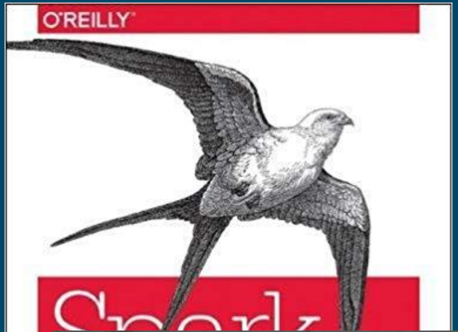
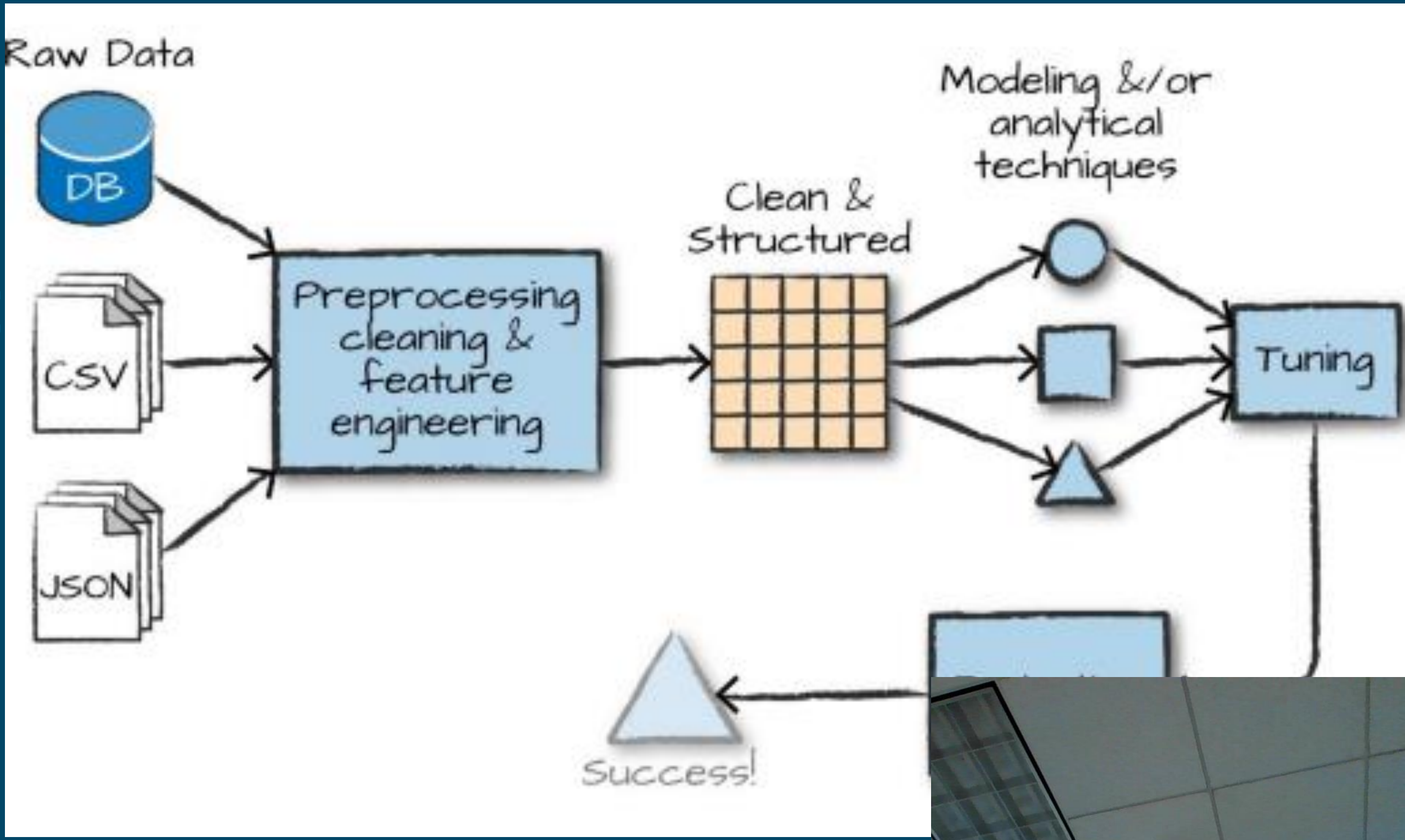
Spark Analytics
An Introductory Example

Chapter 24, Definitive guide summary



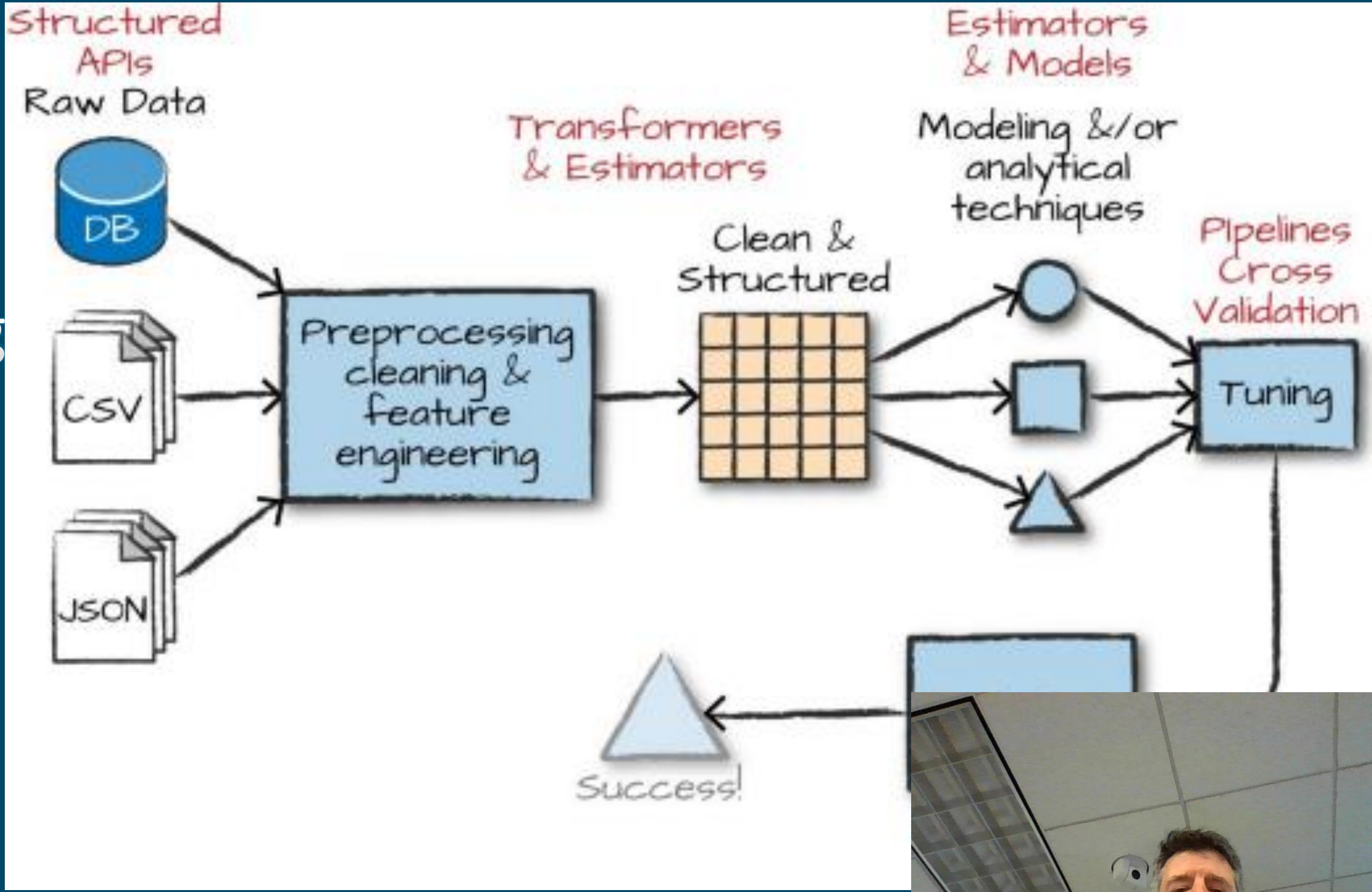
Common processing sequence

1. Collect data
2. Explore and Visualize data
3. Clean data
4. Transform data for modeling
5. Model data (e.g., regression)
6. Predict using model
7. Evaluate model prediction
8. Visualize results


Raw Data
DB
CSV
JSON
Preprocessing cleaning & feature engineering
Clean & Structured
Modeling &/or analytical techniques
Tuning
Success!

Spark provides ML API's

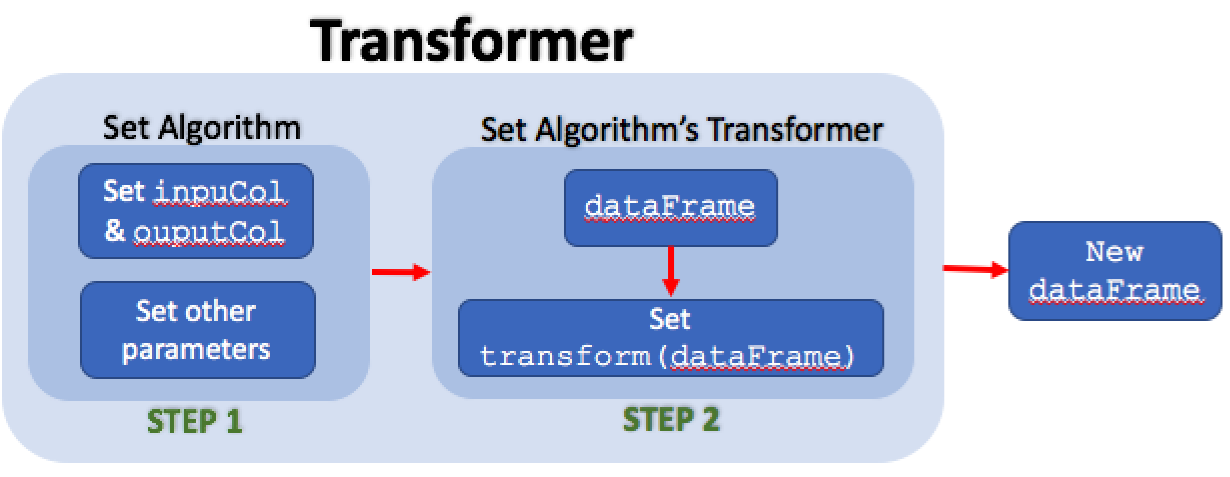
1. Collect data
2. Explore and Visualize data
3. Clean data
4. Transform data for modeling
5. Model data (e.g., regression)
6. Predict using model
7. Evaluate model prediction
8. Visualize results


Structured APIs
Raw Data
DB
CSV
JSON
Transformers & Estimators
Preprocessing cleaning & feature engineering
Clean & Structured
Estimators & Models
Modeling &/or analytical techniques
Pipelines Cross Validation
Tuning
Success!

https://spark.apache.org/docs/latest/ml-pipeline.html

Transformations are central

- Implements the transform() method, which converts a DataFrame to another DataFrame
  - Transformations copy the input to the output and add additional columns for subsequent processing



Transformer

Set Algorithm

Set inpuCol & ouputCol

Set other parameters

STEP 1

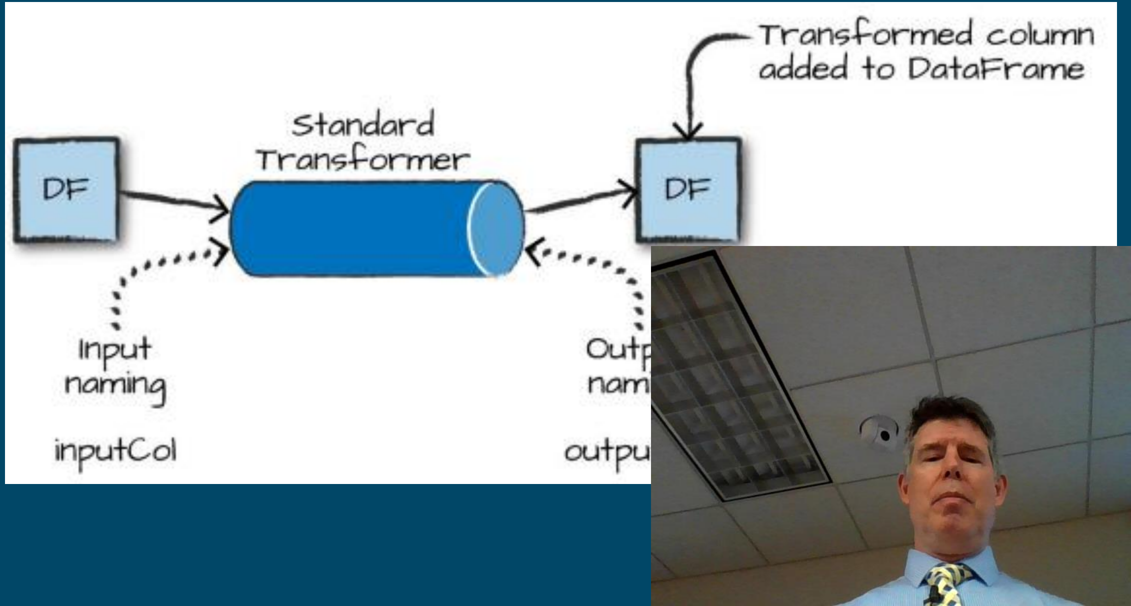
Set Algorithm's Transformer

dataFrame

Set transform(dataFrame)

STEP 2

New dataFrame



Transformed column added to DataFrame

DF

Standard Transformer

DF

Input naming

inputCol

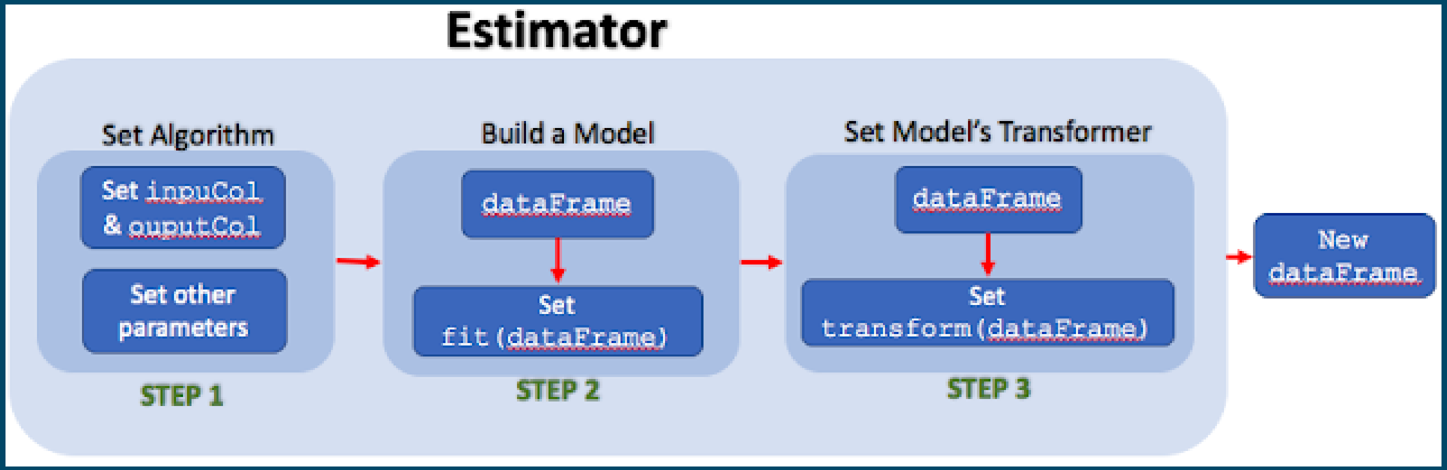
Outp nam

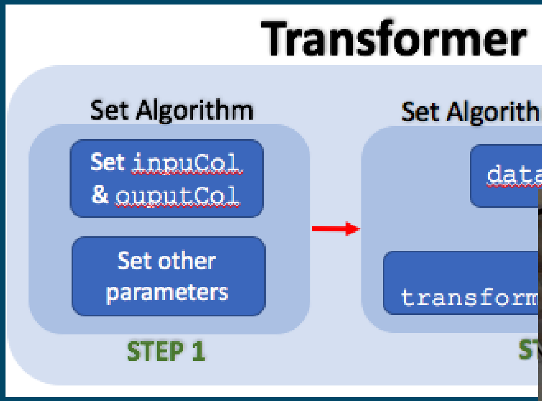
outpu

https://pysparktutorial.blogspot.com/2018/02/transformer-vs-estimator.html

Estimator creates a Transformer

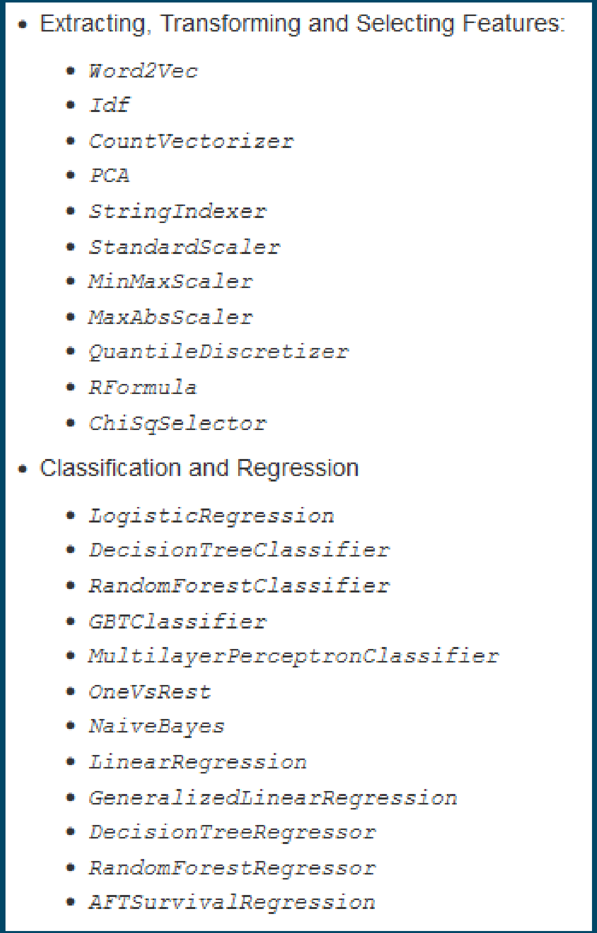
- Implements the fit() method, which applies an algorithm to a DataFrame and produces a model, which is a Transformer
  - Estimators apply fit() and then transform()



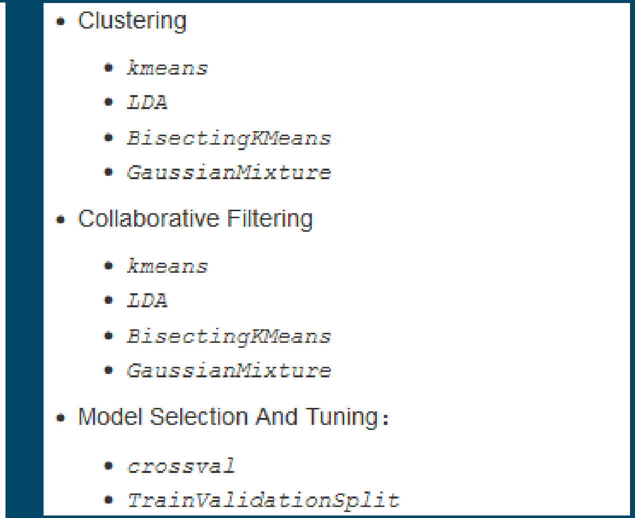


Estimators and transformers

Estimators

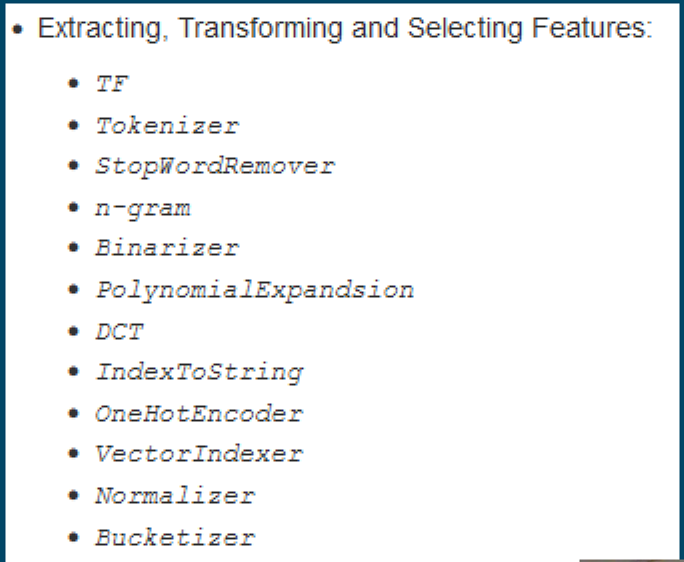


- Extracting, Transforming and Selecting Features:
  - *Word2Vec*
  - *Idf*
  - *CountVectorizer*
  - *PCA*
  - *StringIndexer*
  - *StandardScaler*
  - *MinMaxScaler*
  - *MaxAbsScaler*
  - *QuantileDiscretizer*
  - *RFormula*
  - *ChiSqSelector*
- Classification and Regression
  - *LogisticRegression*
  - *DecisionTreeClassifier*
  - *RandomForestClassifier*
  - *GBTClassifier*
  - *MultilayerPerceptronClassifier*
  - *OneVsRest*
  - *NaiveBayes*
  - *LinearRegression*
  - *GeneralizedLinearRegression*
  - *DecisionTreeRegressor*
  - *RandomForestRegressor*
  - *AFTSurvivalRegression*



- Clustering
  - *kmeans*
  - *LDA*
  - *BisectingKMeans*
  - *GaussianMixture*
- Collaborative Filtering
  - *kmeans*
  - *LDA*
  - *BisectingKMeans*
  - *GaussianMixture*
- Model Selection And Tuning:
  - *crossval*
  - *TrainValidationSplit*

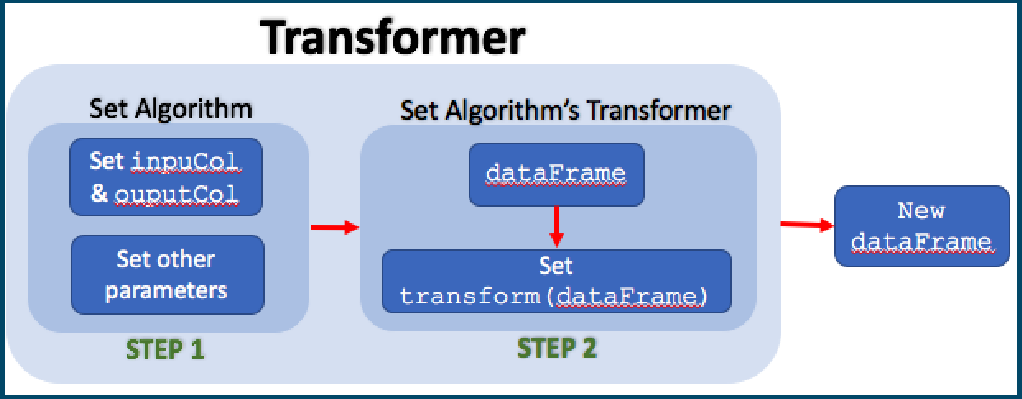
Transformers

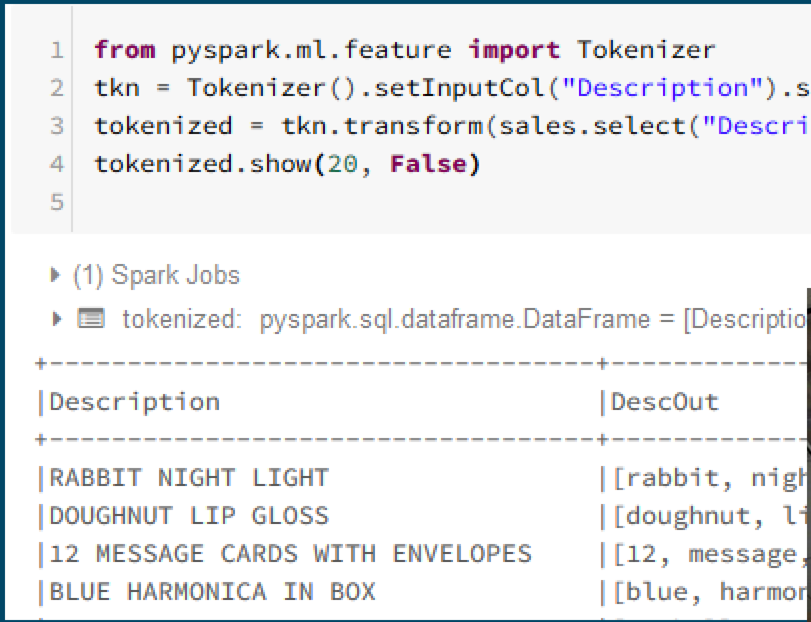


- Extracting, Transforming and Selecting Features:
  - *TF*
  - *Tokenizer*
  - *StopWordRemover*
  - *n-gram*
  - *Binarizer*
  - *PolynomialExpandsion*
  - *DCT*
  - *IndexToString*
  - *OneHotEncoder*
  - *VectorIndexer*
  - *Normalizer*
  - *Bucketizer*
  - *ElementwiseProduct*
  - *SQLTransformer*
  - *VectorAssembler*
  - *VectorSlicer*

Example transformer: Tokenizer

- Transformer
  - One pass through data to transformation the data



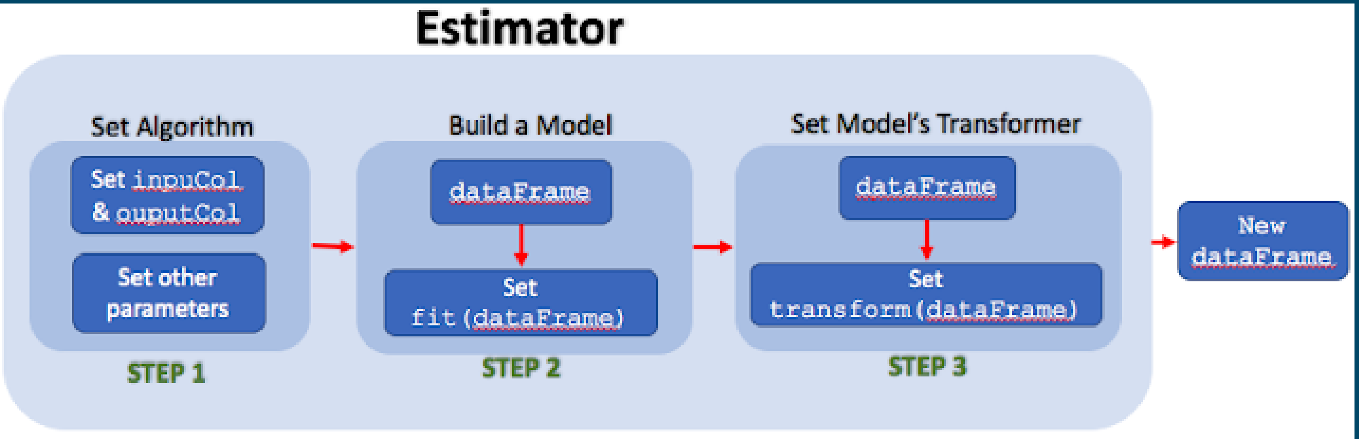


```python
from pyspark.ml.feature import Tokenizer
tkn = Tokenizer().setInputCol("Description").setOutputCol("DescOut")
tokenized = tkn.transform(sales.select("Description"))
tokenized.show(20, False)
```

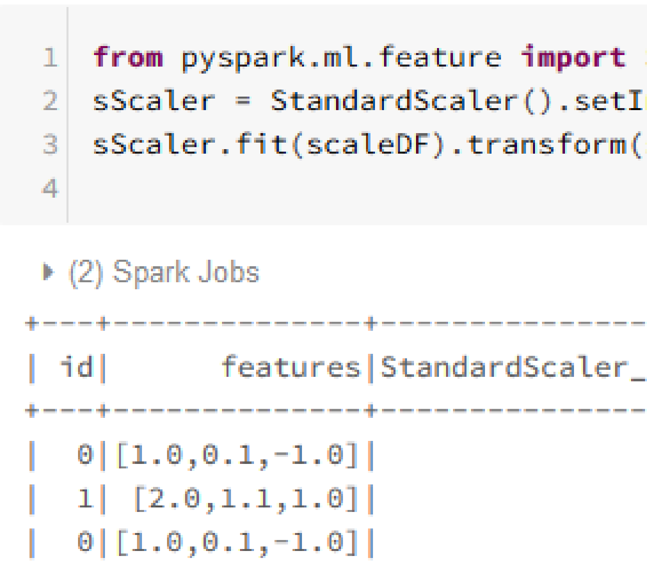
```
▸ (1) Spark Jobs
▸ tokenized:  pyspark.sql.dataframe.DataFrame = [Descriptio
+-----------------------------------+-----------
|Description                        |DescOut
+-----------------------------------+-----------
|RABBIT NIGHT LIGHT                 |[rabbit, nigh
|DOUGHNUT LIP GLOSS                 |[doughnut, li
|12 MESSAGE CARDS WITH ENVELOPES    |[12, message,
|BLUE HARMONICA IN BOX              |[blue, harmon
```

Example estimator: Scaler

- Estimator
  - First pass through data, with fit(), to determine data range for algorithm
    - Scaler returns values, [0,1] with mean of 0 and variance of 1
    - Must review values, fit(), before doing transform()
  - Second pass through data, with transform(), to transform the data



Estimator



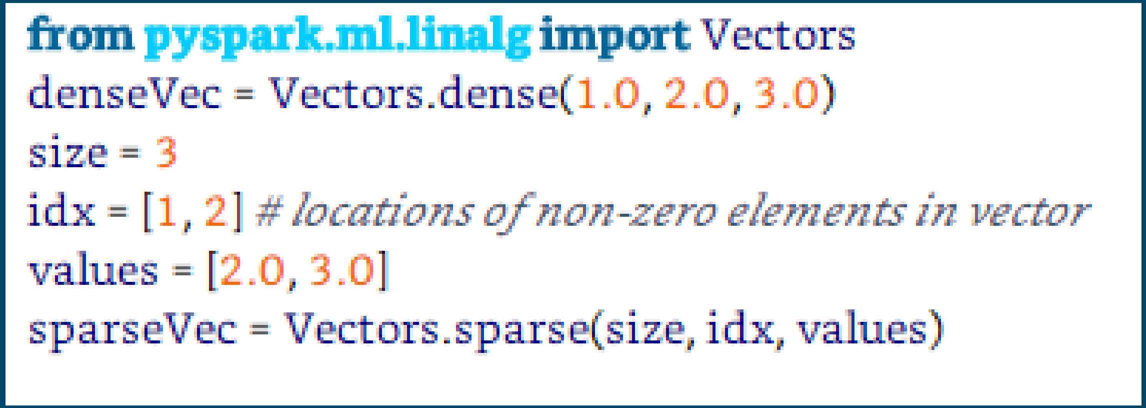
```
from pyspark.ml.feature import StandardScaler
sScaler = StandardScaler().setInputCol("features")
sScaler.fit(scaleDF).transform(scaleDF).show()

```

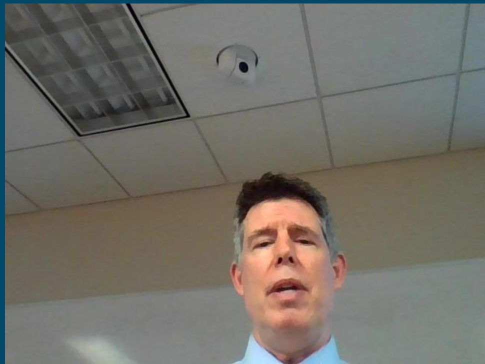
```
+---+--------------+---------------
| id|      features|StandardScaler_
+---+--------------+---------------
|  0|[1.0,0.1,-1.0]|
|  1| [2.0,1.1,1.0]|
|  0|[1.0,0.1,-1.0]|
```

ML lib represents Features as a vector

- Transformations create features
- Features must be a vector of Double values
  - sparse (where most of the elements are zero)
  - dense (where there are many unique values)

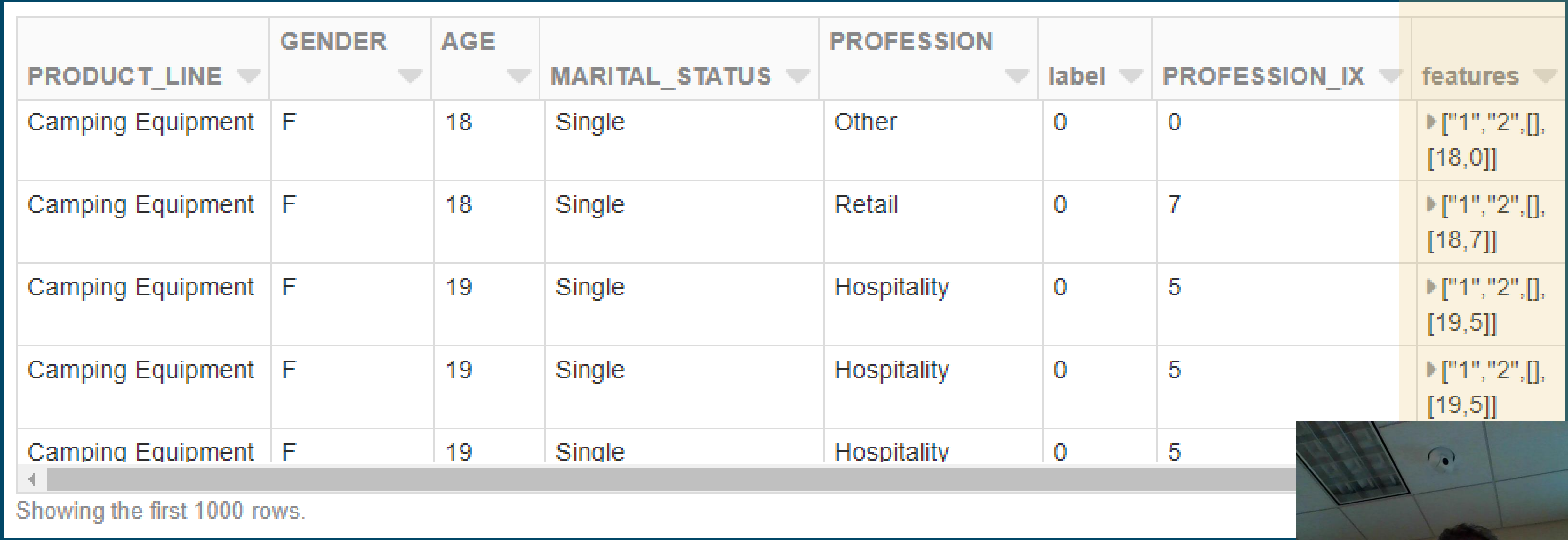


```
from pyspark.ml.linalg import Vectors
denseVec = Vectors.dense(1.0, 2.0, 3.0)
size = 3
idx = [1, 2] # locations of non-zero elements in vector
values = [2.0, 3.0]
sparseVec = Vectors.sparse(size, idx, values)
```



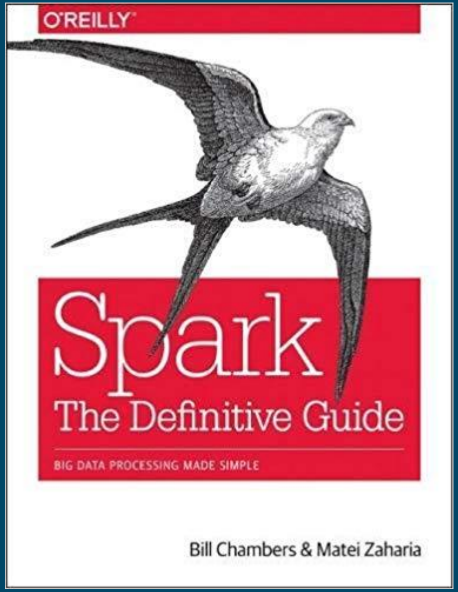
ML DataFrame: data + features

Feature vector



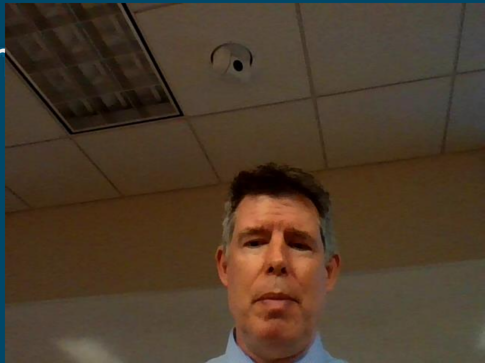
PRODUCT_LINE	GENDER	AGE	MARITAL_STATUS	PROFESSION	label	PROFESSION_IX	features
Camping Equipment	F	18	Single	Other	0	0	["1","2",[],[18,0]]
Camping Equipment	F	18	Single	Retail	0	7	["1","2",[],[18,7]]
Camping Equipment	F	19	Single	Hospitality	0	5	["1","2",[],[19,5]]
Camping Equipment	F	19	Single	Hospitality	0	5	["1","2",[],[19,5]]
Camping Equipment	F	19	Single	Hospitality	0	5	

Showing the first 1000 rows.


O'REILLY
Spark
The Definitive Guide
BIG DATA PROCESSING MADE SIMPLE
Bill Chambers & Matei Zaharia

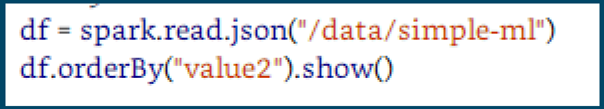
MLlib Example

Chapter 24 exar

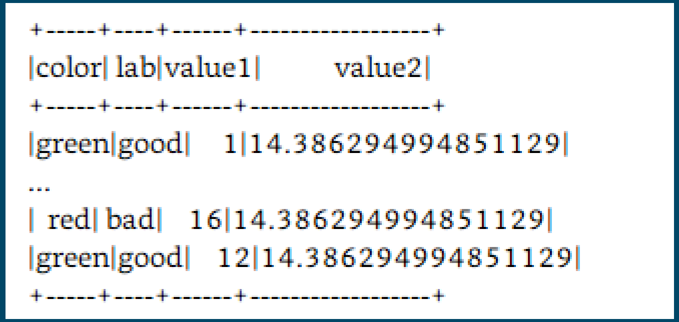


Classifying colors as good or bad

- Read the data

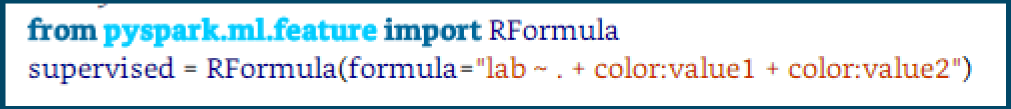


```
df = spark.read.json("/data/simple-ml")
df.orderBy("value2").show()
```



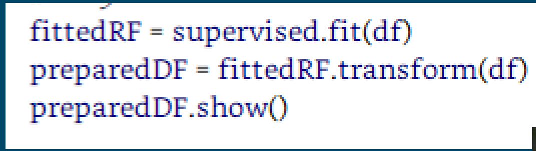
```
+-----+----+------+------------------+
|color| lab|value1|            value2|
+-----+----+------+------------------+
|green|good|     1|14.386294994851129|
...
|  red| bad|    16|14.386294994851129|
|green|good|    12|14.386294994851129|
+-----+----+------+------------------+
```

- Create an R formula for regression
  - A kind of estimator

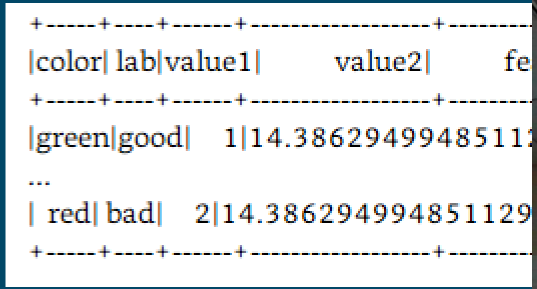


```
from pyspark.ml.feature import RFormula
supervised = RFormula(formula="lab ~ . + color:value1 + color:value2")
```

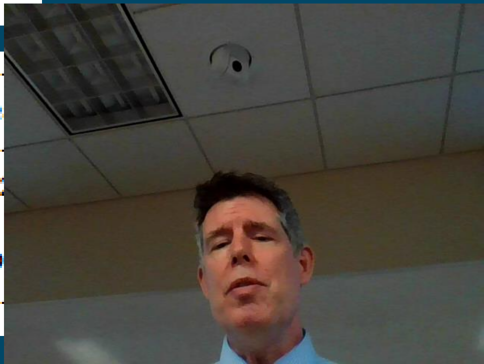
- Fit the R-formula model to the data
  - 1st (fit) it reads all data, checking for categorical values
  - 2nd (transform) it transforms the DF according to the R-formula in preparation for input to a model (notice the features and label columns)



```
fittedRF = supervised.fit(df)
preparedDF = fittedRF.transform(df)
preparedDF.show()
```

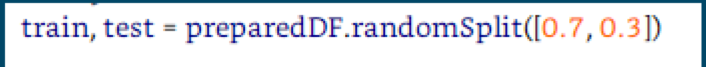


```
+-----+----+------+------------------+--------
|color| lab|value1|            value2|       fe
+-----+----+------+------------------+--------
|green|good|     1|14.38629499485111
...
|  red| bad|     2|14.386294994851129
+-----+----+------+------------------+--------
```



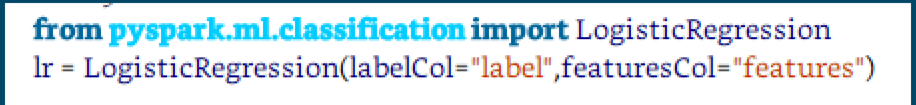
Applying a model to the prepared features

- Split DataFrame into training and test datasets



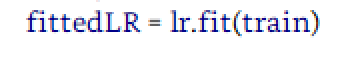
```
train, test = preparedDF.randomSplit([0.7, 0.3])
```

- Create an instance of the regression (estimator)



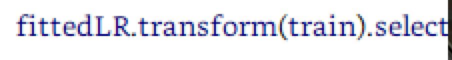
```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="label",featuresCol="features")
```

- Fit the regression estimator to the training data, which creates a model
  - This action starts a Spark job

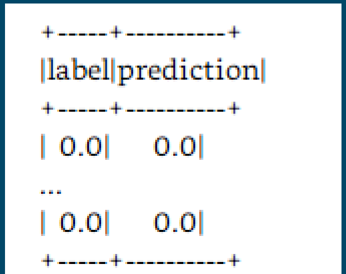


```
fittedLR = lr.fit(train)
```

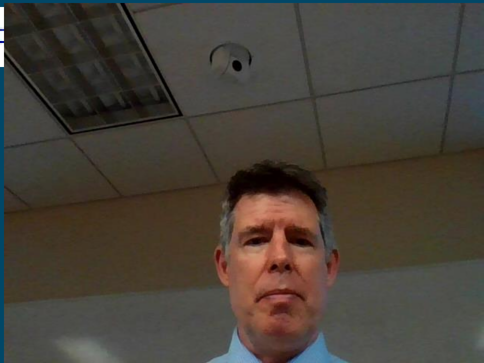
- Transform the model for predictions
  - Creates prediction (column) for labels



```
fittedLR.transform(train).select
```

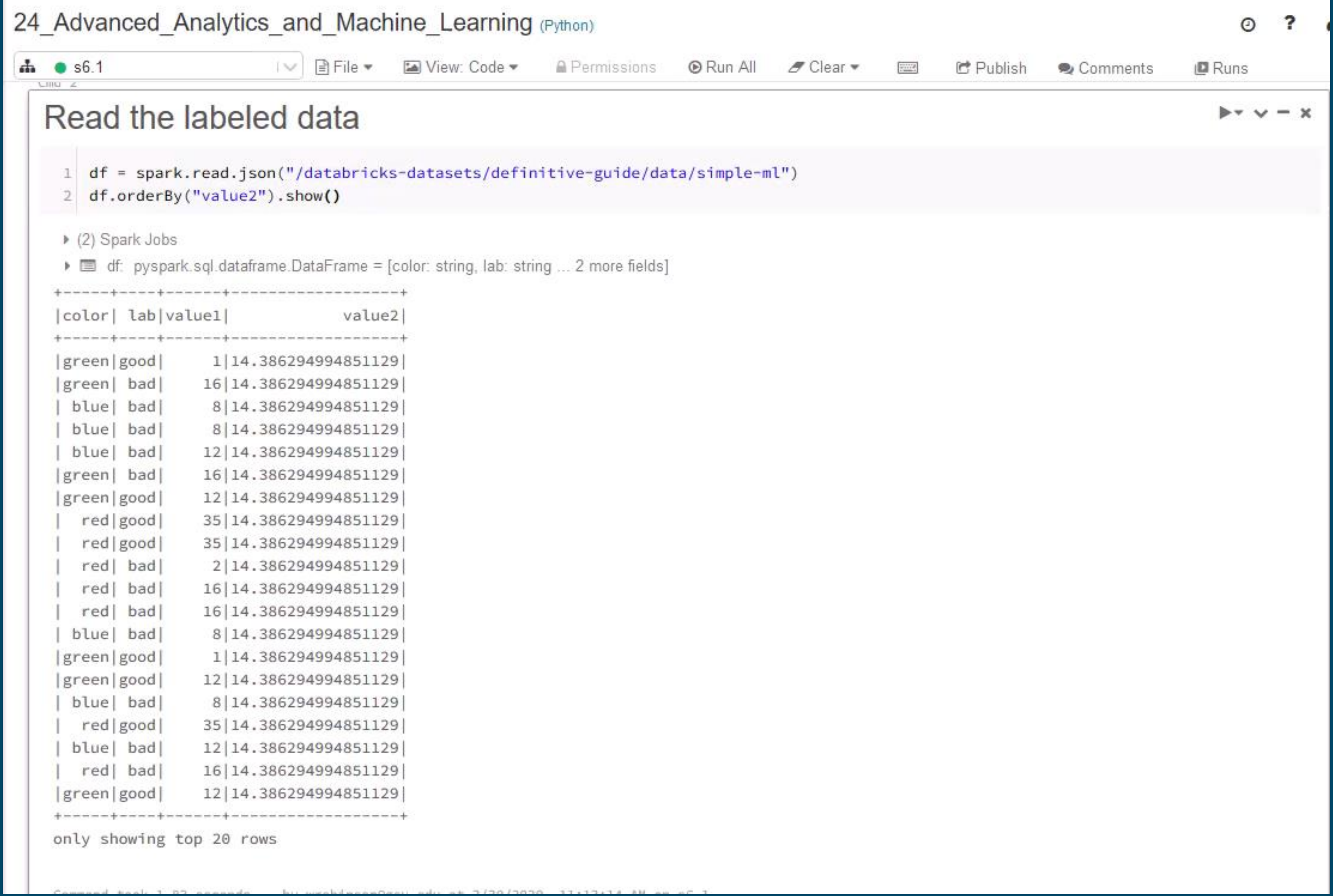


```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
...
|  0.0|       0.0|
+-----+----------+
```



Run Definitive Guide examples on Databricks

- Get code from GitHub
  - https://github.com/databricks/Spark-The-Definitive-Guide
- To upload a notebook (for a book chapter)
  - Navigate to the notebook you would like to import
  - Navigate to the **RAW** version of the file and save that to your computer
  - Select a folder on DataBricks, right-click to Import, and select your downloaded file
  - Replace the data paths in each notebook
    - from /data
    - to /databricks-datasets/definitive-guide/data


24_Advanced_Analytics_and_Machine_Learning (Python)
s6.1
File
View: Code
Permissions
Run All
Clear
Publish
Comments
Runs

Read the labeled data

```
df = spark.read.json("/databricks-datasets/definitive-guide/data/simple-ml")
df.orderBy("value2").show()
```

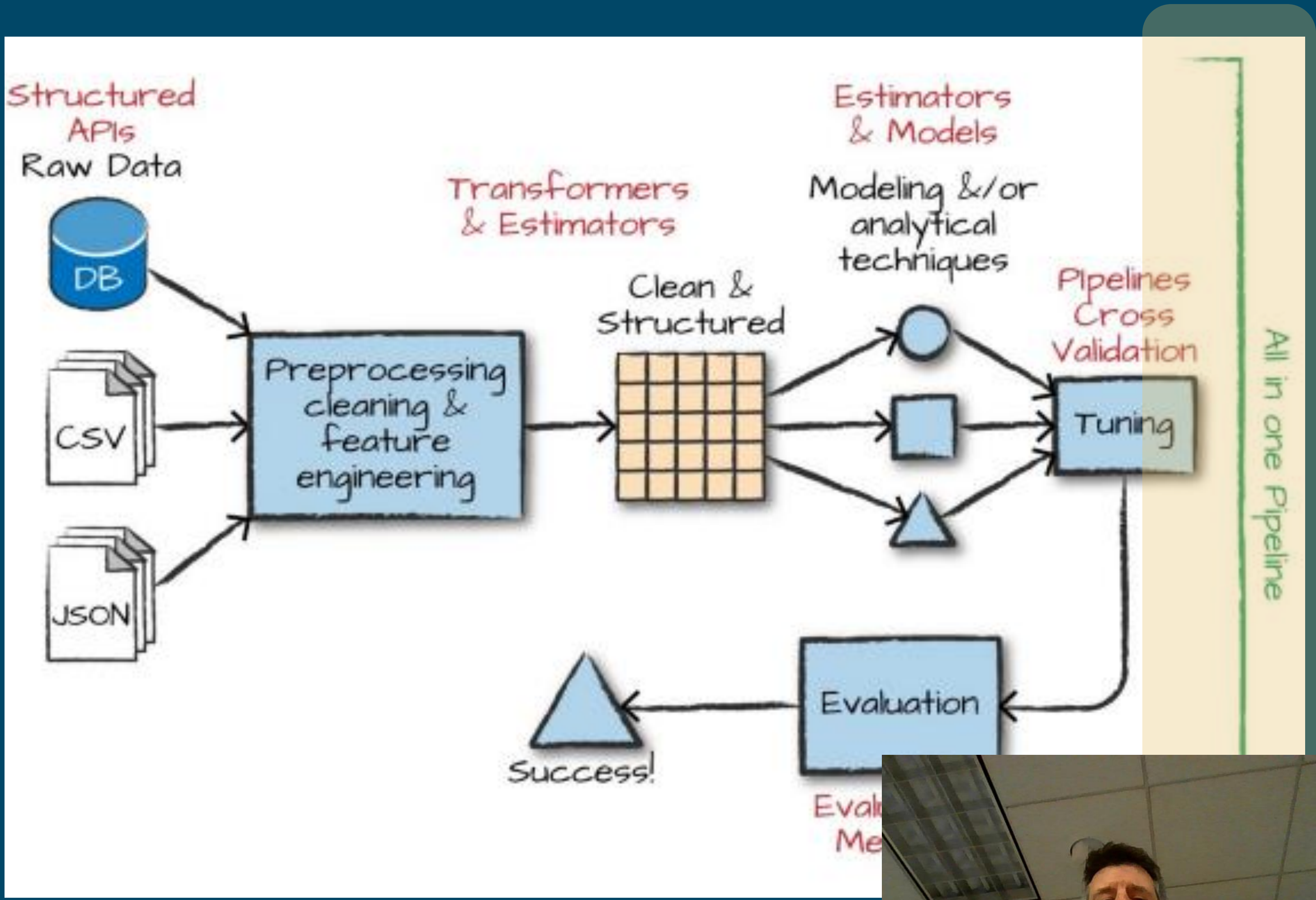
(2) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [color: string, lab: string ... 2 more fields]

```
+-----+----+------+------------------+
|color| lab|value1|            value2|
+-----+----+------+------------------+
|green|good|     1|14.386294994851129|
|green| bad|    16|14.386294994851129|
| blue| bad|     8|14.386294994851129|
| blue| bad|     8|14.386294994851129|
| blue| bad|    12|14.386294994851129|
|green| bad|    16|14.386294994851129|
|green|good|    12|14.386294994851129|
|  red|good|    35|14.386294994851129|
|  red|good|    35|14.386294994851129|
|  red| bad|     2|14.386294994851129|
|  red| bad|    16|14.386294994851129|
|  red| bad|    16|14.386294994851129|
| blue| bad|     8|14.386294994851129|
|green|good|     1|14.386294994851129|
|green|good|    12|14.386294994851129|
| blue| bad|     8|14.386294994851129|
|  red|good|    35|14.386294994851129|
| blue| bad|    12|14.386294994851129|
|  red| bad|    16|14.386294994851129|
|green|good|    12|14.386294994851129|
+-----+----+------+------------------+
only showing top 20 rows
```

Let’s do it again, but with a pipeline

Pipeline for transformations, estimators, evaluators

1. Collect data
2. Explore and Visualize data
3. Clean data
4. Transform data for modeling
5. Model data (e.g., regression)
6. Predict using model
7. Evaluate model prediction
8. Visualize results


Structured APIs
Raw Data
DB
CSV
JSON
Preprocessing cleaning & feature engineering
Transformers & Estimators
Clean & Structured
Estimators & Models
Modeling &/or analytical techniques
Pipelines Cross Validation
Tuning
All in one Pipeline
Evaluation
Success!
Eval
Me

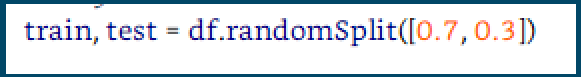
Spark ML Pipeline

- Inspired by Python’s Scikit-learn ML API
- A Spark ML Pipeline chains multiple Transformers and Estimators together to specify an ML workflow
  - A Pipeline is a class that is part of the ML API for simplifying the specification of programming steps needed to create ML models

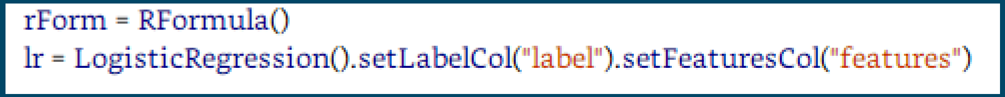
https://spark.apache.org/docs/latest/ml-pipeline.html#how-it-works

Creating the pipeline 1/2

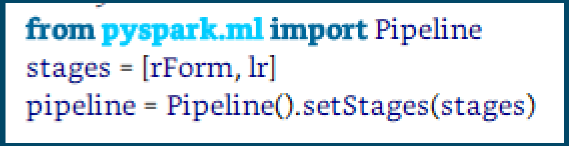
- Train & test datasets
- Pipeline elements
  - R formula
  - Regression
- Pipeline
- Parameter Grid
  - R formula
    - 2 versions (in list)
  - Regression
    - 2 parameters each with various values (in lists)



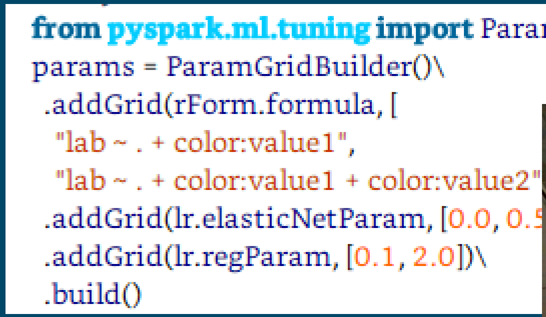
```
train, test = df.randomSplit([0.7, 0.3])
```



```
rForm = RFormula()
lr = LogisticRegression().setLabelCol("label").setFeaturesCol("features")
```



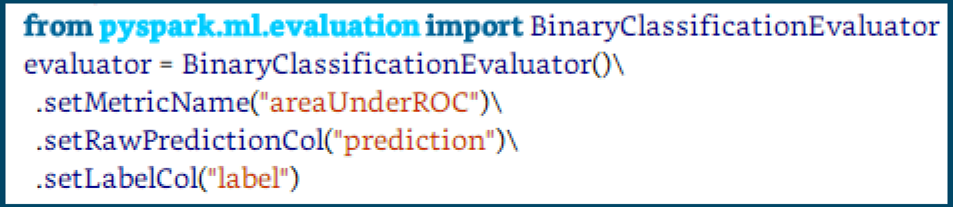
```
from pyspark.ml import Pipeline
stages = [rForm, lr]
pipeline = Pipeline().setStages(stages)
```



```
from pyspark.ml.tuning import ParamGridBuilder
params = ParamGridBuilder()\
 .addGrid(rForm.formula, [
  "lab ~ . + color:value1",
  "lab ~ . + color:value1 + color:value2"
 .addGrid(lr.elasticNetParam, [0.0, 0.5
 .addGrid(lr.regParam, [0.1, 2.0])\
 .build()
```

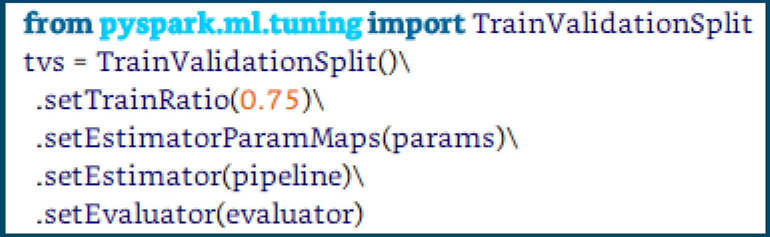
Creating the pipeline 2/2

- Evaluator
  - Used by pipeline to pick best model



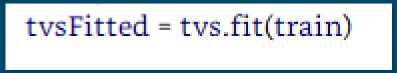
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()\
 .setMetricName("areaUnderROC")\
 .setRawPredictionCol("prediction")\
 .setLabelCol("label")
```

- Define training datasets
  - TrainValidationSplit (or CrossValidator)
    - Runs the pipeline
    - Set the params, pipeline, & evaluator



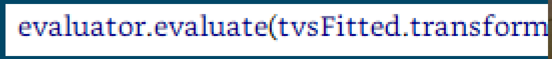
```
from pyspark.ml.tuning import TrainValidationSplit
tvs = TrainValidationSplit()\
 .setTrainRatio(0.75)\
 .setEstimatorParamMaps(params)\
 .setEstimator(pipeline)\
 .setEvaluator(evaluator)
```

- Run the pipeline
  - fit() method on TrainValidationSplit
    - Calls fit() & transform() on elements of pipeline

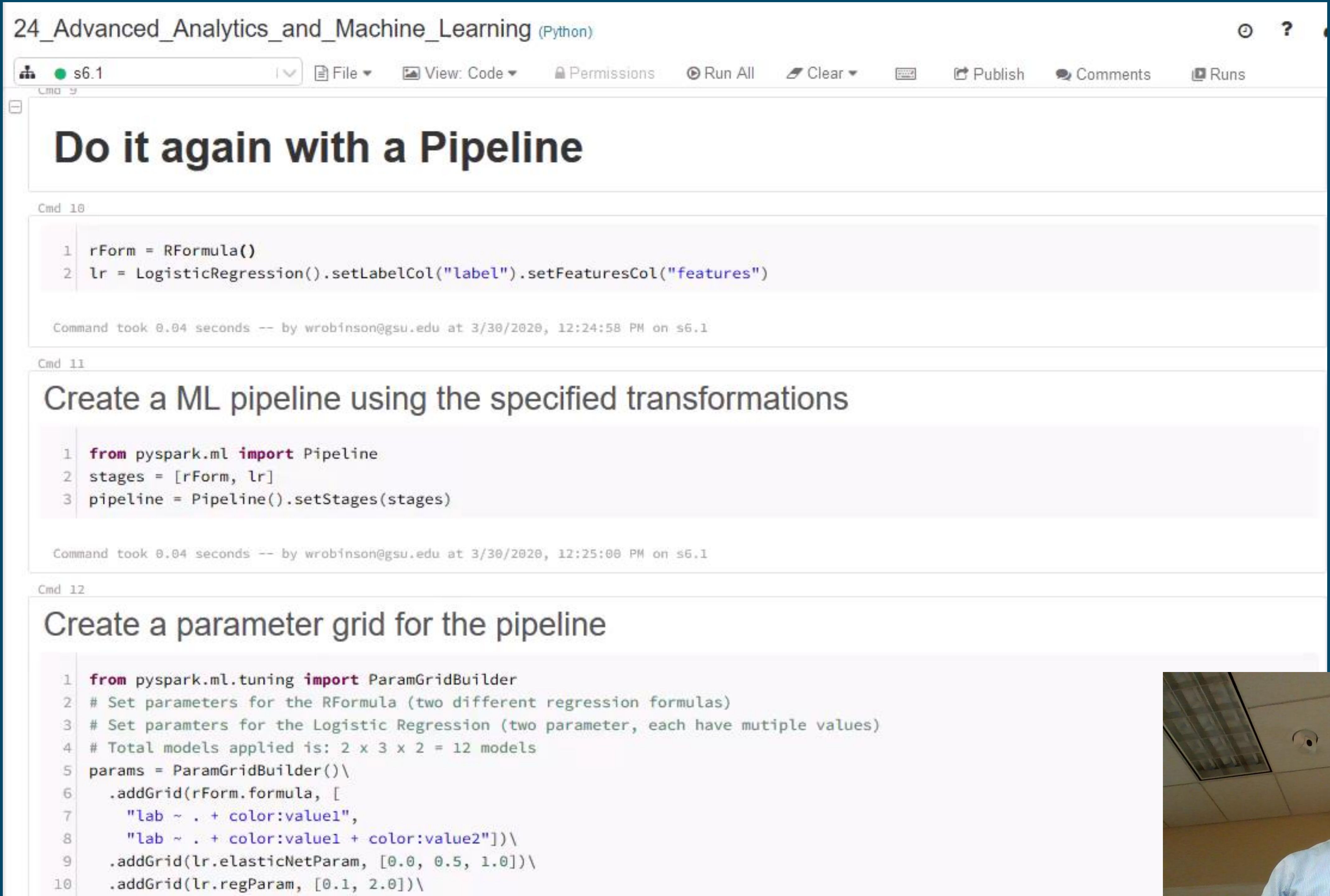


```
tvsFitted = tvs.fit(train)
```

- Evaluate the best model on test data
  - tvs.Fitted is the best model



```
evaluator.evaluate(tvsFitted.transform
```



Do it again with a Pipeline

Cmd 10

```
rForm = RFormula()
lr = LogisticRegression().setLabelCol("label").setFeaturesCol("features")
```

Command took 0.04 seconds -- by wrobinson@gsu.edu at 3/30/2020, 12:24:58 PM on s6.1

Cmd 11

Create a ML pipeline using the specified transformations

```
from pyspark.ml import Pipeline
stages = [rForm, lr]
pipeline = Pipeline().setStages(stages)
```

Command took 0.04 seconds -- by wrobinson@gsu.edu at 3/30/2020, 12:25:00 PM on s6.1

Cmd 12

Create a parameter grid for the pipeline

```
from pyspark.ml.tuning import ParamGridBuilder
# Set parameters for the RFormula (two different regression formulas)
# Set paramters for the Logistic Regression (two parameter, each have mutiple values)
# Total models applied is: 2 x 3 x 2 = 12 models
params = ParamGridBuilder()\
  .addGrid(rForm.formula, [
    "lab ~ . + color:value1",
    "lab ~ . + color:value1 + color:value2"])\
  .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])\
  .addGrid(lr.regParam, [0.1, 2.0])\
```

Important to remember

- Two essential elements of ML API
  - Transformations: transform() method take DataFrame input and add columns to new DataFrame
  - Estimators: fit() method uses algorithm to creates model, followed by transform, which adds new columns
- Transformation adds two columns
  - Features: dense vector of double values for the X columns (independent vars)
  - Label: double values of the predictor column (Y column)
- Pipeline simplifies development
  - Estimators and transformations are declared in a list for pipeline
  - ParameterGridBuilder specifies parameters for pipeline elements
  - Using fit(), TrainValidationSplit (or CrossValidator) runs the pipeline, applying th parameters and an evaluator