Pyspark Analytics-PART ii

July 25, 2022

1 Part ii: Regional-analysis

[1]:
```
# starting a SparkSession and creating a spark instance
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

[1]: 'C:\\spark-3.0.3-bin-hadoop2.7'

[2]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Task2').getOrCreate()
```

1:Load the data file into a Spark DataFrame (1st DataFrame). Describe the structure of the created data frame

[3]:
```
# Reading the CSV file DataSet and decribing its structure
region1 = spark.read.csv('D:\\Datascience\\Region_info.csv',header=True,␣
↪inferSchema=True)
region1.show(5)
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+--------------------+
|population|fertility| HIV|        CO2|BMI_male|
GDP|BMI_female|life|child_mortality|              Region|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+--------------------+
|      null|     null|null|       null|    null| null|      null|null|
null|                null|
|  34811059|     2.73| 0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|Middle East & Nor…|
|      null|     null|null|       null|    null| null|      null|null|
null|                null|
|  19842251|     6.43| 2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|  Sub-Saharan Africa|
|      null|     null|null|       null|    null| null|      null|null|
null|                null|
+----------+---------+----+-----------+--------+-----+----------+----+----------
```

```
-----+--------------------+
only showing top 5 rows
```

[4]:
```
region1.printSchema() #Structure of the Dataset
```

```
root
 |-- population: integer (nullable = true)
 |-- fertility: double (nullable = true)
 |-- HIV: double (nullable = true)
 |-- CO2: double (nullable = true)
 |-- BMI_male: double (nullable = true)
 |-- GDP: integer (nullable = true)
 |-- BMI_female: double (nullable = true)
 |-- life: double (nullable = true)
 |-- child_mortality: double (nullable = true)
 |-- Region: string (nullable = true)
```

2. Create a new DataFrame (2nd DataFrame) by removing the 'region' column

[5]:
```
# delete single column and creating a new dataframe named Data
data = region1.drop('Region')
data.show(10)
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|population|fertility| HIV|        CO2|BMI_male|
GDP|BMI_female|life|child_mortality|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|      null|     null|null|       null|    null| null|      null|null|
null|
|  34811059|     2.73| 0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  19842251|     6.43| 2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  40381860|     2.24| 0.5|4.785169983| 27.5017|14646|  118.8915|75.5|
15.4|
|      null|     null|null|       null|    null| null|      null|null|
null|
|   2975029|      1.4| 0.1|1.804106217|25.35542| 7383|  132.8108|72.5|
20.0|
|      null|     null|null|       null|    null| null|      null|null|
null|
```

```
|  21370348|      1.96| 0.1|18.01631327|27.56373|41312|  117.3755|81.5|
5.2|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
only showing top 10 rows
```

[6]:
```python
# checking the schema after deletion of the raws
```

[7]:
```python
data.printSchema() # Checking if the Region column is removed
```

```
root
 |-- population: integer (nullable = true)
 |-- fertility: double (nullable = true)
 |-- HIV: double (nullable = true)
 |-- CO2: double (nullable = true)
 |-- BMI_male: double (nullable = true)
 |-- GDP: integer (nullable = true)
 |-- BMI_female: double (nullable = true)
 |-- life: double (nullable = true)
 |-- child_mortality: double (nullable = true)
```

3. Use a graph, explore and describe the relationship between 'fertility' feature and 'life' feature in the 2nd DataFrame

[8]:
```python
# Staistical Summary of both Fertility and Life
data.select('fertility', 'life').describe().show()
```

```
+-------+------------------+-----------------+
|summary|         fertility|             life|
+-------+------------------+-----------------+
|  count|               139|              139|
|   mean| 3.005107913669065|69.60287769784175|
| stddev|1.6153544802816209|9.122189401943691|
|    min|              1.28|             45.2|
|    max|              7.59|             82.6|
+-------+------------------+-----------------+
```

[9]:
```python
# Converting spark Dataframe to Python
import pandas as pd
data = data.toPandas()
```

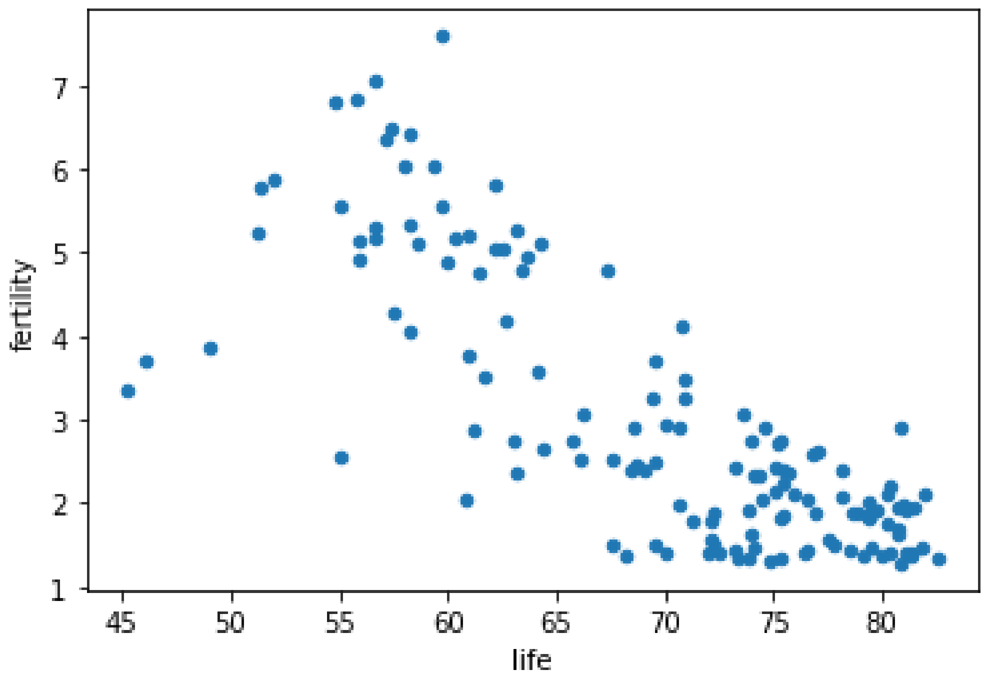
[10]:
```python
# Removing missing Data
data.shape
data.dropna().head(3)
```

[10]:
```
   population  fertility  HIV       CO2  BMI_male      GDP  BMI_female  life  \
1  34811059.0       2.73  0.1  3.328945  24.59620  12314.0    129.9049  75.3
3  19842251.0       6.43  2.0  1.474353  22.25083   7103.0    130.1247  58.3
5  40381860.0       2.24  0.5  4.785170  27.50170  14646.0    118.8915  75.5

   child_mortality
1             29.5
3            192.0
5             15.4
```

[11]:
```
#  Visualization of Data ( second dataframe)
data.plot.scatter(x='life',y='fertility')
```

[11]: <AxesSubplot:xlabel='life', ylabel='fertility'>



[12]:
```
data =  spark.read.option('header','true').csv('D:\\Datascience\\Region_info.
 ↪csv', inferSchema=True)
type(data)
```

[12]: pyspark.sql.dataframe.DataFrame

[13]:
```
# delete single column and creating a new dataframe named Data
data = data.drop('Region')
data.show(10)
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|population|fertility| HIV|        CO2|BMI_male|
GDP|BMI_female|life|child_mortality|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|      null|     null|null|       null|    null| null|      null|null|
null|
|  34811059|     2.73| 0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  19842251|     6.43| 2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  40381860|     2.24| 0.5|4.785169983| 27.5017|14646|  118.8915|75.5|
15.4|
|      null|     null|null|       null|    null| null|      null|null|
null|
|   2975029|      1.4| 0.1|1.804106217|25.35542| 7383|  132.8108|72.5|
20.0|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  21370348|     1.96| 0.1|18.01631327|27.56373|41312|  117.3755|81.5|
5.2|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
only showing top 10 rows
```

[14]:
```python
data.na.drop(how='any')
data.show(5)
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|population|fertility| HIV|        CO2|BMI_male|
GDP|BMI_female|life|child_mortality|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|      null|     null|null|       null|    null| null|      null|null|
null|
|  34811059|     2.73| 0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  19842251|     6.43| 2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|
```

```
|      null|     null|null|       null|    null| null|      null|null|
null|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
only showing top 5 rows
```

[ ]:

4. Use Spark SQL query to display the 'fertility' and 'life' columns in the 2nd DataFrame where 'fertility' is great than 1.0 and 'life' is greater than 70.

[15]:

```
data.createOrReplaceTempView("Table")
spark.sql("SELECT fertility,life from Table where fertility>0 AND life>70")#␣
↪collect()
data.show(10)
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|population|fertility| HIV|        CO2|BMI_male|
GDP|BMI_female|life|child_mortality|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|      null|     null|null|       null|    null| null|      null|null|
null|
|  34811059|     2.73| 0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  19842251|     6.43| 2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  40381860|     2.24| 0.5|4.785169983| 27.5017|14646|  118.8915|75.5|
15.4|
|      null|     null|null|       null|    null| null|      null|null|
null|
|   2975029|      1.4| 0.1|1.804106217|25.35542| 7383|  132.8108|72.5|
20.0|
|      null|     null|null|       null|    null| null|      null|null|
null|
|  21370348|     1.96| 0.1|18.01631327|27.56373|41312|  117.3755|81.5|
5.2|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
only showing top 10 rows
```

5. Build a linear regression model to predict life expectancy (the 'life' column) in the 2nd

DataFrame using the 'fertility' column as the predictor. Conduct performance evaluation for the model and make conclusions.

[16]:
```
data.printSchema()
```

```
root
 |-- population: integer (nullable = true)
 |-- fertility: double (nullable = true)
 |-- HIV: double (nullable = true)
 |-- CO2: double (nullable = true)
 |-- BMI_male: double (nullable = true)
 |-- GDP: integer (nullable = true)
 |-- BMI_female: double (nullable = true)
 |-- life: double (nullable = true)
 |-- child_mortality: double (nullable = true)
```

[17]:
```
data.columns
```

[17]:
```
['population',
 'fertility',
 'HIV',
 'CO2',
 'BMI_male',
 'GDP',
 'BMI_female',
 'life',
 'child_mortality']
```

[18]:
```
data
```

[18]:
```
DataFrame[population: int, fertility: double, HIV: double, CO2: double,
BMI_male: double, GDP: int, BMI_female: double, life: double, child_mortality:
double]
```

[19]:
```
# Remmoving missing values and droping the non-numeric column Region.
data.na.drop().show(5)
```

```
+----------+---------+---+-----------+--------+-----+----------+----+-----------
----+
|population|fertility|HIV|        CO2|BMI_male|
GDP|BMI_female|life|child_mortality|
+----------+---------+---+-----------+--------+-----+----------+----+-----------
----+
|  34811059|     2.73|0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|
|  19842251|     6.43|2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|
|  40381860|     2.24|0.5|4.785169983| 27.5017|14646|  118.8915|75.5|
```

```
15.4|
|   2975029|      1.4|0.1|1.804106217|25.35542| 7383|  132.8108|72.5|
20.0|
|  21370348|     1.96|0.1|18.01631327|27.56373|41312|  117.3755|81.5|
5.2|
+----------+---------+---+-----------+--------+-----+----------+----+-----------
----+
only showing top 5 rows
```

[20]:
```
# counting the number of null Values.
for col in data.columns:
    print(col.ljust(20), data.filter(data[col].isNull()).count())
```

```
population           139
fertility            139
HIV                  139
CO2                  139
BMI_male             139
GDP                  139
BMI_female           139
life                 139
child_mortality      139
```

[21]:
```
data.select('fertility','life').summary('mean', '50%', 'max').show()
```

```
+-------+-----------------+-----------------+
|summary|        fertility|             life|
+-------+-----------------+-----------------+
|   mean|3.005107913669065|69.60287769784175|
|    50%|             2.41|             72.0|
|    max|             7.59|             82.6|
+-------+-----------------+-----------------+
```

[22]:
```
# filling the missing Value with 50% life value
data = data.na.fill(72.0)
```

[23]:
```
# counting the number of null Values.
for col in data.columns:
    print(col.ljust(20), data.filter(data[col].isNull()).count())
```

```
population           0
fertility            0
HIV                  0
CO2                  0
BMI_male             0
GDP                  0
BMI_female           0
```

```
life                 0
child_mortality      0
```

[24]:
```python
# Invoking VectorAssembler for grouping the required features
from pyspark.ml.feature import VectorAssembler
```

[25]:
```python
featureassembler=VectorAssembler(inputCols=['fertility'],
                                 outputCol='feature')
```

[26]:
```python
# transform the element of the input and Independant feature column
output=featureassembler.transform(data)
```

[27]:
```python
output.columns
```

[27]:
```
['population',
 'fertility',
 'HIV',
 'CO2',
 'BMI_male',
 'GDP',
 'BMI_female',
 'life',
 'child_mortality',
 'feature']
```

[28]:
```python
model_output= output.select("feature", "life")
```

[29]:
```python
model_output.show(5)
```

```
+-------+----+
|feature|life|
+-------+----+
| [72.0]|72.0|
| [2.73]|75.3|
| [72.0]|72.0|
| [6.43]|58.3|
| [72.0]|72.0|
+-------+----+
only showing top 5 rows
```

Model Training using Linear Regression

[30]:
```python
from pyspark.ml.regression import LinearRegression
#train_test_split
#featuresCol will be the input column and labelCol will be the target column
train_data, test_data= model_output.randomSplit([0.8, 0.2])
lr=LinearRegression(featuresCol='feature', labelCol='life')
lr=lr.fit(train_data)
```

Perfomance Evaluation

[31]:
```
# Getting the coefficients
lr.coefficients
```

[31]:
```
DenseVector([0.0276])
```

[32]:
```
# getting intercepts
lr.intercept
```

[32]:
```
69.84713458608961
```

Prediction with linear model

[33]:
```
pred=lr.evaluate(test_data)
```

[34]:
```
pred.meanAbsoluteError, pred.meanSquaredError
```

[34]:
```
(4.126167160095794, 45.68901889022429)
```

[35]:
```
pred.predictions.show(5)
```

```
+-------+----+-----------------+
|feature|life|       prediction|
+-------+----+-----------------+
| [1.33]|75.3|69.88387569244462|
| [1.37]|80.0|69.88498068812447|
| [1.38]|68.2|69.88525693704443|
|  [1.4]|72.5|69.88580943488435|
| [1.41]|80.4|69.88608568380432|
+-------+----+-----------------+
only showing top 5 rows
```

6. Build a Lasso regression model to predict life expectancy (the 'life column) in the 2nd DataFrame using all other columns as the predictor. Conduct performance evaluation for the model and make conclusions.

Lasso Regression model

[36]:
```
logr =  spark.read.option('header','true').csv('D:\\Datascience\\Region_info.
 ↪csv', inferSchema=True)
type(logr)
```

[36]:
```
pyspark.sql.dataframe.DataFrame
```

[37]:
```
logr.show(5)
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+--------------------+
|population|fertility| HIV|        CO2|BMI_male|
```

```
GDP|BMI_female|life|child_mortality|              Region|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+--------------------+
|      null|     null|null|       null|    null| null|      null|null|
null|                null|
|  34811059|     2.73| 0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|Middle East & Nor…|
|      null|     null|null|       null|    null| null|      null|null|
null|                null|
|  19842251|     6.43| 2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|  Sub-Saharan Africa|
|      null|     null|null|       null|    null| null|      null|null|
null|                null|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+--------------------+
only showing top 5 rows
```

[38]:

```python
# delete missing values in the rows and making a new dataframe
lasso_data = logr.na.drop().drop('Region')
lasso_data.show(10)
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|population|fertility| HIV|        CO2|BMI_male|
GDP|BMI_female|life|child_mortality|
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
|  34811059|     2.73| 0.1|3.328944661| 24.5962|12314|  129.9049|75.3|
29.5|
|  19842251|     6.43| 2.0|1.474353388|22.25083| 7103|  130.1247|58.3|
192.0|
|  40381860|     2.24| 0.5|4.785169983| 27.5017|14646|  118.8915|75.5|
15.4|
|   2975029|      1.4| 0.1|1.804106217|25.35542| 7383|  132.8108|72.5|
20.0|
|  21370348|     1.96| 0.1|18.01631327|27.56373|41312|  117.3755|81.5|
5.2|
|   8331465|     1.41| 0.3|8.183160018|26.46741|43952|  124.1394|80.4|
4.6|
|   8868713|     1.99| 0.1|5.109538292|25.65117|14365|  128.6024|70.6|
43.3|
|    348587|     1.89| 3.1|3.131921321|27.24594|24373|  124.3862|72.2|
14.5|
| 148252473|     2.38|0.06|0.319161002|20.39742| 2265|  125.0307|68.4|
55.9|
|    277315|     1.83| 1.3|6.008278835|26.38439|16075|   126.394|75.3|
15.4|
```

```
+----------+---------+----+-----------+--------+-----+----------+----+----------
-----+
only showing top 10 rows
```

[39]:
```python
lasso_data.dtypes
```

[39]:
```
[('population', 'int'),
 ('fertility', 'double'),
 ('HIV', 'double'),
 ('CO2', 'double'),
 ('BMI_male', 'double'),
 ('GDP', 'int'),
 ('BMI_female', 'double'),
 ('life', 'double'),
 ('child_mortality', 'double')]
```

[40]:
```python
# Checking for Missing Values
for col in lasso_data.columns:
    print(col.ljust(20), lasso_data.filter(lasso_data[col].isNull()).count())
```

```
population          0
fertility           0
HIV                 0
CO2                 0
BMI_male            0
GDP                 0
BMI_female          0
life                0
child_mortality     0
```

Lasso prediction Model Building

[41]:
```python
# StringIndexer: Converts string categories to numerical categories.
# Vector Assembler: Special to Spark API. We will find detail shortly.
# Logistic regression based on Lasso regularization.
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import numpy as np
import pyspark.sql.functions as F
from pyspark.ml.feature import OneHotEncoder, StringIndexer, StandardScaler
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.types import FloatType
```

[42]:
```
#VectorAssembler is a transformer that combines a given list of columns into a␣
↪single vector column.
#It is useful for combining raw features and features generated by different␣
↪feature transformers
from pyspark.ml.feature import VectorAssembler
predictorsassembler =␣
↪VectorAssembler(inputCols=['population','fertility','HIV','CO2','BMI_male','GDP','BMI_femal
                                                    'child_mortality'],
                                    outputCol='predictors')
```

[43]:
```
# The .select() is a transformation function that is used to select the columns␣
↪from DataFrame and Dataset
lasso_output = predictorsassembler.transform(lasso_data).
↪select('predictors','life')
```

[44]:
```
lasso_output.show(10)
```

```
+--------------------+----+
|          predictors|life|
+--------------------+----+
|[3.4811059E7,2.73...|75.3|
|[1.9842251E7,6.43...|58.3|
|[4.038186E7,2.24,...|75.5|
|[2975029.0,1.4,0....|72.5|
|[2.1370348E7,1.96...|81.5|
|[8331465.0,1.41,0...|80.4|
|[8868713.0,1.99,0...|70.6|
|[348587.0,1.89,3....|72.2|
|[1.48252473E8,2.3...|68.4|
|[277315.0,1.83,1....|75.3|
+--------------------+----+
only showing top 10 rows
```

[45]:
```
lasso_output.columns
```

[45]: ['predictors', 'life']

[46]:
```
# The .select() is a transformation function  used to select the columns from␣
↪DataFrame and Dataset
```

model training

[47]:
```
# Now we split the training data into the train and test part(0.8, 0.2␣
↪respectively)
train_lasso, test_lasso = lasso_output.randomSplit([0.8, 0.2],seed=42)
```

[48]:
```
train_lasso.show(4, truncate=False) # displaying train data
```

```
+-----------------------------------------------------------------+----+
|predictors                                                       |life|
+-----------------------------------------------------------------+----+
|[277315.0,1.83,1.3,6.008278835,26.38439,16075.0,126.394,15.4]   |75.3|
|[306165.0,2.91,2.4,1.36012592,27.02255,8293.0,120.9224,20.1]    |70.7|
|[321026.0,2.38,0.06,3.277725768,23.21991,12029.0,123.3223,16.0]|78.2|
|[348587.0,1.89,3.1,3.131921321,27.24594,24373.0,124.3862,14.5]  |72.2|
+-----------------------------------------------------------------+----+
only showing top 4 rows
```

[49]:

```
test_lasso.show(4, truncate=False)# dislaying test data
```

```
+----------------------------------------------------------------+----+
|predictors                                                      |life|
+----------------------------------------------------------------+----+
|[310033.0,2.12,0.3,6.821903051,27.20687,42294.0,118.7381,2.7]  |82.0|
|[485079.0,1.63,0.3,22.16807969,27.43404,95001.0,122.3705,2.8]  |80.7|
|[665414.0,5.05,0.06,0.178853064,22.06131,1440.0,132.1354,91.2]|62.6|
|[843206.0,2.74,0.1,1.277779556,26.53078,7129.0,127.4768,24.0]  |65.7|
+----------------------------------------------------------------+----+
only showing top 4 rows
```

[50]:

```
# Linear Model and Evaluation Matrix
evaluator = MulticlassClassificationEvaluator(labelCol='life',␣
 ↪metricName='accuracy')
```

[51]:

```
lasso = LogisticRegression(labelCol='life')
```

[52]:

```
# lasso Regression
evaluator
```

[52]:

```
MulticlassClassificationEvaluator_fb37b604287b
```