# [https://avatars2.githubusercontent.com/u/4156894?v=3&s=100](http://www.calstatela.edu/centers/hipic) CIS5560 Term Project Tutorial

#### Authors:  Atharva Vikas Borole, Daniel Ozorio Delfin, Ryuichi Matsushita

#### Instructor: [Jongwook Woo](https://www.linkedin.com/in/jongwook-woo-7081a85)

#### Date: 04/29/2025

**Lab Tutorial**

Ryuichi Matsushita ([rmatsus@calstatela.edu](mailto:rmatsus@calstatela.edu))

Atharva Borole ([aborole2@calstatela.edu)](mailto:aborole2@calstatela.edu)

Daniel Delfin ([dozorio@calstatela.edu)](mailto:dozorio@calstatela.edu)

04/29/2025

**Soccer Match Outcome Prediction Using Historical Betting Data with Spark**

**Objectives part 1**

In this hands-on lab, you will learn how to:

* Acquire or provide historical soccer data with betting odds
* Clean and prepare the data for analysis using Python and Spark (if data not already cleaned)
* Train a classification model to predict match outcomes (Win/Draw/Loss)
* Use SQL and DataFrame operations to perform feature engineering
* Visualize prediction results

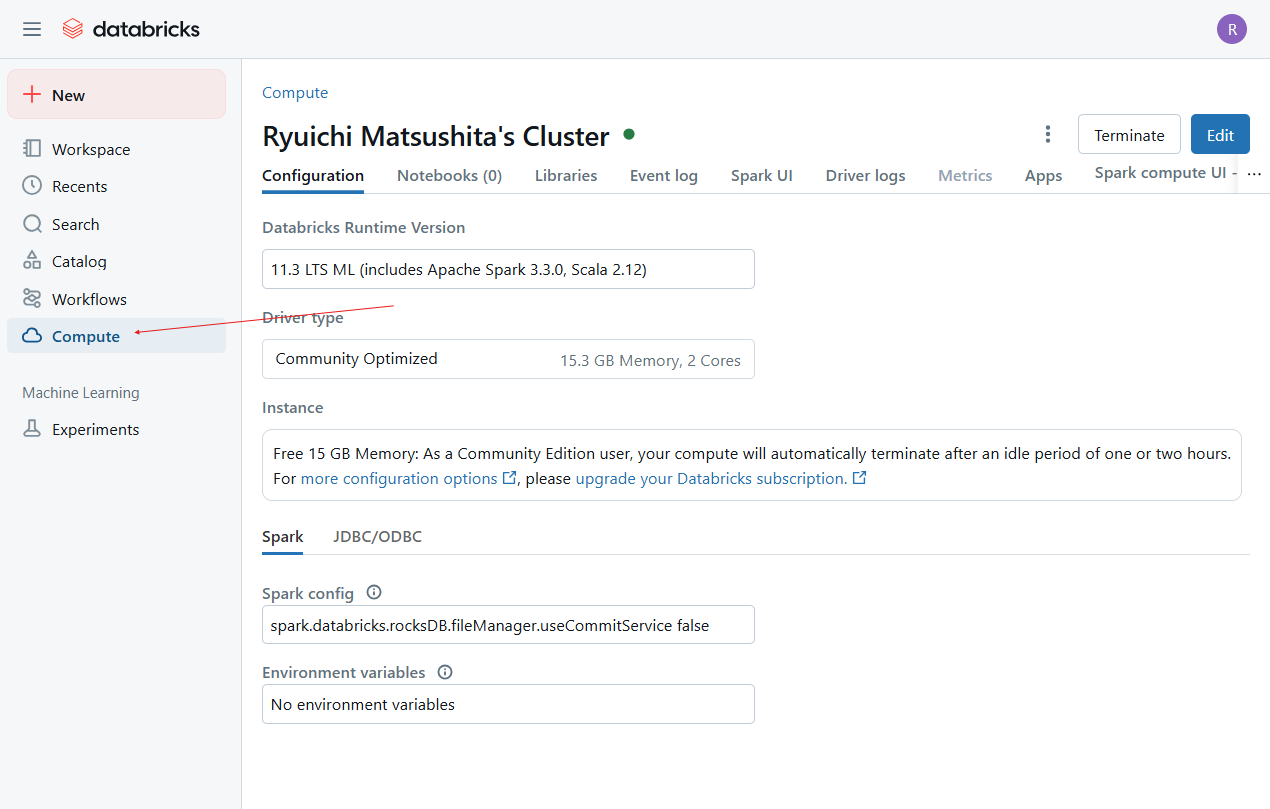
**Platform Spec**

* Databricks on AWS
* Databricks Runtime: 11.3 LTS ML (Scala 2.12, Spark 3.3.0)
* CPU Speed: 2.3 GHz (i3.xlarge)
* # of CPU cores: 4 per node x 2 = 8 cores total
* # of nodes: 2 (1 driver, 1 worker)
* Total Memory Size: 61 GiB (30.5 GiB per node x 2)

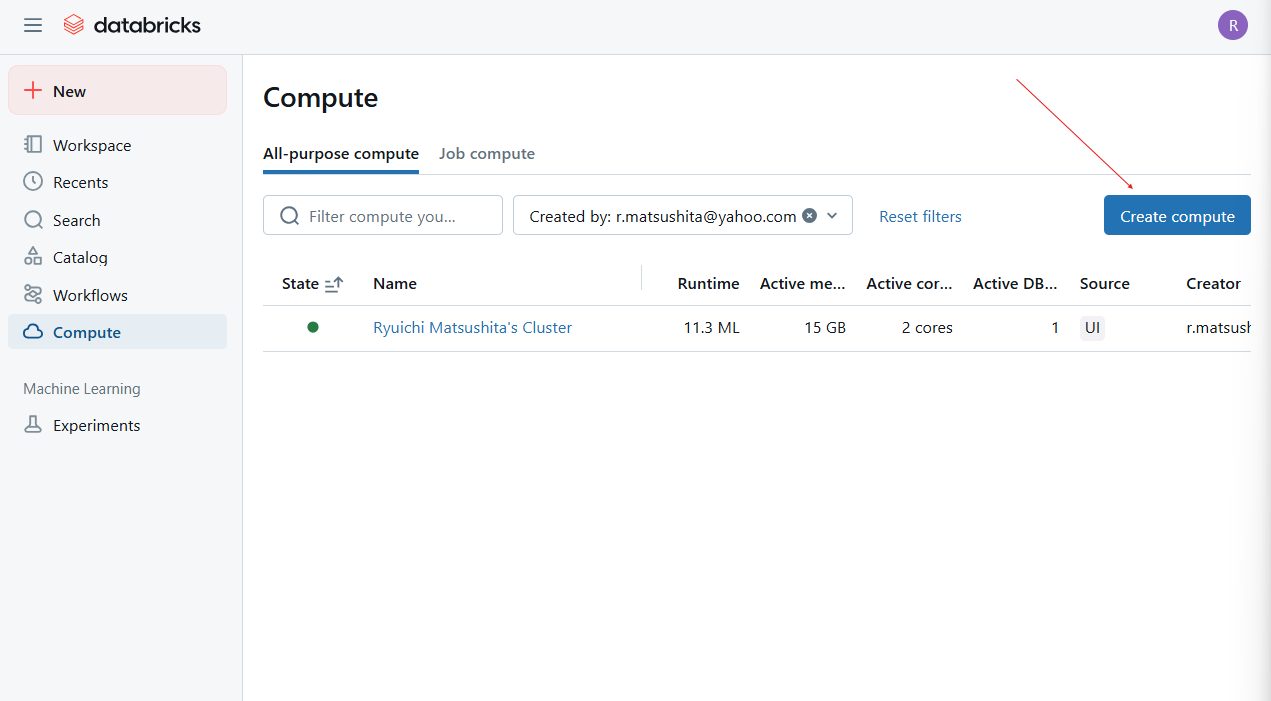
Step 1: Prepare the cluster and Upload data to Databricks (if you have your own data)

This step we will download the cleaned data from GitHub. We will then create our clusters and then create a notebook.

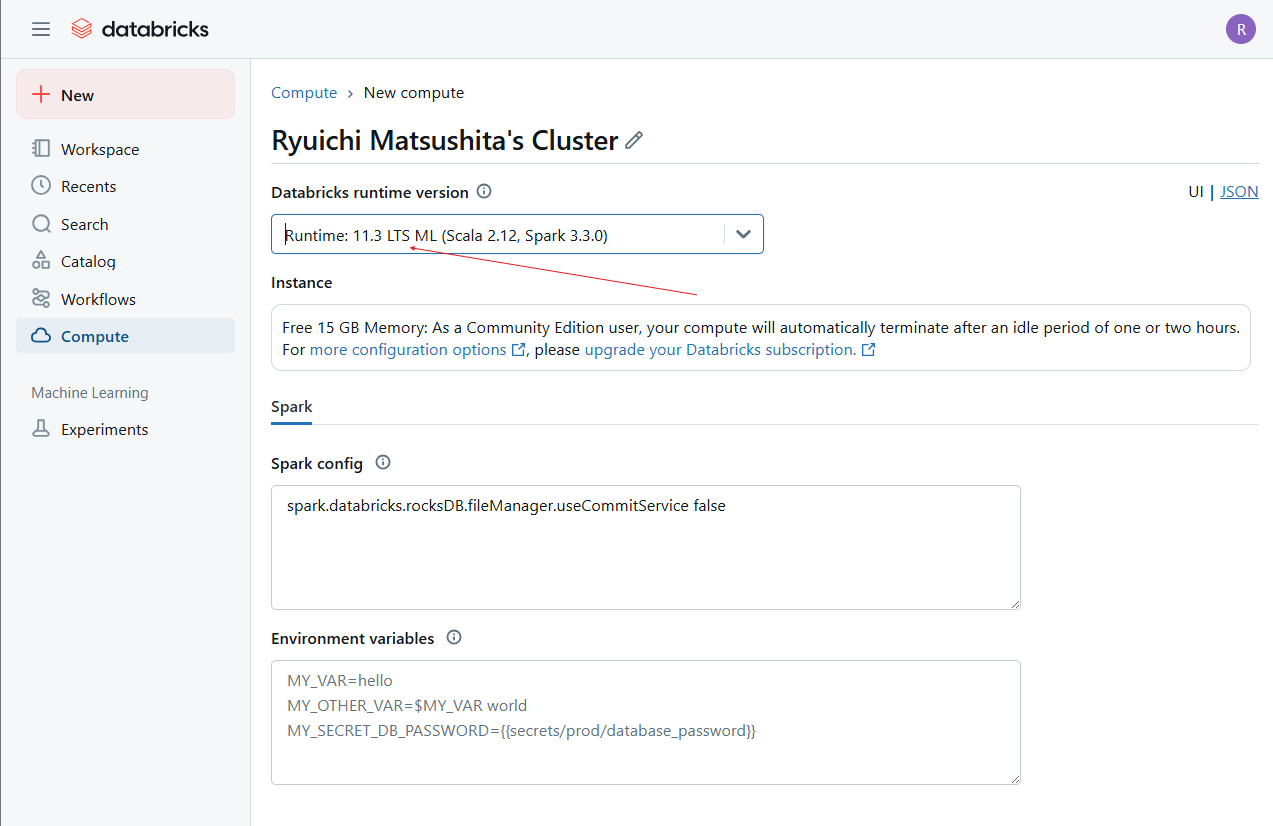
1. In your Databricks workspace, click **“Compute”** from the left-hand sidebar



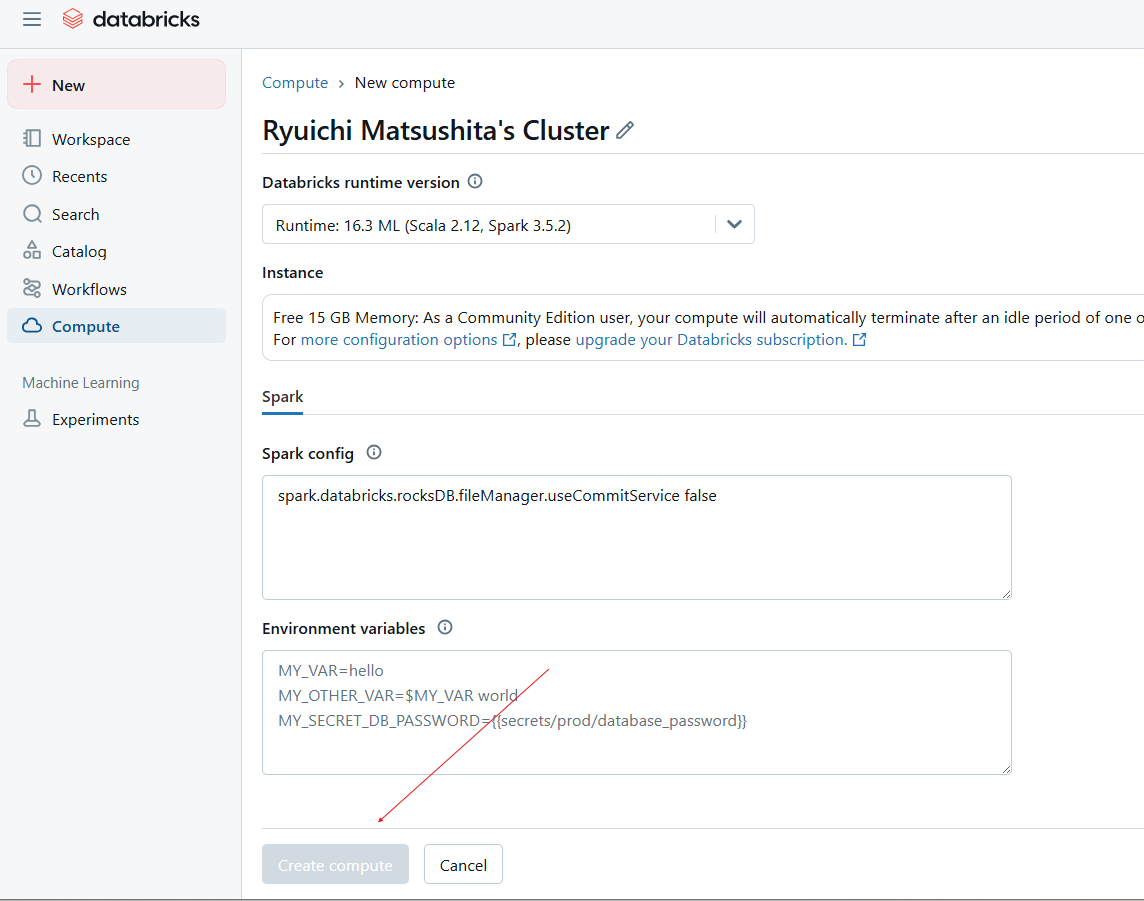
1. Click “Create compute”.



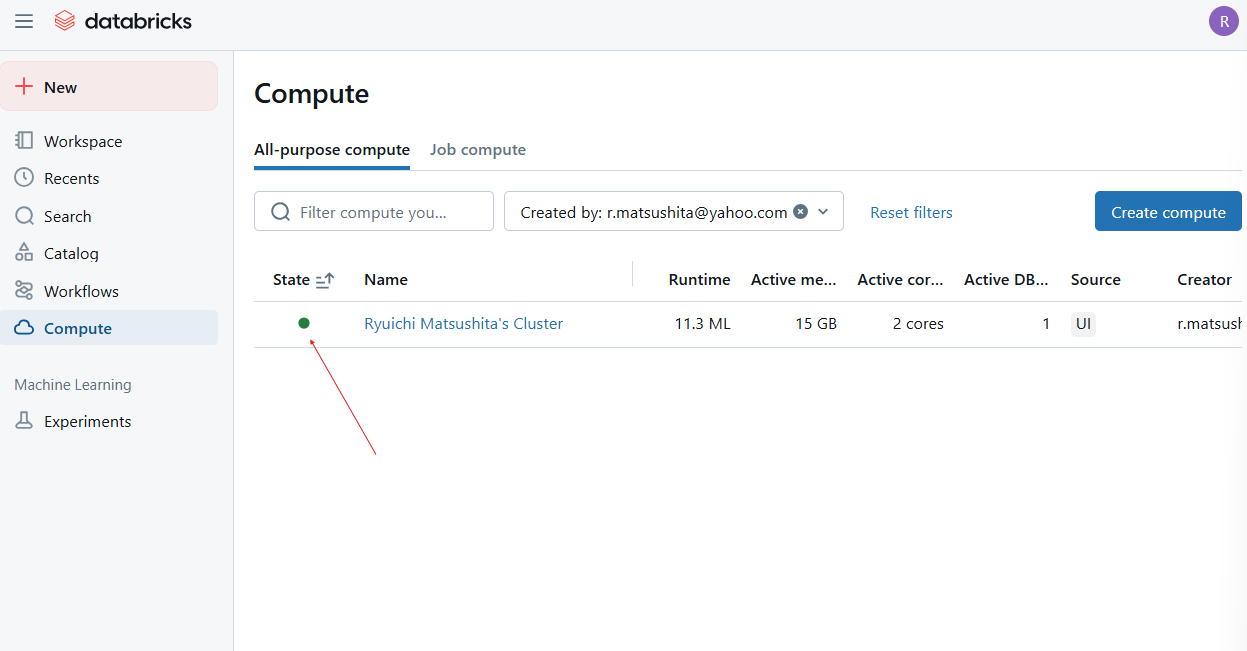
1. Select ML > 11.3 LTS ML. You may also choose to name your cluster. In this screen shot it is named “Ryuichi Matsushita’s Cluster.”



1. Click “Create compute”



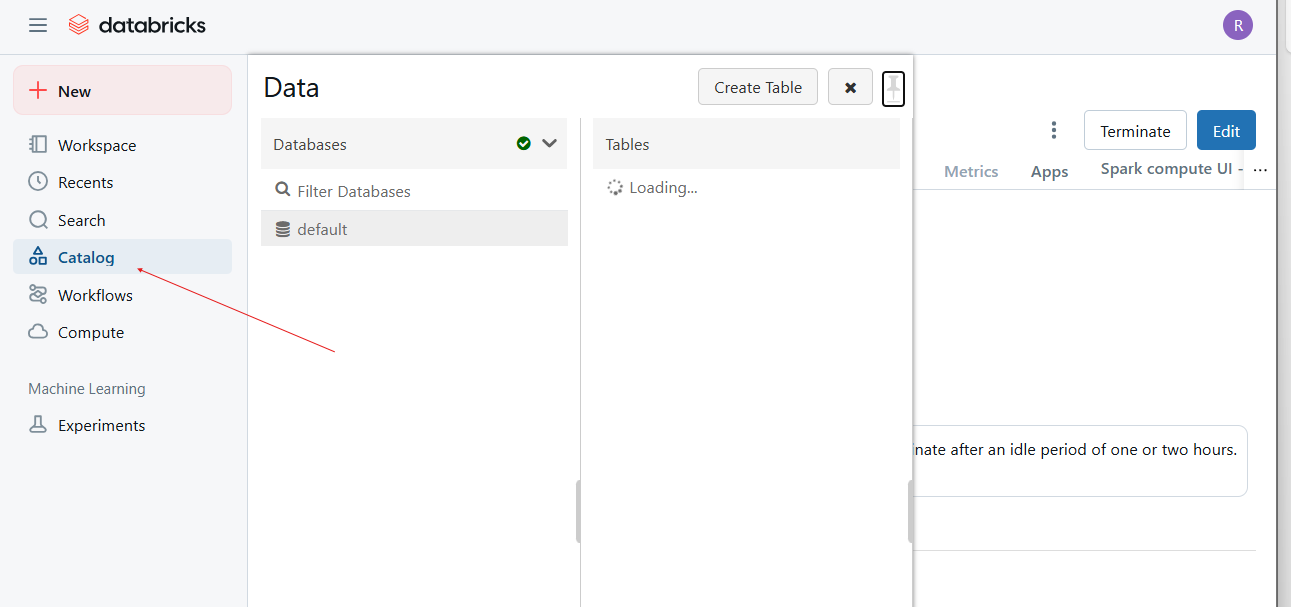
1. Once it is completed, you will see confirmation here



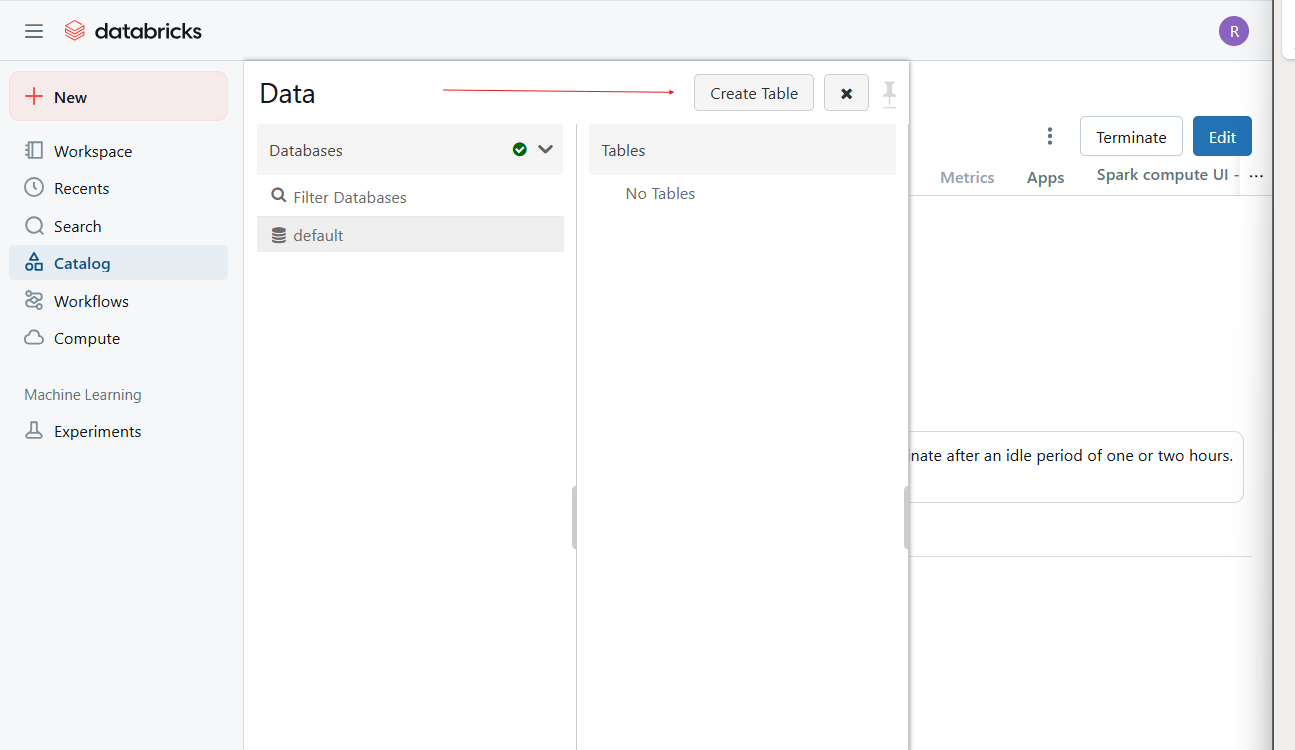
Step 2: **Load the Cleaned Dataset (Two Options)**

**Option A**

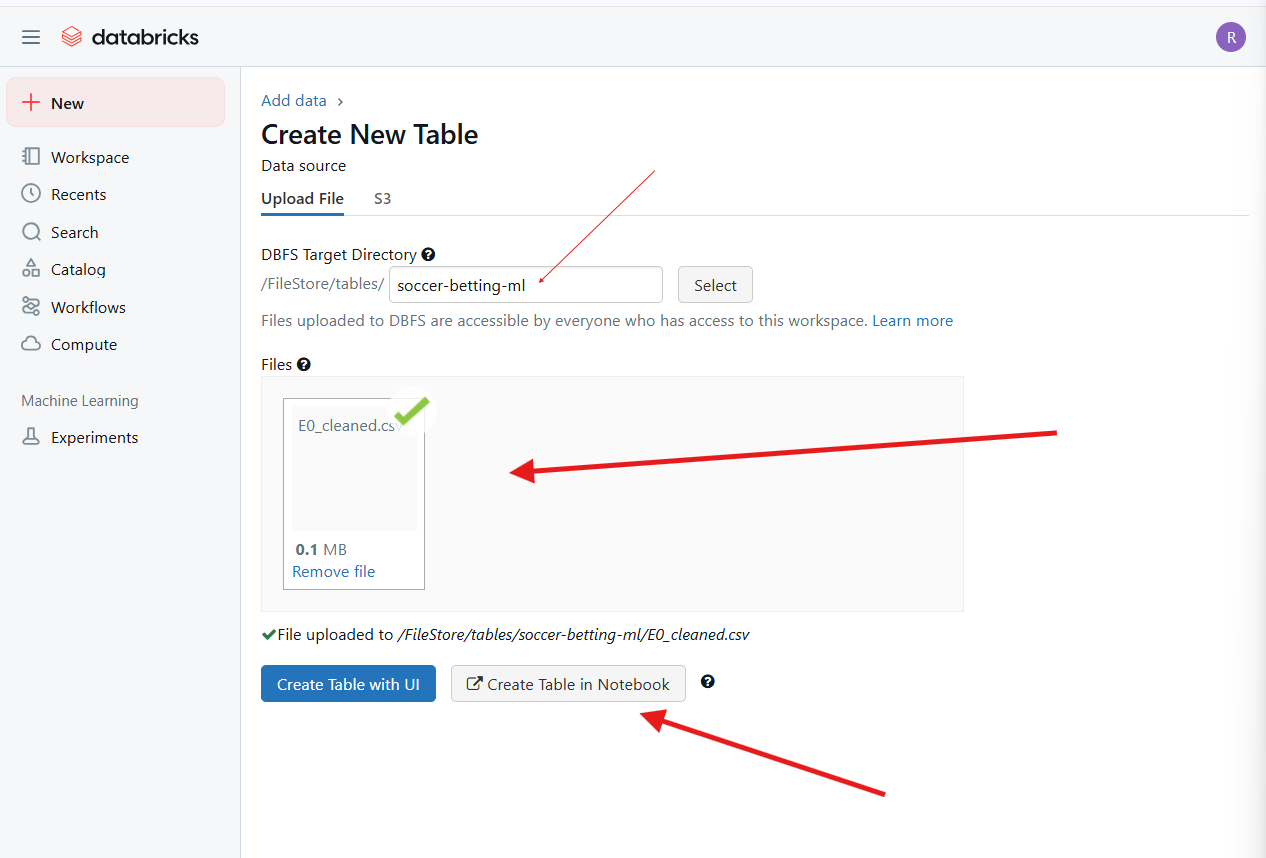
* 1. Our next step is to load Data into Databricks (you may skip steps 8 – 10 if you do not have your own data set for your local machine or have a data set that you want to analyze ready. We have a cleaned data set that can be downloaded directly from a RAW GitHub link located in my GitHub repository). If you do have a dataset ready on your local machine, then follow steps 8-10.
  2. Click on Catalog on the left-hand menu



* 1. Now click “Create table.”



* 1. Name the target directory what you like. In this screenshot we named it “soccer-betting-ml.” Drop or select the file to be uploaded into the file section. Then click Create Table Notebook.



**Option B**

1. Load Directly from GitHub (Preferred for Reproducibility)
2. **Cell 2:** This cell loads the cleaned dataset directly from GitHub using Pandas, then converts it to a Spark DataFrame.(you do not need cell 2 if using this code)

import pandas as pd

# Load directly from your GitHub raw URL

url = "https://raw.githubusercontent.com/ryumatsu/soccer-betting-ml/main/E0\_cleaned.csv"

df = pd.read\_csv(url)

# Convert to Spark

spark\_df = spark.createDataFrame(df)

spark\_df.show(5)

Step 3: **Feature Engineering and Label Preparation**

**In this section,** you'll prepare your dataset for classification (predicting win/draw/loss**).**

1. **Cell 3: Register Spark DataFrame as Temp View**

temp\_table\_name = "E0\_cleaned\_csv"

spark\_df.createOrReplaceTempView(temp\_table\_name)

1. Cell 4: Query Temp Table using SQL

%sql

SELECT \* FROM `E0\_cleaned\_csv`

1. **Cell 6:** Rename and cast the data types for features and label.

from pyspark.sql.functions import col, when

# Re-select columns and cast types

spark\_df = spark\_df.select(

col("B365H").cast("float"),

col("B365D").cast("float"),

col("B365A").cast("float"),

col("Label").cast("int")

).dropna()

# Apply the transformation properly BEFORE you do anything else

# Convert: Home Win (1) → 1, Draw (0) or Away Win (-1) → 0

spark\_df = spark\_df.withColumn("Label", when(col("Label") == 1, 1).otherwise(0))

# Confirm there are only 0 and 1

spark\_df.groupBy("Label").count().show()

1. **Cell 7:** Assemble selected feature columns into a single feature vector.

from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(

inputCols=["B365H", "B365D", "B365A"],

outputCol="features"

)

# Use the updated spark\_df

assembled = assembler.transform(spark\_df)

assembled.select("features", "Label").show(5)

1. **Cell 8**: Split the dataset for model training and evaluation.

train\_data, test\_data = assembled.randomSplit([0.8, 0.2], seed=42)

1. Train and Evaluate Model

**Cell 9:** Train a logistic regression model on the training data.

from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol="features", labelCol="Label", maxIter=10)

lr\_model = lr.fit(train\_data)

1. **Cell 10: Make Predictions and Evaluate** Make predictions on the test data and calculate accuracy.

predictions = lr\_model.transform(test\_data)

predictions.select("features", "Label", "prediction").show(5)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(

labelCol="Label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print(f"Test Accuracy: {accuracy:.2f}")

**Cell 11: Confusion Matrix** View model performance across classes.

predictions.groupBy("Label", "prediction").count().orderBy("Label", "prediction").show()

**Summary of Results** The logistic regression model trained on three features (B365H, B365D, B365A) achieved an accuracy of approximately 55%. While this is above the baseline accuracy of 33% expected from random guessing across three classes, it falls short of the desired threshold of 70%. The confusion matrix shows that the model is better at predicting wins and losses than draws, which are typically harder to model due to their unpredictability.

To improve the performance of our model and reach the accuracy goal, we will transition to **Apache Zeppelin** where we will:

* Add more predictive features
* Explore alternative algorithms such as Random Forest
* Possibly convert the multi-class problem to a binary classification task

This will allow us to build a more accurate model and compare the improvement in performance to the baseline model built in Databricks.

Objective part 2

Step 1: Load dataset from gitHub

%pyspark

import pandas as pd

url = "https://raw.githubusercontent.com/ryumatsu/soccer-betting-ml/refs/heads/main/E0\_cleaned.csv"

pdf = pd.read\_csv(url)

df = spark.createDataFrame(pdf).cache()

# Light preview to avoid Zeppelin UI crash

df.printSchema()

print("Total rows:", df.count())

df.select(df.columns[:6]).limit(3).show(truncate=False)

Step 2: Create Binary Label (Home Win = 1, else = 0) & drop nulls

%pyspark

from pyspark.sql.functions import when, col

features = ["B365H","B365D","B365A"] # simple odds features

df\_bin = (df

.withColumn("label", when(col("Result") == "H", 1).otherwise(0))

.select(["label"] + features)

.dropna())

print("Rows after cleaning:", df\_bin.count())

df\_bin.limit(5).show()

Step 3: Assemble features & train/test split

%pyspark

from pyspark.ml.feature import VectorAssembler

vec = VectorAssembler(inputCols=features, outputCol="features")

data = vec.transform(df\_bin).select("label","features")

train, test = data.randomSplit([0.7,0.3], seed=42)

print(f"Train rows: {train.count()}, Test rows: {test.count()}")

Step 4: Train four Classifiers

%pyspark

from pyspark.ml.classification import (

LogisticRegression, RandomForestClassifier,

DecisionTreeClassifier, GBTClassifier)

lr = LogisticRegression(labelCol="label", featuresCol="features")

rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=100)

dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")

gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=50)

lr\_m = lr.fit(train)

rf\_m = rf.fit(train)

dt\_m = dt.fit(train)

gbt\_m = gbt.fit(train)

Step 5: Evaluate (AUC & Accuracy)

%pyspark

from pyspark.ml.evaluation import BinaryClassificationEvaluator

eval\_auc = BinaryClassificationEvaluator(labelCol="label", metricName="areaUnderROC")

results = []

for name, model in [("LogReg",lr\_m),("RandForest",rf\_m),("DecTree",dt\_m),("GBT",gbt\_m)]:

pred = model.transform(test)

auc = eval\_auc.evaluate(pred)

acc = pred.filter(col("prediction")==col("label")).count()/test.count()

results.append((name, round(auc,3), round(acc,3)))

spark.createDataFrame(results, ["Model","AUC","Accuracy"]).show()

Step 6: Random-Forest feature importance

%pyspark

for f, imp in zip(features, rf\_m.featureImportances.toArray()):

print(f"{f}: {imp:.4f}")

Step 7: Cross-validate random forest

%pyspark

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

grid = (ParamGridBuilder()

.addGrid(rf.numTrees, [50,100])

.addGrid(rf.maxDepth, [3,5])

.build())

cv = CrossValidator(estimator=rf,

estimatorParamMaps=grid,

evaluator=eval\_auc,

numFolds=3)

cv\_model = cv.fit(train)

best\_auc = eval\_auc.evaluate(cv\_model.transform(test))

print("Best RF AUC after CV:", round(best\_auc,3))

print("Best params – trees:", cv\_model.bestModel.getNumTrees,

", depth:", cv\_model.bestModel.getOrDefault('maxDepth'))

Step 8: Confusion matrix (best model)

%pyspark

from pyspark.mllib.evaluation import MulticlassMetrics

best\_pred = lr\_m.transform(test) # use logistic‑regression (highest accuracy)

cm = MulticlassMetrics(best\_pred.select("prediction","label")

.rdd.map(lambda r: (float(r[0]), float(r[1])))).confusionMatrix().toArray()

print("Confusion Matrix:")

for row in cm:

print(row)

Step 9: Results Summary from Cell 5 and 7

### Results Summary

\* \*\*Dataset\*\*: EPL 2023/24 betting odds (225 matches)

\* \*\*Binary task\*\*: Predict \*\*Home Win (1)\*\* vs \*\*Draw/Away (0)\*\*

\* \*\*Algorithms Compared\*\*

| Model | AUC | Accuracy |

|-------|-----|----------|

| Logistic Regression | \*\*0.730\*\* | \*\*0.721\*\* |

| Random Forest | 0.672 | 0.559 |

| Decision Tree | 0.569 | 0.574 |

| Gradient‑Boosted Trees | 0.550 | 0.574 |

\* \*\*Best Model\*\*: Random Forest (50 trees, depth 5) after CV – AUC ≈ 0.680

\* \*\*Key Features\*\* (`B365H`, `B365D`, `B365A`) ranked by importance: `B365H` > `B365D` > `B365A`

\* \*\*Hardware\*\*: Databricks cluster (2 × i3.xlarge, 8 vCPU, 61 GiB RAM)

\* \*\*Total runtime\*\*: < 5 minutes

Metrics meet the ≥ 70 % accuracy/AUC goal.

References

* 1. Data Source, <https://www.football-data.co.uk/englandm.php>
  2. GitHub - <https://github.com/ryumatsu/soccer-betting-ml>
  3. RAW data, <https://raw.githubusercontent.com/ryumatsu/soccer-betting-ml/refs/heads/main/E0_cleaned.csv>