H_2O + IBM DSX Demo

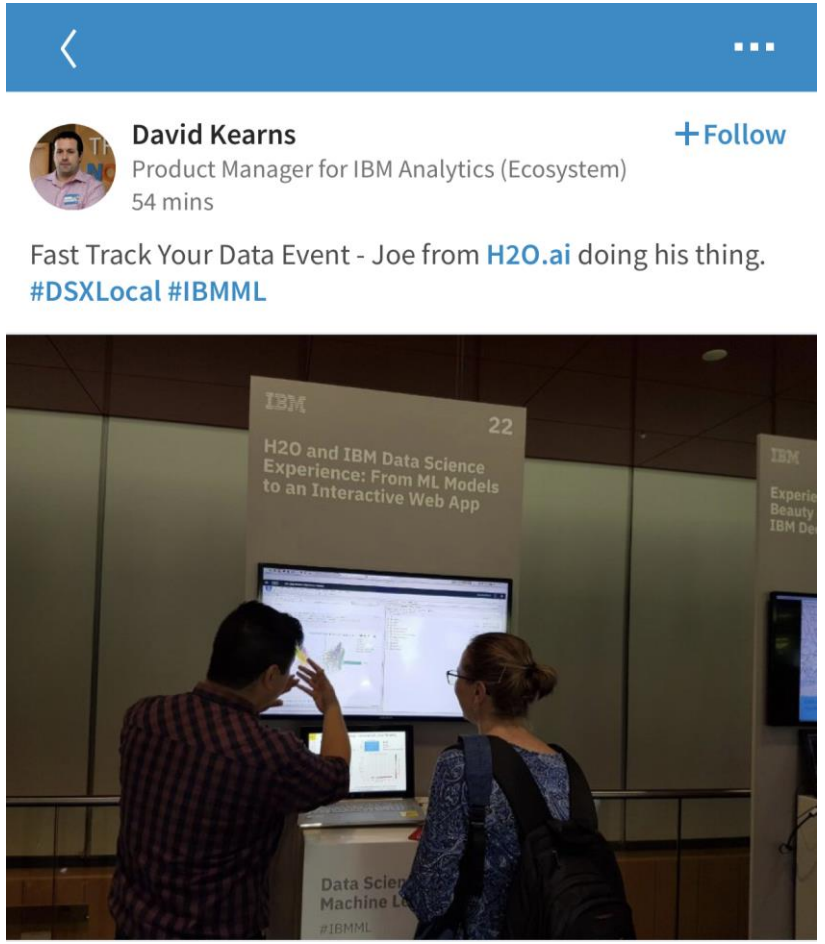
Introduction to First Demo

Jo-fai Chow

About this Presentation

- First H_2O + IBM DSX Demo
  - IBM Conference in Munich (June)
  - Collaboration
  - Data
  - Analysis

- Walk-through
  - R Notebook
  - Shiny App



H2O.ai

First H_2O + IBM DSX Demo

Quick Summary

- Objectives
  - Showing H_2O can work seamlessly with other R packages on IBM DSX.
  - Using an interactive demo to make machine learning look interesting.

- Code & Data
  - https://github.com/woobe/h2o_demo_for_ibm_dsx

Collaboration

IBM Fast Track Your Data (ML Conference) Munich

- H_2O.ai
  - Venkatesh Yadav (US)
  - Vinod Iyengar (US)
  - Jo-fai Chow (UK)
- IBM
  - Deepak Rangarao (US)
  - David Kearns (Ireland)
  - Marcus Boone (US)

H_2O.ai

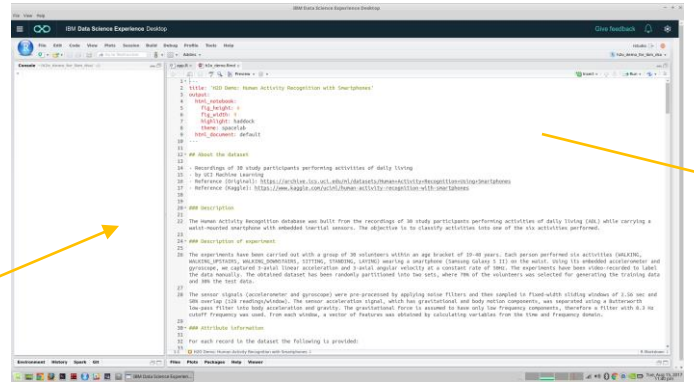
Data

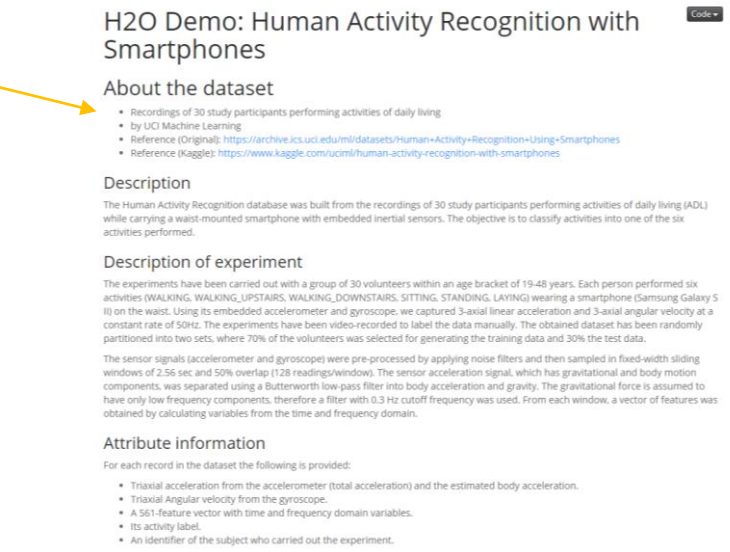
- Smartphone Sensor Data
  - 561 Features
  - 6 Activities
    - Walking
    - Walking Upstairs
    - Walking Downstairs
    - Sitting
    - Standing
    - Laying
  - Train (7k)
  - Test (3k)

H2O.ai

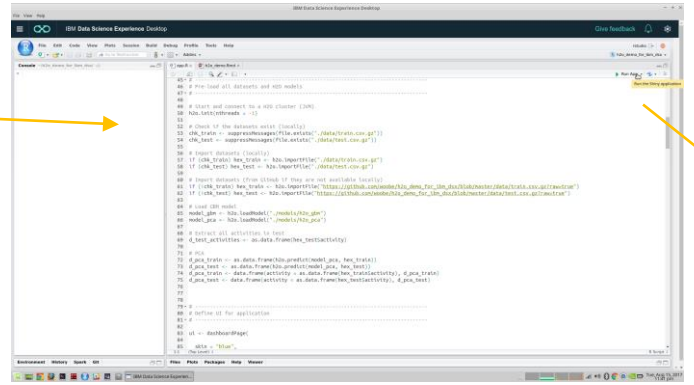
Analysis

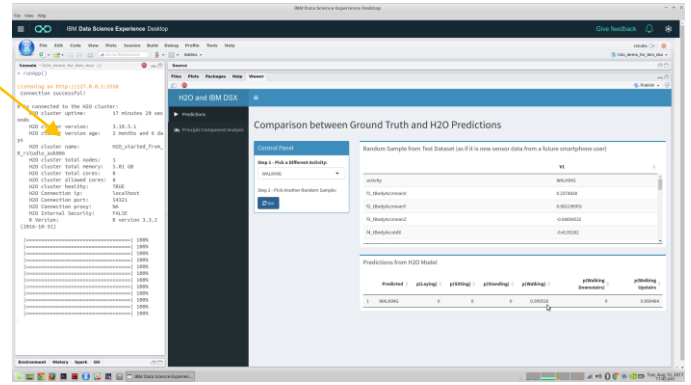
- R Notebook
  - h2o_demo.Rmd
  - h2o_demo.nb.html (rendered)





- Shiny App
  - app.R
  - Click **Run App** in RStudio





R Notebook (Walk-through)

H2O Demo: Human Activity Recognition with Smartphones

Code

h2o_demo.nb.html

About the dataset

- Recordings of 30 study participants performing activities of daily living
- by UCI Machine Learning
- Reference (Original): https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones
- Reference (Kaggle): https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones

Description

The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities performed.

Description of experiment

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

Attribute information

For each record in the dataset the following is provided:

- Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
- Triaxial Angular velocity from the gyroscope.
- A 561-feature vector with time and frequency domain variables.
- Its activity label.
- An identifier of the subject who carried out the experiment.

Background info about the dataset

Relevant papers

- Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine. International Workshop of Ambient Assisted Living (IWAAL 2012). Vitoria-Gasteiz, Spain. Dec 2012
- Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge L. Reyes-Ortiz. Energy Efficient Smartphone-Based Activity Recognition using Fixed-Point Arithmetic. Journal of Universal Computer Science. Special Issue in Ambient Assisted Living: Home Care. Volume 19, Issue 9. May 2013
- Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine. 4th International Workshop of Ambient Assited Living, IWAAL 2012, Vitoria-Gasteiz, Spain, December 3-5, 2012. Proceedings. Lecture Notes in Computer Science 2012, pp 216-223.

Step 1 - Start and connect to a H2O cluster (JVM)

R packages for this demo

```
# Pre-load all R packages
suppressPackageStartupMessages(library(data.table))
suppressPackageStartupMessages(library(h2o))
suppressPackageStartupMessages(library(plotly))
```

Hide

```
# Start and connect to a H2O cluster (JVM)
h2o.init(nthreads = -1)
```

```
H2O is not running yet, starting it now...

Note:  In case of errors look at the following log files:
    /tmp/Rtmp9896Ad/h2o_joe_started_from_r.out
    /tmp/Rtmp9896Ad/h2o_joe_started_from_r.err
```

Start and connect to a H_2O Cluster on IBM DSX

```
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-0ubuntu1.16.04.2-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

```
Starting H2O JVM and connecting: . Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         1 seconds 385 milliseconds
    H2O cluster version:        3.10.5.1
    H2O cluster version age:    10 days
    H2O cluster name:           H2O_started_from_R_joe_knw156
    H2O cluster total nodes:    1
    H2O cluster total memory:   5.21 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  8
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    R Version:                  R version 3.4.0 (2017-04-21)
```

Hide

```
h2o.no_progress() # disable progress bar in notebbok
```

Step 2 - Importing datasets into H2O

Import data

```
# Check if the datasets exist (locally)
chk_train <- suppressMessages(file.exists("./data/train.csv.gz"))
chk_test <- suppressMessages(file.exists("./data/test.csv.gz"))
# Import datasets (locally)
if (chk_train) hex_train <- h2o.importFile("./data/train.csv.gz")
if (chk_test) hex_test <- h2o.importFile("./data/test.csv.gz")
# Import datasets (from GitHub if they are not available locally)
if (!chk_train) hex_train <- h2o.importFile("https://github.com/woobe/h2o_demo_for_ibm_dsx/blob/master/data/train.csv.gz?raw=true")
if (!chk_test) hex_test <- h2o.importFile("https://github.com/woobe/h2o_demo_for_ibm_dsx/blob/master/data/test.csv.gz?raw=true")
```

Step 3 - Exploratory Analysis

Begin exploratory analysis

```
# Dimensions
# 'Train' dataset has 7352 rows and 562 columns
# 'Test' dataset has 2947 rows and 562 columns
dim(hex_train)
```

```
[1] 7352  562
```

Hide

```
dim(hex_test)
```

```
[1] 2947  562
```

Hide

```
# First few records
# First column is the label 'activity'
# Rest of the columns (V1 to V561) are sensors data
head(hex_train)
```

	activity <fctr>	f1_tBodyAccmeanX <dbl>	f2_tBodyAccmeanY <dbl>	f3_tBodyAccmeanZ <dbl>	f4_tBodyAccstdX <dbl>	f5_tBodyAccstdY <dbl>
1	STANDING	0.2885845	-0.02029417	-0.1329051	-0.9952786	-0.9831106
2	STANDING	0.2784188	-0.01641057	-0.1235202	-0.9982453	-0.9753002
3	STANDING	0.2796531	-0.01946716	-0.1134617	-0.9953796	-0.9671870
4	STANDING	0.2791739	-0.02620065	-0.1232826	-0.9960915	-0.9834027
5	STANDING	0.2766288	-0.01656966	-0.1153619	-0.9981386	-0.9808173
6	STANDING	0.2771988	-0.01009785	-0.1051373	-0.9973350	-0.9904868

```r
# Extract 'activity' columns for other graphics packages in R
d_activity_train <- as.data.frame(hex_train$activity)
d_activity_test <- as.data.frame(hex_test$activity)
# Count acitivity
d_freq_train <- as.data.frame(table(d_activity_train))
d_freq_test <- as.data.frame(table(d_activity_test))
d_freq <- merge(d_freq_train, d_freq_test, by.x = "d_activity_train", by.y = "d_activity_test", sort = FALSE)
colnames(d_freq) <- c("activity", "freq_train", "freq_test")
d_freq
```

Simple count

activity <fctr>	freq_train <int>	freq_test <int>
LAYING	1407	537
SITTING	1286	491
STANDING	1374	532
WALKING	1226	496
WALKING_DOWNSTAIRS	986	420
WALKING_UPSTAIRS	1073	471

6 rows

Hide

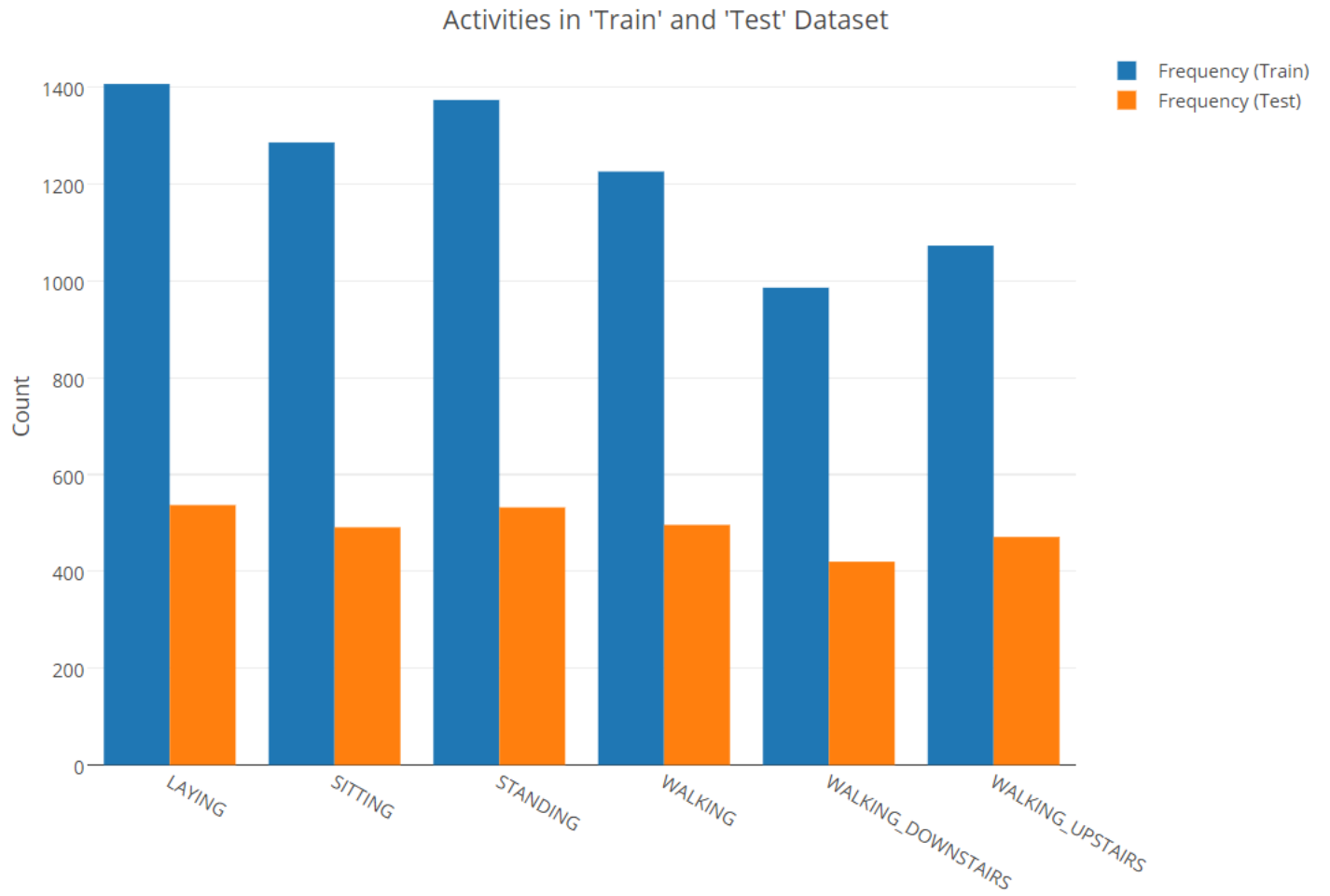
```r
# Visualize 'activity' in both 'train' and 'test'
p <- plot_ly(d_freq, x = ~activity, y = ~freq_train, type = 'bar', name = 'Frequency (Train)') %>%
  add_trace(y = ~freq_test, name = 'Frequency (Test)') %>%
  layout(title = "Activities in 'Train' and 'Test' Dataset") %>%
  layout(yaxis = list(title = 'Count'), xaxis = list(title = "")) %>%
  layout(margin = list(b = 90)) %>%
  layout(barmode = "group")
p
```

Activities in 'Train' and 'Test' Dataset

Hide

```
# Visualize 'activity' in both 'train' and 'test'
p <- plot_ly(d_freq, x = ~activity, y = ~freq_train, type = 'bar', name = 'Frequency (Train)') %>%
  add_trace(y = ~freq_test, name = 'Frequency (Test)') %>%
  layout(title = "Activities in 'Train' and 'Test' Dataset") %>%
  layout(yaxis = list(title = 'Count'), xaxis = list(title = "")) %>%
  layout(margin = list(b = 90)) %>%
  layout(barmode = "group")
p
```

Visualize the counts



Hide

```
# Look at relationship between sensor data `f1_tBodyAccmeanX` and activity
d_f1 <- data.frame(V1_train = as.data.frame(hex_train$f1_tBodyAccmeanX), activity = as.data.frame(hex_train$activity))
```

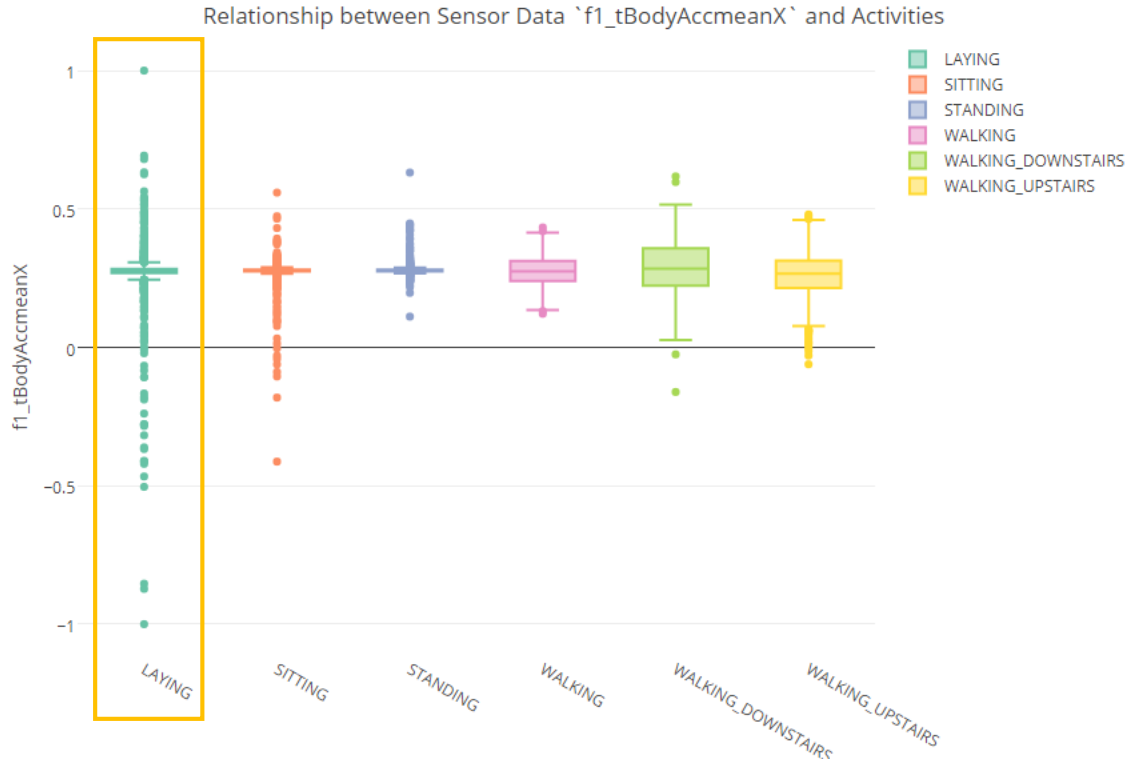
```
# Look at relationship between sensor data `f1_tBodyAccmeanX` and activity
d_f1 <- data.frame(V1_train = as.data.frame(hex_train$f1_tBodyAccmeanX), activity = as.data.frame(hex_train$activity))
head(d_f1)
```

	f1_tBodyAccmeanX <dbl>	activity <fctr>
1	0.2885845	STANDING
2	0.2784188	STANDING
3	0.2796531	STANDING
4	0.2791739	STANDING
5	0.2766288	STANDING
6	0.2771988	STANDING

6 rows

Hide

```
p <- plot_ly(d_f1, y = ~f1_tBodyAccmeanX, color = ~activity, type = "box") %>%
     layout(title = "Relationship between Sensor Data `f1_tBodyAccmeanX` and Activities") %>%
     layout(yaxis = list(title = 'f1_tBodyAccmeanX'), xaxis = list(title = "")) %>%
     layout(margin = list(b = 90))
p
```



Look at first feature (one of the sensor measurements)

The distribution of values for **laying** is quite different to other activities. This is understandable as the phone movement is quite different when you use your smartphone while lying down.

Hide

```
# Principal Component Analysis
# 95% of variance in original data captured by first five principal components
model_pca <- h2o.prcomp(training_frame = hex_train,
                    x = 2:562,
                    model_id = "h2o_pca",
                    k = 5)
model_pca
```

```
Model Details:
==============

H2ODimReductionModel: pca
Model ID:  h2o_pca
Importance of components:
                            pc1      pc2      pc3      pc4      pc5
Standard deviation     16.190113 4.587733 1.570451 1.441637 0.980662
Proportion of Variance  0.864715 0.069434 0.008136 0.006856 0.003173
Cumulative Proportion   0.864715 0.934148 0.942285 0.949141 0.952313


H2ODimReductionMetrics: pca

No model metrics available for PCA
```

… but we won't be able to explore all 561 features one by one …

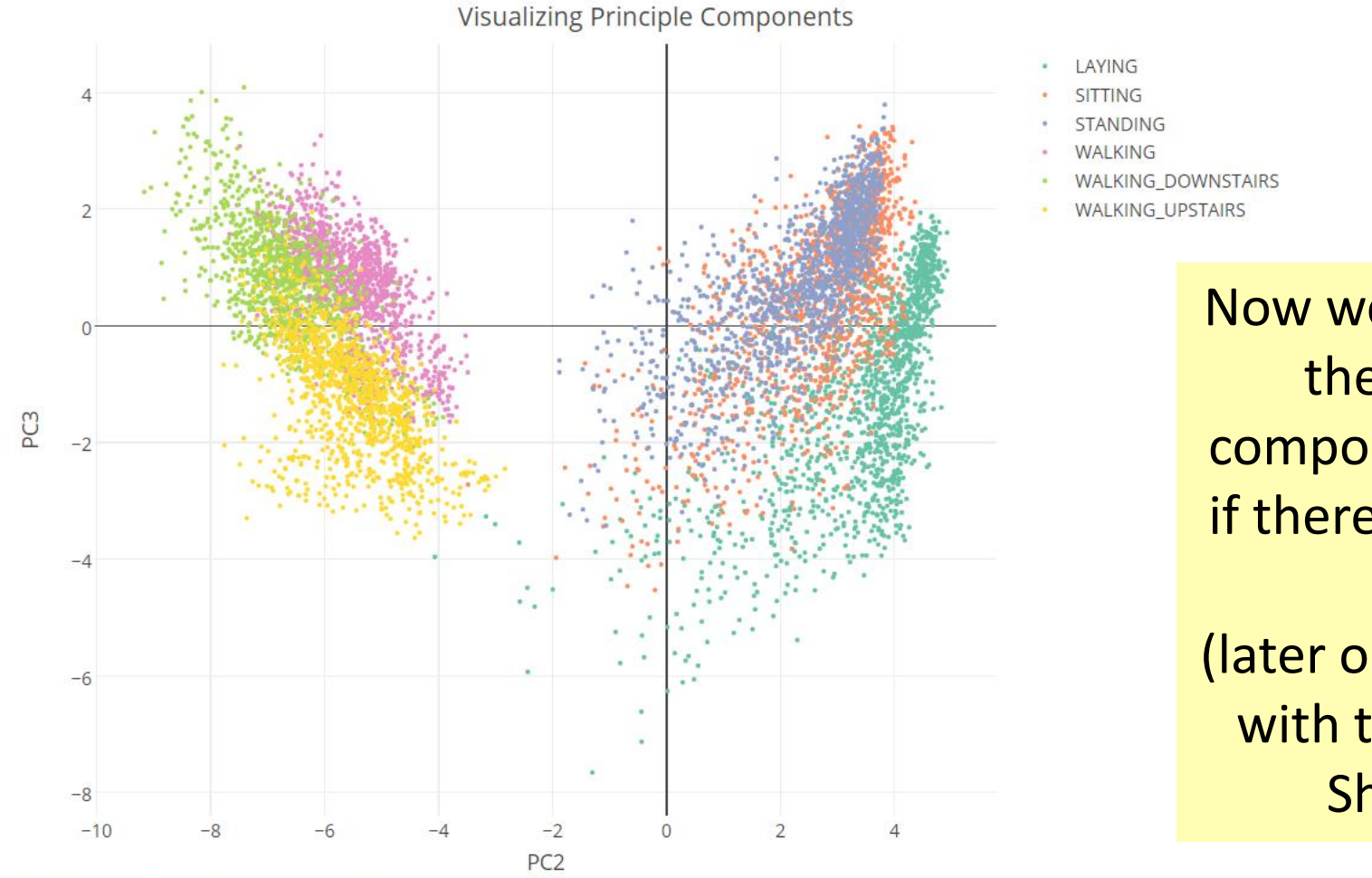
… so we use principle component analysis in H_2O to reduce the dimensionality from 561 to 5.

```
# Visualize principle components with activity labels
d_pca <- as.data.frame(h2o.predict(model_pca, hex_train))
d_pca <- data.frame(d_pca, as.data.frame(hex_train$activity))
head(d_pca)
```

	PC1 <dbl>	PC2 <dbl>	PC3 <dbl>	PC4 <dbl>	PC5 <dbl>	activity <fctr>
1	19.17406	3.199837	0.4452233	1.8616369	-0.58875544	STANDING
2	19.31925	3.086634	0.9333813	1.8616152	-0.30543163	STANDING
3	19.39820	2.946190	1.4208175	1.7053176	0.05042238	STANDING
4	19.34186	3.280947	1.8193346	1.2823227	-0.61599070	STANDING
5	19.38915	3.342314	2.1122827	1.2133754	-0.32844535	STANDING
6	19.48550	3.214922	2.2482216	0.8627183	0.02153694	STANDING

Hide

```
p <- plot_ly(data = d_pca, x = ~PC2, y = ~PC3, color = ~activity,
             type = "scatter", mode = "markers", marker = list(size = 3)) %>%
     layout(title = "Visualizing Principle Components")
p
```



Now we can visualize the principle components and see if there is any cluster

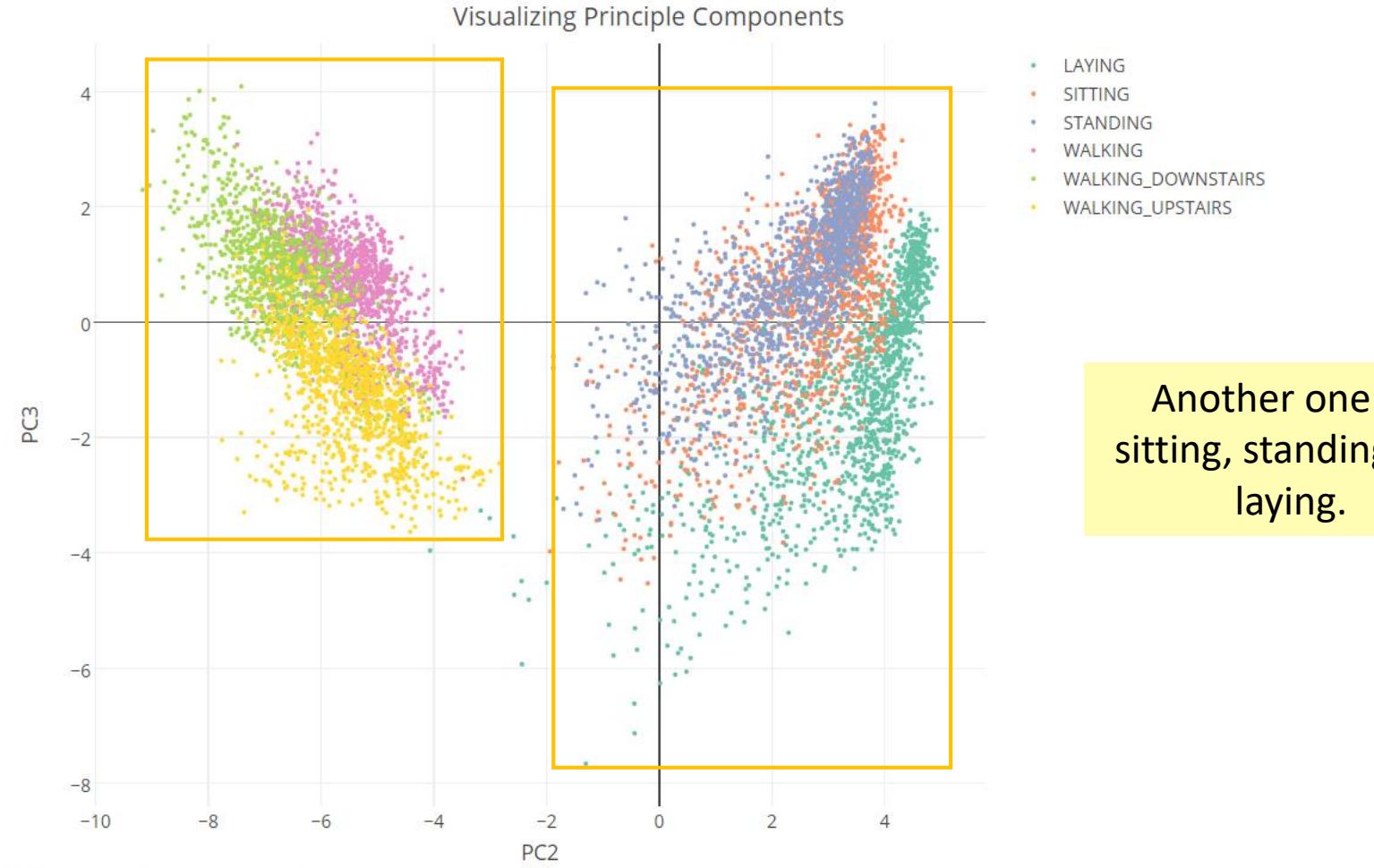
(later on you can play with this using the Shiny App)

From the graph above, we can see that:

- it could be difficult to distinguish between **Standing** and **Sitting** as there are large overlaps in their sensor data.
- **Laying** has its own cluster so it should be easy to classify.
- **Walking**, **Walking Upstairs** and **Walking Downstairs** are understandably closer to each other yet they are quite different to **Sitting**, **Standing** and **Laying**.

Hide

```
p <- plot_ly(data = d_pca, x = ~PC2, y = ~PC3, color = ~activity,
             type = "scatter", mode = "markers", marker = list(size = 3)) %>%
     layout(title = "Visualizing Principle Components")
p
```



First, we can see two major clusters. One for movement (walking, walking upstairs and walking downstairs)

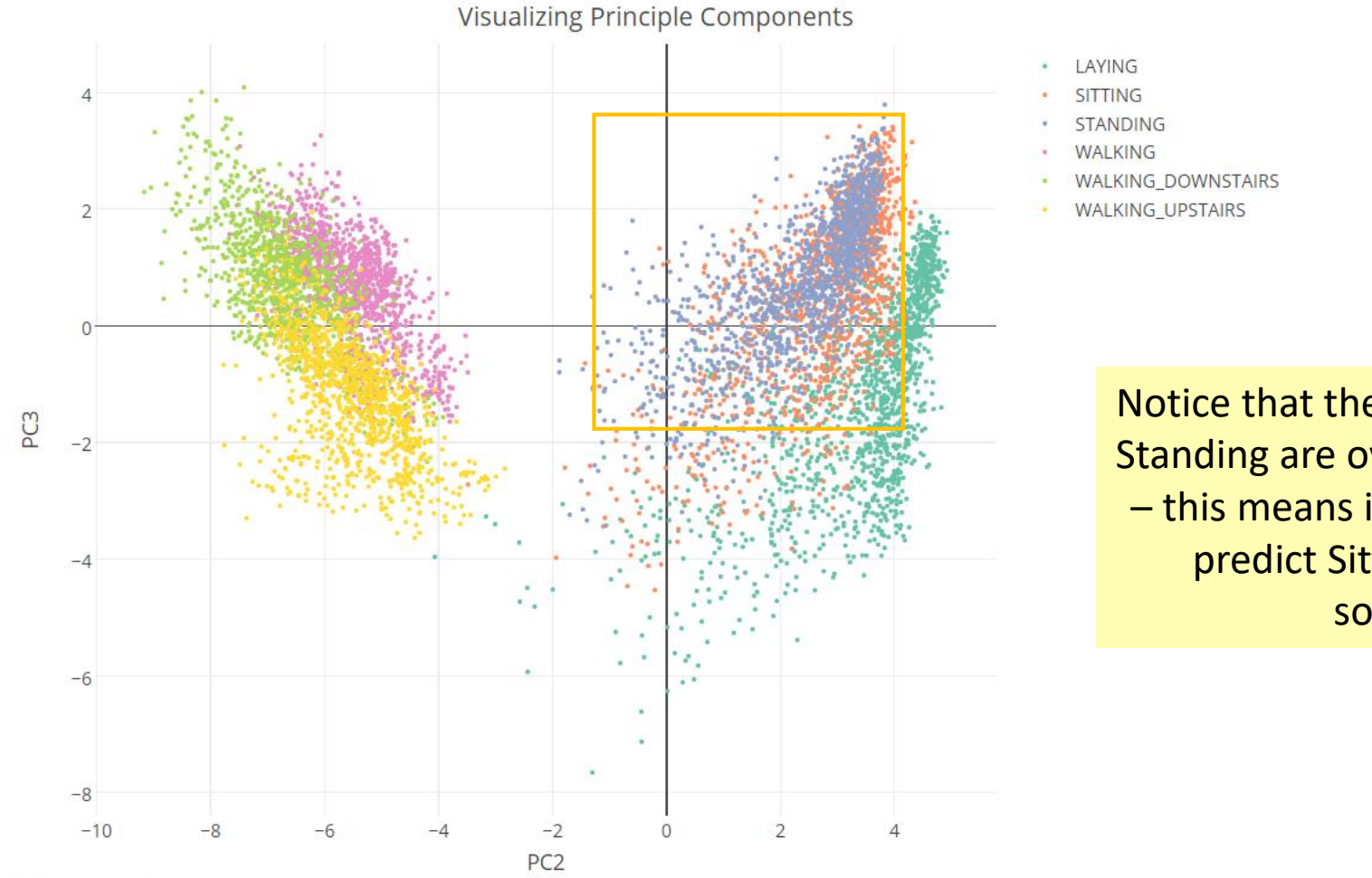
Another one for sitting, standing and laying.

From the graph above, we can see that:

- it could be difficult to distinguish between **Standing** and **Sitting** as there are large overlaps in their sensor data.
- **Laying** has its own cluster so it should be easy to classify.
- **Walking**, **Walking Upstairs** and **Walking Downstairs** are understandably closer to each other yet they are quite different to **Sitting**, **Standing** and **Laying**.

Hide

```
p <- plot_ly(data = d_pca, x = ~PC2, y = ~PC3, color = ~activity,
             type = "scatter", mode = "markers", marker = list(size = 3)) %>%
     layout(title = "Visualizing Principle Components")
p
```



Notice that the cluster of Sitting and Standing are overlapping each other – this means it could be difficult to predict Sitting and Standing sometimes.

From the graph above, we can see that:

- it could be difficult to distinguish between **Standing** and **Sitting** as there are large overlaps in their sensor data.
- **Laying** has its own cluster so it should be easy to classify.
- **Walking**, **Walking Upstairs** and **Walking Downstairs** are understandably closer to each other yet they are quite different to **Sitting**, **Standing** and **Laying**.

Step 4 - Build and evalutate a predictive model using H2O's Gradient Boosting Machine (GBM) algorithm

Hide

```
# Define target and features for model training
target <- "activity"
features <- setdiff(colnames(hex_train), target) # i.e. using the records of all 561 sensors
```

Hide

```
# Build a GBM model with cross-validation and early stopping
model <- h2o.gbm(x = features,
                 y = target,
                 training_frame = hex_train,
                 model_id = "h2o_gbm",
                 ntrees = 500,
                 learn_rate = 0.05,
                 learn_rate_annealing = 0.999,
                 max_depth = 7,
                 sample_rate = 0.9,
                 col_sample_rate = 0.9,
                 nfolds = 3,
                 fold_assignment = "Stratified",
                 stopping_metric = "logloss",
                 stopping_rounds = 5,
                 score_tree_interval = 10,
                 seed = 1234)
```

Hide

```
# Print out model summary
model
```

```
Model Details:
==============

H2OMultinomialModel: gbm
Model ID:  h2o_gbm
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth min_leaves max_leaves mean_lea
ves
1             290                    1740            1459657         1         7    6.99655          2         83    55.64
828


H2OMultinomialMetrics: gbm
** Reported on training data. **

Training Set Metrics:
```

After some initial exploratory analysis, we can build a GBM model

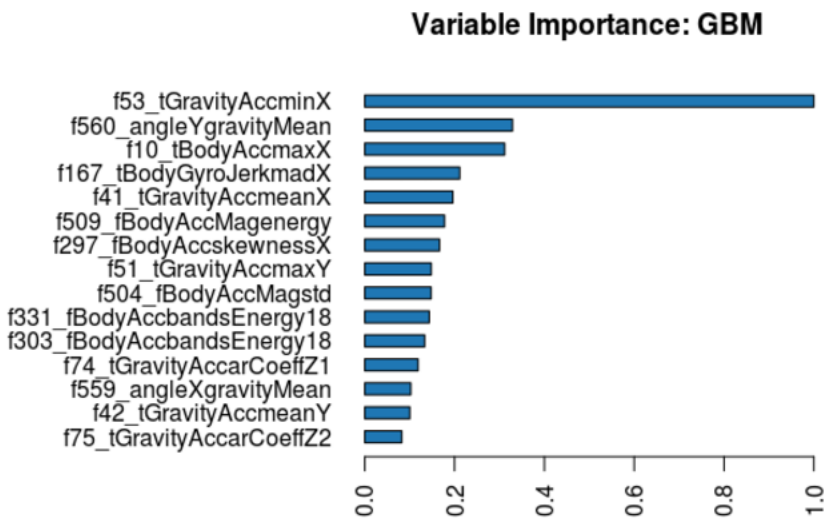
Hide

```
# Look at variable importance in this GBM model
h2o.varimp(model)
```

```
Variable Importances:
               variable relative_importance scaled_importance percentage
1    f53_tGravityAccminX         9839.461914          1.000000   0.204901
2 f560_angleYgravityMean         3236.193359          0.328899   0.067392
3       f10_tBodyAccmaxX         3065.950928          0.311597   0.063847
4 f167_tBodyGyroJerkmadX         2081.444580          0.211540   0.043345
5   f41_tGravityAccmeanX         1930.086060          0.196158   0.040193

---
                        variable relative_importance scaled_importance percentage
556   f494_fBodyGyrobandsEnergy4148            0.008722          0.000001   0.000000
557   f471_fBodyGyrobandsEnergy3348            0.007785          0.000001   0.000000
558         f99_tBodyAccJerkenergyZ            0.004651          0.000000   0.000000
559         f98_tBodyAccJerkenergyY            0.003739          0.000000   0.000000
560        f362_fBodyAccJerkenergyY            0.002965          0.000000   0.000000
561 f548_fBodyBodyGyroJerkMagenergy            0.002210          0.000000   0.000000
```

Hide

```
# Visualize variable importance
h2o.varimp_plot(model, num_of_features = 15)
```



We can look at variable importance (see which features are the main drivers)

Step 5 - Make and evalutate predictions

We can use the model to make predictions

Hide

```
# Make predictions
yhat_test <- h2o.predict(model, hex_test)
head(yhat_test)
```

predict <fctr>	LAYING <dbl>	SITTING <dbl>	STANDING <dbl>	WALKING <dbl>	WALKING_DOWNSTAIRS <dbl>	WALKING_UPSTAIRS <dbl>
1 STANDING	8.627448e-08	7.071596e-06	0.9999925	1.011563e-07	9.650777e-08	1.042965e-07
2 STANDING	1.300119e-08	4.897534e-07	0.9999995	1.457190e-08	1.443481e-08	1.518905e-08
3 STANDING	1.201143e-08	9.275131e-06	0.9999907	1.302589e-08	1.357375e-08	1.392128e-08
4 STANDING	1.151841e-08	1.617590e-06	0.9999983	1.249811e-08	1.302733e-08	1.338994e-08
5 STANDING	1.128372e-08	8.109414e-07	0.9999991	1.223912e-08	1.283396e-08	1.312683e-08
6 STANDING	2.985663e-08	7.707183e-06	0.9999922	3.320442e-08	3.409307e-08	3.472674e-08

6 rows

Hide

```
# Evaluate predictions
h2o.performance(model, newdata = hex_test)
```

H2OMultinomialMetrics: glrm

```
# Evaluate predictions
h2o.performance(model, newdata = hex_test)
```

We can look at the performance on unseen test dataset

```
H2OMultinomialMetrics: gbm

Test Set Metrics:
=====================

MSE: (Extract with `h2o.mse`) 0.06358653
RMSE: (Extract with `h2o.rmse`) 0.2521637
Logloss: (Extract with `h2o.logloss`) 0.3214465
Mean Per-Class Error: 0.07575778
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
                  LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS  Error          Rate
LAYING               537       0        0       0                  0                0 0.0000 =     0 / 537
SITTING                0     401       89       0                  0                1 0.1833 =    90 / 491
STANDING               0      40      492       0                  0                0 0.0752 =    40 / 532
WALKING                0       0        0     481                  4               11 0.0302 =    15 / 496
WALKING_DOWNSTAIRS     0       0        0      10                378               32 0.1000 =    42 / 420
WALKING_UPSTAIRS       0       1        0      24                  6              440 0.0658 =    31 / 471
Totals               537     442      581     515                388              484 0.0740 = 218 / 2,947

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
=======================================================================
Top-6 Hit Ratios:
  k hit_ratio
1 1  0.926026
2 2  0.990159
3 3  0.994231
4 4  0.996607
5 5  1.000000
6 6  1.000000
```

As expected: - It is easy to classify **Laying** - It is difficult to distinguish between **Sitting** and **Standing**

```
# Evaluate predictions
h2o.performance(model, newdata = hex_test)
```

As expected, Laying is easy to predict (with no error on test data)

```
H2OMultinomialMetrics: gbm

Test Set Metrics:
=====================

MSE: (Extract with `h2o.mse`) 0.06358653
RMSE: (Extract with `h2o.rmse`) 0.2521637
Logloss: (Extract with `h2o.logloss`) 0.3214465
Mean Per-Class Error: 0.07575778
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
                   LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS  Error          Rate
LAYING                537       0        0       0                  0                0 0.0000 =     0 / 537
SITTING                 0     401       89       0                  0                1 0.1833 =    90 / 491
STANDING                0      40      492       0                  0                0 0.0752 =    40 / 532
WALKING                 0       0        0     481                  4               11 0.0302 =    15 / 496
WALKING_DOWNSTAIRS      0       0        0      10                378               32 0.1000 =    42 / 420
WALKING_UPSTAIRS        0       1        0      24                  6              440 0.0658 =    31 / 471
Totals                537     442      581     515                388              484 0.0740 = 218 / 2,947

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
=======================================================================
Top-6 Hit Ratios:
  k hit_ratio
1 1  0.926026
2 2  0.990159
3 3  0.994231
4 4  0.996607
5 5  1.000000
6 6  1.000000
```

As expected: - It is easy to classify **Laying** - It is difficult to distinguish between **Sitting** and **Standing**

```
# Evaluate predictions
h2o.performance(model, newdata = hex_test)
```

```
H2OMultinomialMetrics: gbm

Test Set Metrics:
=====================

MSE: (Extract with `h2o.mse`) 0.06358653
RMSE: (Extract with `h2o.rmse`) 0.2521637
Logloss: (Extract with `h2o.logloss`) 0.3214465
Mean Per-Class Error: 0.07575778
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
                  LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS  Error          Rate
LAYING               537       0        0       0                  0                0 0.0000 =     0 / 537
SITTING                0     401       89       0                  0                1 0.1833 =    90 / 491
STANDING               0      40      492       0                  0                0 0.0752 =    40 / 532
WALKING                0       0        0     481                  4               11 0.0302 =    15 / 496
WALKING_DOWNSTAIRS     0       0        0      10                378               32 0.1000 =    42 / 420
WALKING_UPSTAIRS       0       1        0      24                  6              440 0.0658 =    31 / 471
Totals               537     442      581     515                388              484 0.0740 = 218 / 2,947

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
=======================================================================
Top-6 Hit Ratios:
  k hit_ratio
1 1  0.926026
2 2  0.990159
3 3  0.994231
4 4  0.996607
5 5  1.000000
6 6  1.000000
```

Yet, it is quite easy to get confused between **Sitting** and **Standing** (as there is a huge overlap of sensor measurements)

As expected: - It is easy to classify **Laying** - It is difficult to distinguish between **Sitting** and **Standing**

```
# Evaluate predictions
h2o.performance(model, newdata = hex_test)
```

```
H2OMultinomialMetrics: gbm

Test Set Metrics:
=====================

MSE: (Extract with `h2o.mse`) 0.06358653
RMSE: (Extract with `h2o.rmse`) 0.2521637
Logloss: (Extract with `h2o.logloss`) 0.3214465
Mean Per-Class Error: 0.07575778
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
                  LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS  Error          Rate
LAYING               537       0        0       0                  0                0 0.0000 =     0 / 537
SITTING                0     401       89       0                  0                1 0.1833 =    90 / 491
STANDING               0      40      492       0                  0                0 0.0752 =    40 / 532
WALKING                0       0        0     481                  4               11 0.0302 =    15 / 496
WALKING_DOWNSTAIRS     0       0        0      10                378               32 0.1000 =    42 / 420
WALKING_UPSTAIRS       0       1        0      24                  6              440 0.0658 =    31 / 471
Totals               537     442      581     515                388              484 0.0740 = 218 / 2,947

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
=======================================================================
Top-6 Hit Ratios:
  k hit_ratio
1 1  0.926026
2 2  0.990159
3 3  0.994231
4 4  0.996607
5 5  1.000000
6 6  1.000000
```

Similarly, there is some confusion between different walking movements.

As expected: - It is easy to classify **Laying** - It is difficult to distinguish between **Sitting** and **Standing**

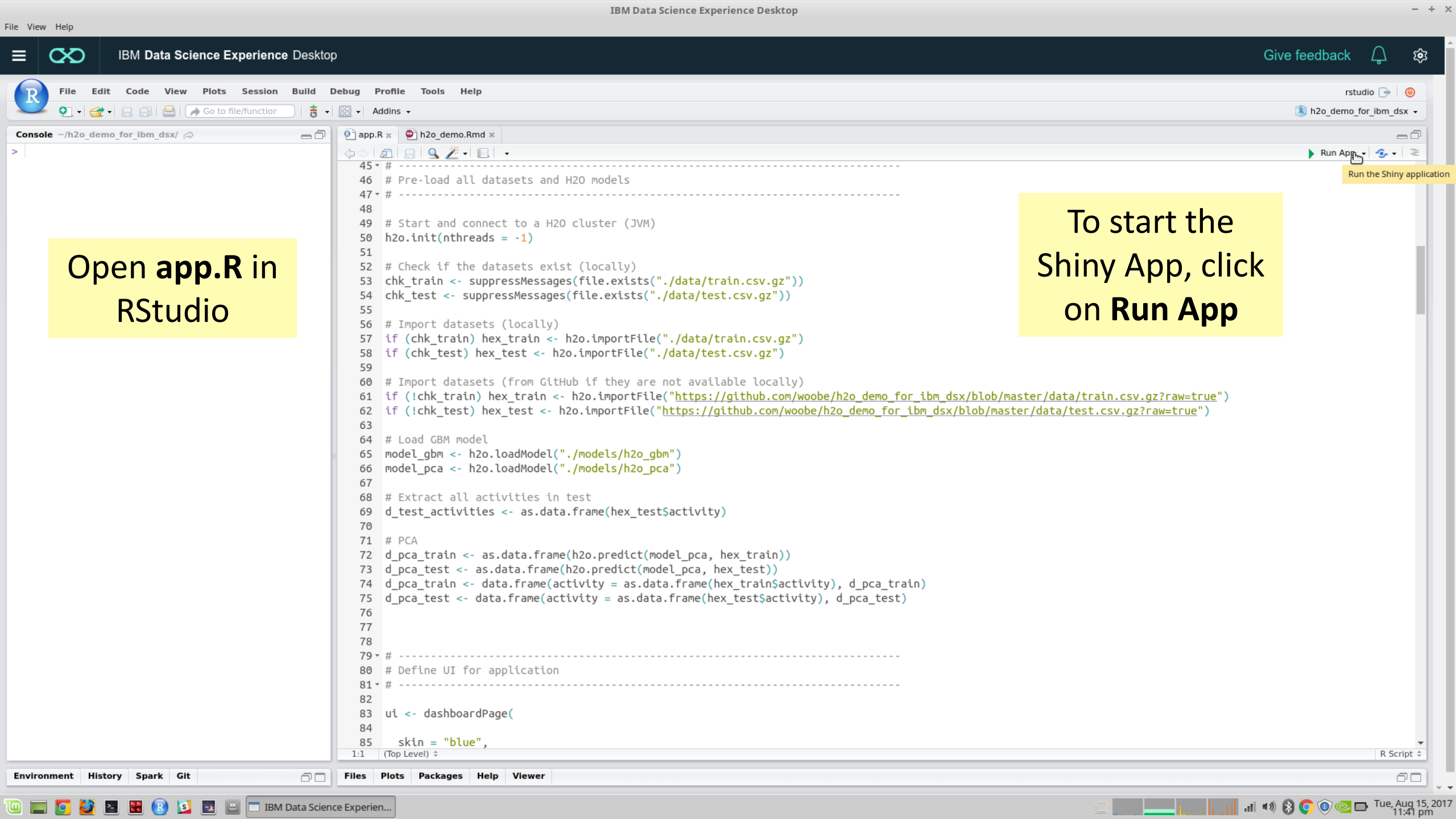
```
# Evaluate predictions
h2o.performance(model, newdata = hex_test)
```

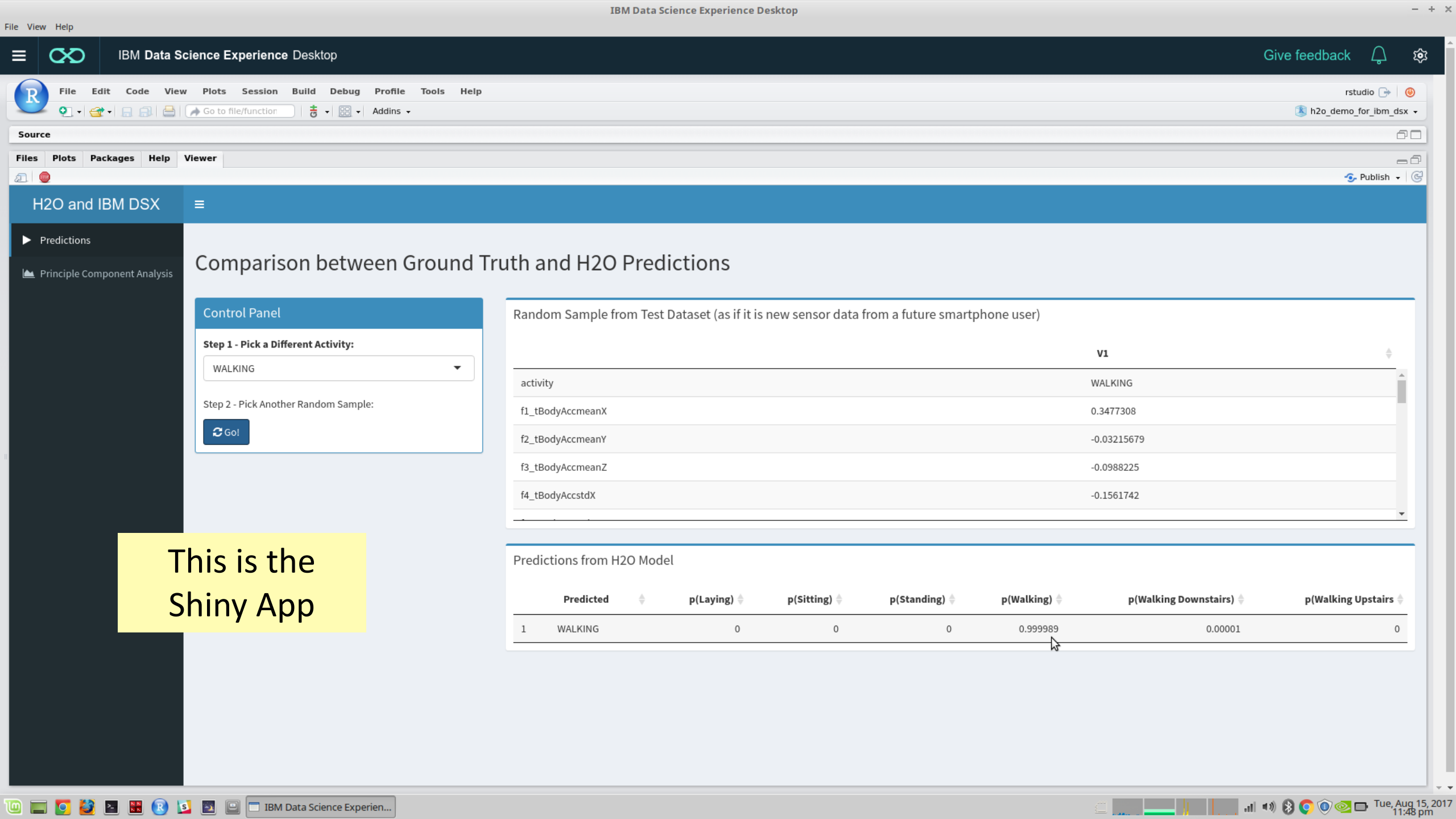
```
H2OMultinomialMetrics: gbm

Test Set Metrics:
=====================

MSE: (Extract with `h2o.mse`) 0.06358653
RMSE: (Extract with `h2o.rmse`) 0.2521637
Logloss: (Extract with `h2o.logloss`) 0.3214465
Mean Per-Class Error: 0.07575778
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
                  LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS  Error          Rate
LAYING               537       0        0       0                  0                0 0.0000 =     0 / 537
SITTING                0     401       89       0                  0                1 0.1833 =    90 / 491
STANDING               0      40      492       0                  0                0 0.0752 =    40 / 532
WALKING                0       0        0     481                  4               11 0.0302 =    15 / 496
WALKING_DOWNSTAIRS     0       0        0      10                378               32 0.1000 =    42 / 420
WALKING_UPSTAIRS       0       1        0      24                  6              440 0.0658 =    31 / 471
Totals               537     442      581     515                388              484 0.0740 = 218 / 2,947

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
=======================================================================
Top-6 Hit Ratios:
  k hit_ratio
1 1  0.926026
2 2  0.990159
3 3  0.994231
4 4  0.996607
5 5  1.000000
6 6  1.000000
```

Overall, it is still a decent model with 7.4% error on test data

As expected: - It is easy to classify **Laying** - It is difficult to distinguish between **Sitting** and **Standing**

Shiny App (Walk-through)

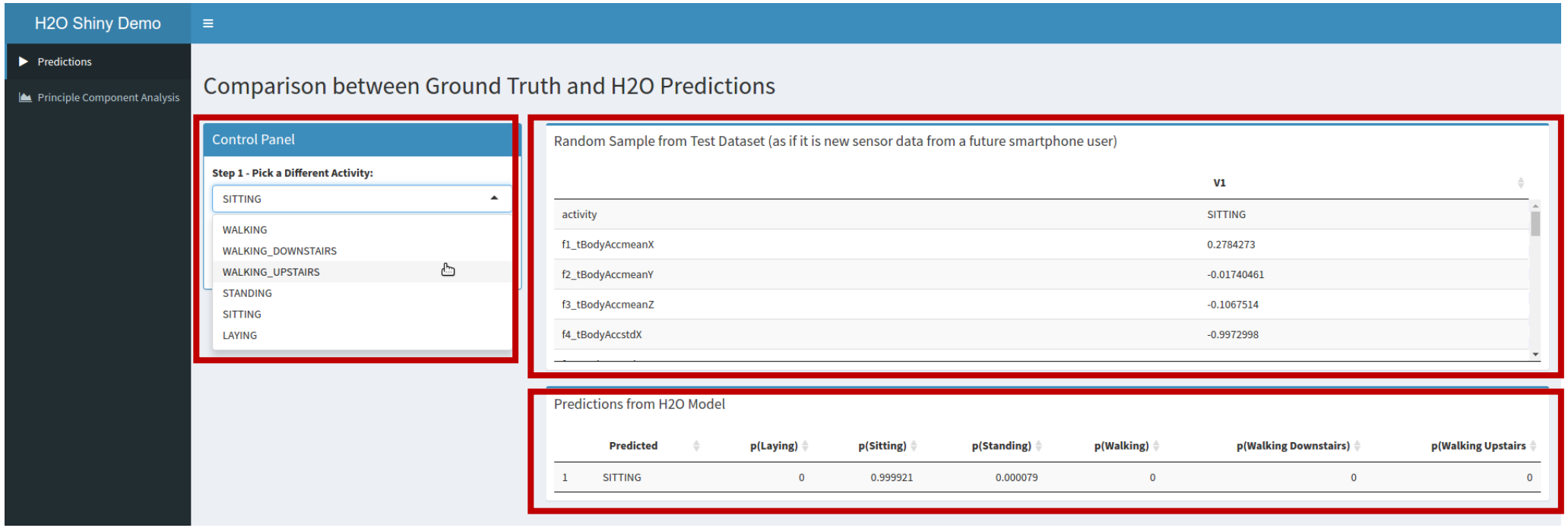

IBM Data Science Experience Desktop
File View Help
IBM Data Science Experience Desktop
Give feedback
File Edit Code View Plots Session Build Debug Profile Tools Help
rstudio
Addins
h2o_demo_for_ibm_dsx
Console ~/h2o_demo_for_ibm_dsx/
app.R
h2o_demo.Rmd
Run App
Run the Shiny application
45 # --
46 # Pre-load all datasets and H2O models
47 # --
48
49 # Start and connect to a H2O cluster (JVM)
50 h2o.init(nthreads = -1)
51
52 # Check if the datasets exist (locally)
53 chk_train <- suppressMessages(file.exists("./data/train.csv.gz"))
54 chk_test <- suppressMessages(file.exists("./data/test.csv.gz"))
55
56 # Import datasets (locally)
57 if (chk_train) hex_train <- h2o.importFile("./data/train.csv.gz")
58 if (chk_test) hex_test <- h2o.importFile("./data/test.csv.gz")
59
60 # Import datasets (from GitHub if they are not available locally)
61 if (!chk_train) hex_train <- h2o.importFile("https://github.com/woobe/h2o_demo_for_ibm_dsx/blob/master/data/train.csv.gz?raw=true")
62 if (!chk_test) hex_test <- h2o.importFile("https://github.com/woobe/h2o_demo_for_ibm_dsx/blob/master/data/test.csv.gz?raw=true")
63
64 # Load GBM model
65 model_gbm <- h2o.loadModel("./models/h2o_gbm")
66 model_pca <- h2o.loadModel("./models/h2o_pca")
67
68 # Extract all activities in test
69 d_test_activities <- as.data.frame(hex_test$activity)
70
71 # PCA
72 d_pca_train <- as.data.frame(h2o.predict(model_pca, hex_train))
73 d_pca_test <- as.data.frame(h2o.predict(model_pca, hex_test))
74 d_pca_train <- data.frame(activity = as.data.frame(hex_train$activity), d_pca_train)
75 d_pca_test <- data.frame(activity = as.data.frame(hex_test$activity), d_pca_test)
76
77
78
79 # --
80 # Define UI for application
81 # --
82
83 ui <- dashboardPage(
84
85 skin = "blue",
1:1 (Top Level)
R Script
Open app.R in RStudio
To start the Shiny App, click on Run App
Environment History Spark Git
Files Plots Packages Help Viewer
IBM Data Science Experien...
Tue, Aug 15, 2017 11:41 pm


IBM Data Science Experience Desktop
File View Help
IBM Data Science Experience Desktop
Give feedback
File Edit Code View Plots Session Build Debug Profile Tools Help
rstudio
Go to file/function
Addins
h2o_demo_for_ibm_dsx
Source
Files Plots Packages Help Viewer
Publish
H2O and IBM DSX
Predictions
Principle Component Analysis
Comparison between Ground Truth and H2O Predictions
Control Panel
Step 1 - Pick a Different Activity:
WALKING
Step 2 - Pick Another Random Sample:
Go!
Random Sample from Test Dataset (as if it is new sensor data from a future smartphone user)
V1
activity WALKING
f1_tBodyAccmeanX 0.3477308
f2_tBodyAccmeanY -0.03215679
f3_tBodyAccmeanZ -0.0988225
f4_tBodyAccstdX -0.1561742
Predictions from H2O Model
Predicted p(Laying) p(Sitting) p(Standing) p(Walking) p(Walking Downstairs) p(Walking Upstairs
1 WALKING 0 0 0 0.999989 0.00001 0
This is the Shiny App
IBM Data Science Experien...
Tue, Aug 15, 2017 11:48 pm

H_2O GBM -> Shiny

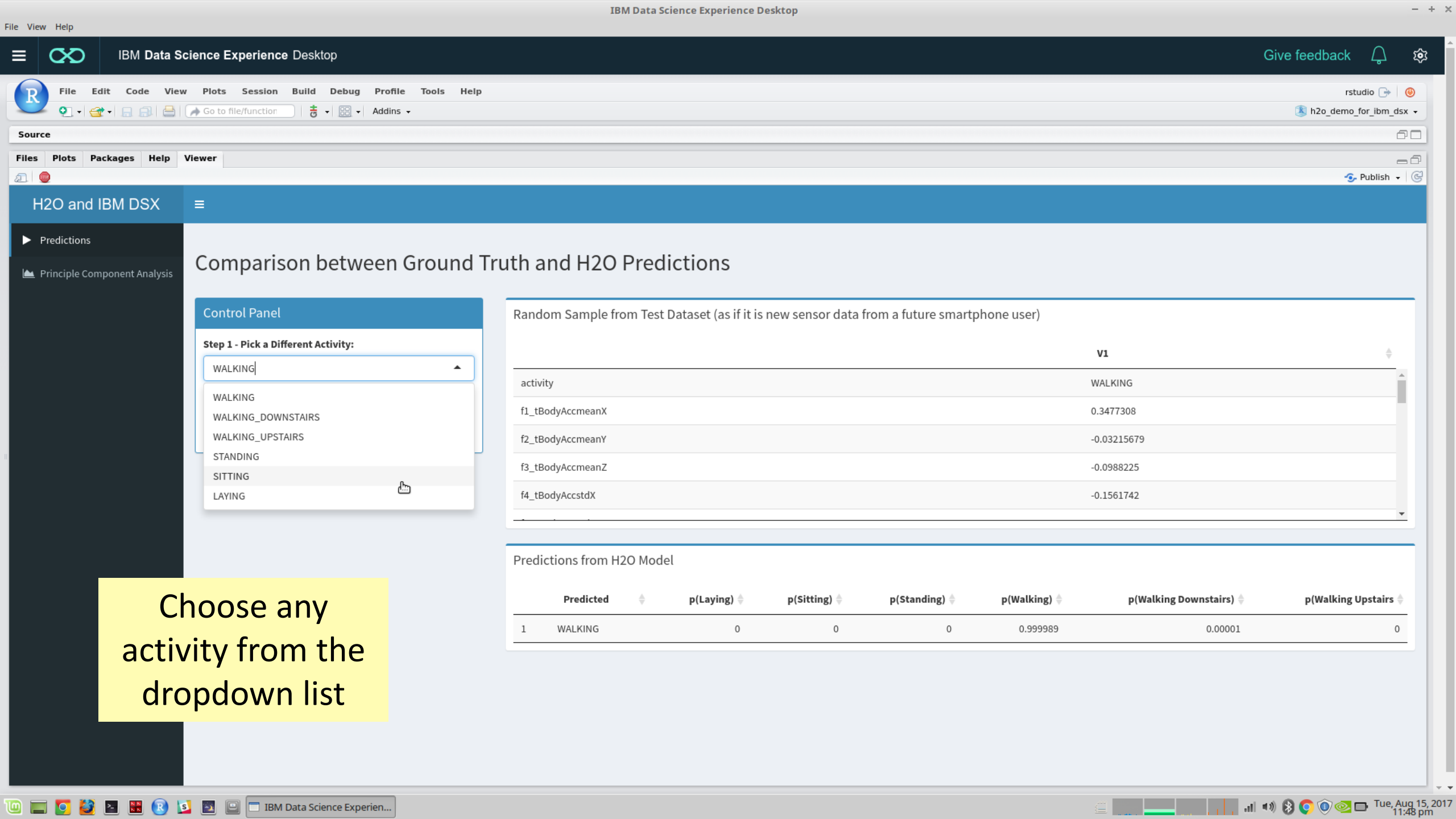
Controls

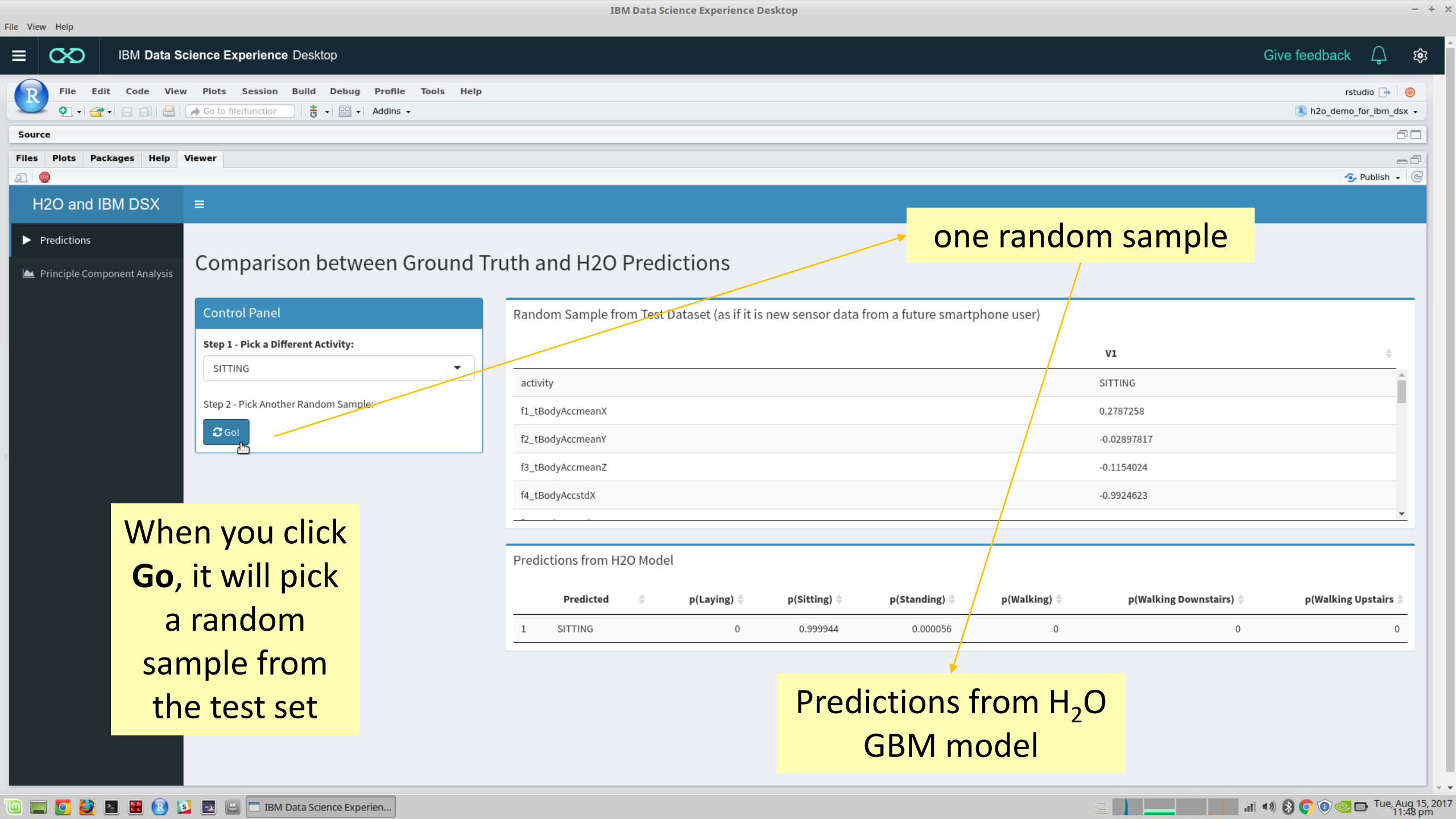
Random Sample from Test (3k) with Label



Outputs from H_2O GBM

H2O.ai


IBM Data Science Experience Desktop
File View Help
IBM Data Science Experience Desktop
Give feedback
File Edit Code View Plots Session Build Debug Profile Tools Help
rstudio
Go to file/function
Addins
h2o_demo_for_ibm_dsx
Source
Files Plots Packages Help Viewer
Publish
H2O and IBM DSX
Predictions
Principle Component Analysis
Comparison between Ground Truth and H2O Predictions
Control Panel
Step 1 - Pick a Different Activity:
WALKING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
STANDING
SITTING
LAYING
Random Sample from Test Dataset (as if it is new sensor data from a future smartphone user)
V1
activity WALKING
f1_tBodyAccmeanX 0.3477308
f2_tBodyAccmeanY -0.03215679
f3_tBodyAccmeanZ -0.0988225
f4_tBodyAccstdX -0.1561742
Predictions from H2O Model
Predicted p(Laying) p(Sitting) p(Standing) p(Walking) p(Walking Downstairs) p(Walking Upstairs
1 WALKING 0 0 0 0.999989 0.00001 0
Choose any activity from the dropdown list
IBM Data Science Experien...
Tue, Aug 15, 2017 11:48 pm


IBM Data Science Experience Desktop
File View Help
IBM Data Science Experience Desktop
Give feedback
File Edit Code View Plots Session Build Debug Profile Tools Help
rstudio
Addins
h2o_demo_for_ibm_dsx
Source
Files Plots Packages Help Viewer
Publish
H2O and IBM DSX
Predictions
Principle Component Analysis
Comparison between Ground Truth and H2O Predictions
Control Panel
Step 1 - Pick a Different Activity:
SITTING
Step 2 - Pick Another Random Sample:
Go!
Random Sample from Test Dataset (as if it is new sensor data from a future smartphone user)
V1
activity SITTING
f1_tBodyAccmeanX 0.2787258
f2_tBodyAccmeanY -0.02897817
f3_tBodyAccmeanZ -0.1154024
f4_tBodyAccstdX -0.9924623
Predictions from H2O Model
Predicted p(Laying) p(Sitting) p(Standing) p(Walking) p(Walking Downstairs) p(Walking Upstairs
1 SITTING 0 0.999944 0.000056 0 0 0
one random sample
When you click Go, it will pick a random sample from the test set
Predictions from H2O GBM model
IBM Data Science Experien...
Tue, Aug 15, 2017 11:48 pm

IBM Data Science Experience Desktop

File View Help

IBM Data Science Experience Desktop

Give feedback

File Edit Code View Plots Session Build Debug Profile Tools Help

rstudio

Go to file/function Addins

h2o_demo_for_ibm_dsx

Source

Files Plots Packages Help Viewer

Publish

Second tab will take you to the PCA visualization

H2O and IBM DSX

Predictions

Principle Component Analysis

Visualizing Principle Components from H2O Model

Control Panel

Dataset:

- (•) train
- ( ) test

Variable for X-Axis:

- ( ) PC1
- (•) PC2
- ( ) PC3
- ( ) PC4
- ( ) PC5

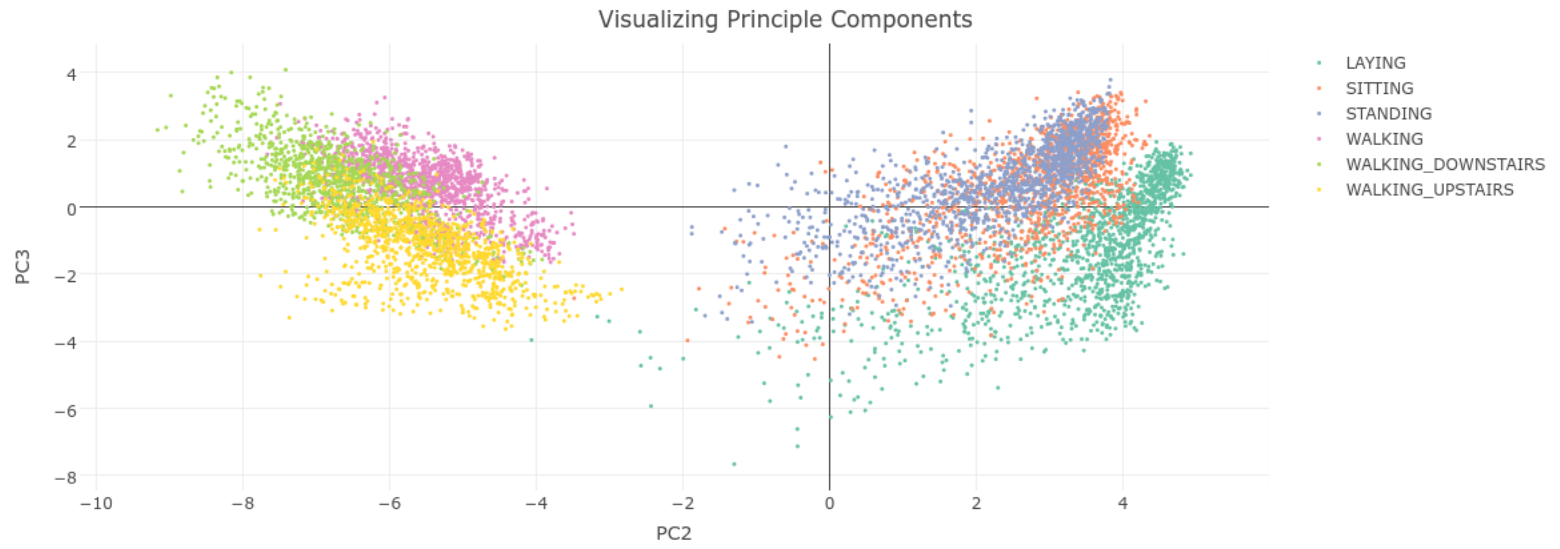
Variable for Y-Axis:

- ( ) PC1
- ( ) PC2
- (•) PC3
- ( ) PC4
- ( ) PC5

Refresh Graph:

Go!

Interactive Plot



IBM Data Science Experien...

Tue, Aug 15, 2017
11:49 pm

IBM Data Science Experience Desktop

File View Help

IBM Data Science Experience Desktop

Give feedback

File Edit Code View Plots Session Build Debug Profile Tools Help

rstudio

Go to file/function Addins

h2o_demo_for_ibm_dsx

Source

Files Plots Packages Help Viewer

Publish

It is interactive, you can zoom in/out and click on any point

H2O and IBM DSX

Predictions

Principle Component Analysis

Visualizing Principle Components from H2O Model

Control Panel

Dataset:

- train
- test

Variable for X-Axis:

- PC1
- PC2
- PC3
- PC4
- PC5

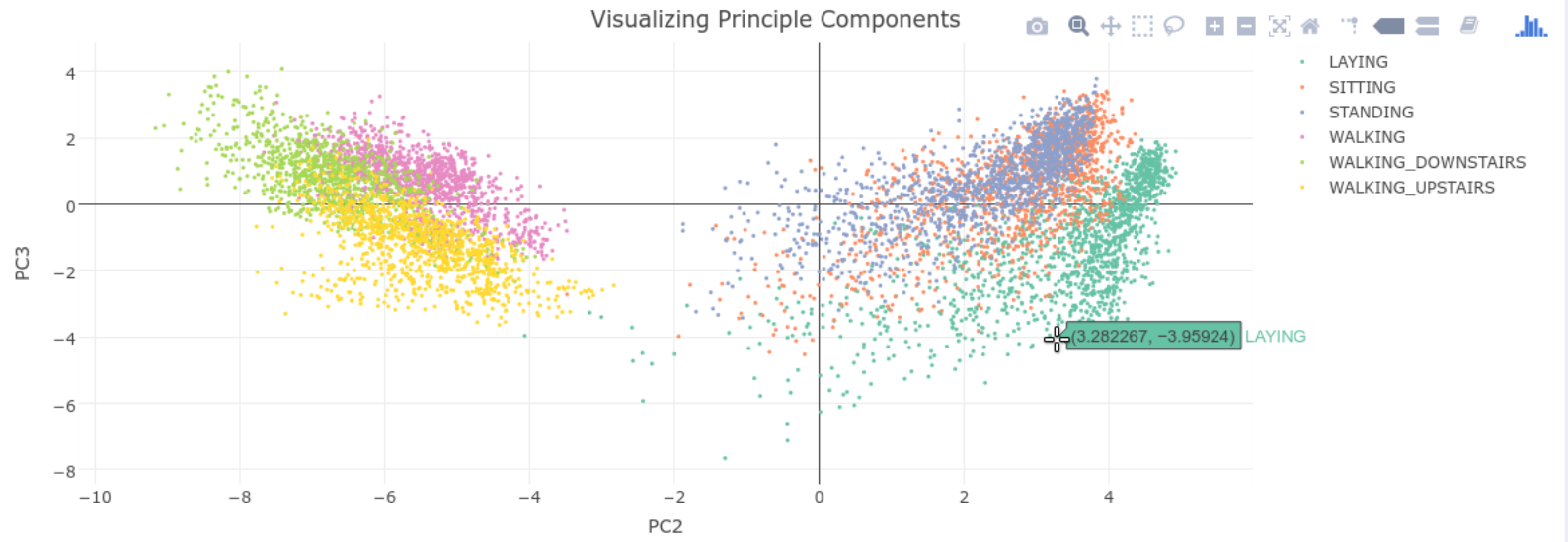
Variable for Y-Axis:

- PC1
- PC2
- PC3
- PC4
- PC5

Refresh Graph:

Go!

Interactive Plot



IBM Data Science Experien...

Tue, Aug 15, 2017
11:49 pm

IBM Data Science Experience Desktop

File View Help

IBM Data Science Experience Desktop

Give feedback

File Edit Code View Plots Session Build Debug Profile Tools Help

rstudio

Go to file/function Addins

h2o_demo_for_ibm_dsx

Source

Files Plots Packages Help Viewer

Publish

You can try the visualization with different principle components

H2O and IBM DSX

Predictions

Principle Component Analysis

Visualizing Principle Components from H2O Model

Control Panel

Dataset:

- (•) train
- ( ) test

Variable for X-Axis:

- ( ) PC1
- ( ) PC2
- (•) PC3
- ( ) PC4
- ( ) PC5

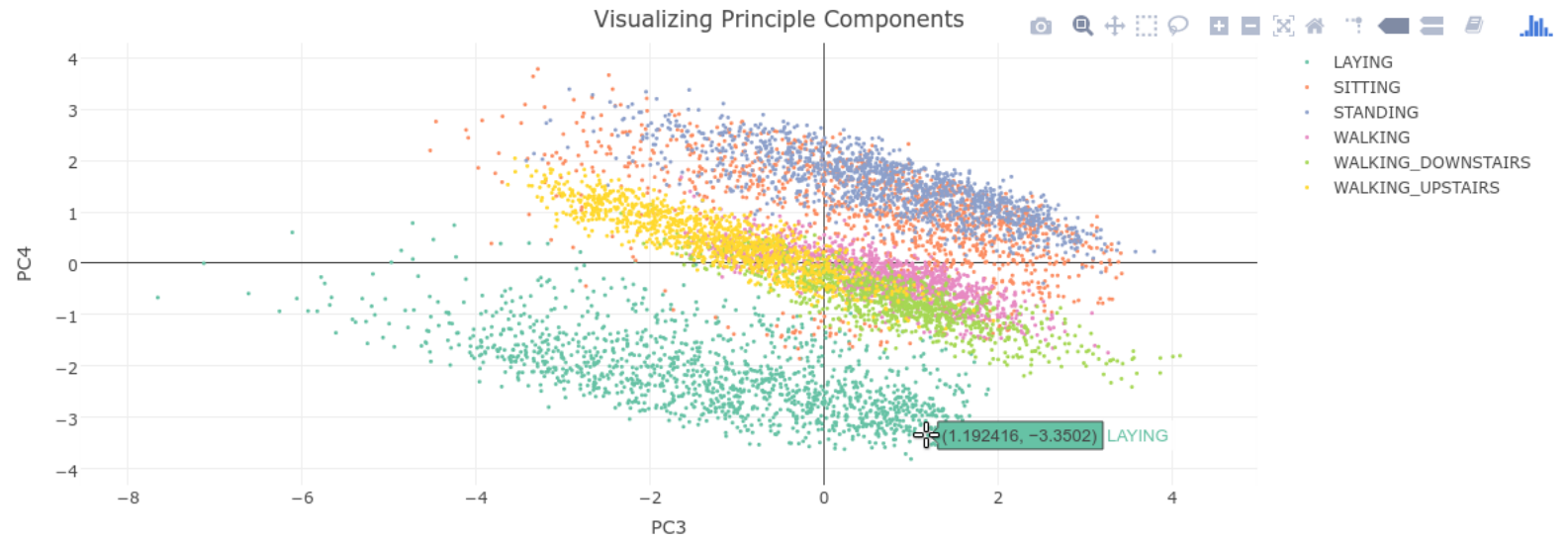
Variable for Y-Axis:

- ( ) PC1
- ( ) PC2
- ( ) PC3
- (•) PC4
- ( ) PC5

Refresh Graph:

Go!

Interactive Plot



IBM Data Science Experien...

Tue, Aug 15, 2017
11:50 pm

IBM Data Science Experience Desktop

File View Help

IBM Data Science Experience Desktop

Give feedback

File Edit Code View Plots Session Build Debug Profile Tools Help

rstudio

Go to file/function Addins

h2o_demo_for_ibm_dsx

Source

Files Plots Packages Help Viewer

Publish

You can try the visualization with different principle components

H2O and IBM DSX

Predictions

Principle Component Analysis

Visualizing Principle Components from H2O Model

Control Panel

Dataset:

- train
- test

Variable for X-Axis:

- PC1
- PC2
- PC3
- PC4
- PC5

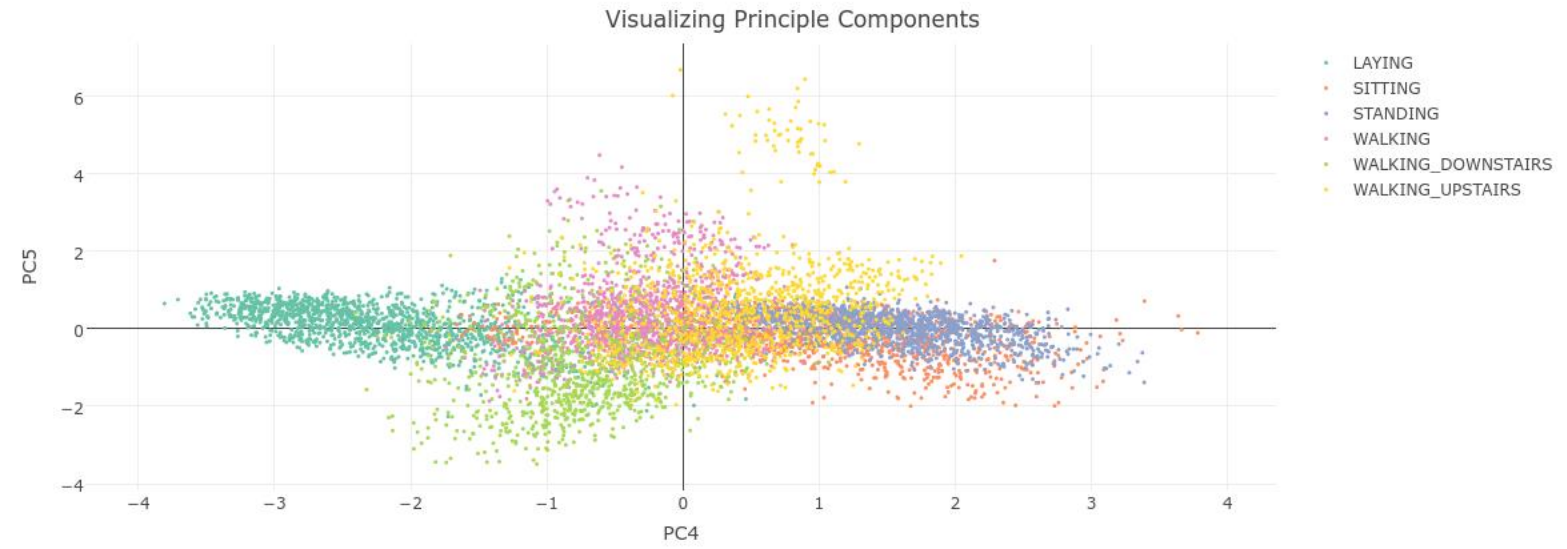
Variable for Y-Axis:

- PC1
- PC2
- PC3
- PC4
- PC5

Refresh Graph:

Go!

Interactive Plot



IBM Data Science Experien...

Tue, Aug 15, 2017
11:51 pm

IBM Data Science Experience Desktop

File View Help

IBM Data Science Experience Desktop

Give feedback

File Edit Code View Plots Session Build Debug Profile Tools Help

rstudio

Go to file/function Addins

h2o_demo_for_ibm_dsx

Source

Files Plots Packages Help Viewer

Publish

You can also try the visualization with principle components from test data

H2O and IBM DSX

Predictions

Principle Component Analysis

Visualizing Principle Components from H2O Model

Control Panel

Dataset:

- train
- test

Variable for X-Axis:

- PC1
- PC2
- PC3
- PC4
- PC5

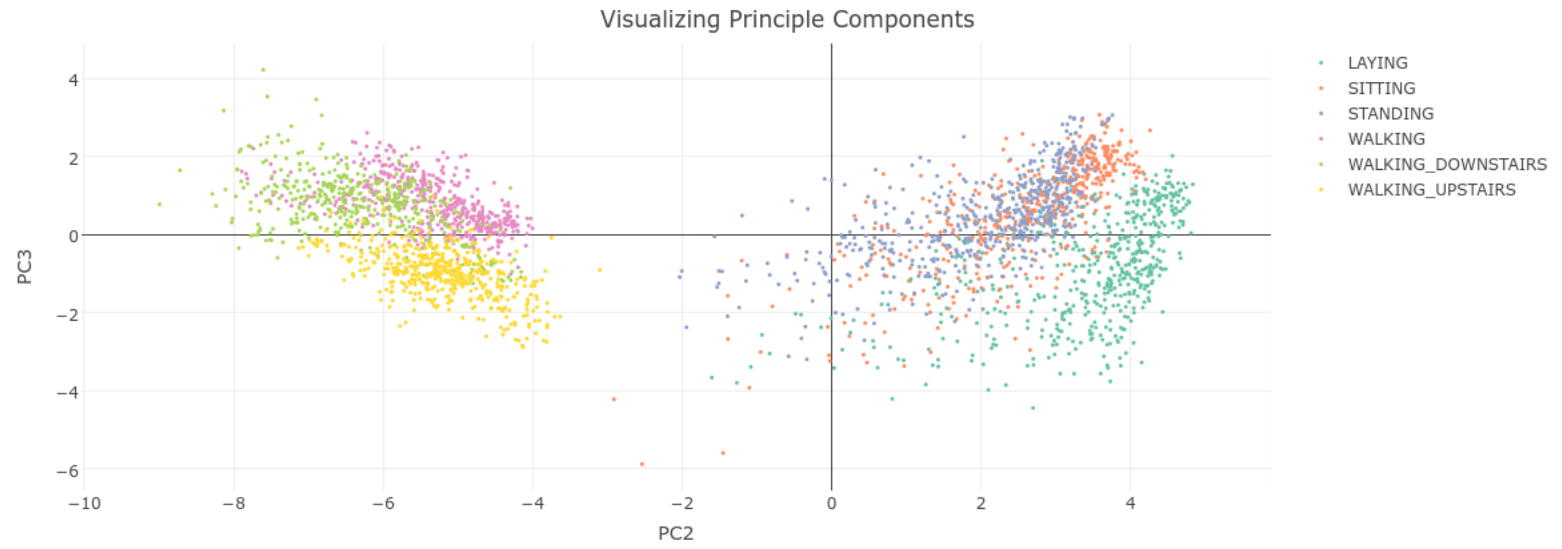
Variable for Y-Axis:

- PC1
- PC2
- PC3
- PC4
- PC5

Refresh Graph:

Go!

Interactive Plot



IBM Data Science Experien...

Tue, Aug 15, 2017
11:51 pm