Las conclusiones finales sobre el ejercicio KNN, son las siguientes.

WINE

En el ejercicio del vino, con la optimización mediante la búsqueda de mejores parámetros, mediante un algoritmo donde se utiliza todas las variables posibles para el modelo de KNeighborsClassifier

Tras esto conseguimos lo siguiente.

El valor máx de porcentaje es:0.6433566433566433, con k=24, y w=distance

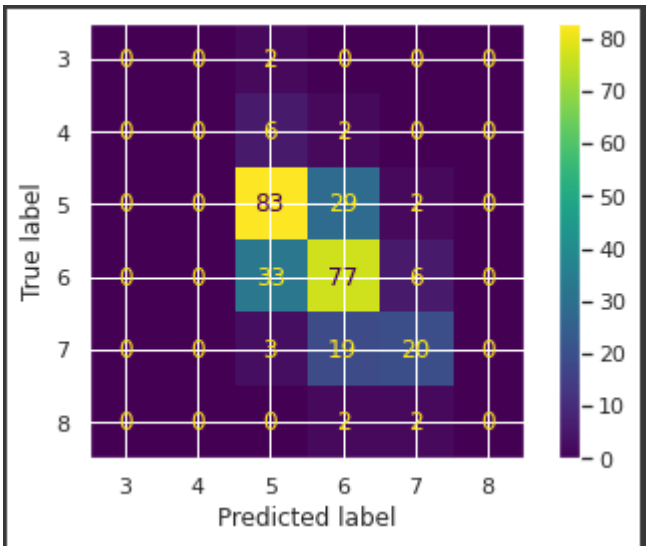
Una vez ya sabemos los mejores resultados, con estos parámetros, lo utilizamos para nuestro modelo, el cual hará que mejore hasta un 10%, comparado con la última vez, que se utilizó parámetros aleatorios.

Posteriormente con ese mismo modelo utilziando kf = KFold(n_splits=5)

Obtendremos una medida de bondad del modelo (accuracy score o)

```
Metrica del modelo 1.0
Metricas cross_validation [0.6744186  0.62790698 0.58479532 0.60818713 0.63157895]
Media de cross_validation 0.6253773969808242
Metrica en Test 0.6433566433566433
```

Finalmente haremos nuestra predicción con nuestro modelo optimizado, y como punto final, creamos nuestra matriz de confusión.



Donde vemos que aumenta nuestros valores.

HEART

Repitiendo el proceso anterior en en el caso del corazón, obtenemos una mínima empeoramiento,tanto en el porcentaje de clasificación, como en la matriz de confusión. Solo con una diferencia, anteriormente se usó el modelo de BernoulliNB(), mientras que ahora se utiliza KNN.

Como desventaja de usar este modelo, donde se ha visto mejor reflejado es en la matriz de confusión donde obtenemos una ligera incrementación a la hora de clasificar.

BIKES

A diferencia de los dos anteriores ejercicios de clasificación, este es un problema de regresión.

Una de las principales diferencias es a la hora de elegir los datos de entrenamiento y test, que se hace de la siguiente manera.

```
train_bike = df_filtrado.iloc[:501]
test_bike = df_filtrado.iloc[502:]

# train
x_train = train_bike[['temperature','humidity','windspeed']]
y_train = train_bike['count']

# test
x_test = test_bike[['temperature','humidity','windspeed']]
y_test = test_bike['count']
```

Donde en este caso elegimos los datos en base de su ubicación, finalmente asignamos las columnas.

Una vez creados los datos, utilizamos nuestro modelo, con una diferencia a los anteriores, ahora utilizamos el KNeighborsRegressor.

Procedemos a instanciar nuestro modelo, y a diferencia de buscar el score de acierto, obtenemos otros datos.

```python
# evaluacion
from sklearn.metrics import mean_squared_error
mean_squared_error = mean_squared_error(y_model,y_test)
print(f"error cuadrático medio: {mean_squared_error}")


from sklearn.metrics import mean_absolute_error
mean_absolute_error = mean_absolute_error(y_model,y_test)
print(f"error absoluto medio: {mean_absolute_error}")


```

```
error cuadrático medio: 95079.71379124143
error absoluto medio: 246.3994807607776
```

Ahora procedemos a representar gráficamente los valores predichos con los valores reales.

Como vemos nuestro modelo, en ciertas ocasiones si acierta casi plenamente, mientras que en otras se queda atrás, y tiene valores alejados, con un cierto margen considerable.

Comenzamos con nuestra optimización como anteriormente hemos realizado, con la diferencia de buscar el error absoluto menor.

```
[20]  print(f"El valor min de error absoluto es:{min(errores_absoluto)}, con k={k_min}, y w={w_min}")
```

```
El valor min de error absoluto es:243.08643326097726, con k=30, y w=distance
```

Una vez con estos, podemos hacer un modelo un poco más acertado.

Y como vemos, mejoramos ciertamente en algunos puntos, la predicción. Haciendo que nuestra predicción se acerque más a los datos.