Julian Peña Reyes

Análisis de datos

TRABAJO FINAL ANALISIS DATOS

VER UN NUEVO VINO A QUE CALIDAD PERTENECE (cluster)***** 6 CLUSTERS de calidades
Hacer el análisis con todo lo que hemos visto.

Solución

Cargamos la librería necesaria para nuestro análisis y también cargamos nuestra base de datos.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.patches as mpatches
import seaborn as sb
import seaborn as sns #graficas y estadistica



from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

import os #sistema op


os.chdir('C:/Users/Julian/Documents/MMA/PYTHONLUZ')
cwd=os.getcwd() # asigan variable swd al directorio
xls_file = "Winequality.xls"
df = pd.read_csv(xls_file, header= 0, sep=';')
```

Observemos en el gráfico de abajo las variables que tenemos y nuestro “y” quality.

Llamamos la descripción estadística de nuestra base

```
print(df.describe())
```

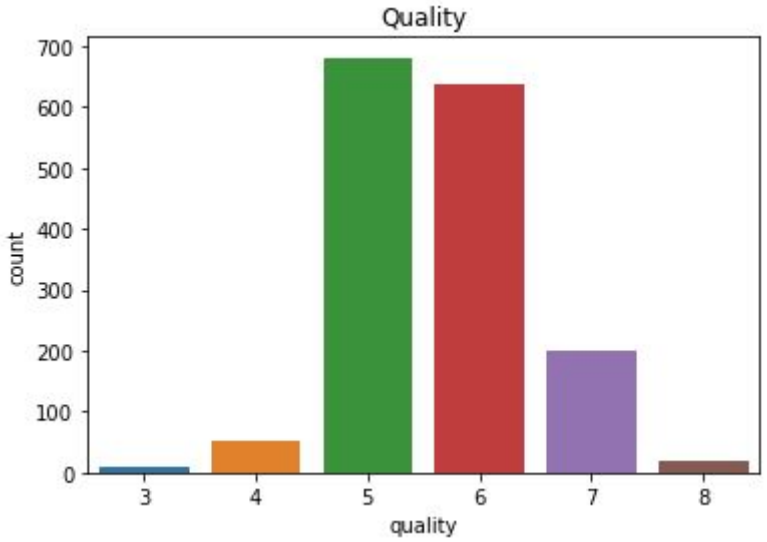
```
[20 rows x 12 columns]
```

	fixed acidity	volatile acidity	...	alcohol	quality
count	1599.000000	1599.000000	...	1599.000000	1599.000000
mean	8.319637	0.527821	...	10.422983	5.636023
std	1.741096	0.179060	...	1.065668	0.807569
min	4.600000	0.120000	...	8.400000	3.000000
25%	7.100000	0.390000	...	9.500000	5.000000
50%	7.900000	0.520000	...	10.200000	6.000000
75%	9.200000	0.640000	...	11.100000	6.000000
max	15.900000	1.580000	...	14.900000	8.000000

Contamos los datos por tipo de calidad.

```
sns.countplot(x= "quality", data = df)
plt.title("Quality")
plt.show()
```

Y obtenemos el siguiente gráfico.



Veamos aquí que tenemos n neighbors= 6 en los cuales podemos segmentar los datos dependiendo de su calidad.

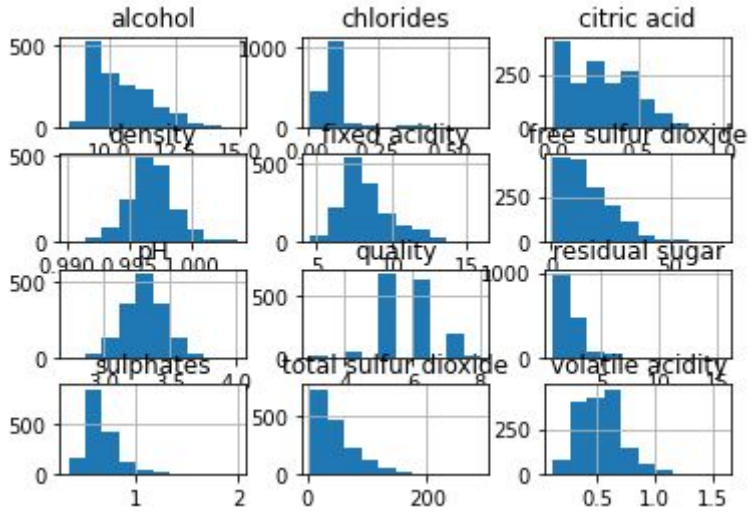
Veamos las frecuencias relativas de las calidades con

```
print(df["quality"].value_counts(normalize = True))
```

```
5    0.425891
6    0.398999
7    0.124453
4    0.033146
8    0.011257
3    0.006254
Name: quality, dtype: float64
```

Ahora grafiquemos las distribuciones de las variables

```
df.hist()
plt.show()
```



Con esto vemos que no todas tienen distribución normal y es posible que esto modifique el modelo.

Ahora creamos el modelo

Localizamos en x las que no sean quiality y en y quality

```
X = df.loc[:,df.columns != 'quiality']
y = df['quality'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
scaler = MinMaxScaler() # necesito escalamiento del dato max al min
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#k-Nearest Neighbor

n_neighbors = 6

knn = KNeighborsClassifier(n_neighbors)
knn.fit(X_train, y_train)
print('Accuracy training: {:.2f}'
     .format(knn.score(X_train, y_train)))
print('Accuracy test: {:.2f}'
     .format(knn.score(X_test, y_test)))
```

Tenemos como resultado buenas exactitudes

```
Accuracy training: 0.94
Accuracy test: 0.93
```

Ahora necesitamos identificar la precisión del modelo

```
pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

Vemos que el F1 score es muy bajito del 57% y esto debido a que las calidades 3 y 8 no tienen ninguna precisión, sin embargo una buena exactitud.

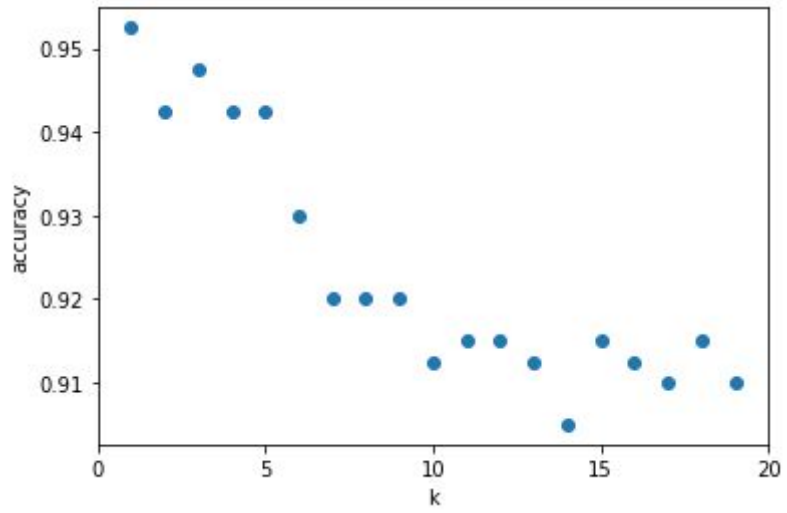
```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.88      0.50      0.64        14
           5       0.94      0.98      0.96       169
           6       0.95      0.97      0.96       170
           7       0.83      0.85      0.84        40
           8       0.00      0.00      0.00         5

    accuracy                           0.93       400
   macro avg       0.60      0.55      0.57       400
weighted avg       0.91      0.93      0.92       400
```

Con esto ya podemos graficar las exactitudes para ver el mejor valor de k

```
k_range = range(1, 20)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)
    scores.append(knn.score(X_test, y_test))
plt.figure()
plt.xlabel('k')
plt.ylabel('accuracy')
plt.scatter(k_range, scores)
plt.xticks([0,5,10,15,20])
```



Y vemos que el k donde hay mejor precisión es 1

Finalmente con esto allí ya podemos predecir sobre nuevas muestras con nuestro clasificador

```
clf = KNeighborsClassifier(n_neighbors, weights='distance')
clf.fit(X, y)
print(clf.predict([......]))
```