By the end of this activity, you will be able to perform the following in Spark:

1. Generate a categorical variable from a numeric variable
2. Aggregate the features into one single column
3. Randomly split the data into training and test sets
4. Create a decision tree classifier to predict days with low humidity.

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

Step 0. **Reconfigure number of CPUs in Cloudera VM.** For this hands on exercise, we need at least 2 virtual CPUs for the Cloudera VM.

Step 1. **Open Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/RCneZE7PEeaqTxIkdCEfsw_c491f272226b35805e44abef7a7a22a9_browser-icon.png?expiry=1576195200000&hmac=IgnSjtD4aVW3sZ0hffsGeaSFt5B-nYbc6mUl6I3JYOc

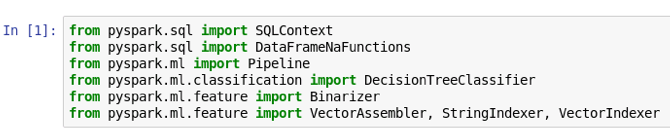
Navigate to *localhost:8889/tree/Downloads/big-data-4*:

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/9Zu58oqhEeaKKwpaECzIKQ_361b99533aaa8d7cde3e3df56b69b3f5_browser.png?expiry=1576195200000&hmac=I-Eqw1LRaxwPY03e_qgglpjpnwQeIS9rPNwRjk6s-0I

Open the handling classification notebook by clicking on *classification.ipynb:*

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/2kmUBosjEeav3w41xGA1rQ_d0033fb9a6418c93bd2462cb45b0c5bb_notebook.png?expiry=1576195200000&hmac=WCLqUFksH0OZgRpfqsYDcIrMaG2Dp1TqBXM7BkpYoDw

Step 2. **Load classes and data.** Execute the first cell in the notebook to load the classes used for this exercise.



Next, execute the second cell which loads the weather data into a DataFrame and prints the columns.



Execute the third cell, which defines the columns in the weather data we will use for the decision tree classifier.



Step 3. **Drop unused and missing data.** We do not need the *number* column in our data, so let's remove it from the DataFrame:

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/x9sFGYshEea8gwpyjKjbvQ_b26fd7010d56f823fb219c12f7eebc0f_drop-number.png?expiry=1576195200000&hmac=5iJPtKjBP6WIr8M_hyqOSYaXNpKWjwI24VQhF8WqFbg

Next, let's remove all rows with missing data:

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/zan9gIshEeaDbw65org3Dw_aa3928798858a82430330c67acfff4cc_drop-missing.png?expiry=1576195200000&hmac=SMvA1oy_BbD9vZsgrabJDYeyN5N1PaRwH5EGuphUjSo

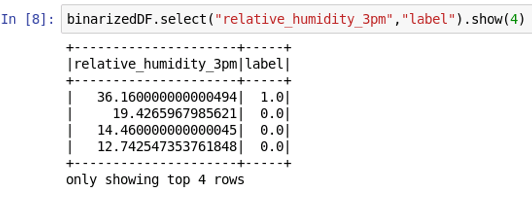
We can print the number of rows and columns in our DataFrame:

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/1oHvNIshEeaTPwpQDCu1cw_6627bb1343890e81acb3fd93384fa377_count-rows-cols.png?expiry=1576195200000&hmac=-ccAcRQzZccQ3riTQl9gtobBVkUiRaQ9FNGI7jeXDJ0

Step 4.**Create categorical variable.** Let's create a categorical variable to denote if the humidity is not low. If the value is less than 25%, then we want the categorical value to be 0, otherwise the categorical value should be 1. We can create this categorical variable as a column in a DataFrame using *Binarizer:*

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/5mtFQoshEeaDbw65org3Dw_1c28f0fedefb0316dfa952e31ce3708d_create-binarized.png?expiry=1576195200000&hmac=UlFSbKou8mtXD2EDx-BVFiCO32ACCnj-VHu0sslIdG0

The *threshold* argument specifies the threshold value for the variable, *inputCol* is the input column to read, and *outputCol* is the name of the new categorical column. The second line applies the *Binarizer* and creates a new DataFrame with the categorical column. We can look at the first four values in the new DataFrame:



The first row's humidity value is greater than 25% and the label is 1. The other humidity values are less than 25% and have labels equal to 0.

Step 5. **Aggregate features.** Let's aggregate the features we will use to make predictions into a single column:

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/DVk9tosiEeaTPwpQDCu1cw_dc8f891a8aedd17ed7188c097e29b6ce_create-agg-features.png?expiry=1576195200000&hmac=E-35tMsX90wCWQ5tnNYQGJh71ko5QhPneAVOmHfEofE

The *inputCols* argument specifies our list of column names we defined earlier, and *outputCol* is the name of the new column. The second line creates a new DataFrame with the aggregated features in a column.

Step 6.**Split training and test data.** We can split the data by calling *randomSplit():*

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/Gc4tNosiEeav3w41xGA1rQ_fb81ccc9d382c62509efd5e0f867485d_create-training-test.png?expiry=1576195200000&hmac=UN5tI-l2Pn5L1Jk5ydKETgayqTRLpCh7_S47S_RWQF4

The first argument is how many parts to split the data into and the *approximate*size of each. This specifies two sets of 80% and 20%. Normally, the seed should not be specified, but we use a specific value here so that everyone will get the same decision tree.

We can print the number of rows in each DataFrame to check the sizes (1095 \* 80% = 851.2):

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/Lc4cTYsiEea8gwpyjKjbvQ_334d0e49db6fc20b5d8fb84cfd7997e7_num-training-test.png?expiry=1576195200000&hmac=_FtyWwlIHpMd1hDExLMY08XY8t8IF_P6zy5pB5b4V_8

**NOTE:** if you get values (859, 205), then your Cloudera VM is most likely configured to only using 1 CPU. You need to reconfigure the VM to use 2 CPUs.

Step 7. **Create and train decision tree.** Let's create the decision tree:



The *labelCol* argument is the column we are trying to predict, *featuresCol* specifies the aggregated features column, *maxDepth* is stopping criterion for tree induction based on maximum depth of tree, *minInstancesPerNode* is stopping criterion for tree induction based on minimum number of samples in a node, and *impurity* is the impurity measure used to split nodes.

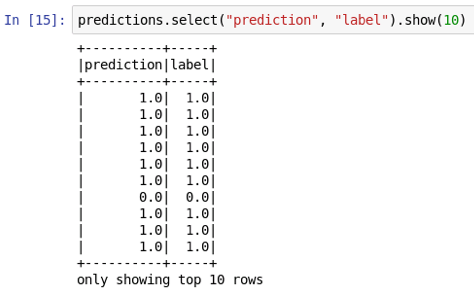
We can create a model by training the decision tree. This is done by executing it in a *Pipeline:*

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/_96pOYsiEeaghhLNmbkUnw_fa5f04ba7eb20ca73b72d715e17d25d7_train-model.png?expiry=1576195200000&hmac=uP0MPj0qv5t5KRFlrOCCDNRYqY761s-DuYYQASC-Hr4

Let's make predictions using our test data set:

https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/TVoEx4sjEea8gwpyjKjbvQ_1ad9a1d87cbe2a9fc5a4d7623e8fafff_create-predictions.png?expiry=1576195200000&hmac=EXBDtjytC2YPYyxy9QvGY13iAf0dT-Lb3Tc5JqHwDJM

Looking at the first ten rows in the prediction, we can see the prediction matches the input:



Step 8. **Save predictions to CSV.** Finally, let's save the predictions to a CSV file. In the next Spark hands-on activity, we will evaluate the accuracy.

Let's save only the *prediction*and *label* columns to a CSV file:

