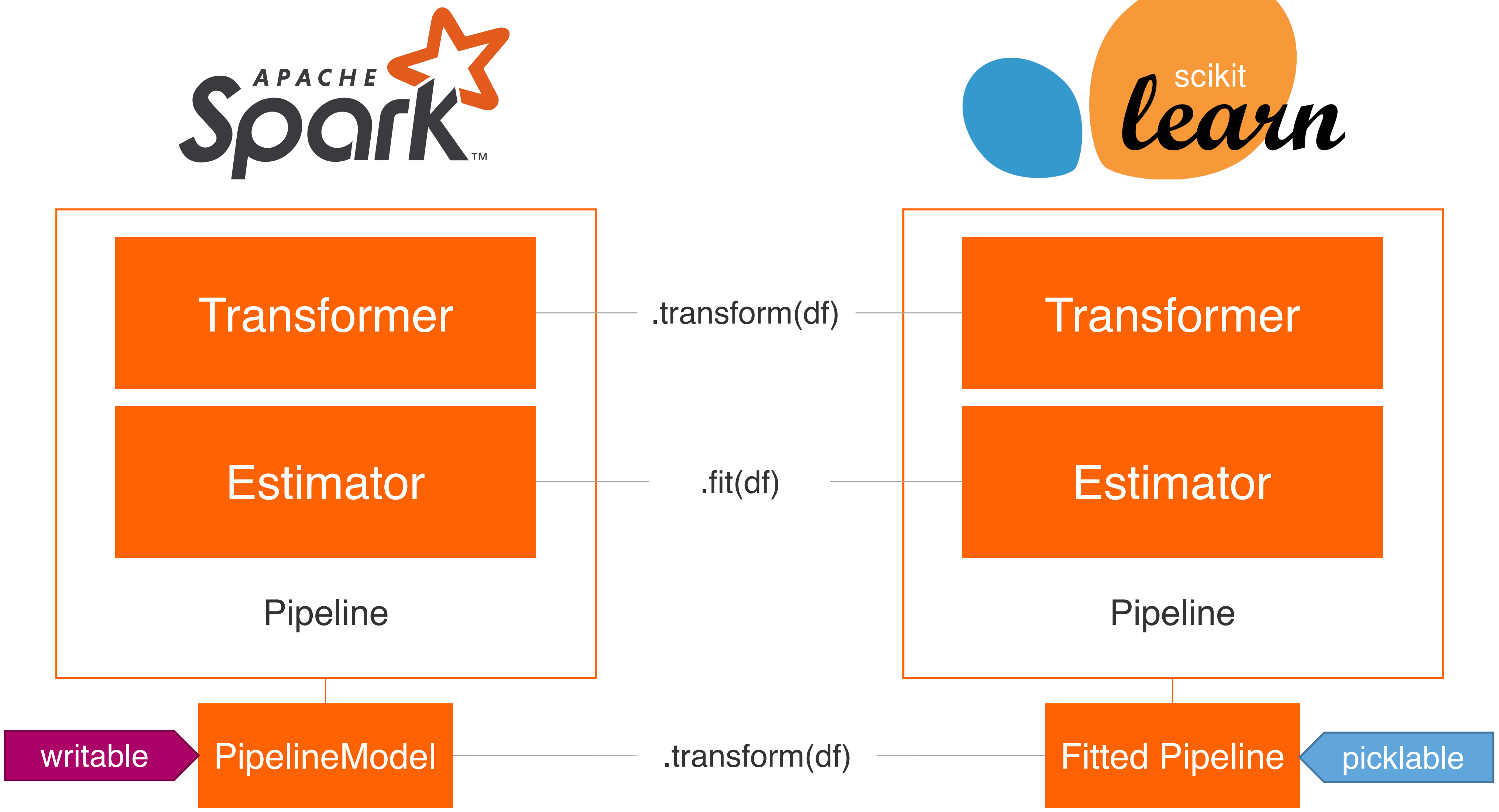
Spark MLLib

Tao Ruangyam, ING Analytics - Frankfurt Hub

Istanbul, 2020

Inspired by Scikit-Learn



Apache Spark

Transformer

Estimator

Pipeline

writable

PipelineModel

scikit learn

Transformer

Estimator

Pipeline

Fitted Pipeline

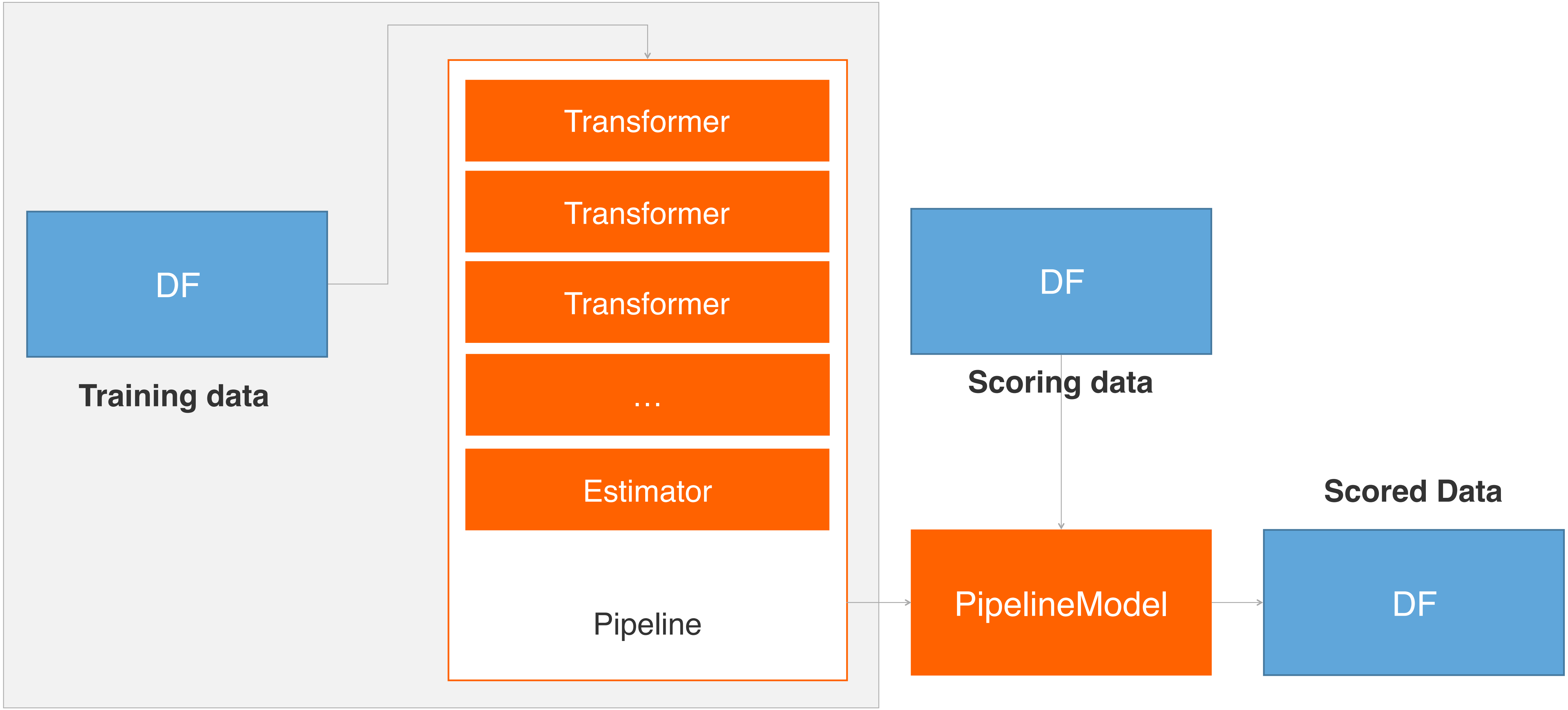
picklable

.transform(df)

.fit(df)

.transform(df)

Machine Learning in Spark



DF

Training data

Transformer

Transformer

Transformer

…

Estimator

Pipeline

DF

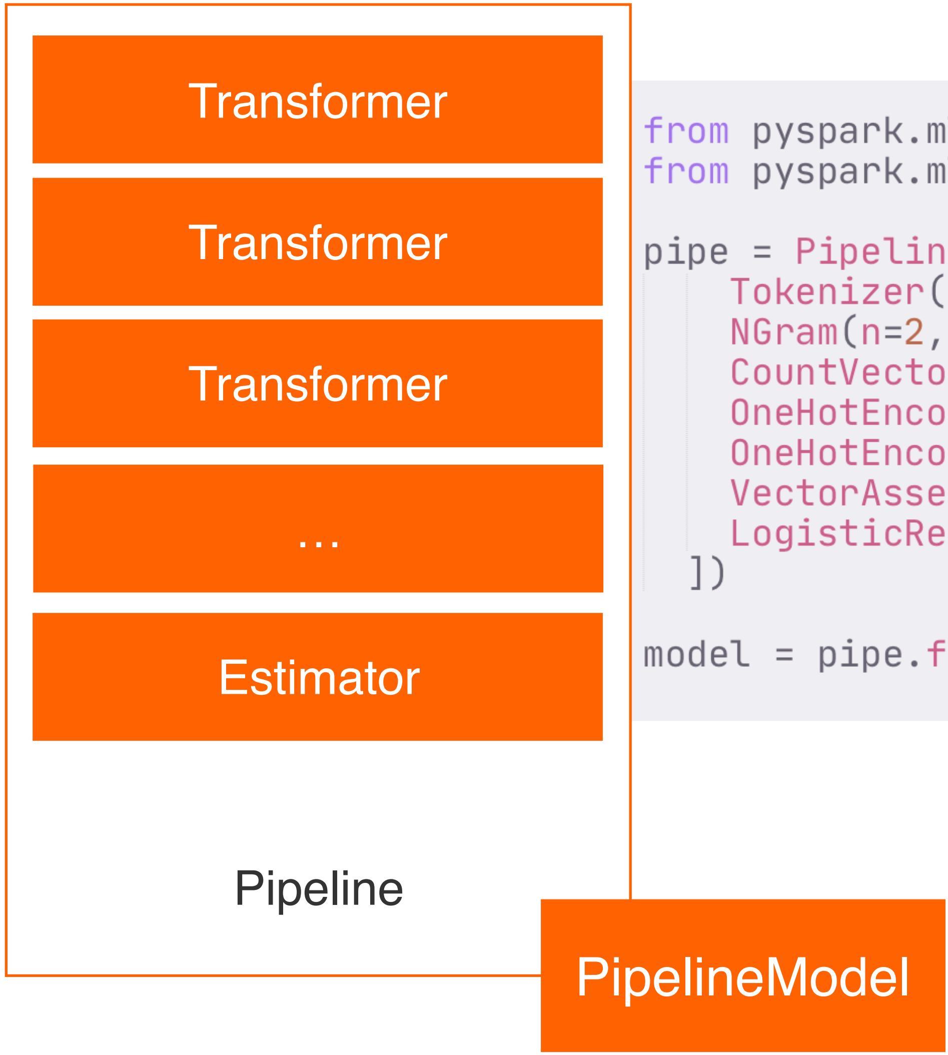
Scoring data

PipelineModel

Scored Data

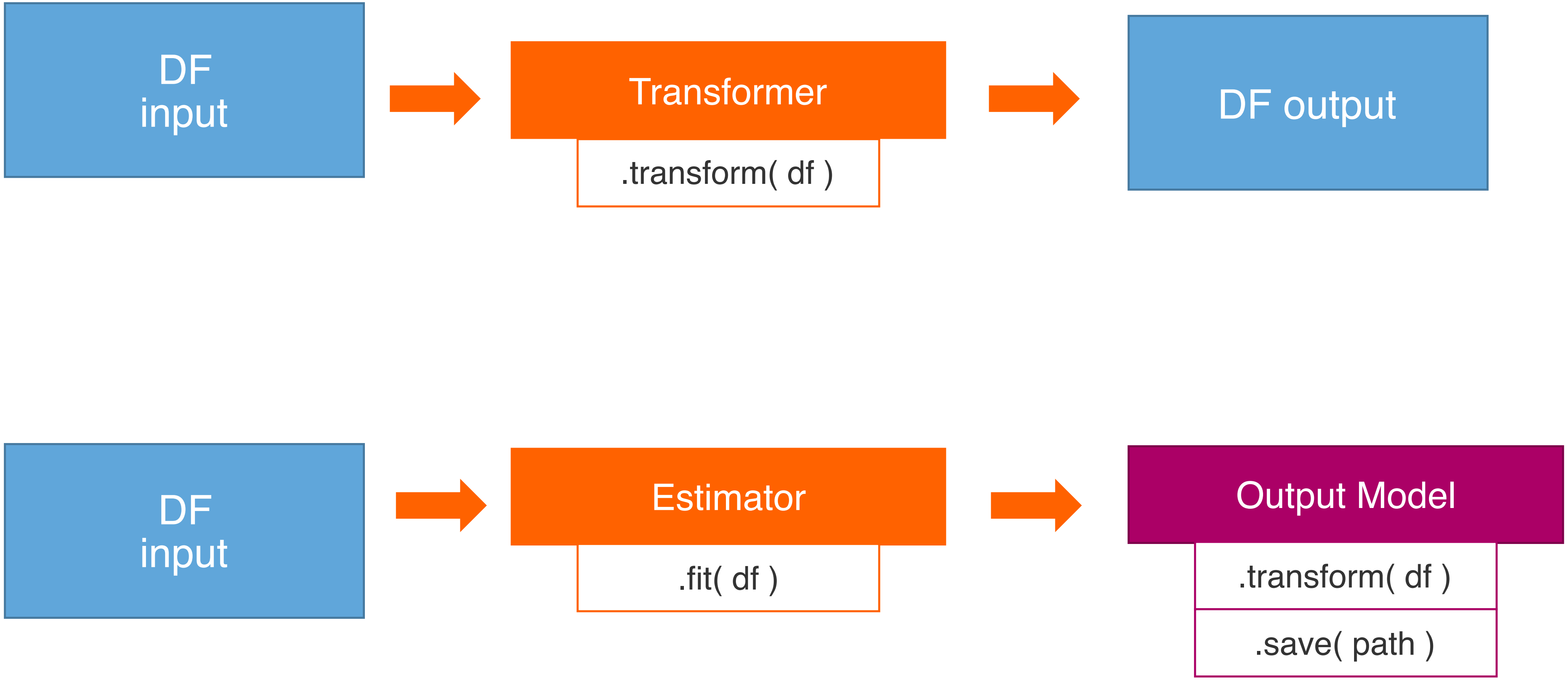
DF

Machine Learning in Spark



```
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder

pipe = Pipeline(stages=[
    Tokenizer(inputCol="desc", outputCol="words"),
    NGram(n=2, inputCol="words", outputCol="ngrams"),
    CountVectorizer(inputCol="ngrams", outputCol="ngrams"),
    OneHotEncoder(inputCol="type", outputCol="type-num"),
    OneHotEncoder(inputCol="manufacturer", outputCol="mf-num"),
    VectorAssembler(inputCols=["ngrams","type-num","mf-num"], outputCol="features"),
    LogisticRegression(featuresCol="features", labelCol="sold")
  ])

model = pipe.fit(train_df)
```

Transformer vs Estimator



Some interesting feature transformers

pyspark.ml.feature	Arguments	Output Column Type
Bucketizer	**splits**=[0, 10, 20, float("inf")]	Double
CountVectorizer	**minTF**=0.0, **maxTF**=1.0, …	Vector of Double
Imputer	**strategy**="mean"	Double
Normalizer	**p**=2.0	Vector of Double
OneHotEncoder		Double
PCA	**k**=3	Vector of Double
….		

Some interesting estimators

pyspark.ml.classification

- LinearSVC
- LogisticRegression
- DecisionTreeClassifier
- RandomForestClassifier
- GBTClassifier
- …

pyspark.ml.clustering

- KMeans
- GaussianMixture
- LDA
- …

Absent approaches

- DBScan
- Xmeans
- Catboost
- Etc.

Typical Estimator

pyspark.ml.classification

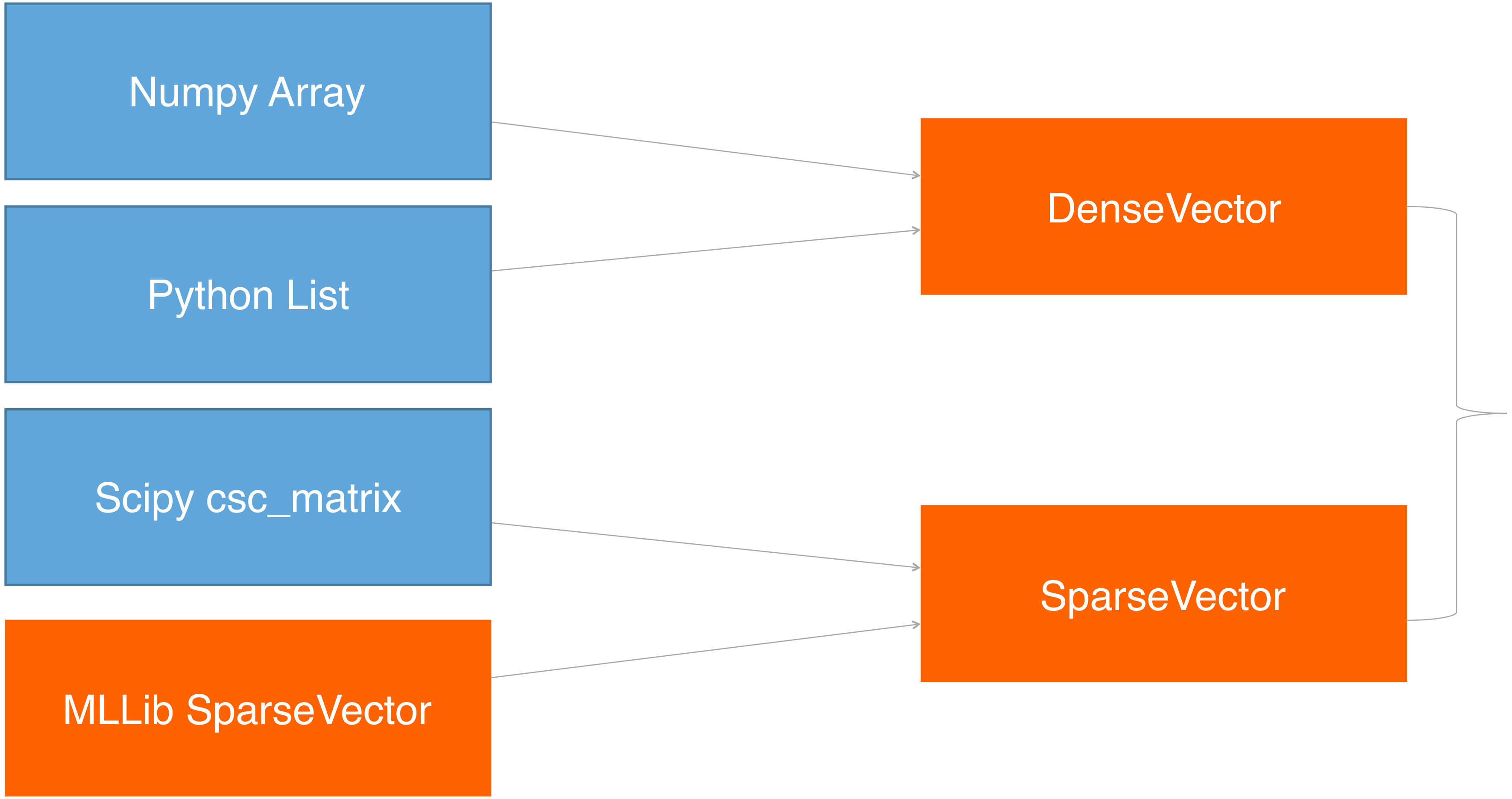
```
LinearSVC( featuresCol="foo",
           labelCol="label",
           predictionCol="target",
           …)
```

pyspark.ml.clustering

```
KMeans( featuresCol="foo",
        predictionCol="target",
        … )
```

Other hyperparameters

Compatible Column Types (as Vector)



Input to any **Transformer** which takes a **Vector column**

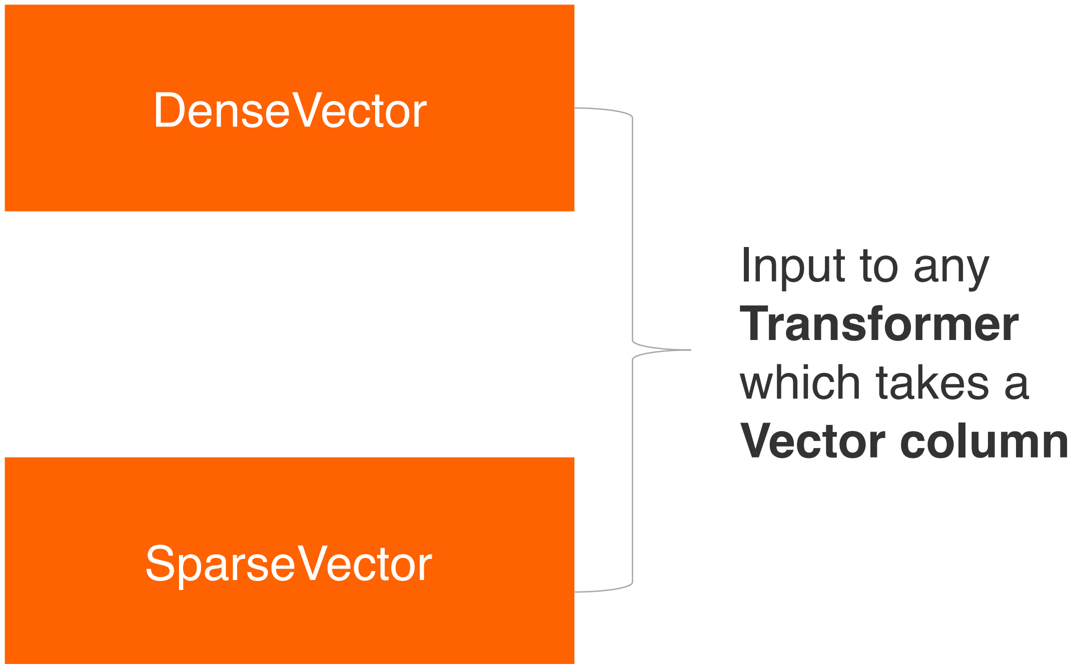
DenseVector vs. SparseVector

```
np.array( [1.5, 2.0, 1.3, 0.1])
```

1.5	2.0	1.3	0.1

```
from pyspark.mllib.linalg import Vectors
Vectors.sparse( 13, {4: 1, 5: 1, 8: 1, 11: 2})
```

0	0	0	0	1	1	0	0	1	0	0	2	0

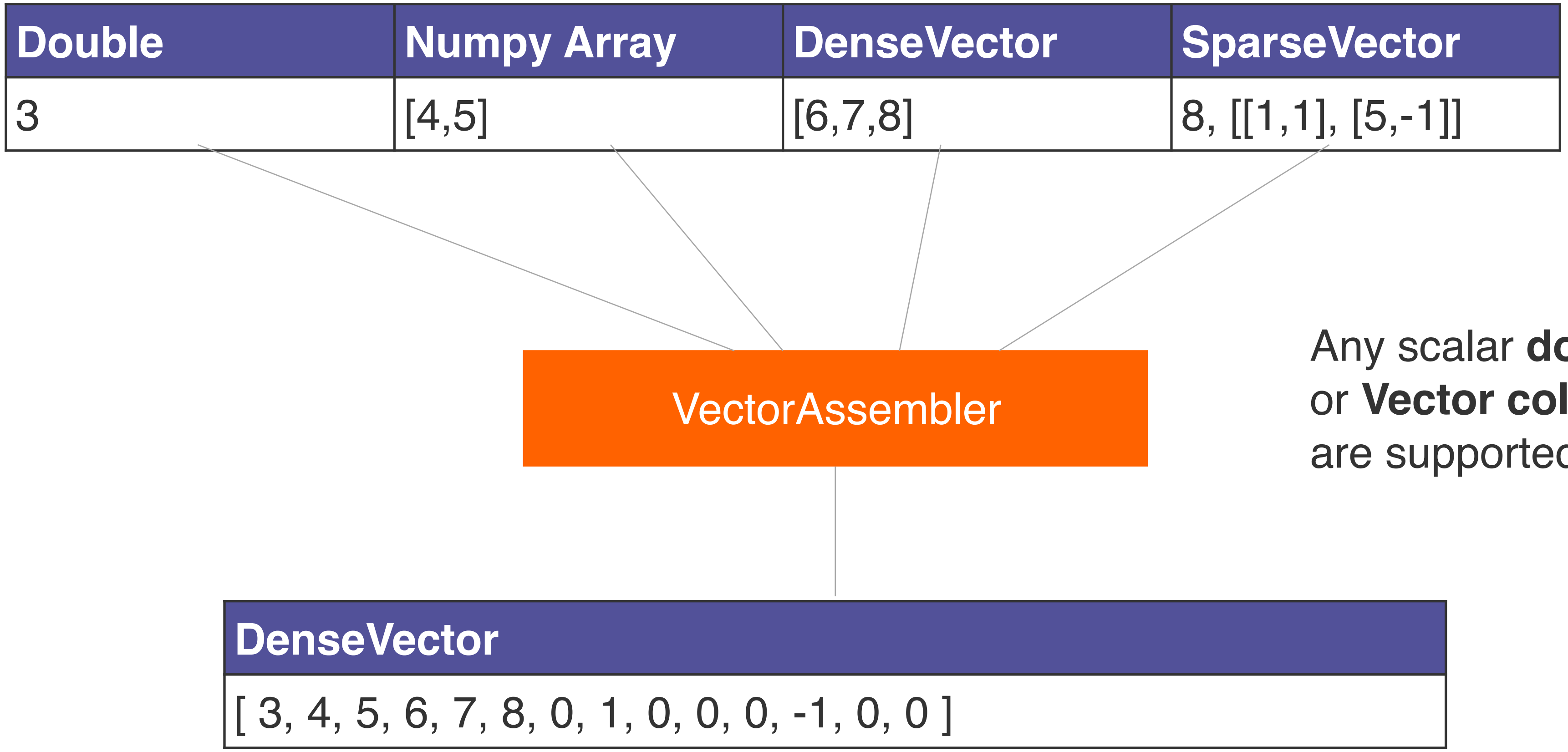


DenseVector

SparseVector

Input to any **Transformer** which takes a **Vector column**

VectorAssembler



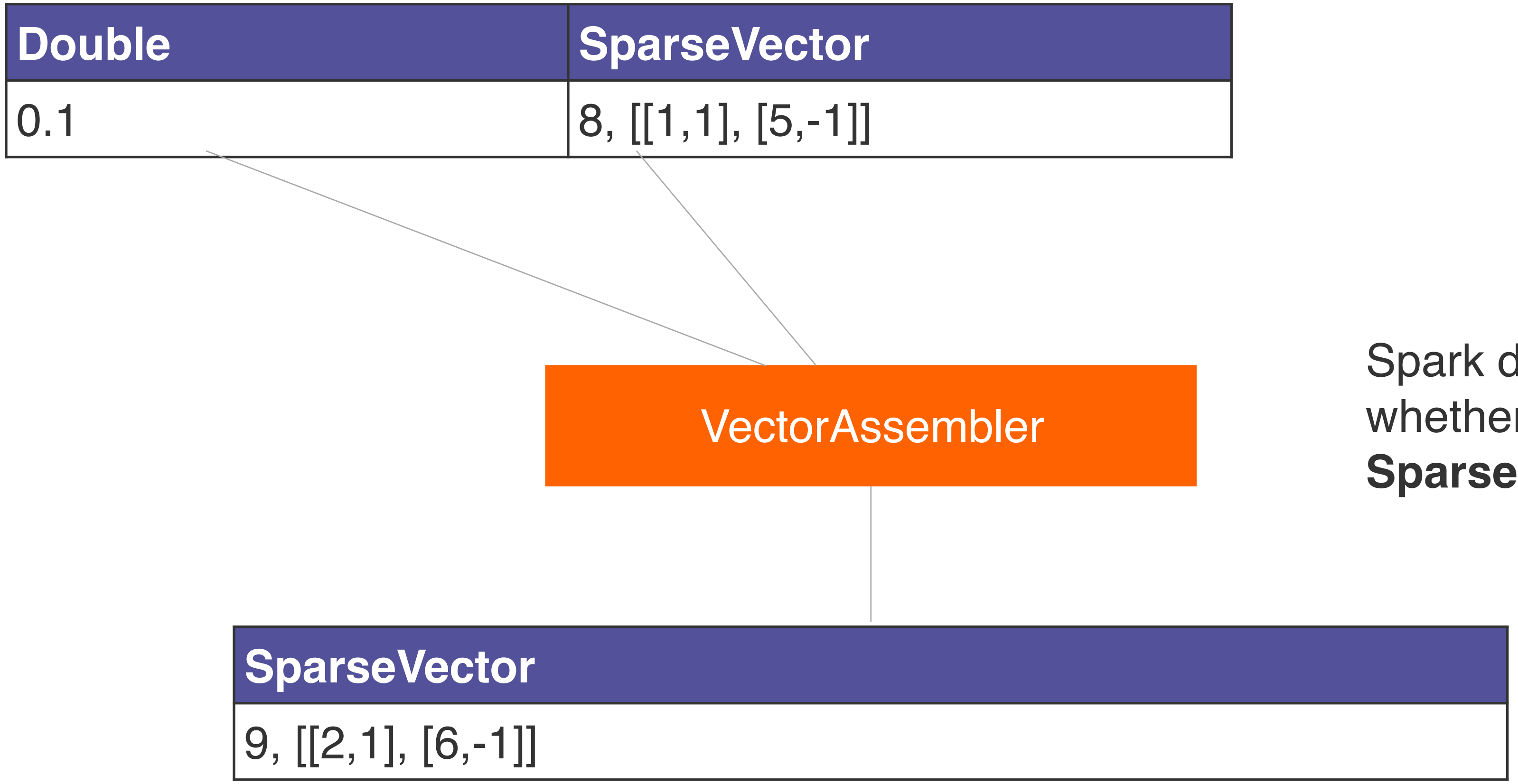
Double	Numpy Array	DenseVector	SparseVector
3	[4,5]	[6,7,8]	8, [[1,1], [5,-1]]

VectorAssembler

DenseVector
[3, 4, 5, 6, 7, 8, 0, 1, 0, 0, 0, -1, 0, 0]

Any scalar **double column**,
or **Vector column**
are supported by **VectorAssembler**

VectorAssembler



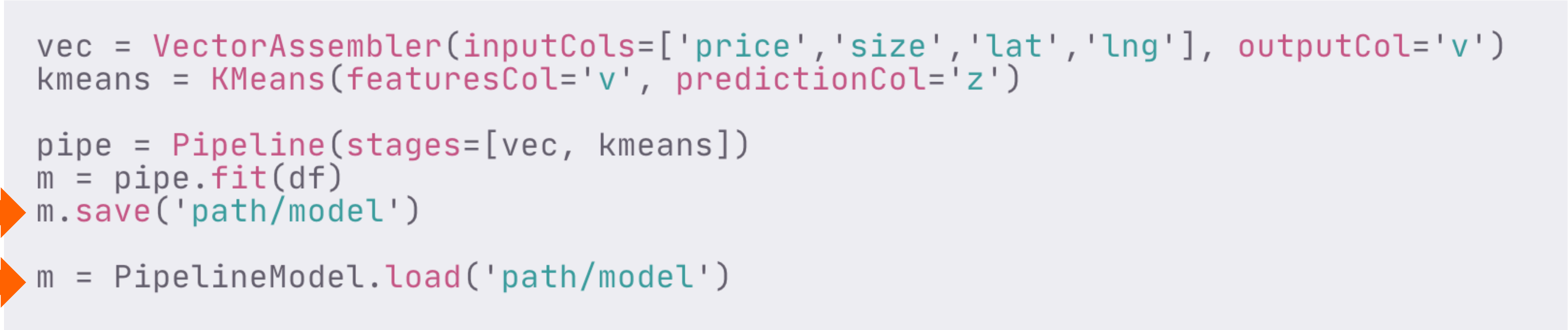
Double	SparseVector
0.1	8, [[1,1], [5,-1]]

VectorAssembler

SparseVector
9, [[2,1], [6,-1]]

Spark decides automatically whether the output should be **Sparse** or **Dense**

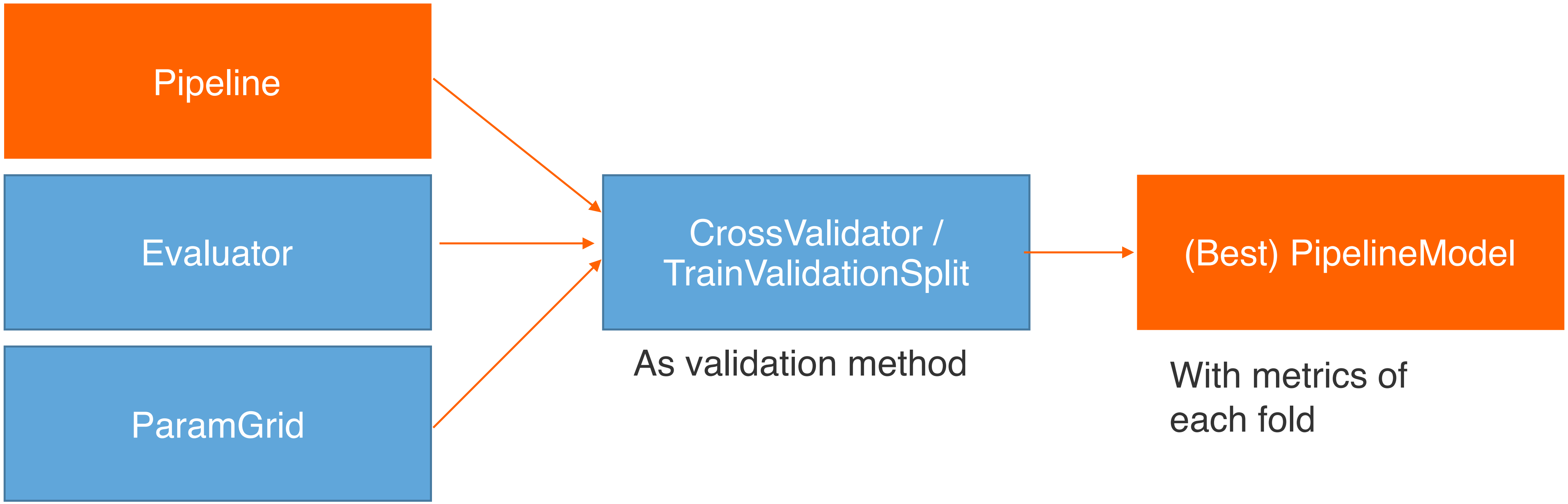
Model save & load



```
vec = VectorAssembler(inputCols=['price','size','lat','lng'], outputCol='v')
kmeans = KMeans(featuresCol='v', predictionCol='z')

pipe = Pipeline(stages=[vec, kmeans])
m = pipe.fit(df)
m.save('path/model')

m = PipelineModel.load('path/model')
```

Model file will store **Pipeline Metadata** separately from **Serialised Model object**

Cross Validation in Spark



Pipeline

Evaluator

ParamGrid

CrossValidator / TrainValidationSplit

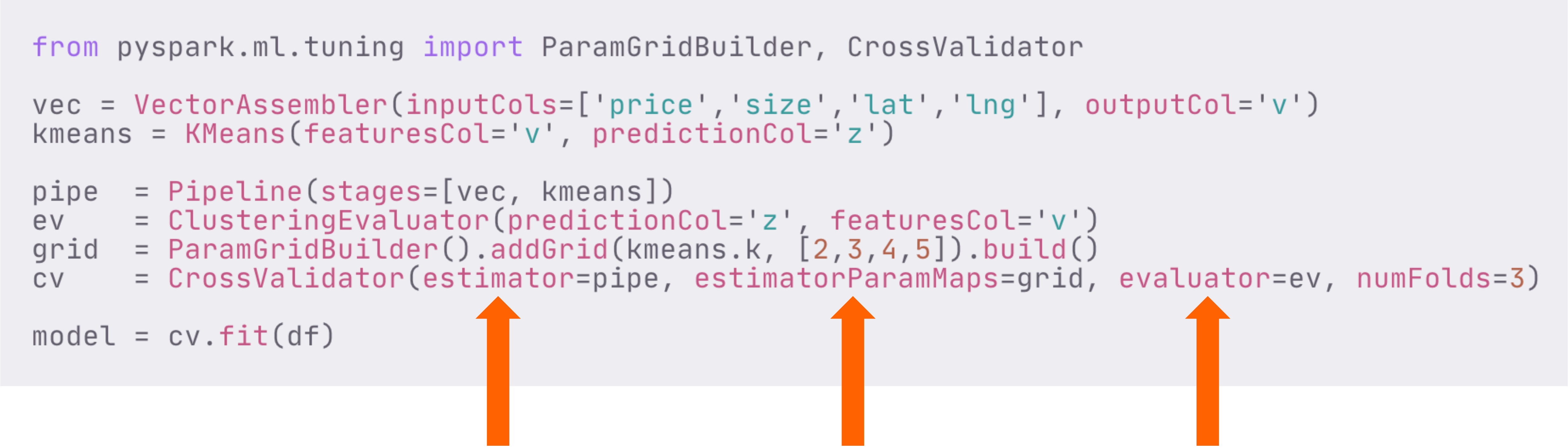
(Best) PipelineModel

As validation method

With metrics of each fold

Defines all grid parameters to explore

Cross Validation



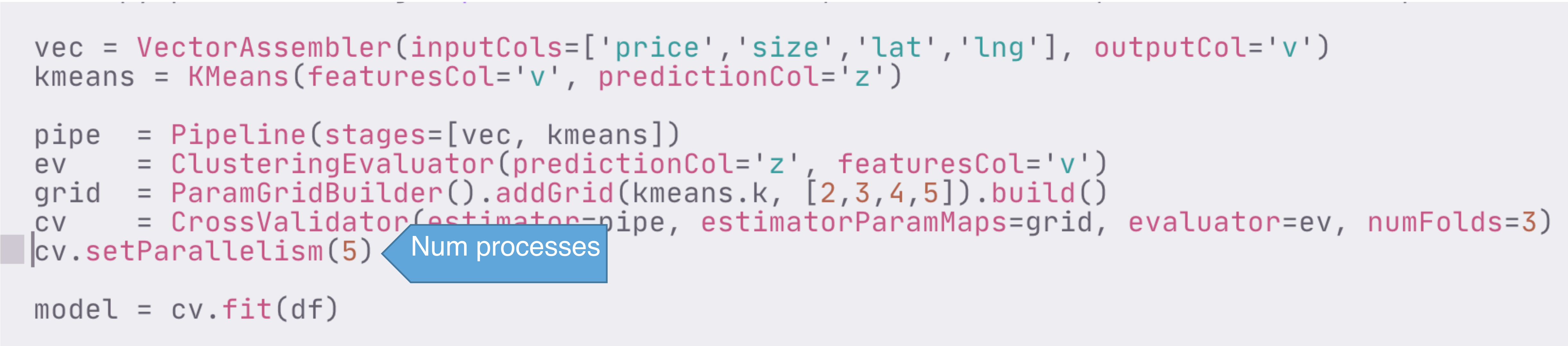
```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

vec = VectorAssembler(inputCols=['price','size','lat','lng'], outputCol='v')
kmeans = KMeans(featuresCol='v', predictionCol='z')

pipe  = Pipeline(stages=[vec, kmeans])
ev    = ClusteringEvaluator(predictionCol='z', featuresCol='v')
grid  = ParamGridBuilder().addGrid(kmeans.k, [2,3,4,5]).build()
cv    = CrossValidator(estimator=pipe, estimatorParamMaps=grid, evaluator=ev, numFolds=3)

model = cv.fit(df)
```

Assign pipeline / estimator to fit

Assign grid search

Assign evaluation method

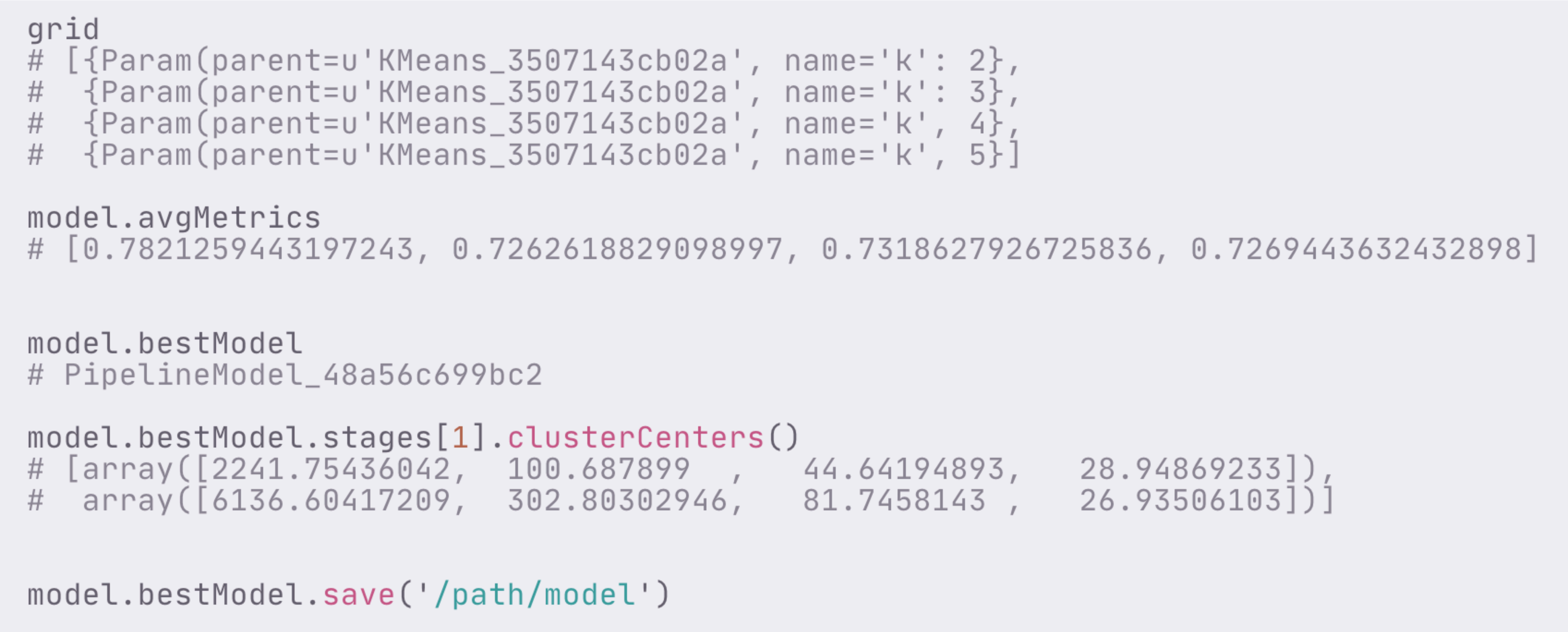
Parallel Cross Validation !



```
vec = VectorAssembler(inputCols=['price','size','lat','lng'], outputCol='v')
kmeans = KMeans(featuresCol='v', predictionCol='z')

pipe  = Pipeline(stages=[vec, kmeans])
ev    = ClusteringEvaluator(predictionCol='z', featuresCol='v')
grid  = ParamGridBuilder().addGrid(kmeans.k, [2,3,4,5]).build()
cv    = CrossValidator(estimator=pipe, estimatorParamMaps=grid, evaluator=ev, numFolds=3)
cv.setParallelism(5)

model = cv.fit(df)
```

Num processes

Cross Validation Results



```
grid
# [{Param(parent=u'KMeans_3507143cb02a', name='k': 2},
#  {Param(parent=u'KMeans_3507143cb02a', name='k': 3},
#  {Param(parent=u'KMeans_3507143cb02a', name='k', 4},
#  {Param(parent=u'KMeans_3507143cb02a', name='k', 5}]

model.avgMetrics
# [0.782125944319724З, 0.7262618829098997, 0.7318627926725836, 0.7269443632432898]


model.bestModel
# PipelineModel_48a56c699bc2

model.bestModel.stages[1].clusterCenters()
# [array([2241.75436042,  100.687899  ,   44.64194893,   28.94869233]),
#  array([6136.60417209,  302.80302946,   81.7458143 ,   26.93506103])]


model.bestModel.save('/path/model')
```