Spark Setup & Data Loading

python

CopyEdit

```
from pyspark.sql import SparkSession
```

→ To start working with big data using Spark.

python

CopyEdit

```
spark = SparkSession.builder.appName("StockMarketAnalysis").getOrCreate()
```

→ Initializes a Spark session named "StockMarketAnalysis".

python

CopyEdit

```
df = spark.read.csv("/content/infolimpioavanzadoTarget.csv", header=True, inferSchema=True)
```

→ Loads the stock data CSV file (with header and type inference).

🔷 Select and Order Relevant Columns

python

CopyEdit

```
df = df.select("Date", "Open", "High", "Low", "Close", "Volume")
df = df.orderBy("Date")
```

→ Chooses only important stock columns and orders them by date.

🔷 Add Lag Feature

python

CopyEdit

```
from pyspark.sql.functions import col, lag
from pyspark.sql.window import Window

windowSpec = Window.orderBy("Date")
df = df.withColumn("Prev_Close", lag("Close").over(windowSpec))
df = df.na.drop()
```

→ Adds a new column Prev_Close which stores the previous day's closing price (used for trend analysis). Then removes rows with nulls.

◈ Feature Engineering & Scaling

python

CopyEdit

```
from pyspark.ml.feature import VectorAssembler, MinMaxScaler
```

→ Prepares data for machine learning (converts multiple columns into one feature vector and scales it).

python

CopyEdit

```
feature_cols = ["Open", "High", "Low", "Volume", "Prev_Close"]

assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

df = assembler.transform(df)
```

→ Combines feature columns into a single vector called features.

python

CopyEdit

```
scaler = MinMaxScaler(inputCol="features", outputCol="scaled_features")

scaler_model = scaler.fit(df)

df = scaler_model.transform(df)
```

→ Scales the feature vector values between 0 and 1 to help the neural network train better.

◈ Convert to Pandas and Prepare for Deep Learning

python

CopyEdit

```
pandas_df = df.select("scaled_features", "Close").toPandas()
```

→ Converts the Spark DataFrame to a Pandas DataFrame so that TensorFlow/Keras can work with it.

python

CopyEdit

```
X = np.array([np.array(x) for x in pandas_df["scaled_features"]])

y = pandas_df["Close"].values
```

→ Creates feature matrix X and target y (actual close price).

python

CopyEdit

split = int(0.8 * len(X))

X_train, X_test = X[:split], X[split:]

y_train, y_test = y[:split], y[split:]

→ Splits the data into 80% training and 20% testing.

◈ **Reshape Data for LSTM**

python

CopyEdit

X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))

X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

→ LSTM expects input in 3D shape: (samples, time steps, features)

◈ **Build the LSTM Model**

python

CopyEdit

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

model = Sequential()

model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))

model.add(Dropout(0.2))

model.add(Dense(1))

→ Creates an LSTM neural network:

- 50 LSTM units
- Dropout layer to prevent overfitting
- Dense layer for outputting one predicted value

🔷 Compile and Train Model

python

CopyEdit

```
model.compile(optimizer='adam', loss='mse')
model.summary()
```

→ Compiles the model using **Mean Squared Error (MSE)** as loss.

python

CopyEdit

```
model.fit(X_train, y_train, epochs=1, batch_size=16, validation_data=(X_test, y_test), verbose=1)
```

→ Trains the model on your training data for 1 epoch (you can increase this for better accuracy).

🔷 Predict and Evaluate

python

CopyEdit

```
y_pred = model.predict(X_test)
```

→ Predicts stock prices on the test data.

python

CopyEdit

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

→ Calculates how accurate the predictions are. Lower MSE = better performance.

🔷 Visualize Predictions

python

CopyEdit

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(y_test, label='Actual Prices', color='blue')
```

```
plt.plot(y_pred, label='Predicted Prices', color='orange')

plt.title("Stock Price Prediction - Actual vs Predicted")

plt.xlabel("Time Steps")

plt.ylabel("Price")

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```