

# SDSC Summer Institute 2022

## Scalable Machine Learning



# Spark Hands-On

Mai H. Nguyen, Ph.D.

# Spark Topics

- **Spark**
  - History & Design Goals
  - RDDs
  - DataFrames
  - Spark Architecture
  - Spark API
  - Spark Core & Libraries
- **Spark Hands-On**
  - Scaling
  - Cluster Analysis
- **R on Spark**

# Setup

- **Login to Expanse**

- Open terminal window on local machine
- `ssh login.expanse.sdsc.edu -l <xdtr_account>`

- **Pull latest from repo**

- `git pull`
- URL: <https://github.com/sdsc/sdsc-summer-institute-2022>

# Server Setup for PySpark - Command Line

- **In terminal window**
  - jupyter-shared-spark
    - `galileo launch --account ${SI2022_ACCOUNT} --reservation ${SI2022_RESERVATION_CPU} --partition shared --cpus 4 --memory 16 --time-limit 04:00:00 --env-modules singularitypro --sif ${SI2022_CONTAINER_DIR}/ciml/2021/pyspark-latest.sif --bind /cm,/expanse,/scratch --quiet`
- **To check queue**
  - `squeue -u $USER`

# Data Setup

- In terminal window in Jupyter Lab, do the following
- Go to local scratch directory

```
cd /scratch$USER/job_*
```

```
pwd # Something like /scratch/xdtrXXX/job_XXXXXX
```

- **Get data**

- cp \$SI2022\_DATA\_DIR/BookReviews\_5M.txt.zip .
- unzip BookReviews\_5M.txt.zip

- cp \$SI2022\_DATA\_DIR/minute\_weather.csv.zip .
- unzip minute\_weather.csv.zip

- ls # Should see BookReviews\_5M.txt & minute\_weather.csv

Don't forget the  
period at the end!



# PySpark Scaling Hands-On

- **Data**

- BookReviews\_5M.txt

- Source : <https://jmcauley.ucsd.edu/data/amazon/>

- **Notebook**

- pyspark\_demo.ipynb
- Location: <si\_dir>/4.3b\_scalable\_ml/

- **To do**

- Change number of cores: 1, 2, 4
- Note difference in execution times

# GETTING EXECUTION TIMES

- In notebook, execution time is printed out in cell before Spark session is stopped (next to last cell)
- Need to restart the kernel and run all cells without stopping to get accurate execution time:
  - Run -> Restart Kernel and Run All Cells
- Find mean and standard deviation of execution times over 3 runs for
  - 1 core, 2 cores, and 4 cores

```
import pyspark
from pyspark.sql import SparkSession
```

```
conf = pyspark.SparkConf().setAll([
    ('spark.master', 'local[2]'),
    ('spark.app.name', 'PySpark Demo)])
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Specify number of cores.  
“\*” uses all available cores





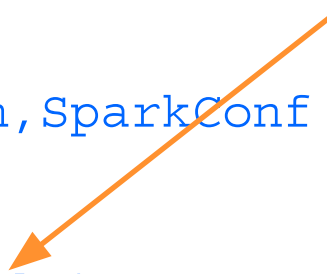
# SPARK SESSION

```
import pyspark
from pyspark.sql import SparkSession, SparkConf

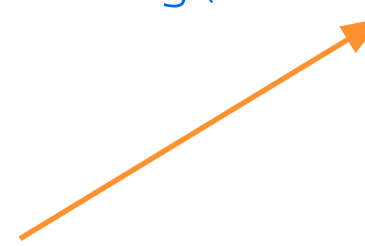
conf = SparkConf().setAll([
    ('spark.master', 'local[*]'),
    ('spark.app.name', 'PySpark Demo')])

spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Use \* to use all available cores, or integer value to specify number of cores to use



Configuration parameters for Spark session



Get existing Spark session or create new one



# SPARK PROGRAM STRUCTURE

- **Start Spark session**
  - `spark = SparkSession.builder.config(conf=conf).getOrCreate()`
- **Create distributed dataset**
  - `df = spark.read.csv("data.csv",header="True")`
- **Apply transformations**
  - `new_df = df.filter(col("dept") == "Sales")`
- **Perform actions**
  - `df.collect()`
- **Stop Spark session**
  - `spark.stop()`

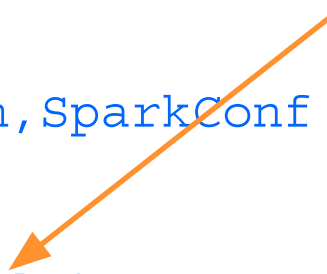
# START SPARK SESSION

```
import pyspark
from pyspark.sql import SparkSession, SparkConf

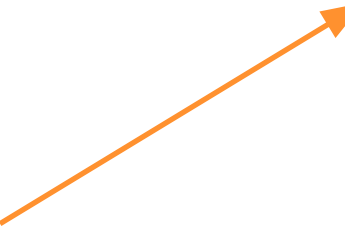
conf = SparkConf().setAll([
    ('spark.master', 'local[*]'),
    ('spark.app.name', 'PySpark Demo')])

spark = SparkSession.builder.config(conf=conf).getOrCreate()
```


Use \* to use all available cores, or integer value to specify number of cores to use



Configuration parameters for Spark session



Get existing Spark session or create new one



# LOAD DATA

```
df = spark.read.text("file.txt")
```

```
df = spark.read.csv("file.csv",  
                    header=True  
                    inferSchema=True).cache()
```

Indicates whether  
column headers exist



Automatically infer  
data types of columns



Cache data



# DROP ROWS WITH NULLS

- **Drop rows with null values**

```
df.dropna()  
df.dropna(how='any')  
df.dropna(how='all')
```

- **Check number of rows before and after dropping rows**

```
df.count()
```

# CREATE FEATURE VECTOR COLUMN

- **Create feature vector column**
  - Combines given list of columns into single vector column
  - To feed data to machine learning models

```
from pyspark.ml.feature import VectorAssembler
```

```
features = ['air_temp', 'relative_humidity']  
assembler = VectorAssembler(inputCols=features,  
                             outputCol='featureVector')
```

```
features_df = assembler.transform(df)  
features_df.show()
```

air_temp	relative_humidity
62.96	63.9



air_temp	relative_humidity	featureVector
62.96	63.9	[62.96, 63.9]

New column  
appended to  
features\_df

# PARTITION DATA

- Partition available data into train and test data sets

```
train_df, test_df = df.randomSplit(0.8, 0.2), seed=<seed>)
```

Weights for splits



Random number  
generator seed



# SCALE DATA

- **Scale input data values**

- Standardize values to have zero mean and unit standard deviation
- Each feature is scaled separately
- Create scale transformer using train data, then apply to train/test data

```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol="features_unscaled",
                        outputCol="features_scaled",
                        withStd=True, withMean=True)
scalerModel = scaler.fit(train_df)
```

```
train_df = scalerModel.transform(train_df)
test_df  = scalerModel.transform(test_df)
```



# BUILD MODEL


- **Build decision tree classifier**

- Create model
- Use fit() to train model

```
from pyspark.ml.classification import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier(  
    featuresCol='featureVector',  
    labelCol='label',  
    predictionCol='prediction')
```

Can specify name of  
columns for features,  
labels, and predictions




```
dt_model = dt.fit(<train>)
```

# APPLY MODEL

- **Apply trained model**
  - Use transform()

```
predictions = <model>.transform(<data>)
```

Will have <data> and  
new 'prediction'  
column appended at  
the end



Trained  
model



Input data



# EVALUATE CLASSIFICATION MODEL

- **Evaluator for classification model**
  - Calculates F1, precision, recall, accuracy

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

mc_evaluator = MulticlassClassificationEvaluator(
    predictionCol='prediction',
    labelCol='label',
    metricName='f1')

mc_evaluator.evaluate(<predictions>)
```

Column with predictions

Column with labels

Evaluation metric

Contains predictions and labels

# PySpark Cluster Analysis Hands-On

- **Data**

- Weather station measurements

- **Task**

- Perform cluster analysis to identify different weather patterns

- **Approach**

- Spark k-means

- **Notebook**

- Location: <ciml\_dir>/3.6\_spark
- pyspark-clustering.ipynb # Starter notebook
- pyspark-clustering-wOutput.ipynb # Has cell outputs
- pyspark-clustering-soln.ipynb # Solution

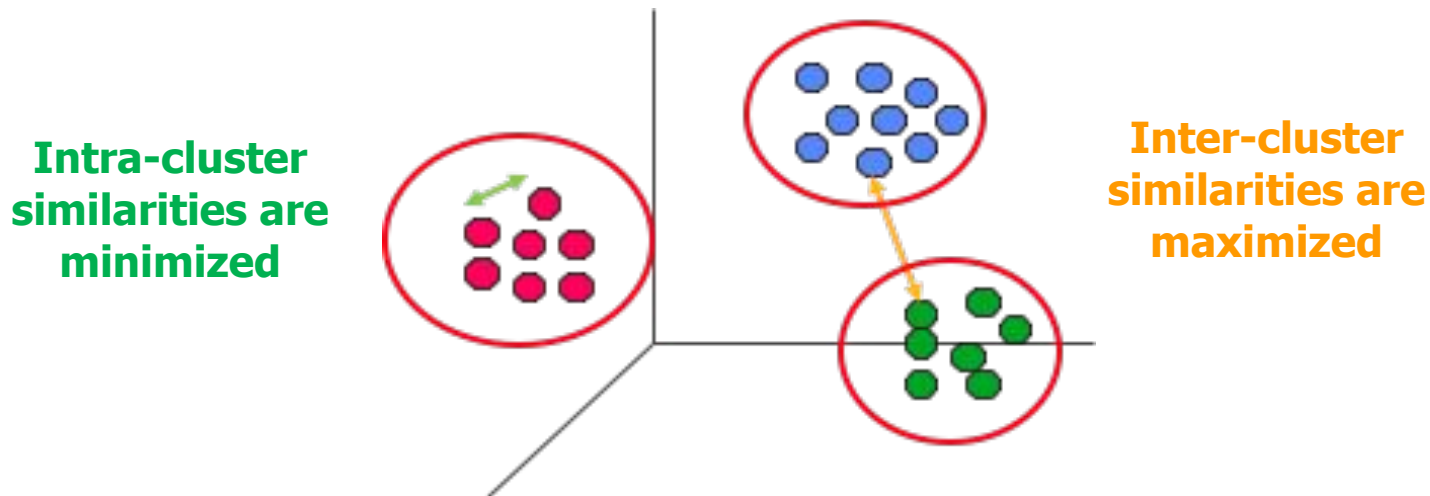
# Dataset Description

- **Measurements from weather station on Mt. Woodson, San Diego**
- **Air temperature, humidity, wind speed, wind direction, etc.**
- **Three years of data: Sep. 2011 - Sep. 2014**
  - minute\_weather.csv: measurement every minute
- **Source**
  - <http://hpwren.ucsd.edu>



# Cluster Analysis

- **Cluster analysis divides data into groups**
  - Grouping is based on some similarity measure.
  - Samples within a cluster are more similar to each other than to samples in other clusters.



<http://www-users.cs.umn.edu/~kumar/dmbook/index.php>

# *k*-Means Clustering

- **Partitional**
  - Clusters are divided into non-overlapping subsets
- **Centroid-Based**
  - Cluster represented by central vector
- **Simple, classic clustering technique**
  - Data points are grouped into  $k$  clusters
  - Cluster defined by cluster mean

- **Algorithm**

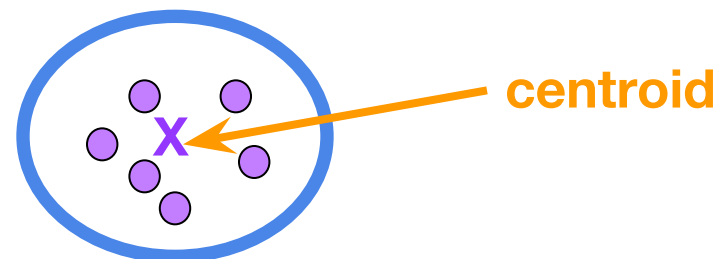
Select  $k$  initial *centroids* (cluster centers)

Repeat

Assign each sample to closest centroid

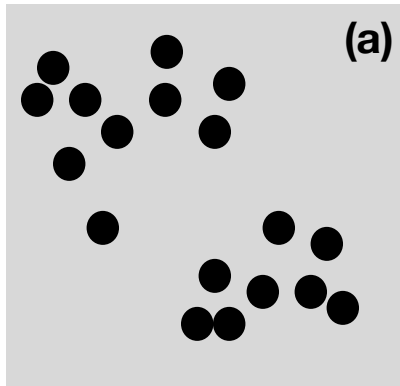
Calculate mean of cluster to determine new centroid

Until some stopping criterion is reached

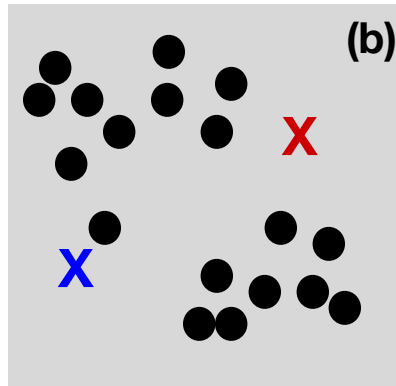




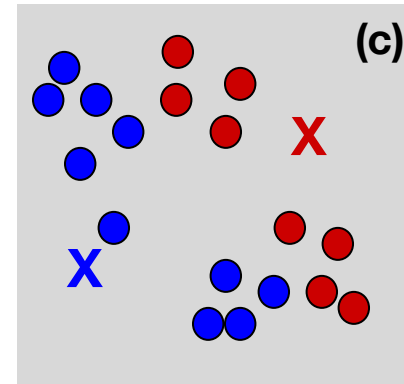
# *k*-Means Clustering Illustration



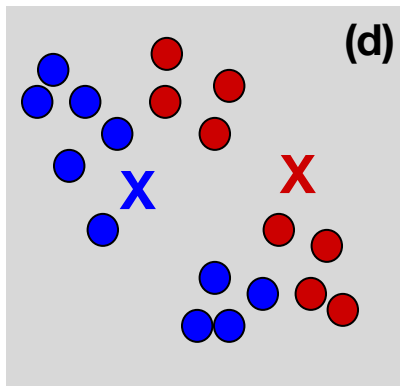
Original samples



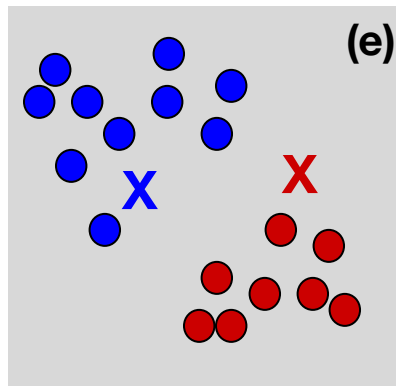
Initial Centroids



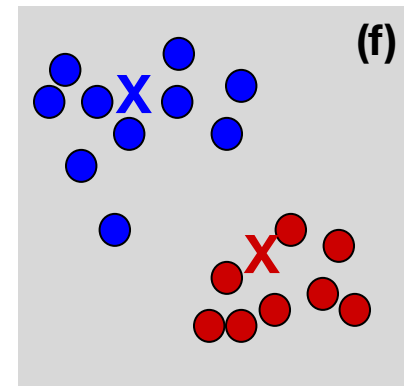
Assign Samples



Re-calculate Centroids



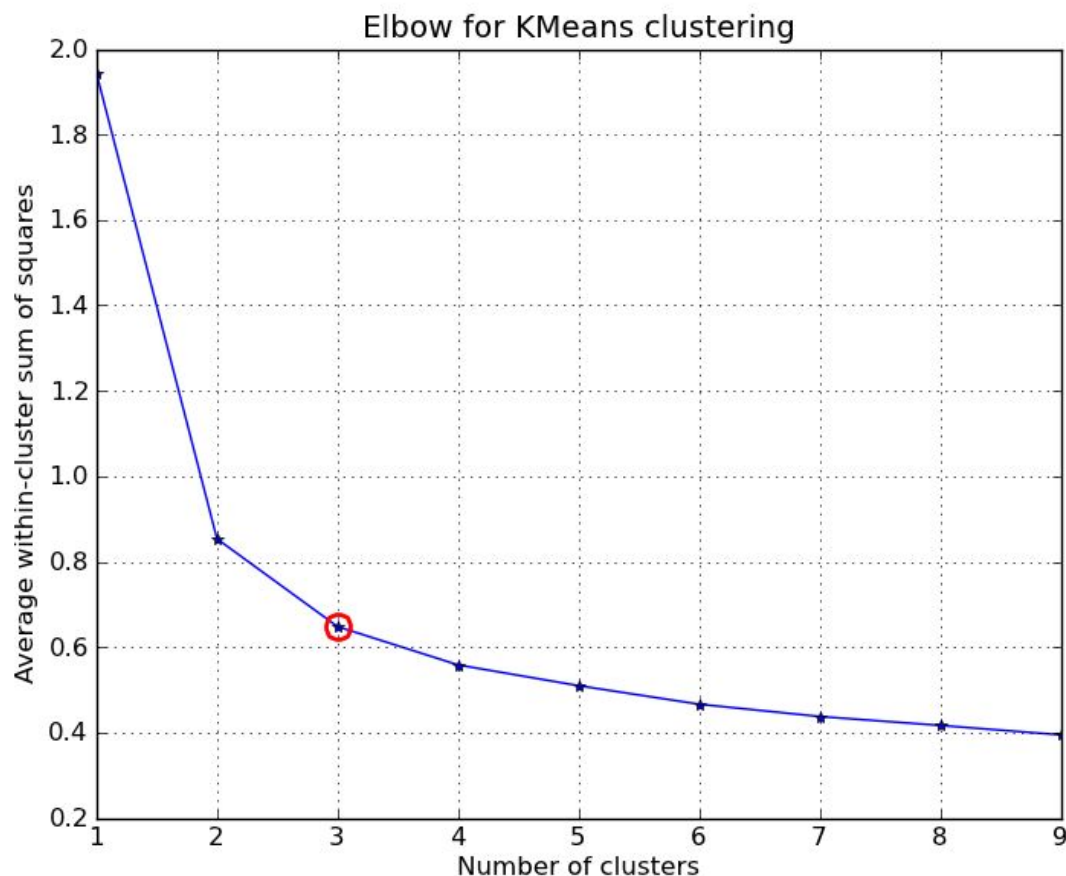
Assign Samples



Re-calculate  
Centroids

# Choosing Number of Clusters ( $k$ )

- **Elbow method**
  - Plot cluster evaluation metric (e.g., WSSE) vs. different values for  $k$
  - “Elbow” in plot suggests value(s) for  $k$



<http://stackoverflow.com/questions/6645895/calculating-the-percentage-of-variance-measure-for-k-means>

# Evaluating Clustering Results

- **Within-Cluster Sum of Squared Error (WSSE)**
- For each sample, error is distance to centroid.  
Then, **WSSE** is computed as:

$$WSSE = \sum_{i=1}^K \sum_{x \in C_i} \|x - m_i\|^2$$

$x$ : data sample in cluster  $C_i$

$m_i$ : cluster centroid (i.e., mean of cluster)

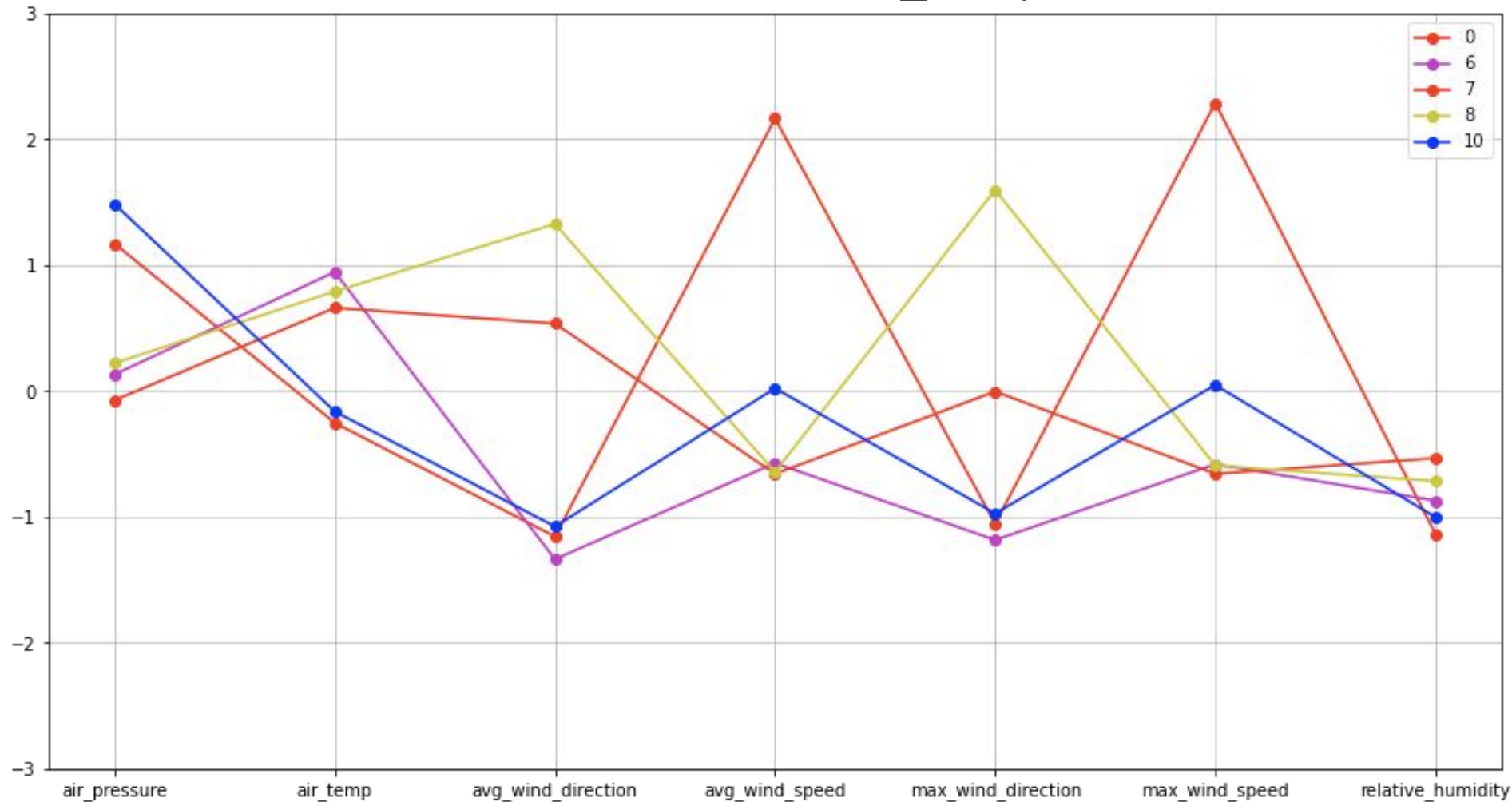
$\|x - m_i\|^2$ : Euclidean distance between  $m_i$  and  $x$

# Clustering Hands-On Overview

- **Setup**
  - Start Spark
  - Load modules
- **Load data**
  - Specify schema
  - Read in data from “minute\_weather.csv”
- **Explore data**
  - Look at schema, number of rows, summary statistics
- **Prepare data**
  - Drop nulls
  - Create feature vector
- **Perform k-means cluster analysis**
  - Use elbow plot to determine k
  - Build k-means model
- **Evaluate clusters**
  - Plot cluster profiles
- **Stop Spark session**

# Cluster Profile: Parallel Plots

```
utils.parallel_plot(centersNamed[centersNamed['relative_humidity'] < -0.5],  
numClusters, colors=colors_used);
```



# PySpark Cluster Analysis Hands-On

- **Code**

- pyspark-clustering.ipynb
  - Notebook for hands-on
  - Add your code where indicated: # ==> YOUR CODE HERE
- pyspark-clustering-wOutput.ipynb
  - Has cell outputs
- pyspark-clustering-soln.ipynb
  - Solution
- **utils.py**
  - Has utility functions

- **Resources**

- [Apache Spark](#)
- [PySpark Documentation — PySpark Documentation](#)
- [Spark SQL and DataFrames Guide](#)
- Python for Data Science Cheat Sheet (pdf)

# PANDAS API ON SPARK

- **Pandas API integrated into Spark as of v. 3.2**
- **Allows for simple porting of already existing Pandas code to Spark**
  - Same code can run with pandas (single node) and Spark (distributed system)
- **Can use Spark to scale to multiple machines**
- **Can take advantage of unified analytics provided by Spark**
  - Can access functionality provided by other Spark libraries
- **Currently cover 83% of pandas API.**
- **To use**

```
from pandas import read_csv  
from pyspark.pandas import read_csv  
pdf = read_csv("data.csv")
```

# Spark Topics

- **Spark**
  - History & Design Goals
  - RDDs
  - DataFrames
  - Spark Architecture
  - Spark API
  - Spark Core & Libraries
- **Spark Hands-On**
  - Scaling
  - Cluster Analysis
- **R on Spark**



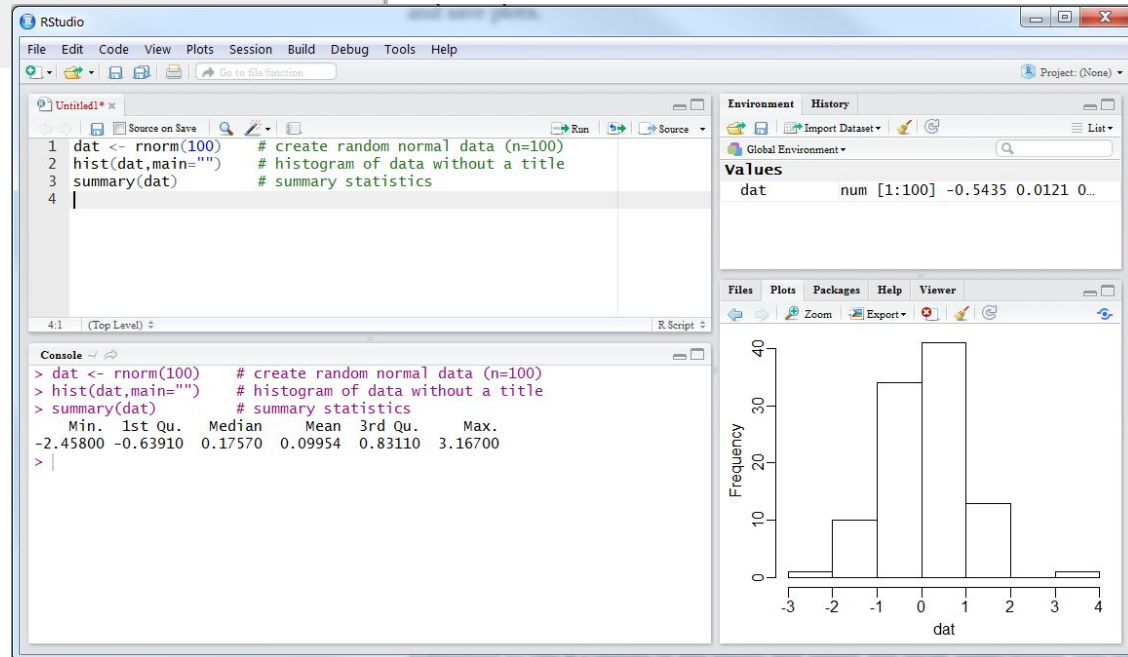
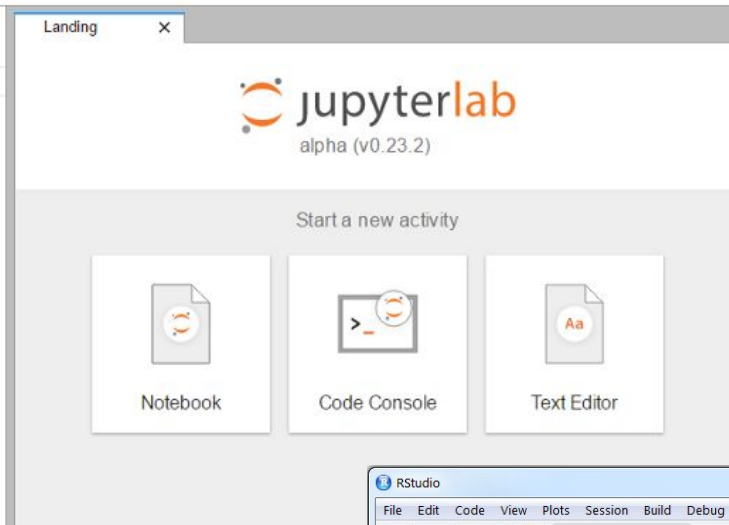
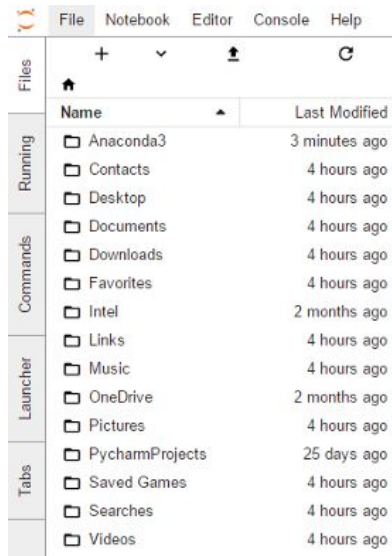
# SparkR

- **R package that provides frontend to use Spark from R**
- **Supports distributed machine learning using R API**
- **Allows R script to connect to Spark cluster**
- **Can use with R shell or RStudio or other R IDEs.**

# SparkR

- **Uses MLlib for machine learning functionality**
- **Familiar R syntax:**
  - Read contents of file into a Spark dataframe
    - `newdata <- read.df ("data.txt", source="csv")`
  - R formula operators for model fitting:
    - `model <- spark.randomForest(training, label ~ features, "classification", numTrees = 10)`
  - Get summary of fitted model
    - `summary(model)`
  - Apply model to make predictions
    - `predictions <- predict(model, testDF)`
  - Save model
    - `write.ml (model, "mymodel")`

# Running SparkR Code





**sparklyr**



- **R interface for Spark**
- **R Legacy**
  - From Rstudio
  - Available on CRAN
- **Spark backend to dplyr and SQL**
  - Interactively manipulate Spark data using dplyr and SQL
- **Access to Spark functionality**
  - Interface to Spark MLlib algorithms
- **Connect to Spark clusters**

# sparklyr

- **Setup**

- `install.packages("sparklyr")`
- `library(sparklyr)`
- `spark_install ()`

- **Connect to Spark**

- `sc <- spark_connect (master="local")`

- **Using dplyr**

- `flights_sdf %>% group_by(tailnum)`  
`%>% filter(count > 20)`

- **Machine Learning**

- `model <- ml_random_forest (`  
`train_sdf, quality ~ ., type="classification")`

# sparklyr Functions

- **Spark Operations**
  - Manage Spark connections (e.g., `spark_config()`)
- **Spark Data Manipulation**
  - Read data in Spark DataFrame and perform operations (e.g., `spark_read_csv()`, `tbl_cache()`)
- **Feature Transformers**
  - Transform data (e.g., `ml_pca()`, `ft_bucketizer()`)
- **Distributed Machine Learning**
  - Access Spark Mllib algorithms (e.g., `ml_kmeans()`)
- **Streaming**
  - Support streaming data operations (e.g., `stream_read_json()`)
- **Extensions**
  - Interface to platforms for big data analysis, graph analytics, production

# Machine Learning Algorithms in SparkR

## Machine Learning

### Algorithms

SparkR supports the following machine learning algorithms currently:

#### Classification

- `spark.logit`: Logistic Regression
- `spark.mlp`: Multilayer Perceptron (MLP)
- `spark.naiveBayes`: Naive Bayes
- `spark.svmLinear`: Linear Support Vector Machine

#### Regression

- `spark.survreg`: Accelerated Failure Time (AFT) Survival Model
- `spark.glm` or `glm`: Generalized Linear Model (GLM)
- `spark.isoreg`: Isotonic Regression

#### Tree

- `spark.gbt`: Gradient Boosted Trees for Regression and Classification
- `spark.randomForest`: Random Forest for Regression and Classification

#### Clustering

- `spark.bisectingKmeans`: Bisecting k-means
- `spark.gaussianMixture`: Gaussian Mixture Model (GMM)
- `spark.kmeans`: K-Means
- `spark.lda`: Latent Dirichlet Allocation (LDA)

#### Collaborative Filtering

- `spark.als`: Alternating Least Squares (ALS)

#### Frequent Pattern Mining

- `spark.fpGrowth`: FP-growth

#### Statistics

- `spark.kstest`: Kolmogorov-Smirnov Test

# Resources

- **Spark**
  - <https://spark.apache.org/>
- **PySpark API**
  - <https://spark.apache.org/docs/latest/api/python/index.html>
- **Spark DataFrame**
  - <https://spark.apache.org/docs/latest/sql-programming-guide.html>
- **MLlib**
  - <https://spark.apache.org/mllib/>
- **SparkR**
  - <https://spark.apache.org/docs/latest/sparkr.html>
- **SparkR API**
  - <https://spark.apache.org/docs/latest/api/R/>
- **sparklyr**
  - <https://spark.rstudio.com/>



# Spark Topics

- **Spark**
  - History & Design Goals
  - RDDs
  - DataFrames
  - Spark Architecture
  - Spark API
  - Spark Core & Libraries
- **Spark Hands-On**
  - Scaling
  - Cluster Analysis
- **R on Spark**