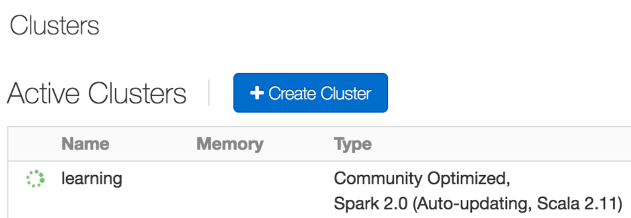
Databricks Community Edition

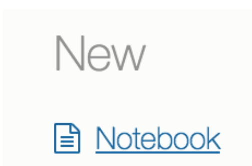
Step 1:

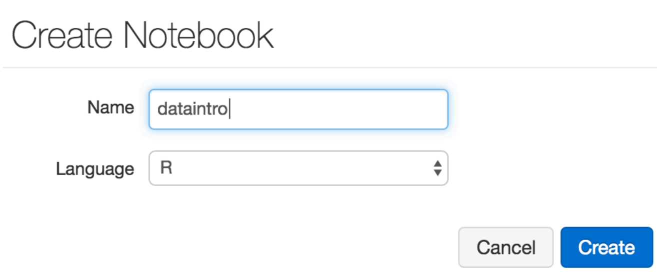
Log into https://community.cloud.databricks.com/

Step 2: Create a cluster:



Step 3: Create a new notebook:





Let's code!

We'll start with the built-in Faithful data set by converting it to a SparkR data frame:

```
dim(faithful)
class(faithful)
```

```
faithful_sark_df <- as.DataFrame(faithful)

# select only one column
head(select(faithful_sark_df, faithful_sark_df $eruptions))

# another way
head(select(faithful_sark_df, "eruptions"))

#Filtering in a dplyr syntax
head(filter(faithful_sark_df, faithful_sark_df$eruptions < 2))



# distinct
mtcars_spark_df <- as.DataFrame(mtcars)
head(select(mtcars_spark_df, mtcars_spark_df$cyl))
head(distinct(select(mtcars_spark_df, mtcars_spark_df$cyl)))

# filter
showDF(filter(mtcars_spark_df, mtcars_spark_df$hp > 200))

# arrange
head(arrange(mtcars_spark_df, desc(mtcars_spark_df$mpg)))

# summarize
head(summarize(groupBy(mtcars_spark_df, mtcars_spark_df$gear),
count=n(mtcars_spark_df$gear)))
```

Piping

```
# install.packages('magrittr')
library(magrittr)
library(magrittr)
groupBy(mtcars_spark_df, mtcars_spark_df$cyl) %>%
agg('mean_mpg' = mean(mtcars_spark_df$mpg)) %>%
arrange(mtcars_spark_df$cyl) %>%
head
```

Modeling – Linear Regression

Now let’s fit a Generalized Gaussian Linear Model (for more information see: https://spark.apache.org/docs/latest/sparkr.html#machine-learning):

```
iris_spark_df <- as.DataFrame(iris)
head(iris_spark_df)

# Fit a linear model over the dataset
model <- glm(Sepal_Length ~ Sepal_Width + Species, data=iris_spark_df,
family="gaussian")

# model coefficients are return in a similar format to R's native glm()
summary(model)
predictions <- predict(model, newData = iris_spark_df)
head(select(predictions, "Sepal_Length", "prediction"))
```

Let’s look at a more complex dataset:

http://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf

Diabetes study with 442 diabetes patients were measured on 10 baseline variables. Y is the response variable, a measure of disease progression one year after baseline.

```
install.packages('lars')
library(lars)

data(diabetes)
diabetes_all <- data.frame(cbind(diabetes$x, y = diabetes$y))
head(diabetes_all)
outcome_name <- 'y'

set.seed(1234)
splitIndex <- base::sample(nrow(diabetes_all), floor(0.75*nrow(diabetes_all)))
train_diabetes <- diabetes_all[ splitIndex,]
test_diabetes <- diabetes_all[-splitIndex,]



train_diabetes_sp <- as.DataFrame(train_diabetes)
test_diabetes_sp <- as.DataFrame(test_diabetes)



model <- glm(y~age+sex+bmi+map+tc+ldl+hdl+tch+ltg+glu,
data=train_diabetes_sp, family='gaussian')
summary(model)
```

```
predictions <- predict(model, newData = test_diabetes_sp)
names(predictions)

predictions_details <- select(predictions, predictions$label,
predictions$prediction)
```

The root mean squared error (RMSE) is often used to validate linear regression models. It is also very easy to calculate:

```
predictions_details <- select(predictions, predictions$label,
predictions$prediction)

predictions_details <- collect(predictions_details)

rmse <- sqrt(mean((predictions_details$label -
predictions_details$prediction)^2)) print(rmse)
```

Modeling – Logistic Regression

```
titanic <- read.csv('http://math.ucdenver.edu/RTutorial/titanic.txt',sep='\t')

# create a new feature out of names
titanic$Title <- ifelse(grepl('Mr ',titanic$Name),'Mr',
                        ifelse(grepl('Mrs ',titanic$Name),'Mrs',
                               ifelse(grepl('Miss',titanic$Name),'Miss','Nothing')))
titanic$Title <- as.factor(titanic$Title)

# impute missing ages
titanic$Age[is.na(titanic$Age)] <- median(titanic$Age, na.rm=T)

titanic <- titanic[c('PClass', 'Age', 'Sex', 'Title', 'Survived')]

# We binarize all non-numeric fields
charcolumns <- names(titanic[sapply(titanic, is.factor)])
for (colname in charcolumns) {
        print(paste(colname,length(unique(titanic[,colname]))))
        for (newcol in unique(titanic[,colname])) {
                if (!is.na(newcol))
                titanic[,paste0(colname,"_",newcol)] <-
ifelse(titanic[,colname]==newcol,1,0)
```

```
        }
        titanic <- titanic[,setdiff(names(titanic),colname)]
}

titanic$Survived <- as.factor(ifelse(titanic$Survived == 1, 'yes', 'no'))

set.seed(1234)
splitIndex <- base::sample ( nrow(titanic), floor(0.75*nrow(titanic)))
trainDF <- titanic[ splitIndex,]
testDF <- titanic[-splitIndex,]

# convert everything to spark data frames
train_titanic_sp <- as.DataFrame(trainDF)
test_titanic_sp <- as.DataFrame(testDF)
class(train_titanic_sp)


model <- spark.glm(train_titanic_sp,
Survived~Age+PClass_1st+PClass_2nd+Sex_female+Title_Miss+Title_Mr+Title_
Mrs, family = "binomial")

predictions <- predict(model, newData = test_titanic_sp )
names(predictions)


# Extract the label and the predictions:
predictions_details <- select(predictions, predictions$label,
predictions$prediction)

# make sql temp view
createOrReplaceTempView(predictions_details, "predictions_details")
```

Let's calculate the accuracy manually:

```
TP <- sql("SELECT count(label) FROM predictions_details WHERE label = 1
AND prediction > 0.5")
TP <- collect(TP)[[1]]
TN <- sql("SELECT count(label) FROM predictions_details WHERE label = 0
AND prediction < 0.5")
TN <- collect(TN)[[1]]
FP <- sql("SELECT count(label) FROM predictions_details WHERE label = 0
AND prediction > 0.5")
FP <- collect(FP)[[1]]
FN <- sql("SELECT count(label) FROM predictions_details WHERE label = 1
```

```
AND prediction < 0.5")
FN <- collect(FN)[[1]]
accuracy = (TP + TN)/(TP + TN + FP + FN) print(paste0(round(accuracy * 100,2),
'%’
```

Naive Bayes Model

(For more information see Spark documentation at https://spark.apache.org/docs/latest/sparkr.html#naive-bayes-model)

```
# Fit a Bernoulli naive Bayes model with spark.naiveBayes
titanic <- as.data.frame(Titanic)
titanicDF <- createDataFrame(titanic[titanic$Freq > 0, -5])
nbDF <- titanicDF
nbTestDF <- titanicDF
nbModel <- spark.naiveBayes(nbDF, Survived ~ Class + Sex + Age)

# Model summary summary(nbModel)
# Prediction
nbPredictions <- predict(nbModel, nbTestDF) showDF(nbPredictions)
```

KMeans Model

```
# KMeans Model
# Fit a k-means model with spark.kmeans
irisDF <- suppressWarnings(createDataFrame(iris))
kmeansDF <- irisDF
kmeansTestDF <- irisDF
kmeansModel <- spark.kmeans(kmeansDF, ~ Sepal_Length + Sepal_Width +
Petal_Length + Petal_Width,
k = 3)

# Model summary summary(kmeansModel)
# Get fitted result from the k-means model
showDF(fitted(kmeansModel))

# Prediction
kmeansPredictions <- predict(kmeansModel, kmeansTestDF)
```

```
showDF(kmeansPredictions)
```