Procesamiento de datos masivos

Jesús Morán

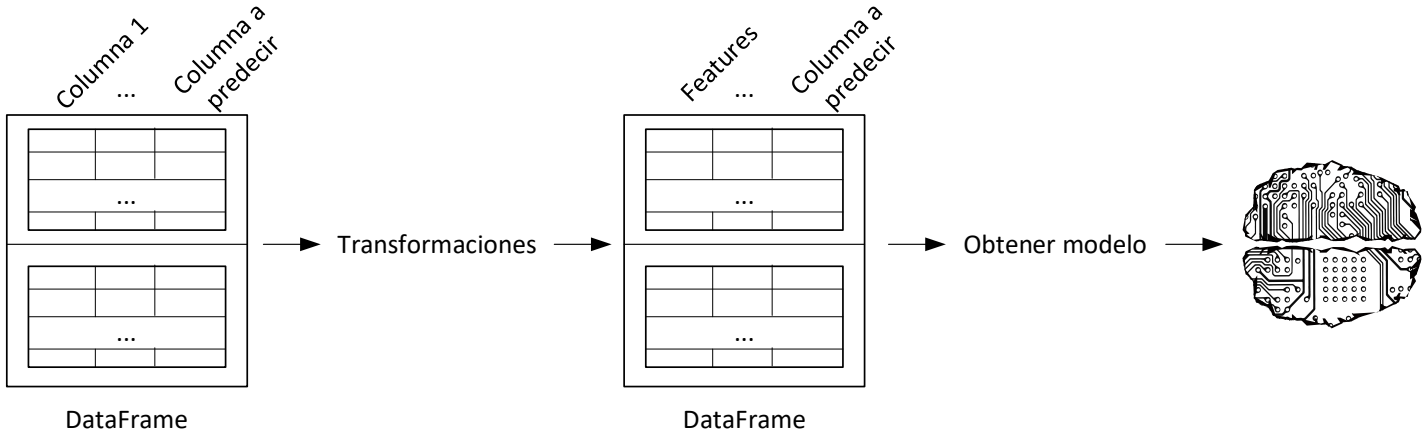
viu | Universidad Internacional de Valencia

17.01.22

- Librerías Spark ML(lib)
- Fases: entrenamiento, pruebas y producción
- Pipeline
- Feature Engineering
- Modelos
- Personalización

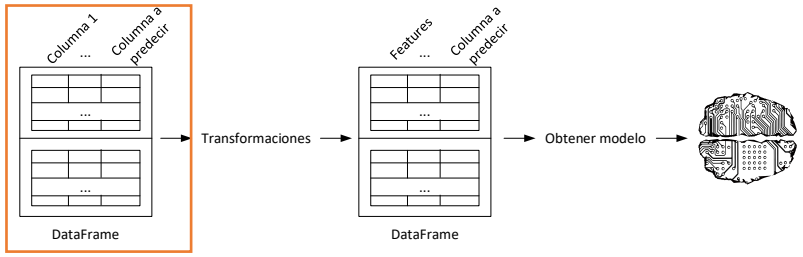
- Spark MLlib:
  - Machine Learning sobre la API RDD
  - Modo mantenimiento: desde spark 2.0 no se añaden nuevas funcionalidades
- Spark.ML:
  - Machine Learning sobre API DataFrame

- Fase de entrenamiento:

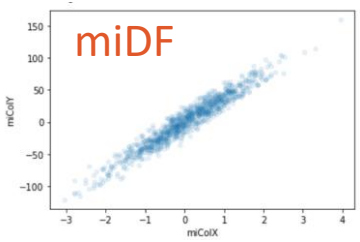


  - El modelo se utilizará para predecir

- Fase de entrenamiento:
  - DataFrame de entrenamiento:




```
from pyspark.sql import SparkSession
from sklearn.datasets import make_regression
import pandas as pd

#Creamos un dataframe de pandas con dos columnas correlacionadas de forma aleatoria
miX, miY = make_regression(n_samples = 1000,
                           n_features = 1,
                           n_informative = 1,
                           n_targets = 1,
                           noise = 10,
                           random_state = 1)

miPandasDF = pd.DataFrame({"miColX": miX.flatten(),
                           "miColY": miY})

spark = SparkSession.builder.appName('miEjemplo').getOrCreate()
miDF = spark.createDataFrame(miPandasDF)
```

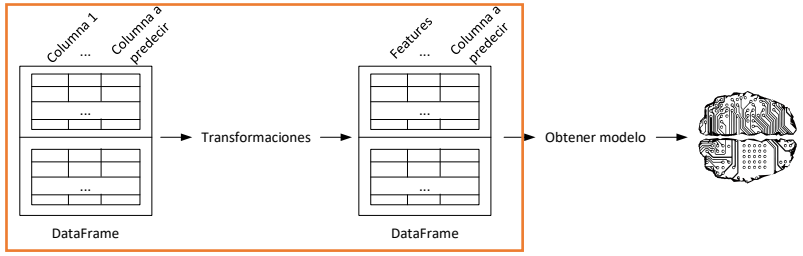


Columna a predecir

miDF:

miColX	miColY
-1.0678765764865552	-22.338036449635013
-0.2915945955008327	-4.09375012893803
0.05080775477602897	14.731253346489734
0.6218035043055724	38.24855703084982
-0.5457741679825677	-15.619815045857369
1.5550159943330888	66.61775597496538
-0.2097529354225541	-3.235168106189642

- Fase de entrenamiento:
  - Tansformación(es):
    - Transformer: Transforma un DataFrame en otro (**feature engineeing**)




```
from pyspark.ml.feature import VectorAssembler

#Creamos el transformador
miVectorAssembler = VectorAssembler(inputCols = ["miColX"], outputCol = "miFeatures")

#Transformamos los datos
miDF_transformado = miVectorAssembler.transform(miDF)

miDF_transformado.show(truncate = False)
```

Columna a predecir

Features con las que predecir

miDF

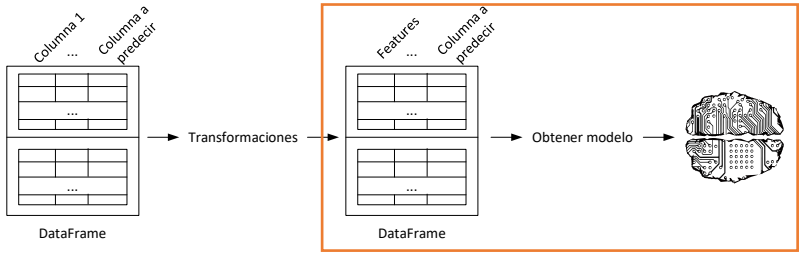
```
+-------------------+-------------------+
|miColX             |miColY             |
+-------------------+-------------------+
|-1.0678765764865552|-22.338036449635013|
|-0.2915945955008327|-4.09375012893803  |
|0.05080775477602897|14.731253346489734 |
|0.6218035043055724 |38.24855703084982  |
|-0.5457741679825677|-15.619815045857369|
|1.5550159943330888 |66.61775597496538  |
|-0.2097529354225541|-3.235168106189642 |
```

→ Transformación (vectorAssembler) →

miDF_transformado

```
+-------------------+-------------------+----------------------+
|miColX             |miColY             |miFeatures            |
+-------------------+-------------------+----------------------+
|-1.0678765764865552|-22.338036449635013|[-1.0678765764865552] |
|-0.2915945955008327|-4.09375012893803  |[-0.2915945955008327] |
|0.05080775477602897|14.731253346489734 |[0.05080775477602897] |
|0.6218035043055724 |38.24855703084982  |[0.6218035043055724]  |
|-0.5457741679825677|-15.619815045857369|[-0.5457741679825677] |
|1.5550159943330888 |66.61775597496538  |[1.5550159943330888]  |
|-0.2097529354225541|-3.235168106189642 |[-0.2097529354225541]|
```

- Fase de entrenamiento:
  - Obtener el modelo:
    - Estimator: Transforma un DataFrame en un modelo (**entrenamiento**)
    - Modelo: Hace la predicción de un DataFrame (**pruebas o producción**)
      - Se ajusta a los datos de entrenamiento

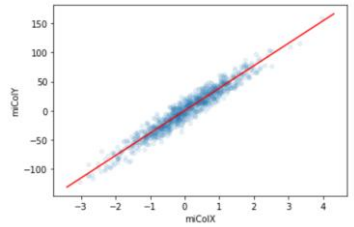



```
from pyspark.ml.regression import LinearRegression

#Creamos el estimador
miLr = LinearRegression(featuresCol= "miFeatures", labelCol = "miColY", predictionCol = "miPrediccion")

#Creamos el modelo (ajustamos a los datos)
miLrModel = miLr.fit(miDF_transformado)
```

```
>7
```

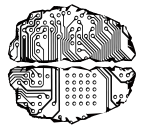
```
+-------------------+-------------------+--------------------+
|miColX             |miColY             |miFeatures          |
+-------------------+-------------------+--------------------+
|-1.0678765764865552|-22.338036449635013|[-1.0678765764865552]|
|-0.2915945955008327|-4.09375012893803  |[-0.2915945955008327]|
|0.05080775477602897|14.731253346489734 |[0.05080775477602897]|
|0.6218035043055724 |38.24855703084982  |[0.6218035043055724] |
|-0.5457741679825677|-15.619815045857369|[-0.5457741679825677]|
|1.5550159943330888 |66.61775597496538  |[1.5550159943330888] |
|-0.2097529354225541|-3.235168106189642 |[-0.2097529354225541]|
```

miDF_transformado

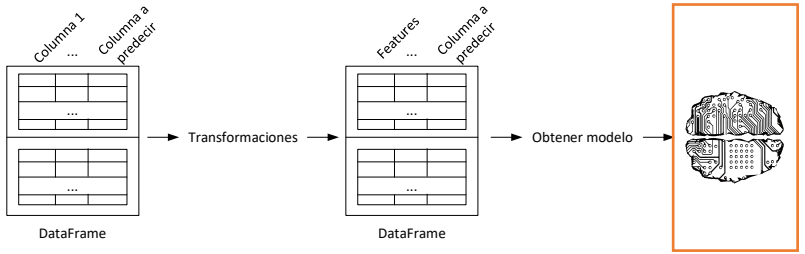


miLrModel

→ Estimator (LinearRegression) →

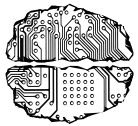


- Fase de entrenamiento:
  - Mostrar resumen del modelo:
    - LinearRegressionTrainingSummary
    - Depende del modelo, cambia la clase

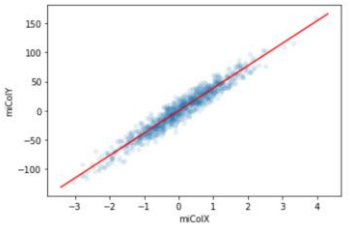



```
miResumenEntrenamientoLr = miLrModel.summary
print("Resumen de entrenamiento:")
print(f"Coeficientes: {miLrModel.coefficients}")
print(f"Pendiente: {miLrModel.coefficients[0]}")
print(f"Inercept: {miLrModel.intercept}")
print(miResumenEntrenamientoLr.coefficientStandardErrors)
```

miLrModel



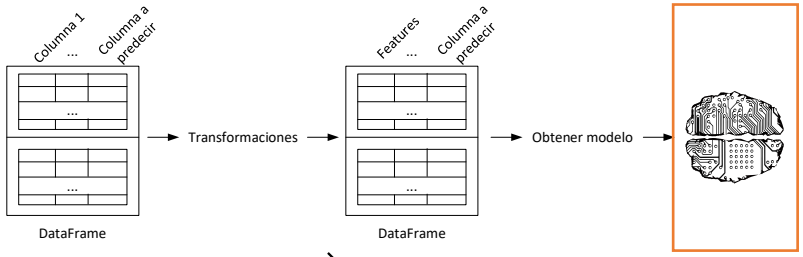
resumen

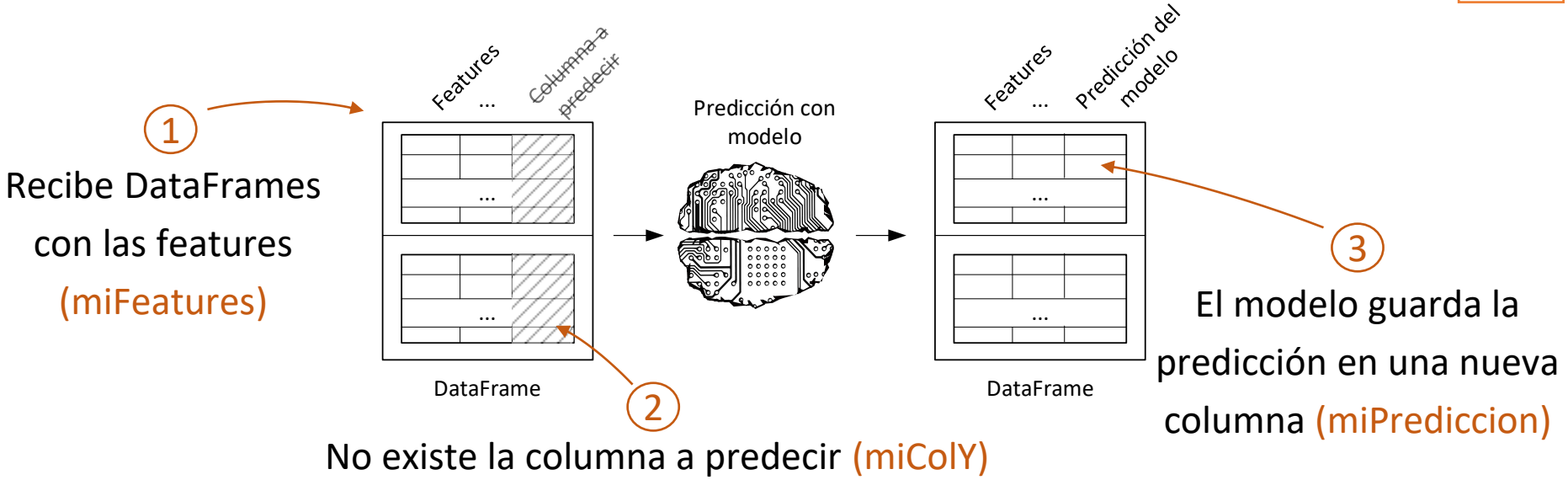



```
Coeficientes: [38.47831607877498]
Pendiente: 38.47831607877498
Inercept: 0.2905002991350191
[0.3233131012624349, 0.31741962805979596]
...
```

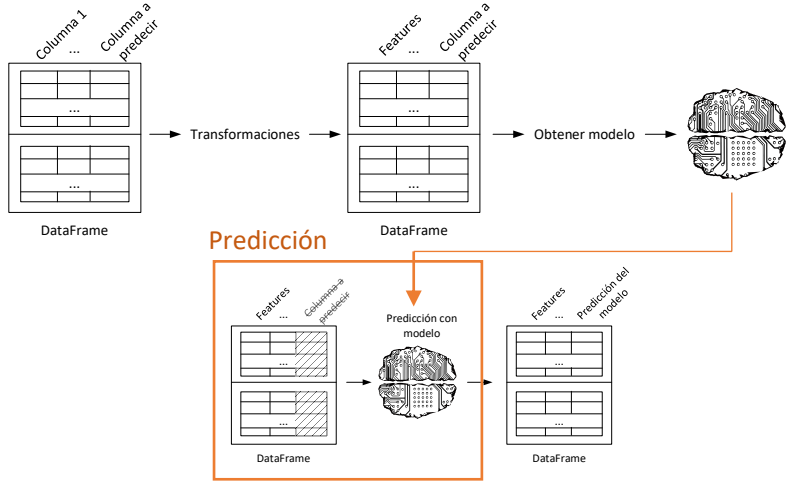
```
>8
```

- Fase de entrenamiento:
  - Resultado: modelo (miLrModel)



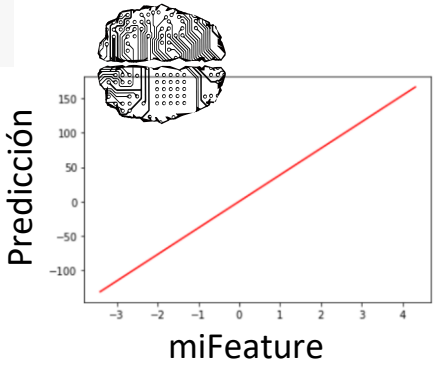


- Fase de entrenamiento:
  - Resultado: modelo (miLrModel)

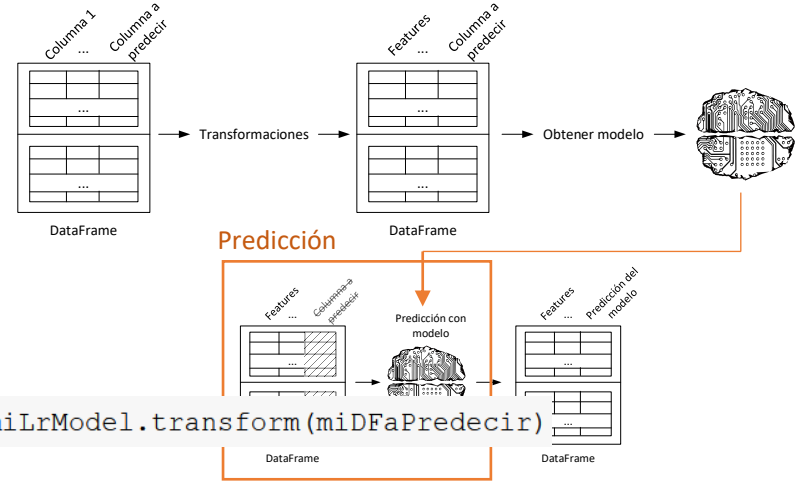



```
from pyspark.sql import Row
import pyspark.ml.linalg as ml

miDFaPredecir = spark.createDataFrame([
    Row(miFeatures = ml.DenseVector([0.5])),
    Row(miFeatures = ml.DenseVector([3.5]))
])
```

```
+-----------+
|miFeatures|
+-----------+
|[0.5]      |
|[3.5]      |
+-----------+
```



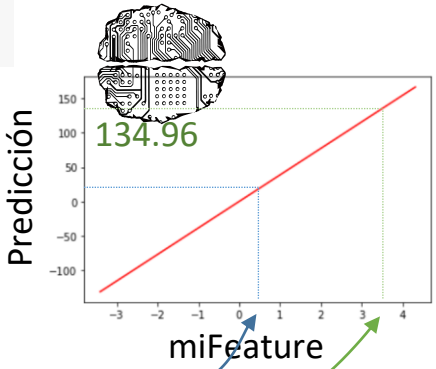
- Fase de entrenamiento:
  - Resultado: modelo (miLrModel)



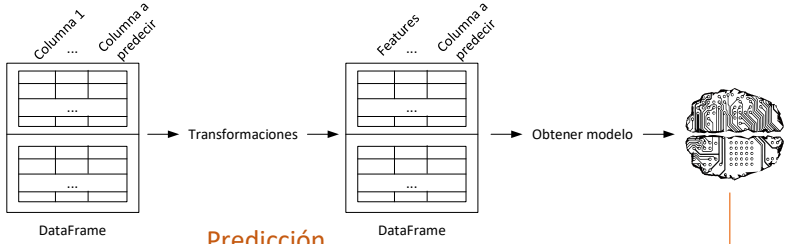

```
from pyspark.sql import Row
import pyspark.ml.linalg as ml

miDFaPredecir = spark.createDataFrame([
    Row(miFeatures = ml.DenseVector([0.5])),
    Row(miFeatures = ml.DenseVector([3.5]))
])
```

```
miDFconPrediccion = miLrModel.transform(miDFaPredecir)
```

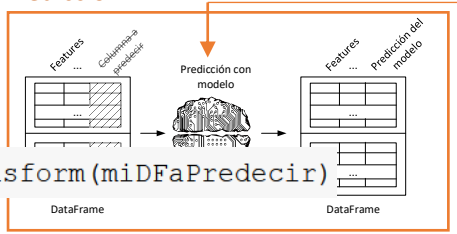
```
+-----------+
|miFeatures|
+-----------+
|[0.5]      |
|[3.5]      |
+-----------+
```



- Fase de entrenamiento:
  - Resultado: modelo (miLrModel)



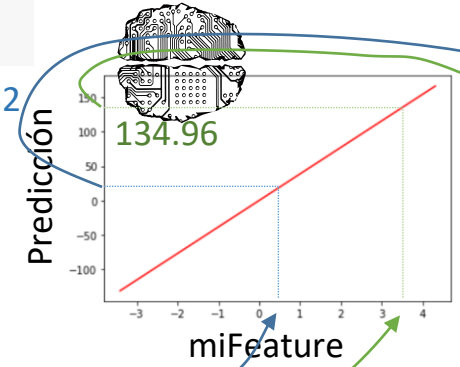
Predicción




```
from pyspark.sql import Row
import pyspark.ml.linalg as ml

miDFaPredecir = spark.createDataFrame([
    Row(miFeatures = ml.DenseVector([0.5])),
    Row(miFeatures = ml.DenseVector([3.5]))
])
```

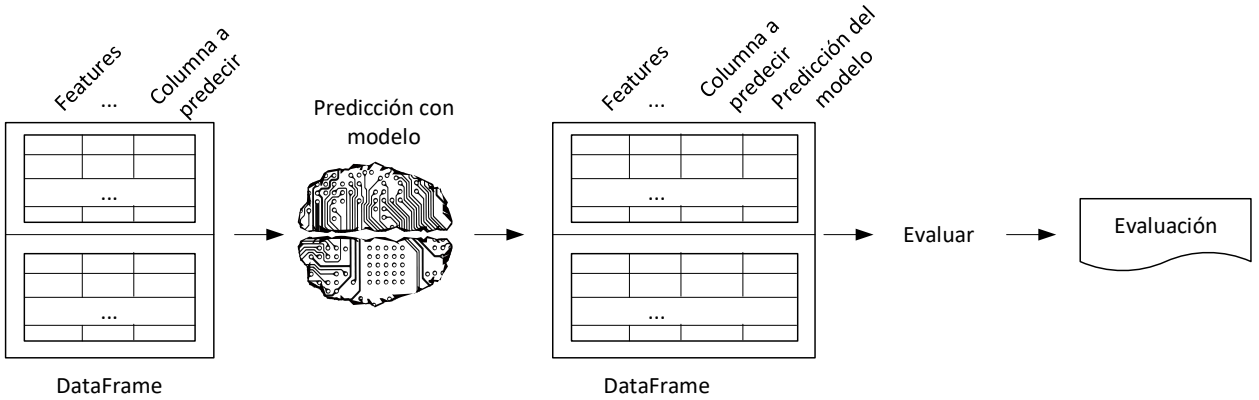
```
miDFconPrediccion = miLrModel.transform(miDFaPredecir)
```

```
+-----------+
|miFeatures|
+-----------+
|[0.5]      |
|[3.5]      |
+-----------+
```



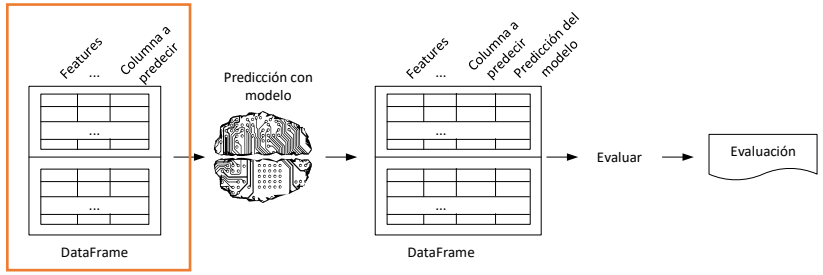

```
miDFconPrediccion.show(truncate = False)
+-----------+------------------+
|miFeatures|miPrediccion      |
+-----------+------------------+
|[0.5]      |19.52965833852251 |
|[3.5]      |134.96460657484747|
+-----------+------------------+
```

- Fase de entrenamiento
- **Fase de pruebas**:



  - DataFrame de prueba: columna con valor a predecir
  - Se compara con la predicción del modelo

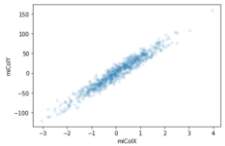
- Fase de entrenamiento
- **Fase de pruebas**:



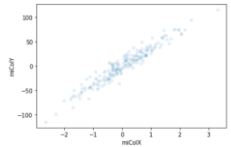
  - Dividir los datos en entrenamiento y prueba

```
miDF_entrenamiento, miDF_test = miDF.randomSplit([0.8, 0.2], seed = 1)
```

80% de los datos para entrenar el modelo

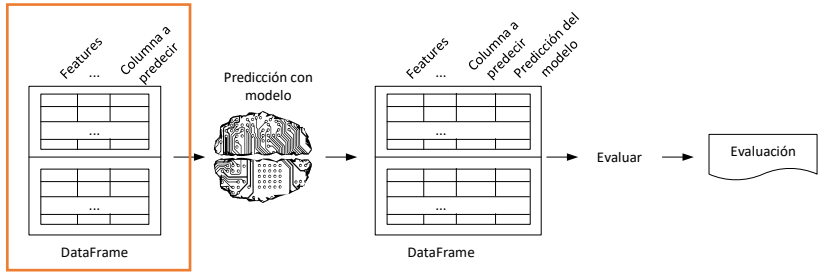


20% de los datos para evaluar el modelo



Estos datos no se utilizan para entrenar
Puede que el modelo nunca los haya visto
Contienen cuál es el valor a predecir

- Fase de entrenamiento
- **Fase de pruebas**:
  - Obtener la features ejecutando las transformaciones




```
miDFtest.show(truncate = False)
```

```
+-------------------+-------------------+
|miColX             |miColY             |
+-------------------+-------------------+
|-2.3015386968802827|-98.8925300891861  |
|-1.7879128911997157|-68.20000855930708 |
|-1.6993336047222958|-60.81550376846737 |
|-1.674195807618932 |-50.086788432071124|
|-1.6374495930083417|-62.3041951990063  |
```

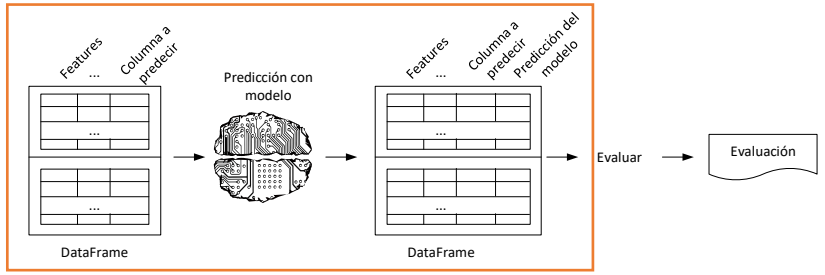
Transformaciones

```
miDF_test_transformado.show(truncate = False)
```

```
+-------------------+-------------------+----------------------+
|miColX             |miColY             |miFeatures            |
+-------------------+-------------------+----------------------+
|-2.3015386968802827|-98.8925300891861  |[-2.3015386968802827]|
|-1.7879128911997157|-68.20000855930708 |[-1.7879128911997157]|
|-1.6993336047222958|-60.81550376846737 |[-1.6993336047222958]|
|-1.674195807618932 |-50.086788432071124|[-1.674195807618932] |
|-1.6374495930083417|-62.3041951990063  |[-1.6374495930083417]|
```

```
miDF_test_transformado = miVectorAssembler.transform(miDFtest)
```

- Fase de entrenamiento
- **Fase de pruebas**:
  - Predecir




```
miDF_test_transformado.show(truncate = False)
```

```
+-------------------+-------------------+---------------------+
|miColX             |miColY             |miFeatures           |
+-------------------+-------------------+---------------------+
|-2.3015386968802827|-98.8925300891861  |[-2.3015386968802827]|
|-1.7879128911997157|-68.20000855930708 |[-1.7879128911997157]|
|-1.6993336047222958|-60.81550376846737 |[-1.6993336047222958]|
|-1.674195807618932 |-50.086788432071124|[-1.674195807618932] |
|-1.6374495930083417|-62.3041951990063  |[-1.6374495930083417]|
```

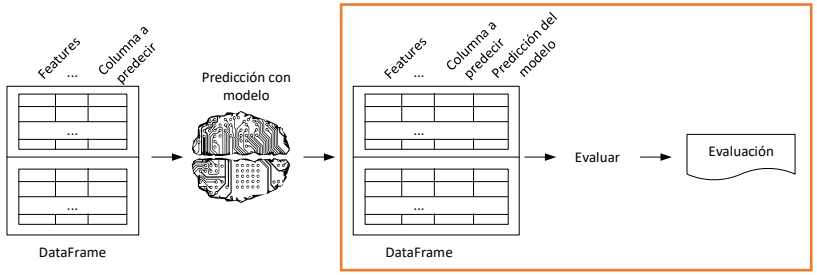
Predicción

```
miPrediccionTest.show(truncate = False)
```

```
+-------------------+-------------------+---------------------+-------------------+
|miColX             |miColY             |miFeatures           |miPrediccion       |
+-------------------+-------------------+---------------------+-------------------+
|-2.3015386968802827|-98.8925300891861  |[-2.3015386968802827]|-88.78564246873839 |
|-1.7879128911997157|-68.20000855930708 |[-1.7879128911997157]|-68.87842541973056 |
|-1.6993336047222958|-60.81550376846737 |[-1.6993336047222958]|-65.4452507963961  |
|-1.674195807618932 |-50.086788432071124|[-1.674195807618932] |-64.47095476699876 |
|-1.6374495930083417|-62.3041951990063  |[-1.6374495930083417]|-63.046737249319385|
```

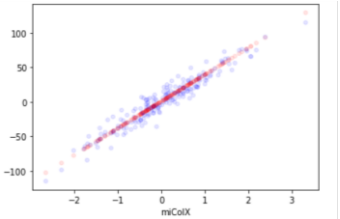
```
miPrediccionTest = miLrModel.transform(miDF_test_transformado)
```

- Fase de entrenamiento
- **Fase de pruebas**:
  - Evaluator: obtiene métricas de calidad el modelo (**pruebas**)




```
miPrediccionTest.show(truncate = False)
```

```
+------------------+------------------+--------------------+------------------+
|miColX            |miColY            |miFeatures          |miPrediccion      |
+------------------+------------------+--------------------+------------------+
|-2.3015386968802827|-98.8925300891861  |[-2.3015386968802827]|-88.78564246873839 |
|-1.7879128911997157|-68.20000855930708 |[-1.7879128911997157]|-68.87842541973056 |
|-1.6993336047222958|-60.81550376846737 |[-1.6993336047222958]|-65.4452507963961  |
|-1.674195807618932 |-50.086788432071124|[-1.674195807618932] |-64.47095476699876 |
|-1.6374495930083417|-62.3041951990063  |[-1.6374495930083417]|-63.046737249319385|
```

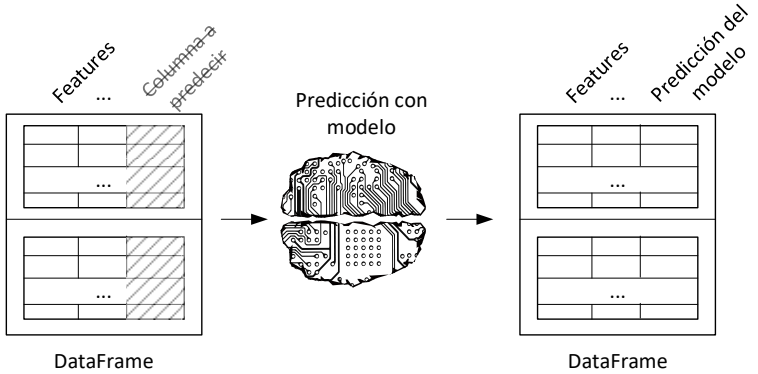


Evaluación

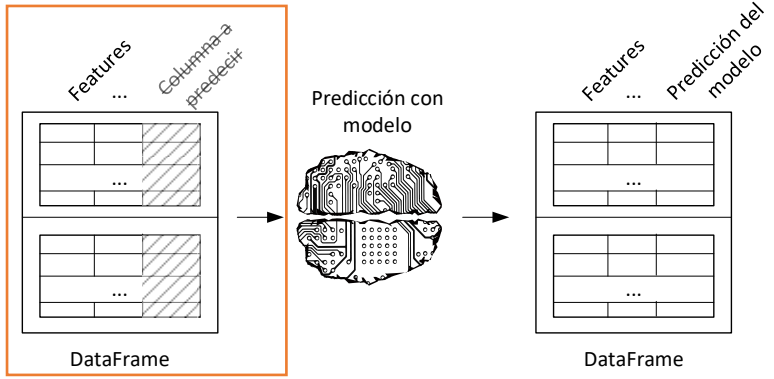
```
El R-squared es 0.9251344582533008
```

```
miEvaluator = RegressionEvaluator(predictionCol = "miPrediccion", labelCol = "miColY")
miPrediccionTest_r2 = miEvaluator.setMetricName("r2").evaluate(miPrediccionTest)
print(f'El R-squared es {miPrediccionTest_r2}')
```

- Fase de entrenamiento
- Fase de pruebas
- **Fase de producción:**
  - Transformar el DataFrame hasta conseguir las features
  - Predecir



- Fase de entrenamiento
- Fase de pruebas
- **Fase de producción:**



Se cambia la semilla para tener nuevos datos

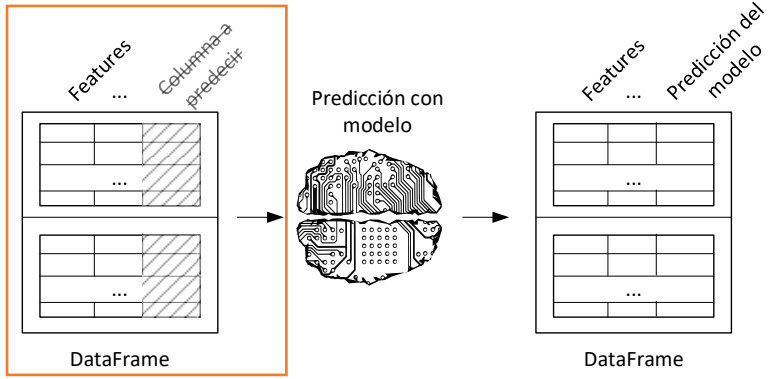
```
miX, miY = make_regression(n_samples = 50,
                           n_features = 1,
                           n_informative = 1,
                           n_targets = 1,
                           noise = 10,
                           random_state = 2)

miPandasDF = pd.DataFrame({"miColX": miX.flatten(),
                           "miColY": miY})

miDFProduccion = spark.createDataFrame(miPandasDF)
miDFProduccion = miDFProduccion.drop("miColY")
```

```
miDFProduccion.show(truncate = False)
```

```
+--------------------+
|miColX              |
+--------------------+
|-0.878107893240342  |
|0.6113407795737174  |
|0.11272650481664892 |
|0.5514540445464243  |
|-1.1179254451135168 |
```

Se elimina la columna que se quiere predecir (en producción no se conoce)

- Fase de entrenamiento
- Fase de pruebas
- **Fase de producción:**



- Transformar el DataFrame para obtener las features

```
miDFProduccion.show(truncate = False)
```

```
+---------------------+
|miColX               |
+---------------------+
|-0.878107893240342   |
|0.6113407795737174   |
|0.11272650481664892  |
|0.5514540445464243   |
|-1.1179254451135168  |
```

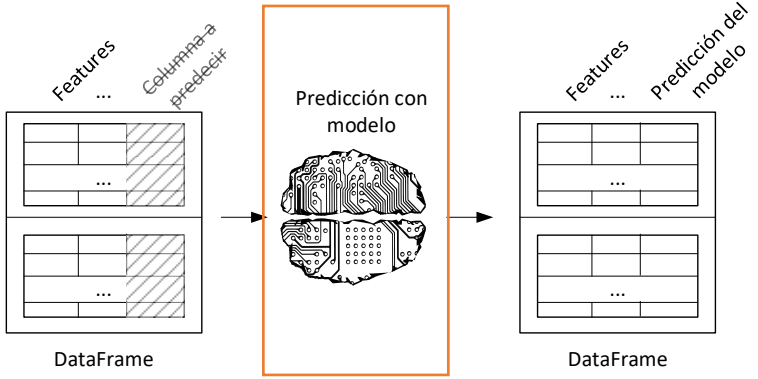
Transformaciones →

```
miDFProduccion_transformados.show(truncate = False)
```

```
+---------------------+-----------------------+
|miColX               |miFeatures             |
+---------------------+-----------------------+
|-0.878107893240342   |[-0.878107893240342]   |
|0.6113407795737174   |[0.6113407795737174]   |
|0.11272650481664892  |[0.11272650481664892]  |
|0.5514540445464243   |[0.5514540445464243]   |
|-1.1179254451135168  |[-1.1179254451135168]  |
```

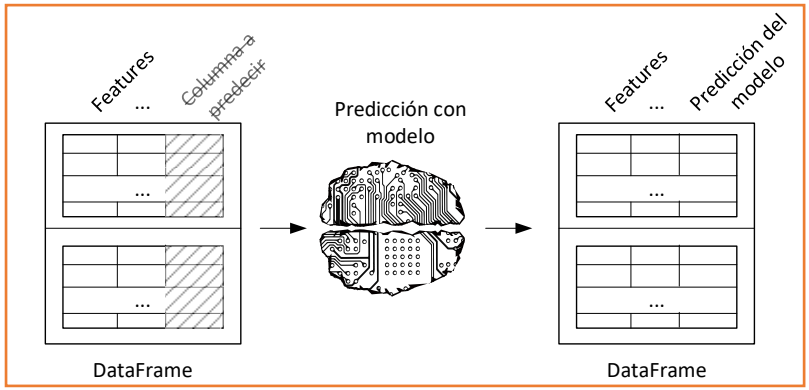
```
miVectorAssembler = VectorAssembler(inputCols = ["miColX"], outputCol = "miFeatures")
miDFProduccion_transformados = miVectorAssembler.transform(miDFProduccion)
```

- Fase de entrenamiento
- Fase de pruebas
- **Fase de producción:**
  - Cargar modelo:
    - Se guardo previamente: `miLrModel.save("miModeloGuardado")`
    - Cargar: `miLrModelCargado = LinearRegressionModel.load("miModeloGuardado")`



Fases de Machine Learning

- Fase de entrenamiento
- Fase de pruebas
- **Fase de producción:**




```
miDFProduccion_transformados.show(truncate = False)
```

```
+---------------------+------------------------+
|miColX               |miFeatures              |
+---------------------+------------------------+
|-0.878107893240342   |[-0.878107893240342]    |
|0.6113407795737174   |[0.6113407795737174]    |
|0.11272650481664892  |[0.11272650481664892]   |
|0.5514540445464243   |[0.5514540445464243]    |
|-1.1792544511351681  |[-1.1792544511351681]   |
```

Predicción

```
miDFProduccion_prediccion.show(truncate = False)
```

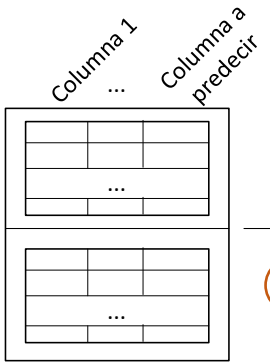
```
+---------------------+------------------------+--------------------+
|miColX               |miFeatures              |miPrediccion        |
+---------------------+------------------------+--------------------+
|-0.878107893240342   |[-0.878107893240342]    |-33.61601187598163  |
|0.6113407795737174   |[0.6113407795737174]    |24.112353181101348  |
|0.11272650481664892  |[0.11272650481664892]   |4.786956206215395   |
|0.5514540445464243   |[0.5514540445464243]    |21.79125050409493   |
|-1.1792544511351681  |[-1.1792544511351681]   |-42.910911011696165 |
```

```
miDFProduccion_prediccion = miLrModelCargado.transform(miDFProduccion_transformados)
```

- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase

Pipelines

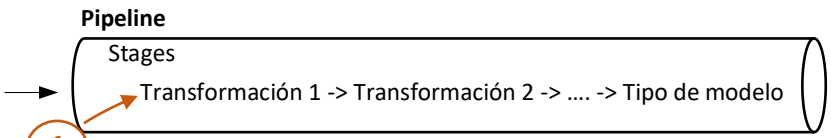
- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase
  - Fase de entrenamiento:

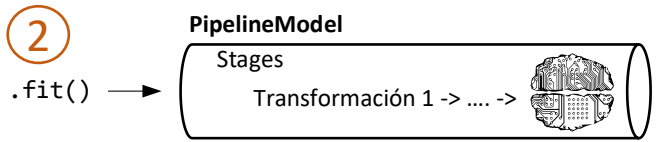


DataFrame

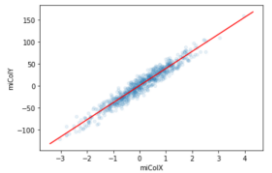
miColX	miColY
-1.0678765764865552	-22.3380364496350 13
-0.2915945955008327	-4.09375012893803
0.05080775477602897	14.731253346489734
0.6218035043055724	38.24855703084982
-0.5457741679825677	-15.619815045857369

Dataset en plano (sin features)





VectorAssembler +



```
miPipelineModel = miPipeline.fit(miDF_entrenamiento)
```

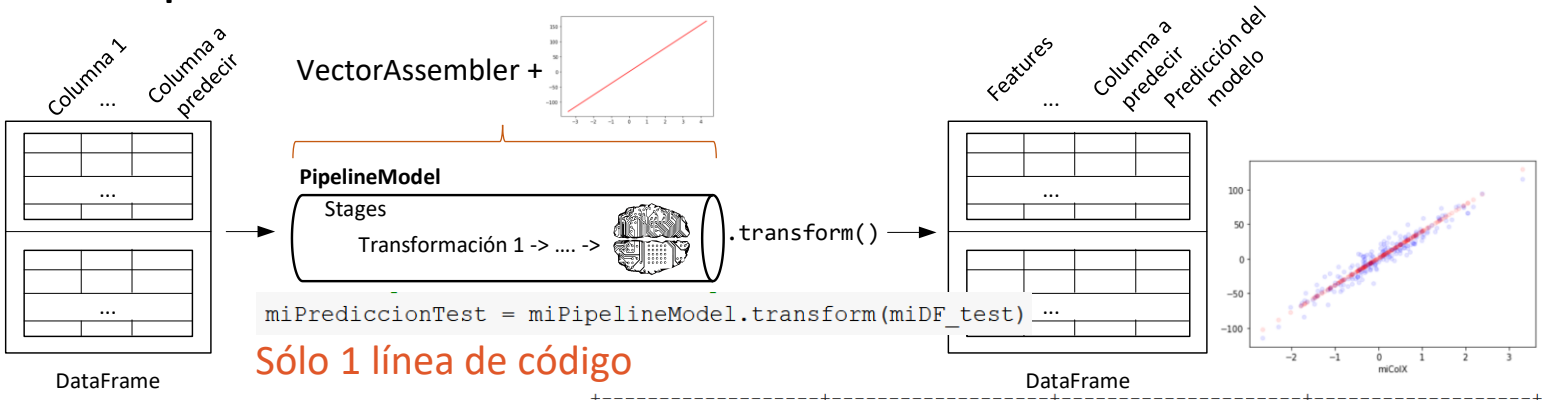
```
#Creamos las stages
miVectorAssembler = VectorAssembler(inputCols = ["miColX"], outputCol = "miFeatures")
miLr = LinearRegression(featuresCol= "miFeatures", labelCol = "miColY", predictionCol = "miPrediccion"

#Creamos el pipeline
miPipeline = Pipeline(stages = [miVectorAssembler,
                                miLr])
```

- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase
  - Fase de entrenamiento:
    - Obtener información del modelo:

```
miLrModel = miPipelineModel.stages[-1]
miResumenEntrenamientoLr = miLrModel.summary
print("Resumen de entrenamiento:")
print(f"Coeficientes: {miLrModel.coefficients}")
print(f"Pendiente: {miLrModel.coefficients[0]}")
print(f"Inercept: {miLrModel.intercept}")
print(miResumenEntrenamientoLr.coefficientStandardErrors)
```

```
Resumen de entrenamiento:
Coeficientes: [38.75821041084623]
Pendiente: 38.75821041084623
Inercept: 0.41787861365244516
[0.3592893130906634, 0.356091916460506]
```

- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase
  - Fase de pruebas:

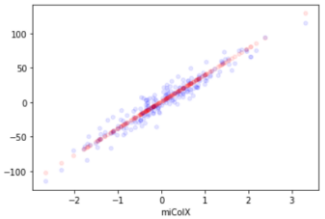



```
+-------------------+-------------------+
|miColX             |miColY             |
+-------------------+-------------------+
|-2.3015386968802827|-98.8925300891861  |
|-1.7879128911997157|-68.20000855930708 |
|-1.6993336047222958|-60.81550376846737 |
|-1.674195807618932 |-50.086788432071124|
|-1.637449593008341|-62.3041951990063  |
```

```
+-------------------+-------------------+--------------------+-------------------+
|miColX             |miColY             |miFeatures          |miPrediccion       |
+-------------------+-------------------+--------------------+-------------------+
|-2.3015386968802827|-98.8925300891861  |[-2.3015386968802827]|-88.78564246873839 |
|-1.7879128911997157|-68.20000855930708 |[-1.7879128911997157]|-68.87842541973056 |
|-1.6993336047222958|-60.81550376846737 |[-1.6993336047222958]|-65.4452507963961  |
|-1.674195807618932 |-50.086788432071124|[-1.674195807618932] |-64.47095476699876 |
```

- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase
  - Fase de pruebas:
    - Evaluación

```
miEvaluator = RegressionEvaluator(predictionCol = "miPrediccion", labelCol = "miColY")
miPrediccionTest_r2 = miEvaluator.setMetricName("r2").evaluate(miPrediccionTest)
print(f'El R-squared es {miPrediccionTest_r2}')
```



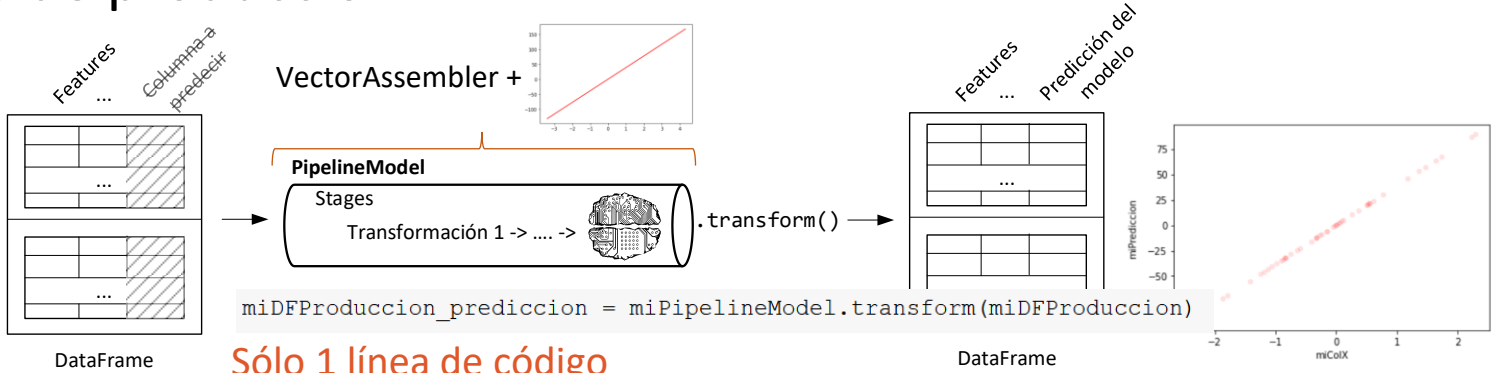

```
+------------------+------------------+--------------------+-------------------+
|miColX            |miColY            |miFeatures          |miPrediccion       |
+------------------+------------------+--------------------+-------------------+
|-2.3015386968802827|-98.8925300891861  |[-2.3015386968802827]|-88.78564246873839 |
|-1.7879128911997157|-68.20000855930708 |[-1.7879128911997157]|-68.87842541973056 |
|-1.6993336047222958|-60.81550376846737 |[-1.6993336047222958]|-65.4452507963961  |
|-1.674195807618932 |-50.086788432071124|[-1.674195807618932] |-64.47095476699876 |
```

Evaluación → `El R-squared es 0.9251344582533008`

- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase
  - Fase de pre-producción:
    - Guardar PipelineModel: `miPipelineModel.save("miModelo")`
    - Cargar PipelineModel:

```
from pyspark.ml import PipelineModel

miPipelineModel = PipelineModel.load("miModelo")
```

- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase
  - Fase de producción:




```
+--------------------+
|miColX              |
+--------------------+
|-0.878107893240342  |
|0.6113407795737174  |
|0.11272650481664892 |
|0.5514540445464243  |
|-1.1179254451135168 |
```

```
+--------------------+--------------------+--------------------+
|miColX              |miFeatures          |miPrediccion        |
+--------------------+--------------------+--------------------+
|-0.878107893240342  |[-0.878107893240342]|-33.61601187598163  |
|0.6113407795737174  |[0.6113407795737174]|24.112353181101348  |
|0.11272650481664892 |[0.11272650481664892]|4.786956206215395  |
|0.5514540445464243  |[0.5514540445464243]|21.79125050409493   |
|-1.1179254451135168 |[-1.1179254451135168]|-42.910911011696165|
```

- Integran transformaciones y estimadores (**stages**)
- Se pueden utilizar en cualquier fase
- Hiper-parámetros:
  - TrainValidationSplit
  - CrossValidation

Feature Engineering

- Feature Transformers
  - Tokenizer
  - StopWordsRemover
  - n-gram
  - Binarizer
  - PCA
  - PolynomialExpansion
  - Discrete Cosine Transform (DCT)
  - StringIndexer
  - IndexToString
  - OneHotEncoder
  - VectorIndexer
  - Interaction
  - Normalizer
  - StandardScaler
  - RobustScaler
  - MinMaxScaler
  - MaxAbsScaler
  - Bucketizer
  - ElementwiseProduct
  - SQLTransformer
  - VectorAssembler
  - VectorSizeHint
  - QuantileDiscretizer
  - Imputer
- Feature Extractors
  - TF-IDF
  - Word2Vec
  - CountVectorizer
  - FeatureHasher
- Feature Selectors
  - VectorSlicer
  - RFormula
  - ChiSqSelector
  - UnivariateFeatureSelector
  - VarianceThresholdSelector
- Locality Sensitive Hashing
  - LSH Operations
    - Feature Transformation
    - Approximate Similarity Join
    - Approximate Nearest Neighbor Search
  - LSH Algorithms
    - Bucketed Random Projection for Euclidean Distance
    - MinHash for Jaccard Distance

Obtenido de https://spark.apache.org/docs/latest/ml-features.html

Ejemplo: StringIndexer

Transforma categorías en IDs

```
+--------------------+-------------+--------------------+-------------+
|          Store name|Category Name|Volume Sold (liters)|Sale (Dollars)|
+--------------------+-------------+--------------------+-------------+
|         Hy-Vee Food|      Tequila|                24.0|       252.72|
|         Hy-Vee Food|      Tequila|                 1.5|        47.08|
|         Hy-Vee Food|       Mezcal|                 4.5|        180.0|
|Bootleggin' Barzi...|       Mezcal|                0.75|        19.89|
+--------------------+-------------+--------------------+-------------+
```

```
+--------------------+-------------+--------------------+-------------+-----------------+
|          Store name|Category Name|Volume Sold (liters)|Sale (Dollars)|CategoryNameIndex|
+--------------------+-------------+--------------------+-------------+-----------------+
|         Hy-Vee Food|      Tequila|                24.0|       252.72|              1.0|
|         Hy-Vee Food|      Tequila|                 1.5|        47.08|              1.0|
|         Hy-Vee Food|       Mezcal|                 4.5|        180.0|              0.0|
|Bootleggin' Barzi...|       Mezcal|                0.75|        19.89|              0.0|
+--------------------+-------------+--------------------+-------------+-----------------+
```

```
from pyspark.ml.feature import StringIndexer

miRDD = spark.sparkContext.parallelize([("Hy-Vee Food", "Tequila", 24.0, 252.72),
                                        ("Hy-Vee Food", "Tequila", 1.5, 47.08),
                                        ("Hy-Vee Food", "Mezcal", 4.5, 180.0),
                                        ("Bootleggin' Barzini's Fin", "Mezcal", 0.75, 19.89)])

miSchema = StructType([
                       StructField(name = 'Store name',            dataType = StringType(),  nullable = True),
                       StructField(name = 'Category Name',         dataType = StringType(),  nullable = True),
                       StructField(name = 'Volume Sold (liters)',  dataType = FloatType(),   nullable = True),
                       StructField(name = 'Sale (Dollars)',        dataType = FloatType(),   nullable = True)
                      ])

miDF = spark.createDataFrame(miRDD, schema = miSchema)
miDF.printSchema()
miDF.show()


miIndexer = StringIndexer(inputCol = "Category Name", outputCol = "CategoryNameIndex")
miModel = miIndexer.fit(miDF)
miDFIndexed = miModel.transform(miDF)
```

Ejemplo: IndexToString

Transforma IDs en categorías

```
+------------------------+------------------+-------------+-----------------+
|Store name              |Volume Sold (liters)|Sale (Dollars)|CategoryNameIndex|
+------------------------+------------------+-------------+-----------------+
|Hy-Vee Food             |24.0              |252.72       |0.0              |
|Hy-Vee Food             |1.5               |47.08        |0.0              |
|Hy-Vee Food             |4.5               |180.0        |1.0              |
|Bootleggin' Barzini's Fin|0.75              |19.89        |1.0              |
+------------------------+------------------+-------------+-----------------+
```

```
+------------------------+------------------+-------------+-----------------+-------------+
|Store name              |Volume Sold (liters)|Sale (Dollars)|CategoryNameIndex|Category Name|
+------------------------+------------------+-------------+-----------------+-------------+
|Hy-Vee Food             |24.0              |252.72       |0.0              |Tequila      |
|Hy-Vee Food             |1.5               |47.08        |0.0              |Tequila      |
|Hy-Vee Food             |4.5               |180.0        |1.0              |Mezcal       |
|Bootleggin' Barzini's Fin|0.75              |19.89        |1.0              |Mezcal       |
+------------------------+------------------+-------------+-----------------+-------------+
```

```
miDFSinCategoria = miDFIndexed.drop("Category Name")
miDFSinCategoria.show(truncate = False)
```

```
miConverter = IndexToString(inputCol = "CategoryNameIndex", outputCol = "Category Name")
miConvertedDF = miConverter.transform(miDFSinCategoria)
miConvertedDF.show(truncate = False)
```

El mapeo categorías <-> IDs lo tiene guardado como metadatos del schema

```
miDFSinCategoria.schema["CategoryNameIndex"].metadata
```

```
{'ml_attr': {'name': 'CategoryNameIndex',
  'type': 'nominal',
  'vals': ['Tequila', 'Mezcal']}}
```

- **Cortar datos**: recibe un umbral y corta los datos inferiores
  - Crear los parámetros: umbral
    - El Transformador puede recibir un umbral para cortar los datos

```
class HasUmbral(Params):
  umbral = Param(Params._dummy(), "umbral", "umbral maximo de los datos", typeConverter = TypeConverters.toFloat)

  def __init__(self):
      super(HasUmbral, self).__init__()
      self._setDefault(umbral = None)

  def setUmbral(self, value):
      return self._set(umbral = value)

  def getUmbral(self):
      return self.getOrDefault(self.umbral)
```

- **Cortar datos**: recibe un umbral y corta los datos inferiores
  - Crear los parámetros: umbral
  - Crear el transformador:

Recibe columna de entrada y salida

Indicamos que recibirá el parámetro umbral

```
class CortarDatos(Transformer, HasInputCol, HasOutputCol, HasUmbral, DefaultParamsReadable, DefaultParamsWritable):
  @keyword_only
  def __init__(self, inputCol = None, outputCol = None, umbral = None):
    super(CortarDatos, self).__init__()
    kwargs = self._input_kwargs
    self.setParams(**kwargs)

  @keyword_only
  def setParams(self, inputCol = None, outputCol = None, umbral = None):
    kwargs = self._input_kwargs
    return self._set(**kwargs)

  def _transform(self, dataset):
    output_column = self.getOutputCol()
    input_column = self.getInputCol()
    umbral = self.getUmbral()
    return dataset.withColumn(output_column,
                              when(col(input_column) < umbral, umbral)
                              .otherwise(col(input_column)))
```

Se puede guardar un PipelineModel que tenga este transformador

Transforma los datos cortándolos

- **Cortar datos**: recibe un umbral y corta los datos inferiores
  - Crear los parámetros: umbral
  - Crear el transformador
  - Ejemplo:

```
#Ejecutando la transformación directamente
miCortador = CortarDatos(inputCol = "miColX", outputCol = "miColXcortada", umbral = 0)
print("Dataset después de ejecutar el transformador:")
miDFCortado = miCortador.transform(miDF)
miDFCortado.show(truncate = False)

#Mostramos el resultado de la trnasformación
miDFCortado.to_pandas_on_spark().plot.scatter(x = 'miColXcortada', y = 'miColY', alpha = 0.1)
```

miDF

```
+--------------------+-------------------+
|miColX              |miColY             |
+--------------------+-------------------+
|-1.0678765764865552 |-22.338036449635013|
|-0.2915945955008327 |-4.09375012893803  |
|0.05080775477602897 |14.731253346489734 |
|0.6218035043055724  |38.24855703084982  |
|-0.5457741679825677 |-15.619815045857369|
```

miDFCortado

```
+--------------------+-------------------+--------------------+
|miColX              |miColY             |miColXcortada       |
+--------------------+-------------------+--------------------+
|-1.0678765764865552 |-22.338036449635013|0.0                 |
|-0.2915945955008327 |-4.09375012893803  |0.0                 |
|0.05080775477602897 |14.731253346489734 |0.05080775477602897 |
|0.6218035043055724  |38.24855703084982  |0.6218035043055724  |
|-0.5457741679825677 |-15.619815045857369|0.0                 |
```

- **Media Estimator**: obtiene un modelo con la media
  - Crear los parámetros: media
    - El modelo tiene que ser guardar la media del entrenamiento

```
#Parametro que vamos a utilizar en el modelo.
class HasMedia(Params):
  media = Param(Params._dummy(), "media", "media de los datos", typeConverter = TypeConverters.toFloat)

  def __init__(self):
      super(HasMedia, self).__init__()
      self._setDefault(media = None)

  def setMedia(self, value):
      return self._set(media = value)
```

- **Media Estimator**: obtiene un modelo con la media
  - Crear los parámetros: media
  - Crear el modelo que devuelve la media de entrenamiento

```python
#Creamos el modelo. Es un modelo sencillo que contiene la media de los datos de entrenamiento y devuelve la media para toda
class MediaModel(Model, HasInputCol, HasPredictionCol, HasMedia, DefaultParamsReadable, DefaultParamsWritable):

  @keyword_only
  def __init__(self, inputCol = None, predictionCol = None, media = None):
    super(MediaModel, self).__init__()
    kwargs = self._input_kwargs
    self.setParams(**kwargs)

  @keyword_only
  def setParams(self, inputCol = None, predictionCol = None, media = None):
    kwargs = self._input_kwargs
    return self._set(**kwargs)

  def _transform(self, dataset):
    input_column = self.getInputCol()
    prediction_column = self.getPredictionCol()

    #Obtenemos la informacion que se define lo que debe predecir el modelo (en este caso del entrenamiento solo nos quedamos
    miMediaEntrenamiento = self.getMedia()

    #Devolvemos un dataset que por cada x de input_column nos devuelva una prediccion en la columna prediction_column
    return dataset.withColumn(prediction_column, lit(miMediaEntrenamiento))
```

El modelo recibe una media

El modelo recibe una la columna de entrada y en la que predice

Media con la que se crea el modelo (la de entrenamiento)

Independientemente de la entrada, devuelve la media del entrenamiento

- **Media Estimator**: obtiene un modelo con la media
  - Crear los parámetros: media
  - Crear el modelo que devuelve la media de entrenamiento
  - Crear el estimador que entrena y devuelve el modelo

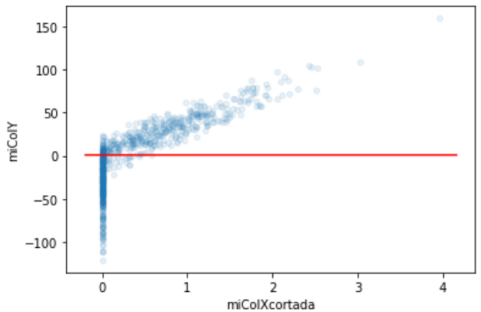
```
#Clase que se encarga de entrenar y devolvernos un modelo
class MediaEstimator(Estimator, HasInputCol, HasPredictionCol, DefaultParamsReadable, DefaultParamsWritable):
  @keyword_only
  def __init__(self, inputCol = None, predictionCol = None):
    super(MediaEstimator, self).__init__()
    kwargs = self._input_kwargs
    self.setParams(**kwargs)

  @keyword_only
  def setParams(self, inputCol = None, predictionCol = None):
    kwargs = self._input_kwargs
    return self._set(**kwargs)

  def _fit(self, dataset):
    input_column = self.getInputCol()
    prediction_column = self.getPredictionCol()

    #Entrenamos
    miMediaEntrenamiento = dataset.agg(mean(prediction_column).alias("miMedia")).first()["miMedia"] #Obtenemos

    #Devolvemos el modelo que tiene esa media
    return MediaModel(inputCol = input_column,
                      predictionCol = prediction_column,
                      media = miMediaEntrenamiento)
```

Se calcula la media de los datos de entrenamiento

Se crea un modelo que siempre devuelva esa media

- **Media Estimator**: obtiene un modelo con la media
  - Crear los parámetros: media
  - Crear el modelo que devuelve la media de entrenamiento
  - Crear el estimador que entrena y devuelve el modelo
  - Ejemplo:

Cortado + Modelo media

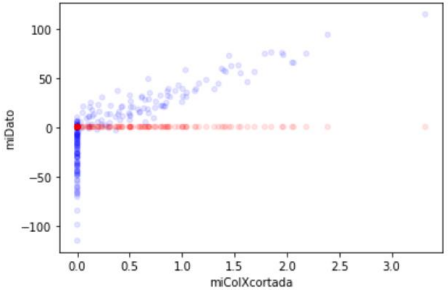


Entrenamiento

```
#Dividimos en entrenamiento y test
miDF_entrenamiento, miDF_test = miDF.randomSplit([0.8, 0.2], seed = 1)

#Creamos las stages
miCortador = CortarDatos(inputCol = "miColX", outputCol = "miColXcortada", umbral = 0)
miMediaEstimator = MediaEstimator(inputCol =  "miColXcortada", predictionCol =  "miPrediccion")

#Creamos el pipeline
miPipeline = Pipeline(stages = [miCortador,
                                miMediaEstimator])

#Entrenamos
miPipelineModel = miPipeline.fit(miDF_entrenamiento)
```

- **Media Estimator**: obtiene un modelo con la media
  - Crear los parámetros: media
  - Crear el modelo que devuelve la media de entrenamiento
  - Crear el estimador que entrena y devuelve el modelo
  - Ejemplo:

Cortado + Modelo media



Pruebas

```
#Hacemos predicciones con el de pruebas
miPrediccionTest = miPipelineModel.transform(miDF_test)
```

```
#Evaluamos
miEvaluator = RegressionEvaluator(predictionCol = "miPrediccion", labelCol = "miColY")
miPrediccionTest_r2 = miEvaluator.setMetricName("r2").evaluate(miPrediccionTest)
print(f'El R-squared es {miPrediccionTest_r2}')
```

```
El R-squared es -0.0011514208446574692
```

```
+-------------------+-------------------+-------------+-------------------+
|miColX             |miColY             |miColXcortada|miPrediccion       |
+-------------------+-------------------+-------------+-------------------+
|-2.3015386968802827|-98.8925300891861  |0.0          |0.4116259060394846|
|-1.7879128911997157|-68.20000855930708 |0.0          |0.4116259060394846|
|-1.6993336047222958|-60.81550376846737 |0.0          |0.4116259060394846|
|-1.674195807618932 |-50.0867884320711 24|0.0          |0.4116259060394846|
|-1.637449593008341 7|-62.3041951990063 |0.0          |0.4116259060394846|
```

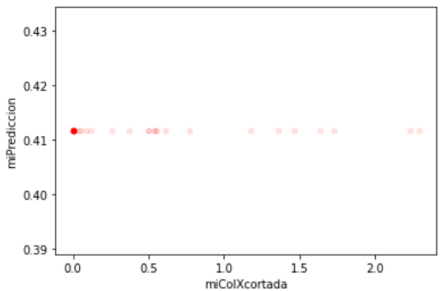
- **Media Estimator**: obtiene un modelo con la media
  - Crear los parámetros: media
  - Crear el modelo que devuelve la media de entrenamiento
  - Crear el estimador que entrena y devuelve el modelo
  - Ejemplo:

Producción

```
#Predecimos los datos de producción
miDFProduccion_prediccion = miPipelineModel.transform(miDFProduccion)
```

```
+--------------------+--------------------+------------------+
|miColX              |miColXcortada       |miPrediccion      |
+--------------------+--------------------+------------------+
|-0.878107893240342  |0.0                 |0.4116259060394846|
|0.6113407795737174  |0.6113407795737174  |0.4116259060394846|
|0.11272650481664892 |0.11272650481664892 |0.4116259060394846|
|0.5514540445464243  |0.5514540445464243  |0.4116259060394846|
|-1.1179254451135168 |0.0                 |0.4116259060394846|
```

Cortado + Modelo media



Resumen Spark ML

- Transformers: Transforma un DataFrame en otro (**feature engineeing**)
  `miDataFrame_transformado = transformador.transform(miDataFrame)`
- Estimators: Transforma un DataFrame en un transformador (ej. modelo) (**entrenamiento**)
  `miModelo = estimador.fit(miDataFrame)`
- Model: Hace la predicción de un DataFrame (**pruebas o producción**)
  `miDataFrame_predicción = miModelo.transform(miDataFrame)`

- Pipeline: ejecuta transformers y estimators anidados para conseguir un modelo (**feature engineeing y entrenamiento**)
  PipelineModel: ejecuta transformers y modelo anidados para predecir **(pruebas o producción)**
  `miPipeline = Pipeline(stages = [transformador1, transformador2,… estimador])`
  `.fit` para obtener modelo y `.transform` para predecir (transformando automáticamente)
- Evaluator: obtiene métricas de calidad el modelo (**pruebas**)
  `evaluador.evaluate(miDataFrame)`
- Param: parámetro que se pueden cambiar para obtener mejores modelos (**optimización**)

- Librerías Spark ML(lib)
- Fases: entrenamiento, pruebas y producción
- Pipeline
- Feature Engineering
- Modelos
- Personalización

Gracias

viu | Universidad Internacional de Valencia