019-03-25
23	6.000000	138.00	6.0	6.0	2019-03-12	2017-04-13

Note:

In case customer has a credit note GROSS_PRICE_AMT_min will be negativ (<0). It might make more sense to use the min positive amount but will leave it for now

In [38]:

```
# Add aggregate features to dataframe
mrg_agg.columns = ['_'.join(col) for col in mrg_agg.columns]
mrg_agg.head()
```

Out[38]:

Main_Phone_Num_count	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min
22	5.159091	113.50	11.7	2.0	2019-02-06	2017-07-08
187	2.892246	540.85	8.0	1.0	2019-03-22	2017-04-12
3	8.333333	25.00	10.0	5.0	2018-09-18	2018-02-23
1	5.200000	5.20	5.2	5.2	2019-03-25	2019-03-25
23	6.000000	138.00	6.0	6.0	2019-03-12	2017-04-13

In [39]:

```
mrg_agg['GROSS_PRICE_AMT_mean'] = mrg_agg['GROSS_PRICE_AMT_mean'].round(decimals=4)
```

In [40]:

```
# Rename Main_Phone_Num_count
mrg_agg.rename(columns={"Main_Phone_Num_count": "NF_Num_Transactions"}, inplace=True)
mrg_agg.head()
```

Out[40]:

NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min
22	5.1591	113.50	11.7	2.0	2019-02-06	2017-07-08
187	2.8922	540.85	8.0	1.0	2019-03-22	2017-04-12
3	8.3333	25.00	10.0	5.0	2018-09-18	2018-02-23
1	5.2000	5.20	5.2	5.2	2019-03-25	2019-03-25
23	6.0000	138.00	6.0	6.0	2019-03-12	2017-04-13

In [41]:

```
# Create MERCHANT_PAYMENT_TYPE Feature (proper one-hot-encoding will be done later)

mrg_agg = mrg_agg.join(pd.crosstab(mrg_demo_data['Main_Phone_Num'], mrg_demo_data['MERCHANT_PAYMENT_TYPE'], dropna=False).add_prefix('HAS_')).reset_index()
mrg_agg.head()
```

Out[41]:

	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min	HAS_E_pay	HAS_N_Pay	HAS_UNKNOWN
0	22	5.1591	113.50	11.7	2.0	2019-02-06	2017-07-08	22	0	0

	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min	HAS_E_pay	HAS_N_Pay	HAS_UNKNOWN
1	187	2.8922	540.85	8.0	1.0	2019-03-22	2017-04-12	0	187	0
2	3	8.3333	25.00	10.0	5.0	2018-09-18	2018-02-23	0	3	0
3	1	5.2000	5.20	5.2	5.2	2019-03-25	2019-03-25	0	1	0
4	23	6.0000	138.00	6.0	6.0	2019-03-12	2017-04-13	0	23	0

In [42]:

```
# Create one-hot-encoding for merchants (did mobile user purchase from merchant XYZ?)

#mrg_agg = mrg_agg.join(pd.crosstab(mrg_demo_data['Main_Phone_Num'], mrg_demo_data['NF_MERCHANT'], dropna=False).add_prefix('HAS_COMP_')).reset_index()

#mrg_agg.head(20)
```

Merge Demographics with aggregated Transaction Data

In [43]:

```
# Merge Demographics with aggregated Transactions
# Using left join to keep all demographics records. Customers without transactions will have NaN values

df = df_demo.merge(mrg_agg, left_on='Main_Phone_Num', right_on='Main_Phone_Num', how='left')
df.head()
```

Out[43]:

	Co_Nplay_Typ_Id	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Stack_Typ_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Prod_Typ_Id	Cust_Seg_Id	Cust_Class_Id	Party_Typ_Id	Cust_Hier_Typ_Id	Ind_Gender	Ind_Birth_Dt	Ind_Age	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Cust_Lifecycle_Stat_Id	Cust_Lifecycle_Typ_Id	Cust_Stat_Id	First_No_Pay_Dt	Last_no_pay_Dt	Bad_Pay_Count	Flag_Last_6_Month	N T
0	1PMoPost	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1997-03-11	22.0	CH	DE	DE	Current Customer	Current	Active	2017-05-09	2018-11-13	8.0	1.0	8
1	1PMoPost	2019-03-09	ACTIVATED	2015-12-21	35736109	N	35.0	35.0	39	5-2CEGS	Y	Bundle	Recurring	Inventory	1202	E	Ind	Master	M	2002-07-05	16.0	BA	DE	DE	Current Customer	Current	Active	NaN	NaN	NaN	NaN	2
2	1PMoPost	2018-07-01	ACTIVATED	2015-11-27	35460707	N	59.0	59.0	40	5-2E1W4	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	F	1987-06-06	31.0	CH	FR	FR	Current Customer	Current	Active	NaN	NaN	NaN	NaN	N
3	1PMoPost	2018-08-01	ACTIVATED	2012-10-04	35240209	N	69.0	69.0	77	5-2E1WT	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1998-07-21	20.0	CH	DE	DE	Current Customer	Current	Active	NaN	NaN	NaN	NaN	3
4	1PMoPost	2018-11-13	ACTIVATED	2015-10-13	35721309	N	55.0	55.0	41	5-2E1XN	Y	Bundle	Recurring	Inventory	1203	E	Ind	Master	F	1998-07-13	20.0	CH	DE	EN	Current Customer	Current	Active	NaN	NaN	NaN	NaN	1

In [44]:

```
# Backup
df_backup_2 = df.copy()
```

In [45]:

```
df.info()
```

```
Int64Index: 515211 entries, 0 to 515210
```

```
Data columns (total 44 columns):
Main_Phone_Num               515211 non-null int64
Co_Nplay_Typ_Id              515211 non-null object
Start_Dt                     515211 non-null object
Subs_Stat_Id                 515211 non-null object
Subscr_Since_Dt              515211 non-null object
Tac_Id                       515211 non-null object
Stack_Typ_Id                 515211 non-null object
Hectare_Cell_X_Coordinate    515211 non-null int64
Hectare_Cell_Y_Coordinate    515211 non-null int64
List_Recurring_Chrg_Amt      515211 non-null float64
Actual_Recurring_Chrg_Amt    515211 non-null float64
Subs_Age_Months              515211 non-null int64
Prod_Id                      515211 non-null object
Reg_Relevant_Flag            515211 non-null object
Prod_Item_Typ_Id             515211 non-null object
Price_Typ_Id                 515211 non-null object
Prod_Typ_Id                  515211 non-null object
Cust_Seg_Id                  515211 non-null int64
Cust_Class_Id                515211 non-null object
Party_Typ_Id                 515211 non-null object
Cust_Hier_Typ_Id             515211 non-null object
Ind_Gender                   515211 non-null object
Ind_Birth_Dt                 515211 non-null object
Ind_Age                      515130 non-null float64
Ind_Nationality_Code         515209 non-null object
Written_Language_Code        515211 non-null object
Oral_Language_Code           515211 non-null object
Cust_Lifecycle_Stat_Id       515211 non-null object
Cust_Lifecycle_Typ_Id        515211 non-null object
Cust_Stat_Id                 515211 non-null object
First_No_Pay_Dt              7798 non-null object
Last_no_pay_Dt               7798 non-null object
Bad_Pay_Count                7798 non-null float64
Flag_Last_6_Month            7798 non-null float64
NF_Num_Transactions          232355 non-null float64
GROSS_PRICE_AMT_mean         232355 non-null float64
GROSS_PRICE_AMT_sum          232355 non-null float64
GROSS_PRICE_AMT_max          232355 non-null float64
GROSS_PRICE_AMT_min          232355 non-null float64
REV_EFF_TS_max               232355 non-null datetime64[ns]
REV_EFF_TS_min               232355 non-null datetime64[ns]
HAS_E_pay                  232355 non-null float64
HAS_N_Pay                232355 non-null float64
HAS_UNKNOWN                  232355 non-null float64
dtypes: datetime64[ns](2), float64(13), int64(5), object(24)
memory usage: 176.9+ MB
```

Feature Analysis

Missing Values

In [46]:

```
# Check columns for NULL value
df.isna().sum()
```

Out[46]:

```
Main_Phone_Num                    0
Co_Nplay_Typ_Id                   0
Start_Dt                          0
Subs_Stat_Id                      0
Subscr_Since_Dt                   0
Tac_Id                            0
Stack_Typ_Id                      0
Hectare_Cell_X_Coordinate         0
Hectare_Cell_Y_Coordinate         0
List_Recurring_Chrg_Amt           0
Actual_Recurring_Chrg_Amt         0
Subs_Age_Months                   0
Prod_Id                           0
Reg_Relevant_Flag                 0
Prod_Item_Typ_Id                  0
Price_Typ_Id                      0
Prod_Typ_Id                       0
Cust_Seg_Id                       0
Cust_Class_Id                     0
Party_Typ_Id                      0
Cust_Hier_Typ_Id                  0
Ind_Gender                        0
Ind_Birth_Dt                      0
Ind_Age                          81
Ind_Nationality_Code              2
Written_Language_Code             0
Oral_Language_Code                0
Cust_Lifecycle_Stat_Id            0
Cust_Lifecycle_Typ_Id             0
Cust_Stat_Id                      0
First_No_Pay_Dt              507413
Last_no_pay_Dt               507413
Bad_Pay_Count                507413
Flag_Last_6_Month            507413
NF_Num_Transactions          282856
GROSS_PRICE_AMT_mean         282856
GROSS_PRICE_AMT_sum          282856
GROSS_PRICE_AMT_max          282856
GROSS_PRICE_AMT_min          282856
REV_EFF_TS_max               282856
REV_EFF_TS_min               282856
HAS_E_pay                  282856
HAS_N_Pay                282856
HAS_UNKNOWN                  282856
dtype: int64
```

Finding:

- Ind_Age and Ind_Nationality_Code contain some NULL values
- 3 features with lots of NULL values
- Target Flag_Last_6_Month with lots of NULL values

- New features from transaction data with NULL values

In [47]:

```
# Closer look at Ind_Age
df.loc[df['Ind_Age'].isnull()]
```

Out[47]:

	Co_Nplay_Typ_Id	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Stack_Typ_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Prod_Typ_Id	Cust_Seg_Id	Cust_Class_Id	Party_Typ_Id	Cust_Hier_Typ_Id	Ind_Gender	Ind_Birth_Dt	Ind_Age	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Cust_Lifecycle_Stat_Id	Cust_Lifecycle_Typ_Id	Cust_Stat_Id	First_No_Pay_Dt	Last_no_pay_Dt	Bad_Pay_Count	Flag_Last_6_Month
20648	1PMoPost	2018-01-19	ACTIVATED	2009-10-09	35535406	N	35.0	35.0	113	epb-AAA_BBB_light	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	F	1900-01-01	NaN	CH	DE	DE	Current Customer	Current	Active	NaN	NaN	NaN	NaN
21536	1PMoPost	2018-01-19	ACTIVATED	2001-01-16	35880205	N	59.0	59.0	218	5-2E1W4	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	M	1900-01-01	NaN	CH	IT	IT	Current Customer	Current	Active	NaN	NaN	NaN	NaN
48056	1PMoPost	2018-01-19	ACTIVATED	2005-02-10	35975108	N	65.0	65.0	169	epb-AAA_BBB_XS	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	F	1911-11-11	NaN	CH	FR	FR	Current Customer	Current	Active	NaN	NaN	NaN	NaN
48782	1PMoPost	2019-02-26	ACTIVATED	2012-07-05	35879308	N	80.0	80.0	80	5-30HS1	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	M	1911-11-11	NaN	CH	DE	DE	Current Customer	Current	Active	NaN	NaN	NaN	NaN
58420	1PMoPost	2018-08-11	ACTIVATED	2010-07-31	35993706	N	35.0	35.0	104	epb-AAA_BBB_light	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	F	1911-11-11	NaN	CH	DE	DE	Current Customer	Current	Active	NaN	NaN	NaN	NaN
65062	1PMoPost	2018-01-19	ACTIVATED	2011-03-03	35897907	N	59.0	59.0	97	5-2E1WJ	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	M	1900-01-01	NaN	CH	DE	DE	Current Customer	Current	Active	NaN	NaN	NaN	NaN

81 rows × 44 columns

Finding:

- Only two different birthdates are set for Ind_Age = NULL (1900-01-01 / 1911-11-11) -> looks like dummy birthdates -> drop rows
- All records have Cust_Seg_Id = 1300 -> check with data owner if this was set deliberitly

In [48]:

```
# Drop rows Ind_Age = NULL
df_stats = df
df = df.dropna(subset=['Ind_Age'])

rows_dropped = df_stats.shape[0] - df.shape[0]
pprint('No. of dropped rows: ' + str(rows_dropped))
```

```
'No. of dropped rows: 81'
```

In [49]:

```
df.loc[df['Ind_Nationality_Code'].isnull()]
```

Out[49]:

	Co_Nplay_Typ_Id	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Stack_Typ_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Prod_Typ_Id	Cust_Seg_Id	Cust_Class_Id	Party_Typ_Id	Cust_Hier_Typ_Id	Ind_Gender	Ind_Birth_Dt	Ind_Age	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Cust_Lifecycle_Stat_Id	Cust_Lifecycle_Typ_Id	Cust_Stat_Id	First_No_Pay_Dt	Last_no_pay_Dt	Bad_Pay_Count	Flag_Last_6_Month
429018	1PMoPost	2018-07-29	ACTIVATED	2001-12-11	35608609	N	80.0	80.0	207	epb-AAA_BBB_S	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	F	1977-12-07	41.0	NaN	EN	EN	Current Customer	Current	Active	NaN	NaN	NaN	NaN
462553	1PMoPost	2019-04-02	ACTIVATED	2015-08-27	35782108	N	80.0	80.0	43	5-30HS1	Y	Bundle	Recurring	Inventory	1300	E	Ind	Master	M	1975-07-21	43.0	NaN	EN	EN	Current Customer	Current	Active	NaN	NaN	NaN	NaN

In [50]:

```
# Closer look at records where Ind_Nationality_Code is NULL, maybe a "standard" can be set
df.loc[(df['Written_Language_Code'] == 'EN') & (df['Oral_Language_Code'] == 'EN')]
```

OUTPUT REMOVED (ANONYMIZATION)

Finding:

To many different Ind_Nationality_Code -> drop rows Ind_Nationality_Code = NULL

In [51]:

```
# Drop rows Ind_Nationality_Code = NULL
df_stats = df
df = df.dropna(subset=['Ind_Nationality_Code'])

rows_dropped = df_stats.shape[0] - df.shape[0]
pprint('No. of dropped rows: ' + str(rows_dropped))
```

```
'No. of dropped rows: 2'
```

In [52]:

```
# Check out Bad_Pay_Count and Flag_Last_6_Month

pprint(df['Bad_Pay_Count'].value_counts(dropna = False))
pprint(df['Flag_Last_6_Month'].value_counts(dropna = False))
```

```
NaN      507330
 1.0       2966
 2.0       1917
 3.0       1258
 4.0        761
 5.0        460
 6.0        250
 7.0        110
 8.0         41
 9.0         26
 11.0         7
 10.0         2
Name: Bad_Pay_Count, dtype: int64
NaN     507330
 1.0      5850
 0.0      1948
Name: Flag_Last_6_Month, dtype: int64
```

In [53]:

```
# Change Bad_Pay_Count and Flag_Last_6_Month from NaN to 0
```

```
# Change data type to integer

df['Bad_Pay_Count'].fillna(0, inplace = True)
df['Flag_Last_6_Month'].fillna(0, inplace = True)

df['Bad_Pay_Count'] = df.Bad_Pay_Count.astype(int)
df['Flag_Last_6_Month'] = df.Flag_Last_6_Month.astype(int)
```

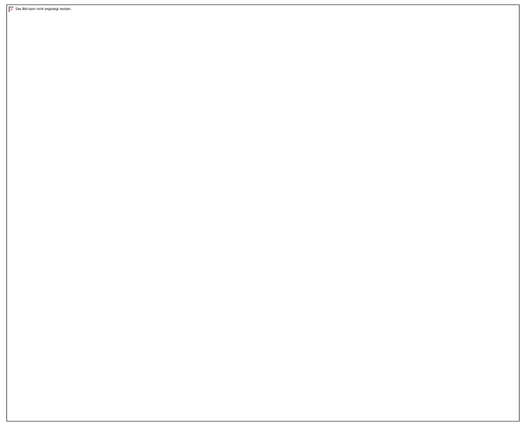
In [54]:

```
# Replace NaN from transaction data with 0

df['NF_Num_Transactions'].fillna(0, inplace = True)
df['GROSS_PRICE_AMT_mean'].fillna(0, inplace = True)
df['GROSS_PRICE_AMT_sum'].fillna(0, inplace = True)
df['GROSS_PRICE_AMT_max'].fillna(0, inplace = True)
df['GROSS_PRICE_AMT_min'].fillna(0, inplace = True)
df['REV_EFF_TS_max'].fillna(0, inplace = True)
df['REV_EFF_TS_min'].fillna(0, inplace = True)
df['HAS_E_pay'].fillna(0, inplace = True)
df['HAS_N_Pay'].fillna(0, inplace = True)
df['HAS_UNKNOWN'].fillna(0, inplace = True)
```

Flag_Last_6_Month Distribution

In [55]:

```
# Show Flag_Last_6_Month information

didnt_pay = df['Flag_Last_6_Month'].value_counts()[1]
paid = df['Flag_Last_6_Month'].value_counts()[0]

didnt_pay_per = didnt_pay / df.shape[0] * 100
paid_per = paid / df.shape[0] * 100

plt.figure(figsize=(5, 4))
sns.countplot(df['Flag_Last_6_Month'])

plt.xlabel('Flag_Last_6_Month', size=15, labelpad=15)
plt.ylabel('Mobile Users', size=15, labelpad=15)
plt.xticks((0, 1), ['paid ({0:.2f}%)'.format(paid_per), 'didnt_pay ({0:.2f}%)'.format(didnt_pay_per)])
plt.tick_params(axis='x', labelsize=13)
plt.tick_params(axis='y', labelsize=13)

plt.title('Flag_Last_6_Month Distribution', size=15, y=1.05)
plt.show()

print('{} of {} mobile users did not pay in the last 6 months -> {:.2f}% of the dataset'.format(didnt_pay, df.shape[0], didnt_pay_per))
print('{} of {} mobile users paid in the last 6 months -> {:.2f}% of the dataset'.format(paid, df.shape[0], paid_per))
```




```
5850 of 515128 mobile users did not pay in the last 6 months -> 1.14% of the dataset
509278 of 515128 mobile users paid in the last 6 months -> 98.86% of the dataset
```

Finding:

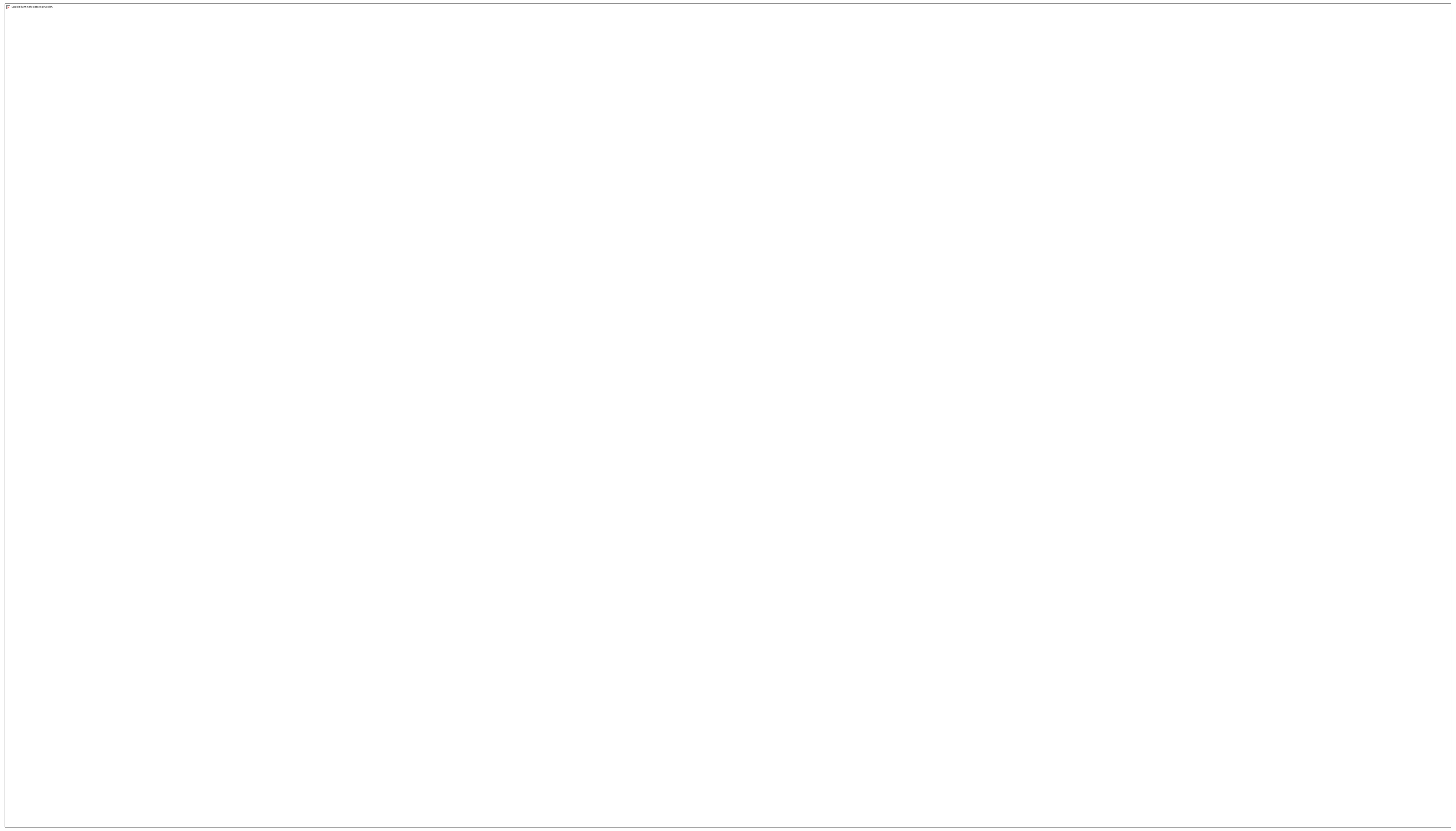
Highly Imbalanced Data!! -> must be addressed when defining the models

In [56]:

```
# Function to show Flag_Last_6_Month Distribution for numerical features

def show_dist_num(dataframe, col1, col2):
    cont_features = [col1, col2]

    flag = dataframe['Flag_Last_6_Month'] == 1

    fig, axs = plt.subplots(ncols=2, nrows=2, figsize=(20, 20))
    plt.subplots_adjust(right=1.5)

    for i, feature in enumerate(cont_features):
        # Distribution of Flag_Last_6_Month in feature
        sns.distplot(dataframe[~flag][feature], label='paid', hist=True, color='#2ecc71', ax=axs[0][i])
        sns.distplot(dataframe[flag][feature], label='did not pay', hist=True, color='#e74c3c', ax=axs[0][i])

        # Distribution of feature in dataset
        sns.distplot(dataframe[feature], label='Demographics Data', hist=False, color='#e74c3c', ax=axs[1][i])
        #sns.distplot(df_test[feature], label='Test Set', hist=False, color='#2ecc71', ax=axs[1][i])

        axs[0][i].set_xlabel('')
        axs[1][i].set_xlabel('')

        for j in range(2):
            axs[i][j].tick_params(axis='x', labelsize=20)
            axs[i][j].tick_params(axis='y', labelsize=20)

        axs[0][i].legend(loc='upper right', prop={'size': 20})
        axs[1][i].legend(loc='upper right', prop={'size': 20})
        axs[0][i].set_title('Distribution of Flag_Last_6_Month in {}'.format(feature), size=20, y=1.05)

    axs[1][0].set_title('Distribution of {} Feature'.format(col1), size=20, y=1.05)
    axs[1][1].set_title('Distribution of {} Feature'.format(col2), size=20, y=1.05)

    plt.show()
```

In [57]:

```
# Show distribution
show_dist_num(df, 'Cust_Seg_Id', 'Ind_Age')
```



In [58]:

```
df_demo_cust_seg = df[["Cust_Seg_Id","Flag_Last_6_Month"]].groupby('Cust_Seg_Id').mean()
df_demo_cust_seg.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[58]:

	Flag_Last_6_Month
Cust_Seg_Id	
1203	0.019283
1204	0.019035
1102	0.012149
1202	0.011677
1103	0.010753
1300	0.008684
1101	0.006751
1400	0.001654

In [59]:

```
df_demo_age = df[["Ind_Age","Flag_Last_6_Month"]].groupby('Ind_Age').mean()
df_demo_age.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[59]:

	Flag_Last_6_Month
Ind_Age	
27.0	0.021465
22.0	0.021289
26.0	0.021002
28.0	0.020651
23.0	0.020155
25.0	0.019068
29.0	0.018855
24.0	0.018474
31.0	0.018098
20.0	0.017896

Findings:

Cust_Seg_Id = 1200 has the most Users but also above average non payers -> investigate segments later
Ind_Age between approx. 20 - 30 has the most non payers

In [60]:

```
# Show distribution
show_dist_num(df, 'List_Recurring_Chrg_Amt', 'Actual_Recurring_Chrg_Amt')
```

Das Bild kann nicht angezeigt werden.

In [61]:

```
df_demo_act_amt = df[["Actual_Recurring_Chrg_Amt","Flag_Last_6_Month"]].groupby('Actual_Recurring_Chrg_Amt').mean()
df_demo_act_amt.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[61]:

	Flag_Last_6_Month
Actual_Recurring_Chrg_Amt	
0.0	0.033333
89.0	0.031443
140.0	0.028992
200.0	0.025287
100.0	0.023721
69.0	0.021330
139.0	0.020105
199.0	0.018568
169.0	0.018256
10.0	0.015385

In [62]:

```
df_demo_list_amt = df[["List_Recurring_Chrg_Amt","Flag_Last_6_Month"]].groupby('List_Recurring_Chrg_Amt').mean()
df_demo_list_amt.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[62]:

	Flag_Last_6_Month
List_Recurring_Chrg_Amt	
0.0	0.034483
89.0	0.031433
140.0	0.028992
200.0	0.025280
100.0	0.023720
69.0	0.021330
139.0	0.020105
199.0	0.018568
169.0	0.018256
99.0	0.015308

Findings:

Most non payers have a mobile subscription fee of just above CHF 100
List_Recurring_Chrg_Amt and Actual_Recurring_Chrg_Amt hardly differ -> drop List_Recurring_Chrg_Amt later

In [63]:

```
# Show distribution
show_dist_num(df, 'NF_Num_Transactions', 'GROSS_PRICE_AMT_mean')
```

Das Bild kann nicht angezeigt werden.

In [64]:

```
df_demo_num_trx = df[["NF_Num_Transactions","Flag_Last_6_Month"]].groupby('NF_Num_Transactions').mean()
df_demo_num_trx.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[64]:

	Flag_Last_6_Month
NF_Num_Transactions	
2921.0	1.0
1232.0	1.0
1055.0	1.0
550.0	1.0
743.0	1.0
652.0	1.0
523.0	1.0
650.0	1.0
439.0	1.0
445.0	0.5

In [65]:

```
df_demo_amt_mean = df[["GROSS_PRICE_AMT_mean","Flag_Last_6_Month"]].groupby('GROSS_PRICE_AMT_mean').mean()
df_demo_amt_mean.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[65]:

	Flag_Last_6_Month
GROSS_PRICE_AMT_mean	
11.5128	1.0
23.9323	1.0
13.9533	1.0
11.0118	1.0
24.8800	1.0
24.8878	1.0
24.8971	1.0
24.9067	1.0
9.1763	1.0
11.9865	1.0

In [66]:

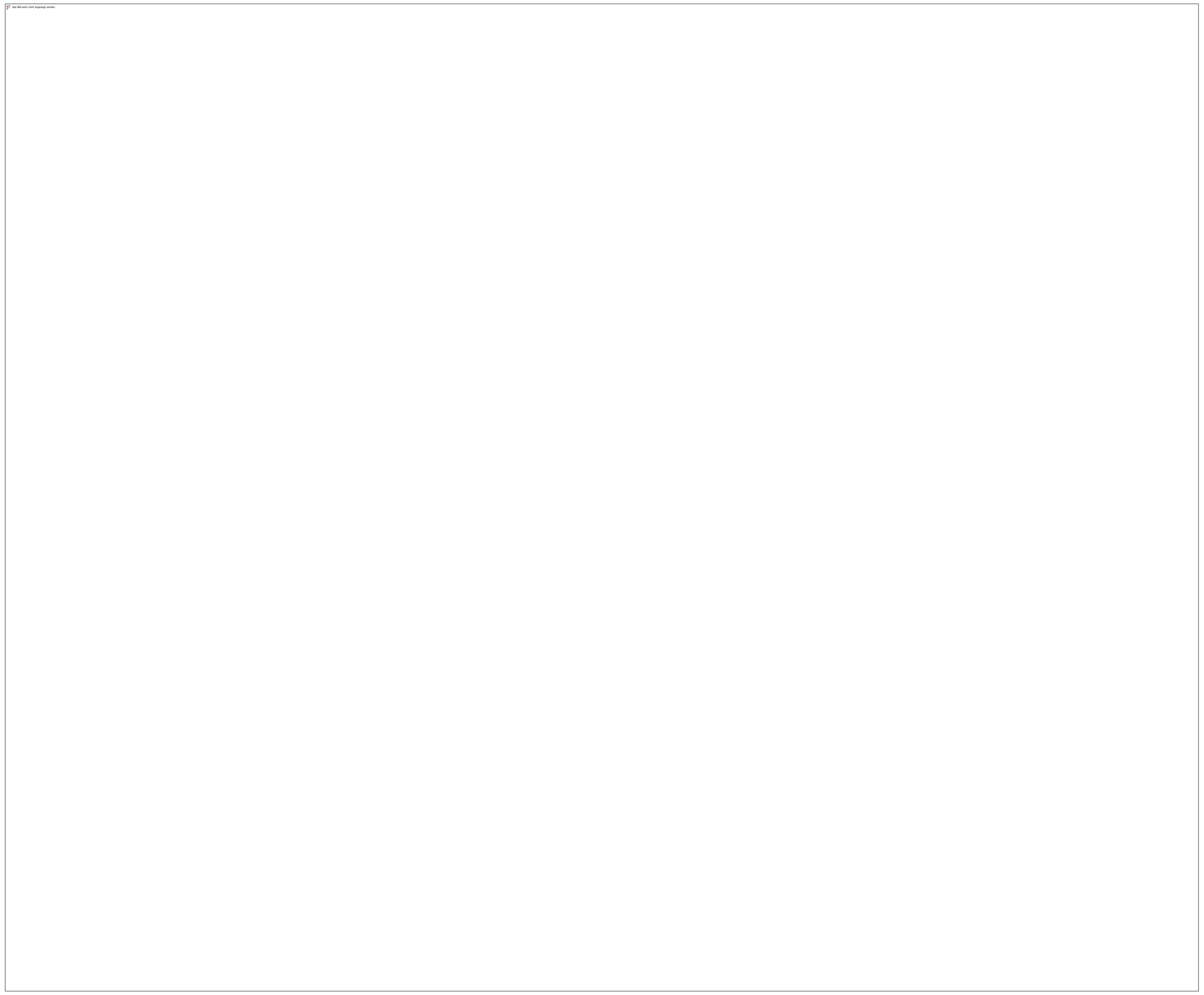
```
# Function to show Flag_Last_6_Month Distribution for categorical features

def show_dist_cat(dataframe, cols):
    cat_features = cols

    fig, axs = plt.subplots(ncols=2, nrows=3, figsize=(20, 20))
    plt.subplots_adjust(right=1.5, top=1.25)

    for i, feature in enumerate(cat_features, 1):
        plt.subplot(2, 3, i)
        sns.countplot(x=feature, hue='Flag_Last_6_Month', data=dataframe)

        plt.xlabel('{}'.format(feature), size=20, labelpad=15)
        plt.ylabel('Mobile User Count', size=20, labelpad=15)
        plt.tick_params(axis='x', labelsize=20)
        plt.tick_params(axis='y', labelsize=20)

        plt.legend(['payed', 'did not pay'], loc='upper center', prop={'size': 18})
        plt.title('Count of Flag_Last_6_Month in {} Feature'.format(feature), size=20, y=1.05)

    plt.show()
```

In [67]:

```
# Function to show Flag_Last_6_Month Distribution for single feature

def show_dist_cat_single(dataframe, col):

    fig, axs = plt.subplots(figsize=(22, 9))
    sns.countplot(x=col, hue='Flag_Last_6_Month', data=dataframe)

    plt.xlabel(col, size=15, labelpad=20)
    plt.ylabel('Mobile User Count', size=15, labelpad=20)
    plt.tick_params(axis='x', labelsize=15)
    plt.tick_params(axis='y', labelsize=15)

    plt.legend(['payed', 'did not pay'], loc='upper right', prop={'size': 15})
    plt.title('Count of Flag_Last_6_Month in {} Feature'.format(col), size=15, y=1.05)

    plt.show()
```

In [68]:

```
# Show distribution for categorical features
# Function can take max. 6 features as input

cat_cols = ['Subscr_Since_Dt', 'Tac_Id', 'Subs_Age_Months', 'Prod_Id', 'Subs_Stat_Id', 'Bad_Pay_Count']
show_dist_cat(df, cat_cols)
```



In [69]:

```
# Show distribution for categorical features
# Function can take max. 6 features as input

cat_cols = ['Ind_Gender', 'Ind_Nationality_Code', 'Written_Language_Code', 'Oral_Language_Code']
show_dist_cat(df, cat_cols)
```

Das Bild kann nicht angezeigt werden.

In [70]:

```
df_gender = df[["Ind_Gender","Flag_Last_6_Month"]].groupby('Ind_Gender').mean()
df_gender.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[70]:

	Flag_Last_6_Month
Ind_Gender	
M	0.014289
F	0.008355
U	0.005102

In [71]:

```
df_nat = df[["Ind_Nationality_Code","Flag_Last_6_Month"]].groupby('Ind_Nationality_Code').mean()
df_nat.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[71]:

	Flag_Last_6_Month
Ind_Nationality_Code	
AZ	0.130435
NI	0.100000
TG	0.080000
LA	0.062500
AO	0.052632
BO	0.050505
MN	0.047619
DO	0.045603
GM	0.045455

	Flag_Last_6_Month
Ind_Nationality_Code	
SN	0.045455

In [72]:

```
df_wlc = df[["Written_Language_Code","Flag_Last_6_Month"]].groupby('Written_Language_Code').mean()
df_wlc.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[72]:

	Flag_Last_6_Month
Written_Language_Code	
EN	0.011874
DE	0.011598
FR	0.011025
IT	0.008871

In [73]:

```
df_olc = df[["Oral_Language_Code","Flag_Last_6_Month"]].groupby('Oral_Language_Code').mean()
df_olc.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[73]:

	Flag_Last_6_Month
Oral_Language_Code	
ES	0.076923
98	0.043478
EN	0.015264
DE	0.010090
FR	0.009614
IT	0.007642
DA	0.000000
EL	0.000000
PT	0.000000
SV	0.000000

In [74]:

```
# Show map
# temp, not finished, delete 0 X, Y coordinate

df_hect = df.copy()
# remove outliers
df_hect = df_hect.drop(df_hect[(df_hect.Hectare_Cell_X_Coordinate < 400000) & (df_hect.Hectare_Cell_Y_Coordinate < 50000)].index)
df_hect.plot(kind="scatter", x='Hectare_Cell_X_Coordinate', y='Hectare_Cell_Y_Coordinate', alpha=0.4)

#df_demo_hect = df_demo.copy()
#df_demo_hect.plot(kind="scatter", x='Hectare_Cell_X_Coordinate', y='Hectare_Cell_Y_Coordinate', alpha=0.4,
#             s=df_demo_hect['Flag_Last_6_Months'), label="flag", figsize=(10,7),
#             c='Flag_Last_6_Months', cmap=plt.get_cmap("jet"), colorbar=True,
#)
#plt.legend()
```

Out[74]:

Population Density Map

Finding:

Data has pretty much the same distribution as population density and is evenly distributed across the country

Correlations

In [75]:

```
# Correlations

df_corr = df.drop(['Main_Phone_Num'], axis=1).corr().abs().unstack().sort_values(kind="quicksort", ascending=False).reset_index()
df_corr.rename(columns={"level_0": "Feature 1", "level_1": "Feature 2", 0: 'Correlation Coefficient'}, inplace=True)
df_corr.drop(df_corr.iloc[1::2].index, inplace=True)
df_corr_nd = df_corr.drop(df_corr[df_corr['Correlation Coefficient'] == 1.0].index)
```

In [76]:

```
# Show highly correlated features

corr = df_corr_nd['Correlation Coefficient'] > 0.1
df_corr_nd[corr]
```

Out[76]:

	Feature 1	Feature 2	Correlation Coefficient
18	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	0.999982
20	Ind_Age	Cust_Seg_Id	0.854754
22	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_min	0.800345
24	Flag_Last_6_Month	Bad_Pay_Count	0.787465
26	GROSS_PRICE_AMT_sum	HAS_N_Pay	0.774242
28	HAS_N_Pay	NF_Num_Transactions	0.745932
30	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_mean	0.737396
32	NF_Num_Transactions	GROSS_PRICE_AMT_sum	0.709506
34	HAS_E_pay	NF_Num_Transactions	0.686664
36	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	0.608585
38	Hectare_Cell_X_Coordinate	Hectare_Cell_Y_Coordinate	0.578420
40	Subs_Age_Months	Ind_Age	0.506452

	Feature 1	Feature 2	Correlation Coefficient
42	HAS_N_Pay	GROSS_PRICE_AMT_max	0.501122
44	Cust_Seg_Id	Subs_Age_Months	0.450995
46	GROSS_PRICE_AMT_max	NF_Num_Transactions	0.390063
48	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	0.346199
50	GROSS_PRICE_AMT_min	GROSS_PRICE_AMT_max	0.338134
52	GROSS_PRICE_AMT_max	Ind_Age	0.241065
54	Cust_Seg_Id	GROSS_PRICE_AMT_max	0.228499
56	GROSS_PRICE_AMT_sum	HAS_E_pay	0.221269
58	HAS_N_Pay	GROSS_PRICE_AMT_mean	0.200127
60	Cust_Seg_Id	Actual_Recurring_Chrg_Amt	0.192441
62	Cust_Seg_Id	List_Recurring_Chrg_Amt	0.192414
64	Ind_Age	GROSS_PRICE_AMT_mean	0.179273
66	Cust_Seg_Id	GROSS_PRICE_AMT_mean	0.167413
68	GROSS_PRICE_AMT_max	Actual_Recurring_Chrg_Amt	0.162362
70	GROSS_PRICE_AMT_max	List_Recurring_Chrg_Amt	0.162346
72	HAS_N_Pay	Ind_Age	0.159184
74	HAS_N_Pay	Bad_Pay_Count	0.154368
76	GROSS_PRICE_AMT_sum	Bad_Pay_Count	0.153594
78	HAS_N_Pay	Cust_Seg_Id	0.151923
80	Ind_Age	Actual_Recurring_Chrg_Amt	0.149450
82	Ind_Age	List_Recurring_Chrg_Amt	0.149412
84	Actual_Recurring_Chrg_Amt	GROSS_PRICE_AMT_sum	0.145044
86	GROSS_PRICE_AMT_sum	List_Recurring_Chrg_Amt	0.145036
88	NF_Num_Transactions	GROSS_PRICE_AMT_mean	0.144494
90	GROSS_PRICE_AMT_max	Subs_Age_Months	0.144222
92	Flag_Last_6_Month	HAS_N_Pay	0.140798
94	Actual_Recurring_Chrg_Amt	NF_Num_Transactions	0.140743
96	List_Recurring_Chrg_Amt	NF_Num_Transactions	0.140734
98	GROSS_PRICE_AMT_max	Bad_Pay_Count	0.140702
100	Cust_Seg_Id	GROSS_PRICE_AMT_sum	0.140339
102	Ind_Age	GROSS_PRICE_AMT_sum	0.140292
104	Flag_Last_6_Month	GROSS_PRICE_AMT_sum	0.139701
106	HAS_N_Pay	Actual_Recurring_Chrg_Amt	0.138598
108	HAS_N_Pay	List_Recurring_Chrg_Amt	0.138588
110	Flag_Last_6_Month	GROSS_PRICE_AMT_max	0.133237
112	NF_Num_Transactions	Bad_Pay_Count	0.122870
114	NF_Num_Transactions	Flag_Last_6_Month	0.114615
116	GROSS_PRICE_AMT_mean	Actual_Recurring_Chrg_Amt	0.110896
118	GROSS_PRICE_AMT_mean	List_Recurring_Chrg_Amt	0.110881
120	NF_Num_Transactions	Cust_Seg_Id	0.105571
122	NF_Num_Transactions	Ind_Age	0.101240

Finding:

- Lots of features are correlated > 0.1

- Very high correlation between Bad_Pay_Count and target Flag_Last_6_Month. I suspect data leakage -> drop Bad_Pay_Count and related columns

In [77]:

```
# Correlation with target Flag_Last_6_Month
corr_matrix = df.corr()
corr_matrix['Flag_Last_6_Month'].sort_values(ascending=False)
```

Out[77]:

```
Flag_Last_6_Month            1.000000
Bad_Pay_Count                0.787465
HAS_N_Pay                0.140798
GROSS_PRICE_AMT_sum          0.139701
GROSS_PRICE_AMT_max          0.133237
NF_Num_Transactions          0.114615
GROSS_PRICE_AMT_mean         0.065515
Actual_Recurring_Chrg_Amt    0.056371
List_Recurring_Chrg_Amt      0.056368
HAS_E_pay                  0.017683
HAS_UNKNOWN                  0.015354
Hectare_Cell_Y_Coordinate    0.006350
Main_Phone_Num               0.001651
Hectare_Cell_X_Coordinate   -0.001539
GROSS_PRICE_AMT_min         -0.003434
Subs_Age_Months             -0.045191
Cust_Seg_Id                 -0.058158
Ind_Age                     -0.061621
Name: Flag_Last_6_Month, dtype: float64
```

Finding:

- Little correlation except for Bad_Pay_Count!
- New features have higher correlation than already existing features

In [78]:

```
# Correlation Heatmap

fig, axs = plt.subplots(nrows=2, figsize=(25, 25))

sns.heatmap(df.drop(['Main_Phone_Num'], axis=1).corr(), ax=axs[0], annot=True, square=True, cmap='coolwarm', annot_kws={'size': 9})

for i in range(2):
    axs[i].tick_params(axis='x', labelsize=12)
    axs[i].tick_params(axis='y', labelsize=12)

axs[0].set_title('Correlations', size=15)

#plt.show()
```

Out[78]:

```
Text(0.5, 1.0, 'Correlations')
```

Das Bild kann nicht angezeigt werden.

Data Preprocessing

Prepare data for "categorical friendly models" (more preprocessing for other models later)

Drop Columns with unique values

In [79]:

```
# Backup
df_backup_2 = df.copy()
```

In [80]:

In [83]:

```
# Drop all columns that have only one value

no_cols = len(df.columns)

for col in df.columns:
    if len(df[col].unique()) == 1:
        df.drop(col,inplace=True,axis=1)
        pprint('dropped ' + col)

no_cols_new = no_cols - len(df.columns)
pprint('No. of dropped columns: ' + str(no_cols_new))
```

```
'No. of dropped columns: 8'
```

In [84]:

```
df.info()
```

```
Int64Index: 515128 entries, 0 to 515210
Data columns (total 36 columns):
Main_Phone_Num                515128 non-null int64
Start_Dt                      515128 non-null object
Subs_Stat_Id                  515128 non-null object
Subscr_Since_Dt               515128 non-null object
Tac_Id                        515128 non-null object
Stack_Typ_Id                  515128 non-null object
Hectare_Cell_X_Coordinate     515128 non-null int64
Hectare_Cell_Y_Coordinate     515128 non-null int64
List_Recurring_Chrg_Amt       515128 non-null float64
Actual_Recurring_Chrg_Amt     515128 non-null float64
Subs_Age_Months               515128 non-null int64
Prod_Id                       515128 non-null object
Reg_Relevant_Flag             515128 non-null object
Prod_Item_Typ_Id              515128 non-null object
Price_Typ_Id                  515128 non-null object
Cust_Seg_Id                   515128 non-null int64
Ind_Gender                    515128 non-null object
Ind_Birth_Dt                  515128 non-null object
Ind_Age                       515128 non-null float64
Ind_Nationality_Code          515128 non-null object
Written_Language_Code         515128 non-null object
Oral_Language_Code            515128 non-null object
First_No_Pay_Dt               7798 non-null object
Last_no_pay_Dt                7798 non-null object
Bad_Pay_Count                 515128 non-null int64
Flag_Last_6_Month             515128 non-null int64
NF_Num_Transactions           515128 non-null float64
GROSS_PRICE_AMT_mean          515128 non-null float64
GROSS_PRICE_AMT_sum           515128 non-null float64
GROSS_PRICE_AMT_max           515128 non-null float64
GROSS_PRICE_AMT_min           515128 non-null float64
REV_EFF_TS_max                515128 non-null object
REV_EFF_TS_min                515128 non-null object
HAS_E_pay                   515128 non-null float64
HAS_N_Pay                 515128 non-null float64
HAS_UNKNOWN                   515128 non-null float64
dtypes: float64(11), int64(7), object(18)
memory usage: 165.4+ MB
```

Bad_Pay_Count

In [85]:

```
# Drop Bad_Pay_Count and related columns as they appear to support data leakage

df = df.drop('First_No_Pay_Dt', 1)
df = df.drop('Last_no_pay_Dt', 1)
df = df.drop('Bad_Pay_Count', 1)
```

Tac_Id

In [86]:

```
pprint(df['Tac_Id'].value_counts(dropna = False))
```

```
35680809    4515
35716409    4089
35280209    3731
35240209    3652
35904108    3617
35966409    3500
35263108    1962
            ...
35345008       1
35931008       1
Name: Tac_Id, Length: 6678, dtype: int64
```

Finding:

Too many different values for one-hot-encoding -> keep values (convert to Integer)

In [87]:

```
df.loc[(df.Tac_Id == 'UNKNOWN')]

value is trying
```

Out[87]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Stack_Typ_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Birth_Dt	Ind_Age	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean
48588	2017-09-22	ACTIVATED	2012-07-11	UNKNOWN	N	55.0	55.0	80	5-2E1XS	Y	Bundle	Recurring	1203	M	1994-07-20	24.0	PT	DE	EN	0	0.0	0.0
75638	2017-11-25	ACTIVATED	2012-02-16	UNKNOWN	N	35.0	35.0	85	epb-AAA_BBB_light	Y	Bundle	Recurring	1400	M	1959-04-22	59.0	NL	DE	EN	0	0.0	0.0
140179	2018-10-26	ACTIVATED	1996-06-19	UNKNOWN	M	25.0	25.0	273	5-Q6NV	Y	Subscription	Recurring	1400	M	1953-09-21	65.0	CH	DE	DE	0	0.0	0.0
149410	2016-01-27	ACTIVATED	2010-03-31	UNKNOWN	N	9.8	9.8	108	5-1KB2E	Y	Bundle	Recurring	1400	M	1948-10-22	70.0	CH	FR	FR	0	0.0	0.0
160239	2019-01-09	ACTIVATED	2007-08-28	UNKNOWN	N	19.0	19.0	139	5-2E1UL	Y	Bundle	Recurring	1102	M	1969-05-04	49.0	IT	FR	FR	0	0.0	0.0
161258	2017-11-23	ACTIVATED	2009-12-03	UNKNOWN	N	35.0	35.0	112	epb-AAA_BBB_light	Y	Bundle	Recurring	1300	M	1979-07-07	39.0	PT	FR	FR	0	0.0	0.0
170524	2017-12-15	ACTIVATED	2005-02-07	UNKNOWN	N	10.0	5.0	169	5-2E1U1	Y	Bundle	Recurring	1400	M	1961-02-02	58.0	CH	DE	DE	0	0.0	0.0

76 rows × 33 columns

In [88]:

```
# Change UNKNOWN to 99999999 and convert data type

df.loc[(df.Tac_Id == 'UNKNOWN'), 'Tac_Id'] = 99999999
df['Tac_Id'] = df.Tac_Id.astype(int)
```

Ind_Age, Ind_Birth_Dt and Cust_Seg_Id

In [89]:

```
# Age
pprint(df['Ind_Age'].max())
pprint(df['Ind_Age'].min())
```

```
118.0
13.0
```

In [90]:

```
# Check out Age > 105 (oldest living person in Switzerland is currently 116 years old, so 118 cannot be true...)
df.loc[(df.Ind_Age > 105), ['Ind_Birth_Dt', 'Ind_Age', 'Cust_Seg_Id']]
```

Out[90]:

	Ind_Birth_Dt	Ind_Age	Cust_Seg_Id
48656	1909-11-01	109.0	1300
79399	1910-01-01	109.0	1300
178939	1909-04-26	109.0	1300
203863	1901-01-01	118.0	1300
224644	1909-08-03	109.0	1300
255269	1901-01-01	118.0	1300
329135	1908-10-29	110.0	1300
345398	1911-02-05	108.0	1300
350122	1913-01-01	106.0	1300

Finding:

- Ind_Age 118 and Ind_Birth_Dt 1901-01-01 looks like wrong data -> drop rows
- Other birthdates YYYY-01-01 also look suspicious but will leave them

In [91]:

```
# Drop rows age > 116 and birthdate 1901-01-01
df_stats = df
df = df.drop(df[(df.Ind_Birth_Dt =='1901-01-01') & (df.Ind_Age > 116)].index)
rows_dropped = df_stats.shape[0] - df.shape[0]
pprint('No. of dropped rows: ' + str(rows_dropped))
```

```
'No. of dropped rows: 2'
```

In [92]:

```
# Closer look at Cust_Seg and Age
df.groupby(
    ['Cust_Seg_Id']
).agg(
    {
        'Ind_Age': [min, max],
        'Cust_Seg_Id': "count"
```

```
    }
)
```

Out[92]:

	Ind_Age		Cust_Seg_Id
	min	max	count
Cust_Seg_Id			
1101	17.0	95.0	2518
1102	15.0	95.0	8231
1103	23.0	84.0	93
1202	13.0	20.0	35881
1203	20.0	27.0	148317
1204	27.0	31.0	45233
1300	31.0	110.0	161793
1400	55.0	100.0	113060

Finding:

- 1101, 1102 and 1103 don't make sense -> leave it for now
- 1300 should only go up to 55 -> further investigation

Best approach might be to create new age group feature and drop Cust_Seg_Id

In [93]:

```
# Closer look at Cust_Seg_Id, check birthdate for dummy values
df.loc[(df.Cust_Seg_Id == 1300) & (df.Ind_Age > 55), ['Ind_Birth_Dt', 'Ind_Age']]
```

Out[93]:

	Ind_Birth_Dt	Ind_Age
48656	1909-11-01	109.0
79399	1910-01-01	109.0
108266	1917-12-28	101.0
178939	1909-04-26	109.0
194752	1915-07-02	103.0
213469	1915-07-02	103.0
224644	1909-08-03	109.0
285727	1917-09-01	101.0
289469	1914-12-24	104.0
293001	1913-07-11	105.0
297664	1916-08-11	102.0
314305	1915-09-26	103.0
329135	1908-10-29	110.0
345398	1911-02-05	108.0
350122	1913-01-01	106.0

Finding:

Looks like wrong Cust_Seg_Id classification -> change to Seg 1400

In [94]:

```
# Change Cust_Seg_Id
df.loc[(df.Cust_Seg_Id == 1300) & (df.Ind_Age > 55), 'Cust_Seg_Id'] = 1400
```

In [95]:

```
# Check Cust_Seg and Age again
df.groupby(
    ['Cust_Seg_Id']
).agg(
    {
        'Ind_Age': [min, max],
        'Cust_Seg_Id': "count"
    }
)
```

Out[95]:

	Ind_Age		Cust_Seg_Id
	min	max	count
Cust_Seg_Id			
1101	17.0	95.0	2518
1102	15.0	95.0	8231
1103	23.0	84.0	93
1202	13.0	20.0	35881
1203	20.0	27.0	148317
1204	27.0	31.0	45233
1300	31.0	55.0	161778
1400	55.0	110.0	113075

In [96]:

```
# Drop Ind_Age and Ind_Birth_Dt columns (not necessary anymore)
df = df.drop('Ind_Age', 1)
df = df.drop('Ind_Birth_Dt', 1)
```

In [97]:

```
df.head()
```

Out[97]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Stack_Typ_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS AMT
0	2019-03-16	ACTIVATED	2011-03-02	35608109	N	69.0	69.0	97	5-2E1WT	Y	Bundle	Recurring	1203	F	CH	DE	DE	1	8.0	6.4750	51.80
1	2019-03-09	ACTIVATED	2015-12-21	35736109	N	35.0	35.0	39	5-2CEGS	Y	Bundle	Recurring	1202	M	BA	DE	DE	0	20.0	17.7900	355.80
2	2018-07-01	ACTIVATED	2015-11-27	35460707	N	59.0	59.0	40	5-2E1W4	Y	Bundle	Recurring	1300	F	CH	FR	FR	0	0.0	0.0000	0.00
3	2018-08-01	ACTIVATED	2012-10-04	35240209	N	69.0	69.0	77	5-2E1WT	Y	Bundle	Recurring	1203	F	CH	DE	DE	0	30.0	2.7667	83.00
4	2018-11-13	ACTIVATED	2015-10-13	35721309	N	55.0	55.0	41	5-2E1XN	Y	Bundle	Recurring	1203	F	CH	DE	EN	0	17.0	4.9900	84.83

In [98]:

```
df.info()
```

```
Int64Index: 515126 entries, 0 to 515210
```

```
Data columns (total 31 columns):
Main_Phone_Num              515126 non-null int64
Start_Dt                    515126 non-null object
Subs_Stat_Id                515126 non-null object
Subscr_Since_Dt             515126 non-null object
Tac_Id                      515126 non-null int64
Stack_Typ_Id                515126 non-null object
Hectare_Cell_X_Coordinate   515126 non-null int64
Hectare_Cell_Y_Coordinate   515126 non-null int64
List_Recurring_Chrg_Amt     515126 non-null float64
Actual_Recurring_Chrg_Amt   515126 non-null float64
Subs_Age_Months             515126 non-null int64
Prod_Id                     515126 non-null object
Reg_Relevant_Flag           515126 non-null object
Prod_Item_Typ_Id            515126 non-null object
Price_Typ_Id                515126 non-null object
Cust_Seg_Id                 515126 non-null int64
Ind_Gender                  515126 non-null object
Ind_Nationality_Code        515126 non-null object
Written_Language_Code       515126 non-null object
Oral_Language_Code          515126 non-null object
Flag_Last_6_Month           515126 non-null int64
NF_Num_Transactions         515126 non-null float64
GROSS_PRICE_AMT_mean        515126 non-null float64
GROSS_PRICE_AMT_sum         515126 non-null float64
GROSS_PRICE_AMT_max         515126 non-null float64
GROSS_PRICE_AMT_min         515126 non-null float64
REV_EFF_TS_max              515126 non-null object
REV_EFF_TS_min              515126 non-null object
HAS_E_pay                 515126 non-null float64
HAS_N_Pay               515126 non-null float64
HAS_UNKNOWN                 515126 non-null float64
dtypes: float64(10), int64(7), object(14)
memory usage: 145.8+ MB
```

Subs_Stat_Id

In [99]:

```
pprint(df['Subs_Stat_Id'].value_counts(dropna = False))
```

```
ACTIVATED         513320
POST_SUSPENDED      1806
Name: Subs_Stat_Id, dtype: int64
```

In [100]:

```
# Convert Subs_Stat_Id into 0 and 1
df["Subs_Stat_Id"] = df["Subs_Stat_Id"].map({"POST_SUSPENDED": 0, "ACTIVATED": 1})
```

Subs_Age_Months

In [101]:

```
pprint(df['Subs_Age_Months'].value_counts(dropna = False))
```

```
75     6362
87     5548
63     5398
```

```
77     5115
80     4894
79     4815
67     4751
76     4693
68     4661
51     4641
73     3852
       ...
297      41
Name: Subs_Age_Months, Length: 292, dtype: int64
```

Note:

Maybe create binning feature later

Stack_Typ_Id

In [102]:

```
pprint(df['Stack_Typ_Id'].value_counts(dropna = False))
```

```
N    515125
M         1
Name: Stack_Typ_Id, dtype: int64
```

In [103]:

```
# Drop Stack_Typ_Id as it has only 1 record differs from the others
df = df.drop('Stack_Typ_Id', 1)

# Alternative:
# Convert Stack_Typ_Id into 0 and 1
# df["Stack_Typ_Id"] = df["Stack_Typ_Id"].map({"M": 0, "N": 1})
```

List_Recurring_Chrg_Amt, Actual_Recurring_Chrg_Amt

In [104]:

```
# Check difference between Actual_Recurring_Chrg_Amt and List_Recurring_Chrg_Amt (discount)
df.loc[(df['Actual_Recurring_Chrg_Amt'] != df['List_Recurring_Chrg_Amt'] )]
```

Out[104]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS AMT
1814	2019-03-13	1	2007-10-01	35946907	80.0	79.00	138	5-30HRH	Y	Bundle	Recurring	1400	F	CH	DE	DE	0	0.0	0.0000	0.00
58593	2018-07-15	1	2013-12-09	35304709	80.0	79.00	63	epb-AAA_BBB_Pro_S	Y	Bundle	Recurring	1101	M	CH	DE	DE	0	33.0	11.8818	392.10
95785	2019-04-01	1	1995-03-17	35482609	99.0	79.00	288	5-2EKWN	Y	Bundle	Recurring	1400	M	CH	DE	DE	0	0.0	0.0000	0.00
144287	2018-01-19	1	2002-11-29	35926706	99.0	79.00	196	5-2EKWN	Y	Bundle	Recurring	1300	M	CH	DE	DE	0	0.0	0.0000	0.00

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	List_Recurring_Chrg_Amt	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS AMT
151135	2018-01-19	1	2013-06-10	1362900	99.0	79.00	69	5-2EKWN	Y	Bundle	Recurring	1300	M	CH	DE	EN	0	0.0	0.0000	0.00
166290	2018-10-20	1	1995-05-05	86417403	200.0	197.00	287	epb-AAA_BBB_Pro_XL	Y	Bundle	Recurring	1101	M	CH	DE	DE	0	1.0	0.9900	0.99
170524	2017-12-15	1	2005-02-07	99999999	10.0	5.00	169	5-2E1U1	Y	Bundle	Recurring	1400	M	CH	DE	DE	0	0.0	0.0000	0.00
188145	2019-02-11	1	1999-04-01	35500107	99.0	79.00	240	5-2EKWN	Y	Bundle	Recurring	1400	M	CH	DE	DE	0	0.0	0.0000	0.00
204867	2018-01-19	1	2000-01-21	35541507	99.0	79.00	230	5-2EKWN	Y	Bundle	Recurring	1400	M	CH	DE	DE	0	0.0	0.0000	0.00

75 rows × 30 columns

Finding:

Only a few users have different list and actual prices -> no further action

In [105]:

```
# Drop List_Recurring_Chrg_Amt as it highly correlates with Actual_Recurring_Chrg_Amt
df = df.drop('List_Recurring_Chrg_Amt', 1)
```

In [106]:

```
pprint(df['Actual_Recurring_Chrg_Amt'].value_counts(dropna = False))
```

```
80.00     115024
100.00     64458
69.00      53680
35.00      47567
59.00      42886
65.00      38089
99.00      24418
55.00      20470
29.00      18054
75.00      17635
79.00      15611
140.00      9244
49.00       8875
33.00       4770
45.00       4349
44.00       3808
19.80       3516
200.00      3480
89.00       3244
29.80       2182
129.00      2110
139.00      2089
179.00      1955
169.00      1479
9.80        1445
19.00       1069
```

```
39.00       907
12.00       827
199.00      754
25.00       575
15.00       310
34.00        92
10.00        65
5.00         47
0.00         30
68.00         2
8.00          2
18.00         2
40.18         1
197.00        1
98.00         1
88.00         1
53.00         1
30.80         1
Name: Actual_Recurring_Chrg_Amt, dtype: int64
```

In [107]:

```
df.loc[(df['Actual_Recurring_Chrg_Amt'] == 0)]
```

Out[107]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum
7382	2018-10-06	1	2013-12-12	35728009	0.0	63	epb-AAA_BBB_XTRA_S	Y	Bundle	Recurring	1203	M	CH	DE	EN	0	73.0	8.8808	648.30
26841	2018-09-06	1	2007-02-16	35942208	0.0	145	epb-AAA_BBB_S	Y	Bundle	Recurring	1300	F	CH	DE	DE	0	0.0	0.0000	0.00
101262	2019-04-02	1	2013-11-12	86626404	0.0	64	epb-AAA_BBB_XTRA_M	Y	Bundle	Recurring	1203	M	CH	DE	EN	0	38.0	17.7774	675.54
105958	2018-03-08	1	1999-03-30	35926006	0.0	240	epb-AAA_BBB_S	Y	Bundle	Recurring	1300	F	CH	DE	DE	0	0.0	0.0000	0.00
112372	2018-01-19	1	2003-09-04	35569507	0.0	187	epb-AAA_BBB_M	Y	Bundle	Recurring	1400	M	CH	DE	DE	0	4.0	20.5000	82.00
136525	2018-12-19	1	2014-05-05	35966409	0.0	59	epb-AAA_BBB_M	Y	Bundle	Recurring	1300	M	CH	DE	DE	0	39.0	3.6333	141.70
142830	2018-01-19	1	1997-08-28	35260109	0.0	259	epb-AAA_BBB_S	Y	Bundle	Recurring	1400	M	CH	DE	DE	0	0.0	0.0000	0.00
152856	2018-09-15	1	2000-06-15	35498709	0.0	225	epb-AAA_BBB_S	Y	Bundle	Recurring	1400	M	CH	DE	DE	0	0.0	0.0000	0.00

Finding:

Some Users seam not to pay any mobile subscribtion fee... -> leave it

Hectare_Cell_Coordinate

In [108]:

```
# Drop x/y coordinates due to lack of time to further explore
# Find location on https://map.geo.admin.ch
# https://www.geo.admin.ch/de/geo-dienstleistungen/datenbezug.html#ui-collapse-483

df = df.drop('Hectare_Cell_X_Coordinate', 1)
df = df.drop('Hectare_Cell_Y_Coordinate', 1)
```

Prod_Id

In [109]:

```
pprint(df['Prod_Id'].value_counts(dropna = False))
```

```
epb-AAA_BBB_M         48580
epb-AAA_BBB_S         47784
5-2E1WT                44513
                       ...
5-24CCV                    6
5-2E1UV                    6
epb-AAA_Home_Sxx          1
Name: Prod_Id, Length: 144, dtype: int64
```

Note:

Do some more preprocessing later

Reg_Relevant_Flag

In [110]:

```
pprint(df['Reg_Relevant_Flag'].value_counts(dropna = False))
```

```
Y    515117
N         9
Name: Reg_Relevant_Flag, dtype: int64
```

In [111]:

```
# Convert Reg_Relevant_Flag into 0 and 1
df["Reg_Relevant_Flag"] = df["Reg_Relevant_Flag"].map({"N": 0, "Y": 1})
```

Prod_Item_Typ_Id

In [112]:

```
pprint(df['Prod_Item_Typ_Id'].value_counts(dropna = False))
```

```
Bundle                      515116
Bundle Effective Product         8
Mobile Effective Product         1
Subscription                     1
Name: Prod_Item_Typ_Id, dtype: int64
```

In [113]:

```
# Convert Prod_Item_Typ_Id into 0 and 1
cleanup_prod_item = {"Prod_Item_Typ_Id":     {"Bundle Effective Product": 0,
    "Mobile Effective Product": 0,
    "Subscription": 0,
    "Bundle": 1}}
```

```
df = df.replace(cleanup_prod_item)
```

Price_Typ_Id

In [114]:

```
pprint(df['Price_Typ_Id'].value_counts(dropna = False))
```

```
Recurring    515117
Inventory         9
Name: Price_Typ_Id, dtype: int64
```

In [115]:

```
# Convert Price_Typ_Id into 0 and 1
df["Price_Typ_Id"] = df["Price_Typ_Id"].map({"Inventory": 0, "Recurring": 1})
```

In [116]:

```
df.info()
```

```
Int64Index: 515126 entries, 0 to 515210
Data columns (total 27 columns):
Main_Phone_Num              515126 non-null int64
Start_Dt                    515126 non-null object
Subs_Stat_Id                515126 non-null int64
Subscr_Since_Dt             515126 non-null object
Tac_Id                      515126 non-null int64
Actual_Recurring_Chrg_Amt   515126 non-null float64
Subs_Age_Months             515126 non-null int64
Prod_Id                     515126 non-null object
Reg_Relevant_Flag           515126 non-null int64
Prod_Item_Typ_Id            515126 non-null int64
Price_Typ_Id                515126 non-null int64
Cust_Seg_Id                 515126 non-null int64
Ind_Gender                  515126 non-null object
Ind_Nationality_Code        515126 non-null object
Written_Language_Code       515126 non-null object
Oral_Language_Code          515126 non-null object
Flag_Last_6_Month           515126 non-null int64
NF_Num_Transactions         515126 non-null float64
GROSS_PRICE_AMT_mean        515126 non-null float64
GROSS_PRICE_AMT_sum         515126 non-null float64
GROSS_PRICE_AMT_max         515126 non-null float64
GROSS_PRICE_AMT_min         515126 non-null float64
REV_EFF_TS_max              515126 non-null object
REV_EFF_TS_min              515126 non-null object
HAS_E_pay                 515126 non-null float64
HAS_N_Pay               515126 non-null float64
HAS_UNKNOWN                 515126 non-null float64
dtypes: float64(9), int64(9), object(9)
memory usage: 110.0+ MB
```

Ind_Gender

In [117]:

```
df['Ind_Gender'].value_counts(dropna = False)
```

Out[117]:

```
M    260751
F    253983
U       392
Name: Ind_Gender, dtype: int64
```

Finding:

Drop rows with Ind_Gender = 'U' (too much effort to clean data)

In [118]:

```
# Cleanup Ind_Gender

pprint(df['Ind_Gender'].unique())

# Drop unknown gender (U)
df_stats = df
df = df.drop(df[df.Ind_Gender =='U'].index)
rows_dropped = df_stats.shape[0] - df.shape[0]
pprint('No. of dropped rows: ' + str(rows_dropped))

# Convert gender into categorical value 0 and 1
df["Ind_Gender"] = df["Ind_Gender"].map({"M": 0, "F": 1})

pprint(df['Ind_Gender'].unique())
```

```
array(['F', 'M', 'U'], dtype=object)
'No. of dropped rows: 392'
array([1, 0])
```

In [119]:

```
df.head()
```

Out[119]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum
0	2019-03-16	1	2011-03-02	35608109	69.0	97	5-2E1WT	1	1	1	1203	1	CH	DE	DE	1	8.0	6.4750	51.80
1	2019-03-09	1	2015-12-21	35736109	35.0	39	5-2CEGS	1	1	1	1202	0	BA	DE	DE	0	20.0	17.7900	355.80
2	2018-07-01	1	2015-11-27	35460707	59.0	40	5-2E1W4	1	1	1	1300	1	CH	FR	FR	0	0.0	0.0000	0.00
3	2018-08-01	1	2012-10-04	35240209	69.0	77	5-2E1WT	1	1	1	1203	1	CH	DE	DE	0	30.0	2.7667	83.00
4	2018-11-13	1	2015-10-13	35721309	55.0	41	5-2E1XN	1	1	1	1203	1	CH	DE	EN	0	17.0	4.9900	84.83

Ind_Nationality_Code

In [120]:

```
pprint(df['Ind_Nationality_Code'].value_counts(dropna = False))
```

```
CH    441303
```

```
PT     14141
IT     11080
DE      8229
FR      4307
XK      4296
RS      3079
ES      2848
TR      2324
MK      2052
BA      1860
HR      1587
ER       268
       ...
FJ         1
KM         1
AN         1
Name: Ind_Nationality_Code, Length: 182, dtype: int64
```

Finding:

Many categories -> group / aggregate less used categories

In [121]:

```
# Cleanup Ind_Nationality_Code

min_cat = 400 # threshold -> change category if below

# Change to 'OTHER' if below count threshold
df.loc[df.groupby('Ind_Nationality_Code').Ind_Nationality_Code.transform('count').lt(min_cat), 'Ind_Nationality_Code'] = 'OTHER'

pprint(df['Ind_Nationality_Code'].value_counts(dropna = False))
```

```
CH       441303
PT        14141
IT        11080
OTHER      8993
DE         8229
FR         4307
XK         4296
RS         3079
ES         2848
TR         2324
MK         2052
BA         1860
HR         1587
AT         1291
BR         1253
UK          972
LK          853
TH          656
NL          618
RU          576
PL          534
LI          495
HU          488
```

```
BE          480
US          419
Name: Ind_Nationality_Code, dtype: int64
```

Written_Language_Code

In [122]:

```
pprint(df['Written_Language_Code'].value_counts(dropna = False))
```

```
DE    360807
FR    133429
IT     17806
EN      2692
Name: Written_Language_Code, dtype: int64
```

Note:

Convert to one-hot-encoding later

Oral_Language_Code

In [123]:

```
pprint(df['Oral_Language_Code'].value_counts(dropna = False))
```

```
DE    267913
EN    140291
FR     94274
IT     12164
PT        46
98        23
ES        13
DA         5
SV         2
EL         2
ZH         1
Name: Oral_Language_Code, dtype: int64
```

Finding:

- Group less used languages
- Oral_Language_Code = 98? -> leave it
- ZH = Chinese not züridütsch...

In [124]:

```
# Cleanup Oral_Language_Code

# Change to 'OTHER' if not in DE, EN, FR or IT
o_codes = ['DE', 'EN', 'FR', 'IT']
df.loc[~df['Oral_Language_Code'].isin(o_codes), 'Oral_Language_Code'] = 'OTHER'
```

Note:

Convert to one-hot-encoding later

Outlier Detection

In [125]:

```
df.head()
```

Out[125]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum
0	2019-03-16	1	2011-03-02	35608109	69.0	97	5-2E1WT	1	1	1	1203	1	CH	DE	DE	1	8.0	6.4750	51.80
1	2019-03-09	1	2015-12-21	35736109	35.0	39	5-2CEGS	1	1	1	1202	0	BA	DE	DE	0	20.0	17.7900	355.80
2	2018-07-01	1	2015-11-27	35460707	59.0	40	5-2E1W4	1	1	1	1300	1	CH	FR	FR	0	0.0	0.0000	0.00
3	2018-08-01	1	2012-10-04	35240209	69.0	77	5-2E1WT	1	1	1	1203	1	CH	DE	DE	0	30.0	2.7667	83.00
4	2018-11-13	1	2015-10-13	35721309	55.0	41	5-2E1XN	1	1	1	1203	1	CH	DE	EN	0	17.0	4.9900	84.83

In [126]:

```
df_outlier = df
```

In [127]:

```
# Outlier detection

def detect_outliers(df_outlier,n,features):
    """
    Takes a dataframe of features and returns a list of the indices
    corresponding to the observations containing more than n outliers according
    to the Tukey method.
    """
    outlier_indices = []

    # iterate over features(columns)
    for col in features:
        # 1st quartile (25%)
        Q1 = np.percentile(df_outlier[col], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df_outlier[col],75)
        # Interquartile range (IQR)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df_outlier[(df_outlier[col] < Q1 - outlier_step) | (df_outlier[col] > Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
```

```
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers

Outliers_to_drop = detect_outliers(df_outlier,2,['Actual_Recurring_Chrg_Amt',
                                              'Subs_Age_Months',
                                              'GROSS_PRICE_AMT_max'])
```

In [128]:

```
# Show outliers
df.loc[Outliers_to_drop]
```

Out[128]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum

Finding:

No outliers found

In [129]:

```
# Drop outliers
# df = df.drop(Outliers_to_drop, axis = 0).reset_index(drop=True)
```

Feature Engineering

Mobile Provider Code

In [130]:

```
# Extract old mobile provider code (079, 078, etc.)

df_feat = df.reset_index(drop=True)
df_feat['NF_Mobile_Provider_Code'] = df_feat['Main_Phone_Num'].astype(str).str[2:4].astype(np.int64)
```

In [131]:

```
pprint(df_feat['NF_Mobile_Provider_Code'].value_counts(dropna = False))
```

```
79    447565
78     31042
76     25381
77     10461
75       161
37       124
Name: NF_Mobile_Provider_Code, dtype: int64
```

Note:

Maybe convert to one-hot-encoding later

In [132]:

```
show_dist_cat_single(df_feat, 'NF_Mobile_Provider_Code')
```

Das Bild kann nicht angezeigt werden.

In [133]:

```
df_feat_prov = df_feat[["NF_Mobile_Provider_Code","Flag_Last_6_Month"]].groupby('NF_Mobile_Provider_Code').mean()
df_feat_prov.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[133]:

	Flag_Last_6_Month
NF_Mobile_Provider_Code	
75	0.031056
77	0.013861
76	0.013002
78	0.011597
79	0.011189
37	0.000000

Finding:

- 79 has the most non payers (absolute)
- 75 has the highest ratio of non payers

In [134]:

```
df_feat.head(20)
```

Out[134]:

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_ AMT_
0	2019-03-16	1	2011-03-02	35608109	69.0	97	5-2E1WT	1	1	1	1203	1	CH	DE	DE	1	8.0	6.4750	51.80	15.00
1	2019-03-09	1	2015-12-21	35736109	35.0	39	5-2CEGS	1	1	1	1202	0	BA	DE	DE	0	20.0	17.7900	355.80	80.00

	Start_Dt	Subs_Stat_Id	Subscr_Since_Dt	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_
2	2018-07-01	1	2015-11-27	35460707	59.0	40	5-2E1W4	1	1	1	1300	1	CH	FR	FR	0	0.0	0.0000	0.00	0.00
3	2018-08-01	1	2012-10-04	35240209	69.0	77	5-2E1WT	1	1	1	1203	1	CH	DE	DE	0	30.0	2.7667	83.00	3.00
4	2018-11-13	1	2015-10-13	35721309	55.0	41	5-2E1XN	1	1	1	1203	1	CH	DE	EN	0	17.0	4.9900	84.83	4.99
5	2018-10-21	1	2014-09-26	35763109	80.0	54	epb-AAA_BBB_XTRA_S	1	1	1	1203	1	IT	DE	EN	0	1.0	19.0000	19.00	19.00
6	2019-03-29	1	2015-10-01	35297809	80.0	42	5-30HQX	1	1	1	1203	1	CH	DE	EN	0	33.0	15.5664	513.69	99.99
7	2019-02-27	1	2013-10-28	35531508	80.0	65	5-30HS1	1	1	1	1203	0	MK	DE	EN	0	102.0	4.9500	504.90	4.95
8	2019-03-30	1	2004-04-28	35523008	80.0	179	5-30HS1	1	1	1	1300	1	CH	DE	DE	0	0.0	0.0000	0.00	0.00
9	2019-01-05	1	2012-08-09	35487009	80.0	79	epb-AAA_BBB_S	1	1	1	1300	1	XK	DE	EN	0	0.0	0.0000	0.00	0.00
10	2019-03-01	1	2014-02-28	35304809	100.0	61	epb-AAA_BBB_M	1	1	1	1204	0	RS	DE	EN	0	51.0	4.9900	254.49	4.99

Months since Start Date

In [135]:

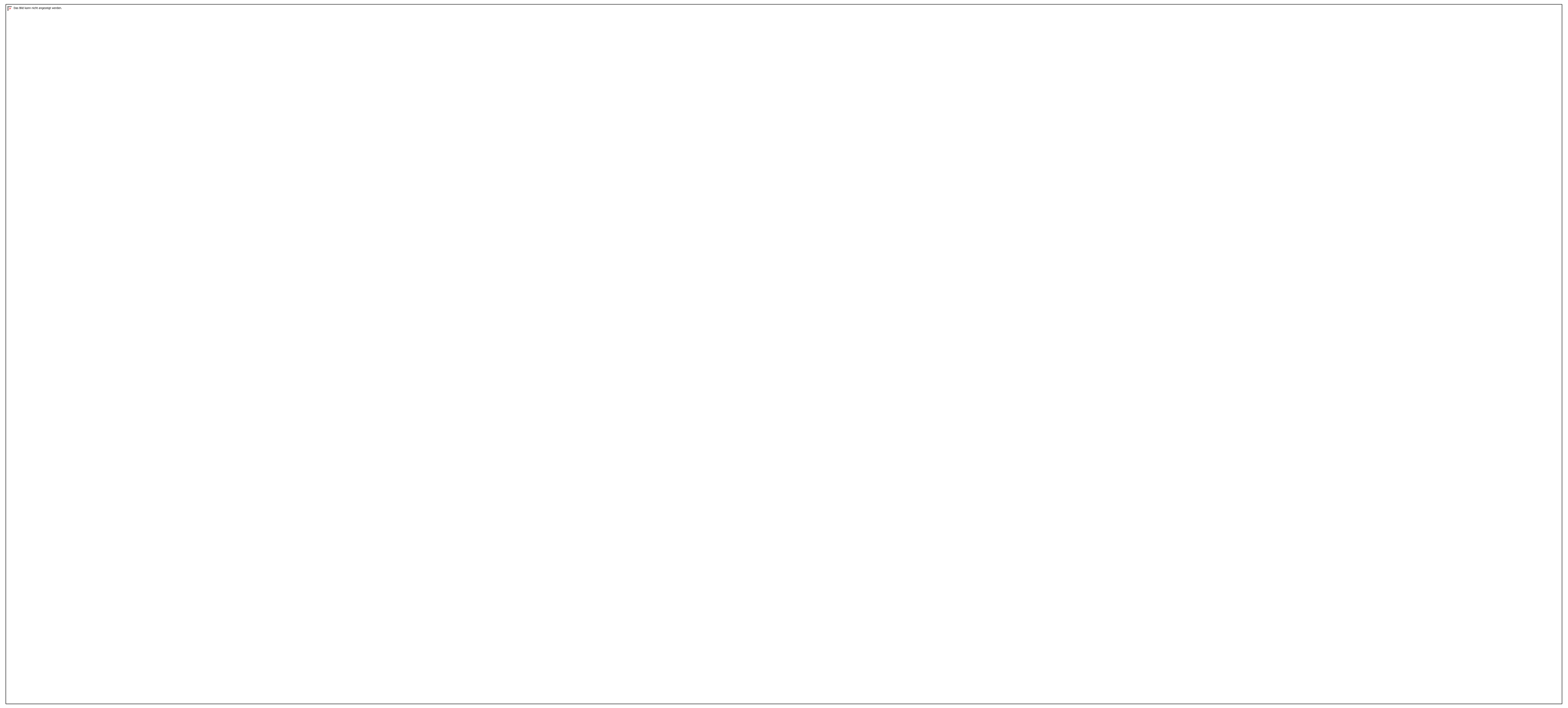
```
# Make use of Start_Dt
# Start_Dt = Date of last mobile contract modification

current_date = pd.to_datetime('today')

# No. of months since Start_Dt
df_feat['Start_Dt'] =  pd.to_datetime(df_feat['Start_Dt'])

nf_start_dt_months = ((current_date - df_feat['Start_Dt'])/30).dt.days
df_feat['NF_Start_Dt_Months'] = nf_start_dt_months

df_feat = df_feat.drop(['Start_Dt'],axis=1) # drop original column
df_feat = df_feat.drop(['Subscr_Since_Dt'],axis=1)

#df_feat['NF_Start_Dt_Months'] = ((current_date - df_feat['Start_Dt'])/30).dt.days

pprint(df_feat['NF_Start_Dt_Months'].value_counts(dropna = False))
```

```
18    151448
5      41757
4      39353
8      32332
```

```
7      31834
9      30082
6      29600
10     25452
14     25219
11     23201
12     21814
13     18916
15     14738
16     14088
17     11604
20      1314
22       665
23       553
21       376
24       169
19       148
25        40
26        16
27        13
43         1
52         1
Name: NF_Start_Dt_Months, dtype: int64
```

In [136]:

```
show_dist_cat_single(df_feat, 'NF_Start_Dt_Months')
```



In [137]:

```
df_feat_start_m = df_feat[["NF_Start_Dt_Months","Flag_Last_6_Month"]].groupby('NF_Start_Dt_Months').mean()
df_feat_start_m.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[137]:

	Flag_Last_6_Month
NF_Start_Dt_Months	
7	0.018439
6	0.015405
8	0.014660
4	0.014459
5	0.014441
10	0.013634
11	0.013232
9	0.012965
12	0.011873
13	0.011683

In [138]:

```
df_feat.head()
```

Out[138]:

	Subs_Stat_Id	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_P AMT_
0	1	35608109	69.0	97	5-2E1WT	1	1	1	1203	1	CH	DE	DE	1	8.0	6.4750	51.80	15.00	-5.00
1	1	35736109	35.0	39	5-2CEGS	1	1	1	1202	0	BA	DE	DE	0	20.0	17.7900	355.80	80.00	0.92
2	1	35460707	59.0	40	5-2E1W4	1	1	1	1300	1	CH	FR	FR	0	0.0	0.0000	0.00	0.00	0.00
3	1	35240209	69.0	77	5-2E1WT	1	1	1	1203	1	CH	DE	DE	0	30.0	2.7667	83.00	3.00	1.00
4	1	35721309	55.0	41	5-2E1XN	1	1	1	1203	1	CH	DE	EN	0	17.0	4.9900	84.83	4.99	4.99

Monthly Expenditure Ratio

In [139]:

```
# Ratio of average monthly expenditure to monthly abo cost, replace NaN with 0
# Use Subs_Age_Months (Months since mobile subscription) and not no. of months since first purchase
# Set 0 if Actual_Recurring_Chrg_Amt = 0 -> otherwise will raise an inifite error in the fit method

df_feat = df_feat.assign(NF_Ratio_Month=np.where(df_feat['Actual_Recurring_Chrg_Amt'] != 0, df_feat['GROSS_PRICE_AMT_sum'] / df_feat['Subs_Age_Months'] /
df_feat['Actual_Recurring_Chrg_Amt'], 0))
df_feat['NF_Ratio_Month'] = df_feat['NF_Ratio_Month'].round(decimals=4)
df_feat['NF_Ratio_Month'].fillna(0, inplace = True)
```

In [140]:

```
#temp
df_feat.loc[df_feat['NF_Ratio_Month'].isnull()]
```

Out[140]:

	Subs_Stat_Id	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	R

In [141]:

```
df_feat.head()
```

Out[141]:

	Subs_Stat_Id	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code	Written_Language_Code	Oral_Language_Code	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min
0	1	35608109	69.0	97	5-2E1WT	1	1	1	1203	1	CH	DE	DE	1	8.0	6.4750	51.80	15.00	-5.00
1	1	35736109	35.0	39	5-2CEGS	1	1	1	1202	0	BA	DE	DE	0	20.0	17.7900	355.80	80.00	0.92
2	1	35460707	59.0	40	5-2E1W4	1	1	1	1300	1	CH	FR	FR	0	0.0	0.0000	0.00	0.00	0.00
3	1	35240209	69.0	77	5-2E1WT	1	1	1	1203	1	CH	DE	DE	0	30.0	2.7667	83.00	3.00	1.00
4	1	35721309	55.0	41	5-2E1XN	1	1	1	1203	1	CH	DE	EN	0	17.0	4.9900	84.83	4.99	4.99

In [142]:

```
#show_dist_cat_single(df_feat, 'NF_Ratio_Month')
```

In [143]:

```
df_feat_ratio_m = df_feat[["NF_Ratio_Month","Flag_Last_6_Month"]].groupby('NF_Ratio_Month').mean()
df_feat_ratio_m.sort_values(by='Flag_Last_6_Month', ascending=False).head(10)
```

Out[143]:

	Flag_Last_6_Month
NF_Ratio_Month	
1.0633	1.0
0.7854	1.0
1.1292	1.0
0.6764	1.0
0.7871	1.0
1.1231	1.0
0.4613	1.0
1.1156	1.0
0.5747	1.0
0.5755	1.0

Note:

Data should be binned for a meaningful evaluation

In [144]:

```
# Backup temp
df_feat_backup_3 = df_feat.copy()
```

In [145]:

```
# Backup temp
#df_feat = df_feat_backup_3.copy()
```

One-hot-encoding

One column per category, with a 1 or 0 in each cell for if the row contained that column's category Used for features with low no. of dimensions

In [146]:

```
# One-hot-encoding
df_feat = pd.get_dummies(df_feat, columns=['Written_Language_Code'], drop_first=True)
df_feat = pd.get_dummies(df_feat, columns=['Oral_Language_Code'], drop_first=True)
df_feat = pd.get_dummies(df_feat, columns=['NF_Mobile_Provider_Code'], drop_first=True)
```

In [147]:

```
df_feat.info()
```

```
RangeIndex: 514734 entries, 0 to 514733
Data columns (total 37 columns):
Main_Phone_Num                514734 non-null int64
Subs_Stat_Id                  514734 non-null int64
Tac_Id                        514734 non-null int64
Actual_Recurring_Chrg_Amt     514734 non-null float64
Subs_Age_Months               514734 non-null int64
Prod_Id                       514734 non-null object
Reg_Relevant_Flag             514734 non-null int64
Prod_Item_Typ_Id              514734 non-null int64
Price_Typ_Id                  514734 non-null int64
Cust_Seg_Id                   514734 non-null int64
Ind_Gender                    514734 non-null int64
Ind_Nationality_Code          514734 non-null object
Flag_Last_6_Month             514734 non-null int64
NF_Num_Transactions           514734 non-null float64
GROSS_PRICE_AMT_mean          514734 non-null float64
GROSS_PRICE_AMT_sum           514734 non-null float64
GROSS_PRICE_AMT_max           514734 non-null float64
GROSS_PRICE_AMT_min           514734 non-null float64
REV_EFF_TS_max                514734 non-null object
REV_EFF_TS_min                514734 non-null object
HAS_E_pay                   514734 non-null float64
HAS_N_Pay                 514734 non-null float64
HAS_UNKNOWN                   514734 non-null float64
NF_Start_Dt_Months            514734 non-null int64
NF_Ratio_Month                514734 non-null float64
Written_Language_Code_EN      514734 non-null uint8
Written_Language_Code_FR      514734 non-null uint8
Written_Language_Code_IT      514734 non-null uint8
Oral_Language_Code_EN         514734 non-null uint8
Oral_Language_Code_FR         514734 non-null uint8
Oral_Language_Code_IT         514734 non-null uint8
Oral_Language_Code_OTHER      514734 non-null uint8
NF_Mobile_Provider_Code_75    514734 non-null uint8
NF_Mobile_Provider_Code_76    514734 non-null uint8
NF_Mobile_Provider_Code_77    514734 non-null uint8
NF_Mobile_Provider_Code_78    514734 non-null uint8
NF_Mobile_Provider_Code_79    514734 non-null uint8
dtypes: float64(10), int64(11), object(4), uint8(12)
memory usage: 104.1+ MB
```

In [148]:

```
# Change HAS_ features to proper one-hot-encoding (0/1)
mrg_agg.loc[(mrg_agg.HAS_E_pay > 0), 'HAS_E_pay'] = 1
mrg_agg.loc[(mrg_agg.HAS_N_Pay > 0), 'HAS_N_Pay'] = 1
mrg_agg.loc[(mrg_agg.HAS_UNKNOWN > 0), 'HAS_UNKNOWN'] = 1

# Change data type to Integer
mrg_agg['HAS_E_pay'] = mrg_agg.HAS_E_pay.astype(int)
mrg_agg['HAS_N_Pay'] = mrg_agg.HAS_E_pay.astype(int)
mrg_agg['HAS_UNKNOWN'] = mrg_agg.HAS_E_pay.astype(int)

mrg_agg.head()
```

Out[148]:

	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min	HAS_E_pay	HAS_N_Pay	HAS_UNKNOWN
0	22	5.1591	113.50	11.7	2.0	2019-02-06	2017-07-08	1	1	1
1	187	2.8922	540.85	8.0	1.0	2019-03-22	2017-04-12	0	0	0
2	3	8.3333	25.00	10.0	5.0	2018-09-18	2018-02-23	0	0	0
3	1	5.2000	5.20	5.2	5.2	2019-03-25	2019-03-25	0	0	0
4	23	6.0000	138.00	6.0	6.0	2019-03-12	2017-04-13	0	0	0

Binary Encoding

first the categories are encoded as ordinal, then those integers are converted into binary code, then the digits from that binary string are split into separate columns. This encodes the data in fewer dimensions that one-hot, but with some distortion of the distances

In [149]:

```
# Binary Encoding
ce_bin = ce.BinaryEncoder(cols=['Prod_Id', 'Ind_Nationality_Code'])
df_feat = ce_bin.fit_transform(df_feat)
df_feat
```

Out[149]:

	Subs_Stat_Id	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id_0	Prod_Id_1	Prod_Id_2	Prod_Id_3	Prod_Id_4	Prod_Id_5	Prod_Id_6	Prod_Id_7	Prod_Id_8	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender
0	1	35608109	69.0	97	0	0	0	0	0	0	0	0	1	1	1	1	1203	1
1	1	35736109	35.0	39	0	0	0	0	0	0	0	1	0	1	1	1	1202	0
2	1	35460707	59.0	40	0	0	0	0	0	0	0	1	1	1	1	1	1300	1
3	1	35240209	69.0	77	0	0	0	0	0	0	0	0	1	1	1	1	1203	1

	Ind_Nationality_Code_0	Ind_Nationality_Code_1	Ind_Nationality_Code_2	Ind_Nationality_Code_3	Ind_Nationality_Code_4	Ind_Nationality_Code_5	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min	HAS_E_pay	HAS_N_Pay	HAS_UNKNOWN	NF_Start_Dt_Months	NF_Ratio_Month	Written_Language_Code_EN	Written_Language_Code_FR	W_ ag
0	0	0	0	0	0	1	1	8.0	6.4750	51.80	15.00	-5.00	2019-03-19 00:00:00	2019-02-01 00:00:00	0.0	8.0	0.0	4	0.0077	0	0	0
1	0	0	0	0	1	0	0	20.0	17.7900	355.80	80.00	0.92	2019-02-23 00:00:00	2018-05-02 00:00:00	0.0	20.0	0.0	5	0.2607	0	0	0
2	0	0	0	0	0	1	0	0.0	0.0000	0.00	0.00	0.00	0	0	0.0	0.0	0.0	13	0.0000	0	1	0
3	0	0	0	0	0	1	0	30.0	2.7667	83.00	3.00	1.00	2019-03-10	2017-04-10	0.0	30.0	0.0	12	0.0156	0	0	0

	Subs_Stat_Id	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id_0	Prod_Id_1	Prod_Id_2	Prod_Id_3	Prod_Id_4	Prod_Id_5	Prod_Id_6	Prod_Id_7	Prod_Id_8	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code_0	Ind_Nationality_Code_1	Ind_Nationality_Code_2	Ind_Nationality_Code_3	Ind_Nationality_Code_4	Ind_Nationality_Code_5	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min	HAS_E_pay	HAS_N_Pay	HAS_UNKNOWN	NF_Start_Dt_Months	NF_Ratio_Month	Written_Language_Code_EN	Written_Language_Code_FR	W_ ag
																															00:00:00	00:00:00								
[illegible]	1	35721309	55.0	41	0	0	0	0	0	0	1	0	0	1	1	1	1203	1	0	0	0	0	0	1	0	17.0	4.9900	84.83	4.99	4.99	2018-08-27 00:00:00	2018-05-07 00:00:00	0.0	0.0	17.0	9	0.0376	0	0	0
[illegible]	1	35763109	80.0	54	0	0	0	0	0	0	1	0	1	1	1	1	1203	1	0	0	0	0	1	1	0	1.0	19.0000	19.00	19.00	19.00	2018-01-06 00:00:00	2018-01-06 00:00:00	0.0	1.0	0.0	9	0.0044	0	0	0
[illegible]	1	35297809	80.0	42	0	0	0	0	0	0	1	1	0	1	1	1	1203	1	0	0	0	0	0	1	0	33.0	15.5664	513.69	99.99	1.00	2019-03-20 00:00:00	2017-05-01 00:00:00	0.0	33.0	0.0	4	0.1529	0	0	0
[illegible]	1	35531508	80.0	65	0	0	0	0	0	0	1	1	1	1	1	1	1203	0	0	0	0	1	0	0	0	102.0	4.9500	504.90	4.95	4.95	2019-03-20 00:00:00	2017-04-13 00:00:00	102.0	0.0	0.0	5	0.0971	0	0	0

514734 rows × 50 columns

In [150]:

```
df_feat.head()
```

Out[150]:

	Subs_Stat_Id	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id_0	Prod_Id_1	Prod_Id_2	Prod_Id_3	Prod_Id_4	Prod_Id_5	Prod_Id_6	Prod_Id_7	Prod_Id_8	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code_0	Ind_Nationality_Code_1	Ind_Nationality_Code_2	Ind_Nationality_Code_3	Ind_Nationality_Code_4	Ind_Nationality_Code_5	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min	HAS_E_pay	HAS_N_Pay	HAS_UNKNOWN	NF_Start_Dt_Months	NF_Ratio_Month	Written_Language_Code_EN	Written_Language_Code_FR	W_ ag
[illegible]	1	35608109	69.0	97	0	0	0	0	0	0	0	0	1	1	1	1	1203	1	0	0	0	0	0	1	1	8.0	6.4750	51.80	15.00	-5.00	2019-03-19 00:00:00	2019-02-01 00:00:00	0.0	8.0	0.0	4	0.0077	0	0	0
[illegible]	1	35736109	35.0	39	0	0	0	0	0	0	0	1	0	1	1	1	1202	0	0	0	0	0	1	0	0	20.0	17.7900	355.80	80.00	0.92	2019-02-23 00:00:00	2018-05-02 00:00:00	0.0	20.0	0.0	5	0.2607	0	0	0
[illegible]	1	35460707	59.0	40	0	0	0	0	0	0	0	1	1	1	1	1	1300	1	0	0	0	0	0	1	0	0.0	0.0000	0.00	0.00	0.00	0	0	0.0	0.0	0.0	13	0.0000	0	1	0
[illegible]	1	35240209	69.0	77	0	0	0	0	0	0	0	0	1	1	1	1	1203	1	0	0	0	0	0	1	0	30.0	2.7667	83.00	3.00	1.00	2019-03-10 00:00:00	2017-04-10 00:00:00	0.0	30.0	0.0	12	0.0156	0	0	0

	Subs_Stat_Id	Tac_Id	Actual_Recurring_Chrg_Amt	Subs_Age_Months	Prod_Id_0	Prod_Id_1	Prod_Id_2	Prod_Id_3	Prod_Id_4	Prod_Id_5	Prod_Id_6	Prod_Id_7	Prod_Id_8	Reg_Relevant_Flag	Prod_Item_Typ_Id	Price_Typ_Id	Cust_Seg_Id	Ind_Gender	Ind_Nationality_Code_0	Ind_Nationality_Code_1	Ind_Nationality_Code_2	Ind_Nationality_Code_3	Ind_Nationality_Code_4	Ind_Nationality_Code_5	Flag_Last_6_Month	NF_Num_Transactions	GROSS_PRICE_AMT_mean	GROSS_PRICE_AMT_sum	GROSS_PRICE_AMT_max	GROSS_PRICE_AMT_min	REV_EFF_TS_max	REV_EFF_TS_min	HAS_E_pay	HAS_N_Pay	HAS_UNKNOWN	NF_Start_Dt_Months	NF_Ratio_Month	Written_Language_Code_EN	Written_Language_Code_FR	W_ag
‹	1	35721309	55.0	41	0	0	0	0	0	0	1	0	0	1	1	1	1203	1	0	0	0	0	0	1	0	17.0	4.9900	84.83	4.99	4.99	2018-08-27 00:00:00	2018-05-07 00:00:00	0.0	0.0	17.0	9	0.0376	0	0	0

In [151]:

```
# Change data type to Integer
df_feat['HAS_E_pay'] = df_feat.HAS_E_pay.astype(int)
df_feat['HAS_N_Pay'] = df_feat.HAS_E_pay.astype(int)
df_feat['HAS_UNKNOWN'] = df_feat.HAS_E_pay.astype(int)
```

In [152]:

```
df_feat.info()
```

```
RangeIndex: 514734 entries, 0 to 514733
Data columns (total 50 columns):
Main_Phone_Num                514734 non-null int64
Subs_Stat_Id                  514734 non-null int64
Tac_Id                        514734 non-null int64
Actual_Recurring_Chrg_Amt     514734 non-null float64
Subs_Age_Months               514734 non-null int64
Prod_Id_0                     514734 non-null int64
Prod_Id_1                     514734 non-null int64
Prod_Id_2                     514734 non-null int64
Prod_Id_3                     514734 non-null int64
Prod_Id_4                     514734 non-null int64
Prod_Id_5                     514734 non-null int64
Prod_Id_6                     514734 non-null int64
Prod_Id_7                     514734 non-null int64
Prod_Id_8                     514734 non-null int64
Reg_Relevant_Flag             514734 non-null int64
Prod_Item_Typ_Id              514734 non-null int64
Price_Typ_Id                  514734 non-null int64
Cust_Seg_Id                   514734 non-null int64
Ind_Gender                    514734 non-null int64
Ind_Nationality_Code_0        514734 non-null int64
Ind_Nationality_Code_1        514734 non-null int64
Ind_Nationality_Code_2        514734 non-null int64
Ind_Nationality_Code_3        514734 non-null int64
Ind_Nationality_Code_4        514734 non-null int64
Ind_Nationality_Code_5        514734 non-null int64
Flag_Last_6_Month             514734 non-null int64
NF_Num_Transactions           514734 non-null float64
GROSS_PRICE_AMT_mean          514734 non-null float64
GROSS_PRICE_AMT_sum           514734 non-null float64
GROSS_PRICE_AMT_max           514734 non-null float64
GROSS_PRICE_AMT_min           514734 non-null float64
REV_EFF_TS_max                514734 non-null object
REV_EFF_TS_min                514734 non-null object
```

```
HAS_E_pay                   514734 non-null int64
HAS_N_Pay                 514734 non-null int64
HAS_UNKNOWN                   514734 non-null int64
NF_Start_Dt_Months            514734 non-null int64
NF_Ratio_Month                514734 non-null float64
Written_Language_Code_EN      514734 non-null uint8
Written_Language_Code_FR      514734 non-null uint8
Written_Language_Code_IT      514734 non-null uint8
Oral_Language_Code_EN         514734 non-null uint8
Oral_Language_Code_FR         514734 non-null uint8
Oral_Language_Code_IT         514734 non-null uint8
Oral_Language_Code_OTHER      514734 non-null uint8
NF_Mobile_Provider_Code_75    514734 non-null uint8
NF_Mobile_Provider_Code_76    514734 non-null uint8
NF_Mobile_Provider_Code_77    514734 non-null uint8
NF_Mobile_Provider_Code_78    514734 non-null uint8
NF_Mobile_Provider_Code_79    514734 non-null uint8
dtypes: float64(7), int64(29), object(2), uint8(12)
memory usage: 155.1+ MB
```

In [153]:

```
# Correlation with target Flag_Last_6_Month
corr_matrix = df_feat.corr()
corr_matrix['Flag_Last_6_Month'].sort_values(ascending=False)
```

Out[153]:

```
Flag_Last_6_Month             1.000000
GROSS_PRICE_AMT_sum           0.139789
GROSS_PRICE_AMT_max           0.133276
NF_Ratio_Month                0.121247
NF_Num_Transactions           0.114625
GROSS_PRICE_AMT_mean          0.065548
Actual_Recurring_Chrg_Amt     0.056395
Oral_Language_Code_EN         0.022524
Ind_Nationality_Code_3        0.017877
HAS_UNKNOWN                   0.017687
HAS_N_Pay                 0.017687
HAS_E_pay                   0.017687
Prod_Id_8                     0.015027
Ind_Nationality_Code_4        0.012143
Ind_Nationality_Code_2        0.010318
Tac_Id                        0.006626
Prod_Id_3                     0.005647
Prod_Id_6                     0.004846
NF_Mobile_Provider_Code_76    0.003526
NF_Mobile_Provider_Code_77    0.003397
NF_Mobile_Provider_Code_75    0.003287
Prod_Id_5                     0.001866
Main_Phone_Num                0.001652
Oral_Language_Code_OTHER      0.001309
NF_Mobile_Provider_Code_78    0.000564
Ind_Nationality_Code_1        0.000459
Written_Language_Code_EN      0.000360
Prod_Id_1                     0.000165
Subs_Stat_Id                 -0.000459
Written_Language_Code_FR     -0.001837
```

```
GROSS_PRICE_AMT_min          -0.003416
Prod_Item_Typ_Id             -0.003686
Price_Typ_Id                 -0.003936
Reg_Relevant_Flag            -0.003936
NF_Mobile_Provider_Code_79   -0.004184
Written_Language_Code_IT     -0.004443
Oral_Language_Code_IT        -0.005454
Prod_Id_2                    -0.007498
Oral_Language_Code_FR        -0.007776
Prod_Id_4                    -0.008626
Ind_Nationality_Code_5       -0.012716
Ind_Gender                   -0.027996
NF_Start_Dt_Months           -0.033836
Prod_Id_7                    -0.035726
Subs_Age_Months              -0.045252
Cust_Seg_Id                  -0.058184
Prod_Id_0                          NaN
Ind_Nationality_Code_0             NaN
Name: Flag_Last_6_Month, dtype: float64
```

In [154]:

```
df_feat.info()
```

```
RangeIndex: 514734 entries, 0 to 514733
Data columns (total 50 columns):
Main_Phone_Num                514734 non-null int64
Subs_Stat_Id                  514734 non-null int64
Tac_Id                        514734 non-null int64
Actual_Recurring_Chrg_Amt     514734 non-null float64
Subs_Age_Months               514734 non-null int64
Prod_Id_0                     514734 non-null int64
Prod_Id_1                     514734 non-null int64
Prod_Id_2                     514734 non-null int64
Prod_Id_3                     514734 non-null int64
Prod_Id_4                     514734 non-null int64
Prod_Id_5                     514734 non-null int64
Prod_Id_6                     514734 non-null int64
Prod_Id_7                     514734 non-null int64
Prod_Id_8                     514734 non-null int64
Reg_Relevant_Flag             514734 non-null int64
Prod_Item_Typ_Id              514734 non-null int64
Price_Typ_Id                  514734 non-null int64
Cust_Seg_Id                   514734 non-null int64
Ind_Gender                    514734 non-null int64
Ind_Nationality_Code_0        514734 non-null int64
Ind_Nationality_Code_1        514734 non-null int64
Ind_Nationality_Code_2        514734 non-null int64
Ind_Nationality_Code_3        514734 non-null int64
Ind_Nationality_Code_4        514734 non-null int64
Ind_Nationality_Code_5        514734 non-null int64
Flag_Last_6_Month             514734 non-null int64
NF_Num_Transactions           514734 non-null float64
GROSS_PRICE_AMT_mean          514734 non-null float64
GROSS_PRICE_AMT_sum           514734 non-null float64
GROSS_PRICE_AMT_max           514734 non-null float64
```

```
GROSS_PRICE_AMT_min           514734 non-null float64
REV_EFF_TS_max                514734 non-null object
REV_EFF_TS_min                514734 non-null object
HAS_E_pay                   514734 non-null int64
HAS_N_Pay                 514734 non-null int64
HAS_UNKNOWN                   514734 non-null int64
NF_Start_Dt_Months            514734 non-null int64
NF_Ratio_Month                514734 non-null float64
Written_Language_Code_EN      514734 non-null uint8
Written_Language_Code_FR      514734 non-null uint8
Written_Language_Code_IT      514734 non-null uint8
Oral_Language_Code_EN         514734 non-null uint8
Oral_Language_Code_FR         514734 non-null uint8
Oral_Language_Code_IT         514734 non-null uint8
Oral_Language_Code_OTHER      514734 non-null uint8
NF_Mobile_Provider_Code_75    514734 non-null uint8
NF_Mobile_Provider_Code_76    514734 non-null uint8
NF_Mobile_Provider_Code_77    514734 non-null uint8
NF_Mobile_Provider_Code_78    514734 non-null uint8
NF_Mobile_Provider_Code_79    514734 non-null uint8
dtypes: float64(7), int64(29), object(2), uint8(12)
memory usage: 155.1+ MB
```

In [155]:

```
#temp
df_feat_backup_4 = df_feat.copy()
```

In [409]:

```
#temp
df_feat = df_feat_backup_4.copy()
```

Modeling

Preparation

In [410]:

```
# Function to show basic confusion matrix
def show_conf_mat(y_test, y_pred):
    conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
    print('Confusion matrix:\n', conf_mat)

    labels = ['Class 0 (PAYS)', 'Class 1 (DOES NOT PAY)']
    fig = plt.figure()
    ax = fig.add_subplot(111)
    cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
    fig.colorbar(cax)
    ax.set_xticklabels([''] + labels)
    ax.set_yticklabels([''] + labels)
    plt.xlabel('Predicted')
    plt.ylabel('Expected')
    plt.show()
```

In [411]:

```
# Drop unnecessary columns
```

```
df_feat = df_feat.drop(['Main_Phone_Num'],axis=1)
df_feat = df_feat.drop(['REV_EFF_TS_max'],axis=1)
df_feat = df_feat.drop(['REV_EFF_TS_min'],axis=1)
```

In [412]:

```
# temp
df_feat_backup_5 = df_feat.copy()
```

In [413]:

```
#temp
print(np.all(np.isfinite(df_feat))) # -> problem?
#print(np.all(np.isfinite(X_train))) # -> problem?
#print(np.all(np.isfinite(y_train)))
```

```
True
```

In [414]:

```
# Prepare Training and Testing sets

#y = df_feat.['Flag_Last_6_Month']
#X = df_feat.drop(columns=['Flag_Last_6_Month'])

y = df_feat.Flag_Last_6_Month
X = df_feat.drop('Flag_Last_6_Month', axis=1)

# Simple split for Training and Test Data
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.25, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[414]:

```
((386050, 46), (128684, 46), (386050,), (128684,))
```

In [415]:

```
#temp
print("df_feat: ",type(df_feat))
print("X: ",type(X))
print("X_train: ",type(X_train))
print("y_train: ",type(y_train))
print("X_test: ",type(X_test))
print("y_test: ",type(y_test))
```

```
df_feat:
X:
X_train:
y_train:
X_test:
y_test:
```

In [416]:

```
print('y_train class counts')
print(y_train.value_counts())
print('y_test class counts')
print(y_test.value_counts())
```

```
y_train class counts
0    381664
1      4386
Name: Flag_Last_6_Month, dtype: int64
y_test class counts
```

```
0    127222
1      1462
Name: Flag_Last_6_Month, dtype: int64
```

Basic Models

In [428]:

```
# Basic function for training and some analysis
def run_model(clf):
    print("Model: ",clf)
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    clf.score(X_train, y_train)
    acc_clf = round(clf.score(X_train, y_train) * 100, 2)
    print("score: ",round(acc_clf,2,), "%")

    #f1 = round(f1_score(y_test, y_pred),2)
    #print("f1-score: ",f1)

    #recall = round(recall_score(y_test, y_pred),2)
    #print("recall-score: ",recall)

    # predict probabilities
    probs = clf.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    auc = roc_auc_score(y_test, probs)
    print('AUC: %.3f' % auc)

    # Classification Report
    target_names = ['Class 0 (PAYS)', 'Class 1 (DOES NOT PAY)']
    print(classification_report(y_test, y_pred, target_names=target_names))

    # Check if both classes are predicted
    print("Check if both classes are predicted:")
    print("y_pred: ",np.unique(y_pred))
    print("y_test: ",np.unique(y_test))

    # Show Confusion Matrix
    show_conf_mat(y_test, y_pred)
```

In [429]:

```
clf_rf = RandomForestClassifier(n_estimators=100)
run_model(clf_rf)
```

```
Model:  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
score:  100.0 %
```

```
AUC: 0.847
                       precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      1.00      0.99    127222
Class 1 (DOES NOT PAY)       0.00      0.00      0.00      1462

             micro avg       0.99      0.99      0.99    128684
             macro avg       0.49      0.50      0.50    128684
          weighted avg       0.98      0.99      0.98    128684

Check if both classes are predicted:
y_pred:  [0]
y_test:  [0 1]
Confusion matrix:
 [[127222      0]
 [  1462      0]]
/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
```

In [430]:

```
# Decision Tree
clf_dt = DecisionTreeClassifier()
run_model(clf_dt)
```

```
Model:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.523
                       precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      0.99      0.99    127222
Class 1 (DOES NOT PAY)       0.05      0.06      0.05      1462

             micro avg       0.98      0.98      0.98    128684
             macro avg       0.52      0.52      0.52    128684
```

```
          weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125443   1779]
 [  1373     89]]
```

In [431]:

```
# Logistic Regression
clf_lr = LogisticRegression()
run_model(clf_lr)
```

```
Model:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
score:  98.86 %
AUC: 0.505
                        precision    recall  f1-score   support

       Class 0 (PAYS)       0.99      1.00      0.99    127222
Class 1 (DOES NOT PAY)       0.00      0.00      0.00      1462

            micro avg       0.99      0.99      0.99    128684
            macro avg       0.49      0.50      0.50    128684
         weighted avg       0.98      0.99      0.98    128684

Check if both classes are predicted:
y_pred:  [0]
y_test:  [0 1]
/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Confusion matrix:
 [[127222      0]
```

```
 [  1462      0]]
```

In [432]:

```
# KNN
clf_knn = KNeighborsClassifier(n_neighbors = 3)
run_model(clf_knn)
```

```
Model:  KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
           weights='uniform')
score:  98.91 %
AUC: 0.556
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      1.00      0.99    127222
Class 1 (DOES NOT PAY)       0.05      0.01      0.01      1462

             micro avg       0.99      0.99      0.99    128684
             macro avg       0.52      0.50      0.50    128684
          weighted avg       0.98      0.99      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[127019    203]
 [  1452     10]]
```

In [433]:

```
# AdaBoost
clf_ada = AdaBoostClassifier(DecisionTreeClassifier(random_state=2),random_state=42,learning_rate=0.1)
run_model(clf_ada)
```

```
Model:  AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
```

```
            min_weight_fraction_leaf=0.0, presort=False, random_state=2,
            splitter='best'),
          learning_rate=0.1, n_estimators=50, random_state=42)
score:  100.0 %
AUC: 0.521
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      0.99      0.99    127222
Class 1 (DOES NOT PAY)       0.04      0.06      0.05      1462

             micro avg       0.98      0.98      0.98    128684
             macro avg       0.52      0.52      0.52    128684
          weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125424   1798]
 [  1379     83]]
```

Finding:

Accuracy is extremely high but some models won't be able to predict the customers who will not pay! => Highly imbalanced data

Handling Class Imbalance

Class Weight / Cost Function

In [ ]:

```
# Train model
clf_svm = SVC(kernel='linear',
            class_weight='balanced', # penalize
            C=1.0,
            random_state=0),
            probability=True)

run_model(clf_svm)

# Accuracy?
#print(accuracy_score(y, y_pred))

# AUROC?
#prob_svm = svm.predict_proba(X_train)
```

```
#prob_svm = [p[1] for p in prob_svm]
#print(roc_auc_score(y, prob_svm))
```

Note:

Takes a long time to train on my local machine -> Code will not be executed

In [437]:

```
# Random Forest Classifier with class_weight
clf_rf_class_weight = RandomForestClassifier(n_estimators=100, class_weight='balanced')
run_model(clf_rf_class_weight)
```

```
Model:  RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=100, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
score:  100.0 %
AUC: 0.835
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      1.00      0.99    127222
Class 1 (DOES NOT PAY)       0.00      0.00      0.00      1462

             micro avg       0.99      0.99      0.99    128684
             macro avg       0.49      0.50      0.50    128684
          weighted avg       0.98      0.99      0.98    128684

Check if both classes are predicted:
y_pred:  [0]
y_test:  [0 1]
Confusion matrix:
 [[127222      0]
 [  1462      0]]
/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
```

In [438]:

```
# Decision Tree with class_weight
clf_dt_class_weight = DecisionTreeClassifier(class_weight='balanced')
run_model(clf_dt_class_weight)
```

```
Model:  DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.519
                        precision    recall  f1-score   support

       Class 0 (PAYS)       0.99      0.99      0.99    127222
Class 1 (DOES NOT PAY)       0.05      0.05      0.05      1462

            micro avg       0.98      0.98      0.98    128684
            macro avg       0.52      0.52      0.52    128684
         weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125797   1425]
 [  1389     73]]
```

In [439]:

```
# Test different class_weights (faster to compute than SVM)
for w in [1,5,10,100, 1000]:
    print('--- Weight of {} ---'.format(w))
    clf_dt_class_weight = DecisionTreeClassifier(class_weight={0:1,1:w})
    #clf_lr_class_weight = LogisticRegression(class_weight={0:1,1:w})
    run_model(clf_dt_class_weight)
```

```
--- Weight of 1 ---
Model:  DecisionTreeClassifier(class_weight={0: 1, 1: 1}, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.523
                        precision    recall  f1-score   support
```

```
       Class 0 (PAYS)       0.99      0.99      0.99    127222
Class 1 (DOES NOT PAY)       0.05      0.06      0.05      1462

            micro avg       0.98      0.98      0.98    128684
            macro avg       0.52      0.52      0.52    128684
         weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125441   1781]
 [  1373     89]]
```


```
--- Weight of 5 ---
Model:  DecisionTreeClassifier(class_weight={0: 1, 1: 5}, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.522
                        precision    recall  f1-score   support

       Class 0 (PAYS)       0.99      0.99      0.99    127222
Class 1 (DOES NOT PAY)       0.05      0.06      0.05      1462

            micro avg       0.98      0.98      0.98    128684
            macro avg       0.52      0.52      0.52    128684
         weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125624   1598]
 [  1378     84]]
```

```
--- Weight of 10 ---
Model:  DecisionTreeClassifier(class_weight={0: 1, 1: 10}, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.526
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      0.99      0.99    127222
Class 1 (DOES NOT PAY)       0.06      0.06      0.06      1462

             micro avg       0.98      0.98      0.98    128684
             macro avg       0.52      0.53      0.52    128684
          weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125637   1585]
 [  1368     94]]
```

```
--- Weight of 100 ---
Model:  DecisionTreeClassifier(class_weight={0: 1, 1: 100}, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.519
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      0.99      0.99    127222
```

```
Class 1 (DOES NOT PAY)       0.05      0.05      0.05      1462

             micro avg       0.98      0.98      0.98    128684
             macro avg       0.52      0.52      0.52    128684
          weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125859   1363]
 [  1390     72]]
```


```
--- Weight of 1000 ---
Model:  DecisionTreeClassifier(class_weight={0: 1, 1: 1000}, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.520
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      0.99      0.99    127222
Class 1 (DOES NOT PAY)       0.05      0.05      0.05      1462

             micro avg       0.98      0.98      0.98    128684
             macro avg       0.52      0.52      0.52    128684
          weighted avg       0.98      0.98      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125930   1292]
 [  1389     73]]
```

Down-Sampling

Downsample majority class

In [440]:

```
# Down-Sampling
# Prepare Dataframe

# Concat Training Data -> only re-sample on training set
X = pd.concat([X_train, y_train], axis=1)

# Separate majority and minority classes
df_majority = X[X.Flag_Last_6_Month==0]
df_minority = X[X.Flag_Last_6_Month==1]

# Down-sample majority class
df_majority_downsampled = resample(df_majority,
                                 replace=False,     # sample with replacement
                                 n_samples=len(df_minority),    # to match majority class
                                 random_state=42) # reproducible results

# Combine minority class with downsampled minority class
df_downsampled = pd.concat([df_minority, df_majority_downsampled])

# Display new class counts
df_downsampled.Flag_Last_6_Month.value_counts()
```

Out[440]:

```
1    4386
0    4386
Name: Flag_Last_6_Month, dtype: int64
```

In [447]:

```
# Create Training Data
y_train = df_downsampled.Flag_Last_6_Month
X_train = df_downsampled.drop('Flag_Last_6_Month', axis=1)
```

In [448]:

```
# Random Forest after Down-Sampling
clf_rf_down = RandomForestClassifier(n_estimators=100)
run_model(clf_rf_down)
```

```
Model:  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
```

```
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
score:  100.0 %
AUC: 0.883
                        precision    recall  f1-score   support

        Class 0 (PAYS)       1.00      0.75      0.86    127222
Class 1 (DOES NOT PAY)       0.04      0.91      0.08      1462

             micro avg       0.76      0.76      0.76    128684
             macro avg       0.52      0.83      0.47    128684
          weighted avg       0.99      0.76      0.85    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[95841 31381]
 [  136  1326]]
```

In [449]:

```
# Decision Tree after Down-Sampling
clf_dt_down = DecisionTreeClassifier()
run_model(clf_dt_down)
```

```
Model:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.746
                        precision    recall  f1-score   support

        Class 0 (PAYS)       1.00      0.76      0.86    127222
Class 1 (DOES NOT PAY)       0.03      0.73      0.06      1462

             micro avg       0.76      0.76      0.76    128684
             macro avg       0.51      0.75      0.46    128684
          weighted avg       0.99      0.76      0.85    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
```

```
 [[96294 30928]
 [  388  1074]]
```

In [450]:

```
# Logistic Regression after Down-Sampling
clf_lr_down = LogisticRegression()
run_model(clf_lr_down)
```

```
Model:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
score:  74.46 %
/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this
warning.
  FutureWarning)
AUC: 0.868
                       precision    recall  f1-score   support

        Class 0 (PAYS)       1.00      0.87      0.93    127222
Class 1 (DOES NOT PAY)       0.05      0.62      0.10      1462

             micro avg       0.87      0.87      0.87    128684
             macro avg       0.52      0.75      0.51    128684
          weighted avg       0.98      0.87      0.92    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[111234  15988]
 [   557    905]]
```

In [451]:

```
# KNN after Down-Sampling
clf_knn_down = KNeighborsClassifier(n_neighbors = 3)
run_model(clf_knn_down)
```

```
Model:  KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
           weights='uniform')
score:  85.77 %
AUC: 0.783
                        precision    recall  f1-score   support

        Class 0 (PAYS)       1.00      0.74      0.85    127222
Class 1 (DOES NOT PAY)       0.03      0.74      0.06      1462

             micro avg       0.74      0.74      0.74    128684
             macro avg       0.51      0.74      0.46    128684
          weighted avg       0.99      0.74      0.84    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[94703 32519]
 [  379  1083]]
```

In [452]:

```
# AdaBoost after Down-Sampling
clf_ada_down = AdaBoostClassifier(DecisionTreeClassifier(random_state=2),random_state=42,learning_rate=0.1)
run_model(clf_ada_down)
```

```
Model:  AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=2,
            splitter='best'),
          learning_rate=0.1, n_estimators=50, random_state=42)
score:  100.0 %
AUC: 0.746
                        precision    recall  f1-score   support

        Class 0 (PAYS)       1.00      0.76      0.86    127222
Class 1 (DOES NOT PAY)       0.03      0.73      0.06      1462

             micro avg       0.76      0.76      0.76    128684
             macro avg       0.51      0.75      0.46    128684
          weighted avg       0.99      0.76      0.85    128684

Check if both classes are predicted:
```

```
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[96444 30778]
 [  388  1074]]
```

In [ ]:

SMOTE

Synthetic Minority Oversampling Technique

In [453]:

```
df_feat = df_feat_backup_5.copy()
```

In [454]:

```
# Try Synthetic Minority Oversampling Technique (SMOTE)

y = df_feat.Flag_Last_6_Month
X = df_feat.drop('Flag_Last_6_Month', axis=1)

# setting up testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

sm = SMOTE(random_state=42, ratio=1.0)
X_train, y_train = sm.fit_sample(X_train, y_train)  # modifies X_train and y_train into numpy.ndarray

#X_train = pd.DataFrame(X_train, columns=X.columns)
#y_train = pd.Series(y_train)
```

```
/Users/of/anaconda3/lib/python3.7/site-packages/imblearn/utils/deprecation.py:53: DeprecationWarning: 'ratio' is deprecated from 0.4 and will be removed in 0.6 for the estimator . Use
'sampling_strategy' instead.
  category=DeprecationWarning)
```

In [455]:

```
#temp
print("df_feat: ",type(df_feat))
print("X: ",type(X))
print("X_train: ",type(X_train))
print("y_train: ",type(y_train))
print("X_test: ",type(X_test))
print("y_test: ",type(y_test))
```

```
df_feat:
X:
X_train:
y_train:
```

```
X_test:
y_test:
```

In [456]:

```
# Random Forest after SMOTE
clf_rf_smote = RandomForestClassifier(n_estimators=100)
run_model(clf_rf_smote)
```

```
Model:  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
score:  100.0 %
AUC: 0.839
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      1.00      0.99    127230
Class 1 (DOES NOT PAY)       0.50      0.00      0.00      1454

             micro avg       0.99      0.99      0.99    128684
             macro avg       0.74      0.50      0.50    128684
          weighted avg       0.98      0.99      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[127229      1]
 [  1453      1]]
```

In [457]:

```
# Decision Tree after SMOTE
clf_dt_smote = DecisionTreeClassifier()
run_model(clf_dt_smote)
```

```
Model:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
score:  100.0 %
AUC: 0.525
                        precision    recall  f1-score   support
```

```
       Class 0 (PAYS)       0.99      0.98      0.99    127230
Class 1 (DOES NOT PAY)       0.05      0.07      0.06      1454

            micro avg       0.97      0.97      0.97    128684
            macro avg       0.52      0.53      0.52    128684
         weighted avg       0.98      0.97      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125290   1940]
 [  1358     96]]
```

In [458]:

```
# Logistic Regression after SMOTE
clf_lr_smote = LogisticRegression()
run_model(clf_lr_smote)
```

```
Model:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
/Users/of/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
score:  74.54 %
AUC: 0.875
                        precision    recall  f1-score   support

       Class 0 (PAYS)       0.99      0.88      0.93    127230
Class 1 (DOES NOT PAY)       0.05      0.61      0.10      1454

            micro avg       0.87      0.87      0.87    128684
            macro avg       0.52      0.74      0.51    128684
         weighted avg       0.98      0.87      0.92    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[111391  15839]
 [   571    883]]
```

In [459]:

```
# KNN after SMOTE
clf_knn_smote = KNeighborsClassifier(n_neighbors = 3)
run_model(clf_knn_smote)
```

```
Model:  KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
           weights='uniform')
score:  97.73 %
AUC: 0.648
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      0.93      0.96    127230
Class 1 (DOES NOT PAY)       0.05      0.31      0.08      1454

             micro avg       0.92      0.92      0.92    128684
             macro avg       0.52      0.62      0.52    128684
          weighted avg       0.98      0.92      0.95    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[117911   9319]
 [  1007    447]]
```

In [460]:

```
# AdaBoost after SMOTE
clf_ada_smote = AdaBoostClassifier(DecisionTreeClassifier(random_state=2),random_state=42,learning_rate=0.1)
run_model(clf_ada_smote)
```

```
Model:  AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=2,
```

```
           splitter='best'),
          learning_rate=0.1, n_estimators=50, random_state=42)
score:  100.0 %
AUC: 0.527
                        precision    recall  f1-score   support

        Class 0 (PAYS)       0.99      0.98      0.99    127230
Class 1 (DOES NOT PAY)       0.05      0.07      0.06      1454

             micro avg       0.97      0.97      0.97    128684
             macro avg       0.52      0.53      0.52    128684
          weighted avg       0.98      0.97      0.98    128684

Check if both classes are predicted:
y_pred:  [0 1]
y_test:  [0 1]
Confusion matrix:
 [[125295   1935]
 [  1353    101]]
```

In [ ]:

In [ ]:

In [461]:

Finding:

- No feature stands out -> GROSS_PRICE_AMT_sum, Tac_Id and NF_Ratio_Month seam to play the most significant role
- Reg_Relevant_Flag, Prod_Item_Typ_Id and Price_Typ_Id are "full features" (not one-hot-encodings) that seam not to be important -> drop them

In [147]:

```
# Drop irrelevant features
X_train = X_train.drop('Reg_Relevant_Flag', 1)
X_train = X_train.drop('Prod_Item_Typ_Id', 1)
X_train = X_train.drop('Price_Typ_Id', 1)

X_test = X_test.drop('Reg_Relevant_Flag', 1)
X_test = X_test.drop('Prod_Item_Typ_Id', 1)
X_test = X_test.drop('Price_Typ_Id', 1)

X_train.shape, X_test.shape
```

Out[147]:

```
((386050, 43), (128684, 43))
```

In [148]:

```
# Train again
random_forest = RandomForestClassifier(n_estimators=100, oob_score = True)
random_forest.fit(X_train, y_train)
```

```
Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, y_train)

acc_random_forest = round(random_forest.score(X_train, y_train) * 100, 2)
print(round(acc_random_forest,2,), "%")
```

```
100.0 %
```

In [149]:

```
print("oob score:", round(random_forest.oob_score_, 4)*100, "%")
```

```
oob score: 98.86 %
```

Hyperparameter Tuning with Pipeline

In [467]:

```
scaler = StandardScaler()
param_grid = dict(C=np.logspace(-5, 5, 11), penalty=['l1', 'l2'])
#clf_lr = LogisticRegression(random_state=42)
#cv = GridSearchCV(estimator=clf_lr, param_grid=param_grid, scoring='average_precision')
cv = GridSearchCV(estimator=clf_lr_smote, param_grid=param_grid, scoring='average_precision')
pipeline = make_pipeline(scaler, cv)

pipeline.fit(X_train, y_train)

y_true = y_test
y_pred = pipeline.predict(X_test)
y_score = pipeline.predict_proba(X_test)[:, 1]
```

```
 Returns
```

Evaluation

Confusion Matrix

In [493]:

```
# Classification Report
target_names = ['Class 0 (PAYS)', 'Class 1 (DOES NOT PAY)']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
                        precision    recall  f1-score   support

        Class 0 (PAYS)       1.00      0.81      0.89    127230
Class 1 (DOES NOT PAY)       0.04      0.76      0.08      1454

             micro avg       0.81      0.81      0.81    128684
             macro avg       0.52      0.79      0.49    128684
          weighted avg       0.99      0.81      0.88    128684
```

In [490]:

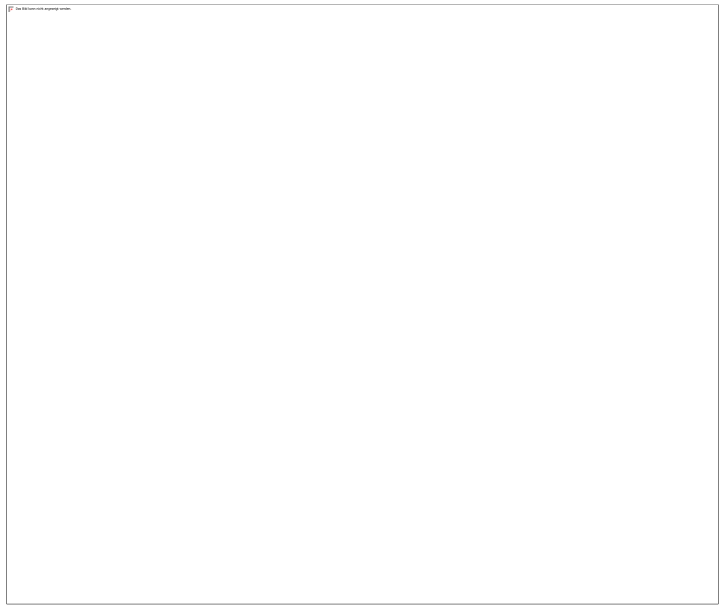
```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
```

```
                      normalize=True):
"""
given a sklearn confusion matrix (cm), make a nice plot

Arguments
---------
cm:           confusion matrix from sklearn.metrics.confusion_matrix

target_names: given classification classes such as [0, 1, 2]
              the class names, for example: ['high', 'medium', 'low']

title:        the text to display at the top of the matrix

cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
              see http://matplotlib.org/examples/color/colormaps_reference.html
              plt.get_cmap('jet') or plt.cm.Blues

normalize:    If False, plot the raw numbers
              If True, plot the proportions

Usage
-----
plot_confusion_matrix(cm           = cm,                  # confusion matrix created by
                                                          # sklearn.metrics.confusion_matrix
                      normalize    = True,                # show proportions
                      target_names = y_labels_vals,       # list of names of the classes
                      title        = best_estimator_name) # title of graph

Citiation
---------
http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

"""
import matplotlib.pyplot as plt
import numpy as np
import itertools

accuracy = np.trace(cm) / float(np.sum(cm))
misclass = 1 - accuracy

if cmap is None:
    cmap = plt.get_cmap('Blues')

plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()

if target_names is not None:
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names, rotation=45)
    plt.yticks(tick_marks, target_names)

if normalize:
```

```
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()
```

In [492]:

```
plot_confusion_matrix(cm=confusion_matrix(y_true=y_true, y_pred=y_pred),
                      normalize    = True,
                      target_names = ['Class 0 (PAYS)','Class 1 (DOES NOT PAY)'],
                      title        = "Confusion Matrix, Normalized")
```



Precision and Recall

In [472]:

```
from sklearn.metrics import precision_score, recall_score

print("Precision:", precision_score(y_train, predictions))
print("Recall:",recall_score(y_train, predictions))
```

```
Precision: 0.8309613257325421
Recall: 0.616060012157545
```

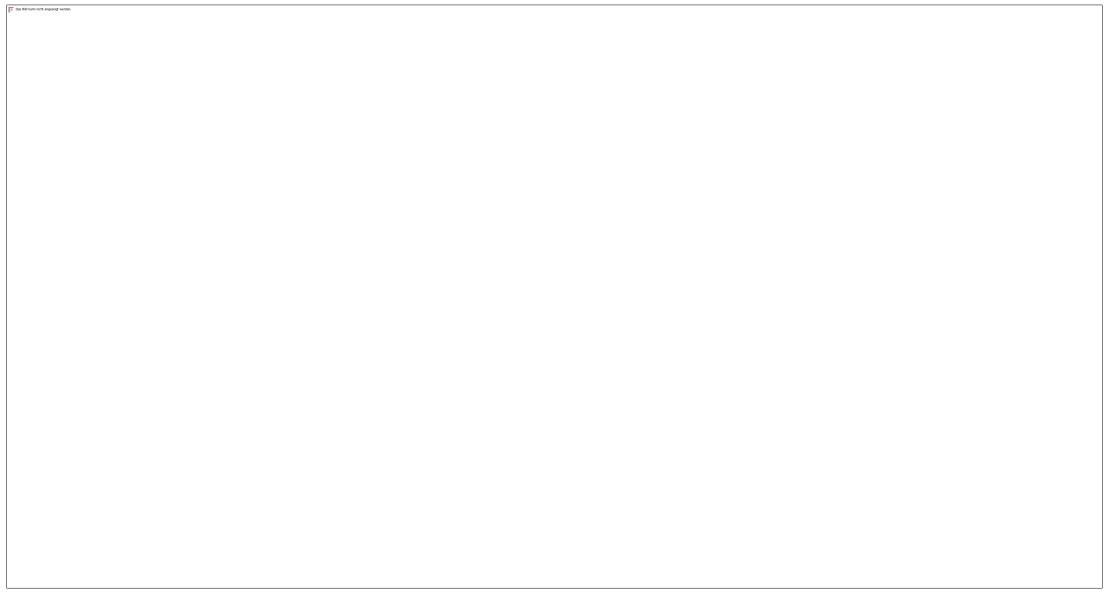
In [473]:

```
# Precision and Recall Curve
```

```
from sklearn.metrics import precision_recall_curve

# getting the probabilities of our predictions
y_scores = clf_lr_smote.predict_proba(X_train)
y_scores = y_scores[:,1]

precision, recall, threshold = precision_recall_curve(y_train, y_scores)
```

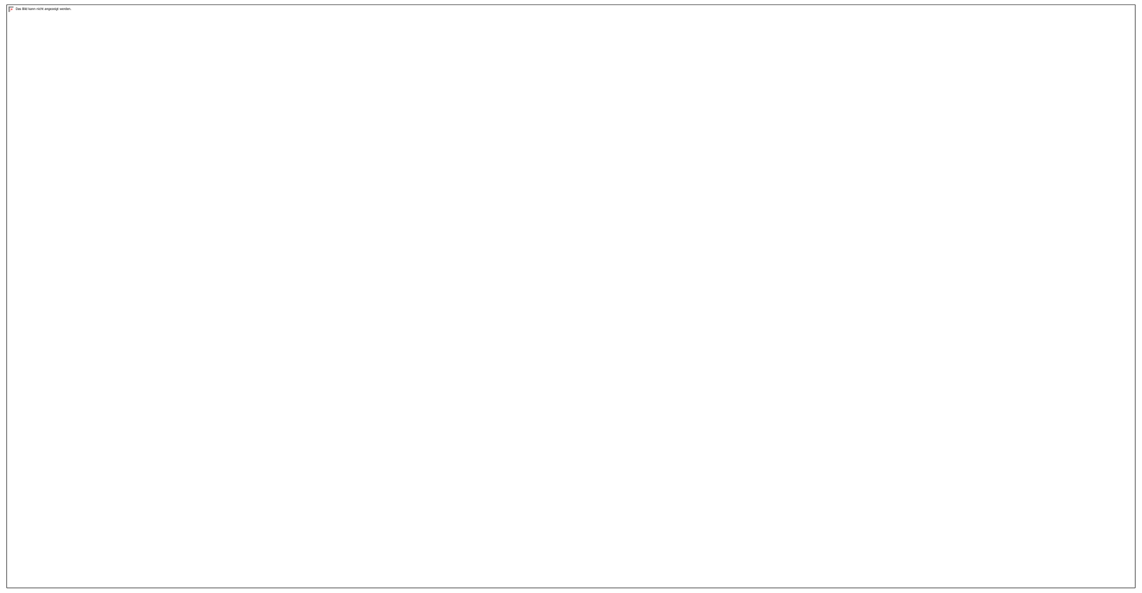
In [474]:

```
def plot_precision_and_recall(precision, recall, threshold):
    plt.plot(threshold, precision[:-1], "r-", label="precision", linewidth=5)
    plt.plot(threshold, recall[:-1], "b", label="recall", linewidth=5)
    plt.xlabel("threshold", fontsize=19)
    plt.legend(loc="upper right", fontsize=19)
    plt.ylim([0, 1])

plt.figure(figsize=(14, 7))
plot_precision_and_recall(precision, recall, threshold)
plt.show()
```



In [475]:

```
def plot_precision_vs_recall(precision, recall):
    plt.plot(recall, precision, "g--", linewidth=2.5)
    plt.ylabel("recall", fontsize=19)
    plt.xlabel("precision", fontsize=19)
    plt.axis([0, 1.5, 0, 1.5])

plt.figure(figsize=(14, 7))
plot_precision_vs_recall(precision, recall)
plt.show()
```



F-Score

In [476]:

```
from sklearn.metrics import f1_score
f1_score(y_train, predictions)
```

Out[476]:

```
0.7075528619082799
```

ROC AUC

ROC AUC Curve

In [477]:

```
from sklearn.metrics import roc_curve
# compute true positive rate and false positive rate
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_scores)
```

In [478]:

```
# plotting them against each other
def plot_roc_curve(false_positive_rate, true_positive_rate, label=None):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'r', linewidth=4)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (FPR)', fontsize=16)
    plt.ylabel('True Positive Rate (TPR)', fontsize=16)

plt.figure(figsize=(14, 7))
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()
```

Das Bild kann nicht angezeigt werden.

ROC AUC Score

In [479]:

```
from sklearn.metrics import roc_auc_score
r_a_score = roc_auc_score(y_train, y_scores)
print("ROC-AUC-Score:", r_a_score)
```

```
ROC-AUC-Score: 0.8799482937502328
```