M2.855_20202_PEC1-Enunciado

March 10, 2021

M2.855 · Modelos avanzados de minería de datos · PEC1

2020-2 · Máster universitario en Ciencia de datos (Data science)

Estudios de Informática, Multimedia y Telecomunicación

1 PEC 1: Preparación de datos

A lo largo de esta práctica veremos cómo aplicar diferentes tecnicas para la carga y preparación de datos:

Carga de un conjunto de datos

Análisis de los datos 2.1 Análisis estadístico básico 2.2 Análisis exploratorio de los datos

Reducción de la dimensionalidad

Entrenamiento y test

Importante: Cada uno de los ejercicios puede suponer varios minutos de ejecución, por lo que la entrega debe hacerse en formato notebook y en formato html donde se vea el código, los resultados y comentarios de cada ejercicio. Se puede exportar el notebook a html desde el menú File → Download as → HTML.

Para realizar esta PEC, necesitaremos las siguientes librerías:

Nombre y apellidos: Francisco Javier Melchor González

[1]:
```
from IPython.display import Image
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import datasets
from sklearn import preprocessing
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪precision_recall_fscore_support
from sklearn.model_selection import train_test_split, cross_val_score
```

```
from sklearn import tree
from six import StringIO
import pydotplus

import matplotlib.pyplot as plt


pd.set_option('display.max_columns', None)

%matplotlib inline
```

2 1. Carga del conjunto de datos (1 punto)

En primer lugar, debéis cargar el conjunto de datos Wine recognition (más información en el enlace https://archive.ics.uci.edu/ml/datasets/Wine). Se puede descargar de internet o se puede cargar directamente desde la librería "scikit-learn", que incorpora un conjunto de datasets muy conocidos y usados para la minería de datos y machine learning: http://scikit-learn.org/stable/datasets/index.html.

Ejercicio: Cargad el conjunto de datos "Wine Recognition" y mostrad:

- el número y nombre de los atributos (variables que podrían ser usadas para predecir la respuesta "wine_class") - el número de filas del conjunto de datos - verificad si hay o no "missing values" y en qué columnas

Sugerencia: Si usáis sklearn (sklearn.datasets.load_wine), explorad las diferents 'keys' del objecto obtenido.

Sugerencia: Igual os resulta útil pasar los datos (atributos + target) a un dataframe de pandas.

Solución:

- **Carga de datos:**

[2]:
```
wine = pd.read_csv('../data/wine.csv')
wine.head()
```

[2]:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	\
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	

	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	0.28	2.29	5.64	1.04	3.92	1065
1	0.26	1.28	4.38	1.05	3.40	1050
2	0.30	2.81	5.68	1.03	3.17	1185
3	0.24	2.18	7.80	0.86	3.45	1480

```
4                  0.39      1.82        4.32  1.04  2.93       735
```

[3]:
```
wine.rename(columns={"Alcohol":"alcohol", "Mg":"magnesium", "Color.int":
 ↪"color_intensity", "Wine":"wine_class"},inplace=True)
```

- **Número y nombre de los atributos (variables que podrían ser usadas para predecir la respuesta "wine_class")**

[4]:
```
len(wine.columns[1:])
```

[4]:
```
13
```

[5]:
```
wine.columns[1:]
```

[5]:
```
Index(['alcohol', 'Malic.acid', 'Ash', 'Acl', 'magnesium', 'Phenols',
       'Flavanoids', 'Nonflavanoid.phenols', 'Proanth', 'color_intensity',
       'Hue', 'OD', 'Proline'],
      dtype='object')
```

- **El número de filas del conjunto de datos**

[6]:
```
len(wine)
```

[6]:
```
178
```

- **Verificad si hay o no "missing values" y en qué columnas**

[7]:
```
wine.isna().sum()
```

[7]:
```
wine_class              0
alcohol                 0
Malic.acid              0
Ash                     0
Acl                     0
magnesium               0
Phenols                 0
Flavanoids              0
Nonflavanoid.phenols    0
Proanth                 0
color_intensity         0
Hue                     0
OD                      0
Proline                 0
dtype: int64
```

El método anterior genera la suma de todos los elementos con valor "NA" para cada una de las columnas presentes en el dataset, como se puede observar todas las columnas tienen un valor 0 a la derecha, lo que indica que no existen valores NA en este dataset

3 2. Análisis de los datos (3 puntos)

3.1 2.1 Análisis estadístico básico

Ejercicio: Realizad un análisis estadístico básico: - Variables categóricas:

- Calculad la frecuencia.
- Haced un gráfico de barras.

- Variables numéricas: - Calculad estadísticos descriptivos básicos: media, mediana, desviación estándar, … - Haced un histograma de las variables: alcohol, magnesium y color_intensity

Sugerencia: podéis usar la librería 'pandas' y sus funciones 'describe' y 'value_counts'

Solución:

[8]:
```
wine.dtypes
```

[8]:
```
wine_class                int64
alcohol                 float64
Malic.acid              float64
Ash                     float64
Acl                     float64
magnesium                 int64
Phenols                 float64
Flavanoids              float64
Nonflavanoid.phenols    float64
Proanth                 float64
color_intensity         float64
Hue                     float64
OD                      float64
Proline                   int64
dtype: object
```

3.1.1 Variables categóricas

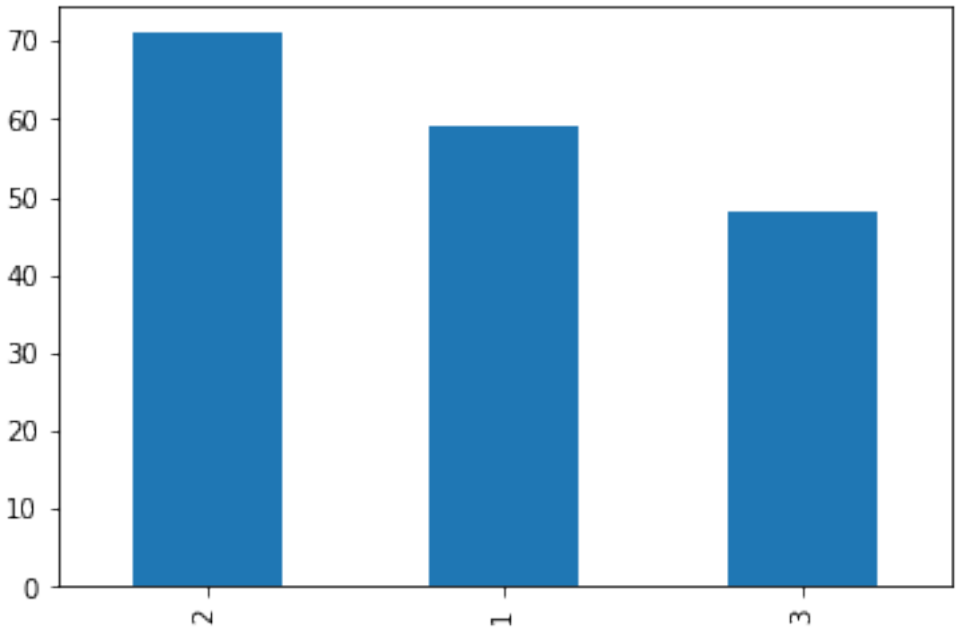
[9]:
```
wine['wine_class'] = wine['wine_class'].astype('category')
```

[10]:
```
wine['wine_class'].value_counts()
```

[10]:
```
2    71
1    59
3    48
Name: wine_class, dtype: int64
```

[11]:
```
wine['wine_class'].value_counts().plot(kind='bar')
```

[11]:
```
<AxesSubplot:>
```



3.1.2 Variables numéricas

[12]:
```
wine.describe()
```

[12]:

	alcohol	Malic.acid	Ash	Acl	magnesium	Phenols
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000

	Flavanoids	Nonflavanoid.phenols	Proanth	color_intensity
count	178.000000	178.000000	178.000000	178.000000
mean	2.029270	0.361854	1.590899	5.058090
std	0.998859	0.124453	0.572359	2.318286
min	0.340000	0.130000	0.410000	1.280000
25%	1.205000	0.270000	1.250000	3.220000
50%	2.135000	0.340000	1.555000	4.690000
75%	2.875000	0.437500	1.950000	6.200000
max	5.080000	0.660000	3.580000	13.000000

	Hue	OD	Proline
count	178.000000	178.000000	178.000000

```
mean      0.957449    2.611685   746.893258
std       0.228572    0.709990   314.907474
min       0.480000    1.270000   278.000000
25%       0.782500    1.937500   500.500000
50%       0.965000    2.780000   673.500000
75%       1.120000    3.170000   985.000000
max       1.710000    4.000000  1680.000000
```

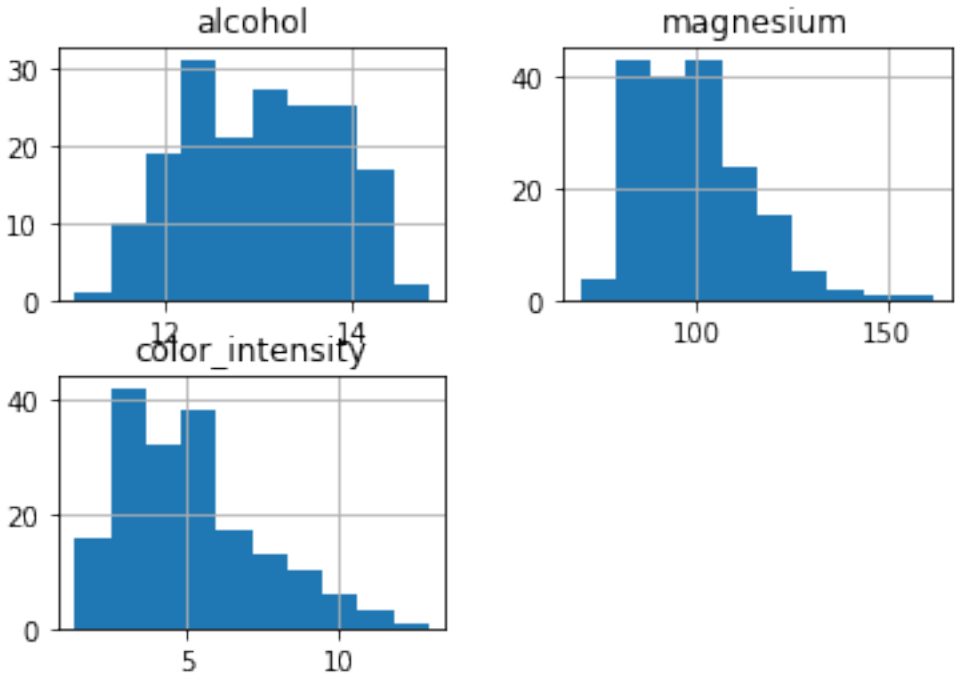
[13]:
```
wine.columns
```

[13]:
```
Index(['wine_class', 'alcohol', 'Malic.acid', 'Ash', 'Acl', 'magnesium',
       'Phenols', 'Flavanoids', 'Nonflavanoid.phenols', 'Proanth',
       'color_intensity', 'Hue', 'OD', 'Proline'],
      dtype='object')
```

[14]:
```
wine_selected = wine[['alcohol','magnesium','color_intensity']]
```

[15]:
```
wine_selected.hist()
```

[15]:
```
array([[<AxesSubplot:title={'center':'alcohol'}>,
        <AxesSubplot:title={'center':'magnesium'}>],
       [<AxesSubplot:title={'center':'color_intensity'}>, <AxesSubplot:>]],
      dtype=object)
```



Análisis: Comentad los resultados.

Solución:

- **Con respecto a la única variable categórica que hay en el dataset**, que es la variable ***Wine***, la cual permite clasificar los tipos de vinos que hay en el dataset, podemos observar en el histograma generado que existen 3 clases de vinos y que la **clase que más predomina es la clase "2"**, seguida de la "1" y de la "3" respectivamente.

  A pesar de tener más casos en unas que en otras, **no parece ser un dataset muy desbalanceado**, ya que no hay una gran diferencia entre unas clases y otras

- Por otro lado, **con respecto al conjunto total de variables numéricas**, podemos observar que **existen valores atípicos** en las siguientes variables:

  - ***Malic.acid***
  - ***Ash***
  - ***Acl***
  - ***magnesium***
  - ***Proanth***
  - ***color_intensity***

- Y con respecto a las variables ***alcohol, magnesium y color_intensity***, **sus distribuciones son**: **uniforme** en el caso de la variable ***alcohol*** y **unimodal con sesgo hacia la izquierda** en el caso de ***magnesium y color_intensity***

3.2 2.2 Análisis exploratorio de los datos

En este ejercicio exploraremos la relación de algunos de los atributos numéricos con la variable respuesta ("wine_class"), tanto gráficamente como cualitativamente, y analizaremos las diferentes correlaciones. Para empezar, seleccionaremos sólo 3 atributos para explorar: alcohol, magnesium y color_intensity.

[16]:
```
feats_to_explore = ['alcohol', 'magnesium', 'color_intensity']
```

Ejercicio: Usando una librería gráfica, como por ejemplo "matplotlib", realizad un gráfico del histograma de valores para cada uno de los atributos seleccionados, separados por los valores de la clase respuesta. Los tres gráficos tienen que estar sobrepuestos, es decir, por ejemplo, en el histograma de la feature "alcohol" tiene que haber en un solo gráfico tres histogramas, uno por cada clase de vino. Añadid una leyenda para saber a qué clase corresponde cada histograma.

La finalidad es observar cómo se distribuye cada uno de los atributos en función de la clase que tengan, para poder identificar de manera visual y rápida si algunos atributos permiten diferenciar de forma clara las diferentes clases de vinos.

Sugerencia: podéis usar el parámetro "alpha" en los gráficos para que se aprecien los tres histogramas.

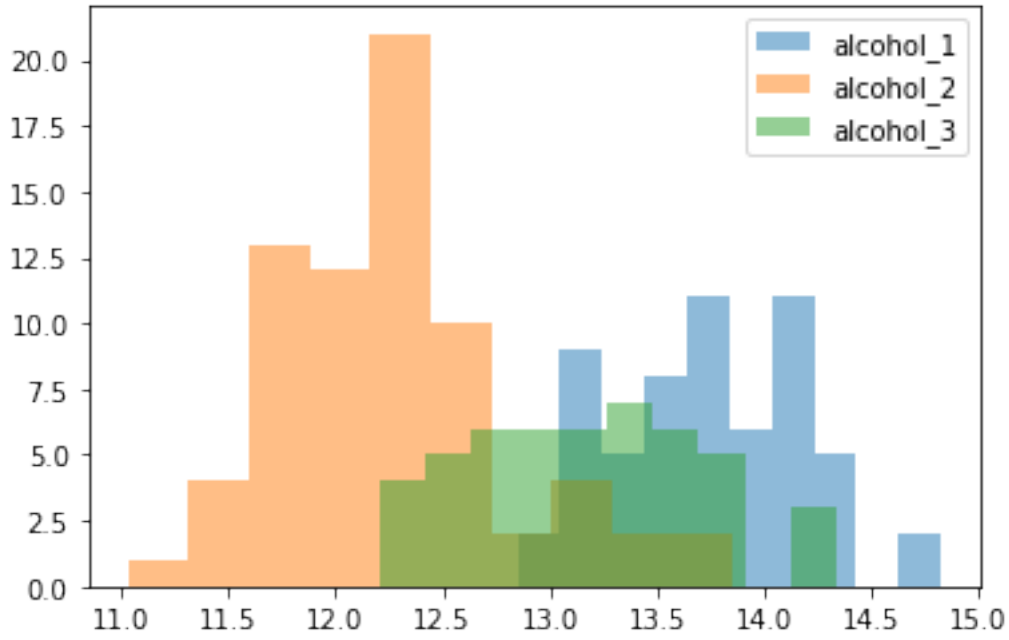
Solución:

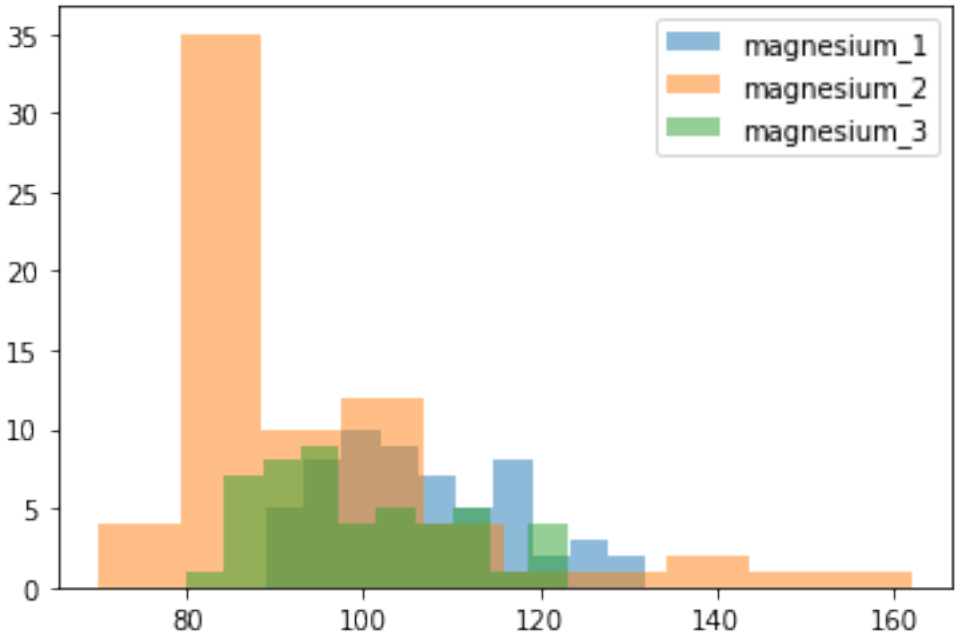
[17]:
```
feats_to_analysis = feats_to_explore.copy()
```

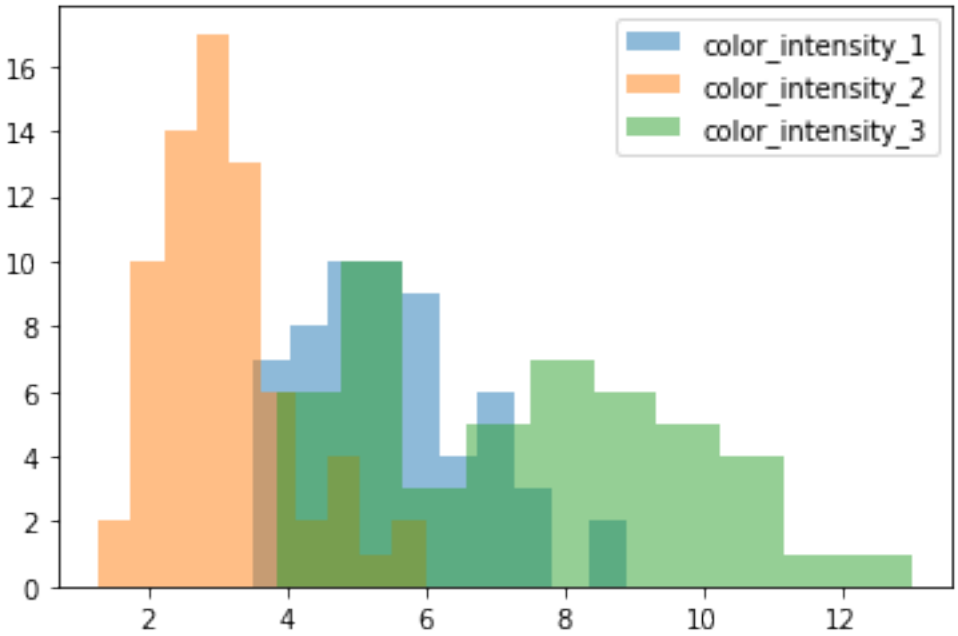
[18]:
```
feats_to_analysis.append('wine_class')
```

[19]:
```
wine_analysis = wine[feats_to_analysis]
```

[20]:
```
for col in wine_analysis:
    if col != 'wine_class':
        plt.hist(wine_analysis[wine_analysis['wine_class'] == 1][col], label =␣
↪col + '_1', alpha=0.5)
        plt.hist(wine_analysis[wine_analysis['wine_class'] == 2][col], label =␣
↪col + '_2', alpha=0.5)
        plt.hist(wine_analysis[wine_analysis['wine_class'] == 3][col], label =␣
↪col + '_3', alpha=0.5)
        plt.legend(loc='upper right')
        plt.show()
```







Análisis: Mirando los histogramas, ¿qué atributo parece tener más peso a la hora de clasificar un vino? ¿Cuál parece tener menos peso?

Solución:

Si observamos los histogramas obtenidos **el atributo que parece tener más peso a la hora de**

clasificar el vino es el alcohol, pues se puede ver que los 3 gráficos generados por cada uno de los valores de la variable Wine son los que más separados se encuentran unos de otros, con respecto a los 3 gráficos para las demás variables

Por otro lado, **el atributo que parece tener menos peso a la hora de clasificar el vino es el magnesium**, pues se puede ver que los 3 gráficos generados por cada uno de los valores de la variable Wine son los que más superpuestos se encuentran entre ellos, con respecto a los 3 gráficos generados para las demás variables

Ejercicio: Usando los histogramas anteriores, añadid una línea vertical indicando la media de cada uno de los histogramas (tres por gráfico). Pintad las líneas del mismo color que el histograma para que quede claro a cuál hacen referencia.

Añadid a la leyenda la clase de vino y la desviación estándar en cuestión.

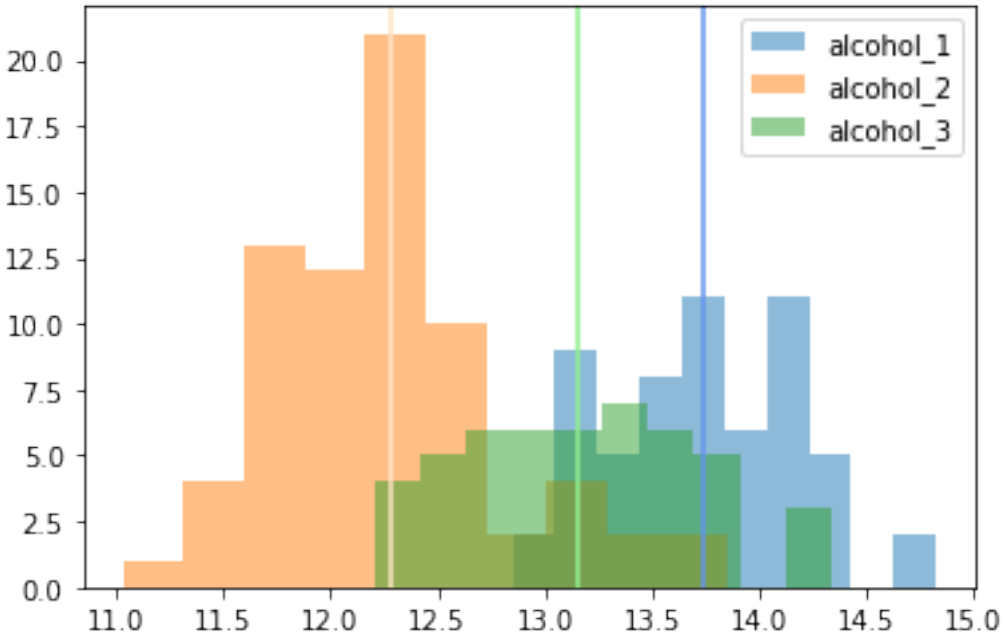
La finalidad es verificar numéricamente las diferencias identificadas anteriormente de forma visual.

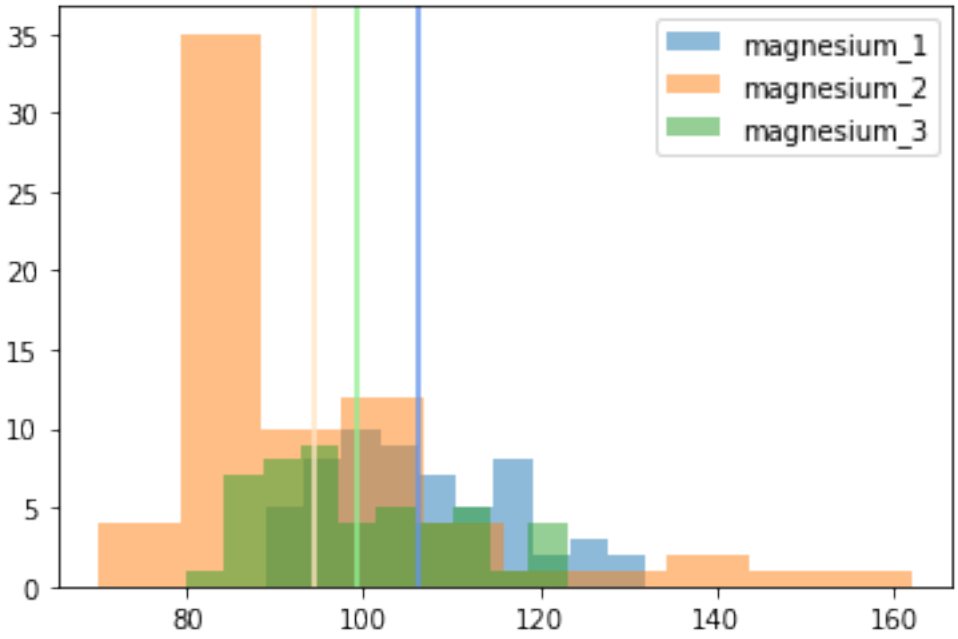
Sugerencia: podeis usar "axvline", de matplotlib axis, para las líneas verticales.

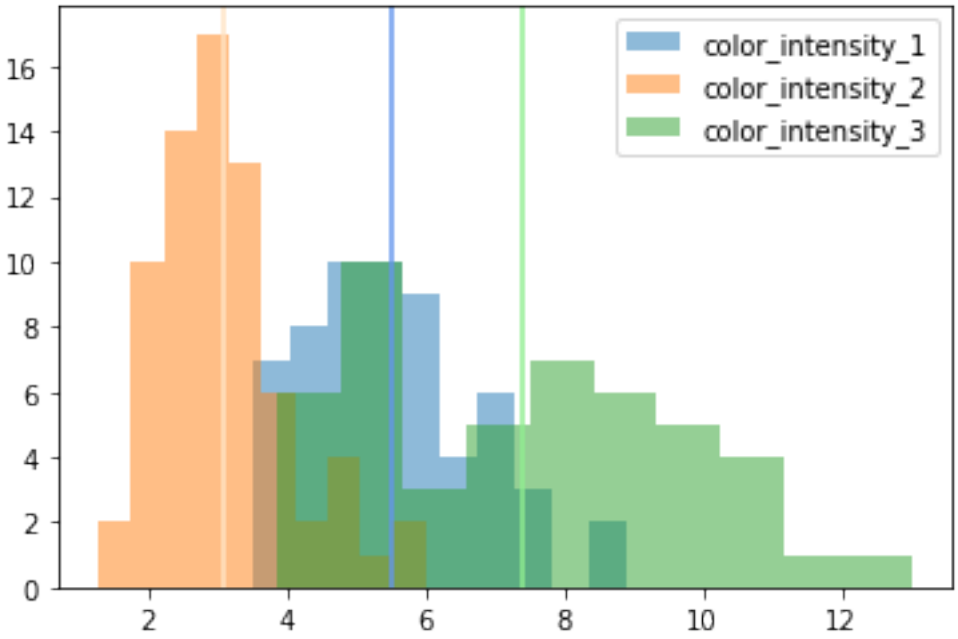
Solución:

[21]:
```
import statistics
```

[22]:
```
for col in wine_analysis:
    if col != 'wine_class':
        plt.hist(wine_analysis[wine_analysis['wine_class'] == 1][col], label =␣
↪col + '_1', alpha=0.5)
        plt.axvline(statistics.mean(wine_analysis[wine_analysis['wine_class']␣
↪== 1][col]), color= 'cornflowerblue')
        plt.hist(wine_analysis[wine_analysis['wine_class'] == 2][col], label =␣
↪col + '_2', alpha=0.5)
        plt.axvline(statistics.mean(wine_analysis[wine_analysis['wine_class']␣
↪== 2][col]), color= 'bisque')
        plt.hist(wine_analysis[wine_analysis['wine_class'] == 3][col], label =␣
↪col + '_3', alpha=0.5)
        plt.axvline(statistics.mean(wine_analysis[wine_analysis['wine_class']␣
↪== 3][col]), color= 'lightgreen')
        plt.legend(loc='upper right')
        plt.show()
```


alcohol_1
alcohol_2
alcohol_3
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0

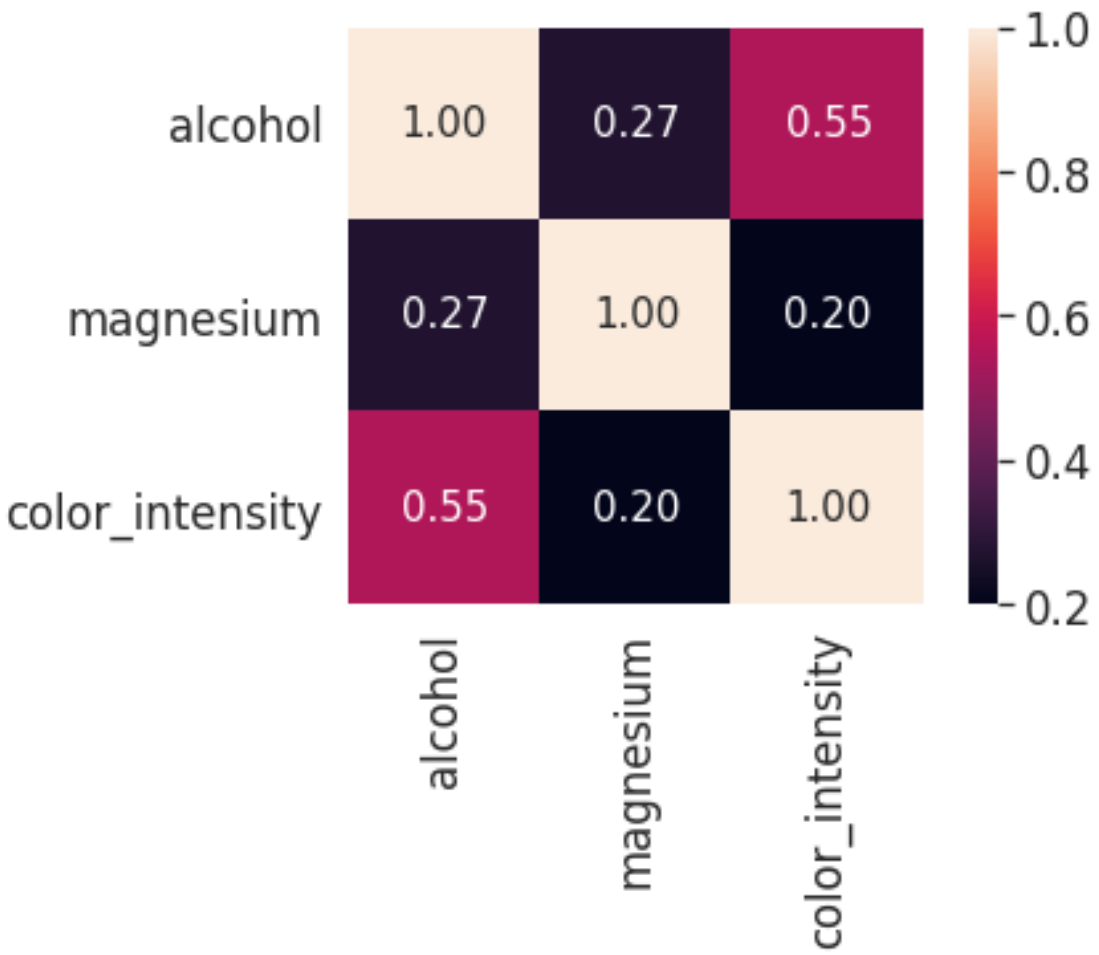

magnesium_1
magnesium_2
magnesium_3
35
30
25
20
15
10
5
0
80
100
120
140
160



Ejercicio: Calculad y mostrad la correlación entre las tres variables que estamos analizando.

Solución:

[23]:
```
cm = np.corrcoef(wine_analysis[feats_to_explore].values.T)
sns.set(font_scale=1.5)
hm = sns.heatmap(cm,
cbar=True,
annot=True,
square=True,
fmt='.2f',
annot_kws={'size': 15},
yticklabels=feats_to_explore,
xticklabels=feats_to_explore)
plt.show()
```



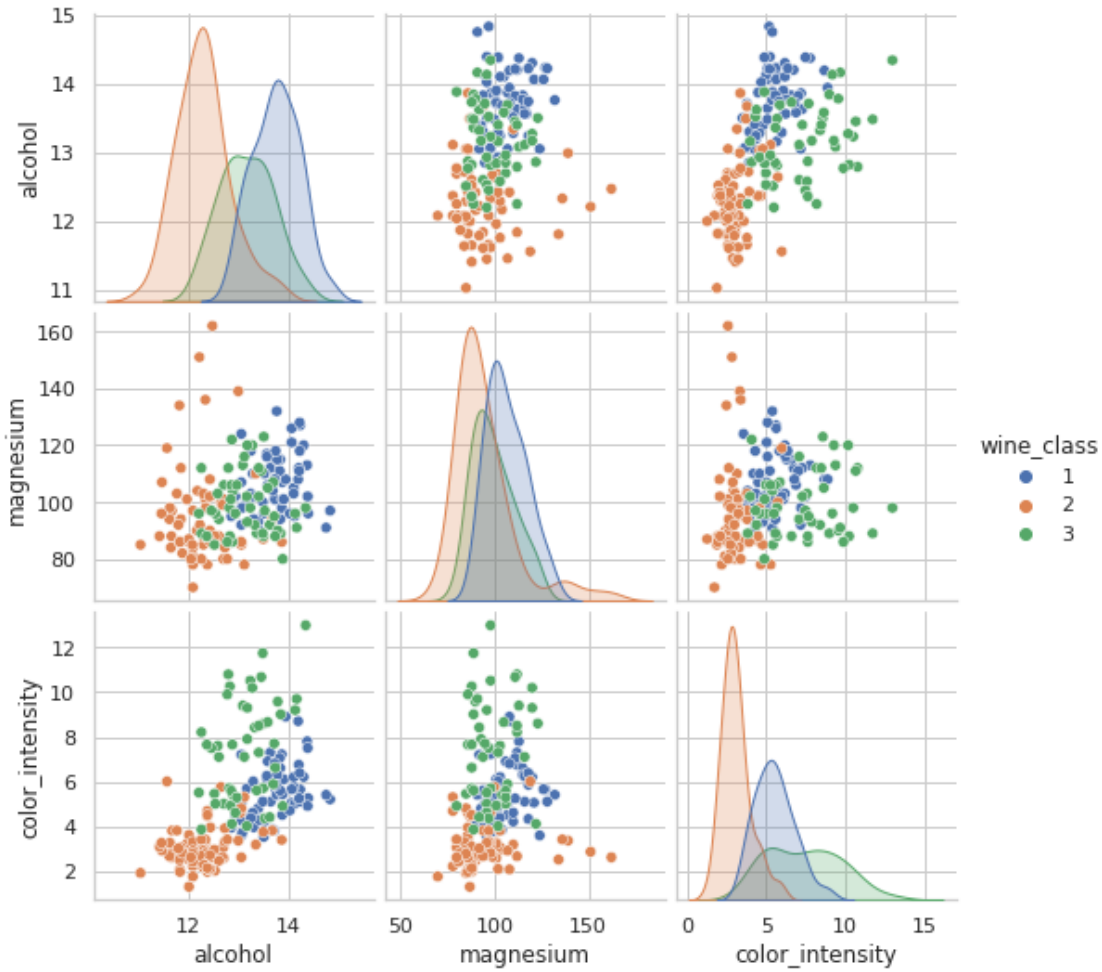
Ejercicio: Representad gráficamente las relaciones entre estas variables (scatterplots). Diferenciad con colores diferentes las diferentes clases.

La finalidad es poder observar y analizar las correlaciones de manera gráfica entre algunas de las variables.

Sugerencia: podéis usar la función "pairplot" de la librería 'seaborn' con el parámetro "hue".

Solución:

[24]:
```
sns.set(style='whitegrid', context='notebook')
sns.pairplot(wine_analysis, hue='wine_class')
plt.show()
```



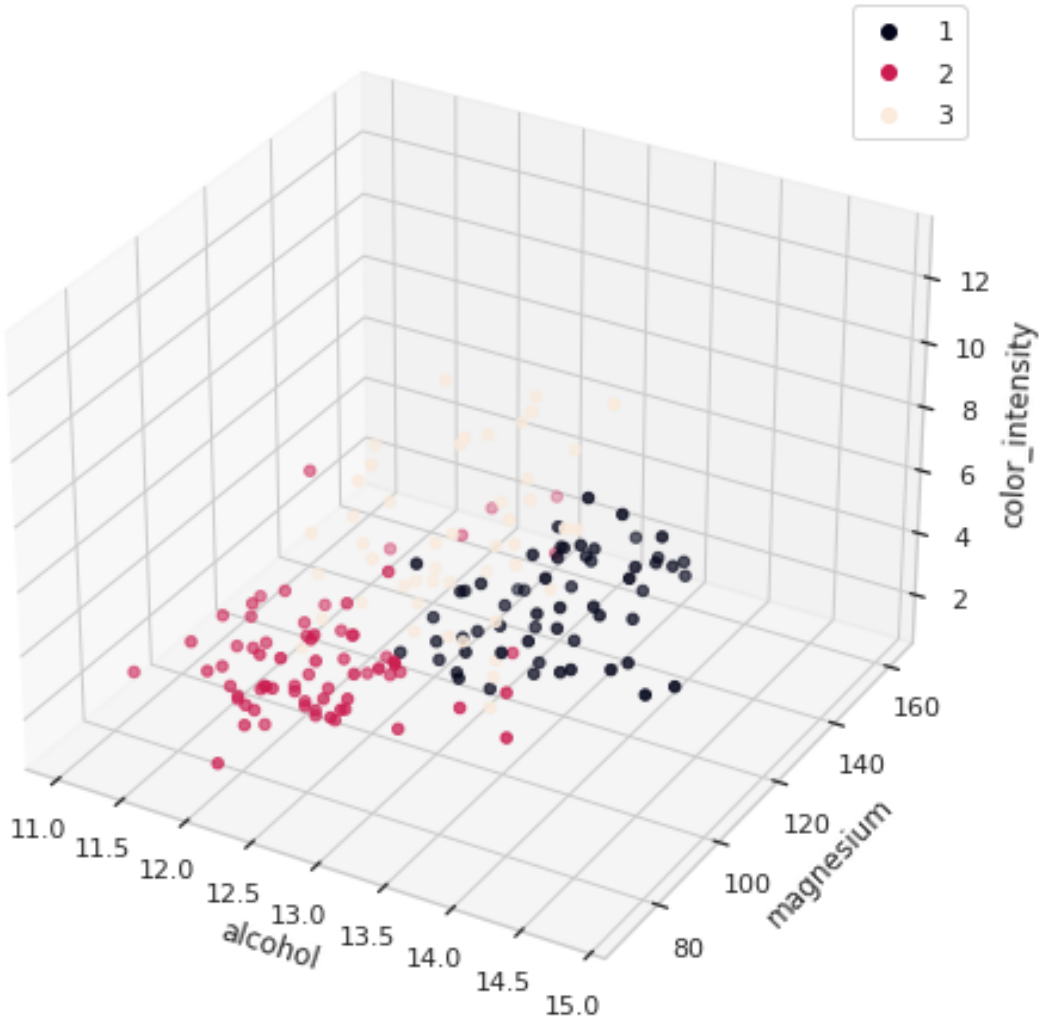
Ejercicio: Representad en 3D las tres variables. Poned nombres en los ejes y diferenciad con colores diferentes las diferentes clases de vino.

La finalidad es complementar los gráficos anteriores y poder observar qué variables discriminan mejor entre las tres clases de vino.

Solución:

[25]:
```
fig = plt.figure(figsize=(6,6))
ax = Axes3D(fig)

x = wine_analysis[feats_to_explore[0]]
y = wine_analysis[feats_to_explore[1]]
z = wine_analysis[feats_to_explore[2]]

ax.set_xlabel(feats_to_explore[0])
ax.set_ylabel(feats_to_explore[1])
ax.set_zlabel(feats_to_explore[2])
```

```
sc = ax.scatter(x, y, z, c=wine_analysis['wine_class'])
plt.legend(*sc.legend_elements())

plt.show()
```



Análisis: Observando las correlaciones, ¿qué variables son las que tienen una correlación más fuerte? ¿Cuadra el resultado numérico con los gráficos obtenidos?

Solución:

Observando las correlaciones, las variables que resultan tener una **correlación más fuerte** son: ***color_intensity y alcohol***. Lo cual, **cuadra con los gráficos obtenidos**, ya que **podemos ver a través del scatter plot y del gráfico 3D, que a medida que aumenta el valor de cualquiera de las dos variables, también va aumentando el de la otra**, aunque no de una forma muy notoria ya que a pesar de ser las dos variables que más se correlacionan entre sí, **dicha**

correlación tiene el valor de 0.55

4 3. Reducción de la dimensionalidad (3 puntos)

En este ejercicio se aplicarán métodos de reducción de la dimensionalidad al conjunto original de datos. El objetivo es reducir el conjunto de atributos a un nuevo conjunto con menos dimensiones. Así en vez de trabajar con 3 variables elegidas al azar, usaremos la información de todos los atributos.

Ejercicio: Aplicad el método de reducción de la dimensionalidad, Principal Component Analysis (PCA), para reducir a 2 dimensiones el dataset entero con todas las features. Generad un gráfico en 2D con el resultado del PCA usando colores diferentes para cada una de las clases de la respuesta (wine_class), con el objetivo de visualizar si es posible separar eficientemente las clases con este método.

NOTA: Tened cuidado y no incluyáis la variable objetivo "wine class" a la reducción de dimensionalidad. Queremos poder explicar la variable objetivo en funcion del resto de variables reducidas a dos dimensiones.

Sugerencia: No es necesario que programéis el algoritmo, podéis usar la implementación disponible en la librería de "scikit-learn".

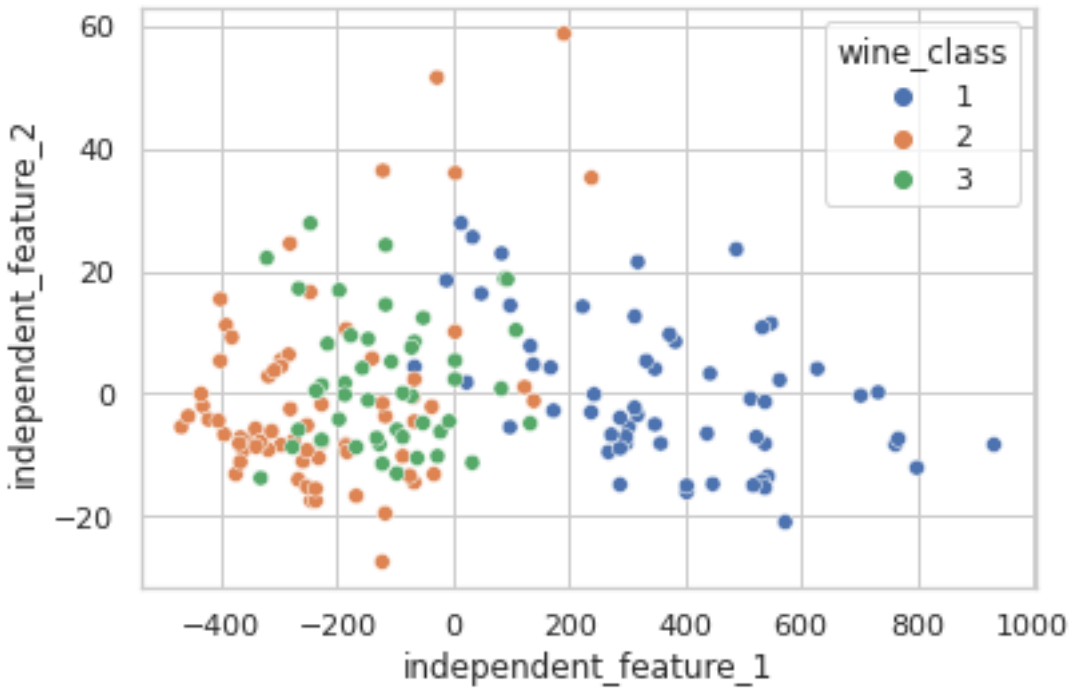
Solución:

[26]:
```
pca = PCA(n_components=2)
wine_pca = pca.fit(wine[wine.columns[1:]]).transform(wine[wine.columns[1:]])
```

[27]:
```
wine_pca = pd.DataFrame(data = wine_pca, columns=['independent_feature_1',␣
↪'independent_feature_2'])
wine_pca['wine_class'] = wine.wine_class
```

[28]:
```
sns.scatterplot(data=wine_pca, x="independent_feature_1",␣
↪y="independent_feature_2", hue="wine_class")
```

[28]:
```
<AxesSubplot:xlabel='independent_feature_1', ylabel='independent_feature_2'>
```



Ejercicio: Repetid la reducción de dimensionalidad, pero en este caso usando TSNE. Podéis encontrar más información sobre este algoritmo en el link:

https://distill.pub/2016/misread-tsne/

Igual que antes, generad un gráfico en 2D con el resultado del PCA usando colores diferentes para cada una de las clases de la respuesta (wine_class), con el objetivo de visualizar si es posible separar eficientemente las clases con este método.

Sugerencia: No es necesario que programéis el algoritmo, podéis usar la implementación disponible en la librería de "scikit-learn".

Sugerencia: A parte de especificar el número de componentes, probad a usar el parámetro "perplexity".

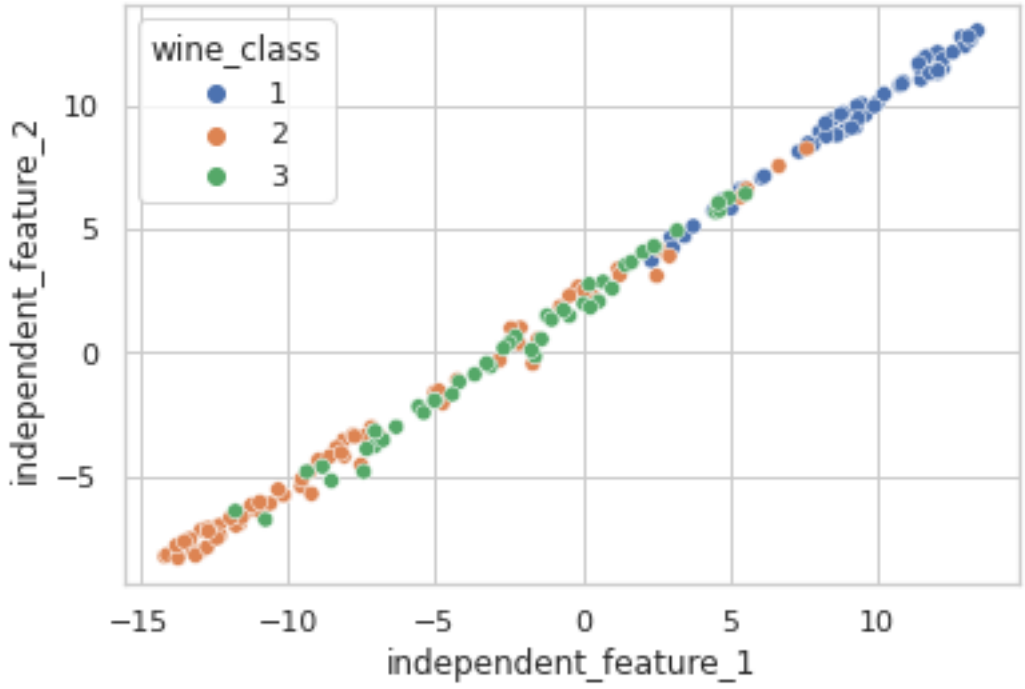
Solución:

[29]:
```
tsne = TSNE(n_components=2)
wine_tsne = tsne.fit_transform(wine[wine.columns[1:]])
```

[30]:
```
wine_tsne = pd.DataFrame(data = wine_tsne, columns=['independent_feature_1',␣
↪'independent_feature_2'])
wine_tsne['wine_class'] = wine.wine_class
```

[31]:
```
sns.scatterplot(data=wine_tsne, x="independent_feature_1",␣
↪y="independent_feature_2", hue="wine_class")
```

[31]: <AxesSubplot:xlabel='independent_feature_1', ylabel='independent_feature_2'>



[32]:
```
wine['wine_class'].value_counts()
```

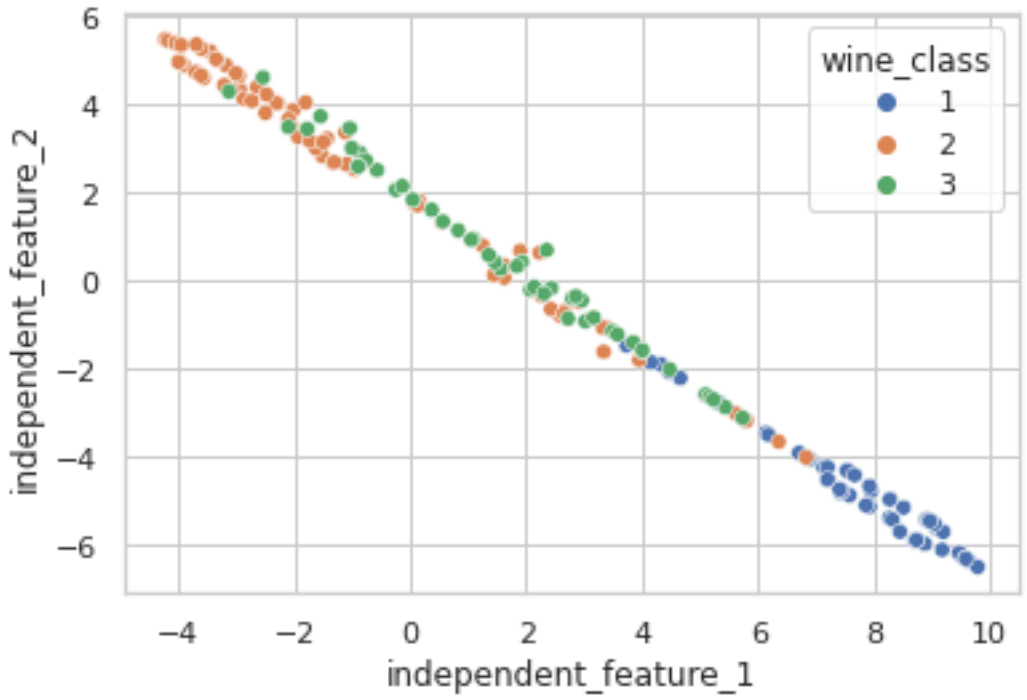
[32]:
```
2    71
1    59
3    48
Name: wine_class, dtype: int64
```

[33]:
```
tsne = TSNE(n_components=2, perplexity=50)
wine_tsne = tsne.fit_transform(wine[wine.columns[1:]])
```

[34]:
```
wine_tsne = pd.DataFrame(data = wine_tsne, columns=['independent_feature_1',␣
↪'independent_feature_2'])
wine_tsne['wine_class'] = wine.wine_class
```

[35]:
```
sns.scatterplot(data=wine_tsne, x="independent_feature_1",␣
↪y="independent_feature_2", hue="wine_class")
```

[35]: <AxesSubplot:xlabel='independent_feature_1', ylabel='independent_feature_2'>

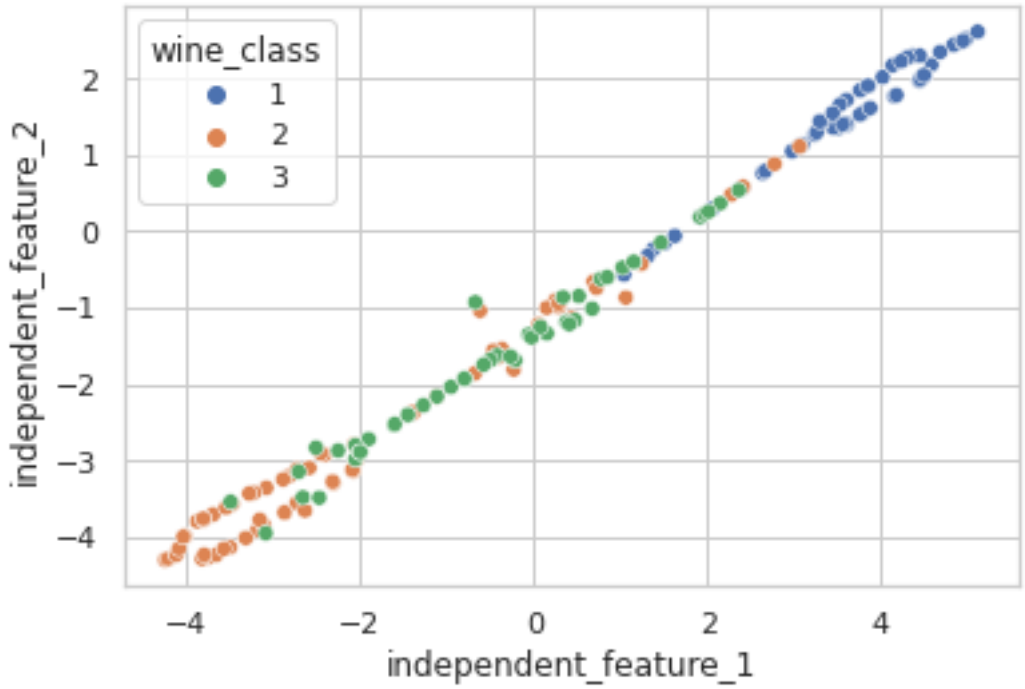


[36]:
```
tsne = TSNE(n_components=2, perplexity=71)
wine_tsne = tsne.fit_transform(wine[wine.columns[1:]])
```

[37]:
```
wine_tsne = pd.DataFrame(data = wine_tsne, columns=['independent_feature_1',␣
↪'independent_feature_2'])
wine_tsne['wine_class'] = wine.wine_class
```

[38]:
```
sns.scatterplot(data=wine_tsne, x="independent_feature_1",␣
↪y="independent_feature_2", hue="wine_class")
```

[38]: <AxesSubplot:xlabel='independent_feature_1', ylabel='independent_feature_2'>



Análisis: Observando los dos gráficos, ¿crees que ha funcionado bien la reducción de dimensionalidad? ¿Ha conseguido separar las clases correctamente? ¿Cuál de los dos métodos ha funcionado mejor?

¿Por qué obtenemos resultados tan diferentes?

Solución:

- En el caso de la reducción a través del algoritmo **PCA**, parece que **las clases 2 y 3 se solapan mucho entre sí**, esto puede ser porque tengan valores similares en las diferentes columnas en ambas clases.
- En el caso de la reducción a través del algoritmo **T-SNE**, parece que **consigue separar mejor las 3 clases presentes en los datos**, sobre todo **al aumentar el parámetro perplexity** hasta el valor 71 que es el conjunto de elementos que pertenecen a la clase más numerosa, que es la número 2. La razón por la cual he aumentado el valor y por la cual mejora, es por la forma en la que está implementada el algoritmo, ya que trata de agrupar los datos en diferentes grupos según la distancia de un punto a otro, y el parámetro **perplexity** representa el número máximo de elementos por grupo a la hora de dividirlos.

La razón por la que obtenemos resultados tan diferentes es por la forma de funcionamiento que tienen cada uno de los algoritmos, el algoritmo **PCA trata de maximizar la varianza y preservar las grandes distancias por pares de objetos**, lo que quiere decir que los elementos que resultan ser diferentes combinando los valores de todas sus variables, terminan estando muy separados, ya que trata de preservar la estructura global de los datos, mientras que **T-SNE trata de preservar solo pequeñas distancias por pares o similitudes locales**, lo que quiere decir que trata de preservar la estructura presente en los grupos que existen en los datos.

5 4. Entrenamiento y test (3 puntos)

En este último ejercicio se trata de aplicar un método de aprendizaje supervisado, concretamente el clasificador Decision Tree (un árbol de decisión), para predecir la clase a la que pertenece cada vino y evaluar la precisión obtenida con el modelo. Para eso usaremos:

```
- El conjunto de datos original con todos los atributos
- El conjunto de datos reducido a sólo 2 atributos con PCA
- El conjunto de datos reducido a sólo 2 atributos con TSNE
```

Ejercicio: Usando el conjunto de datos original:

- Dividid el dataset en train y test.
- Definid un modelo Decision Tree (fijando max_depth = 5 para mantener el modelo simple).
- Aplicad validación cruzada con el modelo definido y el dataset de train (con cv=5 ya es suficiente).
- Calculad la media y la desviación estándar de la validación cruzada.

Sugerencia: Para separar entre train y test podéis usar train_test_split de sklearn.

Sugerencia: Para entrenar un modelo decision tree podéis usar 'DecisionTreeClassifier' de sklearn.

Sugerencia: Para aplicar validación cruzada podéis usar 'cross_val_score' de sklearn.

Solución:

[39]:
```
X_train, X_test, y_train, y_test = train_test_split(wine[wine.columns[1:]],␣
↪wine['wine_class'], test_size=0.3, random_state=1)
```

[40]:
```
dt_clf = DecisionTreeClassifier(max_depth = 5)

cv_scores = cross_val_score(dt_clf, X_train, y_train, cv=5)
print("mean: {:.3f} (std: {:.3f})".format(cv_scores.mean(),
                                          cv_scores.std()),
                                          end="\n\n" )
```

```
mean: 0.887 (std: 0.046)
```

Ejercicio: Haced el fit con todo el conjunto de train, y generad el árbol de decisión correspondiente.

Sugerencia: Para dibujar árboles de decisión, ver este link:

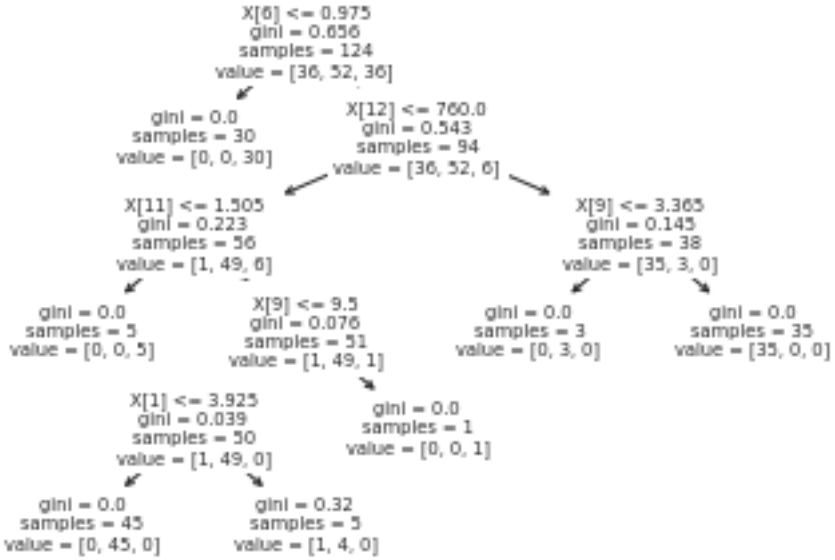
https://towardsdatascience.com/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-1c50b4aa68dc

Solución:

[41]:
```
dt_clf.fit(X_train, y_train)
```

[41]:
```
DecisionTreeClassifier(max_depth=5)
```

[42]:
```
tree.plot_tree(dt_clf);
```



Ejercicio: Repetid el mismo procedimiento que en el ejercicio anterior con el dataset reducido a 2 dimensiones con PCA.

Solución:

[43]:
```
X_train, X_test, y_train, y_test = train_test_split(wine_pca[wine_pca.columns[0:
↪2]], wine_pca['wine_class'], test_size=0.3, random_state=1)
```

[44]:
```
dt_clf_pca = DecisionTreeClassifier(max_depth = 5)

cv_scores = cross_val_score(dt_clf_pca, X_train, y_train, cv=5)
print("mean: {:.3f} (std: {:.3f})".format(cv_scores.mean(),
                                          cv_scores.std()),
                                          end="\n\n" )
```

```
mean: 0.677 (std: 0.082)
```

[45]:
```
dt_clf_pca.fit(X_train, y_train)
```

[45]:
```
DecisionTreeClassifier(max_depth=5)
```

[46]:
```
tree.plot_tree(dt_clf_pca);
```

Ejercicio: Repetid el mismo procedimiento que en el ejercicio anterior con el dataset reducido a 2 dimensiones con TSNE.

Solución:

[47]:
```
X_train, X_test, y_train, y_test = train_test_split(wine_tsne[wine_tsne.
↪columns[0:2]], wine_tsne['wine_class'], test_size=0.3, random_state=1)
```

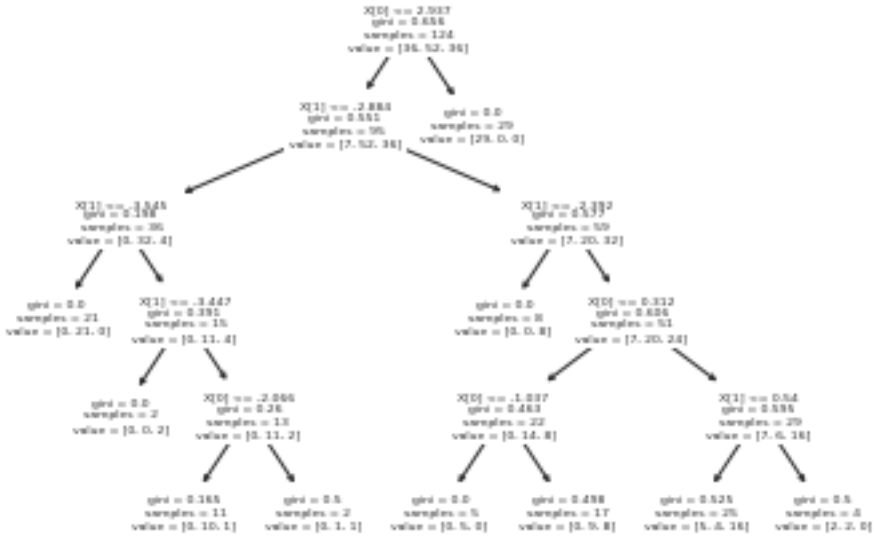
[48]:
```
dt_clf_tsne = DecisionTreeClassifier(max_depth = 5)

cv_scores = cross_val_score(dt_clf_tsne, X_train, y_train, cv=5)
print("mean: {:.3f} (std: {:.3f})".format(cv_scores.mean(),
                                          cv_scores.std()),
                                          end="\n\n" )
```

```
mean: 0.685 (std: 0.053)
```

[49]:
```
dt_clf_tsne.fit(X_train, y_train)
```

[49]:
```
DecisionTreeClassifier(max_depth=5)
```

[50]:
```
tree.plot_tree(dt_clf_tsne);
```



Análisis: ¿Con qué datos ha funcionado mejor? ¿Tiene sentido? ¿Cuadra con los resultados que hemos visto en el ejercicio 3?

Solución:

Los datos con los que ha funcionado mejor el modelo creado ha sido con el conjunto de datos total, lo cual **tiene sentido, porque finalmente**, a pesar del funcionamiento de los algoritmos utilizado, **al aplicar la reducción de la dimensionalidad**, aunque uno funcione mejor que otro, al aplicar la misma **se pierde una cierta cantidad de información.**

Si comparamos los resultados obtenidos por el dataset resultante de la reducción realizada con los algoritmos **PCA y T-SNE**, podemos observar que **el algoritmo T-SNE obtiene mejores resultados**, pero la diferencia es poca. La razón por la que obtiene mejores resultados es **porque el algoritmo T-SNE ha logrado separar mejor las diferentes clases presentes en los datos que el algoritmo PCA**, tal y como hemos visto en el ejercicio 3

Ejercicio: Con el mejor modelo que hayáis obtenido:

- Generad predicciones sobre el dataset de test.
- Calculad la precisión de las predicciones obtenidas y la matriz de confusión asociada.

Sugerencia: Para calcular la precisión y la matriz de confusión podéis usar las funciones dentro del módulo "metrics" de sklearn.

Solución:

[51]:
```
X_train, X_test, y_train, y_test = train_test_split(wine[wine.columns[1:]],␣
↪wine['wine_class'], test_size=0.3, random_state=1)
```

[52]:
```
scores = dt_clf.predict(X_test)
```

[53]:
```
wine['wine_class'].value_counts()
```

[53]:
```
2    71
1    59
3    48
Name: wine_class, dtype: int64
```

[54]:
```
def print_metrics(labels, scores):
    conf = confusion_matrix(labels, scores)
    print('                 Confusion matrix')
    print('                       Class 1          Class 2          Class 3')
    print('Actual Class 1        %6d' % conf[0,0] + '             %5d' %␣
↪conf[0,1] + '             %5d' % conf[0,2])
    print('Actual Class 2        %6d' % conf[1,0] + '             %5d' %␣
↪conf[1,1] + '             %5d' % conf[1,2])
    print('Actual Class 3        %6d' % conf[2,0] + '             %5d' %␣
↪conf[2,1] + '             %5d' % conf[2,2])

    print('')
    print('Accuracy        %0.2f' % accuracy_score(labels, scores))
    metrics = precision_recall_fscore_support(labels, scores)
    print(' ')
    print('           Class 1                 Class 2                Class 3')
    print('Num case   %0.2f' % metrics[3][0] + '                      %0.2f' %␣
↪metrics[3][1] + '                 %0.2f' % metrics[3][2] )
    print('Precision   %0.2f' % metrics[0][0] + '                       %0.2f' %␣
↪metrics[0][1] + '                  %0.2f' % metrics[0][2] )
    print('Recall      %0.2f' % metrics[1][0] + '                       %0.2f' %␣
↪metrics[1][1] + '                  %0.2f' % metrics[1][2] )
    print('F1          %0.2f' % metrics[2][0] + '                       %0.2f' %␣
↪metrics[2][1] + '                  %0.2f' % metrics[2][2] )
```

[55]:
```
print_metrics(y_test, scores)
```

```
                 Confusion matrix
                       Class 1          Class 2          Class 3
Actual Class 1             21                2                0
Actual Class 2              0               18                1
Actual Class 3              0                0               12

Accuracy        0.94

           Class 1                 Class 2                Class 3
Num case   23.00                    19.00                  12.00
Precision   1.00                     0.90                   0.92
Recall      0.91                     0.95                   1.00
F1          0.95                     0.92                   0.96
```