Sprint 8 : Tasca M8 T01

Exercici 1

Descarrega el dataset adjunt, de registres de publicacions a Facebook sobre Tailàndia, i classifica els diferents registres utilitzant l'algorisme de K-means

In [207…

```
# Tratamiento de datos
# ==============================================================================
import pandas as pd
import numpy as np

# Gráficos
# ==============================================================================
import matplotlib.pyplot as plt
from matplotlib import style
import matplotlib.ticker as ticker
import seaborn as sns
from pprint import pprint

# Preprocesado y análisis
# ==============================================================================
#import statsmodels.api as sm
#import pingouin as pg
from scipy import stats
import random as rd
from imblearn.over_sampling import SMOTE

# Preprocesado y modelado
# ==============================================================================
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from kneed import KneeLocator


# Test Estadísticos
# ==============================================================================
from scipy.stats import shapiro
from scipy.stats import ttest_ind
from scipy.stats import ttest_rel

# Ajuste de distribuciones
# ==============================================================================
from scipy import stats
import inspect
from statsmodels.distributions.empirical_distribution import ECDF
import scipy.cluster.hierarchy as shc



# Configuración matplotlib
# ==============================================================================
plt.style.use('ggplot')
from statsmodels.graphics.gofplots import qqplot
from matplotlib import pyplot

# Configuración warnings
# ==============================================================================
import warnings
warnings.filterwarnings('ignore')
```

1.1 Introducción

The 'Facebook Live Sellers in Thailand' is a dataset curated in UCI Machine Learning Datasets. The data contains 7050 observations and twelve attributes. The data is about live selling feature on the Facebook platform. Each record consists of information about the time live information of sale is posted to Facebook and engagements in the data. The engagements are regular Facebook interactions such as share and emotion rection. Details and academic publications relating to the data is available from the source https://archive.ics.uci.edu/ml/datasets/Facebook+Live+Sellers+in+Thailand.

Facebook Live Sellers in Thailand Data Set

Abstract: Facebook pages of 10 Thai fashion and cosmetics retail sellers. Posts of a different nature (video, photos, statuses, and links). Engagement metrics consist of comments, shares, and reactions.

- Data Set Characteristics: Multivariate
- Number of Instances: 7051

- Area: Business
- Attribute Characteristics: Integer
- Number of Attributes:12
- Date Donated: 2019-04-22
- Associated Tasks: Clustering
- Missing Values?: N/A
- Number of Web Hits: 65141

Source: Nassim Dehouche, Mahidol University International College, nassim.deh '@' mahidol.edu

Data Set Information:

- The variability of consumer engagement is analysed through a Principal Component Analysis, highlighting the changes induced by the use of Facebook Live. The seasonal component is analysed through a study of the averages of the different engagement metrics for different time-frames (hourly, daily and monthly). Finally, we identify statistical outlier posts, that are qualitatively analyzed further, in terms of their selling approach and activities.

Attribute Information:

- status_id
- status_type
- status_published
- num_reactions
- num_comments
- num_shares
- num_likes
- num_loves
- num_wows
- num_hahas
- num_sads
- num_angrys

Relevant Papers:

- Nassim Dehouche and Apiradee Wongkitrungrueng. Facebook Live as a Direct Selling Channel, 2018, Proceedings of ANZMAC 2018: The 20th Conference of the Australian and New Zealand Marketing Academy. Adelaide (Australia), 3-5 December 2018. Dehouche, N., 2020, Dataset on usage and engagement patterns for Facebook Live sellers in Thailand. Data in Brief, 30(105661).
- Wongkitrungrueng, A., Dehouche, N., and Assarut, N., 2020, Live streaming commerce from the seller's perspective: implications for online relationship marketing. Journal of Martketing Management, 36(5-6).

Citation Request:

- Nassim Dehouche and Apiradee Wongkitrungrueng. Facebook Live as a Direct Selling Channel, 2018, Proceedings of ANZMAC 2018: The 20th Conference of the Australian and New Zealand Marketing Academy. Adelaide (Australia), 3-5 December 2018.

In [208…
```
data= pd.read_csv(r"C:\Users\hecto\OneDrive\Documentos\IT Data Science\Sprint8_DS\Publicacions Facebook Thailandia.csv")
data.head(3)
```

Out[208]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1

In [209…
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   status_id         7050 non-null   object 
 1   status_type       7050 non-null   object 
 2   status_published  7050 non-null   object 
 3   num_reactions     7050 non-null   int64  
 4   num_comments      7050 non-null   int64  
 5   num_shares        7050 non-null   int64  
 6   num_likes         7050 non-null   int64  
 7   num_loves         7050 non-null   int64  
 8   num_wows          7050 non-null   int64  
 9   num_hahas         7050 non-null   int64  
 10  num_sads          7050 non-null   int64  
 11  num_angrys        7050 non-null   int64  
 12  Column1           0 non-null      float64
 13  Column2           0 non-null      float64
 14  Column3           0 non-null      float64
 15  Column4           0 non-null      float64
dtypes: float64(4), int64(9), object(3)
memory usage: 881.4+ KB
```

In [210…
```
data.describe().T
```

Out[210]:

	count	mean	std	min	25%	50%	75%	max
num_reactions	7050.0	230.117163	462.625309	0.0	17.0	59.5	219.00	4710.0
num_comments	7050.0	224.356028	889.636820	0.0	0.0	4.0	23.00	20990.0
num_shares	7050.0	40.022553	131.599965	0.0	0.0	0.0	4.00	3424.0
num_likes	7050.0	215.043121	449.472357	0.0	17.0	58.0	184.75	4710.0
num_loves	7050.0	12.728652	39.972930	0.0	0.0	0.0	3.00	657.0
num_wows	7050.0	1.289362	8.719650	0.0	0.0	0.0	0.00	278.0
num_hahas	7050.0	0.696454	3.957183	0.0	0.0	0.0	0.00	157.0
num_sads	7050.0	0.243688	1.597156	0.0	0.0	0.0	0.00	51.0
num_angrys	7050.0	0.113191	0.726812	0.0	0.0	0.0	0.00	31.0
Column1	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Column2	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Column3	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Column4	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [211…
```
data.isnull().sum()
```

Out[211]:
```
status_id              0
status_type            0
status_published       0
num_reactions          0
num_comments           0
num_shares             0
num_likes              0
num_loves              0
num_wows               0
num_hahas              0
num_sads               0
num_angrys             0
Column1             7050
Column2             7050
Column3             7050
Column4             7050
dtype: int64
```

- Como se puede observar las variables Columns 1,2,3 y 4 no tiene valores asignados, por tanto procederemos a eliminarlas ya que no aportan nada al anaálisis

In [212…
```
data = data[data.columns[:-4]]
data.head(3)
```

Out[212]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1

In [213…
```
data.nunique()
```

Out[213]:
```
status_id           6997
status_type            4
status_published    6913
num_reactions       1067
num_comments         993
num_shares           501
num_likes           1044
num_loves            229
num_wows              65
num_hahas             42
num_sads              24
num_angrys            14
dtype: int64
```

- El resultado de los datos únicos refleja que existe una direfencia en la variable status_id con 6997 datos no repetidos, respecto ea total de datos es de 7050. La diferencia entre 7050 y 6997 son 53 datos que están repetidos y vamos a proceder a su análisis.

In [214…
```
duplicated_data = data[data['status_id'].duplicated() == True]
duplicated_data.head()
```

Out[214]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
1698	246675545449582_326883450762124	photo	10/11/2013 8:23	211	2	0	211	0	
1729	246675545449582_429583263825475	photo	9/11/2013 0:12	537	16	1	537	0	
6221	819700534875473_1002372733274918	video	6/10/2018 3:43	376	20	3	354	19	
6222	819700534875473_1001982519980606	photo	6/9/2018 22:53	255	7	4	249	6	
6223	819700534875473_1000607730118085	photo	6/7/2018 7:01	1704	21	3	1685	15	

In [215…
```
data[data.status_id == '246675545449582_326883450762124']
```

Out[215]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wo
1488	246675545449582_326883450762124	photo	2/14/2014 3:07	211	2	0	211	0	
1698	246675545449582_326883450762124	photo	10/11/2013 8:23	211	2	0	211	0	

In [216…
```
data[data.status_id == '246675545449582_429583263825475']
```

Out[216]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wo
1408	246675545449582_429583263825475	photo	4/22/2014 5:43	537	16	1	537	0	
1729	246675545449582_429583263825475	photo	9/11/2013 0:12	537	16	1	537	0	

In [217…
```
data[data.status_id == '819700534875473_1002372733274918']
```

Out[217]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
6170	819700534875473_1002372733274918	video	6/10/2018 3:43	376	20	3	354	19	
6221	819700534875473_1002372733274918	video	6/10/2018 3:43	376	20	3	354	19	

- Dado que aunque el indicador de Status_id está repetido, pero en algunos casos la variable status_published es diferente, no podemos eliminar los elementos duplicados al tener una diferencias en la fecha y hora de publicación. No podemos concluir que son registros exáctos y tampoco el porcentaje de elementos repetidos (53/7050 = 0,75%) nos parecen relevantes.

In [218…
```
data=data.drop(["status_id","status_published"],axis=1)
data.head(3)
```

Out[218]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	video	529	512	262	432	92	3	1	1	0
1	photo	150	0	0	150	0	0	0	0	0
2	video	227	236	57	204	21	1	1	0	0

1.2 Transformación de las variables Categóricas

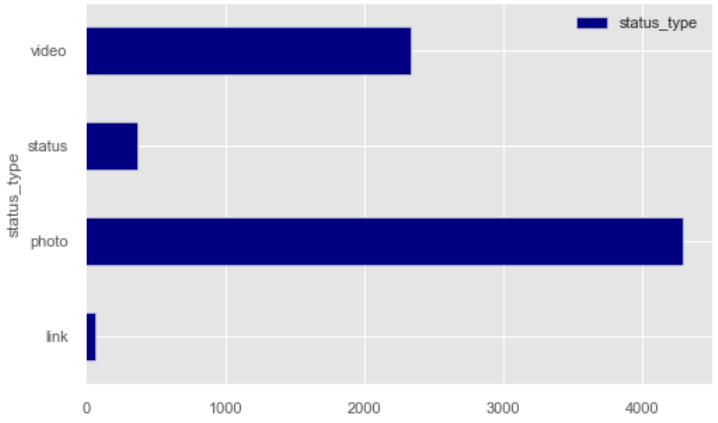
In [219…
```
status= data.groupby("status_type")[["status_type"]].count()
status
```

Out[219]:

status_type	status_type
link	63
photo	4288
status	365
video	2334

In [220…
```
status.plot(kind="barh", color="navy")
```

Out[220]:
```
<AxesSubplot:ylabel='status_type'>
```



- Status_type es una variable categórica con los tipos: video, status, photo, y link. Photo y video represetntan el 94% del total de los datos con 4288 y 2334 respectivamente.
- Para poder aplicar los modelos apliaremos LabelEncoder sobre la variable categórica, para convertir las equiquetas en variables numéricas.

In [221…
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
data['status_type']=le.fit_transform(data['status_type'])
data.head(3)
```

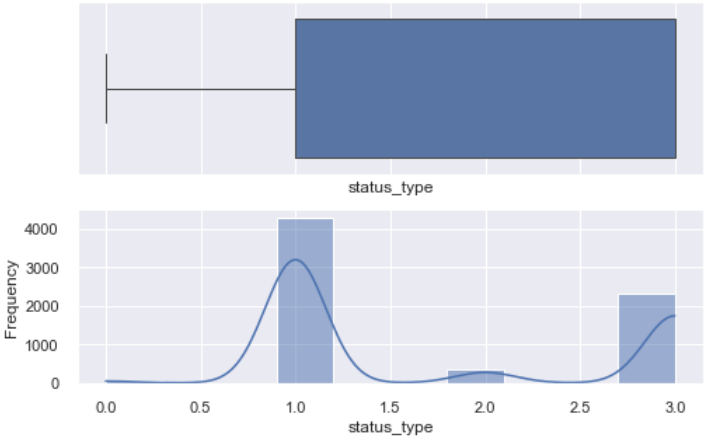
Out[221]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	3	529	512	262	432	92	3	1	1	0
1	1	150	0	0	150	0	0	0	0	0
2	3	227	236	57	204	21	1	1	0	0

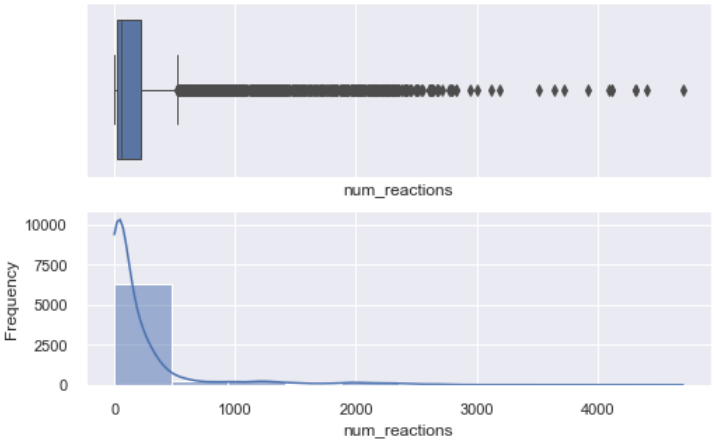
1.3 Análisis Gráfico de las variables

In [222…
```
import matplotlib.pyplot as plt
import seaborn as sns

for i in data.columns:
    plt.figure()
    plt.tight_layout()
    sns.set(rc={"figure.figsize":(8, 5)})
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True)
    plt.gca().set(xlabel= i,ylabel='Frequency')
    sns.boxplot(data[i],  ax=ax_box , linewidth= 1.0)
    sns.histplot(data[i], ax=ax_hist , bins = 10,kde=True)
```

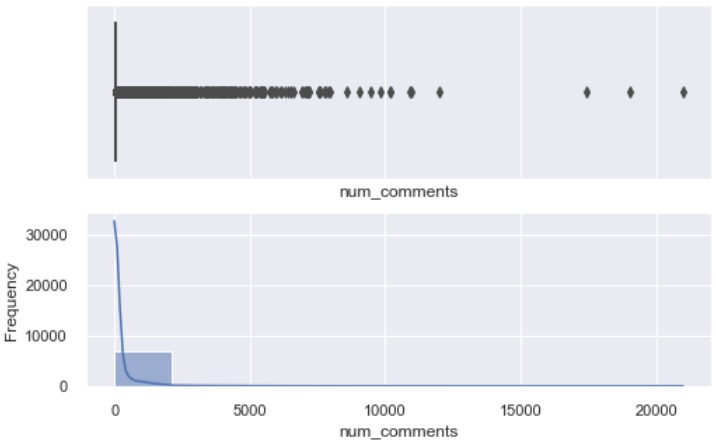
```
<Figure size 576x360 with 0 Axes>
```



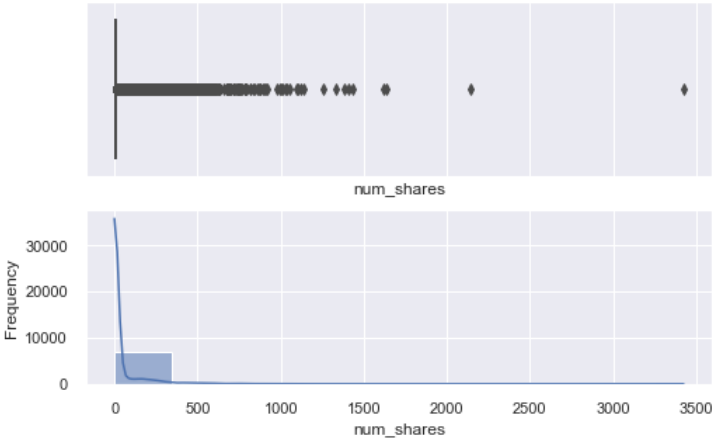

```
<Figure size 576x360 with 0 Axes>
```



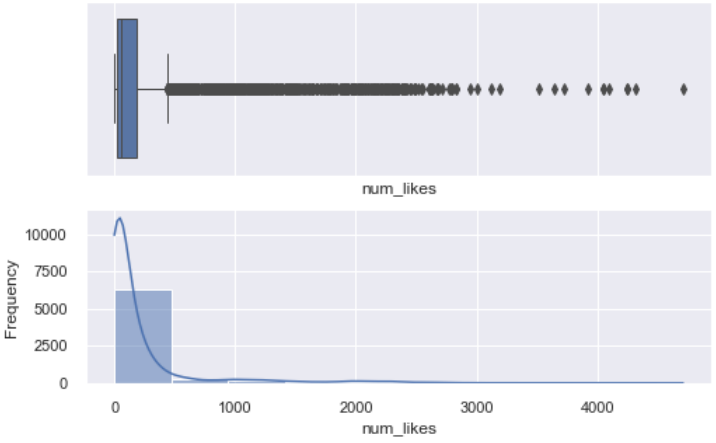

```
<Figure size 576x360 with 0 Axes>
```



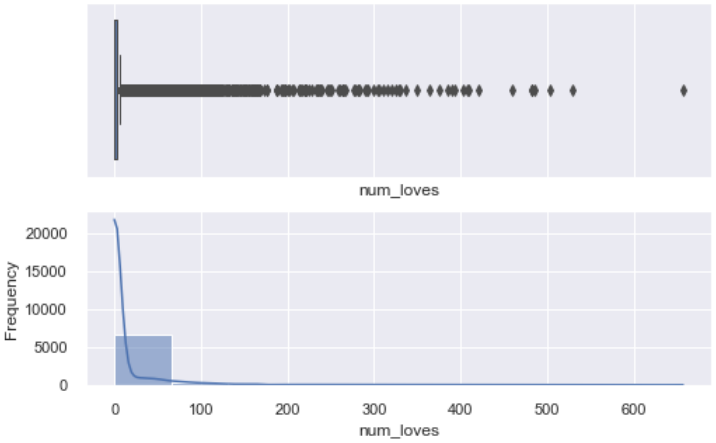

```
<Figure size 576x360 with 0 Axes>
```



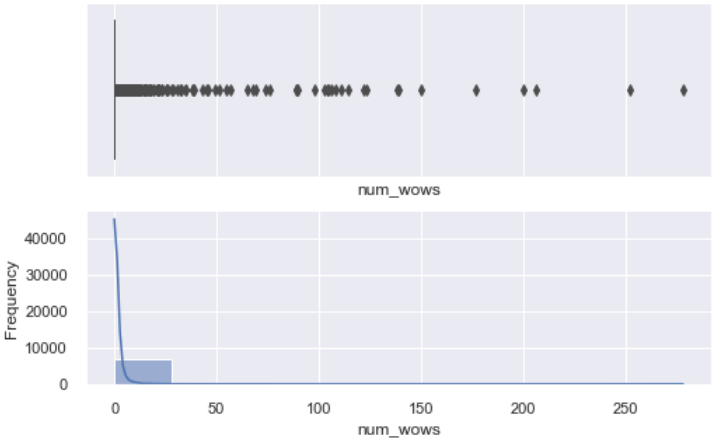

```
<Figure size 576x360 with 0 Axes>
```



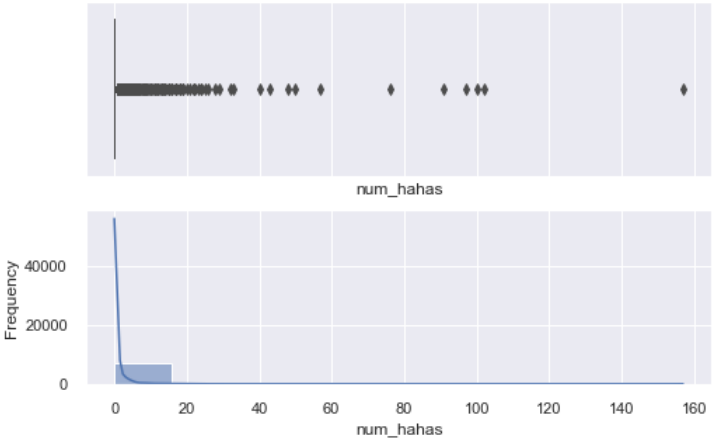

```
<Figure size 576x360 with 0 Axes>
```



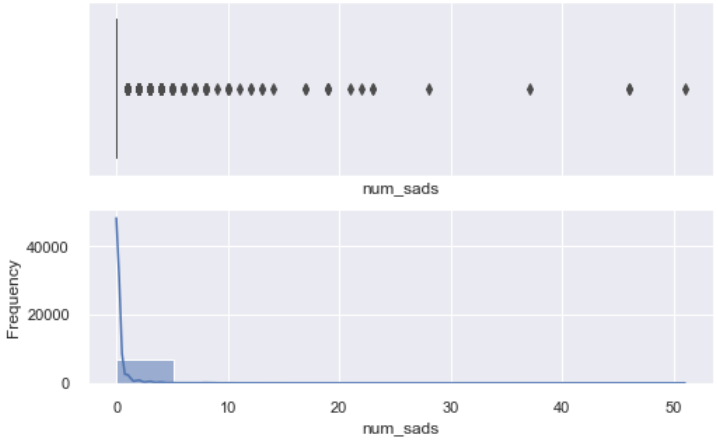

```
<Figure size 576x360 with 0 Axes>
```



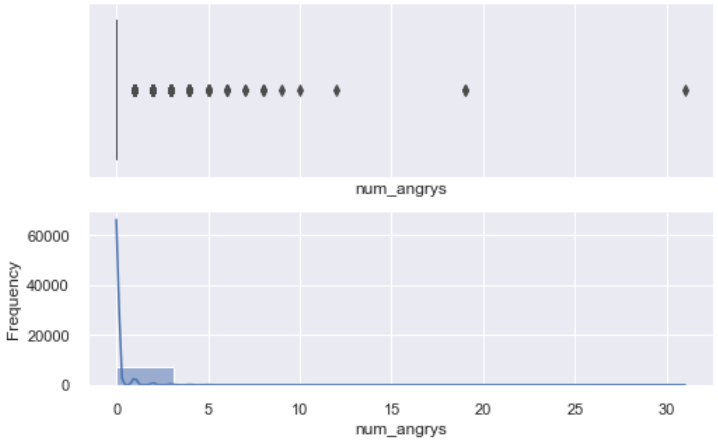

```
<Figure size 576x360 with 0 Axes>
```




```
<Figure size 576x360 with 0 Axes>
```




```
<Figure size 576x360 with 0 Axes>
```

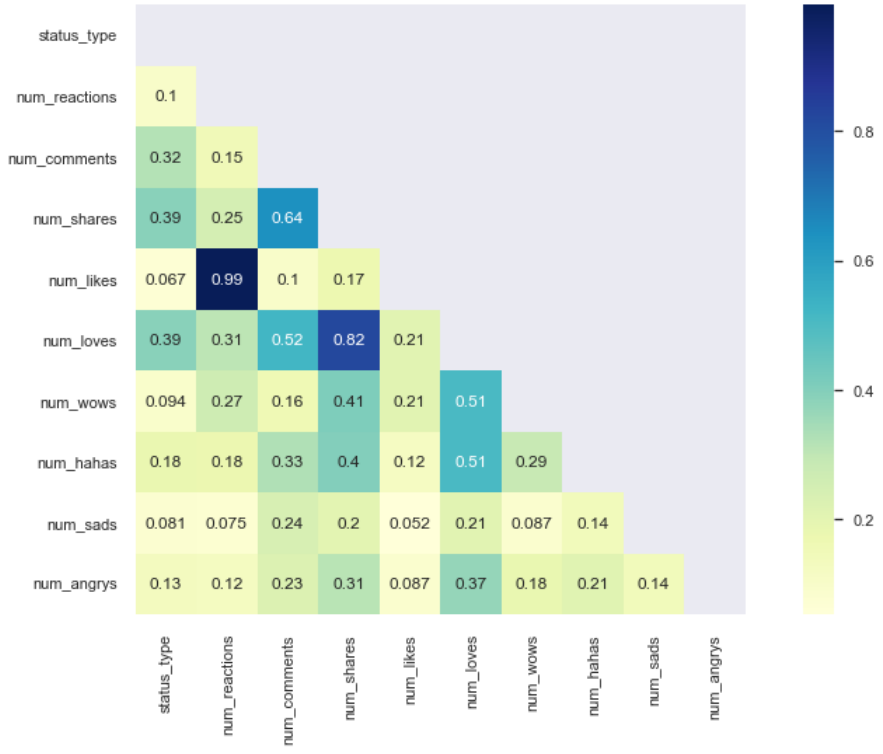


- Como podemos observar, tanto en los gráficos boxplot como en la distribución de Frecuencias, existe una gran dispersión en los datos y la mayoría presentan numerosos outliers, que deberemos tratar en el apartado 2. Preprocesing Data.

1.4 Correlación de las variables

In [223…
```
plt.figure(figsize=(15,8))
upp_mat = np.triu(data.corr())
sns.heatmap(data.corr(),cmap="YlGnBu", square = True,annot=True, mask = upp_mat)
```

Out[223]:
```
<AxesSubplot:>
```



- Existe una baja correlación en general entre las variables explicativas, salvo en el caso de las variables num_likes y num_reactions, con un

coeficiente de correlación de 0,99, y entre num_loves y num_shares con un coeficiente de 0,82.

- También existe una elevada correlación, con doeficientes superiores al 0,5 entre, num_wows y num_loves, num_hahas y num_loves y num_loves y num_comments.
- Aunque aplicaremos en este caso modelos de aprendizaje no supervisado, vamos a evaluar si tener en cuenta o no, la multicolinealidad, afecta al cálculo del PCA (Principal Component Analysis).

1.5 Estimación de la multicolinealidad con VIF (Variance Inflation Factor)

Según Hanke, y Wichern (2010)la multicolinealidad es la situación en la cual las variables independientes de una ecuación de regresión múltiple están sumamente intercorrelacionadas. Es decir, existe una relación lineal entre dos o más variables independientes. La multicolinealidad se refiere a que dos o más variables pueden contener información repetida. El impacto que tiene la multicolinealidad en las variables se puede medir mediante un indicador que se conoce como Factor de Inflación de la Varianza o VIF por sus siglas en inglés (variance inflation factor):

- VIF = 1 / (1-R^2)
- Si el valor de VIF es cercano a 1 se puede interpretar que la multicolinealidad no es un gran problema para la variable en cuestión. Un valor de VIF más grande que 1implica que el coeficiente de esa variable puede ser modificado de manera importante conforme se tomen en cuenta a las demás variables en el modelo de regresión.En última instancia,un valor de VIF alto implica que existe información redundante en las variables independientesy que, por lo tanto, cada vez que una de estas variables se modifique,el coeficiente de otra variable puede ser afectado.
- La regla general para decidir sobre la corección de la multicolinealidad es la siguiente:
  1. VIF=1 significa que no existe correlación entre esta variable independiente y cualquier otra.
  2. 1< VIF < 5 sugiere una correlación moderada pero no sería necesario resolverla.
  3. VIF > 5 son niveles críticos de multicolinealidad.

In [224…
```
# Shape del dataset original:

data_mc=data
print(data_mc.shape)

# Se divide entre conjunto de variables predictoras y conjunto variables objetivo:

target_label = "status_type"
pred_labels = data_mc.columns.to_list()
pred_labels.remove(target_label)

print(len(pred_labels))

# Conjunto de datos predictor y el conjunto objetivo:

x_pred = data_mc[pred_labels]
y_target = data_mc[target_label]
```

```
(7050, 10)
9
```

In [225…
```
# LinearRegression para el calculo de las Ri
from sklearn.linear_model import LinearRegression

def calculateVIF(var_predictoras_df):
    var_pred_labels = list(var_predictoras_df.columns)
    num_var_pred = len(var_pred_labels)

    lr_model = LinearRegression()

    result = pd.DataFrame(index = ['VIF'], columns = var_pred_labels)
    result = result.fillna(0)

    for ite in range(num_var_pred):
        x_features = var_pred_labels[:]
        y_feature = var_pred_labels[ite]
        x_features.remove(y_feature)

        x = var_predictoras_df[x_features]
        y = var_predictoras_df[y_feature]

        lr_model.fit(var_predictoras_df[x_features], var_predictoras_df[y_feature])

        result[y_feature] = 1/(1 - lr_model.score(var_predictoras_df[x_features], var_predictoras_df[y_feature]))

    return result
```

In [226…
```
calculateVIF(x_pred.copy(deep = True)).T
```

Out[226]:

	VIF
num_reactions	3.108463e+07
num_comments	1.792287e+00
num_shares	3.823288e+00
num_likes	2.934227e+07
num_loves	2.322467e+05
num_wows	1.104074e+04
num_hahas	2.273337e+03
num_sads	3.712007e+02
num_angrys	7.760214e+01

- En nuestro caso solo dos variables tienen valores de VIF inferiores a 5, num_comments y num_shares, el resto de variables tienen VIF muy elevados que requieren atención.
- Para eliminar las variables que generan la multicolinealidad aplicaremos una función SelectDataUsingVIF que lo que hace es generar un bucle que termina cuando el valor máximo del VIF es inferior al criterio. En el bucle se elimina la columna de dataframe con el valor de VIF más elevado en cada momento y se vuelve a calcular el VIF de todas las variables.

In [227…
```
def selectDataUsingVIF(var_predictoras_df, max_VIF = 5):
    result = var_predictoras_df.copy(deep = True)

    VIF = calculateVIF(result)

    while VIF.values.max() > max_VIF:
        col_max = np.where(VIF == VIF.values.max())[1][0]
        features = list(result.columns)
        features.remove(features[col_max])
        result = result[features]

        VIF = calculateVIF(result)

    return result
```

In [228…
```
calculateVIF(selectDataUsingVIF(x_pred)).T
```

Out[228]:

	VIF
num_comments	1.792220
num_shares	3.804506
num_likes	1.061638
num_loves	3.970589
num_wows	1.403558
num_hahas	1.371272
num_sads	1.076220
num_angrys	1.167430

- Al final de proceso se obsera que ha sido eliminada la variable num_reactions y el nuevo valor VIF para el resto de las variables es en todos los casos <5.

1.6 Preprocesing Data

1.6.1 Evaluación de la normalidad de las variables

In [229…
```
data.isnull().values.any()
```

Out[229]:
```
False
```

In [230…
```
# create a function that checks if the distribution is normal:
def check_normal_distribution(data):

    for i in data[features]:
        stat, p_value_norm = shapiro(data[i])
        print(f'Results for {i}:')
        print('stat=%.3f, p=%.3f' % (stat, p_value_norm))

        if p_value_norm < 0.05 :
            print("Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed")
            print('----------------------------------------------------------------------------------------------')
        else:
            print("Fail to reject null hypothesis at 95€ Significance Level  >> The data is normally distributed")
            print('----------------------------------------------------------------------------------------------')
```

In [231…
```
features =["status_type","num_reactions","num_comments","num_shares","num_likes","num_loves","num_wows","num_hahas","num_sads"
check_normal_distribution(data[features])
```

```
Results for status_type:
stat=0.665, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_reactions:
stat=0.510, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_comments:
stat=0.261, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_shares:
stat=0.337, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_likes:
stat=0.489, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_loves:
stat=0.353, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_wows:
stat=0.110, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_hahas:
stat=0.148, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_sads:
stat=0.130, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
Results for num_angrys:
stat=0.135, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distributed
----------------------------------------------------------------------------------------------
```

- Ninguna de las variables explicativas sigue una función normal y además tienen numerosos outlaers por lo que aplicaremos RobustScales en el procesamiento de datos.

1.6.2 RobustScaler

RobustScaler aplica un escalado a las características de forma que sea más robusto a los outliers. Concretamente elimina la mediana y escala los datos de acuerdo a un rango de cuartiles (por defecto el rango intercuantil: el rango de valores entre el 1er cuartil y el 3er cuartil).

Razones para utilizar pruebas robustas:

- Son estables respecto a pequeñas desviaciones del modelo paramétrico asumido (normalidad y homocedasticidad).
- A diferencia de los procedimientos no paramétricos, los procedimientos estadísticos robustos no tratan de comportarse necesariamente bien para una amplia clase de modelos, pero son de alguna manera óptimos en un entorno de cierta distribución de probabilidad, por ejemplo, normal.
- Solucionan los problemas de influencia de los outliers.
- Son más potentes que las pruebas paramétricas y no paramétricas cuando los datos no son normales y/o no son homocedásticos.

In [232…
```
from sklearn.preprocessing import RobustScaler
robScaColumns = ["num_reactions","num_comments","num_shares","num_likes","num_loves","num_wows","num_hahas","num_sads","num_ar
scalerRob = preprocessing.RobustScaler().fit(data[robScaColumns])# RobustEscaler
data[robScaColumns] = scalerRob.transform(data[robScaColumns])
```

In [233…
```
data.describe().T
```

Out[233]:

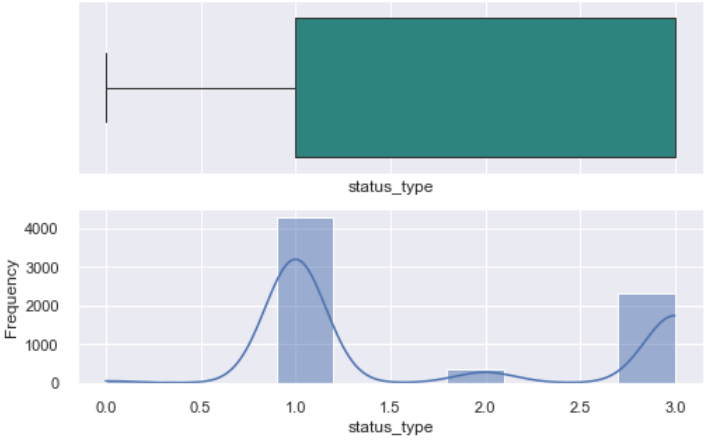
	count	mean	std	min	25%	50%	75%	max
status_type	7050.0	1.704965	0.942399	0.000000	1.000000	1.0	3.000000	3.000000
num_reactions	7050.0	0.844639	2.290224	-0.294554	-0.210396	0.0	0.789604	23.022277
num_comments	7050.0	9.580697	38.679862	-0.173913	-0.173913	0.0	0.826087	912.434783
num_shares	7050.0	10.005638	32.899991	0.000000	0.000000	0.0	1.000000	856.000000
num_likes	7050.0	0.936174	2.679418	-0.345753	-0.244411	0.0	0.755589	27.731744
num_loves	7050.0	4.242884	13.324310	0.000000	0.000000	0.0	1.000000	219.000000
num_wows	7050.0	1.289362	8.719650	0.000000	0.000000	0.0	0.000000	278.000000
num_hahas	7050.0	0.696454	3.957183	0.000000	0.000000	0.0	0.000000	157.000000
num_sads	7050.0	0.243688	1.597156	0.000000	0.000000	0.0	0.000000	51.000000
num_angrys	7050.0	0.113191	0.726812	0.000000	0.000000	0.0	0.000000	31.000000

1.6.3 Análisis Gráfico de las variables con RobustScaler

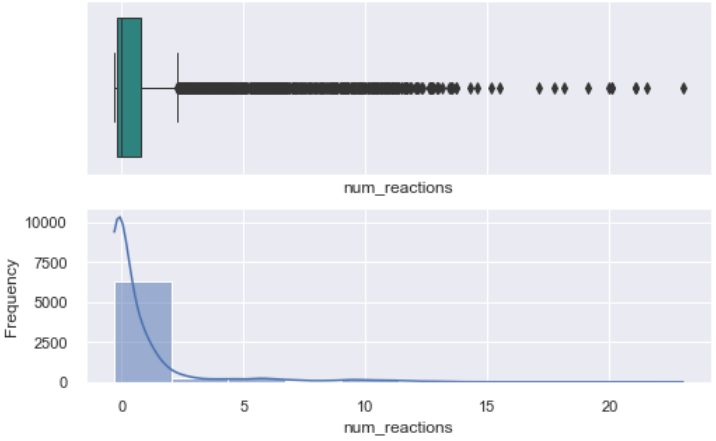
In [234…

```
for i in data.columns:
    plt.figure()
    plt.tight_layout()
    sns.set(rc={"figure.figsize":(8, 5)})
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True)
    plt.gca().set(xlabel= i,ylabel='Frequency')
    sns.boxplot(data[i],  ax=ax_box , linewidth= 1.0, palette="viridis")
    sns.histplot(data[i], ax=ax_hist , bins = 10,kde=True, palette="viridis")
```

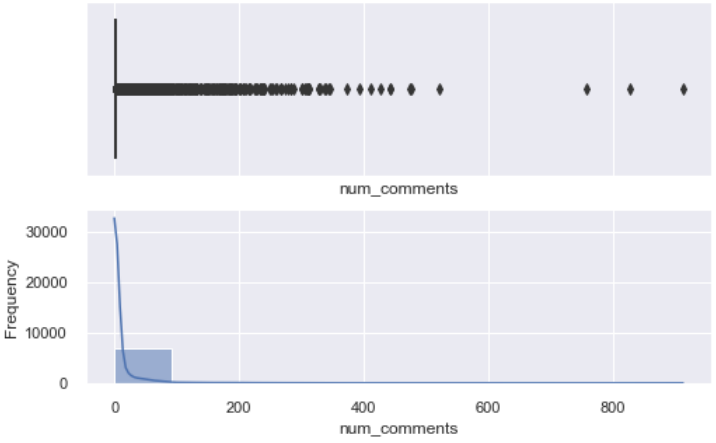
```
<Figure size 576x360 with 0 Axes>
```



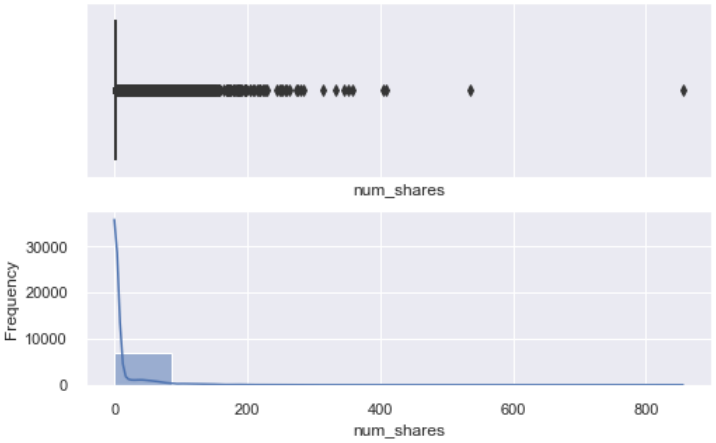

```
<Figure size 576x360 with 0 Axes>
```



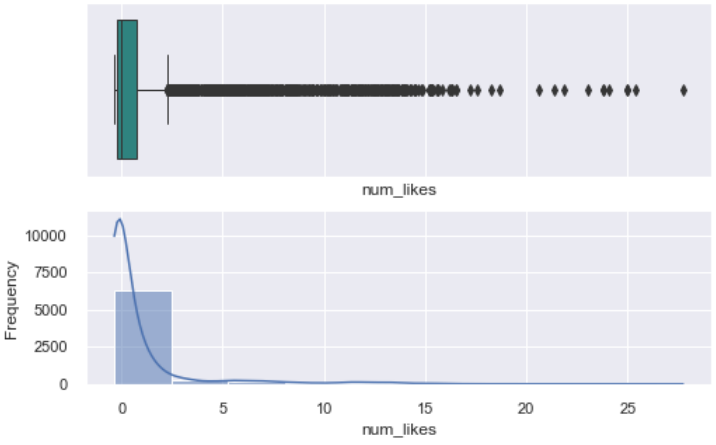

```
<Figure size 576x360 with 0 Axes>
```



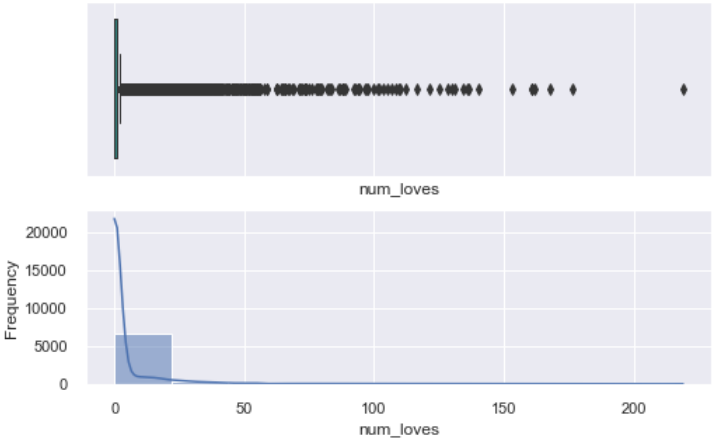
<Figure size 576x360 with 0 Axes>



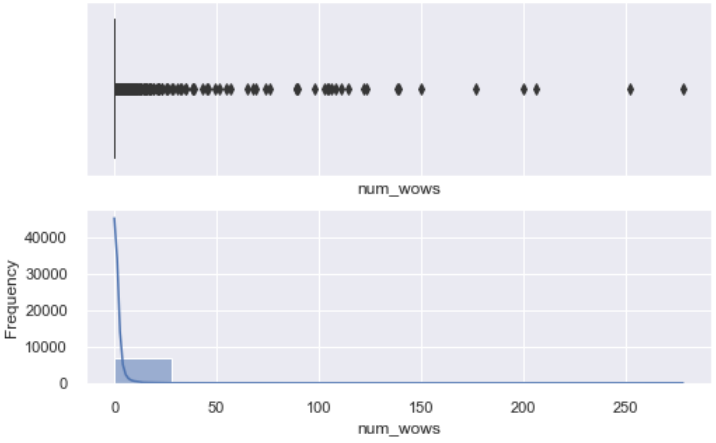
<Figure size 576x360 with 0 Axes>



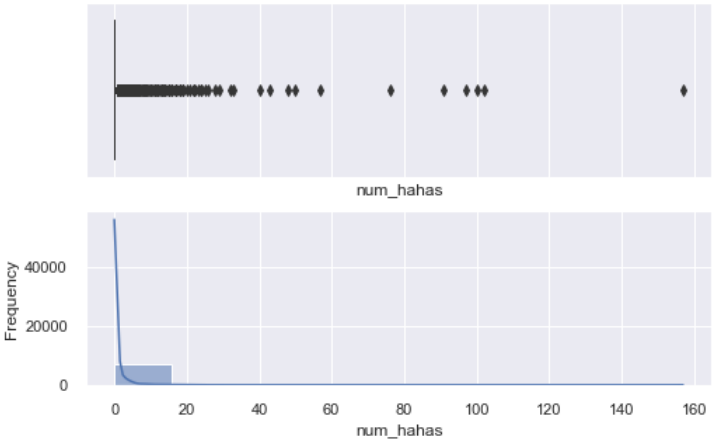
<Figure size 576x360 with 0 Axes>



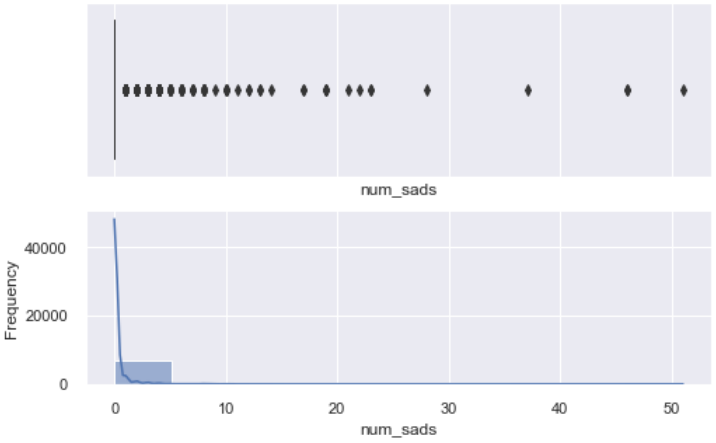
<Figure size 576x360 with 0 Axes>



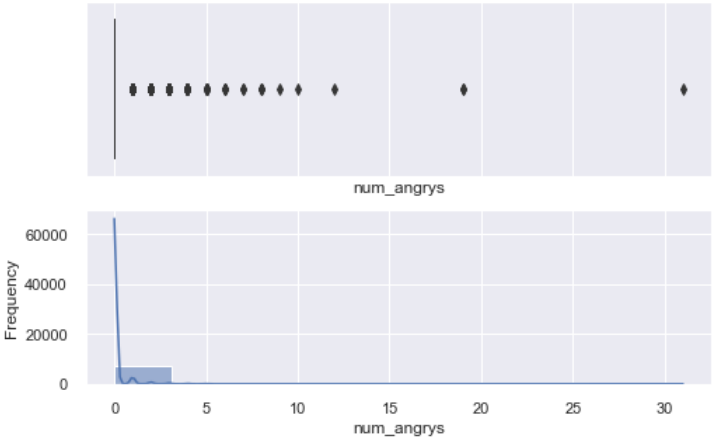

```
<Figure size 576x360 with 0 Axes>
```




```
<Figure size 576x360 with 0 Axes>
```




```
<Figure size 576x360 with 0 Axes>
```



- Después de aplicar RobustScaler hemos reducido la escla de la dispersión de las variables, pero no hemos solucionado de forma efectiva los outliers, y esto puede distosionar los resultados de la aplicación de los modelos de aprendizaje no supervisado, como es el caso de KMeans, que no se comporta bien cuando las variables presentan tantos outliers.

1.7 PCA (Principal Component Analysis)

1.7.1 PCA con todas las variables

In [235…
```
from sklearn.decomposition import PCA

X = data.drop(['status_type'],axis=1)
y = data['status_type']
```

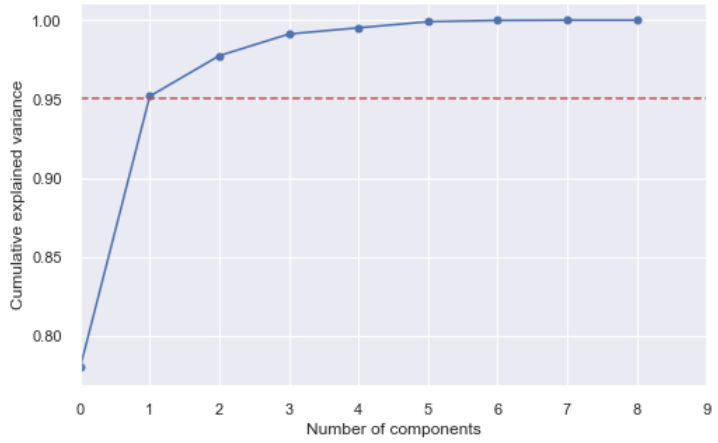
In [236…
```
X.head()
```

Out[236]:

	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	2.324257	22.086957	65.50	2.229508	30.666667	3.0	1.0	1.0	0.0
1	0.448020	-0.173913	0.00	0.548435	0.000000	0.0	0.0	0.0	0.0
2	0.829208	10.086957	14.25	0.870343	7.000000	1.0	1.0	0.0	0.0
3	0.254950	-0.173913	0.00	0.315946	0.000000	0.0	0.0	0.0	0.0
4	0.759901	-0.173913	0.00	0.870343	3.000000	0.0	0.0	0.0	0.0

In [237…
```
pca = PCA().fit(X)
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker = "o")
plt.axhline(0.95, color = "r", linestyle = "--")
plt.xlim([0, 9])
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

Out[237]:
```
Text(0, 0.5, 'Cumulative explained variance')
```



- En este caso, la primera componente explica el 95% de la varianza observada en los datos y las dos siguientes componentes, no superan el 10% de varianza explicada.

In [238…
```
pca = PCA(n_components=1)
pca.fit(X)
df_pca = pd.DataFrame(pca.transform(X), columns=['pca1'], index=data.index)
df_pca.head()
```

Out[238]:

	pca1
0	49.098473
1	-14.527011
2	3.545756
3	-14.531109
4	-13.895270

1.7.2 PCA sin Multicolinealidad

In [239…
```
X_sinMcl = data.drop(['status_type','num_reactions'],axis=1)
y_sinMcl = data['status_type']
```

In [240…
```
X_sinMcl.head()
```

Out[240]:

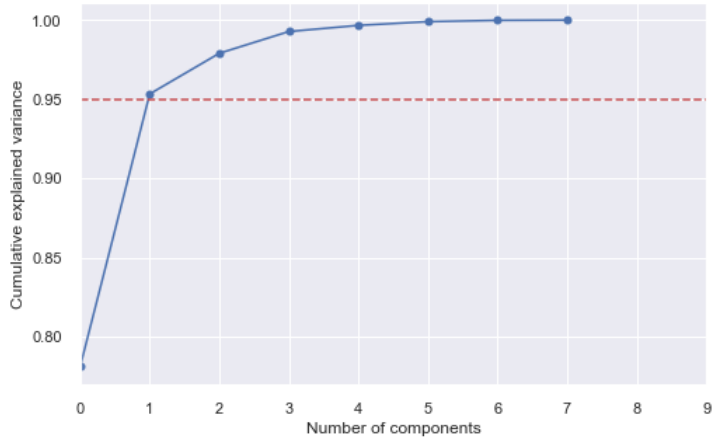
	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	22.086957	65.50	2.229508	30.666667	3.0	1.0	1.0	0.0
1	-0.173913	0.00	0.548435	0.000000	0.0	0.0	0.0	0.0
2	10.086957	14.25	0.870343	7.000000	1.0	1.0	0.0	0.0
3	-0.173913	0.00	0.315946	0.000000	0.0	0.0	0.0	0.0
4	-0.173913	0.00	0.870343	3.000000	0.0	0.0	0.0	0.0

In [241…

```
pca = PCA().fit(X_sinMcl)
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker = "o")
plt.axhline(0.95, color = "r", linestyle = "--")
plt.xlim([0, 9])
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

Out[241]:

```
Text(0, 0.5, 'Cumulative explained variance')
```



- Los resultados obenidos son similares a los obtenidos en el apartado anterior, pero no cambian los critetios de selección de componentes, ya que la primera componente acumul el 95% de la explicación de la varianza.
- Sin embargo, en este caso, parece que las tres siguientes componentes acumulan hasta el 100% de la explicación de la varianza, mientras en el caso anterior necesitabamos una componente adicional para alcanzar dicho porcentaje.

In [242…

```
pca_sin = PCA(n_components=1)
pca_sin.fit(X_sinMcl)
df_pca_sin = pd.DataFrame(pca_sin.transform(X_sinMcl), columns=['pca1'], index=data.index)
df_pca_sin.head()
```

Out[242]:

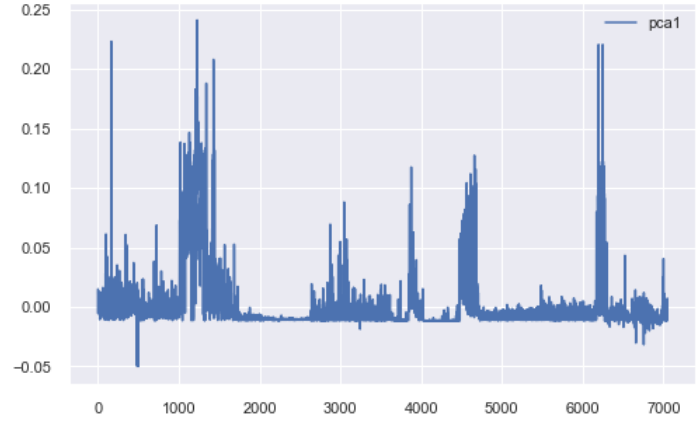
	pca1
0	49.083497
1	-14.523455
2	3.545969
3	-14.525461
4	-13.895130

In [243…

```
diferencias=df_pca-df_pca_sin
diferencias.plot()
```

Out[243]:

```
<AxesSubplot:>
```



- Como se observa en el gráfico, las diferencias en los valores en el PCA calculado sobre el total de variables y eliminando la variable que genera la multicolinealidad nos superan en ningun momento el rango de (-0.05, 0,25), por lo que utilizaremos el Data Frame completo.

1.8 Modelos

1.8.1 K-means

El algoritmo K-means (MacQueen, 1967) agrupa las observaciones en un número predefinido de K clusters de forma que, la suma de las varianzas internas de los clusters, sea lo menor posible. El algoritmo empleado sigue el siguiente proceso:

- Especificar el número K de clusters que se quieren crear.
- Seleccionar de forma aleatoria k observaciones del set de datos como centroides iniciales.
- Asignar cada una de las observaciones al centroide más cercano.
- Para cada uno de los K clusters generados en el paso 3, recalcular su centroide.
- Repetir los pasos 3 y 4 hasta que las asignaciones no cambien o se alcance el número máximo de iteraciones establecido.

Este algoritmo garantiza que, en cada paso, se reduzca la intra-varianza total de los clusters hasta alcanzar un óptimo local. Debido a que el algoritmo de K-means no evalúa todas las posibles distribuciones de las observaciones sino solo parte de ellas, los resultados obtenidos dependen de la asignación aleatoria inicial (paso 2). Por esta razón, es importante ejecutar el algoritmo varias veces (25-50), cada una con una asignación aleatoria inicial distinta, y seleccionar aquella que haya conseguido una menor varianza total.

Ventajas y desventajas

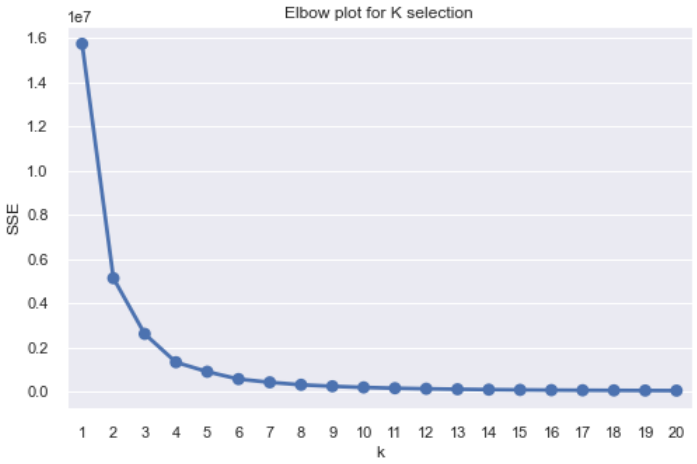
- K-means es uno de los métodos de clustering más utilizados. Destaca por la sencillez y velocidad de su algoritmo, sin embargo, presenta una serie de limitaciones que se deben tener en cuenta.
- Requiere que se indique de antemano el número de clusters que se van a crear. Esto puede ser complicado si no se dispone de información adicional sobre los datos con los que se trabaja. Se han desarrollado varias estrategias para ayudar a identificar potenciales valores óptimos de K (elbow, shilouette), pero todas ellas son orientativas.
- Dificultad para detectar clusters alargados o con formas irregulares.
- Las agrupaciones resultantes pueden variar dependiendo de la asignación aleatoria inicial de los centroides. Para minimizar este problema, se recomienda repetir el proceso de clustering entre 25-50 veces y seleccionar como resultado definitivo el que tenga menor suma total de varianza interna. Aun así, solo se puede garantizar la reproducibilidad de los resultados si se emplean semillas.
- Presenta problemas de robustez frente a outliers. La única solución es excluirlos o recurrir a otros métodos de clustering más robustos como K-medoids (PAM).

1.8.1 a) Número óptimo de Clusters con el Método Elbow

- El método Elbow, también conocido como método del codo, sigue una estrategia de probar un rango de valores del hiperparámetro en cuestión, representar gráficamente los resultados obtenidos con cada uno, e identificar aquel punto de la curva (codo) a partir del cual la mejora deja de ser notable.
- En los casos de partitioning clustering, como por ejemplo K-means, las observaciones se agrupan de una forma tal que se minimiza la varianza total intra-cluster.
- El método Elbow calcula la varianza total intra-cluster en función del número de clusters y escoge como óptimo aquel valor a partir del cual añadir más clusters apenas consigue mejoría.

In [244…
```
from sklearn.cluster import KMeans
from kneed import KneeLocator

def elbow_plot(data):
    """Create elbow plot from normalized data"""
    sse = {}
    sse_r = []
    for k in range(1, 21):
        kmeans = KMeans(n_clusters=k, random_state=1)
        kmeans.fit(data)
        sse[k] = kmeans.inertia_
        sse_r.append(kmeans.inertia_)

    plt.title('Elbow plot for K selection')
    plt.xlabel('k')
    plt.ylabel('SSE')
    sns.pointplot(x=list(sse.keys()),
                  y=list(sse.values()))
    plt.show()
    return sse_r
```

In [245…
```
sse = elbow_plot(df_pca) #Se ejecuta con PCA
```



In [246…
```
kl = KneeLocator(range(1, 21), sse, curve="convex", direction="decreasing")
kl.elbow
```

Out[246]:
```
4
```

1.8.1 b) KMeans Clusters

In [247…
```
k_means = KMeans(n_clusters=4, random_state=1)
k_means.fit(X)
```

Out[247]:
```
KMeans(n_clusters=4, random_state=1)
```

In [248…
```
y_pred = k_means.predict(X)
y_pred
```

Out[248]:
```
array([2, 0, 0, ..., 0, 0, 0])
```

In [249…
```
data['Cluster'] = y_pred
data.head()
```

Out[249]:

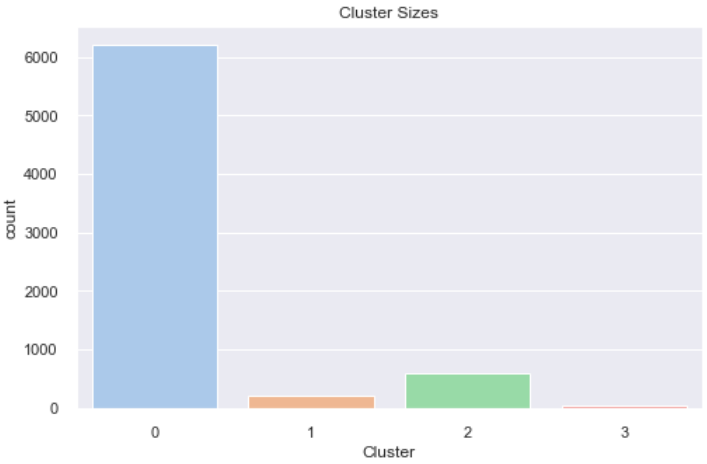
	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys	Cluster
0	3	2.324257	22.086957	65.50	2.229508	30.666667	3.0	1.0	1.0	0.0	2
1	1	0.448020	-0.173913	0.00	0.548435	0.000000	0.0	0.0	0.0	0.0	0
2	3	0.829208	10.086957	14.25	0.870343	7.000000	1.0	1.0	0.0	0.0	0
3	1	0.254950	-0.173913	0.00	0.315946	0.000000	0.0	0.0	0.0	0.0	0
4	1	0.759901	-0.173913	0.00	0.870343	3.000000	0.0	0.0	0.0	0.0	0

In [250…
```
df_pca["Cluster"] = k_means.labels_
```

In [251…
```
df_pca.Cluster.value_counts()
```

Out[251]:
```
0    6208
2     601
1     207
3      34
Name: Cluster, dtype: int64
```

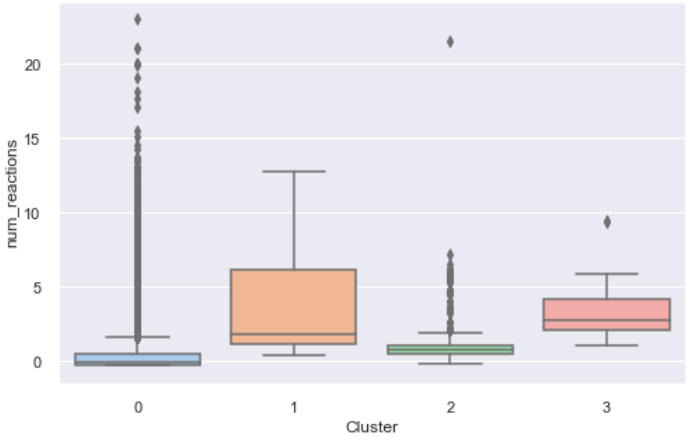
In [252…
```
sns.countplot(x = df_pca.Cluster, palette ="pastel").set(
    title = "Cluster Sizes")
plt.show()
```

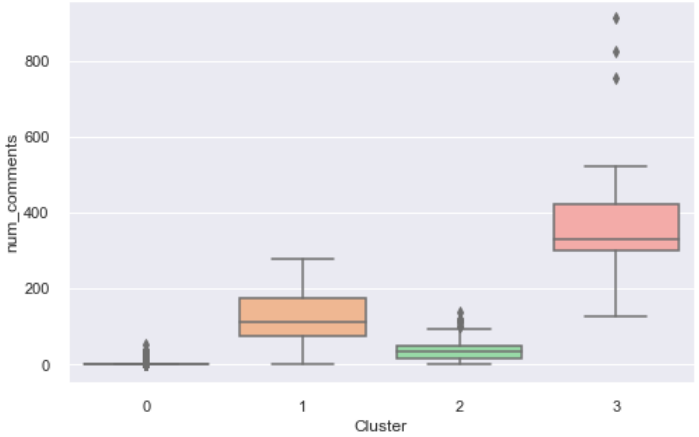


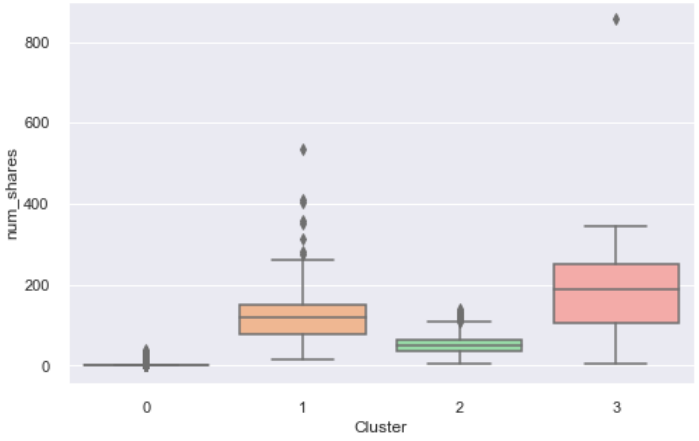
- La distribución de los valores del Data Frame entre los diferentes clusters no es homogénea, y el cluster 0 acumula el 88,05% de los valores.

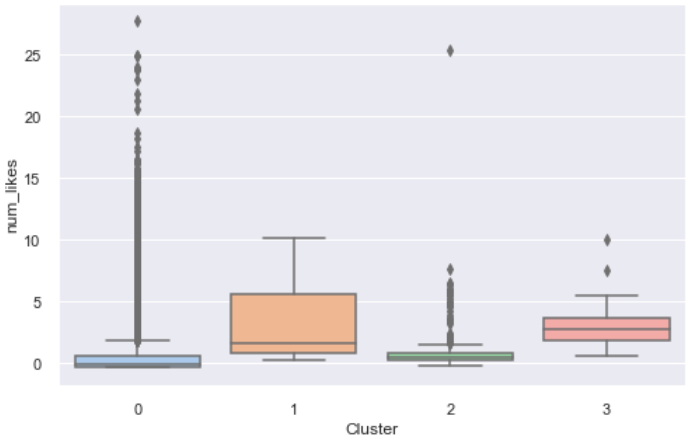
1.8.1 c) Visualizacíón Clusters en función de las variables

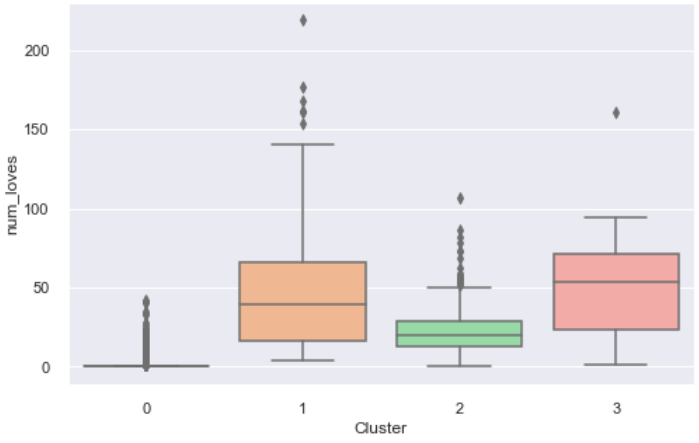
In [253…
```
columns = data.columns[1:10]

for i in columns:
    plt.figure()
    sns.set(rc={"figure.figsize":(8, 5)})
    sns.boxplot(x='Cluster', y=i, data=data, palette="pastel")
```

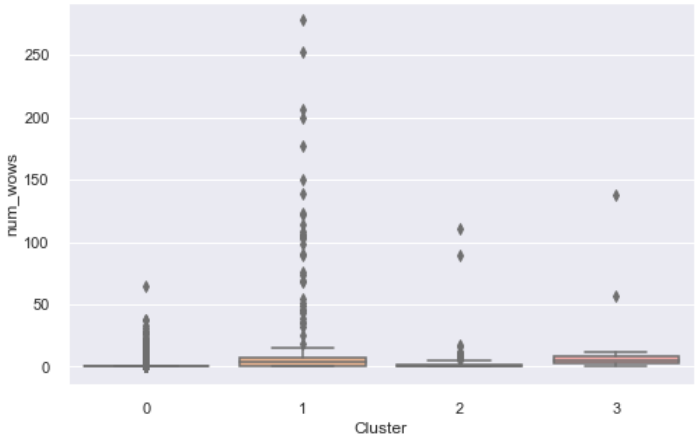


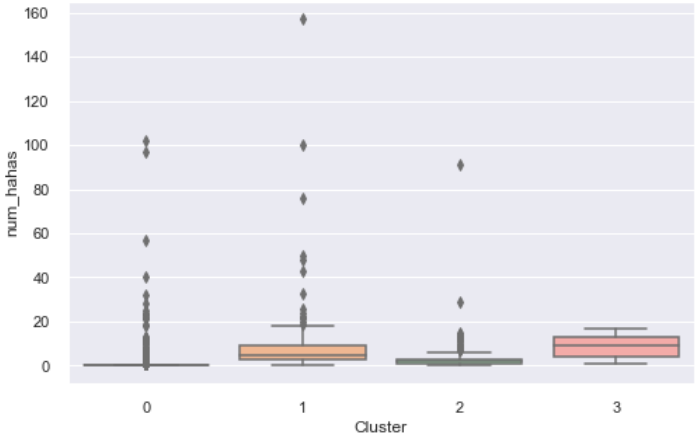


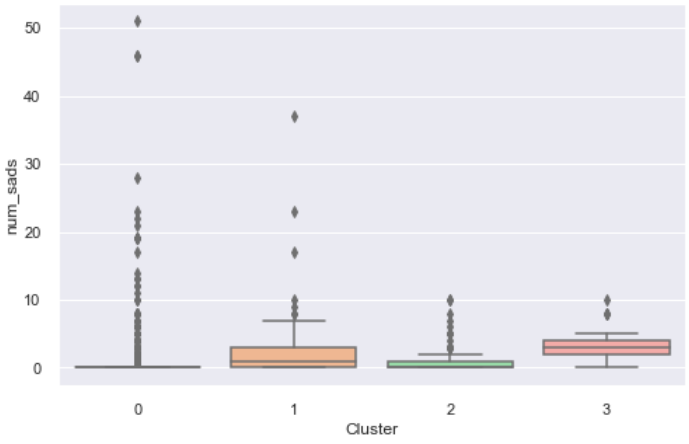

num_shares
800
600
400
200
0
0
1
2
3
Cluster

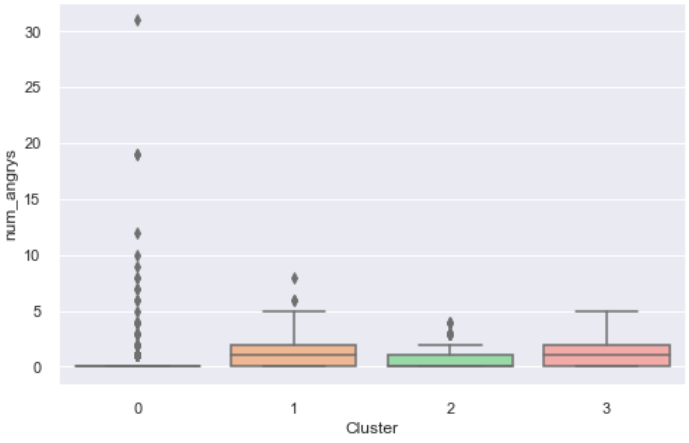

num_likes
25
20
15
10
5
0
0
1
2
3
Cluster


num_loves
200
150
100
50
0
0
1
2
3
Cluster


num_wows
250
200
150
100
50
0
0
1
2
3
Cluster



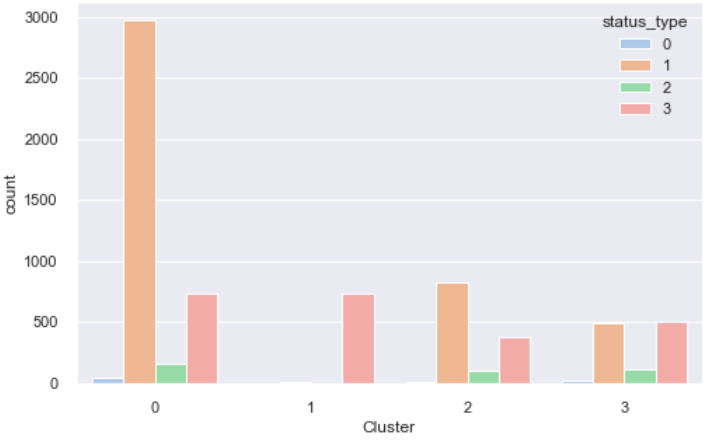




- De las gráficas de boxplot de las diferentes variables podemos observar que las primeras: num_reactions, num_comments, num_shares, num_loves y num_likes tienen valores de mediana y de desviación estándar bastante diferntes entre los distintos clusters; el resto de variaales sin embargo no presentan grandes diferencias en la distribución de los cuartiles.
- Otro factor destacable es la existencia de numerosos outliers en el clúster 0 y 1, en la mayoría de las variables, lo que probablemente está distorsionando la clasificación de los valores en los diferentes clústers.

In [327…
```
ax = sns.countplot(x="Cluster", hue="status_type", data=data, palette="pastel")
#place legend in upper left of plot
plt.legend(loc='upper right', title="status_type")
```

Out[327]:
```
<matplotlib.legend.Legend at 0x1f44b44ec70>
```

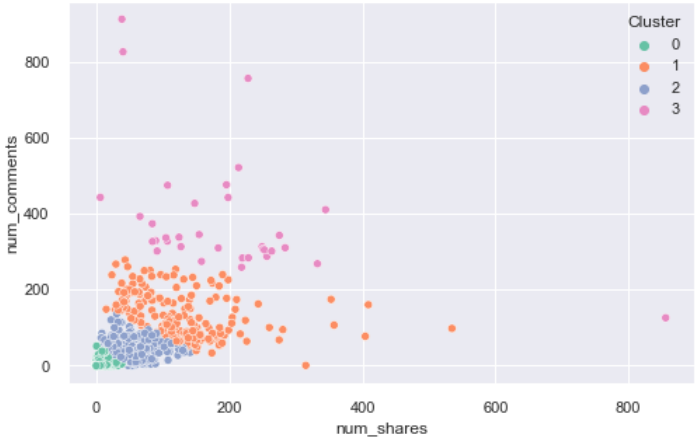


- El cluster 0 concentra el mayor número de datos de la variable status_type, en sus cuatro categorías: link, photo, status y video.
- El cluster 1 solo tiene datos de videos y los cluster 2 y 3 tienen datos del tipo photo, status y video.

1.8.1 d) Visualización de datos los Clusters por pares de variables

In [256…
```
sns.scatterplot(x = 'num_shares', y = 'num_comments',hue='Cluster',data =data,legend='full',palette="Set2")
```

Out[256]:
```
<AxesSubplot:xlabel='num_shares', ylabel='num_comments'>
```



- Los clusters 0,1 y 2 aparecen definidos entre los valores de las variables num_comments y num_shares, el cluster 3 si bien está diferenciado del resto, tiene una mayor dispersión de valores.

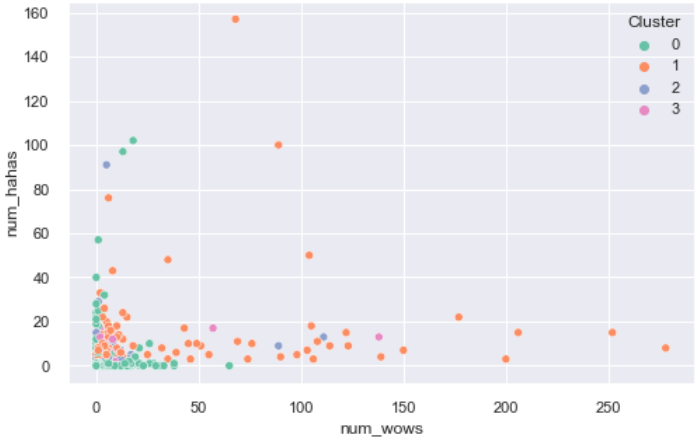
In [257…
```
sns.scatterplot(x = 'num_likes', y = 'num_loves',hue='Cluster',data =data,legend='full',palette="Set2")
```

Out[257]:
```
<AxesSubplot:xlabel='num_likes', ylabel='num_loves'>
```

- Entre las variables num_loves y num_likes también aparecen definidos los clusters 0 y 1, pero los clusters 2 y 3 se intercalan en el area de otros clusters.

In [258…
```
sns.scatterplot(x = 'num_wows', y = 'num_hahas',hue='Cluster',data =data,legend='full',palette="Set2")
```

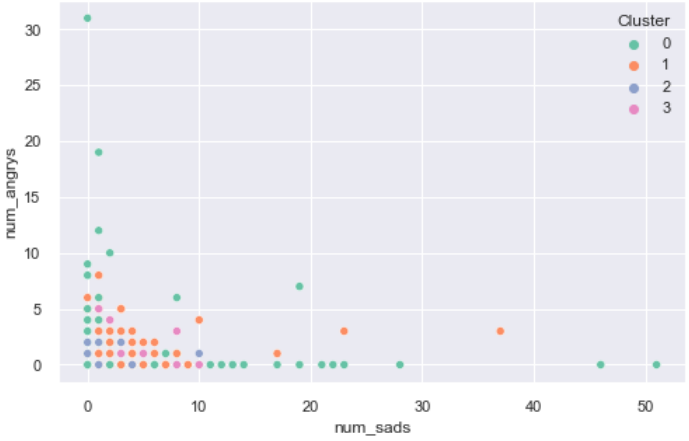
Out[258]:
```
<AxesSubplot:xlabel='num_wows', ylabel='num_hahas'>
```



- En el gráfico de las variables num_hahas y num_wows no se aprecia una diferencia significativa entre las posiciones de los diferentes Clusters

In [259…
```
sns.scatterplot(x = 'num_sads', y = 'num_angrys',hue='Cluster',data =data,legend='full',palette="Set2")
```

Out[259]:
```
<AxesSubplot:xlabel='num_sads', ylabel='num_angrys'>
```



- Igual que en caso anterior, no hay diferencias en la distribución de los valores de los clusters para estas variables, num_angrys y num_sads, y se debe al menor número de datos de las mismas informados en el data frame, por lo que su influencia en la clasificación de los cluster es menor.

1.8.2 K-medoids

K-medoids es un método de clustering muy similar a K-means en cuanto a que ambos agrupan las observaciones en K clusters, donde K es un valor preestablecido por el analista. La diferencia es que, en K-medoids, cada cluster está representado por una observación presente en el cluster (medoid), mientras que en K-means cada cluster está representado por su centroide, que se corresponde con el promedio de todas las observaciones del cluster, pero con ninguna en particular.

Una definición más exacta del término medoid es: elemento dentro de un cluster cuya distancia promedio entre él y todos los demás elementos del mismo cluster es lo menor posible. Se corresponde con el elemento más central del cluster y por lo tanto puede considerarse como el más representativo. El hecho de utilizar "medoids" en lugar de centroides hace de K-medoids un método más robusto que K-means, viéndose menos afectado por outliers o ruido.

In [260…
```
from sklearn_extra.cluster import KMedoids
kmedoids = KMedoids(n_clusters=4, random_state=1).fit(X)
kmedoids.labels_
```

Out[260]:
```
array([1, 2, 3, ..., 0, 2, 0], dtype=int64)
```

In [261…
```
kmedoids.cluster_centers_
```

Out[261]:
```
array([[-0.21039604, -0.13043478,  0.        , -0.24441133,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ],
       [ 1.16584158, 46.17391304, 63.5       ,  0.91803279, 26.33333333,
         0.        ,  3.        ,  0.        ,  1.        ],
       [ 0.63118812,  0.26086957,  0.5       ,  0.75707899,  0.66666667,
         0.        ,  0.        ,  0.        ,  0.        ],
       [ 2.85891089,  2.60869565,  2.        ,  3.39195231,  2.66666667,
         2.        ,  0.        ,  0.        ,  0.        ]])
```

In [262…
```
df2_pca=df_pca.drop("Cluster",axis=1)
df2_pca["Cluster"] = kmedoids.labels_
```

```
df2_pca.head()
```

Out[262]:

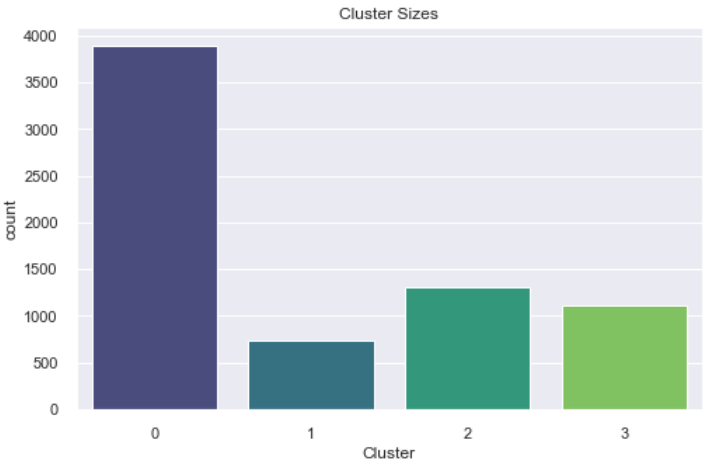
	pca1	Cluster
0	49.098473	1
1	-14.527011	2
2	3.545756	3
3	-14.531109	0
4	-13.895270	2

1.8.2 a) KMedoids Clusters

In [263…
```
df2_pca.Cluster.value_counts()
```

Out[263]:
```
0    3896
2    1308
3    1109
1     737
Name: Cluster, dtype: int64
```

In [264…
```
sns.countplot(x = df2_pca.Cluster, palette ="viridis").set(title = "Cluster Sizes")
plt.show()
```



- La distribución de los Cluster en este csso es un poco más homogénea entre el 1,2 y 3, pero el cluster 0 sigue acumulando el 55,26% de los datos

In [265…
```
data = data.drop("Cluster", axis=1)
data['Cluster'] = kmedoids.labels_
data.head()
```

Out[265]:

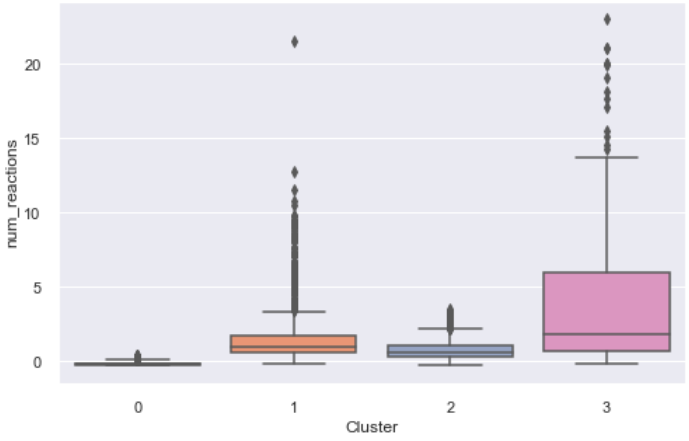
	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys	Cluster
0	3	2.324257	22.086957	65.50	2.229508	30.666667	3.0	1.0	1.0	0.0	1
1	1	0.448020	-0.173913	0.00	0.548435	0.000000	0.0	0.0	0.0	0.0	2
2	3	0.829208	10.086957	14.25	0.870343	7.000000	1.0	1.0	0.0	0.0	3
3	1	0.254950	-0.173913	0.00	0.315946	0.000000	0.0	0.0	0.0	0.0	0
4	1	0.759901	-0.173913	0.00	0.870343	3.000000	0.0	0.0	0.0	0.0	2

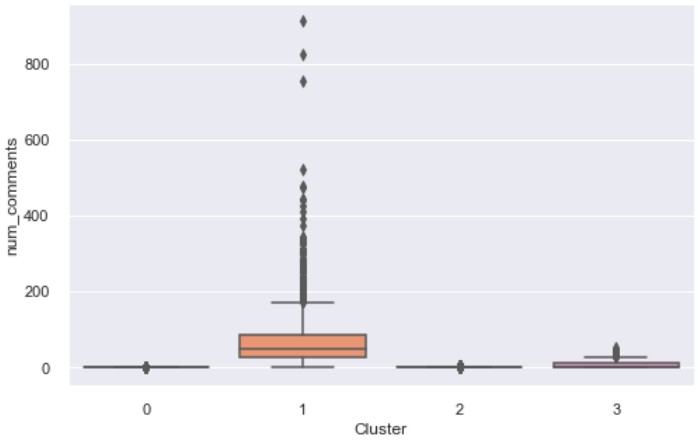
In [266…
```
data.Cluster.value_counts() # comprobación de la asignación de cluster en el Data Frame
```

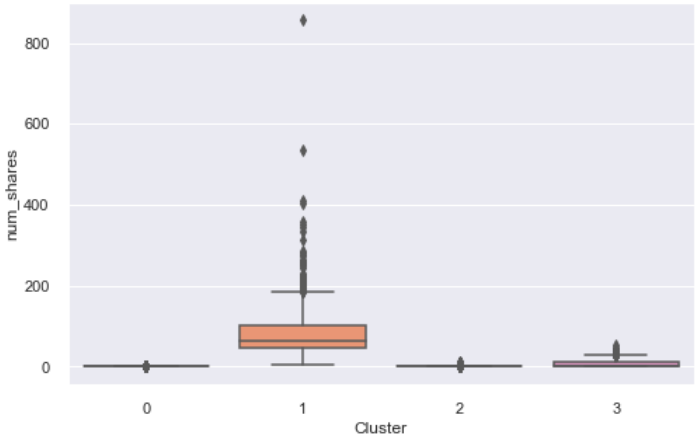
Out[266]:
```
0    3896
2    1308
3    1109
1     737
Name: Cluster, dtype: int64
```

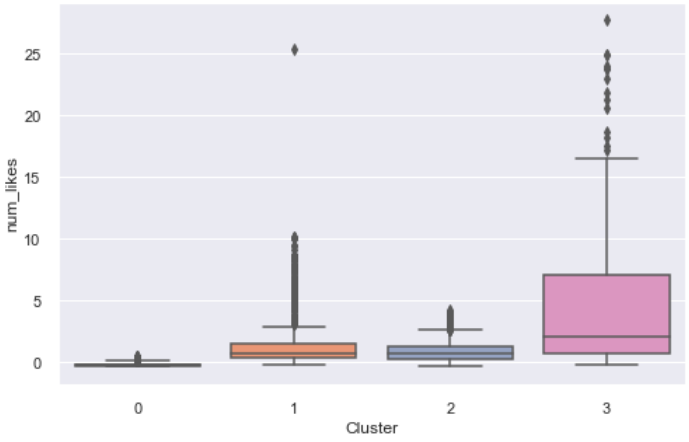
1.8.2 b) Visualización Clusters en función de las variables

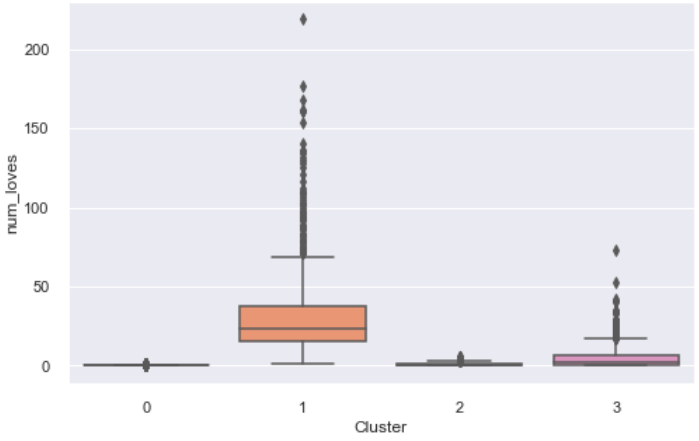
In [333…
```
columns = data.columns[1:10]

for i in columns:
    plt.figure()
    sns.set(rc={"figure.figsize":(8, 5)})
    sns.boxplot(x='Cluster', y=i, data=data, palette="Set2")
```

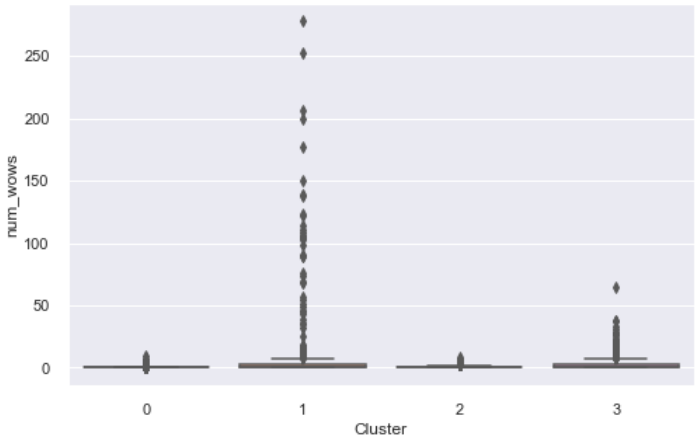

num_reactions
20
15
10
5
0
0
1
2
3
Cluster

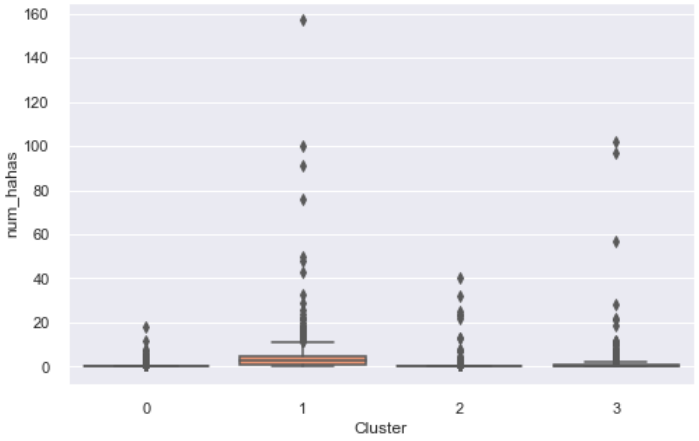

num_comments
800
600
400
200
0
0
1
2
3
Cluster

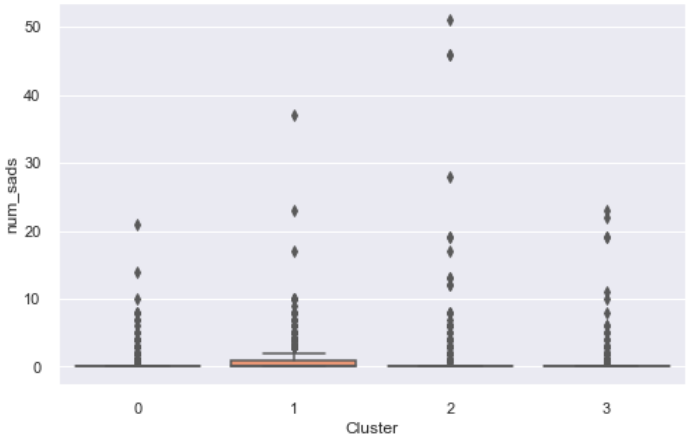

num_shares
800
600
400
200
0
0
1
2
3
Cluster

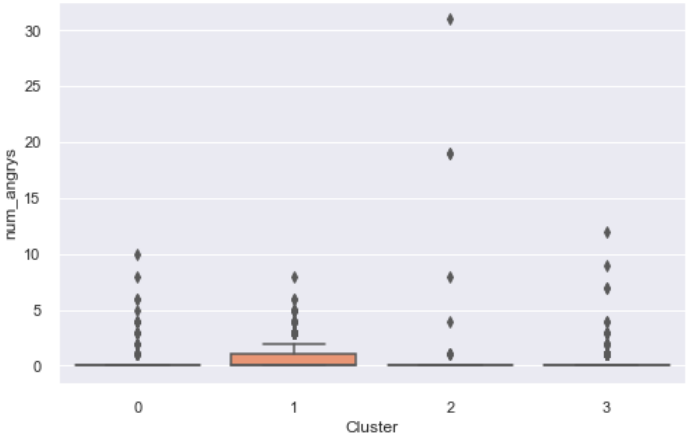

num_likes
25
20
15
10
5
0
0
1
2
3
Cluster


num_loves
200
150
100
50
0
0
1
2
3
Cluster


num_wows
250
200
150
100
50
0
0
1
2
3
Cluster


num_hahas
160
140
120
100
80
60
40
20
0
0
1
2
3
Cluster

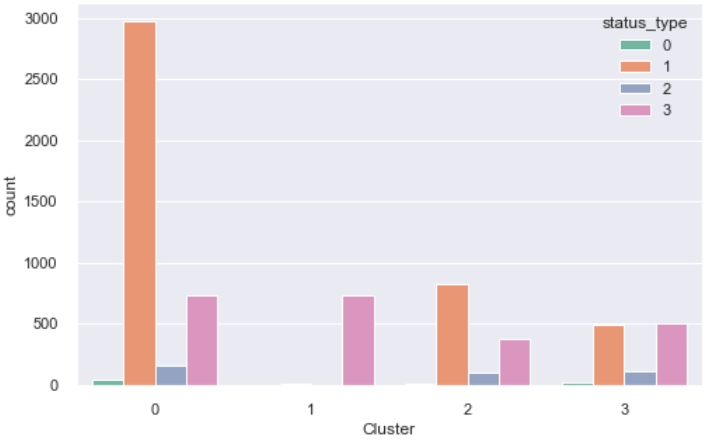

num_sads
50
40
30
20
10
0
0
1
2
3
Cluster



- Como en el modelo anterior, existen variables en las cuales las gráficas boxplot son significativamente diferentes entre clusters, pero siguen produciéndose sistorsiones derivadas de numerosos outliers.

In [334…]
```
ax = sns.countplot(x="Cluster", hue="status_type", data=data, palette="Set2")
#place legend in upper left of plot
plt.legend(loc='upper right', title="status_type")
```

Out[334]:
```
<matplotlib.legend.Legend at 0x1f44b67a6a0>
```



- Como en el modelo anterior, el cluster 0 es el que tiene valores de status_type de sus cuatro tipos: like, photo, status y video.
- Con este nuevo modelo, también los clusters 2 y 3 cuentan con un mayor número de elementos del tipo: photo, status y vídeo.
- Tambíen el cluster 1 tiene asignados más elementos del tipo video de la variable status_type.

Exercici 2

Classifica els diferents registres utilitzant l'algorisme de clustering jeràrquic.

2.1 Selección de una muestra del DataFrame

In [335…]
```
df_sample = data.sample(frac = 0.5, random_state = 0)
```

In [336…]
```
from sklearn.decomposition import PCA

X_sample = df_sample.drop(['status_type'],axis=1)
y_sample = df_sample['status_type']
```

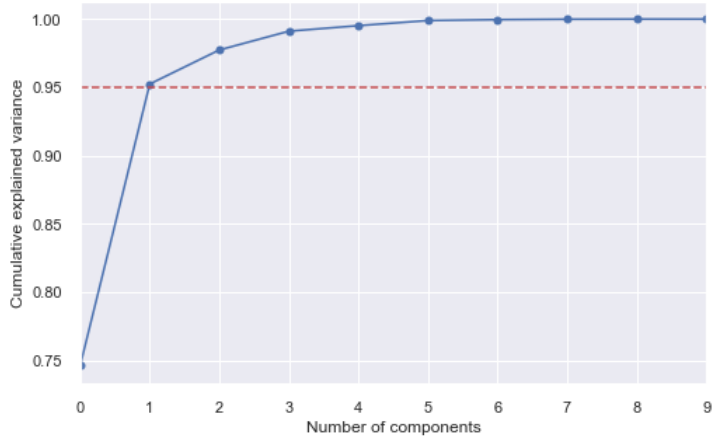
In [337…]
```
X_sample.shape
```

Out[337]:
```
(3525, 10)
```

2.2 PCA (Principal Components Analysis)

In [338…]
```
pca = PCA().fit(X_sample)
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker = "o")
plt.axhline(0.95, color = "r", linestyle = "--")
plt.xlim([0, 9])
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

Out[338]:
```
Text(0, 0.5, 'Cumulative explained variance')
```



También en este caso, la primera componente explica el 95% de la varianza observada en los datos y las dos siguientes componentes, no superan el 10% de varianza explicada.

In [339…
```
pca = PCA(n_components=1)
pca.fit(X_sample)
df_pca_agg = pd.DataFrame(pca.transform(X_sample), columns=['pca1'], index=df_sample.index) #'pca2','pca3','pca4'
df_pca_agg.head()
```

Out[339]:

	pca1
4412	-15.108972
4565	236.610688
5660	-15.074925
2724	-15.037243
6649	-14.362794

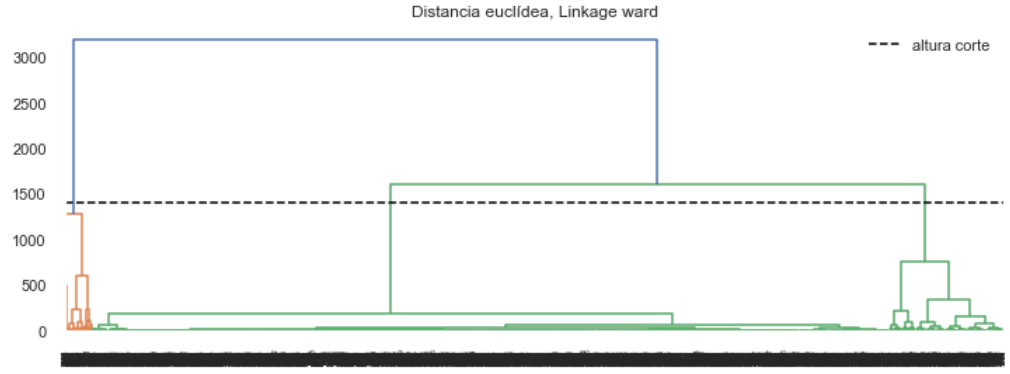
In [347…
```
df_pca_agg.shape
```

Out[347]:
```
(3525, 2)
```

2.3 Dendrograma

Un dendrograma es un diagrama en forma de árbol que se utiliza para visualizar la relación entre grupos. Cuanto mayor sea la distancia de las líneas verticales en el dendrograma, mayor será la distancia entre esos grupos. La clave para interpretar un dendrograma es concentrarse en la altura a la que se unen dos objetos.

In [348…
```
import scipy.cluster.hierarchy as shc

fig, ax = plt.subplots(1, 1, figsize=(12, 4))
altura_corte = 1400
shc.dendrogram(shc.linkage(df_pca_agg, method = "ward"), ax=ax)
ax.set_title("Distancia euclídea, Linkage ward")
ax.axhline(y=altura_corte, c = 'black', linestyle='--', label='altura corte')
ax.legend();
```



- Hemos seleccionado tres clusters, si bien podrían ser dos si seguimos la clasificación por colores, pero dada la tendencia a concentrar los datos en el cluster 0, suponemos que 3 clusters aportarán mas información a la clasificación, si bien no tienen porqué ser el número óptimo de clusters.

2.4 AgglomerativeClustering Model

In [349…
```
from sklearn.cluster import AgglomerativeClustering
```

```
agglo = AgglomerativeClustering(n_clusters = 3, affinity = "euclidean", linkage = "ward")
y_pred_agg= agglo.fit_predict(df_pca_agg)
y_pred_agg
```

Out[349]:
```
array([2, 0, 2, ..., 2, 2, 2], dtype=int64)
```

In [350…
```
df_sample['Cluster'] = y_pred_agg
df_sample.head()
```

Out[350]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys	Cluster
4412	3	-0.289604	-0.173913	0.00	-0.339791	0.000000	0.0	0.0	0.0	0.0	2
4565	3	10.779703	111.608696	203.75	9.138599	125.333333	252.0	15.0	1.0	2.0	0
5660	1	-0.254950	-0.130435	0.00	-0.298063	0.000000	0.0	0.0	0.0	0.0	2
2724	1	-0.254950	-0.173913	0.00	-0.304024	0.333333	0.0	0.0	0.0	0.0	2
6649	1	-0.012376	0.260870	0.25	-0.029806	0.000000	4.0	0.0	0.0	0.0	2

In [351…
```
df_pca_agg["Cluster"] = agglo.labels_
```

In [352…
```
df_pca_agg.head()
```

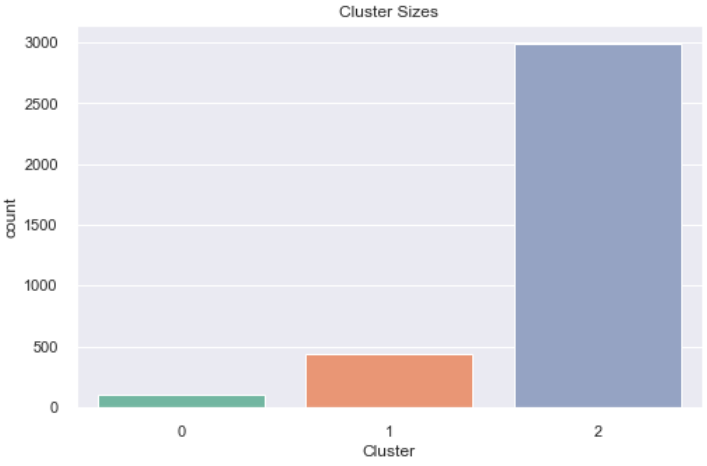
Out[352]:

	pca1	Cluster
4412	-15.108972	2
4565	236.610688	0
5660	-15.074925	2
2724	-15.037243	2
6649	-14.362794	2

In [353…
```
df_pca_agg.Cluster.value_counts()
```

Out[353]:
```
2    2991
1     431
0     103
Name: Cluster, dtype: int64
```

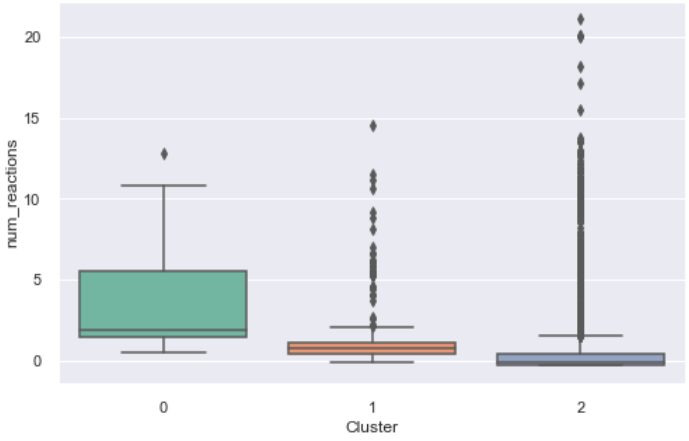
In [354…
```
sns.countplot(x = df_pca_agg.Cluster, palette = "Set2").set(title = "Cluster Sizes")
plt.show()
```

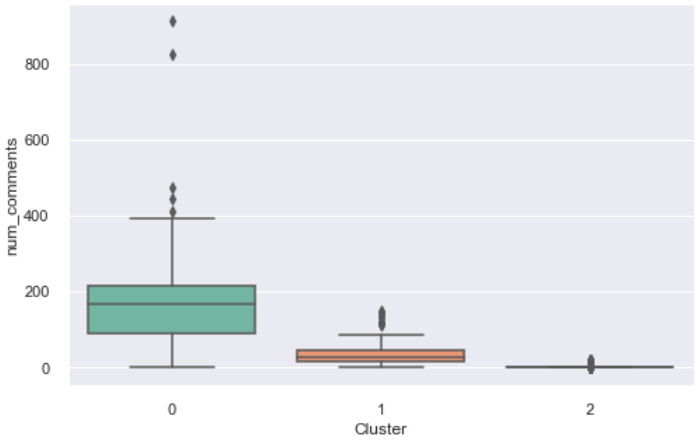


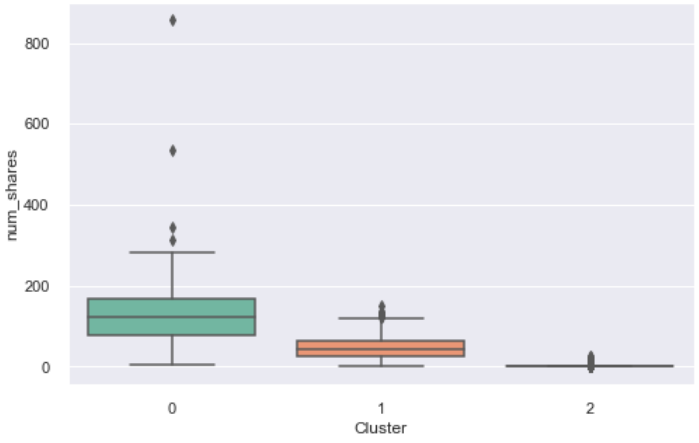
- De nuevo la clasificación de valores se concentra en un mismo cluster, pero en este caso se corresponde con el cluster 2, que acumula el 84,85% de los datos.

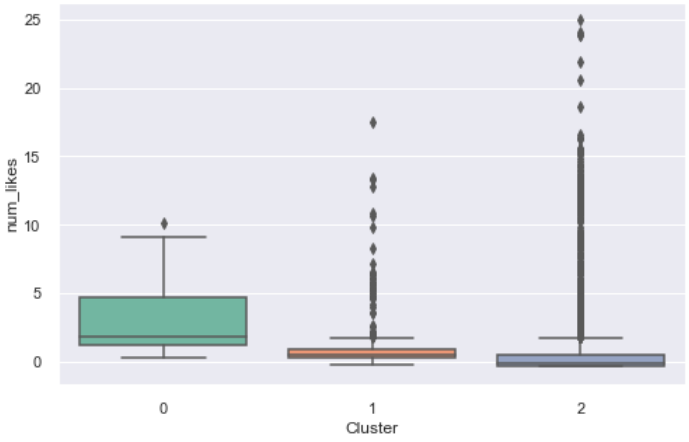
2.5 Visualizacíon Clusters en función de las variables

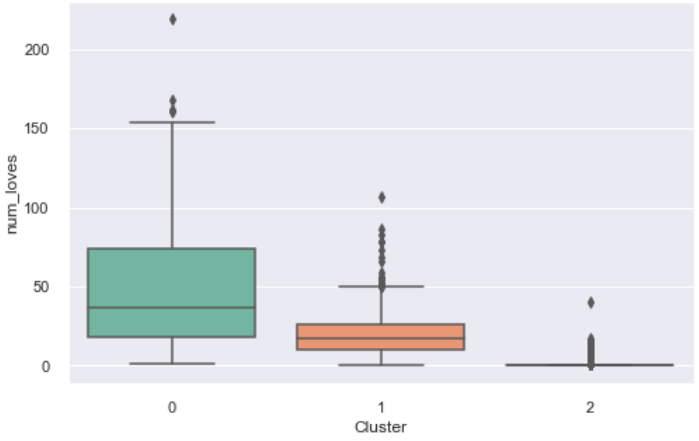
In [355…
```
columns = df_sample.columns[1:10]

for i in columns:
    plt.figure()
    sns.set(rc={"figure.figsize":(8, 5)})
    sns.boxplot(x='Cluster', y=i, data=df_sample, palette="Set2")
```

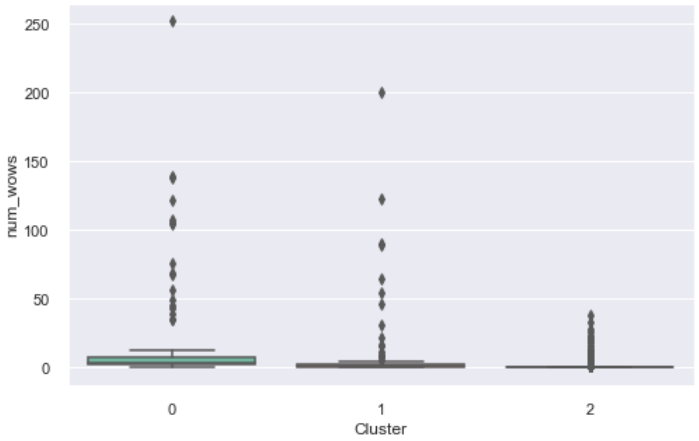

num_reactions
20
15
10
5
0
0
1
2
Cluster

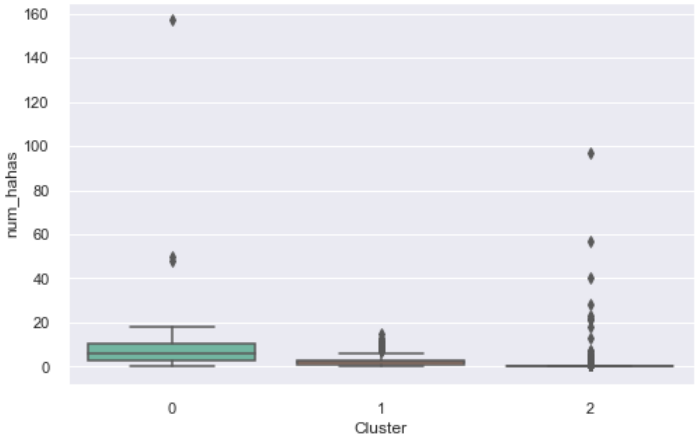

num_comments
800
600
400
200
0
0
1
2
Cluster

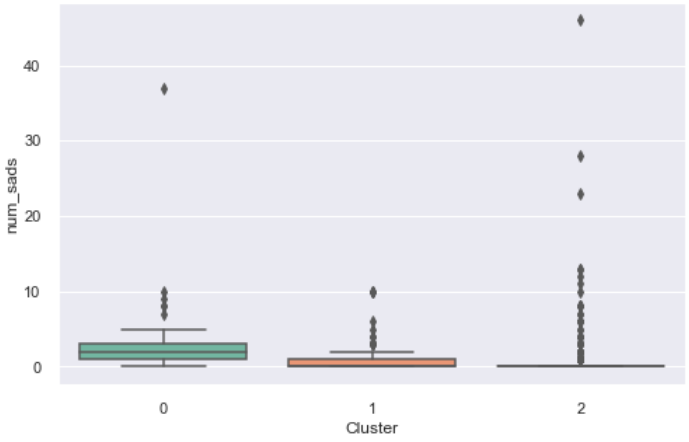

num_shares
800
600
400
200
0
0
1
2
Cluster

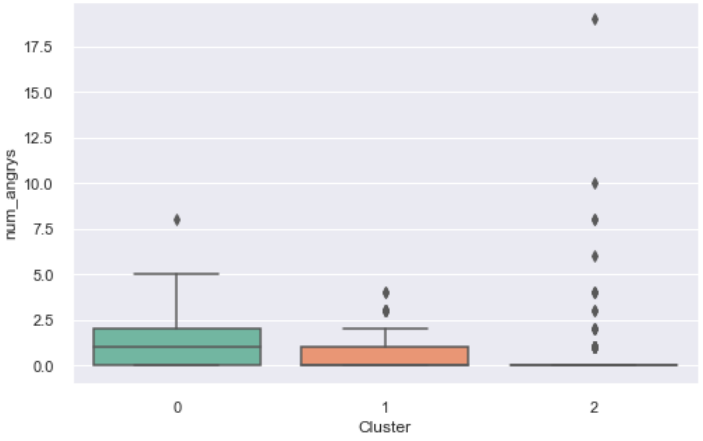

num_likes
25
20
15
10
5
0
0
1
2
Cluster


num_loves
200
150
100
50
0
0
1
2
Cluster


num_wows
250
200
150
100
50
0
0
1
2
Cluster


num_hahas
160
140
120
100
80
60
40
20
0
0
1
2
Cluster

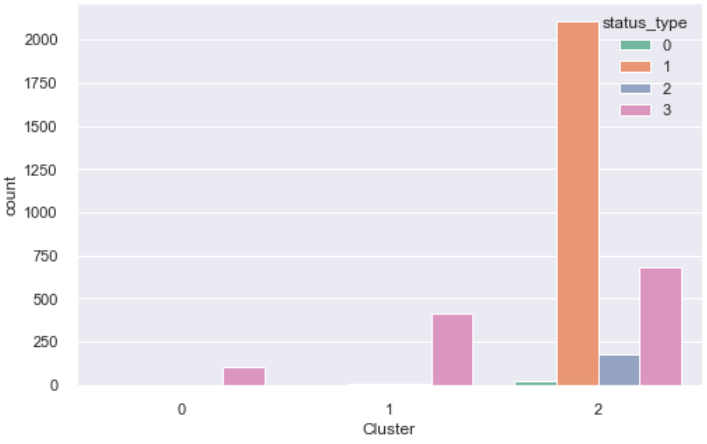

num_sads
40
30
20
10
0
0
1
2
Cluster



- En general la visualización de los datos de cada variable por cada cluster ofrece una representación más homogénea, aunque siguen apareciendo numerosos ouliers, pero tiene menos efecto en la distribución de los datos en relación a la mediana.
- A difenrencia de las clasificaciones anteriores, los outliers se concentran más en el custer 2, que es el que tiene más datos, de la misma forma que en los modelos anteriores sucedía con el cluster 0.

In [356…
```
ax = sns.countplot(x="Cluster", hue="status_type", data=df_sample, palette="Set2")
#place legend in upper left of plot
plt.legend(loc='upper right', title="status_type")
```

Out[356]:
```
<matplotlib.legend.Legend at 0x1f45254ea00>
```



- En esta clasificación el cluster 2 tiene datos con las cuatro características de la variable status_type: link, photo, status y video, mientras que los clusters 0 y 1 solo tienen datos correspondientes al tipo video.

Exercici 3

Calcula el rendiment del clustering mitjançant un paràmetre com pot ser silhouette.

El método de average silhouette considera como número óptimo de clusters aquel que maximiza la media del silhouette coeficient de todas las observaciones. El silhouette coeficient (si) cuantifica cómo de buena es la asignación que se ha hecho de una observación comparando su similitud con el resto de observaciones de su cluster frente a las de los otros clusters. Su valor puede estar entre -1 y 1, siendo valores próximos a 1 un indicativo de que la observación se ha asignado al cluster correcto.

Para cada observación i , el silhouette coeficient (si) se obtiene del siguiente modo:

- Calcular el promedio de las distancias (llámese ai) entre la observación i y el resto de observaciones que pertenecen al mismo cluster. Cuanto menor sea ai , mejor ha sido la asignación de i a su cluster.
- Calcular la distancia promedio entre la observación i y el resto de clusters. Entendiendo por distancia promedio entre i y un determinado cluster C como la media de las distancias entre i y las observaciones del cluster C .
- Identificar como bi a la menor de las distancias promedio entre i y el resto de clusters, es decir, la distancia al cluster más próximo (neighbouring cluster).
- Calcular el valor de silhouette como: si= (bi−ai)/max(ai,bi)

Se considera como número óptimo de clusters aquel que maximiza la media del silhouette coeficient de todas las observaciones.

3.1 Silhouette - Modelo KMeans

In [357…
```
from sklearn.metrics import silhouette_score
kmeans_silhouette = silhouette_score(df_pca, k_means.labels_).round(5)
```

In [358…
```
kmeans_silhouette
```

Out[358]:
```
0.88853
```

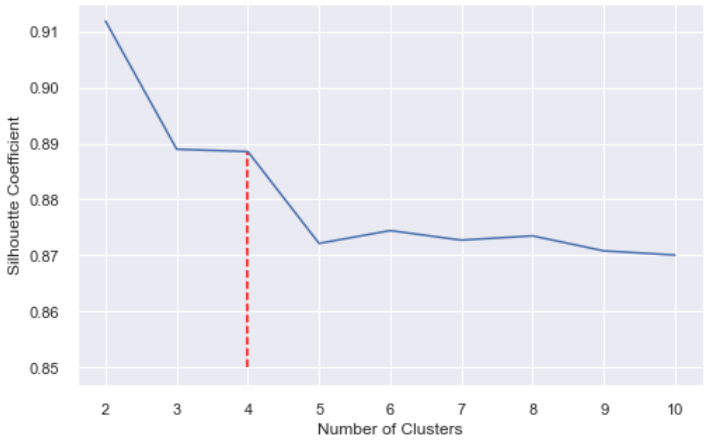
In [359…
```
data.head(3)
```

Out[359]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys	Cluster
0	3	2.324257	22.086957	65.50	2.229508	30.666667	3.0	1.0	1.0	0.0	1
1	1	0.448020	-0.173913	0.00	0.548435	0.000000	0.0	0.0	0.0	0.0	2
2	3	0.829208	10.086957	14.25	0.870343	7.000000	1.0	1.0	0.0	0.0	3

In [360…
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# A list holds the silhouette coefficients for each k
silhouette_coefficients = []

# silhouette starts at 2 clusters!!
for k in range(2,11):
    kmeans = KMeans(n_clusters= k, random_state=1)
    kmeans.fit(X) #data.drop(['status_type'],axis=1))
    score = silhouette_score(df_pca, kmeans.labels_)
    silhouette_coefficients.append(score)
```

In [361…
```
# plot the previous result
plt.plot(range(2, 11), silhouette_coefficients)
plt.xticks(range(2, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.vlines(x= 4, ymax = 0.88853, ymin= 0.85,  color = 'red', linestyles = '--')
plt.grid(True)
plt.show()
```



- La gráfica del coeficiente de Silhouette indica que el número óptimo de clúster es 2 con un coeficiente de 0.91, por tanto superior al que hemos utiizado en nuesto modelo que ha sido de 4 clústers.
- Según la gráfica no existe diferencia en cuanto al valor del coeficiente Silhouette si en el modelo se calcula con 3 ó 4 clusters.

3.2 Silhouette - KMedoids

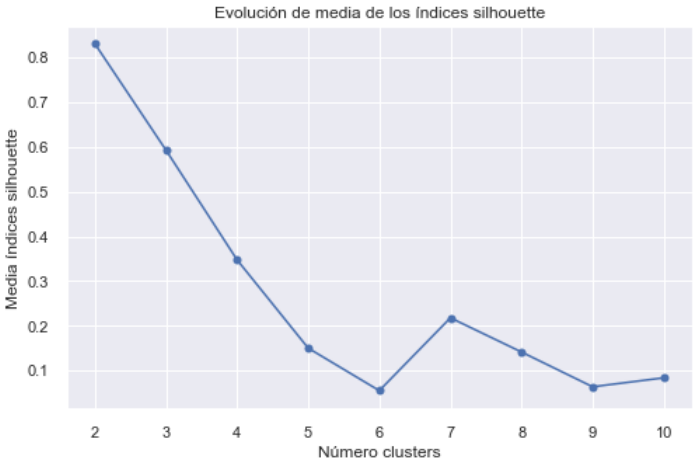
In [362…
```
kmed_silhouette = silhouette_score(df2_pca,kmedoids.labels_).round(5)
kmed_silhouette
```

Out[362]:
```
0.55464
```

- El resultado del coeficiente silhouette es particularmente bajo en este caso, ya que para la valoración del mismo hemos asignado 4 clusters para poder comparar con el modelo Kmeans.
- Vamos a evaluar cómo se comporta el coeficinete de silhoutte para estimar el óptimo de clusters que optimizan el modelo Kmedoids.

In [363…
```
# Método silhouette para identificar el número óptimo de clusters
# ==============================================================================
range_n_clusters = range(2, 11)
valores_medios_silhouette = []

for n_clusters in range_n_clusters:
    modelo = KMedoids(n_clusters=n_clusters, random_state=1)
    cluster_labels = modelo.fit_predict(X)
    silhouette_avg = silhouette_score(X, cluster_labels)
    valores_medios_silhouette.append(silhouette_avg)

fig, ax = plt.subplots(1, 1, figsize=(8, 5))
```

```
ax.plot(range_n_clusters, valores_medios_silhouette, marker='o')
ax.set_title("Evolución de media de los índices silhouette")
ax.set_xlabel('Número clusters')
ax.set_ylabel('Media índices silhouette');
```



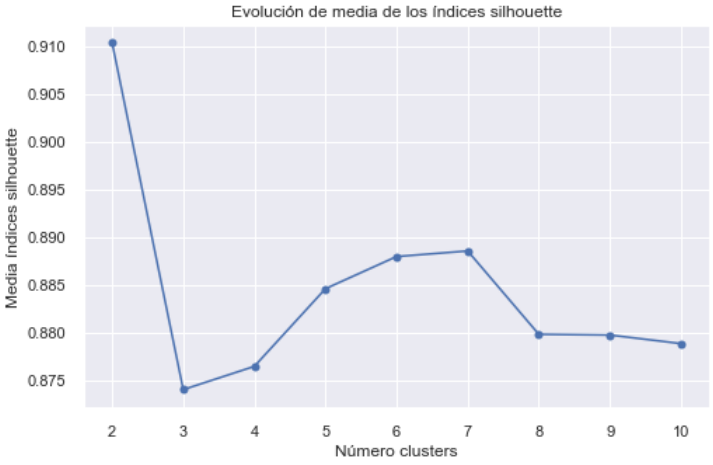
- Como en el caso anterior el número optimo de clusters es 2 y el valor del coeficiente silhouette se sitúa en 0.8

3.3 Silhouette - Modelo AgglomerativeClustering

In [364…
```
agglo_silhouette = silhouette_score(df_pca_agg, agglo.labels_).round(5)
agglo_silhouette
```

Out[364]:
```
0.87408
```

In [365…
```
# Método silhouette para identificar el número óptimo de clusters
# ==============================================================================
range_n_clusters = range(2, 11)
valores_medios_silhouette = []

for n_clusters in range_n_clusters:
    modelo = AgglomerativeClustering(n_clusters = n_clusters, affinity = "euclidean", linkage = "ward")

    cluster_labels = modelo.fit_predict(df_pca_agg)
    silhouette_avg = silhouette_score(df_pca_agg, cluster_labels)
    valores_medios_silhouette.append(silhouette_avg)

fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.plot(range_n_clusters, valores_medios_silhouette, marker='o')
ax.set_title("Evolución de media de los índices silhouette")
ax.set_xlabel('Número clusters')
ax.set_ylabel('Media índices silhouette');
```



- Como en los dos casos anteriores el óptimo de custers en función del coeficiente silhouette es de 2, con un valor máximo de 0.91.
- El valor de silhouette decrece hasta 3 clusters, para volver a crecer hasta alcanzar un nuevo máximo en 7 cluster con un valor de silhouette de 0.888.

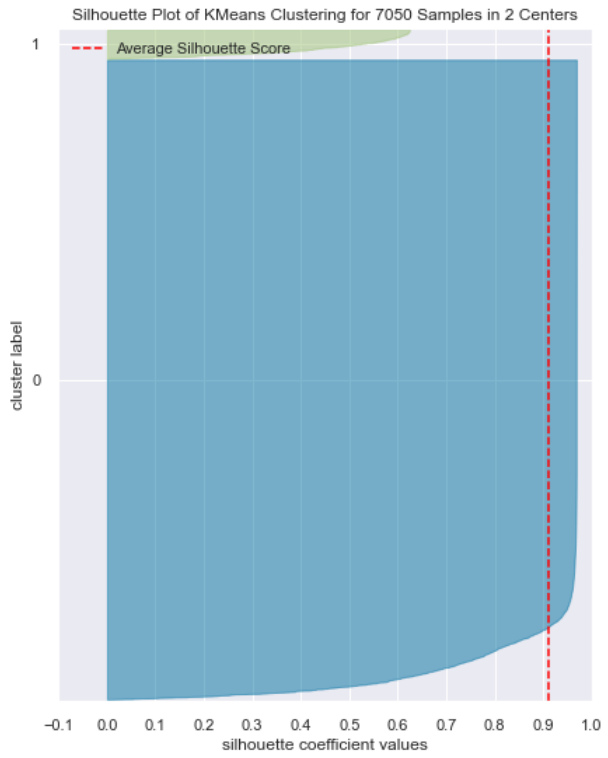
3.4 Gráfico de valores de Silhouette con Yellowbrick

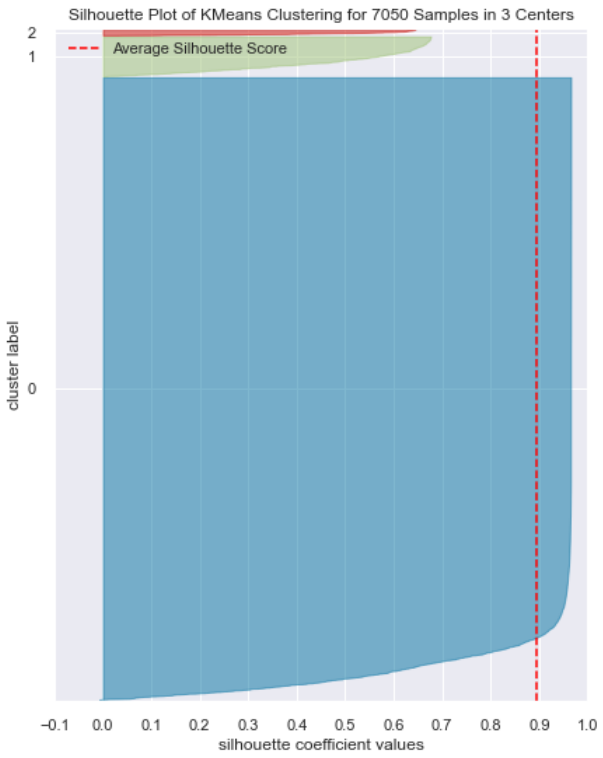
In [184…
```
from yellowbrick.cluster import SilhouetteVisualizer
```

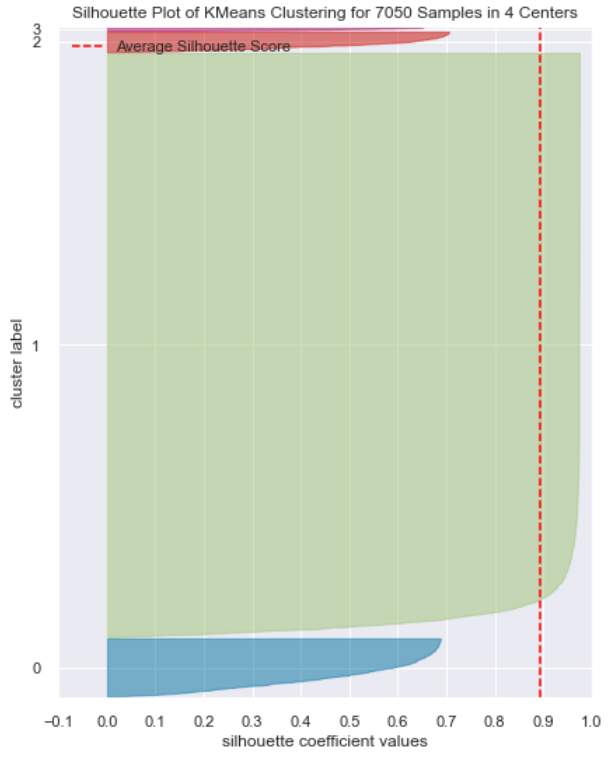
In [185…
```
# silhouette score plots with Yellowbrick
dict_score = dict()
```

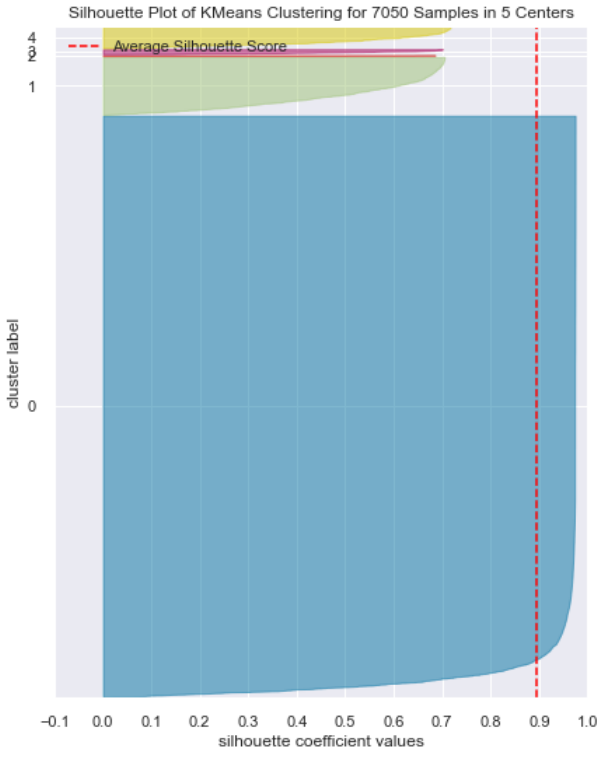
```
fig, ax = plt.subplots(3, 2, figsize=(15,30))

for i in range(2,8):
    km = KMeans(n_clusters=i, random_state=1)

    q, mod = divmod(i, 2)
    vis = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod], is_fitted=False)
    vis.fit(df_pca)
    vis.finalize()
    dict_score[i] = vis.silhouette_score_

print("silhouette scores for k clusters:")
_ = [print(k,":",f'{v:.3f}') for k,v in dict_score.items()]

K_sil_a3 = max(dict_score, key=dict_score.get)          # optimal clusters
sil_opt_a3 = dict_score[K_sil_a3]                       # optimal (maximal) silhouette score
print("maximum silhouette score for", f'{K_sil_a3:.0f} clusters: ', f'{sil_opt_a3:.3f}')
```

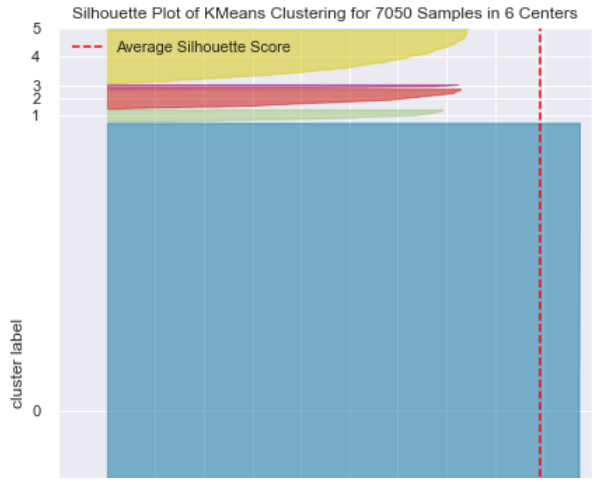
```
silhouette scores for k clusters:
2 : 0.912
3 : 0.894
4 : 0.893
5 : 0.894
6 : 0.894
7 : 0.886
maximum silhouette score for 2 clusters:  0.912
```

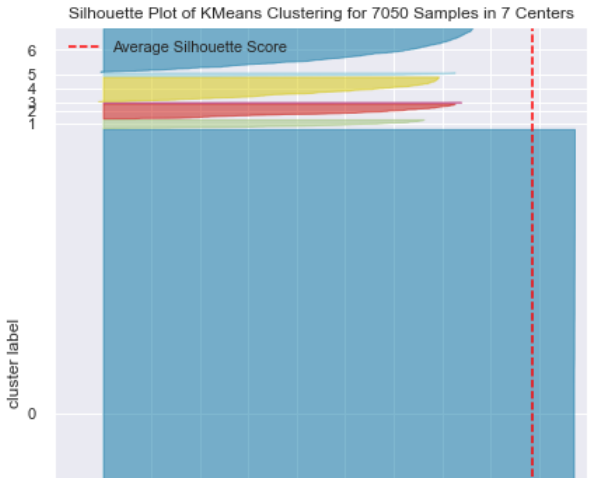

Silhouette Plot of KMeans Clustering for 7050 Samples in 2 Centers
Average Silhouette Score
cluster label
silhouette coefficient values

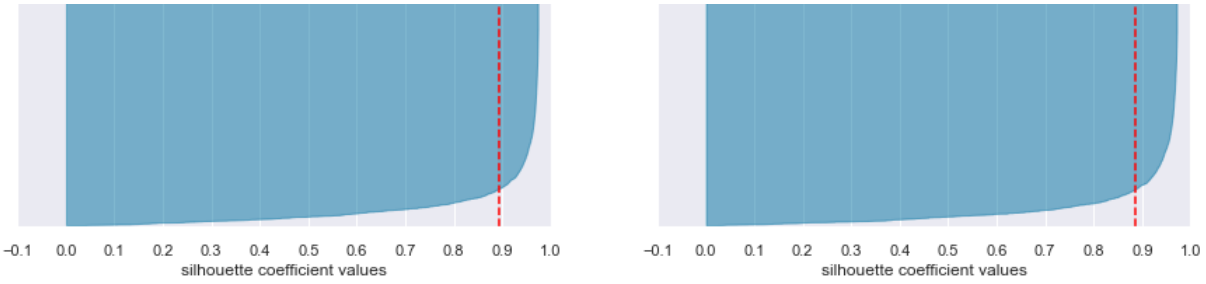

Silhouette Plot of KMeans Clustering for 7050 Samples in 3 Centers
Average Silhouette Score
cluster label
silhouette coefficient values


Silhouette Plot of KMeans Clustering for 7050 Samples in 4 Centers
Average Silhouette Score
cluster label
silhouette coefficient values


Silhouette Plot of KMeans Clustering for 7050 Samples in 5 Centers
Average Silhouette Score
cluster label
silhouette coefficient values


Silhouette Plot of KMeans Clustering for 7050 Samples in 6 Centers
Average Silhouette Score
cluster label


Silhouette Plot of KMeans Clustering for 7050 Samples in 7 Centers
Average Silhouette Score
cluster label



- Los datos obtenidos con la esta libreria son similares a los calculados en apartados anteriores con el modelo KMeans, y el óptimo corresponde a la evaluación con 2 clusters con un valor de 0.912 similar a los datos otenidos en apartados anteriores.
- Existe un factor diferencial cuando calcula k=4 (clusters) ya que agrupa el mayor número de datos en el cluster 1, a diferencia de nuesto análisis que los concentraba en el cluster 0, y del resto de gráficos del propio Yellowbrick que para todos los k!=4, los acumula en el cluster 0.