PRÁCTICA 2: Limpieza y análisis de los datos

January 4, 2022

Alejandro Pérez Manrique

Estudios de Informática, Multimedia y Telecomunicaciones

Índice de contenido

---

1 Descripción del dataset

En la PRA 1 habíamos extraído mediante *Web Scraping* información sobre una página de videojuegos. El problema que poseía dicho *dataset* era la baja cantidad de atributos numéricos, además que dicho conjunto de datos estaban muy bien estructurados y limpios dado que la página web de la

que se extrajo era muy sencilla de tratar. Por estas razones, para no hacer la PRA 2 muy sencilla se ha optado por escoger otro juego de datos con una mayor cantidad de atributos numéricos y del cual desconocemos por completo como se encuentran.

1.1 Descripción e importancia del dataset

El *dataset* que escogeremos para esta ocasión será Wine Quality. El dataset contiene 6497 registros sobre vinos. Los atributos que encontramos serían: Acidez volatil, Acidez permanente, Acidez cítrica, Azúcar residual, Cloro, Dióxido de sulfuro libre, Total de dióxido de azufre, Densidad, pH, Sulfatos, Grado de alcohol y calidad del vino. Dicho dataset está creado a partir de la información ofrecida por *Vinho Verde*.

El dataset resulta muy interesante ya que ofrece las características fisico-químicas de los vinos lo que hace que sea más atractivo su análisis, además de ser parámetros más fácilmente controbales en las bodegas al no depender de un sommier especializado en vino con lo que ello conllevaría. Las características fisico-químicas se pueden obtener muy fácilmente con la instrumentación adecuada que toda bodega podría tener.

1.2 Objetivo del análisis

El vino es una bebida alcohólica elaborada con uvas fermentadas. La levadura consume el azúcar de la uva y lo convierte en etanol, dióxido de carbono y calor. Es una bebida alcohólica de agradable sabor. Definitivamente, será interesante analizar los atributos fisico-químicos del vino y comprender sus relaciones y su significado con la calidad y las clasificaciones de los tipos de vinos para dar respuestas a las siguientes preguntas:

- **¿Qué tipo de vino se trata si la muestra tiene los atributos que posee?**
- Predecir la calidad de cada muestra de vino **¿qué calidad tiene el vino con las características que posee?**

2 Integración y selección de los datos de interés a analizar

El paso de integrar y la seleccionar los datos de interés tiene especial relevancia para aquellos proyectos en los que se tienen más de una fuente de datos. El objetivo es poder crear un *dataset* global que recoja toda la información de las diferentes fuentes y se haga una selección de aquellos registros que sean de especial interés.

En nuestro caso, solo disponemos de un dataset con el que trabajar, además, no contamos con ningún grupo de especial interés ya que disponemos de dos objetivos, clasificar los vinos en blancos o tintos y predecir su calidad. Si descartasemos un grupo de datos podría darnos lugar a errores en nuestro futuro modelo de minería de datos. Una propuesta es que una vez realizado el modelo para la clasificación entre tintos y blancos, escogamos del dataset aquellos vinos tintos o blancos por separado y los tratemos como unidad.

Haremos una breve visualización del conjunto de datos con alguna que otra información de interés antes de comenzar nuestras tareas de limpieza y análisis.

[1]:
```
# Importamos librerías que emplearemos a lo largo de todo
# el proyecto. Este chunck será modificado
```

```
# tantas veces como veamos que necesitamos añadir una librería
# para el tratamiento de los datos.

import pandas as pd
import warnings
# Omitiremos las advertencias para no entorpecer en el
# visualizado
warnings.filterwarnings('ignore')

from matplotlib import pyplot as plt
from matplotlib.ticker import FormatStrFormatter
from matplotlib.colors import ListedColormap
from matplotlib import style
style.use('ggplot') or plt.style.use('ggplot')
%matplotlib inline
import seaborn as sns
from scipy.stats import f_oneway, skew, boxcox
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np
from sklearn.linear_model import LinearRegression
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

[2]:
```
# Realizamos la carga del dataset
df_wine = pd.read_csv('data/winequalityN.csv')

# Visualizamos las primeras 5 filas para observar como
# nos encontramos los atributos
df_wine.head(3)
```

[2]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6

Se puede observar por tanto que tenemos dos variables objetivos para nuestros modelos, la calidad y el tipo de vino.

Atributos principales	Target
fixed acidity	type
volatile acidity	quality
citric acid	
residual sugar	
chlorides	
free sulfur dioxide	
total sulfur dioxide	
density	
pH	
sulphates	
alcohol	

Continuemos explorando un poco más el conjunto de datos.

[3]:
```
# Información de los tipos de datos que tenemos
df_wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

[4]:
```
# Observemos por encima los valores únicos del dataset
df_unique = df_wine.apply(lambda x: [x.unique()])
```

Con esta breve exploración de los datos pasaremos a comenzar con nuestras tareas de tratamiento de valores nulos, que como podemos ver, tenemos en nuestro dataset. También realizaremos una

exploración de los valores extremos y valoraremos si es posible eliminarlos o por el contrario, deberemos mantenerlos para que nuestro dataset no pierda coherencia.

Por último, haremos una transformación del atributo *quality*, crearemos una nueva columna en el que daremos un sistema de notas similar al que se tiene en la UOC, es decir, empleando las notas A,B,C y D.

[5]:
```
# Realizamos la asignación de notas en una columna 'rating'
df_wine['rating'] = None
df_wine.loc[df_wine.quality < 5, 'rating'] = 'D'
df_wine.loc[df_wine.quality == 5, 'rating'] = 'C'
df_wine.loc[df_wine.quality == 6, 'rating'] = 'C'
df_wine.loc[df_wine.quality == 7, 'rating'] = 'B'
df_wine.loc[df_wine.quality == 8, 'rating'] = 'B'
df_wine.loc[df_wine.quality == 9, 'rating'] = 'A'
df_wine.loc[df_wine.quality == 10, 'rating'] = 'A'
```

3 Limpieza de los datos

3.1 Tratamiento de los valores nulos

En este apartado lo interesante será tratar los valores nulos pero los valores que tengan 0 habrá que tener cuidado en eliminarlos ya que la naturaleza de nuestros datos tienen posibilidad de adquirir como valor el 0, un vino puede estar suspenso con un 0, la acidez podría adquierir un valor de 0…

Dado que contamos con muy pocos valores nulos, eliminaremos dichos registros. Hay que tener en cuenta que tenemos más de 6000 registros, eliminar unos 38 como máximo (no supone ni un 1% del conjunto de los datos) para tal juego de datos es apenas eliminar nada.

Otro paso a realizar debería ser la eliminación de duplicados, pero nuestra hipótesis se basará en que puede que haya vinos que posean exactamente las mismas características, lo cual no sería raro en un juego de datos con más de 6000 registros.

[6]:
```
df_wine.dropna(inplace=True)
```

3.2 Tratamiento de valores *outliers*

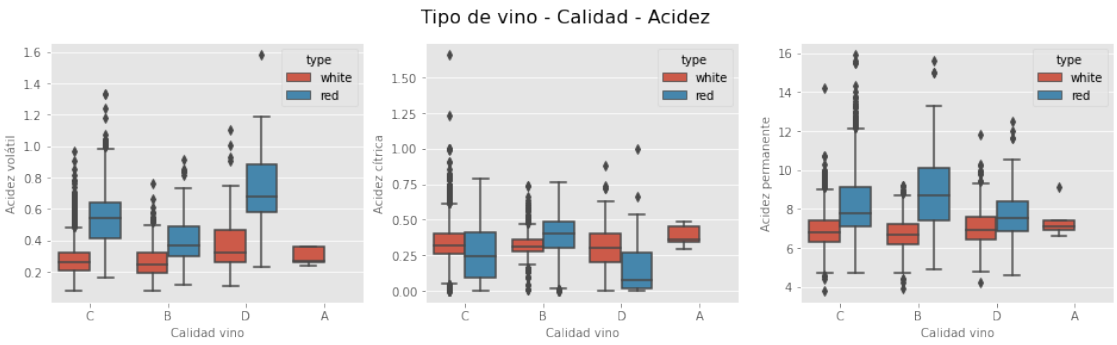
Nuestro conjunto de datos puede contener numerosos valores extremos. Hay que tener especial cuidado con dichos valores pues pueden perturbar nuestra muestra pero también tenemos que tener cuidado con su eliminación porque pueden ser datos totalmente reales.

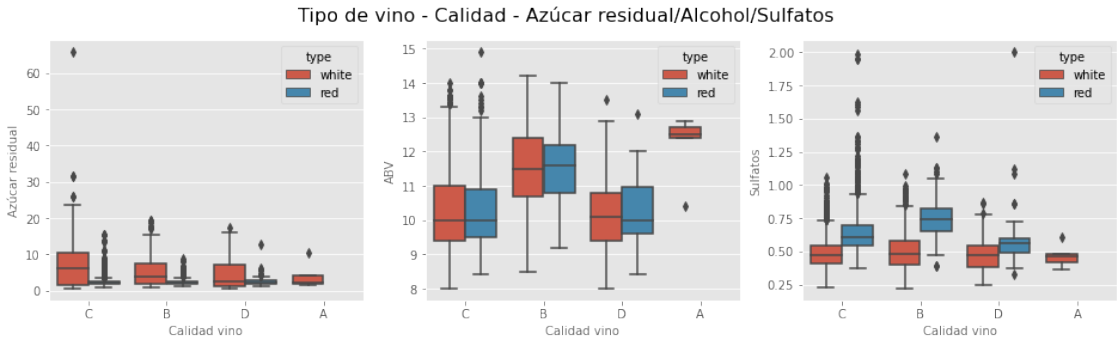
Hagamos una visualicación de los valores extremos empleando lo más común en estos casos que son las gráficas *boxplot* en aquellos atributos que merecen la pena observar: *volatile acidity*, *citric acid*, *fixed acidity*, *residual sugar*, *alcohol* y *sulphates*.

[7]:
```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
f.suptitle('Tipo de vino - Calidad - Acidez', fontsize=16)

sns.boxplot(x='rating', y='volatile acidity', hue='type', data=df_wine, ax=ax1)
ax1.set_xlabel("Calidad vino",size = 10,alpha=0.9)
```

```
ax1.set_ylabel("Acidez volátil",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='citric acid', hue='type', data=df_wine, ax=ax2)
ax2.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax2.set_ylabel("Acidez cítrica",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='fixed acidity', hue='type', data=df_wine, ax=ax3)
ax3.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax3.set_ylabel("Acidez permanente",size = 10,alpha=0.9)

f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
f.suptitle('Tipo de vino - Calidad - Azúcar residual/Alcohol/Sulfatos',␣
 ↪fontsize=16)

sns.boxplot(x='rating', y='residual sugar', hue='type', data=df_wine, ax=ax1)
ax1.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax1.set_ylabel("Azúcar residual",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='alcohol', hue='type', data=df_wine, ax=ax2)
ax2.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax2.set_ylabel("ABV",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='sulphates', hue='type', data=df_wine, ax=ax3)
ax3.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax3.set_ylabel("Sulfatos",size = 10,alpha=0.9)
```

[7]: Text(0, 0.5, 'Sulfatos')





Se puede ver que hay outliers. En esta parte como hemos comentado, hay infinidad de posibilidades y no siempre encontraremos defensores de la eliminación absoluta de los valores extremos. En nuestro caso, por dejar un ejemplo didáctico, realizaremos la eliminación de aquellos registros que contengan outliers empleando los quartiles 75 y 25, y manejando el rango interquartil IQR empleado en los *boxplots*.

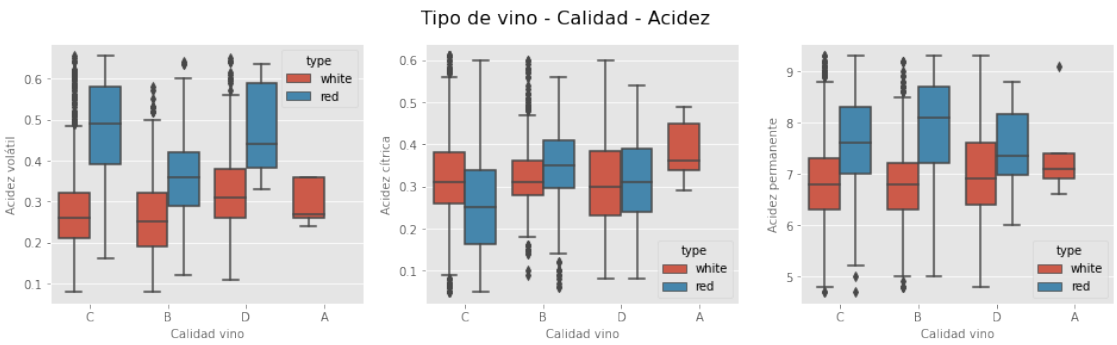
[8]:
```
def drop_outliers(df, column):

    q3=(np.percentile(df[column],75))
    q1=(np.percentile(df[column],25))

    iqr= 1.5 * (q3 - q1)

    df.drop(df[df[column] > (iqr + np.percentile(df[column], 75))].index,␣
↪inplace=True)
    df.drop(df[df[column] < (np.percentile(df[column], 25) - iqr)].index,␣
↪inplace=True)
```

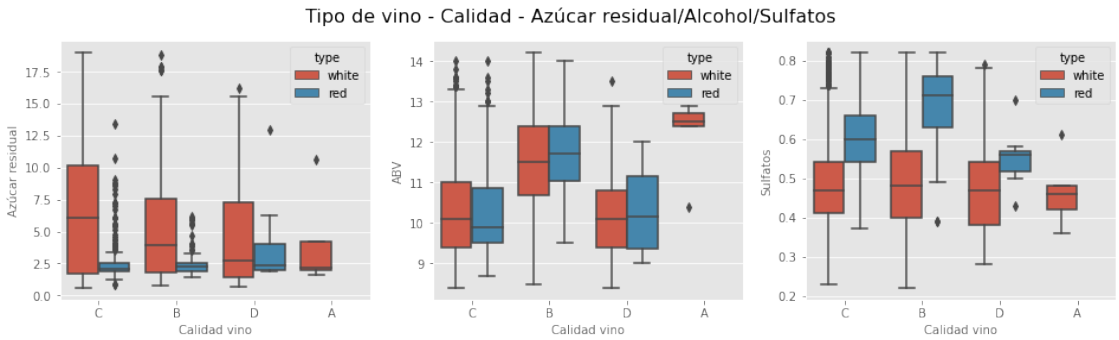
Con esto habremos definido una función que nos permitirá eliminar los *outliers*. Es hora de ponerlo en práctica.

[9]:
```
# Eliminamos outliers de volatile acidity
drop_outliers(df_wine,'volatile acidity')
# Eliminamos outliers de volatile acidity
drop_outliers(df_wine,'citric acid')
# Eliminamos outliers de volatile acidity
drop_outliers(df_wine,'fixed acidity')
# Eliminamos outliers de volatile acidity
drop_outliers(df_wine,'residual sugar')
# Eliminamos outliers de volatile acidity
drop_outliers(df_wine,'alcohol')
# Eliminamos outliers de volatile acidity
drop_outliers(df_wine,'sulphates')
```

[10]:
```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
f.suptitle('Tipo de vino - Calidad - Acidez', fontsize=16)

sns.boxplot(x='rating', y='volatile acidity', hue='type', data=df_wine, ax=ax1)
ax1.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax1.set_ylabel("Acidez volátil",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='citric acid', hue='type', data=df_wine, ax=ax2)
ax2.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax2.set_ylabel("Acidez cítrica",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='fixed acidity', hue='type', data=df_wine, ax=ax3)
ax3.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax3.set_ylabel("Acidez permanente",size = 10,alpha=0.9)

f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
f.suptitle('Tipo de vino - Calidad - Azúcar residual/Alcohol/Sulfatos',␣
↪fontsize=16)

sns.boxplot(x='rating', y='residual sugar', hue='type', data=df_wine, ax=ax1)
ax1.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax1.set_ylabel("Azúcar residual",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='alcohol', hue='type', data=df_wine, ax=ax2)
ax2.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax2.set_ylabel("ABV",size = 10,alpha=0.9)

sns.boxplot(x='rating', y='sulphates', hue='type', data=df_wine, ax=ax3)
ax3.set_xlabel("Calidad vino",size = 10,alpha=0.9)
ax3.set_ylabel("Sulfatos",size = 10,alpha=0.9)
```

[10]:
```
Text(0, 0.5, 'Sulfatos')
```





A pesar de haber tratado los *outliers* se siguen observando. Esto es debido a que a medida que vayamos eliminando regustros, las medidas como la media y la mediana seguirán cambiando por la eliminación de datos. En este aspecto, podemos entender a aquellos defensores del mantenimiento de los valores extremos ya que como se puede observar, esta tarea muchas veces se puede volver infinita hasta eliminar tal cantidad de registros que nuestro conjunto de datos se verá reducido e inutilizado. Nosotros, para mostrar un ejemplo hemos realizado únicamente la eliminación de los mismos una única vez, esto nos ha permitido al menos eliminar aquellos valores extremos que se diferenciaban del resto de manera muy exagerada, quedandonos un conjunto de datos algo más reducido pero sin perder muchos registros. A continuación, se observarán medidas estadísticas básicas y se podrá comprobar como se han suprimido gran cantidad de registros (contabamos inicialmente con 6497).

[11]:
```
df_wine.describe()
```

[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	\
count	5191.000000	5191.000000	5191.000000	5191.000000	
mean	6.940406	0.296828	0.317885	5.779426	
std	0.846616	0.113688	0.100207	4.695020	
min	4.700000	0.080000	0.050000	0.600000	
25%	6.400000	0.220000	0.260000	1.800000	
50%	6.900000	0.270000	0.310000	4.200000	
75%	7.400000	0.350000	0.370000	8.700000	
max	9.300000	0.655000	0.610000	18.950000	

	chlorides	free sulfur dioxide	total sulfur dioxide	density	\
count	5191.000000	5191.000000	5191.000000	5191.000000	
mean	0.049688	32.926411	127.005683	0.994215	
std	0.023705	17.231473	50.396558	0.002825	
min	0.009000	2.000000	6.000000	0.987110	
25%	0.036000	20.000000	97.000000	0.991900	
50%	0.045000	31.000000	126.000000	0.994200	
75%	0.054000	44.000000	161.000000	0.996400	
max	0.346000	289.000000	440.000000	1.002410	

	pH	sulphates	alcohol	quality

count	5191.000000	5191.000000	5191.000000	5191.000000
mean	3.209667	0.502712	10.525794	5.872086
std	0.152790	0.113495	1.201731	0.864530
min	2.790000	0.220000	8.400000	3.000000
25%	3.100000	0.420000	9.500000	5.000000
50%	3.200000	0.490000	10.400000	6.000000
75%	3.310000	0.570000	11.400000	6.000000
max	3.850000	0.820000	14.200000	9.000000

Cabe destacar que además hemos obtenido con esta primera eliminación de los valores extremos un conjunto de datos que cumple con la normativa **Reglamento CE 1493/99 y 423/2008, el "Codex Internacional de prácticas enológicas" y el "Codex enológico internacional" (productos)**, el cual establece los valores límites de la composición de productos enológicos, que en nuestro caso como ejemplo, teníamos vinos que superaban el límite de 1 gr/dm^3 de acidez cítrica y que con la primera eliminación de valores extremos, dichos valores se han suprimido.

4 Análisis de los datos

4.1 Selección de los grupos de datos

En nuestro caso, no escogeremos ningún grupo de datos ya que todos los registros nos son útiles, con la eliminación de los valores extremos y nulos ya hemos realizado una primera criba y en este paso, estableceremos que atributos son los que realmente deberemos tomar para el futuro modelo de datos. Por lo tanto, tomaremos todos los atributos para realizar este análisis y realizaremos las comprobaciones oportunas.

4.2 Comprobación de la normalidad y homogeneidad de la varianza

Conviene realizar un análisis estadístico inferencial. En la asignatura *"Tipología y ciclo de vida de los datos"* podemos encontrar diferentes técnicas para realizar dicho análisis: *"Este tipo de análisis tiene por objetivo modelar lo datos a través de una distribución conocida"*. Se parte de la premisa de que el conjunto de datos que se esta estudiando representa una fracción de la totalidad de un gran grupo mayor, y el objetivo será inferir en cómo es ese grupo, tomando como hipótesis que se asume un grado de error dado que solo contamos con una fracción de ese grupo mayor.

Para análisis inferenciales entre uno o dos grupos se suele realizar la comprobación de la normalidad con métodos como los tests de Kolmogorov-Smirnov o el de Shaphiro-Wilk, además de tenerse en cuenta el teorema central de límite, siendo este último el más potente, también se realiza una comprobación de la homocedasticidad. En la comparación de dos grupos se emplea el famoso test de Student si tratamos hipótesis de tipo paramétrico o si no es el caso, se emplea Wilcoxon o Mann-Whitney.

El problema es que en nuestro conjunto de datos estamos comparando más de dos grupos, eso nos lleva a tener problemas con los métodod inferenciales anteriores. No obstante, contamos con el análisis de varianza unidireccional conocido como ANOVA, análisis de un solo factor, tratandose de una extensión de la prueba t de Student, que tiene por objetivo comprar las medias entre más de dos grupos de datos. Pero ANOVA es para problemas de tipo paramétrico, en caso de no ser así, para el tipo no paramétrico contaríamos con el test de Kruskal-Wallis.

Para nuestro conjunto de datos, vamos a emplear la prueba de Fisher en el ANOVA y el valor

asociado de p-value para determinar la significancia y conocer si podemos descartar la hipótesis nula. Podríamos resumir lo anterior diciendo que un valor alto en la prueba de Fisher y un p-value pequeño significará una correlación entre las variables y el target y un valor de p-value cerca del 0 indicará significancia estadística muy probable. Además añadir que el nivel límite de significancia que se suele aplicar es α=0.05.

Emplearemos las diferencias más significativas que hemos observado en el pequeño análisis estadistico anterior:

[12]:
```
# Definiremos una función que nos calcule la significancia
# y cruzaremos varias variables de modo que conozcamos si la
# hipótesis nula es rechazada o no. Dado que tenemos dos atributos
# que son target, elaboraremos dos funciones para comprobar
# su inferencia:

def statical_inference_type(attribute):
    Fisher, p_value = f_oneway(df_wine[df_wine.type == 'white'][attribute],
                               df_wine[df_wine.type == 'red'][attribute])
    if p_value <= 0.05:
        hypothesis = 'rechazada'
    else:
        hypothesis = 'aceptada'
    print('Para {}: Fisher: {:.4f} --- p_value: {:.6f} --- La hipótesis nula es␣
 ↪{}'
          .format(attribute, Fisher, p_value, hypothesis))

def statical_inference_rating(attribute):
    Fisher, p_value = f_oneway(df_wine[df_wine.rating == 'D'][attribute],
                               df_wine[df_wine.rating == 'C'][attribute],
                               df_wine[df_wine.rating == 'B'][attribute],
                               df_wine[df_wine.rating == 'A'][attribute])
    if p_value <= 0.05:
        hypothesis = 'rechazada'
    else:
        hypothesis = 'aceptada'
    print('Para {}: Fisher: {:.4f} --- p_value: {:.6f} --- La hipótesis nula es␣
 ↪{}'
          .format(attribute, Fisher, p_value, hypothesis))

list_check = ['fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol']
# Probamos primero con el target type
print('ANOVA para la variable objetivo Type (Tipo de vino):')
for i in list_check:
    statical_inference_type(i)
print('\nANOVA para la variable objetivo rating (Calidad del vino):')
for j in list_check:
```

```
    statical_inference_rating(j)
```

```
ANOVA para la variable objetivo Type (Tipo de vino):
Para fixed acidity: Fisher: 597.7568 --- p_value: 0.000000 --- La hipótesis nula
es rechazada
Para volatile acidity: Fisher: 2398.9987 --- p_value: 0.000000 --- La hipótesis
nula es rechazada
Para citric acid: Fisher: 172.1322 --- p_value: 0.000000 --- La hipótesis nula
es rechazada
Para residual sugar: Fisher: 413.4444 --- p_value: 0.000000 --- La hipótesis
nula es rechazada
Para chlorides: Fisher: 1547.9264 --- p_value: 0.000000 --- La hipótesis nula es
rechazada
Para free sulfur dioxide: Fisher: 741.6098 --- p_value: 0.000000 --- La
hipótesis nula es rechazada
Para total sulfur dioxide: Fisher: 2543.6985 --- p_value: 0.000000 --- La
hipótesis nula es rechazada
Para density: Fisher: 394.0978 --- p_value: 0.000000 --- La hipótesis nula es
rechazada
Para pH: Fisher: 583.3769 --- p_value: 0.000000 --- La hipótesis nula es
rechazada
Para sulphates: Fisher: 909.7206 --- p_value: 0.000000 --- La hipótesis nula es
rechazada
Para alcohol: Fisher: 7.0339 --- p_value: 0.008023 --- La hipótesis nula es
rechazada

ANOVA para la variable objetivo rating (Calidad del vino):
Para fixed acidity: Fisher: 8.8849 --- p_value: 0.000007 --- La hipótesis nula
es rechazada
Para volatile acidity: Fisher: 29.7817 --- p_value: 0.000000 --- La hipótesis
nula es rechazada
Para citric acid: Fisher: 4.2664 --- p_value: 0.005126 --- La hipótesis nula es
rechazada
Para residual sugar: Fisher: 13.1581 --- p_value: 0.000000 --- La hipótesis nula
es rechazada
Para chlorides: Fisher: 69.0689 --- p_value: 0.000000 --- La hipótesis nula es
rechazada
Para free sulfur dioxide: Fisher: 10.8632 --- p_value: 0.000000 --- La hipótesis
nula es rechazada
Para total sulfur dioxide: Fisher: 15.1397 --- p_value: 0.000000 --- La
hipótesis nula es rechazada
Para density: Fisher: 155.9318 --- p_value: 0.000000 --- La hipótesis nula es
rechazada
Para pH: Fisher: 3.0283 --- p_value: 0.028275 --- La hipótesis nula es rechazada
Para sulphates: Fisher: 2.5859 --- p_value: 0.051411 --- La hipótesis nula es
aceptada
Para alcohol: Fisher: 305.4769 --- p_value: 0.000000 --- La hipótesis nula es
rechazada
```

Es decir, la hipótesis nula no se cumple para ningún caso.

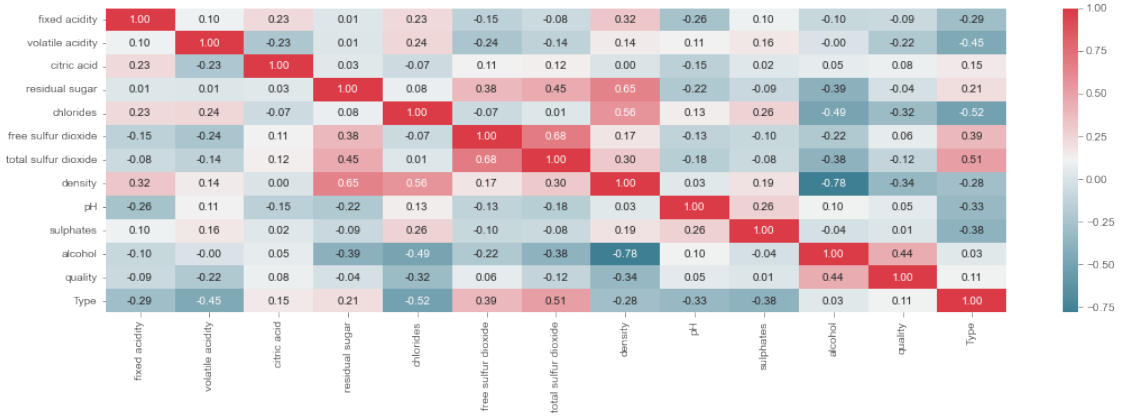
4.3 Aplicación de pruebas estadísticas

El siguiente paso más común es estudiar la correlatividad de las variables. Para ello primero deberemos preparar un poco el dataset para que el resultado de la matriz de correlatividad que podamos extraer sea lo más realista posible, tratando las variables *target*.

En la asignatura *"Tipología y ciclo de vida de lo datos"* nos enseñan a que el coeficiente de correlación de Pearson es el más utilizado entre las variables relacionadas linealmente aunque requiere que la distribución entre variables sea normal y cumpla el criterio de homocedasticidad, cosa que con p-value por debajo del nivel de significancia no suele cumplir. Para la alternativa no paramétrica se tiene la correlación de Spearman que mide el grado de dependencia entre dos variables también y no conlleva a ninguna suposición sobre la distribución de los datos. En la página de cienciadedatos.net podemos ver ejemplos de como aplicar ambas correlaciones además de una explicación más exahustiva sobre la correlación.

[13]:
```
# Tratamos las variables target:
type_encoder = LabelEncoder()
target_type = type_encoder.fit_transform(df_wine.type.values)
df_wine['Type'] = target_type

type_rating = {'D':0, 'C':1, 'B':2, 'A':3}
target_rating = df_wine.rating.map(type_rating)
```

[14]:
```
# Realizamos lo mismo para la correlación con el coeficiente de Spearman
correlation = df_wine.corr(method='spearman')
correlation_columns = correlation.Type.keys()
total_correlation = correlation.loc[correlation_columns, correlation_columns]
plt.figure(figsize=(18, 5))
sns.heatmap(total_correlation, cmap=sns.diverging_palette(220, 10,␣
 ↪as_cmap=True), annot=True, fmt=".2f")
sns.set(font_scale=1.5)
plt.show()
```



Observando la correlatividad podremos hacernos una idea de los atributos que parece que no vamos a necesitar.

Si es verdad que hemos visto la correlación entre las variables dos a dos, pero nos hemos dejado pasar el problema de la multicolinealidad. Uno de los métodos más empleados es el de usar el Factor de Inflación de la Varianza (FIV), donde dicho método realiza una regresión lineal de un parámetro sobre cada uno de los demás atributos. El descubrir que atributos nos perturbarían nuestro set de datos nos dará la posibilidad de eliminar algunos de ellos, lo que nos reducirá la dimensionalidad.

En it-swarm_es nos enseñan como poder gestionar este paso con Python, sin embargo, puede que necesitemos una librería que no este instalada en nuestro ordenador (statsmodels). En ese caso en profesordata tenemos una alternativa empleando únicamente sklearn.

[15]:
```
# Realizaremos una copia de nuestro dataset para mantener
# un dataset sin ningún tipo de cambio
df_wine_origin = df_wine.copy(deep=True)

# Separamos las variables que emplearemos como target
target_type = 'type'
target_quality = 'quality'
target_rating = 'rating'
columns_df = df_wine.columns.to_list()
columns_df.remove(target_type)
columns_df.remove(target_quality)
columns_df.remove(target_rating)
x_pred = df_wine[columns_df]
```

[16]:
```
# Elaboraremos una función que nos permita fácilmente
# llamarla como el ejemplo que tenemos en la web:
# Debo importar LinearRegression para el calculo de las Ri
def cVIF(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

# Probamos la función:
cVIF(x_pred.copy(deep=True)).T
```

[16]:

	VIF
fixed acidity	3.416255
volatile acidity	1.714843
citric acid	1.255825
residual sugar	14.056745
chlorides	1.611110
free sulfur dioxide	2.039623
total sulfur dioxide	3.312256
density	35.448543
pH	2.705052
sulphates	1.385276
alcohol	9.836234
Type	5.745717

Normalmente, cuando una variable tiene un VIF mayor a 5, significa que es una variable que se debe eliminar por presentar una gran multicolinealidad con el resto. Habrá que realizar este paso repetidas veces para eliminar en cada paso un único atributo, ya que el eliminar dos atributos de golpe podría llevarnos a un error.

Eliminamos pues primero la densidad y también las variables que nos han servido como atributos a predecir (es decir, *Type*).

[17]:
```
df_wine_origin = df_wine.copy(deep=True)
target_type = 'type'
target_Type = 'Type'
target_quality = 'quality'
target_rating = 'rating'
density = 'density'
columns_df = df_wine.columns.to_list()
columns_df.remove(target_type)
columns_df.remove(target_rating)
columns_df.remove(target_quality)
columns_df.remove(density)
columns_df.remove(target_Type)
x_pred = df_wine[columns_df]

cVIF(x_pred.copy(deep=True)).T
```

[17]:

	VIF
fixed acidity	1.329258
volatile acidity	1.371979
citric acid	1.244622
residual sugar	1.505944
chlorides	1.409053
free sulfur dioxide	1.923509
total sulfur dioxide	2.243720
pH	1.321639
sulphates	1.168597
alcohol	1.565171

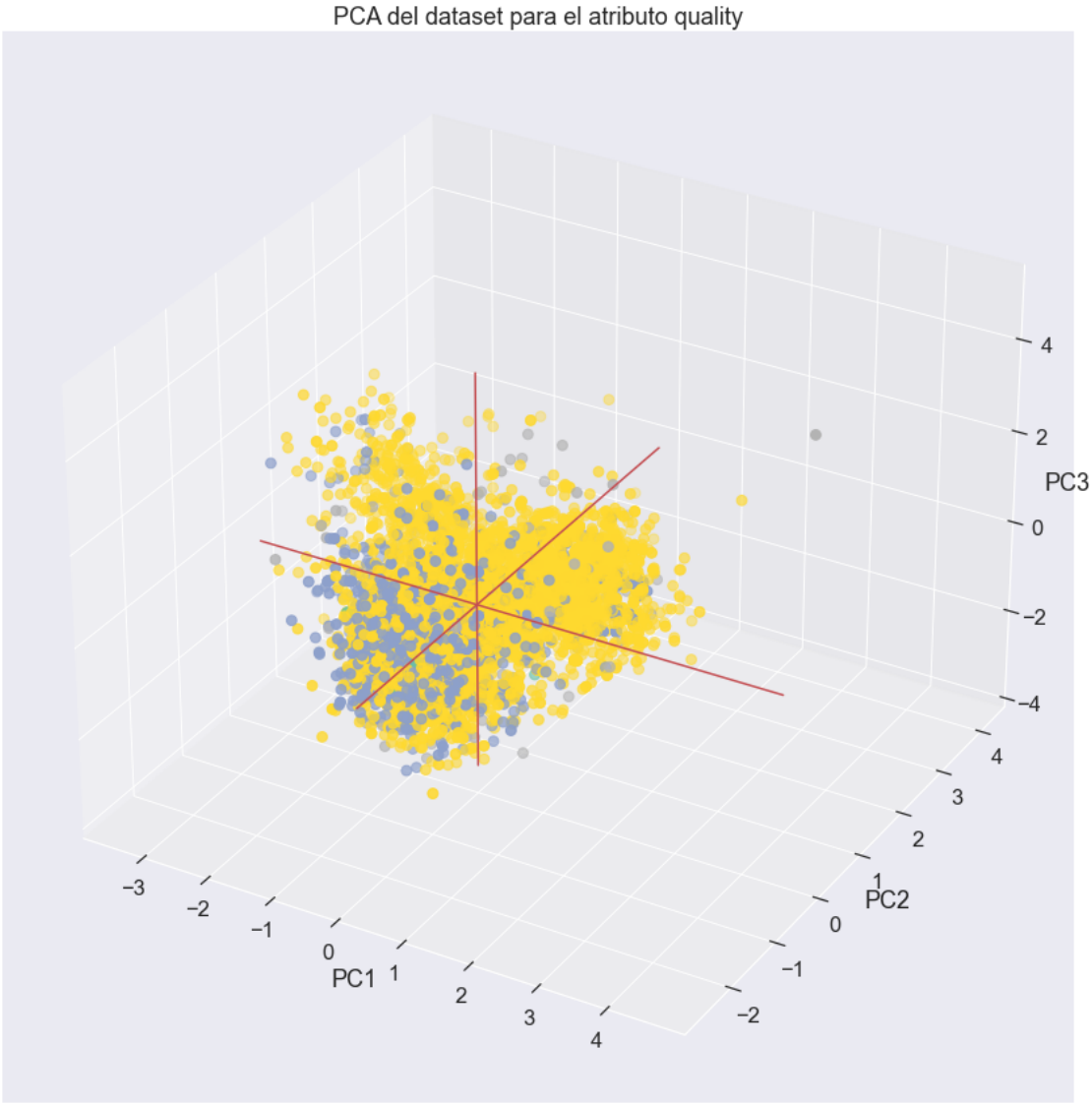
De este modo, habremos hecho una selección de las variables a emplear, eliminamos *density* de nuestro set de datos.

Dado este punto, podríamos eliminar aquellas variables que sean muy colineales y multicolineales, pero para terminar este punto, estableceremos un estudio sobre las componentes principales PCA aprovechando que nuestro dataset es puramente numérico. Las componentes principales son variables nuevas que poseen independencia entre ellas, sería como realizar un cambio de base en un espacio multidimensional de tal manera que los ejes fueran lo más ortogonalmente posible. Esto nos podrá reducir notoriamente el número de variables de nuestro dataset sin perder información relevante.

[18]:
```
# Tratamos las variables target:
type_encoder = LabelEncoder()
target_type = type_encoder.fit_transform(df_wine.type.values)
df_wine['Type'] = target_type

type_rating = {'D':0, 'C':1, 'B':2, 'A':3}
target_rating = df_wine.rating.map(type_rating)

def pca_get(dataset, target, attr):
    scale = StandardScaler()
    dataset = pd.DataFrame(scale.fit_transform(dataset), index=dataset.index)
    pca_recop = PCA(random_state=101, whiten=True).fit(dataset)
    color = target
    resultados=pd.DataFrame(pca_recop.transform(dataset),
                        columns=['PCA%i' % i for i in range(dataset.shape[1])],
                        index=dataset.index)
    fig = plt.figure(figsize=(15,15))
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(resultados['PCA0'], resultados['PCA1'], resultados['PCA2'],
               c=color, cmap="Set2_r", s=60)
    xAxisLine = ((min(resultados['PCA0']), max(resultados['PCA0'])), (0, 0),␣
 ↪(0,0))
    ax.plot(xAxisLine[0], xAxisLine[1], xAxisLine[2], 'r')
    yAxisLine = ((0, 0), (min(resultados['PCA1']), max(resultados['PCA1'])),␣
 ↪(0,0))
```

```
    ax.plot(yAxisLine[0], yAxisLine[1], yAxisLine[2], 'r')
    zAxisLine = ((0, 0), (0,0), (min(resultados['PCA2']),␣
↪max(resultados['PCA2'])))
    ax.plot(zAxisLine[0], zAxisLine[1], zAxisLine[2], 'r')
    ax.set_xlabel("PC1")
    ax.set_ylabel("PC2")
    ax.set_zlabel("PC3")
    ax.set_title("PCA del dataset para el atributo " + (attr))
    plt.show()
    X_train , X_test, y, y_test = train_test_split(dataset ,
                                                   target, test_size=0.3,
                                                   random_state=0)
    KNC = KNeighborsClassifier(algorithm = 'ball_tree', leaf_size = 14,
                               n_neighbors = 12, p  = 1, weights = 'distance')
    KNC = KNC.fit(X_train, y)
    print('KNeighbors Classifier Training Accuracy: {:2.4%}'
          .format(accuracy_score(y, KNC.predict(X_train))))
    y_pred = KNC.predict(X_test)
    print('KNeighbors Classifier Test Accuracy: {:2.4%}'
          .format(accuracy_score(y_test, y_pred)))
    print('_' * 40)
    print('\nExactitud de', attr, 'Predicción por el número de componentes PCA:
↪\n')
    PcaAccu = pd.DataFrame(columns=['Components', 'Var_ratio',
                                    'Train_Acc', 'Test_Acc'])
    for componets in np.arange(1, dataset.shape[1]):
        variance_ratio = sum(pca_recop.explained_variance_ratio_[:
↪componets])*100
        pca = PCA(n_components=componets, random_state=101, whiten=True)
        X_train_pca = pca.fit_transform(X_train)
        Components = X_train_pca.shape[1]
        KNC = KNeighborsClassifier(algorithm = 'ball_tree',
                                   leaf_size = 12, n_neighbors = 12,
                                   p  = 1, weights = 'distance')
        KNC = KNC.fit(X_train_pca, y)
        Training_Accuracy = accuracy_score(y, KNC.predict(X_train_pca))
        X_test_pca = pca.transform(X_test)
        y_pred = KNC.predict(X_test_pca)
        Test_Accuracy = accuracy_score(y_test, y_pred)
        PcaAccu = PcaAccu.append(pd.DataFrame([(Components,
                                               variance_ratio, Training_Accuracy,
                                               Test_Accuracy)],
                                              columns=['Components', 'Var_ratio',
                                                       'Train_Acc', 'Test_Acc']))
    PcaAccu.set_index('Components', inplace=True)
    display(PcaAccu.sort_values(by='Test_Acc', ascending=False))
```

```
columns = df_wine.columns
columns = list(columns.drop(['type', 'rating', 'quality']))
pca_rating = pca_get(df_wine.loc[:, columns],target_rating, 'quality')
```




```
KNeighbors Classifier Training Accuracy: 100.0000%
KNeighbors Classifier Test Accuracy: 82.8626%
----------------------------------------

Exactitud de quality Predicción por el número de componentes PCA:

            Var_ratio  Train_Acc  Test_Acc
Components
```

```
7          87.526268       1.0  0.831194
8          92.013389       1.0  0.830552
10         98.570264       1.0  0.829910
9          95.735356       1.0  0.827985
11         99.866635       1.0  0.825417
6          82.444324       1.0  0.812580
5          76.321473       1.0  0.810013
2          48.703767       1.0  0.799101
4          69.831710       1.0  0.795250
3          60.645172       1.0  0.793325
1          26.200235       1.0  0.765725
```

Dependiendo de cual fuese nuestra variable *target* podríamos emplear 7 componentes principales (si deseamos predecir la calidad del vino).

Para no reducir tan drásticamente la cantidad de variables y poder conservar la comprensión del *dataset* de manera más sencilla, se empleará el uso de la correlación y la multicolinealidad para definir el conjunto de datos definitivo, este será eliminando por tanto la variable densidad del mismo.

[19]:
```
df_wine_origin = df_wine.copy(deep=True)
target_type = 'type'
target_quality = 'quality'
density = 'density'
columns_df = df_wine.columns.to_list()
columns_df.remove(target_type)
columns_df.remove(target_quality)
columns_df.remove(density)
df_tocsv = df_wine[columns_df]

# Exportamos dataset tratado
df_tocsv.to_csv('data/df_wine_cleaned.csv', index=False, sep=',')
```

5 Resolución del problema

Gracias a las anteriores acciones en nuestro *dataset* original, ahora contaríamos con un conjunto de datos con el que trabajar y poder resolver nuestro problema.

Para no extender más la práctica con código y dado que el objetvo de esta PRA es realmente la de tratar los datos, se ha generado un archivo plano .py en el que se resuelve el problema con modelado y usando el *dataset* final.

Los pasos realizados en la práctica nos han permitido perfeccionar un *dataset* el cual desconocíamos en un principio. Tratando los nulos y los valores extremos podremos eliminar aquellos valores que perturban los valores estadísticos que son de vital importancia en los modelos de minería de datos. Con las técnicas estadísticas hemos logrado descubrir que atributos nos son más interesantes y cuáles deberíamos de eliminar si quisieramos mantener un *dataset* reducido sin perder información. Con la resolución del problema daríamos las respuestas que buscamos, la de poder detectar cuando un vino tendrá una buena calidad y esto es especialmente útil para una empresa ya que permitirá

tener controlado los parámetros para asegurarse un éxito rotundo. En el directorio de GitHub se podrá ver como se ha elaborado el modelo y estar disponible para cuando se quiera predecir la calidad del vino.

6 Bibliografía y recursos web

Para la elaboración de esta práctica se han seguido los siguientes recursos:

- Subirats Maté, L., Pérez Trenard, D.O., Calvo González, M. (2021). Introducción a la limpieza y análisis de los datos. UOC.
- VanderPlas, J. (2016). Python Data Science Handbook. O'Reilly.
- VanderPlas, J. (2016). A Whirlwind Tour of Python. O'Reilly.
- Módulo de teoría de la asignatura minería de datos: Proceso de minería de datos (2021).
- Módulo de teoría de la asignatura minería de datos: Preprocesado de datos (2021).
- Módulo de teoría de la asignatura minería de datos: Gestión de características (2021).
- Módulo de teoría de la asignatura minería de datos: Caso de estudio (2021).
- Módulo de teoría de la asignatura minería de datos: Evaluación de modelos (2021)
- Módulo de teoría de la asignatura minería de datos: Modelos no supervisados (2021).
- Método de Fisher por Ruben J. Romo
- Para el empleo de todas las herramientas de la librería sklearn de Python.
- Dataset para el empleado en la práctica: Obtenido a traves de la RStudio y transformado a .csv.
- Torrano, C., Recuero, P., Ramirez, F., Hernández, S., Torres, J. (2019). Machine Learning aplicado a Ciberseguridad: Técnicas y ejemplos en la detección de amenazas. OxWord.
- Alonso, C., González, P., Garrido, J., Lukic, I., Pérez, D., Ramos, A., Barroso, D., Palop, I., Selvi, J., Picó, J., Aguayo, J.M., (2013). Hacking de dispositivos iOS: iPhone & iPad. OxWord.
- Información sobre la correlación lineal aplicada en Python.

Contribuciones	Firma
Investigación previa	Alejandro Pérez Manrique
Redacción de las respuestas	Alejandro Pérez Manrique
Desarrollo código	Alejandro Pérez Manrique