Développement d’un système de détection de fraudes en utilisant la librairie « MLlib » d’Apache Spark

Présenté par

Bouchaara Oumayma
Idhika Oumaima

Encadré par

Mme Zaidouni Dounia

Introduction

Chaque année, la fraude coûte à l'économie mondiale plus de 5 000 milliards de dollars. En plus de faire payer un lourd tribut aux victimes individuelles, la fraude a un impact sur les entreprises sous la forme d'une perte de revenus et de productivité, ainsi que d'une réputation et de relations avec les clients endommagées. Les praticiens de l'IA sont en première ligne de cette bataille, construisant et déployant des modèles ML sophistiqués pour détecter les fraudes qui permettent aux organisations

1 **Présentation du projet**

2 **Analyse exploratoire et modélisation**

3 **Deploiement par Streamlit**

Presentation du projet

Problematique

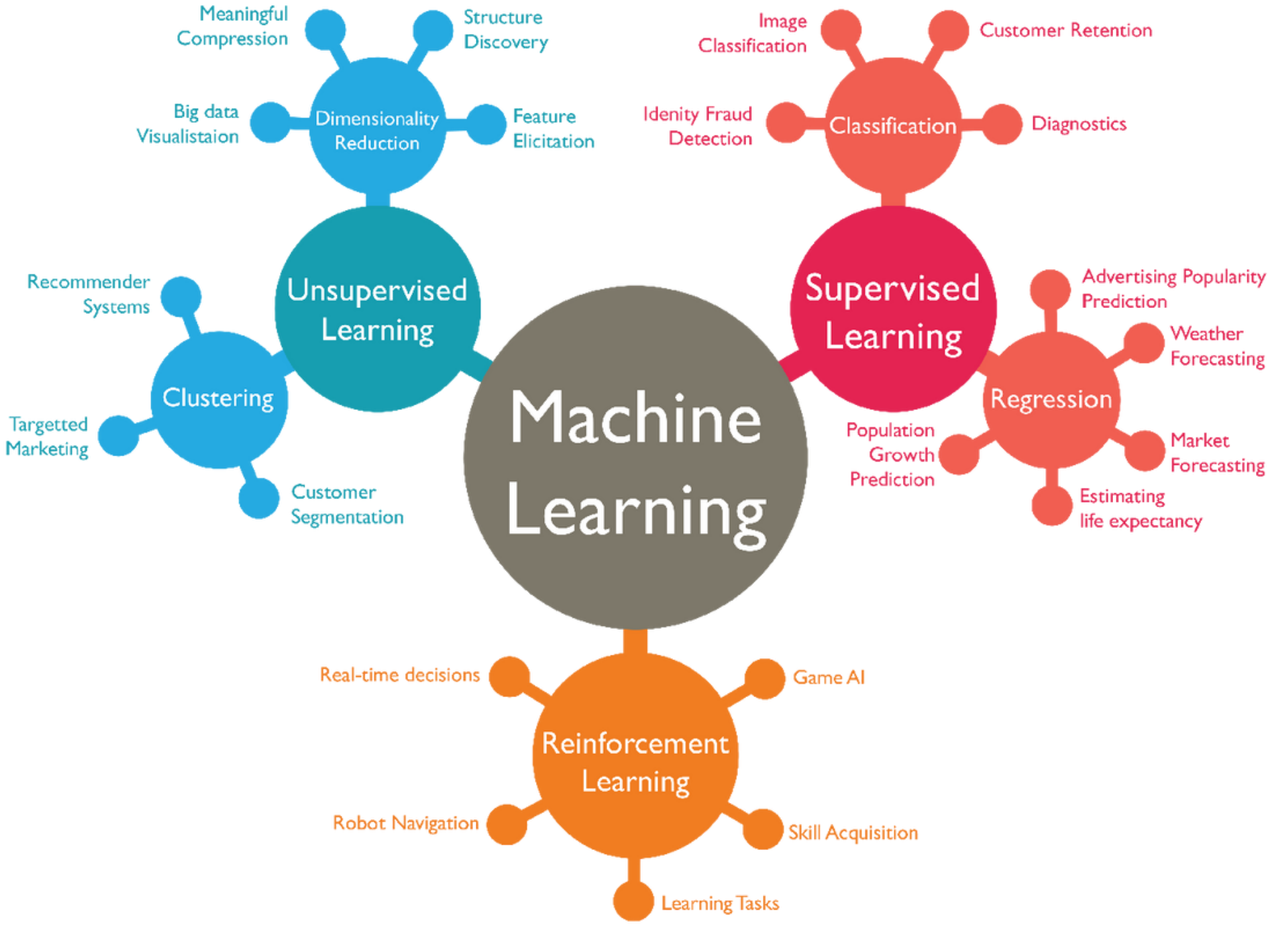
les transactions sont en pleine croissance

Les fraudeurs étant de plus en plus habiles à trouver et à exploiter les failles des systèmes

les fraudes occupent le devant des soucis des banques

Solution


BIG DATA


Machine Learning
Unsupervised Learning
Meaningful Compression
Structure Discovery
Dimensionality Reduction
Big data Visualistaion
Feature Elicitation
Clustering
Recommender Systems
Targetted Marketing
Customer Segmentation
Supervised Learning
Image Classification
Customer Retention
Classification
Idenity Fraud Detection
Diagnostics
Regression
Advertising Popularity Prediction
Weather Forecasting
Market Forecasting
Estimating life expectancy
Population Growth Prediction
Reinforcement Learning
Real-time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks

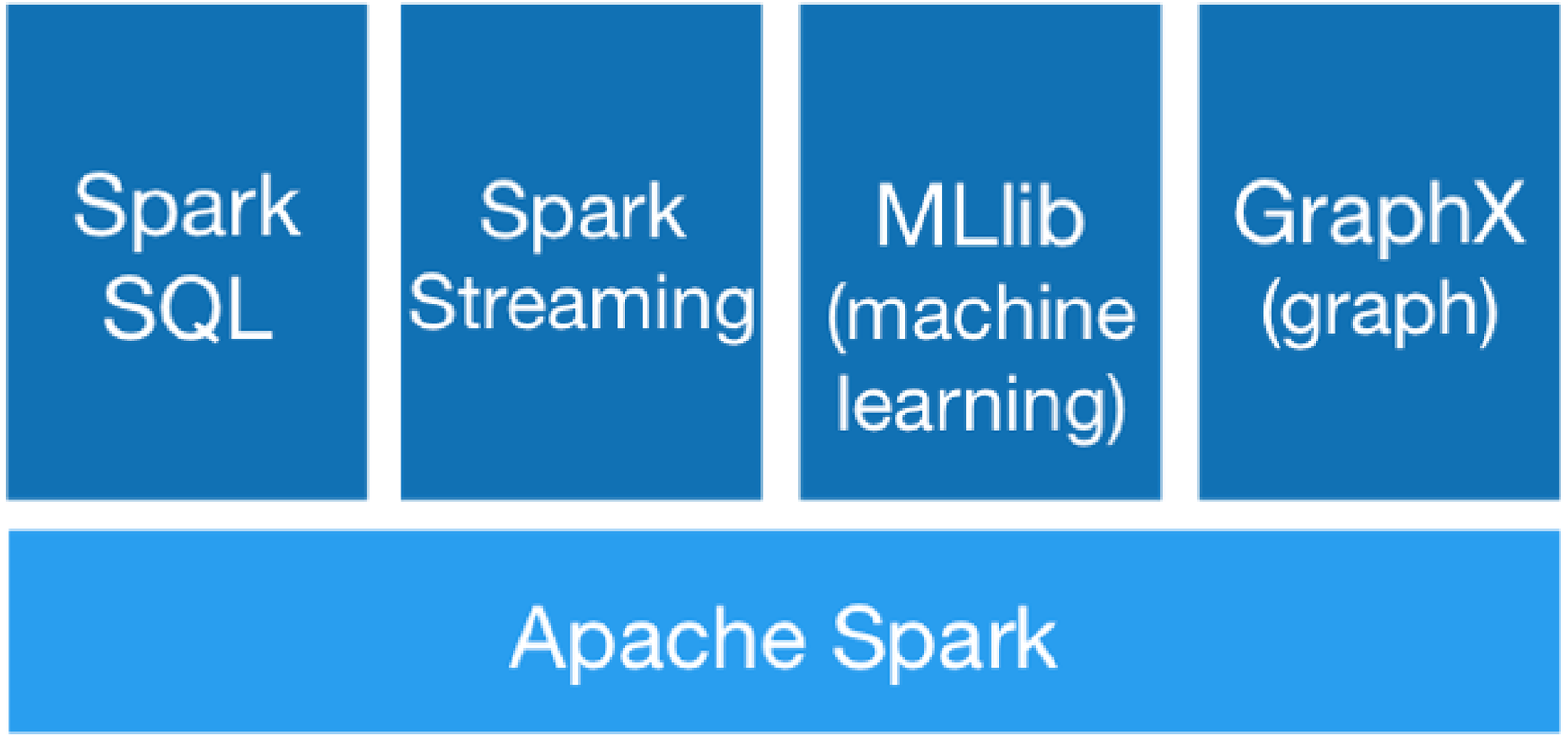
Description du jeu des donnees

le dataset sur lequel on a travaillé est récupérée depuis Kaggle:
https://www.kaggle.com/datasets/ealaxi/paysim1

l'ensemble des colonnes sont les suivantes:

- **Step**
- **type**
- **amount**
- **nameOrig**
- **oldbalanceOrig**
- **newbalanceOrig**
- **nameDest**
- **oldbalanceDest**
- **newbalanceDest**
- **isFraud**
- **isFlaggedFraud**

Spark et MLlib



Analyse exploratoire et modélisation

Analyse statistique et visualisation

```
from pyspark.sql import SparkSession , SQLContext
from pyspark.sql.functions import col,isnan, when, count
import seaborn as sns
import matplotlib.pyplot as plt
import pyspark.sql.functions as f
```

Create spark session

```
spark = SparkSession.builder.master('local[*]').appName('Fraud Sys detection').getOrCreate()
```

Importation des données et création d'une session spark

```
data.printSchema()
```

```
root
 |-- step: string (nullable = true)
 |-- type: string (nullable = true)
 |-- amount: string (nullable = true)
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrg: string (nullable = true)
 |-- newbalanceOrig: string (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: string (nullable = true)
 |-- newbalanceDest: string (nullable = true)
 |-- isFraud: string (nullable = true)
 |-- isFlaggedFraud: string (nullable = true)
```

Shema du jeu des données

```
cols_to_convert = ['step','amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest','isFraud','isFl
```

```
for col_name in cols_to_convert:
    data = data.withColumn(col_name, data[col_name].cast('float').alias(col_name))
```

```
data.printSchema()
```

```
root
 |-- step: float (nullable = true)
 |-- type: string (nullable = true)
 |-- amount: float (nullable = true)
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrg: float (nullable = true)
 |-- newbalanceOrig: float (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: float (nullable = true)
 |-- newbalanceDest: float (nullable = true)
 |-- isFraud: float (nullable = true)
 |-- isFlaggedFraud: float (nullable = true)
```

Conversion des types de donnees

```
print((data.count(),len(data.columns)))
```

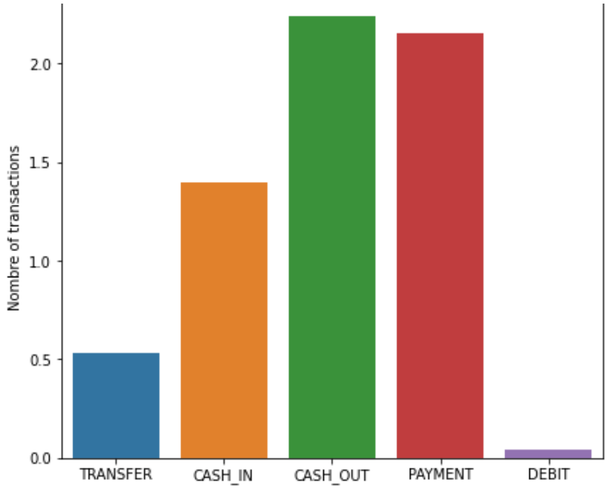
```
(6362620, 11)
```

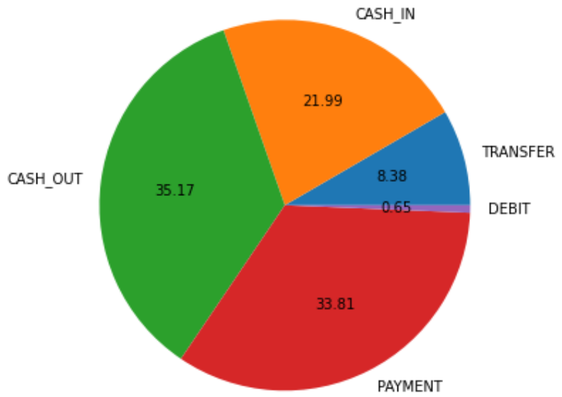
```
data.describe(['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']).show()
```

```
+-------+----------------+-----------------+-----------------+-----------------+-----------------+
|summary|          amount|    oldbalanceOrg|   newbalanceOrig|   oldbalanceDest|   newbalanceDest|
+-------+----------------+-----------------+-----------------+-----------------+-----------------+
|  count|         6362620|          6362620|          6362620|          6362620|          6362620|
|   mean|179861.9035579989| 833883.1040482761|855113.6685519279|1100701.6665156623|1224996.398184315|
| stddev|603858.2316772637|2888242.6729989178|2924048.502910317|3399180.1117577194|3674128.941013759|
|    min|             0.0|              0.0|              0.0|              0.0|              0.0|
|    max|      9.244552E7|       5.958504E7|       4.958504E7|      3.56015904E8|     3.56179264E8|
+-------+----------------+-----------------+-----------------+-----------------+-----------------+
```

les mesures statiques de l'ensemble des donnees

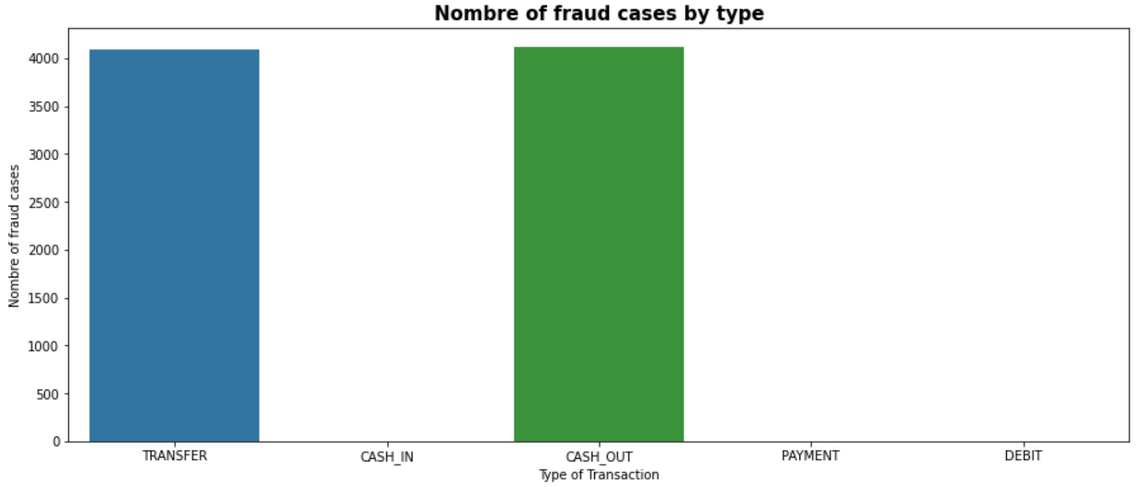
	type	count
0	TRANSFER	532909
1	CASH_IN	1399284
2	CASH_OUT	2237500
3	PAYMENT	2151495
4	DEBIT	41432



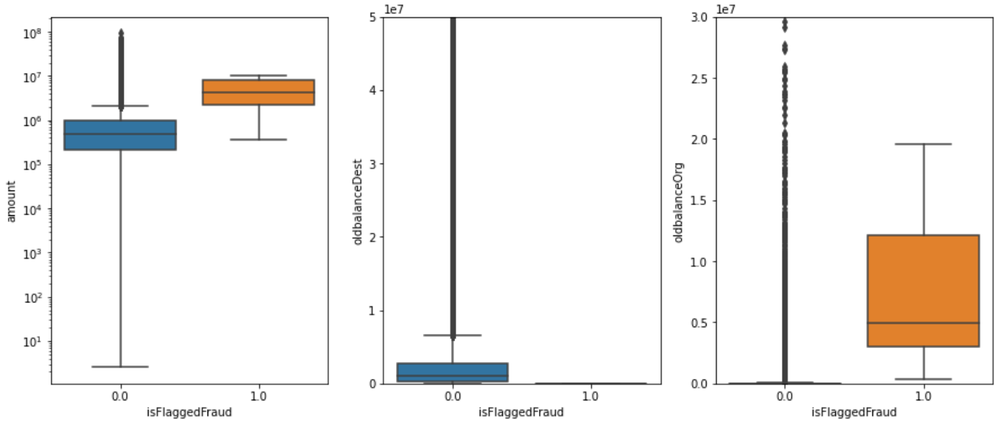


Nombre des transactions par type

	type	sum(isFraud)
0	TRANSFER	4097.0
1	CASH_IN	0.0
2	CASH_OUT	4116.0
3	PAYMENT	0.0
4	DEBIT	0.0



la somme des transactions qui sont réellement frauduleuses par type de transaction



comparaison entre les transactions de type TRANSFER détectées par le système traditionnel et celles non détectées

Count null values

```
data.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in data.columns]).show()
```

```
+----+----+------+--------+-------------+--------------+--------+-------------+--------------+-------+-------------
-+
|step|type|amount|nameOrig|oldbalanceOrg|newbalanceOrig|nameDest|oldbalanceDest|newbalanceDest|isFraud|isFlaggedFrau
d|
+----+----+------+--------+-------------+--------------+--------+-------------+--------------+-------+-------------
-+
|   0|   0|     0|       0|            0|             0|       0|            0|             0|      0|
0|
+----+----+------+--------+-------------+--------------+--------+-------------+--------------+-------+-------------
-+
```

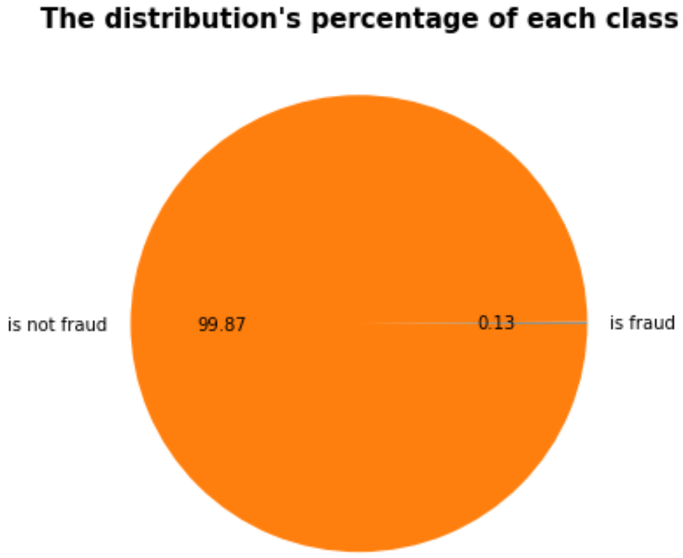
Count duplicate records

```
data.groupBy(data.columns).count().where(f.col('count') > 1).select(f.sum('count')).show()
```

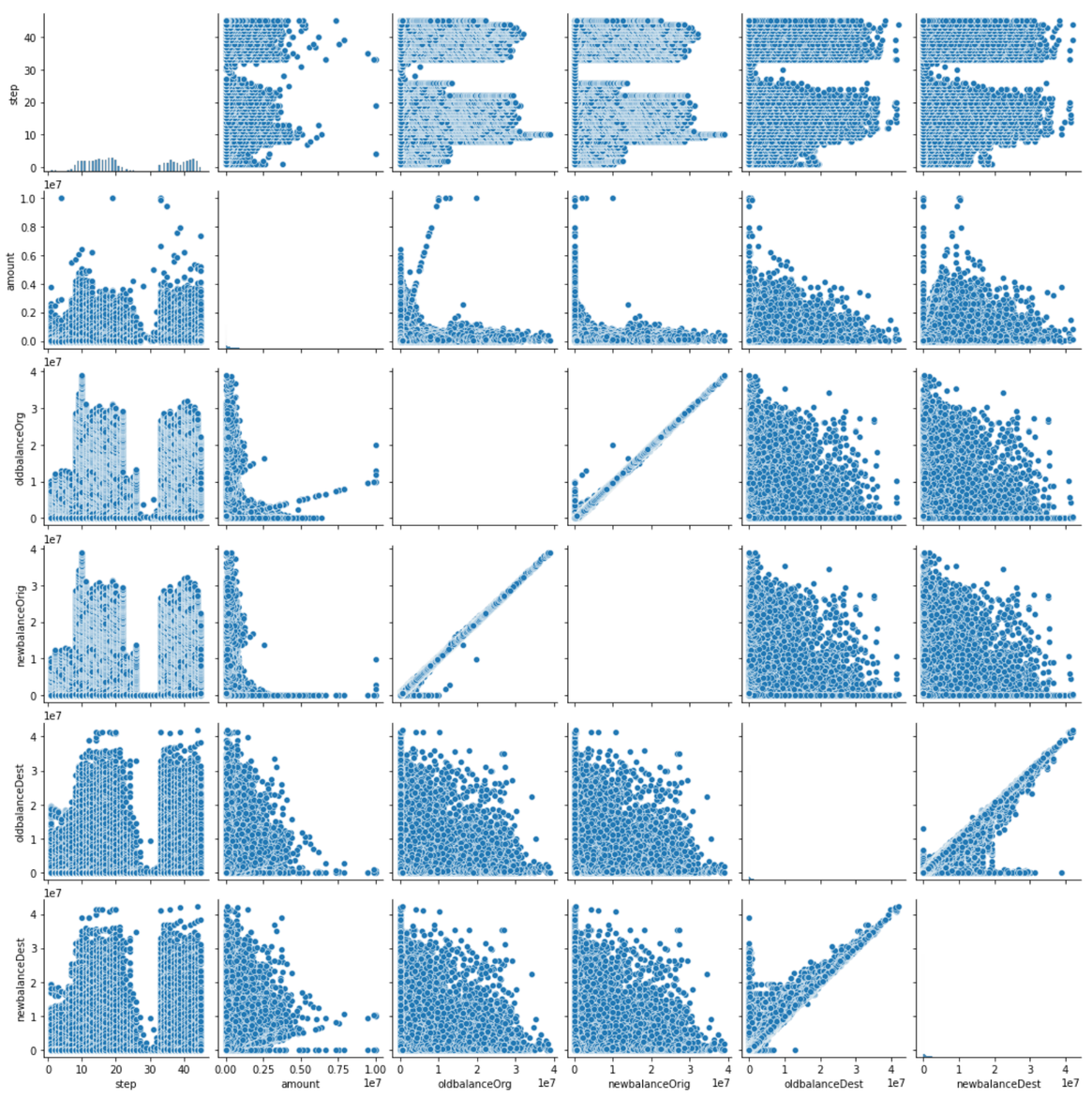
```
+----------+
|sum(count)|
+----------+
|      null|
+----------+
```

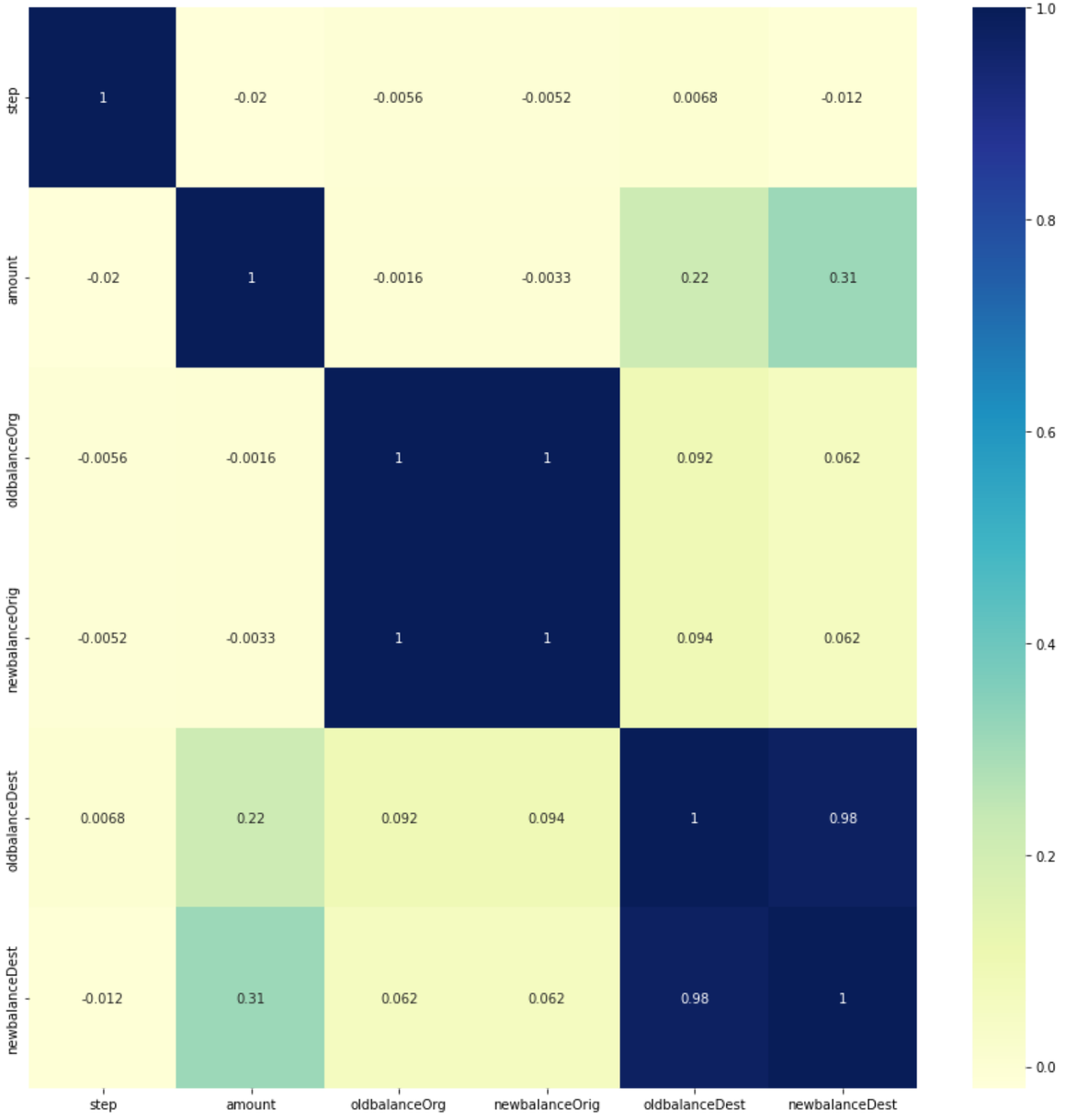
s'assurer qu'on n'a ni valeur manquante ni ligne dupliquée

	isFraud	count
0	1.0	8213
1	0.0	6354407



le pourcentage des transactions(frauduleuses/non frauduleuses





la correlation entre les variables

Prétraitement des données

```
cols = ("step","isFlaggedFraud","nameOrig","nameDest")

data = data.drop(*cols)
```

```
data.select('type').distinct().collect()
```

```
[Row(type='TRANSFER'),
 Row(type='CASH_IN'),
 Row(type='CASH_OUT'),
 Row(type='PAYMENT'),
 Row(type='DEBIT')]
```

```
data = data.filter((data.type=="TRANSFER") | (data.type=="CASH_OUT"))
```

éliminer les colonnes qui ne sont pas importantes et puis on supprime tous les transactions qui ne sont pas de type TRANSFER et CASH_OUT

```
from pyspark.ml.feature import StringIndexer

type_indexer = StringIndexer(inputCol="type", outputCol="numeric_type")
data = type_indexer.fit(data).transform(data)
data = data.drop('type')
data.show(5)
```

```
+---------+-------------+--------------+-------------+--------------+-------+------------+
|   amount|oldbalanceOrg|newbalanceOrig|oldbalanceDest|newbalanceDest|isFraud|numeric_type|
+---------+-------------+--------------+-------------+--------------+-------+------------+
|    181.0|        181.0|           0.0|          0.0|           0.0|    1.0|         1.0|
|    181.0|        181.0|           0.0|      21182.0|           0.0|    1.0|         0.0|
|229133.94|      15325.0|           0.0|       5083.0|       51513.44|    0.0|         0.0|
| 215310.3|        705.0|           0.0|      22425.0|           0.0|    0.0|         1.0|
|311685.88|      10835.0|           0.0|       6267.0|     2719173.0|    0.0|         1.0|
+---------+-------------+--------------+-------------+--------------+-------+------------+
only showing top 5 rows
```

vectoriser la colonne de type de transactions pour la convertir en colonne numérique

```
+---------+-------------+--------------+-------------+--------------+-------+------------+--------------------+--------------------+
|   amount|oldbalanceOrg|newbalanceOrig|oldbalanceDest|newbalanceDest|isFraud|numeric_type|            features|      scaledFeatures|
+---------+-------------+--------------+-------------+--------------+-------+------------+--------------------+--------------------+
|    181.0|        181.0|           0.0|          0.0|           0.0|    1.0|         1.0|[181.0,181.0,0.0,...|[1.95790991277889...|
|    181.0|        181.0|           0.0|      21182.0|           0.0|    1.0|         0.0|[181.0,181.0,0.0,...|[1.95790991277889...|
|229133.94|      15325.0|           0.0|       5083.0|       51513.44|    0.0|         0.0|[229133.9375,1532...|[0.02478583467320...|
| 215310.3|        705.0|           0.0|      22425.0|           0.0|    0.0|         1.0|[215310.296875,70...|[0.02329050633010...|
+---------+-------------+--------------+-------------+--------------+-------+------------+--------------------+--------------------+
only showing top 4 rows
```

rassembler les colonnes "Features" ou "inputs" , et après on les normalise grâce à MinMaxScaler

```
(train, test )= df.randomSplit([0.9, 0.1], seed=2022)
```

```
train.show(5)
```

```
+------+-------------+--------------+-------------+--------------+-------+------------+--------------------+
|amount|oldbalanceOrg|newbalanceOrig|oldbalanceDest|newbalanceDest|isFraud|numeric_type|            features|
+------+-------------+--------------+-------------+--------------+-------+------------+--------------------+
|  0.37|      25032.0|       25031.63|          0.0|           0.0|    0.0|         0.0|[0.37000000476837...|
|  1.58|          0.0|           0.0|     197938.48|       70090.2|    0.0|         0.0|[1.58000004291534...|
|  1.65|          0.0|           0.0|     6653022.5|     6653024.5|    0.0|         0.0|[1.64999997615814...|
|   1.9|      18399.0|       18397.1|   1.1306069E7|   1.1306071E7|    0.0|         0.0|[1.89999997615814...|
|  2.05|          0.0|           0.0|     243703.05|      243705.1|    0.0|         0.0|[2.04999995231628...|
+------+-------------+--------------+-------------+--------------+-------+------------+--------------------+
only showing top 5 rows
```

```
train.groupBy('isFraud').count().show(truncate = False)
```

```
+-------+-------+
|isFraud|count  |
+-------+-------+
|1.0    |7403   |
|0.0    |2486403|
+-------+-------+
```

diviser les données entre 90% pour "train data" et 10% pour "test data "

Prédiction et évaluation des différents algorithmes

Logistic Regression

```
tp = prediction_LR[(prediction_LR.isFraud == 1) & (prediction_LR.prediction == 1)].count()
tn = prediction_LR[(prediction_LR.isFraud == 0) & (prediction_LR.prediction == 0)].count()
fp = prediction_LR[(prediction_LR.isFraud == 0) & (prediction_LR.prediction == 1)].count()
fn = prediction_LR[(prediction_LR.isFraud == 1) & (prediction_LR.prediction == 0)].count()
recall_LR = tp/(tp+fn)
precision_LR = tp/(tp+fp)
f1_score_LR = 2*(recall_LR*precision_LR)/(recall_LR+precision_LR)
print("Recall : ",recall_LR)
print("Precision : ", precision_LR)
print("F1 Score : ", f1_score_LR)
```

```
Recall :  0.4691358024691358
Precision :  0.9047619047619048
F1 Score :  0.6178861788617886
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol="isFraud")
areaUnderROC_LR = evaluator.evaluate(prediction_LR, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_LR)
```

```
Area under ROC = 0.9831340807452497
```

```
areaUnderPR_LR = evaluator.evaluate(prediction_LR, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_LR)
```

```
Area under PR = 0.6665939046903049
```

Naive bayes

```
tp = prediction_NB[(prediction_NB.isFraud == 1) & (prediction_NB.prediction == 1)].count()
tn = prediction_NB[(prediction_NB.isFraud == 0) & (prediction_NB.prediction == 0)].count()
fp = prediction_NB[(prediction_NB.isFraud == 0) & (prediction_NB.prediction == 1)].count()
fn = prediction_NB[(prediction_NB.isFraud == 1) & (prediction_NB.prediction == 0)].count()
recall_NB = tp/(tp+fn)
precision_NB = tp/(tp+fp)
f1_score_NB = 2*(recall_NB*precision_NB)/(recall_NB+precision_NB)
print("Recall : ",recall_NB)
print("Precision : ", precision_NB)
print("F1 Score : ", f1_score_NB)
```

```
Recall :  0.8641975308641975
Precision :  0.01032189568987127
F1 Score :  0.02040013405802381
```

```
areaUnderROC_NB = evaluator.evaluate(prediction_NB, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_NB)
```

```
Area under ROC = 0.5626975644150937
```

```
areaUnderPR_NB = evaluator.evaluate(prediction_NB, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_NB)
```

```
Area under PR = 0.008746362879704742
```

Random forest

```
tp = prediction_RF[(prediction_RF.isFraud == 1) & (prediction_RF.prediction == 1)].count()
tn = prediction_RF[(prediction_RF.isFraud == 0) & (prediction_RF.prediction == 0)].count()
fp = prediction_RF[(prediction_RF.isFraud == 0) & (prediction_RF.prediction == 1)].count()
fn = prediction_RF[(prediction_RF.isFraud == 1) & (prediction_RF.prediction == 0)].count()
recall_RF = tp/(tp+fn)
precision_RF = tp/(tp+fp)
f1_score_RF = 2*(recall_RF*precision_RF)/(recall_RF+precision_RF)
print("Recall : ",recall_RF)
print("Precision : ", precision_RF)
print("F1 Score : ", f1_score_RF)
```

```
Recall :  0.5679012345679012
Precision :  0.9956709956709957
F1 Score :  0.7232704402515723
```

```
areaUnderROC_LR = evaluator.evaluate(prediction_RF, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_LR)
```

```
Area under ROC = 0.921505064654637
```

```
# Area under precision-recall curve
areaUnderPR_RF = evaluator.evaluate(prediction_RF, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_RF)
```

```
Area under PR = 0.7501002010064396
```

Gradient boosted tree

```
tp = prediction_GBT[(prediction_GBT.isFraud == 1) & (prediction_GBT.prediction == 1)].count()
tn = prediction_GBT[(prediction_GBT.isFraud == 0) & (prediction_GBT.prediction == 0)].count()
fp = prediction_GBT[(prediction_GBT.isFraud == 0) & (prediction_GBT.prediction == 1)].count()
fn = prediction_GBT[(prediction_GBT.isFraud == 1) & (prediction_GBT.prediction == 0)].count()
recall_GBT = tp/(tp+fn)
precision_GBT = tp/(tp+fp)
f1_score_GBT = 2*(recall_GBT*precision_GBT)/(recall_GBT+precision_GBT)
print("Recall : ",recall_GBT)
print("Precision : ", precision_GBT)
print("F1 Score : ", f1_score_GBT)
```

```
Recall :  0.6691358024691358
Precision :  0.9713261648745519
F1 Score :  0.7923976608187135
```

```
#Area under ROC curve
# Area under ROC curve
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol="isFraud")
areaUnderROC_GBT = evaluator.evaluate(prediction_GBT, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_GBT)
```

```
Area under ROC = 0.9895590708060567
```

```
# Area under precision-recall curve
areaUnderPR_GBT = evaluator.evaluate(prediction_GBT, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_GBT)
```

```
Area under PR = 0.8230876904767537
```

Récapitulatif

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100

Precision

Choix du modèle

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746

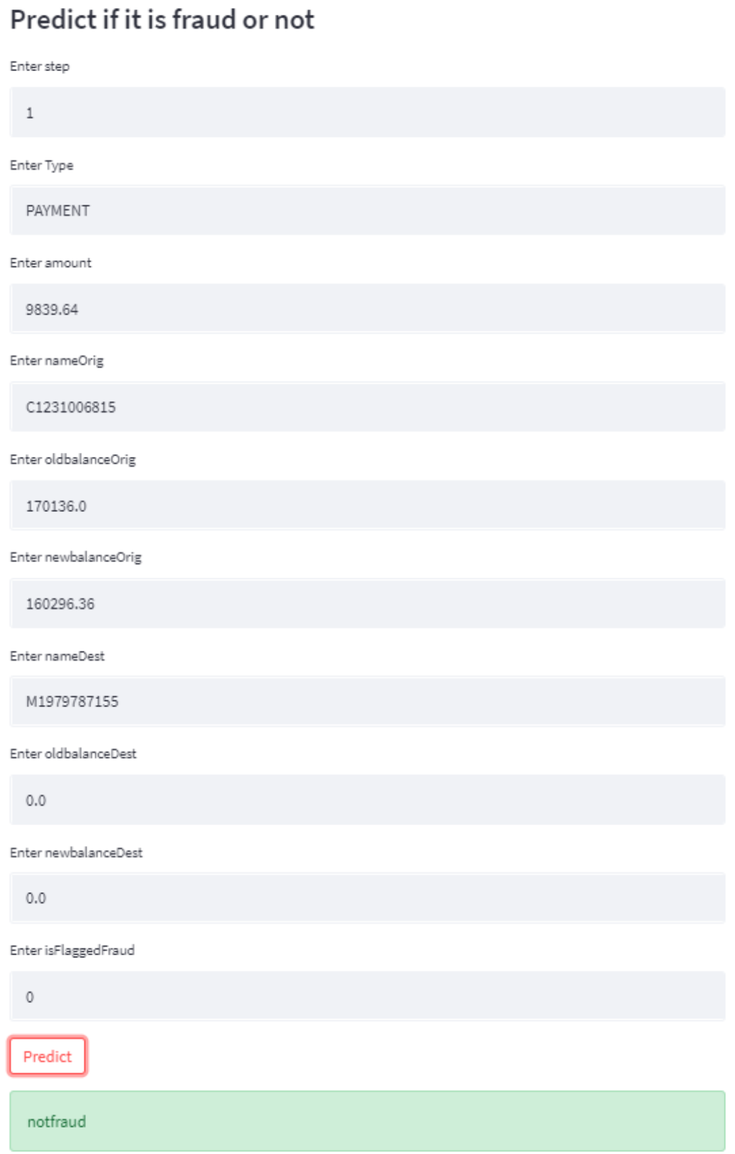
Random Forest

Déploiement par Streamlit

on finit par déployer le modèle de machine Learning déjà implémenté a l'aide du Streamlit

```python
import streamlit as st
import ktrain
import pickle
import joblib
st.title('Fraud detection')
st.subheader('Predict if it is fraud or not')
step=st.text_input('Enter step')
Type=st.text_input('Enter Type')
amount=st.text_input('Enter amount')
nameOrig=st.text_input('Enter nameOrig')
oldbalanceOrig=st.text_input('Enter oldbalanceOrig')
newbalanceOrig=st.text_input('Enter newbalanceOrig')
nameDest=st.text_input('Enter nameDest')
oldbalanceDest=st.text_input('Enter oldbalanceDest')
newbalanceDest=st.text_input('Enter newbalanceDest')
isFlaggedFraud=st.text_input('Enter isFlaggedFraud')
if st.button("Predict"):
    model=joblib.load('modelgbt1.pkl')
    data = list(map([step,Type,amount, nameOrig,oldbalanceOrig,newbalan
    result, probs = model.predict(data)
    if result==0:
            st.success('notfraud')
    elif result==1:
            st.error('fraud')

```



Merci de votre attention