Predicția bolilor de inimă folosind PySpark, MLlib și TensorFlow

Mihai Andrei-Alexandru, *Big Data Project*, *iunie 2025*

Cuprins

---

1. Introducere

Prezentarea succintă a setului de date

Setul de date utilizat este **Heart Failure Prediction**, disponibil pe Kaggle:
https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction

Setul de date conține 918 înregistrări despre pacienți, fiecare cu 12 caracteristici medicale și o variabilă țintă `HeartDisease` care indică dacă pacientul **are sau nu boală cardiovasculară**.

Obiective

Obiectivul proiectului este:

- **procesarea**, **curățarea** și **analizarea** setului de date folosind *Spark*;
- **antrenarea** și **evaluarea** a două modele *ML*;
- **aplicarea** unei metode de *DL* cu *TensorFlow*;
- **implementarea** unui *pipeline* și *UDF*;
- **integrarea** unui *flux de date în timp real* cu *Spark Streaming*.

---

2. Procesarea datelor cu Spark

Vom folosi *PySpark* pentru a **analiza** și **prelucra** datele. Se vor aplica **agregări** și **transformări** folosind atât *DataFrame API* cât și *Spark SQL*.

```
from pyspark.sql import SparkSession

spark =
SparkSession.builder.appName("HeartFailurePrediction").getOrCreate()

# Citirea setului de date
df = spark.read.option("header", True).option("inferSchema",
True).csv("heart.csv")
df.printSchema()
df.show(5)
```

```
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-
defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
25/06/15 09:56:30 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where
applicable

root
 |-- Age: integer (nullable = true)
 |-- Sex: string (nullable = true)
 |-- ChestPainType: string (nullable = true)
 |-- RestingBP: integer (nullable = true)
 |-- Cholesterol: integer (nullable = true)
 |-- FastingBS: integer (nullable = true)
 |-- RestingECG: string (nullable = true)
 |-- MaxHR: integer (nullable = true)
 |-- ExerciseAngina: string (nullable = true)
 |-- Oldpeak: double (nullable = true)
 |-- ST_Slope: string (nullable = true)
 |-- HeartDisease: integer (nullable = true)

+---+---+-------------+---------+-----------+---------+----------
+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|
MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------
+-----+--------------+-------+--------+------------+
```

```
| 40|  M|          ATA|      140|        289|        0|    Normal|
172|             N|    0.0|      Up|           0|
| 49|  F|          NAP|      160|        180|        0|    Normal|
156|             N|    1.0|    Flat|           1|
| 37|  M|          ATA|      130|        283|        0|        ST|
98|             N|    0.0|      Up|           0|
| 48|  F|          ASY|      138|        214|        0|    Normal|
108|             Y|    1.5|    Flat|           1|
| 54|  M|          NAP|      150|        195|        0|    Normal|
122|             N|    0.0|      Up|           0|
+---+---+-------------+---------+-----------+---------+----------
+-----+--------------+-------+--------+------------+
only showing top 5 rows
```

```
# Statistici descriptive
df.describe().show()

# Distribuția țintei (HeartDisease)
df.groupBy("HeartDisease").count().show()
```

```
25/06/15 09:56:38 WARN SparkStringUtils: Truncated the string
representation of a plan since it was too large. This behavior can be
adjusted by setting 'spark.sql.debug.maxToStringFields'.


+-------+------------------+----+-------------+------------------
+------------------+------------------+----------+------------------
+--------------+------------------+--------+-------------------+
|summary|               Age| Sex|ChestPainType|         RestingBP|
Cholesterol|         FastingBS|RestingECG|             MaxHR|
ExerciseAngina|           Oldpeak|ST_Slope|       HeartDisease|
+-------+------------------+----+-------------+------------------
+------------------+------------------+----------+------------------
+--------------+------------------+--------+-------------------+
|  count|               918| 918|          918|               918|
918|               918|       918|               918|           918|
918|     918|                 918|
|   mean|53.510893246187365|NULL|         NULL|132.39651416122004|
198.7995642701525|0.2331154684095860 5|      NULL|136.80936819172112|
NULL|0.8873638344226581|    NULL| 0.5533769063180828|
| stddev|  9.43261650673202|NULL|         NULL|18.514154119907808|
109.38414455220345|0.42304562473930296|      NULL| 25.46033413825029|
NULL|1.0665701510493264|    NULL|0.49741373828459706|
|    min|                28|   F|          ASY|                 0|
0|                 0|       LVH|                60|             N|
-2.6|    Down|                   0|
|    max|                77|   M|           TA|               200|
603|                 1|        ST|               202|             Y|
6.2|      Up|                   1|
+-------+------------------+----+-------------+------------------
```

```
+-----------------+------------------+----------+-----------------
+--------------+------------------+--------+------------------+

+------------+-----+
|HeartDisease|count|
+------------+-----+
|           1|  508|
|           0|  410|
+------------+-----+
```

Curățare și transformare date

Verificăm *valori lipsă* și **realizăm transformări** simple: *conversia la lowercase*, *eliminarea duplicatelor*.

```
# Verificare valori lipsă
from pyspark.sql.functions import col, isnan, when, count, trim

# Pentru fiecare coloana din set-ul nostru de date, numaram valorile
egale cu NULL sau stringurile vide.
df.select([
    count(
        when(
            col(c).isNull() | (trim(col(c)) == ""), c
        )
    ).alias(c)
    for c in df.columns
]).show()

# Eliminare duplicate (dacă există)
df = df.dropDuplicates()
```

```
+---+---+-------------+---------+-----------+---------+----------
+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|
MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------
+-----+--------------+-------+--------+------------+
|  0|  0|            0|        0|          0|        0|         0|
0|             0|      0|       0|           0|
+---+---+-------------+---------+-----------+---------+----------
+-----+--------------+-------+--------+------------+
```

Din fericire set-ul de date pe care l-am ales **nu a continut** *NULL* sau *stringuri vide*, dar acest lucru **nu este valabil** pentru orice dataset, deci acesta prelucrare **este obligatorie**.

Grupări și agregări cu DataFrame API

Agregare: procentul bolnavilor pe gen:

```
from pyspark.sql.functions import round, avg

df.groupBy("Sex") \
  .agg(round(avg("HeartDisease") * 100,
2).alias("HeartDiseaseRatePercent")) \
  .show()
```

```
+---+-----------------------+
|Sex|HeartDiseaseRatePercent|
+---+-----------------------+
|  F|                  25.91|
|  M|                  63.17|
+---+-----------------------+
```

Grupare pe categorii de vârstă:

```
from pyspark.sql.functions import when, col

df = df.withColumn("AgeGroup", when(col("Age") < 40, "<40")
                               .when((col("Age") >= 40) & (col("Age") <
60), "40-59")
                               .otherwise("60+"))

df.groupBy("AgeGroup") \
  .agg(round(avg("HeartDisease") * 100,
2).alias("HeartDiseaseRatePercent")) \
  .orderBy("AgeGroup") \
  .show()
```

```
+--------+-----------------------+
|AgeGroup|HeartDiseaseRatePercent|
+--------+-----------------------+
|   40-59|                  50.77|
|     60+|                  73.12|
|     <40|                   32.5|
+--------+-----------------------+
```

Interogări cu Spark SQL

Rata bolii în funcție de *glicemie* (**FastingBS**) și *tensiune arterială* (**RestingBP**):

```
df = df.withColumn("HighBP", when(col("RestingBP") >= 130,
1).otherwise(0))
```

```
df.createOrReplaceTempView("heart")

spark.sql("""
    SELECT
       FastingBS AS Diabetic,
       HighBP,
       COUNT(*) AS Total,
       ROUND(AVG(HeartDisease) * 100, 2) AS HeartDiseaseRatePercent
    FROM heart
    GROUP BY FastingBS, HighBP
    ORDER BY HeartDiseaseRatePercent DESC
""").show()
```

```
+--------+------+-----+----------------------+
|Diabetic|HighBP|Total|HeartDiseaseRatePercent|
+--------+------+-----+----------------------+
|       1|     0|   80|                  80.0|
|       1|     1|  134|                  79.1|
|       0|     1|  409|                 50.86|
|       0|     0|  295|                 44.07|
+--------+------+-----+----------------------+
```

---

3. Metode de Machine Learning

Vom **construi** și **evalua** *două modele* de *clasificare binară* pentru a **prezice** apariția *bolii cardiace* (`HeartDisease`):

1. *Logistic Regression*
2. *Random Forest Classifier*

Scopul este să comparăm performanța celor două metode.

```
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml import Pipeline

# Selectăm coloanele numerice
feature_cols = ['Age', 'RestingBP', 'Cholesterol', 'FastingBS',
'MaxHR', 'Oldpeak']

# Transformăm și coloanele categorice în numeric (dacă există)
categorical_cols = ['Sex', 'ChestPainType', 'RestingECG',
'ExerciseAngina', 'ST_Slope']
indexers = [StringIndexer(inputCol=col, outputCol=col+"_idx") for col
in categorical_cols]
```

```
# Asamblare vector caracteristici
assembler = VectorAssembler(
    inputCols=feature_cols + [col + "_idx" for col in
categorical_cols],
    outputCol="features"
)

# Etichetă (HeartDisease este deja 0/1 dar asigurăm consistența)
label_indexer = StringIndexer(inputCol="HeartDisease",
outputCol="label")

# Pipeline de preprocesare
pipeline = Pipeline(stages=indexers + [assembler, label_indexer])
data = pipeline.fit(df).transform(df).select("features", "label")
data.show(5, truncate=False)
```

```
+-----------------------------------------------------+-----+
|features                                             |label|
+-----------------------------------------------------+-----+
|(11,[0,1,2,4,7,10],[40.0,140.0,289.0,172.0,2.0,1.0])|1.0  |
|[49.0,160.0,180.0,0.0,156.0,1.0,1.0,1.0,0.0,0.0,0.0]|0.0  |
|[37.0,130.0,283.0,0.0,98.0,0.0,0.0,2.0,2.0,0.0,1.0] |1.0  |
|[48.0,138.0,214.0,0.0,108.0,1.5,1.0,0.0,0.0,1.0,0.0]|0.0  |
|(11,[0,1,2,4,7,10],[54.0,150.0,195.0,122.0,1.0,1.0])|1.0  |
+-----------------------------------------------------+-----+
only showing top 5 rows
```

Împărțirea datelor

Vom împărți setul de date în două subseturi: antrenare (70%) și test (30%).

```
train_data, test_data = data.randomSplit([0.7, 0.3], seed=42)
print(f"Train: {train_data.count()}, Test: {test_data.count()}")
```

```
Train: 681, Test: 237
```

Logistic Regression

- Vrem să **prezicem** dacă o persoană are *boală cardiacă* (*HeartDisease = 1*) pe baza unor *caracteristici clinice*.
- **Logistic Regression** este o alegere bună pentru *clasificare binară*, deoarece este *interpretabil*, *rapid de antrenat* și oferă o primă linie de bază.
- **Aplicăm** *modelul* și **evaluăm** *acuratețea* pe *setul de testare*.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

```
lr = LogisticRegression(featuresCol="features", labelCol="label")
lr_model = lr.fit(train_data)
lr_predictions = lr_model.transform(test_data)

evaluator = BinaryClassificationEvaluator(labelCol="label",
metricName="areaUnderROC")
lr_auc = evaluator.evaluate(lr_predictions)
print(f"AUC (Logistic Regression): {lr_auc:.4f}")
```

```
AUC (Logistic Regression): 0.8969
```

Random Forest Classifier

La fel ca mai sus, dorim să facem clasificare binară.

Random Forest este un model de tip "ensemble" ce poate surprinde relații non-liniare între variabile. Este mai robust decât Logistic Regression, dar mai lent.

Antrenăm un model RF și comparăm performanța sa cu Logistic Regression.

```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol="features", labelCol="label",
numTrees=100, seed=42)
rf_model = rf.fit(train_data)
rf_predictions = rf_model.transform(test_data)

rf_auc = evaluator.evaluate(rf_predictions)
print(f"AUC (Random Forest): {rf_auc:.4f}")
```

```
AUC (Random Forest): 0.9155
```

Concluzii ML

Pentru ambele modele am folosit *AUC* - (**Area Under Curve**).

Aceasta este o *metrică standard* folosită pentru *evaluarea performanței* unui model de *clasificare binară*, în special când clasele sunt *dezechilibrate*.

Model	AUC
Logistic Regression	*~ 0.8969*
Random Forest	*~ 0.9155*

Random Forest a oferit performanță mai bună datorită capacității sale de a **modela relații complexe** între trăsături. Totuși, *Logistic Regression* este **util** ca *model de bază* și pentru *interpretabilitate*.

---

4. Utilizarea unui Data Pipeline complet

Vom **construi** un *Pipeline* care include:

- **Preprocesare**: *indexare categorice* + *VectorAssembler*
- **Model ML**: *Logistic Regression*

Ne dorim să automatizăm întreaga secvență de transformări și învățare automată.

Etape:

1. **Indexare categorică**: Coloanele *string* sunt transformate în coloane numerice folosind *StringIndexer*.
2. **VectorAssembler**: Toate coloanele numerice și cele indexate sunt combinate într-un singur vector de trăsături (*features*) necesar pentru modele ML.
3. **Label Indexing**: Coloana țintă *HeartDisease* este transformată în eticheta *label*.
4. **Modelul**: Se utilizează *LogisticRegression* pentru clasificare binară.
5. **Pipeline**: Toți pașii anteriori sunt legați într-un *Pipeline* unitar ce permite aplicarea secvențială a etapelor.
6. **Antrenare & Predicție**: *fit()* antrenează pipeline-ul pe date, *transform()* aplică modelul pe același set.
7. **Evaluare**: Se utilizează *BinaryClassificationEvaluator* cu metrica *areaUnderROC* pentru a evalua performanța modelului.

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression

# Refacem pipeline-ul complet: indexeri + assembler + model
indexers = [StringIndexer(inputCol=col, outputCol=col+"_idx") for col
in categorical_cols]
assembler = VectorAssembler(
    inputCols=feature_cols + [col + "_idx" for col in
categorical_cols],
    outputCol="features"
)
label_indexer = StringIndexer(inputCol="HeartDisease",
outputCol="label")

# Logistic Regression model
lr = LogisticRegression(featuresCol="features", labelCol="label")
```

```
# Pipeline complet
full_pipeline = Pipeline(stages=indexers + [assembler, label_indexer,
lr])

# Antrenare + predicție
pipeline_model = full_pipeline.fit(df)
predictions = pipeline_model.transform(df)

# Evaluare rapidă
evaluator = BinaryClassificationEvaluator(labelCol="label",
metricName="areaUnderROC")
pipeline_auc = evaluator.evaluate(predictions)
print(f"AUC cu pipeline complet (Logistic Regression):
{pipeline_auc:.4f}")
```

```
25/06/15 09:57:34 WARN InstanceBuilder: Failed to load implementation
from:dev.ludovic.netlib.blas.JNIBLAS

AUC cu pipeline complet (Logistic Regression): 0.9194
```

---

5. Utilizarea unei funcții definite de utilizator (UDF)

Voi crea o funcție care calculează un scor de risc personalizat în funcție de vârstă și colesterol, urmând să adaug acest scor ca o coloană nouă pentru analiză sau input în model.

```
from pyspark.sql.functions import udf
from pyspark.sql.types import DoubleType

# Funcție de calcul risc: rudimentară pentru demonstrație
def risk_score(age, cholesterol):
    score = 0.5 * age + 0.5 * (cholesterol if cholesterol else 0)
    return float(score / 200)  # normalizare

risk_udf = udf(risk_score, DoubleType())

# Aplicare UDF
df_with_risk = df.withColumn("RiskScore", risk_udf(col("Age"),
col("Cholesterol")))
df_with_risk.select("Age", "Cholesterol", "RiskScore").show(5)
```

```
+---+-----------+---------+
|Age|Cholesterol|RiskScore|
+---+-----------+---------+
```

```
| 40|         289|   0.8225|
| 49|         180|   0.5725|
| 37|         283|      0.8|
| 48|         214|    0.655|
| 54|         195|   0.6225|
+---+-----------+---------+
only showing top 5 rows
```

Optimizarea hiperparametrilor (Grid Search)

Voi **aplica** un *Grid Search* și *Cross Validation* pentru *Logistic Regression*.

Îmi doresc să optimizez hiperparametrii, astfel, trebuie să găsesc valoarea optimă pentru *regParam* și *elasticNetParam*.

```
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Pipeline de bază (doar cu Logistic Regression)
pipeline_lr = Pipeline(stages=indexers + [assembler, label_indexer,
lr])

# Grid de parametri
param_grid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.01, 0.1, 0.5]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()

# Cross-validation
crossval = CrossValidator(estimator=pipeline_lr,
                          estimatorParamMaps=param_grid,
                          evaluator=evaluator,
                          numFolds=3)

# Execută căutarea
cv_model = crossval.fit(df)
best_model = cv_model.bestModel

# Afișăm AUC pe modelul optimizat
cv_auc = evaluator.evaluate(best_model.transform(df))
print(f"AUC cu Logistic Regression + param tuning: {cv_auc:.4f}")
```

```
25/06/15 10:13:50 WARN CacheManager: Asked to cache already cached
data.
25/06/15 10:13:50 WARN CacheManager: Asked to cache already cached
data.

AUC cu Logistic Regression + param tuning: 0.9195
```

```
Bad pipe message: %s [b'"Google Chrome";v="137", "Chromium";v="137",
"Not/A)Brand']
Bad pipe message: %s [b'ol: max-age=0\r\nsec-ch-ua: "Google
Chrome";v="137", "Chromium";v="137", "Not/A)Brand";v="24"\r\nsec-ch-
ua-mobile: ?0\r\n']
Bad pipe message: %s [b'c-ch-ua-platform: "Windows"\r\nUpgrade-
Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) A', b'leWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0
Safari/537.36\r\nAccept: text/html,application/xhtml+xml,app']
Bad pipe message: %s
[b'cation/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.7\r\nSec-Fet', b'-Site: none\r\nSec-
Fetch-Mode: navigate\r\nSec-Fetch-User: ?1\r\nSec-Fetch-Dest:
document\r\nAccept-Encodi']
Bad pipe message: %s [b'ol: max-age=0\r\nsec-ch-ua: "Google
Chrome";v="137", "Chromium";v="137", "Not/A)Brand";v="24"\r\nsec-ch-
ua-mobile: ?0\r\n']
Bad pipe message: %s [b'c-ch-ua-platform: "Windows"\r\nUpgrade-
Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) A', b'leWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0
Safari/537.36\r\nAccept: text/html,application/xhtml+xml,app']
Bad pipe message: %s
[b'cation/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.7\r\nSec-Fet', b'-Site: cross-site\r\
nSec-Fetch-Mode: navigate\r\nSec-Fetch-User: ?1\r\nSec-Fetch-Dest:
document\r\nAccept-']
Bad pipe message: %s [b'coding: gzip, deflate, br, zstd\r\nAccept-
Language: en-US,en;q=0.9,ro;']
Bad pipe message: %s [b'ol: max-age=0\r\nsec-ch-ua: "Google
Chrome";v="137", "Chromium";v="137", "Not/A)Brand";v="24"\r\nsec-ch-
ua-mobile: ?0\r\n']
Bad pipe message: %s [b'c-ch-ua-platform: "Windows"\r\nUpgrade-
Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) A', b'leWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0
Safari/537.36\r\nAccept: text/html,application/xhtml+xml,app']
Bad pipe message: %s
[b'cation/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.7\r\nSec-Fet', b'-Site: cross-site\r\
nSec-Fetch-Mode: navigate\r\nSec-Fetch-Dest: document\r\nAccept-
Encoding: gzip, defl']
```

Concluzii

Am **integrat** într-un *pipeline complet* pașii de *preprocesare*, *antrenare* și *evaluare*. Am **utilizat**:

- *UDF* pentru **definirea** unei *logici personalizate de scor de risc*
- *Grid search* și *cross validation* pentru *optimizarea hiperparametrilor*

---

6. Deep Learning cu TensorFlow

Dorim să construim un model de clasificare binară care prezice apariția unei boli cardiace (*HeartDisease = 0 sau 1*) folosind rețele neuronale.

Rețelele neuronale artificiale pot capta relații complexe, neliniare între variabile, mai ales când există mai multe atribute implicate. În comparație cu Logistic Regression sau Random Forest, DL oferă flexibilitate mai mare pentru modelarea relațiilor nelineare.

Vom folosi TensorFlow + Keras pentru a antrena o rețea neuronală simplă cu:

- 2 straturi ascunse (dense)
- Funcția de activare *ReLU*
- Funcția de pierdere *binary_crossentropy*
- Optimizator *Adam*

```
import pandas as pd
from pyspark.sql.functions import col

# Salvăm datele Spark într-un Pandas DataFrame pentru TensorFlow
pandas_df = df.select(
    "Age", "Sex", "ChestPainType", "RestingBP", "Cholesterol",
    "FastingBS", "RestingECG", "MaxHR", "ExerciseAngina",
    "Oldpeak", "ST_Slope", "HeartDisease"
).toPandas()
```

Preprocesare date pentru TensorFlow

Vom converti datele categorice în one-hot encoding și vom scala datele numerice.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Separăm features și label
X = pandas_df.drop("HeartDisease", axis=1)
y = pandas_df["HeartDisease"]

# Coloane categorice și numerice
cat_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina",
"ST_Slope"]
num_cols = ["Age", "RestingBP", "Cholesterol", "FastingBS", "MaxHR",
"Oldpeak"]
```

```
# Pipeline de transformare
preprocessor = ColumnTransformer([
    ("num", StandardScaler(), num_cols),
    ("cat", OneHotEncoder(), cat_cols)
])

# Aplicăm transformările
X_processed = preprocessor.fit_transform(X)

# Împărțire în train/test
X_train, X_test, y_train, y_test = train_test_split(X_processed, y,
test_size=0.2, random_state=42)
```

Construirea modelului TensorFlow

Model secvențial cu 2 straturi ascunse și un strat final cu sigmoid.

```
import tensorflow as tf
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Dense(32, activation='relu',
input_shape=(X_train.shape[1],)),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # output pentru clasificare
binară
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
2025-06-15 09:58:04.517672: I tensorflow/core/util/port.cc:153] oneDNN
custom operations are on. You may see slightly different numerical
results due to floating-point round-off errors from different
computation orders. To turn them off, set the environment variable
`TF_ENABLE_ONEDNN_OPTS=0`.
2025-06-15 09:58:04.518433: I
external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda
drivers on your machine, GPU will not be used.
2025-06-15 09:58:04.523461: I
external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda
drivers on your machine, GPU will not be used.
2025-06-15 09:58:04.532728: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
```

```
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1749981484.549408     952 cuda_dnn.cc:8579] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1749981484.554505     952 cuda_blas.cc:1407] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
W0000 00:00:1749981484.568485     952 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1749981484.568501     952 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1749981484.568502     952 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1749981484.568503     952 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
2025-06-15 09:58:04.572540: I
tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
/usr/local/lib/python3.11/site-packages/keras/src/layers/core/dense.py
:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
2025-06-15 09:58:05.888882: E
external/local_xla/xla/stream_executor/cuda/cuda_platform.cc:51]
failed call to cuInit: INTERNAL: CUDA error: Failed call to cuInit:
UNKNOWN ERROR (303)

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━
┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━
┩
│ dense (Dense)                   │ (None, 32)             │
672 │
├─────────────────────────────────┼────────────────────────┼───────────────
┤
│ dense_1 (Dense)                 │ (None, 16)             │
```

```
528 │
├─────────────────────────────────┼────────────────────────────┼──────────
──────┤
│ dense_2 (Dense)                 │ (None, 1)                  │
17 │
└─────────────────────────────────┴────────────────────────────┴──────────
──────┘

 Total params: 1,217 (4.75 KB)

 Trainable params: 1,217 (4.75 KB)

 Non-trainable params: 0 (0.00 B)
```

Antrenarea modelului

```
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
validation_split=0.2, verbose=0)
```

Evaluarea modelului pe setul de testare

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8468 - loss: 0.3631

Test Accuracy: 0.8750
```

Concluzie Deep Learning

În această secțiune am folosit **TensorFlow** pentru a construi și antrena un model de rețea neuronală care prezice probabilitatea de boală cardiacă pe baza caracteristicilor pacienților. Pașii principali sunt:

1. **Conversie Spark → Pandas**:
   - Datele Spark sunt convertite în format Pandas (`toPandas()`) pentru a fi compatibile cu TensorFlow și Scikit-learn.
2. **Preprocesare date**:
   - Separăm `X` (features) și `y` (eticheta).
   - Coloanele numerice sunt scalate (`StandardScaler`) iar cele categorice sunt codificate (`OneHotEncoder`) cu un `ColumnTransformer`.
3. **Împărțirea datasetului**:
   - Setul este împărțit în subseturi de antrenare și testare (`train_test_split`), cu 80% pentru antrenare și 20% pentru test.
4. **Crearea modelului**:
   - Rețea neuronală secvențială cu 2 straturi ascunse:
     - 32 neuroni → 16 neuroni → 1 neuron de ieșire cu activare `sigmoid` (clasificare binară).

   - Optimizator: `adam`, funcție de pierdere: `binary_crossentropy`.
5. **Antrenare și evaluare**:
   - Modelul este antrenat pe 80% din datele de antrenare, cu validare internă de 20%.
   - Evaluarea se face pe setul de test (`evaluate()`), iar acuratețea este afișată.

Modelul de *rețea neuronală* **oferă** o *acuratețe competitivă* pe *setul de testare*.

`Avantaje`:

- **Capacitate de modelare** *complexă*
- **Poate învăța** *relații subtile* în date

`Dezavantaje`:

- **Necesită** *mai multă putere computațională*
- **Mai puțin interpretabil** decât modele simple