PySpark CrossValidate

A technique for efficiently hyper-tuning across models

Hyperparameter optimization across models

- Hyperparameter optimization
  - Construct a common pipeline, including parameter grid, across multiple models
  - Then, run CrossValidate to find the best among all models and parameters
- Efficient
- Code is simpler than repeating common code for each model

How to efficiently run multiple models on your data set?

Why run multiple models?

Because you want to select the best model with the best parameters

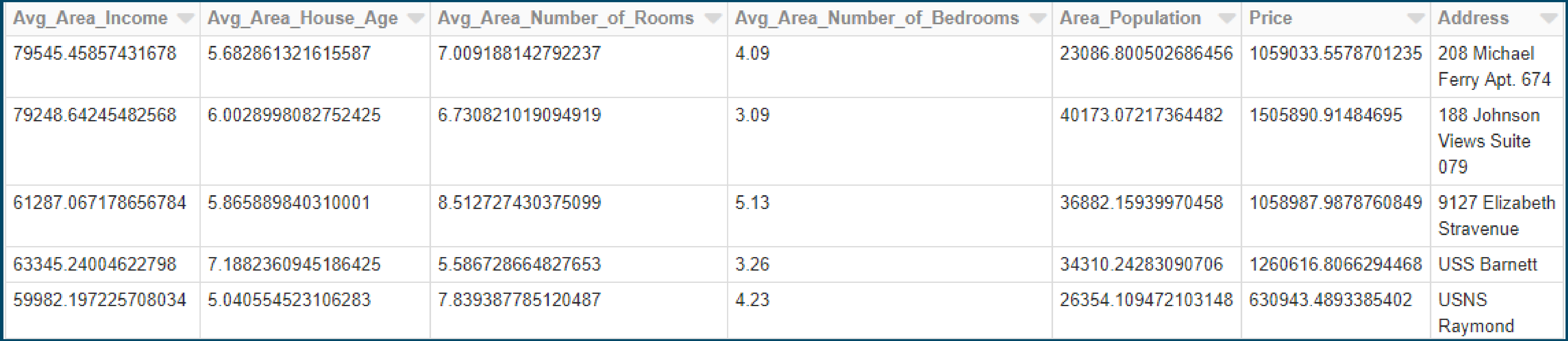
Hyperparameter optimization

- Choosing a set of optimal hyperparameters for a learning algorithm
- A hyperparameter is a parameter whose value is used to control the learning process
  - For example, the *k* in k-means

Grid search for hyperparameter optimization

- Grid search
  - an exhaustive search through a specified subset of the total hyperparameter space of a learning algorithm
  - must be guided by some performance metric, typically measured by cross-validation on the training set

Consider this example: housing data



Avg_Area_Income	Avg_Area_House_Age	Avg_Area_Number_of_Rooms	Avg_Area_Number_of_Bedrooms	Area_Population	Price	Address
79545.45857431678	5.682861321615587	7.009188142792237	4.09	23086.800502686456	1059033.5578701235	208 Michael Ferry Apt. 674
79248.64245482568	6.0028998082752425	6.730821019094919	3.09	40173.07217364482	1505890.91484695	188 Johnson Views Suite 079
61287.067178656784	5.865889840310001	8.512727430375099	5.13	36882.15939970458	1058987.9878760849	9127 Elizabeth Stravenue
63345.24004622798	7.1882360945186425	5.586728664827653	3.26	34310.24283090706	1260616.8066294468	USS Barnett
59982.197225708034	5.040554523106283	7.839387785120487	4.23	26354.109472103148	630943.4893385402	USNS Raymond

Standard pipeline for predicting housing price

- Regression features (RFormula)

```
formula = "{} ~ {}".format("Price", " + ".join(columns))
print("Formula : {}".format(formula))
rformula = RFormula(formula = formula)
```

- Regression

```
lr = LinearRegression()
```

- Pipeline

```
pipeline = Pipeline(stages=[rformula, lr])
```

- Parameter grid

```
paramGrid = ParamGridBuilder()\
            .addGrid(lr.regParam,[0.01, .04])\
            .build()
```

- Cross Validate

```
cv = CrossValidator()\
      .setEstimator(pipeline)\
      .setEvaluator(RegressionEvaluator()\
                          .setMetricName("r2"))\
      .setEstimatorParamMaps(paramGrid)\
      .setNumFolds(3)

cvModel = cv.fit(df)
```

Now, repeat for each of the next *n* models you would like to test!

☹

Repeat pipeline for next *n* models

Features prepared in the pipeline
repeat for each model
Can be numerous and slow

- Regression features (RFormula)

```
formula = "{} ~ {}".format("Price", " + ".join(columns))
print("Formula : {}".format(formula))
rformula = RFormula(formula = formula)
```

- Regression
- Pipeline
- Parameter grid

Only this changes

```
lr = LinearRegression()
```

```
pipeline = Pipeline(stages=[rformula, lr])
```

```
paramGrid = ParamGridBuilder()\
            .addGrid(lr.regParam,[0.01, .04])\
            .build()
```

- Cross Validate

```
cv = CrossValidator()\
       .setEstimator(pipeline)\
       .setEvaluator(RegressionEvaluator()\
                         .setMetricName("r2"))\
       .setEstimatorParamMaps(paramGrid)\
       .setNumFolds(3)

cvModel = cv.fit(df)
```

Rather than copy & paste code to create multiple pipelines, each of which repeats the same feature creation (early in the pipeline),

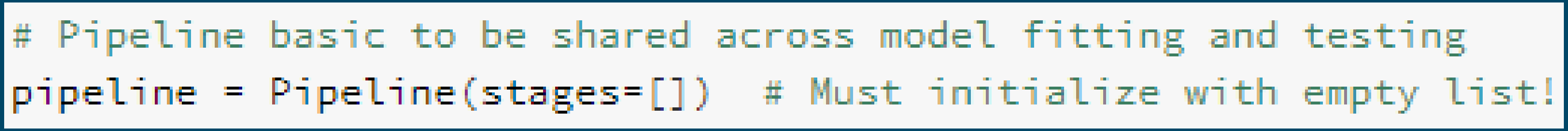
let’s put all the pipelines in the parameter grid.

The grid will use cache rather than rerun the common feature creation (RFormula)

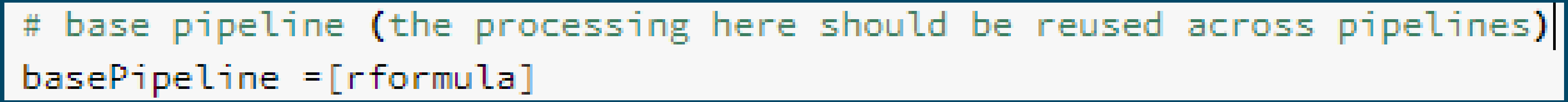
Simpler, faster code

Parameter grid pipeline technique

- Start with empty pipeline (instantiated object)

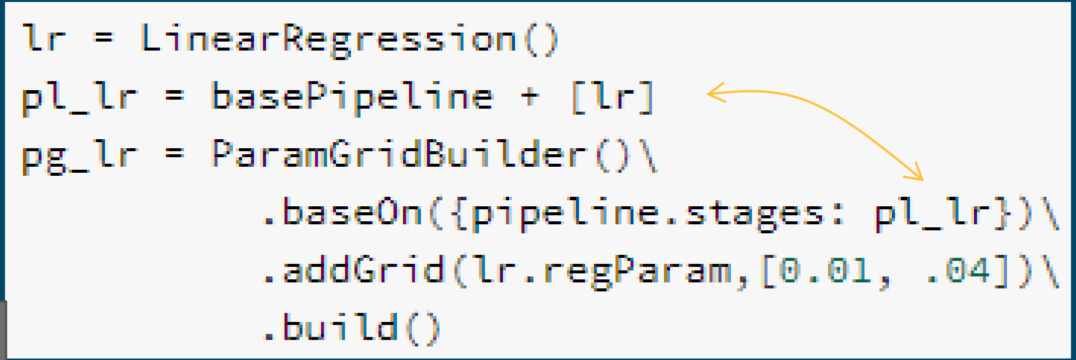


```
# Pipeline basic to be shared across model fitting and testing
pipeline = Pipeline(stages=[])  # Must initialize with empty list!
```



```
# base pipeline (the processing here should be reused across pipelines)
basePipeline =[rformula]
```

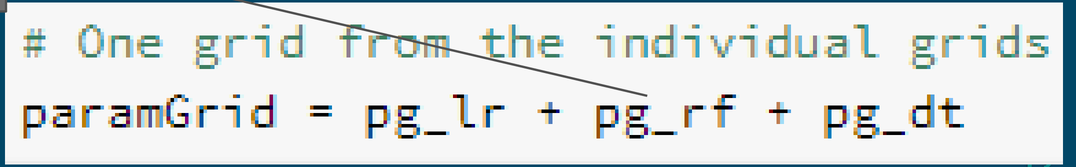
- For each model, in a parameter grid
  - add their unique pipeline objects
  - specify their parameters



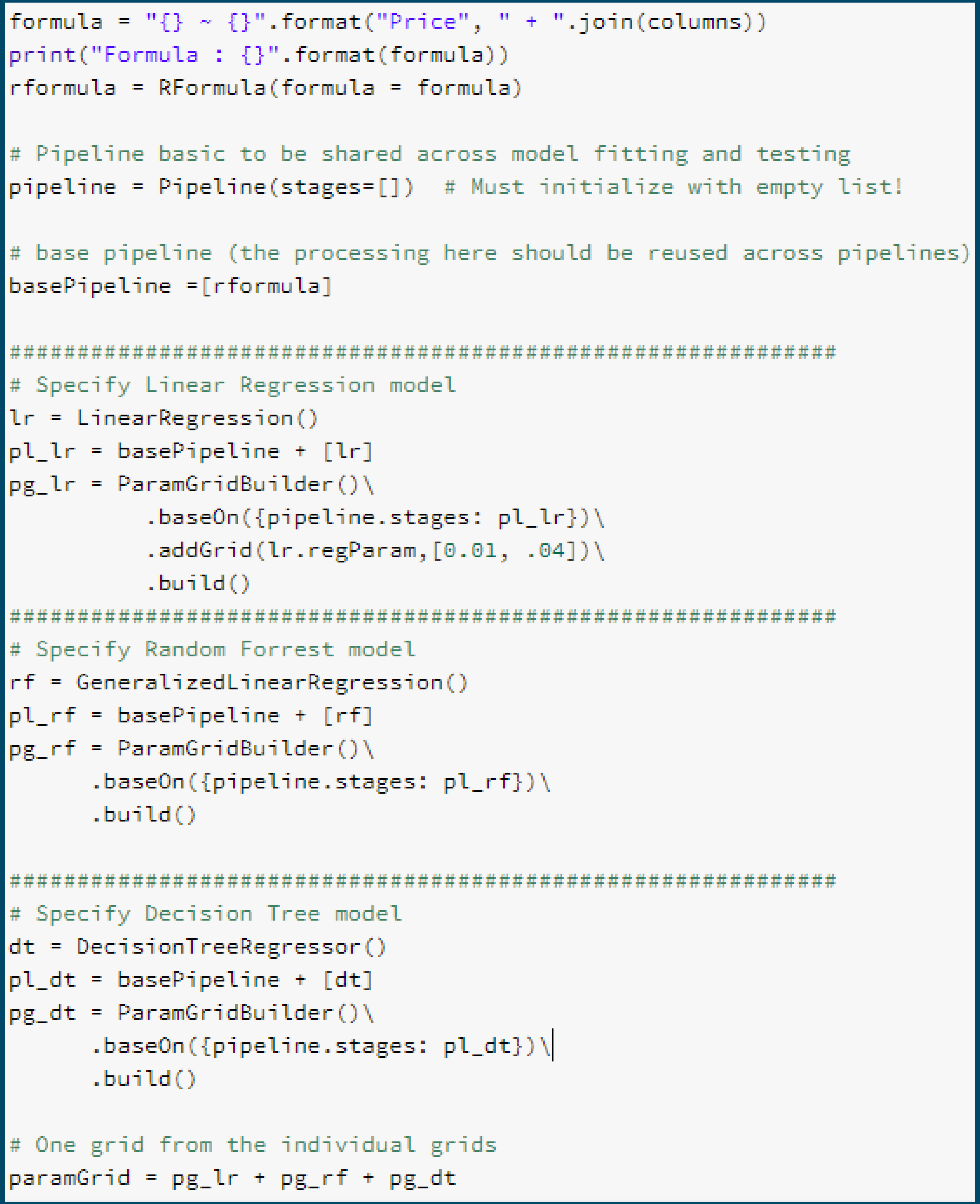
```
lr = LinearRegression()
pl_lr = basePipeline + [lr]
pg_lr = ParamGridBuilder()\
            .baseOn({pipeline.stages: pl_lr})\
            .addGrid(lr.regParam,[0.01, .04])\
            .build()
```

RandomForest model (not shown)

- Finally append all the parameter grids

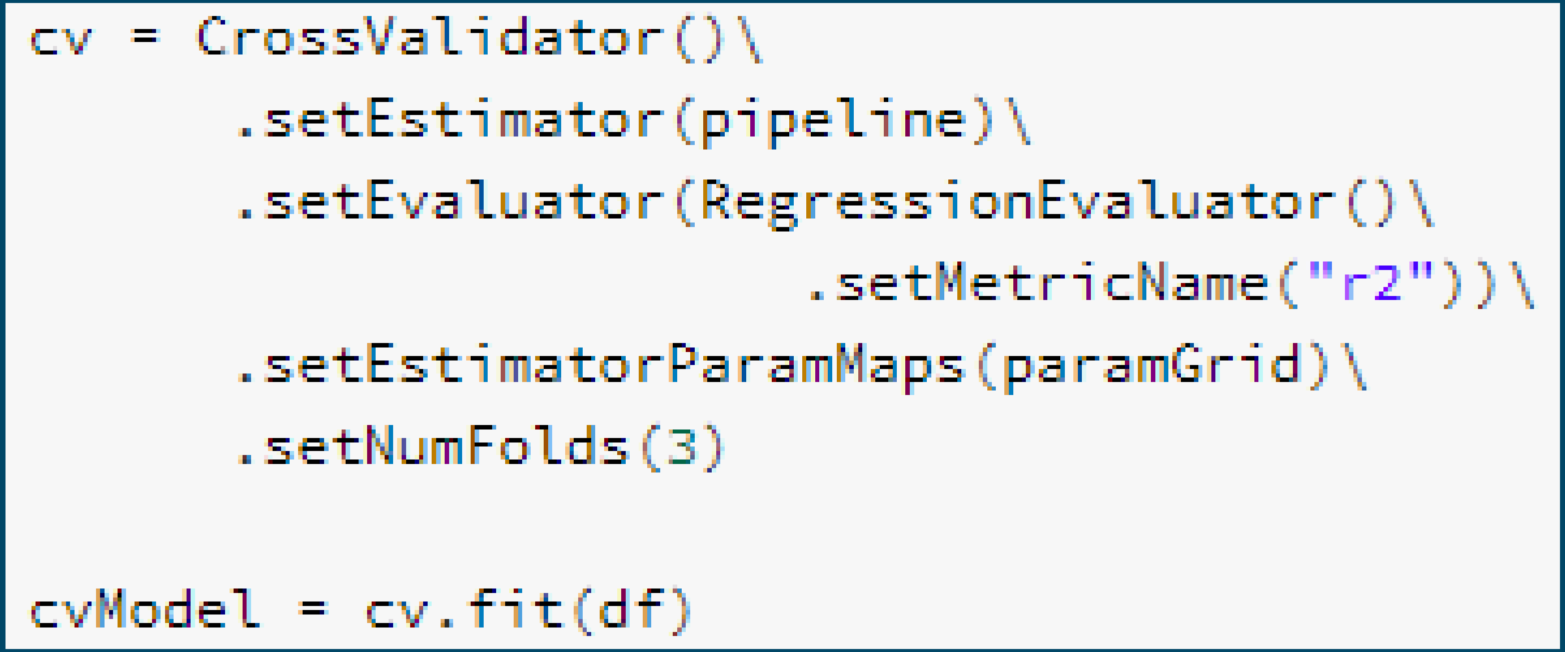


```
# One grid from the individual grids
paramGrid = pg_lr + pg_rf + pg_dt
```



```python
formula = "{} ~ {}".format("Price", " + ".join(columns))
print("Formula : {}".format(formula))
rformula = RFormula(formula = formula)

# Pipeline basic to be shared across model fitting and testing
pipeline = Pipeline(stages=[])  # Must initialize with empty list!

# base pipeline (the processing here should be reused across pipelines)
basePipeline =[rformula]

#############################################################
# Specify Linear Regression model
lr = LinearRegression()
pl_lr = basePipeline + [lr]
pg_lr = ParamGridBuilder()\
          .baseOn({pipeline.stages: pl_lr})\
          .addGrid(lr.regParam,[0.01, .04])\
          .build()
#############################################################
# Specify Random Forrest model
rf = GeneralizedLinearRegression()
pl_rf = basePipeline + [rf]
pg_rf = ParamGridBuilder()\
      .baseOn({pipeline.stages: pl_rf})\
      .build()

#############################################################
# Specify Decision Tree model
dt = DecisionTreeRegressor()
pl_dt = basePipeline + [dt]
pg_dt = ParamGridBuilder()\
      .baseOn({pipeline.stages: pl_dt})\
      .build()

# One grid from the individual grids
paramGrid = pg_lr + pg_rf + pg_dt
```

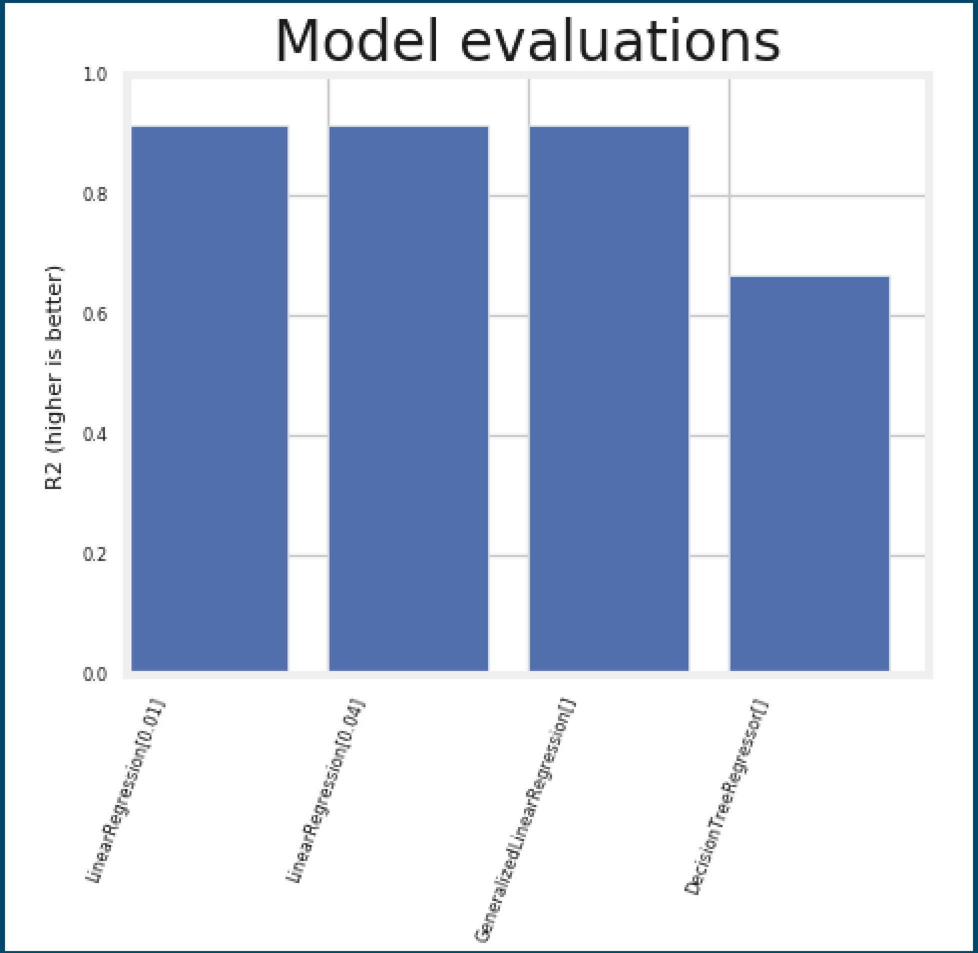
Now, run CrossValidator on the multiple model, multiple parameter pipeline



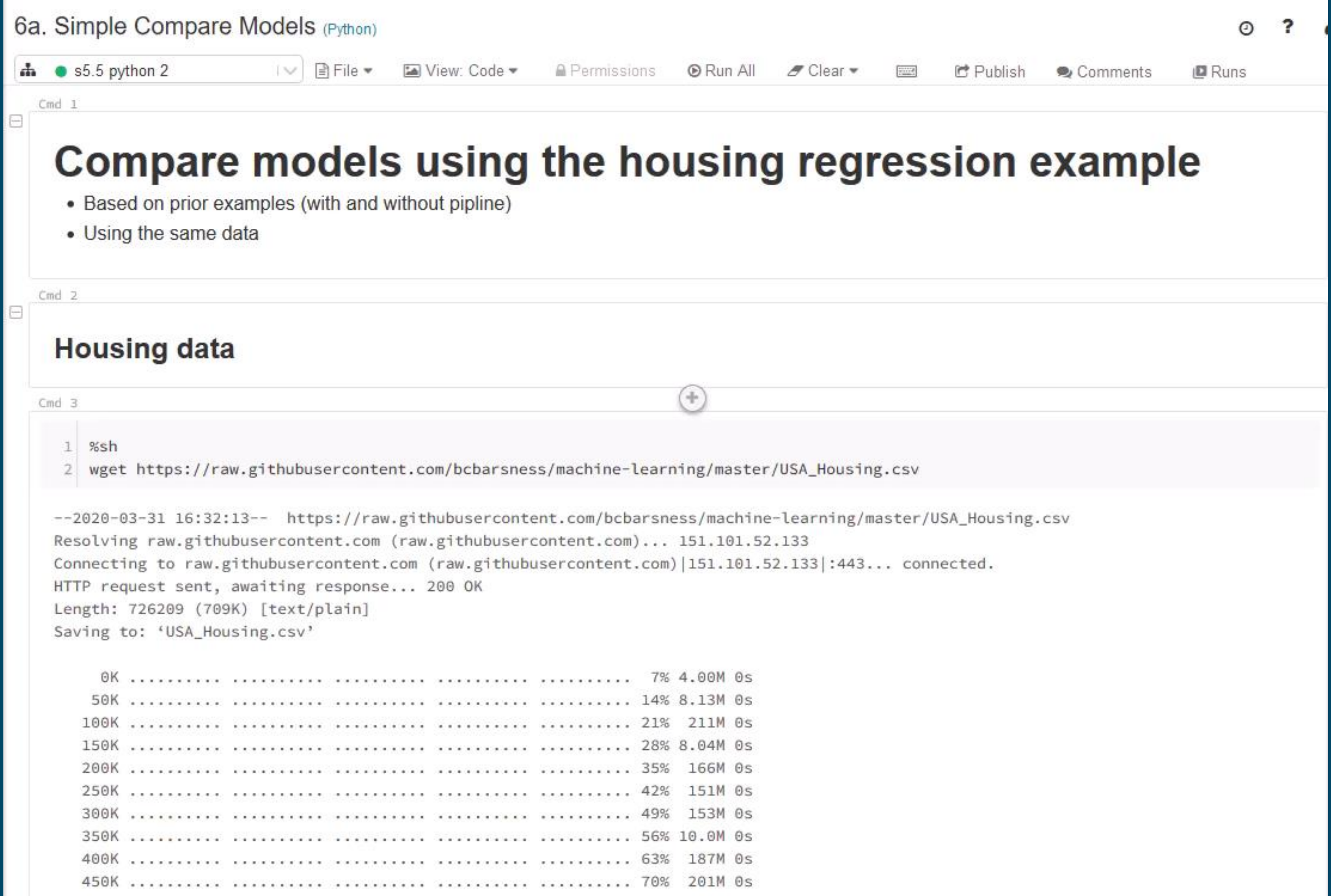
```
cv = CrossValidator()\
      .setEstimator(pipeline)\
      .setEvaluator(RegressionEvaluator()\
                         .setMetricName("r2"))\
      .setEstimatorParamMaps(paramGrid)\
      .setNumFolds(3)

cvModel = cv.fit(df)
```

Finally, get your best model

```
cvModel.getEstimatorParamMaps()[ np.argmax(cvModel.avgMetrics) ]
```



Example in Databricks



Compare models using the housing regression example

- Based on prior examples (with and without pipline)
- Using the same data

Housing data

```
%sh
wget https://raw.githubusercontent.com/bcbarsness/machine-learning/master/USA_Housing.csv
```

```
--2020-03-31 16:32:13--  https://raw.githubusercontent.com/bcbarsness/machine-learning/master/USA_Housing.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.52.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.52.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 726209 (709K) [text/plain]
Saving to: ‘USA_Housing.csv’

     0K .......... .......... .......... .......... ..........  7% 4.00M 0s
    50K .......... .......... .......... .......... .......... 14% 8.13M 0s
   100K .......... .......... .......... .......... .......... 21%  211M 0s
   150K .......... .......... .......... .......... .......... 28% 8.04M 0s
   200K .......... .......... .......... .......... .......... 35%  166M 0s
   250K .......... .......... .......... .......... .......... 42%  151M 0s
   300K .......... .......... .......... .......... .......... 49%  153M 0s
   350K .......... .......... .......... .......... .......... 56% 10.0M 0s
   400K .......... .......... .......... .......... .......... 63%  187M 0s
   450K .......... .......... .......... .......... .......... 70%  201M 0s
```

Important to remember

- Hyperparameter optimization
  - uses grid search to exhaustively search through a specified subset of the total hyperparameter space of a learning algorithm
  - it is be guided by a performance metric, which is used to rank order the parameterized models
  - In PySpark, we use three classes
    - Pipeline, ParamGridBuilder, CrossValidator