Introduction

Health Insurance Cross Sell Prediction

Source: https://www.kaggle.com/datasets/anmolkumar/health-insurance-cross-sell-prediction

Context

Our client is an Insurance company that has provided Health Insurance to its customers now they need your help in building a model to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the company.

Just like medical insurance, there is vehicle insurance where every year customer needs to pay a premium of certain amount to insurance provider company so that in case of unfortunate accident by the vehicle, the insurance provider company will provide a compensation (called 'sum assured') to the customer.

Building a model to predict whether a customer would be interested in Vehicle Insurance is extremely helpful for the company because it can then accordingly plan its communication strategy to reach out to those customers and optimise its business model and revenue.

Now, The main question is to predict, whether the customer would be interested in Vehicle insurance or not, you have information about demographics (gender, age, region code, type), Vehicles (Vehicle Age, Damage), Policy (Premium, sourcing channel) etc.

Imports

In [1]:
```
# Libraries used in the Project

import numpy  as np
import pandas as pd
import boruta as bt
import scikitplot as skplt
import pickle
import seaborn as sns

from matplotlib import pyplot as plt
from sklearn import preprocessing   as pp
from sklearn import linear_model    as lm
from sklearn import model_selection as ms
from sklearn import ensemble        as en
from sklearn import neighbors       as nh
```

pandas:

Main Use: Data Manipulation and Analysis

pandas provides data structures like DataFrames and Series that make it easy to work with structured data, such as CSV files or SQL tables. It allows you to perform data cleaning, transformation, and analysis efficiently.

numpy:

Main Use: Numerical Computations

numpy is a library for numerical computations in Python. It provides support for working with arrays and matrices, making it essential for scientific computing, mathematical operations, and array manipulation.

seaborn:

Main Use: Data Visualization

seaborn is a data visualization library built on top of Matplotlib. It simplifies the creation of informative and attractive statistical graphics, including various types of plots such as scatter plots, bar plots, and heatmaps.

scikitplot:

Main Use: Machine Learning Model Evaluation

scikitplot is a library that extends Scikit-learn's functionality by providing tools for visualizing the performance of machine learning models. It offers functions to create various types of plots, such as ROC curves and confusion matrices, for model evaluation.

matplotlib.pyplot

Main Use: Data Visualization

matplotlib is a widely used library for creating static, animated, or interactive visualizations in Python. pyplot is a module within Matplotlib that provides a simple interface for creating various types of plots and charts.

sklearn.preprocessing:

Main Use: Data Preprocessing for Machine Learning

sklearn.preprocessing contains functions for data preprocessing tasks like scaling, encoding categorical variables, and imputing missing values. These preprocessing steps are crucial for preparing data for machine learning models.

sklearn.model_selection:

Main Use: Model Selection and Evaluation

sklearn.model_selection provides tools for splitting data into training and testing sets, cross-validation, and hyperparameter tuning. It's essential for evaluating and selecting the best machine learning models.

sklearn.ensemble:

Main Use: Ensemble Learning Methods

sklearn.ensemble contains various ensemble learning techniques like Random Forests, Gradient Boosting, and Bagging. Ensemble methods combine multiple models to improve predictive performance.

sklearn.neighbors:

Main Use: Nearest Neighbors Algorithms

sklearn.neighbors implements algorithms for solving nearest neighbors problems, such as k-nearest neighbors (KNN). These algorithms are useful for classification and regression tasks based on proximity to other data points.

sklearn.linear_model:

Main Use: Linear Models for Regression and Classification

sklearn.linear_model provides tools for working with linear models, including linear regression, logistic regression, and other linear-based models commonly used in machine learning.

Helper Fucntions

In [1]:

```
# Just make Jupyter visual better

from IPython.core.display  import HTML

def jupyter_settings():
    %matplotlib inline
    %pylab inline

    plt.style.use( 'bmh' )
    plt.rcParams['figure.figsize'] = [10, 5]
    plt.rcParams['font.size'] = 24

    display( HTML( '<style>.container { width:100% !important; }</style>') )
    pd.options.display.max_columns = None
    pd.options.display.max_rows = None
    pd.set_option( 'display.expand_frame_repr', False )

    sns.set()

jupyter_settings()

'''calculates precision at a given value of k for a dataset. Precision is a metric used to evaluate the performance of a ranking or
recommendation system. It measures the proportion of relevant items among the top k items in a ranked list.'''

def precision_at_k( data, k=10 ):
    data = data.reset_index( drop=True )
    data['ranking'] = data.index + 1
    data['precision_at_k'] = data['response'].cumsum() / data['ranking']

    return ( data.loc[ k, 'precision_at_k'], data )

'''calculates recall at a given rank k for a ranked list of items in the data DataFrame. Recall measures the proportion of relevant items
that were included among the top k items in a ranked list. It does so by computing the cumulative sum of relevant items up to each rank
and dividing it by the total number of relevant items in the dataset.'''

def recall_at_k( data, k=15 ):
    data = data.reset_index( drop=True )
    data['ranking'] = data.index + 1
    data['recall_at_k'] = data['response'].cumsum() / data['response'].sum()

    return ( data.loc[ k, 'recall_at_k'], data )
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 20
     16     pd.set_option( 'display.expand_frame_repr', False )
     18     sns.set()
---> 20 jupyter_settings()
     22 '''calculates precision at a given value of k for a dataset. Precision is a metric used to evaluate the performance of a ranking or
     23 recommendation system. It measures the proportion of relevant items among the top k items in a ranked list.'''
     25 def precision_at_k( data, k=10 ):

Cell In[1], line 14, in jupyter_settings()
     11 plt.rcParams['font.size'] = 24
     13 display( HTML( '<style>.container { width:100% !important; }</style>') )
---> 14 pd.options.display.max_columns = None
     15 pd.options.display.max_rows = None
     16 pd.set_option( 'display.expand_frame_repr', False )

NameError: name 'pd' is not defined
```

Loading Dataset

In [3]:

```
# Load the archive

df_raw = pd.read_csv( r'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\data\\raw\\train.csv',
```

In [4]:

```
# Taking a look at the dataset

df_raw.head()
```

Out[4]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0

Data Description

In [5]:
```python
# Creating the first checkpoint

df1 = df_raw.copy()
```

Renaming the Columns

In [6]:
```python
# Visualizing the columns

df1.columns
```

Out[6]:
```
Index(['id', 'Gender', 'Age', 'Driving_License', 'Region_Code',
       'Previously_Insured', 'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premium',
       'Policy_Sales_Channel', 'Vintage', 'Response'],
      dtype='object')
```

In [7]:
```python
# Tansforming the columns to ease the access later

cols_new = ['id', 'gender', 'age', 'driving_license', 'region_code', 'previously_insured',
            'vehicle_age', 'vehicle_damage', 'annual_premium', 'policy_sales_channel', 'vintage', 'response']

df1.columns = cols_new
```

Data Dimension

In [8]:
```python
# Get to know the dataset dimensions

print( f'Number of Rows {df1.shape[0]}')
print( f'Number of Columns {df1.shape[1]}')
```

```
Number of Rows 381109
Number of Columns 12
```

Data types

In [9]:
```python
# Get to know which types of data we'll work on, and if they need to be changed

df1.dtypes
```

Out[9]:
```
id                       int64
gender                  object
age                      int64
driving_license          int64
region_code            float64
previously_insured       int64
vehicle_age             object
vehicle_damage          object
annual_premium         float64
policy_sales_channel   float64
vintage                  int64
response                 int64
dtype: object
```

Check NAs

In [10]:
```
# Check if we've got NAs to be treated

df1.isna().sum()
```

Out[10]:
```
id                      0
gender                  0
age                     0
driving_license         0
region_code             0
previously_insured      0
vehicle_age             0
vehicle_damage          0
annual_premium          0
policy_sales_channel    0
vintage                 0
response                0
dtype: int64
```

Categorical X Numerical

In [11]:
```
# Divide the Dataset into two new datasets, one numerical types and another categorical

num_attributes = df1.select_dtypes( include=['int64', 'float64'] )
cat_attributes = df1.select_dtypes( exclude=['int64', 'float64', 'datetime64[ns]'] )
```

First view of our Numerical Attributes

In [12]:
```python
# Central Tendency Median and Mean

ct1 = pd.DataFrame( num_attributes.apply( np.mean ) ).T
ct2 = pd.DataFrame( num_attributes.apply( np.median ) ).T

# Dispersion - std, min, max, range, skew, kurtosis

d1 = pd.DataFrame( num_attributes.apply( np.std) ).T
d2 = pd.DataFrame( num_attributes.apply( min ) ).T
d3 = pd.DataFrame( num_attributes.apply( max ) ).T
d4 = pd.DataFrame( num_attributes.apply( lambda x: x.max() - x.min() ) ).T
d5 = pd.DataFrame( num_attributes.apply( lambda x: x.skew() ) ).T
d6 = pd.DataFrame( num_attributes.apply( lambda x: x.kurtosis() ) ).T

# Concatenate

m = pd.concat( [d2, d3, d4, ct1, ct2, d1, d5, d6] ).T.reset_index()
m.columns = ['Attributes', 'min', 'max', 'range', 'mean', 'median', 'std', 'skew', 'kurtosis']
m
```

Out[12]:

	Attributes	min	max	range	mean	median	std	skew	kurtosis
0	id	1.0	381109.0	381108.0	190555.000000	190555.0	110016.691870	9.443274e-16	-1.200000
1	age	20.0	85.0	65.0	38.822584	36.0	15.511591	6.725390e-01	-0.565655
2	driving_license	0.0	1.0	1.0	0.997869	1.0	0.046109	-2.159518e+01	464.354302
3	region_code	0.0	52.0	52.0	26.388807	28.0	13.229871	-1.152664e-01	-0.867857
4	previously_insured	0.0	1.0	1.0	0.458210	0.0	0.498251	1.677471e-01	-1.971871
5	annual_premium	2630.0	540165.0	537535.0	30564.389581	31669.0	17213.132474	1.766087e+00	34.004569
6	policy_sales_channel	1.0	163.0	162.0	112.034295	133.0	54.203924	-9.000081e-01	-0.970810
7	vintage	10.0	299.0	289.0	154.347397	154.0	83.671194	3.029517e-03	-1.200688
8	response	0.0	1.0	1.0	0.122563	0.0	0.327935	2.301906e+00	3.298788

Feature Engineering

In [13]:
```
# Change the yes and no to 1 and 0, to facilitate the handling later

df2 = df1.copy()

# Vehicle Damage Number

df2['vehicle_damage'] = df2['vehicle_damage'].apply( lambda x: 1 if x == 'Yes' else 0 )

# Vehicle Age

df2['vehicle_age'] =  df2['vehicle_age'].apply( lambda x: 'over_2_years' if x == '> 2 Years'
                                               else 'between_1_2_year' if x == '1-2 Year'
                                               else 'below_1_year' )
```

Data Filtering

In [14]:
```
# Here just creating a third checkpoint, for while no filtering needed to start the EDA

df3 = df2.copy()
```

EDA, Exploratory Data Analyze

In [15]:
```
# Fourth Checkpoint

df4 = df3.copy()
```

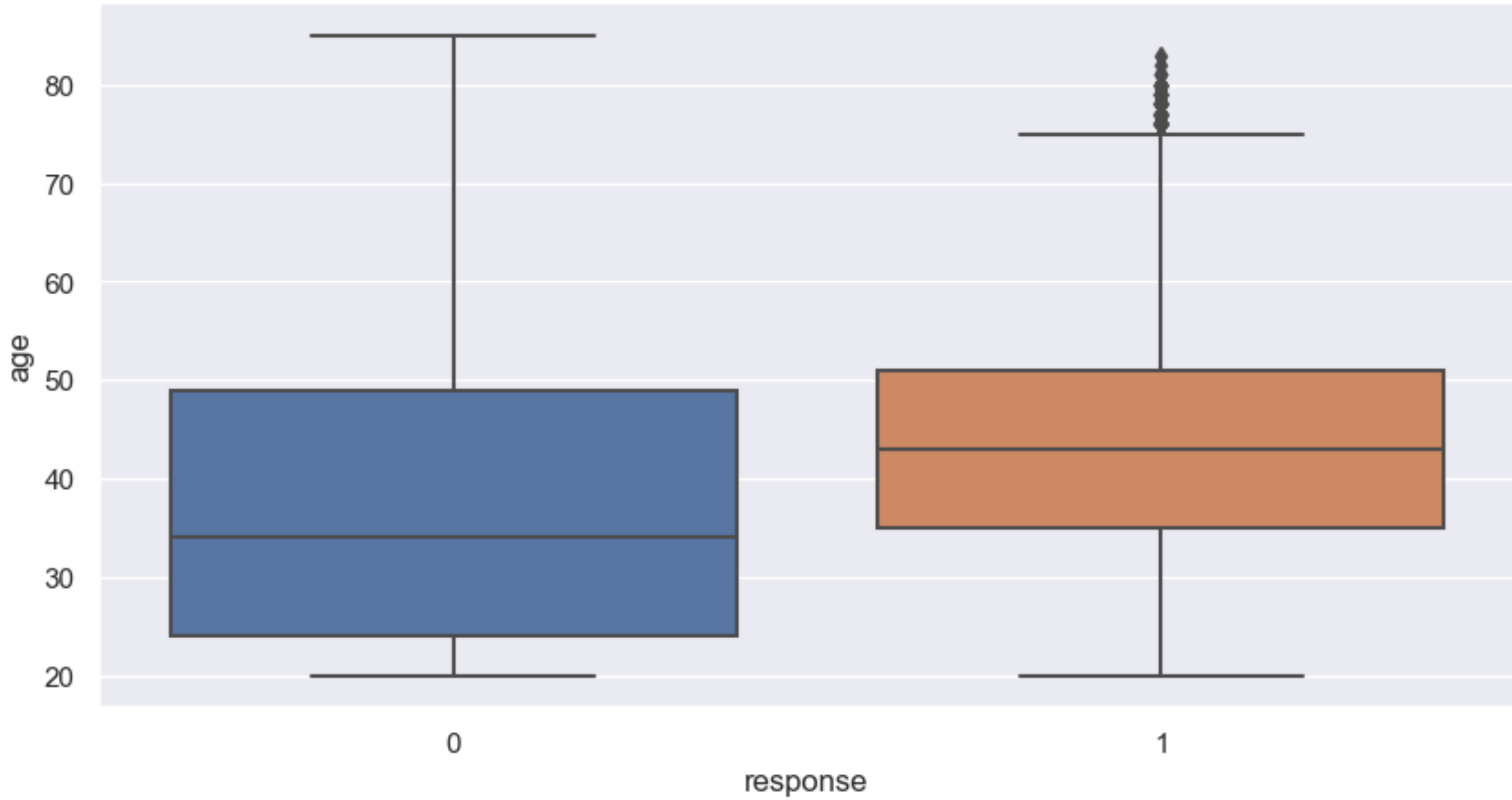
Univariate Analysis

Age (Try to identify which age group has more interest in signing a new insurance.)

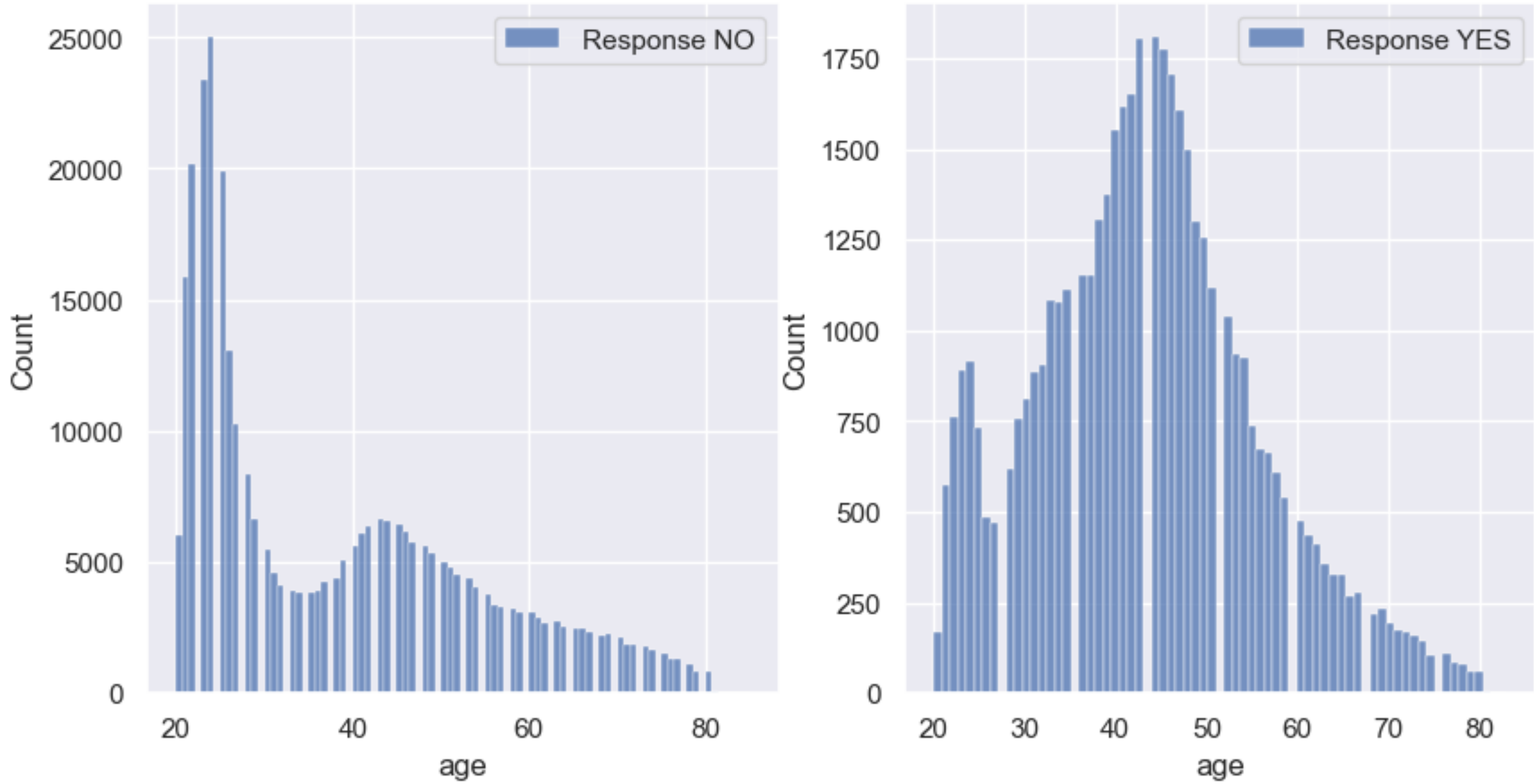
Analyzing the graphics below we may see that for yes answers we have an age concentration between 35 and 50 years, with a higher peak in 42 to 46, moreover, some outliers over 75 years

While no answers has a concentration between 25 to 30, with a higher peak in 22 to 26

In [16]:
```
# Use boxplot to see the distribuition between the age groups according their response

aux0 = df4[['age', 'response']]
sns.boxplot( x='response', y='age', data=aux0 );
```



In [17]:
```
# Histplot will give a best overview of the concetration

# No answer

plt.subplot( 1, 2, 1)

aux0 = df4.loc[df4['response'] == 0, 'age']
sns.histplot( aux0, label='Response NO' );
plt.legend()

# Yes answer

plt.subplot( 1, 2, 2)

aux1 = df4.loc[df4['response'] == 1, 'age']
sns.histplot( aux1, label='Response YES' );
plt.legend();
```

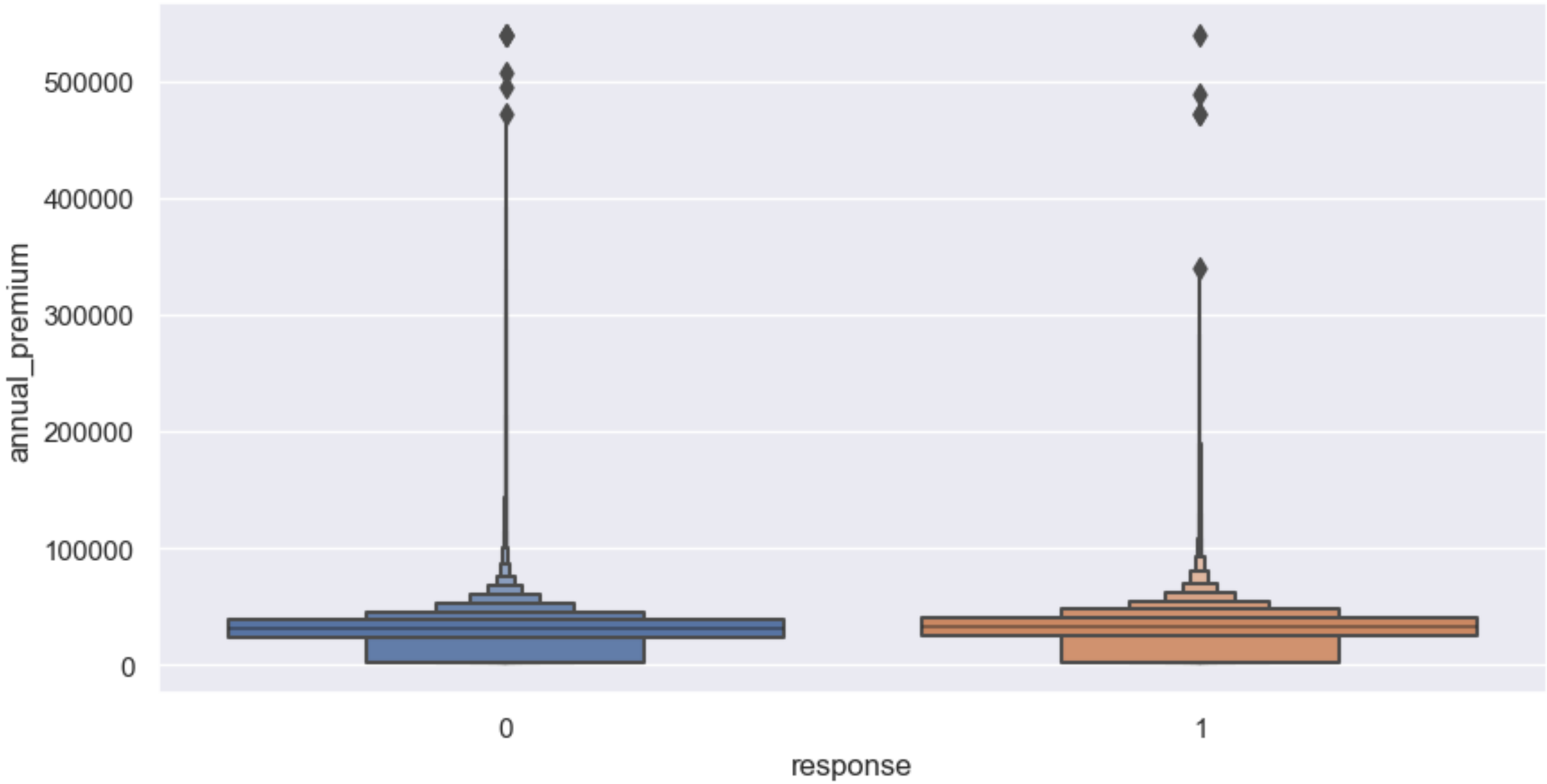


Annual Income (Check the Premium values most bought, see how much the premium value has to do with the client's decision.)

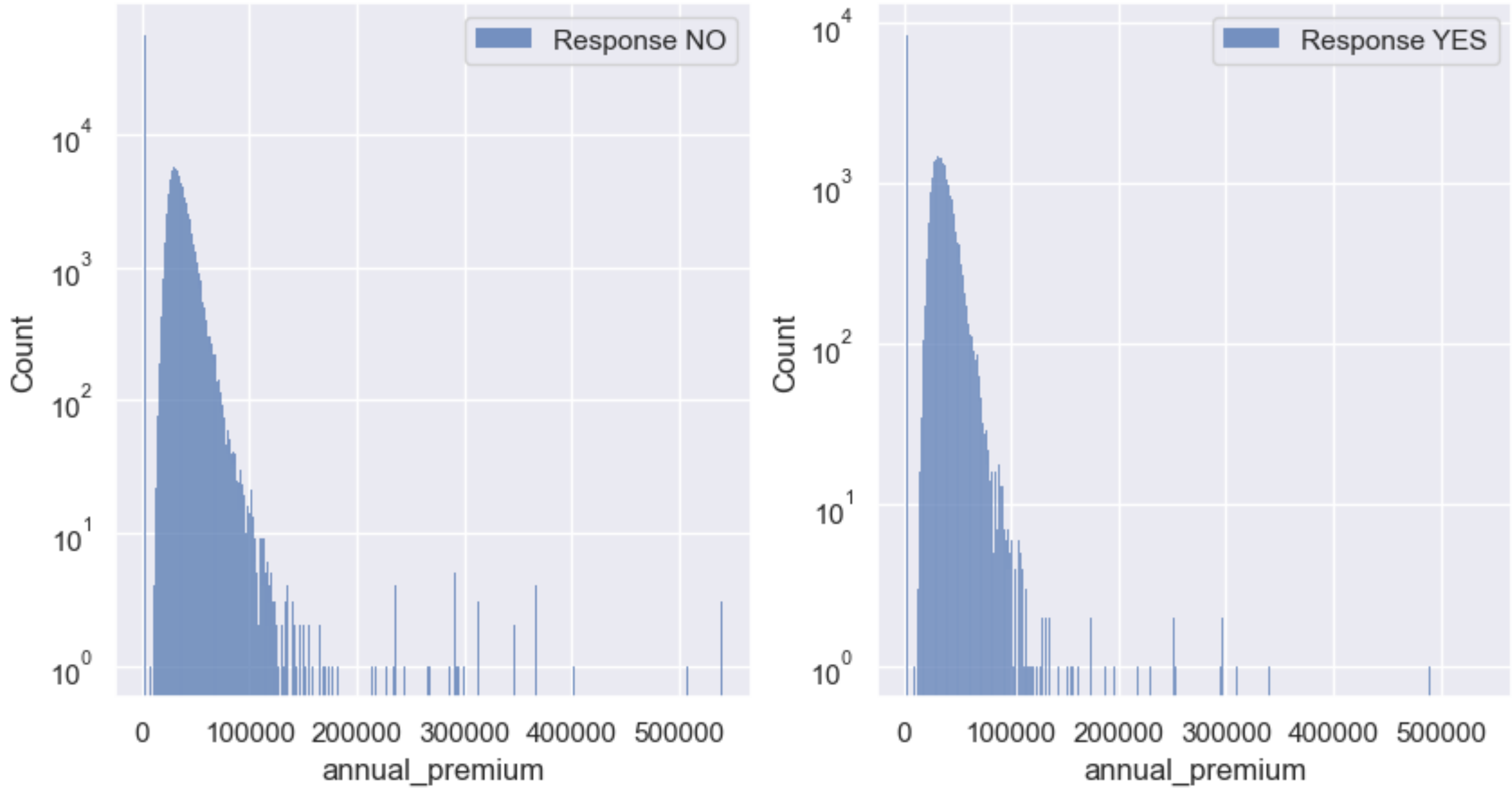
Analyzing the graphics below, we may see that the concentration of answers no and yes stays in a interval between 10k and 100k, and both have almost the same curve, with a peak at 20k.

I also divided that dataset in 2, Yes and No acceptance, as well, I create an annual premium interval between 10k and 80k, to get a closer view that allows us to see better the concentration in 30k annual premium, as in yes as in no.

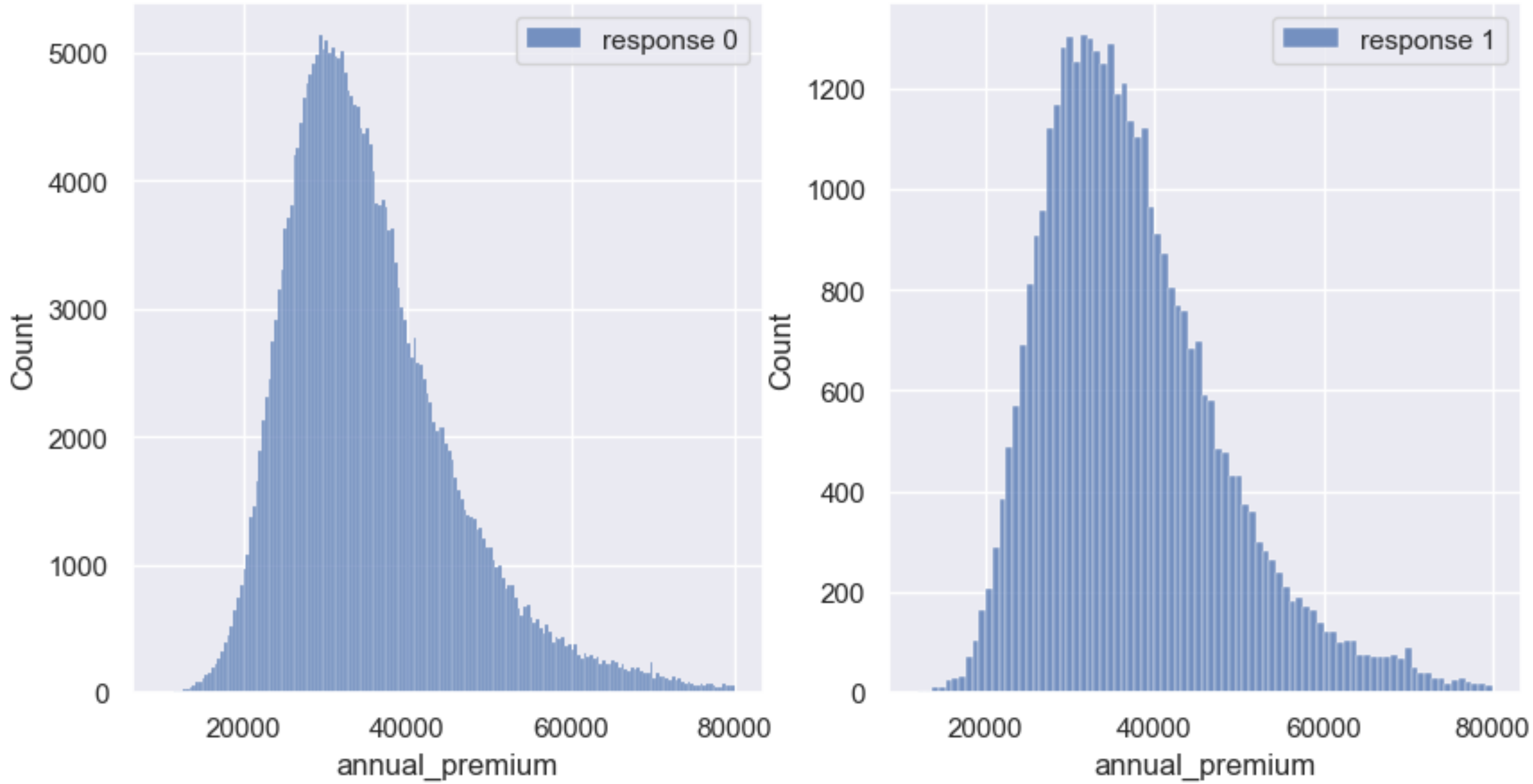
In [18]:
```
# Looking the behavior comparing the value and the answers

aux0 = df4[['annual_premium', 'response', 'id']]
sns.boxenplot( x='response', y='annual_premium', data=aux0 );
```



In [19]:
```
# Histplot will give a best overview of the concentration

# No Answer

plt.subplot( 1, 2, 1)

aux0 = df4.loc[df4['response'] == 0, 'annual_premium']
sns.histplot( aux0, label='Response NO' );
plt.yscale('log')
plt.legend();

# Yes Answer

plt.subplot( 1, 2, 2)

aux1 = df4.loc[df4['response'] == 1, 'annual_premium']
sns.histplot( aux1, label='Response YES' );
plt.yscale('log')
plt.legend();
```



In [20]:
```python
# Separate an interval of high concentration between 10k and 80k to have a better view

aux = df4[ ( df4['annual_premium'] > 10000) & ( df4['annual_premium'] < 80000 ) ]

# No Answer

plt.subplot( 1, 2, 1 )

aux00 = aux.loc[aux['response'] == 0, 'annual_premium']
sns.histplot( aux00, label='response 0' );
plt.legend();

# Yes Answer

plt.subplot( 1, 2, 2 )

aux00 = aux.loc[aux['response'] == 1, 'annual_premium']
sns.histplot( aux00, label='response 1' );
plt.legend();
```



Driving License (Check if having a driver license influence the insurance purchase)

As we can see below, the number of people that don't have driver license is too low just 812 among 380k, so that makes difficult to use it as tendency.

So mostly people that accept the insurance would have a driver license, to be more precisely 12%,

We could say 100% of all yes answers, considering that just 41 non-drivers accepted

In [21]:
```
# Checking the relation of licensed and non licensed drive

aux = df4[['driving_license', 'id']].groupby( 'driving_license').count().reset_index()
aux
```

Out[21]:

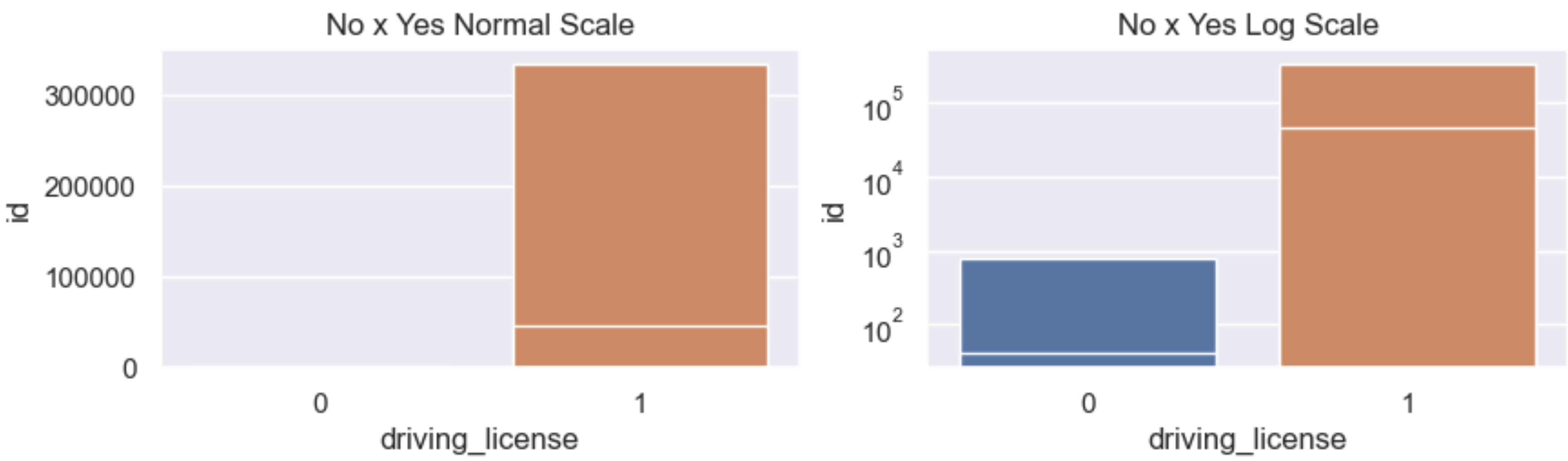
	driving_license	id
0	0	812
1	1	380297

In [22]:
```
# Creating a little dataset, grouping driving license and response to see the relation

aux = df4[['driving_license', 'response', 'id']].groupby( ['driving_license', 'response'] ).count().sort_values( 'id', ascending=False ).re
aux['percentage'] = aux['id'] / aux['id'].sum()
aux
```

Out[22]:

	driving_license	response	id	percentage
0	1	0	333628	0.875414
1	1	1	46669	0.122456
2	0	0	771	0.002023
3	0	1	41	0.000108

In [23]:
```
# Accept the Insurance

aux0 = aux[aux['response'] == 0]

plt.subplot( 2, 2, 1 )

sns.barplot( x='driving_license', y='id', data=aux0 );
plt.title( 'No x Yes Normal Scale' )

plt.subplot( 2, 2, 2 )

sns.barplot( x='driving_license', y='id', data=aux0 );
plt.yscale('log')

# Don't Accept the Insurance

aux0 = aux[aux['response'] == 1]

plt.subplot( 2, 2, 1 )

sns.barplot( x='driving_license', y='id', data=aux0 );

plt.subplot( 2, 2, 2 )

sns.barplot( x='driving_license', y='id', data=aux0 );
plt.yscale('log')
plt.title( 'No x Yes Log Scale' );
```



Region Code (Check where are the regions the insurance is most looked for)

We can see below that the most clients should live in region 28, the most concentration of clients, when we look at the percentage graphic, more than 40% of all acceptance comes from region 28, the other areas don't even reach 10% individually

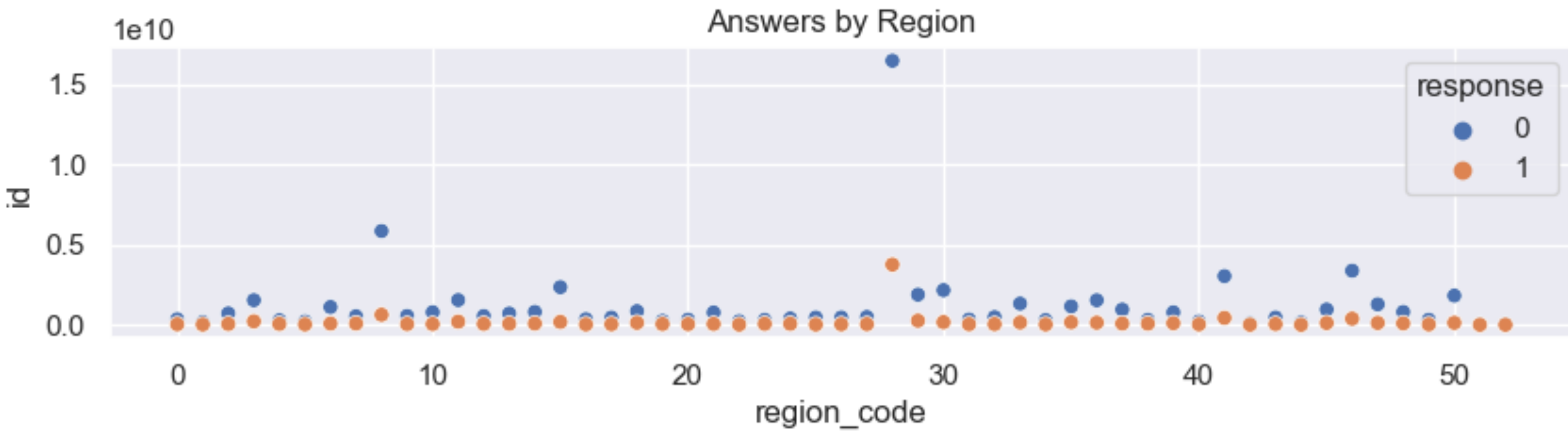
In [25]:
```
# Create a Top 10 Region Rank for positive answer

aux0 = df4.loc[df4['response'] == 1, ['id', 'region_code', 'response']].groupby( ['region_code', 'response'] ).count().sort_values( 'id', a
aux0['percentage'] = aux0['id'] / aux0['id'].sum()
aux0.head( 10 )
```

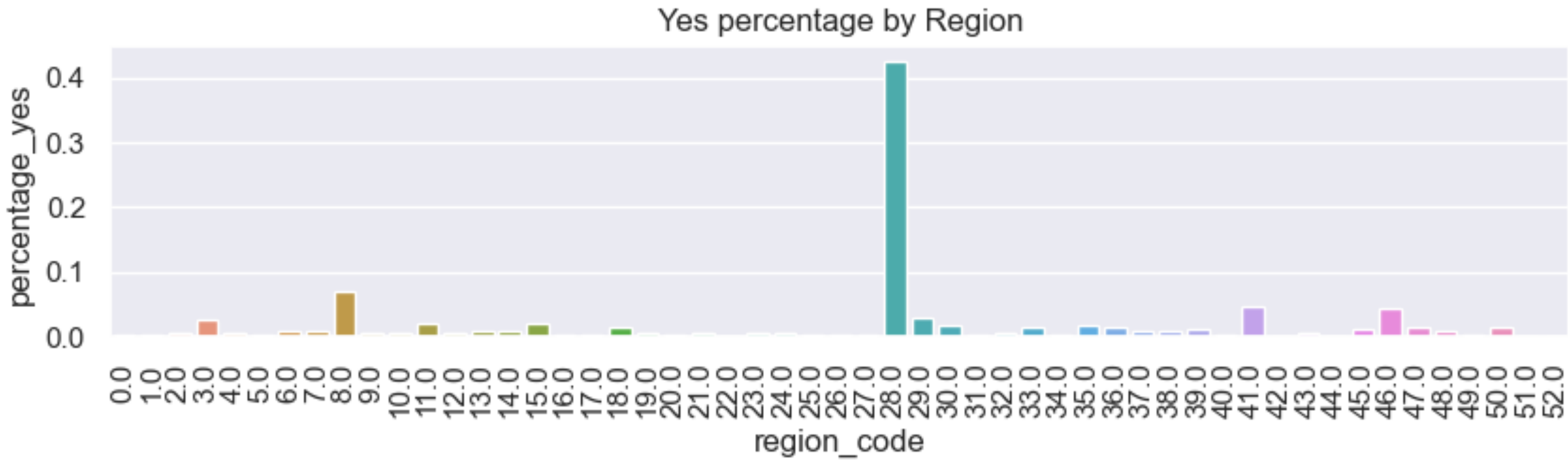
Out[25]:

	region_code	response	id	percentage
0	28.0	1	19917	0.426397
1	8.0	1	3257	0.069728
2	41.0	1	2224	0.047613
3	46.0	1	2032	0.043502
4	29.0	1	1365	0.029223
5	3.0	1	1181	0.025284
6	11.0	1	1041	0.022286
7	15.0	1	958	0.020510
8	30.0	1	900	0.019268
9	35.0	1	865	0.018519

In [26]:

```
# Grouping the clients by Region and Answer

plt.subplot( 2, 1, 1 )

aux0 = df4[['id', 'region_code', 'response']].groupby( ['region_code', 'response'] ).sum().reset_index()
sns.scatterplot( x='region_code', y='id', hue='response', data=aux0 );
plt.title( 'Answers by Region' );

# Percentage of Yes clients

aux0 = df4.loc[df4['response'] == 1, ['region_code', 'response']].groupby( 'region_code').sum().sort_values( 'response', ascending=False ).
aux0['percentage_yes'] = aux0['response'] / aux0['response'].sum()
aux0

plt.subplots_adjust( hspace=0.5 );

plt.subplot( 2, 1, 2 )

sns.barplot( x='region_code', y='percentage_yes', data=aux0 );
plt.xticks( rotation=90 );
plt.title( 'Yes percentage by Region' );
```





Previously Insured (See if the client that has already insured before, has more chances to accept a new insurance)

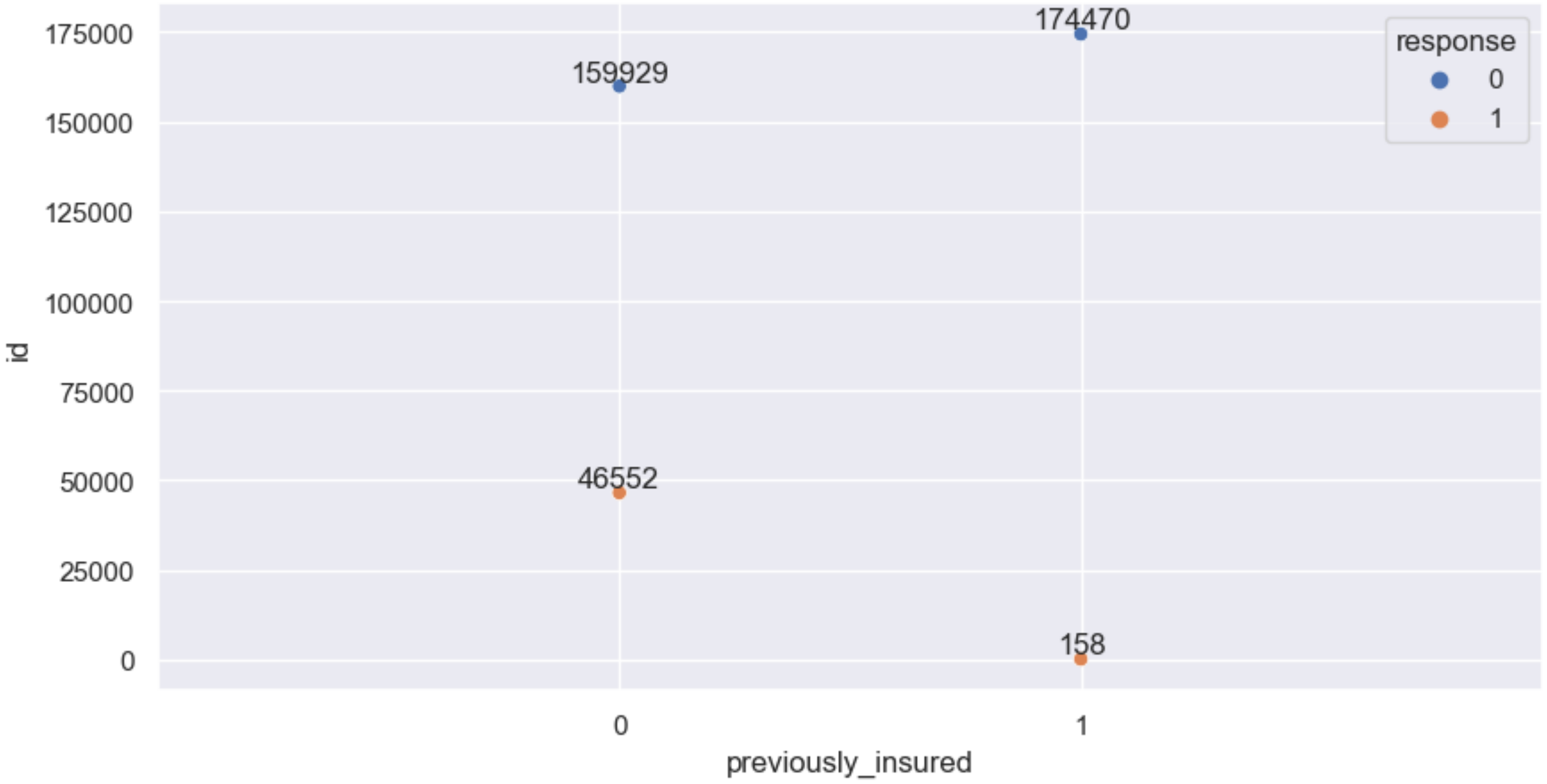
And, looking in the charts below we may see that there isn't a good news to the company, because the number of people that already had the insurance answered mostly no to another one.

A relation of 99% among those that had the insurance

While we had 22,5% of acceptance among those that didn't have a insurance

In [27]:

```python
# Previously Insured

aux0 = df4[['id', 'previously_insured', 'response']].groupby( ['previously_insured', 'response'] ).count().reset_index()
ax = sns.scatterplot( x='previously_insured', y='id', hue='response', data=aux0 );

plt.xticks( [0,1] );
plt.xlim( [-1.0, 2.0 ] );

# Manually annotate the points with their values.

for i, r in aux0.iterrows():
    ax.text( r['previously_insured'], r['id'], str( r['id'] ), ha='center', va='bottom' )
```



In [28]:

```python
# Getting columns Previsouly insured and response to see the percentage relation

pd.crosstab( df4['previously_insured'], df4['response'] ).apply( lambda x: x / x.sum(), axis=1 ).reset_index()
```

Out[28]:

response	previously_insured	0	1
0	0	0.774546	0.225454
1	1	0.999095	0.000905

In [29]:

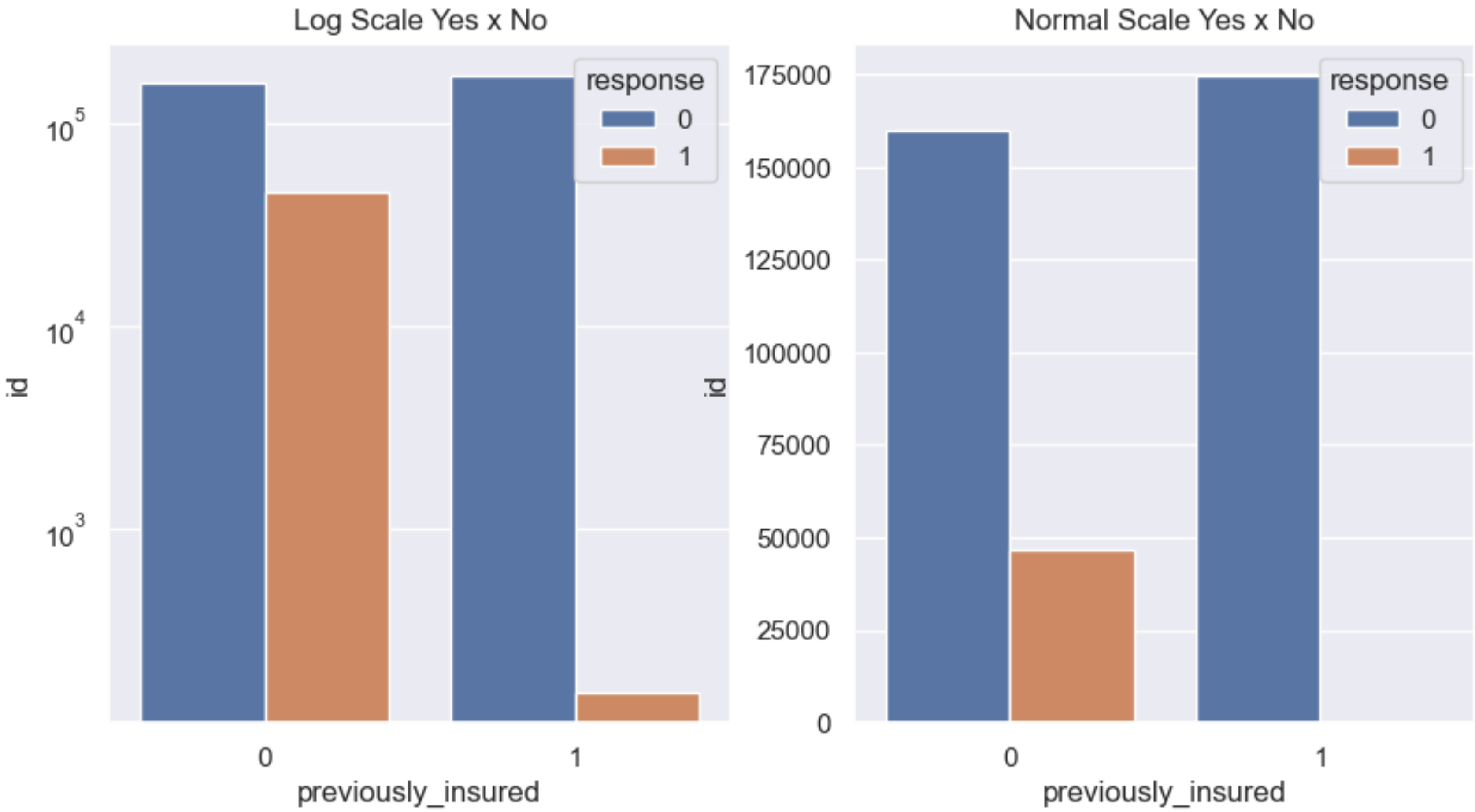
```
# Grouping previouosly insured and response, to create a sorted rank

aux0 = df4[['previously_insured', 'response', 'id']].groupby( ['previously_insured', 'response'] ).count().sort_values( 'id', ascending=Fal
aux0
```

Out[29]:

	previously_insured	response	id
0	1	0	174470
1	0	0	159929
2	0	1	46552
3	1	1	158

In [30]:

```
# Checking the relation with previously insured and response

plt.subplot( 1, 2, 1 )

sns.barplot( x='previously_insured', y='id', hue='response', data=aux0 );
plt.title( 'Log Scale Yes x No' );

# Log scale

plt.yscale('log')

plt.subplot( 1, 2, 2 )

sns.barplot( x='previously_insured', y='id', hue='response', data=aux0 );
plt.title( 'Normal Scale Yes x No' );
```



Vehicle Age (The vehicle age matters whether a client buys or not the insurance car?)

Well, looking below we see that the most acceptance comes from owners of between 1 to 2 years old car, with 9%, we just have few % regarding over 2 years and below 1 year cars both summing up 3% only

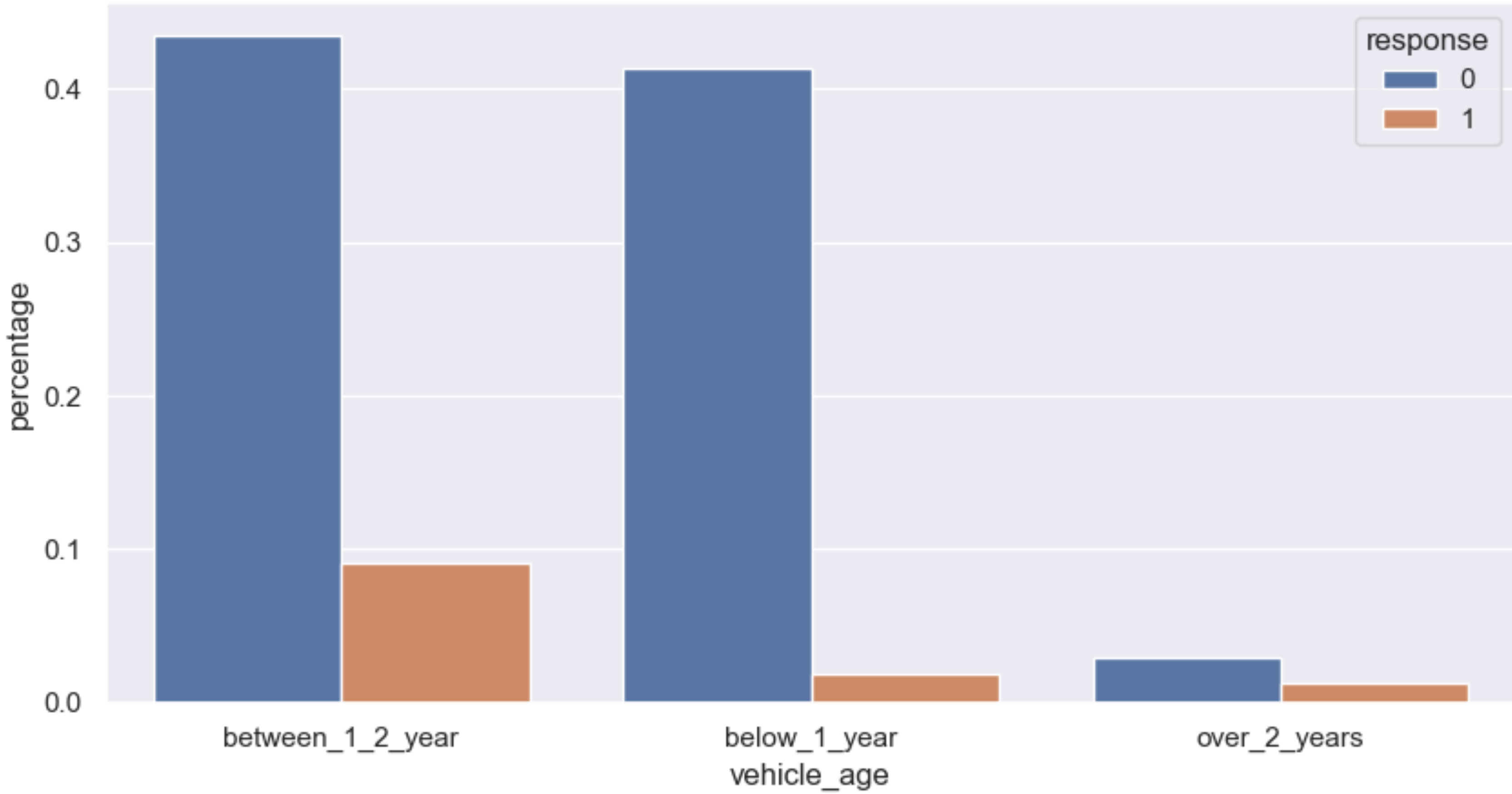
In [31]:
```
# Grouping the Vehicle age and the responses

aux0 = df4[['id', 'vehicle_age', 'response']].groupby( ['vehicle_age', 'response'] ).count().sort_values( 'id', ascending=False ).reset_ind
aux0['percentage'] = aux0['id'] / aux['id'].sum()
aux0
```

Out[31]:

	vehicle_age	response	id	percentage
0	between_1_2_year	0	165510	0.434285
1	below_1_year	0	157584	0.413488
2	between_1_2_year	1	34806	0.091328
3	over_2_years	0	11305	0.029663
4	below_1_year	1	7202	0.018897
5	over_2_years	1	4702	0.012338

In [32]:

```
# Bar chart by each age ratio showing the difference of yes and no individually

sns.barplot( x='vehicle_age', y='percentage', hue='response', data=aux0 );
```



Vehicle Damage (Analyze if the client, that had a vehicle damage, has more probability to buy the insurance)

As We can see the most response yes comes from those who already had any vehicle damage with 12% of the total. while less than half percentage that didn't have vehicle damage before said yes

In [33]:

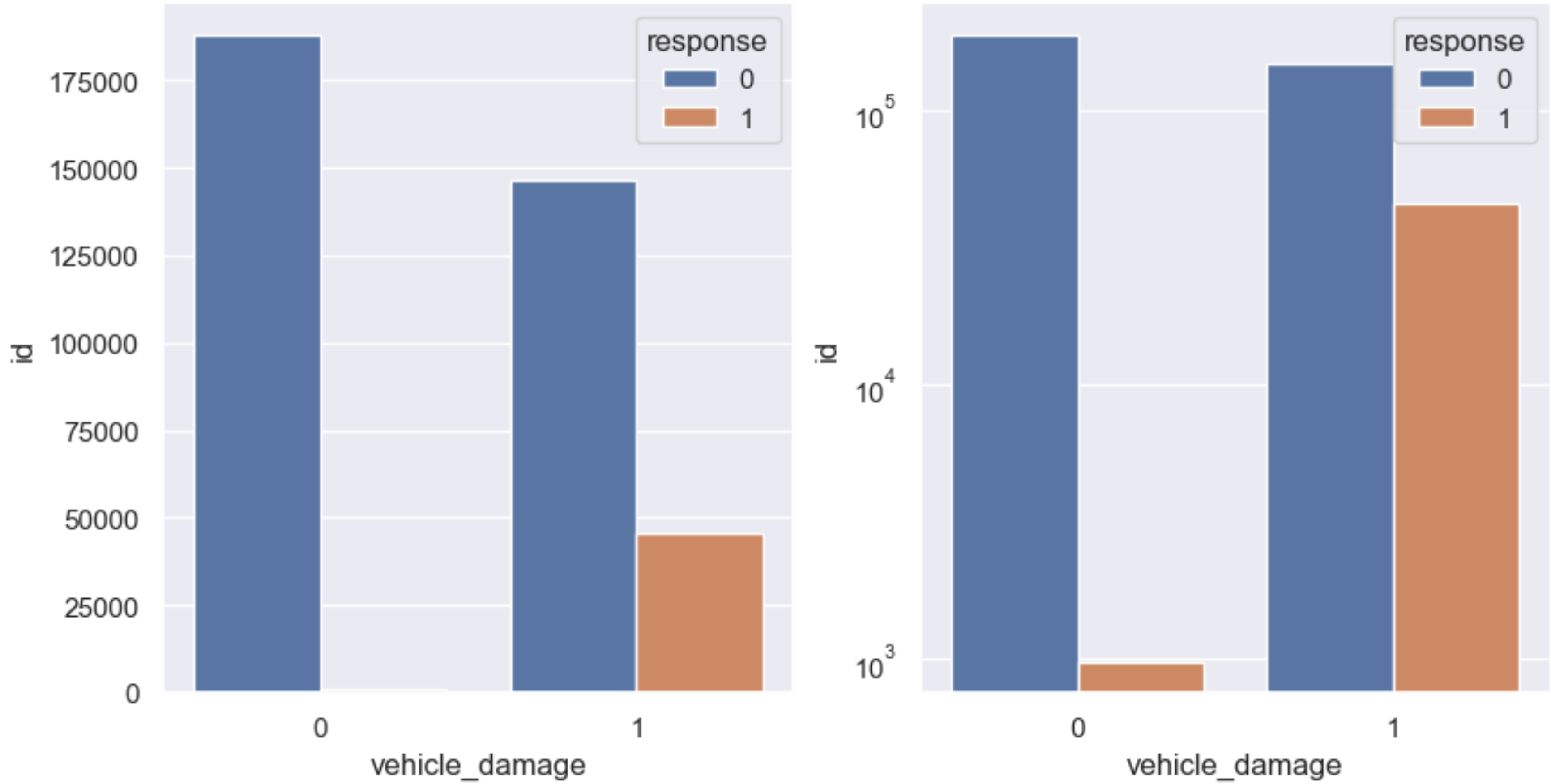
```
# Grouping vehicle demage and response to see a rank with percentage

aux0 = df4[['vehicle_damage', 'response', 'id']].groupby( ['vehicle_damage', 'response'] ).count().sort_values( 'id', ascending=False ).res
aux0['percentage'] = aux0['id'] / aux0['id'].sum()
aux0
```

Out[33]:

	vehicle_damage	response	id	percentage
0	0	0	187714	0.492547
1	1	0	146685	0.384890
2	1	1	45728	0.119987
3	0	1	982	0.002577

In [34]:

```
# Vehicle Damage vs. Response

plt.subplot( 1, 2, 1 )

sns.barplot( x='vehicle_damage', y='id', hue='response', data=aux0 );

# Vehicle Damage vs. Response ( Log Scale )

plt.subplot( 1, 2, 2 )

sns.barplot( x='vehicle_damage', y='id', hue='response', data=aux0 );
plt.yscale('log')
```



Policy Sales Channel (Check if the Channel used to communicate the client influence the response)

As we may see below, we have 155 sales channels, and just a few them have a consider number of contact.

In [35]:
```
# Count how many channels we have in the dataset

df4['policy_sales_channel'].nunique()
```

Out[35]:
```
155
```

In [36]:
```
# Channels with answer 0 ( No )

aux0 = df4[['policy_sales_channel', 'response']]
aux0 = aux0[aux0['response'] == 0].groupby( 'policy_sales_channel' ).count().sort_values( 'response', ascending=False ).reset_index()
aux0.rename(columns={'response': 'response_0'}, inplace=True)

# Channels with answer 1 ( Yes )

aux1 = df4[['policy_sales_channel', 'response']]
aux1 = aux1[aux1['response'] == 1].groupby( 'policy_sales_channel' ).count().sort_values( 'response', ascending=False ).reset_index()
aux1.rename(columns={'response': 'response_1'}, inplace=True)

# Merge both datasets on the 'policy_sales_channel' column

aux01 = pd.merge(aux1, aux0, on='policy_sales_channel', how='inner')
aux01['Acceptance_percentual'] = aux01['response_1'] / ( aux01['response_1'] + aux01['response_0'] )
aux01
```

Out[36]:

	policy_sales_channel	response_1	response_0	Acceptance_percentual
0	26.0	15891	63809	0.199385
1	124.0	13996	59999	0.189148
2	152.0	3858	130926	0.028624
3	156.0	2297	8364	0.215458
4	157.0	1794	4890	0.268402
5	122.0	1720	8210	0.173212
6	154.0	1474	4519	0.245954
7	163.0	880	2013	0.304183
8	160.0	475	21304	0.021810
9	155.0	395	839	0.320097
10	25.0	369	1479	0.199675
11	13.0	275	1590	0.147453
12	55.0	189	1075	0.149525
13	7.0	182	1416	0.113892
14	31.0	160	471	0.253566
15	3.0	159	364	0.304015
16	30.0	156	1254	0.110638
17	158.0	135	357	0.274390
18	12.0	132	651	0.168582
19	125.0	127	899	0.123782
20	8.0	125	1390	0.082508
21	151.0	122	3763	0.031403
22	52.0	115	940	0.109005
23	11.0	108	1095	0.089776
24	29.0	106	737	0.125741
25	4.0	102	407	0.200393
26	24.0	99	651	0.132000

	policy_sales_channel	response_1	response_0	Acceptance_percentual
27	15.0	78	810	0.087838
28	150.0	76	236	0.243590
29	120.0	65	704	0.084525
30	14.0	63	559	0.101286
31	23.0	58	364	0.137441
32	61.0	56	523	0.096718
33	60.0	53	464	0.102515
34	10.0	50	214	0.189394
35	16.0	45	478	0.086042
36	136.0	40	145	0.216216
37	153.0	36	571	0.059308
38	1.0	35	1039	0.032588
39	147.0	34	150	0.184783
40	91.0	29	129	0.183544
41	42.0	26	106	0.196970
42	59.0	25	102	0.196850
43	145.0	23	151	0.132184
44	109.0	21	154	0.120000
45	44.0	20	81	0.198020
46	121.0	19	45	0.296875
47	19.0	19	203	0.085586
48	22.0	18	314	0.054217
49	116.0	18	136	0.116883
50	36.0	17	35	0.326923
51	9.0	17	152	0.100592
52	54.0	16	84	0.160000
53	37.0	15	137	0.098684

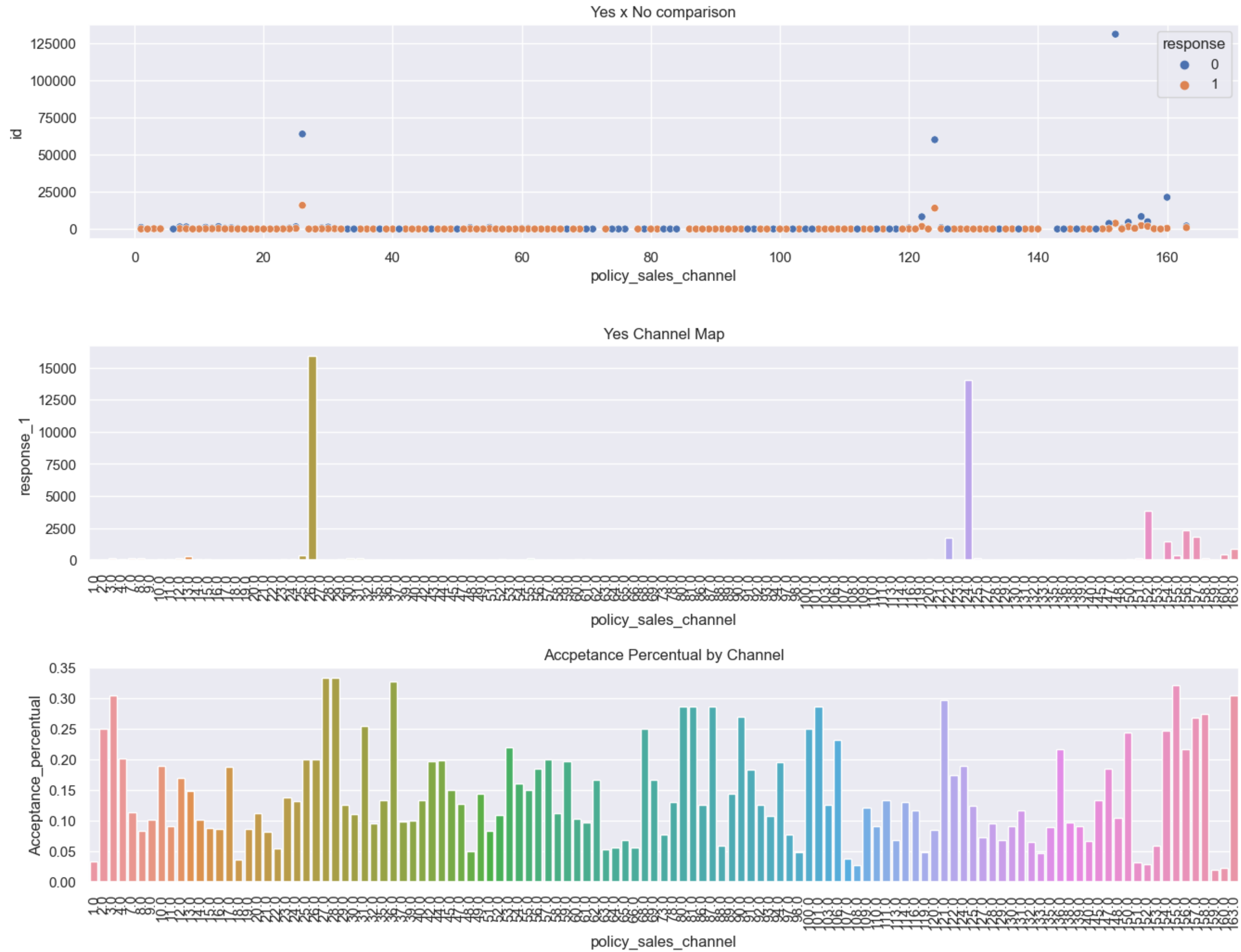
	policy_sales_channel	response_1	response_0	Acceptance_percentual
54	131.0	14	107	0.115702
55	128.0	13	124	0.094891
56	139.0	13	130	0.090909
57	106.0	12	40	0.230769
58	138.0	12	112	0.096774
59	21.0	12	136	0.081081
60	56.0	12	53	0.184615
61	35.0	10	65	0.133333
62	111.0	9	59	0.132353
63	135.0	9	92	0.089109
64	103.0	9	63	0.125000
65	94.0	9	37	0.195652
66	127.0	8	102	0.072727
67	148.0	8	69	0.103896
68	47.0	8	55	0.126984
69	90.0	7	19	0.269231
70	113.0	7	97	0.067308
71	45.0	7	40	0.148936
72	53.0	7	25	0.218750
73	140.0	7	100	0.065421
74	18.0	6	161	0.035928
75	86.0	6	42	0.125000
76	64.0	5	84	0.056180
77	119.0	5	98	0.048544
78	133.0	4	81	0.047059
79	132.0	4	58	0.064516
80	65.0	4	55	0.067797

	policy_sales_channel	response_1	response_0	Acceptance_percentual
81	81.0	4	10	0.285714
82	80.0	4	10	0.285714
83	78.0	3	20	0.130435
84	129.0	3	41	0.068182
85	17.0	3	13	0.187500
86	20.0	3	24	0.111111
87	93.0	3	25	0.107143
88	92.0	3	21	0.125000
89	114.0	3	20	0.130435
90	40.0	2	13	0.133333
91	107.0	2	52	0.037037
92	101.0	2	5	0.285714
93	87.0	2	5	0.285714
94	130.0	2	20	0.090909
95	32.0	2	19	0.095238
96	88.0	2	32	0.058824
97	89.0	2	12	0.142857
98	49.0	2	12	0.142857
99	100.0	2	6	0.250000
100	28.0	1	2	0.333333
101	159.0	1	50	0.019608
102	27.0	1	2	0.333333
103	108.0	1	37	0.026316
104	66.0	1	17	0.055556
105	51.0	1	11	0.083333
106	57.0	1	4	0.200000
107	58.0	1	8	0.111111

	policy_sales_channel	response_1	response_0	Acceptance_percentual
108	62.0	1	5	0.166667
109	63.0	1	18	0.052632
110	68.0	1	3	0.250000
111	39.0	1	9	0.100000
112	69.0	1	5	0.166667
113	2.0	1	3	0.250000
114	97.0	1	12	0.076923
115	98.0	1	20	0.047619
116	48.0	1	19	0.050000
117	110.0	1	10	0.090909
118	73.0	1	12	0.076923

In [37]:

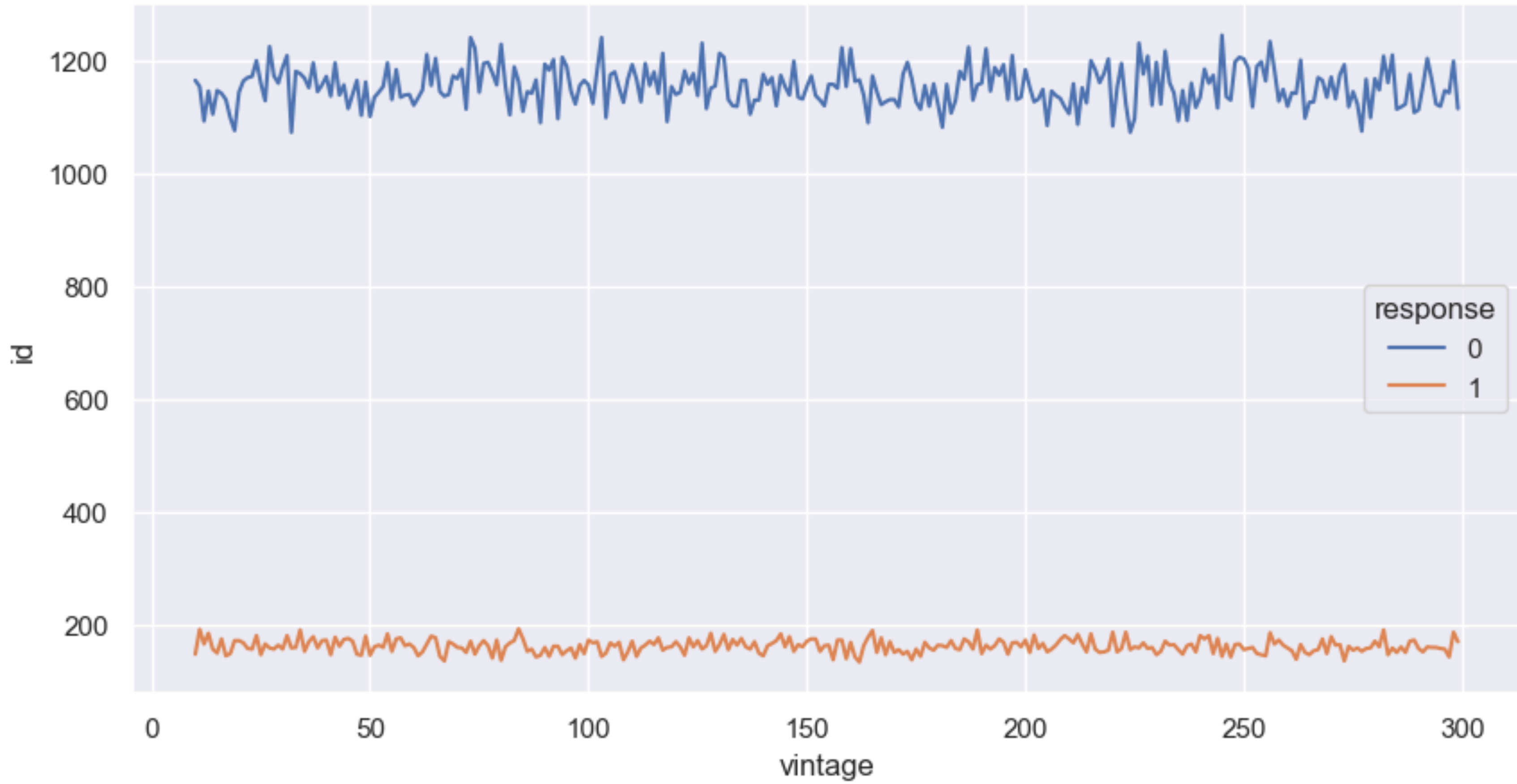
```
# Separating aux with the columns to be analyzed

aux00 = df4[['policy_sales_channel', 'response', 'id']].groupby( ['policy_sales_channel', 'response'] ).count().sort_values( 'id', ascendin

plt.figure(figsize=(16, 12))
plt.subplot( 3, 1, 1 )

# Creating a Scatterplot to compare the yes an no amount in each channel

sns.scatterplot( x='policy_sales_channel', y='id', hue='response', data=aux00 );
plt.title( 'Yes x No comparison' );

plt.subplot( 3, 1, 2 )

plt.subplots_adjust( hspace=0.5 );

# Creating a barplot with just positive answers to compare which channels has more acceptance

sns.barplot( x='policy_sales_channel', y='response_1', data=aux1 );
plt.title( 'Yes Channel Map' );
plt.xticks( rotation=90 );

# Create a barplot to indicate the best percentual acceptance

plt.subplots_adjust( hspace=0.5 );

plt.subplot( 3, 1, 3 )

sns.barplot( x='policy_sales_channel', y='Acceptance_percentual', data=aux01 );
plt.title( 'Accpetance Percentual by Channel' )
plt.xticks( rotation=90 );
```


Yes x No comparison
id
policy_sales_channel
response
0
1
Yes Channel Map
response_1
policy_sales_channel
Accpetance Percentual by Channel
Acceptance_percentual
policy_sales_channel

Vintage (Check if a longer time Client has more chances to buy another insurance)

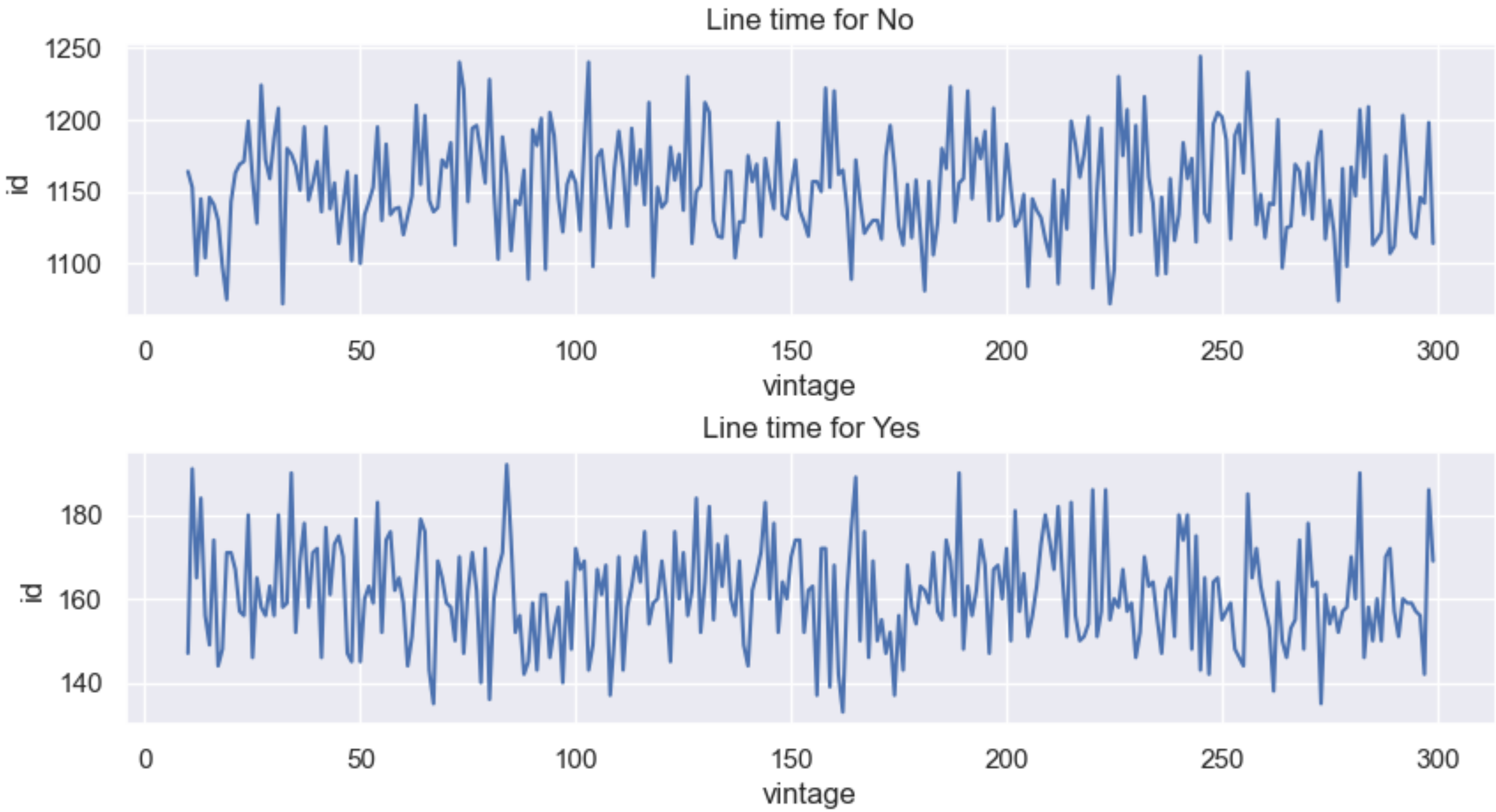
Looking into the line time graphics, it is not possible to say that the amount of time a person has been a client influences whether the person will accept a new insurance or not, the variation is huge during the all period

In [38]:
```
# Grouping vintage and the responses to create a linetime graphic to see the variation throughout time

aux0 = df4[['vintage', 'response', 'id']].groupby( ['vintage', 'response'] ).count().reset_index()
sns.lineplot( x='vintage', y='id', hue='response', data=aux0 );
```



In [39]:

```python
# Linetime for no answers

plt.subplot( 2, 1, 1 )

aux1 = aux0[aux0['response'] == 0 ]
aux1 = aux1.groupby( ['vintage', 'response'] ).sum().reset_index()
sns.lineplot( x='vintage', y='id', data=aux1 );
plt.title( 'Line time for No' );

# Linetime for yes answers

plt.subplots_adjust( hspace=0.5 );
plt.subplot( 2, 1, 2 )

aux1 = aux0[aux0['response'] == 1 ]
aux1 = aux1.groupby( ['vintage', 'response'] ).sum().reset_index()
sns.lineplot( x='vintage', y='id', data=aux1 );
plt.title( 'Line time for Yes' );
```



Data Preparation

In [95]:
```
# Prepare the feature set ('x') and target variable ('y') for machine learning tasks

x = df4.drop( 'response', axis=1 )
y = df4['response'].copy()

# Split the data into training and validation sets for model training and evaluation

x_train, x_validation, y_train, y_validation = ms.train_test_split( x, y, test_size=0.20 )

# Combine the training features and target values into a single DataFrame ('df5') for convenience during further analysis or model training.

df5 = pd.concat( [x_train, y_train], axis=1 )
```

Standarlization

In [41]:
```
# Create a StandardScaler instance 'ss' from scikit-learn's preprocessing module.

ss = pp.StandardScaler()

# Standardize the 'annual_premium' feature in the 'df5' DataFrame using the StandardScaler 'ss'.

df5['annual_premium'] = ss.fit_transform( df5[['annual_premium']].values )

# Save the trained scaler to a file using 'pickle', which can be useful for future data preprocessing.

# pickle.dump( ss, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\pickle\\annual_premiu
```

Rescaling

In [42]:
```
# Create a MinMaxScaler instance for 'age' and 'vintage' features

mms_age = pp.MinMaxScaler()
mms_vintage = pp.MinMaxScaler()

# age Min Max Scaling This line scales the 'age' feature using Min-Max scaling. It transforms the values of 'age' to fall within the range [

df5['age'] = mms_age.fit_transform( df5[['age']].values )

# Vintage Min Max Scaling Similar to 'age', this line scales the 'vintage' feature using Min-Max scaling, transforming its values to the [0,

df5['vintage'] = mms_vintage.fit_transform( df5[['vintage']].values )

# pickle.dump( mms_age, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\pickle\\age_scal
# pickle.dump( mms_vintage, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\pickle\\vint
```

Transformation

Encoding

In [43]:
```
# Calculate the mean response for each gender category - One Hot Encoding / Target Encoding

target_encode_gender = df5.groupby( 'gender' )['response'].mean()

# Map the calculated mean response values back to the 'gender' column

df5.loc[:, 'gender'] = df5['gender'].map( target_encode_gender )

# Calculate the mean response for each region code category - Target Encoding / Frequency Encoding

target_encode_region_code = df5.groupby( 'region_code' )['response'].mean()

# Map the calculated mean response values back to the 'region_code' column

df5.loc[:, 'region_code'] = df5['region_code'].map( target_encode_region_code )

# One-hot encode the 'vehicle_age'. It creates binary columns for each unique value in the 'vehicle_age' column, and the new columns are pre

df5 = pd.get_dummies( df5, prefix='vehicle_age', columns=['vehicle_age'] )

# Calculate the frequency of each policy sales channel and normalize it by the total count

fe_policy_sales_channel = df5.groupby( 'policy_sales_channel' ).size() / len( df5 )

# Map the calculated frequency values back to the 'policy_sales_channel' column

df5.loc[:, 'policy_sales_channel'] = df5['policy_sales_channel'].map( fe_policy_sales_channel )

# pickle.dump( fe_policy_sales_channel, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\
# pickle.dump( target_encode_region_code, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell
# pickle.dump( target_encode_gender, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\pic
```

```
C:\Users\gabre\AppData\Local\Temp\ipykernel_11136\571412953.py:7: FutureWarning: In a future version, `df.iloc[:, i] = newvals` will attemp
t to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or,
if columns are non-unique, `df.isetitem(i, newvals)`
  df5.loc[:, 'gender'] = df5['gender'].map( target_encode_gender )
```

In [44]:
```
df5.head()
```

Out[44]:

	id	gender	age	driving_license	region_code	previously_insured	vehicle_damage	annual_premium	policy_sales_channel	vintage	response	vehi
119290	119291	0.103762	0.769231	1	0.187580	1	0	-1.624562	0.001138	0.993080	0	
283769	283770	0.103762	0.061538	1	0.122428	1	0	-0.000503	0.353383	0.889273	0	
363171	363172	0.138885	0.415385	1	0.187580	0	1	0.741654	0.209084	0.577855	0	
29128	29129	0.103762	0.246154	1	0.080128	0	1	0.469495	0.001660	0.851211	1	
201924	201925	0.103762	0.030769	1	0.090601	1	0	-1.624562	0.056847	0.608997	0	

In [45]:

```
x_validation.head()
```

Out[45]:

	id	gender	age	driving_license	region_code	previously_insured	vehicle_age	vehicle_damage	annual_premium	policy_sales_channel	vintage
168257	168258	Male	55	1	28.0	0	between_1_2_year	0	29112.0	156.0	261
116644	116645	Female	35	1	28.0	0	between_1_2_year	1	38153.0	26.0	18
59564	59565	Female	66	1	28.0	0	between_1_2_year	1	2630.0	156.0	128
218923	218924	Male	20	1	9.0	1	below_1_year	0	2630.0	160.0	201
377906	377907	Male	42	1	28.0	0	between_1_2_year	1	33838.0	124.0	132

Validation Preparation

In [46]:
```
# Map gender using target encoding

x_validation.loc[:, 'gender'] = x_validation.loc[:, 'gender'].map( target_encode_gender)

# Scale age using Min-Max scaling

x_validation.loc[:, 'age'] = mms_age.transform( x_validation[['age']].values )

# Map region code using target encoding

x_validation.loc[:, 'region_code'] = x_validation.loc[:, 'region_code'].map( target_encode_region_code )

# Find columns that start with 'vehicle_age'

vehicle_age_cols = x_validation.filter(like='vehicle_age')

# Apply one-hot encoding to vehicle age columns

x_validation = pd.get_dummies(x_validation, columns=vehicle_age_cols.columns)

# Standardize annual premium using StandardScaler

x_validation.loc[:, 'annual_premium'] = ss.transform( x_validation[['annual_premium']].values )

# Map policy sales channel using frequency encoding

x_validation.loc[:, 'policy_sales_channel'] = x_validation['policy_sales_channel'].map( fe_policy_sales_channel )

# Scale vintage using Min-Max scaling

x_validation.loc[:, 'vintage'] = mms_vintage.transform( x_validation[['vintage']].values )

# Fill missing values with 0

x_validation = x_validation.fillna( 0 )
```

```
C:\Users\gabre\AppData\Local\Temp\ipykernel_11136\2740249866.py:3: FutureWarning: In a future version, `df.iloc[:, i] = newvals` will attem
pt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` o
r, if columns are non-unique, `df.isetitem(i, newvals)`
  x_validation.loc[:, 'gender'] = x_validation.loc[:, 'gender'].map( target_encode_gender)
```

In [47]:
```
x_validation.head()
```

Out[47]:

	id	gender	age	driving_license	region_code	previously_insured	vehicle_damage	annual_premium	policy_sales_channel	vintage	vehicle_age_be
168257	168258	0.138885	0.538462	1	0.187580	0	0	-0.084536	0.027869	0.868512	
116644	116645	0.103762	0.230769	1	0.187580	0	1	0.441232	0.209084	0.027682	
59564	59565	0.103762	0.707692	1	0.187580	0	1	-1.624562	0.027869	0.408304	
218923	218924	0.138885	0.000000	1	0.080128	1	0	-1.624562	0.056847	0.660900	
377906	377907	0.138885	0.338462	1	0.187580	0	1	0.190299	0.194685	0.422145	

Feature Selection

Boruta Algorithm

In [48]:

```
# Prepare the training data: x_train_n contains the features, and y_train_n contains the target values

x_train_n = df5.drop( ['id', 'response'], axis=1 ).values
y_train_n = y_train.values.ravel()

# Define the Machine Learning Model

et = en.ExtraTreesClassifier( n_jobs=-1 )

# Define Boruta Feature Selection

boruta = bt.BorutaPy( et, n_estimators='auto', verbose=2, random_state=42 ).fit( x_train_n, y_train_n )
```

```
Iteration: 	1 / 100
Confirmed: 	0
Tentative: 	12
Rejected: 	0
Iteration: 	2 / 100
Confirmed: 	0
Tentative: 	12
Rejected: 	0
Iteration: 	3 / 100
Confirmed: 	0
Tentative: 	12
Rejected: 	0
Iteration: 	4 / 100
Confirmed: 	0
Tentative: 	12
Rejected: 	0
Iteration: 	5 / 100
Confirmed: 	0
Tentative: 	12
Rejected: 	0
Iteration: 	6 / 100
Confirmed: 	0
Tentative: 	12
Rejected: 	0
Iteration: 	7 / 100
Confirmed: 	0
Tentative: 	12
Rejected: 	0
Iteration: 	8 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	9 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	10 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	11 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	12 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
```

```
Iteration: 	13 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	14 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	15 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	16 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	17 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	18 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	19 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	20 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	21 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	22 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	23 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	24 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
```

```
Iteration:      25 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      26 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      27 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      28 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      29 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      30 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      31 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      32 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      33 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      34 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      35 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
Iteration:      36 / 100
Confirmed:      1
Tentative:      1
Rejected:       10
```

```
Iteration: 	37 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	38 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	39 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	40 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	41 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	42 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	43 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	44 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	45 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	46 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	47 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	48 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
```

```
Iteration: 	49 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	50 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	51 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	52 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	53 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	54 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	55 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	56 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	57 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	58 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	59 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	60 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
```

```
Iteration: 	61 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	62 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	63 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	64 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	65 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	66 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	67 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	68 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	69 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	70 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	71 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	72 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
```

```
Iteration: 	73 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	74 / 100
Confirmed: 	1
Tentative: 	1
Rejected: 	10
Iteration: 	75 / 100
Confirmed: 	1
Tentative: 	0
Rejected: 	11


BorutaPy finished running.

Iteration: 	76 / 100
Confirmed: 	1
Tentative: 	0
Rejected: 	11
```

In [49]:

```python
# Extract the Boolean list of selected features from Boruta and convert it to a Python list

cols_selected = boruta.support_.tolist()

# Extract the names of the best-selected features

x_train_fs = df5.drop( ['id', 'response'], axis=1 )
cols_selected_boruta = x_train_fs.iloc[:, cols_selected].columns.tolist()

# Find the features that were not selected by Boruta

cols_not_selected_boruta = list( np.setdiff1d( x_train_fs.columns, cols_selected_boruta ) )

# Print the names of features not selected by Boruta, selected by Boruta, and Boruta's selection status

print( cols_not_selected_boruta )
print( cols_selected_boruta )
print( cols_selected )
```

```
['annual_premium', 'driving_license', 'gender', 'policy_sales_channel', 'previously_insured', 'region_code', 'vehicle_age_below_1_year', 'vehicle_age_between_1_2_year', 'vehicle_age_over_2_years', 'vehicle_damage', 'vintage']
['age']
[False, True, False, False, False, False, False, False, False, False, False, False]
```

Boruta couldn't bring a satisfied result, giving us just one important column (feature), so we need to use another method.

Feature Importance

In [50]:
```
# Model Definition

forest = en.ExtraTreesClassifier( n_estimators=250, random_state=0, n_jobs=-1 )

# Data Preparation

x_train_n = df5.drop( ['id', 'response'], axis=1 )
y_train_n = y_train.values
forest.fit( x_train_n, y_train_n )
```

Out[50]:
```
ExtraTreesClassifier

ExtraTreesClassifier(n_estimators=250, n_jobs=-1, random_state=0)
```

In [51]:

```python
# Feature Importance Analysis

# Get feature importances from the trained ExtraTreesClassifier

importances = forest.feature_importances_

# Calculate the standard deviation of feature importances

std = np.std( [tree.feature_importances_ for tree in forest.estimators_], axis=0 )

# Sort feature importances in descending order and get the indices

indices = np.argsort( importances)[::-1]

# Print the Feature Ranking

print( 'Feature Ranking' )

# Create an empty DataFrame to store feature names and their importances

df = pd.DataFrame()

# Iterate through feature names and their importances

for i, j in zip( x_train_n, forest.feature_importances_ ):
    aux = pd.DataFrame( {'feature': i, 'importance': j}, index=[0] )
    df = pd.concat( [df, aux], axis=0 )

# Print the DataFrame sorted by importance in descending order

print( df.sort_values( 'importance', ascending=False ))

# Plot the impurity-based feature importances of the forest

plt.figure()
plt.title( 'Feature Importances' )
plt.bar( range( x_train_n.shape[1] ), importances[indices], color='r', yerr=std[indices], align='center' )
plt.xticks( range( x_train_n.shape[1] ), indices )
plt.xlim( [-1, x_train_n.shape[1]] )
plt.show()
```

```
Feature Ranking
                        feature  importance
0                       vintage    0.272453
0                annual_premium    0.244334
0                           age    0.166886
0                   region_code    0.103617
0                vehicle_damage    0.068228
0          policy_sales_channel    0.060241
0            previously_insured    0.055887
0      vehicle_age_below_1_year    0.014287
0  vehicle_age_between_1_2_year    0.006241
0                        gender    0.005146
0      vehicle_age_over_2_years    0.002169
0               driving_license    0.000510
```

Machine Learning Modeling

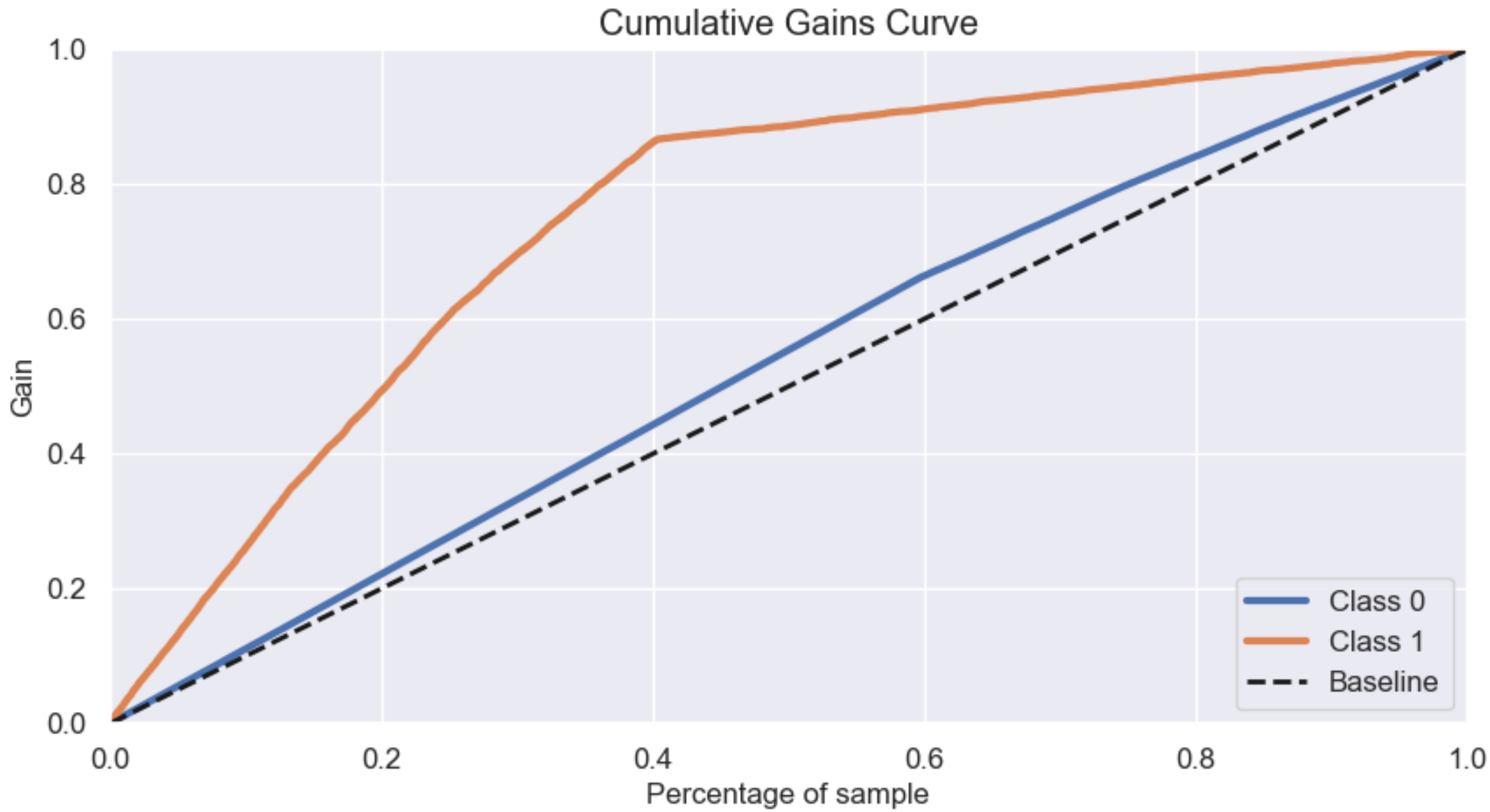
In [52]:
```
# Selecting the best ranked features

cols_selected = ['vintage', 'annual_premium', 'age', 'region_code', 'vehicle_damage', 'policy_sales_channel', 'previously_insured']
```

In [53]:
```
# Selecting Columns for Training and Validation Data

x_train = df5[ cols_selected ]

# Create a validation dataset (x_val) with the same selected feature columns.

x_val = x_validation[ cols_selected ]

# Extract the corresponding target values (y_val) for the validation dataset.

y_val = y_validation
```

KNN Classifier

In [54]:
```
# Model Definition

knn_model = nh.KNeighborsClassifier( n_neighbors=7 )

# Model Training

knn_model.fit( x_train, y_train )

# Model Prediction

yhat_knn = knn_model.predict_proba( x_val )
```

In [55]:
```
# Accumulative Gain

skplt.metrics.plot_cumulative_gain( y_val, yhat_knn );
```



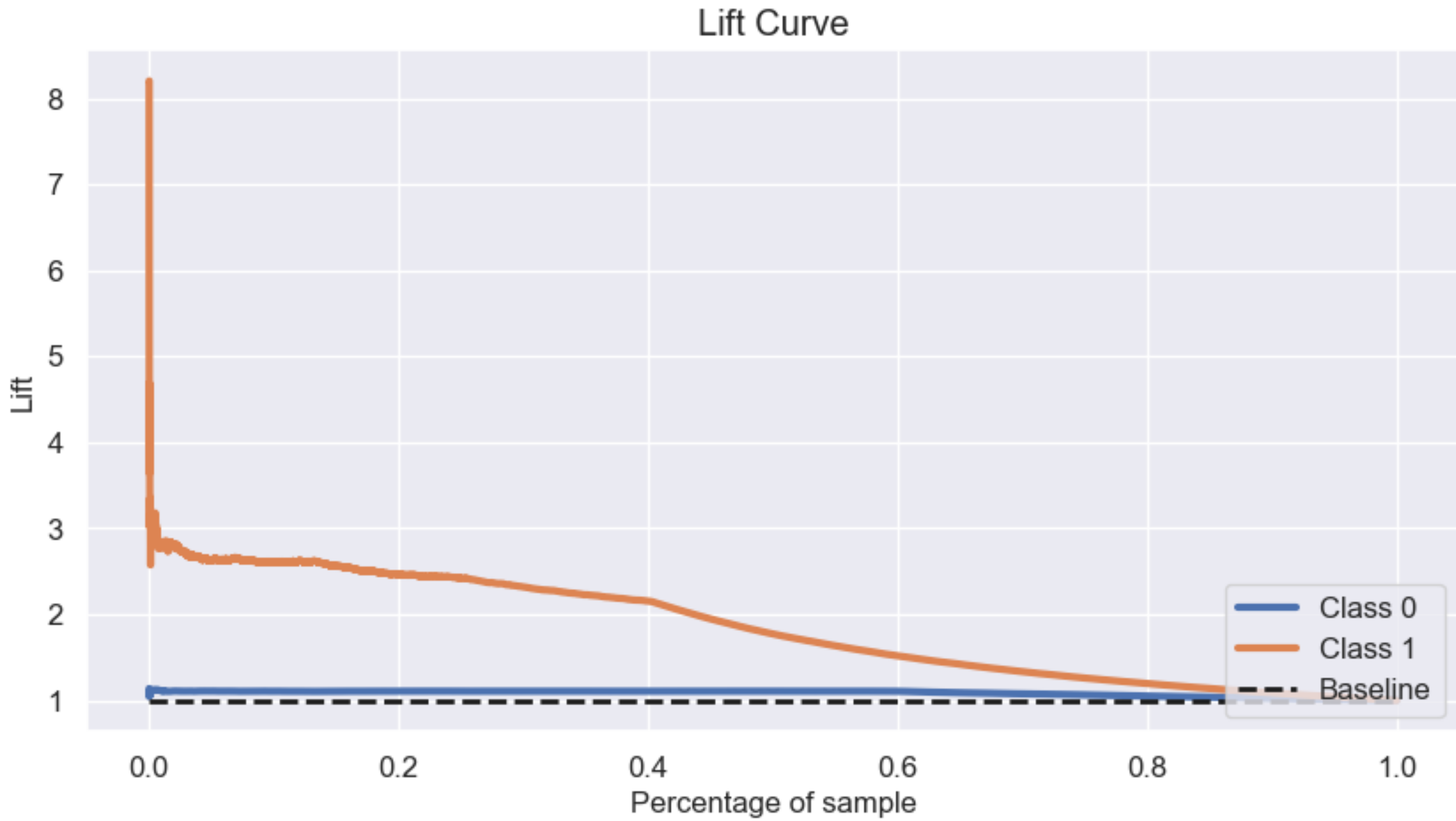
Important Insights:

In a Cumulative Gain chart, a steeper curve indicates that the model is doing a better job of ranking positive instances higher. This means that positive instances are being identified earlier in the ranked list.

The ideal scenario is when the Cumulative Gain curve starts at 0% (since you start with no positive instances captured) and rises steeply to 100% (indicating that all positive instances have been captured before a significant number of negative instances).

Cumulative Gain chart helps you assess the effectiveness of your model in prioritizing positive cases, which can be crucial in scenarios where the cost or impact of false negatives (missing positive cases) is high. It's a tool for evaluating classification models and understanding their performance in terms of positive instance ranking.

In [56]:
```
# Scikitplot library is there to help

skplt.metrics.plot_lift_curve( y_val, yhat_knn );
```



Lift Curve:

The Lift Curve helps you understand how well a classification model, such as a k-nearest neighbors (k-NN) classifier in this case, performs in comparison to a random or baseline model. It is especially useful in scenarios where you are interested in targeting a specific class, like potential customers who are likely to respond to a marketing campaign.

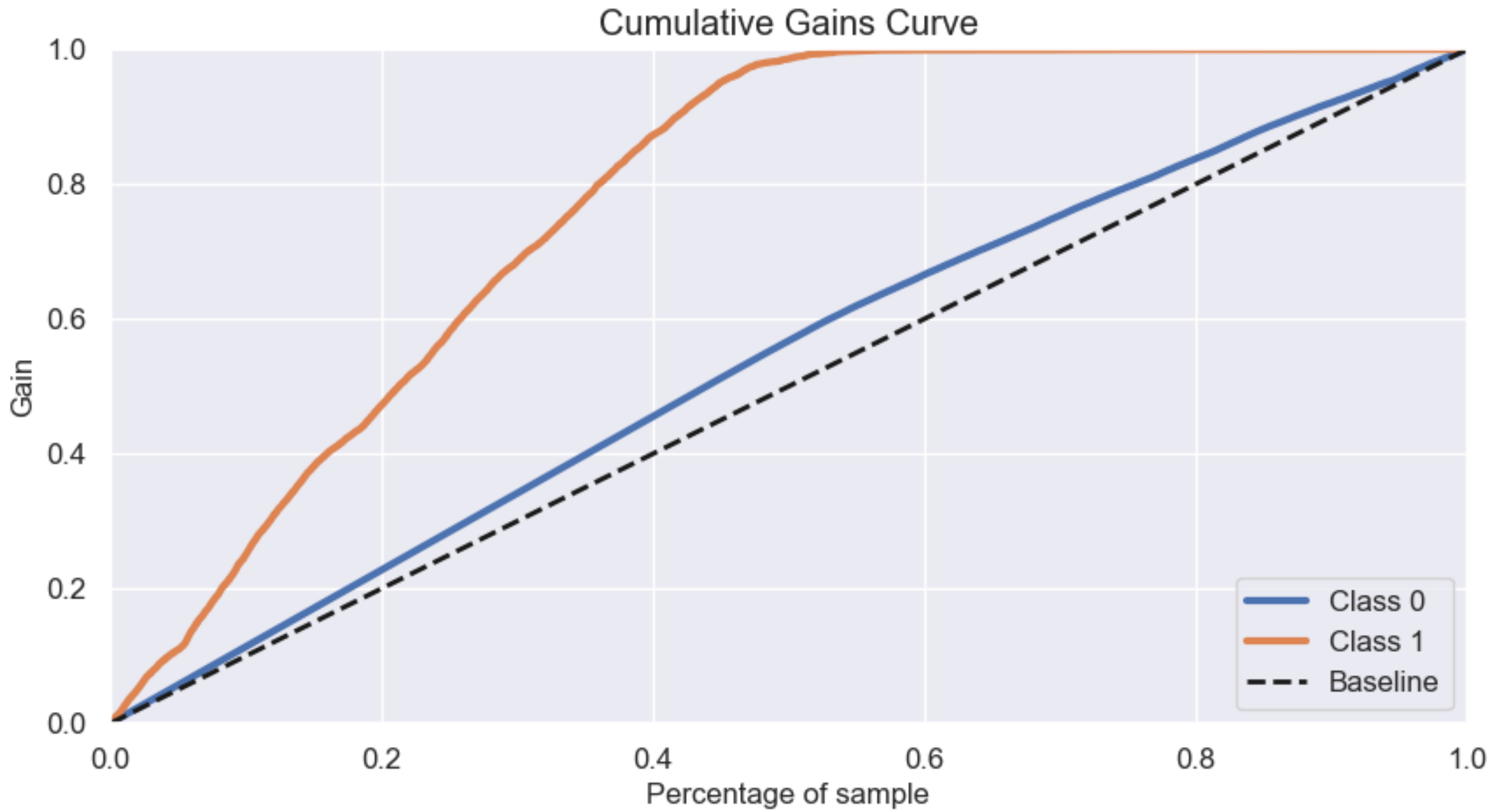
In a Lift Curve, a lift value greater than 1 indicates that the model is better at identifying the target class than random chance. The ideal scenario is when the lift curve starts at a lift value of 1 (indicating performance similar to random) and rises higher as you move along the x-axis. Higher lift values indicate that the model is more effective at identifying the target class.

Lift Curve provides insights into how much better your model is at identifying the target class compared to a random model. It helps you assess the effectiveness of your classification model, especially when you have a specific class of interest, such as positive responses to a marketing campaign.

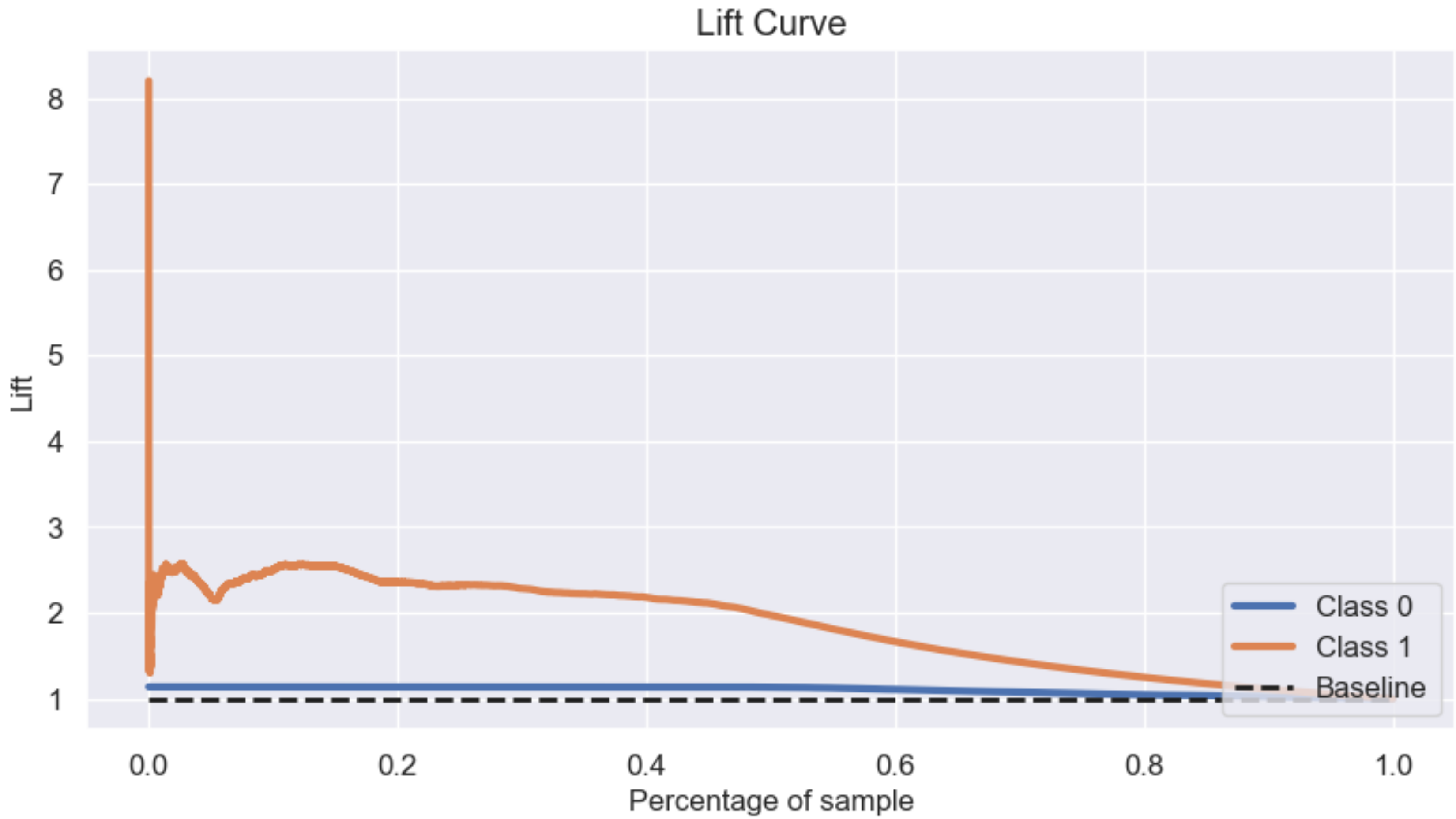
Logistic Regression

In [57]:
```
# Model Definition

lr_model = lm.LogisticRegression( random_state=42 )

# Model Training

lr_model.fit( x_train, y_train )

# Model Prediction

yhat_lr = lr_model.predict_proba( x_val )
```

In [58]:
```
# Accumulative Gain

skplt.metrics.plot_cumulative_gain( y_val, yhat_lr );
```



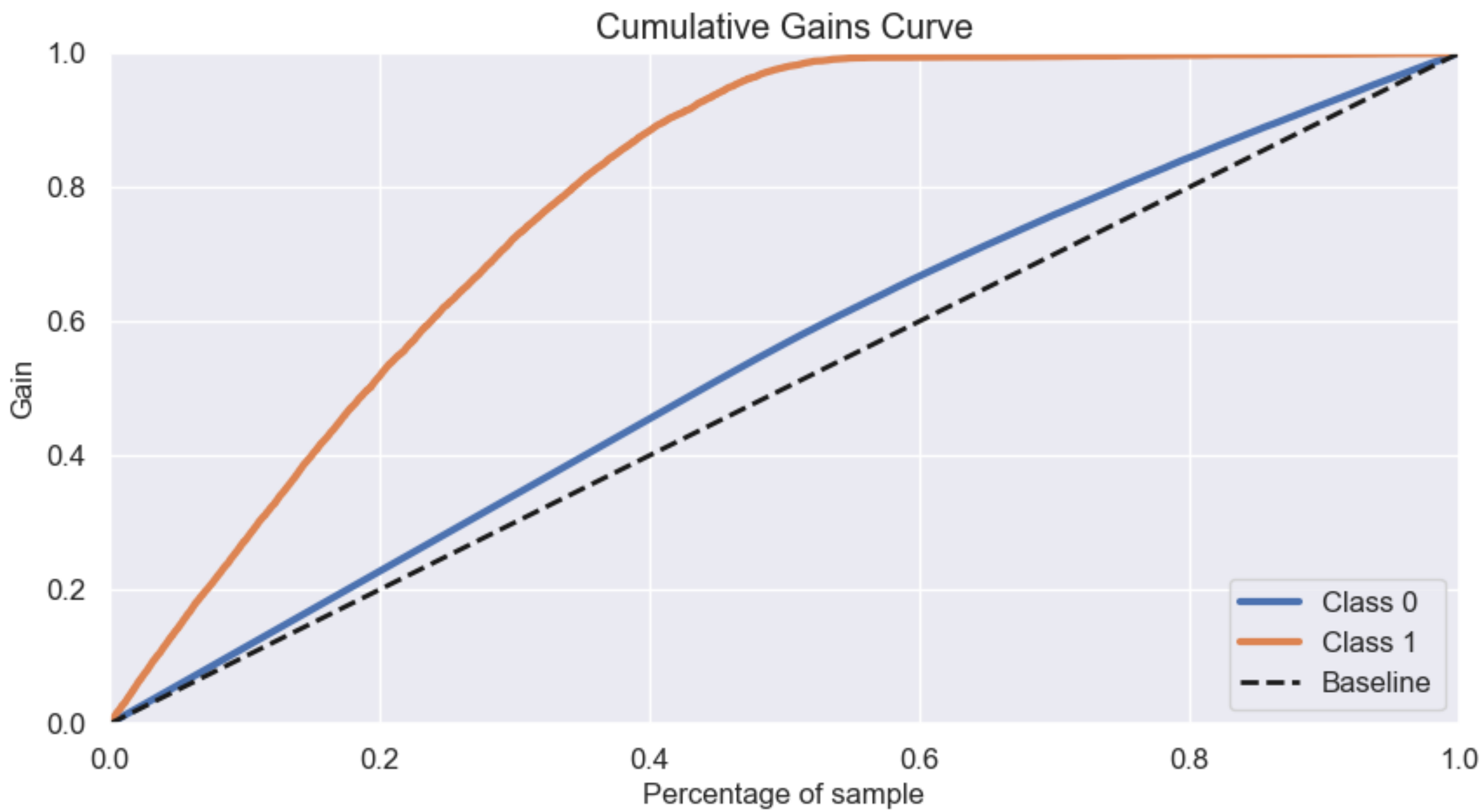
In [59]:
```
# Scikitplot is there to help

skplt.metrics.plot_lift_curve( y_val, yhat_lr );
```



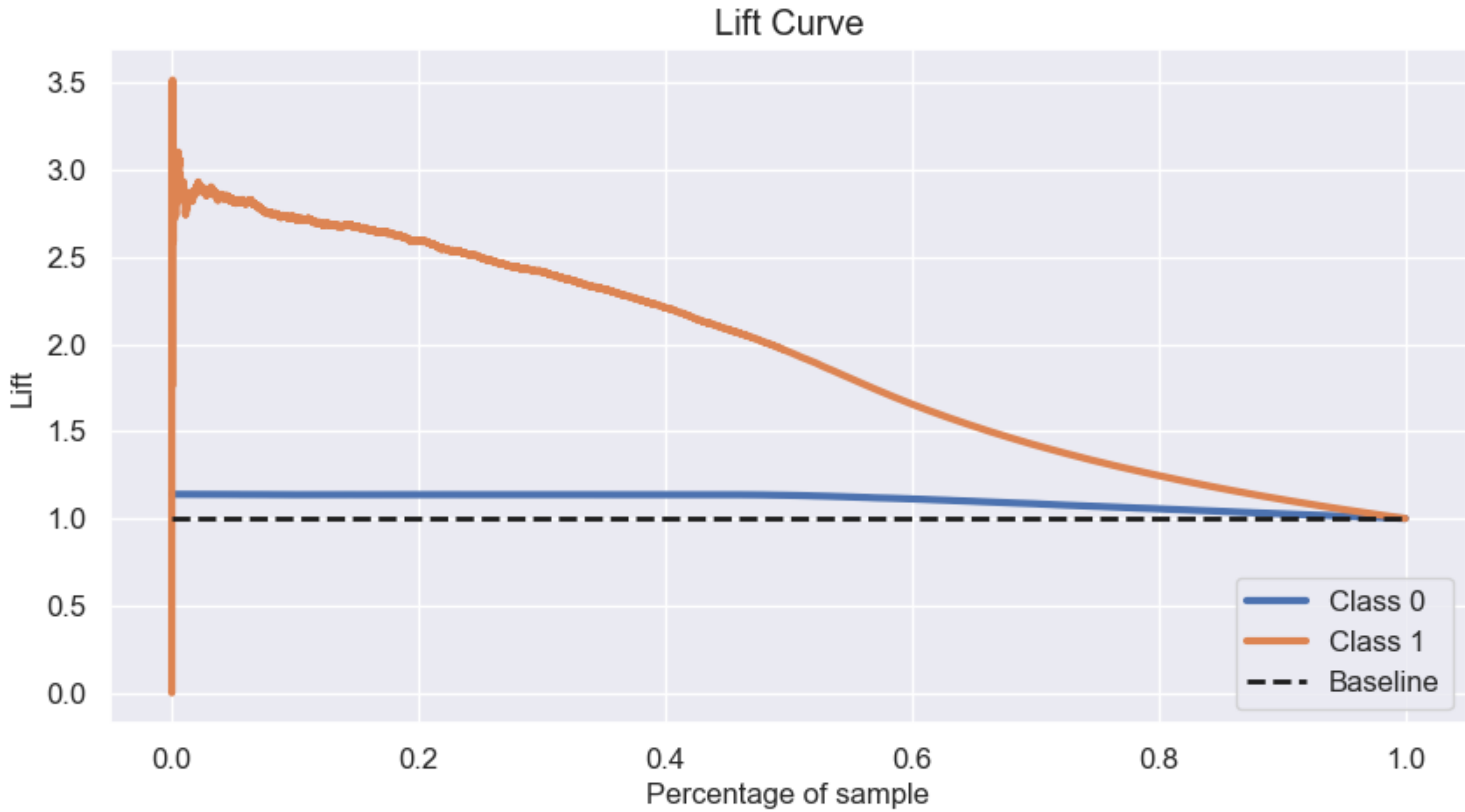
Extra Trees Classifier

In [60]:
```
# Model Definition

et = en.ExtraTreesClassifier( n_estimators=1000, n_jobs=-1, random_state=42 )

# Model Training

et.fit( x_train, y_train )

# Model Prediction

yhat_et = et.predict_proba( x_val )
```

In [62]:
```
# Accumulative Gain

skplt.metrics.plot_cumulative_gain( y_val, yhat_et );
```



In [61]:
```
# Scikitplot Library is there to help

skplt.metrics.plot_lift_curve( y_val, yhat_et );
```

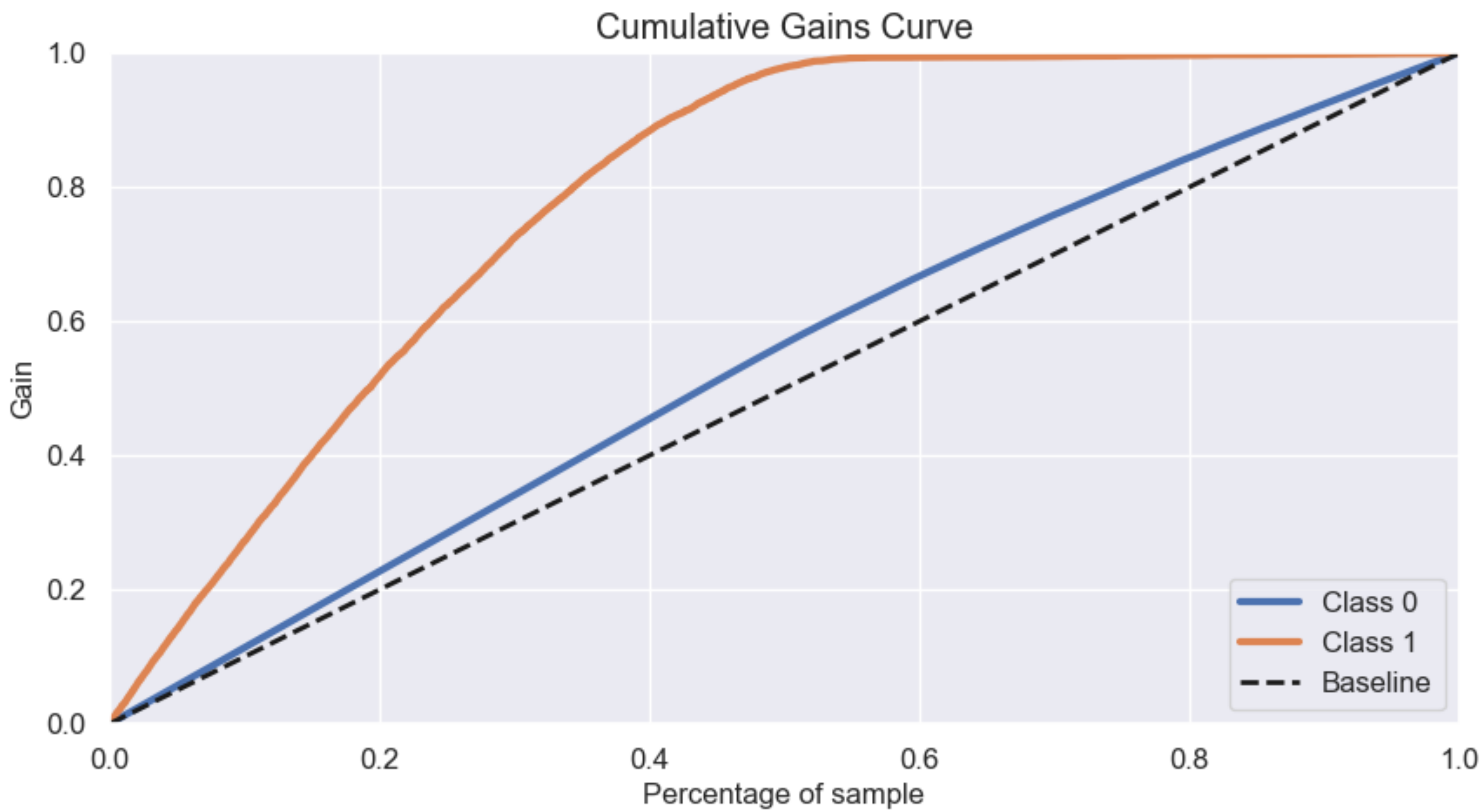


Random Forest

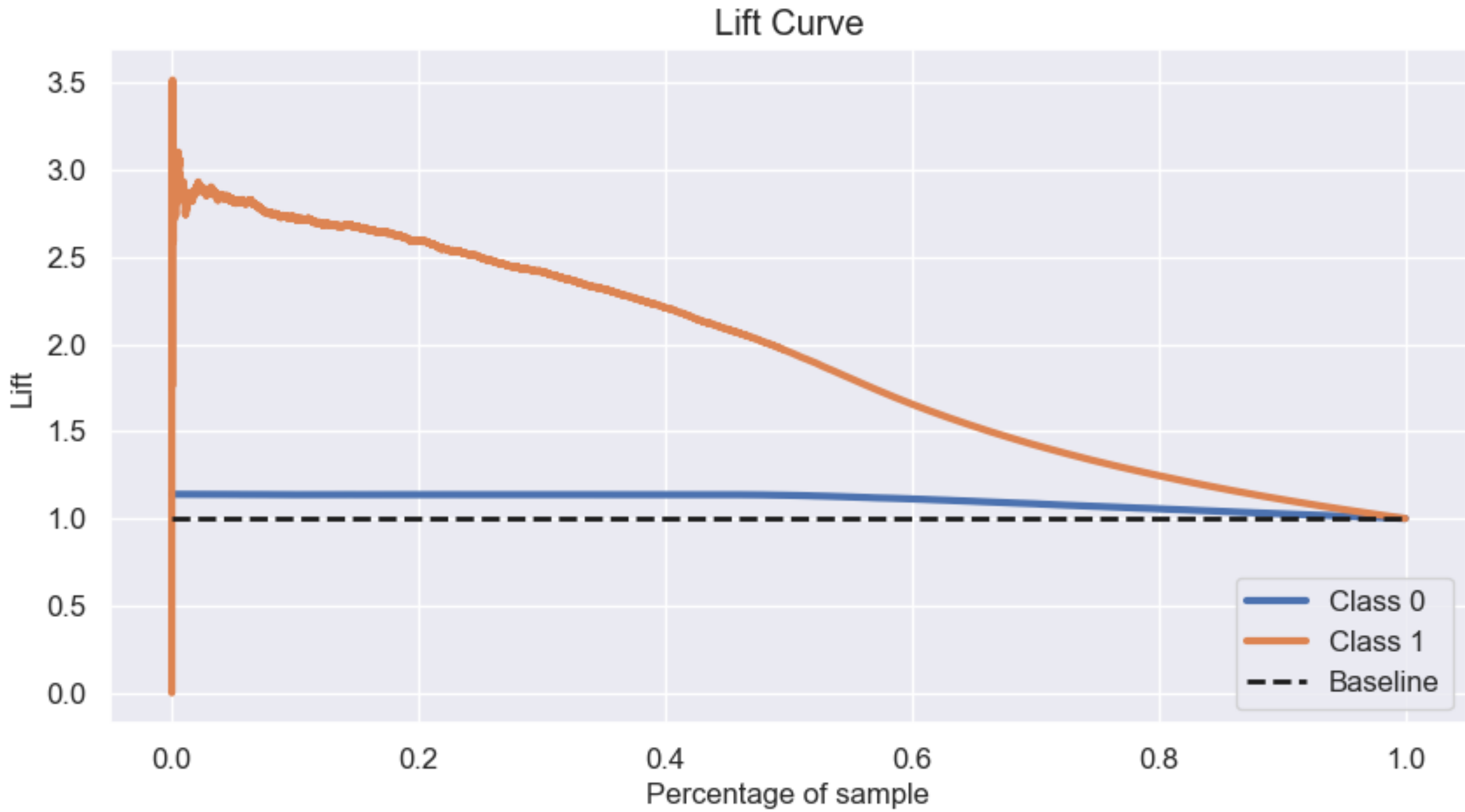
In [63]:

```
# model definition

rf = en.RandomForestClassifier( n_estimators=1000, n_jobs=-1, random_state=42 )

# model training

rf.fit( x_train, y_train )

# model prediction

yhat_rf = et.predict_proba( x_val )
```

In [64]:
```
# Accumulative Gain

skplt.metrics.plot_cumulative_gain( y_val, yhat_rf );
```



In [65]:
```
#Scikitplot library is there to help

skplt.metrics.plot_lift_curve( y_val, yhat_rf );
```



Performance Metrics

```
In [66]: # Create a copy of the validation dataset (x_validation) and the corresponding target values (y_validation)

df8 = x_validation.copy()
df8['response'] = y_validation.copy()

# Calculate the propensity scores, In this case, extracting the probabilities of the positive class (class 1) and converting them to a list

df8['score'] = yhat_et[:, 1].tolist()

# sort clients by propensity score, This orders clients from those with the highest propensity to those with the lowest propensity

df8 = df8.sort_values( 'score', ascending=False )

# Calculate the precision at a specified value of k, the precision at k measures the accuracy of positive predictions among the top k instar

precision_at_20, data = precision_at_k( df8, k=20 )

# Calculate the recall at a specified value of k, the recall at k measures the fraction of actual positive instances captured among the top

recall_at_15, data = recall_at_k( df8, k=15 )
```

High Precision:

Choose this when minimizing false positives is critical. For example, in a spam email filter, you want to avoid classifying legitimate emails as spam (false positives).

High Recall:

Choose this when identifying as many positive instances as possible is crucial, even if it means accepting more false positives. For instance, in medical diagnosis, it's important to detect as many cases of a disease as possible, even if it results in some false alarms.

In practice, you may use both precision and recall together, often summarized using the F1-score, which is the harmonic mean of precision and recall. The choice of metrics should align with your specific objectives and the consequences of false positives and false negatives in your application.

In [67]:
```python
# Import the necessary function for calculating Top-K Accuracy Score from scikit-learn

from sklearn.metrics import top_k_accuracy_score

# Define the true labels for a set of instances

y_true = np.array( [0, 1, 2, 2] )

# Define the predicted probabilities for each class for the same set of instances

y_score = np.array( [[0.5, 0.2, 0.2],  # 0 is in top 2, Predicted probabilities for class 0, 1, and 2 for the first instance
                    [0.3, 0.4, 0.2],   # 1 is in top 2, Predicted probabilities for class 0, 1, and 2 for the second instance
                    [0.2, 0.4, 0.3],   # 2 is in top 2, Predicted probabilities for class 0, 1, and 2 for the third instance
                    [0.7, 0.2, 0.1]] ) # 2 isn't in top 2, Predicted probabilities for class 0, 1, and 2 for the fourth instance

# Calculate the Top-K Accuracy Score, this score measures the accuracy of predicting whether the true label is among the top-K predicted lal

top_k_accuracy_score( y_true, y_score, k=2 )
```

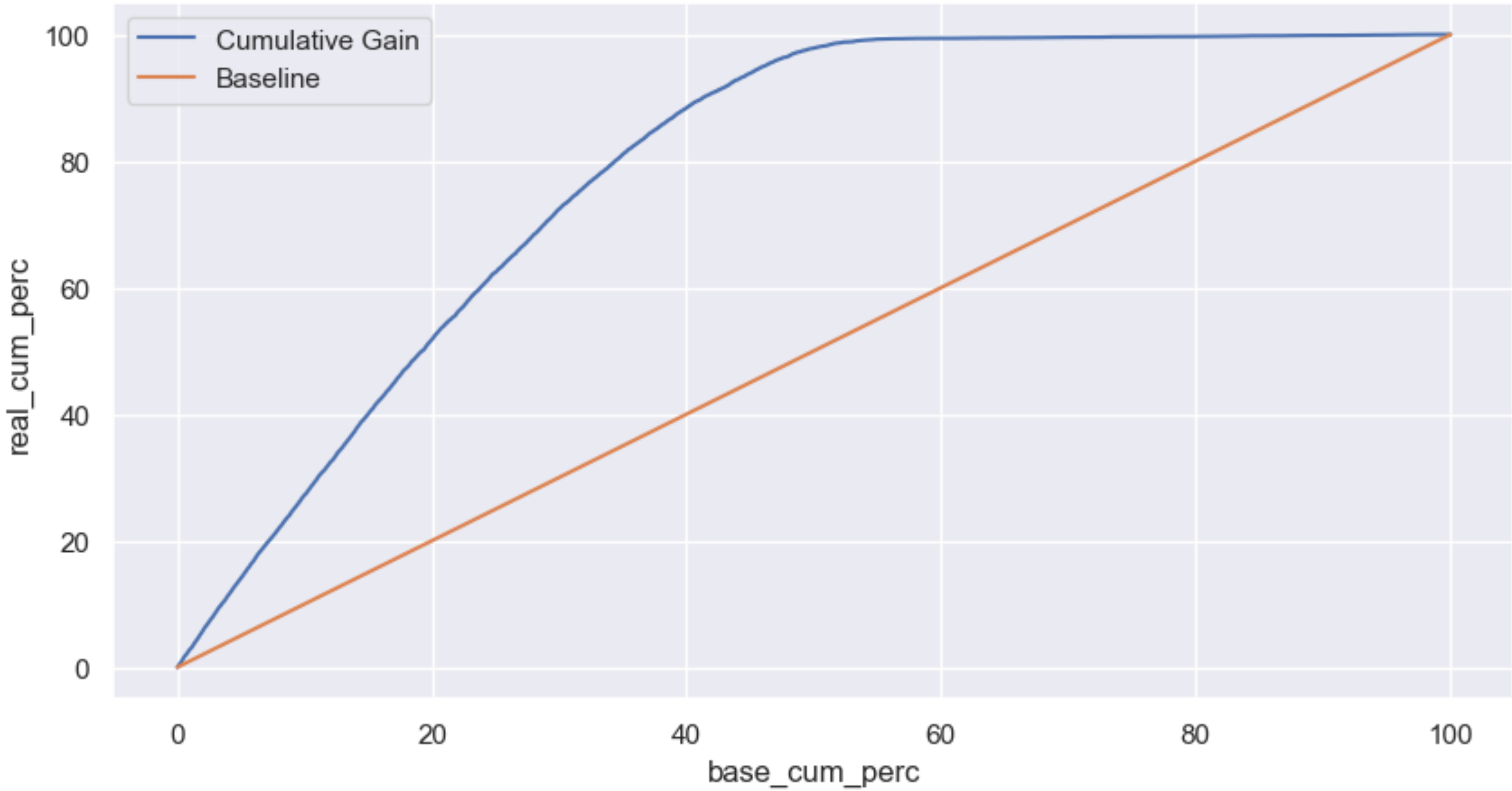
Out[67]:
```
0.75
```

In [68]:
```python
# Define an array of true labels for a set of instances

y_true = np.array( [1, 0, 1, 1, 0, 1, 0, 0] )

# Define an array of predicted probabilities (empty in this example)

y_score = np.array( [] )
```

Cummulative Curve Manually

In [69]:

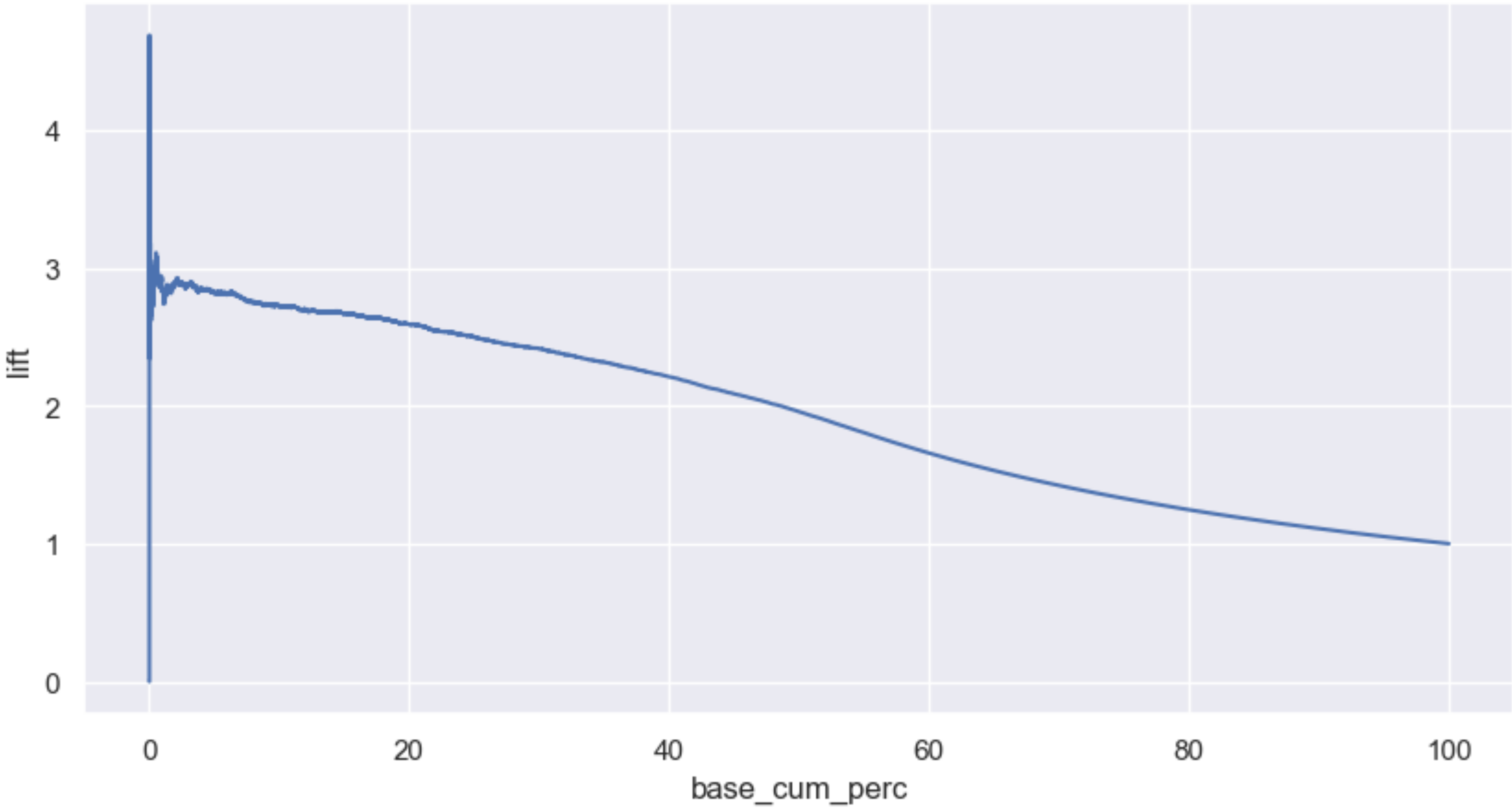
```
# Create a DataFrame to store model predictions and true labels

results = pd.DataFrame()
results['prediction'] = yhat_et[:,1].tolist() # Predicted probabilities for the positive class
results['real'] = y_val.tolist() # True labels (actual outcomes)

# Sort the results DataFrame by prediction in descending order

results = results.sort_values( 'prediction', ascending=False )

# Calculate the percentage of interest (Propensity Score)

results['real_cum'] = results['real'].cumsum()  # Cumulative sum of true positives
results['real_cum_perc'] = 100 * results['real_cum'] / results['real'].sum() # Percentage of true positives

# Calculate the percentage of the base (Clients)

results['base'] = range( 1, len( results ) + 1  ) # A range of integers representing clients
results['base_cum_perc'] = 100 * results['base'] / len( results ) # Percentage of clients

# Create a baseline model for comparison, based on client percentage

results['baseline'] = results['base_cum_perc']

# Create a line plot to visualize the cumulative gain chart

sns.lineplot( x='base_cum_perc', y='real_cum_perc', data=results, label='Cumulative Gain' );
sns.lineplot( x='base_cum_perc', y='baseline', data=results, label='Baseline' );
plt.legend();
```



Lift Curve Manually

In [70]:

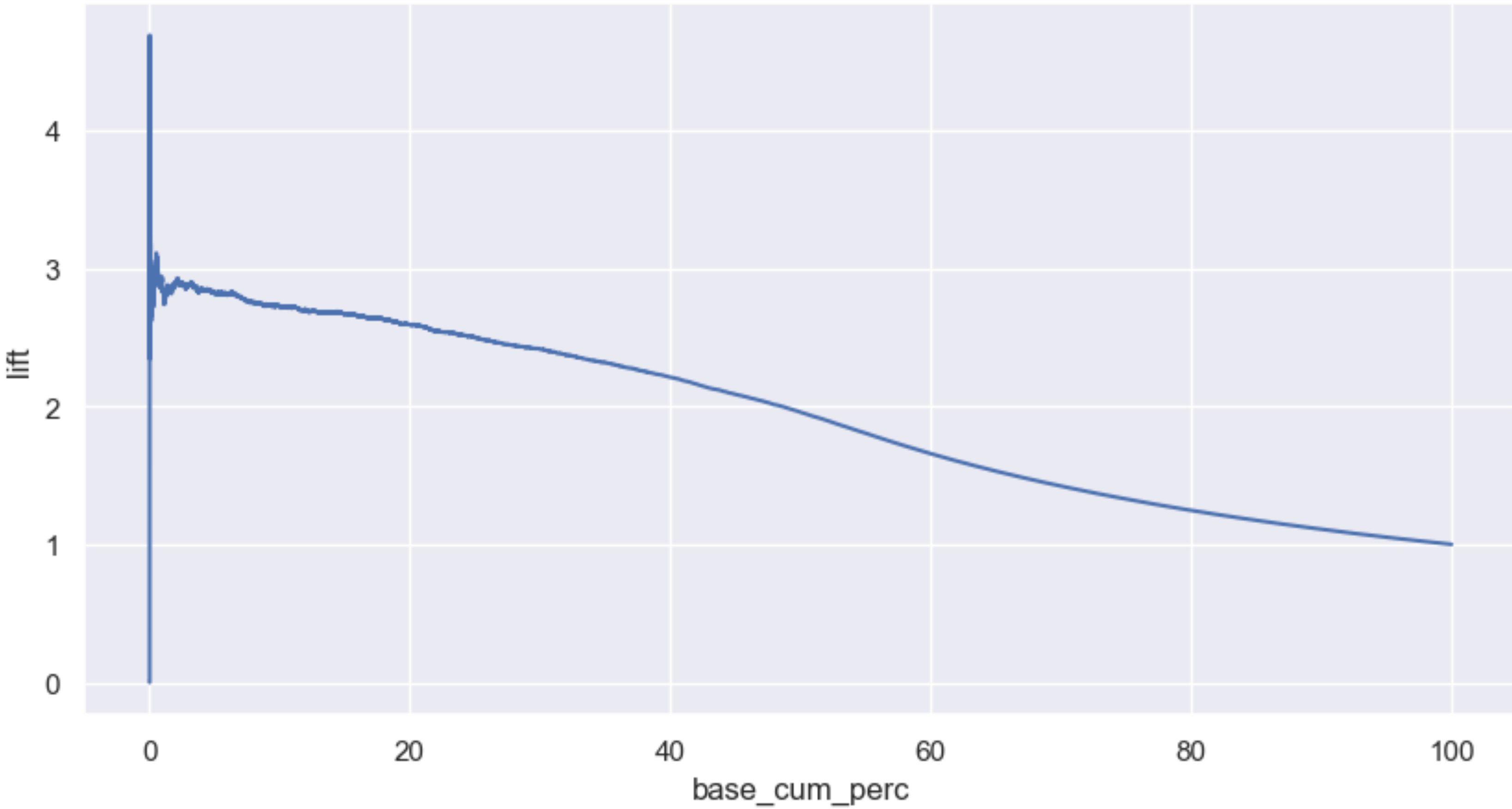
```python
# Create an empty DataFrame to store model performance metrics

results = pd.DataFrame()

# Store the predicted probabilities for the positive class in the DataFrame

results['prediction'] = yhat_et[:,1].tolist()

# Store the true labels (actual outcomes) in the DataFrame

results['real'] = y_val.tolist()

# Sort the DataFrame by predicted probabilities in descending order

results = results.sort_values( 'prediction', ascending=False )

# Calculate the cumulative sum of true positives (Propensity Score)

results['real_cum'] = results['real'].cumsum()

# Calculate the percentage of true positives relative to the total true positives (Propensity Score)

results['real_cum_perc'] = 100 * results['real_cum'] / results['real'].sum()

# Create a range of integers representing the number of clients (Base)

results['base'] = range( 1, len( results ) + 1 )

# Calculate the percentage of clients (Base) relative to the total number of clients

results['base_cum_perc'] = 100 * results['base'] / len( results )

# Create a baseline model based on client percentage

results['baseline'] = results['base_cum_perc']

# Calculate the lift, which is the ratio of Propensity Score to Baseline

results['lift'] = results['real_cum_perc'] / results['base_cum_perc']

# the lift chart helps us understand how well our model is at identifying the target group compared to a baseline model.
# A higher lift value and a steeper curve are indicators of better model performance in targeting the group of interest.

sns.lineplot( x='base_cum_perc', y='lift', data=results );
```



ROI Curve Manually

In [71]:

```
# Create an empty DataFrame to store model performance metrics

results = pd.DataFrame()

# Store the predicted probabilities for the positive class in the DataFrame, Predicted probabilities for the positive class

results['prediction'] = yhat_et[:,1].tolist()

# Store the true labels (actual outcomes) in the DataFrame

results['real'] = y_val.tolist()

# Sort the DataFrame by predicted probabilities in descending order

results = results.sort_values( 'prediction', ascending=False )

# Calculate the cumulative sum of true positives (Propensity Score)

results['real_cum'] = results['real'].cumsum()

# Calculate the percentage of true positives relative to the total true positives (Propensity Score)

results['real_cum_perc'] = 100 * results['real_cum'] / results['real'].sum()

# Create a range of integers representing the number of clients (Base)

results['base'] = range( 1, len( results ) + 1 )

# Calculate the percentage of clients (Base) relative to the total number of clients

results['base_cum_perc'] = 100 * results['base'] / len( results )

# Create a baseline model based on client percentage

results['baseline'] = results['base_cum_perc']

# Calculate the lift, which is the ratio of Propensity Score to Baseline

results['lift'] = results['real_cum_perc'] / results['base_cum_perc']

# This code calculates and visualizes the ROI Curve (Lift Curve), which helps assess the effectiveness of
# a model in targeting a specific group compared to a baseline model.

sns.lineplot( x='base_cum_perc', y='lift', data=results );
```



4
3
2
1
0
lift
0
20
40
60
80
100
base_cum_perc

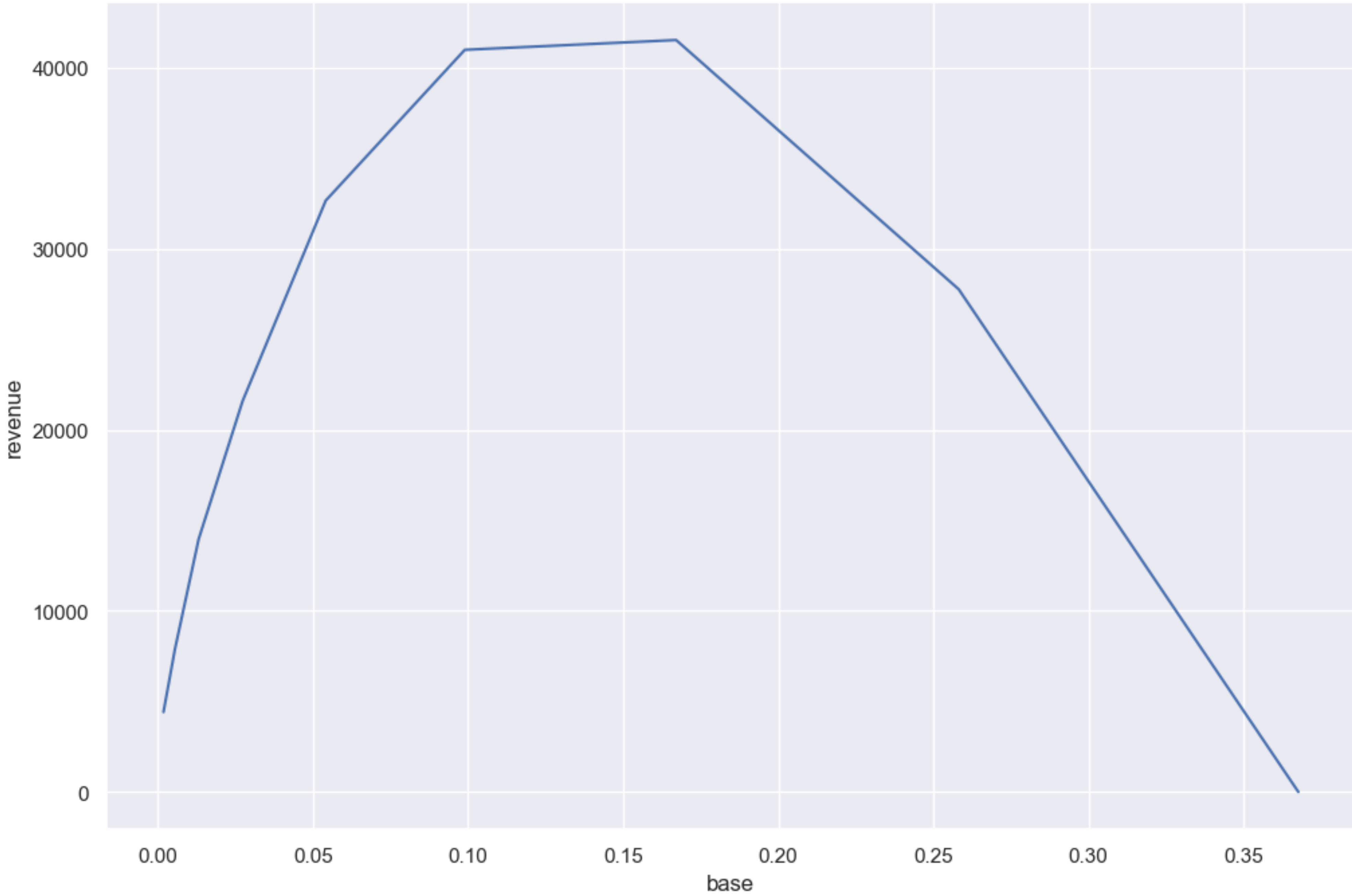
```
In [72]: # Compute the 'bucket' for each prediction based on predefined ranges

results['bucket'] = results['prediction'].apply( lambda x: 0.9 if x >= 0.90 else
                                                           0.8 if ( x >= 0.80) & ( x <= 0.90 ) else
                                                           0.7 if ( x >= 0.70) & ( x <= 0.80 ) else
                                                           0.6 if ( x >= 0.60) & ( x <= 0.70 ) else
                                                           0.5 if ( x >= 0.50) & ( x <= 0.60 ) else
                                                           0.4 if ( x >= 0.40) & ( x <= 0.50 ) else
                                                           0.3 if ( x >= 0.30) & ( x <= 0.40 ) else
                                                           0.2 if ( x >= 0.20) & ( x <= 0.30 ) else
                                                           0.1 if ( x >= 0.10) & ( x <= 0.20 ) else 0.01 )

# Aggregate clients among the defined 'buckets' and calculate the minimum propensity score and count

df = results[['prediction','bucket']].groupby( 'bucket' ).agg( {'min', 'count'} ).reset_index()
df.columns = df.columns.droplevel()
df.columns = ['index', 'clients', 'propensity_score']

# Compute gross revenue and cost for each bucket

df['gross_revenue'] = 40 * df['clients'] * df['propensity_score']
df['cost'] = 4 * df['clients']

# Calculate the cumulative percentage of clients

df['base'] = df['clients'].sort_values( ascending=True ).cumsum() / df['clients'].sum()

# Calculate the net revenue (revenue - cost) for each bucket

df['revenue'] = df['gross_revenue'] - df['cost']

# Sort the DataFrame by 'index' in descending order

df = df.sort_values( 'index', ascending=False )
df

# This code segments clients into buckets based on their predicted propensity scores and computes revenue and cost metrics for each bucket.
```

Out[72]:

	index	clients	propensity_score	gross_revenue	cost	base	revenue
9	0.90	138	0.9	4968.0	552	0.001811	4416.0
8	0.80	282	0.8	9024.0	1128	0.005510	7896.0
7	0.70	582	0.7	16296.0	2328	0.013146	13968.0
6	0.60	1080	0.6	25920.0	4320	0.027315	21600.0
5	0.50	2042	0.5	40840.0	8168	0.054105	32672.0
4	0.40	3417	0.4	54672.0	13668	0.098935	41004.0
3	0.30	5193	0.3	62316.0	20772	0.167065	41544.0
2	0.20	6943	0.2	55544.0	27772	0.258154	27772.0
1	0.10	8345	0.1	33380.0	33380	0.367637	0.0
0	0.01	48200	0.0	0.0	192800	1.000000	-192800.0

In [73]:

```
plt.figure( figsize=(12,8))

# Filter the DataFrame to include only clients with a propensity score greater than or equal to 0.1

aux = df[df['propensity_score'] >= 0.1]

# Create a line plot to visualize the relationship between the cumulative percentage of clients ('base') and revenue for the selected clien

sns.lineplot( x='base', y='revenue', data=aux );

# This code filters and selects clients with a propensity score greater than or equal to 0.1 and then visualizes their cumulative percentage
```



Deploy to Production

In [ ]:
```
# Save the Trained Model

# pickle.dump( et, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\pickle\\model_et_heal
# pickle.dump( lr_model, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\health_insurance_cross-sell\\pickle\\model_l
```

Health Insurance Class

In [ ]:

```
# import pickle
# import numpy as np
# import pandas as pd

# class HealthInsurance( object ):
#     def __init__(self):
#         self.home_path = ''
#         self.annual_premium_scaler            = pickle.load( open( self.home_path + 'parameter/annual_premium_scaler.pkl', 'rb' ) )
#         self.age_scaler                       = pickle.load( open( self.home_path + 'parameter/age_scaler.pkl', 'rb' ) )
#         self.vintage_scaler                   = pickle.load( open( self.home_path + 'parameter/vintage_scaler.pkl', 'rb' ) )
#         self.target_encode_gender_scaler      = pickle.load( open( self.home_path + 'parameter/target_encode_gender_scaler.pkl', 'rb' ) )
#         self.target_encode_region_code_scaler = pickle.load( open( self.home_path + 'parameter/target_encode_region_code_scaler.pkl', 'rb
#         self.fe_policy_sales_channel_scaler   = pickle.load( open( self.home_path + 'parameter/fe_policy_sales_channel_scaler.pkl', 'rb' 

#     def data_cleaning( self, df1 ):

#         # 1.1. Rename Columns
#         cols_new = ['id', 'gender', 'age', 'driving_license', 'region_code', 'previously_insured', 'vehicle_age',
#                     'vehicle_damage', 'annual_premium','policy_sales_channel', 'vintage', 'response']

#         # rename
#         df1.columns = cols_new

#         return df1

#     def feature_engineering( self, df2 ):
#         # 2.0. Feature Engineering

#         # Vehicle Damage Number
#         df2['vehicle_damage'] = df2['vehicle_damage'].apply( lambda x: 1 if x == 'Yes' else 0 )

#         # Vehicle Age
#         df2['vehicle_age'] = df2['vehicle_age'].apply( lambda x: 'over_2_years' if x == '> 2 Years'
#                                                                   else 'between_1_2_year' if x == '1-2 Year'
#                                                                   else 'below_1_year' )

#         return df2

#     def data_preparation( self, df5 ):

#         # anual premium - StandarScaler
#         df5['annual_premium'] = self.annual_premium_scaler.transform( df5[['annual_premium']].values )

#         # Age - MinMaxScaler
#         df5['age'] = self.age_scaler.transform( df5[['age']].values )

#         # Vintage - MinMaxScaler
#         df5['vintage'] = self.vintage_scaler.transform( df5[['vintage']].values )
```

```
#         # gender - One Hot Encoding / Target Encoding
#         df5.loc[:, 'gender'] = df5['gender'].map( self.target_encode_gender_scaler )

#         # region_code - Target Encoding / Frequency Encoding
#         df5.loc[:, 'region_code'] = df5['region_code'].map( self.target_encode_region_code_scaler )

#         # vehicle_age - One Hot Encoding / Frequency Encoding
#         df5 = pd.get_dummies( df5, prefix='vehicle_age', columns=['vehicle_age'] )

#         # policy_sales_channel - Target Encoding / Frequency Encoding
#         df5.loc[:, 'policy_sales_channel'] = df5['policy_sales_channel'].map( self.fe_policy_sales_channel_scaler )

#         # Feature Selection
#         cols_selected = ['annual_premium', 'vintage', 'age', 'region_code', 'vehicle_damage', 'previously_insured',
#                          'policy_sales_channel']

#         return df5[cols_selected]

#     def get_prediction( self, model, original_data, test_data ):

#         # model prediction
#         pred = model.predict_proba( test_data )

#         # join prediction into original data
#         original_data['prediction'] = pred

#         return original_data.to_json( orient='records', date_format='iso' )
```

API Handler

In [ ]:

```
# import pickle
# import pandas as pd
# from flask import Flask, request, Response
# from healthInsurance.HealthInsurance import HealthInsurance
# import os

# # loading model

# model = pickle.load( open( 'model/model_linear_regression.pkl', 'rb' ) )

# # initialize API

# app = Flask(__name__)


# @app.route( '/healthInsurance/predict', methods=['GET', 'POST'])
# def health_insurance_predict():
#     test_json = request.get_json()

#     if test_json:  # there is data
#         if isinstance( test_json, dict ):  # unique example
#             test_raw = pd.DataFrame( test_json, index=[0] )

#         else:  # multiple example
#             test_raw = pd.DataFrame( test_json, columns=test_json[0].keys() )

#         # Instantiate Rossmann class
#         pipeline = HealthInsurance()

#         # data cleaning
#         df1 = pipeline.data_cleaning( test_raw )

#         # feature engineering
#         df2 = pipeline.feature_engineering( df1 )

#         # data preparation
#         df3 = pipeline.data_preparation( df2 )

#         # prediction
#         df_response = pipeline.get_prediction( model, test_raw, df3 )

#         return df_response

#     else:
#         return Response( '{}', status=200, mimetype='application/json' )

# if __name__ == '__main__':
#     port = os.environ.get( 'PORT', 5000 )
```

```
#     app.run( host='0.0.0.0', port=port )
```

API Tester

In [119…

```
import requests

# Loading test dataset

df_test = x_validation
df_test['response'] = y_validation
```

In [120…

```
df_test = df_test.sample(10)
```

In [121…

```
df_test.head()
```

Out[121]:

	id	gender	age	driving_license	region_code	previously_insured	vehicle_age	vehicle_damage	annual_premium	policy_sales_channel	vintage	res
237128	237129	Female	47	1	28.0	1	between_1_2_year	0	37286.0	26.0	66	
27684	27685	Male	37	1	11.0	0	between_1_2_year	1	36168.0	124.0	211	
272910	272911	Female	23	1	39.0	0	below_1_year	1	2630.0	152.0	175	
135626	135627	Female	30	1	33.0	1	below_1_year	0	2630.0	152.0	291	
154606	154607	Female	21	1	43.0	0	between_1_2_year	1	33156.0	26.0	197	

In [122…

```
# Convert Dataframe to json

data = json.dumps( df_test.to_dict( orient='records' ) )
data
```

Out[122]:


```
'[{"id": 237129, "gender": "Female", "age": 47, "driving_license": 1, "region_code": 28.0, "previously_insured": 1, "vehicle_age": "between_1_2_year", "vehicle_damage": 0, "annual_premium": 37286.0, "policy_sales_channel": 26.0, "vintage": 66, "response": 0}, {"id": 27685, "gender": "Male", "age": 37, "driving_license": 1, "region_code": 11.0, "previously_insured": 0, "vehicle_age": "between_1_2_year", "vehicle_damage": 1, "annual_premium": 36168.0, "policy_sales_channel": 124.0, "vintage": 211, "response": 0}, {"id": 272911, "gender": "Female", "age": 23, "driving_license": 1, "region_code": 39.0, "previously_insured": 0, "vehicle_age": "below_1_year", "vehicle_damage": 1, "annual_premium": 2630.0, "policy_sales_channel": 152.0, "vintage": 175, "response": 0}, {"id": 135627, "gender": "Female", "age": 30, "driving_license": 1, "region_code": 33.0, "previously_insured": 1, "vehicle_age": "below_1_year", "vehicle_damage": 0, "annual_premium": 2630.0, "policy_sales_channel": 152.0, "vintage": 291, "response": 0}, {"id": 154607, "gender": "Female", "age": 21, "driving_license": 1, "region_code": 43.0, "previously_insured": 0, "vehicle_age": "between_1_2_year", "vehicle_damage": 1, "annual_premium": 33156.0, "policy_sales_channel": 26.0, "vintage": 197, "response": 1}, {"id": 177727, "gender": "Male", "age": 21, "driving_license": 1, "region_code": 8.0, "previously_insured": 1, "vehicle_age": "below_1_year", "vehicle_damage": 0, "annual_premium": 57970.0, "policy_sales_channel": 160.0, "vintage": 56, "response": 0}, {"id": 299589, "gender": "Male", "age": 23, "driving_license": 1, "region_code": 8.0, "previously_insured": 1, "vehicle_age": "below_1_year", "vehicle_damage": 0, "annual_premium": 50420.0, "policy_sales_channel": 152.0, "vintage": 193, "response": 0}, {"id": 339019, "gender": "Female", "age": 42, "driving_license": 1, "region_code": 48.0, "previously_insured": 0, "vehicle_age": "between_1_2_year", "vehicle_damage": 1, "annual_premium": 2630.0, "policy_sales_channel": 15.0, "vintage": 110, "response": 0}, {"id": 135570, "gender": "Male", "age": 60, "driving_license": 1, "region_code": 28.0, "previously_insured": 0, "vehicle_age": "between_1_2_year", "vehicle_damage": 1, "annual_premium": 41719.0, "policy_sales_channel": 26.0, "vintage": 75, "response": 1}, {"id": 45762, "gender": "Male", "age": 31, "driving_license": 1, "region_code": 21.0, "previously_insured": 0, "vehicle_age": "between_1_2_year", "vehicle_damage": 1, "annual_premium": 2630.0, "policy_sales_channel": 26.0, "vintage": 126, "response": 1}]'
```


In [123…

```python
# API Call

# url = 'http://0.0.0.0:5000/predict'
url = 'https://health-insuarance.onrender.com/healthInsurance/predict'
header = {'Content-type': 'application/json' }

r = requests.post( url, data=data, headers=header )
print( 'Status Code {}'.format( r.status_code ) )
```

```
Status Code 500
```

In [124…

```python
d1 = pd.DataFrame( r.json(), columns=r.json()[0].keys() )
d1.sort_values( 'score', ascending=False ).head()
```

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
File ~\anaconda3\envs\exercises_1\lib\site-packages\requests\models.py:971, in Response.json(self, **kwargs)
    970 try:
--> 971     return complexjson.loads(self.text, **kwargs)
    972 except JSONDecodeError as e:
    973     # Catch JSON-related errors and raise as requests.JSONDecodeError
    974     # This aliases json.JSONDecodeError and simplejson.JSONDecodeError

File ~\anaconda3\envs\exercises_1\lib\json\__init__.py:357, in loads(s, cls, object_hook, parse_float, parse_int, parse_constant, object_pa
irs_hook, **kw)
    354 if (cls is None and object_hook is None and
    355         parse_int is None and parse_float is None and
    356         parse_constant is None and object_pairs_hook is None and not kw):
--> 357     return _default_decoder.decode(s)
    358 if cls is None:

File ~\anaconda3\envs\exercises_1\lib\json\decoder.py:337, in JSONDecoder.decode(self, s, _w)
    333 """Return the Python representation of ``s`` (a ``str`` instance
    334 containing a JSON document).
    335 
    336 """
--> 337 obj, end = self.raw_decode(s, idx=_w(s, 0).end())
    338 end = _w(s, end).end()

File ~\anaconda3\envs\exercises_1\lib\json\decoder.py:355, in JSONDecoder.raw_decode(self, s, idx)
    354 except StopIteration as err:
--> 355     raise JSONDecodeError("Expecting value", s, err.value) from None
    356 return obj, end

JSONDecodeError: Expecting value: line 1 column 1 (char 0)

During handling of the above exception, another exception occurred:

JSONDecodeError                           Traceback (most recent call last)
Cell In[124], line 1
----> 1 d1 = pd.DataFrame( r.json(), columns=r.json()[0].keys() )
      2 d1.sort_values( 'score', ascending=False ).head()

File ~\anaconda3\envs\exercises_1\lib\site-packages\requests\models.py:975, in Response.json(self, **kwargs)
    971     return complexjson.loads(self.text, **kwargs)
    972 except JSONDecodeError as e:
    973     # Catch JSON-related errors and raise as requests.JSONDecodeError
    974     # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
--> 975     raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

In [286…

```python
# API Call
#url = 'http://0.0.0.0:5000/predict'
url = 'https://health-insurance-model.herokuapp.com/predict'
header = {'Content-type': 'application/json' }

r = requests.post( url, data=data, headers=header )
print( 'Status Code {}'.format( r.status_code ) )
```

```
Status Code 200
```

In [287…

```python
d1 = pd.DataFrame( r.json(), columns=r.json()[0].keys() )
d1.sort_values( 'score', ascending=False ).head()
```

Out[287]:

	id	gender	age	driving_license	region_code	previously_insured	vehicle_age	vehicle_damage	annual_premium	policy_sales_channel	vintage	respor
3	363080	0.138780	0.369231	1	0.187988	0	below_1_year	0	0.492486	23.0	0.930796	
7	318230	0.138780	0.230769	1	0.187988	0	below_1_year	0	-0.511883	26.0	0.615917	
0	74147	0.099756	0.092308	1	0.187988	0	below_1_year	0	0.669245	124.0	0.961938	
1	322299	0.138780	0.369231	1	0.187988	0	below_1_year	0	0.839437	124.0	0.761246	
9	107812	0.138780	0.338462	1	0.187988	0	below_1_year	0	3.605010	26.0	0.640138	

In [ ]: