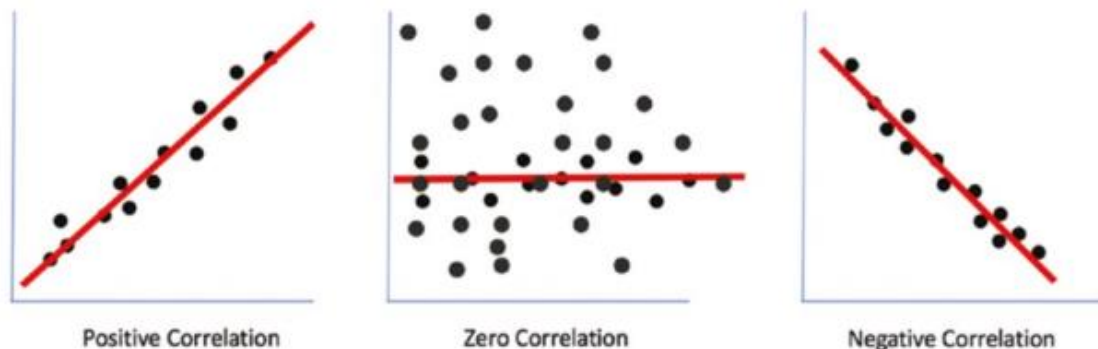


Chapitre 5 :

La bibliotheque MLIB

A. Notes de cours

- ❖ la bibliothèque d'apprentissage automatique de Spark (Mllib) a une capacité à former des modèles à grande échelle et à fournir une formation distribuée.
- ❖ La bibliothèque d'apprentissage automatique de Spark (Mllib) permet aux utilisateurs de créer rapidement des modèles sur un énorme ensemble de données, en plus de prétraiter et de préparer des flux de travail avec le framework Spark lui même.
- ❖ La corrélation est une mesure importante permettant de déterminer s'il existe une relation entre deux variables continues. Elle peut être positive ou négative ou tout simplement qu'il n'y ait pas de corrélation entre deux variables.



- ❖ La corrélation concerne la relation entre les caractéristiques numériques, alors que d'autres types de variables peuvent également être catégorielles. L'un des moyens de valider la relation entre deux variables catégorielles consiste à utiliser un test du chi carré.
- ❖ Nous pouvons convertir la variable numérique/continue en caractéristiques catégorielles (0/1) en utilisant Binarizer dans Mllib.
- ❖ Nous devons déclarer la valeur seuil, afin de convertir la caractéristique numérique en une caractéristique binaire.

- ❖ L'analyse en composantes principales (ACP) est l'une des techniques de transformation qui permet de réduire les dimensions des données tout en gardant intacte au maximum la variation des données.
- ❖ La normalisation fait référence à la transformation des données de manière à ce que les nouvelles données normalisées aient une moyenne de 0 et un écart type de 1. La normalisation se fait à l'aide de la formule suivante :

$$\frac{(x - \text{mean}(x))}{\text{standard dev}(x)}$$

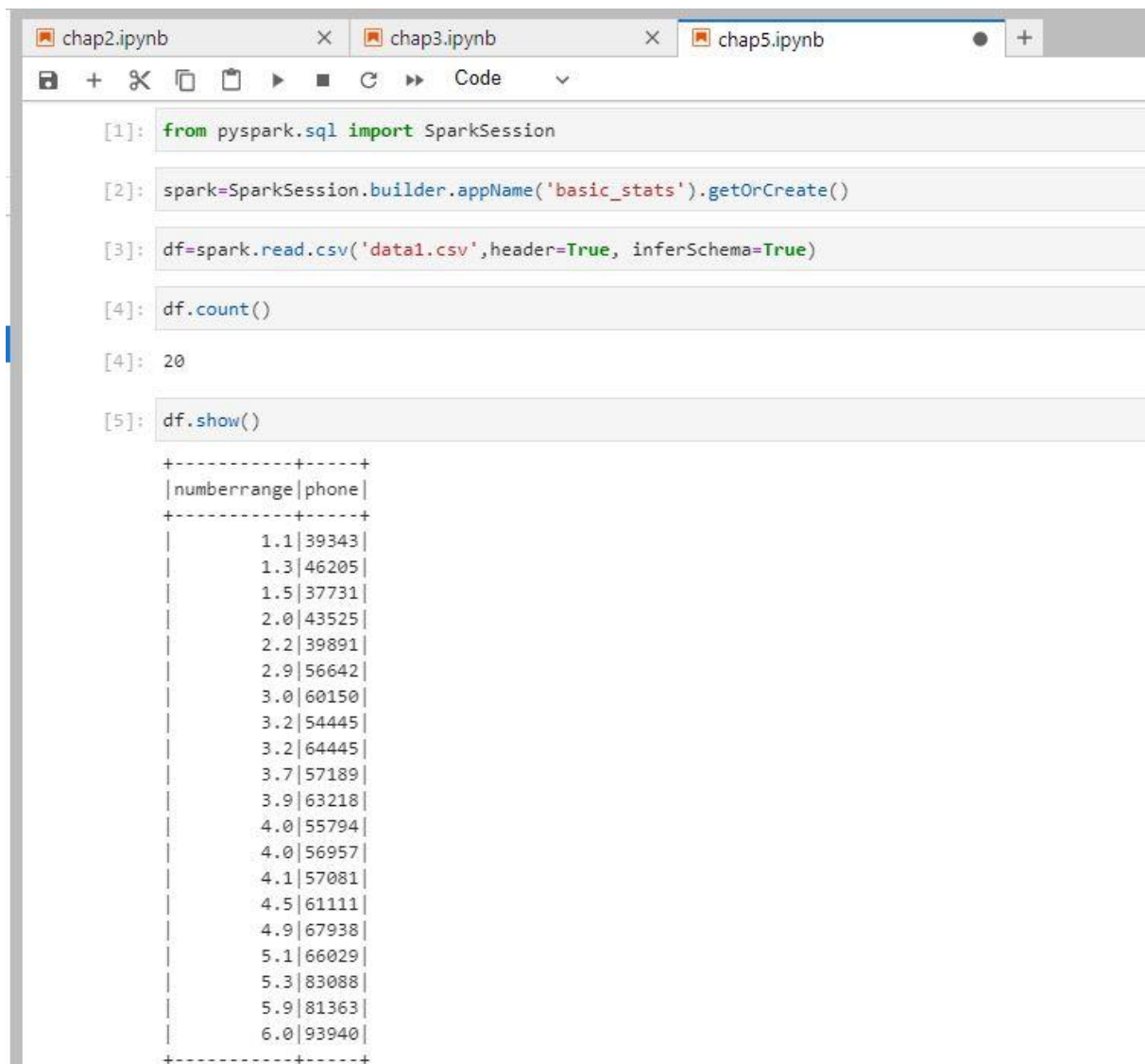
- ❖ La normalisation permet de standardiser les données d'entrée et parfois d'améliorer les performances des modèles d'apprentissage automatique.
- ❖ La mise à l'échelle est une autre technique pour normaliser les données, de sorte que les valeurs se situent dans une plage spécifique. Sa formule est :

$$\frac{x - \min(x)}{\max(x) - \min(x)}$$

- ❖ La mise à l'échelle min-max est une autre version de la mise à l'échelle standard, car elle vous permet de redimensionner les valeurs des caractéristiques entre des limites spécifiques (généralement entre 0 et 1).
- ❖ MaxAbsScaler est un peu différent des outils de mise à l'échelle standard, car il redimensionne chaque valeur de caractéristique entre -1 et 1. Cependant, il ne déplace pas le centre des données et, par conséquent, n'a aucun impact sur la parcimonie.
- ❖ On fait du binning à l'aide de Bucketizer dans Spark
- ❖ Utilise la bibliothèque d'apprentissage automatique de Spark (MLlib) pour créer des modèles de classification.

B. Cas Pratique

- Calcul des corrélations : création de l'objet SparkSession.



```
[1]: from pyspark.sql import SparkSession

[2]: spark=SparkSession.builder.appName('basic_stats').getOrCreate()

[3]: df=spark.read.csv('data1.csv',header=True, inferSchema=True)

[4]: df.count()

[4]: 20

[5]: df.show()
```

numberrange	phone
1.1	39343
1.3	46205
1.5	37731
2.0	43525
2.2	39891
2.9	56642
3.0	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4.0	55794
4.0	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6.0	93940

- Nous combinons ensuite les deux colonnes en une seule representation vectorielle, afin de calculer le coefficient de correlation.

```
[6]: from pyspark.ml.feature import VectorAssembler
```

```
[7]: assembler = VectorAssembler(inputCols=df.columns,outputCol="features")
```

```
[8]: df_new=assembler.transform(df)
```

```
[9]: df_new.show()
```

```
+-----+-----+-----+
|numberrange|phone|features|
+-----+-----+-----+
|1.1|39343|[1.1,39343.0]|
|1.3|46205|[1.3,46205.0]|
|1.5|37731|[1.5,37731.0]|
|2.0|43525|[2.0,43525.0]|
|2.2|39891|[2.2,39891.0]|
|2.9|56642|[2.9,56642.0]|
|3.0|60150|[3.0,60150.0]|
|3.2|54445|[3.2,54445.0]|
|3.2|64445|[3.2,64445.0]|
|3.7|57189|[3.7,57189.0]|
|3.9|63218|[3.9,63218.0]|
|4.0|55794|[4.0,55794.0]|
|4.0|56957|[4.0,56957.0]|
|4.1|57081|[4.1,57081.0]|
|4.5|61111|[4.5,61111.0]|
|4.9|67938|[4.9,67938.0]|
|5.1|66029|[5.1,66029.0]|
|5.3|83088|[5.3,83088.0]|
|5.9|81363|[5.9,81363.0]|
|6.0|93940|[6.0,93940.0]|
+-----+-----+-----+
```

➤ Calcul du coefficient de correlation de **Pearson**

```
[10]: from pyspark.ml.stat import Correlation
```

```
[11]: pearson_corr = Correlation.corr(df_new,'features')
```

```
[12]: pearson_corr.show(2,False)
```

```
+-----+-----+-----+
|pearson(features)|
+-----+-----+-----+
|1.0|0.9075437935323684|0.9075437935323684|1.0|
+-----+-----+-----+
```

- Calcul du coefficient de corrélation de **Spearman**

```
[13]: spearman_corr=Correlation.corr(df_new,'features',"spearman")
```

```
[14]: spearman_corr.show(2, False)
```

```
|spearman(features)
+-----+
|1.0          0.8705796294446093  \n0.8705796294446093  1.0
+-----+
```

- Execution du test de **Chi-Square** a l'aide de spark

```
[15]: df=spark.read.csv('data2.csv',inferSchema=True,header=True)
```

```
[16]: df.count()
```

```
[16]: 20
```

```
[17]: df.show()
```

marital	housing	label
married	no	0
married	no	0
married	yes	0
married	no	0
married	no	0
married	no	0
married	no	0
married	no	0
single	yes	0
single	yes	0
married	no	0
single	yes	0
single	no	0
divorced	yes	0
married	yes	0
married	yes	0
married	yes	0
married	yes	0
married	yes	0
single	no	0

```
[18]: from pyspark.ml.feature import StringIndexer

[19]: marital_indexer = StringIndexer(inputCol="marital",outputCol="marital_num").fit(df)

[20]: df = marital_indexer.transform(df)

[21]: from pyspark.ml.feature import OneHotEncoder

[22]: marital_encoder = OneHotEncoder(inputCol="marital_num",outputCol="marital_vector")

[24]: marital_encoder.setDropLast(False)

[24]: OneHotEncoder_114f0cbf4cfb

[25]: ohe = marital_encoder.fit(df)

[26]: df = ohe.transform(df)

[27]: housing_indexer = StringIndexer(inputCol="housing",outputCol="housing_num").fit(df)

[28]: df = housing_indexer.transform(df)

[29]: housing_encoder = OneHotEncoder(inputCol="housing_num",outputCol="housing_vector")

[30]: housing_encoder.setDropLast(False)

[30]: OneHotEncoder_30178ef03adf

[31]: ohe1 = housing_encoder.fit(df)

[32]: df = ohe1.transform(df)
```

```
[33]: df.show()
```

marital	housing	label	marital_num	marital_vector	housing_num	housing_vector
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
single	yes	0	1.0	(3,[1],[1.0])	1.0	(2,[1],[1.0])
single	yes	0	1.0	(3,[1],[1.0])	1.0	(2,[1],[1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])
single	yes	0	1.0	(3,[1],[1.0])	1.0	(2,[1],[1.0])
single	no	0	1.0	(3,[1],[1.0])	0.0	(2,[0],[1.0])
divorced	yes	0	2.0	(3,[2],[1.0])	1.0	(2,[1],[1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])
single	no	0	1.0	(3,[1],[1.0])	0.0	(2,[0],[1.0])

```
[34]: df_assembler = VectorAssembler(inputCols=['marital_vector','housing_vector'], outputCol="features")
[35]: df = df_assembler.transform(df)
[36]: df.show()
```

marital	housing	label	marital_num	marital_vector	housing_num	housing_vector	features
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])	(5,[0,4],[1.0,1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
single	yes	0	1.0	(3,[1],[1.0])	1.0	(2,[1],[1.0])	(5,[1,4],[1.0,1.0])
single	yes	0	1.0	(3,[1],[1.0])	1.0	(2,[1],[1.0])	(5,[1,4],[1.0,1.0])
married	no	0	0.0	(3,[0],[1.0])	0.0	(2,[0],[1.0])	(5,[0,3],[1.0,1.0])
single	yes	0	1.0	(3,[1],[1.0])	1.0	(2,[1],[1.0])	(5,[1,4],[1.0,1.0])
single	no	0	1.0	(3,[1],[1.0])	0.0	(2,[0],[1.0])	(5,[1,3],[1.0,1.0])
divorced	yes	0	2.0	(3,[2],[1.0])	1.0	(2,[1],[1.0])	(5,[2,4],[1.0,1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])	(5,[0,4],[1.0,1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])	(5,[0,4],[1.0,1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])	(5,[0,4],[1.0,1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])	(5,[0,4],[1.0,1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])	(5,[0,4],[1.0,1.0])
married	yes	0	0.0	(3,[0],[1.0])	1.0	(2,[1],[1.0])	(5,[0,4],[1.0,1.0])
single	no	0	1.0	(3,[1],[1.0])	0.0	(2,[0],[1.0])	(5,[1,3],[1.0,1.0])

➤ Construction d'un modele de classification : pour notre exemple, nous utilisons un jeu de donnees contenant les informations relatives a quelques clients qui veulent effectuer des prets dans une banque. Nous construisons ainsi un modele de classification binaire pour predire si un client particulier doit se voir accorder un prêt, sur la base des connaissances du modele. Les differentes etapes sont :

- Chargez le jeu de donnees
- Effectuer une analyse exploratoire des donnees
- Effectuer des transformations des donnees requises
- Divisez les donnees en sous-ensembles de formation et de test
- Entraîner et evaluer le modele de base sur les donnees
- Effectuer les reglages des hyperparametres
- Construire un modele final avec les meilleurs parametres.


```
[38]: from pyspark.sql import SparkSession

[39]: spark=SparkSession.builder.appName('binary_class').getOrCreate()

[40]: df=spark.read.csv('classification_data.csv',inferSchema=True,header=True)

[41]: print((df.count(),len(df.columns)))

(46751, 12)

[42]: df.printSchema()

root
 |-- loan_id: string (nullable = true)
 |-- loan_purpose: string (nullable = true)
 |-- is_first_loan: integer (nullable = true)
 |-- total_credit_card_limit: integer (nullable = true)
 |-- avg_percentage_credit_card_limit_used_last_year: double (nullable = true)
 |-- saving_amount: integer (nullable = true)
 |-- checking_amount: integer (nullable = true)
 |-- is_employed: integer (nullable = true)
 |-- yearly_salary: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- dependent_number: integer (nullable = true)
 |-- label: integer (nullable = true)
```

```
[43]: df.show(5)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|loan_id|loan_purpose|is_first_loan|total_credit_card_limit|avg_percentage_credit_card_limit_used_last_year|saving_amount|checking_amount|is_employed|yearly_salary|age|dependent_number|label|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  A_1|  personal|      1|      7900|      0.8|      1103|      6393|      1|      16|400|42|0| |
|  A_2|  personal|      0|      3300|      0.29|      2588|      832|      1|      75|500|56|0|
|  A_3|  personal|      0|      7600|      0.9|      1651|      8868|      1|      59|000|46|0|
|  A_4|  personal|      1|      3400|      0.38|      1269|      6863|      1|      26|000|55|8|0|
|  A_5| emergency|      0|      2600|      0.89|      1310|      3423|      1|      9|700|41|4|1|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

- Utilisation de groupby pour compter le nombre d'évenements positif et negatif. Ceci nous permettre de voir que plus d'un tiers de client n'ont pas rembourser leur prêts.


```
[46]: df.groupBy('label').count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|    1|16201|
|    0|30550|
+-----+-----+
```

- En continuant avec notre analyse, on constate que les clients demandent les prêts pour des raisons immobilières, opérationnelles et personnelles.

```
[47]: df.groupBy('loan_purpose').count().show()
```

```
+-----+-----+
|loan_purpose|count|
+-----+-----+
|    others| 6763|
| emergency| 7562|
|  property|11388|
| operations|10580|
|   personal|10458|
+-----+-----+
```

- Transformation des données : comme toutes les variables du dataframe sont numériques à l'exception de l'objet du prêt, nous allons la convertir sous forme numérique.

```
[48]: from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
```

```
[49]: loan_purpose_indexer = StringIndexer(inputCol="loan_purpose", outputCol="loan_index").fit(df)
```

```
[50]: df = loan_purpose_indexer.transform(df)
```

```
[51]: loan_encoder = OneHotEncoder(inputCol="loan_index", outputCol="loan_purpose_vec")
```

```
[53]: loan_encoder.setDropLast(False)
```

```
[53]: OneHotEncoder_b09f280f4b09
```

```
[54]: ohe2 = loan_encoder.fit(df)
```

```
[56]: df = ohe2.transform(df)
```

```
[57]: df.select(['loan_purpose', 'loan_index', 'loan_purpose_vec']).show(3, False)
```

```
+-----+-----+-----+
|loan_purpose|loan_index|loan_purpose_vec|
+-----+-----+-----+
|personal  |2.0      |(5,[2],[1.0])|
|personal  |2.0      |(5,[2],[1.0])|
|personal  |2.0      |(5,[2],[1.0])|
+-----+-----+-----+
```

only showing top 3 rows

- Nous allons maintenant créer un vecteur a caracteristique unique pour l'entrainement du modele.

```
[58]: from pyspark.ml.feature import VectorAssembler

[59]: t_card_limit_used_last_year', 'saving_amount', 'checking_amount', 'is_employed', 'yearly_salary', 'age', 'dependent_number', 'loan_purpose_vec'], outputCol="features")
←
```

```
[60]: df = df_assembler.transform(df)

[62]: df.select(['features', 'label']).show(10, False)
```

features	label
[1.0, 7900.0, 0.8, 1103.0, 6393.0, 1.0, 16400.0, 42.0, 4.0, 0.0, 0.0, 1.0, 0.0, 0.0]	0
[0.0, 3300.0, 0.29, 2588.0, 832.0, 1.0, 75500.0, 56.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0]	0
[0.0, 7600.0, 0.9, 1651.0, 8868.0, 1.0, 59000.0, 46.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0]	0
[1.0, 3400.0, 0.38, 1269.0, 6863.0, 1.0, 26000.0, 55.0, 8.0, 0.0, 0.0, 1.0, 0.0, 0.0]	0
[0.0, 2600.0, 0.89, 1310.0, 3423.0, 1.0, 9700.0, 41.0, 4.0, 0.0, 0.0, 0.0, 1.0, 0.0]	1
(14, [1, 2, 3, 4, 5, 6, 7, 10], [7600.0, 0.51, 1040.0, 2406.0, 1.0, 22900.0, 52.0, 1.0])	0
[1.0, 6900.0, 0.82, 2408.0, 5556.0, 1.0, 34800.0, 48.0, 4.0, 0.0, 1.0, 0.0, 0.0, 0.0]	0
[0.0, 5700.0, 0.56, 1933.0, 4139.0, 1.0, 32500.0, 64.0, 2.0, 0.0, 0.0, 1.0, 0.0, 0.0]	0
[1.0, 3400.0, 0.95, 3866.0, 4131.0, 1.0, 13300.0, 23.0, 3.0, 0.0, 0.0, 1.0, 0.0, 0.0]	0
[0.0, 2900.0, 0.91, 88.0, 2725.0, 1.0, 21100.0, 52.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0]	1

only showing top 10 rows

- Separation des donnees et formation du modele.

```
[63]: model_df=df.select(['features', 'label'])

[64]: training_df, test_df=model_df.randomSplit([0.75, 0.25])

[65]: from pyspark.ml.classification import LogisticRegression

[66]: log_reg=LogisticRegression().fit(training_df)

[67]: lr_summary=log_reg.summary

[68]: lr_summary.accuracy
```

```
[68]: 0.8931814946619218
```

```
[69]: lr_summary.areaUnderROC
```

```
[69]: 0.9585345162334288
```

```
[70]: print(lr_summary.precisionByLabel)
```

```
[0.9233007140967998, 0.8382506632365946]
```

```
[71]: print(lr_summary.recallByLabel)
```

```
[0.912361703981183, 0.8569902194460426]
```

```
[72]: predictions = log_reg.transform(test_df)
```

```
[73]: predictions.show(10)
```

features	label	rawPrediction	probability	prediction
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-4.3770418168056...	[0.01240660819335...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-6.3817512641631...	[0.00168929838842...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-6.3612442923392...	[0.00172423808034...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-7.3048659529449...	[6.7180799049482E...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-3.9692155255151...	[0.01853809264122...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-4.6102664916734...	[0.00985115579443...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-4.9723731633575...	[0.00687904135699...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-3.1288478953892...	[0.04193286786845...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	1	[-3.4085655370913...	[0.03202884029251...	1.0
(14, [0, 1, 2, 3, 4, 7, ...]	0	[4.45805859281647...	[0.98854783901862...	0.0

only showing top 10 rows

```
[74]: model_predictions = log_reg.transform(test_df)
```

```
[75]: model_predictions = log_reg.evaluate(test_df)
```

```
[76]: model_predictions.accuracy
```

```
[76]: 0.8955788749354894
```

```
[77]: model_predictions.areaUnderROC
```

```
[77]: 0.9601298930116144
```

```
[78]: print(model_predictions.recallByLabel)
```

```
[0.9179399367755532, 0.853495290034705]
```

```
[79]: print(model_predictions.precisionByLabel)
```

```
[0.9218253968253968, 0.8467781603541564]
```