

# Getting started with Spark on Azure HDInsight



# Contents

Overview .....	3
Provision an HDInsight Spark Cluster.....	5
Run interactive Spark SQL queries using Zeppelin .....	9
Run interactive Spark SQL queries using Jupyter .....	13
Writing a Machine Learning App using Spark MLlib.....	17
Terms of Use.....	22

# Overview

## Summary

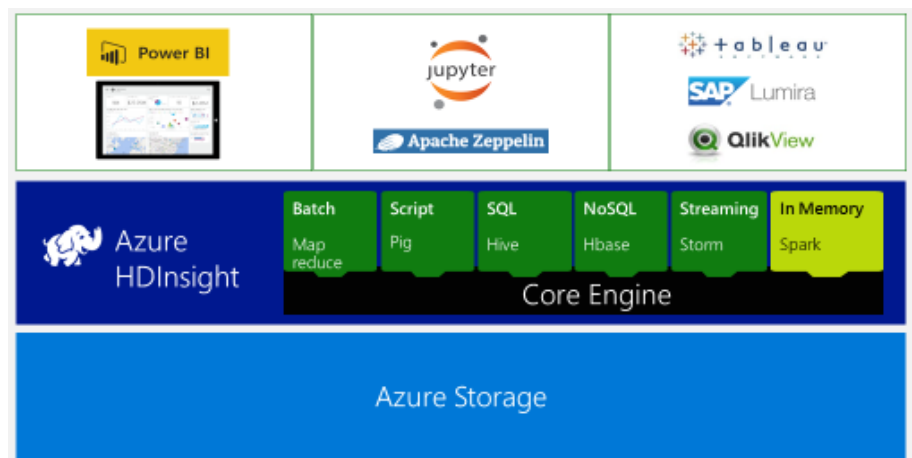
This lab is intended to serve as a general introduction to Spark on HDInsight. We will be covering:

- Provisioning an Apache Spark cluster on HDInsight
- Using web-based notebooks Zeppelin (Scala/Spark SQL) and Jupyter (Python) to run Spark SQL queries on the cluster
- Writing machine learning algorithms using MLlib (Python)

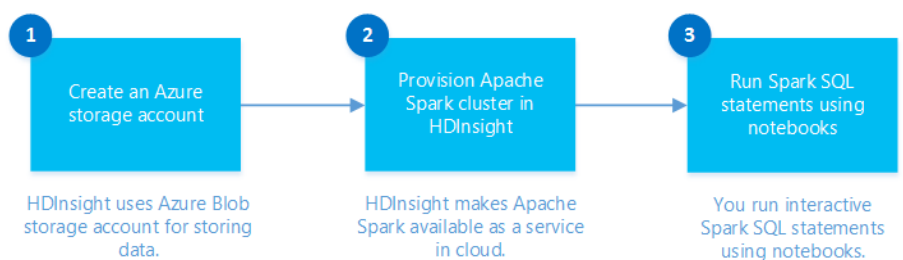
Apache Spark is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Spark's in-memory computation capabilities make it a good choice for iterative algorithms in machine learning and graph computations.

Spark can also be used to perform conventional disk-based data processing. Spark improves the traditional MapReduce framework by avoiding writes to disk in the intermediate stages. Also, Spark is compatible with the Hadoop Distributed File System (HDFS) and Azure Blob storage so the existing data can easily be processed via Spark.

To interact with Spark there are several options including Power BI, Jupyter, Zeppelin and 3<sup>rd</sup> party BI tools (see graphic below).



In the first part of this lab we are going to create an Azure storage account, provision an Apache Spark Cluster in HDInsight and then run some Spark SQL statements using notebooks.



### What is Apache Zeppelin?

Apache Zeppelin is a web-based notebook that enables interactive data analytics. The Zeppelin interpreter allows any language/data-processing-backend to be plugged in. However, current languages included in the Zeppelin interpreter are:

- Scala (with Apache Spark)
- SparkSQL
- Markdown
- Shell

Zeppelin notebooks hosted on the HDInsight Spark Cluster supports Scala and SparkSQL. In this lab we will focus on using Zeppelin for SparkSQL.

Please take a moment to read more details on the [Apache Zeppelin website](#).

### What is Jupyter?

Jupyter is another web-based notebook that enables interactive data analytics. Original Jupyter got its name because it was meant as a notebook frontend for the Ju(lia)Pyt(hon)eR programming languages. Today, Jupyter supports over 40 programming languages.

Jupyter notebooks on the HDInsight Spark cluster support Python. In this lab we will leverage the pyspark package to run Spark SQL queries.

Please take a moment to read more details on the [Jupyter website](#).

### What is MLlib?

MLlib is Apache Spark's scalable machine learning library. MLlib fits into Spark's APIs and interoperates with NumPy in Python (starting in Spark 0.9). You can use any Hadoop data source (e.g. HDFS, HBase, or local files), or Azure blob storage.

MLlib contains the following algorithms: logistic regression, SVM, decision/regression trees, random forest, kmeans clustering, LDA, SVD and QR decomposition, PCA, elastic-net regularization, naïve Bayes, sequential pattern mining, summary stats & hypothesis testing, feature transformations, model evaluation and hyper-parameter tuning.

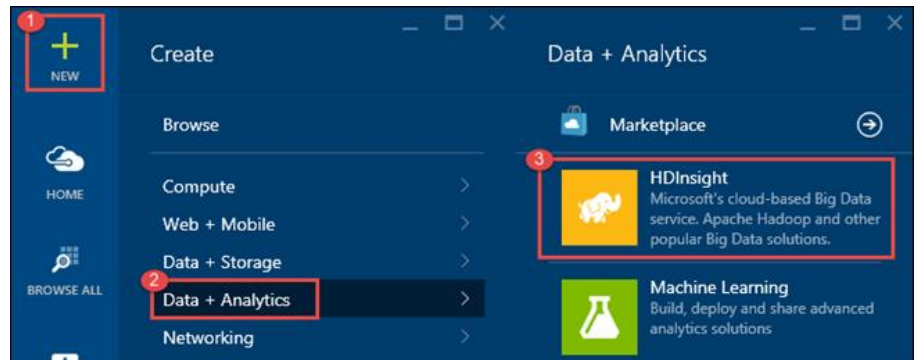
Refer to the [MLlib guide](#) for detailed worked examples.

### Lab Requirements/Prerequisites

Before you begin this tutorial, you must have an Azure subscription. See [Get Azure free trial](#).

# Provision an HDInsight Spark Cluster

1. Sign in to the Azure preview portal.
2. Click **NEW**, click **Data + Analytics**, and then click **HDInsight**.



3. Enter a **Cluster Name**, select Hadoop for the Cluster Type, and from the **Cluster Operating System** drop-down menu, select Windows Server 2012 R2 Datacenter. A green check appears beside the cluster name if it is available.

New HDInsight Cluster

Cluster Name  
mysparkcluster ✓  
.azurehdinsight.net

Cluster Type ⓘ  
Spark (PREVIEW) ▼

Cluster Operating System  
Windows Server 2012 R2 Datacenter ▼

4. If you have more than one subscription, click the **Subscription** entry to select the Azure subscription to use for the cluster.
5. Click **Resource Group** to see a list of existing resource groups and select where to create the cluster. Or, you can click **Create New** and then enter the name of the new resource group. A green check appears to indicate if the new group name is available.
6. Click **Credentials**, and then enter a **Cluster Login Username** and **Cluster Login Password**. If you want to enable Remote Desktop on the cluster node, for Enable Remote Desktop, click Yes, and then specify the required values. This tutorial

does not require remote desktop so you can **skip this**. Click **Select** at the bottom to save the credentials configuration.

## Cluster Credentials

Create login and remote access credentials for the cluster.

Cluster Login Username ⓘ

admin

Cluster Login Password

••••••••✓

Confirm Password

••••••••✓

Enable Remote Desktop?

NO


YES


Select

- Click **Data Source** to choose an existing data source for the cluster, or **create a new one**. When you provision a Hadoop cluster in HDInsight, you specify an Azure Storage account. A specific Blob storage container from that account is designated as the default file system, like in the Hadoop distributed file system (HDFS). By default, the HDInsight cluster is provisioned in the same data center as the storage account you specify. For more information, see [Use Azure Blob storage with HDInsight](#).


**Data Source**

The cluster will use this data source as the primary location for most data access, such as job input and log output.

Selection Method 


From all subscriptions 

---

Select storage account  
hdistorage (South Central US) 


[Or Create New](#)

---

Choose Default Container 

myhadoopcluster

---

Location  
South Central US 

**Select**

Currently you can select an Azure Storage Account as the data source for an HDInsight cluster. Use the following to understand the entries on the Data Source blade.

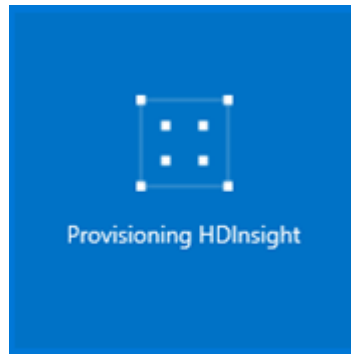
- **Selection Method:** Set this to **From all subscriptions** to enable browsing of storage accounts from all your subscriptions. Set this to **Access Key** if you want to enter the **Storage Name** and **Access Key** of an existing storage account.
- **Select storage account / Create New:** Click **Select storage account** to browse and select an existing storage account you want to associate with the cluster. Or, click **Create New** to create a new storage account. Use the field that appears to enter the name of the storage account. A green check appears if the name is available.
- **Choose Default Container:** Use this to enter the name of the default container to use for the cluster. While you can enter any name here, we recommend using the same name as the cluster so that you can easily recognize that the container is used for this specific cluster.
- **Location:** The geographic region that the storage account is in, or will be created in.

**IMPORTANT:** Selecting the location for the default data source also sets the location of the HDInsight cluster. The cluster and default data source must be located in the same region.

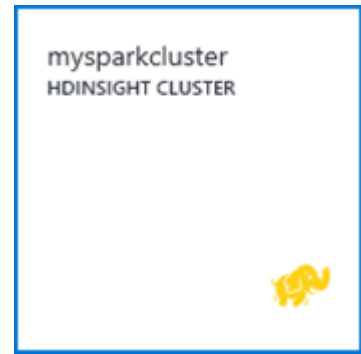
8. Click **Select** to save the data source configuration.

9. By default, the size of the cluster is 4 worker nodes and 2 head nodes. The estimated cost of this cluster is approximately 80pence per hour per node. You can reduce the size of the cluster to 2 worker nodes (head node size cannot be reduced) without affecting the ability to run this lab.
10. On the **New HDInsight Cluster** blade, ensure that Pin to Startboard is selected, and then click Create. This creates the cluster and adds a tile for it to the Startboard of your Azure portal. The icon will indicate that the cluster is provisioning, and will change to display the HDInsight icon once provisioning has completed.

WHILE PROVISIONING



PROVISIONING COMPLETE



**NOTE:** It will take some time for the cluster to be created, usually around 15 minutes. Use the tile on the Startboard, or the **Notifications** entry on the left of the page to check on the provisioning process.

11. Once the provisioning is complete, click the tile for the Spark cluster from the Startboard to launch the cluster blade.



# Run interactive Spark SQL queries using Zeppelin

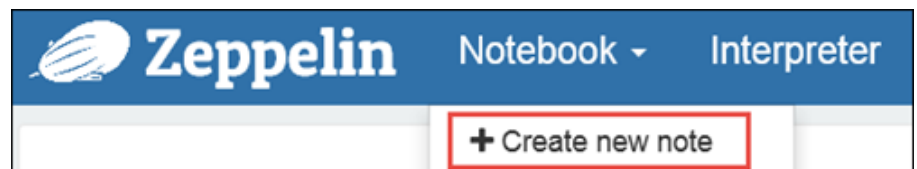
After you have provisioned a cluster, you can use a web-based Zeppelin notebook to run Spark SQL interactive queries against the Spark HDInsight cluster. In this section, we will use a sample data file (hvac.csv) available by default on the cluster to run some interactive Spark SQL queries.

1. From the Azure Preview Portal, from the startboard, click the tile for your Spark cluster (if you pinned it to the startboard). You can also navigate to your cluster under **Browse All > HDInsight Clusters**.
2. From the Spark cluster blade, click **Quick Links**, and then from the Cluster Dashboard blade, click **Zeppelin Notebook**. If prompted, enter the admin credentials for the cluster.

**NOTE:** You may also reach the Zeppelin Notebook for your cluster by opening the following URL in your browser. Replace CLUSTERNAME with the name of your cluster:

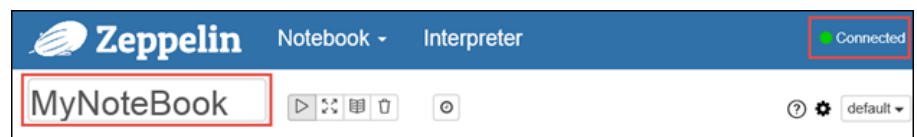
<https://CLUSTERNAME.azurehdinsight.net/zeppelin>

3. Create a new notebook. From the header pane, click **Notebook**, and then click **Create New Note**.



On the same page, under the Notebook heading, you should see a new notebook with the name starting with **Note XXXXXXXXXX**. Click the new notebook.

4. On the webpage for the new notebook, click the heading, and change the name of the notebook if you want to. Press **ENTER** to save the name change. Also, make sure the notebook header shows a **Connected** status in the top-right corner.



5. Load sample data into a temporary table. When you provision a Spark cluster in HDInsight, the sample data file, **hvac.csv**, is copied to the associated storage account under **\HdiSamples\SensorSampleData\hvac**. You can view the directory via the **HDInsight Cluster Dashboard**.

The data contains the following variables:

Date	Time	TargetTemp	ActualTemp	System	SystemAge	BuildingID
06/01/2013	00:00:01	66	58	13	20	4
06/02/2013	01:00:01	69	68	3	20	17
06/03/2013	02:00:01	70	73	17	20	18
06/04/2013	03:00:01	67	63	2	23	15
06/05/2013	04:00:01	68	74	16	9	3
06/06/2013	05:00:01	67	56	13	28	4

**Note that line feeds may not be preserved if you copy-and-paste. Therefore, we have provided snippets in txt files – you can find these in the lab folder.**

```
// Create an RDD using the default Spark context, sc
val hvacText =
sc.textFile("wasb:///HdiSamples/SensorSampleData/hvac/HVAC.csv")

// Define a schema
case class Hvac(date: String, time: String, targettemp: Integer,
actualtemp: Integer, buildingID: String)

// Map the values in the .csv file to the schema
val hvac = hvacText.map(s => s.split(",")).filter(s => s(0) !=
>Date").map(
    s => Hvac(s(0),
        s(1),
        s(2).toInt,
        s(3).toInt,
        s(6)
    )
).toDF()

// Register as a temporary table called "hvac"
hvac.registerTempTable("hvac")
```

In this example, it is worth noting that the target address for the blob storage is used by default, i.e. there is no need to explicitly specify:

```
val hvacText =
sc.textFile("wasb://<<yourClusterName@YourAccount.blob.core.w
indows.net>>HdiSamples/SensorSampleData/hvac/HVAC.csv")
```

Press **SHIFT + ENTER** on the keyboard or click the **Play** button for the paragraph to run the code. The status in the upper right corner of the paragraph should progress from READY, PENDING, RUNNING to FINISHED. The output appears at the bottom of the same paragraph. The screenshot looks like the following:

```
Load Data Into Table
// Create an RDD using the default Spark context, sc
val hvacText = sc.textFile("wasb:///HdiSamples/SensorSampleData/hvac/HVAC.csv")

// Define a schema
case class Hvac(date: String, time: String, targettemp: Integer, actualtemp: Integer, buildingID: String)

// Map the values in the .csv file to the schema
val hvac = hvacText.map(s => s.split(",")).filter(s => s(0) != "Date").map(
  s => Hvac(s(0),
    s(1),
    s(2).toInt,
    s(3).toInt,
    s(6)
  )
).toDF()

// Register as a temporary table called "hvac"
hvac.registerTempTable("hvac")

hvacText: org.apache.spark.rdd.RDD[String] = wasb:///HdiSamples/SensorSampleData/hvac/HVAC.csv MapPartitionsRDD[1] at textFile at <console>:24
defined class Hvac
hvac: org.apache.spark.sql.DataFrame = [date: string, time: string, targettemp: int, actualtemp: int, buildingID: string]
Took 36 seconds
```

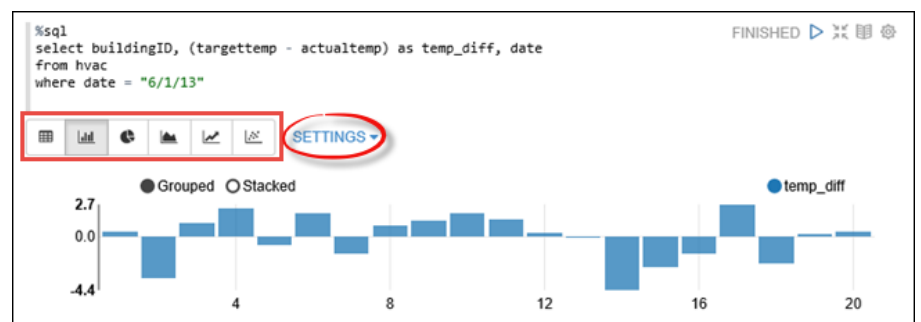
You can also provide a title to each paragraph. From the right-hand corner, click the **Settings** icon, and then click **Show title**.

6. You can now run Spark SQL statements on the hvac table. Write the following query in a new paragraph. The query retrieves the building ID and the difference between the target and actual temperatures for each building on a given date. Press **SHIFT + ENTER**.

```
%sql
SELECT buildingID, (targettemp - actualtemp) AS temp_diff, date
FROM hvac
WHERE date = "6/1/13"
```

The **%sql** statement at the beginning tells the notebook to use the Spark SQL interpreter. You can look at the defined interpreters from the Interpreter tab in the notebook header.

The following screenshot shows the output:



Click the display options (highlighted in rectangle) to switch between different representations for the same output. Click **Settings** to choose what constitutes the key and values in the output. The screen capture above uses **buildingID** as the key and the average of **temp\_diff** as the value.

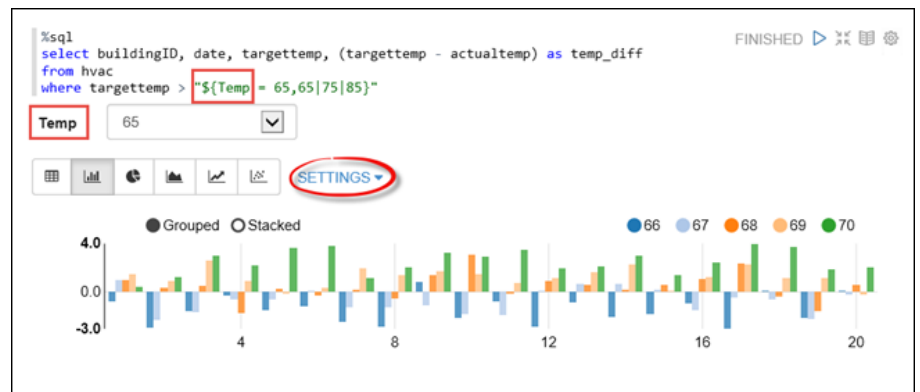
7. You can also run Spark SQL statements using variables in the query. The next code example shows how to define a variable, **Temp**, in the query with the possible values you want to query with. When you first run the query, a drop-down

is automatically populated with the values you specified for the variable.

```
%sql
select buildingID, date, targettemp, (targettemp - actualtemp) as
temp_diff
from hvac
where targettemp > "${Temp = 65,65|75|85}"
```

Write the above code example in a new paragraph and press **SHIFT + ENTER**.

For subsequent queries, you can select a new value from the drop-down and run the query again. Click **Settings** to choose what constitutes the key and values in the output. The screen capture below uses buildingID as the key, the average of temp\_diff as the value, and targettemp as the group.



- Restart the Spark SQL interpreter to exit the application. Click the **Interpreter** tab at the top, and for the Spark interpreter, click **Restart**.

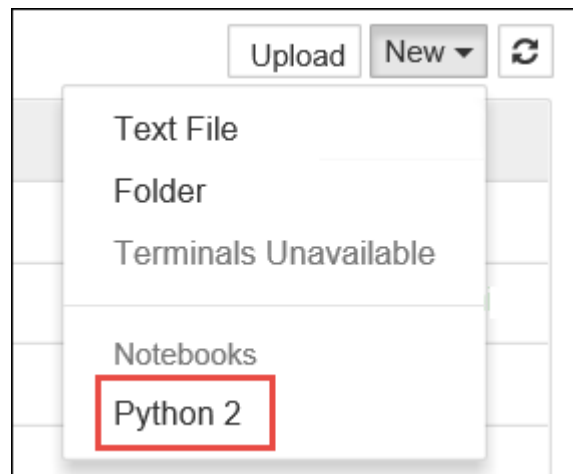
# Run interactive Spark SQL queries using Jupyter

1. From the Azure Preview Portal, from the startboard, click the tile for your Spark cluster (if you pinned it to the startboard). You can also navigate to your cluster under **Browse All > HDInsight Clusters**.
2. From the Spark cluster blade, click **Quick Links**, and then from the **Cluster Dashboard** blade, click **Jupyter Notebook**. If prompted, enter the admin credentials for the cluster.

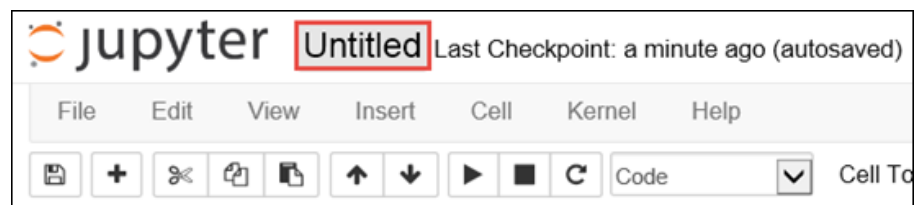
**NOTE:** You may also reach the Jupyter Notebook for your cluster by opening the following URL in your browser. Replace **CLUSTERNAME** with the name of your cluster:

<https://CLUSTERNAME.azurehdinsight.net/jupyter>

3. Create a new notebook. Click **New**, and then click **Python2**.



4. A new notebook is created and opened with the name 'Untitled.pynb'. Click the notebook name at the top, and enter a friendly name.



5. Import the required modules and create the Spark and SQL contexts. Write the following code example in an empty cell, and then press **SHIFT + ENTER**.

```

from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.sql.types import *

# Create Spark and SQL contexts
sc = SparkContext('spark://headnodehost:7077', 'pyspark')
sqlContext = SQLContext(sc)

```

On the first run, the Spark context is established. On subsequent runs, we don't need to repeat that. If you do run this command multiple times you may receive the following error:

Error

```

-----
ValueError Traceback (most recent call last)
<ipython-input-2-ad7e7ec9ed75> in <module>()
4
5 # Create Spark and SQL contexts
----> 6 sc = SparkContext('spark://headnodehost:7077',
'pyspark')
7 sqlContext = SQLContext(sc)

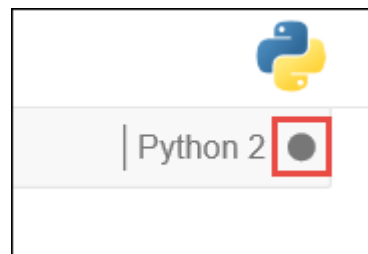
.
.
.

```

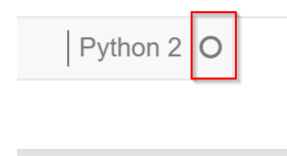
**ValueError: Cannot run multiple SparkContexts at once;**  
existing SparkContext(app=pyspark,  
master=spark://headnodehost:7077) created by \_\_init\_\_ at  
<ipython-input-1-ad7e7ec9ed75>:6

Every time you run a job in Jupyter, your web browser window title will show a **(Busy)** status along with the notebook title. You will also see a solid circle next to the **Python 2** text in the top-right corner. After the job is completed, this will change to a hollow circle.

BUSY



IDLE



- Load sample data into a temporary table. When you provision a Spark cluster in HDInsight, the sample data file, **hvac.csv**, is copied to the associated storage account under **\HdiSamples\SensorSampleData\hvac**.

In an empty cell, paste the following code example and press **SHIFT + ENTER**. This code example registers the data into a temporary table called **hvac**.

```

# Load the data
hvacText =
sc.textFile("wasb:///HdiSamples/SensorSampleData/hvac/HVAC.csv")

# Create the schema
hvacSchema = StructType([StructField("date", StringType(),
False),StructField("time", StringType(),
False),StructField("targettemp", IntegerType(),
False),StructField("actualtemp", IntegerType(),
False),StructField("buildingID", StringType(), False)])

# Parse the data in hvacText
hvac = hvacText.map(lambda s: s.split(",")).filter(lambda s: s[0]
!= "Date").map(lambda s:(str(s[0]), str(s[1]), int(s[2]),
int(s[3]), str(s[6]) ))

# Create a data frame
hvacdf = sqlContext.createDataFrame(hvac,hvacSchema)

# Register the data fram as a table to run queries against
hvacdf.registerAsTable("hvac")

# Run queries against the table and display the data
data = sqlContext.sql("select buildingID, (targettemp -
actualtemp) as temp_diff, date from hvac where date =
\"6/1/13\"")
data.show()

```

7. Once the job is completed successfully, the following output is displayed.

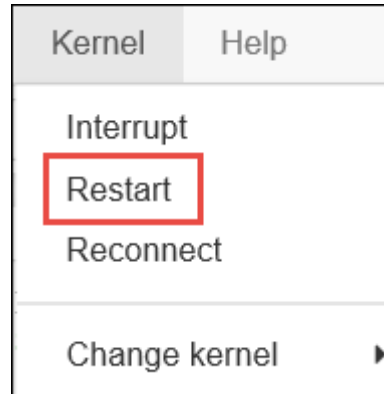
buildingID	temp_diff	date
4	8	6/1/13
3	2	6/1/13
7	-10	6/1/13
12	3	6/1/13
7	9	6/1/13
7	5	6/1/13
3	11	6/1/13
8	-7	6/1/13
17	14	6/1/13
16	-3	6/1/13
8	-8	6/1/13
1	-1	6/1/13
12	11	6/1/13
3	14	6/1/13
6	-4	6/1/13
1	4	6/1/13
19	4	6/1/13
19	12	6/1/13
9	-9	6/1/13
15	-10	6/1/13

In [ ]:

8. You can cast the data into pandas data using:

```
import pandas as pan  
mydata = data.toPandas()
```

9. Restart the kernel to exit the application. From the top menu bar, click **Kernel**, click **Restart**, and then click **Restart** again at the prompt.





# Writing a Machine Learning App using Spark MLlib

1. Continuing with Jupyter, create a new notebook. Click **New**, and then click **Python 2**.
2. A new notebook is created and opened with the name Untitled.pynb. Click the notebook name at the top, and enter a friendly name.



3. Start building your machine learning application. In this application we use a Spark ML pipeline to perform a document classification. In the pipeline, we split the document into words, convert the words into a numerical feature vector, and finally build a prediction model using the feature vectors and labels.

To start building the application, first import the required modules and assign resources to the application. In the empty cell in the new notebook, paste the following snippet, and then press **SHIFT + ENTER**.

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

import os
import sys
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.sql.types import *

from pyspark.mllib.classification import
LogisticRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from numpy import array

# Assign resources to the application
conf = SparkConf()
conf.setMaster('spark://headnodehost:7077')
conf.setAppName('pysparkregression')
conf.set("spark.cores.max", "4")
conf.set("spark.executor.memory", "4g")

sc = SparkContext(conf=conf)
sqlContext = SQLContext(sc)
```

4. You must now load the data (hvac.csv), parse it, and use it to train the model. For this, you define a function that checks whether the actual temperature of the building is greater than the target temperature. If the actual temperature is greater, the building is hot, denoted by the value **1.0**. If the actual temperature is lesser, the building is cold, denoted by the value **0.0**.

Paste the following snippet in an empty cell and press **SHIFT + ENTER**.

```
# List the structure of data for better understanding. Because
the data will be
# loaded as an array, this structure makes it easy to understand
what each element
# in the array corresponds to

# 0 Date
# 1 Time
# 2 TargetTemp
# 3 ActualTemp
# 4 System
# 5 SystemAge
# 6 BuildingID

LabeledDocument = Row("BuildingID", "SystemInfo", "label")

# Define a function that parses the raw CSV file and returns an
object of type LabeledDocument

def parseDocument(line):
    values = [str(x) for x in line.split(',')]
    if (values[3] > values[2]):
        hot = 1.0
    else:
        hot = 0.0

    textValue = str(values[4]) + " " + str(values[5])

    return LabeledDocument((values[6]), textValue, hot)

# Load the raw HVAC.csv file, parse it using the function
data =
sc.textFile("wasb:///HdiSamples/SensorSampleData/hvac/HVAC.csv")

documents = data.filter(lambda s: "Date" not in
s).map(parseDocument)
training = documents.toDF()
```

5. Configure the Spark machine learning pipeline that consists of three stages: tokenizer, hashingTF, and lr. For more information about what is a pipeline and how it works see [Spark machine learning pipeline](#).

Paste the following snippet in an empty cell and press **SHIFT + ENTER**.

```
tokenizer = Tokenizer(inputCol="SystemInfo", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(),
outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

6. Fit the pipeline to the training document. Paste the following snippet in an empty cell and press **SHIFT + ENTER**.

```
model = pipeline.fit(training)
```

7. Verify the training document to checkpoint your progress with the application. Paste the following snippet in an empty cell and press **SHIFT + ENTER**.

```
training.show()
```

This should give the output similar to the following:

```
In [5]: training.show()
```

BuildingID	SystemInfo	label
4	13 20	0.0
17	3 20	0.0
18	17 20	1.0
15	2 23	0.0
3	16 9	1.0
4	13 28	0.0
2	12 24	0.0
16	20 26	1.0
9	16 9	1.0
12	6 5	0.0
15	10 17	1.0
7	2 11	0.0
15	14 2	1.0
6	3 2	0.0
20	19 22	0.0
8	19 11	0.0
6	15 7	0.0
13	12 5	0.0
4	8 22	0.0
7	17 5	0.0

Go back and verify the output against the raw CSV file. For example, the first row the CSV file has this data:

	A	B	C	D	E	F	G
1	Date	Time	TargetTemp	ActualTemp	System	SystemAge	BuildingID
2	6/1/2013	0:00:01	66	58	13	20	4

Notice how the actual temperature is less than the target temperature suggesting the building is cold. Hence in the training output, the value for **label** in the first row is **0.0**, which means the building is not hot.

8. Prepare a data set to run the trained model against. To do so, we would pass on a system ID and system age (denoted as **SystemInfo** in the training output), and the model would predict whether the building with that system ID and system age would be hotter (denoted by 1.0) or cooler (denoted by 0.0).

Paste the following snippet in an empty cell and press **SHIFT + ENTER**.

```
# SystemInfo here is a combination of system ID followed by
system age
Document = Row("id", "SystemInfo")
test = sc.parallelize([(1L, "20 25"),
                      (2L, "4 15"),
                      (3L, "16 9"),
                      (4L, "9 22"),
                      (5L, "17 10"),
                      (6L, "7 22")]) \
    .map(lambda x: Document(*x)).toDF()
```

9. Finally, make predictions on the test data. Paste the following snippet in an empty cell and press **SHIFT + ENTER**.

```
# Make predictions on test documents and print columns of
interest
prediction = model.transform(test)
selected = prediction.select("SystemInfo", "prediction",
                             "probability")
for row in selected.collect():
    print row
```

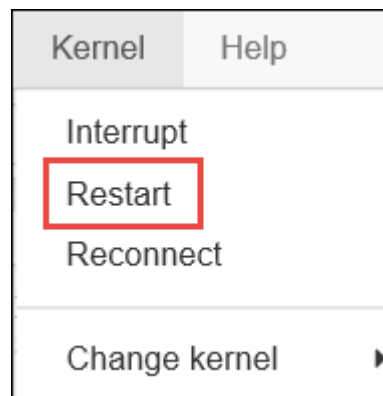
10. You should see an output similar to the following:

```
In [8]: # Make predictions on test documents and print columns of interest
prediction = model.transform(test)
selected = prediction.select("SystemInfo", "prediction", "probability")
for row in selected.collect():
    print row

Row(SystemInfo=u'20 25', prediction=1.0, probability=DenseVector([0.4999, 0.5001]))
Row(SystemInfo=u'4 15', prediction=0.0, probability=DenseVector([0.5016, 0.4984]))
Row(SystemInfo=u'16 9', prediction=1.0, probability=DenseVector([0.4785, 0.5215]))
Row(SystemInfo=u'9 22', prediction=1.0, probability=DenseVector([0.4549, 0.5451]))
Row(SystemInfo=u'17 10', prediction=1.0, probability=DenseVector([0.4925, 0.5075]))
Row(SystemInfo=u'7 22', prediction=0.0, probability=DenseVector([0.5015, 0.4985]))
```

From the first row in the prediction, you can see that for an HVAC system with ID 20 and system age of 25 years, the building will be hot (prediction=1.0). The first value for DenseVector (0.49999) corresponds to the prediction 0.0 and the second value (0.5001) corresponds to the prediction 1.0. In the output, even though the second value is only marginally higher, the model shows prediction=1.0.

11. You can now exit the notebook by restarting the kernel. From the top menu bar, click **Kernel**, click **Restart**, and then click **Restart** again at the prompt.



# Terms of Use

© 2015 Microsoft Corporation. All rights reserved.

By using this Hands-on Lab, you agree to the following terms:

The technology/functionality described in this Hands-on Lab is provided by Microsoft Corporation in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the Hands-on Lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this Hands-on Lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCTIONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

**FEEDBACK.** If you give feedback about the technology features, functionality and/or concepts described in this Hands-on Lab to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

## DISCLAIMER

This lab contains only a portion of the features and enhancements in Microsoft Azure. Some of the features might change in future releases of the product.