Objective

The goal of this section is to learn about dataframes API, SparkSQL and MLlib on Apache Spark.

Getting Started with Dataframes

1. First create spark context:

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
```

2. Import data: Data can be imported from local filesystem, HDFS or S3

```
# Directly import a python list as a RDD
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)

# Read a file from s3
lines = sc.textFile("s3n://csed516/gburg.txt")
```

   Note: the change the bucket name and key above to a s3 bucket and file in your account.

3. RDD to list and back to RDD

```
# Python list -> RDD
Data = [1,2,3,4,5,6]
print(type(Data))
dataRDD = sc.parallelize(Data)
print(type(dataRDD))

# RDD -> list
data = dataRDD.collect()
print(type(dataRDD))
```

4. Simple map and reduce operations on RDD

```
# Map operation to retrieve length of each line
lineLengths = lines.map(lambda s: len(s))

# Reduce operation to add all of the lengths to get the length of the entire document.
totalLength = lineLengths.reduce(lambda a, b: a + b)
print(totalLength)
```

5. Python UDF in Spark

```
# Define a UDF in Python and run it in Spark
def udf(s):
    words = s.split(" ")
    return len(words)

# Import text file from s3 to a RDD.
lines = sc.textFile("s3n://csed516/gburg.txt")

# Use the UDF in map instead of an anonymous function
numWords = lines.map(udf)

# Reduce to word in the entire RDD.
total = numWords.reduce(lambda a,b : a + b)
print (total)
```

6. Word Count implementation

```
# Import text file from s3 to a RDD
lines = sc.textFile("s3n://csed516/gburg.txt")

# Split string to words using flatmap
words = lines.flatMap(lambda s: s.split(" "))

# Create pairs - tuples in following format (word, count)
pairs = words.map(lambda s: (s, 1))

# Word count
counts = pairs.reduceByKey(lambda a, b: a + b)
print("total number of words {}".format(counts.count()))
```

Analysis with dataframes in PySpark

Dataset is a distributed collection of data. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQLs optimized execution engine. Dataset API is available in Scala and Java.
A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a dataframe in R/Python, but with richer optimizations under the hood. The dataframe API is available in Scala, Java, Python and R.

1. Begin again by creating a session

```
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2. Importing data: data can be imported from the local file system, HDFS or S3, in this example we'll import data from S3. We will work with the Iris dataset `https://archive.ics.uci.edu/ml/datasets/Iris`.

```
df = spark.read.csv("s3://csed516sec5/iris.csv", header="true", inferSchema="true")
```

   Note1: Both s3 and s3n work with Amazon EMR, learn more about the various ways to access s3 data from emr at: `https://aws.amazon.com/premiumsupport/knowledge-center/emr-file-system-s3/`.

3. Basic operations from dataframe API. Full python API listed at: `https://spark.apache.org/docs/latest/api/python/pyspark.sql.html`

```
df.count()
df.first()
df.show()

# Print schema
df.printSchema()

# Run queries using dataframe API
df.select("class").show()
df.filter(df['class']!='Iris-setosa').show()
df.groupBy('class').count().show()
```

4. Dataframe Operations and SQL queries:

```
#create temp view before using spark.sql
df.createOrReplaceTempView("iris")
sqlDF = spark.sql("SELECT* FROM iris")
sqlDF.show()
```

5. RDDs and DataFrames

```
from pyspark.sql import Row

# Create spark context
sc = spark.sparkContext

# Read a file from s3as RDD, run wordcount app on it.
lines = sc.textFile('s3n://csed516/gburg.txt')

words = lines.flatMap(lambda l: l.split(" "))
pairs = words.map(lambda s:(s,1))
counts = pairs.reduceByKey(lambda a, b :a+b)

# RDD -> dataFrame
#first split the tuple
words = counts.map(lambda p: Row(word=p[0], freq=int(p[1])))

# Create a dataFrame
schemaWC = spark.createDataFrame(words)
schemaWC.createOrReplaceTempView("words")
highFreqWords = spark.sql("SELECT word FROM words WHERE freq > 5")
highFW = highFreqWords.rdd.map(lambda p : "word: " + p.word).collect()
for w in highFW:
    print(w)
```

Spark and MLlib

MLlib is Sparks machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

1. **ML Algorithms**: common learning algorithms such as classification, regression, clustering, and collaborative filtering
2. **Featurization**: feature extraction, transformation, dimensionality reduction, and selection
3. **Pipelines**: tools for constructing, evaluating, and tuning ML Pipelines
4. **Persistence**: saving and load algorithms, models, and Pipelines
5. **Utilities**: linear algebra, statistics, data handling, etc.

The primary Machine Learning API for Spark is now the DataFrame-based API in the `spark.ml` package. We use the Iris data set to explore supervised and unsupervised methods from Spark MLlib. We'll do supervised and unsupervised training on the iris dataset. The data set has three classes and four features. Each class has 50 instances in the training set.

1. Read data from s3.

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

ds = spark.read.format("libsvm").load("s3://csed516sec5/sample_iris_data.txt")
ds.take(2)
```

2. Unsupervised learning using k-means.

```
from pyspark.ml.clustering import KMeans
# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(ds)

# Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(ds)
```

```
print("Within Set Sum of Squared Errors = " + str(wssse))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

3. Visualize the two clusters

```
# visualize two clusters
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import numpy as np

transformed = model.transform(ds).select("features", "prediction")

pddf = transformed.toPandas()
fv = pddf['features'].values.tolist()
fv = pd.DataFrame(np.asarray(fv))
fv.columns = ['s_1', 's_2','p_1','p_2']

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')
ax.scatter(fv['s_1'],fv['s_2'],fv['p_1'],c=pddf['prediction'])
ax.set_xlabel('s_1')
ax.set_ylabel('s_2')
ax.set_zlabel('p_1')
plt.show()
```

Try this again with k =3 instead.

4. Supervised ML on the same dataset, we'll use logistic regression, OneVsRest for multi-class classification.

```
#Supervised learning
from pyspark.ml.classification import LogisticRegression, OneVsRest
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

#load Data
inputData = spark.read.format("libsvm").load("s3://csed516/sample_iris_data.txt")


# generate the train/test split.
(train, test) = inputData.randomSplit([0.8, 0.2])

# instantiate the base classifier.
lr = LogisticRegression(maxIter=10, tol=1E-6, fitIntercept=True)

# instantiate the One Vs Rest Classifier.
ovr = OneVsRest(classifier=lr)

# train the multiclass model.
ovrModel = ovr.fit(train)
```

5. Compute classification error.

```
predictions = ovrModel.transform(test)

# obtain evaluator.
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
# compute the classification error on test data.
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

REFERENCES:

1. https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-release-components.html
2. https://www.perfectlyrandom.org/2018/08/11/setup-spark-cluster-on-aws-emr/