In [54]:
```
import findspark
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql.types import StructField,StructType,IntegerType,StringType,TimestampType,
from pyspark.ml.feature import VectorAssembler,VectorIndexer,StringIndexer,OneHotEncoder,I
from pyspark.ml.stat import Correlation
from pyspark.ml import Pipeline
from pyspark.sql import functions as F
from pyspark.ml.evaluation import *
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.linalg import DenseVector,SparseVector
```

In [55]:
```
import pandas as pd
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', -1)
```

```
d:\projects\python\pyspark\env-win-v3.7.9\lib\site-packages\ipykernel_launcher.py:4: Futur
eWarning: Passing a negative integer is deprecated in version 1.0 and will not be supporte
d in future version. Instead, use None to not limit the column width.
  after removing the cwd from sys.path.
```

In [56]:
```
findspark.init()
```

In [59]:
```
try: spark.stop()
except: pass
```

In [58]:
```
# By default 12 executors based on CPU core if not specified
spark=SparkSession.builder.appName("SparkML").master("local[4]").getOrCreate()
sc=spark.sparkContext
```

Read DataFrame

In [6]:
```
rdd1=sc.textFile("data/iris/iris.data")
rdd1=rdd1.map(lambda x: x.split(","))
rdd1=rdd1.map(lambda x: Row(sepal_l=float(x[0]), sepal_w=float(x[1]),petal_l=float(x[2]),p
rdd1.collect()
```

Out[6]:
```
[Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=5.1, sepal_w=3.5),
 Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=4.9, sepal_w=3.0),
 Row(cls='Iris-setosa', petal_l=1.3, petal_w=0.2, sepal_l=4.7, sepal_w=3.2),
 Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.2, sepal_l=4.6, sepal_w=3.1),
 Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=5.0, sepal_w=3.6),
 Row(cls='Iris-setosa', petal_l=1.7, petal_w=0.4, sepal_l=5.4, sepal_w=3.9),
 Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.3, sepal_l=4.6, sepal_w=3.4),
 Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.2, sepal_l=5.0, sepal_w=3.4),
 Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=4.4, sepal_w=2.9),
 Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.1, sepal_l=4.9, sepal_w=3.1),
 Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.2, sepal_l=5.4, sepal_w=3.7),
 Row(cls='Iris-setosa', petal_l=1.6, petal_w=0.2, sepal_l=4.8, sepal_w=3.4),
 Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.1, sepal_l=4.8, sepal_w=3.0),
 Row(cls='Iris-setosa', petal_l=1.1, petal_w=0.1, sepal_l=4.3, sepal_w=3.0),
 Row(cls='Iris-setosa', petal_l=1.2, petal_w=0.2, sepal_l=5.8, sepal_w=4.0),
 Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.4, sepal_l=5.7, sepal_w=4.4),
 Row(cls='Iris-setosa', petal_l=1.3, petal_w=0.4, sepal_l=5.4, sepal_w=3.9),
```

```
Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.3, sepal_l=5.1, sepal_w=3.5),
Row(cls='Iris-setosa', petal_l=1.7, petal_w=0.3, sepal_l=5.7, sepal_w=3.8),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.3, sepal_l=5.1, sepal_w=3.8),
Row(cls='Iris-setosa', petal_l=1.7, petal_w=0.2, sepal_l=5.4, sepal_w=3.4),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.4, sepal_l=5.1, sepal_w=3.7),
Row(cls='Iris-setosa', petal_l=1.0, petal_w=0.2, sepal_l=4.6, sepal_w=3.6),
Row(cls='Iris-setosa', petal_l=1.7, petal_w=0.5, sepal_l=5.1, sepal_w=3.3),
Row(cls='Iris-setosa', petal_l=1.9, petal_w=0.2, sepal_l=4.8, sepal_w=3.4),
Row(cls='Iris-setosa', petal_l=1.6, petal_w=0.2, sepal_l=5.0, sepal_w=3.0),
Row(cls='Iris-setosa', petal_l=1.6, petal_w=0.4, sepal_l=5.0, sepal_w=3.4),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.2, sepal_l=5.2, sepal_w=3.5),
Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=5.2, sepal_w=3.4),
Row(cls='Iris-setosa', petal_l=1.6, petal_w=0.2, sepal_l=4.7, sepal_w=3.2),
Row(cls='Iris-setosa', petal_l=1.6, petal_w=0.2, sepal_l=4.8, sepal_w=3.1),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.4, sepal_l=5.4, sepal_w=3.4),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.1, sepal_l=5.2, sepal_w=4.1),
Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=5.5, sepal_w=4.2),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.1, sepal_l=4.9, sepal_w=3.1),
Row(cls='Iris-setosa', petal_l=1.2, petal_w=0.2, sepal_l=5.0, sepal_w=3.2),
Row(cls='Iris-setosa', petal_l=1.3, petal_w=0.2, sepal_l=5.5, sepal_w=3.5),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.1, sepal_l=4.9, sepal_w=3.1),
Row(cls='Iris-setosa', petal_l=1.3, petal_w=0.2, sepal_l=4.4, sepal_w=3.0),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.2, sepal_l=5.1, sepal_w=3.4),
Row(cls='Iris-setosa', petal_l=1.3, petal_w=0.3, sepal_l=5.0, sepal_w=3.5),
Row(cls='Iris-setosa', petal_l=1.3, petal_w=0.3, sepal_l=4.5, sepal_w=2.3),
Row(cls='Iris-setosa', petal_l=1.3, petal_w=0.2, sepal_l=4.4, sepal_w=3.2),
Row(cls='Iris-setosa', petal_l=1.6, petal_w=0.6, sepal_l=5.0, sepal_w=3.5),
Row(cls='Iris-setosa', petal_l=1.9, petal_w=0.4, sepal_l=5.1, sepal_w=3.8),
Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.3, sepal_l=4.8, sepal_w=3.0),
Row(cls='Iris-setosa', petal_l=1.6, petal_w=0.2, sepal_l=5.1, sepal_w=3.8),
Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=4.6, sepal_w=3.2),
Row(cls='Iris-setosa', petal_l=1.5, petal_w=0.2, sepal_l=5.3, sepal_w=3.7),
Row(cls='Iris-setosa', petal_l=1.4, petal_w=0.2, sepal_l=5.0, sepal_w=3.3),
Row(cls='Iris-versicolor', petal_l=4.7, petal_w=1.4, sepal_l=7.0, sepal_w=3.2),
Row(cls='Iris-versicolor', petal_l=4.5, petal_w=1.5, sepal_l=6.4, sepal_w=3.2),
Row(cls='Iris-versicolor', petal_l=4.9, petal_w=1.5, sepal_l=6.9, sepal_w=3.1),
Row(cls='Iris-versicolor', petal_l=4.0, petal_w=1.3, sepal_l=5.5, sepal_w=2.3),
Row(cls='Iris-versicolor', petal_l=4.6, petal_w=1.5, sepal_l=6.5, sepal_w=2.8),
Row(cls='Iris-versicolor', petal_l=4.5, petal_w=1.3, sepal_l=5.7, sepal_w=2.8),
Row(cls='Iris-versicolor', petal_l=4.7, petal_w=1.6, sepal_l=6.3, sepal_w=3.3),
Row(cls='Iris-versicolor', petal_l=3.3, petal_w=1.0, sepal_l=4.9, sepal_w=2.4),
Row(cls='Iris-versicolor', petal_l=4.6, petal_w=1.3, sepal_l=6.6, sepal_w=2.9),
Row(cls='Iris-versicolor', petal_l=3.9, petal_w=1.4, sepal_l=5.2, sepal_w=2.7),
Row(cls='Iris-versicolor', petal_l=3.5, petal_w=1.0, sepal_l=5.0, sepal_w=2.0),
Row(cls='Iris-versicolor', petal_l=4.2, petal_w=1.5, sepal_l=5.9, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.0, petal_w=1.0, sepal_l=6.0, sepal_w=2.2),
Row(cls='Iris-versicolor', petal_l=4.7, petal_w=1.4, sepal_l=6.1, sepal_w=2.9),
Row(cls='Iris-versicolor', petal_l=3.6, petal_w=1.3, sepal_l=5.6, sepal_w=2.9),
Row(cls='Iris-versicolor', petal_l=4.4, petal_w=1.4, sepal_l=6.7, sepal_w=3.1),
Row(cls='Iris-versicolor', petal_l=4.5, petal_w=1.5, sepal_l=5.6, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.1, petal_w=1.0, sepal_l=5.8, sepal_w=2.7),
Row(cls='Iris-versicolor', petal_l=4.5, petal_w=1.5, sepal_l=6.2, sepal_w=2.2),
Row(cls='Iris-versicolor', petal_l=3.9, petal_w=1.1, sepal_l=5.6, sepal_w=2.5),
Row(cls='Iris-versicolor', petal_l=4.8, petal_w=1.8, sepal_l=5.9, sepal_w=3.2),
Row(cls='Iris-versicolor', petal_l=4.0, petal_w=1.3, sepal_l=6.1, sepal_w=2.8),
Row(cls='Iris-versicolor', petal_l=4.9, petal_w=1.5, sepal_l=6.3, sepal_w=2.5),
Row(cls='Iris-versicolor', petal_l=4.7, petal_w=1.2, sepal_l=6.1, sepal_w=2.8),
Row(cls='Iris-versicolor', petal_l=4.3, petal_w=1.3, sepal_l=6.4, sepal_w=2.9),
Row(cls='Iris-versicolor', petal_l=4.4, petal_w=1.4, sepal_l=6.6, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.8, petal_w=1.4, sepal_l=6.8, sepal_w=2.8),
Row(cls='Iris-versicolor', petal_l=5.0, petal_w=1.7, sepal_l=6.7, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.5, petal_w=1.5, sepal_l=6.0, sepal_w=2.9),
Row(cls='Iris-versicolor', petal_l=3.5, petal_w=1.0, sepal_l=5.7, sepal_w=2.6),
Row(cls='Iris-versicolor', petal_l=3.8, petal_w=1.1, sepal_l=5.5, sepal_w=2.4),
Row(cls='Iris-versicolor', petal_l=3.7, petal_w=1.0, sepal_l=5.5, sepal_w=2.4),
Row(cls='Iris-versicolor', petal_l=3.9, petal_w=1.2, sepal_l=5.8, sepal_w=2.7),
```

```
Row(cls='Iris-versicolor', petal_l=5.1, petal_w=1.6, sepal_l=6.0, sepal_w=2.7),
Row(cls='Iris-versicolor', petal_l=4.5, petal_w=1.5, sepal_l=5.4, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.5, petal_w=1.6, sepal_l=6.0, sepal_w=3.4),
Row(cls='Iris-versicolor', petal_l=4.7, petal_w=1.5, sepal_l=6.7, sepal_w=3.1),
Row(cls='Iris-versicolor', petal_l=4.4, petal_w=1.3, sepal_l=6.3, sepal_w=2.3),
Row(cls='Iris-versicolor', petal_l=4.1, petal_w=1.3, sepal_l=5.6, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.0, petal_w=1.3, sepal_l=5.5, sepal_w=2.5),
Row(cls='Iris-versicolor', petal_l=4.4, petal_w=1.2, sepal_l=5.5, sepal_w=2.6),
Row(cls='Iris-versicolor', petal_l=4.6, petal_w=1.4, sepal_l=6.1, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.0, petal_w=1.2, sepal_l=5.8, sepal_w=2.6),
Row(cls='Iris-versicolor', petal_l=3.3, petal_w=1.0, sepal_l=5.0, sepal_w=2.3),
Row(cls='Iris-versicolor', petal_l=4.2, petal_w=1.3, sepal_l=5.6, sepal_w=2.7),
Row(cls='Iris-versicolor', petal_l=4.2, petal_w=1.2, sepal_l=5.7, sepal_w=3.0),
Row(cls='Iris-versicolor', petal_l=4.2, petal_w=1.3, sepal_l=5.7, sepal_w=2.9),
Row(cls='Iris-versicolor', petal_l=4.3, petal_w=1.3, sepal_l=6.2, sepal_w=2.9),
Row(cls='Iris-versicolor', petal_l=3.0, petal_w=1.1, sepal_l=5.1, sepal_w=2.5),
Row(cls='Iris-versicolor', petal_l=4.1, petal_w=1.3, sepal_l=5.7, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=6.0, petal_w=2.5, sepal_l=6.3, sepal_w=3.3),
Row(cls='Iris-virginica', petal_l=5.1, petal_w=1.9, sepal_l=5.8, sepal_w=2.7),
Row(cls='Iris-virginica', petal_l=5.9, petal_w=2.1, sepal_l=7.1, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=5.6, petal_w=1.8, sepal_l=6.3, sepal_w=2.9),
Row(cls='Iris-virginica', petal_l=5.8, petal_w=2.2, sepal_l=6.5, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=6.6, petal_w=2.1, sepal_l=7.6, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=4.5, petal_w=1.7, sepal_l=4.9, sepal_w=2.5),
Row(cls='Iris-virginica', petal_l=6.3, petal_w=1.8, sepal_l=7.3, sepal_w=2.9),
Row(cls='Iris-virginica', petal_l=5.8, petal_w=1.8, sepal_l=6.7, sepal_w=2.5),
Row(cls='Iris-virginica', petal_l=6.1, petal_w=2.5, sepal_l=7.2, sepal_w=3.6),
Row(cls='Iris-virginica', petal_l=5.1, petal_w=2.0, sepal_l=6.5, sepal_w=3.2),
Row(cls='Iris-virginica', petal_l=5.3, petal_w=1.9, sepal_l=6.4, sepal_w=2.7),
Row(cls='Iris-virginica', petal_l=5.5, petal_w=2.1, sepal_l=6.8, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=5.0, petal_w=2.0, sepal_l=5.7, sepal_w=2.5),
Row(cls='Iris-virginica', petal_l=5.1, petal_w=2.4, sepal_l=5.8, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=5.3, petal_w=2.3, sepal_l=6.4, sepal_w=3.2),
Row(cls='Iris-virginica', petal_l=5.5, petal_w=1.8, sepal_l=6.5, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=6.7, petal_w=2.2, sepal_l=7.7, sepal_w=3.8),
Row(cls='Iris-virginica', petal_l=6.9, petal_w=2.3, sepal_l=7.7, sepal_w=2.6),
Row(cls='Iris-virginica', petal_l=5.0, petal_w=1.5, sepal_l=6.0, sepal_w=2.2),
Row(cls='Iris-virginica', petal_l=5.7, petal_w=2.3, sepal_l=6.9, sepal_w=3.2),
Row(cls='Iris-virginica', petal_l=4.9, petal_w=2.0, sepal_l=5.6, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=6.7, petal_w=2.0, sepal_l=7.7, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=4.9, petal_w=1.8, sepal_l=6.3, sepal_w=2.7),
Row(cls='Iris-virginica', petal_l=5.7, petal_w=2.1, sepal_l=6.7, sepal_w=3.3),
Row(cls='Iris-virginica', petal_l=6.0, petal_w=1.8, sepal_l=7.2, sepal_w=3.2),
Row(cls='Iris-virginica', petal_l=4.8, petal_w=1.8, sepal_l=6.2, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=4.9, petal_w=1.8, sepal_l=6.1, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=5.6, petal_w=2.1, sepal_l=6.4, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=5.8, petal_w=1.6, sepal_l=7.2, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=6.1, petal_w=1.9, sepal_l=7.4, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=6.4, petal_w=2.0, sepal_l=7.9, sepal_w=3.8),
Row(cls='Iris-virginica', petal_l=5.6, petal_w=2.2, sepal_l=6.4, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=5.1, petal_w=1.5, sepal_l=6.3, sepal_w=2.8),
Row(cls='Iris-virginica', petal_l=5.6, petal_w=1.4, sepal_l=6.1, sepal_w=2.6),
Row(cls='Iris-virginica', petal_l=6.1, petal_w=2.3, sepal_l=7.7, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=5.6, petal_w=2.4, sepal_l=6.3, sepal_w=3.4),
Row(cls='Iris-virginica', petal_l=5.5, petal_w=1.8, sepal_l=6.4, sepal_w=3.1),
Row(cls='Iris-virginica', petal_l=4.8, petal_w=1.8, sepal_l=6.0, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=5.4, petal_w=2.1, sepal_l=6.9, sepal_w=3.1),
Row(cls='Iris-virginica', petal_l=5.6, petal_w=2.4, sepal_l=6.7, sepal_w=3.1),
Row(cls='Iris-virginica', petal_l=5.1, petal_w=2.3, sepal_l=6.9, sepal_w=3.1),
Row(cls='Iris-virginica', petal_l=5.1, petal_w=1.9, sepal_l=5.8, sepal_w=2.7),
Row(cls='Iris-virginica', petal_l=5.9, petal_w=2.3, sepal_l=6.8, sepal_w=3.2),
Row(cls='Iris-virginica', petal_l=5.7, petal_w=2.5, sepal_l=6.7, sepal_w=3.3),
Row(cls='Iris-virginica', petal_l=5.2, petal_w=2.3, sepal_l=6.7, sepal_w=3.0),
Row(cls='Iris-virginica', petal_l=5.0, petal_w=1.9, sepal_l=6.3, sepal_w=2.5),
Row(cls='Iris-virginica', petal_l=5.2, petal_w=2.0, sepal_l=6.5, sepal_w=3.0),
```

```
Row(cls='Iris-virginica', petal_l=5.4, petal_w=2.3, sepal_l=6.2, sepal_w=3.4),
Row(cls='Iris-virginica', petal_l=5.1, petal_w=1.8, sepal_l=5.9, sepal_w=3.0)]
```

In [7]:

```
schema1=StructType([
    StructField("sepal_l",DoubleType(),False),
    StructField("sepal_w",DoubleType(),False),
    StructField("petal_l",DoubleType(),False),
    StructField("petal_w",DoubleType(),False),
    StructField("cls",StringType(),False),
])
df1=spark.createDataFrame(rdd1,schema1)
print(df1.printSchema())
print(df1.show(5))
```

```
root
 |-- sepal_l: double (nullable = false)
 |-- sepal_w: double (nullable = false)
 |-- petal_l: double (nullable = false)
 |-- petal_w: double (nullable = false)
 |-- cls: string (nullable = false)

None
+-------+-------+-------+-------+-----------+
|sepal_l|sepal_w|petal_l|petal_w|        cls|
+-------+-------+-------+-------+-----------+
|    5.1|    3.5|    1.4|    0.2|Iris-setosa|
|    4.9|    3.0|    1.4|    0.2|Iris-setosa|
|    4.7|    3.2|    1.3|    0.2|Iris-setosa|
|    4.6|    3.1|    1.5|    0.2|Iris-setosa|
|    5.0|    3.6|    1.4|    0.2|Iris-setosa|
+-------+-------+-------+-------+-----------+
only showing top 5 rows

None
```

In [8]:

```
import pandas as pd
pdf = pd.DataFrame({
        'x1': ['a','a','b','b', 'c', 'd'],
        'x2': ['apple', 'orange', 'orange','orange', 'peach', 'peach'],
        'x3': [1, 1, 2, 2, 2, 4],
        'x4': [2.4, 2.5, 3.5, 1.4, 2.1,1.5],
        'y1': [1, 0, 1, 0, 0, 1],
        'y2': ['yes', 'no', 'no', 'yes', 'yes', 'yes']
    })
df2 = spark.createDataFrame(pdf)
df2.show()
```

```
+---+------+---+---+---+---+
| x1|    x2| x3| x4| y1| y2|
+---+------+---+---+---+---+
|  a| apple|  1|2.4|  1|yes|
|  a|orange|  1|2.5|  0| no|
|  b|orange|  2|3.5|  1| no|
|  b|orange|  2|1.4|  0|yes|
|  c| peach|  2|2.1|  0|yes|
|  d| peach|  4|1.5|  1|yes|
+---+------+---+---+---+---+
```

Dense vector and sparse vector

A vector can be represented in dense and sparse formats. A dense vector is a regular vector that has each elements printed. A sparse vector use three components to represent a vector but with less memory.

In [9]:
```
dv = DenseVector([1.0,0.,0.,0.,4.5,0])
dv
```

Out[9]:
```
DenseVector([1.0, 0.0, 0.0, 0.0, 4.5, 0.0])
```

Three components of a sparse vector

- vector size
- indices of active elements
- values of active elements

In the above dense vector:

- vector size = 6
- indices of active elements = [0, 4]
- values of active elements = [1.0, 4.5]

We can use the SparseVector() function to create a sparse vector. The first argument is the vector size, the second argument is a dictionary. The keys are indices of active elements and the values are values of active elements.

In [10]:
```
sv = SparseVector(6, {0:1.0, 4:4.5})
sv
```

Out[10]:
```
SparseVector(6, {0: 1.0, 4: 4.5})
```

Convert sparse vector to dense vector

In [11]:
```
DenseVector(sv.toArray())
```

Out[11]:
```
DenseVector([1.0, 0.0, 0.0, 0.0, 4.5, 0.0])
```

Convert dense vector to sparse vector

In [12]:
```
active_elements_dict = {index: value for index, value in enumerate(dv) if value != 0}
print(active_elements_dict)
print(SparseVector(len(dv), active_elements_dict))
```

```
{0: 1.0, 4: 4.5}
(6,[0,4],[1.0,4.5])
```

VectorAssembler

Assemble feature columns into one single feacturesCol with VectorAssembler

In [13]:
```
assembler1 = VectorAssembler(inputCols = ["sepal_l", "sepal_w", "petal_l", "petal_w"], out
assembled1 = assembler1.transform(df1)
```

```
assembled1.show(3)
```

```
+-------+-------+-------+-------+-----------+-----------------+
|sepal_l|sepal_w|petal_l|petal_w|        cls|         features|
+-------+-------+-------+-------+-----------+-----------------+
|    5.1|    3.5|    1.4|    0.2|Iris-setosa|[5.1,3.5,1.4,0.2]|
|    4.9|    3.0|    1.4|    0.2|Iris-setosa|[4.9,3.0,1.4,0.2]|
|    4.7|    3.2|    1.3|    0.2|Iris-setosa|[4.7,3.2,1.3,0.2]|
+-------+-------+-------+-------+-----------+-----------------+
only showing top 3 rows
```

Split data into train and test set

In [15]:

```
assembled2=assembled1.drop("sepal_l","sepal_w","petal_l","petal_w")
(trainingData,testData)=assembled2.randomSplit([0.6,0.4])
trainingData.show(5)
testData.show(5)
```

```
+-----------+-----------------+
|        cls|         features|
+-----------+-----------------+
|Iris-setosa|[4.4,3.0,1.3,0.2]|
|Iris-setosa|[4.4,3.2,1.3,0.2]|
|Iris-setosa|[4.6,3.1,1.5,0.2]|
|Iris-setosa|[4.6,3.6,1.0,0.2]|
|Iris-setosa|[4.7,3.2,1.3,0.2]|
+-----------+-----------------+
only showing top 5 rows

+-----------+-----------------+
|        cls|         features|
+-----------+-----------------+
|Iris-setosa|[4.3,3.0,1.1,0.1]|
|Iris-setosa|[4.4,2.9,1.4,0.2]|
|Iris-setosa|[4.5,2.3,1.3,0.3]|
|Iris-setosa|[4.6,3.2,1.4,0.2]|
|Iris-setosa|[4.6,3.4,1.4,0.3]|
+-----------+-----------------+
only showing top 5 rows
```

In [16]:

```
assembled2.rdd.map(lambda x: x['features']).take(5)
```

Out[16]:

```
[DenseVector([5.1, 3.5, 1.4, 0.2]),
 DenseVector([4.9, 3.0, 1.4, 0.2]),
 DenseVector([4.7, 3.2, 1.3, 0.2]),
 DenseVector([4.6, 3.1, 1.5, 0.2]),
 DenseVector([5.0, 3.6, 1.4, 0.2])]
```

In [17]:

```
assembled2.rdd.map(lambda x: list(x['features'])).take(5)
```

Out[17]:

```
[[5.1, 3.5, 1.4, 0.2],
 [4.9, 3.0, 1.4, 0.2],
 [4.7, 3.2, 1.3, 0.2],
 [4.6, 3.1, 1.5, 0.2],
 [5.0, 3.6, 1.4, 0.2]]
```

In [18]:

```
dense_features_col_udf = F.udf(lambda x: x.toArray().tolist(), returnType=ArrayType(Double
assembled2.withColumn("SparseDenseToArray",dense_features_col_udf(assembled2.features)).pr
```

```
root
 |-- cls: string (nullable = false)
 |-- features: vector (nullable = true)
 |-- SparseDenseToArray: array (nullable = true)
 |    |-- element: double (containsNull = true)
```

Correlation

In [19]:
```
display(Correlation.corr(assembled1, "features").toPandas())
```

	pearson(features)
0	DenseMatrix([[1. , -0.10936925, 0.87175416, 0.81795363],\n [-0.10936925, 1. , -0.4205161 , -0.35654409],\n [0.87175416, -0.4205161 , 1. , 0.9627571],\n [0.81795363, -0.35654409, 0.9627571 , 1.]])

Binarizer

add new col and convert it to binary float form
if greater than threshold than 1.0, else 0.0

Bucketizer

add new col and convert it to float form
if [0, 2.1, 2.5, 3.5], then 0.0-2.0 is 0.0 2.1-2.4 is 1.0 2.5-3.5 is 2.0

StringIndexer

Converts categorical or other data types to float values

OneHotEncoder

Converts float values to float binary array for representation

IndexToString

Convert index value back to its original labels

Transform:

A feature transformer might take a DataFrame, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended. A learning model might take a DataFrame, read the column containing feature vectors, predict the label for each feature vector, and output a new DataFrame with predicted labels appended as a column.

In [20]:
```
all_stages2 = [StringIndexer(inputCol=c, outputCol='idx_' + c) for c in ['x1', 'x2', 'x3']
               [OneHotEncoder(inputCol='idx_' + c, outputCol='ohe_' + c) for c in ['x1', 'x2
              [Binarizer(threshold=2.1,inputCol="x4",outputCol="b_x4")] + \
              [Bucketizer(splits=[0, 2.1, 2.5, 3.5],inputCol="x4",outputCol="buck_x4")]
all_stages2.append(IndexToString(inputCol="idx_x2",outputCol="idx_string_x2",labels=["orar
all_stages2
```

```
[StringIndexer_c235889022aa,
```

Out[20]:
```
 StringIndexer_0b1402550aa7,
 StringIndexer_7e823f890c1f,
 OneHotEncoder_abb9c64f7773,
 OneHotEncoder_f6bd5e696a99,
 OneHotEncoder_90b65ea90797,
 Binarizer_88fa439f5847,
 Bucketizer_00ca78f559e6,
 IndexToString_5dddc9ef8668]
```

Pipeline

Pipeline is a sequence of stages which consists of Estimators and/or Transformers. Estimator has fit method and Transformer has transform method. Therefore, we can say, a pipeline is a sequence of fit and transform methods. When it is a fit method, it applies to the input data and turns into a transform method. Then the transform method applies to the fitted data and output transformed data. The transformed data output from previous stage has to be an acceptable input to the next stage's fit/transform method.

In [21]:
```
df2_pipe_fit = Pipeline(stages=all_stages2).fit(df2)
print(df2_pipe_fit.stages[1].labels)
{x._java_obj.getOutputCol(): x.labels for x in df2_pipe_fit.stages if isinstance(x, Strin
```

```
['orange', 'peach', 'apple']
```

Out[21]:
```
{'idx_x1': ['a', 'b', 'd', 'c'],
 'idx_x2': ['orange', 'peach', 'apple'],
 'idx_x3': ['2', '1', '4']}
```

In [22]:
```
all_stages2[-1].setLabels(df2_pipe_fit.stages[1].labels)
df2_pipe_tran=df2_pipe_fit.transform(df2)
display(df2_pipe_tran.toPandas())
```

	x1	x2	x3	x4	y1	y2	idx_x1	idx_x2	idx_x3	ohe_x1	ohe_x2	ohe_x3	b_x4	buck_x4	idx_string_x2
0	a	apple	1	2.4	1	yes	0.0	2.0	1.0	(1.0, 0.0, 0.0)	(0.0, 0.0)	(0.0, 1.0)	1.0	1.0	apple
1	a	orange	1	2.5	0	no	0.0	0.0	1.0	(1.0, 0.0, 0.0)	(1.0, 0.0)	(0.0, 1.0)	1.0	2.0	orange
2	b	orange	2	3.5	1	no	1.0	0.0	0.0	(0.0, 1.0, 0.0)	(1.0, 0.0)	(1.0, 0.0)	1.0	2.0	orange
3	b	orange	2	1.4	0	yes	1.0	0.0	0.0	(0.0, 1.0, 0.0)	(1.0, 0.0)	(1.0, 0.0)	0.0	0.0	orange
4	c	peach	2	2.1	0	yes	3.0	1.0	0.0	(0.0, 0.0, 0.0)	(0.0, 1.0)	(1.0, 0.0)	0.0	1.0	peach
5	d	peach	4	1.5	1	yes	2.0	1.0	2.0	(0.0, 0.0, 1.0)	(0.0, 1.0)	(0.0, 0.0)	0.0	0.0	peach

In [23]:
```
assembler2 = VectorAssembler(inputCols = ['ohe_x1', 'ohe_x2', 'ohe_x3', 'x4'], outputCol =
assembled2 = assembler2.transform(df2_pipe_tran)
assembled2.limit(3).toPandas()
```

Out[23]:

	x1	x2	x3	x4	y1	y2	idx_x1	idx_x2	idx_x3	ohe_x1	ohe_x2	ohe_x3	b_x4	buck_x4	idx_string_x2	fea

	x1	x2	x3	x4	y1	y2	idx_x1	idx_x2	idx_x3	ohe_x1	ohe_x2	ohe_x3	b_x4	buck_x4	idx_string_x2	fea
0	a	apple	1	2.4	1	yes	0.0	2.0	1.0	(1.0, 0.0, 0.0)	(0.0, 0.0)	(0.0, 1.0)	1.0	1.0	apple	(1. 0. 0. 1.
1	a	orange	1	2.5	0	no	0.0	0.0	1.0	(1.0, 0.0, 0.0)	(1.0, 0.0)	(0.0, 1.0)	1.0	2.0	orange	(1. 0. 0. 1.
2	b	orange	2	3.5	1	no	1.0	0.0	0.0	(0.0, 1.0, 0.0)	(1.0, 0.0)	(1.0, 0.0)	1.0	2.0	orange	(0. 0. 0. 0.

example

In [24]:

```
featureIndexer=VectorIndexer(inputCol="features",outputCol="indexedFeatures",maxCategories
string_to_idx=StringIndexer(inputCol="cls",outputCol="indexedCls")
lr=LogisticRegression(featuresCol="indexedFeatures",labelCol="indexedCls")
idx_to_string=IndexToString(inputCol="prediction",outputCol="predCls")
```

In [39]:

```
df_featureIndexer1_fit=featureIndexer.fit(assembled1)
df_featureIndexer1_tran=df_featureIndexer1_fit.transform(assembled1)
df_featureIndexer1_tran.show(5,True)

df_string_to_idx1_fit=string_to_idx.fit(df_featureIndexer1_tran)
df_string_to_idx1_tran=df_string_to_idx1_fit.transform(df_featureIndexer1_tran)
df_string_to_idx1_tran.show(5)

(trainingData1,testData1)=df_string_to_idx1_tran.randomSplit([0.6,0.4])

df_lr_fit=lr.fit(df_string_to_idx1_tran)
df_lr_trans=df_lr_fit.transform(df_string_to_idx1_tran)
df_lr_trans.show(5)

idx_to_string.setLabels(df_string_to_idx1_fit.labels)
df_idx_to_string_trans=idx_to_string.transform(df_lr_trans)
df_idx_to_string_trans.show(5)
```

```
+-------+-------+-------+-------+-----------+-----------------+-----------------+
|sepal_l|sepal_w|petal_l|petal_w|        cls|         features|  indexedFeatures|
+-------+-------+-------+-------+-----------+-----------------+-----------------+
|    5.1|    3.5|    1.4|    0.2|Iris-setosa|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|
|    4.9|    3.0|    1.4|    0.2|Iris-setosa|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|
|    4.7|    3.2|    1.3|    0.2|Iris-setosa|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|
|    4.6|    3.1|    1.5|    0.2|Iris-setosa|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|
|    5.0|    3.6|    1.4|    0.2|Iris-setosa|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|
+-------+-------+-------+-------+-----------+-----------------+-----------------+
only showing top 5 rows

+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+
|sepal_l|sepal_w|petal_l|petal_w|        cls|         features|  indexedFeatures|indexedCl
s|
+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+
|    5.1|    3.5|    1.4|    0.2|Iris-setosa|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|        2.
0|
```

```
|    4.9|    3.0|    1.4|    0.2|Iris-setosa|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|       2.
0|
|    4.7|    3.2|    1.3|    0.2|Iris-setosa|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|       2.
0|
|    4.6|    3.1|    1.5|    0.2|Iris-setosa|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|       2.
0|
|    5.0|    3.6|    1.4|    0.2|Iris-setosa|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|       2.
0|
+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+
only showing top 5 rows

+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+--------------------+--------------------+----------+
|sepal_l|sepal_w|petal_l|petal_w|        cls|         features|  indexedFeatures|indexedCl
s|       rawPrediction|         probability|prediction|
+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+--------------------+--------------------+----------+
|    5.1|    3.5|    1.4|    0.2|Iris-setosa|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|       2.
0|[-6.5893371790432...|[4.01427089598418...|       2.0|
|    4.9|    3.0|    1.4|    0.2|Iris-setosa|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|       2.
0|[0.71902195476911...|[6.17170588715625...|       2.0|
|    4.7|    3.2|    1.3|    0.2|Iris-setosa|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|       2.
0|[-3.9723225596865...|[7.98811414499879...|       2.0|
|    4.6|    3.1|    1.5|    0.2|Iris-setosa|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|       2.
0|[-2.6281648871468...|[7.41234709793209...|       2.0|
|    5.0|    3.6|    1.4|    0.2|Iris-setosa|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|       2.
0|[-8.8759964616956...|[2.76228783523617...|       2.0|
+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+--------------------+--------------------+----------+
only showing top 5 rows

+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+--------------------+--------------------+----------+-----------+
|sepal_l|sepal_w|petal_l|petal_w|        cls|         features|  indexedFeatures|indexedCl
s|       rawPrediction|         probability|prediction|    predCls|
+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+--------------------+--------------------+----------+-----------+
|    5.1|    3.5|    1.4|    0.2|Iris-setosa|[5.1,3.5,1.4,0.2]|[5.1,3.5,1.4,0.2]|       2.
0|[-6.5893371790432...|[4.01427089598418...|       2.0|Iris-setosa|
|    4.9|    3.0|    1.4|    0.2|Iris-setosa|[4.9,3.0,1.4,0.2]|[4.9,3.0,1.4,0.2]|       2.
0|[0.71902195476911...|[6.17170588715625...|       2.0|Iris-setosa|
|    4.7|    3.2|    1.3|    0.2|Iris-setosa|[4.7,3.2,1.3,0.2]|[4.7,3.2,1.3,0.2]|       2.
0|[-3.9723225596865...|[7.98811414499879...|       2.0|Iris-setosa|
|    4.6|    3.1|    1.5|    0.2|Iris-setosa|[4.6,3.1,1.5,0.2]|[4.6,3.1,1.5,0.2]|       2.
0|[-2.6281648871468...|[7.41234709793209...|       2.0|Iris-setosa|
|    5.0|    3.6|    1.4|    0.2|Iris-setosa|[5.0,3.6,1.4,0.2]|[5.0,3.6,1.4,0.2]|       2.
0|[-8.8759964616956...|[2.76228783523617...|       2.0|Iris-setosa|
+-------+-------+-------+-------+-----------+-----------------+-----------------+---------
-+--------------------+--------------------+----------+-----------+
only showing top 5 rows
```

In [26]:

```
pipeline1=Pipeline(stages=[string_to_idx,featureIndexer,lr,idx_to_string])
model1=pipeline1.fit(trainingData)
predictions1=model1.transform(testData)
predictions1.select("features","cls","predCls").show(5)
```

```
+-----------------+-----------+-----------+
|         features|        cls|    predCls|
+-----------------+-----------+-----------+
|[4.3,3.0,1.1,0.1]|Iris-setosa|Iris-setosa|
|[4.4,2.9,1.4,0.2]|Iris-setosa|Iris-setosa|
|[4.5,2.3,1.3,0.3]|Iris-setosa|Iris-setosa|
|[4.6,3.2,1.4,0.2]|Iris-setosa|Iris-setosa|
```

```
|[4.6,3.4,1.4,0.3]|Iris-setosa|Iris-setosa|
+-----------------+-----------+-----------+
only showing top 5 rows
```

In [27]:

```python
print(df_featureIndexer1_fit, df_featureIndexer1_tran)
print(df_string_to_idx1_fit, df_string_to_idx1_tran)
print(df_lr_fit, df_lr_trans)
print(df_idx_to_string_trans)
print(featureIndexer, string_to_idx, lr, idx_to_string)
print(pipeline1, model1, predictions1)
```

```
VectorIndexer_2e0659ca79f2 DataFrame[sepal_l: double, sepal_w: double, petal_l: double, pe
tal_w: double, cls: string, features: vector, indexedFeatures: vector]
StringIndexer_406ae13022f9 DataFrame[sepal_l: double, sepal_w: double, petal_l: double, pe
tal_w: double, cls: string, features: vector, indexedFeatures: vector, indexedCls: double]
LogisticRegressionModel: uid = LogisticRegression_e97f15a8f282, numClasses = 3, numFeature
s = 4 DataFrame[sepal_l: double, sepal_w: double, petal_l: double, petal_w: double, cls: s
tring, features: vector, indexedFeatures: vector, indexedCls: double, rawPrediction: vecto
r, probability: vector, prediction: double]
DataFrame[sepal_l: double, sepal_w: double, petal_l: double, petal_w: double, cls: string,
features: vector, indexedFeatures: vector, indexedCls: double, rawPrediction: vector, prob
ability: vector, prediction: double, predCls: string]
VectorIndexer_2e0659ca79f2 StringIndexer_406ae13022f9 LogisticRegression_e97f15a8f282 Inde
xToString_21ad3f83439b
Pipeline_73d495a10e2a PipelineModel_1486829d6db2 DataFrame[cls: string, features: vector,
indexedCls: double, indexedFeatures: vector, rawPrediction: vector, probability: vector, p
rediction: double, predCls: string]
```

Parameter grid

In [32]:

```python
from pyspark.ml.tuning import ParamGridBuilder
param_grid = ParamGridBuilder().\
    addGrid(lr.regParam, [0, 0.5, 1, 2]).\
    addGrid(lr.elasticNetParam, [0, 0.5, 1]).\
    build()
```

MulticlassClassificationEvaluator

In [30]:

```python
evaluator=MulticlassClassificationEvaluator(
    labelCol="indexedCls",
    predictionCol="prediction",metricName="accuracy"
)
accu=evaluator.evaluate(predictions1)
print("Test Error: %g, "%(1-accu))
```

```
Test Error: 0.103448,
```

In [31]:

```python
predictions1.filter(predictions1.predCls!=predictions1.cls).select("features","cls","pred(
```

```
+-----------------+---------------+---------------+
|         features|            cls|        predCls|
+-----------------+---------------+---------------+
|[5.9,3.2,4.8,1.8]|Iris-versicolor| Iris-virginica|
|[6.2,2.2,4.5,1.5]|Iris-versicolor| Iris-virginica|
|[6.0,2.7,5.1,1.6]|Iris-versicolor| Iris-virginica|
|[6.3,2.8,5.1,1.5]| Iris-virginica|Iris-versicolor|
|[7.2,3.0,5.8,1.6]| Iris-virginica|Iris-versicolor|
+-----------------+---------------+---------------+
```

```
only showing top 5 rows
```

Cross-validation model

In [37]:

```
from pyspark.ml.tuning import CrossValidator
cv = CrossValidator(estimator=lr, estimatorParamMaps=param_grid, evaluator=evaluator, numF

cv_model = cv.fit(df_string_to_idx1_tran)
```

Prediction on training data

In [46]:

```
pred_training_cv = cv_model.transform(trainingData1)
show_columns = ['features', 'cls', 'prediction', 'rawPrediction', 'probability']
pred_training_cv.select(show_columns).limit(5).toPandas()
```

Out[46]:

	features	cls	prediction	rawPrediction	probability
0	[4.4, 3.0, 1.3, 0.2]	Iris-setosa	2.0	[-2.3453877654173603, -59.957462945880195, 62.30285071129756]	[8.387388596409817e-29, 7.998718685188395e-54, 1.0]
1	[4.4, 3.2, 1.3, 0.2]	Iris-setosa	2.0	[-5.740152822308001, -64.68840685296995, 70.42855967527795]	[8.324314284215018e-34, 2.0866375828194127e-59, 1.0]
2	[4.6, 3.1, 1.5, 0.2]	Iris-setosa	2.0	[-2.62816488714682, -59.515490786020266, 62.14365567316711]	[7.412347097932093e-29, 1.4591637089202677e-53, 1.0]
3	[4.6, 3.2, 1.4, 0.2]	Iris-setosa	2.0	[-4.443573364742935, -62.94192356241943, 67.38549692716238]	[6.383092411382714e-32, 2.509114271731677e-57, 1.0]
4	[4.6, 3.4, 1.4, 0.3]	Iris-setosa	2.0	[-7.426301245892418, -65.43221101591053, 72.85851226180297]	[1.3575271153426762e-35, 8.731763857042433e-61, 1.0]

Prediction on test data

In [47]:

```
pred_test_cv = cv_model.transform(testData1)
pred_test_cv.select(show_columns).limit(5).toPandas()
```

Out[47]:

	features	cls	prediction	rawPrediction	probability
0	[4.3, 3.0, 1.1, 0.1]	Iris-setosa	2.0	[-3.5827535936672383, -64.6628022790355, 68.24555587270274]	[6.387950678996389e-32, 1.8994744486123638e-58, 1.0]
1	[4.4, 2.9, 1.4, 0.2]	Iris-setosa	2.0	[-0.5299792878212379, -56.53103016948103, 57.06100945730227]	[9.73983008783691e-26, 4.651617566717906e-50, 1.0]
2	[4.5, 2.3, 1.3, 0.3]	Iris-setosa	2.0	[10.537603863648133, -40.8157415836193, 30.278137719971184]	[2.671745430052335e-09, 1.331436689054821e-31, 0.9999999973282545]
3	[4.7, 3.2, 1.3, 0.2]	Iris-setosa	2.0	[-3.9723225596865994, -63.660123151425594, 67.6324457111122]	[7.988114144998791e-32, 9.557885600608829e-58, 1.0]
4	[4.8, 3.4, 1.6, 0.2]	Iris-setosa	2.0	[-6.4237330149176906, -64.86542335610437, 71.28915637102209]	[1.7771372371188298e-34, 7.392942674466309e-60, 1.0]

Intercept and coefficients of the regression model

In [51]:
```
print('Intercept: ' + str(cv_model.bestModel.interceptVector) + "\n"
      'coefficients: ' + str(cv_model.bestModel.coefficientMatrix))
```

```
Intercept: [20.289499212385532,-22.348602233153855,2.0591030207683243]
coefficients: DenseMatrix([[  5.89276754, -16.97382528,   1.18025949,   4.12037176],
             [  3.42761234, -23.65471954,  10.60960823,  22.40656454],
             [ -9.32037988,  40.62854482, -11.78986772, -26.52693629]])
```

Parameters from the best model

In [52]:
```
print('The best RegParam is: ', cv_model.bestModel._java_obj.getRegParam(), "\n",
      'The best ElasticNetParam is: cv_model.bestModel._java_obj.getElasticNetParam()')
```

```
The best RegParam is:  0.0
 The best ElasticNetParam is: cv_model.bestModel._java_obj.getElasticNetParam()
```

VectorIndexer

In [86]:
```
from pyspark.ml.linalg import Vectors
dfr = spark.createDataFrame([(Vectors.dense([-1.0, 0.0]),),
    (Vectors.dense([0.0, 1.0]),), (Vectors.dense([0.0, 2.0]),)], ["a"])
dfr.head()
```

Out[86]:
```
Row(a=DenseVector([-1.0, 0.0]))
```

In [87]:
```
indexer = VectorIndexer(maxCategories=2, inputCol="a")
indexer.setOutputCol("indexed")
model2 = indexer.fit(dfr)
indexer.getHandleInvalid()
model2.transform(dfr).head()
```

Out[87]:
```
Row(a=DenseVector([-1.0, 0.0]), indexed=DenseVector([1.0, 0.0]))
```

In [ ]: