Mithali Manjunath – SEC01 (NUID 002879571)

Big Data System Engineering with Scala  
Fall 2024

Spark Assignment 2



**- GitHub Repo URL -** <https://github.com/mithali-m/Spark_Assignment/tree/main>

**- List of Tasks Implemented**

1. Load the dataset using Spark Step

2. Perform Exploratory Data Analysis on the titanic dataset Step

3. Perform Feature Engineering Step

4. Use the train.csv to train a Machine Learning model & test it on the test.csv. Predict if the records in test.csv survived or not. (1 = Survived, 0 = Dead)

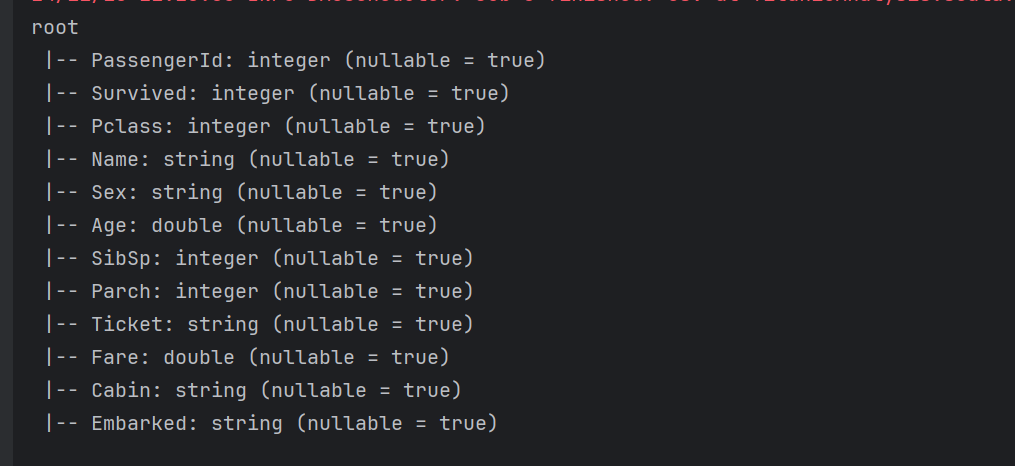
**- Code**

**TitanicAnalysis.scala**

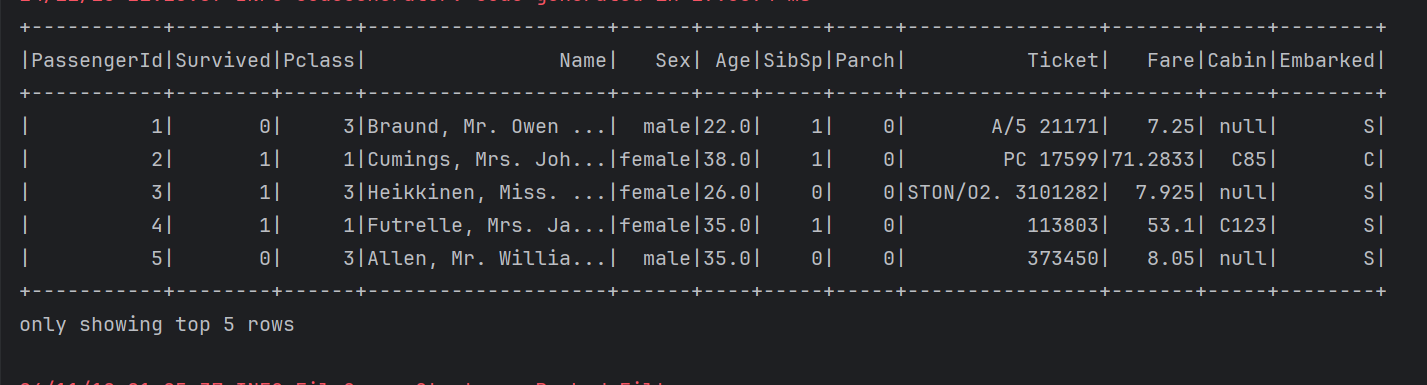
import org.apache.spark.sql.{SparkSession, DataFrame}  
import org.apache.spark.sql.functions.\_  
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}  
import org.apache.spark.ml.classification.LogisticRegression  
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator  
  
object TitanicAnalysis {  
  
 def main(args: Array[String]): Unit = {  
 val spark = SparkSession.*builder*()  
 .appName("Titanic Analysis")  
 .master("local[\*]")  
 .getOrCreate()  
  
 // Load the data  
 val trainData = *loadCSV*(spark, "src/titanic/train.csv")  
 val testData = *loadCSV*(spark, "src/titanic/test.csv")  
  
 // Display schema and first few rows  
 trainData.printSchema()  
 trainData.show(5)  
  
 // Exploratory Data Analysis (EDA)  
 // Check for missing values  
 trainData.select(trainData.columns.map(c => *sum*(*col*(c).isNull.cast("int")).alias(c)): \_\*).show()  
  
 // Summary statistics for 'Age' and 'Fare'  
 trainData.describe("Age", "Fare").show()  
  
 // Survival rate (distribution of 'Survived')  
 trainData.groupBy("Survived").count().show()  
  
 // Feature Engineering: Create new attributes and preprocess data  
 val preprocessedTrainData = *featureEngineering*(trainData, isTest = false)  
 val preprocessedTestData = *featureEngineering*(testData, isTest = true)  
  
 // Build and Train the Model  
 val model = *trainModel*(preprocessedTrainData)  
  
 // Make predictions on the test data  
 val predictions = model.transform(preprocessedTestData)  
  
 // Evaluate the model  
 *evaluateModel*(predictions)  
 }  
  
 def loadCSV(spark: SparkSession, path: String): DataFrame = {  
 spark.read.option("header", "true")  
 .option("inferSchema", "true")  
 .csv(path)  
 }  
  
 def featureEngineering(data: DataFrame, isTest: Boolean): DataFrame = {  
 // Check if 'Survived' exists in the train data and only retain it for training  
 if (!isTest && !data.columns.contains("Survived")) {  
 throw new Exception("The 'Survived' column is missing from the training dataset.")  
 }  
  
 // Handle missing values by filling 'Age' with 30, 'Embarked' with 'S', 'Fare' with 0  
 val dataWithNoMissingValues = data  
 .na.fill(Map("Age" -> 30, "Embarked" -> "S", "Fare" -> 0.0)) // Fill missing values  
  
 // Convert categorical columns to numerical using StringIndexer  
 val sexIndexer = new StringIndexer().setInputCol("Sex").setOutputCol("SexIndexed")  
 val embarkedIndexer = new StringIndexer().setInputCol("Embarked").setOutputCol("EmbarkedIndexed")  
 val pclassIndexer = new StringIndexer().setInputCol("Pclass").setOutputCol("PclassIndexed")  
  
 // Apply transformations to the dataset  
 val indexedData = sexIndexer.fit(dataWithNoMissingValues).transform(dataWithNoMissingValues)  
 val indexedData2 = embarkedIndexer.fit(indexedData).transform(indexedData)  
 val finalIndexedData = pclassIndexer.fit(indexedData2).transform(indexedData2)  
  
 // Feature Engineering: Assemble features into a vector (excluding 'Survived' here)  
 val assembler = new VectorAssembler()  
 .setInputCols(Array("PclassIndexed", "Age", "SibSp", "Parch", "Fare", "SexIndexed", "EmbarkedIndexed"))  
 .setOutputCol("features")  
 .setHandleInvalid("skip") // Skip invalid rows with null values  
  
 // Drop 'Survived' for test data, but retain it for training  
 if (isTest) {  
 assembler.transform(finalIndexedData)  
 } else {  
 assembler.transform(finalIndexedData).select("Survived", "features")  
 }  
 }  
  
 // Train a Logistic Regression model  
 def trainModel(data: DataFrame) = {  
 // Create and train a Logistic Regression model  
 val logisticRegression = new LogisticRegression()  
 .setLabelCol("Survived") // Target column (Survived)  
 .setFeaturesCol("features") // Features column  
  
 val model = logisticRegression.fit(data) // Use 'data' instead of 'train'  
 model  
 }  
  
 // Evaluate the model  
 def evaluateModel(predictions: DataFrame): Unit = {  
 // Configure the evaluator with the appropriate columns  
 val evaluator = new BinaryClassificationEvaluator()  
 .setLabelCol("Survived") // Correct label column for evaluation  
 .setRawPredictionCol("prediction")  
  
 // Evaluate the model  
 val accuracy = evaluator.evaluate(predictions)  
 *println*(s"Model Accuracy: **$**accuracy")  
 }  
}

**- Unit tests / Results**

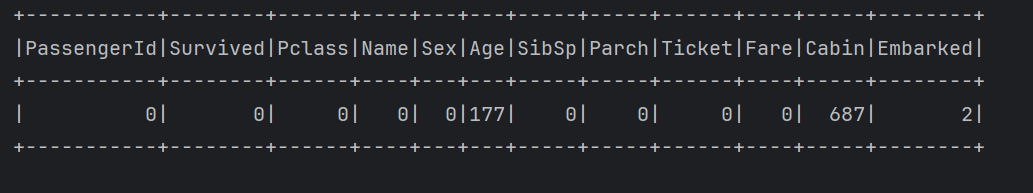
Displays schema of train data



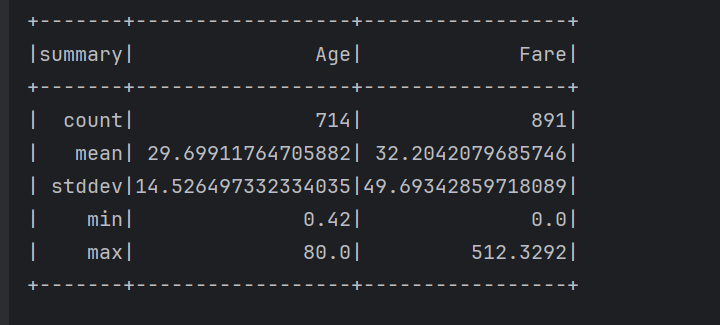
First five rows of train data



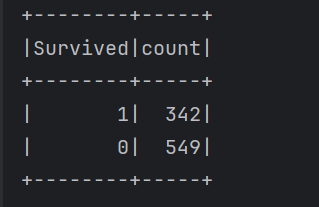
Checking missing values for each column in train data



Summary statistics of ‘Age’ and ‘Fare’



Survived column count



Logistic Regression Model’s Accuracy

