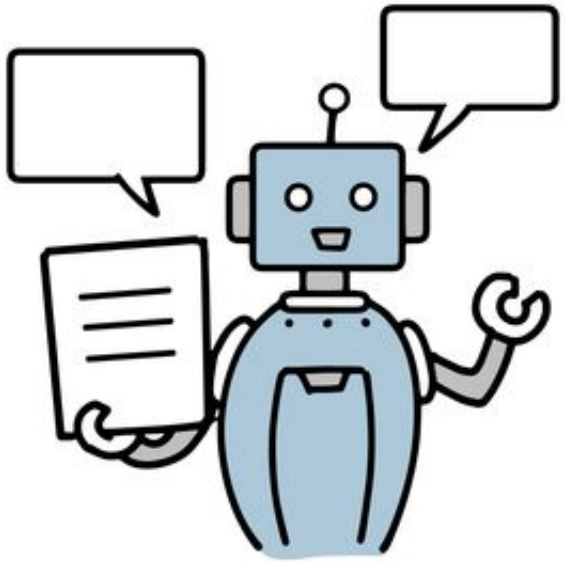
Clasificación de calidad de vino blanco

Juan David Pallares Pallares
Daniel Sebastián Badillo Neira

Contenido

1. **Motivación.**
2. **Objetivos.**
3. **Información del dataset.**
4. **Procesamiento.**
5. **ML.**
6. **DL.**
7. **DBSCAN.**
8. **K-means.**
9. **Resultados.**



1. Motivación

Este proyecto explora y compara modelos de aprendizaje supervisado (ML y DL) y no supervisado (K-Means y DBScan) aplicados a la clasificación de la calidad del vino. Se analiza su comportamiento, capacidad de generalización y efectividad, con el objetivo de entender sus fortalezas, limitaciones y aplicaciones prácticas en problemas reales de clasificación.

2. Objetivos

General

Desarrollar modelos de IA supervisados y no supervisados para predecir la calidad del vino, utilizando técnicas de Machine Learning (ML) y Deep Learning (DL), comparando su desempeño.

Específicos

Aplicar K-Means y DBScan para agrupar vinos según sus características y validar la coherencia de los clústeres.

3. Información del Dataset

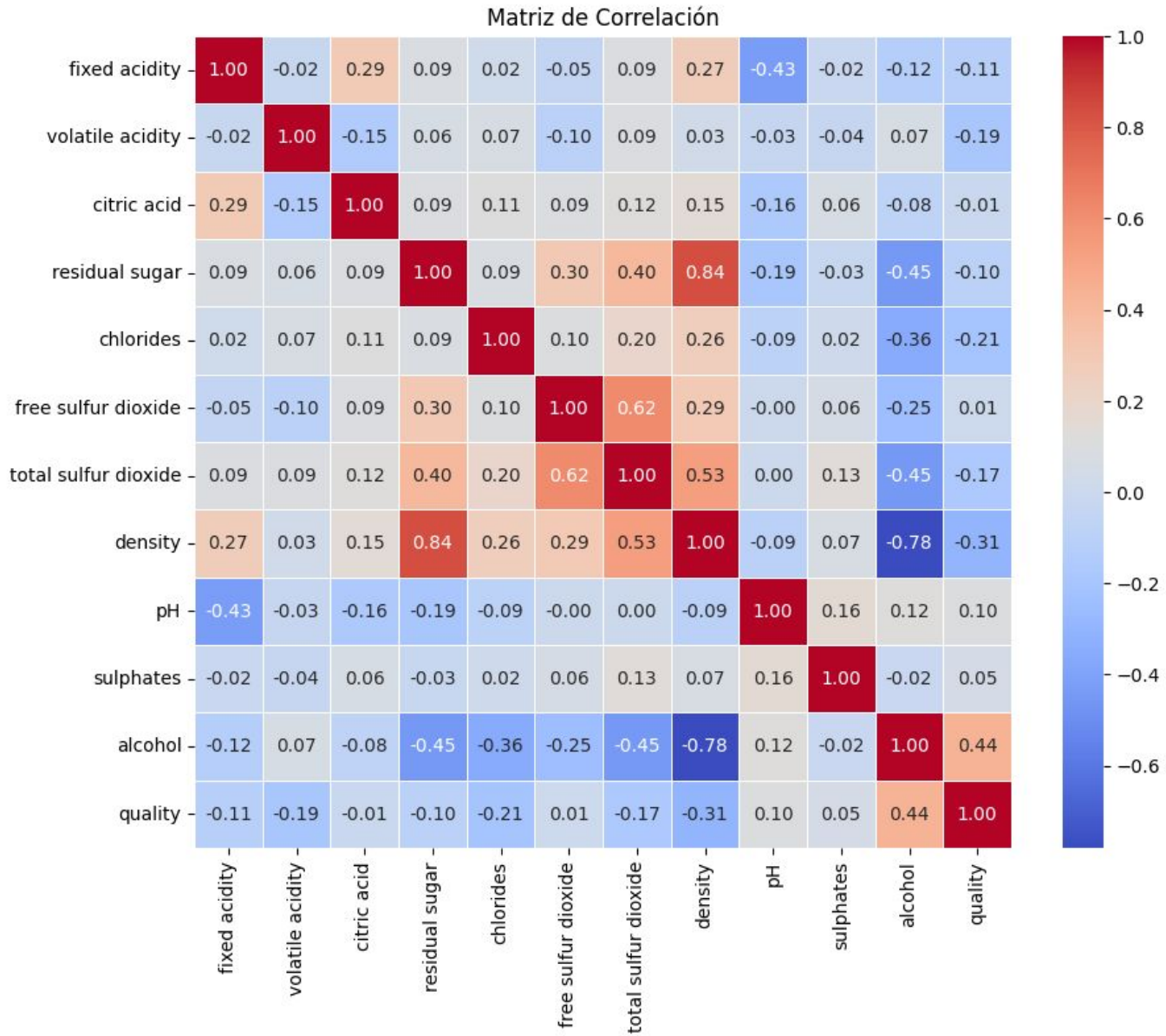
Se trata de un dataset que contiene información acerca de la calidad de múltiples tipos de vino blanco, evaluada con un valor discreto entre 0 y 9, según ciertas características del vino como nivel de alcohol, pH, densidad, azúcar residual, distintos niveles de acidez, entre otros.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

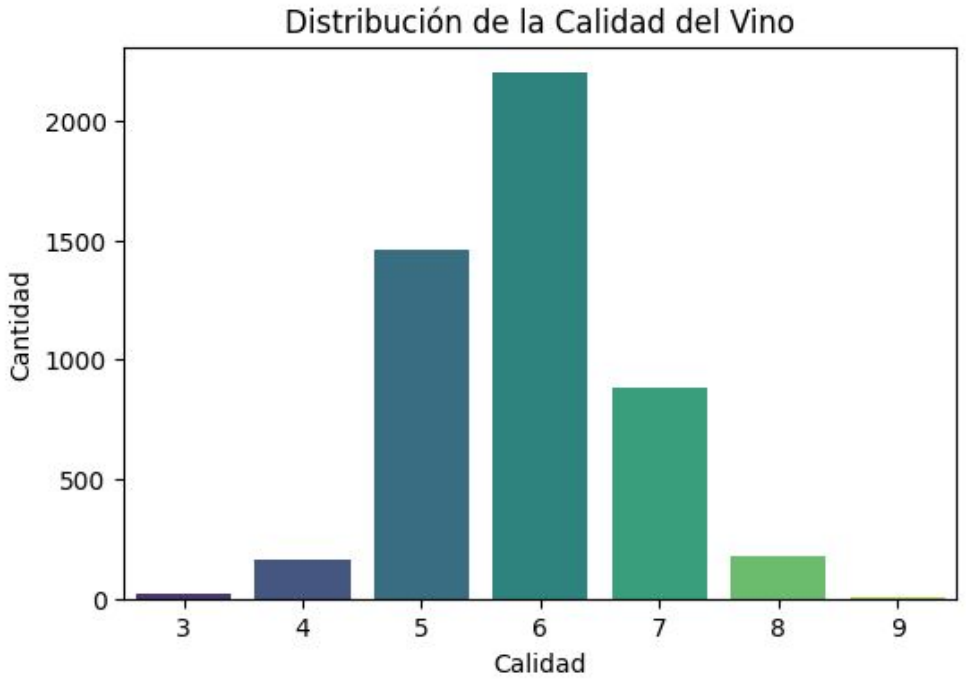
```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

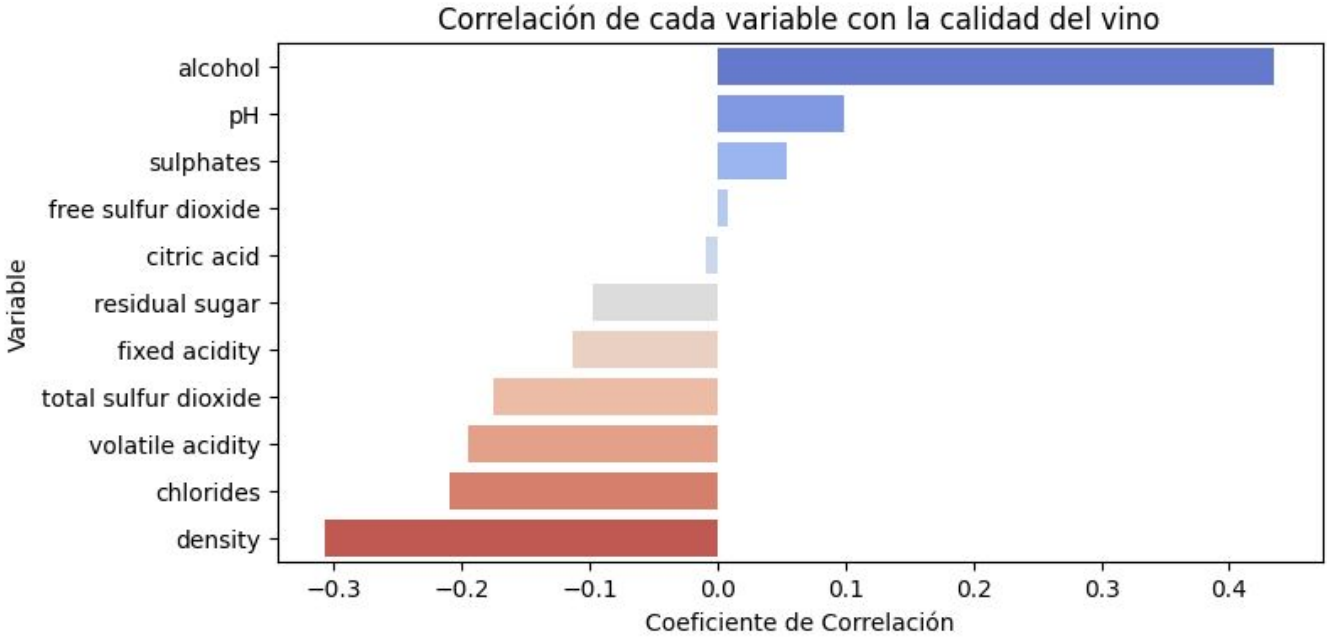
```
#@title Analisis General
print(df.describe().round(2))

       fixed acidity  volatile acidity  citric acid  residual sugar  \
count        4898.00           4898.00      4898.00         4898.00   
mean            6.85              0.28         0.33            6.39   
std             0.84              0.10         0.12            5.07   
min             3.80              0.08         0.00            0.60   
25%             6.30              0.21         0.27            1.70   
50%             6.80              0.26         0.32            5.20   
75%             7.30              0.32         0.39            9.90   
max            14.20              1.10         1.66           65.80   

       chlorides  free sulfur dioxide  total sulfur dioxide  density       pH  \
count    4898.00              4898.00               4898.00  4898.00  4898.00   
mean        0.05                35.31                138.36     0.99     3.19   
std         0.02                17.01                 42.50     0.00     0.15   
min         0.01                 2.00                  9.00     0.99     2.72   
25%         0.04                23.00                108.00     0.99     3.09   
50%         0.04                34.00                134.00     0.99     3.18   
75%         0.05                46.00                167.00     1.00     3.28   
max         0.35               289.00                440.00     1.04     3.82   

       sulphates  alcohol  quality  
count    4898.00  4898.00  4898.00  
mean        0.49    10.51     5.88  
std         0.11     1.23     0.89  
min         0.22     8.00     3.00  
25%         0.41     9.50     5.00  
50%         0.47    10.40     6.00  
75%         0.55    11.40     6.00  
max         1.08    14.20     9.00  
```

Matriz de correlación



4. Procesamiento





Características seleccionadas para el entrenamiento

Con ayuda de la gráfica anterior se seleccionaron aquellas features cuyo coeficiente de relación con la variable quality fue menor a -0.1 y mayor a 0.05. Por lo que estas son:

alcohol

pH

residual sugar

total sulfur dioxide

chlorides

sulphates

fixed acidity

volatile acidity

density

Clasificar vino según calidad

Para mejorar el preprocesamiento, se va a clasificar un vino según si es buen vino o mal vino aquellos que tengan más de 5 en calidad serán clasificados como un buen vino, aquellos con 5 o menos serán clasificados como mal vino.

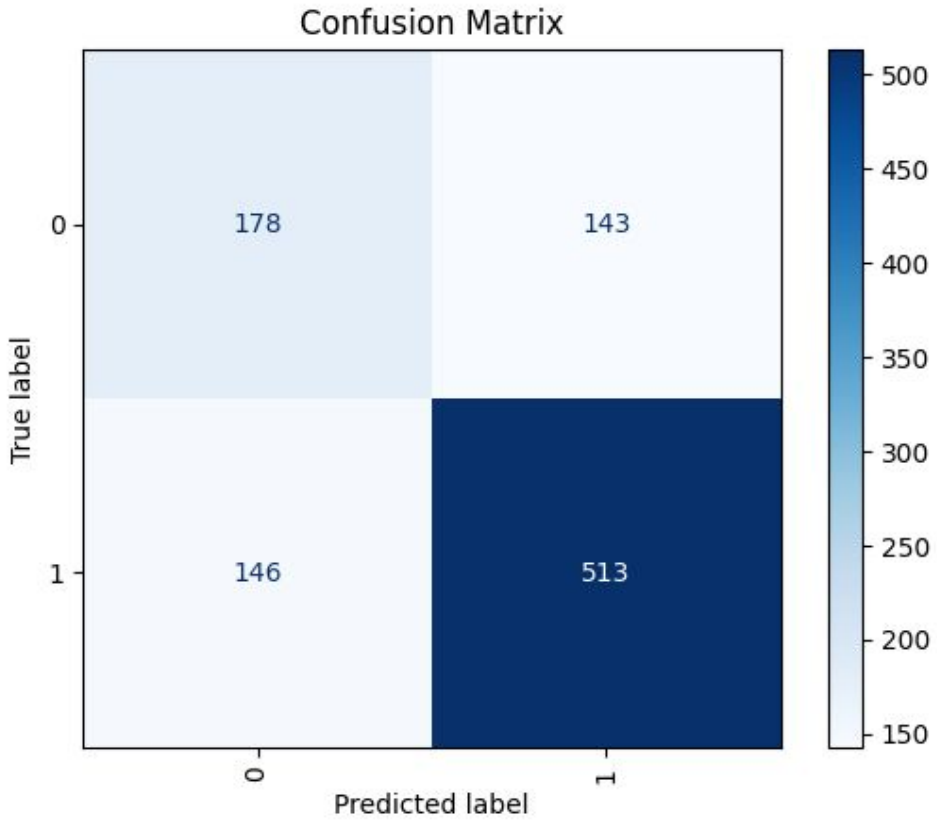
Clasificar vino según calidad

```
[ ] #@title Clasificar vino según calidad
    # Para mejorar el preprocesamiento, se va a clasisficar un vino según si es buen vino o mal vino
    # aquellos que tengan más de 5 en calidad serán clasificados como un buen vino, aquellos con 5 o menos serán clasificados como mal vino

    df['quality_binary'] = df['quality'].apply(lambda x: 1 if x > 5 else 0)
    df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	quality_binary
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	1
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	1
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1

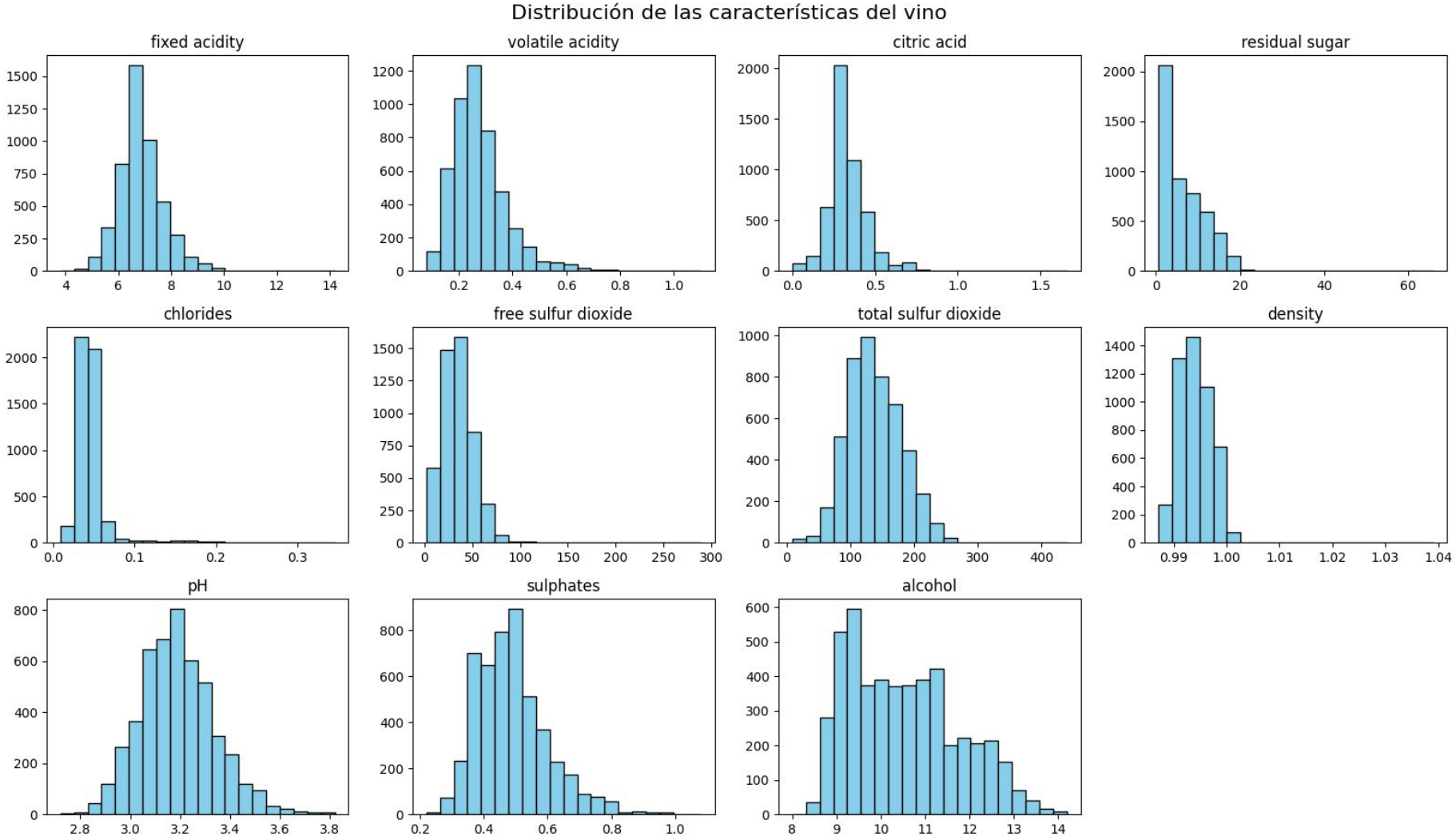
4. ML

Gaussian Naive Bayes




```
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Accuracy: 0.71
Cross Validation
Accuracy promedio: 0.33
¿Por qué?


Distribución de las características del vino
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol

Random Forest

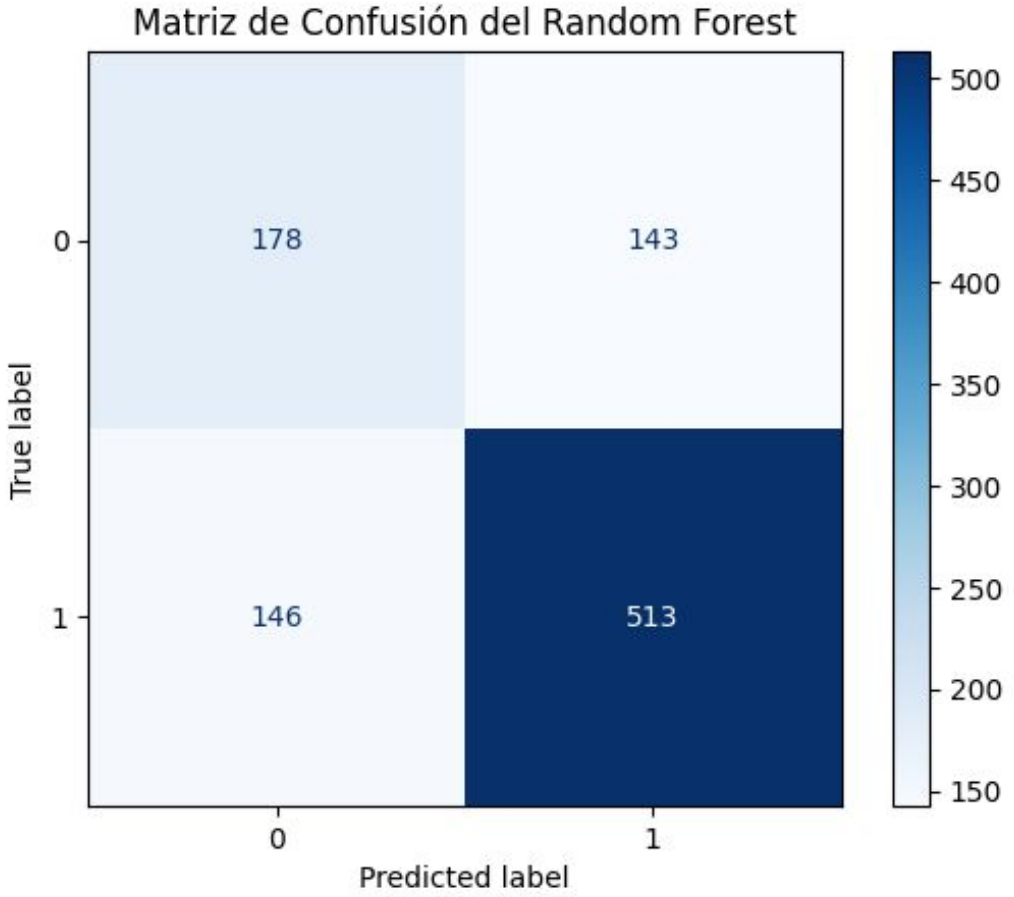
Modelo

```
#@title Modelo
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train,y_train)
accuracy_rf = accuracy_score(rf.predict(X_test), y_test)
print(f"Accuracy: {accuracy_rf:.2f}")
```

```
Accuracy: 0.82
```

Accuracy: 0.82
Cross Validation
Accuracy promedio: 0.84

	precision	recall	f1-score	support
0	0.55	0.55	0.55	321
1	0.78	0.78	0.78	659
accuracy			0.71	980
macro avg	0.67	0.67	0.67	980
weighted avg	0.71	0.71	0.71	980



SVM

```
svc = LinearSVC(random_state=42, max_iter=10000)
svc.fit(X_train_scaled, y_train)
y_pred = svc.predict(X_test_scaled)
accuracy_svc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy_svc:.2f}")
```

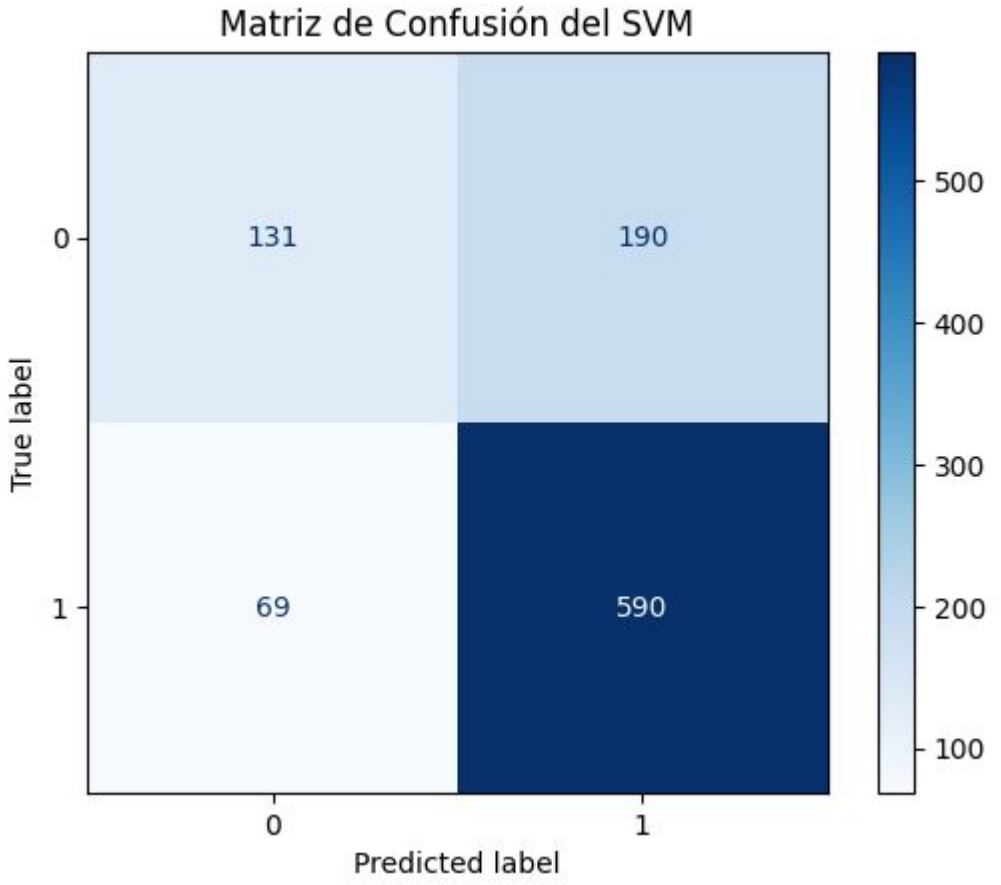
```
Accuracy: 0.74
```

Cross Validation

```
[23] #@title Cross Validation
     cross_val_score_svc = cross_val_score(svc, X, y, cv=KFold(10, shu
     print("accuracy %.3f (+/- %.5f)"%(np.mean(cross_val_score_svc), r
```

```
accuracy 0.752 (+/- 0.01607)
```

Accuracy: 0.74
Cross Validation
Accuracy promedio: 0.75



Comparación de Resultados

```
resultados = pd.DataFrame({
    'Modelo': ['Random Forest', 'Linear SVC', 'GaussianNB'],
    'Accuracy': [accuracy_rf, accuracy_svc, accuracy_NB],
    'Accuracy Cross Validation': [np.mean(cross_val_score_rf), np.mean(cross_val_score_svc), np.mean(cross_val_score_NB)]
})

resultados
```

	Modelo	Accuracy	Accuracy Cross Validation
0	Random Forest	0.827551	0.835235
1	Linear SVC	0.735714	0.751743
2	GaussianNB	0.705102	0.337924

4. DL

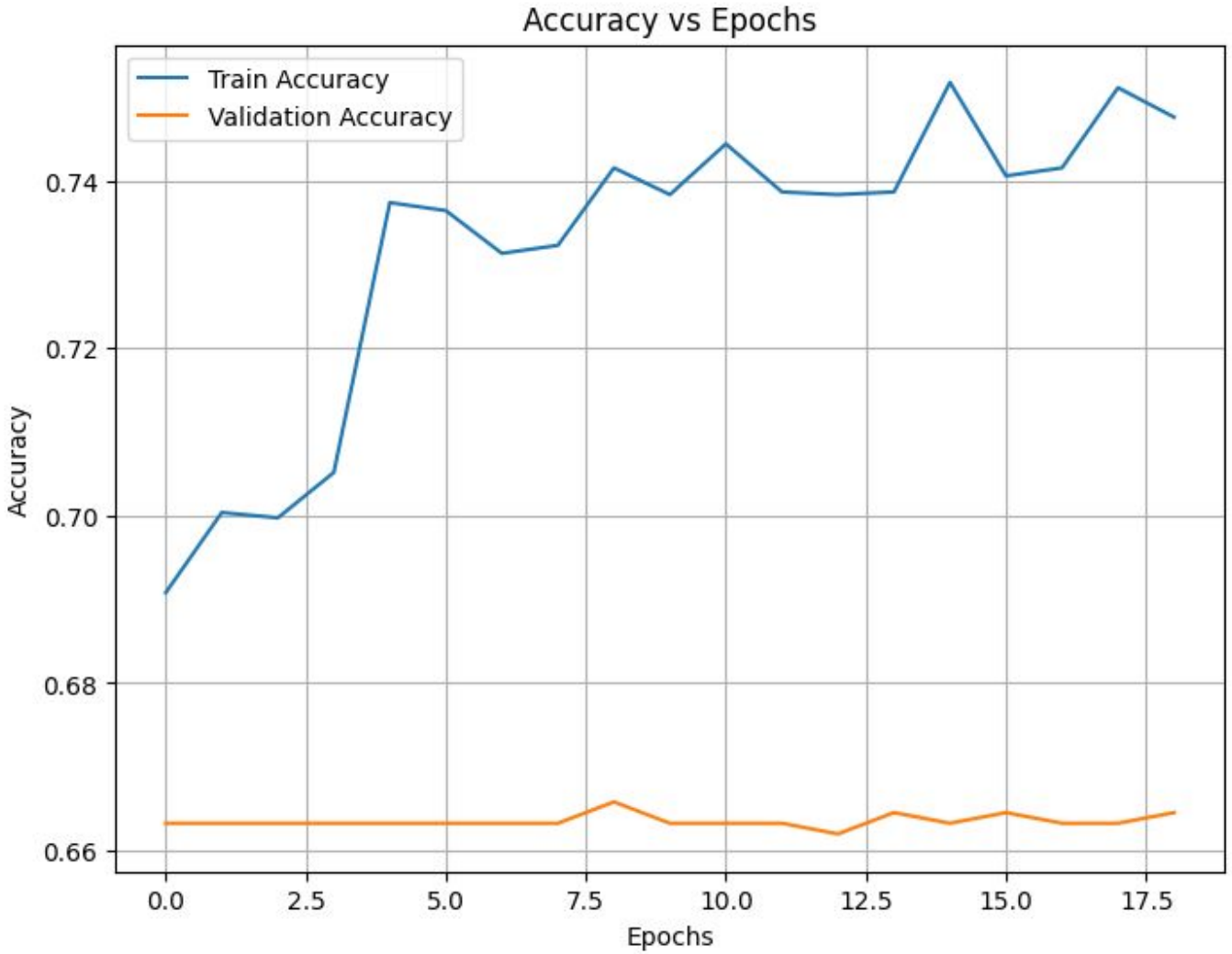
```
# Codificación de etiquetas
encoder = LabelEncoder()
y_train_encoded = encoder.fit_transform(y_train)
y_test_encoded = encoder.transform(y_test)

# Número de clases
num_classes = len(encoder.classes_)

# One-Hot Encoding
y_train_ohe = to_categorical(y_train_encoded, num_classes=num_classes)
y_test_ohe = to_categorical(y_test_encoded, num_classes=num_classes)

# Definición del modelo
model = Sequential([
    Flatten(input_shape=X_train_scaled[0].shape),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dense(128, activation='relu'),
    Dense(2, activation='softmax')  # Salida para multi-clase
])
```

```
# Compilación
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping
early_stopping = EarlyStopping(monitor='val_accuracy', patience=10, restore_best_weights=True)
```

```
# Entrenamiento
history = model.fit(X_train_scaled, y_train_ohe, epochs=100, batch_size=40, validation_split=0.2, callbacks=[early_stopping])
# Gráfica de la evolución de la accuracy
plt.figure(figsize=(8,6))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Epochs')
plt.legend()
plt.grid(True)
plt.show()
```

Test Accuracy: 0.67

Test Loss: 0.71


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Accuracy
0.74
0.72
0.70
0.68
0.66
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

Modelos No Supervisados

DBSCAN

```
df = pd.read_csv(file_path, delimiter=';')
df['quality_binary'] = df['quality'].apply(lambda x: 1 if x > 5 else 0)
y = df['quality_binary']
X = df.drop(['quality', 'quality_binary'], axis=1)

# Estandarizar datos
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)

# DBSCAN
dbscan = DBSCAN(eps=1.5, min_samples=5)
dbscan.fit(scaled_features)

print(np.unique(dbscan.labels_))
accuracy = accuracy_score(y, dbscan.labels_)
print(f"Accuracy: {accuracy:.2f}")
# print(classification_report(y, dbscan.labels_))
```

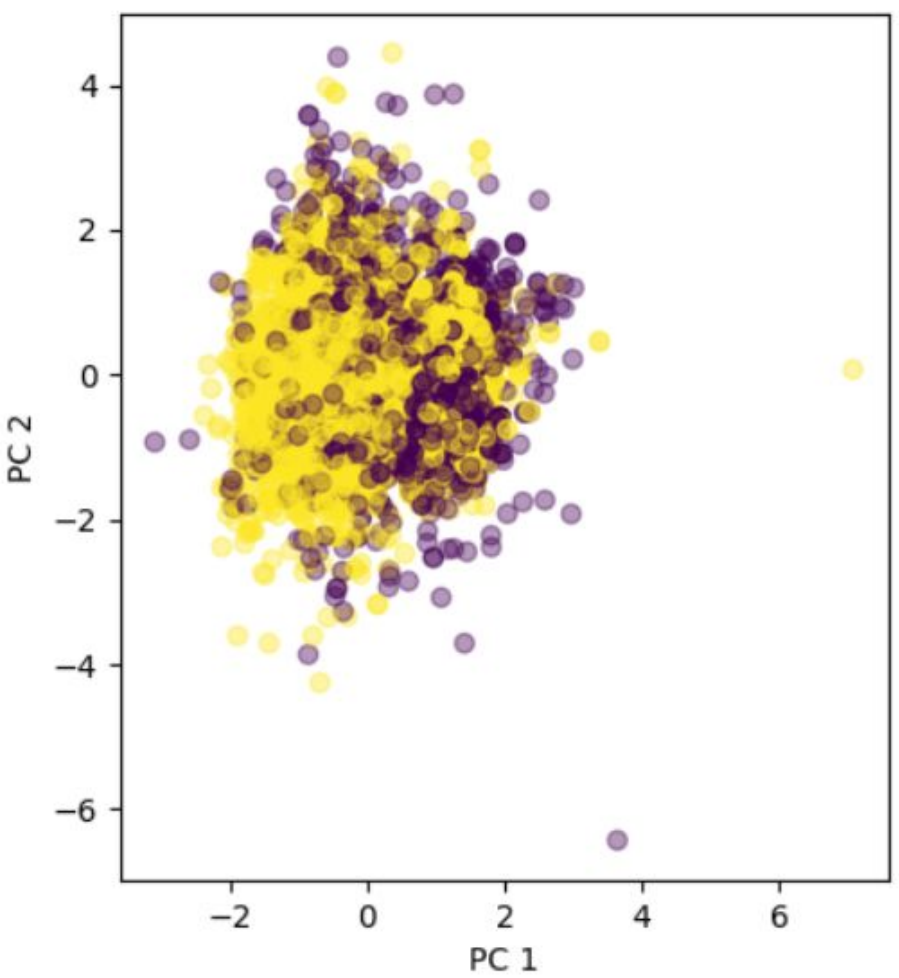
```
[-1  0  1  2  3  4  5  6  7  8  9 10 11 12 13]
Accuracy: 0.24
```

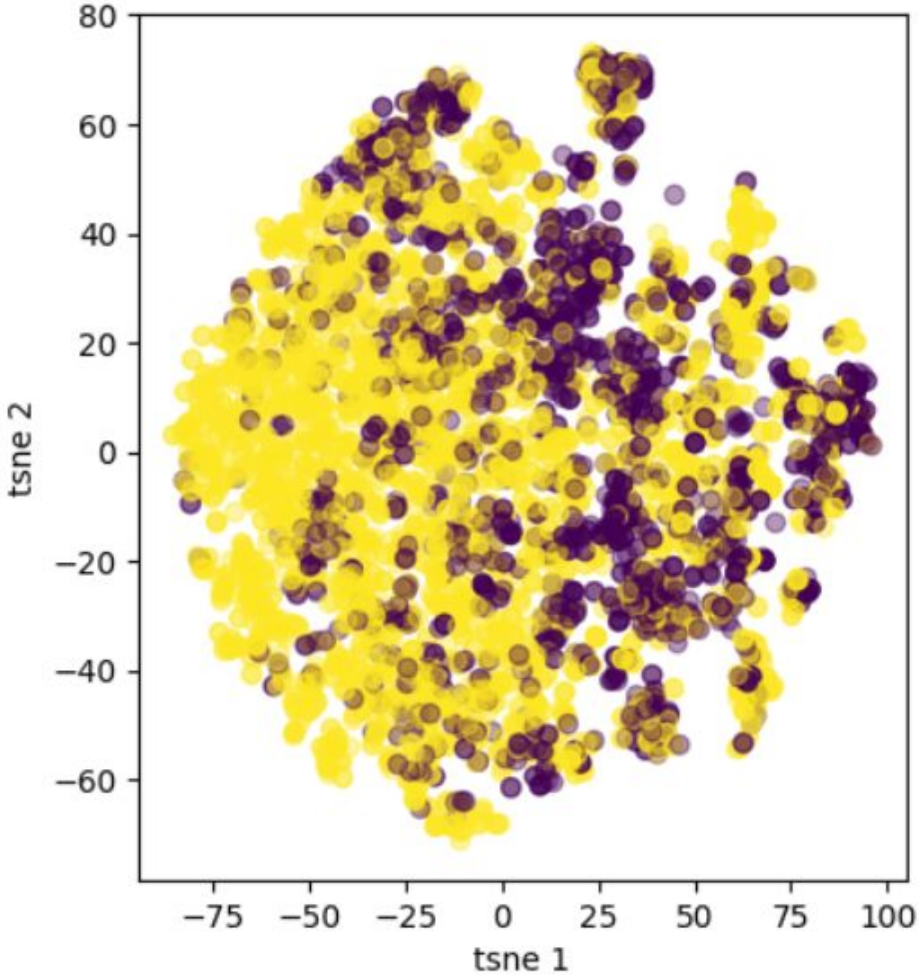
Accuracy: 0.24

Reducción de Dimensionalidad

```
#@title Reducción de dimensionalidad

pca_wine = PCA(n_components= 2, whiten=True)
X_pca = pca_wine.fit_transform(scaled_features)

tsne_wine = TSNE(n_components=2, learning_rate='auto')
X_tsne = tsne_wine.fit_transform(scaled_features)
```

Reducción de Dimensionalidad





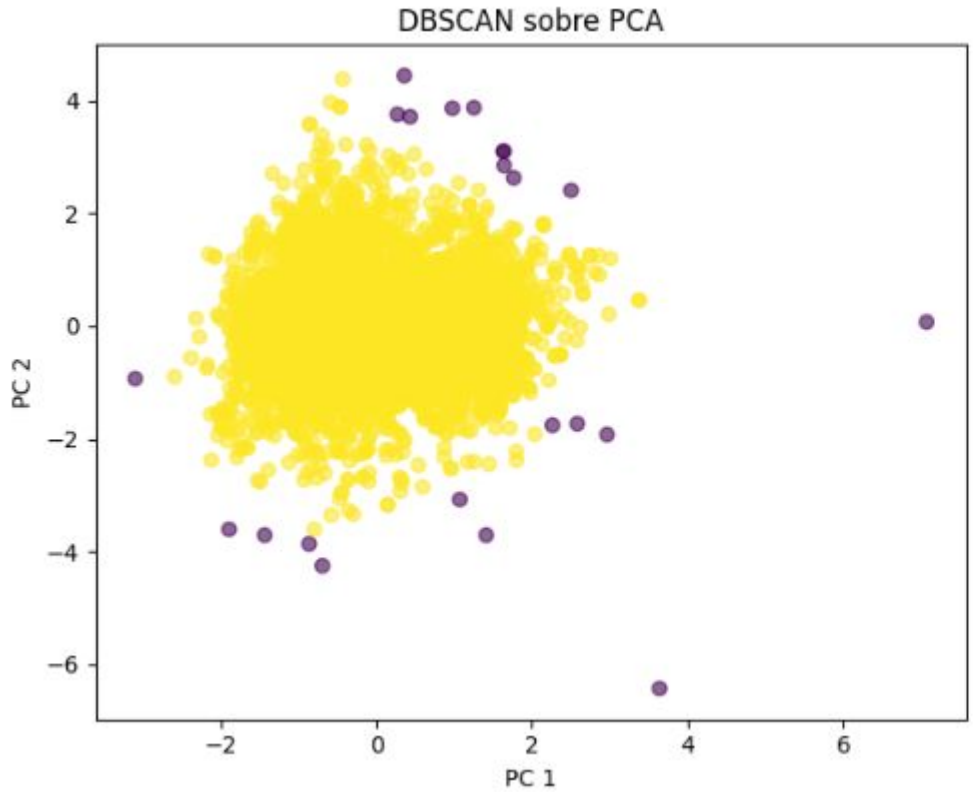
DBSCAN para PCA y T-SNE

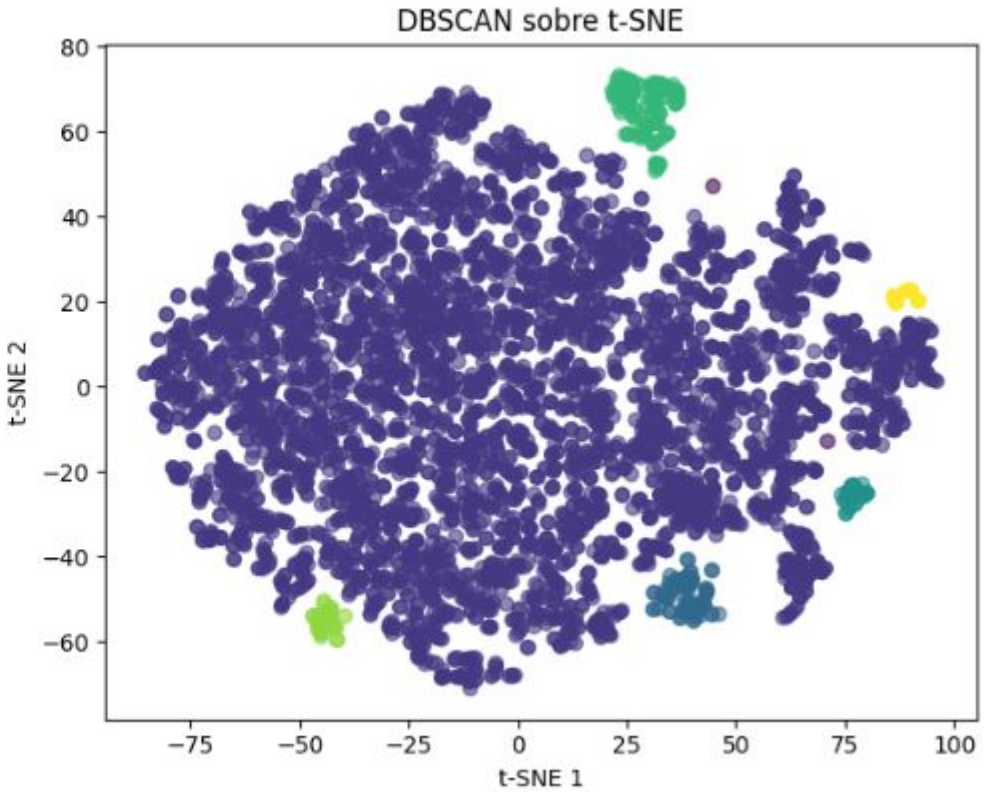
DBSCAN sobre PCA y TSNE

```python
#@title DBSCAN sobre PCA y TSNE
from collections import Counter

# DBSCAN sobre PCA
dbscan_pca = DBSCAN(eps=0.5, min_samples=5)  # Puedes ajustar eps
labels_pca = dbscan_pca.fit_predict(X_pca)
print("PCA Clusters:", np.unique(labels_pca))
print("PCA Label Count:", Counter(labels_pca))

# DBSCAN sobre t-SNE
dbscan_tsne = DBSCAN(eps=5, min_samples=5)  # t-SNE suele requerir mayor eps
labels_tsne = dbscan_tsne.fit_predict(X_tsne)
print("t-SNE Clusters:", np.unique(labels_tsne))
print("t-SNE Label Count:", Counter(labels_tsne))
```

```
PCA Clusters: [-1  0]
PCA Label Count: Counter({np.int64(0): 4876, np.int64(-1): 22})
t-SNE Clusters: [-1  0  1  2  3  4  5]
t-SNE Label Count: Counter({np.int64(0): 4619, np.int64(3): 122, np.int64(1): 76, np.int64(2): 37, np.int64(4): 30, np.int64(5): 12, np.int64(-1): 2})
```

DBSCAN para PCA y T-SNE





Resultado

```
#@title PCA DBSCAN Report
accuracy = accuracy_score(y, dbscan_pca.labels_)
print(accuracy)
print(classification_report(y, dbscan_pca.labels_))
```

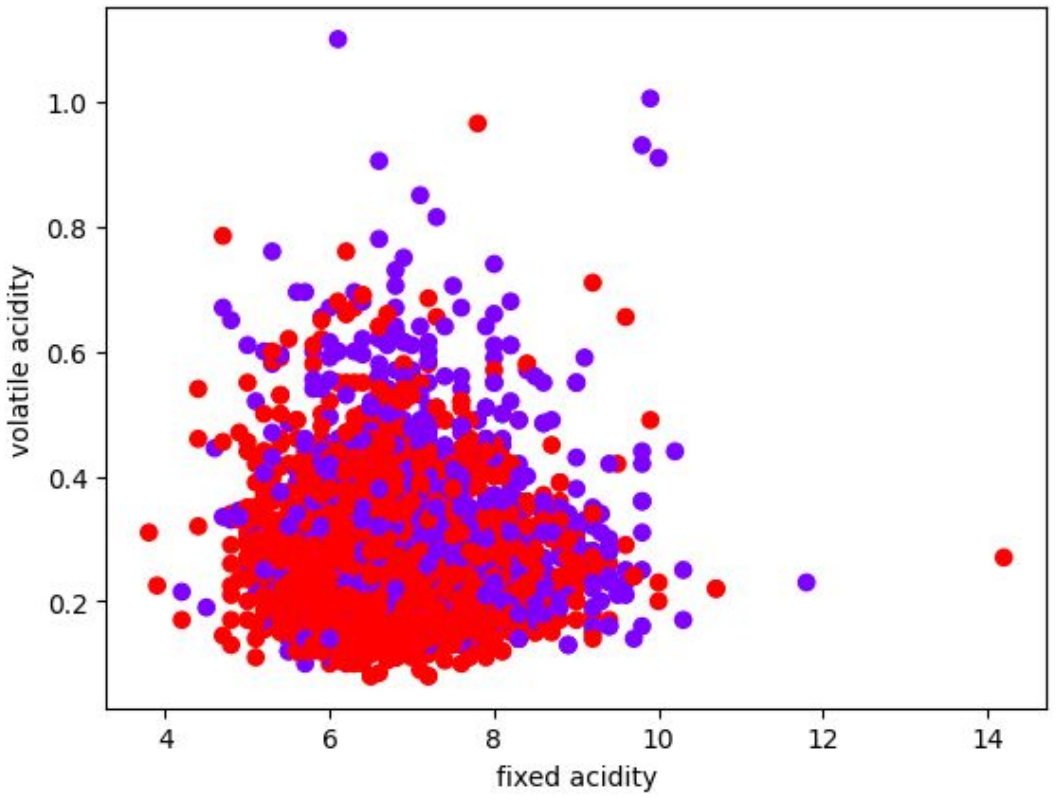
```
0.3319722335647203
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.33      0.99      0.50      1640
           1       0.00      0.00      0.00      3258

    accuracy                           0.33      4898
   macro avg       0.11      0.33      0.17      4898
weighted avg       0.11      0.33      0.17      4898
```

```
#@title TSNE DBSCAN Report
accuracy = accuracy_score(y, dbscan_tsne.labels_)
print(accuracy)
print(classification_report(y, dbscan_tsne.labels_))
```

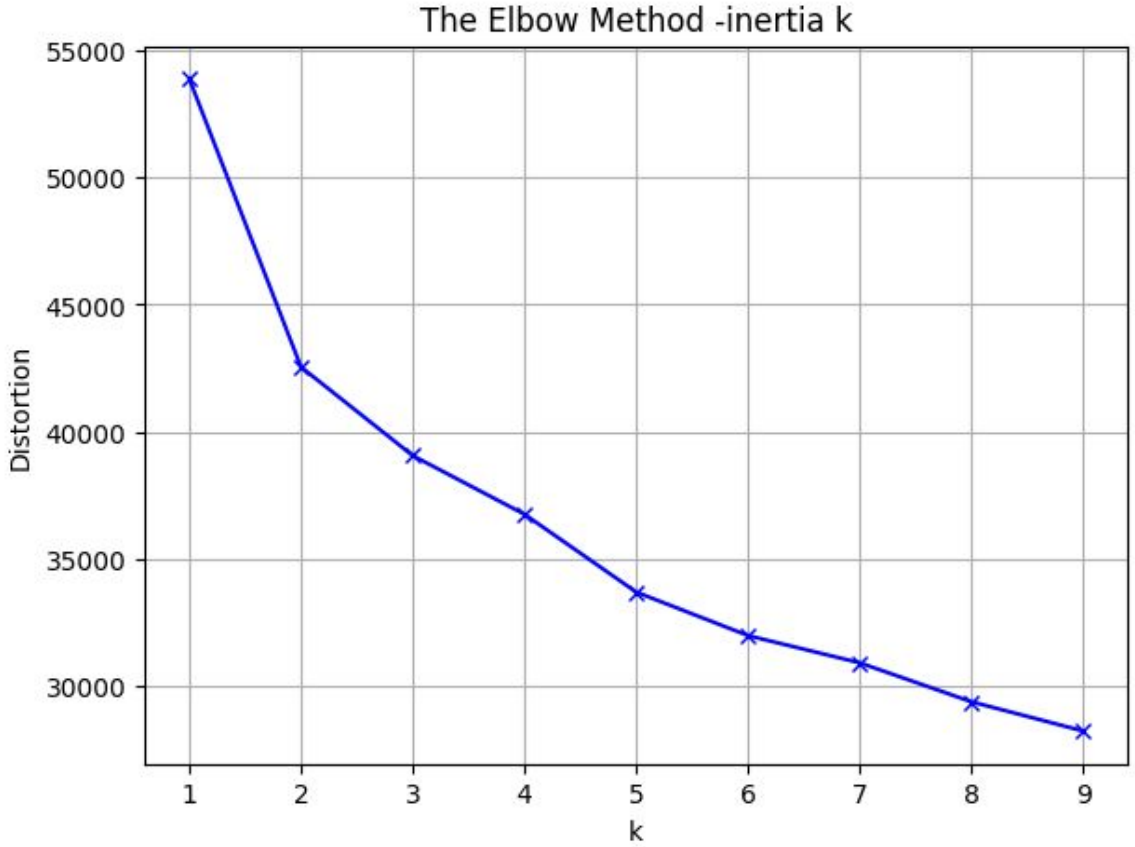
```
0.31318905675786035
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.33      0.92      0.48      1640
           1       0.41      0.01      0.02      3258
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0

    accuracy                           0.31      4898
   macro avg       0.10      0.13      0.07      4898
weighted avg       0.38      0.31      0.17      4898
```

KMeans

```
plt.scatter(X.iloc[:,0], X.iloc[:,1], c=y, cmap='rainbow');
y = df['quality_binary']
X = df.drop(['quality', 'quality_binary'], axis=1)
# Estandarizar datos
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

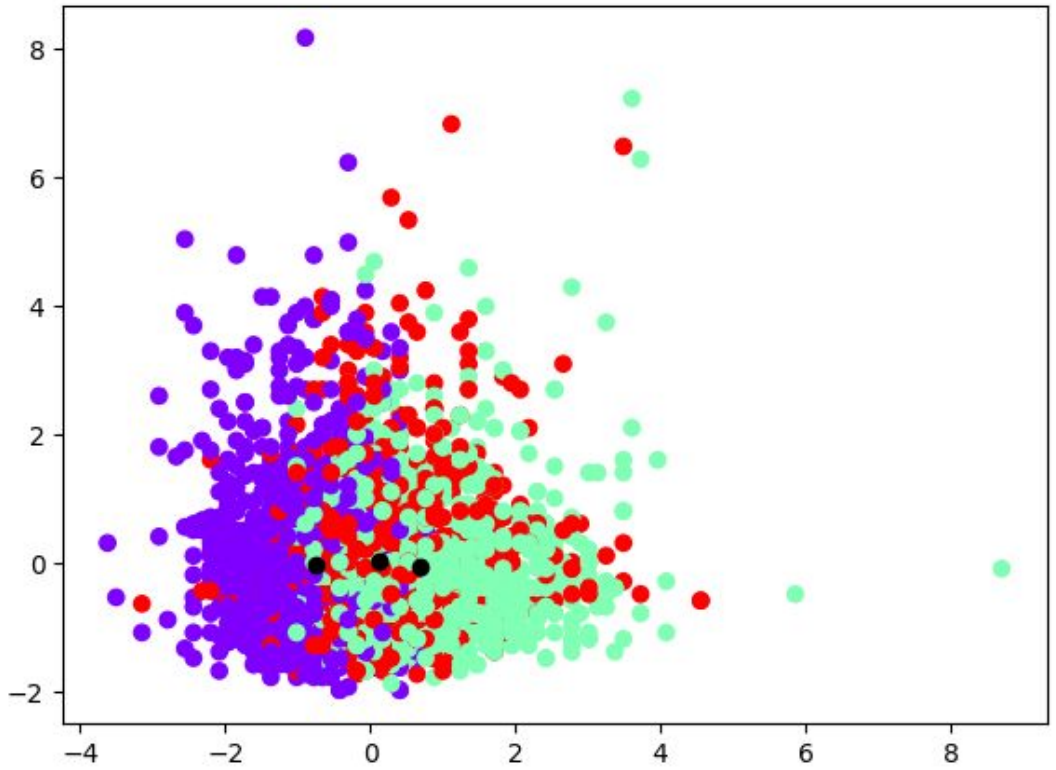


Elbow Method



K Means (k = 3)

Accuracy: 0.27



Basándonos en la gráfica, el número de clusters k adecuado serían 2 o 3.

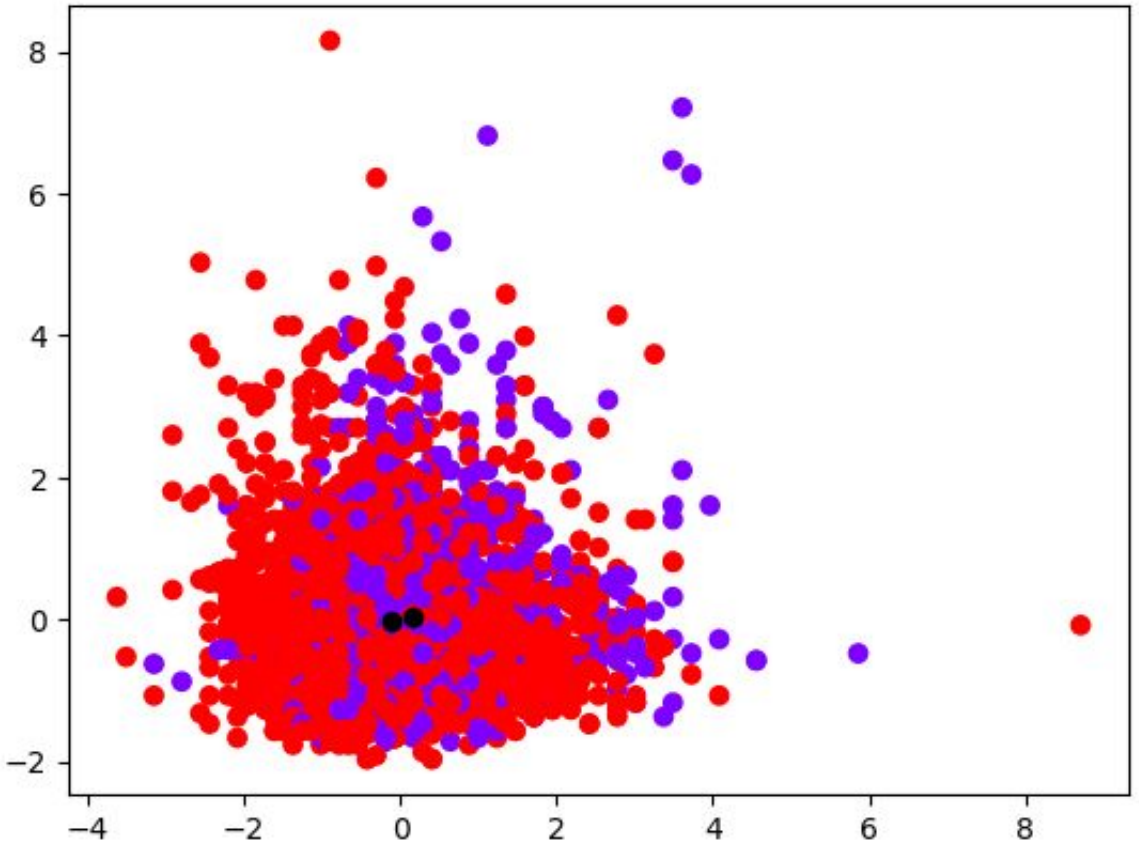
K Means (k = 2)

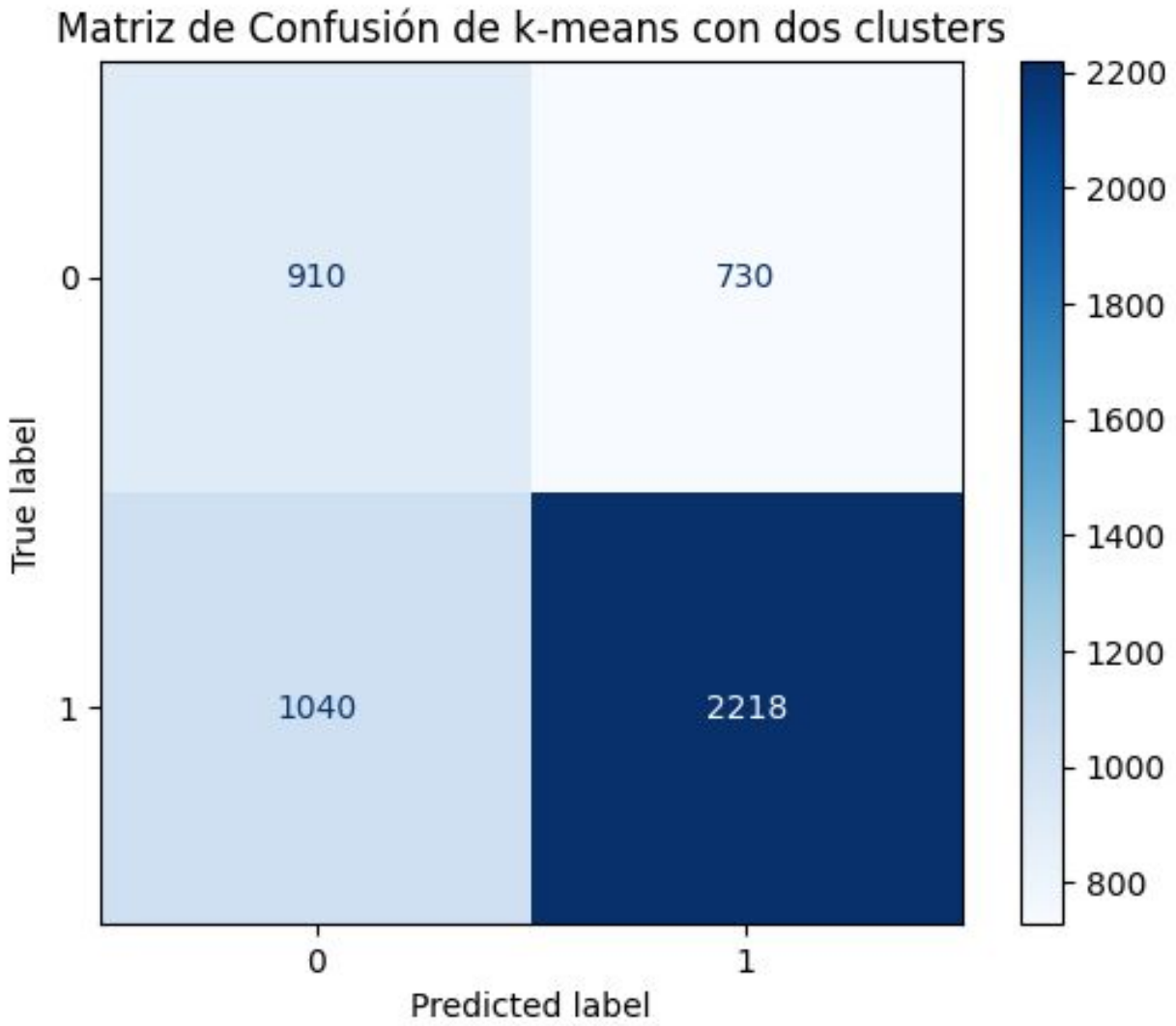
Kmeans con 2 clusters

```
#@title Kmeans con 2 clusters
kmeans = KMeans(n_clusters=2, random_state=27)
kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
print(y)
```

Accuracy: 0.64

	precision	recall	f1-score	support
0	0.47	0.55	0.51	1640
1	0.75	0.68	0.71	3258
accuracy			0.64	4898
macro avg	0.61	0.62	0.61	4898
weighted avg	0.66	0.64	0.65	4898




Matriz de Confusión de k-means con dos clusters
910
730
1040
2218
0
1
True label
0
1
Predicted label
2200
2000
1800
1600
1400
1200
1000
800

Resultados

	Modelo	Accuracy	Accuracy Cross Validation	Test Loss
0	Random Forest	0.823469	0.836257	NaN
1	Linear SVC	0.735714	0.752343	NaN
2	GaussianNB	0.705102	0.337918	NaN
3	DL	0.674490	NaN	0.260362
4	KMeans (k=3)	0.274194	NaN	NaN
5	KMeans (k=2)	0.638628	NaN	NaN
6	KMeans PCA	0.368722	NaN	NaN
7	KMeans TSNE	0.337281	NaN	NaN
8	DBSCAN	0.243365	NaN	NaN
9	DBSCAN PCA	0.331972	NaN	NaN
10	DBSCAN TSNE	0.313189	NaN	NaN

El mejor modelo de entre todos los utilizados en este proyecto finalmente fue el **RANDOM FOREST**.

Preguntas

Universidad Industrial de Santander

UIS 1948 2025

¡Gracias!