# VinkosTestDS

May 29, 2023

[1]:
```
import pandas as pd
import numpy as np
from plotnine import *
from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm


from sklearn.metrics import roc_curve, classification_report
import matplotlib.pyplot as plt
from tqdm import tqdm



import shap
```

## 1 EDA

El análisis exploratorio permitió observar que la mayoría de las variables del conjunto de datos, salvo por la columna correspondiente a la fecha, son útiles para la construcción de modelos de clasificación y de clustering dado que presentan un tipo de dado que fácil de manejar, ya que no requiere de ningún encoding adicional y en general el dataset solo requiere de un preprocesamiento sencillo: el cual se limita a cuidar que un registro no se repita, la fecha no sea nula y los demás campos sean del tipo adecuado (flotante).

- Se identificó que de los 17,287 registros originales de los archivos se tienen duplicados, después de eliminarlos se retuvieron 9,143 para continuar con el trabajo.

- La columna `c1` (renombrada `Class`) presenta el 21% de registros con el valor `1`. Se realiza una prueba paramétrica de media para probar este valor (la cual no se rechaza) así como su respectivo intervalo de confianza a 95%. Este valor es importante ya que será el valor baseline que servirá como referencia para medir la varianza de los algoritmos de clasificación que se prueben, es decir que servirá para medir que tanto difiere el error en los conjuntos de train y en el test.

[2]:
```
data = pd.read_csv("info_01.csv")
df_temp = pd.read_csv("info_02.csv")

data  = pd.merge(data, df_temp, how="left", on =['id', 'id2']).drop_duplicates()
data =data.rename(columns ={ 'v1': 'Lux', 'v2': 'Temp', 'v3': 'Fecha', 'v4':␣
 ↪'CO2', 'v5':'Humedad' , 'v6':'CocienteHumedad',  'c1': 'Class'})
data = data.drop(columns='id2') # es igual a la id, por el tamanio del df no␣
 ↪hace falta llamar al gc
data
```

[2]:

| | id | CO2 | Humedad | CocienteHumedad | Lux | Class | Temp | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 721.2 | 27.3 | 0.004793 | 426.0 | 1 | 23.2 | |
| 4 | 2 | 714.0 | 27.3 | 0.004783 | 429.5 | 1 | 23.1 | |
| 8 | 3 | 713.5 | 27.2 | 0.004779 | 426.0 | 1 | 23.1 | |
| 12 | 4 | 708.2 | 27.2 | 0.004772 | 426.0 | 1 | 23.1 | |
| 16 | 5 | 704.5 | 27.2 | 0.004757 | 426.0 | 1 | 23.1 | |
| … | … | … | … | … | … | … | … | |
| 34552 | 8139 | 787.2 | 36.1 | 0.005579 | 433.0 | 1 | 21.1 | |
| 34556 | 8140 | 789.5 | 36.0 | 0.005563 | 433.0 | 1 | 21.1 | |
| 34560 | 8141 | 798.5 | 36.1 | 0.005596 | 433.0 | 1 | 21.1 | |
| 34564 | 8142 | 820.3 | 36.3 | 0.005621 | 433.0 | 1 | 21.1 | |
| 34568 | 8143 | 821.0 | 36.2 | 0.005612 | 447.0 | 1 | 21.1 | |

| | Fecha |
|---|---|
| 0 | 2015-02-04 17:51:00 |
| 4 | 2015-02-04 17:51:59 |
| 8 | 2015-02-04 17:53:00 |
| 12 | 2015-02-04 17:54:00 |
| 16 | 2015-02-04 17:55:00 |
| … | … |
| 34552 | 2015-02-10 09:29:00 |
| 34556 | 2015-02-10 09:29:59 |
| 34560 | 2015-02-10 09:30:59 |
| 34564 | 2015-02-10 09:32:00 |
| 34568 | 2015-02-10 09:33:00 |

[9143 rows x 8 columns]

[3]:
```
res = stats.ttest_1samp(data.Class, popmean=0.21)
print(res)
```

```
res.confidence_interval()
```

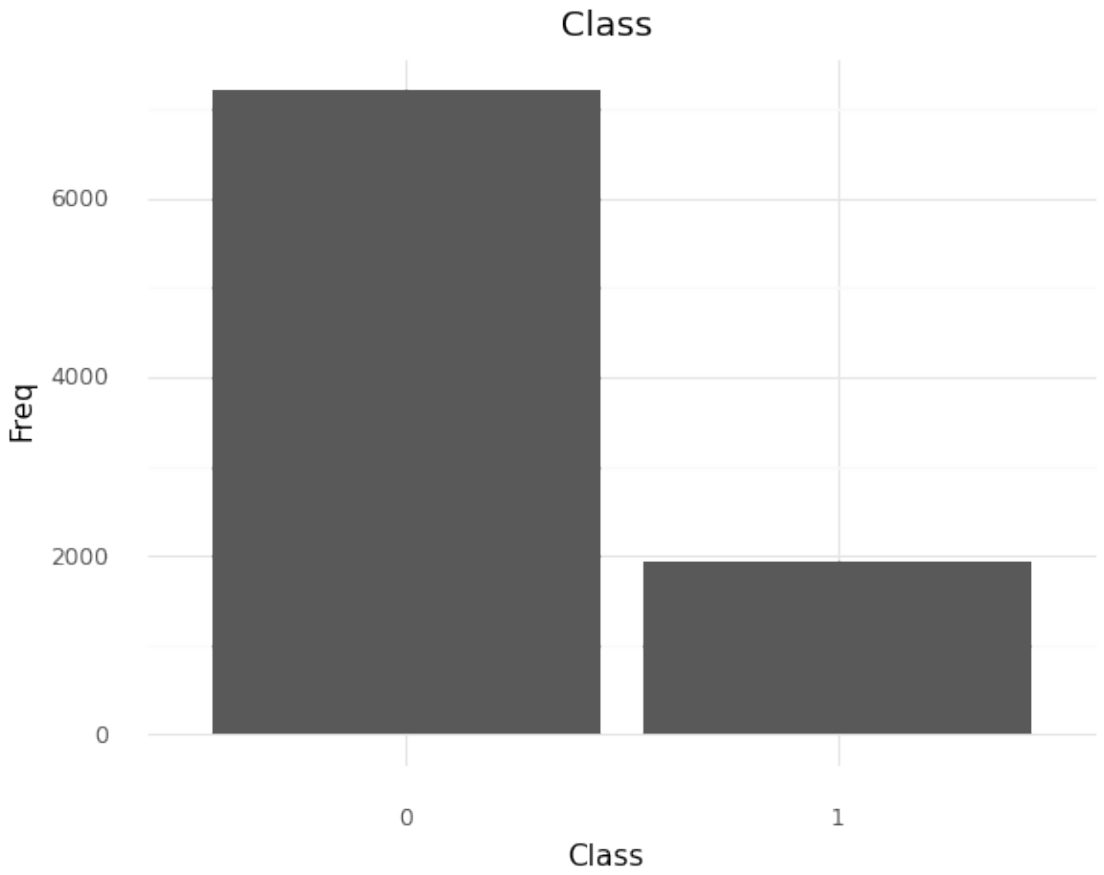
```
TtestResult(statistic=0.25549308257337144, pvalue=0.7983480334511068, df=9142)
```

[3]:
```
ConfidenceInterval(low=0.2027241622605664, high=0.2194567411628176)
```

[4]:
```
data = data[ ['id', 'Lux', 'Temp', 'Fecha', 'CO2', 'Humedad' ,␣
 ↪'CocienteHumedad', 'Class'] ]
```

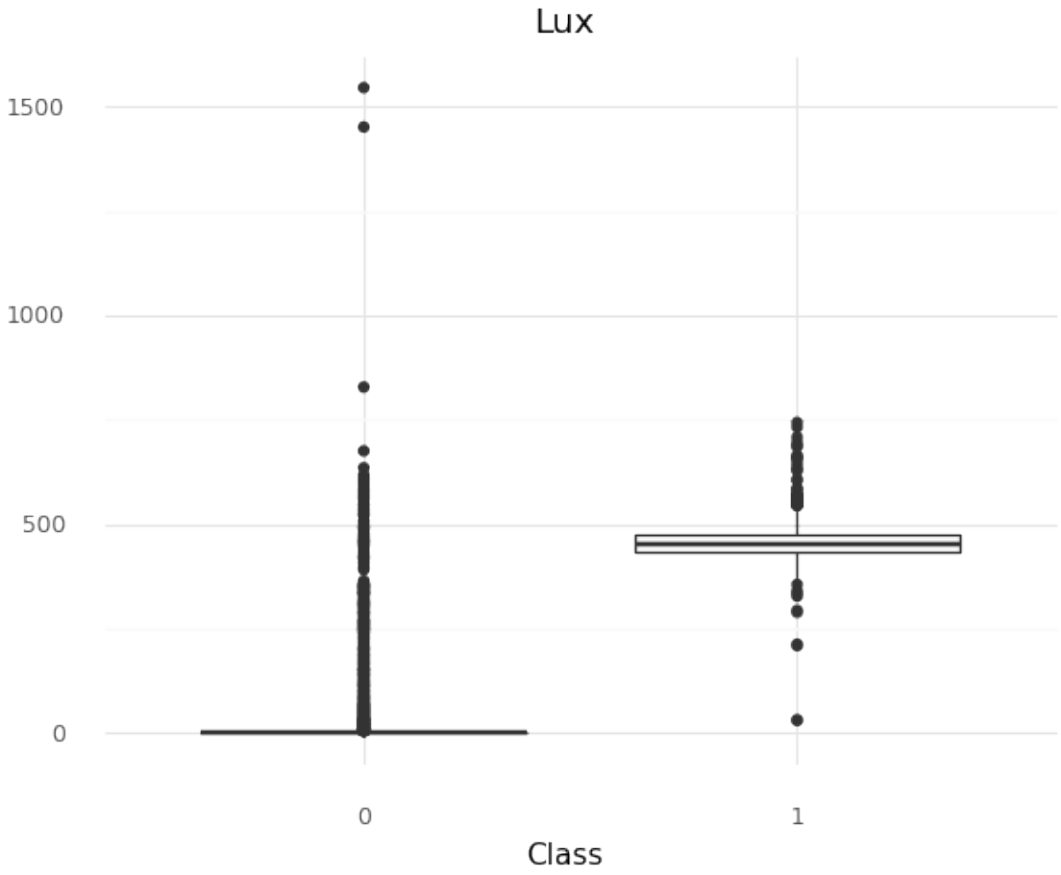
Se realizó un análisis exploratorio transversal para las variables `v1`(`Lux`), `v2` (`Temp`), `v4`(`CO2`) , `v5`(`Humedad`) y `v6`(`CocienteHumedad`), el cual consistió en observar la distribución de las variables a través de los dos valores de las clases `0`y `1`. Para las cinco variables es notorio que tanto para las medianas y sus rangos intercuartiles las distribuciones son diferentes, una prueba de medias para dos grupos de muestras para cada variable donde cada grupo proviene de un valor de clase diferentes permite corroborar esta afirmación.

[5]:
```
ggplot(data, aes(x='factor(Class)')) + geom_bar() + xlab("Class") +␣
 ↪ylab("Freq")+ ggtitle("Class")+ theme_minimal()
```




[5]:
```
<ggplot: (382830343)>
```

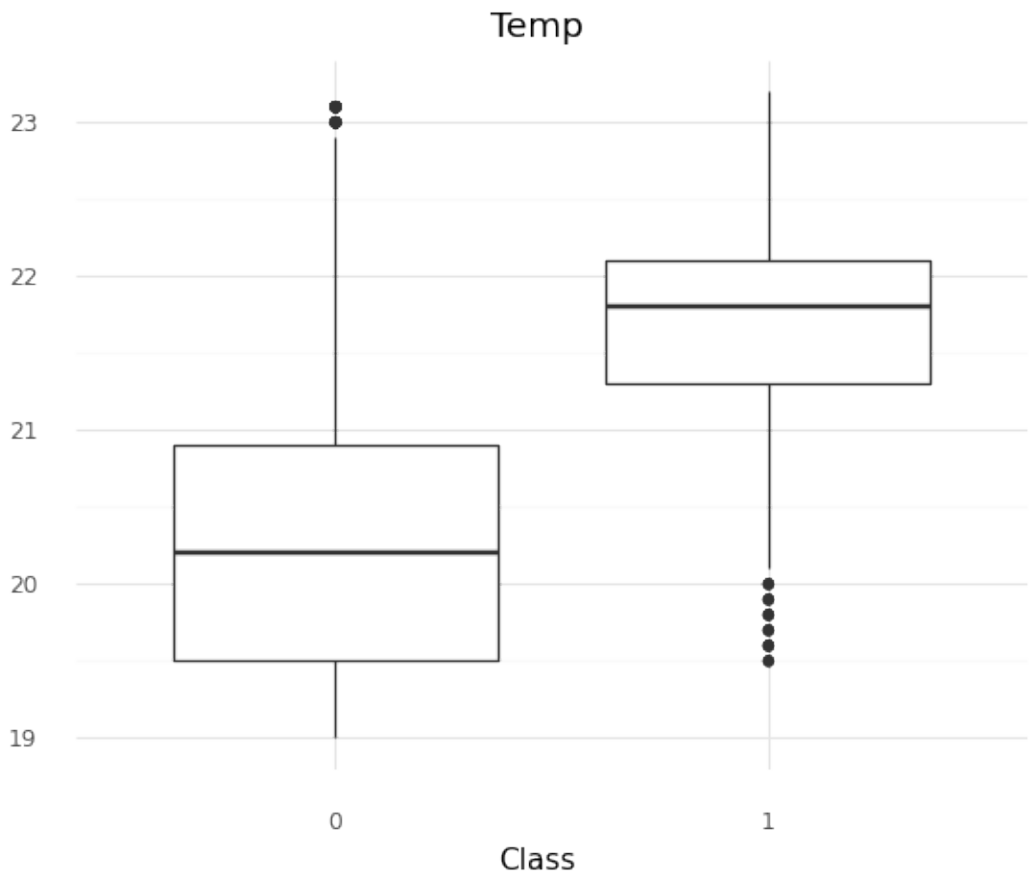
[6]:
```
ggplot ( data, aes(x='factor(Class)', y = 'Lux')) +xlab("Class") + ylab("")+␣
↪geom_boxplot()  + ggtitle("Lux") +theme_minimal()
```




[6]:
```
<ggplot: (382830589)>
```

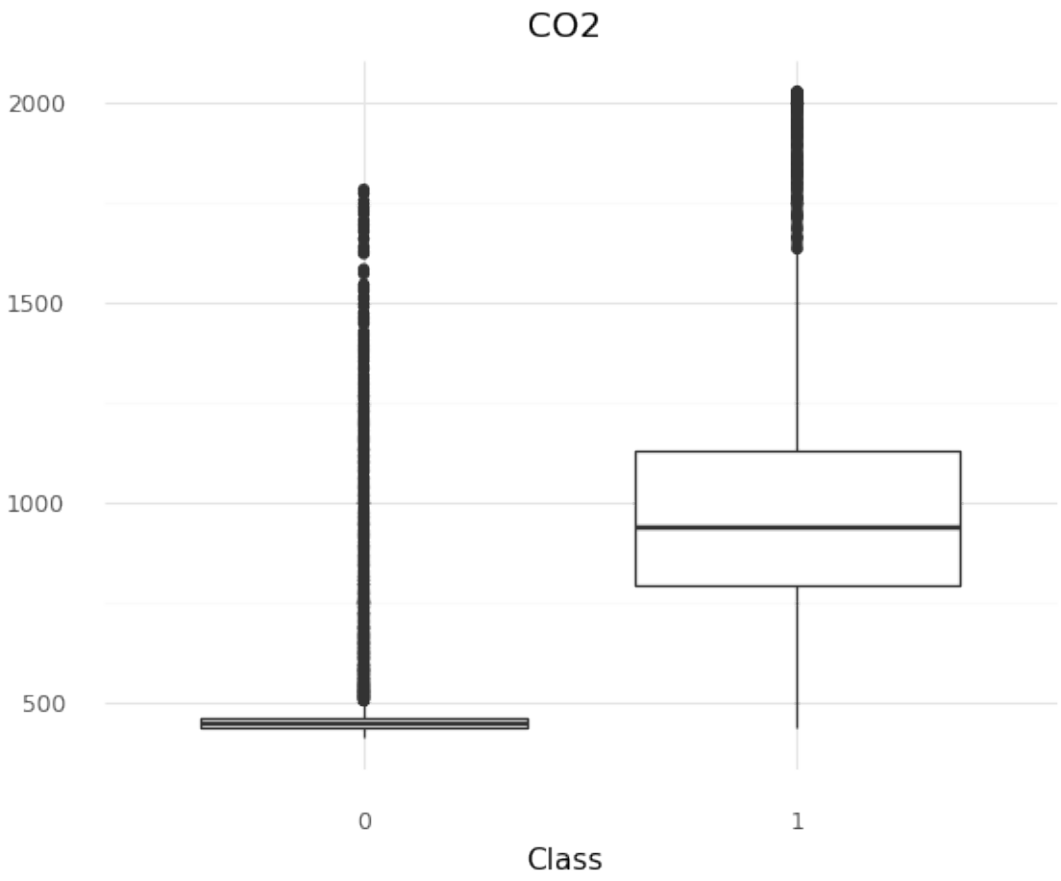
Aunque la variable Lux presenta algunas observaciones con valores outliers, estos se encuentran dentro de rango para ser considerados valores posibles pues 1,000 Lux son valores para un día nublado (aunque la variable Lux presenta algunas observaciones con valores outliers, estos se encuentran dentro de rango para ser considerados valores posibles pues 1000 Lux son valores para un día nublado

[7]:
```
ggplot ( data, aes(x='factor(Class)', y = 'Temp')) + xlab("Class") +␣
↪ylab("")+geom_boxplot() + ggtitle("Temp")+ theme_minimal()
```



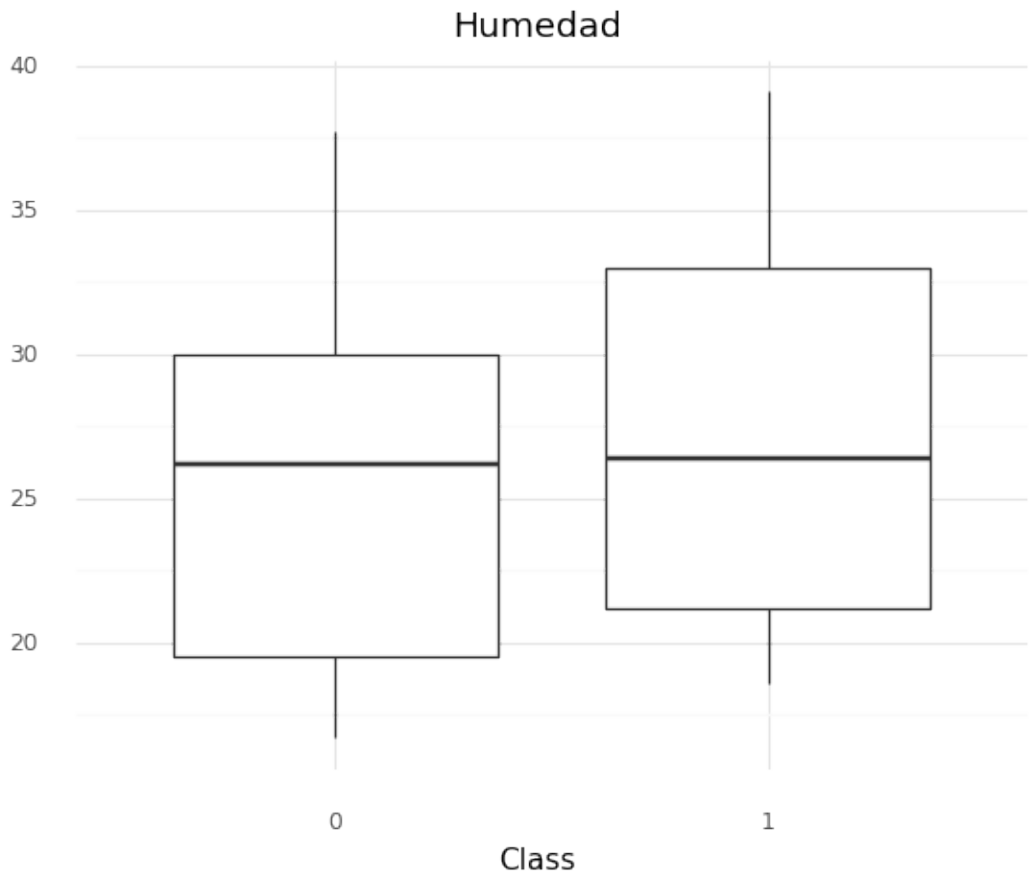

[7]: <ggplot: (382998962)>

[8]:
```
ggplot ( data, aes(x='factor(Class)', y = 'CO2')) + geom_boxplot() +␣
↪xlab("Class") + ylab("")+ ggtitle("CO2")+ theme_minimal()
```



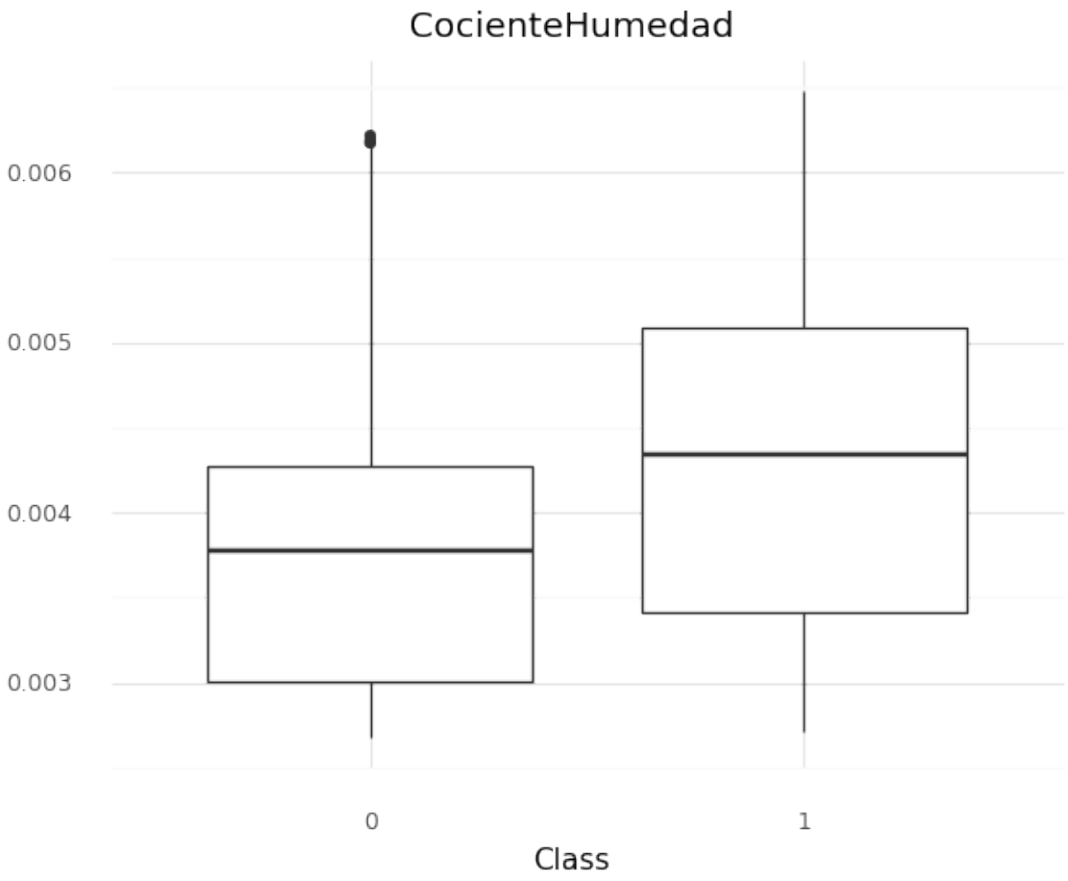

[8]: <ggplot: (383036989)>

[9]:
```
ggplot ( data, aes(x='factor(Class)', y = 'Humedad')) +  xlab("Class") +␣
↪ylab("")+geom_boxplot() +ggtitle("Humedad")+ theme_minimal()
```




[9]: <ggplot: (383075874)>

[10]:
```
ggplot ( data, aes(x='factor(Class)', y = 'CocienteHumedad')) + xlab("Class") +␣
↪ylab("")+ geom_boxplot()  +ggtitle("CocienteHumedad")+ theme_minimal()
```




[10]: <ggplot: (382832755)>

[11]:
```
rng = np.random.default_rng()

for c in ['Lux', 'Temp', 'CO2', 'Humedad', 'CocienteHumedad']:
    print(c)
    grupo_0 = data.loc[data.Class==0., c ].copy()
    grupo_1 = data.loc[data.Class==1., c ].copy()
    print(stats.ttest_ind( grupo_0 , grupo_1, permutations=10000,␣
 ↪random_state=rng) ) # prueba de medias, usamos el test exacto pues la␣
 ↪implementacion aproximada subestima el p-valor
```

```
Lux
Ttest_indResult(statistic=-206.7621217826058, pvalue=9.999000099990002e-05)
Temp
Ttest_indResult(statistic=-61.101216097723544, pvalue=9.999000099990002e-05)
CO2
Ttest_indResult(statistic=-96.27084451059736, pvalue=9.999000099990002e-05)
Humedad
```

```
Ttest_indResult(statistic=-12.430296881852382, pvalue=9.999000099990002e-05)
CocienteHumedad
Ttest_indResult(statistic=-29.74157630832111, pvalue=9.999000099990002e-05)
```

[12]:
```
data.Fecha  = pd.to_datetime(data.Fecha)
```

Al inspeccionar los datos notamos que la columna v3 (Fecha) contiene varios registros con valores nulos. Puesto que las columnas tipo datetime permiten extraer información importante como si los registros provienen de un día laboral o no, de un fin de semana, detectar horas pico, etc. Se procedió a analizar los datos de manera longitudinal. Se graficaron las columnas mencionadas en el párrafo anterior vistas como series de tiempo considerando Fechacomo su índice. Al considerar remover de la muestra los valores nulos se alteraría la proporción de la Clase 0/ 1 al valor 20.1% que se encuentra fuera del intervalo de confianza de 95% por lo que se decide no utilizar en lo posterior la columna Fecha.

[13]:
```
data.loc[data.Fecha.isna(), :].describe()
```

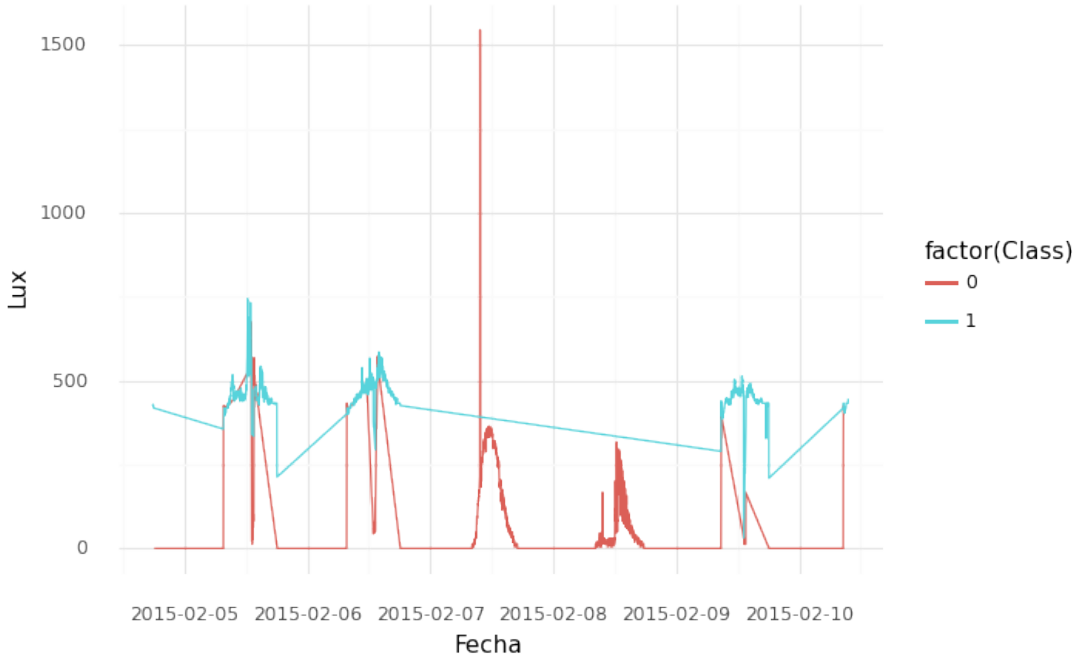
[13]:

| | id | Lux | Temp | CO2 | Humedad | \ |
|---|---|---|---|---|---|---|
| count | 1000.000000 | 1000.000000 | 1000.000000 | 1000.00000 | 1000.000000 | |
| mean | 4135.112000 | 112.987700 | 20.575500 | 597.65820 | 25.718700 | |
| std | 2286.263686 | 190.578773 | 1.030355 | 305.82421 | 5.466458 | |
| min | 7.000000 | 0.000000 | 19.000000 | 415.80000 | 16.800000 | |
| 25% | 2120.750000 | 0.000000 | 19.675000 | 438.00000 | 20.400000 | |
| 50% | 4275.000000 | 0.000000 | 20.400000 | 452.00000 | 26.300000 | |
| 75% | 6045.500000 | 144.625000 | 21.300000 | 628.07500 | 30.500000 | |
| max | 8122.000000 | 604.800000 | 23.100000 | 2028.50000 | 39.100000 | |

| | CocienteHumedad | Class |
|---|---|---|
| count | 1000.000000 | 1000.000000 |
| mean | 0.003848 | 0.201000 |
| std | 0.000835 | 0.400949 |
| min | 0.002674 | 0.000000 |
| 25% | 0.003094 | 0.000000 |
| 50% | 0.003792 | 0.000000 |
| 75% | 0.004344 | 0.000000 |
| max | 0.006451 | 1.000000 |

[14]:
```
#ggplot(data, aes(x='Fecha', y='Class', color='factor(Class)')) + geom_line() +␣
 ↪theme_minimal()
ggplot(data, aes(x='Fecha', y='Lux', color='factor(Class)')) + geom_line() +␣
 ↪theme_minimal()
```

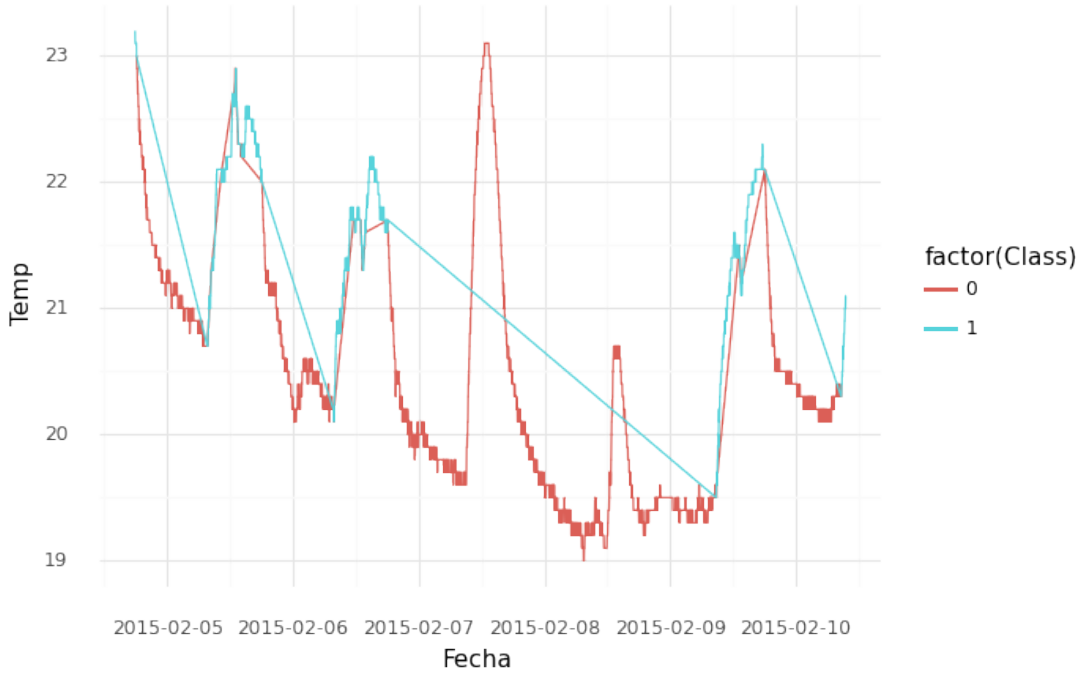
```
geom_path: Removed 201 rows containing missing values.
```




[14]: <ggplot: (383039537)>

[15]:
```
ggplot(data, aes(x='Fecha', y='Temp', color='factor(Class)')) + geom_line() +␣
↪theme_minimal()
```

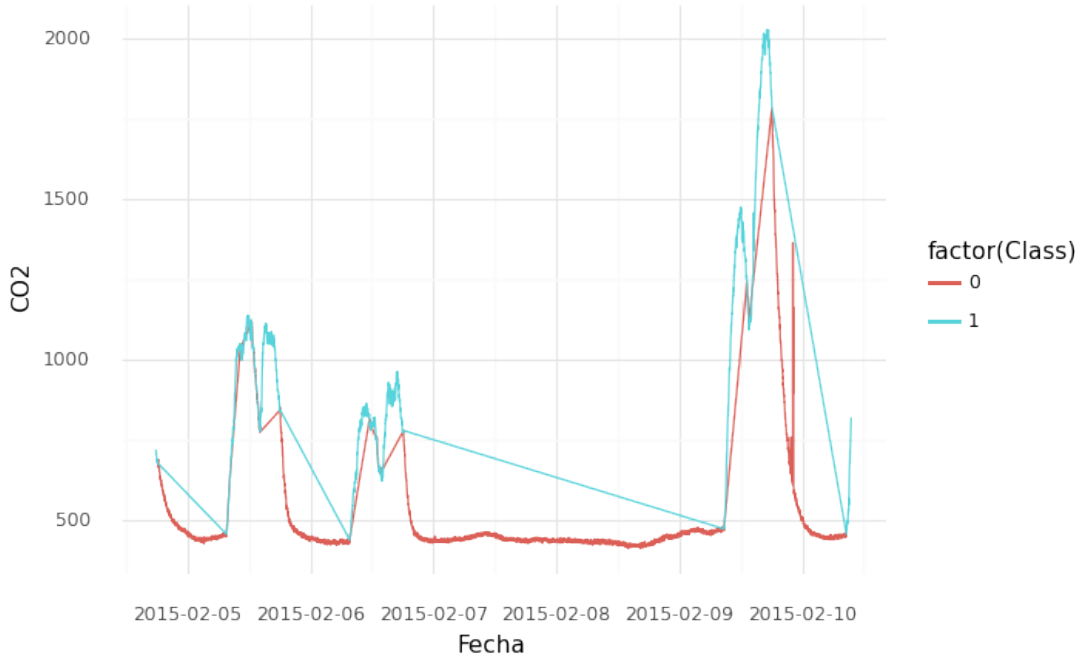
```
geom_path: Removed 201 rows containing missing values.
```




[15]: <ggplot: (383019746)>

[16]:
```
ggplot(data, aes(x='Fecha', y='CO2', color='factor(Class)')) + geom_line() +␣
↪theme_minimal()
```

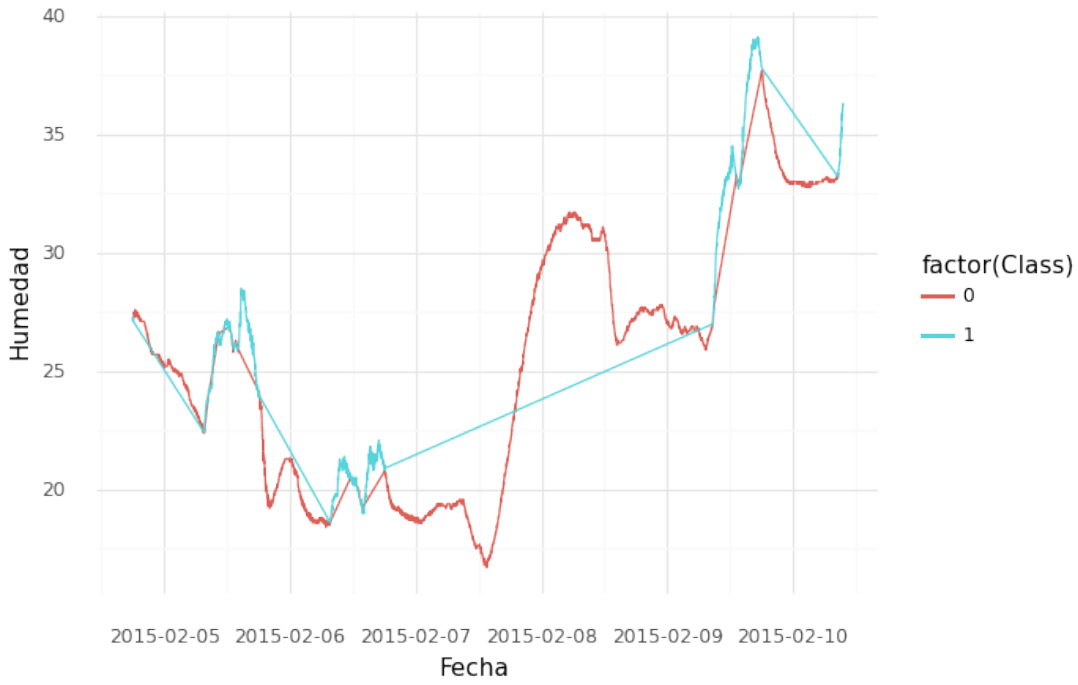
```
geom_path: Removed 201 rows containing missing values.
```




[16]: <ggplot: (383152595)>

[17]:
```
ggplot(data, aes(x='Fecha', y='Humedad', color='factor(Class)')) + geom_line()␣
↪+ theme_minimal()
```

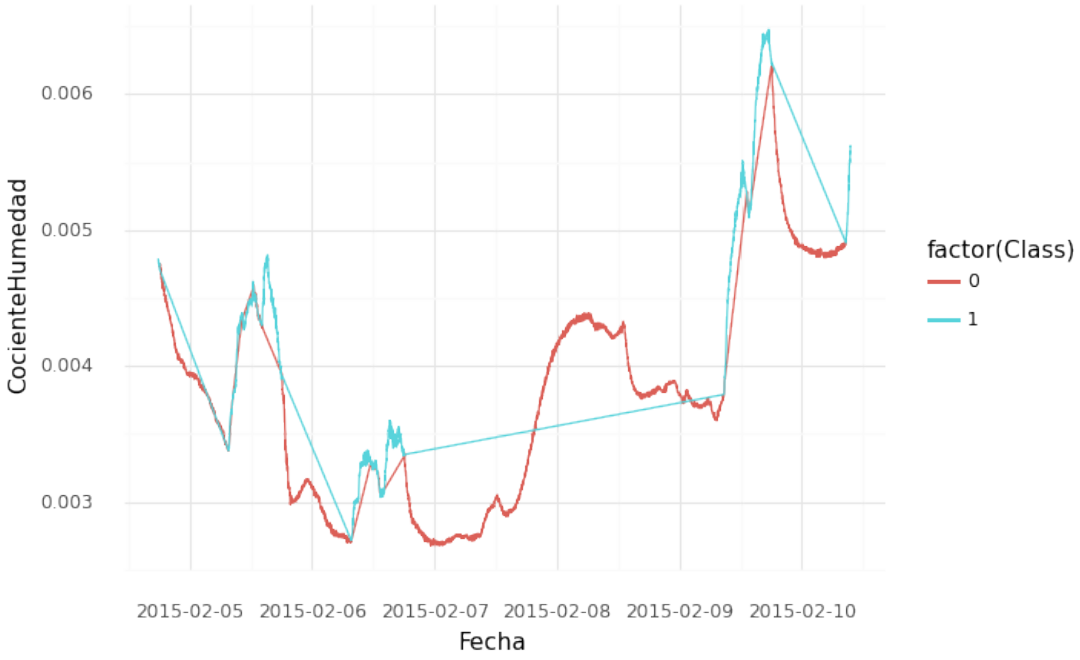
```
geom_path: Removed 201 rows containing missing values.
```




[17]: <ggplot: (383784623)>

[18]:
```
ggplot(data, aes(x='Fecha', y='CocienteHumedad', color='factor(Class)')) +␣
↪geom_line() + theme_minimal()
```

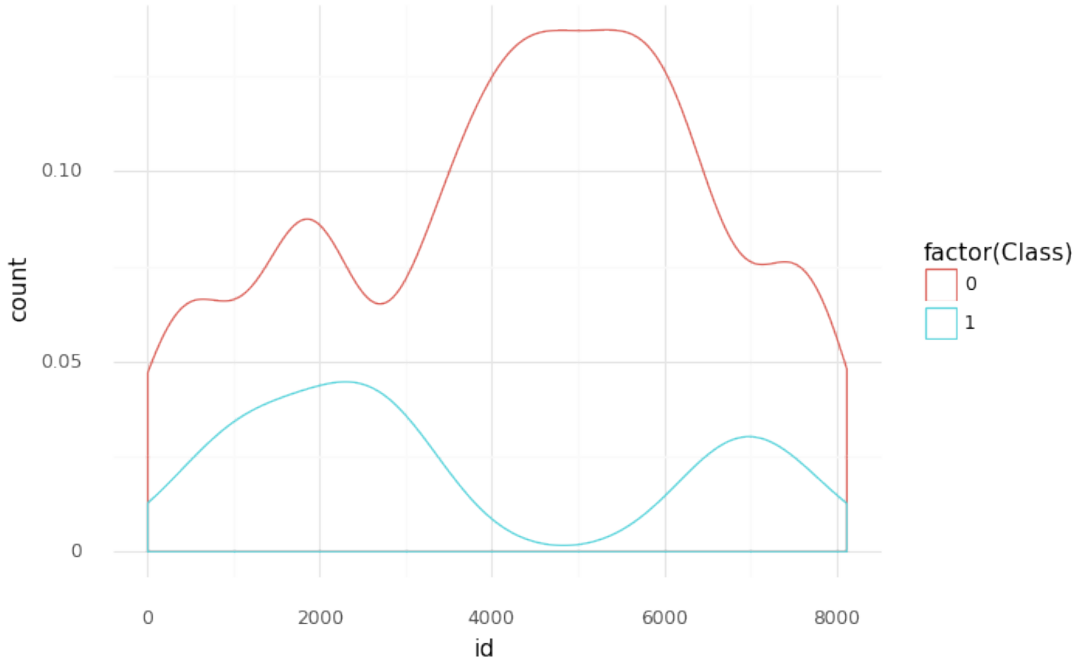
```
geom_path: Removed 201 rows containing missing values.
```




[18]: <ggplot: (383824297)>

Para concluir la sección del análisis exploratorio se grafica un histograma de los ids de los registros donde la `Fecha`es nula donde se puede observar que su ausencia se distribuye de manera no uniforme por lo que en caso de ser requerido podríamos estimar sus valores con los supuestos de datos `Missing and Random` y estimarlos por medio de máxima verosimilitud o por medio de técnicas supervisadas.

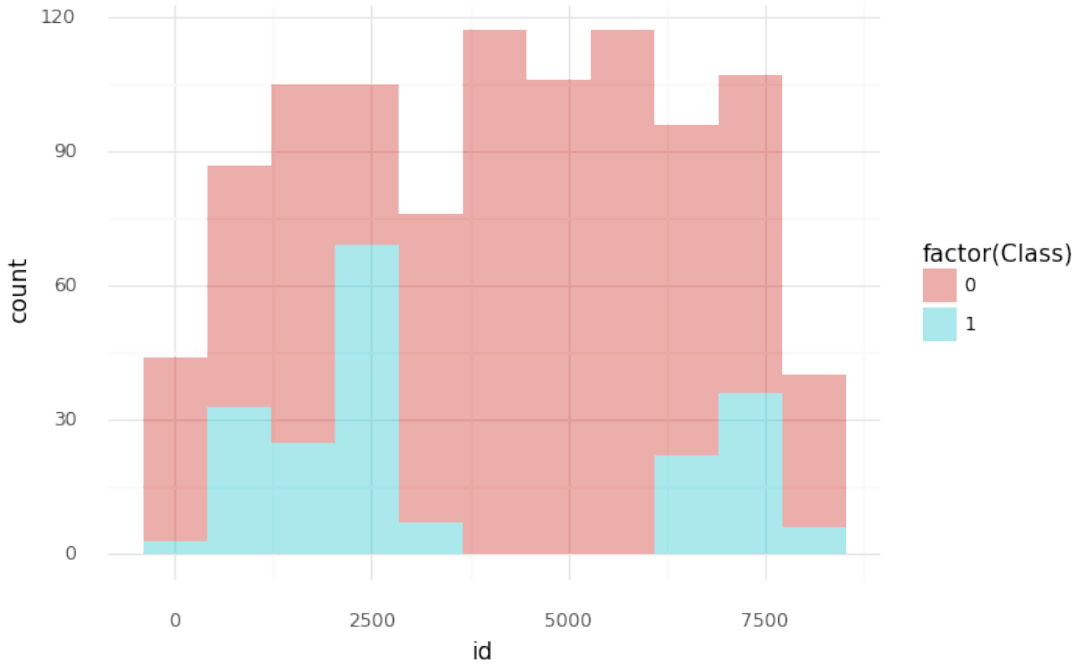
[19]:
```
index = data.Fecha.isna()
indice = np.array(list(data.loc[index, :].index ))
MCR = data.loc[indice, :].copy()
ggplot(MCR, aes(x="id", color='factor(Class)')) +␣
 ↪geom_density(aes(y=after_stat('count'))) + theme_minimal()
```




[19]: <ggplot: (383900154)>

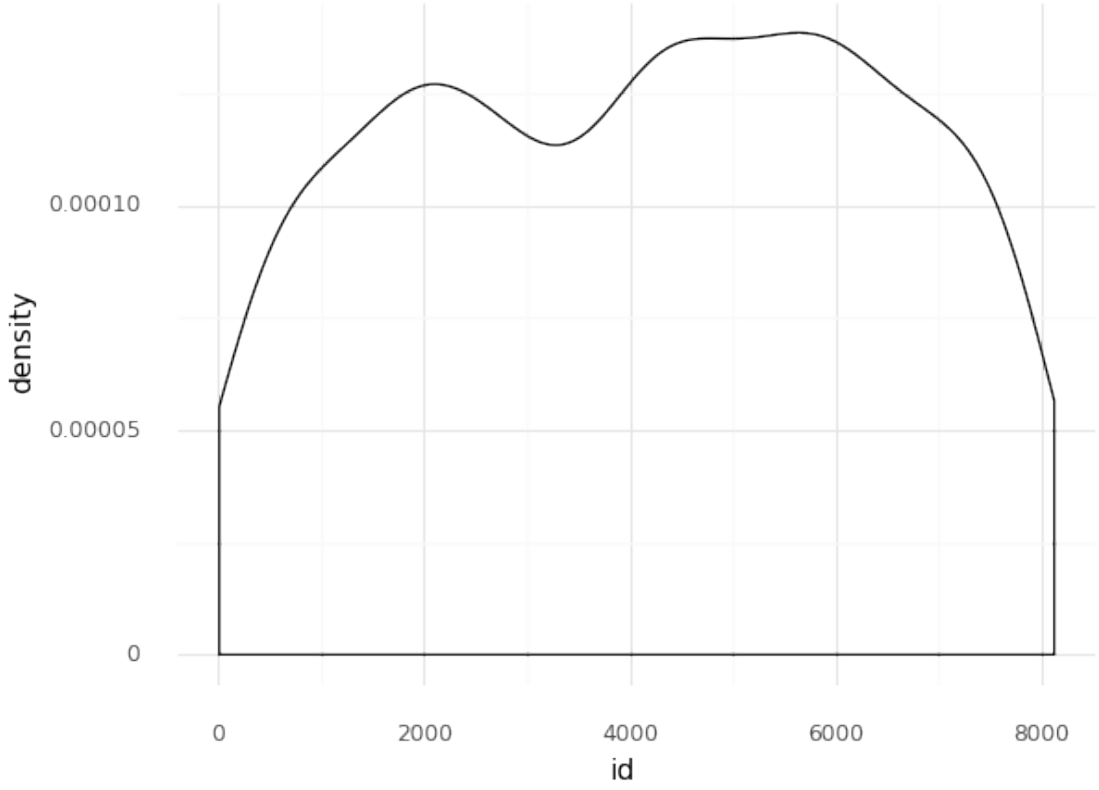
[20]:
```
ggplot(MCR, aes(x="id", fill='factor(Class)')) + geom_histogram(alpha=0.5) +␣
↪theme_minimal()
```

```
'stat_bin()' using 'bins = 11'. Pick better value with 'binwidth'.
```



[20]: <ggplot: (383907069)>

[21]:
```
ggplot(MCR, aes(x="id")) + geom_density(y=after_stat('count*density')) +␣
↪theme_minimal()
```




[21]: <ggplot: (383896549)>

# 2 Algoritmo de clasificación

## 2.1 Preprocesamiento

Como parte del preprocesamiento del conjunto de datos para utilizarlo en los algoritmos solo estandarizamos restando a cada columna su media y dividiendo entre su desviación estándar. Lo anterior por las siguientes razones:

1. Como las variables que tenemos son magnitudes físicas esto permite controlar la varianza de las mediciones. Ya que desconocemos las condiciones en que se colectaron los datos además en que se miden en diferentes unidades de medida.
2. Mayor precisión numérica en los algoritmos al trabajar con unidades cercanas al cero.

3. Al trabajar con estandarización media / desviación estándar no nos preocupamos con observar valores inusuales como en el caso de la estandarización [0,1]

[22]:
```
scaler = StandardScaler()
data_T = scaler.fit(data[['Lux', 'Temp', 'CO2', 'Humedad', 'CocienteHumedad'␣
 ↪]]).transform(data[['Lux', 'Temp', 'CO2', 'Humedad', 'CocienteHumedad' ]])
```

[23]:
```
p = .8
seed= 0
rng = np.random.default_rng(seed)
train_x, test_x, train_y , test_y = train_test_split( data_T, data[['Class']],␣
 ↪train_size=p, random_state=seed)
```

En cuanto a algoritmos de clasificación emplearemos tres: regresión logística, adaboost y knn.

La regresión logística porque es un algoritmo fácil de interpretar y de explicar, además permite regularización ridge y lasso y existen implementación ya probadas que no hacen uso exhaustivo de recursos en memoria y son eficientes en tiempo.

Al igual que que la regresión logística, Adaboost es un algoritmo fácil de interpretar y existen implementaciones confiables aunque el número de parámetros a tunear es mayor que en el caso anterior.

Finalmente Knn es otro algoritmo fácilmente interpretable con pocos parámetros y con implementaciones confiables aunque en este caso los tiempos de ejecución son mayores.

Si bien nuestra primera opción de algoritmo puede verse como opción general ya que es una opción analitica y considera a toda la muestra en un solo paso, la segunda es un método de boosting y la tercera es un método "model-free", consideramos que las familias de algoritmos están representadas en general.

[24]:
```
classifiers = [   LogisticRegression(),
    AdaBoostClassifier(),
    KNeighborsClassifier()]

for clf in classifiers:
    # train
    print(clf)
    clf.fit(train_x, train_y)
    #y_train_out = clf.predict(train_x)
    #print(classification_report(train_y, y_train_out))

    # testing
    y_val_out = clf.predict(test_x)
    print(classification_report(test_y, y_val_out))

clf = LogisticRegression().fit(train_x, train_y)
explainer = shap.Explainer(clf, train_x, feature_names=['Lux', 'Temp', 'CO2',␣
 ↪'Humedad', 'CocienteHumedad' ] )
shap_values = explainer(test_x)
```

```
LogisticRegression()
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1442
           1       0.96      0.98      0.97       387

    accuracy                           0.99      1829
   macro avg       0.98      0.99      0.98      1829
weighted avg       0.99      0.99      0.99      1829

AdaBoostClassifier()

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1442
           1       0.98      0.99      0.98       387

    accuracy                           0.99      1829
   macro avg       0.99      0.99      0.99      1829
weighted avg       0.99      0.99      0.99      1829

KNeighborsClassifier()
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1442
           1       0.98      0.99      0.99       387

    accuracy                           0.99      1829
   macro avg       0.99      0.99      0.99      1829
weighted avg       0.99      0.99      0.99      1829


A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples,), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
```

[25]:
```
m_confu = test_y.copy()
m_confu["y_hat"] = clf.predict(test_x)
m_confu.groupby(["Class", "y_hat"])["Class"].count()
```

[25]:
```
Class  y_hat
0      0        1425
       1          17
```

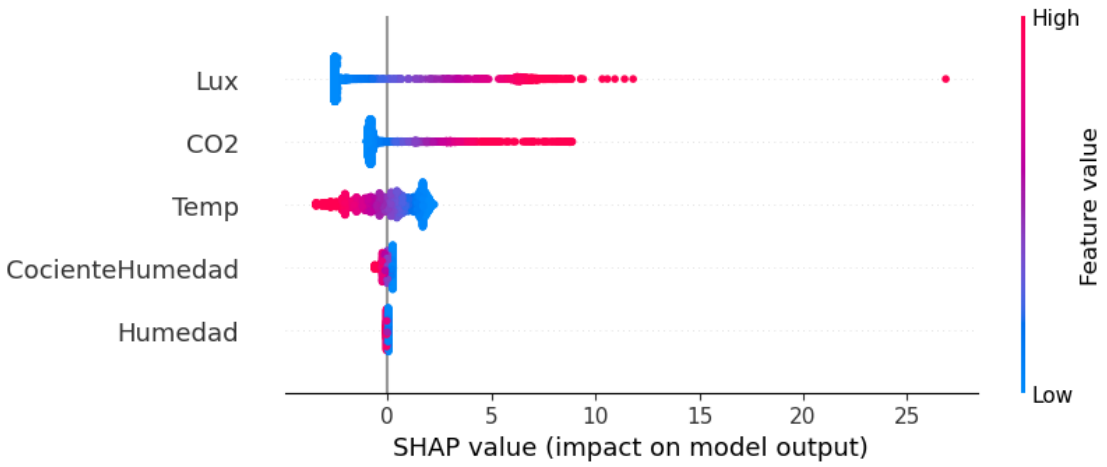
```
1      0           7
       1         380
Name: Class, dtype: int64
```

[26]:
```
tp = 1425#3549#1425
fp = 7#51#7
fn = 17#15 #17
presicion = (tp)/ (tp +fp)
recall = (tp)/(tp + fn)
f1 = 2/ ( (presicion)**(-1) + (recall)**(-1) )

f1
```

[26]:
```
0.9916492693110648
```

[27]:
```
shap.plots.beeswarm(shap_values)
```




Para evaluar los modelos emplearemos la métrica F1 que es una "media ponderada" de la precisión y del recall de los modelos.

En vista de que los modelos presentan desempeño bastante similares sobre los conjuntos train y test, se opta por emplear el modelo de regresión logística en vista de que es más sencillo de interpretar y de buscar hiperparametros además de que su implementación requiere de menor tiempo de ejecución.

Para la regresión logística tenemos un F1 score de 99%

Como notas: Se reportan los resultados para un conjunto de train de 80% de la muestra original sin embargo se tienen resultados similares para un train desde 10% de la muestra original Por el punto anterior se realizó un análisis sobre el aporte de las variables en el modelo seleccionado, regresión logística, cuyos resultados se pueden ver en la gráfica de la celda 25.

El aporte de las variables en orden ascendente es `Lux`, `CO2`, `Temp`, `CocienteHumedad` y `Humedad`. Las columnas `Lux` y `CO2` presentan en conjunto una importancia sobre el modelo mayor que el resto de

las variables, además permiten visualizar que las observaciones con valores altos en estas variables tienen un poder discriminatorio importante por lo que no remover outliers parece apuntar a ser una buena decisión y mantendremos esta idea para los algoritmos de clusterización.

## 3 Algoritmo de clusterización

Como algoritmo de clusterización se comenzó con K-means pues: - Es un algoritmo fácil de interpretar - Sólo tiene un hiperparámetro ($k$) - Su implementación es eficiente ( pues estamos trabajando en dimensiones bajas -número de features es cuatro-)

Al recordar los resultados de la gráfica de Shap sobre la importancia de valores individuales de algunas features se consideró utilizar tambien Kmedoides pues además de las ventajas de Kmeans este presenta poca sensibilidad a valores atípicos.

El primer paso en la especificación de los dos modelos es la elección del número de clusters $k$. Para ello procuraremos elegir el número de clusters en donde se presente la diferencia más significativa en las siguientes gráficas donde el eje horizontal es el número de cluster y el eje vertical es la diferencia entre la inercia de ese modelo (con $k$ custers ) y la inercia del modelo con $k+1$ clusters.

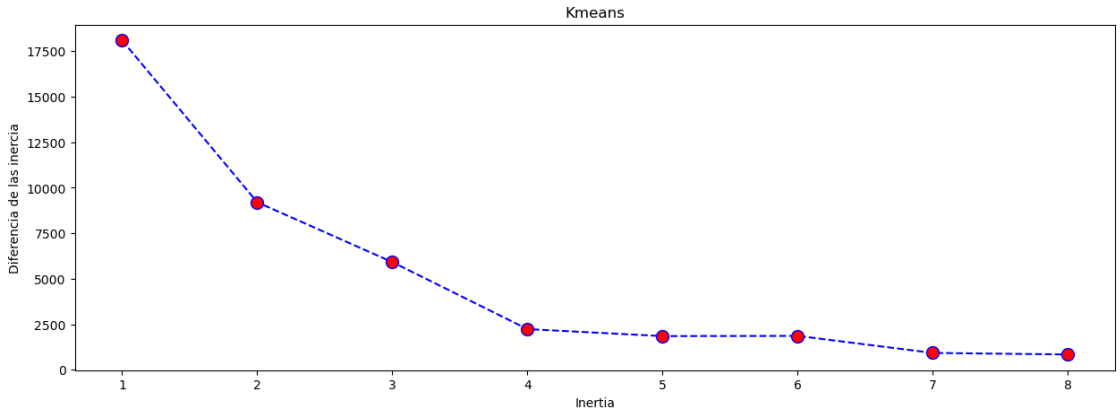
[28]:
```
k1 = []
inertia_s1 = []

for i in tqdm(range(1,10)):
    k1.append(i)
    kmeans1 = KMeans(n_clusters=i, random_state=seed ).fit(data_T)
    inertia_s1.append(kmeans1.inertia_)

# plot
plt.figure(figsize=(15,5))
plt.plot(k1[0:(len(k1)-1)], inertia_s1[0:(len(k1)-1)]- np.roll(inertia_s1,␣
 ↪-1)[0:(len(k1)-1)] ,color='blue', linestyle='dashed', marker='o',␣
 ↪markerfacecolor='red', markersize=10)
plt.title('Kmeans')
plt.xlabel('Inertia')
plt.ylabel('Diferencia de las inercia')
plt.show()
```

```
  0%|                                                    | 0/9 [00:00<?,
?it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
 11%|                                             | 1/9 [00:00<00:00,
9.51it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
 22%|                                          | 2/9 [00:00<00:00,
7.81it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
 33%|                                       | 3/9 [00:00<00:00,
7.56it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
```

```
 44%|                                     | 4/9 [00:00<00:00,
6.20it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
 56%|                                   | 5/9 [00:00<00:00,
6.10it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
 67%|                                 | 6/9 [00:00<00:00,
6.01it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
 78%|                               | 7/9 [00:01<00:00,
5.71it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
 89%|                             | 8/9 [00:01<00:00,
5.71it/s]The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
100%|                           | 9/9 [00:01<00:00,  6.05it/s]
```



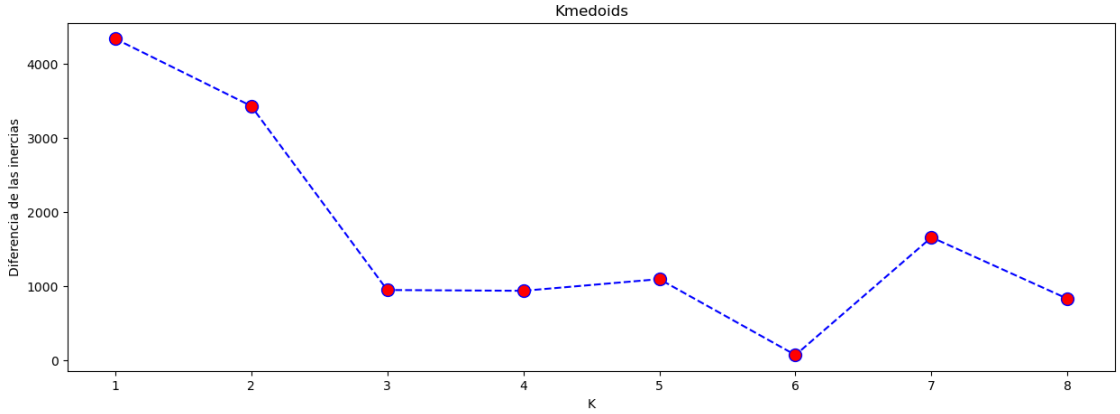

[29]:
```
k2 = []
inertia_s2 = []
seed= 0

for i in tqdm(range(1,10)):
    k2.append(i)
    kmedois1 = KMedoids(n_clusters=i, random_state=seed).fit(data_T)
    inertia_s2.append(kmedois1.inertia_)

# plot
plt.figure(figsize=(15,5))
plt.plot(k2[0:(len(k2)-1)], inertia_s2[0:(len(k2)-1)]- np.roll(inertia_s2,␣
 ↪-1)[0:(len(k2)-1)],color='blue', linestyle='dashed', marker='o',␣
 ↪markerfacecolor='red', markersize=10)
plt.title('Kmedoids')
```

```
plt.xlabel('K')
plt.ylabel('Diferencia de las inercias')
plt.show()
```

```
100%|                          | 9/9 [00:11<00:00,  1.27s/it]
```




[30]:
```
#k = 2
#k_medoids = KMedoids(n_clusters=3, random_state=seed).fit(data_T)
#k_means = KMeans(n_clusters=3, random_state=seed).fit(data_T)
#pca = PCA(n_components=2)
#data_plot = pca.fit(data_T).transform(data_T)
#data_plot = pd.DataFrame(data=data_plot)
#data_plot.columns = [ "PC" + str(_) for _ in range(1, k+1)]
#data_plot["Class"] = data["Class"]
#data_plot['Cluster_means']   = k_medoids.labels_
#data_plot['Cluster_medois']   = k_means.labels_
```

[31]:
```
#ggplot( data_plot, aes( x="PC1", y="PC2")) + geom_point( aes(color=␣
 ↪"factor(Cluster_means)" ), alpha=.1)+ theme_minimal()
```

[32]:
```
#ggplot( data_plot, aes( x="PC1", y="PC2")) + geom_point( aes(color=␣
 ↪"factor(Cluster_medois)"), alpha=.1 )+ theme_minimal()
```

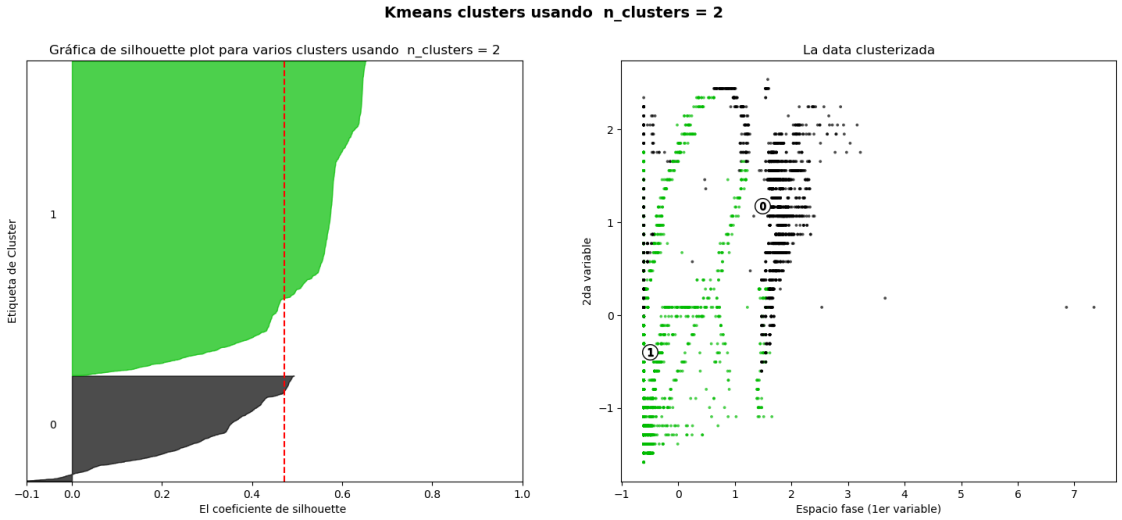
[33]:
```
#ggplot( data_plot, aes( x="PC1", y="PC2", color= "factor(Class)")) +␣
 ↪geom_point(alpha=.1 )+ theme_minimal()
```

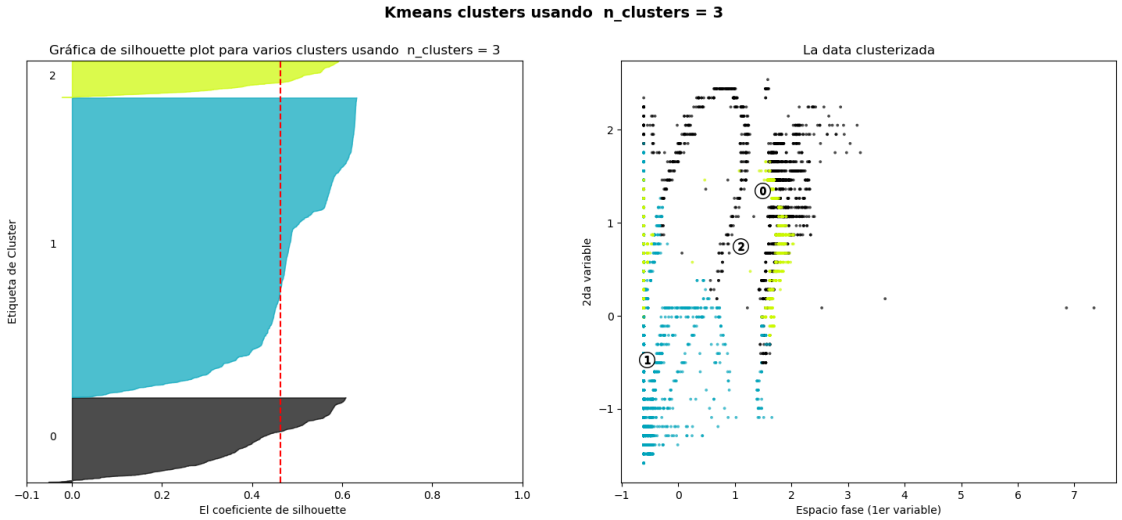
De las gráficas anteriores podemos ver que para Kmeans no tenemos una sugerencia de grupos contundente. Por su parte para K Medoides con 3 grupos la intuición parece sugerir que tendremos un número adecuado de clusters. Para apoyar lo anterior veremos la representación gráfica de la coherencia de los grupos utilizando el score de Silhouette para los clusters de 2 y 3 para ambos métodos.

[34]:
```
X, y = data_T, data["Class"]
range_n_clusters = [2,3]
for n_clusters in range_n_clusters:
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])
    clusterer = KMeans(n_clusters=n_clusters, n_init="auto", random_state=10)
    cluster_labels = clusterer.fit_predict(X)
    silhouette_avg = silhouette_score(X, cluster_labels)
    print(        "Para n_clusters =", n_clusters,    "Silhouette_score :", ␣
 ↪silhouette_avg   )
    sample_silhouette_values = silhouette_samples(X, cluster_labels)
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels␣
 ↪== i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(            np.arange(y_lower, y_upper), 0,    ␣
 ↪ith_cluster_silhouette_values,            facecolor=color, edgecolor=color, ␣
 ↪     alpha=0.7)
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 10  # 10 for the 0 samples
    ax1.set_title("Gráfica de silhouette plot para varios clusters usando ␣
 ↪n_clusters = %d"  % n_clusters)
    ax1.set_xlabel("El coeficiente de silhouette ")
    ax1.set_ylabel("Etiqueta de Cluster")
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(         X[:, 0], X[:, 1], marker=".", s=30, lw=0, alpha=0.7,␣
 ↪c=colors, edgecolor="k"  )
    centers = clusterer.cluster_centers_
    ax2.scatter(        centers[:, 0], centers[:, 1],    marker="o", ␣
 ↪c="white",    alpha=1,  s=200,  edgecolor="k"  )
    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor="k")
    ax2.set_title("La data clusterizada")
    ax2.set_xlabel("Espacio fase (1er variable)")
    ax2.set_ylabel("2da variable")
    plt.suptitle(  "Kmeans clusters usando  n_clusters = %d"  % n_clusters,  ␣
 ↪fontsize=14, fontweight="bold"  )
```

```
plt.show()
```

```
Para n_clusters = 2 Silhouette_score : 0.4712670535211348
Para n_clusters = 3 Silhouette_score : 0.46392661259723456
```

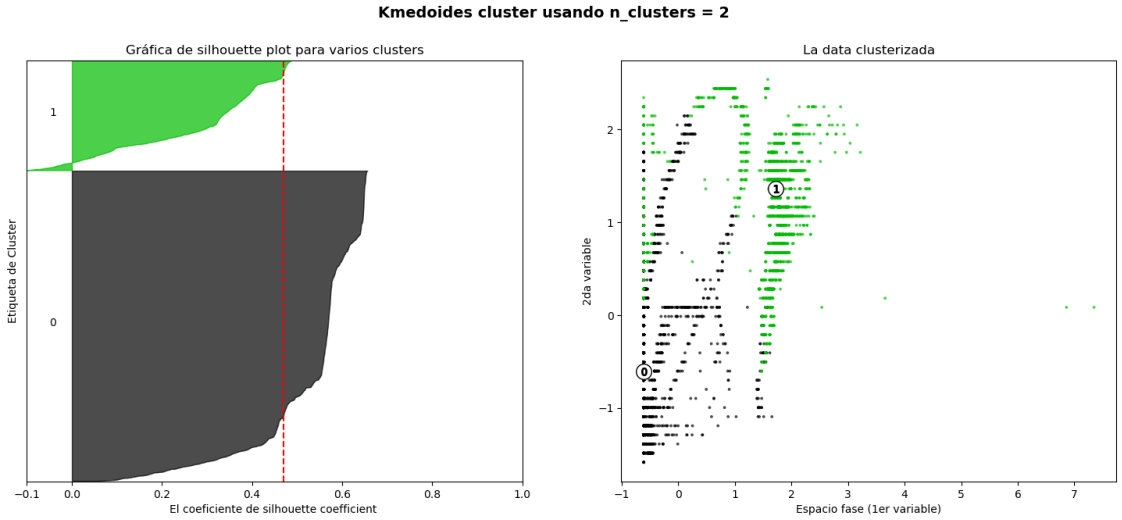


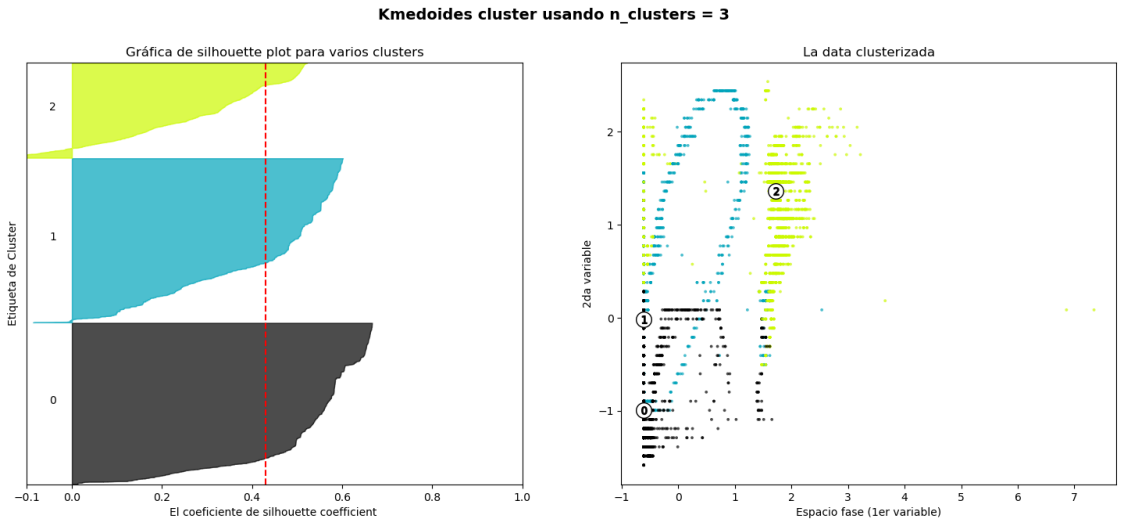




[35]:
```
X, y = data_T, data["Class"]
range_n_clusters = [2,3]
for n_clusters in range_n_clusters:
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])
```

```
    clusterer = KMedoids(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)
    silhouette_avg = silhouette_score(X, cluster_labels)
    print(        "Para n_clusters =", n_clusters,    "Silhouette_score :", ␣
 ↪silhouette_avg   )
    sample_silhouette_values = silhouette_samples(X, cluster_labels)
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels␣
 ↪== i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(            np.arange(y_lower, y_upper), 0,    ␣
 ↪ith_cluster_silhouette_values,            facecolor=color, edgecolor=color, ␣
 ↪     alpha=0.7)
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 10  # 10 for the 0 samples
    ax1.set_title("Gráfica de silhouette plot para varios clusters")
    ax1.set_xlabel("El coeficiente de silhouette coefficient ")
    ax1.set_ylabel("Etiqueta de Cluster")
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(         X[:, 0], X[:, 1], marker=".", s=30, lw=0, alpha=0.7,␣
 ↪c=colors, edgecolor="k"   )
    centers = clusterer.cluster_centers_
    ax2.scatter(        centers[:, 0], centers[:, 1],    marker="o", ␣
 ↪c="white",    alpha=1,  s=200,  edgecolor="k"   )
    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor="k")
    ax2.set_title("La data clusterizada")
    ax2.set_xlabel("Espacio fase (1er variable)")
    ax2.set_ylabel("2da variable")
    plt.suptitle(  "Kmedoides cluster usando n_clusters = %d"  % n_clusters,  ␣
 ↪fontsize=14, fontweight="bold"   )
plt.show()
```

```
Para n_clusters = 2 Silhouette_score : 0.47062967921197607
Para n_clusters = 3 Silhouette_score : 0.42945671233717553
```



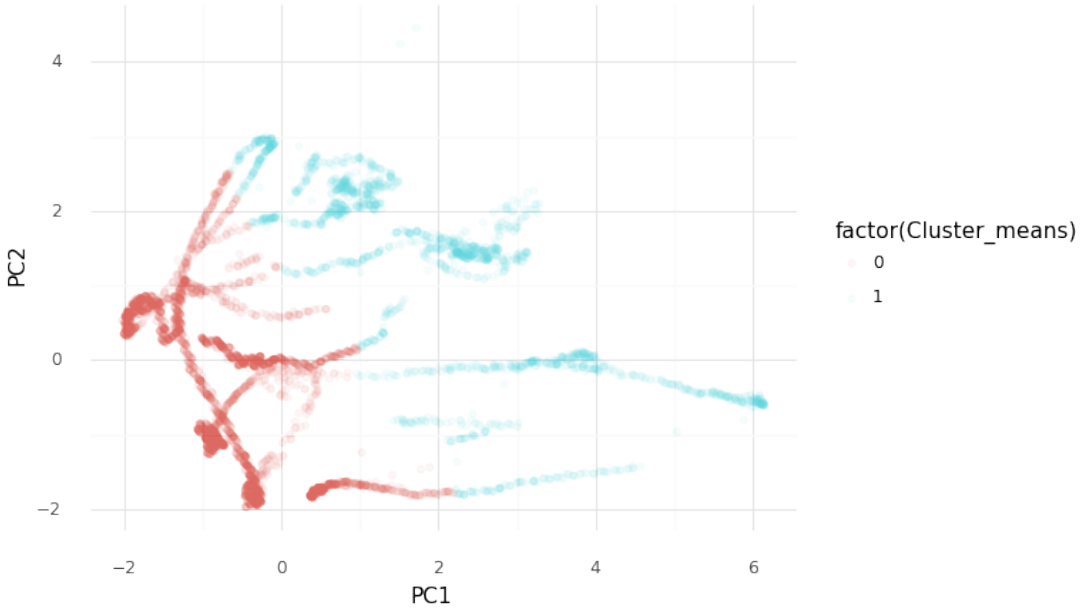

[36]:
```
k = 2
seed=0
k_medoids = KMedoids(n_clusters=k, random_state=seed).fit(data_T)
k_means = KMeans(n_clusters=k, random_state=seed).fit(data_T)
pca = PCA(n_components=2)
data_plot = pca.fit(data_T).transform(data_T)
data_plot = pd.DataFrame(data=data_plot)
data_plot.columns = [ "PC" + str(_) for _ in range(1, 3)]
data_plot["Class"] = data["Class"]
data_plot['Cluster_means']  = k_medoids.labels_
data_plot['Cluster_medois']  = k_means.labels_
```

```
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the
value of `n_init` explicitly to suppress the warning
```

Finalmente como el coeficiente de Silhouette desciende en los dos métodos cuando asciende el número de clusters se decide que 2 clusters es suficiente. Ahora mientras que kmeans tiene un score de Silhouette mayor por milésimas frente a K Medoides para 2 clusters consideramos que esta ganancia es marginal frente a la robustez frente a outliers de K Medoides que permite a este último generalizar sobre nuevas observaciones fuera de nuestro dataset.

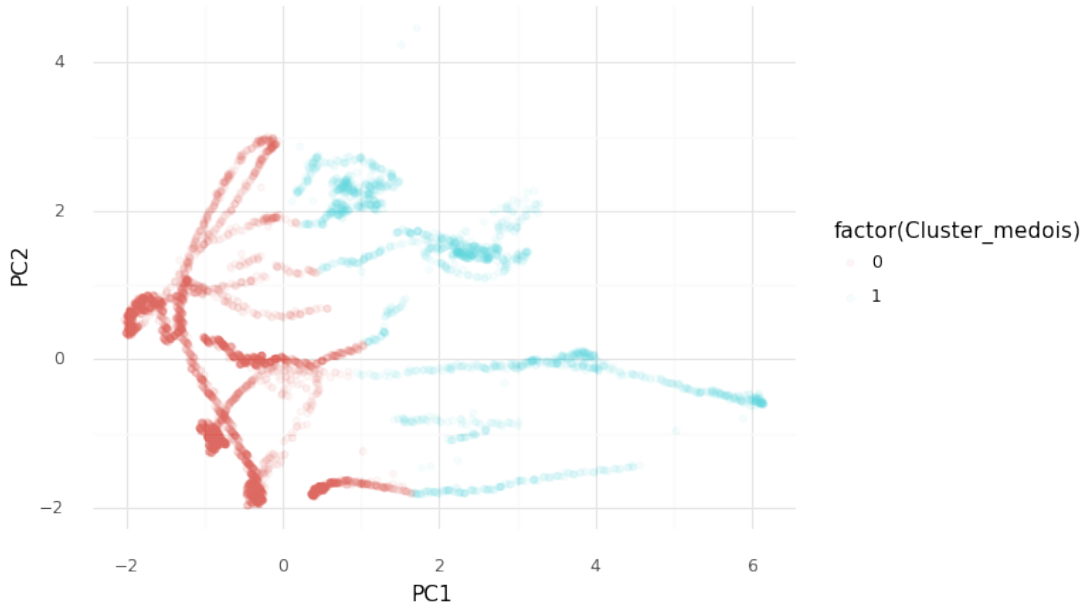
Finalmente optamos por K Medoides con dos clusters como algoritmo final y visualizamos la data con la clasificación final en el espacio de las dos componentes principales (que explican más del 85% de la varianza total).

[37]:
```
ggplot( data_plot, aes( x="PC1", y="PC2")) + geom_point( aes(color=␣
↪"factor(Cluster_means)" ), alpha=.05)+ theme_minimal()
```




[37]:
```
<ggplot: (393210507)>
```

[38]:
```
ggplot( data_plot, aes( x="PC1", y="PC2")) + geom_point( aes(color=␣
↪"factor(Cluster_medois)"), alpha=.05 )+ theme_minimal()
```




[38]: <ggplot: (402571481)>

[ ]: