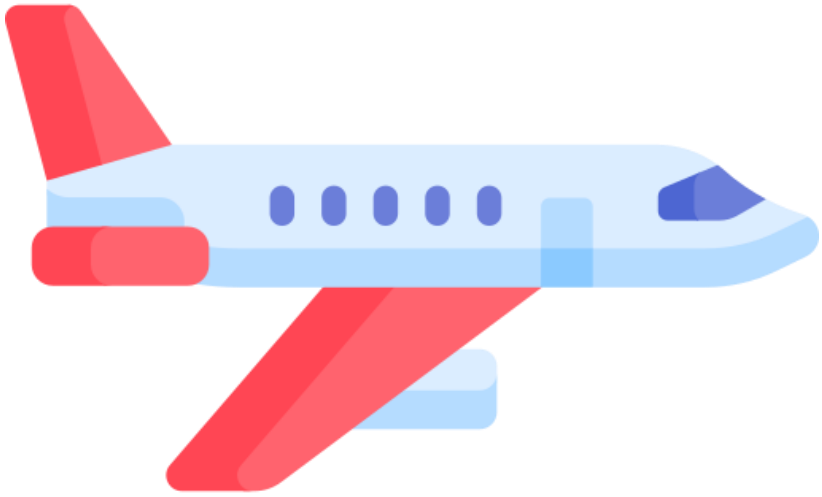
Analiza și predicția satisfacției pasagerilor unei companii aeriene



1. Introducere: prezentarea setului de date și enunțarea obiectivelor

Setul de date folosit conține informații despre persoanele ce au efectuat zboruri cu o anumită companie aeriană. Scopurile următoarelor interogări și metode de Machine Learning folosite sunt de a descoperi care sunt factorii ce influențează satisfacția pasagerilor, relațiile dintre aceștia și de a prezice nivelul general de mulțumire al pasagerilor față de companie.

Fișierul .csv conține următoarele coloane relevante:

- **Gender**: genul persoanei (**Male** sau **Female**)
- **Customer Type**: tipul clientului din punct de vedere al frecvenței utilizării companiei (**Loyal Customer** sau **disloyal Customer**)
- **Age**: vârsta pasagerilor
- **Type of Travel**: scopul călătoriei (**Personal Travel** sau **Business travel**)
- **Class**: clasa de zbor (**Eco**, **Eco Plus** sau **Business**)

- **Flight distance**: distanța zborului
- **Inflight wifi service**: nivelul de satisfacție cu serviciul wifi la bord (**0** - N/A, **1-5**)
- **Departure/Arrival time convenient**: nivelul de satisfacție cu orele la care au fost programate decolare/aterizarea
- **Ease of Online booking**: nivelul de satisfacție cu procesul de rezervare online a biletelor
- **Gate location**: nivelul de satisfacție cu locația porții de îmbarcare
- **Food and drink**: nivelul de satisfacție cu mâncarea și băutura de la bord
- **Online boarding**: nivelul de satisfacție legat de procesul îmbarcării online
- **Seat comfort**: nivelul de satisfacție legat de confortul locului din avion
- **Inflight entertainment**: nivelul de satisfacție cu mijloacele de divertisment de la bord
- **On-board service**: nivelul de satisfacție cu serviciile de la bord
- **Leg room service**: nivelul de satisfacție cu spațiul locului din avion
- **Baggage handling**: nivelul de satisfacție cu procesul de lăsare/colectare a bagajelor
- **Check-in service**: nivelul de satisfacție cu procesul de Check In
- **Inflight service**: nivelul de satisfacție legat de serviciile însoțitorilor de zbor
- **Cleanliness**: nivelul de satisfacție legat de curățenie
- **Departure Delay in Minutes**: numărul de minute de întârziere pe care le-a avut aeronava la plecare
- **Arrival Delay in Minutes**: numărul de minute de întârziere pe care le-a avut aeronava la aterizare
- **Satisfaction**: nivelul general de satisfacție legat de companie (**satisfied** sau **neutral or dissatisfied**)

Setul de date împreună cu mai multe detalii poate fi găsit la https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction.

2. Implementarea de script-uri Spark pentru procesarea datelor, pregătirea, curățarea, transformarea acestora etc. Vor fi prezente grupări și agregări de date. Se vor utiliza Dataframes și Spark SQL (ambele).

Pentru a avea o înțelegere mai bună asupra informațiilor, vom inspecta setul de date afișând primele 20 de linii.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("airline").getOrCreate()
airline = spark.read.csv("/content/drive/MyDrive/Proiect Big
Data/airline_satisfaction.csv", header=True, inferSchema=True)

airline.show()
```

```
1 airline = spark.read.csv("/content/drive/MyDrive/Proiect Big Data/airline_satisfaction.csv", header=True, inferSchema=True)
2
3 airline.show()
```

```
+---+------+------+----------------+---+--------------+--------+--------------+--------------------+--------------------------------+
|_c0|    id|Gender|   Customer Type|Age| Type of Travel|   Class|Flight Distance|Inflight wifi service|Departure/Arrival time convenient|
+---+------+------+----------------+---+--------------+--------+--------------+--------------------+--------------------------------+
|  0| 70172|  Male|   Loyal Customer| 13|Personal Travel|Eco Plus|           460|                   3|                               4|
|  1|  5047|  Male|disloyal Customer| 25|Business travel|Business|           235|                   3|                               2|
|  2|110028|Female|   Loyal Customer| 26|Business travel|Business|          1142|                   2|                               2|
|  3| 24026|Female|   Loyal Customer| 25|Business travel|Business|           562|                   2|                               5|
|  4|119299|  Male|   Loyal Customer| 61|Business travel|Business|           214|                   3|                               3|
|  5|111157|Female|   Loyal Customer| 26|Personal Travel|     Eco|          1180|                   3|                               4|
|  6| 82113|  Male|   Loyal Customer| 47|Personal Travel|     Eco|          1276|                   2|                               4|
|  7| 96462|Female|   Loyal Customer| 52|Business travel|Business|          2035|                   4|                               3|
|  8| 79485|Female|   Loyal Customer| 41|Business travel|Business|           853|                   1|                               2|
|  9| 65725|  Male|disloyal Customer| 20|Business travel|     Eco|          1061|                   3|                               3|
| 10| 34991|Female|disloyal Customer| 24|Business travel|     Eco|          1182|                   4|                               5|
| 11| 51412|Female|   Loyal Customer| 12|Personal Travel|Eco Plus|           308|                   2|                               4|
| 12| 98628|  Male|   Loyal Customer| 53|Business travel|     Eco|           834|                   1|                               4|
| 13| 83502|  Male|   Loyal Customer| 33|Personal Travel|     Eco|           946|                   4|                               2|
| 14| 95789|Female|   Loyal Customer| 26|Personal Travel|     Eco|           453|                   3|                               2|
| 15|100580|  Male|disloyal Customer| 13|Business travel|     Eco|           486|                   2|                               1|
| 16| 71142|Female|   Loyal Customer| 26|Business travel|Business|          2123|                   3|                               3|
| 17|127461|  Male|   Loyal Customer| 41|Business travel|Business|          2075|                   4|                               4|
| 18| 70354|Female|   Loyal Customer| 45|Business travel|Business|          2486|                   4|                               4|
| 19| 66246|  Male|   Loyal Customer| 38|Personal Travel|     Eco|           460|                   2|                               3|
+---+------+------+----------------+---+--------------+--------+--------------+--------------------+--------------------------------+
only showing top 20 rows
```

Deoarece primele două coloane (numărul curent și id-ul) nu ne sunt de folos, putem renunța la ele.

De asemenea, observăm o inconsistență între anumite clase cu privire la scrierea cu majuscule sau minuscule (Personal **T**ravel vs Business **t**ravel, **L**oyal Customer vs **d**isloyal Customer) pe care le vom remedia.

```
from pyspark.sql.functions import initcap, substring, translate

airline_clean = airline.drop("_c0", "id")
airline_clean = airline_clean.withColumn("Customer Type",
initcap(airline_clean["Customer Type"]))
airline_clean = airline_clean.withColumn("Type of Travel", translate("Type
of Travel", 't', 'T'))

airline_clean.show()
```

```
from pyspark.sql.functions import initcap, substring, translate

airline_clean = airline.drop("_c0", "id")
airline_clean = airline_clean.withColumn("Customer Type", initcap(airline_clean["Customer Type"]))
airline_clean = airline_clean.withColumn("Type of Travel", translate("Type of Travel", 't', 'T'))

airline_clean.show()
```

```
+------+-----------------+---+---------------+--------+---------------+---------------------+----------
|Gender|    Customer Type|Age| Type of Travel|   Class|Flight Distance|Inflight wifi service|Departure/
+------+-----------------+---+---------------+--------+---------------+---------------------+----------
|  Male|   Loyal Customer| 13|Personal Travel|Eco Plus|            460|                    3|
|  Male|Disloyal Customer| 25|Business Travel|Business|            235|                    3|
|Female|   Loyal Customer| 26|Business Travel|Business|           1142|                    2|
|Female|   Loyal Customer| 25|Business Travel|Business|            562|                    2|
|  Male|   Loyal Customer| 61|Business Travel|Business|            214|                    3|
|Female|   Loyal Customer| 26|Personal Travel|     Eco|           1180|                    3|
|  Male|   Loyal Customer| 47|Personal Travel|     Eco|           1276|                    2|
|Female|   Loyal Customer| 52|Business Travel|Business|           2035|                    4|
|Female|   Loyal Customer| 41|Business Travel|Business|            853|                    1|
|  Male|Disloyal Customer| 20|Business Travel|     Eco|           1061|                    3|
|Female|Disloyal Customer| 24|Business Travel|     Eco|           1182|                    4|
|Female|   Loyal Customer| 12|Personal Travel|Eco Plus|            308|                    2|
|  Male|   Loyal Customer| 53|Business Travel|     Eco|            834|                    1|
|  Male|   Loyal Customer| 33|Personal Travel|     Eco|            946|                    4|
|Female|   Loyal Customer| 26|Personal Travel|     Eco|            453|                    3|
|  Male|Disloyal Customer| 13|Business Travel|     Eco|            486|                    2|
|Female|   Loyal Customer| 26|Business Travel|Business|           2123|                    3|
|  Male|   Loyal Customer| 41|Business Travel|Business|           2075|                    4|
|Female|   Loyal Customer| 45|Business Travel|Business|           2486|                    4|
|  Male|   Loyal Customer| 38|Personal Travel|     Eco|            460|                    2|
+------+-----------------+---+---------------+--------+---------------+---------------------+----------
only showing top 20 rows
```

De asemenea, dorim să numărăm valorile null de pe fiecare coloană pentru a decide dacă le vom înlocui sau șterge.

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

airline_clean.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c)
for c in airline_clean.columns]).show()
print("Dataset contains:", airline_clean.count(), "rows.")
```

```
# calculăm numărul de valori null din fiecare coloană
from pyspark.sql.functions import col, isnan, when, count

airline_clean.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in airline_clean.columns]).show()
print("Dataset contains:", airline_clean.count(), "rows.")
```

```
---------+---------------+---------------+---------------+--------------+---------------+----------+-------------------------+-----------------------+
rtainment|On-board service|Leg room service|Baggage handling|Checkin service|Inflight service|Cleanliness|Departure Delay in Minutes|Arrival Delay in Minutes|
---------+---------------+---------------+---------------+--------------+---------------+----------+-------------------------+-----------------------+
       0|              0|              0|              0|             0|              0|         0|                        0|                    310|
---------+---------------+---------------+---------------+--------------+---------------+----------+-------------------------+-----------------------+

Dataset contains: 103904 rows.
```

Observăm că singura coloană ce conține valori null este **Arrival Delay in Minutes** (310). Deoarece acest număr este foarte mic față de numărul total de linii din setul de date (103904) alegem să ștergem intrările ce conțin valori null și să afișăm statisticile și schema datelor.

```
airline_data = airline_clean.na.drop()
```

```
airline_data.describe().show()
```

```
1 airline_data = airline_clean.na.drop()
2
3 airline_data.describe().show()
```

```
+-------+------+----------------+-----------------+--------------+--------+-----------------+--------------------+------------------------------+
|summary|Gender|   Customer Type|              Age| Type of Travel|   Class|  Flight Distance|Inflight wifi service|Departure/Arrival time convenient|
+-------+------+----------------+-----------------+--------------+--------+-----------------+--------------------+------------------------------+
|  count|103594|          103594|           103594|        103594|  103594|           103594|              103594|                        103594|
|   mean|  null|            null|39.380466050157345|          null|    null|1189.3252022317895|   2.729752688379636|            3.0600806996544203|
| stddev|  null|            null|15.113125274720122|          null|    null| 997.2972348259818|   1.327866240825716|            1.5252328832650188|
|    min|Female|Disloyal Customer|                7|Business Travel|Business|               31|                   0|                             0|
|    max|  Male|  Loyal Customer|               85|Personal Travel|Eco Plus|             4983|                   5|                             5|
+-------+------+----------------+-----------------+--------------+--------+-----------------+--------------------+------------------------------+
```

```
airline_data.printSchema()
```

```
1 # afișăm schema
2 airline_data.printSchema()
```

```
root
 |-- Gender: string (nullable = true)
 |-- Customer Type: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Type of Travel: string (nullable = true)
 |-- Class: string (nullable = true)
 |-- Flight Distance: integer (nullable = true)
 |-- Inflight wifi service: integer (nullable = true)
 |-- Departure/Arrival time convenient: integer (nullable = true)
 |-- Ease of Online booking: integer (nullable = true)
 |-- Gate location: integer (nullable = true)
 |-- Food and drink: integer (nullable = true)
 |-- Online boarding: integer (nullable = true)
 |-- Seat comfort: integer (nullable = true)
 |-- Inflight entertainment: integer (nullable = true)
 |-- On-board service: integer (nullable = true)
 |-- Leg room service: integer (nullable = true)
 |-- Baggage handling: integer (nullable = true)
 |-- Checkin service: integer (nullable = true)
 |-- Inflight service: integer (nullable = true)
 |-- Cleanliness: integer (nullable = true)
 |-- Departure Delay in Minutes: integer (nullable = true)
 |-- Arrival Delay in Minutes: double (nullable = true)
 |-- satisfaction: string (nullable = true)
```

Cu ajutorul operațiilor pe dataframe-uri pentru fiecare clasă și nivel general de satisfacție calculăm numărul de adulți ce au călătorit în scop profesional.

```
airline_data.filter((airline_data["Age"] >= 18) & (airline_data["Type of Travel"] == "Business Travel")) \
            .groupBy(["Class", "satisfaction"]) \
            .count() \
            .orderBy("Class", ascendng = [False]) \
            .show(truncate=False)
```

```
1 airline_data.filter((airline_data["Age"] >= 18) & (airline_data["Type of Travel"] == "Business Travel")) \
2             .groupBy(["Class", "satisfaction"]) \
3             .count() \
4             .orderBy("Class", ascendng = [False]) \
5             .show(truncate=False)
```

```
+--------+----------------------+-----+
|Class   |satisfaction          |count|
+--------+----------------------+-----+
|Business|neutral or dissatisfied|12787|
|Business|satisfied             |33663|
|Eco     |neutral or dissatisfied|13306|
|Eco     |satisfied             |5803 |
|Eco Plus|neutral or dissatisfied|2199 |
|Eco Plus|satisfied             |1473 |
+--------+----------------------+-----+
```

Cu ajutorul Spark SQL vrem să afișăm pentru fiecare gen și clasă: vârsta mediană, distanța minimă și maximă parcursă de persoanele satisfăcute de companie și ale căror întârzieri nu depășesc media.

```
airline_data.createOrReplaceTempView("airline")

spark.sql("""SELECT Gender, Class, MEDIAN(Age) AS median_age,
                    MIN(`Flight Distance`) AS min_distance, MAX(`Flight Distance`) AS max_distance
             FROM airline
             WHERE satisfaction = 'satisfied'
             AND `Arrival Delay in Minutes` <= (SELECT AVG(`Arrival Delay in Minutes`) FROM airline)
             AND `Departure Delay in Minutes` <= (SELECT AVG(`Departure Delay in Minutes`) FROM airline)
             GROUP BY Gender, Class
             ORDER BY 1, 2, 5 DESC""").show()
```

```
airline_data.createOrReplaceTempView("airline")
```

```
spark.sql("""SELECT Gender, Class, MEDIAN(Age) AS median_age,
                    MIN(`Flight Distance`) AS min_distance, MAX(`Flight Distance`) AS max_distance
             FROM airline
             WHERE satisfaction = 'satisfied'
             AND `Arrival Delay in Minutes` <= (SELECT AVG(`Arrival Delay in Minutes`) FROM airline)
             AND `Departure Delay in Minutes` <= (SELECT AVG(`Departure Delay in Minutes`) FROM airline)
             GROUP BY Gender, Class
             ORDER BY 1, 2, 5 DESC""").show()
```

```
+------+--------+----------+------------+------------+
|Gender|   Class|median_age|min_distance|max_distance|
+------+--------+----------+------------+------------+
|Female|Business|      44.0|          56|        4983|
|Female|     Eco|      39.0|          31|        4963|
|Female|Eco Plus|      38.0|          67|        2917|
|  Male|Business|      44.0|          67|        4983|
|  Male|     Eco|      39.0|          56|        4963|
|  Male|Eco Plus|      39.0|          31|        4983|
+------+--------+----------+------------+------------+
```

Pentru fiecare tip de client, clasă și gen calculăm media, minimul și maximul mediei satisfacției față de serviciile de la bord (acolo unde există evaluări) pentru grupurile care conțin minim 4000 de persoane.

Pentru a afla media notelor pe care le-au acordat pasagerii anumitor servicii vom crea o funcție definită de utilizator ce primește ca parametru un număr arbitrar de valorii de tip *nivel de satisfacție* și returnează media celor diferite de 0 (valoare ce semnifică absența unei note).

```
def avg_satisfaction(*args):
  sat_sum = 0
  sat_nr = 0
  for sat in args:
    if sat > 0:
      sat_sum += sat
      sat_nr += 1
  return sat_sum / sat_nr


# înregistrăm funcția definită anterior ca udf în Spark
from pyspark.sql.functions import udf
from pyspark.sql.types import DoubleType

avg_satisfaction_udf = udf(lambda *z : avg_satisfaction(*z), DoubleType())
```

```
# adăugăm o coloană ce reține media notelor serviciilor de la  bord
airline_data_avg_sat = airline_data \
 .withColumn("Average on board satisfaction", \
  avg_satisfaction_udf(col("Inflight wifi service"), \
                       col("Food and drink"), col("Seat comfort"), \
                       col("Inflight entertainment"), \
                       col("On-board service"), col("Leg room service"), \
                       col("Inflight service"), col("Cleanliness")))

airline_data_avg_sat.show(5)
```

```
airline_data_avg_sat = airline_data.withColumn("Average on board satisfaction", \
                                       avg_satisfaction_udf(col("Inflight wifi service"), \
                                                            col("Food and drink"), \
                                                            col("Seat comfort"), \
                                                            col("Inflight entertainment"), \
                                                            col("On-board service"), \
                                                            col("Leg room service"), \
                                                            col("Inflight service"), \
                                                            col("Cleanliness")))

# adăugăm o coloană separată ce reține media notelor serviciilor de la bord
airline_data_avg_sat.show(5)
```

```
-----+---------------+--------------+---------------+----------+-----------------------+---------------------+------------------+--------------------------+
rvice|Baggage handling|Checkin service|Inflight service|Cleanliness|Departure Delay in Minutes|Arrival Delay in Minutes|      satisfaction|Average on board satisfaction|
-----+---------------+--------------+---------------+----------+-----------------------+---------------------+------------------+--------------------------+
   3|              4|             4|              5|         5|                     25|                 18.0|neutral or dissat...|                     4.375|
   5|              3|             1|              4|         1|                      1|                  6.0|neutral or dissat...|                     2.125|
   3|              4|             4|              4|         5|                      0|                  0.0|         satisfied|                     4.125|
   5|              3|             1|              4|         2|                     11|                  9.0|neutral or dissat...|                     2.625|
   4|              4|             3|              3|         3|                      0|                  0.0|         satisfied|                       3.5|
-----+---------------+--------------+---------------+----------+-----------------------+---------------------+------------------+--------------------------+
```

```
from pyspark.sql.functions import min, max, avg

airline_data_avg_sat.groupBy(["Customer Type", "Class", "Gender"]) \
              .agg(min("Average on board satisfaction").alias("min_sat"), \
                   avg("Average on board satisfaction").alias("avg_sat"), \
                   max("Average on board satisfaction").alias("max_sat"), \
                   count('*').alias("count")) \
              .filter("count > 4000").drop("count").show()
```

```
airline_data_avg_sat.groupBy(["Customer Type", "Class", "Gender"]) \
                    .agg(min("Average on board satisfaction").alias("min_sat"), \
                         avg("Average on board satisfaction").alias("avg_sat"), \
                         max("Average on board satisfaction").alias("max_sat"), \
                         count('*').alias("count")) \
                    .filter("count > 4000") \
                    .drop("count") \
                    .show()
```

```
+-----------------+--------+------+-------+------------------+-------+
|    Customer Type|   Class|Gender|min_sat|           avg_sat|max_sat|
+-----------------+--------+------+-------+------------------+-------+
|   Loyal Customer|     Eco|Female|  1.125|3.1217449942940165|    5.0|
|Disloyal Customer|     Eco|  Male|   1.25| 3.031726595076255|    5.0|
|   Loyal Customer|Business|Female|    1.0| 3.569775085175314|    5.0|
|   Loyal Customer|Business|  Male|    1.0| 3.573174505508376|    5.0|
|Disloyal Customer|     Eco|Female|  1.125| 3.012797365754813|    5.0|
|   Loyal Customer|     Eco|  Male|  1.125| 3.148047057722893|    5.0|
+-----------------+--------+------+-------+------------------+-------+
```

Pentru fiecare nivel de întârzieri (stabilit cu ajutorul unei funcții definite de utilizator ce calculează suma dintre întârzierile de la decolare și aterizare și asignează una din clasele *No Delay*, *Small*, *Average*, *Long*) afișăm media distanțelor pentru tipul cel mai frecvent de pasageri din grupul respectiv.

```
def delay(departure, arrival):
  sum_minutes = departure + arrival
  if sum_minutes == 0:
    return "No Delay"
  elif sum_minutes > 0 and sum_minutes <= 10:
    return "Small"
  elif sum_minutes > 10 and sum_minutes <= 60:
    return "Average"
  elif sum_minutes > 60:
    return "Long"


# înregistrăm funcția în sesiune pentru a o pute folosi în cereri SQL
from pyspark.sql.types import StringType

spark.udf.register("delay_udf", delay, StringType())

spark.sql("""WITH aux AS (SELECT delay_udf(`Departure Delay in Minutes`,
`Arrival Delay in Minutes`) AS Delay, `Customer Type`, ROUND(AVG(`Flight
Distance`), 2) AS AvgDist, COUNT(*) AS c
                          FROM airline
                          GROUP BY delay_udf(`Departure Delay in Minutes`,
`Arrival Delay in Minutes`), `Customer Type`)
       SELECT Delay, `Customer Type`, AvgDist
       FROM aux a
       WHERE c = (SELECT MAX(c) FROM aux WHERE delay = a.delay)""").show()
```

```
spark.sql("""WITH aux AS (SELECT delay_udf(`Departure Delay in Minutes`, `Arrival Delay in Minutes`) AS Delay,
                               `Customer Type`, ROUND(AVG(`Flight Distance`), 2) AS AvgDist, COUNT(*) AS c
                          FROM airline
                          GROUP BY delay_udf(`Departure Delay in Minutes`, `Arrival Delay in Minutes`), `Customer Type`)
            SELECT Delay, `Customer Type`, AvgDist
            FROM aux a
            WHERE c = (SELECT MAX(c)
                       FROM aux
                       WHERE delay = a.delay)""").show()
```

```
+--------+--------------+-------+
|   Delay| Customer Type|AvgDist|
+--------+--------------+-------+
|No Delay|Loyal Customer|1276.39|
| Average|Loyal Customer|1321.95|
|   Small|Loyal Customer|1326.44|
|    Long|Loyal Customer|1275.85|
+--------+--------------+-------+
```

3. Aplicarea a cel puțin două metode ML folosind Spark MLlib

Scopul următoarelor metode de Machine Learning este de a prezice nivelul general de satisfacție a unui pasager (**satisfied** sau **neutral or dissatisfied**) față de compania aeriană pe baza informațiilor despre zboruri și a evaluărilor diferitelor servicii.

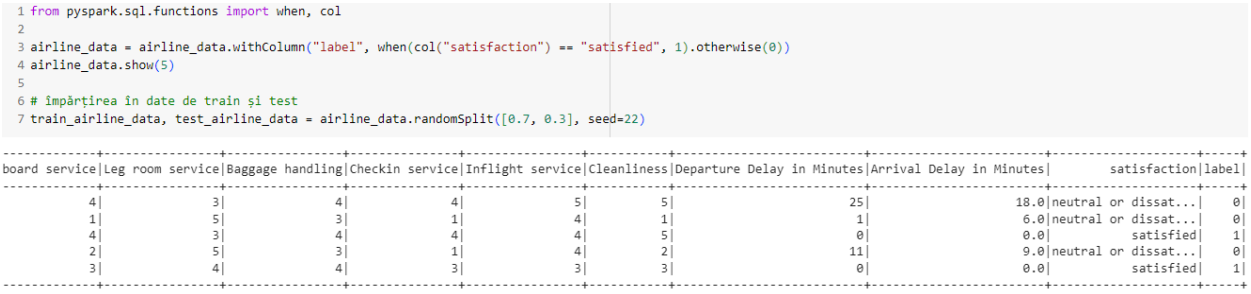
Pentru toate modelele propuse setul de date de test va avea o proporție de 30% și vom transforma variabilele categoriale **Gender**, **Customer Type**, **Type of Travel** și **Class** în variabile numerice cu ajutorul *One-Hot Encoding*. De asemenea, valorile coloanei țintă **satisfaction** vor fi transformate în valori binare astfel:

- **satisfied** -> 1
- **neutral or dissatisfied** -> 0

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

airline_data = airline_data.withColumn("label", when(col("satisfaction")
== "satisfied", 1).otherwise(0))
airline_data.show(5)

# împărțirea în date de train și test
train_airline_data, test_airline_data = airline_data.randomSplit([0.7,
0.3], seed=22)
```



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

airline_data = airline_data.withColumn("label", when(col("satisfaction") == "satisfied", 1).otherwise(0))
airline_data.show(5)

# împărțirea în date de train și test
train_airline_data, test_airline_data = airline_data.randomSplit([0.7, 0.3], seed=22)
```

```
+------------+---------------+---------------+--------------+---------------+----------+--------------------------+-------------------------+--------------------+-----+
|board service|Leg room service|Baggage handling|Checkin service|Inflight service|Cleanliness|Departure Delay in Minutes|Arrival Delay in Minutes|        satisfaction|label|
+------------+---------------+---------------+--------------+---------------+----------+--------------------------+-------------------------+--------------------+-----+
|           4|              3|              4|             4|              5|         5|                        25|                     18.0|neutral or dissat...|    0|
|           1|              5|              3|             1|              4|         1|                         1|                      6.0|neutral or dissat...|    0|
|           4|              3|              4|             4|              4|         5|                         0|                      0.0|           satisfied|    1|
|           2|              5|              3|             1|              4|         2|                        11|                      9.0|neutral or dissat...|    0|
|           3|              4|              4|             3|              3|         3|                         0|                      0.0|           satisfied|    1|
+------------+---------------+---------------+--------------+---------------+----------+--------------------------+-------------------------+--------------------+-----+
```

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

# transformarea variabilelor categoriale
gender_indexer = StringIndexer(inputCol="Gender", outputCol="Gender
Index")
customer_type_indexer = StringIndexer(inputCol="Customer Type",
outputCol="Customer Type Index")
```

```
travel_type_indexer = StringIndexer(inputCol="Type of Travel",
outputCol="Travel Type Index")
class_indexer = StringIndexer(inputCol="Class", outputCol="Class Index")
ohe = OneHotEncoder(inputCols=["Gender Index", "Customer Type Index",
"Travel Type Index", "Class Index"], \
                    outputCols=["Gender OHE", "Customer Type OHE", "Travel
Type OHE", "Class OHE"])

# alegerea și asamblarea coloanelor de tip feature
cols = [c for c in airline_data.columns if c not in ["Gender", "Customer
Type", "Type of Travel", "Class", "satisfaction", "label"]]
cols.extend(["Gender OHE", "Customer Type OHE", "Travel Type OHE", "Class
OHE"])
assembler = VectorAssembler(inputCols=cols, outputCol="features")
```

Regresie Logistică

Deoarece problema este una de clasificare binară, vom folosi pentru început **regresia logistică** pentru a face predicțiile.

Pentru a înlănțui pașii necesari pentru obținerea rezultatelor vom folosi un Pipeline.

```
from pyspark.ml.classification import LogisticRegression

# definirea modelului
log_reg = LogisticRegression(featuresCol="features", labelCol="label")

# definirea pipeline-ului
from pyspark.ml import Pipeline

log_reg_pipeline = Pipeline(stages=[gender_indexer, customer_type_indexer,
travel_type_indexer, class_indexer, ohe, assembler, log_reg])

# antrenarea modelului
fit_log_reg_model = log_reg_pipeline.fit(train_airline_data)

# efectuarea predicțiilor
pred_log_reg = fit_log_reg_model.transform(test_airline_data)
```

Pentru evaluarea modelului vom folosi evaluatorul pentru clasificare binară cu metrica Area Under the Curve. Cu cât această valoare este mai aproape de 1, cu atât modelul are o performanță mai bună.

```
# evaluarea modelului cu ajutorul metricii Area Under the Curve
from pyspark.ml.evaluation import BinaryClassificationEvaluator

log_reg_eval =
BinaryClassificationEvaluator(rawPredictionCol='prediction',
labelCol='label')
log_reg_eval.evaluate(pred_log_reg)
```

```
1 # evaluarea modelului cu ajutorul metricii Area Under the Curve
2 from pyspark.ml.evaluation import BinaryClassificationEvaluator
3
4 log_reg_eval = BinaryClassificationEvaluator(rawPredictionCol='prediction', labelCol='label')
5 log_reg_eval.evaluate(pred_log_reg)
```

```
0.8690452202942274
```

Observăm că AUC este aproximativ **0.87** ceea ce denotă un clasificator bun. De asemenea, vom afișa **matricea de confuzie** pentru a afla numărul de predicții fals-pozitive și fals-negative.

```
# afișarea matricei de confuzie
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.types import FloatType

preds_and_labels_log_reg = pred_log_reg.select(["prediction",
"label"]).withColumn("label",col("label").cast(FloatType())).orderBy("pred
iction")
metrics = MulticlassMetrics(preds_and_labels_log_reg.rdd.map(tuple))
print(metrics.confusionMatrix().toArray())
```

```
1 # afișarea matricei de confuzie
2 from pyspark.mllib.evaluation import MulticlassMetrics
3 from pyspark.sql.types import FloatType
4
5 preds_and_labels_log_reg = pred_log_reg.select(["prediction", "label"]).withColumn("label", col("label").cast(FloatType())).orderBy("prediction")
6 metrics = MulticlassMetrics(preds_and_labels_log_reg.rdd.map(tuple))
7 print(metrics.confusionMatrix().toArray())
```

```
/usr/local/lib/python3.10/dist-packages/pyspark/sql/context.py:157: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
  warnings.warn(
[[16041.  1677.]
 [ 2255. 11227.]]
```

Random Forest

Fiind o problemă de clasificare, un al doilea model pe care îl putem încerca este **Random Forest**.

Deoarece acest algoritm are un număr semnificativ de hiperparametrii vom dori optimizarea acestora prin intermediul **Grid Search** (pentru reglarea *bootstrap*, *maxDepth* și *numTrees*) și al metodei **Cross Validation** cu 3 diviziuni.

De asemenea, vom utiliza un pipeline pentru crearea caracteristicilor în formatul dorit și antrenarea modelului.

```
# instanţierea modelului
from pyspark.ml.classification import RandomForestClassifier

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

# definirea pipeline-ului
rf_pipeline = Pipeline(stages=[gender_indexer, customer_type_indexer,
travel_type_indexer, class_indexer, ohe, assembler, rf])

# definirea gridului de parametrii
from pyspark.ml.tuning import ParamGridBuilder

paramGrid = ParamGridBuilder() \
            .addGrid(rf.bootstrap, [True]) \
            .addGrid(rf.maxDepth, [5, 10, 15]) \
            .addGrid(rf.numTrees, [100, 200]) \
            .build()


# aplicarea Cross Validation cu 3 diviziuni
from pyspark.ml.tuning import CrossValidator

crossval = CrossValidator(estimator=rf_pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=3)

# antrenarea modelului
rf_fit = crossval.fit(train_airline_data)

# efectuarea predicţiilor
pred_rf = rf_fit.transform(test_airline_data)
```

Pentru a putea face cu ușurință o comparație între metode vom folosi aceeași metrică Area Under the Curve pentru evaluare.

```
# evaluarea modelului cu ajutorul metricii Area Under the Curve
rf_eval = BinaryClassificationEvaluator(rawPredictionCol='prediction',
labelCol='label')
rf_eval.evaluate(pred_rf)
```

```
1 # evaluarea modelului cu ajutorul metricii Area Under the Curve
2 rf_eval = BinaryClassificationEvaluator(rawPredictionCol='prediction', labelCol='label')
3 rf_eval.evaluate(pred_rf)
```

```
0.9563298279382983
```

Cu o valoare de **0.95** observăm că optimizarea hiperparametrilor modelului Random Forest a adus o îmbunătățire semnificativă față de regresia logistică cu parametrii default.

Pentru această metodă vom afișa atât matricea de confuzie cât și metrici suplimentare precum acuratețe, precizie, recall, f1.

```
# afișarea matricei de confuzie
preds_and_labels_rf = pred_rf.select(["prediction",
"label"]).withColumn("label",
col("label").cast(FloatType())).orderBy("prediction")
metrics = MulticlassMetrics(preds_and_labels_rf.rdd.map(tuple))
print(metrics.confusionMatrix().toArray())
```

```
1 # afișarea matricei de confuzie
2 preds_and_labels_rf = pred_rf.select(["prediction", "label"]).withColumn("label", col("label").cast(FloatType())).orderBy("prediction")
3 metrics = MulticlassMetrics(preds_and_labels_rf.rdd.map(tuple))
4 print(metrics.confusionMatrix().toArray())
```

```
[[17281.   437.]
 [  845. 12637.]]
```

```
# afișarea de metrici suplimentare (accuracy, precision, recall, f1)
print(f"Accuracy: {metrics.accuracy}")
print(f"Precision for 'satisfied': {metrics.precision(1.0)} \t Precision
for 'neutral or dissatisfied': {metrics.precision(0.0)}")
print(f"Recall for 'satisfied': {metrics.recall(1.0)} \t Recall for
'neutral or dissatisfied': {metrics.recall(0.0)}")
print(f"f1 score for 'satisfied': {metrics.fMeasure(1.0)} \t f1 score for
'neutral or dissatisfied': {metrics.fMeasure(0.0)}")
```

```
# afișarea de metrici suplimentare (accuracy, precision, recall, f1)
print(f"Accuracy: {metrics.accuracy}")
print(f"Precision for 'satisfied': {metrics.precision(1.0)} \t Precision for 'neutral or dissatisfied': {metrics.precision(0.0)}")
print(f"Recall for 'satisfied': {metrics.recall(1.0)} \t Recall for 'neutral or dissatisfied': {metrics.recall(0.0)}")
print(f"f1 score for 'satisfied': {metrics.fMeasure(1.0)} \t f1 score for 'neutral or dissatisfied': {metrics.fMeasure(0.0)}")
```

```
Accuracy: 0.9589102564102564
Precision for 'satisfied': 0.9665748814440875    Precision for 'neutral or dissatisfied': 0.9533818823789032
Recall for 'satisfied': 0.9373238391929981       Recall for 'neutral or dissatisfied': 0.9753358166835986
f1 score for 'satisfied': 0.9517246573279109     f1 score for 'neutral or dissatisfied': 0.9642339024662425
```

Observăm cum clasificatorul are o performanță ușor mai bună pentru clasa **neutral or dissatisfied**.

4. Utilizarea a cel puțin unui Data Pipeline

Atât în metodele de la punctul 3, cât și la punctul 7 au fost utilizate Pipeline-uri pentru codificarea variabilelor categorice, asamblarea coloanelor de tip feature și definirea modelului. Acestea au fost utile în ușurarea înlănțuirii pașilor și aplicarea lor în procesele de antrenare și predicție.

5. Utilizarea unei funcții definite de utilizator (UDF), optimizarea hiperparametrilor.

La punctul 1 am folosit două funcții definite de utilizator: `avg_satisfaction_udf` pentru a afla media evaluărilor pasagerilor pentru anumite servicii și `delay_udf` ce atribuie una din categoriile *No Delay, Small, Average, Long* pe baza numărului de minute de întârziere pe care l-a avut zborul. Prima funcție a fost utilizată în cadrul operațiilor pe DataFrame-uri în timp ce cea de-a doua a fost folosită în cereri SQL.

La modelul Random Forest de la punctul 3 am folosit procesul de optimizare al hiperparametrilor *bootstrap*, *maxDepth* și *numTrees* cu metoda Cross Validation pentru a crește performanța. Avantajul unei astfel de abordări a fost obținerea unui scor foarte bun de **0.95**, în timp ce dezavantajele constau în timpul îndelungat de rulare pentru antrenarea si evaluarea a 18 modele.

6. Aplicarea a cel puțin unei metode DL cu ajutorul Tensorflow

Vom prezice una dintre valorile coloanei **satisfaction** (**satisfied** sau **neutral or dissatisfied**) cu ajutorul rețelelor neuronale adânci.

Am ales această abordare atât datorită creșterii în popularitate și performanță a metodelor **Deep Learning** cât și datorită librăriilor **Keras** și **Tensorflow** din Python care oferă API-uri ușor de folosit pentru construirea rețelelor neuronale.

Pentru început vom aplica aceiași pași de citire, transformare și pregătire a setului de date, de data aceasta cu ajutorul DataFrame-urilor **Pandas** ce sunt compatibile cu **Tensorflow**.

```
import numpy as np
import pandas as pd

airline = pd.read_csv("/content/drive/MyDrive/Proiect Big
Data/airline_satisfaction.csv")
airline.head()
```

```
1 import numpy as np
2 import pandas as pd
3
4 airline = pd.read_csv("/content/drive/MyDrive/Proiect Big Data/airline_satisfaction.csv")
5 airline.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	...	Inflight entertainment
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	...	5
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	...	1
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	...	5
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	...	2
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	...	3

5 rows × 25 columns

```
# numărarea de valori null
airline.isna().sum()
```

```
1 # numărarea de valori null
2 airline.isna().sum()
```

```
Checkin service                    0
Inflight service                   0
Cleanliness                        0
Departure Delay in Minutes         0
Arrival Delay in Minutes         310
satisfaction                       0
```

```
# eliminarea liniilor cu valori null și a coloanelor Unnamed: 0 și id
airline.dropna(inplace=True)
airline.drop(columns=["Unnamed: 0", "id"], inplace=True)

# encodarea variabilelor categoriale
numeric_features = pd.get_dummies(airline, columns=["Gender", "Customer
Type", "Type of Travel", "Class"])
numeric_features.head()
```

```
1 # encodarea variabilelor categoriale
2 numeric_features = pd.get_dummies(airline, columns=["Gender", "Customer Type", "Type of Travel", "Class"])
3 numeric_features.head()
```

	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	Seat comfort	Inflight entertainment	...	satisfaction	Gender_Female	Gender_Male	Customer Type_Loyal Customer
0	13	460	3	4	3	1	5	3	5	5	...	neutral or dissatisfied	0	1	1
1	25	235	3	2	3	3	1	3	1	1	...	neutral or dissatisfied	0	1	0
2	26	1142	2	2	2	2	5	5	5	5	...	satisfied	1	0	1
3	25	562	2	5	5	5	2	2	2	2	...	neutral or dissatisfied	1	0	1
4	61	214	3	3	3	3	4	5	5	3	...	satisfied	0	1	1

5 rows × 28 columns

```
# convertirea coloanei satisfaction în 0 și 1
numeric_features["satisfaction"] =
numeric_features["satisfaction"].replace({"satisfied": 1, "neutral or
dissatisfied": 0})

# extragerea coloanelor features și target
X = numeric_features.drop(columns=["satisfaction"])
y = numeric_features["satisfaction"]

print(X.shape)
print(y.shape)
```

```
1 # convertirea coloanei satisfaction în 0 și 1
2 numeric_features["satisfaction"] = numeric_features["satisfaction"].replace({"satisfied": 1, "neutral or dissatisfied": 0})
3
4 # extragerea coloanelor features și target
5 X = numeric_features.drop(columns=["satisfaction"])
6 y = numeric_features["satisfaction"]
7
8 print(X.shape)
9 print(y.shape)
```

```
(103594, 27)
(103594,)
```

Pentru această metodă vom împărți setul de date în antrenare, validare și testare pentru a oferi rețelei neuronale date (de validare) cu ajutorul cărora să își îmbunătățească performanța.

```
from sklearn.model_selection import train_test_split

X_rest, X_test, y_rest, y_test = train_test_split(X, y, test_size=0.2,
random_state=100, stratify=y)
X_train, X_valid, y_train , y_valid = train_test_split(X_rest, y_rest,
test_size=0.2, random_state=100, stratify=y_rest)

print(y_train.shape)
print(y_valid.shape)
print(y_test.shape)
```

```
1 from sklearn.model_selection import train_test_split
2
3 X_rest, X_test, y_rest, y_test = train_test_split(X, y, test_size=0.2, random_state=100, stratify=y)
4 X_train, X_valid, y_train , y_valid = train_test_split(X_rest, y_rest, test_size=0.2, random_state=100, stratify=y_rest)
5
6 print(y_train.shape)
7 print(y_valid.shape)
8 print(y_test.shape)
```

```
(66300,)
(16575,)
(20719,)
```

Vom crea o rețea neuronală cu un strat de intrare cu 128 de neuroni, 2 straturi ascunse a câte 64, respectiv 32 de neuroni și un start de ieșire cu 1 neuron ce va prezice probabilitatea ca pasagerul să fie satisfăcut.

Toate straturile vor avea *Relu* ca funcție de activare cu excepția stratului de ieșire care va avea *Sigmoid*.

De asemenea, vom aplica între straturi un mecanism de regularizare cu ajutorul unor straturi de tip *Dropout* pentru a evita overfitting-ul.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(units=128, activation="relu", input_shape=(27,)))
model.add(Dropout(0.5))
model.add(Dense(units=64, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units=32, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units=1, activation="sigmoid"))

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

```
# antrenarea modelului pe 70 de epoci
history = model.fit(x=X_train, y=y_train, epochs=70,
validation_data=(X_valid, y_valid))
```

```
1 # antrenarea modelului pe 70 de epoci
2 history = model.fit(x=X_train, y=y_train, epochs=70, validation_data=(X_valid, y_valid))
```

```
Epoch 42/70
2072/2072 [==============================] - 5s 3ms/step - loss: 0.3117 - accuracy: 0.8806 - val_loss: 0.3162 - val_accuracy: 0.8703
Epoch 43/70
2072/2072 [==============================] - 7s 3ms/step - loss: 0.3085 - accuracy: 0.8815 - val_loss: 0.3090 - val_accuracy: 0.8716
Epoch 44/70
2072/2072 [==============================] - 6s 3ms/step - loss: 0.3061 - accuracy: 0.8825 - val_loss: 0.2958 - val_accuracy: 0.8787
Epoch 45/70
2072/2072 [==============================] - 5s 3ms/step - loss: 0.3035 - accuracy: 0.8830 - val_loss: 0.3113 - val_accuracy: 0.8748
Epoch 46/70
2072/2072 [==============================] - 7s 3ms/step - loss: 0.3130 - accuracy: 0.8795 - val_loss: 0.2625 - val_accuracy: 0.9064
```

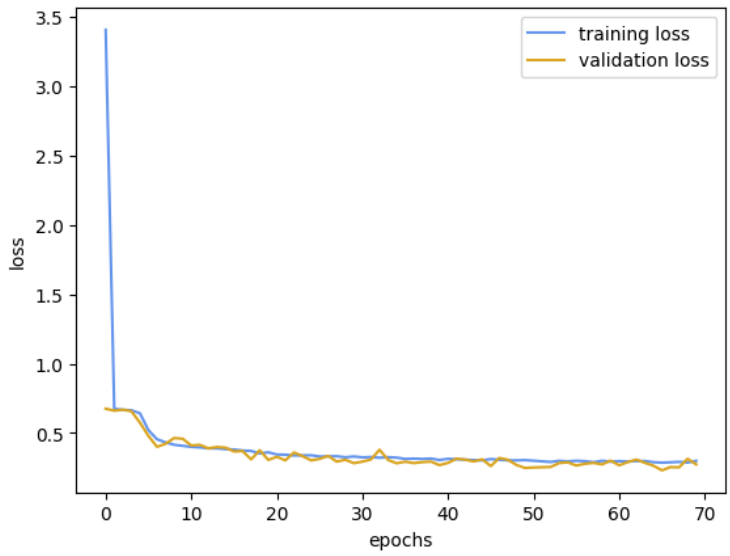
Pentru a observa progresul acurateței pe datele de train și validare de-a lungul etapei de antrenare vom crea un line plot ce monitorizează evoluția funcției de loss.

```
import matplotlib.pyplot as plt

plt.plot(history.history["loss"], c="cornflowerblue", label="training
loss")
plt.plot(history.history["val_loss"], c="goldenrod", label="validation
loss")
plt.legend()
plt.xlabel("epochs")
plt.ylabel("loss")
```

```
4 plt.plot(history.history["loss"], c="cornflowerblue", label="training loss")
5 plt.plot(history.history["val_loss"], c="goldenrod", label="validation loss")
6 plt.legend()
7 plt.xlabel("epochs")
8 plt.ylabel("loss")
```

```
Text(0, 0.5, 'loss')
```



Vom evalua modelul considerând că o probabilitate a ultimului neuron mai mare de 0.5 indică satisfacția pasagerilor. Afișăm raportul de clasificare oferit de sklearn ce include acuratețea, precizia, recall-ul și scorul f1.

```
# calcularea predicțiilor
y_pred = (model.predict(X_test) > 0.5).reshape((-1,))

# afișare classification report pentru metrici precum precision, recall,
f1 pentru fiecare clasă
# și metrici globale precum accuracy
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
1 # calcularea predicțiilor
2 y_pred = (model.predict(X_test) > 0.5).reshape((-1,))
3
4 # afișare classification report pentru metrici precum precision, recall, f1 pentru fiecare clasă
5 # și metrici globale precum accuracy
6 from sklearn.metrics import classification_report
7
8 print(classification_report(y_test, y_pred))
```

```
648/648 [==============================] - 1s 1ms/step
              precision    recall  f1-score   support

           0       0.83      0.99      0.90     11740
           1       0.98      0.74      0.84      8979

    accuracy                           0.88     20719
   macro avg       0.91      0.86      0.87     20719
weighted avg       0.90      0.88      0.88     20719
```

Pentru a afla numărul de predicții fals-pozitive și fals-negative afișăm matricea de confuzie.

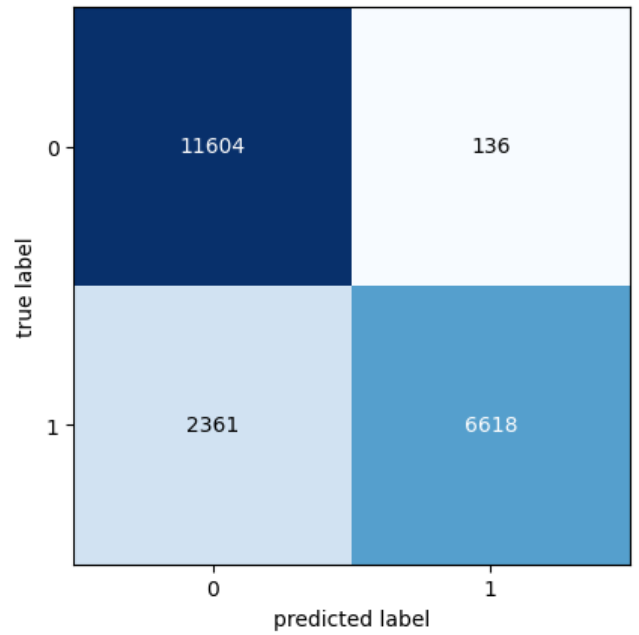
```
# afișarea matricei de confuzie
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

conf = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(conf)
```

```python
# afișarea matricei de confuzie
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

conf = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(conf)
```

```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='predicted label', ylabel='true label'>)
```



Din metricile de mai sus putem observa că modelul este sensibil la fals negative, fiind un număr semnificativ de instanțe când pasageri satisfăcuți au fost prezișI ca neutrii sau nesatisfăcuți.

Precizia este foarte bună pentru clasa "satisfied", în timp ce pentru clasa "neutral or dissatisfied" excelează recall-ul.

Acuratețea de **88%** se află între rezultatele obținute cu Regresia Logistică având parametrii default și Random Forest cu optimizarea hiperparametrilor.

7. Creați un proces de streaming (din orice sursă) folosind Spark Streaming

În continuare vom folosi clasificatorul cu ajutorul căruia am obținut cele mai bune performanțe (**Random Forest**) pentru a prezice **în timp real** satisfacția pasagerilor din setul de date de test față de compania aeriană.

Împărțim setul de date de test în 10 și salvăm fișierele într-un folder separat pentru a simula procesul de streaming.

```
# repartiționăm datele de test în 10 și salvăm fișierele create
test_data = test_airline_data.repartition(10)

import os
import shutil

dir_path = "/content/drive/MyDrive/Proiect Big Data/streaming/"

if os.path.exists(dir_path):
  shutil.rmtree(dir_path, ignore_errors=True)

test_data.write.format("CSV").option("header", True).save(dir_path)

# definirea schemei pentru procesul de streaming
from pyspark.sql.types import StructType, StructField, IntegerType,
StringType, DoubleType

schema = StructType(
   [StructField("Gender", StringType(), True),
    StructField("Customer Type", StringType(), True),
    StructField("Age", IntegerType(), True),
    StructField("Type of Travel", StringType(), True),
    StructField("Class", StringType(), True),
    StructField("Flight Distance", IntegerType(), True),
    StructField("Inflight wifi service", IntegerType(), True),
    StructField("Departure/Arrival time convenient", IntegerType(), True),
    StructField("Ease of Online booking", IntegerType(), True),
    StructField("Gate location", IntegerType(), True),
    StructField("Food and drink", IntegerType(), True),
    StructField("Online boarding", IntegerType(), True),
    StructField("Seat comfort", IntegerType(), True),
    StructField("Inflight entertainment", IntegerType(), True),
    StructField("On-board service", IntegerType(), True),
    StructField("Leg room service", IntegerType(), True),
    StructField("Baggage handling", IntegerType(), True),
    StructField("Checkin service", IntegerType(), True),
    StructField("Inflight service", IntegerType(), True),
    StructField("Cleanliness", IntegerType(), True),
    StructField("Departure Delay in Minutes", IntegerType(), True),
    StructField("Arrival Delay in Minutes", DoubleType(), True),
    StructField("satisfaction", StringType(), True),
    StructField("label", IntegerType(), True)])
```

Apoi citim datele sub formă de stream, aplicăm modelul antrenat pe ele și verificăm că într-adevăr a fost definit un proces de streaming.

```
sourceStream = spark.readStream.schema(schema) \
                               .option("maxFilesPerTrigger", 1) \
                               .csv(dir_path, header=True)

streamingPredictions = rf_fit.transform(sourceStream)

streamingPredictions.isStreaming
```

```
1 sourceStream = spark.readStream.schema(schema) \
2                             .option("maxFilesPerTrigger", 1) \
3                             .csv(dir_path, header=True)
4
5 streamingPredictions = rf_fit.transform(sourceStream)
6
7 streamingPredictions.isStreaming
```

```
True
```

Urmează evaluarea modelului și afișarea predicțiilor. Pentru a scoate în evidență procesul de streaming vom aștepta 10 secunde între apelurile modelului și vom afișa de fiecare dată ultimele 5 predicții și numărul de linii ale DataFrame-ului.

```
# efectuarea predicțiilor
query =
streamingPredictions.writeStream.format("memory").queryName("predictions")
.start()

from pyspark.sql.functions import monotonically_increasing_id
import time

time.sleep(10)

pred = spark.sql("SELECT * FROM predictions")

pred.withColumn("index", monotonically_increasing_id()) \
    .orderBy(col("index").desc()).drop("index") \
    .select(["Gender", "Customer Type", "Age", "Type of Travel", "Class",
"label", "probability", "prediction"]).show(5)

print(f"\nNumber of lines: {pred.count()}\nCurrent AUC:
{rf_eval.evaluate(pred)}")
```

```
from pyspark.sql.functions import monotonically_increasing_id
import time

time.sleep(10)

pred = spark.sql("SELECT * FROM predictions")

pred.withColumn("index", monotonically_increasing_id()) \
    .orderBy(col("index").desc()).drop("index") \
    .select(["Gender", "Customer Type", "Age", "Type of Travel", "Class", "label", "probability", "prediction"]).show(5)

print(f"\nNumber of lines: {pred.count()}\nCurrent AUC: {rf_eval.evaluate(pred)}")
```

```
+------+-------------+---+--------------+--------+-----+--------------------+----------+
|Gender| Customer Type|Age| Type of Travel|   Class|label|         probability|prediction|
+------+-------------+---+--------------+--------+-----+--------------------+----------+
|Female|Loyal Customer| 34|Business travel|Business|    1|[0.00675710154727...|       1.0|
|  Male|Loyal Customer| 35|Business travel|     Eco|    1|[0.15491418905770...|       1.0|
|Female|Loyal Customer| 49|Personal Travel|     Eco|    0|[0.99935691604687...|       0.0|
|  Male|Loyal Customer| 28|Personal Travel|Business|    0|[0.95385580357144...|       0.0|
|Female|Loyal Customer| 52|Personal Travel|Business|    0|[0.96886090550094...|       0.0|
+------+-------------+---+--------------+--------+-----+--------------------+----------+
only showing top 5 rows


Number of lines: 6240
Current AUC: 0.9582375174930883
```

```
time.sleep(10)

pred = spark.sql("SELECT * FROM predictions")

pred.withColumn("index", monotonically_increasing_id()) \
    .orderBy(col("index").desc()).drop("index") \
    .select(["Gender", "Customer Type", "Age", "Type of Travel", "Class",
"label", "probability", "prediction"]).show(5)

print(f"\nNumber of lines: {pred.count()}\nCurrent AUC:
{rf_eval.evaluate(pred)}")
```

```
time.sleep(10)

pred = spark.sql("SELECT * FROM predictions")

pred.withColumn("index", monotonically_increasing_id()) \
    .orderBy(col("index").desc()).drop("index") \
    .select(["Gender", "Customer Type", "Age", "Type of Travel", "Class", "label", "probability", "prediction"]).show(5)

print(f"\nNumber of lines: {pred.count()}\nCurrent AUC: {rf_eval.evaluate(pred)}")
```

```
+------+-------------+---+--------------+--------+-----+--------------------+----------+
|Gender| Customer Type|Age| Type of Travel|   Class|label|         probability|prediction|
+------+-------------+---+--------------+--------+-----+--------------------+----------+
|Female|Loyal Customer| 79|Business travel|Business|    0|[0.95080963005910...|       0.0|
|Female|Loyal Customer| 34|Personal Travel|     Eco|    0|[0.98937353239300...|       0.0|
|Female|Loyal Customer| 33|Business travel|Business|    1|[0.00400980220227...|       1.0|
|Female|Loyal Customer| 15|Business travel|Business|    0|[0.91237485695802...|       0.0|
|  Male|Loyal Customer|  8|Personal Travel|Business|    1|[0.55988486601351...|       0.0|
+------+-------------+---+--------------+--------+-----+--------------------+----------+
only showing top 5 rows


Number of lines: 15601
Current AUC: 0.9555831224815547
```

```
time.sleep(10)

pred = spark.sql("SELECT * FROM predictions")

pred.withColumn("index", monotonically_increasing_id()) \
    .orderBy(col("index").desc()).drop("index") \
    .select(["Gender", "Customer Type", "Age", "Type of Travel", "Class",
"label", "probability", "prediction"]).show(5)

print(f"\nNumber of lines: {pred.count()}\nCurrent AUC:
{rf_eval.evaluate(pred)}")
```

```
1 time.sleep(10)
2
3 pred = spark.sql("SELECT * FROM predictions")
4
5 pred.withColumn("index", monotonically_increasing_id()) \
6     .orderBy(col("index").desc()).drop("index") \
7     .select(["Gender", "Customer Type", "Age", "Type of Travel", "Class", "label", "probability", "prediction"]).show(5)
8
9 print(f"\nNumber of lines: {pred.count()}\nCurrent AUC: {rf_eval.evaluate(pred)}")
```

```
+------+----------------+---+---------------+--------+-----+--------------------+----------+
|Gender|   Customer Type|Age| Type of Travel|   Class|label|         probability|prediction|
+------+----------------+---+---------------+--------+-----+--------------------+----------+
|  Male|  Loyal Customer| 34|Personal Travel|     Eco|    0|[0.99901953384427...|       0.0|
|  Male|  Loyal Customer| 40|Business travel|Business|    0|[0.64287888443651...|       0.0|
|  Male|disloyal Customer| 26|Business travel|     Eco|    0|[0.89145069444397...|       0.0|
|Female|  Loyal Customer| 49|Personal Travel|     Eco|    0|[0.98666903019781...|       0.0|
|  Male|  Loyal Customer| 57|Business travel|     Eco|    0|[0.98354241106963...|       0.0|
+------+----------------+---+---------------+--------+-----+--------------------+----------+
only showing top 5 rows


Number of lines: 21841
Current AUC: 0.9567474969906583
```

```
query.stop()
```

Observăm că pe parcursul procesului de streaming se adaugă noi înregistrări asupra cărora se efectuează predicții, metrica AUC învârtindu-se în permanență în jurul **0.95**.