How to Perform Linear Regression in PySpark: A Step-by-Step Guide

Authored by

stats writer

January 18, 2026

Linear regression is a fundamental statistical method used extensively in MLlib to quantify the relationship between one or more predictor variables and a continuous dependent variable. It serves as a powerful foundation for predictive modeling.

This comprehensive, step-by-step guide demonstrates how to efficiently implement and fit a **linear regression model** using the PySpark framework, focusing on practical data preparation and robust model interpretation.

Understanding the Linear Regression Model

Before diving into the code, it is essential to understand that we are aiming to model a continuous outcome variable (the exam score) based on multiple input features (hours studied and prep exams taken). This methodology falls under **multiple linear regression**. Our goal is to determine the coefficients that best fit the observed data, allowing us to make accurate predictions about student performance.

The elegance of linear regression lies in its simplicity and interpretability, making it a crucial technique in data analysis and machine learning workflows, especially when using distributed computing tools like PySpark for handling large datasets.

Step 1: Setting up the PySpark Environment and Data Creation

The initial phase involves setting up the environment by initializing a **SparkSession** and defining the input data. We will create a PySpark DataFrame containing simulated student performance metrics, including hours spent studying, the number of preparatory exams taken, and the final exam score.

This dataset serves as the foundation for our predictive analysis, where 'hours' and 'prep_exams' will be our predictor variables, and 'score' will be the target variable. The following code snippet demonstrates the necessary setup and data definition in Python:

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

#define data
data = ,
,
,
,
,
,

```
,
,
,
,
,
,
,
,
,
,
,
,
]

#define column names
columns =

#create dataframe using data and column names
df = spark.createDataFrame(data, columns)

#view first five rows of dataframe
df.limit(5).show()

+-----+----------+-----+
|hours|prep_exams|score|
+-----+----------+-----+
| 1| 1| 76|
| 2| 3| 78|
| 2| 3| 85|
| 4| 5| 88|
| 2| 2| 72|
+-----+----------+-----+
```


Step 2: Defining the Multiple Linear Regression Equation

For this analysis, we are employing a multiple linear regression framework, utilizing both 'hours' and 'prep_exams' to predict the 'Exam Score'. The structural equation we aim to solve is defined as follows:

Exam Score = β0 + β1(hours) + β2(prep exams)

Here, **β0** represents the intercept, and **β1** and **β2** represent the coefficients (or weights) assigned to the corresponding predictor variables. The fitting process determines the optimal values for these beta coefficients by minimizing the squared difference between the predicted and actual scores.

Step 3: Data Preparation using VectorAssembler

PySpark's MLlib requires that all feature columns (predictor variables) be combined into a single vector column before training any model. We accomplish this crucial transformation using the **VectorAssembler** function, which is essential for ensuring compatibility with machine learning pipelines.

The **VectorAssembler** takes specified input columns ('hours' and 'prep_exams') and merges them into a new output column, which we label 'features'. This aggregated feature vector is the mandatory input format for PySpark's regression algorithms.

from pyspark.ml.feature import VectorAssembler

#specify predictor variables
assembler = VectorAssembler(inputCols=, outputCol='features')

#format DataFrame for linear regression
data = assembler.transform(df)
data = data.select('features', 'score')

Crucially, we define 'hours' and 'prep_exams' as our input columns for feature generation, and 'score' remains isolated as the designated label column (response variable) for the upcoming model training step.

Step 4: Training the Linear Regression Model

With the data successfully assembled into the required feature vector format, we can proceed to initialize and train the actual LinearRegression estimator from the PySpark MLlib. This process involves defining which column serves as the features, which serves as the label, and what the resulting prediction column should be named.

We instantiate the **LinearRegression** object, specifying 'features' as the input feature column and 'score' as the label column. The `.fit(data)` method then executes the training process, calculating the optimal coefficients necessary to minimize the residual sum of squares, which is the

core objective of Ordinary Least Squares (OLS) regression.

from pyspark.ml.regression import LinearRegression

#specify linear regression model to use
lin_reg = LinearRegression(featuresCol='features',
labelCol='score',
predictionCol='pred_score')

#fit linear regression model to data
fit = lin_reg.fit(data)

The resulting `fit` object encapsulates the trained model, including the calculated intercept, coefficients, and comprehensive summary statistics, which we are now ready to analyze.

Step 5: Analyzing Model Coefficients and Metrics

The final step in the modeling process is to extract and analyze the summary statistics provided by the fitted object. We are particularly interested in the intercept, the coefficients assigned to our predictor variables, and key performance metrics like the p-values and R-squared score.

The following Python commands utilize the `fit.intercept`, `fit.coefficients`, and the `fit.summary` attributes to display the core results of our linear regression calculation:

#view model intercept and coefficients
print(fit.intercept, fit.coefficients)

67.67352554133275

#view p-values of intercept and coefficents
print(fit.summary.pValues)

#view RMSE of model
print(fit.summary.r2)

0.7340272170388176

Based on these outputs, we can summarize the derived numerical parameters of the model:

Intercept (β0): **67.674**
Coefficient of Hours (β1): **5.556**
Coefficient of Prep Exams (β2): **-0.602**

Interpreting the Fitted Regression Equation

By substituting the calculated coefficients and intercept back into our original multiple linear regression formula, we derive the finalized prediction equation:

Exam Score = 67.674 + 5.556(hours) - 0.602(prep_exams)

This equation now allows us to predict the final exam score for any student, given their study habits. The interpretation of the coefficients is crucial: holding the number of prep exams constant, every additional hour of studying is associated with an increase of **5.556 points** in the exam score. Conversely, holding study hours constant, every additional prep exam taken is associated with a slight decrease of **0.602 points**, suggesting a potentially complex or confounding relationship between the variables.

For instance, if a student studies for 3 hours and takes 2 prep exams, the estimated score is calculated as follows: 67.674 + 5.556*(3) - 0.602*(2), resulting in an estimated exam score of **83.1**.

Evaluating Statistical Significance (p-values)

Statistical significance is evaluated by examining the p-values associated with the intercept and each coefficient. A p-value below a conventional significance level (typically 0.05) indicates that the relationship between the predictor and the response variable is unlikely due to random chance. The output provided the p-values in array form, corresponding to :

P-value of Intercept: **1.01 x 10-5 (<0.001)**
P-value of Hours: **0.519**
P-value of Prep Exams: **1.47 x 10-14 (<0.0001)**

We observe that the p-value for 'Prep Exams' is extremely small, confirming a **statistically significant** association with the final exam score. However, the p-value for 'Hours' (0.519) is much greater than 0.05, indicating that, in the presence of the 'Prep Exams' variable, the hours studied does **not** have a statistically significant association with the exam score in this specific multiple linear regression model configuration.

In practical modeling terms, the lack of significance for 'Hours' might prompt us to consider simplifying the model by removing this variable and running a simpler linear regression using 'Prep Exams' as the sole predictor, thereby potentially improving model parsimony and reducing complexity.

Assessing Model Fit (R-squared)

Finally, we examine the R-squared (R2) value, a critical metric that measures the proportion of the

variance in the dependent variable (Exam Score) that is predictable from the independent variables (Hours and Prep Exams).

R-squared of model: **0.734**

An R-squared value of 0.734 signifies that **73.4%** of the total variation observed in the students' final exam scores can be effectively explained by the combined effect of the number of hours studied and the number of preparatory exams taken. This indicates a relatively strong fit for the predictive model.

Conclusion and Next Steps

This tutorial successfully demonstrated how to implement and interpret a multiple linear regression model using PySpark. We covered data assembly using **VectorAssembler**, model fitting, coefficient interpretation, and statistical testing using p-values and R-squared. Mastering these steps is crucial for leveraging the power of distributed computing for machine learning tasks.

Note: For a comprehensive overview of all available regression summary metrics, including RMSE, MAE, and explained variance, refer directly to the official PySpark ML documentation.

The following tutorials explain how to perform other common tasks in PySpark: