

In [1]:

```
# here comes the imports!

import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt;
%matplotlib inline

import datetime
import time
```

In [2]:

```
# imports relativos a machine learning
import sklearn
from sklearn.model_selection import train_test_split

import h2o
from h2o.automl import H2OAutoML

from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_auc_score
```

Versões dos pacotes utilizados!

Isto é relevante com respeito às questões 3 e 4, em que é pedido para especificar as versões das bibliotecas de Python para que os resultados possam ser reproduzidos.

In [3]:

```
print('A versão do numpy é {}'.format(np.__version__))
print('A versão do pandas é {}'.format(pd.__version__))
print('A versão do seaborn é {}'.format(sns.__version__))
print('A versão do sklearn é {}'.format(sklearn.__version__))
print('A versão do h2o é {}'.format(h2o.__version__))
```

A versão do numpy é 1.16.2
A versão do pandas é 0.23.4
A versão do seaborn é 0.9.0
A versão do sklearn é 0.20.3
A versão do h2o é 3.18.0.2

vamos ler o dataset utilizando o pandas!

In [4]:

```
data_json='./dataset.json'
df=pd.read_json(data_json)
```

In [5]:

df

Out[5]:

	branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_id	quantity	re
0	0	143	0	0.0	854	292.91	21804	10	10
1	0	433	0	0.0	246	287.19	5486	20	16
2	0	486	0	0.0	1420	184.84	22662	12	24
3	0	107	0	0.0	1963	189.18	3956	18	28
4	0	768	0	0.0	1786	66.87	4730	5	17
5	0	740	0	0.0	2854	226.80	1835	10	07

In [7]:

```
# verificando o tipo das variaveis  
df.dtypes
```

Out[7]:

```
branch_id           int64  
customer_code       int64  
group_code          int64  
is_churn            float64  
item_code           int64  
item_total_price    float64  
order_id            int64  
quantity            int64  
register_date        object  
sales_channel        int64  
segment_code        int64  
seller_code         int64  
total_price         float64  
unit_price          float64  
dtype: object
```

Os tipos das variaveis parecem bem razoaveis

A coluna 'register_date' que é do tipo objeto, ela representa datas

Vamos ver as dimensões desse DataFrame!

Podemos acessar isso através do atributo .shape

In [8]:

```
df.shape
```

Out[8]:

```
(204428, 14)
```

In [9]:

```
print('Temos {0} linhas.'.format(df.shape[0]))  
print('Temos {0} colunas.'.format(df.shape[1]))
```

Temos 204428 linhas.
Temos 14 colunas.

Verificando se há duplicatas!

In [10]:

```
print(df.duplicated().any())  
print(df.duplicated().sum())
```

True
53

Há 53 duplicatas! Vamos retirá-las.

In [11]:

```
df.drop_duplicates(inplace=True)
```

In [12]:

```
# verificando, novamente, se há duplicatas  
  
print(df.duplicated().any())  
print(df.duplicated().sum())
```

False
0

Não há mais valores duplicados!

In []:

Vamos ver valores faltantes nesse DataFrame

In [13]:

```
df.isnull().any()
```

Out[13]:

```
branch_id      False
customer_code   False
group_code      False
is_churn        True
item_code       False
item_total_price False
order_id        False
quantity        False
register_date    False
sales_channel   False
segment_code    False
seller_code     False
total_price     False
unit_price      False
dtype: bool
```

In [14]:

```
df.isnull().sum()
```

Out[14]:

```
branch_id      0
customer_code   0
group_code      0
is_churn      1909
item_code       0
item_total_price 0
order_id        0
quantity        0
register_date    0
sales_channel    0
segment_code     0
seller_code      0
total_price      0
unit_price       0
dtype: int64
```

só há valores faltantes na coluna 'is_churn'

Há 1909 valores faltantes na coluna 'is_churn', o que pode parecer bastante, mas o correto é ver como esse número se compara com o total de linhas

In [15]:

```
print('Porcentagem de valores faltantes {0}%'.format(100*1909/df.shape[0]))
```

Porcentagem de valores faltantes 0.9340672782874617%

Logo, temos menos de 1% de valores faltantes!

Vamos proceder removendo esses valores, que são relativamente poucos, que estão faltando na coluna 'is_churn'

In [16]:

```
df.dropna(inplace=True)
```

Vamos verificar se ainda há valores faltantes

In [17]:

```
df.isnull().any()
```

Out[17]:

branch_id	False
customer_code	False
group_code	False
is_churn	False
item_code	False
item_total_price	False
order_id	False
quantity	False
register_date	False
sales_channel	False
segment_code	False
seller_code	False
total_price	False
unit_price	False
dtype:	bool

Não há mais valores faltantes!

Lidando com a coluna que é do tipo objeto, que

representa data!

In [18]:

```
df['register_date']
```

Out[18]:

```
0      2017-11-10T00:00:00Z
1      2011-05-16T00:00:00Z
2      2018-01-24T00:00:00Z
3      2010-07-28T00:00:00Z
4      2010-12-17T00:00:00Z
5      2009-05-07T00:00:00Z
6      2013-12-17T00:00:00Z
7      2009-09-29T00:00:00Z
8      2017-11-09T00:00:00Z
9      2008-08-19T00:00:00Z
10     2011-01-27T00:00:00Z
11     2013-07-31T00:00:00Z
12     2016-07-19T00:00:00Z
13     2015-02-18T00:00:00Z
14     2017-06-05T00:00:00Z
15     2008-01-18T00:00:00Z
16     2017-07-10T00:00:00Z
17     2018-01-29T00:00:00Z
18     2018-06-13T00:00:00Z
19     2016-03-15T00:00:00Z
20     2016-03-10T00:00:00Z
21     2011-07-09T00:00:00Z
22     2018-03-21T00:00:00Z
23     2016-10-24T00:00:00Z
24     2010-05-07T00:00:00Z
25     2011-09-22T00:00:00Z
26     2008-06-26T00:00:00Z
27     2011-04-29T00:00:00Z
28     2017-02-02T00:00:00Z
29     2013-12-04T00:00:00Z
...
204398 2014-02-21T00:00:00Z
204399 2009-02-19T00:00:00Z
204400 2015-08-28T00:00:00Z
204401 2017-08-15T00:00:00Z
204402 2012-09-29T00:00:00Z
204403 2015-03-16T00:00:00Z
204404 2010-07-27T00:00:00Z
204405 2015-02-11T00:00:00Z
204406 2015-04-15T00:00:00Z
204407 2015-11-17T00:00:00Z
204408 2015-05-29T00:00:00Z
204409 2009-07-30T00:00:00Z
204410 2016-11-17T00:00:00Z
204411 2016-09-23T00:00:00Z
204412 2017-01-24T00:00:00Z
204413 2018-05-18T00:00:00Z
204414 2016-08-16T00:00:00Z
204415 2014-01-24T00:00:00Z
204416 2008-07-10T00:00:00Z
204417 2012-08-21T00:00:00Z
204418 2015-10-02T00:00:00Z
```



```
204419    2017-10-19T00:00:00Z
204420    2018-06-28T00:00:00Z
204421    2018-01-24T00:00:00Z
204422    2008-09-09T00:00:00Z
204423    2017-02-15T00:00:00Z
204424    2018-05-23T00:00:00Z
204425    2014-10-20T00:00:00Z
204426    2015-08-18T00:00:00Z
204427    2013-02-26T00:00:00Z
```

Name: register_date, Length: 202466, dtype: object

In [19]:

```
# vamos lidar com essa coluna!
```

redefinindo os índices

In [21]:

```
df=df.reset_index(drop=True)
```

In []:

In [22]:

```
# Usando série temporal!
```

In [23]:

```
df3=df.copy()
```

In [24]:

```
df3['Date']=pd.to_datetime(df3.register_date)
```

In [25]:

```
df3.sort_values(by='Date',inplace=True)
```

In [26]:

```
df3.drop('register_date',axis=1,inplace=True)
```

In [27]:

```
df3
```

Out[27]:

	branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_id	quantity	sa
91229	0	588	0	0.0	1768	166.04	0	2	
163387	0	588	0	0.0	2675	255.90	0	20	
51446	0	588	0	0.0	282	363.08	0	20	
81273	0	114	2	1.0	2675	382.65	1	50	
132216	0	435	0	0.0	2624	495.83	2	32	
13263	0	681	0	0.0	1778	116.56	6	3	

In [28]:

```
#df3=df3.reset_index(drop=True,inplace=True)  
df3=df3.reset_index(drop=True)
```

In [29]:

df3

Out[29]:

	branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_id	quantity	sa
0	0	588	0	0.0	1768	166.04	0	2	
1	0	588	0	0.0	2675	255.90	0	20	
2	0	588	0	0.0	282	363.08	0	20	
3	0	114	2	1.0	2675	382.65	1	50	
4	0	435	0	0.0	2624	495.83	2	32	
5	0	681	0	0.0	1778	116.56	6	3	

In [30]:

```
# Extracting date features
#df3['Date'].dt.weekofyear

df3['dayofmonth'] = df3['Date'].dt.day
df3['dayofyear'] = df3['Date'].dt.dayofyear
df3['dayofweek'] = df3['Date'].dt.dayofweek
df3['month'] = df3['Date'].dt.month
df3['year'] = df3['Date'].dt.year
df3['weekofyear'] = df3['Date'].dt.weekofyear
df3['is_month_start'] = (df3['Date'].dt.is_month_start).astype(int)
df3['is_month_end'] = (df3['Date'].dt.is_month_end).astype(int)
```

In [31]:

```
df3.drop('Date',axis=1,inplace=True)
```

In [32]:

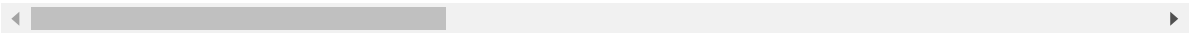
df3

Out[32]:

	branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_
0	0	588	0	0.0	1768	166.04	
1	0	588	0	0.0	2675	255.90	
2	0	588	0	0.0	282	363.08	
3	0	114	2	1.0	2675	382.65	
4	0	435	0	0.0	2624	495.83	
5	0	681	0	0.0	1778	116.56	
6	0	681	0	0.0	416	128.46	
7	0	613	0	0.0	1767	520.61	
8	0	681	0	0.0	282	155.95	
9	0	681	0	0.0	2630	176.85	
10	0	558	0	0.0	2630	863.53	
11	0	435	0	0.0	1774	171.35	
12	0	681	0	0.0	1785	108.26	
13	0	435	0	0.0	1963	165.25	
14	0	613	0	0.0	2630	139.70	
15	0	435	0	0.0	1967	247.93	
16	0	613	0	0.0	390	130.66	
17	0	507	0	0.0	1778	206.39	
18	0	613	0	0.0	360	118.70	
19	0	435	0	0.0	467	185.13	
20	0	681	0	0.0	1787	46.39	
21	0	613	0	0.0	2624	147.05	
22	0	435	0	0.0	2259	162.49	
23	0	507	0	0.0	2630	142.46	
24	0	507	0	0.0	1767	278.32	
25	0	507	0	0.0	246	303.55	
26	0	435	0	0.0	1966	326.06	
27	0	666	0	0.0	1767	624.73	
28	0	666	0	0.0	282	115.49	
29	0	681	0	0.0	1786	46.39	
...	
202436	0	557	0	0.0	2624	218.40	246
202437	0	35	0	0.0	1778	127.85	246

	branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_
202438	0	311	0	0.0	24	71.65	246
202439	0	311	0	0.0	2861	327.50	246
202440	0	311	0	0.0	2860	131.00	246
202441	0	311	0	0.0	2847	334.62	246
202442	0	557	0	0.0	795	329.22	246
202443	0	35	0	0.0	1781	108.66	246
202444	0	35	0	0.0	1414	315.32	246
202445	0	35	0	0.0	1787	51.14	246
202446	0	35	0	0.0	1410	235.43	246
202447	0	311	0	0.0	2844	225.20	246
202448	0	557	0	0.0	1767	376.20	246
202449	0	35	0	0.0	1601	193.12	246
202450	0	311	0	0.0	2865	520.25	246
202451	0	35	0	0.0	1605	396.01	246
202452	0	250	2	0.0	1159	4.23	246
202453	0	35	0	0.0	246	277.26	246
202454	0	250	2	0.0	1740	22.44	246
202455	0	557	0	0.0	2630	327.60	246
202456	0	35	0	0.0	1412	235.43	246
202457	0	311	0	0.0	2866	353.84	246
202458	0	557	0	0.0	359	252.36	246
202459	0	114	2	1.0	773	101.95	246
202460	0	114	2	1.0	889	175.49	246
202461	0	114	2	1.0	1924	76.74	246
202462	0	759	0	0.0	2854	126.01	246
202463	0	759	0	0.0	1970	1211.05	246
202464	0	759	0	0.0	2863	374.40	246
202465	0	759	0	0.0	2624	361.70	246

202466 rows × 21 columns



In []:

In []:

In []:

In []:

Agora vamos às questões!

Question 1 (10 Points)

List as many use cases for the dataset as possible.

Re: Os casos de uso de uso para o conjunto de dados são:

1. saber em função do preço total 'total price' quantos clientes estão comprando certo item, criando intervalos para o preço total onde há maior lucro;
2. saber qual canal de venda gera maior lucro 'sales_channel', determinando uma preferência e onde investir;
3. determinar a partir de 'register_date' (data de pedido) qual ano período gerou maior lucro;
4. determinar a partir de 'total_price' (preço total) qual faixa gera mais lucro, buscando assim maximizar o lucro e otimizar o tipo de cliente;
5. a partir de 'segment_code' (código de segmento que o cliente pertence) otimizar ...
6. a partir de 'group_code' (código de grupo) otimizar o lucro da empresa e encontrar qual é o grupo ótimo de consumidor em que se deve ser investido mais esforços;
7. criar um modelo preditivo a partir de 'is_churn' para obter informações sobre qual grupo, segmento, canal de venda, vendedor, tem maior chance de que esse cliente retorne;
8. estudar qual 'seller_code' (código de vendedor) tem mais sucesso com os clientes.

Question 2 (10 Points)

Pick one of the use cases you listed in question 1 and describe how building a statistical model based on the dataset could best be used to improve the business this data comes from.

Re: Escolhendo o caso de uso 7 da questão 1 podemos criar um modelo estatístico, de machine learning, para prever se um dado cliente voltará, estudando então a influência dos features na predição dos dados.

Question 3 (20 Points)

Implement the model you described in question 2, preferably in Python. The code has to retrieve the data, train and test a statistical model, and report relevant performance criteria. Ideally, we should be able to replicate your analysis from your submitted source-code, so please explicit the versions of the tools and packages you are using.

A versão dos pacotes!

In [33]:

```
print('A versão do numpy é {}'.format(np.__version__))
print('A versão do pandas é {}'.format(pd.__version__))
print('A versão do seaborn é {}'.format(sns.__version__))
print('A versão do sklearn é {}'.format(sklearn.__version__))
print('A versão do h2o é {}'.format(h2o.__version__))
```

A versão do numpy é 1.16.2
A versão do pandas é 0.23.4
A versão do seaborn é 0.9.0
A versão do sklearn é 0.20.3
A versão do h2o é 3.18.0.2

separando em dados de treino e teste

In [34]:

```
#df_ml=df2.copy()
df_ml=df3.copy()
```

In [35]:

```
#msk = np.random.random(len(df_ml)) < 0.8
# criando a mascara
percent=0.66

msk=[]
for i in range(len(df_ml)):
    msk.append(i<=percent*len(df_ml))
msk=np.array(msk)
```

In [36]:

```
train = df_ml[msk]
```

In [37]:

```
test=df_ml[~msk]
```

Vamos utilizar h2o para automatizar os cálculos estatísticos de machine learning.

Vamos inicializar h2o!

In [40]:

```
h2o.init()
```

Checking whether there is an H2O instance running at <http://localhost:54321>. (<http://localhost:54321>.) connected.
Warning: Your H2O cluster version is too old (1 year, 5 months and 6 days)! Please download and install the latest version from <http://h2o.ai/download/> (<http://h2o.ai/download/>)

H2O cluster uptime:	11 secs
H2O cluster timezone:	America/Sao_Paulo
H2O data parsing timezone:	UTC
H2O cluster version:	3.18.0.2
H2O cluster version age:	1 year, 5 months and 6 days !!!
H2O cluster name:	H2O_from_python_vagner_ig2b41
H2O cluster total nodes:	1
H2O cluster free memory:	3.433 Gb
H2O cluster total cores:	8
H2O cluster allowed cores:	8
H2O cluster status:	locked, healthy
H2O connection url:	http://localhost:54321
H2O connection proxy:	None
H2O internal security:	False
H2O API Extensions:	XGBoost, Algos, AutoML, Core V3, Core V4
Python version:	3.6.9 final

In [41]:

```
train.iloc[-1]
```

Out[41]:

branch_id	0.00
customer_code	377.00
group_code	0.00
is_churn	0.00
item_code	1048.00
item_total_price	97.08
order_id	16241.00
quantity	6.00
sales_channel	0.00
segment_code	0.00
seller_code	50.00
total_price	3216.77
unit_price	16.18
dayofmonth	17.00
dayofyear	48.00
dayofweek	2.00
month	2.00
year	2016.00
weekofyear	7.00
is_month_start	0.00
is_month_end	0.00

Name: 133627, dtype: float64

In [42]:

```
test.iloc[1]
```

Out[42]:

```
branch_id          0.00
customer_code      428.00
group_code         0.00
is_churn           0.00
item_code          410.00
item_total_price   138.60
order_id          16245.00
quantity           3.00
sales_channel      0.00
segment_code       0.00
seller_code        39.00
total_price        1697.39
unit_price         45.64
dayofmonth         17.00
dayofyear          48.00
dayofweek          2.00
month              2.00
year              2016.00
weekofyear         7.00
is_month_start     0.00
is_month_end       0.00
Name: 133629, dtype: float64
```

In [43]:

```
train=h2o.H2OFrame(train)
```

```
/home/vagner/anaconda3/lib/python3.6/site-packages/h2o/utils/shared_u
tils.py:170: FutureWarning: Method .as_matrix will be removed in a fu
ture version. Use .values instead.
```

```
data = _handle_python_lists(python_obj.as_matrix().tolist(), -1)[1]
```


```
Parse progress: |
```

```
100%
```

In [44]:

```
test=h2o.H2OFrame(test)
```

```
/home/vagner/anaconda3/lib/python3.6/site-packages/h2o/utils/shared_utils.py:170: FutureWarning: Method .as_matrix will be removed in a future version. Use .values instead.
  data = _handle_python_lists(python_obj.as_matrix().tolist(), -1)[1]
```

Parse progress: |  | 100%

In [45]:

```
# Identify predictors and response
x = train.columns
y = "is_churn"
x.remove(y)
```

In [46]:

```
x
```

Out[46]:

```
['branch_id',
 'customer_code',
 'group_code',
 'item_code',
 'item_total_price',
 'order_id',
 'quantity',
 'sales_channel',
 'segment_code',
 'seller_code',
 'total_price',
 'unit_price',
 'dayofmonth',
 'dayofyear',
 'dayofweek',
 'month',
 'year',
 'weekofyear',
 'is_month_start',
 'is_month_end']
```

In []:

Let us visualize train

In [47]:

```
train
```

branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_id	quan
0	588	0	0	1768	166.04	0	
0	588	0	0	2675	255.9	0	
0	588	0	0	282	363.08	0	
0	114	2	1	2675	382.65	1	
0	435	0	0	2624	495.83	2	
0	681	0	0	1778	116.56	6	
0	681	0	0	416	128.46	6	
0	613	0	0	1767	520.61	4	
0	681	0	0	282	155.95	6	
0	681	0	0	2630	176.85	6	

Out[47]:

Let us visualize the test

In [48]:

```
test
```

branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_id	quan
0	237	0	0	1767	542.66	16232	
0	428	0	0	410	138.6	16245	
0	812	0	0	1760	14.4	16240	
0	636	0	0	204	117.08	16233	
0	103	0	0	361	295.08	16230	
0	636	0	0	1788	180.6	16233	
0	636	0	0	2630	183.32	16233	
0	237	0	0	1767	542.66	16229	
0	377	0	0	1045	149.34	16241	
0	812	0	0	1434	100.3	16240	

Out[48]:

In []:

In []:

In [49]:

```
# For binary classification, response should be a factor  
train[y] = train[y].asfactor()
```

In [50]:

```
test[y] = test[y].asfactor()
```


In [55]:

```
aml.leader
```

```
Model Details
=====
H2OGradientBoostingEstimator : Gradient Boosting Machine
Model Key: GBM_grid_0_AutoML_20190811_201902_model_0

ModelMetricsBinomial: gbm
** Reported on train data. **

MSE: 3.285506325756697e-35
RMSE: 5.731933640366658e-18
LogLoss: 1.484852297690794e-19
Mean Per-Class Error: 0.0
AUC: 1.0
Gini: 1.0
Confusion Matrix (Act/Pred) for max f1 @ threshold = 1.0:

      0      1   Error   Rate
```

In [56]:

```
aml.leaderboard
```

	model_id	auc	logloss
GBM_grid_0_AutoML_20190811_201902_model_0		1,000000	0,000000
GBM_grid_0_AutoML_20190811_201902_model_1		1,000000	0,000000
DRF_0_AutoML_20190811_201902		1,000000	0,005682
XRT_0_AutoML_20190811_201902		0,999999	0,018917
GLM_grid_0_AutoML_20190811_201902_model_0		0,945772	0,174354

Out[56]:

```
# If you need to generate predictions on a test set, you can make
# predictions directly on the `H2OAutoML` object, or on the leader
# model object directly
```

[illegible]

```
(train.as_data_frame)
```

branch_id	customer_code	group_code	is_churn	item_code	item_total_price	order_id	quantity
0	588	0	0	1768	166.04	0	1
0	588	0	0	2675	255.9	0	1
0	588	0	0	282	363.08	0	1
0	114	2	1	2675	382.65	1	1
0	435	0	0	2624	495.83	2	1
0	681	0	0	1778	116.56	6	1
0	681	0	0	416	128.46	6	1
0	613	0	0	1767	520.61	4	1
0	681	0	0	282	155.95	6	1
0	681	0	0	2630	176.85	6	1

```
<bound method H2OFrame.as_data_frame of >
```



```
lb=aml.leaderboard
lb.head( rows=lb.nrows)
```

	model_id	auc	logloss
GBM_grid_0_AutoML_20190811_201902_model_0		1,000000	0,000000
GBM_grid_0_AutoML_20190811_201902_model_1		1,000000	0,000000
DRF_0_AutoML_20190811_201902		1,000000	0,005682
XRT_0_AutoML_20190811_201902		0,999999	0,018917
GLM_grid_0_AutoML_20190811_201902_model_0		0,945772	0,174354

In [60]:

```
print('Generate Predictions!')  
  
testy=aml_leader.predict(test)  
#testy=testy.as_data_frame()
```

```
gbm prediction progress: |██████████████████████████████████|  
█████████| 100%
```

In [61]:

```
# very nice  
testy
```

predict	p0	p1
0	1	1e-19
0	1	1e-19
0	1	1e-19
0	1	1e-19
0	1	1e-19
0	1	1e-19
0	1	1e-19
0	1	1e-19
0	1	3.51619e-19
0	1	1e-19

Out[61]:

In [405]:

```
%(test['is_churn']==testy['predict']).sum()/testy.shape[0]
```

In [62]:

```
test['is_churn']
```

is_churn

0
0
0
0
0
0
0
0
0
0

Out[62]:

In [63]:

```
testy['predict']
```

predict

0
0
0
0
0
0
0
0
0
0
0

Out[63]:

Calculation of some metrics of the predicted model!

The f1_score

In [65]:

```
f1_score(testy['predict'].as_data_frame(),test['is_churn'].as_data_frame())
```

Out[65]:

0.762225147481839

The Recall

In [66]:

```
recall_score(testy['predict'].as_data_frame(),test['is_churn'].as_data_frame())
```

Out[66]:

1.0

The Precision

In [67]:

```
precision_score(testy['predict'].as_data_frame(),test['is_churn'].as_data_frame())
```

Out[67]:

0.6158027414526548

The ROC AUC score

In [68]:

```
roc_auc_score(testy['predict'].as_data_frame(),test['is_churn'].as_data_frame())
```

Out[68]:

0.9600383474541551

O valor de 'ROC AUC Score' acima, que é próximo a 1, mostra que temos um modelo bom para classificação binária do target 'is_churn'.

In []:

In []:

In []:

Question 4 (60 Points)

A. Explain each and every of your design choices, you can use jupyter notebooks. (e.g., preprocessing, model selection, hyper parameters, evaluation criteria). Compare and contrast your choices with alternative methodologies.

Foi escolhido utilizar o h2o para efetuar os cálculos de machine learning, pois é um pacote que otimiza o tempo de cálculo de cada algoritmo e tem uma ótima automatização. O h2o consegue achar o melhor modelo e qual modelo prediz melhor o 'target', juntamente com os parâmetros do modelo que vão gerar os melhores resultados, permitindo saber qual a melhor faixa de parâmetros a serem escolhidos no modelo.

Foi retirada a validação cruzada (cross-validation), colocando o parâmetro nfolds=0, uma vez que ela não é uma boa prática para séries temporais. A melhor técnica no caso para cross validation é a 'walkforward' [<https://medium.com/@samuel.monnier/cross-validation-tools-for-time-series-ffa1a5a09bf9>] (<https://medium.com/@samuel.monnier/cross-validation-tools-for-time-series-ffa1a5a09bf9%5D>). No entanto, essa técnica de cross validation não está implementada no automl.

A métrica mais interessante para o problema de prever 'is_churn', uma vez que é uma classificação binária, é a "ROC AUC" (Area Under the ROC Curve), pois ela está intimamente ligada com a taxa de true positives e falsos positivos.

O melhor modelo foi obtido com base na métrica "ROC AUC", que é feito automaticamente no h2o.

Alternativamente, é interessante olhar para as métricas recall e precision, uma vez que elas estão relacionadas com 'true positives' e 'falsos positivos'. Essas métricas são importantes que elas identificam bem casos em que houve uma predição errada [<https://medium.com/@yashwant140393/the-3-pillars-of-binary-classification-accuracy-precision-recall-d2da3d09f664>] (<https://medium.com/@yashwant140393/the-3-pillars-of-binary-classification-accuracy-precision-recall-d2da3d09f664%5D>).

In []:

In []:

B. Describe how you would improve the model in Question 3 if you had more time.

Seguem a seguir algumas melhorias que poderiam ser feitas no modelo (caso houvesse mais tempo):

1. O modelo poderia ser melhorado usando **feature engineering**, por exemplo, definindo novos features em termos dos antigos.
2. Também poderia ser trabalhada a **remoção de outliers**, removendo, por exemplo, 10% dos dados mais "fora da curva" que seriam os que estão mais distantes da média, com base em uma função custo que é a diferença entre predição (através de uma regressão linear dos dados) e o valor do ponto, e assim eliminando-os do conjunto de dados;
3. Também é possível aumentar o número de modelos testados e o tempo de execução total dos cálculos ou o tempo máximo de cada modelo;
4. Poderíamos também refinar o melhor modelo que foi encontrado (aqui LightGBM) e refinar os parâmetros que o h2o devolve ao final dos cálculos;
5. poderíamos também melhorar as features temporais e adicionar feriados regionais e nacionais.

In []: