**Guys, Line-by-line explanation of the program**

# -\*- coding: utf-8 -\*-

# task12\_part1\_lr\_rdd.py

# Linear Regression with Spark MLlib (RDD API, Spark 1.6)

* The first line declares the file’s text encoding (UTF-8).
* The comments are just metadata: filename and what the script does.

from pyspark import SparkContext

from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD

* Imports:
  + SparkContext is the handle to a running Spark application (it creates RDDs, etc.).
  + LabeledPoint is MLlib’s container for one training example: **label** (the target we want to predict) + **features** (the inputs).
  + LinearRegressionWithSGD is the classic MLlib linear regression trainer that uses **Stochastic Gradient Descent** to learn the best line.

# Step 1: Start SparkContext

sc = SparkContext(appName="Task12\_Part1\_LinearRegression")

* Creates the **SparkContext**. The appName is what you’ll see in Spark logs/UI.
* Under the hood this starts the driver, sets up executors (in local mode, just threads), and prepares to create/operate on **RDDs**.

# Step 2: Load CSV data (local file)

data = sc.textFile("file:///home/cloudera/housing.csv")

* Reads the file as an **RDD of strings**, one line per record.
* The file:/// URI tells Spark to read from the **local filesystem** on the driver.
  + (In true cluster mode you’d usually put the file in HDFS or use a shared path.)

# Step 3: Parse CSV into LabeledPoint (label = price, feature = size)

parsed = data.map(lambda line: line.split(",")) \

.map(lambda parts: LabeledPoint(float(parts[1]), [float(parts[0])]))

* **data.map(lambda line: line.split(","))**  
  Splits each text line on the comma, giving ["size","price"] as strings.
* **Second map** converts those strings into a **LabeledPoint**:
  + float(parts[1]) → the **label** (house **price**).
  + [float(parts[0])] → the **features vector** with a single number (house **size**).  
    (Even for one feature, MLlib expects a vector-like container.)
* Result: parsed is an **RDD[LabeledPoint]** ready for MLlib.

# Step 4: Train Linear Regression model using Stochastic Gradient Descent

model = LinearRegressionWithSGD.train(parsed, iterations=100, step=0.0000001)

* Trains a linear model of the form  
  **prediction = intercept + weight × size**.
* **iterations=100**: number of passes of gradient updates over the data.
* **step=0.0000001**: the **learning rate** (how big each update is).
  + It’s very small here because your feature values (size in thousands) and labels (price in tens of thousands) are large; a big step could diverge.
  + In general, scaling/normalizing features lets you use a larger, safer step.
* Under the hood, Spark iterates over the RDD, computes gradients, and updates intercept and weight to minimize squared error.

# Step 5: Print model parameters

print("Intercept (base price):", model.intercept)

print("Weight (price per sq ft):", model.weights)

* **model.intercept** is the y-intercept: the baseline price when size=0.
* **model.weights** is the learned coefficient(s). With one feature, it’s a single number: the **slope** (“price per square foot” in this simple setup).
* Together they define the learned line:
* predicted\_price = intercept + weight \* size

# Step 6: Test predictions

print("=== Predictions ===")

for size in [1200, 1800, 2500]:

predicted\_price = model.predict([size])

print("House size:", size, "sq ft -> Predicted price:", predicted\_price)

* Makes predictions for three example sizes.
* model.predict([size]) expects a **feature vector** (here, a single-element list).
* Prints the results so you can compare with your intuition/data:
  + If the dataset trend is about **$50 per sq ft**, predictions around 60k, 90k, 125k for 1200/1800/2500 are exactly what you’d expect.

sc.stop()

* **Clean shutdown** of the Spark application. Frees executors and resources.

**A couple of practical notes (for class discussion)**

* **Data format:** this script assumes **no header** in housing.csv. If you had a header line like size,price, you’d want to filter it out:
* header = data.first()
* data\_no\_header = data.filter(lambda line: line != header)
* **Feature scaling:** if features/labels have very different magnitudes, training may require a tiny learning rate (as here). A common alternative is to **scale** the features (e.g., divide size by 1000).
* **RDD vs DataFrame:** you used the **RDD API** here because it avoids modern dependencies (like NumPy/SciPy versions) on this old Spark 1.6 VM. On newer Spark, the **DataFrame-based API (pyspark.ml)** is preferred.