2017 SDSC Summer Institute
Scalable Machine Learning

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

Spark Hands-On

Mai H. Nguyen

Overview

- **Weather station measurements**
- **Data Exploration**
  - Load into Spark DataFrame
  - Describe schema
  - Show summary statistics
  - Calculate correlation between features
- **Cluster to identify different weather patterns**
  - Spark k-means
  - Parallel plots

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
- ***daily_weather.csv***
  - aggregated measurements

Dataset Description

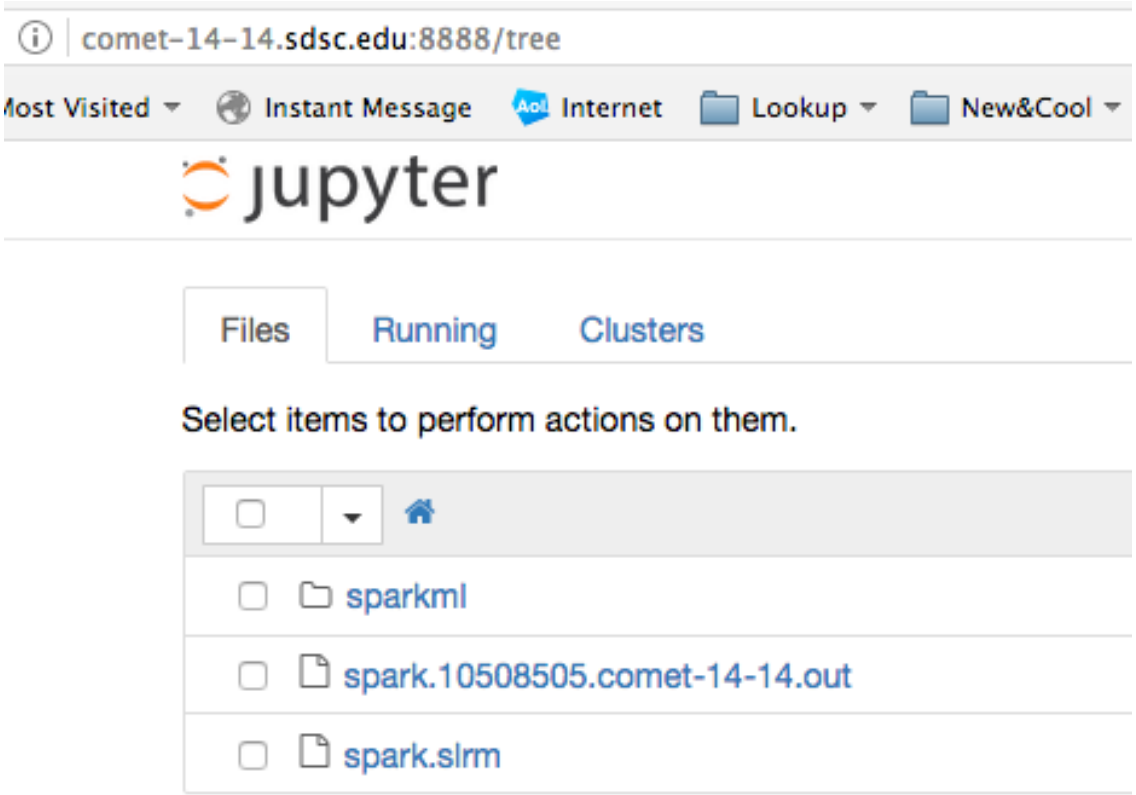
- **Measurements from weather station on Mt. Woodson, San Diego**
- **Air temperature, humidity, wind speed, wind direction, etc.**
- **Three years of data: Sep. 2011 - Sep. 2014**
- ***minute_weather.csv*** — Clustering
  - measurement every minute
- ***daily_weather.csv*** — Data Exploration
  - aggregated measurements

Server Setup

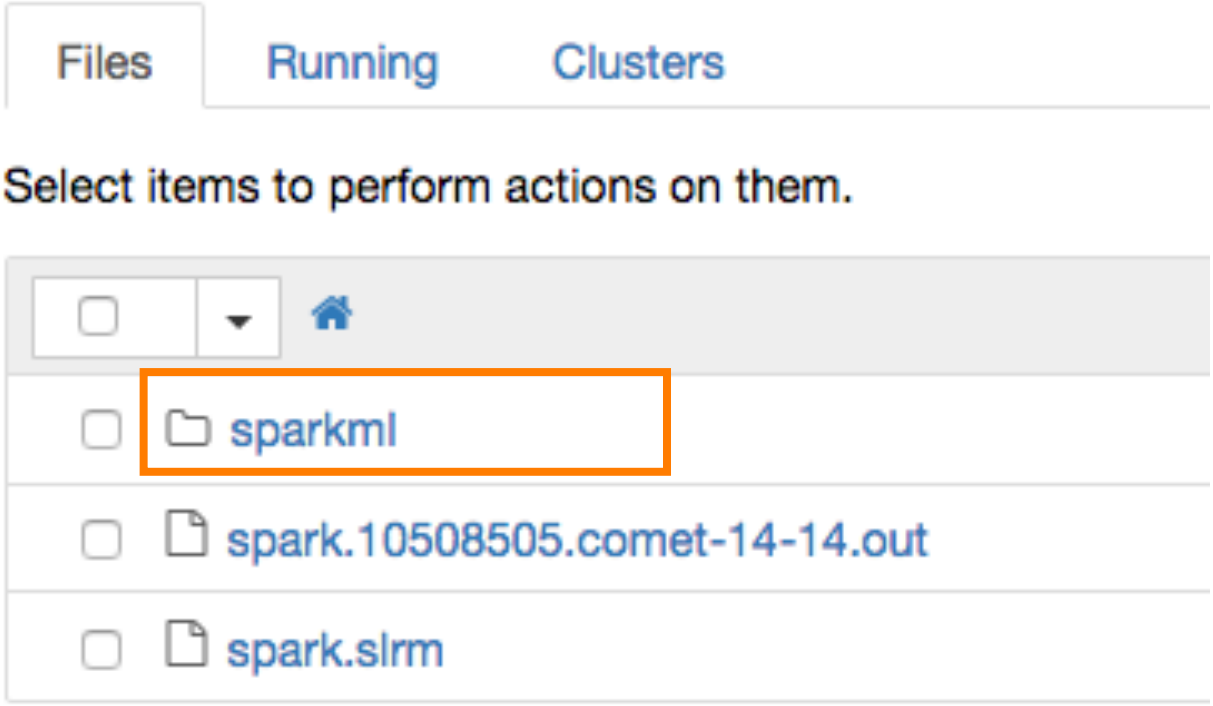
- **Go to Spark directory**
  - *cd SI2017/scalableML/spark*
- **Request compute node**
  - *sbatch spark.slrm*
- **Check queue**
  - *squeue –u $USER*

Browser Setup

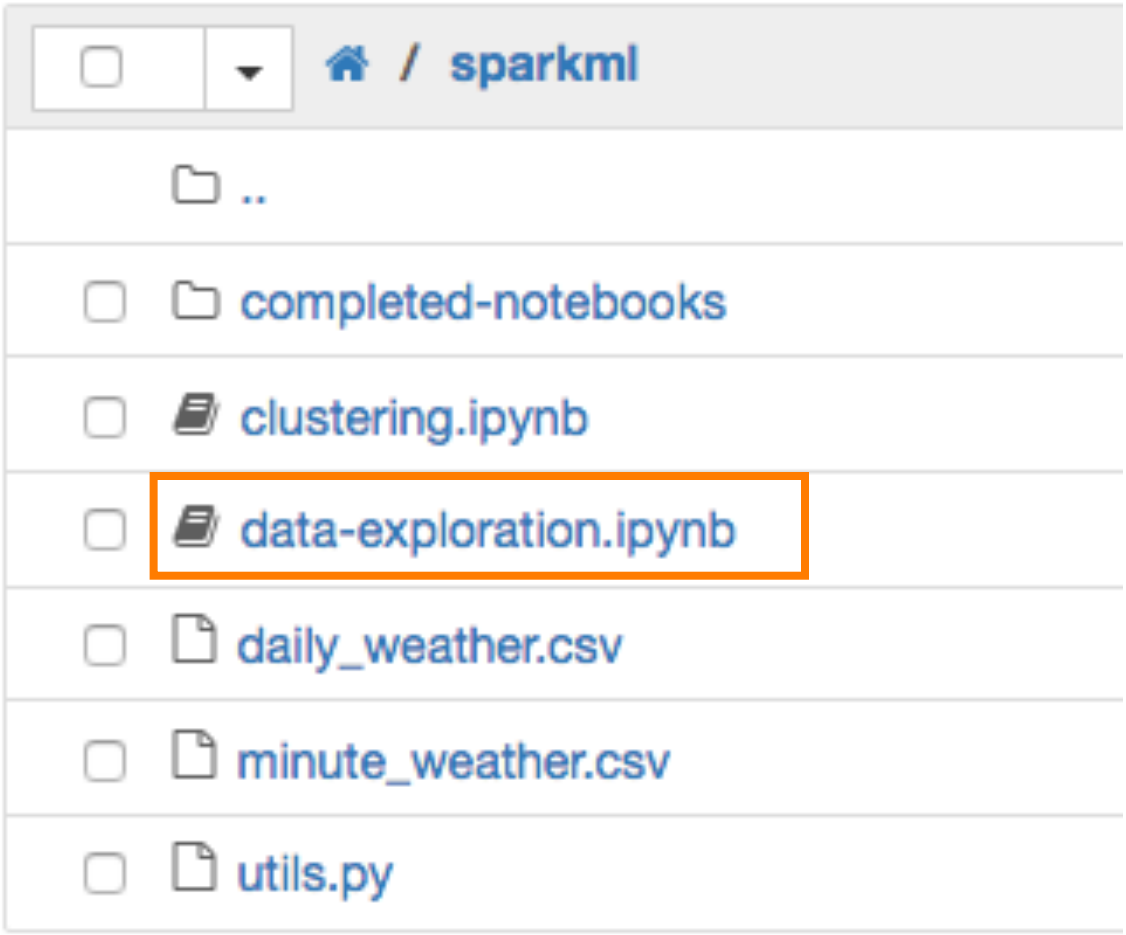
- **In browser, type:**
  - comet-xx-xx.sdsc.edu:8888
  - Should see:



Go to sparkml subdirectory



Open Data Exploration Notebook



Load Data into Spark DataFrame

- Create the Spark SQL Context
- Read the daily weather data into a DataFrame

```
# Load data into Spark dataframe

from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

df = spark.read.format("csv").option("header","true").option("inferSchema","true").load("file:<path>/daily_weather.csv"
```

- Replace <path> with location of file, e.g.:
  - /home/<user>/SI2017/scalableML/spark/sparkml

Examine Schema

df.printSchema()

```
root
 |-- number: integer (nullable = true)
 |-- air_pressure_9am: double (nullable = true)
 |-- air_temp_9am: double (nullable = true)
 |-- avg_wind_direction_9am: double (nullable = true)
 |-- avg_wind_speed_9am: double (nullable = true)
 |-- max_wind_direction_9am: double (nullable = true)
 |-- max_wind_speed_9am: double (nullable = true)
 |-- rain_accumulation_9am: double (nullable = true)
 |-- rain_duration_9am: double (nullable = true)
 |-- relative_humidity_9am: double (nullable = true)
 |-- relative_humidity_3pm: double (nullable = true)
```

Show Summary Statistics

df.describe().toPandas().transpose()

	0	1	2	3	4
summary	count	mean	stddev	min	max
number	1095	547.0	316.24357700987383	0	1094
air_pressure_9am	1092	918.8825513138094	3.184161180386833	907.9900000000024	929.3200000000012
air_temp_9am	1090	64.93300141287072	11.175514003175877	36.752000000000685	98.90599999999992
avg_wind_direction_9am	1091	142.2355107005759	69.13785928889189	15.500000000000046	343.4
avg_wind_speed_9am	1092	5.50828424225493	4.5528134655317185	0.69345139999974	23.554978199999763
max_wind_direction_9am	1092	148.95351796516923	67.23801294602953	28.89999999999991	312.19999999999993
max_wind_speed_9am	1091	7.019513529175272	5.598209170780958	1.1855782000000479	29.84077959999996
rain_accumulation_9am	1089	0.20307895225211126	1.5939521253574893	0.0	24.01999999999907
rain_duration_9am	1092	294.1080522756142	1598.0787786601481	0.0	17704.0
relative_humidity_9am	1095	34.24140205923536	25.472066802250055	6.090000000001012	92.6200000000002
relative_humidity_3pm	1095	35.34472714825898	22.524079453587273	5.3000000000006855	92.2500000000003

Calculate Correlation of Air Temperature vs Humidity

df.stat.corr("air_temp_9am", "relative_humidity_9am")

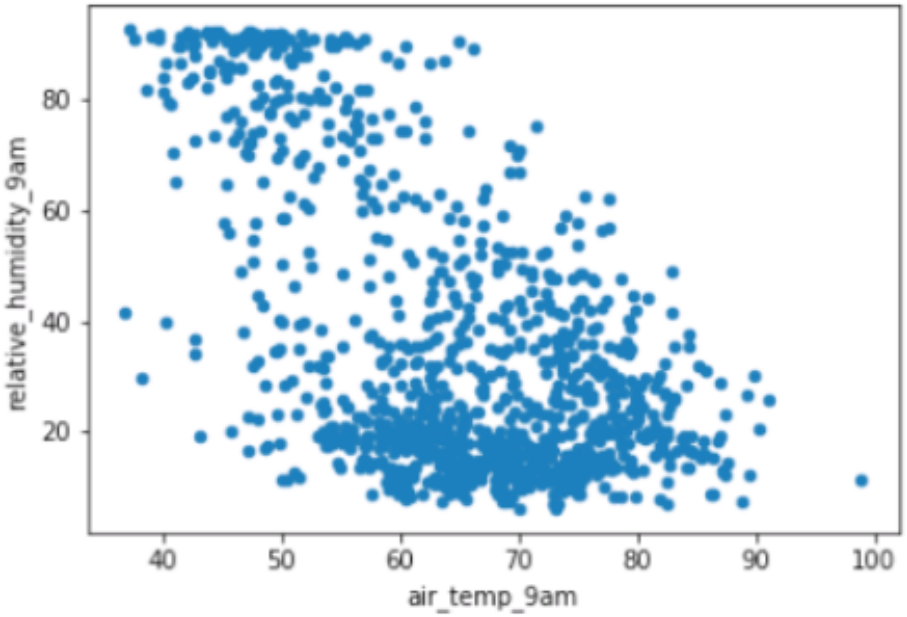
= -0.536670…

Show Plots in Notebook

%matplotlib inline

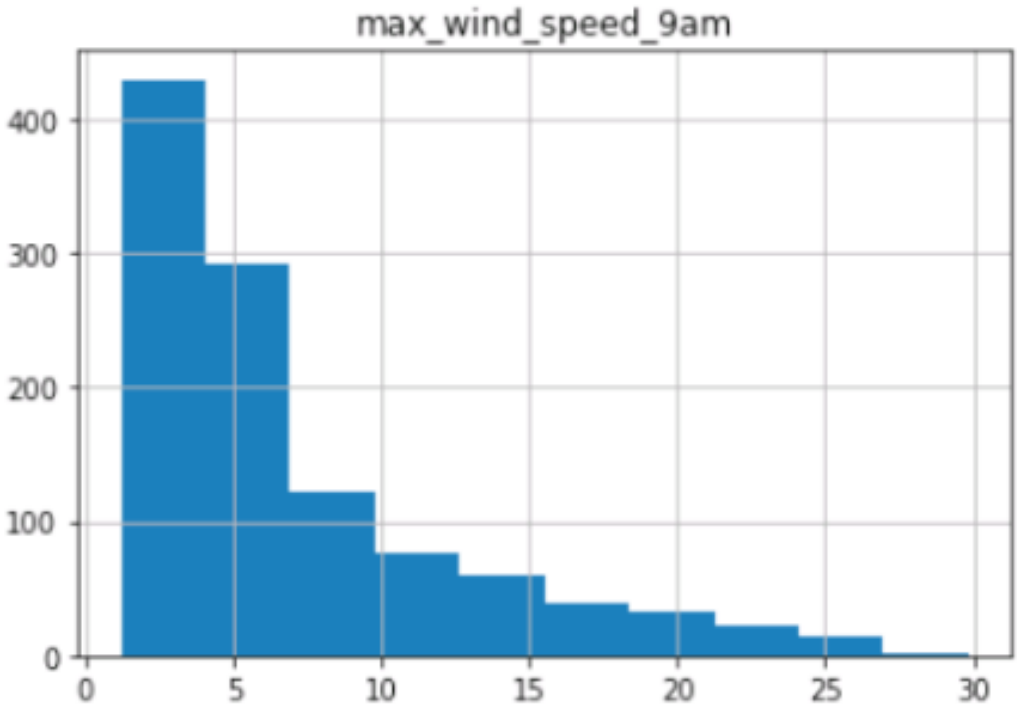
Scatter Plot of Air Temperature vs Humidity

```
df.select('air_temp_9am', 'relative_humidity_9am')
.toPandas()
.plot (kind='scatter', x='air_temp_9am',
       y='relative_humidity_9am')
```



Histogram of Max Wind Speed

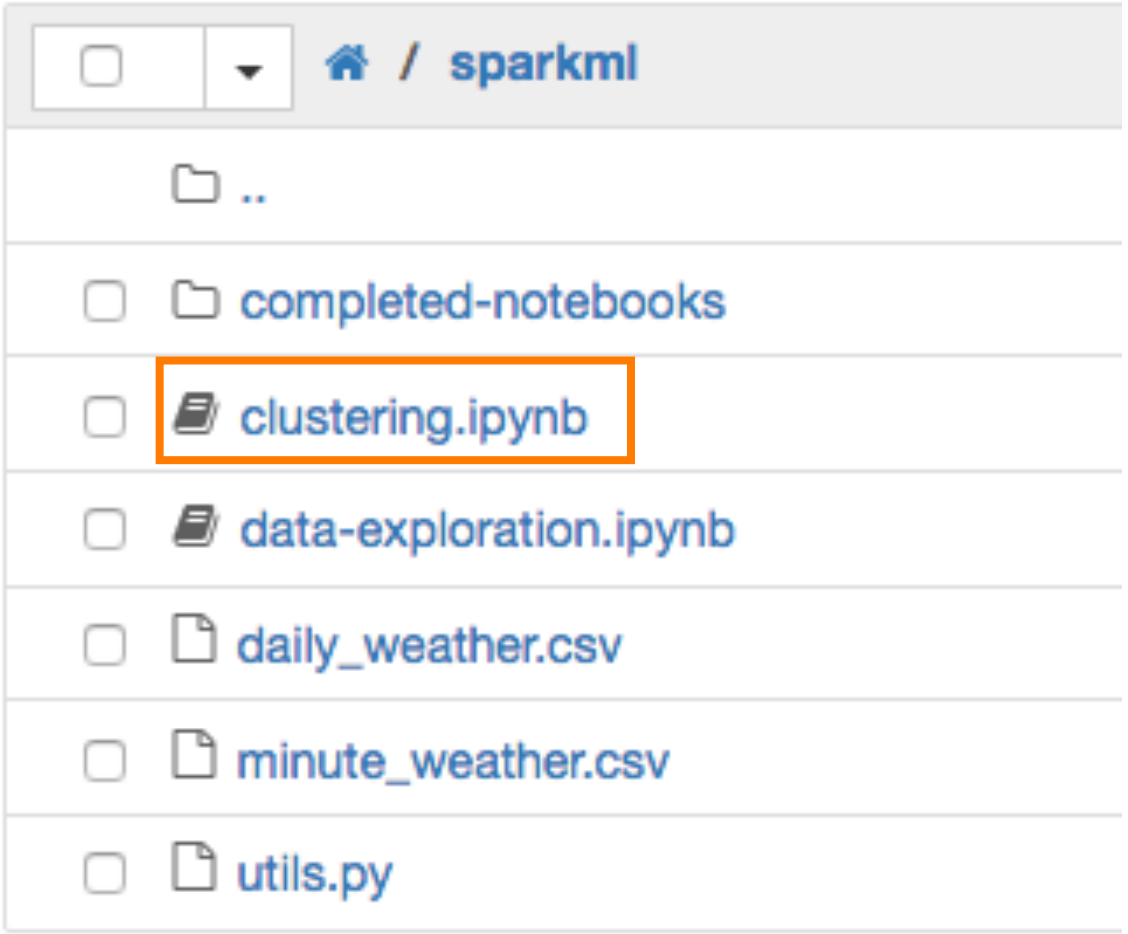
df.select('max_wind_speed_9am').toPandas().hist()



Clustering to Identify Santa Ana Conditions

- **Strong, dry winds in Southern California**
  - wind speed > 30mph
  - wind direction between 10 & 110 degrees (from east)
  - relative humidity < 10%
- **Extreme fire danger**
  - May 2014, swarm of 14 wildfires in San Diego County
  - 2008, Witch Fire, ~200,000 acres
  - 2003, Cedar Fire, ~280,000 acres

Open Clustering Notebook



Load Libraries & Minute Weather Data

In [ ]:
```
# Load libraries

from pyspark.sql import SQLContext
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
import utils
%matplotlib inline
```

In [ ]:
```
# Load minute weather data

sqlContext = SQLContext(sc)
df = spark.read.format("csv").option("header","true").option("inferSchema","true").load("file:<path>/minute_weather.csv
```

- Replace <path> with location of file, e.g.:
  - /home/<user>/SI2017/scalableML/spark/sparkml

Count Rows and Filter Data

- **Count rows**

  df.count()

  = 1587257

Filter data

filteredDF = df.filter((df.rowID % 100) == 0)

filteredDF.count()

= 15873

Show Summary Statistics

filteredDF.describe().toPandas().transpose()

	0	1	2	3	4
summary	count	mean	stddev	min	max
rowID	15873	793600.0	458228.4746717515	0	1587200
air_pressure	15873	916.8291627291587	3.0517222151797943	905.1	929.4
air_temp	15873	61.854689094688936	11.83541379082148	32.36	96.44
avg_wind_direction	15870	161.2875236294896	95.3131612965649	0.0	359.0
avg_wind_speed	15870	2.7928040327662296	2.0705061984600173	0.1	20.1
max_wind_direction	15870	162.70094517958412	92.26960112663167	0.0	359.0
max_wind_speed	15870	3.41462507876495	2.428906406812135	0.1	20.9
min_wind_direction	15870	166.64429741650915	97.82483630682509	0.0	359.0
min_wind_speed	15870	2.1522684310018896	1.7581135042599596	0.0	19.5

Drop Null Data

workingDF = filteredDF.na.drop()
workingDF.count()

= 15869

Create Feature Vector

```
featuresUsed = ['air_pressure', 'air_temp', 'avg_wind_direction', 'avg_wind_speed',
        'max_wind_speed','relative_humidity']
assembler = VectorAssembler(inputCols=featuresUsed, outputCol="features_unscaled")
assembled = assembler.transform(workingDF)
```

Scale Data

```
scaler = StandardScaler(inputCol="features_unscaled", outputCol="features", withStd=True,
scalerModel = scaler.fit(assembled)
scaledData = scalerModel.transform(assembled)
```

Use One-third Data for Elbow Plot

```
# Use one-third data for elbow plot

scaledData = scaledData.select("features", "rowID")

elbowset = scaledData.filter((scaledData.rowID % 3) == 0).select("features")
elbowset.persist()
elbowset.count()
```

```
5289
```

Generate Clusters for Elbow Plot

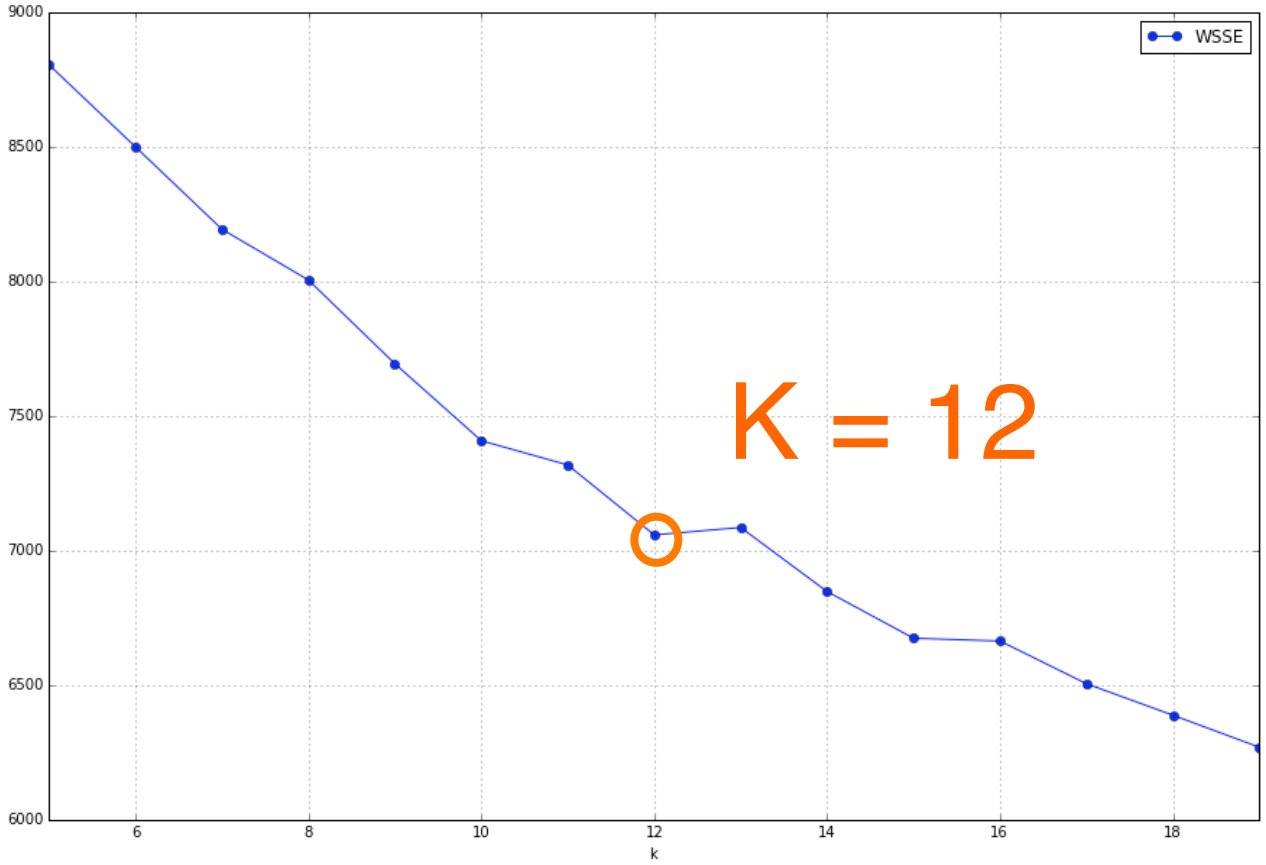
clusters = range(5, 20)

wsseList = utils.elbow(elbowset, clusters)

```
Training for cluster size 5
.....................WSSE = 8804.118354684928
Training for cluster size 6
.....................WSSE = 8525.400895523859
Training for cluster size 7
.....................WSSE = 8123.225352699479
Training for cluster size 8
.....................WSSE = 7876.960906428347
Training for cluster size 9
.....................WSSE = 7887.3819888393555
Training for cluster size 10
.....................WSSE = 7554.953661841033
Training for cluster size 11
.....................WSSE = 7294.206903885911
Training for cluster size 12
```

Show Elbow Plot

utils.elbow_plot(wsseList, clusters)



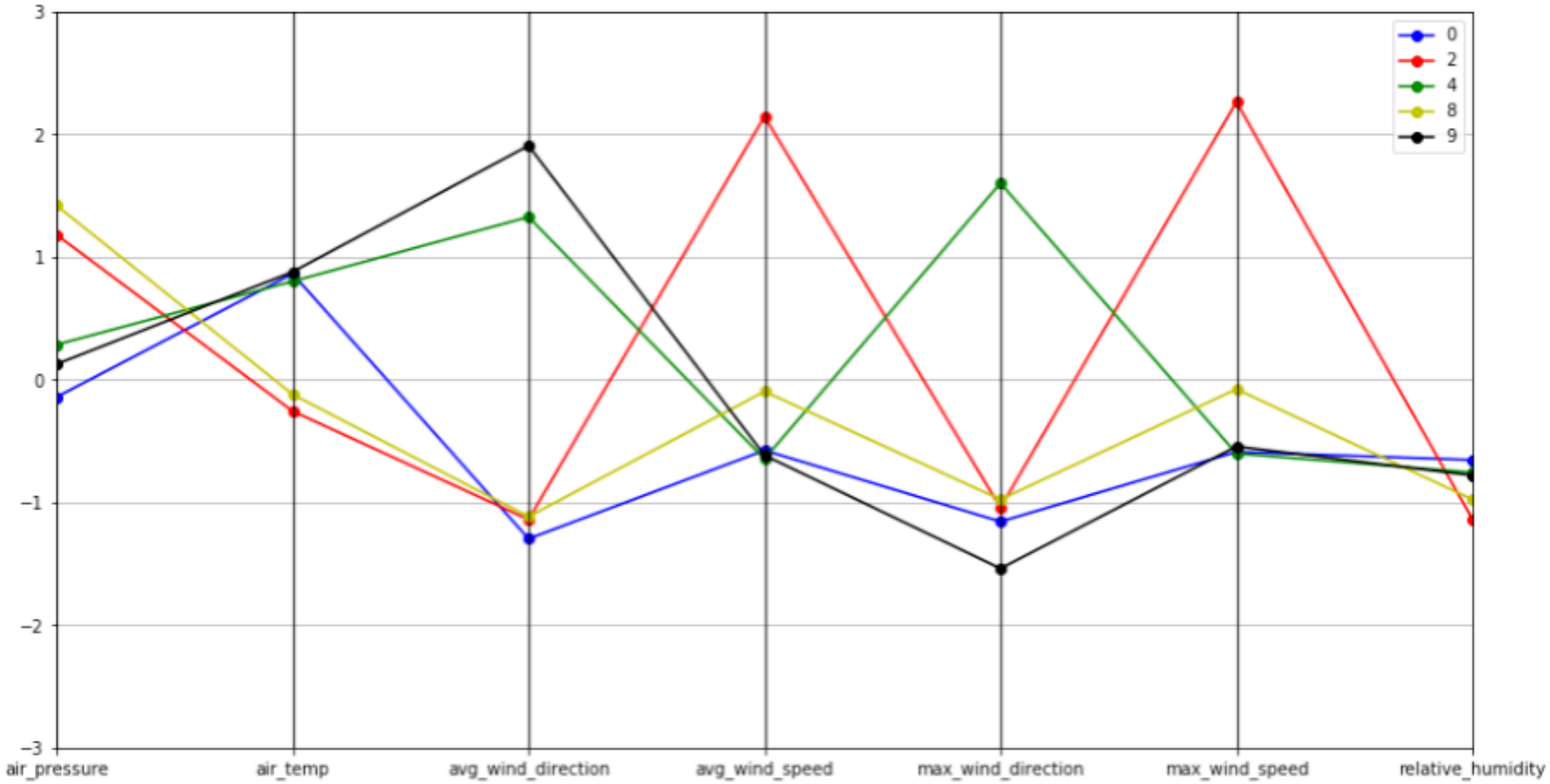
Run KMeans for k = 12 and Extract Cluster Centers

```
# Run KMeans for k = 12

scaledDataFeat = scaledData.select("features")
scaledDataFeat.persist()

kmeans = KMeans(k=12, seed=1)
model = kmeans.fit(scaledDataFeat)
transformed = model.transform(scaledDataFeat)
```

```
# Compute cluster centers

centers = model.clusterCenters()
P = utils.pd_centers(featuresUsed, centers)
centers
```

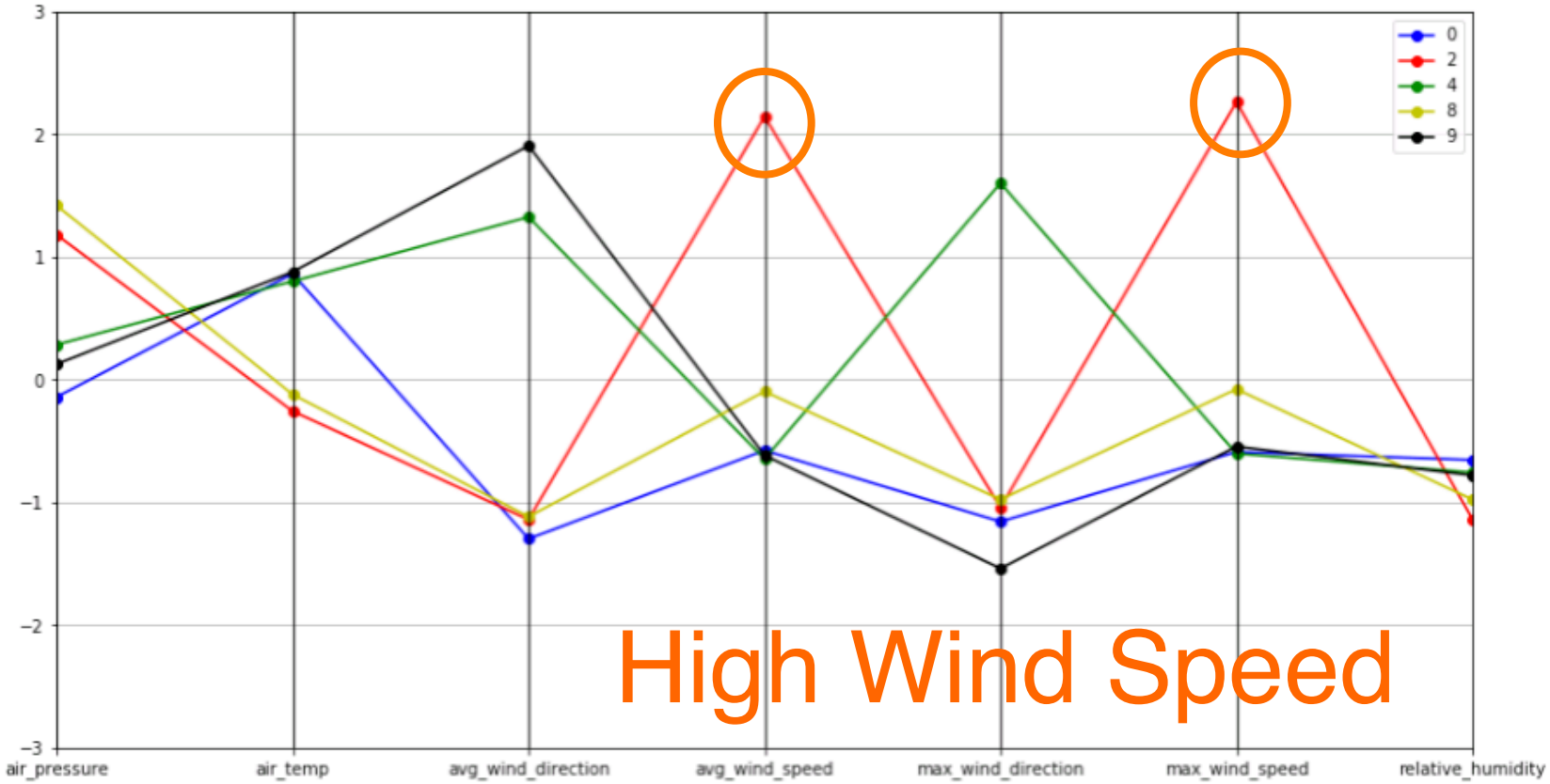
Parallel Plot: Dry Days

utils.parallel_plot(P[P['relative_humidity'] < -0.5], P)



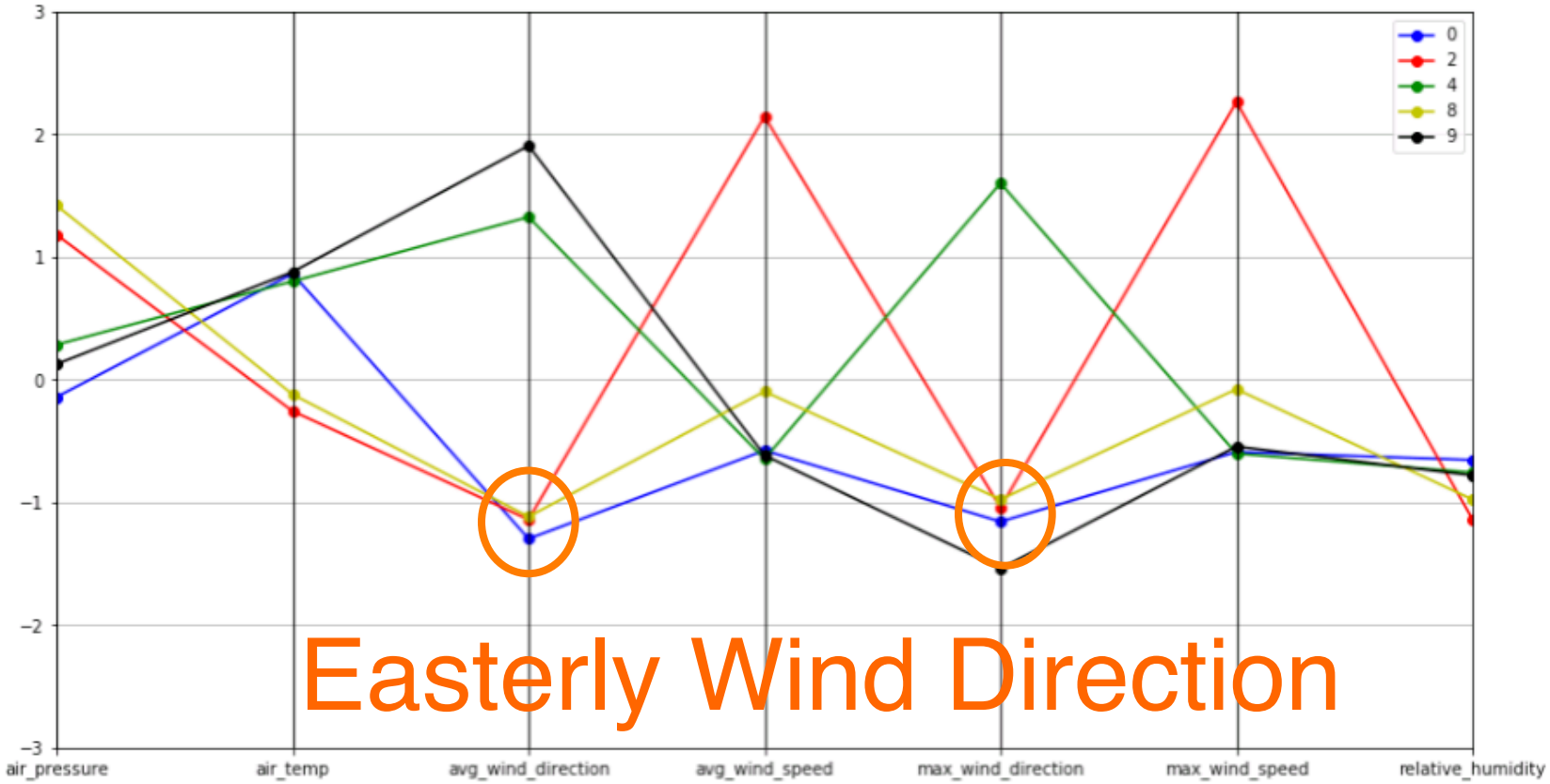
Parallel Plot: Dry Days

utils.parallel_plot(P[P['relative_humidity'] < -0.5], P)



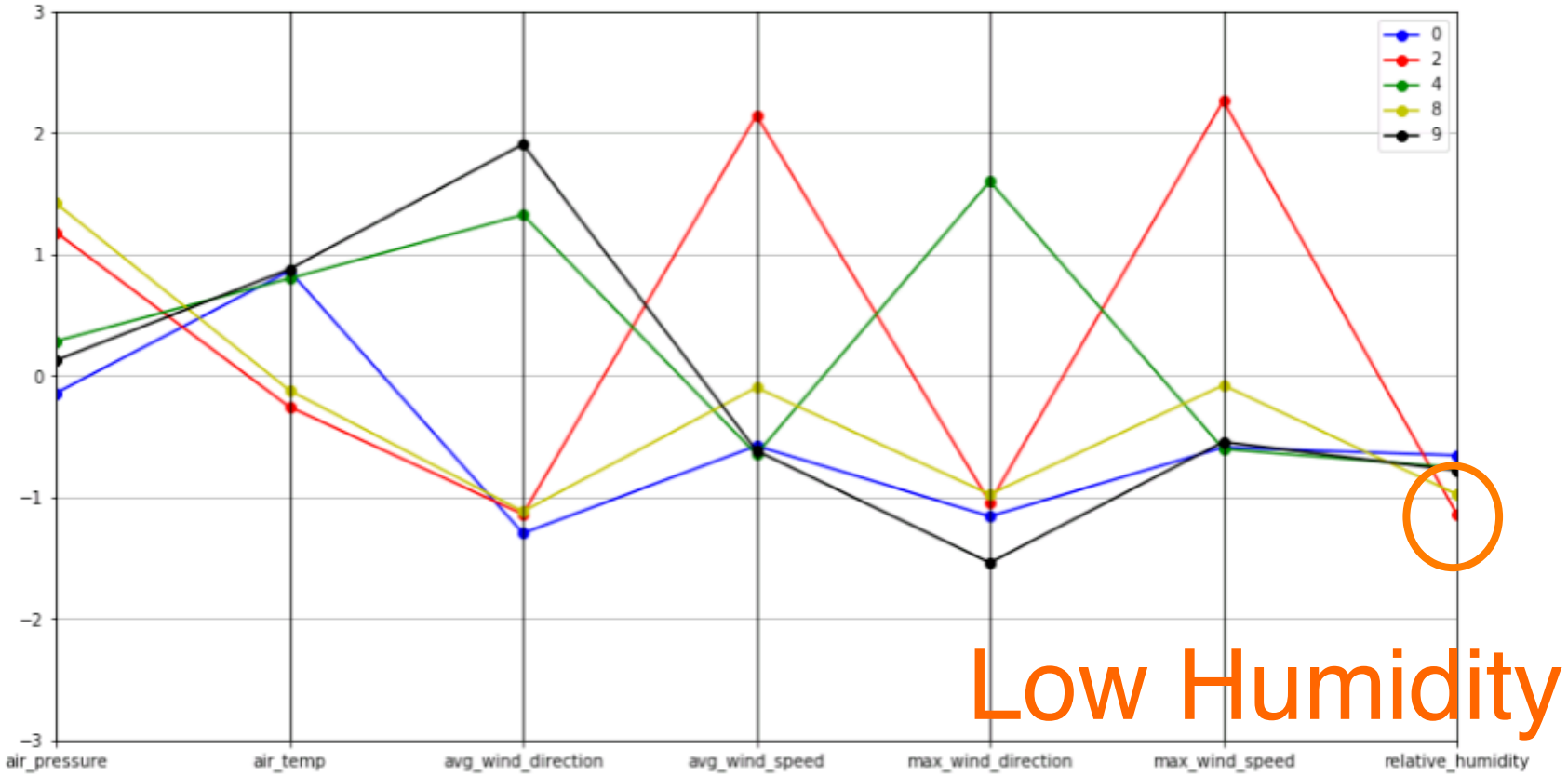
Parallel Plot: Dry Days

utils.parallel_plot(P[P['relative_humidity'] < -0.5], P)



Parallel Plot: Dry Days

utils.parallel_plot(P[P['relative_humidity'] < -0.5], P)



Parallel Plot Exercises

- **Humid days**
  - utils.parallel_plot(P[P['relative_humidity'] > 0.5], P)
- **Hot days**
  - utils.parallel_plot(P[P['air_temp'] > 0.5], P)
- **Cool days**
  - utils.parallel_plot(P[P['air_temp'] < -0.5], P)

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”)

Random Forest Classifier Example

```
# Load training data
df <- read.df("data/mllib/sample_libsvm_data.txt", source = "libsvm")
training <- df
test <- df

# Fit a random forest classification model with spark.randomForest
model <- spark.randomForest(training, label ~ features, "classification", numTrees = 10)

# Model summary
summary(model)

# Prediction
predictions <- predict(model, test)
head(predictions)
```

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

For More on SparkR

- **SparkR**
  - https://spark.apache.org/docs/latest/sparkr.html
- **SparkR Tutorial at useR 2016**
  - https://databricks.com/blog/2016/07/07/sparkr-tutorial-at-user-2016.html
- **Spark R API**
  - https://spark.apache.org/docs/latest/api/R/

Questions?