In [1]:
```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import *

spark = SparkSession\
    .builder\
    .appName("chapter-29-ML-clustering")\
    .getOrCreate()

import os
SPARK_BOOK_DATA_PATH = os.environ['SPARK_BOOK_DATA_PATH']
```

In [2]:
```
spark
```

Out[2]:

SparkSession - hive

SparkContext

Spark UI (http://192.168.1.2:4040)

Version

v2.4.3

Master

local[*]

AppName

PySparkShell

In [3]:
```
from pyspark.ml.feature import VectorAssembler
va = VectorAssembler()\
  .setInputCols(["Quantity", "UnitPrice"])\
  .setOutputCol("features")

sales = va.transform(spark.read.format("csv")
  .option("header", "true")
  .option("inferSchema", "true")
  .load(SPARK_BOOK_DATA_PATH + "/data/retail-data/by-day/*.csv")
  .limit(50)
  .coalesce(1)
  .where("Description IS NOT NULL"))

sales.cache()
```

Out[3]:
```
DataFrame[InvoiceNo: string, StockCode: string, Description: string, Quantity: int, InvoiceDate: ti
mestamp, UnitPrice: double, CustomerID: double, Country: string, features: vector]
```

In [4]:
```
sales.is_cached
```

Out[4]:
```
True
```

In [5]:
```
# should match to limit(50) from spark.read.limit(50)
sales.count()  # to trigger cache
```

Out[5]:
```
50
```

In [7]:
```
sales.show(5, truncate=False)
```

```
+---------+---------+--------------------------------+--------+-------------------+---------+-------
---+--------------+-----------+
|InvoiceNo|StockCode|Description                     |Quantity|InvoiceDate        |UnitPrice|Custome
rID|Country       |features   |
+---------+---------+--------------------------------+--------+-------------------+---------+-------
---+--------------+-----------+
|580538   |23084    |RABBIT NIGHT LIGHT              |48      |2011-12-05 08:38:00|1.79     |14075.0
|United Kingdom|[48.0,1.79]|
|580538   |23077    |DOUGHNUT LIP GLOSS              |20      |2011-12-05 08:38:00|1.25     |14075.0
|United Kingdom|[20.0,1.25]|
|580538   |22906    |12 MESSAGE CARDS WITH ENVELOPES|24      |2011-12-05 08:38:00|1.65     |14075.0
|United Kingdom|[24.0,1.65]|
|580538   |21914    |BLUE HARMONICA IN BOX           |24      |2011-12-05 08:38:00|1.25     |14075.0
|United Kingdom|[24.0,1.25]|
|580538   |22467    |GUMBALL COAT RACK               |6       |2011-12-05 08:38:00|2.55     |14075.0
|United Kingdom|[6.0,2.55] |
+---------+---------+--------------------------------+--------+-------------------+---------+-------
---+--------------+-----------+
only showing top 5 rows
```

In [8]:
```
# COMMAND ----------

from pyspark.ml.clustering import KMeans
km = KMeans().setK(5)
print (km.explainParams())
kmModel = km.fit(sales)
```

```
distanceMeasure: the distance measure. Supported options: 'euclidean' and 'cosine'. (default: eucli
dean)
featuresCol: features column name. (default: features)
initMode: The initialization algorithm. This can be either "random" to choose random points as init
ial cluster centers, or "k-means||" to use a parallel variant of k-means++ (default: k-means||)
initSteps: The number of steps for k-means|| initialization mode. Must be > 0. (default: 2)
k: The number of clusters to create. Must be > 1. (default: 2, current: 5)
maxIter: max number of iterations (>= 0). (default: 20)
predictionCol: prediction column name. (default: prediction)
seed: random seed. (default: 7969353092125344463)
tol: the convergence tolerance for iterative algorithms (>= 0). (default: 0.0001)
```

In [9]:
```
# COMMAND ----------

summary = kmModel.summary
```

In [10]:
```
print (summary.clusterSizes) # number of points
```

```
[10, 8, 29, 2, 1]
```

In [11]:
```
kmModel.computeCost(sales)
centers = kmModel.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[23.2    0.956]
[ 2.5     11.24375]
[7.55172414 2.77172414]
[48.    1.32]
[36.    0.85]
```

In [12]:
```
# COMMAND ----------

from pyspark.ml.clustering import BisectingKMeans
bkm = BisectingKMeans().setK(5).setMaxIter(5)
bkmModel = bkm.fit(sales)


# COMMAND ----------

summary = bkmModel.summary
print (summary.clusterSizes) # number of points
```

```
[16, 8, 13, 10, 3]
```

In [13]:
```
kmModel.computeCost(sales)
centers = kmModel.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[23.2    0.956]
[ 2.5     11.24375]
[7.55172414 2.77172414]
[48.    1.32]
[36.    0.85]
```

In [14]:
```
# COMMAND ----------

from pyspark.ml.clustering import GaussianMixture
gmm = GaussianMixture().setK(5)
print (gmm.explainParams())
model = gmm.fit(sales)


# COMMAND ----------

summary = model.summary
print (model.weights)
model.gaussiansDF.show()
```

```
featuresCol: features column name. (default: features)
k: Number of independent Gaussians in the mixture model. Must be > 1. (default: 2, current: 5)
maxIter: max number of iterations (>= 0). (default: 100)
predictionCol: prediction column name. (default: prediction)
probabilityCol: Column name for predicted class conditional probabilities. Note: Not all models out
put well-calibrated probability estimates! These probabilities should be treated as confidences, no
t precise probabilities. (default: probability)
seed: random seed. (default: -7090211980209472397)
tol: the convergence tolerance for iterative algorithms (>= 0). (default: 0.01)
[0.16503937777770641, 0.35496420094056985, 0.06003637101912308, 0.1999636297743671, 0.2199964204882
3354]
+--------------------+--------------------+
|                mean|                 cov|
+--------------------+--------------------+
|[2.54180583818530...|0.785769315153778...|
|[5.07243095740621...|2.059950971034034...|
|[43.9877864408847...|32.22707068867282...|
|[23.1998836372414...|2.560279258630084...|
|[11.6364190345020...|1.322132750446848...|
+--------------------+--------------------+
```

In [16]:
```
summary.cluster.show(5)
```

```
+----------+
|prediction|
+----------+
|         2|
|         3|
|         3|
|         3|
|         1|
+----------+
only showing top 5 rows
```

In [17]:
```
summary.clusterSizes
```

Out[17]:
```
[8, 18, 3, 10, 11]
```

In [19]:
```
summary.probability.show(5)
```

```
+--------------------+
|         probability|
+--------------------+
|[1.37632400885157...|
|[4.89041912245635...|
|[1.67299627008735...|
|[7.43321003719004...|
|[1.46369160111044...|
+--------------------+
only showing top 5 rows
```

In [20]:
```
# COMMAND ----------

from pyspark.ml.feature import Tokenizer, CountVectorizer
tkn = Tokenizer().setInputCol("Description").setOutputCol("DescOut")
tokenized = tkn.transform(sales.drop("features"))
cv = CountVectorizer()\
  .setInputCol("DescOut")\
  .setOutputCol("features")\
  .setVocabSize(500)\
  .setMinTF(0)\
  .setMinDF(0)\
  .setBinary(True)
cvFitted = cv.fit(tokenized)
prepped = cvFitted.transform(tokenized)
```

In [21]:
```
# COMMAND ----------

from pyspark.ml.clustering import LDA
lda = LDA().setK(10).setMaxIter(5)
print (lda.explainParams())
model = lda.fit(prepped)
```

```
checkpointInterval: set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that t
he cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the check
point directory is not set in the SparkContext. (default: 10)
docConcentration: Concentration parameter (commonly named "alpha") for the prior placed on document
s' distributions over topics ("theta"). (undefined)
featuresCol: features column name. (default: features)
k: The number of topics (clusters) to infer. Must be > 1. (default: 10, current: 10)
keepLastCheckpoint: (For EM optimizer) If using checkpointing, this indicates whether to keep the l
ast checkpoint. If false, then the checkpoint will be deleted. Deleting the checkpoint can cause fa
ilures if a data partition is lost, so set this bit with care. (default: True)
learningDecay: Learning rate, set as anexponential decay rate. This should be between (0.5, 1.0] to
guarantee asymptotic convergence. (default: 0.51)
learningOffset: A (positive) learning parameter that downweights early iterations. Larger values ma
ke early iterations count less (default: 1024.0)
maxIter: max number of iterations (>= 0). (default: 20, current: 5)
optimizeDocConcentration: Indicates whether the docConcentration (Dirichlet parameter for document-
topic distribution) will be optimized during training. (default: True)
optimizer: Optimizer or inference algorithm used to estimate the LDA model.  Supported: online, em
(default: online)
seed: random seed. (default: 7673890338921026109)
subsamplingRate: Fraction of the corpus to be sampled and used in each iteration of mini-batch grad
ient descent, in range (0, 1]. (default: 0.05)
topicConcentration: Concentration parameter (commonly named "beta" or "eta") for the prior placed o
n topic' distributions over terms. (undefined)
topicDistributionCol: Output column with estimates of the topic mixture distribution for each docum
ent (often called "theta" in the literature). Returns a vector of zeros for an empty document. (def
ault: topicDistribution)
```

In [22]:
```
# COMMAND ----------

model.describeTopics(3).show()
```

```
+-----+-------------+--------------------+
|topic|  termIndices|         termWeights|
+-----+-------------+--------------------+
|    0|   [4, 6, 17]||[0.01166617595444...|
|    1|  [13, 9, 66]||[0.01091339619437...|
|    2|[15, 131, 45]||[0.00897001978399...|
|    3| [6, 125, 78]||[0.00902243209856...|
|    4|[103, 55, 62]||[0.00933169978592...|
|    5|  [11, 5, 23]||[0.01496249652116...|
|    6|   [7, 16, 2]||[0.01140631424514...|
|    7| [28, 73, 69]||[0.01535561351823...|
|    8|    [0, 3, 1]||[0.01802752067282...|
|    9| [46, 90, 30]||[0.01169076631125...|
+-----+-------------+--------------------+
```

In [23]:
```
cvFitted.vocabulary


# COMMAND ----------
```

Out[23]:
```
['water',
 'hot',
 'vintage',
 'bottle',
 'paperweight',
 '6',
 'home',
 'doormat',
 'landmark',
 'bicycle',
 'frame',
 'ribbons',
 '',
 'classic',
 'rose',
 'kit',
 'leaf',
 'sweet',
 'bag',
```

In [ ]: