Distributed Big Data Systems - Final Project

by Joris LIMONIER

This notebook uses PySpark to predict how important the traffic is on the B40 road in Luxembourg. The data set is available here and should be in the same directory as this script, named as "datexDataB40.csv".

In [ ]:
```
import numpy as np
import pandas as pd
import plotly.express as px
from pyspark.ml import Pipeline
from pyspark.ml.feature import (
    OneHotEncoder,
    StandardScaler,
    StringIndexer,
    VectorAssembler,
)
from pyspark.ml.stat import Correlation
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, isnan, to_timestamp, when
from pyspark.sql.types import DoubleType
```

In [ ]:
```
spark = SparkSession.builder.getOrCreate()
spark.sparkContext
```

```
22/01/23 13:10:16 WARN Utils: Your hostname, joris-N751JK resolves to a loopback address: 127.0.1.1; using 192.168.1.
84 instead (on interface wlp3s0f0)
22/01/23 13:10:16 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/home/joris/.local/lib/python3.10/site-p
ackages/pyspark/jars/spark-unsafe_2.12-3.2.0.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/01/23 13:10:17 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
22/01/23 13:10:18 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
```

Out[ ]:

SparkContext

Spark UI

Version	v3.2.0
Master	local[*]
AppName	pyspark-shell

In [ ]:

```
col_names = [
    "id",
    "time",
    "latitude",
    "longitude",
    "direction",
    "road",
    "traffic_status",
    "avg_vehicle_speed",
    "vehicle_flow_rate",
    "traffic_concentration",
]

df = spark.read.option("delimiter", ";").csv("datexDataB40.csv")
df = df.toDF(*col_names)
df.show(2)
```

```
+------------+--------------------+---------+---------+---------------+----+-------------+-----------------+-------
---------+--------------------+
|          id|                time| latitude|longitude|      direction|road|traffic_status|avg_vehicle_speed|vehicle
_flow_rate|traffic_concentration|
+------------+--------------------+---------+---------+---------------+----+-------------+-----------------+-------
---------+--------------------+
|B40.HR.18089|2019-11-19T07:39:...|49.493904|5.9472966|outboundFromTown| B40|     congested|             58.0|
114|                 0.0|
|B40.HR.18260|2019-11-19T07:39:...|49.493275|5.9494343|outboundFromTown| B40|     congested|             61.0|
108|                 1.0|
+------------+--------------------+---------+---------+---------------+----+-------------+-----------------+-------
---------+--------------------+
only showing top 2 rows
```

In [ ]:
```
NUM_COL = [
    "avg_vehicle_speed",
    "vehicle_flow_rate",
    "traffic_concentration",
]
CAT_COLS_PRED = ["direction", "traffic_status"]
```

Exploratory Data Analysis (EDA)

We perform EDA on the whole data set, then we will perform a train-test split\ Show a summary of the data

In [ ]:
```
df.summary().show()
```

```
[Stage 2:======================>                                  (2 + 3) / 5]
```

```
+-------+--------------+--------------------+--------------------+--------------------+------------------+------+---
-----------+-----------------+------------------+---------------------+
|summary|            id|                time|            latitude|           longitude|         direction|  road|tra
ffic_status|avg_vehicle_speed| vehicle_flow_rate|traffic_concentration|
+-------+--------------+--------------------+--------------------+--------------------+------------------+------+---
-----------+-----------------+------------------+---------------------+
|  count|        177540|              177540|              177540|              177540|            177540|177540|
177540|           177540|            177540|               177540|
|   mean|          null|                null|  49.494298100002844|   5.947551749999225|              null|  null|
null| 69.1564877967135| 328.4184409147234|   1.1354741680373115|
| stddev|          null|                null|0.001950153663454...|0.003756293765149...|              null|  null|
null|8.522008549670891|310.48324508150665|   1.6174325195370227|
|    min|   B40.HR.18089|2019-11-19T07:39:...|            49.49162|            5.942809|inboundTowardsTown|   B40|
congested|              0.0|                   0|                 0.0|
|    25%|          null|                null|           49.493275|            5.944498|              null|  null|
null|             63.0|              91.0|                 0.0|
|    50%|          null|                null|           49.493904|           5.9472966|              null|  null|
null|             68.0|             249.0|                 1.0|
|    75%|          null|                null|           49.495125|           5.9494743|              null|  null|
null|             74.0|             434.0|                 2.0|
|    max|B40.RTMIH.17853|2019-12-26T09:34:...|            49.49751|            5.953599|  outboundFromTown|   B40|
unknown|             null|                999|                null|
+-------+--------------+--------------------+--------------------+--------------------+------------------+------+---
-----------+-----------------+------------------+---------------------+
```

In [ ]:

```
df.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- time: string (nullable = true)
 |-- latitude: string (nullable = true)
 |-- longitude: string (nullable = true)
 |-- direction: string (nullable = true)
 |-- road: string (nullable = true)
 |-- traffic_status: string (nullable = true)
 |-- avg_vehicle_speed: string (nullable = true)
 |-- vehicle_flow_rate: string (nullable = true)
 |-- traffic_concentration: string (nullable = true)
```

Let's see how many unique values are in each column.

In [ ]:

```
for col_name in df.columns:
    unique_val = df.select(col_name).distinct().collect()
    print(f"--> {col_name}")
    print(f"\tunique values count: {len(unique_val)}")
    if len(unique_val) <= 1000:
        print(f"\tunique values: {[val[col_name] for val in unique_val]}")
```

```
--> id
        unique values count: 10
        unique values: ['B40.HTMIR.17553', 'B40.RH.18089', 'B40.RTMIH.17553', 'B40.HR.18260', 'B40.RH.18610', 'B40.HT
MIR.17853', 'B40.HR.18089', 'B40.RH.18260', 'B40.RTMIH.17853', 'B40.HR.18610']
--> time
        unique values count: 10658
--> latitude
        unique values count: 10
        unique values: ['49.491695', '49.49751', '49.4975', '49.49506', '49.495125', '49.49162', '49.493275', '49.493
332', '49.49396', '49.493904']
--> longitude
        unique values count: 10
        unique values: ['5.9472966', '5.9473367', '5.944604', '5.9494343', '5.953512', '5.942809', '5.944498', '5.949
4743', '5.953599', '5.9429536']
--> direction
        unique values count: 2
        unique values: ['inboundTowardsTown', 'outboundFromTown']
--> road
        unique values count: 1
        unique values: ['B40']
--> traffic_status
        unique values count: 4
        unique values: ['unknown', 'congested', 'freeFlow', 'impossible']
--> avg_vehicle_speed
        unique values count: 121
        unique values: ['102.0', '84.0', '145.0', '75.0', '50.0', '65.0', '66.0', '90.0', '67.0', '47.0', '83.0', '6
3.0', '44.0', '69.0', '109.0', '25.0', '120.0', '108.0', '78.0', '72.0', '59.0', '110.0', '41.0', '51.0', '52.0', '6
4.0', '48.0', '29.0', '71.0', '104.0', '46.0', '106.0', '70.0', '111.0', '114.0', '121.0', '101.0', '35.0', '100.0',
'32.0', '81.0', '73.0', '58.0', '112.0', '99.0', '77.0', '79.0', '68.0', '57.0', '105.0', '85.0', '55.0', '82.0', '10
3.0', '93.0', '53.0', '113.0', '60.0', '31.0', '95.0', '61.0', '33.0', '91.0', '86.0', '56.0', '49.0', '136.0', '88.
0', '89.0', '54.0', '98.0', '76.0', '159.0', '62.0', '87.0', '107.0', '94.0', '92.0', '117.0', '43.0', '97.0', '124.
0', '80.0', '74.0', 'null', '96.0', '116.0', '38.0', '118.0', '45.0', '26.0', '125.0', '12.0', '37.0', '23.0', '34.
0', '27.0', '143.0', '28.0', '135.0', '30.0', '39.0', '128.0', '129.0', '16.0', '42.0', '9.0', '24.0', '131.0', '140.
0', '123.0', '40.0', '137.0', '21.0', '163.0', '36.0', '139.0', '0.0', '126.0', '115.0', '134.0']
--> vehicle_flow_rate
        unique values count: 1704
--> traffic_concentration
        unique values count: 37
        unique values: ['1.0', '25.0', '17.0', '0.0', '9.0', '12.0', '18.0', '10.0', '19.0', '5.0', '6.0', '4.0', '7.
0', '11.0', '2.0', '8.0', '3.0', 'null', '13.0', '20.0', '15.0', '29.0', '14.0', '21.0', '16.0', '22.0', '26.0', '34.
0', '24.0', '28.0', '31.0', '30.0', '33.0', '40.0', '23.0', '35.0', '27.0']
```

We look for missing values

In [ ]:
```
def show_unknown_counts():
    df.select(
        [
            count(
                when(
                    isnan(c)
                    | col(c).isNull()
                    | (col(c) == "null")
                    | (col(c) == "unknown"),
                    c,
                )
            ).alias(c)
            for c in df.columns
        ]
    ).show()


show_unknown_counts()
```

```
[Stage 35:============>                                            (1 + 4) / 5]
+---+----+--------+---------+---------+----+--------------+-----------------+-----------------+----------------------+
| id|time|latitude|longitude|direction|road|traffic_status|avg_vehicle_speed|vehicle_flow_rate|traffic_concentration|
+---+----+--------+---------+---------+----+--------------+-----------------+-----------------+----------------------+
|  0|   0|       0|        0|        0|   0|         10309|            10249|                0|                     8|
+---+----+--------+---------+---------+----+--------------+-----------------+-----------------+----------------------+
```

We deal with missing values in the `traffic_status` .

In [ ]:
```
df.groupby("traffic_status").count().show()
print(f"Total number of rows in the data set: {df.count()}")
```

```
+--------------+------+
|traffic_status| count|
+--------------+------+
|       unknown| 10309|
|     congested|148323|
|      freeFlow| 18904|
|    impossible|     4|
+--------------+------+

Total number of rows in the data set: 177540
```

We see that there are 10309 "unkown" rows for the column `traffic_status` . This represents $\approx 5.8\%$ of the total number of rows. \ It is probably safe to drop these rows since the rest of the data is fairly clean. We also remove the 4 "impossible" values, since they are not numerous enough to make a good classifier.

Note: We could have set the `nullValue` parameter appropriately when reading the csv file initially, but we kept this somewhat less elegant technique to reflect that we initially didn't know the data set and that it is the EDA that allowed us to notice this kind of unexpected issues.

In [ ]:
```
df = df.filter(df["traffic_status"] != "unknown")
df = df.filter(df["traffic_status"] != "impossible")
df.groupby("traffic_status").count().show()
show_unknown_counts()
```

```
+--------------+------+
|traffic_status| count|
+--------------+------+
|     congested|148323|
|      freeFlow| 18904|
+--------------+------+


[Stage 47:============>                                    (1 + 4) / 5]
+---+----+--------+---------+---------+----+--------------+-----------------+-----------------+----------------------+
| id|time|latitude|longitude|direction|road|traffic_status|avg_vehicle_speed|vehicle_flow_rate|traffic_concentration|
+---+----+--------+---------+---------+----+--------------+-----------------+-----------------+----------------------+
|  0|   0|       0|        0|        0|   0|             0|                0|                0|                     0|
+---+----+--------+---------+---------+----+--------------+-----------------+-----------------+----------------------+
```

Actually, removing the "unknown" values from the `traffic_status` also removed the ones from `avg_vehicle_speed` (*i.e.* they were on the same rows)\ We notice that the only road is "B40", so we can drop it.

In [ ]:
```
def drop_if_exists(col):
    """
    Drop the `col` column from `df` if it is in its columns.
    This avoids errors on cell re-run.
    """
    global df
    if col in df.columns:
        df = df.drop(col)


drop_if_exists("road")
```

The number of unique values for `id`, `latitude` and `longitude` is the same (10). \ We strongly suspect that for a given id, the latitude and longitude are always the same. Let's check it.

In [ ]:
```
unique_ids = df.select("id").distinct().collect()
# unique triples (`id`, `latitude`, `longitude`)
unique_triples = df.select("id", "latitude", "longitude").distinct().collect()

print(len(unique_triples) == len(unique_ids) == 10)
```

```
True
```

So an `id` represents a unique camera in a unique location, therefore it has a unique pair (`latitude`, `longitude`).\ This means that the `latitude` and `longitude` columns are redundant, hence we can drop them.

In [ ]:
```
drop_if_exists("latitude")
drop_if_exists("longitude")
```

In [ ]:
```
for num_col in NUM_COL:
    df = df.withColumn(num_col, df[num_col].cast("double"))

# plot
df_num = df.select(NUM_COL + ["traffic_status"]).toPandas()
for i in range(len(NUM_COL)):
    for j in range(i):
        col_x = NUM_COL[i]
        col_y = NUM_COL[j]
        px.scatter(
```

```
            data_frame=df_num,
            x=col_x,
            y=col_y,
            color="traffic_status",
        ).show()
```

There doesn't appear to be (much) correlation between the numerical columns. Interestingly enough however, we noticed that the `traffic_concentration` column seems to be, if not categorical, at least fairly discrete.\ Additionally, we see that the `traffic_status` column is (probably) derived directly from the `avg_vehicle_speed`, so we will remove it for prediction, otherwise, it would be too easy to predict the `traffic_status` column.

In [ ]:
```
df_corr = df.select(NUM_COL)
num_vector_col = "corr_features"
corr_assembler = VectorAssembler(
    inputCols=df_corr.columns,
    outputCol=num_vector_col,
)
df_vect = corr_assembler.transform(df_corr).select(num_vector_col)
Correlation.corr(df_vect, num_vector_col).show(truncate=False)
```

```
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
|pearson(corr_features)
|
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
|1.0                   -0.05410283576509373  -0.33690329840996186  \n-0.05410283576509373  1.0                   0.73
34623348484666    \n-0.33690329840996186  0.7334623348484666    1.0                   |
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+


/home/joris/.local/lib/python3.10/site-packages/pyspark/sql/context.py:125: FutureWarning:

Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
```

We see a strong correlation, but the columns studied are not linearly dependendent. We do not have hundreds of features so we decide not to

drop any.\ Now we deal with the `time` column.

In [ ]:
```
df.select("time").show(2, truncate=False)
```

```
+---------------------------+
|time                       |
+---------------------------+
|2019-11-19T07:39:00.000+01:00|
|2019-11-19T07:39:00.000+01:00|
+---------------------------+
only showing top 2 rows
```

We convert the time column (currently `str`) to a datetime object

In [ ]:
```
df = df.withColumn(
    colName="datetime",
    col=to_timestamp(df.time),
)

df.show(2)
```

```
+------------+--------------------+----------------+-------------+-----------------+-----------------+-------------
-------+-------------------+
|          id|                time|       direction|traffic_status|avg_vehicle_speed|vehicle_flow_rate|traffic_concen
tration|           datetime|
+------------+--------------------+----------------+-------------+-----------------+-----------------+-------------
-------+-------------------+
|B40.HR.18089|2019-11-19T07:39:...|outboundFromTown|     congested|             58.0|            114.0|
0.0|2019-11-19 07:39:00|
|B40.HR.18260|2019-11-19T07:39:...|outboundFromTown|     congested|             61.0|            108.0|
1.0|2019-11-19 07:39:00|
+------------+--------------------+----------------+-------------+-----------------+-----------------+-------------
-------+-------------------+
only showing top 2 rows
```

Now we split the datetime object into several of its compenents.

In [ ]:
```
from pyspark.sql.functions import (
    dayofmonth,
    dayofweek,
```

```
    dayofyear,
    hour,
    minute,
    month,
    weekofyear,
    year,
)

time_props = [
    dayofweek,
    dayofyear,
    dayofmonth,
    hour,
    minute,
    month,
    weekofyear,
    year,
]

if "datetime" in df.columns:  # prevent errors on cell re-run
    for time_prop in time_props:
        df = df.withColumn(
            colName=time_prop.__name__,
            col=time_prop(col("datetime")),
        )
        df.groupby(time_prop.__name__).count().show()
```

```
+---------+-----+
|dayofweek|count|
+---------+-----+
|        1|23183|
|        6|22978|
|        3|25321|
|        5|24200|
|        4|26613|
|        7|23371|
|        2|21561|
+---------+-----+

+---------+-----+
|dayofyear|count|
+---------+-----+
|      329| 4387|
|      330| 4419|
|      325| 4461|
|      328| 4605|
|      327| 4713|
|      324| 4496|
|      326| 4586|
|      331| 4526|
|      323| 3244|
|      332| 4474|
|      336| 3904|
|      333| 4570|
|      339| 4548|
|      335| 4619|
|      338| 4012|
|      334| 4747|
|      337| 4037|
|      346| 4551|
|      340| 4571|
|      347| 4594|
+---------+-----+
only showing top 20 rows

+----------+-----+
|dayofmonth|count|
+----------+-----+
|        26| 5954|
|        27| 4526|
|        22| 9238|
```

```
|        20| 9153|
|        19| 7875|
|        23| 9231|
|        25| 8853|
|        24| 9170|
|        21| 8935|
|        28| 4474|
|         1| 4619|
|         3| 4037|
|         5| 4548|
|         4| 4012|
|        29| 4570|
|         2| 3904|
|        30| 4747|
|        12| 4551|
|        13| 4594|
|         6| 4571|
+----------+-----+
only showing top 20 rows

+----+-----+
|hour|count|
+----+-----+
|  12| 7150|
|  22| 7229|
|   1| 5951|
|  13| 7160|
|  16| 7226|
|   6| 7353|
|   3| 5475|
|  20| 7369|
|   5| 7043|
|  19| 7397|
|  15| 7158|
|   9| 7405|
|  17| 7400|
|   4| 6234|
|   8| 7490|
|  23| 7054|
|   7| 7312|
|  10| 7083|
|  21| 7367|
|  11| 7112|
+----+-----+
```

```
only showing top 20 rows

+------+-----+
|minute|count|
+------+-----+
|    34|16920|
|    44| 8350|
|    54|16716|
|    19|16872|
|     9|16680|
|    59| 8264|
|     4|16578|
|    39|16670|
|    49|16708|
|    24|16692|
|    29| 8436|
|    14| 8324|
|     1|    7|
|    51|   10|
+------+-----+

+-----+------+
|month| count|
+-----+------+
|   11| 53228|
|   12|113999|
+-----+------+

+----------+-----+
|weekofyear|count|
+----------+-----+
|        47|26105|
|        48|31742|
|        49|30427|
|        50|31975|
|        51|31894|
|        52|15084|
+----------+-----+

+----+------+
|year| count|
+----+------+
|2019|167227|
```

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

We see that the `year` column has only one value (2019). It doesn't bring any extra information so we drop it.\ We also drop the `time` and `datetime` columns.

In [ ]:

```
time_props = [time_prop for time_prop in time_props if time_prop != year]
TIME_COLS = [time_prop.__name__ for time_prop in time_props]

df.show(2)
drop_if_exists("year")
drop_if_exists("time")
drop_if_exists("datetime")
df.show(2)
```

```
+------------+--------------------+----------------+--------------+-----------------+-----------------+--------------
-------+------------------+---------+---------+----------+----+------+-----+----------+----+
|          id|                time|       direction|traffic_status|avg_vehicle_speed|vehicle_flow_rate|traffic_concen
tration|          datetime|dayofweek|dayofyear|dayofmonth|hour|minute|month|weekofyear|year|
+------------+--------------------+----------------+--------------+-----------------+-----------------+--------------
-------+------------------+---------+---------+----------+----+------+-----+----------+----+
|B40.HR.18089|2019-11-19T07:39:...|outboundFromTown|     congested|             58.0|            114.0|
0.0|2019-11-19 07:39:00|        3|      323|        19|   7|    39|   11|        47|2019|
|B40.HR.18260|2019-11-19T07:39:...|outboundFromTown|     congested|             61.0|            108.0|
1.0|2019-11-19 07:39:00|        3|      323|        19|   7|    39|   11|        47|2019|
+------------+--------------------+----------------+--------------+-----------------+-----------------+--------------
-------+------------------+---------+---------+----------+----+------+-----+----------+----+
only showing top 2 rows

+------------+----------------+--------------+-----------------+-----------------+--------------------+---------+---
------+----------+----+------+-----+----------+
|          id|       direction|traffic_status|avg_vehicle_speed|vehicle_flow_rate|traffic_concentration|dayofweek|day
ofyear|dayofmonth|hour|minute|month|weekofyear|
+------------+----------------+--------------+-----------------+-----------------+--------------------+---------+---
------+----------+----+------+-----+----------+
|B40.HR.18089|outboundFromTown|     congested|             58.0|            114.0|                 0.0|        3|
323|        19|   7|    39|   11|        47|
|B40.HR.18260|outboundFromTown|     congested|             61.0|            108.0|                 1.0|        3|
323|        19|   7|    39|   11|        47|
+------------+----------------+--------------+-----------------+-----------------+--------------------+---------+---
------+----------+----+------+-----+----------+
only showing top 2 rows
```

We drop the `avg_vehicle_speed` as mentioned before

In [ ]:
```
drop_if_exists("avg_vehicle_speed")
```

Classification

Perform train-test split

In [ ]:
```
train, test = df.randomSplit(weights=[0.8, 0.2], seed=42)
print(f"Number of observations in the train set: {train.count()}")
print(f"Number of observations in the test set: {test.count()}")
```

```
Number of observations in the train set: 133834
[Stage 92:============>                                            (1 + 4) / 5]
Number of observations in the test set: 33393
```

In [ ]:
```
CAT_COLS_PRED = TIME_COLS + ["direction", "id"]
NUM_COLS_PRED = ["vehicle_flow_rate", "traffic_concentration"]
TARGET_COL = "traffic_status"

missing_cols = [
    missing_col
    for missing_col in df.columns
    if missing_col not in CAT_COLS_PRED + NUM_COLS_PRED + [TARGET_COL]
]
if missing_cols == []:
    print("All columns are planned for classification")
else:
    print(f"{missing_cols} are not yet planned")
```

```
All columns are planned for classification
```

Preprocess categorical columns

We OneHotEncode the categorical columns

In [ ]:

```
CAT_COLS_INDEXER = [f"{cat_col}_indexer" for cat_col in CAT_COLS_PRED]
CAT_COLS_ONEHOT = [f"{cat_col}_vec" for cat_col in CAT_COLS_PRED]

cat_stages = [
    StringIndexer(
        inputCols=CAT_COLS_PRED,
        outputCols=CAT_COLS_INDEXER,
    ),
    OneHotEncoder(
        inputCols=CAT_COLS_INDEXER,
        outputCols=CAT_COLS_ONEHOT,
        dropLast=True,
    ),
]

# Show the effect of the categorical stages
Pipeline(stages=cat_stages).fit(train).transform(train).show(2)
```

```
22/01/23 13:11:10 WARN package: Truncated the string representation of a plan since it was too large. This behavior c
an be adjusted by setting 'spark.sql.debug.maxToStringFields'.
```

```
+------------+---------------+-------------+-----------------+--------------------+---------+---------+----------+
----+------+-----+----------+-----------------+-----------------+------------------+------------+--------------+-----
--------+------------------+-----------------+----------+-------------+---------------+--------------+--------------
+--------------+---------+--------------+-------------+-------------+
|          id|       direction|traffic_status|vehicle_flow_rate|traffic_concentration|dayofweek|dayofyear|dayofmonth|
hour|minute|month|weekofyear|dayofweek_indexer|dayofyear_indexer|dayofmonth_indexer|hour_indexer|minute_indexer|month
_indexer|weekofyear_indexer|direction_indexer|id_indexer|dayofweek_vec|  dayofyear_vec|dayofmonth_vec|       hour_vec
|    minute_vec|month_vec|weekofyear_vec|direction_vec|       id_vec|
+------------+---------------+-------------+-----------------+--------------------+---------+---------+----------+
----+------+-----+----------+-----------------+-----------------+------------------+------------+--------------+-----
--------+------------------+-----------------+----------+-------------+---------------+--------------+--------------
+--------------+---------+--------------+-------------+-------------+
|B40.HR.18089|outboundFromTown|     congested|              5.0|                 0.0|        2|      329|        25|
4|    19|   11|        48|              6.0|             31.0|               5.0|        20.0|           1.0|
1.0|               2.0|              0.0|       3.0|    (6,[],[])|(37,[31],[1.0])|(29,[5],[1.0])|(23,[20],[1.0])|(13,
[1],[1.0])|(1,[],[])| (5,[2],[1.0])|(1,[0],[1.0])|(9,[3],[1.0])|
|B40.HR.18089|outboundFromTown|     congested|              5.0|                 0.0|        2|      329|        25|
4|    19|   11|        48|              6.0|             31.0|               5.0|        20.0|           1.0|
1.0|               2.0|              0.0|       3.0|    (6,[],[])|(37,[31],[1.0])|(29,[5],[1.0])|(23,[20],[1.0])|(13,
[1],[1.0])|(1,[],[])| (5,[2],[1.0])|(1,[0],[1.0])|(9,[3],[1.0])|
+------------+---------------+-------------+-----------------+--------------------+---------+---------+----------+
----+------+-----+----------+-----------------+-----------------+------------------+------------+--------------+-----
--------+------------------+-----------------+----------+-------------+---------------+--------------+--------------
+--------------+---------+--------------+-------------+-------------+
only showing top 2 rows
```

Preprocess numerical columns

We scale the numerical columns

In [ ]:

```
num_stages = [
    VectorAssembler(
        inputCols=NUM_COLS_PRED,
        outputCol="assembled_num",
    ),
    StandardScaler(
        inputCol="assembled_num",
        outputCol="scaled_num",
    ),
]
```

```
# Show the effect of the categorical stages
Pipeline(stages=num_stages).fit(train).transform(train).show(20, truncate=False)
```

```
+------------+---------------+-------------+-----------------+---------------------+---------+---------+----------+
----+------+-----+----------+-------------+-------------------------+
|id          |direction      |traffic_status|vehicle_flow_rate|traffic_concentration|dayofweek|dayofyear|dayofmonth|
hour|minute|month|weekofyear|assembled_num|scaled_num               |
+------------+---------------+-------------+-----------------+---------------------+---------+---------+----------+
----+------+-----+----------+-------------+-------------------------+
|B40.HR.18089|outboundFromTown|congested     |5.0              |0.0                  |2        |329      |25        |
4   |19    |11   |48        |[5.0,0.0]    |[0.016142285616544197,0.0]|
|B40.HR.18089|outboundFromTown|congested     |5.0              |0.0                  |2        |329      |25        |
4   |19    |11   |48        |[5.0,0.0]    |[0.016142285616544197,0.0]|
|B40.HR.18089|outboundFromTown|congested     |5.0              |0.0                  |2        |329      |25        |
4   |24    |11   |48        |[5.0,0.0]    |[0.016142285616544197,0.0]|
|B40.HR.18089|outboundFromTown|congested     |7.0              |0.0                  |5        |325      |21        |
4   |19    |11   |47        |[7.0,0.0]    |[0.022599199863161876,0.0]|
|B40.HR.18089|outboundFromTown|congested     |7.0              |0.0                  |5        |325      |21        |
4   |19    |11   |47        |[7.0,0.0]    |[0.022599199863161876,0.0]|
|B40.HR.18089|outboundFromTown|congested     |8.0              |0.0                  |5        |325      |21        |
4   |29    |11   |47        |[8.0,0.0]    |[0.025827656986470715,0.0]|
|B40.HR.18089|outboundFromTown|congested     |9.0              |0.0                  |5        |325      |21        |
4   |39    |11   |47        |[9.0,0.0]    |[0.029056114109779554,0.0]|
|B40.HR.18089|outboundFromTown|congested     |10.0             |0.0                  |4        |324      |20        |
4   |14    |11   |47        |[10.0,0.0]   |[0.032284571233088394,0.0]|
|B40.HR.18089|outboundFromTown|congested     |10.0             |0.0                  |4        |324      |20        |
4   |24    |11   |47        |[10.0,0.0]   |[0.032284571233088394,0.0]|
|B40.HR.18089|outboundFromTown|congested     |10.0             |0.0                  |4        |324      |20        |
4   |24    |11   |47        |[10.0,0.0]   |[0.032284571233088394,0.0]|
|B40.HR.18089|outboundFromTown|congested     |10.0             |0.0                  |5        |325      |21        |
3   |54    |11   |47        |[10.0,0.0]   |[0.032284571233088394,0.0]|
|B40.HR.18089|outboundFromTown|congested     |11.0             |0.0                  |3        |330      |26        |
3   |34    |11   |48        |[11.0,0.0]   |[0.03551302835639723,0.0] |
|B40.HR.18089|outboundFromTown|congested     |11.0             |0.0                  |3        |330      |26        |
3   |34    |11   |48        |[11.0,0.0]   |[0.03551302835639723,0.0] |
|B40.HR.18089|outboundFromTown|congested     |11.0             |0.0                  |4        |324      |20        |
4   |29    |11   |47        |[11.0,0.0]   |[0.03551302835639723,0.0] |
|B40.HR.18089|outboundFromTown|congested     |12.0             |0.0                  |3        |330      |26        |
3   |39    |11   |48        |[12.0,0.0]   |[0.03874148547970607,0.0] |
|B40.HR.18089|outboundFromTown|congested     |12.0             |0.0                  |4        |324      |20        |
2   |44    |11   |47        |[12.0,0.0]   |[0.03874148547970607,0.0] |
|B40.HR.18089|outboundFromTown|congested     |12.0             |0.0                  |4        |324      |20        |
3   |54    |11   |47        |[12.0,0.0]   |[0.03874148547970607,0.0] |
|B40.HR.18089|outboundFromTown|congested     |12.0             |0.0                  |4        |324      |20        |
3   |54    |11   |47        |[12.0,0.0]   |[0.03874148547970607,0.0] |
|B40.HR.18089|outboundFromTown|congested     |12.0             |0.0                  |4        |324      |20        |
4   |19    |11   |47        |[12.0,0.0]   |[0.03874148547970607,0.0] |
```

```
|B40.HR.18089|outboundFromTown|congested     |12.0              |0.0                   |5        |325      |21        |
4   |49    |11   |47        |[12.0,0.0]   |[0.03874148547970607,0.0] |
+------------+---------------+--------------+-----------------+----------------------+---------+---------+----------+
----+------+-----+----------+-------------+--------------------------+
only showing top 20 rows
```

In [ ]:
```
feature_assembler = [
    VectorAssembler(
        inputCols=CAT_COLS_ONEHOT + ["scaled_num"],
        outputCol="features",
    )
]
```

In [ ]:
```
target_stages = [StringIndexer(inputCol=TARGET_COL, outputCol="label")]

# Show the effect of the target_stages
Pipeline(stages=target_stages).fit(train).transform(train).show(2)
```

```
+------------+---------------+--------------+-----------------+----------------------+---------+---------+----------+
----+------+-----+----------+-----+
|          id|       direction|traffic_status|vehicle_flow_rate|traffic_concentration|dayofweek|dayofyear|dayofmonth|
hour|minute|month|weekofyear|label|
+------------+---------------+--------------+-----------------+----------------------+---------+---------+----------+
----+------+-----+----------+-----+
|B40.HR.18089|outboundFromTown|     congested|              5.0|                   0.0|        2|      329|        25|
4|    19|   11|        48|  0.0|
|B40.HR.18089|outboundFromTown|     congested|              5.0|                   0.0|        2|      329|        25|
4|    19|   11|        48|  0.0|
+------------+---------------+--------------+-----------------+----------------------+---------+---------+----------+
----+------+-----+----------+-----+
only showing top 2 rows
```

In [ ]:
```
pipe = Pipeline(stages=cat_stages + num_stages + feature_assembler + target_stages)
preproc_model = pipe.fit(train)
```

In [ ]:
```
preproc_train = preproc_model.transform(train).select("features", "label")
preproc_train.show(2, truncate=False)
```

```
+-----------------------------------------------------------------------------------------+-----+
|features                                                                                 |label|
+-----------------------------------------------------------------------------------------+-----+
|(126,[37,48,92,96,111,114,118,124],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.016142285616544197])|0.0  |
|(126,[37,48,92,96,111,114,118,124],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.016142285616544197])|0.0  |
+-----------------------------------------------------------------------------------------+-----+
only showing top 2 rows
```

In [ ]:
```
preproc_test = preproc_model.transform(test).select("features", "label")
preproc_test.show(2, truncate=False)
```

```
+-----------------------------------------------------------------------------------------+-----+
|features                                                                                 |label|
+-----------------------------------------------------------------------------------------+-----+
|(126,[37,48,92,97,111,114,118,124],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.016142285616544197]) |0.0  |
|(126,[37,48,92,104,111,114,118,124],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.025827656986470715])|0.0  |
+-----------------------------------------------------------------------------------------+-----+
only showing top 2 rows
```

In [ ]:
```
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, CrossValidatorModel, ParamGridBuilder
```

In [ ]:
```
lr = LogisticRegression()
grid = (
    ParamGridBuilder()
    .addGrid(lr.maxIter, [80, 100])
    .addGrid(lr.regParam, [0.0, 1.0])
    .addGrid(lr.elasticNetParam, [0.0, 1.0])
    .build()
)
print(grid)
evaluator = BinaryClassificationEvaluator()
cv = CrossValidator(
    estimator=lr,
    estimatorParamMaps=grid,
```

```
    evaluator=evaluator,
    parallelism=8,
    numFolds=5,
)
cvModel = cv.fit(preproc_train)
print(
    cvModel.avgMetrics
)  # result of cross validation for each combination of parameters
print(
    f"Result of linear regression on the test set: \
        {evaluator.evaluate(cvModel.transform(preproc_test))}"
)
```

```
[{Param(parent='LogisticRegression_29da177c01f2', name='maxIter', doc='max number of iterations (>= 0).'): 80, Param
(parent='LogisticRegression_29da177c01f2', name='regParam', doc='regularization parameter (>= 0).'): 0.0, Param(paren
t='LogisticRegression_29da177c01f2', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. F
or alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0.0}, {Param(parent='LogisticRegre
ssion_29da177c01f2', name='maxIter', doc='max number of iterations (>= 0).'): 80, Param(parent='LogisticRegression_29
da177c01f2', name='regParam', doc='regularization parameter (>= 0).'): 0.0, Param(parent='LogisticRegression_29da177c
01f2', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is a
n L2 penalty. For alpha = 1, it is an L1 penalty.'): 1.0}, {Param(parent='LogisticRegression_29da177c01f2', name='max
Iter', doc='max number of iterations (>= 0).'): 80, Param(parent='LogisticRegression_29da177c01f2', name='regParam',
doc='regularization parameter (>= 0).'): 1.0, Param(parent='LogisticRegression_29da177c01f2', name='elasticNetParam',
doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.0}, {Param(parent='LogisticRegression_29da177c01f2', name='maxIter', doc='max number of iterat
ions (>= 0).'): 80, Param(parent='LogisticRegression_29da177c01f2', name='regParam', doc='regularization parameter (>
= 0).'): 1.0, Param(parent='LogisticRegression_29da177c01f2', name='elasticNetParam', doc='the ElasticNet mixing para
meter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1.0}, {Pa
ram(parent='LogisticRegression_29da177c01f2', name='maxIter', doc='max number of iterations (>= 0).'): 100, Param(par
ent='LogisticRegression_29da177c01f2', name='regParam', doc='regularization parameter (>= 0).'): 0.0, Param(parent='L
ogisticRegression_29da177c01f2', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For a
lpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0.0}, {Param(parent='LogisticRegressio
n_29da177c01f2', name='maxIter', doc='max number of iterations (>= 0).'): 100, Param(parent='LogisticRegression_29da1
77c01f2', name='regParam', doc='regularization parameter (>= 0).'): 0.0, Param(parent='LogisticRegression_29da177c01f
2', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L
2 penalty. For alpha = 1, it is an L1 penalty.'): 1.0}, {Param(parent='LogisticRegression_29da177c01f2', name='maxIte
r', doc='max number of iterations (>= 0).'): 100, Param(parent='LogisticRegression_29da177c01f2', name='regParam', do
c='regularization parameter (>= 0).'): 1.0, Param(parent='LogisticRegression_29da177c01f2', name='elasticNetParam', d
oc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.0}, {Param(parent='LogisticRegression_29da177c01f2', name='maxIter', doc='max number of iterat
ions (>= 0).'): 100, Param(parent='LogisticRegression_29da177c01f2', name='regParam', doc='regularization parameter
(>= 0).'): 1.0, Param(parent='LogisticRegression_29da177c01f2', name='elasticNetParam', doc='the ElasticNet mixing pa
rameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1.0}]
```

```
22/01/23 13:11:22 WARN BlockManager: Block rdd_314_2 already exists on this machine; not re-adding it
22/01/23 13:11:22 WARN BlockManager: Block rdd_314_1 already exists on this machine; not re-adding it
22/01/23 13:11:22 WARN BlockManager: Block rdd_314_0 already exists on this machine; not re-adding it

[0.9141317047568942, 0.91413348583154, 0.8984046629794031, 0.5, 0.9141343130981443, 0.914135903230167, 0.898405550694
878, 0.5]

Result of linear regression on the test set:        0.9156167387910381
```

In [ ]:

```
rf = RandomForestClassifier()
grid = (
    ParamGridBuilder()
    .addGrid(rf.maxDepth, [5, 7, 10])
    .addGrid(rf.maxBins, [8, 16, 32])
    .addGrid(rf.numTrees, [10, 20, 30])
    .build()
)
print(grid)
evaluator = BinaryClassificationEvaluator()
cv = CrossValidator(
    estimator=rf,
    estimatorParamMaps=grid,
    evaluator=evaluator,
    parallelism=8,
    numFolds=5,
)
cvModel = cv.fit(preproc_train)
print(
    cvModel.avgMetrics
)  # result of cross validation for each combination of parameters
print(
    f"Result of random forest on the test set: {evaluator.evaluate(cvModel.transform(preproc_test))}"
)
```

[{Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 8, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 10}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 8, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 8, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 10}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 10}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 5, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 7, Param(parent='RandomForestClassifier_650c430a3c35', name

='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories fo
r any categorical feature.'): 8, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of
trees to train (>= 1).'): 10}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum dep
th of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in rang
e [0, 30].'): 7, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for disc
retizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 8, Param(paren
t='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent
='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1
leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 7, Param(parent='RandomForestCl
assifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 an
d >= number of categories for any categorical feature.'): 8, Param(parent='RandomForestClassifier_650c430a3c35', name
='numTrees', doc='Number of trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name
='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node +
2 leaf nodes. Must be in range [0, 30].'): 7, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc
='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categoric
al feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train
(>= 1).'): 10}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree.
(>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 7,
Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuo
us features. Must be >=2 and >= number of categories for any categorical feature.'): 16, Param(parent='RandomForestC
lassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestCl
assifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; dept
h 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 7, Param(parent='RandomForestClassifier_650c43
0a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of
categories for any categorical feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', d
oc='Number of trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', do
c='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. M
ust be in range [0, 30].'): 7, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of
bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 3
2, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 10},
{Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., de
pth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 7, Param(parent
='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features.
Must be >=2 and >= number of categories for any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c
430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestClassifier_650c4
30a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 in
ternal node + 2 leaf nodes. Must be in range [0, 30].'): 7, Param(parent='RandomForestClassifier_650c430a3c35', name
='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories fo
r any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of
trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum dep
th of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in rang
e [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for dis
cretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 8, Param(pare
nt='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 10}, {Param(paren

t='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 8, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 8, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 10}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 16, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 30}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 10}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 20}, {Param(parent='RandomForestClassifier_650c430a3c35', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 10, Param(parent='RandomForestClassifier_650c430a3c35', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 32, Param(parent='RandomForestClassifier_650c430a3c35', name='numTrees', doc='Number of trees to train (>= 1).'): 30}]

```
22/01/23 13:13:42 WARN BlockManager: Block rdd_4634_1 already exists on this machine; not re-adding it
22/01/23 13:15:31 WARN DAGScheduler: Broadcasting large task binary with size 1388.1 KiB
22/01/23 13:15:32 WARN DAGScheduler: Broadcasting large task binary with size 1383.5 KiB
22/01/23 13:15:33 WARN DAGScheduler: Broadcasting large task binary with size 1334.0 KiB
22/01/23 13:15:35 WARN DAGScheduler: Broadcasting large task binary with size 1330.1 KiB
22/01/23 13:15:37 WARN DAGScheduler: Broadcasting large task binary with size 1936.8 KiB
22/01/23 13:15:37 WARN DAGScheduler: Broadcasting large task binary with size 1930.4 KiB
22/01/23 13:15:56 WARN DAGScheduler: Broadcasting large task binary with size 1328.8 KiB
22/01/23 13:15:58 WARN DAGScheduler: Broadcasting large task binary with size 1387.6 KiB
22/01/23 13:16:00 WARN DAGScheduler: Broadcasting large task binary with size 1914.3 KiB
22/01/23 13:16:05 WARN BlockManager: Block rdd_6900_2 already exists on this machine; not re-adding it
22/01/23 13:18:00 WARN DAGScheduler: Broadcasting large task binary with size 1003.8 KiB
22/01/23 13:18:04 WARN DAGScheduler: Broadcasting large task binary with size 1403.8 KiB
22/01/23 13:18:04 WARN DAGScheduler: Broadcasting large task binary with size 1364.8 KiB
22/01/23 13:18:08 WARN DAGScheduler: Broadcasting large task binary with size 1000.4 KiB
22/01/23 13:18:08 WARN DAGScheduler: Broadcasting large task binary with size 1956.0 KiB
22/01/23 13:18:12 WARN DAGScheduler: Broadcasting large task binary with size 1396.1 KiB
22/01/23 13:18:13 WARN DAGScheduler: Broadcasting large task binary with size 1367.0 KiB
22/01/23 13:18:18 WARN DAGScheduler: Broadcasting large task binary with size 1949.6 KiB
22/01/23 13:18:31 WARN DAGScheduler: Broadcasting large task binary with size 1003.6 KiB
22/01/23 13:18:34 WARN DAGScheduler: Broadcasting large task binary with size 1362.3 KiB
22/01/23 13:18:38 WARN DAGScheduler: Broadcasting large task binary with size 1395.6 KiB
22/01/23 13:18:39 WARN DAGScheduler: Broadcasting large task binary with size 1942.9 KiB
22/01/23 13:18:43 WARN BlockManager: Block rdd_9166_0 already exists on this machine; not re-adding it
22/01/23 13:20:51 WARN DAGScheduler: Broadcasting large task binary with size 1369.0 KiB
22/01/23 13:20:53 WARN DAGScheduler: Broadcasting large task binary with size 1332.6 KiB
22/01/23 13:20:56 WARN DAGScheduler: Broadcasting large task binary with size 1355.6 KiB
22/01/23 13:20:57 WARN DAGScheduler: Broadcasting large task binary with size 1888.3 KiB
22/01/23 13:21:01 WARN DAGScheduler: Broadcasting large task binary with size 1388.6 KiB
22/01/23 13:21:05 WARN DAGScheduler: Broadcasting large task binary with size 1934.5 KiB
22/01/23 13:21:20 WARN DAGScheduler: Broadcasting large task binary with size 1352.5 KiB
22/01/23 13:21:22 WARN DAGScheduler: Broadcasting large task binary with size 1390.4 KiB
22/01/23 13:21:24 WARN DAGScheduler: Broadcasting large task binary with size 1930.3 KiB
22/01/23 13:23:22 WARN DAGScheduler: Broadcasting large task binary with size 1009.4 KiB
22/01/23 13:23:27 WARN DAGScheduler: Broadcasting large task binary with size 1421.1 KiB
22/01/23 13:23:27 WARN DAGScheduler: Broadcasting large task binary with size 1008.9 KiB
22/01/23 13:23:28 WARN DAGScheduler: Broadcasting large task binary with size 1357.7 KiB
22/01/23 13:23:33 WARN DAGScheduler: Broadcasting large task binary with size 1959.3 KiB
22/01/23 13:23:34 WARN DAGScheduler: Broadcasting large task binary with size 1424.9 KiB
22/01/23 13:23:34 WARN DAGScheduler: Broadcasting large task binary with size 1360.3 KiB
22/01/23 13:23:39 WARN DAGScheduler: Broadcasting large task binary with size 1975.0 KiB
22/01/23 13:23:54 WARN DAGScheduler: Broadcasting large task binary with size 1353.9 KiB
22/01/23 13:23:56 WARN DAGScheduler: Broadcasting large task binary with size 1000.9 KiB
```

```
22/01/23 13:23:57 WARN DAGScheduler: Broadcasting large task binary with size 1414.5 KiB
22/01/23 13:23:59 WARN DAGScheduler: Broadcasting large task binary with size 1966.6 KiB
22/01/23 13:25:59 WARN DAGScheduler: Broadcasting large task binary with size 1377.6 KiB
22/01/23 13:26:00 WARN DAGScheduler: Broadcasting large task binary with size 1337.2 KiB
22/01/23 13:26:04 WARN DAGScheduler: Broadcasting large task binary with size 1331.2 KiB
22/01/23 13:26:04 WARN DAGScheduler: Broadcasting large task binary with size 1915.7 KiB
22/01/23 13:26:07 WARN DAGScheduler: Broadcasting large task binary with size 1383.7 KiB
22/01/23 13:26:11 WARN DAGScheduler: Broadcasting large task binary with size 1929.9 KiB
22/01/23 13:26:23 WARN DAGScheduler: Broadcasting large task binary with size 1330.3 KiB
22/01/23 13:26:26 WARN DAGScheduler: Broadcasting large task binary with size 1384.2 KiB
22/01/23 13:26:27 WARN DAGScheduler: Broadcasting large task binary with size 1927.3 KiB
22/01/23 13:26:39 WARN DAGScheduler: Broadcasting large task binary with size 1351.1 KiB

[0.8628619413057041, 0.89753854992108, 0.9022403251163469, 0.8563069096797278, 0.8993684995693207, 0.902437104674513
2, 0.8544414848940226, 0.8988244675661379, 0.9024741109063659, 0.8879509471277993, 0.9078254703199518, 0.908984681017
451, 0.8850041838918704, 0.9089408803244823, 0.9085507895106313, 0.8877360465240318, 0.9097787781777442, 0.9089177428
408262, 0.9054943860463518, 0.9157011895087039, 0.9178925983847981, 0.9077078561231842, 0.9168488634453899, 0.9181536
915834578, 0.90894499664817, 0.9184346075751182, 0.9178427811140766]

Result of random forest on the test set: 0.9177353383890541
```

Conclusion

We get fairly similar results using both Linear Regression or Random Forest. We could try other classifiers and perform GridSearch on more parameters, but this project intends to demonstrate the ability to use (Py)Spark, the data set itself is merely a pretext to justify its use.\ We believe that we demonstrated this ability since we performed the most important Machine Learning Classification tasks using PySpark:

- Exploratory Data Anlalysis
- Data Cleaning
- Data Preprocessing
- Classification
- Hyperparameter tuning
- Model Evaluation