Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
%matplotlib inline
plt.style.use('fivethirtyeight')
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

Loading the Dataset :

In [2]:
```
data = pd.read_csv('ml_project1_data_pt.csv',delimiter =',')
data.head(3)
```

Out[2]:

	ID	ano_nasc	educacao	estado_civil	renda_ano	crianca_casa	adoles_casa	dt_primcomp	recencia_dias	vinho_montante	frutas_montante
0	5524	1957	Graduation	Single	58138.0	0	0	9/4/2012	58	635	88
1	2174	1954	Graduation	Single	46344.0	1	1	3/8/2014	38	11	1
2	4141	1965	Graduation	Together	71613.0	0	0	8/21/2013	26	426	49

In [3]:
```
data.shape
```

Out[3]:
```
(2240, 27)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['ID', 'ano_nasc', 'educacao', 'estado_civil', 'renda_ano', 'crianca_casa', 'adoles_casa', 'dt_primcomp', 'recen
cia_dias', 'vinho_montante', 'frutas_montante', 'carne_montante', 'peixe_montante', 'doces_montante', 'ouro_montante',
'promocoes_desconto', 'promocoes_web', 'promocoes_catalogo', 'promocoes_store', 'num_visit_web_ult_mes', 'Cmp3', 'Cmp
4', 'Cmp5', 'Cmp1', 'Cmp2', 'reclamacoes', 'target'], dtype='object')
```

Data Preprocessing

In [4]:
```
# Checking for null values.
info = pd.DataFrame(data=data.isnull().sum()).T.rename(index={0:'Null values'})
info = info.append(pd.DataFrame(data=data.isnull().sum()/data.shape[0] * 100).T.rename(index={0:'% Null values'}))
info
```

Out[4]:

	ID	ano_nasc	educacao	estado_civil	renda_ano	crianca_casa	adoles_casa	dt_primcomp	recencia_dias	vinho_montante	frutas_montant
Null values	0.0	0.0	0.0	0.0	24.000000	0.0	0.0	0.0	0.0	0.0	0.
% Null values	0.0	0.0	0.0	0.0	1.071429	0.0	0.0	0.0	0.0	0.0	0.

In [5]:
```
# Checking for Duplicates :
data.duplicated().sum()
```

Out[5]: 0

In [6]:
```
data.describe()
```

Out[6]:

	ID	ano_nasc	renda_ano	crianca_casa	adoles_casa	recencia_dias	vinho_montante	frutas_montante	carne_montante	peix
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	
mean	5592.159821	1968.805804	52247.251354	0.444196	0.506250	49.109375	303.935714	26.302232	166.950000	
std	3246.662198	11.984069	25173.076661	0.538398	0.544538	28.962453	336.597393	39.773434	225.715373	
min	0.000000	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000	24.000000	23.750000	1.000000	16.000000	
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000	49.000000	173.500000	8.000000	67.000000	
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000	74.000000	504.250000	33.000000	232.000000	
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	

In [7]:
```
data['dt_primcomp'] = pd.to_datetime(data['dt_primcomp'], errors='coerce')
data['dt_primcomp'] = data['dt_primcomp'].dt.strftime('%m/%Y')
```

In [8]:
```
data['age']= 2020 - data['ano_nasc']
```

In [9]:
```
data['renda_mes_media']= data['renda_ano']/12
```

In [10]:
```
data['campaing_engagement'] =( data['Cmp1']+data['Cmp2']+data['Cmp3']+data['Cmp4']+data['Cmp5']) /5
```

In [11]:
```
data['target'] = data['target'].astype(str)
data['Cmp1'] = data['Cmp1'].astype(str)
data['Cmp2'] = data['Cmp2'].astype(str)
data['Cmp3'] = data['Cmp3'].astype(str)
data['Cmp4'] = data['Cmp4'].astype(str)
data['Cmp5'] = data['Cmp5'].astype(str)
data['reclamacoes'] = data['reclamacoes'].astype(str)
data['digital_profile'] = '0'
data['digital_profile'][(data['num_visit_web_ult_mes']< 5) & (data['promocoes_web']<3)]='1'
```

Exploratory Data Analysis :

In [13]:
```
data.dtypes.groupby(data.dtypes).size()
```

Out[13]:
```
int64      17
float64     3
object     11
dtype: int64
```

In [15]:
```
dtypes = pd.DataFrame(data.dtypes.rename('type')).reset_index().astype('str')
dtypes = dtypes.query('index != "dt_primcomp"',)
dtypes = dtypes.query('index != "ID"')
dtypes = dtypes.query('index != "target"')
numeric = dtypes[(dtypes.type.isin(['int64', 'float64']))]['index'].values
categorical = dtypes[~(dtypes['index'].isin(numeric)) & (dtypes['index'] != 'target')]['index'].values

print('Numeric:\n', numeric)
print('Categorical:\n', categorical)
```

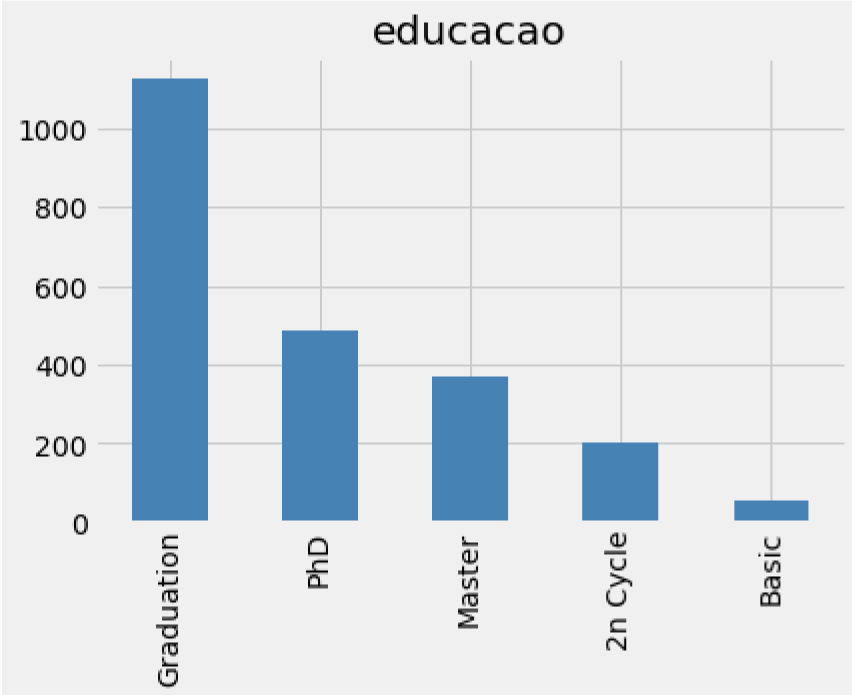
```
Numeric:
 ['ano_nasc' 'renda_ano' 'crianca_casa' 'adoles_casa' 'recencia_dias'
 'vinho_montante' 'frutas_montante' 'carne_montante' 'peixe_montante'
 'doces_montante' 'ouro_montante' 'promocoes_desconto' 'promocoes_web'
 'promocoes_catalogo' 'promocoes_store' 'num_visit_web_ult_mes' 'age'
 'renda_mes_media' 'campaing_engagement']
Categorical:
 ['educacao' 'estado_civil' 'Cmp3' 'Cmp4' 'Cmp5' 'Cmp1' 'Cmp2'
 'reclamacoes' 'digital_profile']
```

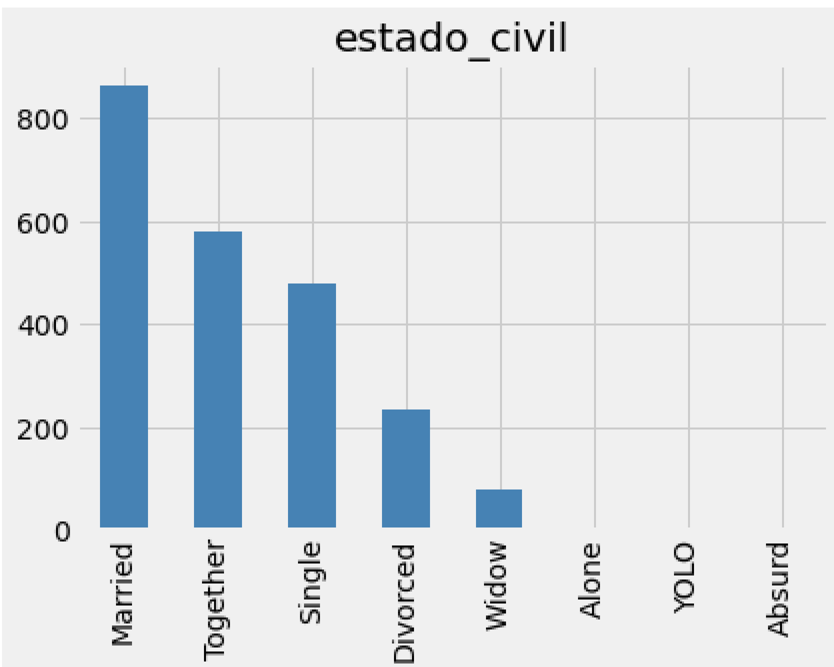
Categorical Data Analysis

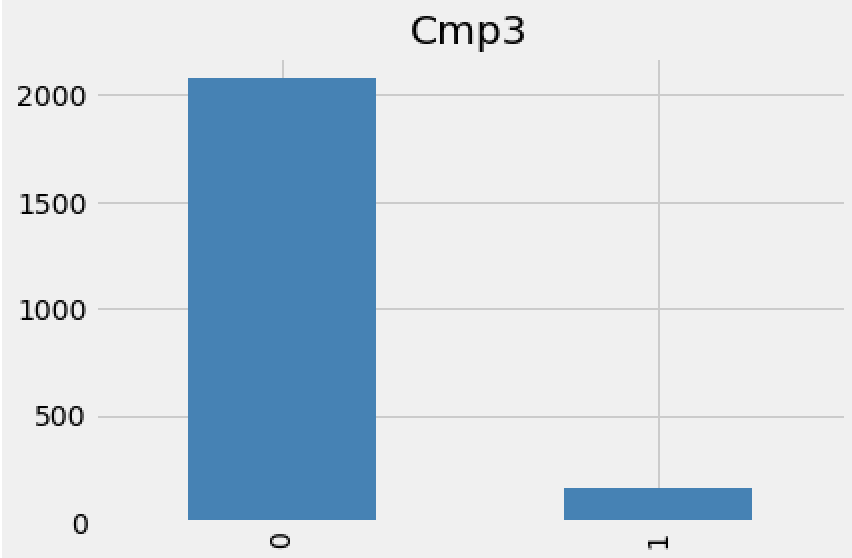
In [830]:
```
pylab.rcParams['figure.figsize'] = (6.0, 4.0)
```

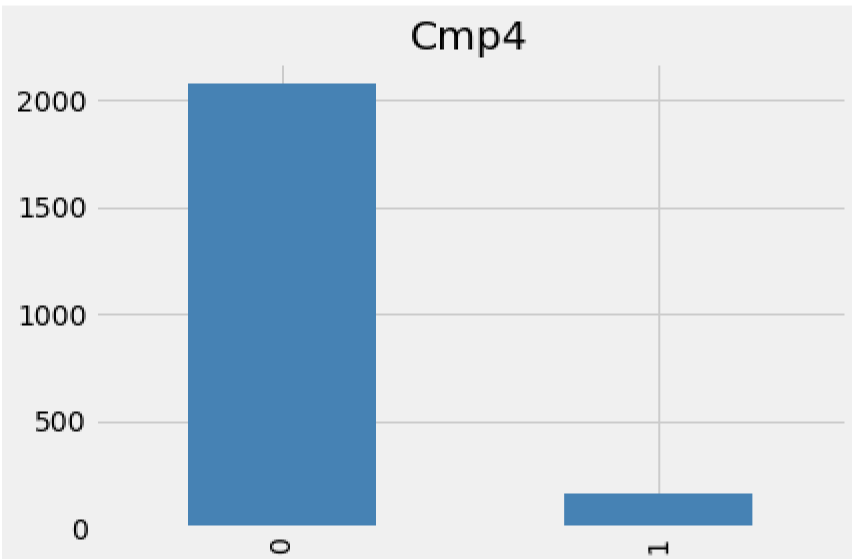
In [16]:

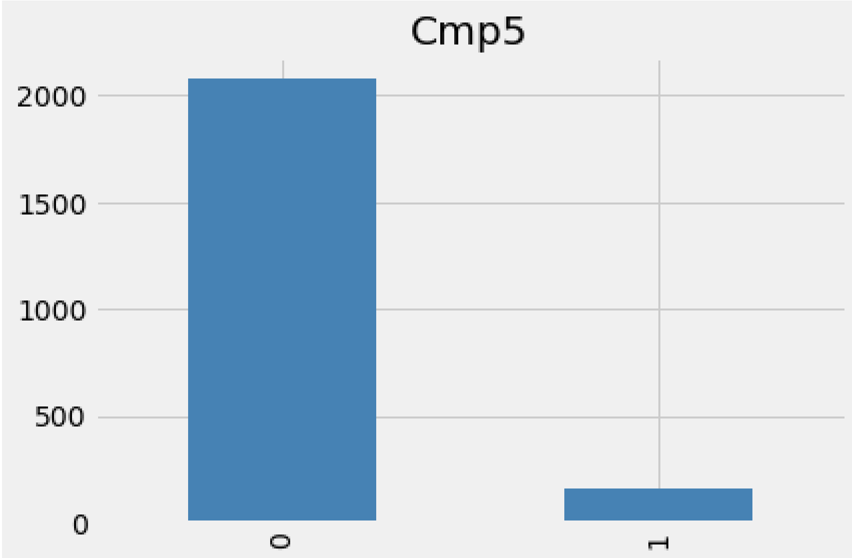
```
for attr in categorical:
  figsize=(8,4)
  plt.figure()
  data[attr].value_counts().plot(kind='bar', color='steelblue');
  plt.title(attr);
```

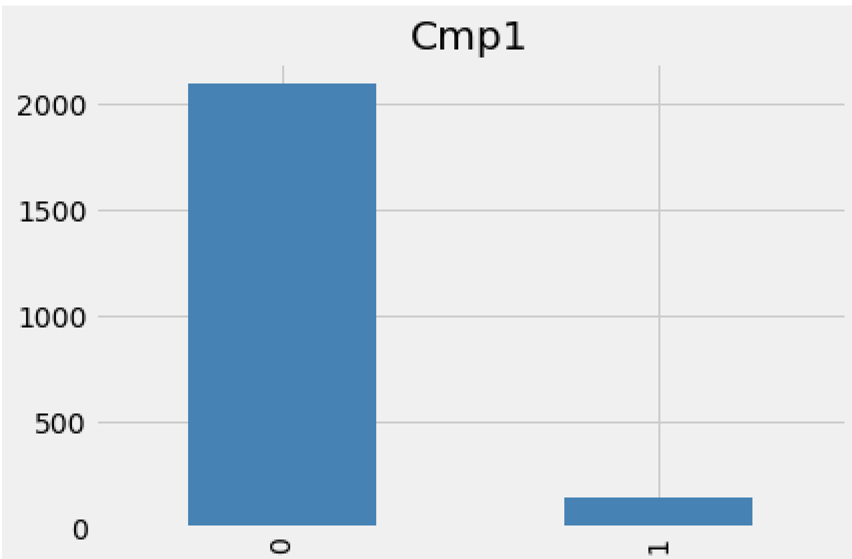

educacao
1000
800
600
400
200
0
Graduation
PhD
Master
2n Cycle
Basic

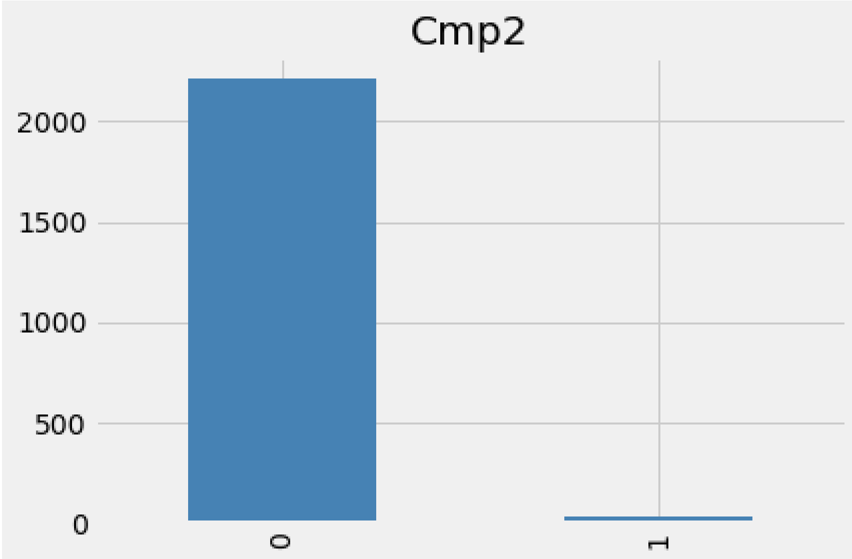

estado_civil
800
600
400
200
0
Married
Together
Single
Divorced
Widow
Alone
YOLO
Absurd

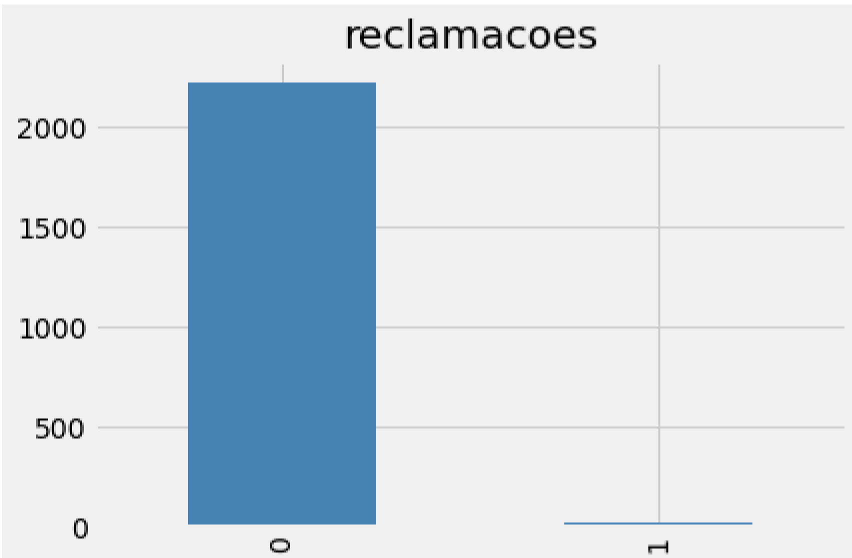

Cmp3
2000
1500
1000
500
0
0
1

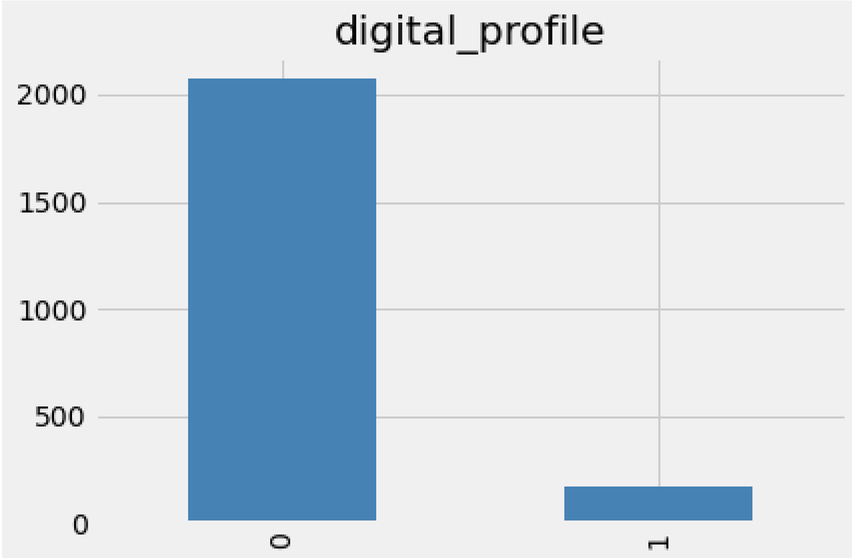

Cmp4
2000
1500
1000
500
0
0
1


Cmp5
2000
1500
1000
500
0
0
1


Cmp1
2000
1500
1000
500
0
0
1


Cmp2
2000
1500
1000
500
0
0
1

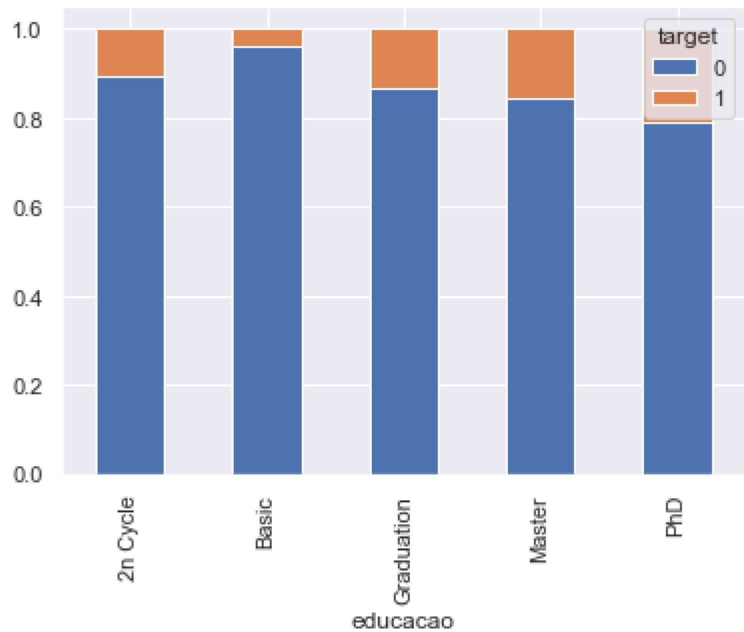

reclamacoes
2000
1500
1000
500
0
0
1

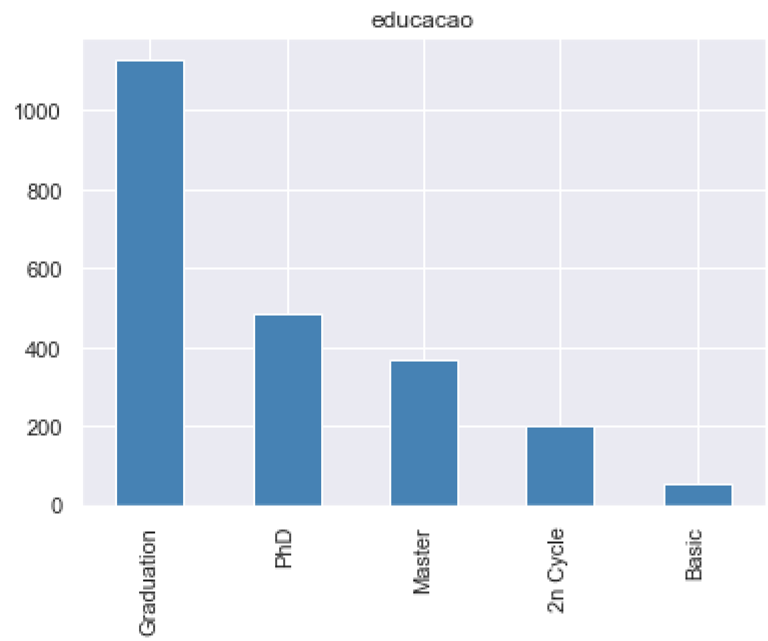

digital_profile
2000
1500
1000
500
0
0
1

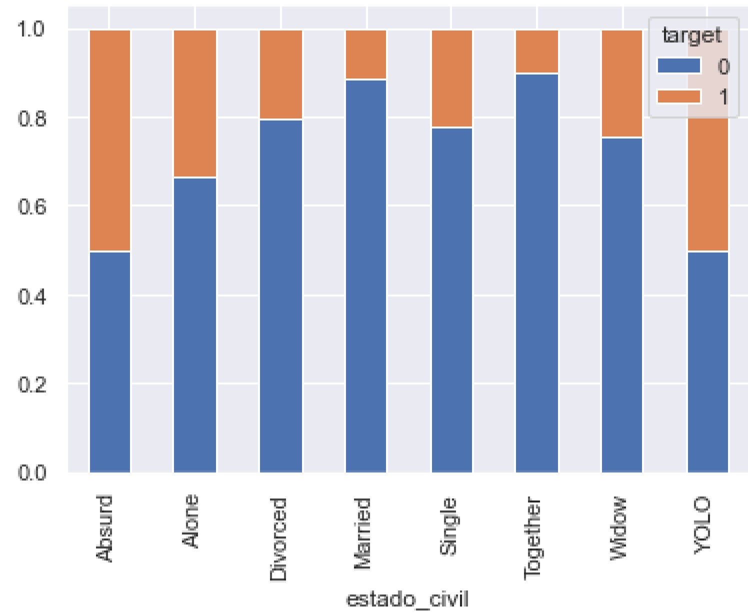
In [56]:
```
for attr in categorical:
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,4))
  outcome_counts = data.groupby([attr, 'target']).size().rename('count').reset_index()
  by_outcome = outcome_counts.pivot(columns='target', values='count', index=attr)
  # Plot the proportions
  by_outcome.div(by_outcome.sum(1), axis=0).plot.bar(stacked=True, ax=ax1);
  # Plot the counts
  data[attr].value_counts().plot.bar(ax=ax2, legend=False,color='steelblue');
  print('Support (%s)\n' % attr)
  print(data[attr].value_counts(), '\n')
  plt.title(attr);
```

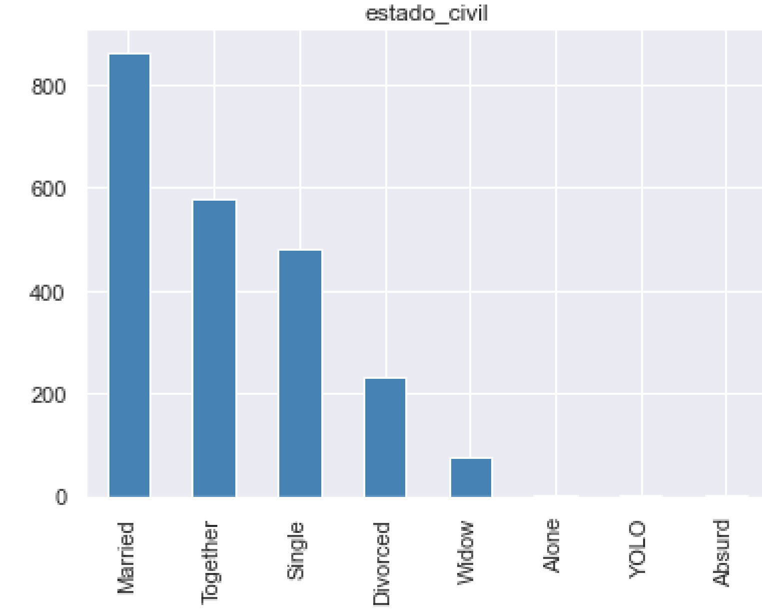
```
Support (educacao)

Graduation    1127
PhD            486
Master         370
2n Cycle       203
Basic           54
Name: educacao, dtype: int64

Support (estado_civil)

Married     864
Together    580
Single      480
Divorced    232
Widow        77
Alone         3
YOLO          2
Absurd        2
Name: estado_civil, dtype: int64

Support (Cmp3)

0    2077
1     163
Name: Cmp3, dtype: int64

Support (Cmp4)

0    2073
1     167
Name: Cmp4, dtype: int64

Support (Cmp5)

0    2077
1     163
Name: Cmp5, dtype: int64

Support (Cmp1)

0    2096
1     144
Name: Cmp1, dtype: int64

Support (Cmp2)
```

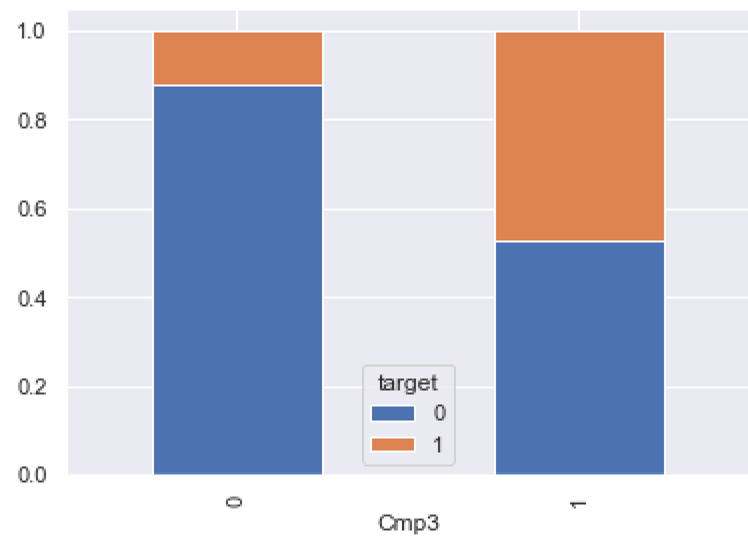
```
0    2210
1      30
Name: Cmp2, dtype: int64

Support (reclamacoes)

0    2219
1      21
Name: reclamacoes, dtype: int64

Support (digital_profile)

0    2071
1     169
Name: digital_profile, dtype: int64
```

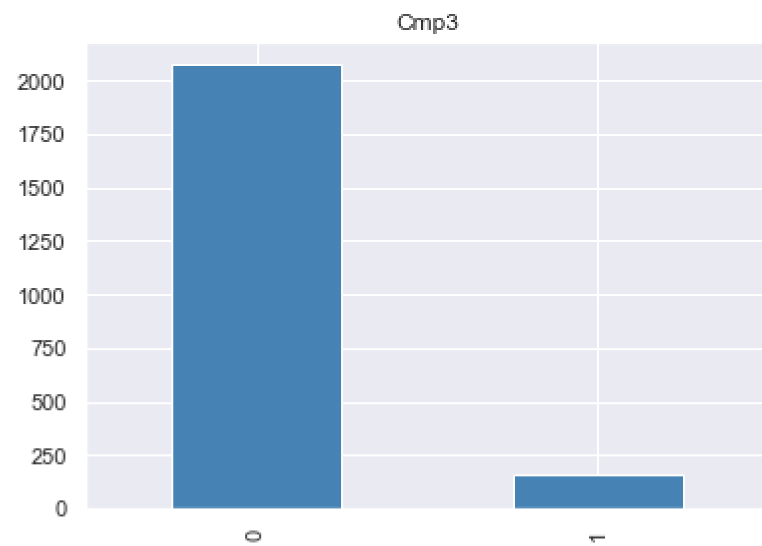


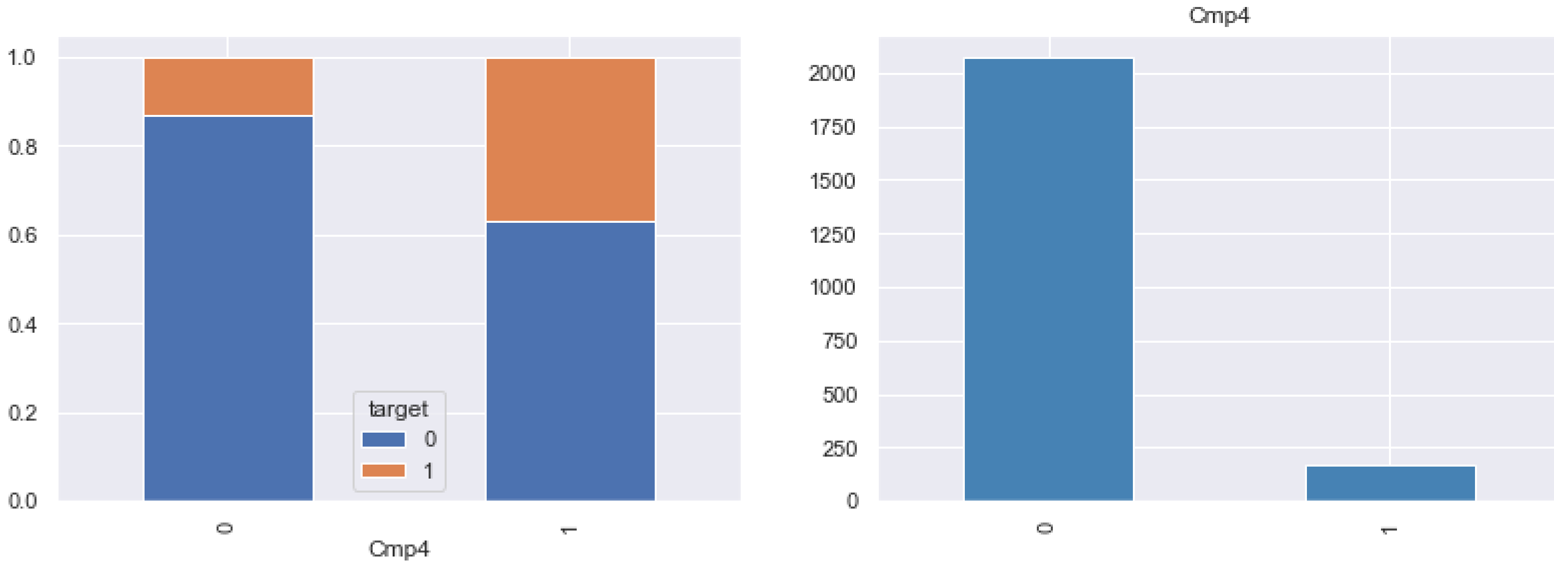


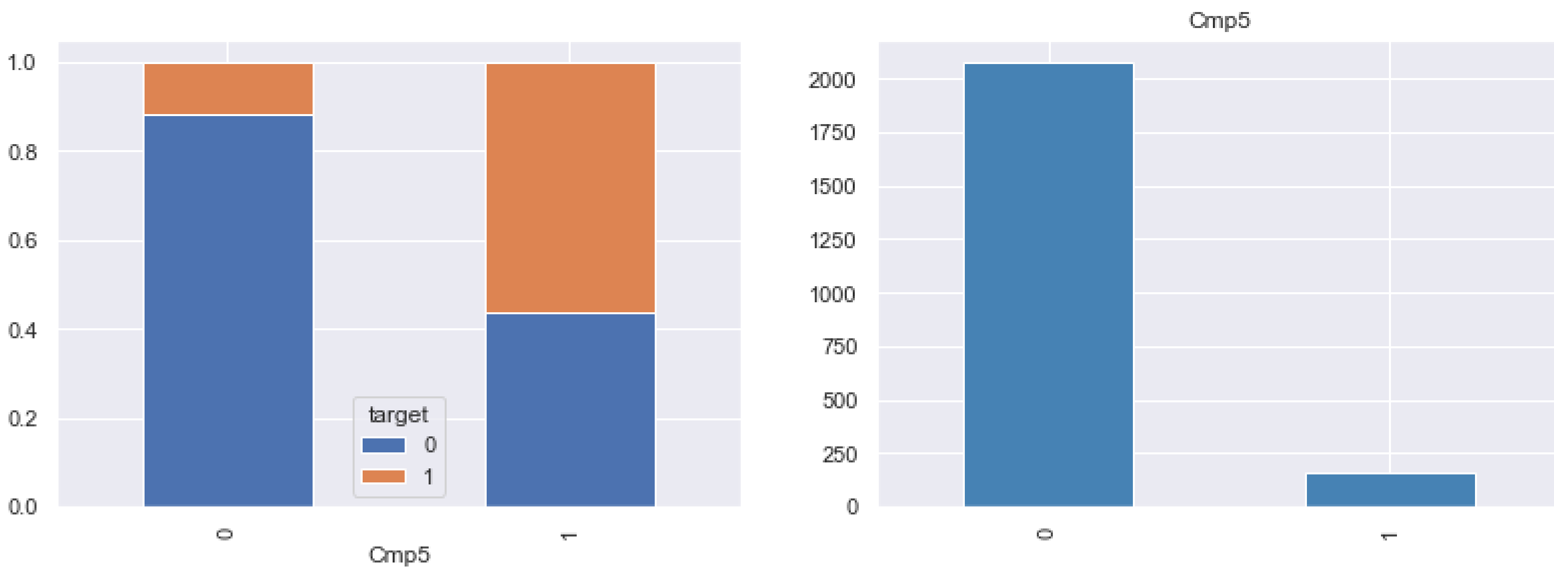

target
0
1
1.0
0.8
0.6
0.4
0.2
0.0
Absurd
Alone
Divorced
Married
Single
Together
Widow
YOLO
estado_civil

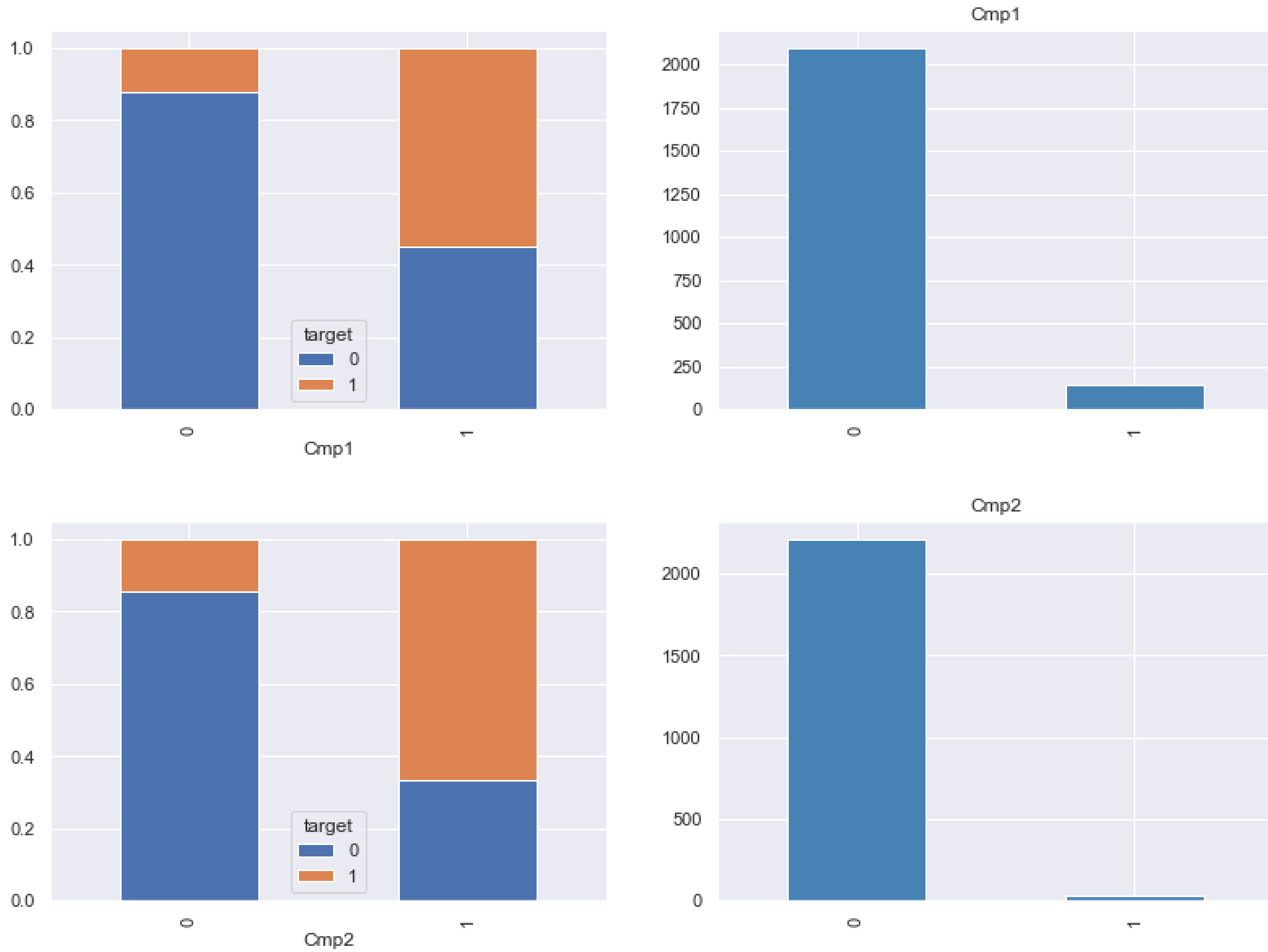

estado_civil
800
600
400
200
0
Married
Together
Single
Divorced
Widow
Alone
YOLO
Absurd

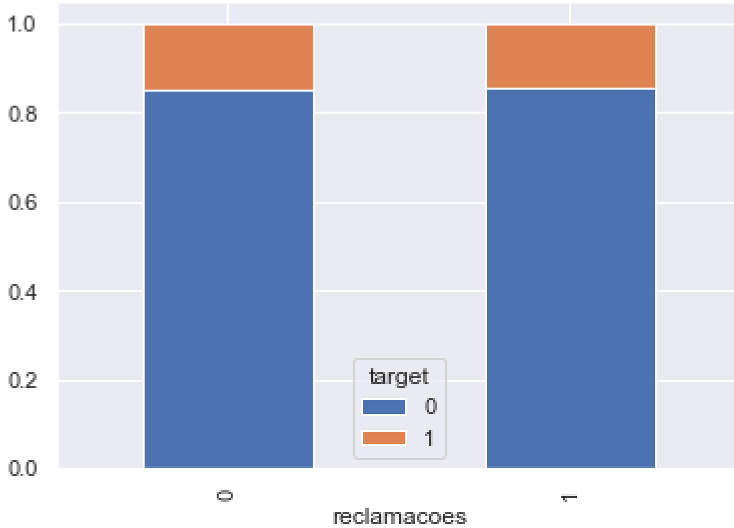

1.0
0.8
0.6
0.4
0.2
0.0
target
0
1
0
1
Cmp3

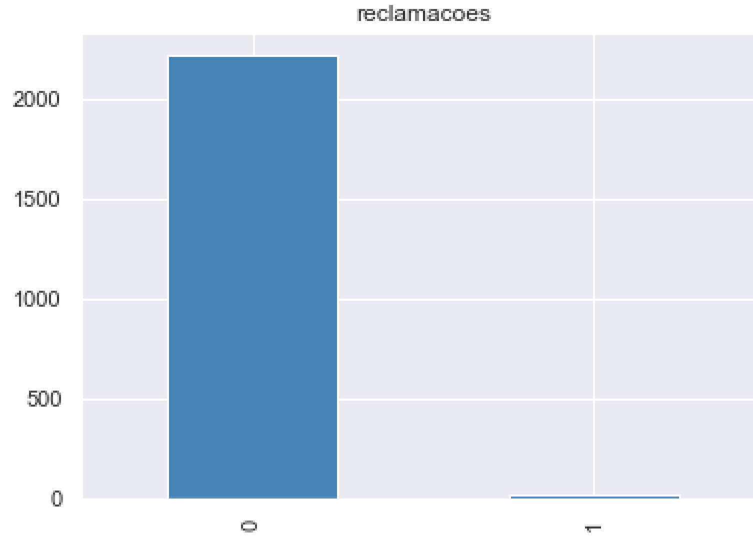

Cmp3
2000
1750
1500
1250
1000
750
500
250
0
0
1

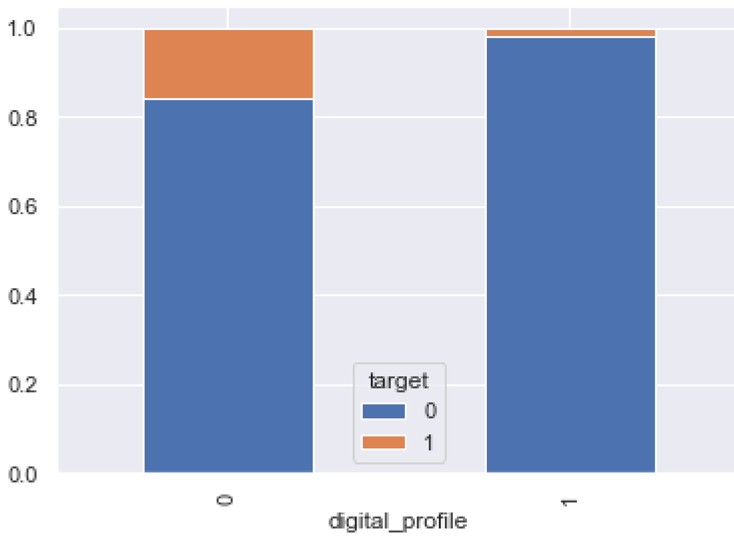

Cmp4
target
0
1
Cmp4
1.0
0.8
0.6
0.4
0.2
0.0
2000
1750
1500
1250
1000
750
500
250
0
0
1

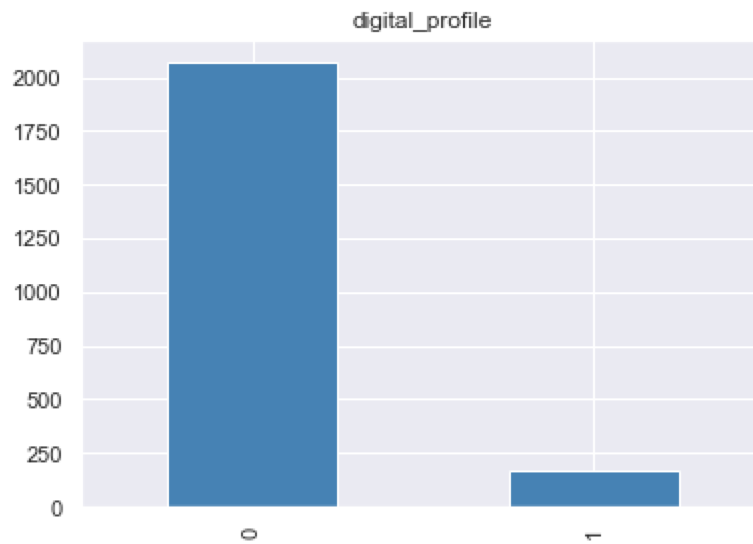

Cmp5
target
0
1
Cmp5
1.0
0.8
0.6
0.4
0.2
0.0
2000
1750
1500
1250
1000
750
500
250
0
0
1


Cmp1
target
0
1
Cmp2
target
0
1
Cmp1
Cmp2





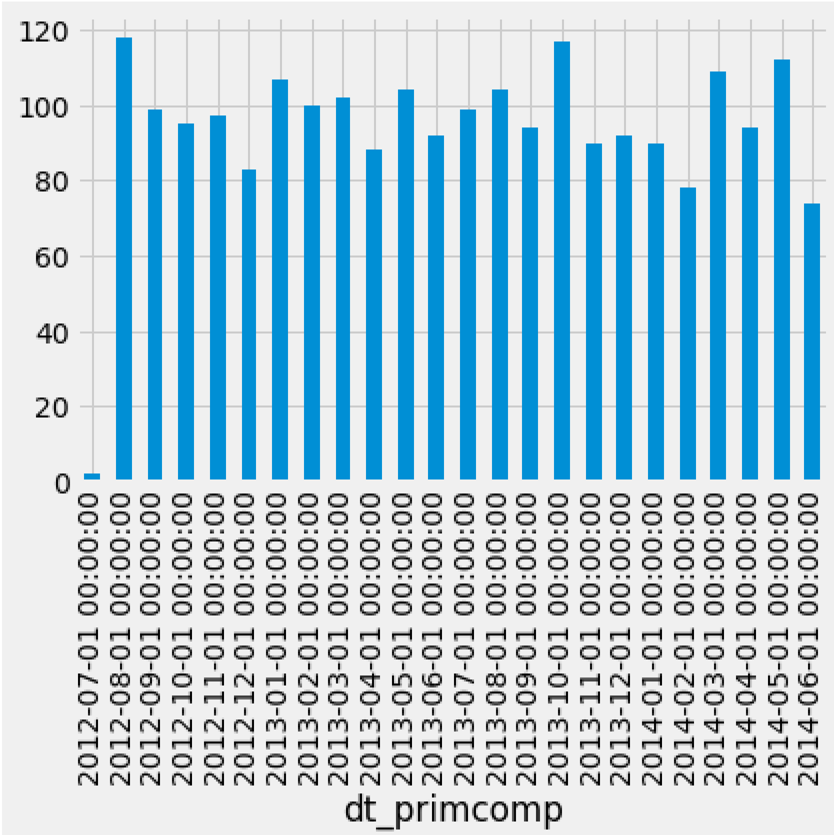




Date Analysis

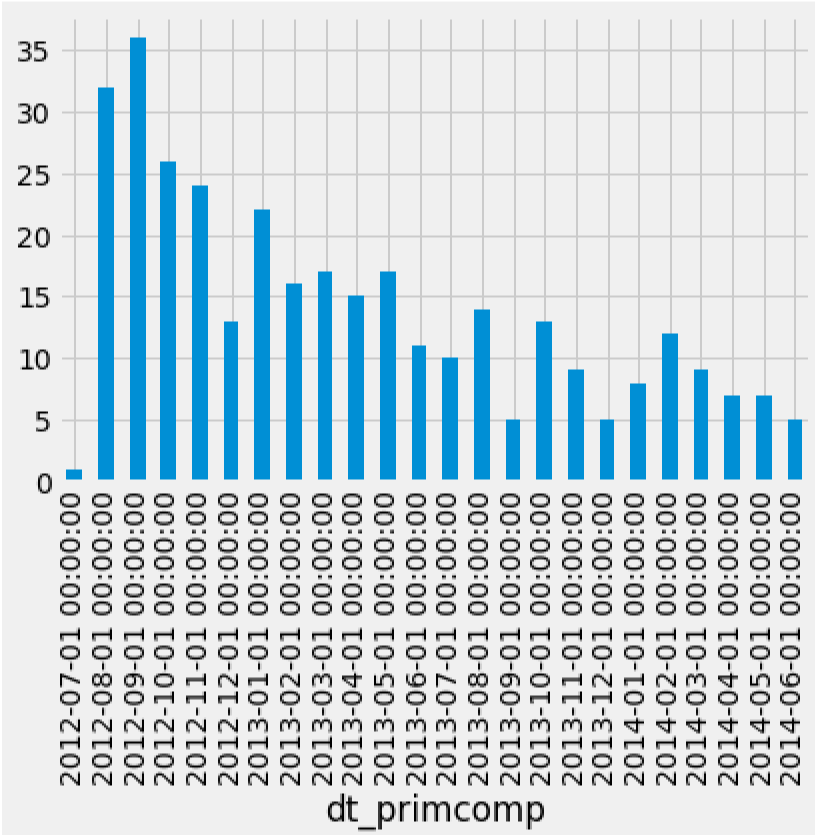
In [18]:

```
data['dt_primcomp'] = pd.to_datetime(data['dt_primcomp'], infer_datetime_format=True)
data.groupby('dt_primcomp')['ID'].nunique().plot(kind='bar')
plt.show()
```



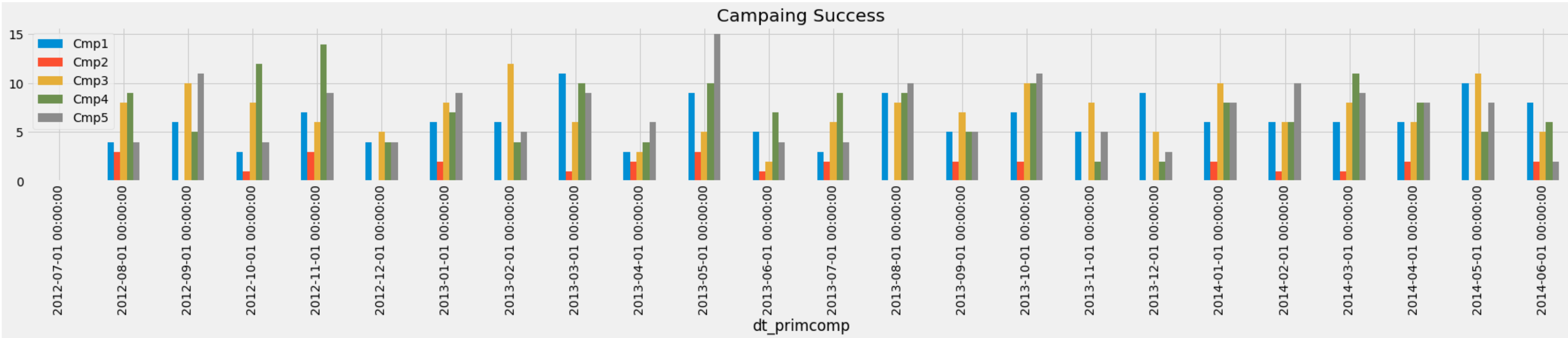
In [21]:

```
data['target'] = data['target'].astype(int)
data.groupby('dt_primcomp')['target'].sum().plot(kind='bar')
plt.show()
```



In [280]:
```python
# campaing
pylab.rcParams['figure.figsize'] = (28, 3)
data.groupby(('dt_primcomp'))['Cmp1','Cmp2','Cmp3','Cmp4','Cmp5'].sum().plot(kind='bar')
plt.title("Campaing Success")
plt.figure( figsize=(20, 18))
plt.show()
```

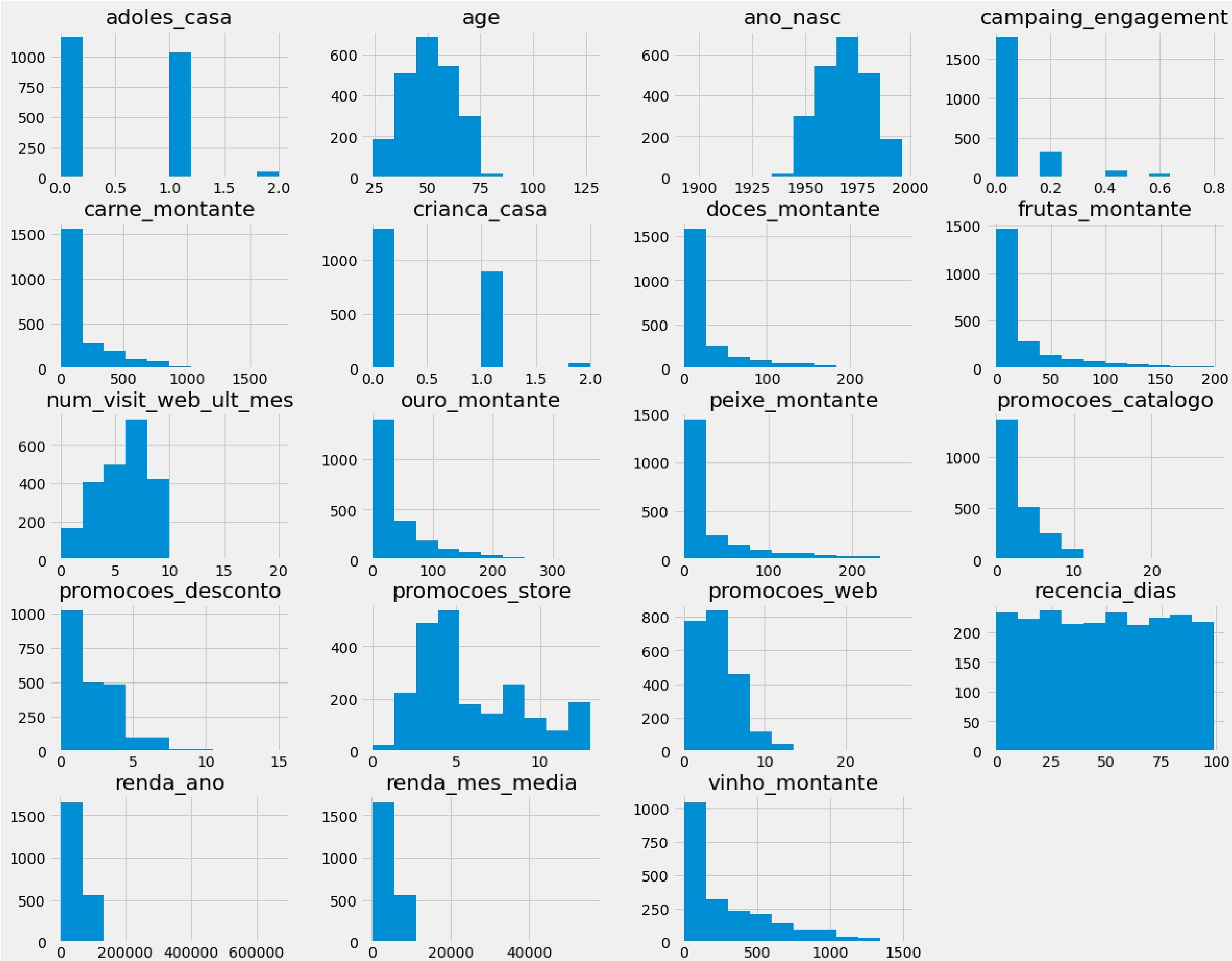
```
C:\Users\patri\anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: Indexing with multiple keys (implic
itly converted to a tuple of keys) will be deprecated, use a list instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```




```
<Figure size 1440x1296 with 0 Axes>
```

Numerical Data Analysis

In [22]:
```
data[numeric].hist(figsize=(18,15));
```



In [23]:

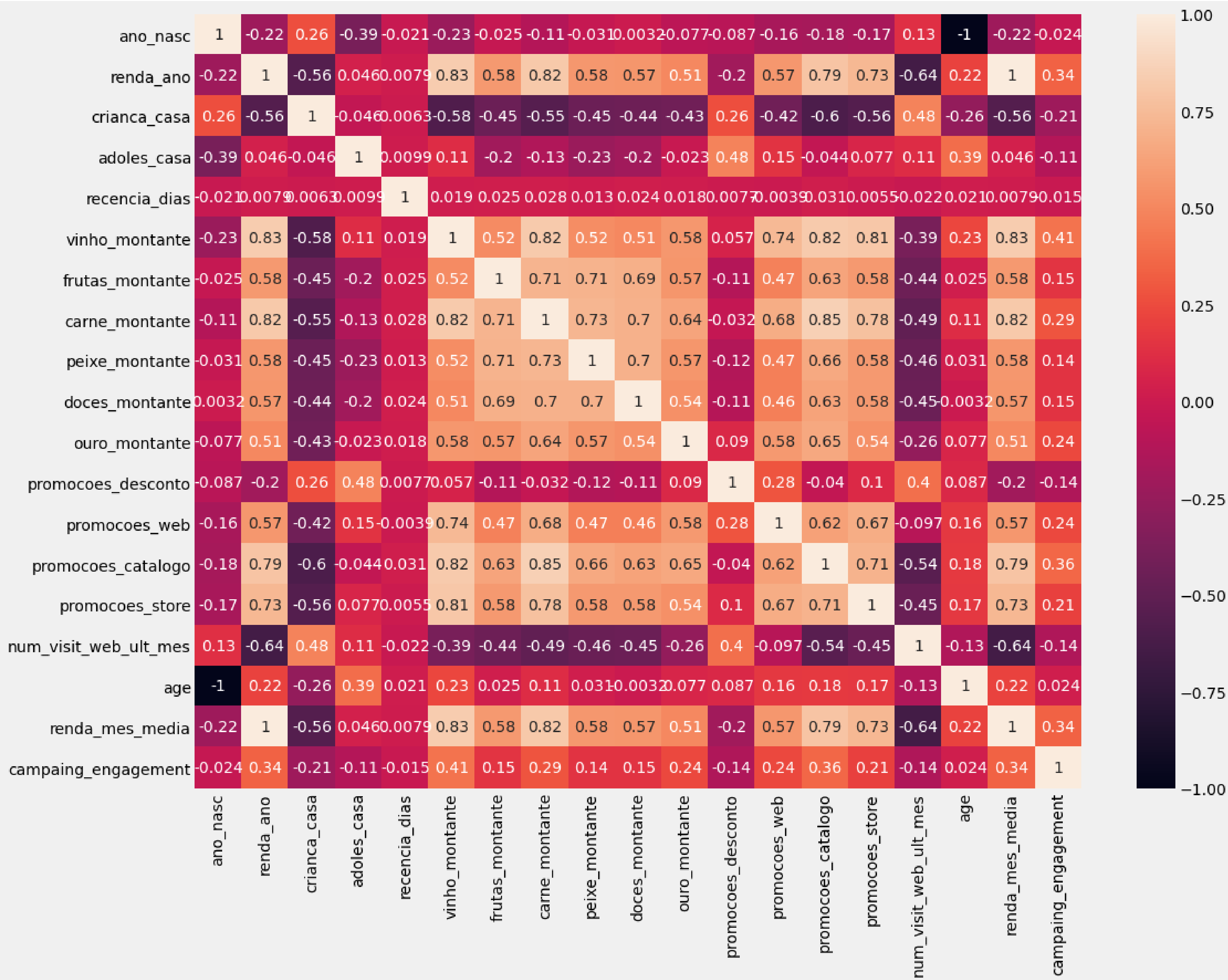
```
data[numeric].describe()
```

Out[23]:

	ano_nasc	renda_ano	crianca_casa	adoles_casa	recencia_dias	vinho_montante	frutas_montante	carne_montante	peixe_montante	c
count	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	
mean	1968.805804	52247.251354	0.444196	0.506250	49.109375	303.935714	26.302232	166.950000	37.525446	
std	11.984069	25173.076661	0.538398	0.544538	28.962453	336.597393	39.773434	225.715373	54.628979	
min	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1959.000000	35303.000000	0.000000	0.000000	24.000000	23.750000	1.000000	16.000000	3.000000	
50%	1970.000000	51381.500000	0.000000	0.000000	49.000000	173.500000	8.000000	67.000000	12.000000	
75%	1977.000000	68522.000000	1.000000	1.000000	74.000000	504.250000	33.000000	232.000000	50.000000	
max	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	259.000000	

In [24]:

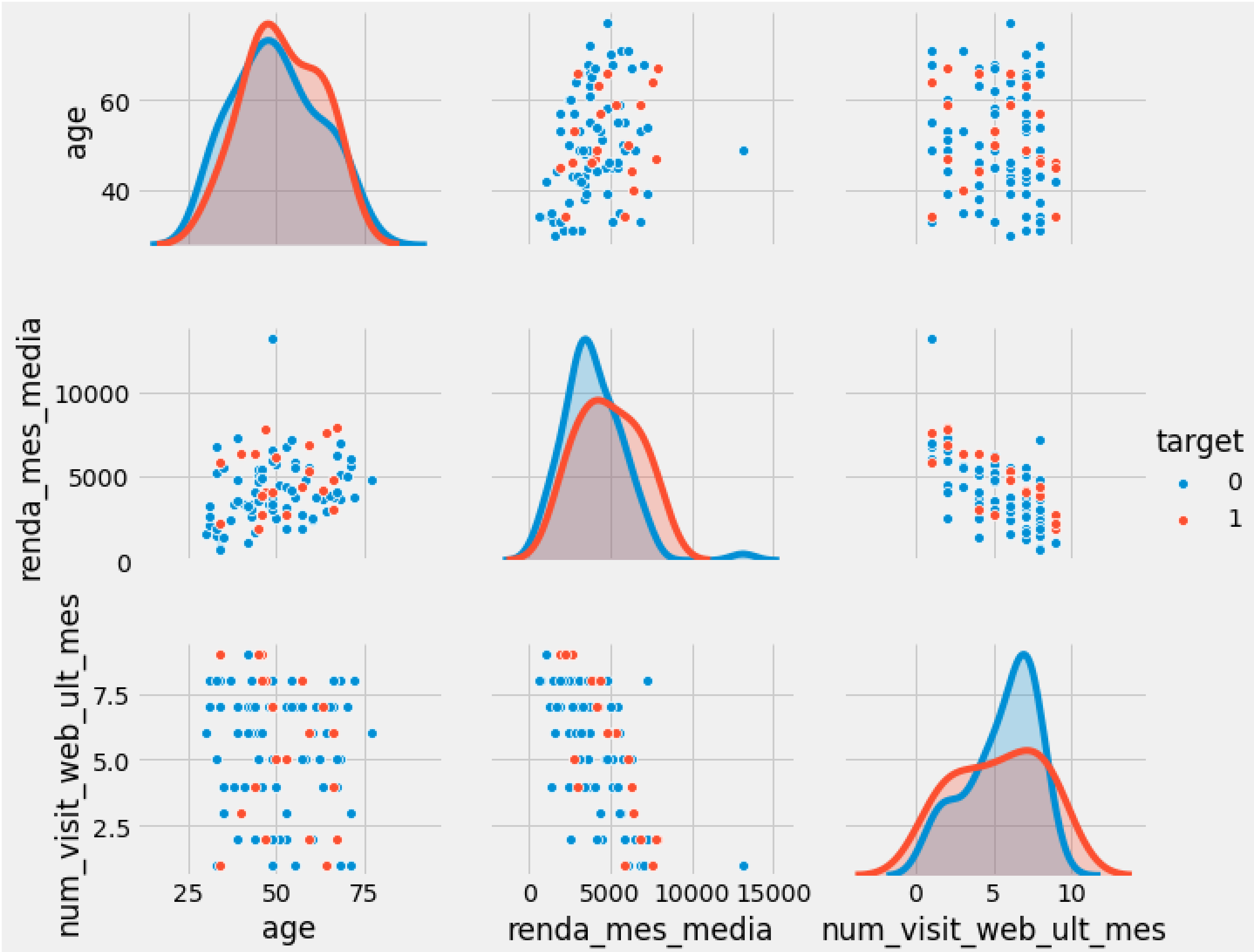
```
plt.figure(figsize=(16,12));
sns.heatmap(data[numeric].corr('spearman'), annot=True);
```



Customer Attributes

In [37]:
```
cust_attrs = ['age', 'renda_mes_media', 'num_visit_web_ult_mes','target']
```

In [38]:
```
data['target'] = data['target'].astype(str)
numeric_outcome = pd.concat([data[numeric], data['target']], axis=1)
sns.pairplot(numeric_outcome[cust_attrs].sample(n=100), hue='target', aspect=1.2);
```



Clustering

1 . Kmeans

In [14]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

In [16]:
```
X=data[['renda_ano', 'crianca_casa', 'adoles_casa',  'recencia_dias', 'vinho_montante', 'frutas_montante', 'carne_monta
nte', 'peixe_montante', 'doces_montante', 'ouro_montante', 'promocoes_desconto', 'promocoes_web', 'promocoes_catalogo',
'promocoes_store', 'num_visit_web_ult_mes','age', 'renda_mes_media','campaing_engagement']]
```

In [17]:
```
X=X.fillna(0)
```

In [48]:
```
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(X)
print(scaled)
```

```
[[ 0.25193856 -0.82521765 -0.92989438 ...  0.98534473  0.25193856
  -0.43903713]
 [-0.20869932  1.03255877  0.90693402 ...  1.23573295 -0.20869932
  -0.43903713]
 [ 0.77823121 -0.82521765 -0.92989438 ...  0.3176428   0.77823121
  -0.43903713]
 ...
 [ 0.20674965 -0.82521765 -0.92989438 ... -1.01776106  0.20674965
   1.03539042]
 [ 0.68574431 -0.82521765  0.90693402 ...  1.06880747  0.68574431
  -0.43903713]
 [ 0.04614739  1.03255877  0.90693402 ...  1.23573295  0.04614739
  -0.43903713]]
```

In [49]:
```
for n_clusters in range(3, 10):
    kmeans = KMeans(init='k-means++', n_clusters = n_clusters, n_init = 30)
    kmeans.fit(scaled)
    clusters = kmeans.predict(scaled)
    sil_avg = silhouette_score(scaled, clusters)
    print("For n_clusters : ", n_clusters, "The average silhouette_score is : ", sil_avg)
```

```
For n_clusters :  3 The average silhouette_score is :  0.2261032879931275
For n_clusters :  4 The average silhouette_score is :  0.15081907973264772
For n_clusters :  5 The average silhouette_score is :  0.15246532108308206
For n_clusters :  6 The average silhouette_score is :  0.15273556967922475
For n_clusters :  7 The average silhouette_score is :  0.15382971566043036
For n_clusters :  8 The average silhouette_score is :  0.15351880625259764
For n_clusters :  9 The average silhouette_score is :  0.14400645664857872
```

In [54]:
```
# Choosing number of clusters as 3:
# Trying Improving the silhouette_score :
n_clusters = 3
sil_avg = -1
while sil_avg < 0.145:
    kmeans = KMeans(init = 'k-means++', n_clusters = n_clusters, n_init = 30)
    kmeans.fit(scaled)
    clusters = kmeans.predict(scaled)
    sil_avg = silhouette_score(scaled, clusters)
    print("For n_clusters : ", n_clusters, "The average silhouette_score is : ", sil_avg)
```

```
For n_clusters :  3 The average silhouette_score is :  0.22575793720918896
```

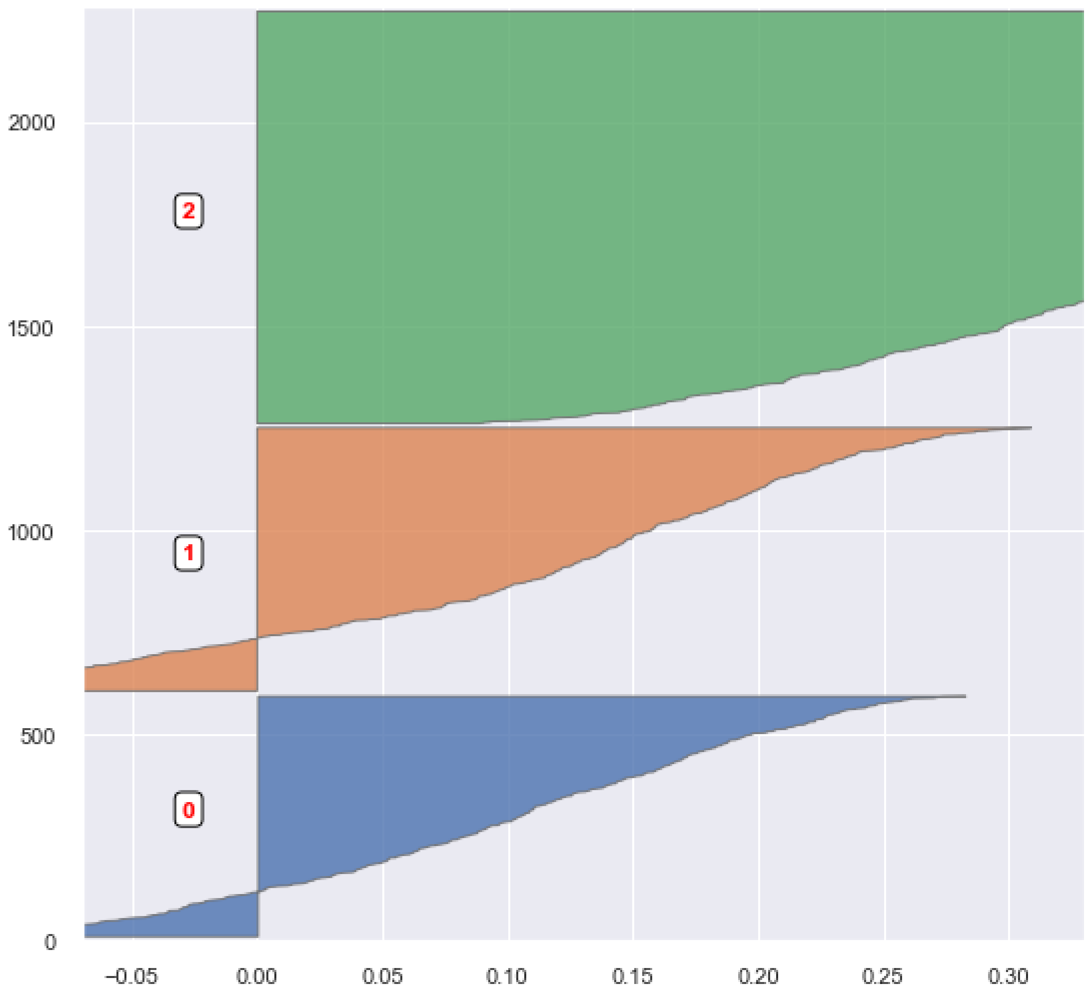
In [55]:
```
# Printing number of elements in each cluster :
pd.Series(clusters).value_counts()
```

Out[55]:
```
2    1008
1     644
0     588
dtype: int64
```

Analysing 3 Cluster

In [56]:
```
def graph_component_silhouette(n_clusters, lim_x, mat_size, sample_silhouette_values, clusters):
    import matplotlib as mpl
    mpl.rc('patch', edgecolor = 'dimgray', linewidth = 1)

    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(8, 8)
    ax1.set_xlim([lim_x[0], lim_x[1]])
    ax1.set_ylim([0, mat_size + (n_clusters + 1) * 10])
    y_lower = 10

    for i in range(n_clusters):
        ith_cluster_silhoutte_values = sample_silhouette_values[clusters == i]
        ith_cluster_silhoutte_values.sort()
        size_cluster_i = ith_cluster_silhoutte_values.shape[0]
        y_upper = y_lower + size_cluster_i

        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhoutte_values, alpha = 0.8)

        ax1.text(-0.03, y_lower + 0.5 * size_cluster_i, str(i), color = 'red', fontweight = 'bold',
                 bbox = dict(facecolor = 'white', edgecolor = 'black', boxstyle = 'round, pad = 0.3'))

        y_lower = y_upper + 10
```

```
In [57]: # Plotting the intra cluster silhouette distances.
         from sklearn.metrics import silhouette_samples
         sample_silhouette_values = silhouette_samples(scaled, clusters)
         graph_component_silhouette(n_clusters, [-0.07, 0.33], len(X), sample_silhouette_values, clusters)
```



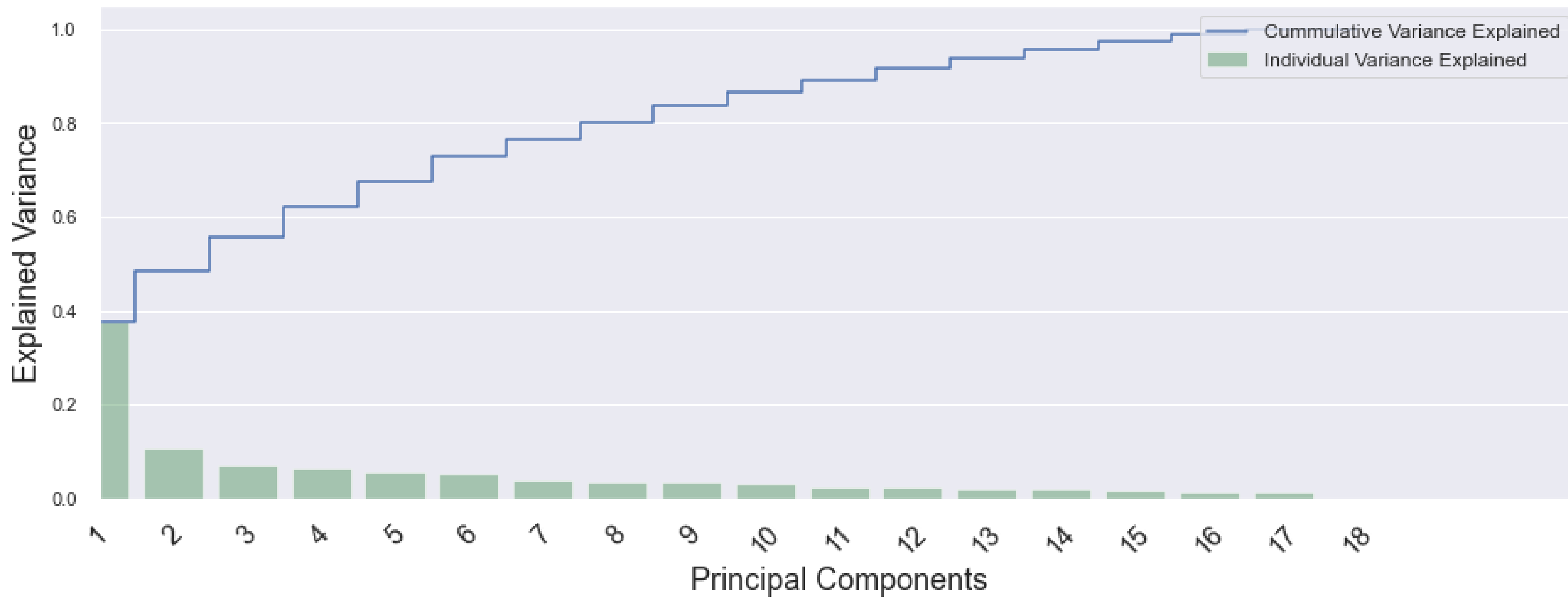
Dimensionality Analysis

PCA

```
In [58]: from sklearn.decomposition import PCA
```

In [59]:
```
pca = PCA()
pca.fit(scaled)
pca_samples = pca.transform(scaled)
```

In [60]:
```
# Checking the amount of variance explained :
fig, ax = plt.subplots(figsize=(14, 5))
sns.set(font_scale=1)
plt.step(range(scaled.shape[1]), pca.explained_variance_ratio_.cumsum(), where = 'mid', label = 'Cummulative Variance E
xplained')
sns.barplot(np.arange(1, scaled.shape[1] + 1), pca.explained_variance_ratio_, alpha = 0.5, color = 'g',
            label = 'Individual Variance Explained')
plt.xlim(0, 20)
plt.xticks(rotation = 45, fontsize = 16)
ax.set_xticklabels([s  for s in ax.get_xticklabels()])

plt.ylabel("Explained Variance", fontsize = 18)
plt.xlabel("Principal Components", fontsize = 18)
plt.legend(loc = 'upper right', fontsize = 12)
plt.show()
```



In [61]:
```
data['fit_segmentacao']= kmeans.labels_
```

In [62]:
```
data
```

Out[62]:

	ID	ano_nasc	educacao	estado_civil	renda_ano	crianca_casa	adoles_casa	dt_primcomp	recencia_dias	vinho_montante	frutas_monta
0	5524	1957	Graduation	Single	58138.0	0	0	09/2012	58	635	
1	2174	1954	Graduation	Single	46344.0	1	1	03/2014	38	11	
2	4141	1965	Graduation	Together	71613.0	0	0	08/2013	26	426	
3	6182	1984	Graduation	Together	26646.0	1	0	02/2014	26	11	
4	5324	1981	PhD	Married	58293.0	1	0	01/2014	94	173	
...	...	...	...	...	...	...	...	...	...	...	
2235	10870	1967	Graduation	Married	61223.0	0	1	06/2013	46	709	
2236	4001	1946	PhD	Together	64014.0	2	1	06/2014	56	406	
2237	7270	1981	Graduation	Divorced	56981.0	0	0	01/2014	91	908	
2238	8235	1956	Master	Together	69245.0	0	1	01/2014	8	428	
2239	9405	1954	PhD	Married	52869.0	1	1	10/2012	40	84	

2240 rows × 32 columns

In [63]:
```
data['fit_segmentacao'] = data['fit_segmentacao'].astype(str)
```

Fit Segmentação Analysis

In [64]:
```
from pandas_profiling import ProfileReport
profile = ProfileReport(data, title="Data Profile Report")
```

In [65]:
```
profile
```

Overview

Dataset statistics

Number of variables	32
Number of observations	2240
Missing cells	48
Missing cells (%)	0.1%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	560.1 KiB
Average record size in memory	256.1 B

Variable types

NUM	18
CAT	14

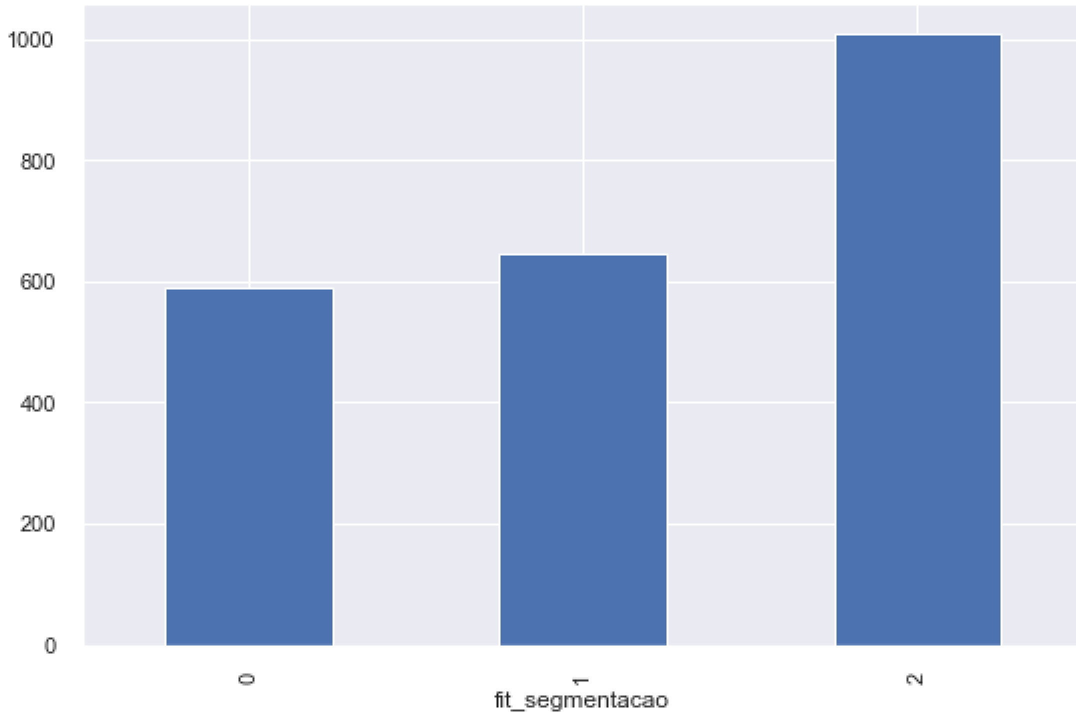
Reproduction

Analysis started	2020-07-21 00:54:36.254745
Analysis finished	2020-07-21 00:56:00.408575
Duration	1 minute and 24.15 seconds
Version	pandas-profiling v2.8.0 (https://github.com/pandas-profiling/pandas-profiling)
Command line	`pandas_profiling --config_file config.yaml [YOUR_FILE.csv]`

Out[65]:

In [188]:
```
# Customer
data.groupby(['fit_segmentacao']).fit_segmentacao.count().sort_values().plot(kind='bar')
data.groupby(['fit_segmentacao']).fit_segmentacao.count().sort_values()
```

Out[188]:
```
fit_segmentacao
0     588
1     644
2    1008
Name: fit_segmentacao, dtype: int64
```

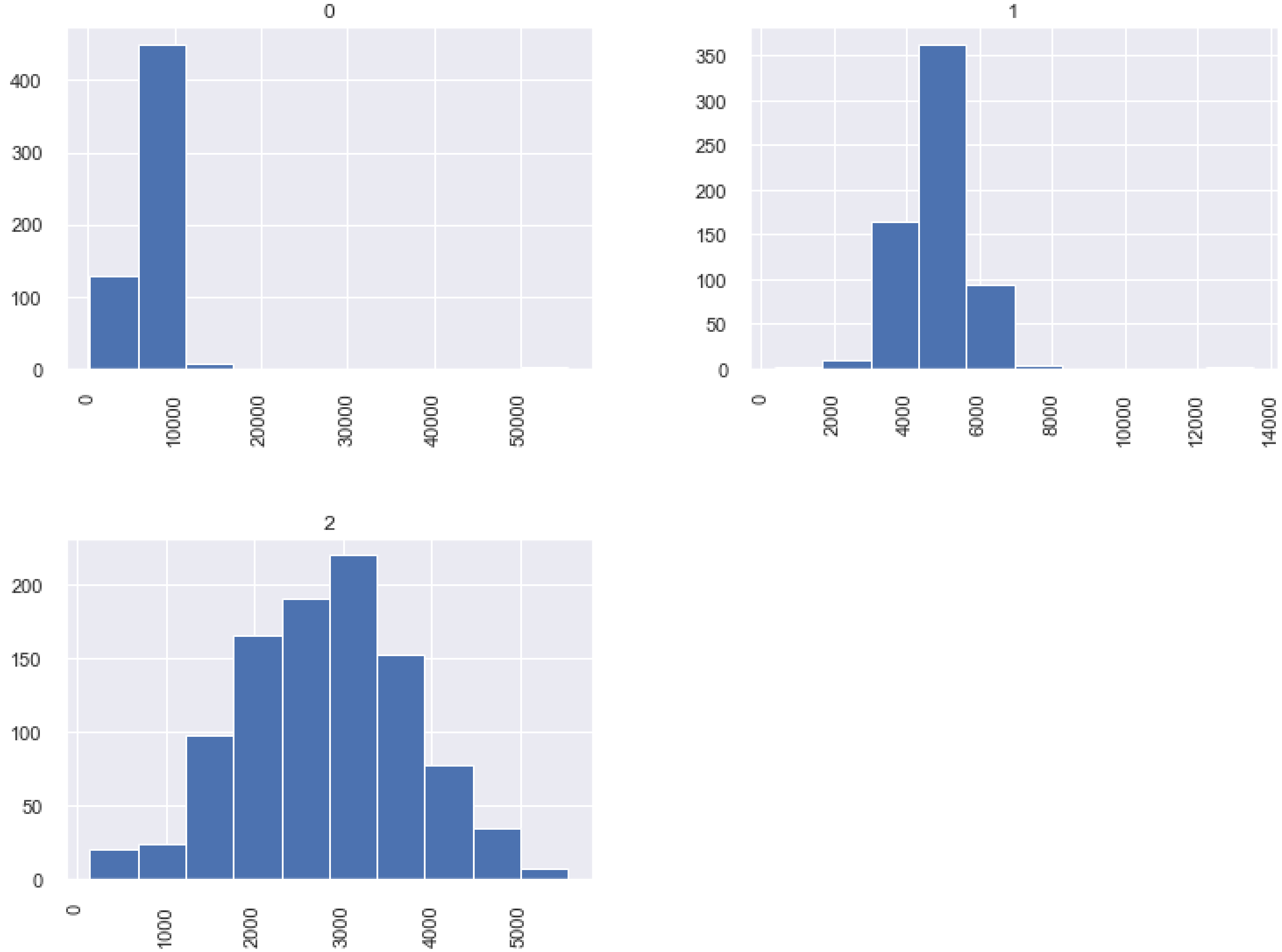


In [176]:
```
num_cluster = data[['renda_mes_media','age','crianca_casa','adoles_casa','recencia_dias','num_visit_web_ult_mes','campaing_engagement']]
```

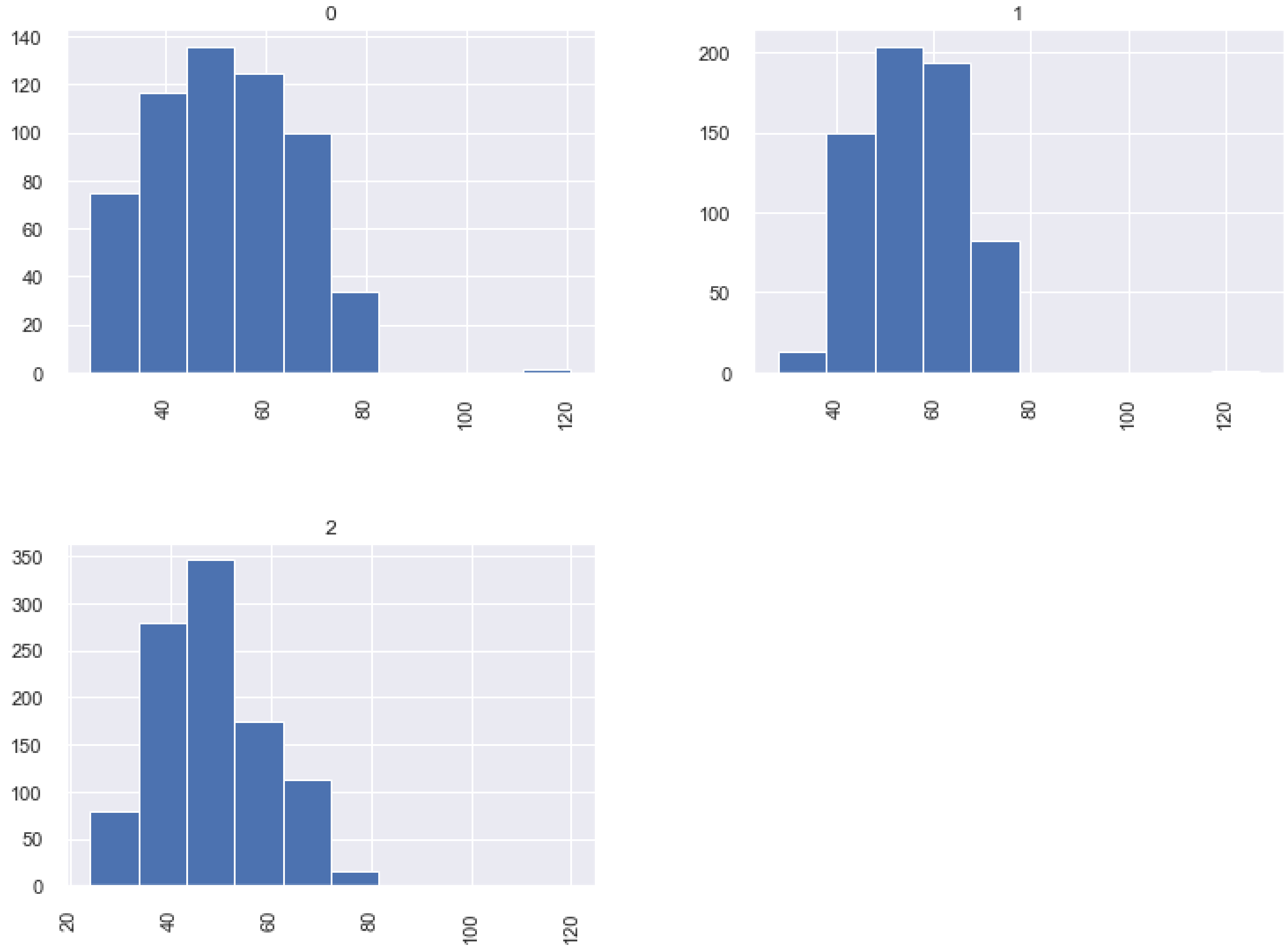
In [177]:

```
for att in num_cluster:
  figsize=(8,4)
  plt.figure()
  data[att].hist(by=data['fit_segmentacao'],figsize=(12,9))
  plt.title(att);
```

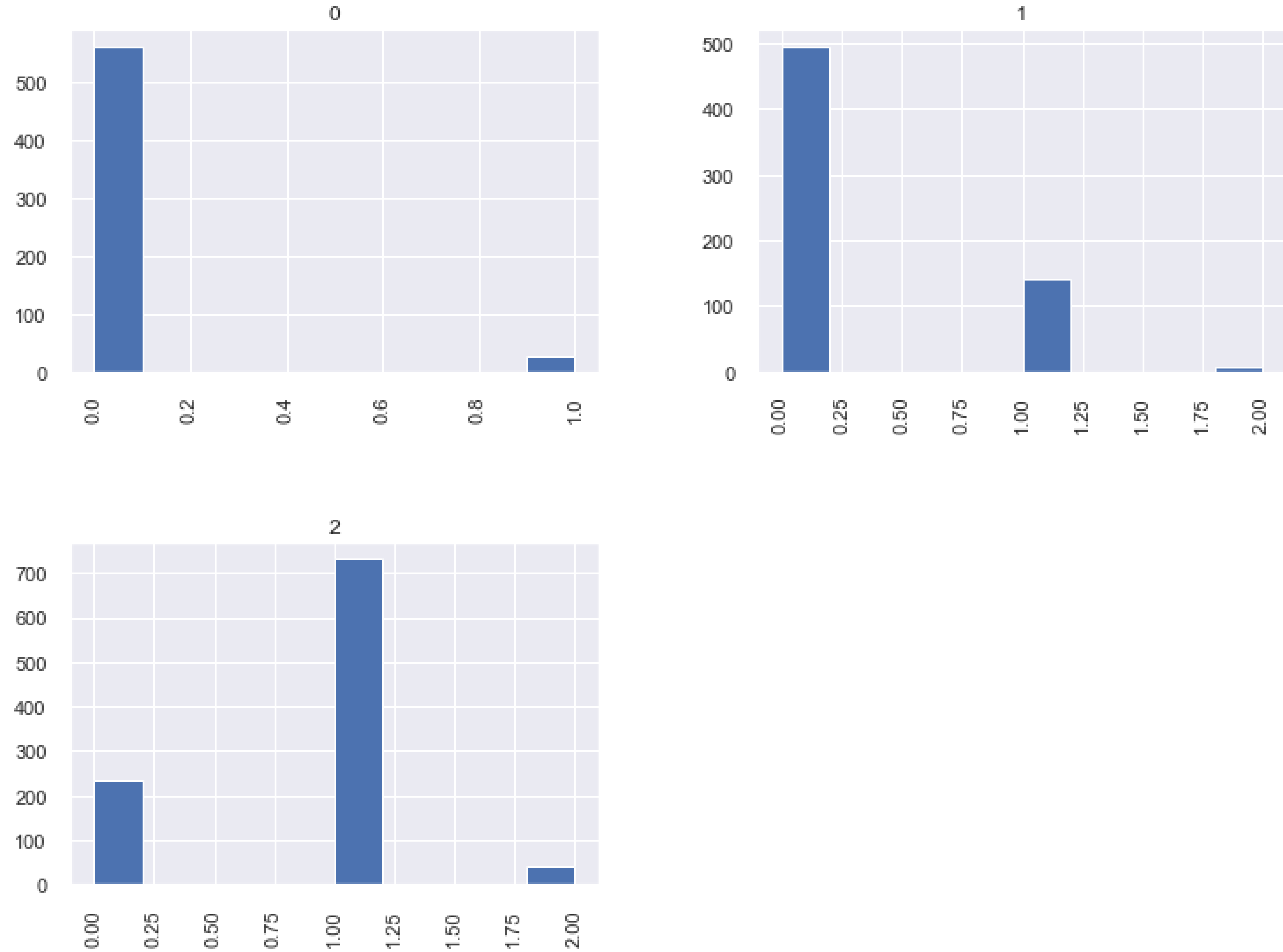
<Figure size 576x396 with 0 Axes>



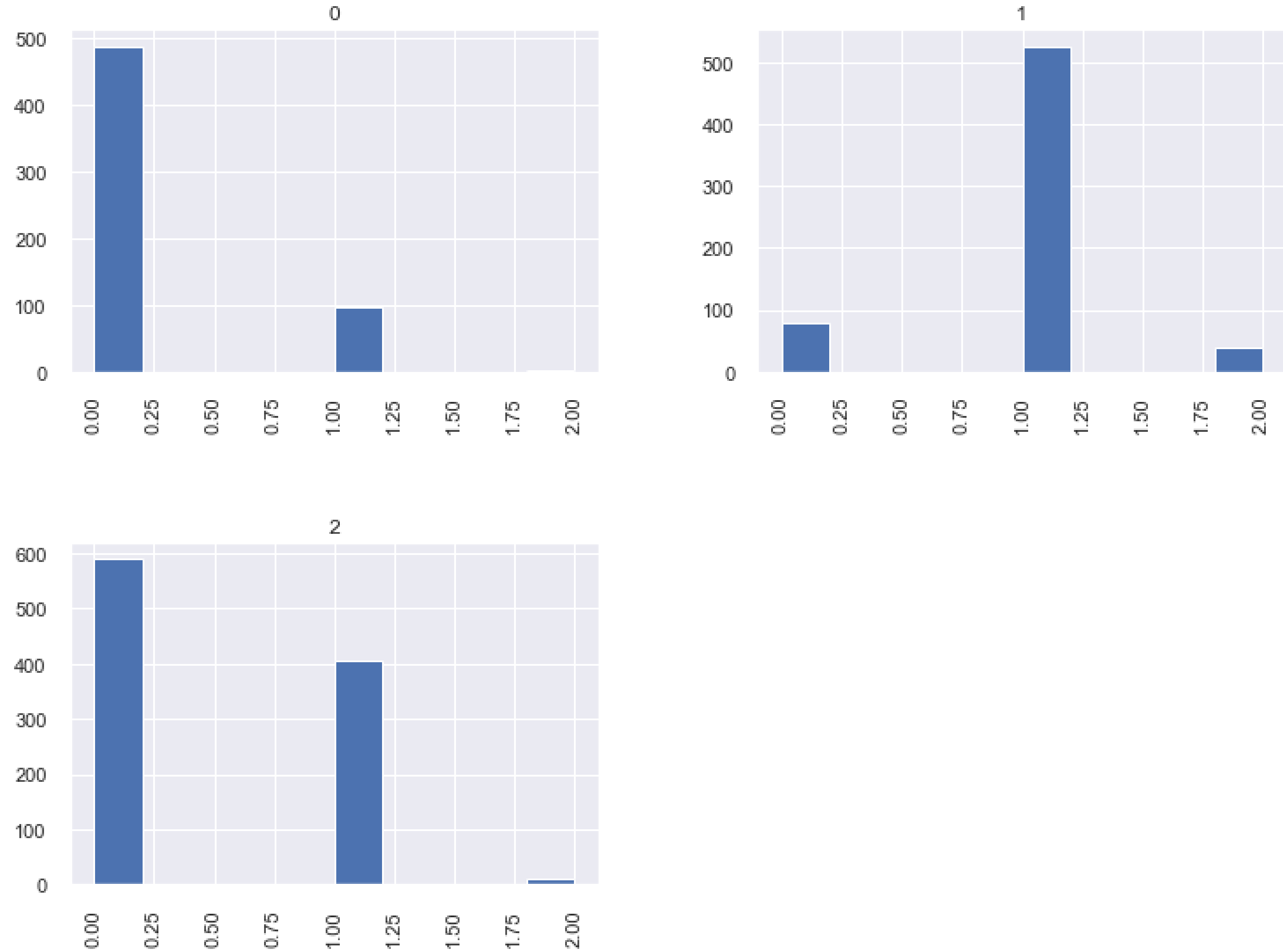
<Figure size 576x396 with 0 Axes>



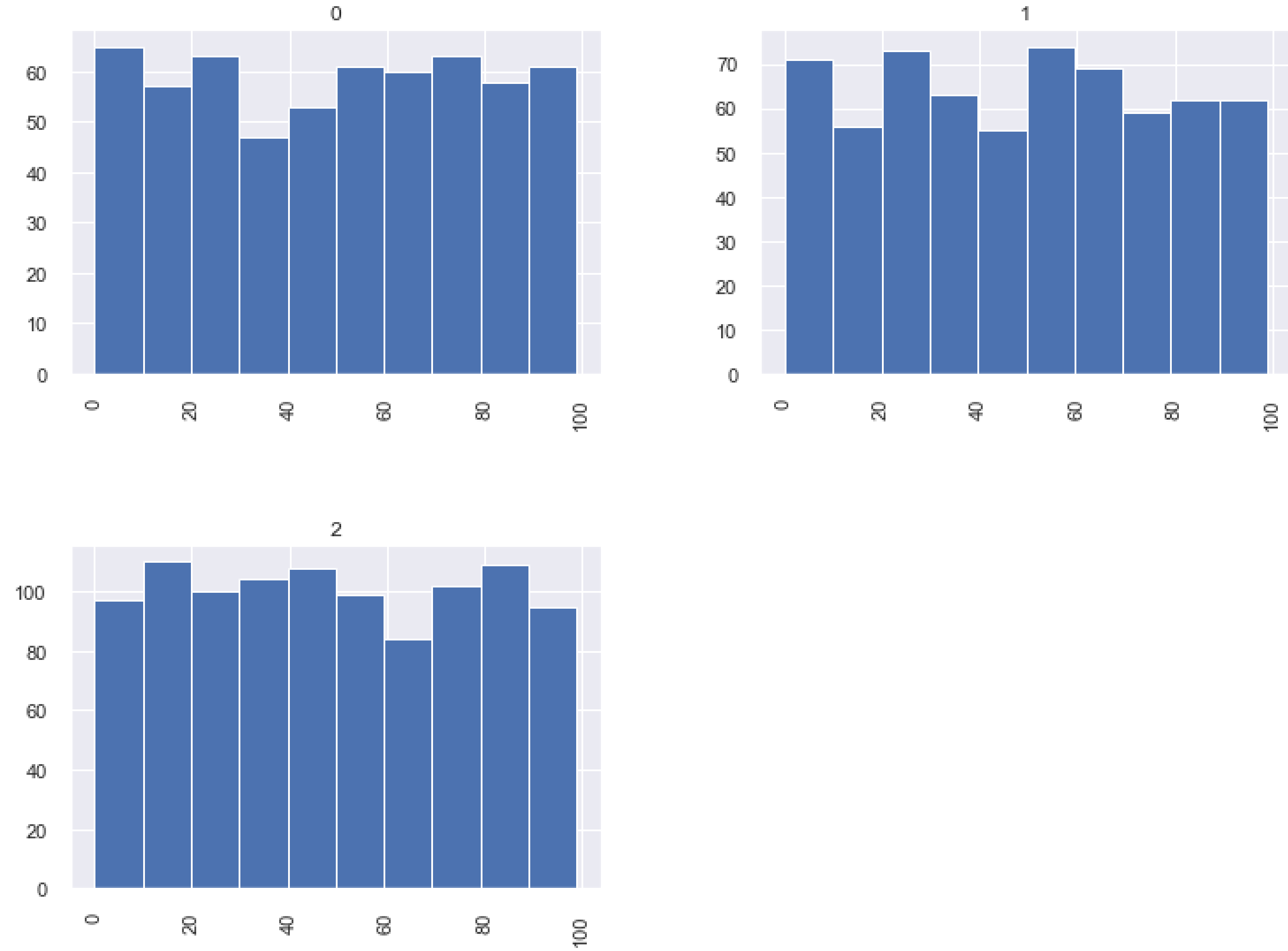
<Figure size 576x396 with 0 Axes>



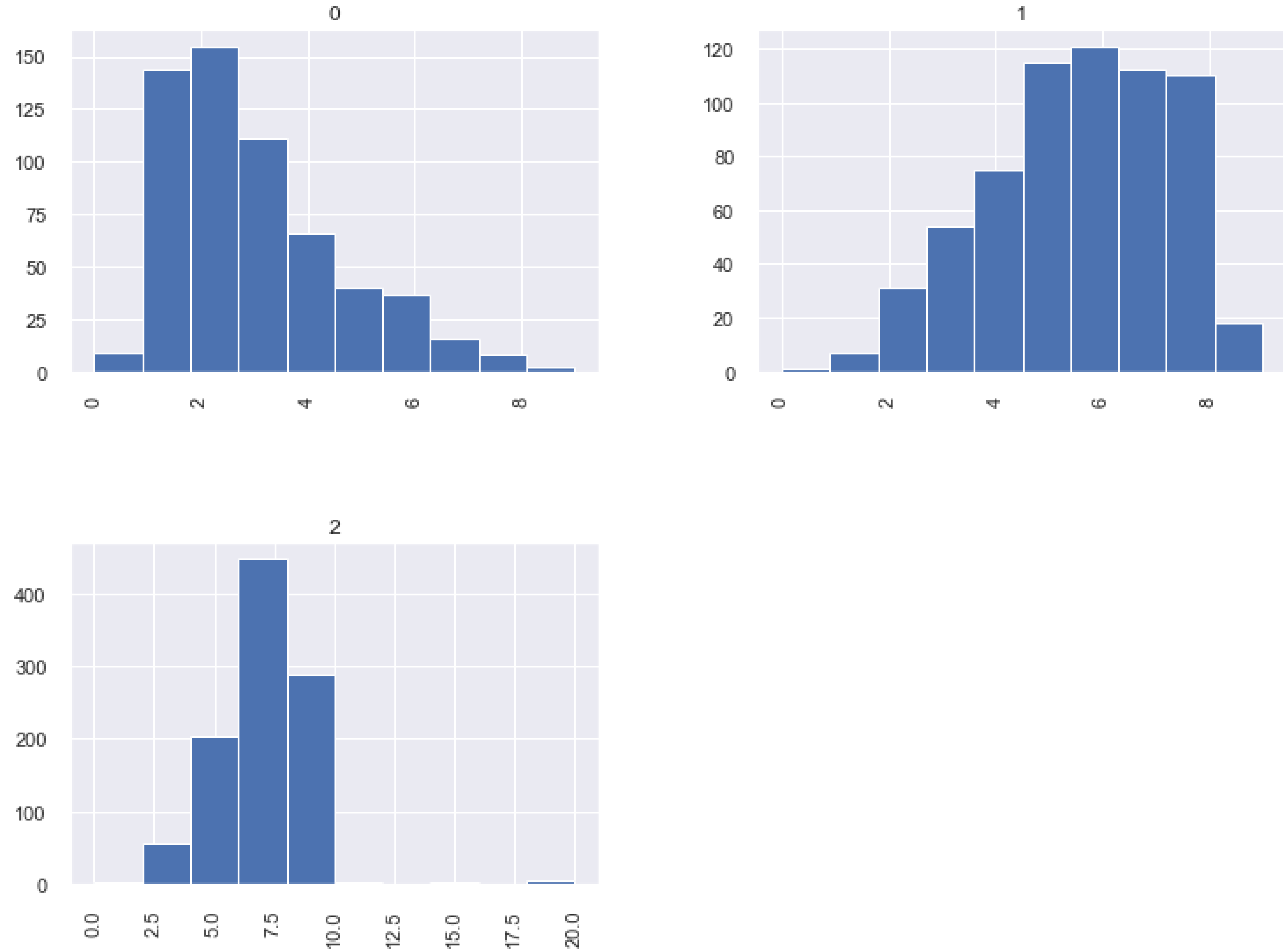
<Figure size 576x396 with 0 Axes>



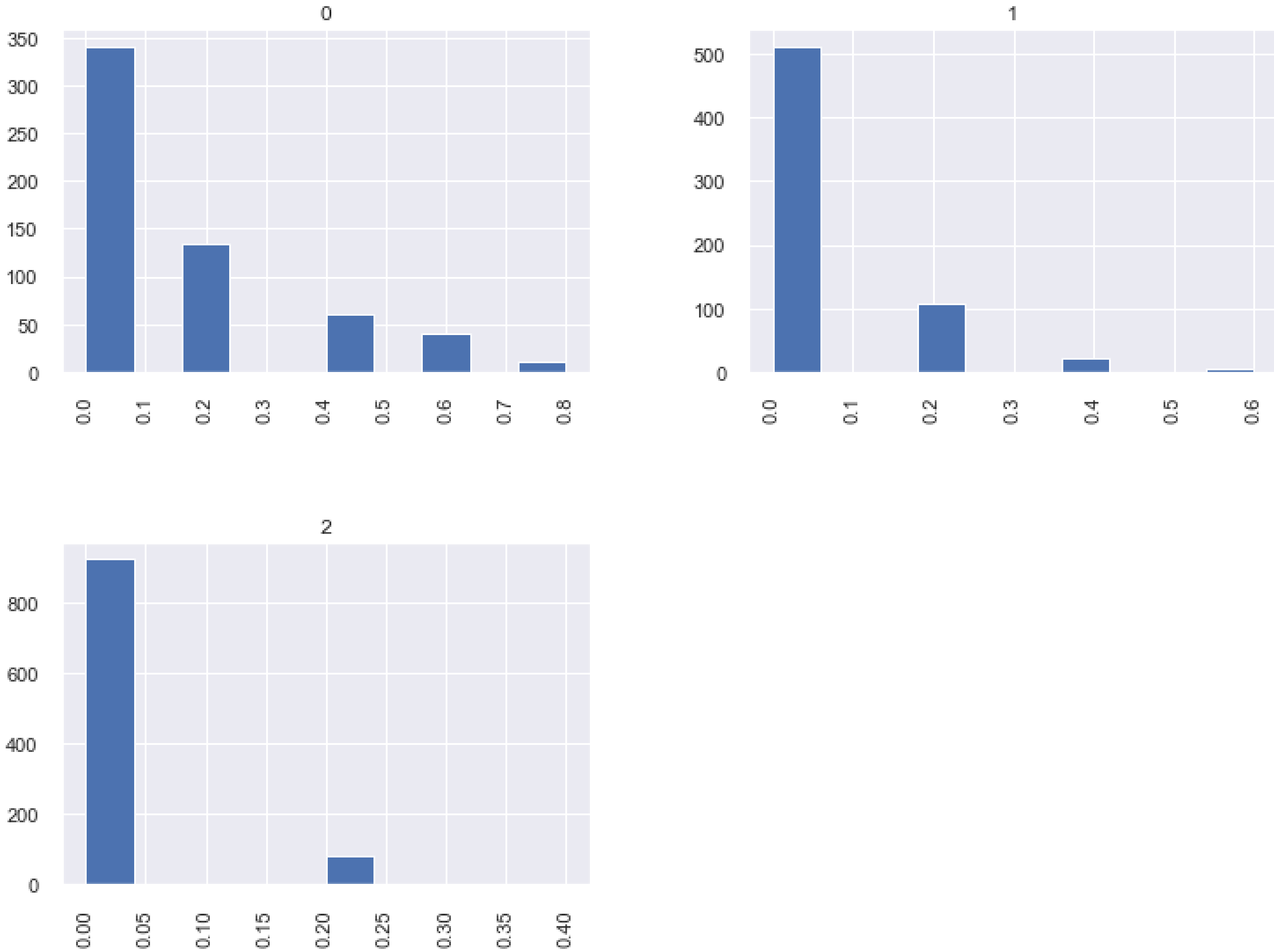
<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>

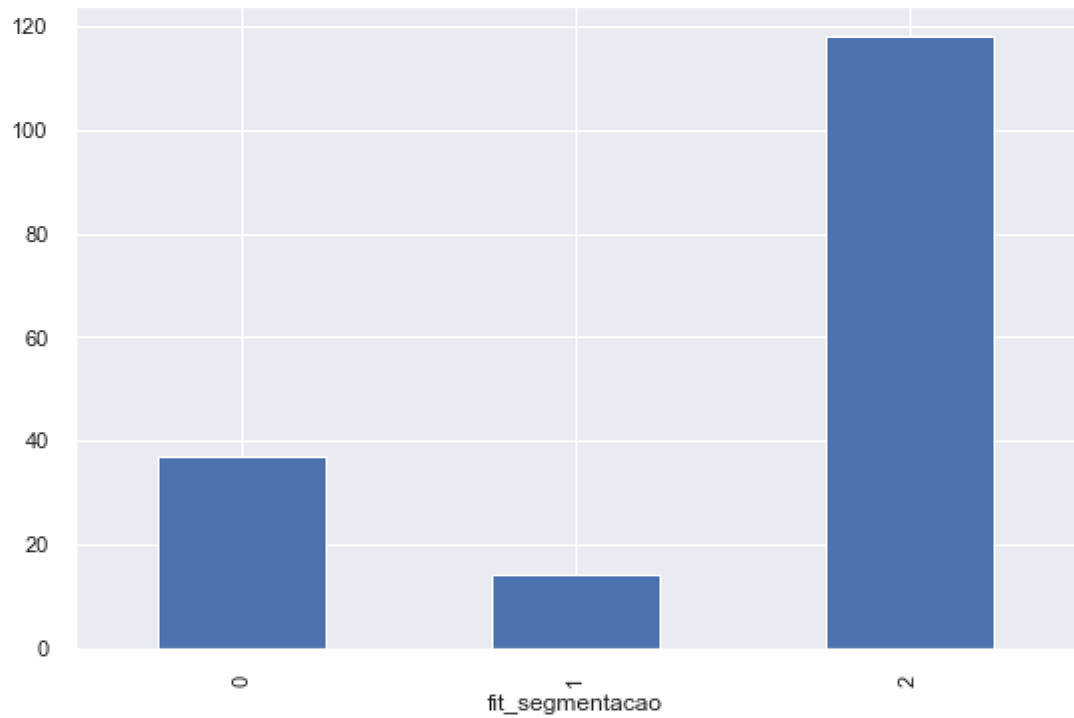

0
1
2

In [178]:
```
for att in num_cluster:
    print (att)

    print( data.groupby('fit_segmentacao', as_index=True)[att].describe())
```

```
renda_mes_media
                 count         mean          std         min          25%          50%        75%           max
fit_segmentacao
0                586.0  6465.720563  2308.773067  203.916667  5836.500000  6369.041667  6860.8750  55555.500000
1                638.0  4823.648119   946.464420  369.000000  4230.937500  4837.958333  5417.5625  13533.083333
2                992.0  2804.360887   964.706871  144.166667  2116.416667  2836.208333  3470.6250   5541.916667
age
                  count       mean        std   min   25%   50%   75%    max
fit_segmentacao
0                 588.0  51.661565  13.646152  25.0  41.0  51.0  63.0  121.0
1                 644.0  55.582298  10.127234  28.0  47.0  55.0  64.0  127.0
2                1008.0  48.118056  11.107355  24.0  40.0  47.0  55.0  120.0
crianca_casa
                  count      mean       std  min  25%  50%  75%  max
fit_segmentacao
0                 588.0  0.044218  0.205753  0.0  0.0  0.0  0.0  1.0
1                 644.0  0.240683  0.452562  0.0  0.0  0.0  0.0  2.0
2                1008.0  0.807540  0.486830  0.0  1.0  1.0  1.0  2.0
adoles_casa
                  count      mean       std  min  25%  50%  75%  max
fit_segmentacao
0                 588.0  0.171769  0.386421  0.0  0.0  0.0  0.0  2.0
1                 644.0  0.936335  0.421782  0.0  1.0  1.0  1.0  2.0
2                1008.0  0.426587  0.518350  0.0  0.0  0.0  1.0  2.0
recencia_dias
                  count       mean        std  min    25%   50%    75%   max
fit_segmentacao
0                 588.0  49.642857  29.420786  0.0  24.00  52.0  74.25  99.0
1                 644.0  48.785714  28.658998  0.0  24.75  50.0  72.00  99.0
2                1008.0  49.004960  28.910418  0.0  24.00  49.0  75.00  99.0
num_visit_web_ult_mes
                  count      mean       std  min  25%  50%  75%   max
fit_segmentacao
0                 588.0  2.835034  1.791043  0.0  1.0  2.0  4.0   9.0
1                 644.0  5.680124  1.873157  0.0  4.0  6.0  7.0   9.0
2                1008.0  6.531746  1.955560  0.0  5.0  7.0  8.0  20.0
campaing_engagement
                  count      mean       std  min  25%  50%  75%  max
fit_segmentacao
0                 588.0  0.142517  0.204286  0.0  0.0  0.0  0.2  0.8
1                 644.0  0.049379  0.106913  0.0  0.0  0.0  0.0  0.6
2                1008.0  0.017659  0.058834  0.0  0.0  0.0  0.0  0.4
```

In [187]:
```
data['digital_profile'] = data['digital_profile'].astype(int)
data.groupby(['fit_segmentacao'])['digital_profile'].sum().plot.bar()
data.groupby(['fit_segmentacao'])['digital_profile'].sum()
```

Out[187]:
```
fit_segmentacao
0     37
1     14
2    118
Name: digital_profile, dtype: int32
```



Classifying the Customers :

In [198]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
class Class_Fit(object):
    def __init__(self, clf, params = None):
        if params:
            self.clf = clf(**params)
        else:
            self.clf = clf()

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def predict(self, x):
        return self.clf.predict(x)

    def grid_search(self, parameters, Kfold):
        self.grid = GridSearchCV(estimator = self.clf, param_grid = parameters, cv = Kfold)

    def grid_fit(self, X, Y):
        self.grid.fit(X, Y)

    def grid_predict(self, X, Y):
        self.predictions = self.grid.predict(X)
        print("Precision: {:.2f} %".format(100 * accuracy_score(Y, self.predictions)))
```

In [197]:
```
data['target'].dtypes
```

Out[197]:
```
dtype('O')
```

In [206]:
```
data['target'] = data['target'].astype(str)
```

In [210]:
```
columns = ['renda_mes_media', 'age','recencia_dias','vinho_montante','frutas_montante','carne_montante','peixe_montante','doces_montante','ouro_montante','promocoes_desconto','promocoes_web','promocoes_catalogo','promocoes_store','num_visit_web_ult_mes','Cmp3','Cmp4','Cmp5','Cmp1','Cmp2','reclamacoes','fit_segmentacao','campaing_engagement']
X = data[columns]
Y = data['target']
```

In [72]:
```
Y=Y.astype(int)
n_instances = len(X)
p_instances = Y.sum() / len(Y)
p_targeted = 0.15
n_targeted = int(n_instances*p_targeted)

print('Number of instances: {:,}'.format(n_instances))
print('Number of conversions {:,}'.format(Y.sum()))
print('Conversion rate: {:.2f}%'.format(p_instances*100.))
print('15% of the population {:,}'.format(n_targeted))
print('Expected number of conversions targetting {:,} @ {:.2f}%: {:,}'.format(n_targeted, p_instances*100., int(p_insta
nces * n_targeted)))
```

```
Number of instances: 2,240
Number of conversions 334
Conversion rate: 14.91%
15% of the population 336
Expected number of conversions targetting 336 @ 14.91%: 50
```

Train, Test Splitting

In [208]:
```
from sklearn.model_selection import train_test_split
```

In [213]:
```
X=X.fillna(0)
X=X.astype(int)
Y=Y.astype(str)
```

In [214]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size = 0.8)
```

Training Models :

In [215]:
```
from sklearn.svm import LinearSVC
from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)
```

In [216]:
```
svc = Class_Fit(clf=LinearSVC)
svc.grid_search(parameters = [{'C':np.logspace(-2,2,10)}], Kfold = 5)
```

In [217]:
```
svc.grid_fit(X=X_train, Y=Y_train)
```

In [218]:
```
svc.grid_predict(X_test, Y_test)
```

```
Precision: 85.04 %
```

In [219]:
```
from sklearn.metrics import confusion_matrix
```

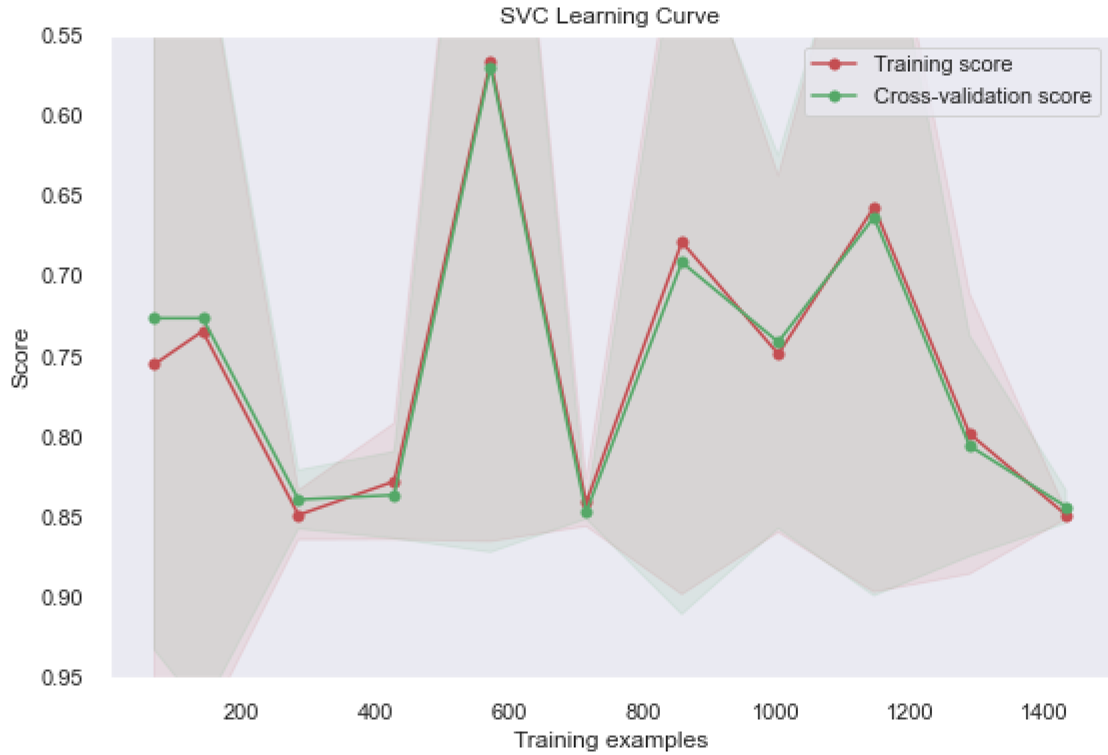
In [220]:
```
class_names = [i for i in range(1,11)]
cnf = confusion_matrix(Y_test, svc.predictions)
cnf
```

Out[220]:
```
array([[380,   1],
       [ 66,   1]], dtype=int64)
```

In [221]:

```python
# Code from sklearn documentation.
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and training learning curve.
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

In [226]:
```
g = plot_learning_curve(svc.grid.best_estimator_, "SVC Learning Curve", X_train, Y_train, ylim=[0.95, 0.55], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
```



Logistic Regression

In [227]:
```
from sklearn.linear_model import LogisticRegression
```

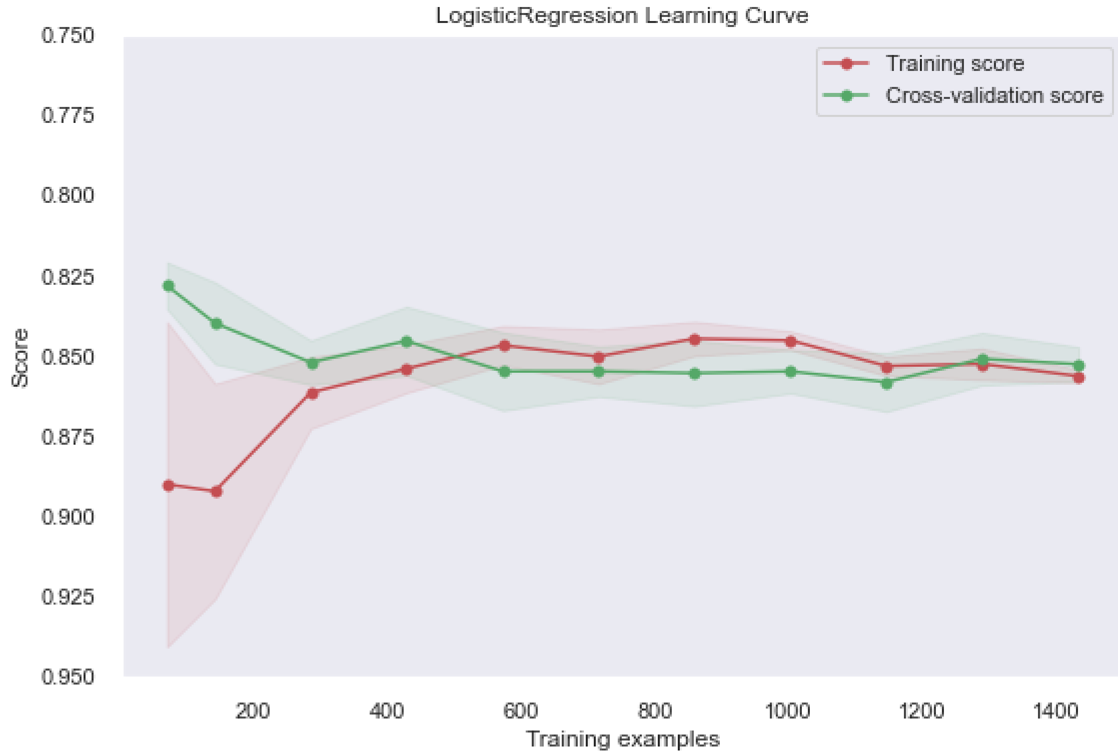
In [228]:
```
lr = Class_Fit(clf = LogisticRegression)
lr.grid_search(parameters = [{'C':np.logspace(-4,6,16)}], Kfold = 17)
lr.grid_fit(X_train, Y_train)
lr.grid_predict(X_test, Y_test)
```

```
Precision: 85.04 %
```

In [229]:
```
cnf = confusion_matrix(Y_test, lr.predictions)
cnf
```

Out[229]:
```
array([[369,  12],
       [ 55,  12]], dtype=int64)
```

In [230]:
```
g = plot_learning_curve(lr.grid.best_estimator_, "LogisticRegression Learning Curve", X_train, Y_train, ylim=[.95, 0.75
], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
```



K-Nearest Neighbours :

In [231]:
```
from sklearn.neighbors import KNeighborsClassifier
```

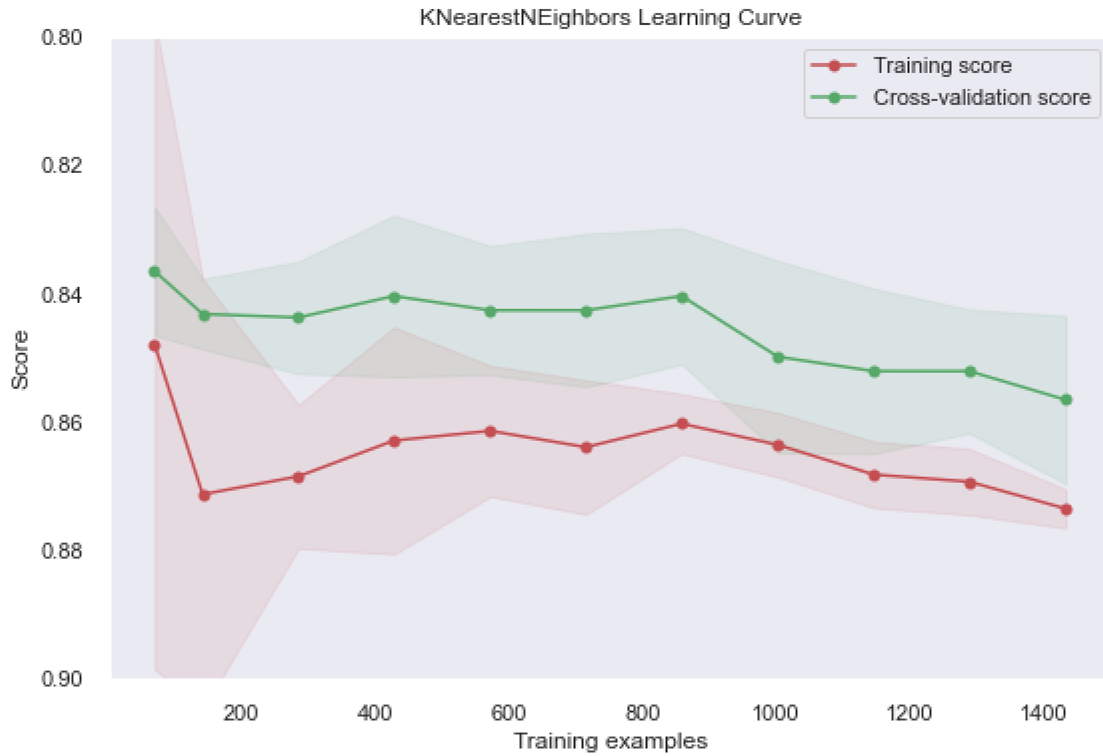
In [232]:
```
knn = Class_Fit(clf = KNeighborsClassifier)
knn.grid_search(parameters = [{'n_neighbors':np.arange(1,50,1)}], Kfold = 10)
knn.grid_fit(X_train, Y_train)
knn.grid_predict(X_test, Y_test)
```

```
Precision: 83.93 %
```

In [233]:
```
cnf = confusion_matrix(Y_test, knn.predictions)
cnf
```

Out[233]:
```
array([[369,  12],
       [ 60,   7]], dtype=int64)
```

In [234]:
```
g = plot_learning_curve(knn.grid.best_estimator_, "KNearestNEighbors Learning Curve", X_train, Y_train, ylim=[.9, 0.8],
cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
```



Decision Trees :

In [235]:
```
from sklearn.tree import DecisionTreeClassifier
```

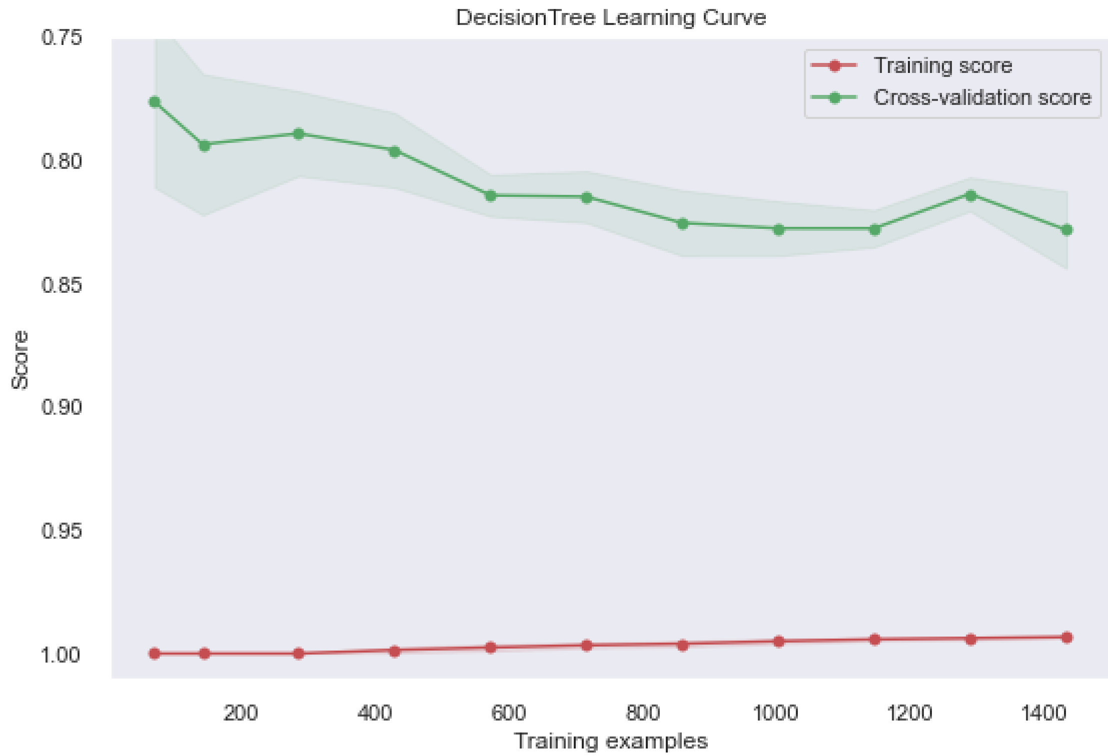
In [236]:
```
tr = Class_Fit(clf = DecisionTreeClassifier)
tr.grid_search(parameters = [{'criterion':['entropy', 'gini'], 'max_features':['sqrt', 'log2']}], Kfold = 3)
tr.grid_fit(X_train, Y_train)
tr.grid_predict(X_test, Y_test)
```

```
Precision: 81.03 %
```

In [237]:
```
cnf = confusion_matrix(Y_test, tr.predictions)
cnf
```

Out[237]:
```
array([[341,  40],
       [ 45,  22]], dtype=int64)
```

In [238]:
```
g = plot_learning_curve(tr.grid.best_estimator_, "DecisionTree Learning Curve", X_train, Y_train, ylim=[1.01, 0.75], cv
= 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
```



Random Forests:

In [239]:
```
from sklearn.ensemble import RandomForestClassifier
```

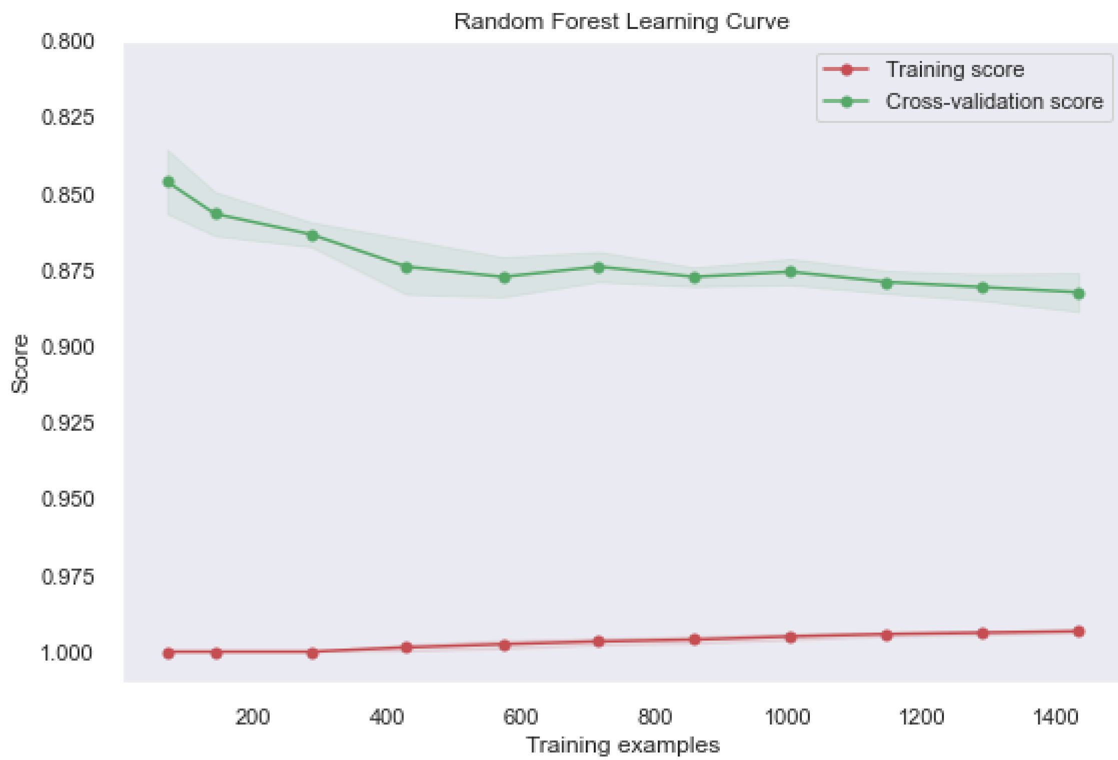
In [240]:
```
rf = Class_Fit(clf = RandomForestClassifier)
rf.grid_search(parameters = [{'criterion':['entropy', 'gini'],
                              'max_features':['sqrt', 'log2'], 'n_estimators':[40, 60, 80, 100, 140]}], Kfold = 5)
rf.grid_fit(X_train, Y_train)
rf.grid_predict(X_test, Y_test)
```

```
Precision: 86.61 %
```

In [258]:
```
cnf = confusion_matrix(Y_test, rf.predictions)
cnf
```

Out[258]:
```
array([[385,   5],
       [ 42,  16]], dtype=int64)
```

In [241]:
```
g = plot_learning_curve(rf.grid.best_estimator_, "Random Forest Learning Curve", X_train, Y_train, ylim=[1.01, 0.8], cv
= 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
```



In [242]:
```
from sklearn.ensemble import AdaBoostClassifier
```

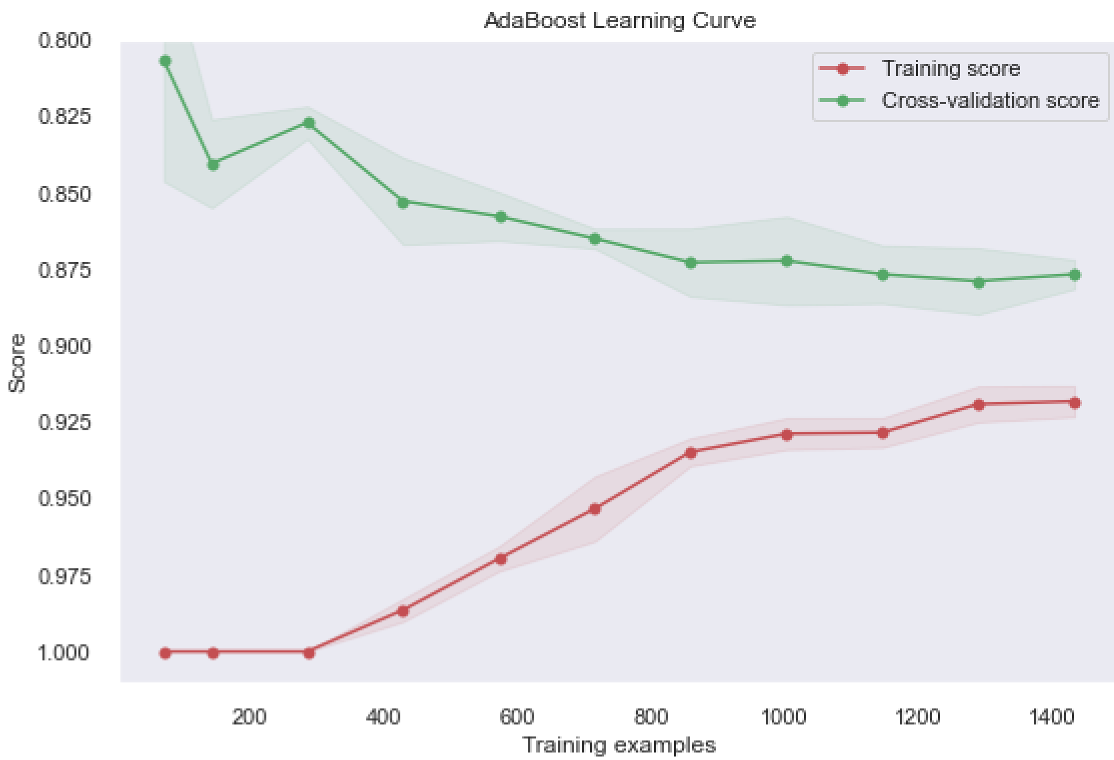
```
In [244]: ada = Class_Fit(clf = AdaBoostClassifier)
          ada.grid_search(parameters = [{'n_estimators':[20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 130]}], Kfold = 8)
          ada.grid_fit(X_train, Y_train)
          ada.grid_predict(X_test, Y_test)
```

```
Precision: 86.16 %
```

```
In [245]: cnf = confusion_matrix(Y_test, ada.predictions)
          cnf
```

```
Out[245]: array([[358,  23],
                 [ 39,  28]], dtype=int64)
```

```
In [246]: g = plot_learning_curve(ada.grid.best_estimator_, "AdaBoost Learning Curve", X_train, Y_train, ylim=[1.01, 0.8], cv = 5
          ,
                                  train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
```



Gradient Boosted Decision Trees :

In [248]:
```python
import xgboost
```

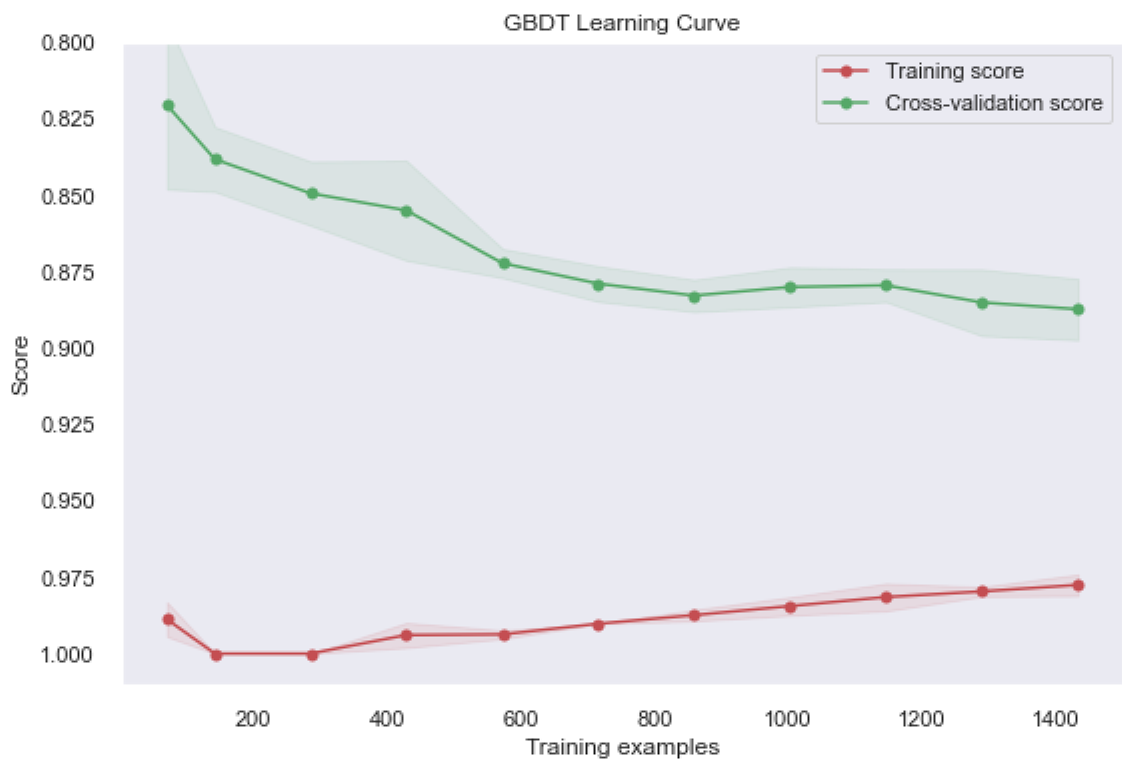
In [249]:
```python
gbdt = Class_Fit(clf = xgboost.XGBClassifier)
gbdt.grid_search(parameters = [{'n_estimators':[20, 30, 40, 50, 60, 70, 80, 90, 100, 120]}], Kfold = 5)
gbdt.grid_fit(X_train, Y_train)
gbdt.grid_predict(X_test, Y_test)
```

```
Precision: 85.71 %
```

In [250]:
```python
cnf = confusion_matrix(Y_test, gbdt.predictions)
cnf
```

Out[250]:
```
array([[363,  18],
       [ 46,  21]], dtype=int64)
```

In [251]:
```python
g = plot_learning_curve(gbdt.grid.best_estimator_, "GBDT Learning Curve", X_train, Y_train, ylim=[1.01, 0.8], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
```



Voting Classifier :

In [252]:
```
rf_best = RandomForestClassifier(**rf.grid.best_params_)
gbdt_best = xgboost.XGBClassifier(**gbdt.grid.best_params_)
svc_best = LinearSVC(**svc.grid.best_params_)
tr_best = DecisionTreeClassifier(**tr.grid.best_params_)
knn_best = KNeighborsClassifier(**knn.grid.best_params_)
lr_best = LogisticRegression(**lr.grid.best_params_)
```

In [253]:
```
from sklearn.ensemble import VotingClassifier
```

In [263]:
```
votingC = VotingClassifier(estimators=[('rf', rf_best), ('gb', gbdt_best), ('knn', knn_best), ('lr', lr_best),('svc', s
vc_best),('tr', tr_best)])
```

In [264]:
```
votingC = votingC.fit(X_train, Y_train)
```

In [265]:
```
predictions = votingC.predict(X_test)
```

In [266]:
```
print("Precision : {:.2f}%".format(100 * accuracy_score(Y_test, predictions)))
```

```
Precision : 85.71%
```

In [267]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       381
           1       0.59      0.15      0.24        67

    accuracy                           0.86       448
   macro avg       0.73      0.57      0.58       448
weighted avg       0.83      0.86      0.82       448
```

Testing Model

In [268]:
```
X
```

Out[268]:

	renda_mes_media	age	recencia_dias	vinho_montante	frutas_montante	carne_montante	peixe_montante	doces_montante	ouro_montante
0	4844	63	58	635	88	546	172	88	88
1	3862	66	38	11	1	6	2	1	6
2	5967	55	26	426	49	127	111	21	42
3	2220	36	26	11	4	20	10	3	5
4	4857	39	94	173	43	118	46	27	15
...	...	...	...	...	...	...	...	...	...
2235	5101	53	46	709	43	182	42	118	247
2236	5334	74	56	406	0	30	0	0	8
2237	4748	39	91	908	48	217	32	12	24
2238	5770	64	8	428	30	214	80	30	61
2239	4405	66	40	84	3	61	2	1	21

2240 rows × 22 columns

In [269]:
```
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(X)
```

In [270]:
```
predictions = votingC.predict(X)
```

In [271]:
```
print("Precision : {:.2f}%".format(100 * accuracy_score(Y, predictions)))
```

```
Precision : 91.83%
```