

Challenge Intermedio: Modelado con Spark ML

PÉREZ ROSAS LUIS ALFREDO MCD UDG 2025-B GRUPO 2

Este challenge lleva a los estudiantes un paso más allá en el uso de PySpark, implementando Spark ML para realizar un modelo predictivo basado en una gran base de datos.

Spark ML (Machine Learning) es la biblioteca de aprendizaje automático de Apache Spark, diseñada para manejar de manera eficiente y escalable grandes volúmenes de datos.

Objetivos

2. Objetivo general:

Aplicar algoritmos de machine learning en grandes bases de datos utilizando Spark ML.

3. Objetivo específico:

Descargar una base de datos desde UCI Machine Learning Repository, limpiarla, y entrenar un modelo predictivo de clasificación utilizando Spark ML.

1. Cargar un dataset público (recomendado: *Wine Quality UCI*) con Spark.
2. Realizar **EDA** con gráficos (histogramas, boxplots, correlación, distribución de la etiqueta).
3. Construir un **pipeline** (VectorAssembler + StandardScaler) y entrenar **al menos dos modelos** (LR, RF).
4. Evaluar con métricas y **gráficas** (ROC, PR, matriz de confusión, importancia de variables).
5. Realizar **búsqueda de hiperparámetros** con **CrossValidator**.

Requisitos técnicos

- Spark 3.4+ con PySpark.
- Para graficar, usar `matplotlib` (una figura por gráfico, **sin** estilos o colores específicos).
- Para graficar desde Spark, convertir a pandas con `toPandas()` cuando sea necesario.

```
In [2]: # === 1) Setup de Spark ===
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder
```

```

    .appName("wine-quality-ml-v2")
    # .config("spark.driver.memory", "4g") # opcional
    .getOrCreate()
)

spark

```

Out[2]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	v3.5.7
Master	local[*]
AppName	wine-quality-ml-v2

Carga del dataset

Base de datos:

Base de datos: Wine Quality Dataset

Plataforma: UCI Machine Learning Repository

Instrucciones: Descargar y cargar los datos en Apache Spark para crear el modelo.

Usa el conjunto **Wine Quality** (red o white) del UCI Repository. Ajusta la ruta al CSV.

In [3]:

```

# === 2) Carga de datos ===
# Ajusta la ruta al archivo CSV (usa el separador ';' del dataset original)
# Ejemplo: path = "/dbfs/FileStore/winequality-red.csv" o path local si corres fuer
path = "data/winequality-red.csv" # <-- CAMBIA ESTA RUTA
raw = spark.read.csv(path, header=True, inferSchema=True, sep=';')
raw.printSchema()
raw.show(10)

```

```

root
|-- fixed acidity: double (nullable = true)
|-- volatile acidity: double (nullable = true)
|-- citric acid: double (nullable = true)
|-- residual sugar: double (nullable = true)
|-- chlorides: double (nullable = true)
|-- free sulfur dioxide: double (nullable = true)
|-- total sulfur dioxide: double (nullable = true)
|-- density: double (nullable = true)
|-- pH: double (nullable = true)
|-- sulphates: double (nullable = true)
|-- alcohol: double (nullable = true)
|-- quality: integer (nullable = true)

+-----+-----+-----+-----+
|fixed acidity|volatile acidity|citric acid|residual sugar|chlorides|free sulfur dio
xide|total sulfur dioxide|density| pH|sulphates|alcohol|quality|
+-----+-----+-----+-----+
|      7.4|        0.7|       0.0|        1.9|     0.076|
11.0|          34.0| 0.9978|3.51|       0.56|     9.4|      5|
|      7.8|        0.88|       0.0|        2.6|     0.098|
25.0|          67.0| 0.9968|3.2|       0.68|     9.8|      5|
|      7.8|        0.76|       0.04|        2.3|     0.092|
15.0|          54.0| 0.997|3.26|       0.65|     9.8|      5|
|     11.2|        0.28|       0.56|        1.9|     0.075|
17.0|          60.0| 0.998|3.16|       0.58|     9.8|      6|
|      7.4|        0.7|       0.0|        1.9|     0.076|
11.0|          34.0| 0.9978|3.51|       0.56|     9.4|      5|
|      7.4|        0.66|       0.0|        1.8|     0.075|
13.0|          40.0| 0.9978|3.51|       0.56|     9.4|      5|
|      7.9|        0.6|       0.06|        1.6|     0.069|
15.0|          59.0| 0.9964|3.3|       0.46|     9.4|      5|
|      7.3|        0.65|       0.0|        1.2|     0.065|
15.0|          21.0| 0.9946|3.39|       0.47|    10.0|      7|
|      7.8|        0.58|       0.02|        2.0|     0.073|
9.0|          18.0| 0.9968|3.36|       0.57|     9.5|      7|
|      7.5|        0.5|       0.36|        6.1|     0.071|
17.0|         102.0| 0.9978|3.35|       0.8|    10.5|      5|
+-----+-----+-----+-----+
-----+-----+-----+-----+
only showing top 10 rows

```

Ingeniería de características y etiqueta binaria

Creamos `label_quality_7_or_more = 1` si `quality >= 7` buena calidad, de lo contrario `0` mala calidad o media. (Puedes justificar otro umbral).

```
In [4]: # === 3) Ingeniería de características y etiqueta ===
from pyspark.sql import functions as F
```

```
wine = raw.withColumn("label_quality_7_or_more", (F.col("quality") >= 7).cast("int"))
wine.select("quality", "label_quality_7_or_more").groupBy("label_quality_7_or_more")

+-----+-----+
|label_quality_7_or_more|count|
+-----+-----+
|                      1|  217|
|                      0| 1382|
+-----+-----+
```

EDA — Gráficas obligatorias

Para graficar, convertimos a pandas con `toPandas()`.

- Histogramas por variable numérica
- Boxplots
- Matriz de correlación
- Distribución de la etiqueta

```
In [5]: # === 4) EDA: Histogramas, Boxplots, Correlación, Distribución de etiqueta ===
import matplotlib.pyplot as plt

pdf = wine.toPandas()

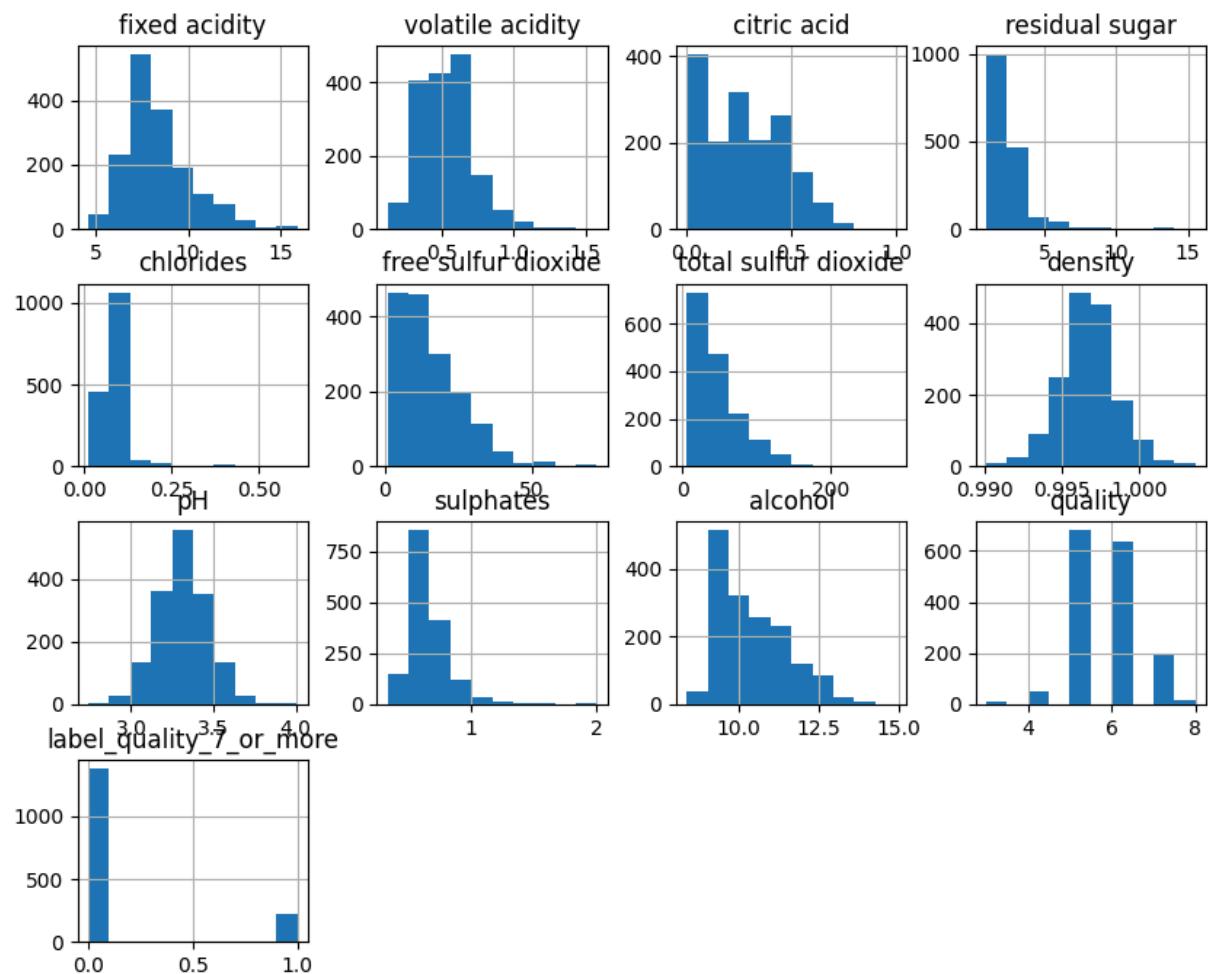
# 4.1 Histogramas
ax = pdf.hist(figsize=(10, 8))
plt.show()

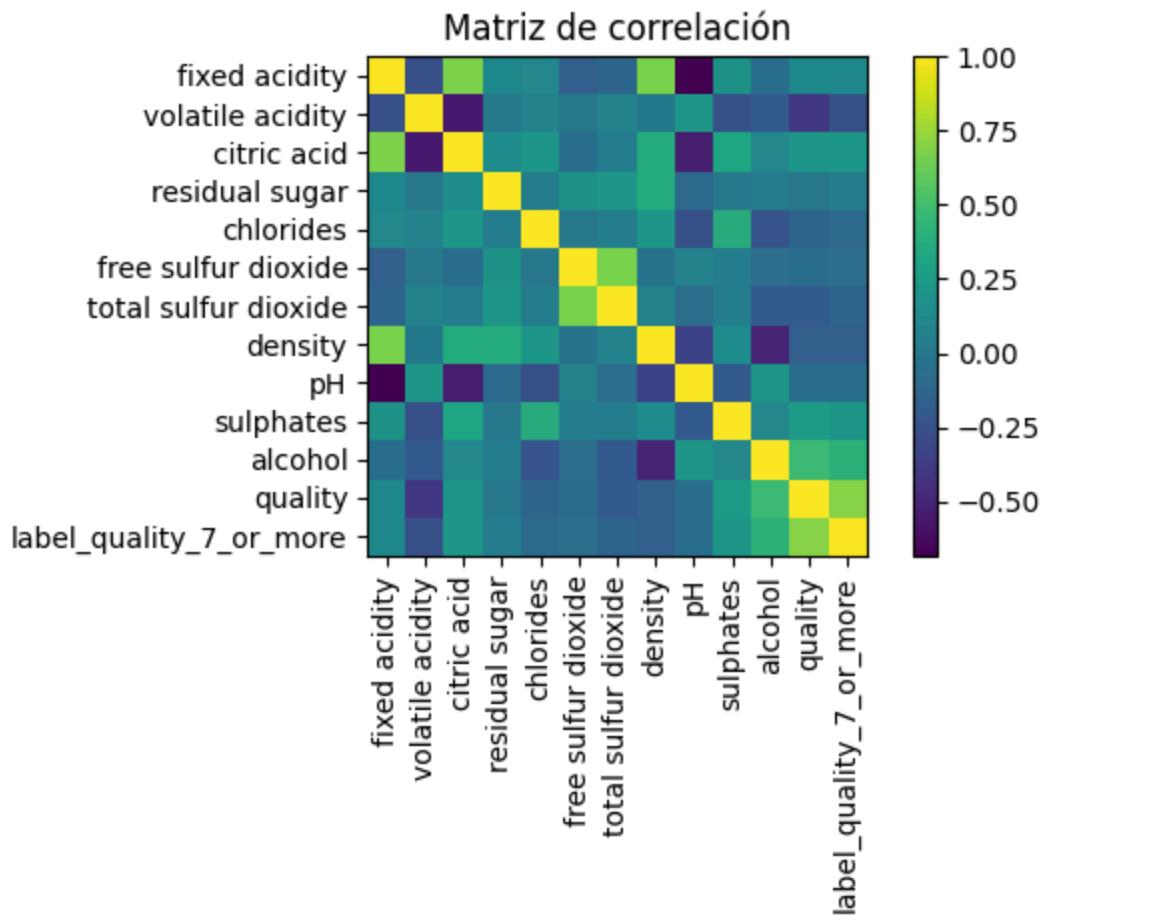
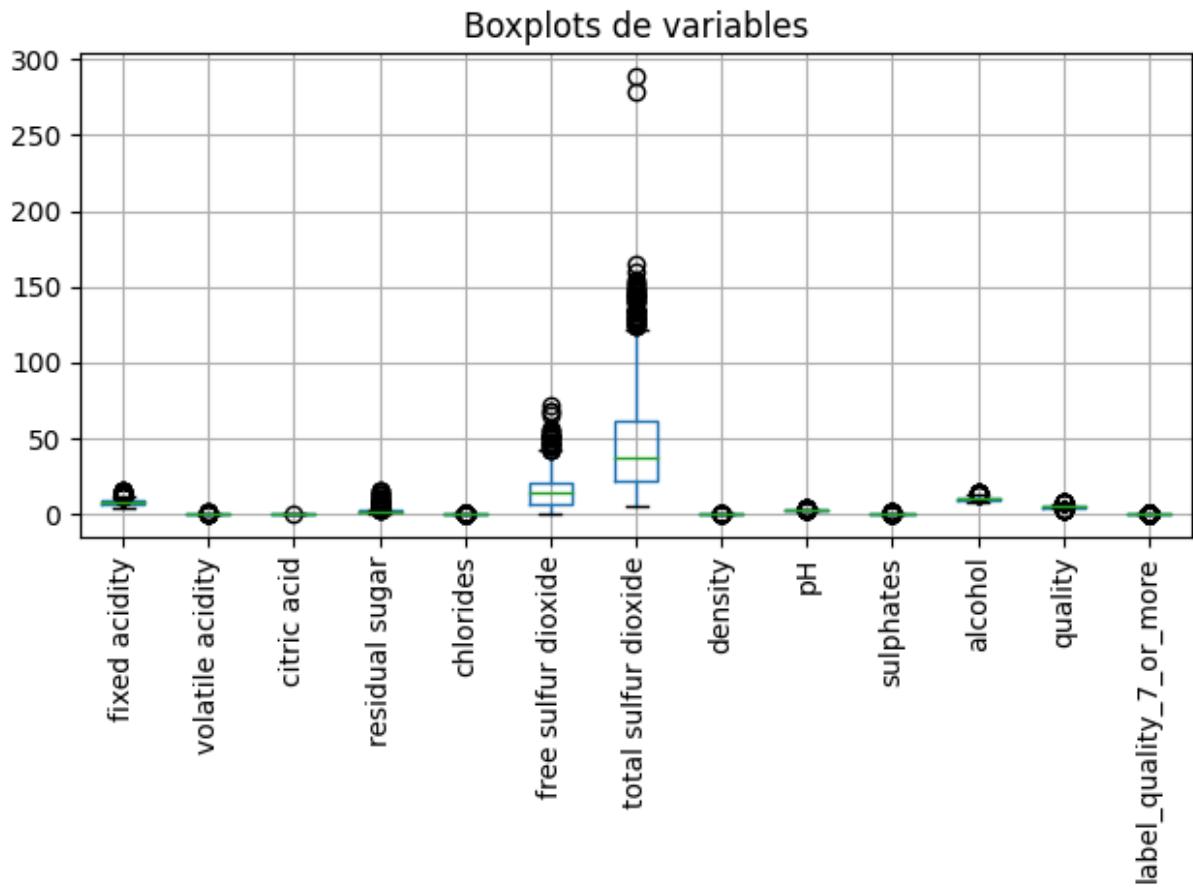
# 4.2 Boxplots
plt.figure()
pdf.boxplot(rot=90)
plt.title("Boxplots de variables")
plt.tight_layout()
plt.show()

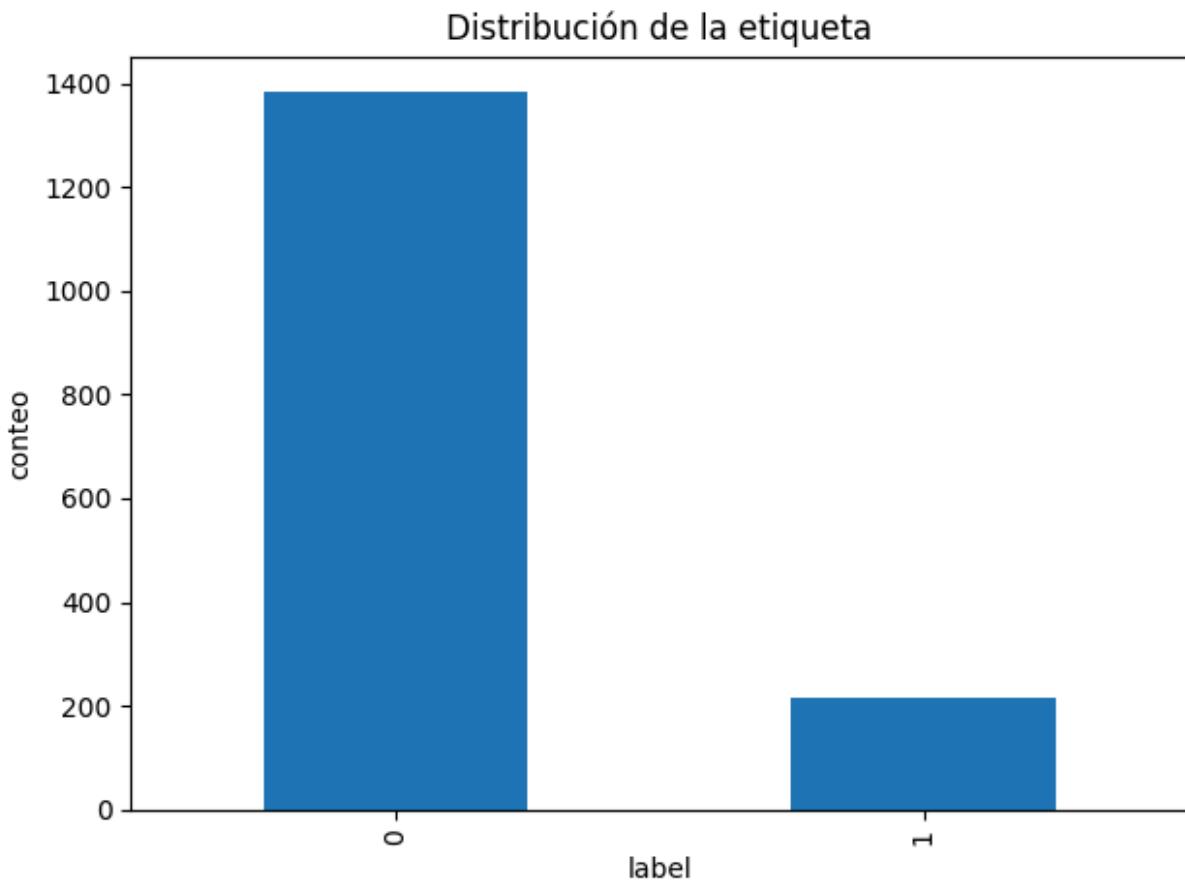
# 4.3 Matriz de correlación (solo numéricas)
plt.figure()
corr = pdf.corr(numeric_only=True)
plt.imshow(corr, interpolation='nearest')
plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
plt.yticks(range(len(corr.columns)), corr.columns)
plt.colorbar()
plt.title("Matriz de correlación")
plt.tight_layout()
plt.show()

# 4.4 Distribución de la etiqueta
plt.figure()
pdf['label_quality_7_or_more'].value_counts().sort_index().plot(kind='bar')
plt.title("Distribución de la etiqueta")
plt.xlabel("label")
plt.ylabel("conteo")
```

```
plt.tight_layout()  
plt.show()
```







Preparación de datos y split

Construiremos el vector de *features* y escalaremos con `StandardScaler`.

```
In [14]: # === 5) Preparación de datos y split ===
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml import Pipeline

feature_cols = [c for c in wine.columns if c not in ['quality', 'label_quality_7_or_more']]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features_raw")
scaler = StandardScaler(inputCol="features_raw", outputCol="features")

pipeline_prep = Pipeline(stages=[assembler, scaler])
prepared = pipeline_prep.fit(wine).transform(wine)

train, test = prepared.randomSplit([0.8, 0.2], seed=42)
train.count(), test.count()
```

Out[14]: (1324, 275)

Modelado: Regresión Logística y Random Forest

Entrenamos y comparamos dos modelos base.

```
In [15]: # === 6) Modelos base (LR y RF) ===
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier

lr = LogisticRegression(featuresCol="features", labelCol="label_quality_7_or_more")
rf = RandomForestClassifier(featuresCol="features", labelCol="label_quality_7_or_more")

lr_model = lr.fit(train)
rf_model = rf.fit(train)

pred_lr = lr_model.transform(test)
pred_rf = rf_model.transform(test)

pred_lr.select("label_quality_7_or_more", "prediction", "probability").show(5, truncate=True)
pred_rf.select("label_quality_7_or_more", "prediction", "probability").show(5, truncate=True)

+-----+-----+
|label_quality_7_or_more|prediction|probability
+-----+-----+
|1|1.0|[0.2097071365846944, 0.7902928634153056]
|0|0.0|[0.8611282558287734, 0.1388717441712266]
|0|0.0|[0.9862567130821678, 0.01374328691783222]
|0|0.0|[0.9930293684927364, 0.006970631507263625]
|1|1.0|[0.3536065744149445, 0.6463934255850555]
+-----+-----+
only showing top 5 rows

+-----+-----+
|label_quality_7_or_more|prediction|probability
+-----+-----+
|1|1.0|[0.42890670329454894, 0.5710932967054512]
|0|0.0|[0.8816963627818465, 0.11830363721815361]
|0|0.0|[0.8099731940629805, 0.1900268059370196]
|0|0.0|[0.9618830311992284, 0.0381169688007717]
|1|1.0|[0.2456994556633248, 0.7543005443366751]
+-----+-----+
only showing top 5 rows
```

Evaluación inicial (AUC)

Calculamos el AUC para ambos modelos.

AUC significa "Area Under the Curve" (Área Bajo la Curva)

Los valores del AUC varían de 0.5 a 1.0:

AUC = 1.0: Indica un rendimiento predictivo perfecto; el modelo puede separar perfectamente las clases positivas y negativas sin errores.

AUC entre 0.8 y 0.9: Se considera un rendimiento excelente.

AUC entre 0.7 y 0.8: Se considera un rendimiento aceptable.

AUC = 0.5: Indica que el modelo no es mejor que una suposición aleatoria (azar) para distinguir entre las clases.

```
In [8]: # === 7) Evaluación (AUC) ===
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(labelCol='label_quality_7_or_more', rawPr
auc_lr = evaluator.evaluate(pred_lr)
auc_rf = evaluator.evaluate(pred_rf)

print("AUC LR:", auc_lr)
print("AUC RF:", auc_rf)
```

AUC LR: 1.0
AUC RF: 1.0

Curvas ROC y Precision-Recall (LR)

Usamos el `summary` del modelo de LR para graficar ROC y PR.

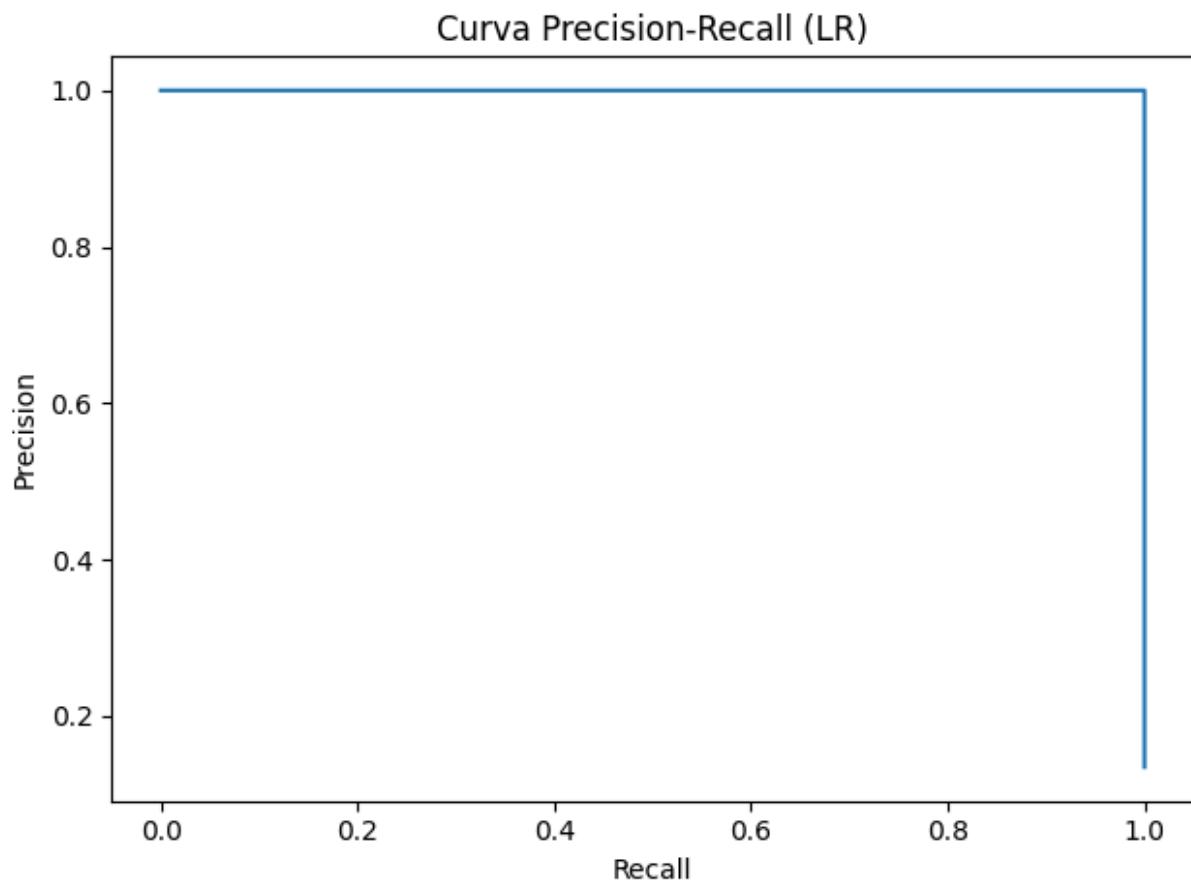
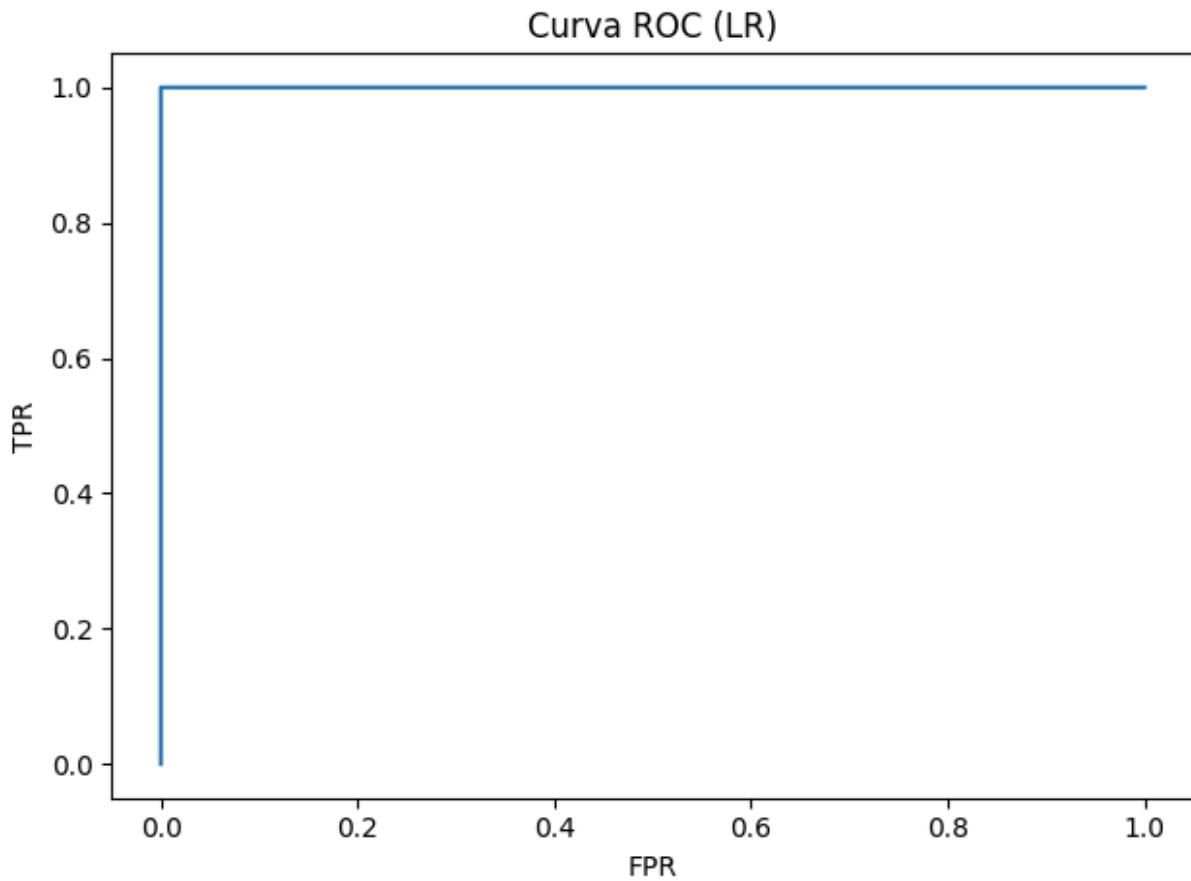
La curva ROC (Receiver Operating Characteristic) es una herramienta gráfica utilizada para evaluar el rendimiento de los modelos de clasificación binaria, mientras que LR comúnmente se refiere a la regresión logística (Logistic Regression), que es un tipo de modelo que a menudo se evalúa con la curva ROC.

```
In [9]: # === 8) Curvas ROC y PR (LR) ===
import matplotlib.pyplot as plt

train_summary = lr_model.summary
roc_df = train_summary.roc.toPandas()
pr_df = train_summary.pr.toPandas()

# ROC
plt.figure()
plt.plot(roc_df['FPR'], roc_df['TPR'])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('Curva ROC (LR)')
plt.tight_layout()
plt.show()

# PR
plt.figure()
plt.plot(pr_df['recall'], pr_df['precision'])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision-Recall (LR)')
plt.tight_layout()
plt.show()
```



Matriz de confusión (RF)

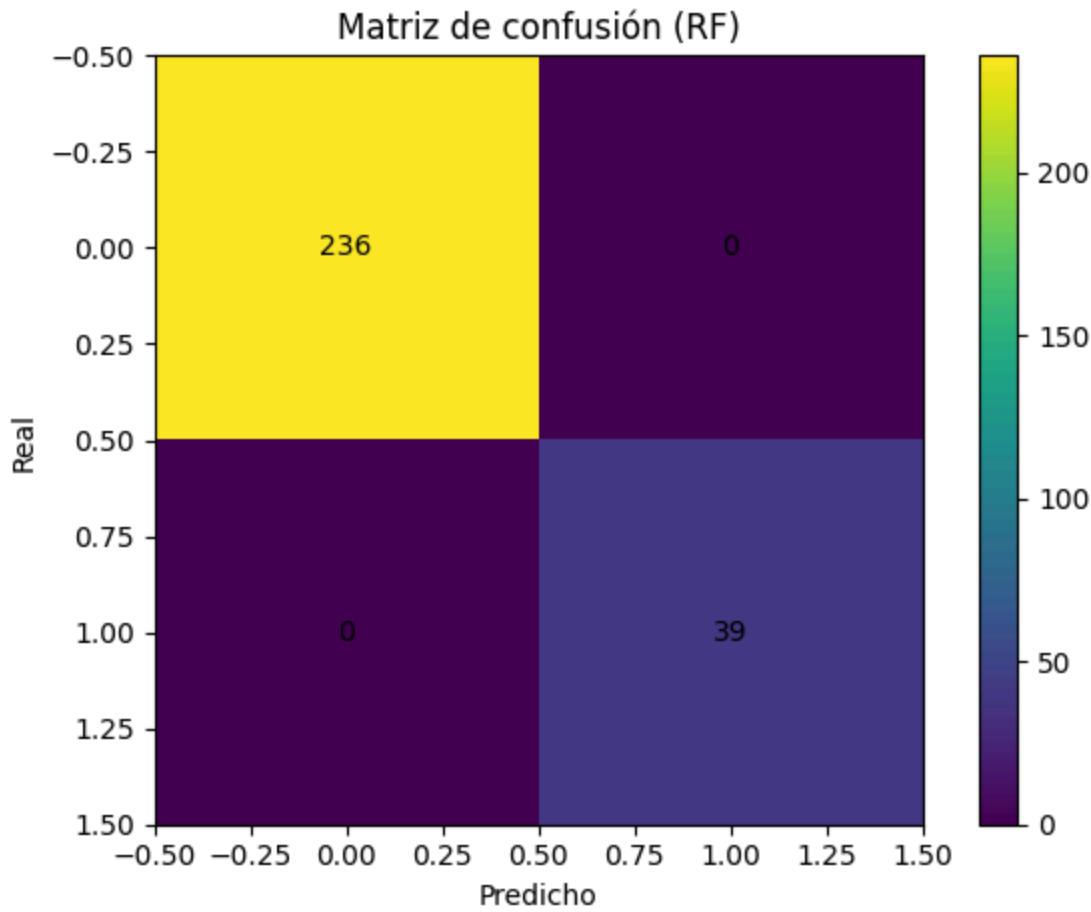
Convertimos a pandas para graficar.

Una matriz de confusión es una tabla que se utiliza para evaluar el rendimiento de un modelo de clasificación en inteligencia artificial, como un modelo de Random Forest (RF). Permite visualizar y comparar los valores reales de los datos con los valores predichos por el modelo.

```
In [10]: # === 9) Matriz de confusión (RF) ===
from sklearn.metrics import confusion_matrix
import numpy as np

preds_rf = pred_rf.select('prediction', 'label_quality_7_or_more').toPandas()
cm = confusion_matrix(preds_rf['label_quality_7_or_more'], preds_rf['prediction'])

plt.figure()
plt.imshow(cm, interpolation='nearest')
plt.title('Matriz de confusión (RF)')
plt.xlabel('Predicho')
plt.ylabel('Real')
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha='center', va='center')
plt.colorbar()
plt.tight_layout()
plt.show()
```



Importancia de variables (RF)

```
In [11]: # === 10) Importancia de variables (RF) ===
import pandas as pd
fi = pd.DataFrame({
    'feature': feature_cols,
    'importance': rf_model.featureImportances.toArray()
}).sort_values('importance', ascending=False)

fi.head(10)
```

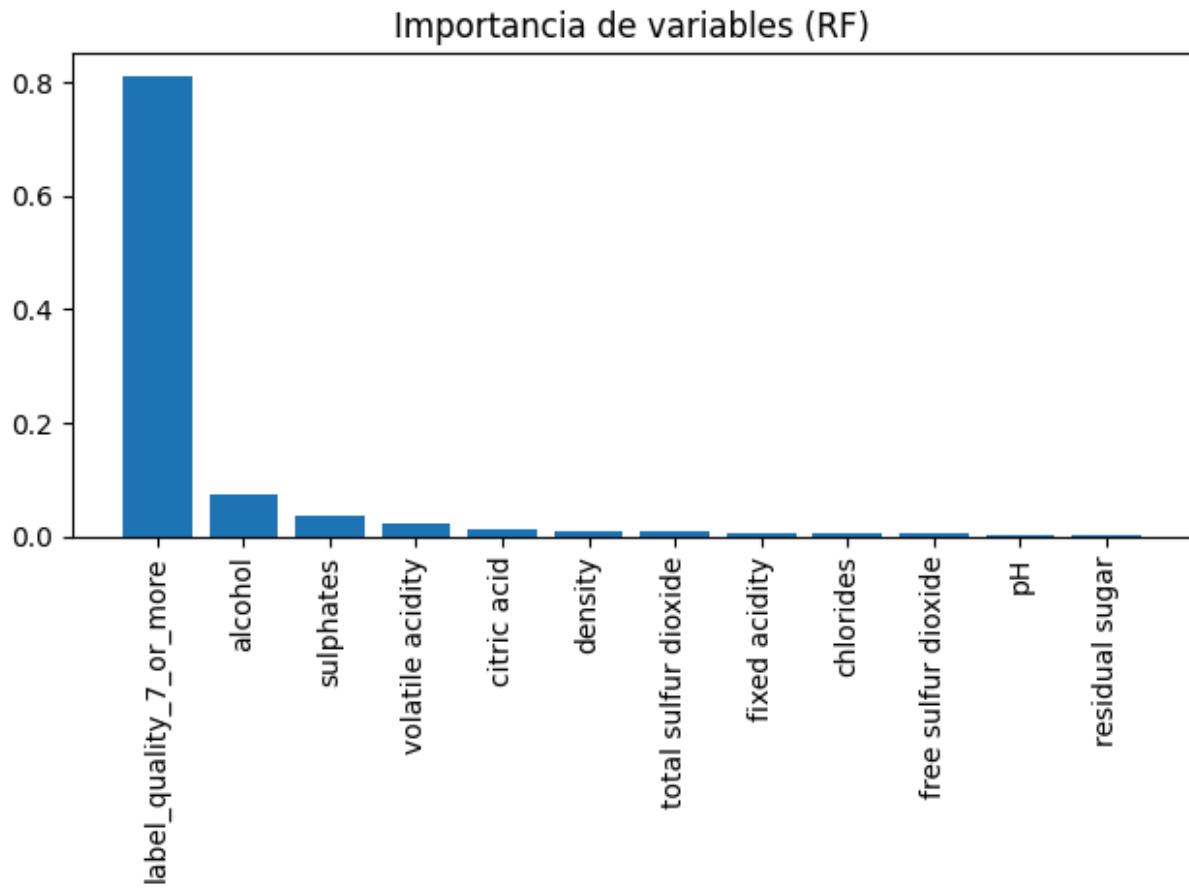
Out[11]:

	feature	importance
11	label_quality_7_or_more	0.809917
10	alcohol	0.073254
9	sulphates	0.038168
1	volatile acidity	0.022624
2	citric acid	0.011713
7	density	0.011019
6	total sulfur dioxide	0.007962
0	fixed acidity	0.007555
4	chlorides	0.005419
5	free sulfur dioxide	0.005400

In [16]:

```
# Gráfico de importancia (una figura, sin estilos de color específicos)
import matplotlib.pyplot as plt

plt.figure()
plt.bar(fi['feature'], fi['importance'])
plt.xticks(rotation=90)
plt.title('Importancia de variables (RF)')
plt.tight_layout()
plt.show()
```



Búsqueda de hiperparámetros + CrossValidator (RF)

```
In [12]: # === 11) Cross-Validation (RF) ===
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

paramGrid = (ParamGridBuilder()
    .addGrid(rf.numTrees, [100, 200, 400])
    .addGrid(rf.maxDepth, [5, 10])
    .build())

cv = CrossValidator(
    estimator=rf,
    estimatorParamMaps=paramGrid,
    evaluator=BinaryClassificationEvaluator(labelCol='label_quality_7_or_more'),
    numFolds=3,
    parallelism=2
)

cv_model = cv.fit(train)
cv_pred = cv_model.bestModel.transform(test)
auc_cv = evaluator.evaluate(cv_pred)

print("AUC RF (CV):", auc_cv)
cv_model.bestModel
```

AUC RF (CV): 1.0

Out[12]: RandomForestClassificationModel: uid=RandomForestClassifier_ed3115acef20, numTrees =100, numClasses=2, numFeatures=12

Comparativa de modelos

Calcula métricas (Accuracy, Precision, Recall, F1) y compáralas.

```
In [13]: # === 12) Comparativa de métricas ===
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def metrics_from_spark_pred(spark_pred_df):
    pdf_pred = spark_pred_df.select('label_quality_7_or_more', 'prediction').toPand
    acc = accuracy_score(pdf_pred['label_quality_7_or_more'], pdf_pred['prediction'])
    pre = precision_score(pdf_pred['label_quality_7_or_more'], pdf_pred['prediction'])
    rec = recall_score(pdf_pred['label_quality_7_or_more'], pdf_pred['prediction'])
    f1 = f1_score(pdf_pred['label_quality_7_or_more'], pdf_pred['prediction'])
    return acc, pre, rec, f1

acc_lr, pre_lr, rec_lr, f1_lr = metrics_from_spark_pred(pred_lr)
acc_rf, pre_rf, rec_rf, f1_rf = metrics_from_spark_pred(pred_rf)
acc_cv, pre_cv, rec_cv, f1_cv = metrics_from_spark_pred(cv_pred)

import pandas as pd
summary = pd.DataFrame({
    'modelo': ['LR', 'RF', 'RF (CV)'],
    'AUC': [auc_lr, auc_rf, auc_cv],
    'Accuracy': [acc_lr, acc_rf, acc_cv],
    'Precision': [pre_lr, pre_rf, pre_cv],
    'Recall': [rec_lr, rec_rf, rec_cv],
    'F1': [f1_lr, f1_rf, f1_cv]
})
summary
```

	modelo	AUC	Accuracy	Precision	Recall	F1
0	LR	1.0	1.0	1.0	1.0	1.0
1	RF	1.0	1.0	1.0	1.0	1.0
2	RF (CV)	1.0	1.0	1.0	1.0	1.0

Conclusiones

- Compara resultados y justifica el modelo recomendado.
- Discute *trade-offs* (complejidad, interpretabilidad, rendimiento).
- (Opcional) Propuestas de mejora: balanceo de clases, más features, MLflow, explicabilidad.

De acuerdo al análisis de datos realizado con herramientas de ciencia de datos (PySpark) a la base de datos "winequality-red.csv" podemos identificar que de manera inicial según la

calidad previamente obtenida; se identifican 217 vinos con buena calidad mientras que 1382 tienen una calidad mala o menor a los estandares del modelo y que son estos datos con los cuales podremos entrenar un modelo de regresion logistica con la finalidad de que el mismo nos sirva para clasificar si un vino es de buena calidad o en su caso de mala calidad.

A fin de identificar la funcionalidad del modelo, se utiliza otro modelo de alta confiabilidad para clasificar un vino de acuerdo a las caracteristicas existentes en la muestra

Los modelos utilizados son:

Un modelo de regresion logistica (LogisticRegression)

Un modelo de arboles aleatorios (RandomForestClassifier)

al comparar los resultados de ambos podemos identificar coincidencias en cuanto a su clasificacion concluyendo que ambos modelos podrian bien ser utilizados para clasificar vinos

Y que el modelo de regresion logistica que es nuestro caso de estudio tiene una alta confiabilidad

El modelo propuesto mayormente en este ejercicio es el de REGRESION LOGISTICA el cual tiene las siguientes caracteristicas:

Complejidad: Baja

Interpretabilidad: Facil de interpretar

Rendimiento: No requiere gran cantidad de recursos de computo

El modelo de regresion logistica propuesto para la resolucion de este problema resulta en una excelente opcion ya que el interes final del modelo es identificar un resultado claro y conciso sobre si el vino en cuestion es de buena calidad SI o NO

PRESENTO 

PÉREZ ROSAS LUIS ALFREDO GRUPO 2 PROCESAMIENTO DE GRANDES BASES DE DATOS

INSTRUCTOR: DR. OSVALDO MEDINA