Healthcare Predictive Analytics within the OR

Ayad Shammout and Denny Lee

June 15th, 2015

About Ayad Shammout

- Director of Business Intelligence, Beth Israel Deaconess Medical Center

- Helped build highly available / disaster recovery infrastructure for BIDMC

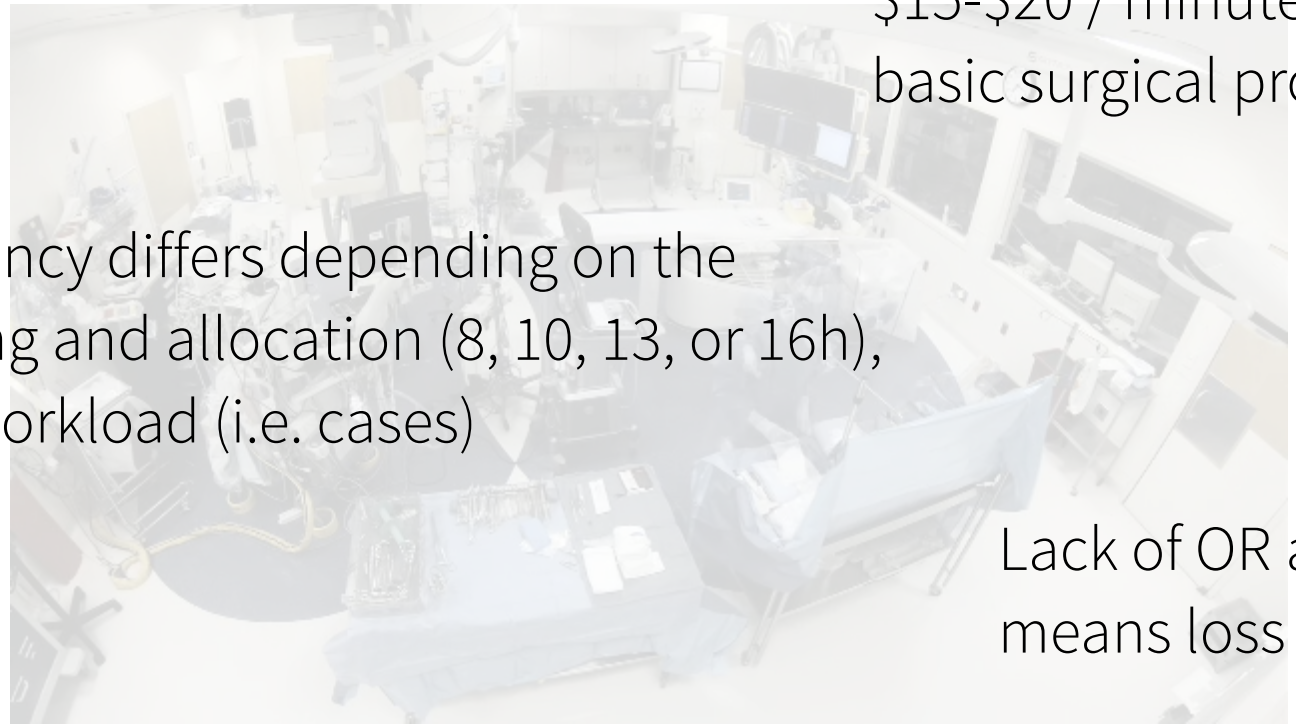
About Denny Lee

- Technology Evangelist, Databricks
- Former Sr. Director of Data Sciences Eng, Concur
- Helped bring Hadoop onto Windows and Azure

About Databricks

- Founded by Apache Spark Creators
- Largest contributor to Spark project, committed to keeping Spark 100% open source
- Databricks is an end-to-end hosted platform

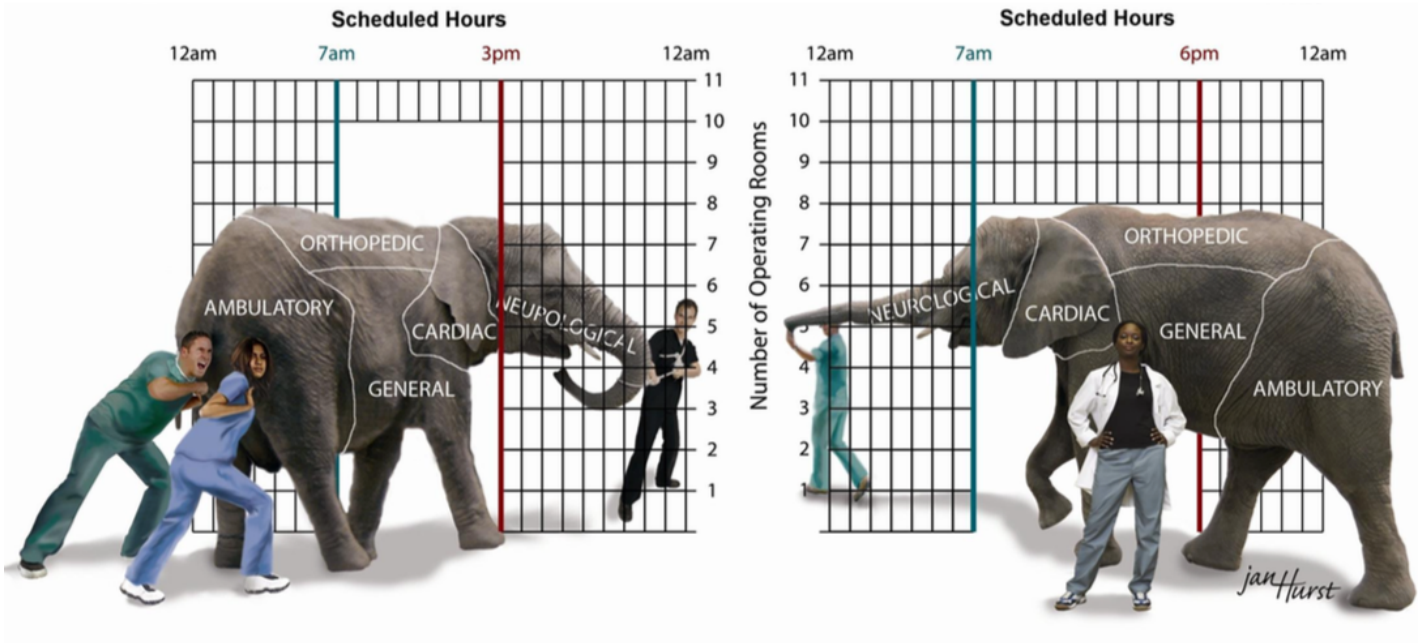
$15-$20 / minute for a basic surgical procedure



OR efficiency differs depending on the OR staffing and allocation (8, 10, 13, or 16h), not the workload (i.e. cases)

Lack of OR availability means loss of patient

Time is an OR's most valuable resource



"You are not going to get the elephant to shrink or change its size. You need to face the fact that the elephant is 8 OR tall and 11hr wide"

Steven Shafer, MD

Operating Room

Better utilization = Better profit margins

Reduce support and maintenance costs

Medical Staff

Better utilization = Better profit margins

Better medical staff efficiencies = Better outcomes

Patients

Shorter wait times and less cancellations

Better medical staff efficiencies = Better outcomes

Develop Predictive Model

- Develop a predictive model that would identify available OR time two weeks in advance.

- Allow us to confirm wait list cases two weeks in advance, instead of when the blocks normally release four days out.

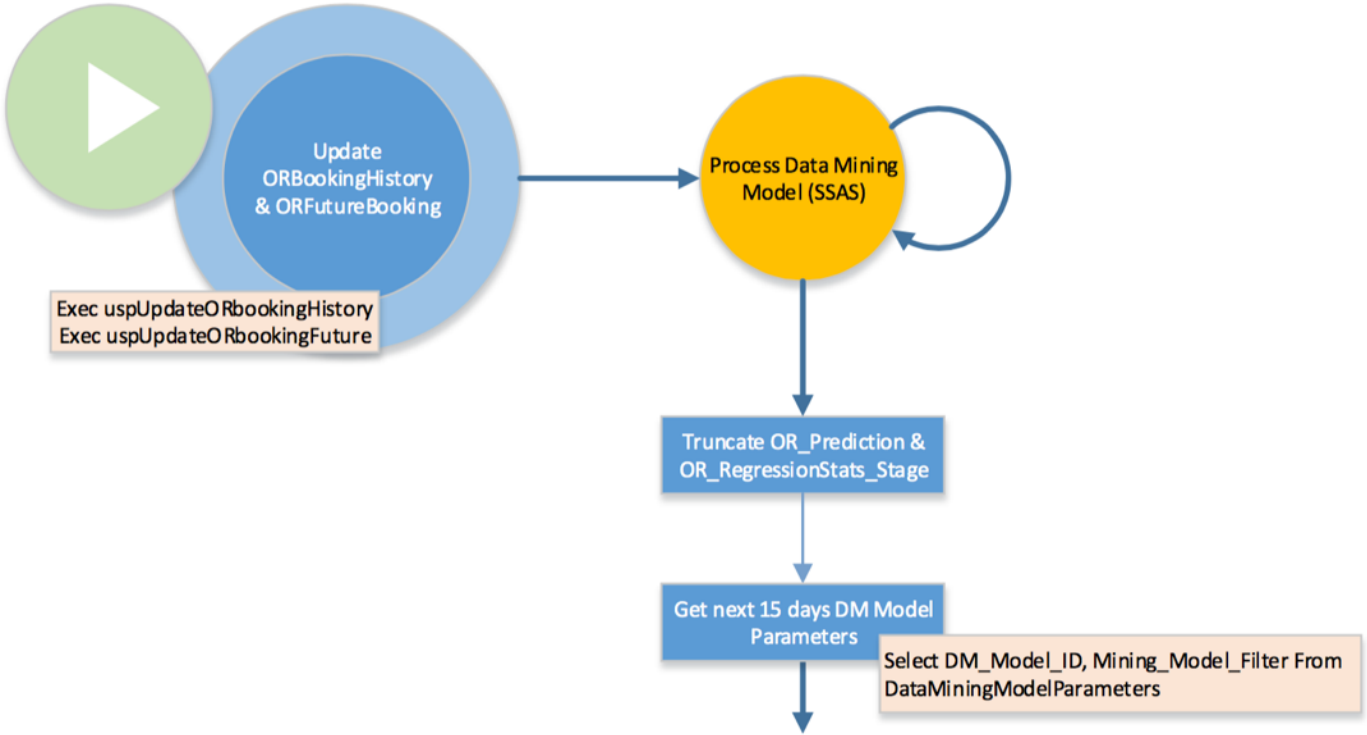
Forecast OR Schedule

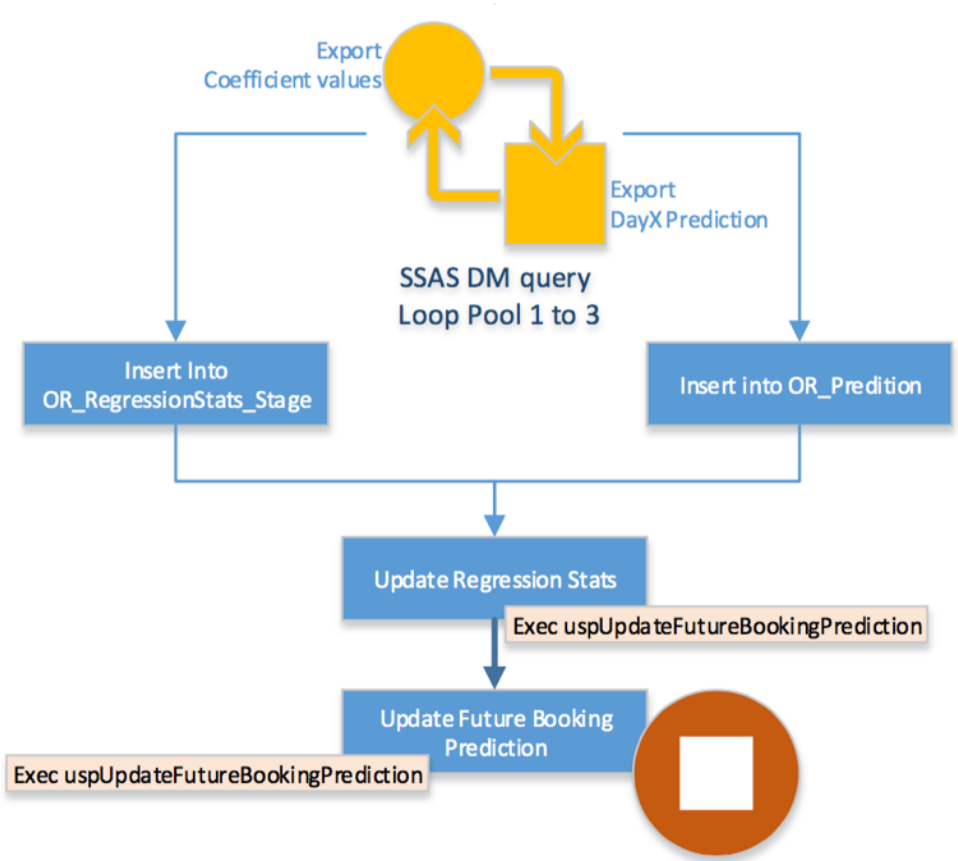
- Case load three weeks in advance
- Book more cases weeks in advance to prevent under-utilization
- Reduce staff overtime and idle time

Background

- Three surgical pools
  - GYN, urology, general surgery, colorectal, surgical oncology
  - Eyes, plastics, ENT
  - Orthopedics, podiatry

- Currently built using SQL Server Data Mining


Update
ORBookingHistory
& ORFutureBooking
Exec uspUpdateORbookingHistory
Exec uspUpdateORbookingFuture
Process Data Mining
Model (SSAS)
Truncate OR_Prediction &
OR_RegressionStats_Stage
Get next 15 days DM Model
Parameters
Select DM_Model_ID, Mining_Model_Filter From
DataMiningModelParameters


Export
Coefficient values
Export
DayX Prediction
SSAS DM query
Loop Pool 1 to 3
Insert Into
OR_RegressionStats_Stage
Insert into OR_Predition
Update Regression Stats
Exec uspUpdateFutureBookingPrediction
Update Future Booking
Prediction
Exec uspUpdateFutureBookingPrediction

demo

OR Block Scheduling

Extract History data and run linear regression with SGD with multiple variables

Build Data Frame

```
case class ORBookingTimeHistory(KeyID: Long, AltKeyID: Long, GroupID: Long, Case_Date: String, DayName_short: String, DayOfWeekNum: Long, DaysAhead: Long,
Day30: Long, Day20: Long, DayX: Long, Day5: Long, RecCreateDt: String)
```

```
defined class ORBookingTimeHistory
```

Command took 0.28s

```
val dfOR = sc.textFile("/mnt/cg.montreal/history/ORBookingTimeHistory.csv").map(_.split(",")).map(m => ORBookingTimeHistory(m(0).toLong, m(1).toLong,
m(2).toLong, m(3), m(4), m(5).toLong, m(6).toLong, m(7).toLong, m(8).toLong, m(9).toLong, m(10).toLong, m(11))).toDF()
```

```
dfOR: org.apache.spark.sql.DataFrame = [KeyID: bigint, AltKeyID: bigint, GroupID: bigint, Case_Date: string, DayName_short: string, DayOfWeekNum: bigint, DaysAhead: bigint,
Day30: bigint, Day20: bigint, DayX: bigint, Day5: bigint, RecCreateDt: string]
```

Command took 0.31s

```
dfOR.registerTempTable("ORBookingTimeHistory")
```

Command took 0.14s

Review Features and Labels (i.e. x and y variables)

```
> val df = sqlContext.sql("SELECT GroupID, DaysAhead, Day30, Day20, DayX, Day5, KeyID FROM ORBookingTimeHistory WHERE GroupID = 1 AND DaysAhead = 1")
df.count
```

```
df: org.apache.spark.sql.DataFrame = [GroupID: bigint, DaysAhead: bigint, Day30: bigint, Day20: bigint, DayX: bigint, Day5: bigint, KeyID: bigint]
res1: Long = 126
```

Command took 1.11s

```
> df.collect.take(10).foreach(println)
```

```
[1,1,27,162,867,747,2878591]
[1,1,48,207,573,519,2878606]
[1,1,264,369,864,615,2878621]
[1,1,333,432,1140,858,2878636]
[1,1,207,300,840,750,2878651]
[1,1,201,318,915,633,2878666]
[1,1,129,225,789,642,2878681]
[1,1,96,189,648,597,2878696]
[1,1,105,171,435,393,2878831]
[1,1,48,147,594,408,2878846]
```

Command took 0.44s

Setup MLLib

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
```

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
```

Command took 0.10s

Defintions

- Features: Day30 (2), Day20 (3), DayX (4),
- Label: Day5 (5)

```
val parsedData = data.map { p =>
  val label = p(5).toString.toDouble
  val features = Array(p(2), p(3), p(4)) map (_.toString.toDouble)
  LabeledPoint(label, Vectors.dense(features))
}
```

```
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[3383] at map at
```

Command took 0.32s

Feature Scaling (Regularization)

```
import org.apache.spark.mllib.feature.StandardScaler
val scaler = new StandardScaler(withMean = true, withStd = true).fit(parsedData.map(x => x.features))
val scaledData = parsedData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.dense(x.features.toArray))))
```

```
import org.apache.spark.mllib.feature.StandardScaler
scaler: org.apache.spark.mllib.feature.StandardScalerModel = org.apache.spark.mllib.feature.StandardScalerModel@4d50448
scaledData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[3386] at map at <console>:34
```

Command took 0.43s

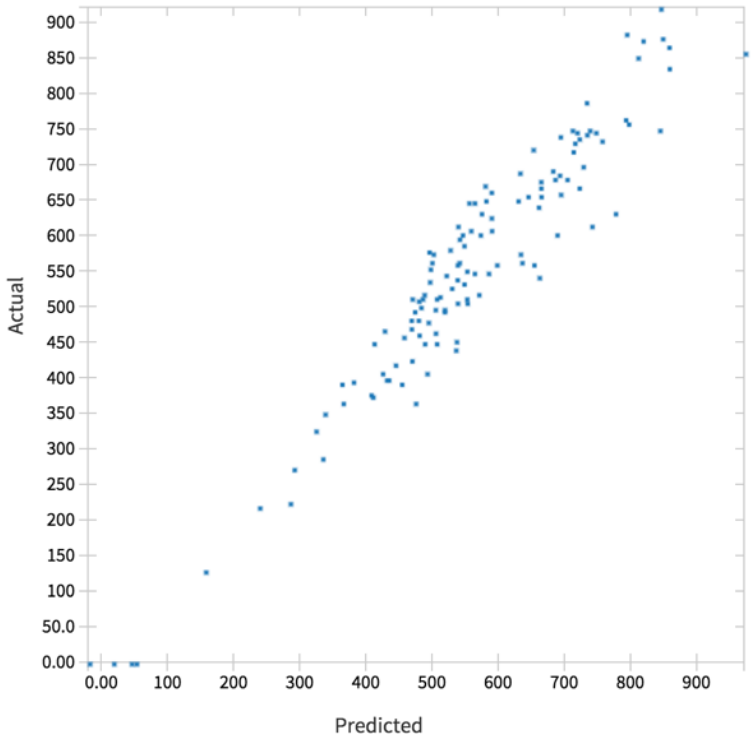
Train the Model

```
val numIterations = 3000
val alpha = 0.1
val algorithm = new LinearRegressionWithSGD()
algorithm.setIntercept(true)
algorithm.optimizer.setNumIterations(numIterations)
algorithm.optimizer.setStepSize(alpha)
val model = algorithm.run(scaledData)
```

```
numIterations: Int = 3000
alpha: Double = 0.1
algorithm: org.apache.spark.mllib.regression.LinearRegressionWithSGD = org.apache.spark.mllib.regression.LinearRegressionWithSGD@19bc16bf
model: org.apache.spark.mllib.regression.LinearRegressionModel = (weights=[-7.266071208249522,25.566012025772352,156.2382377166797], intercept=554.6559567707233)
```

Command took 56.74s

```
> %sql select `_1` as Actual, `_3` as Predicted from LRResults;
```



Plot Options...

Command took 0.27s

Apply Previous Linear Regression with SGD model

For GroupID = 1, DaysAhead = 1 (KeyID = 151), predicted booking is 483

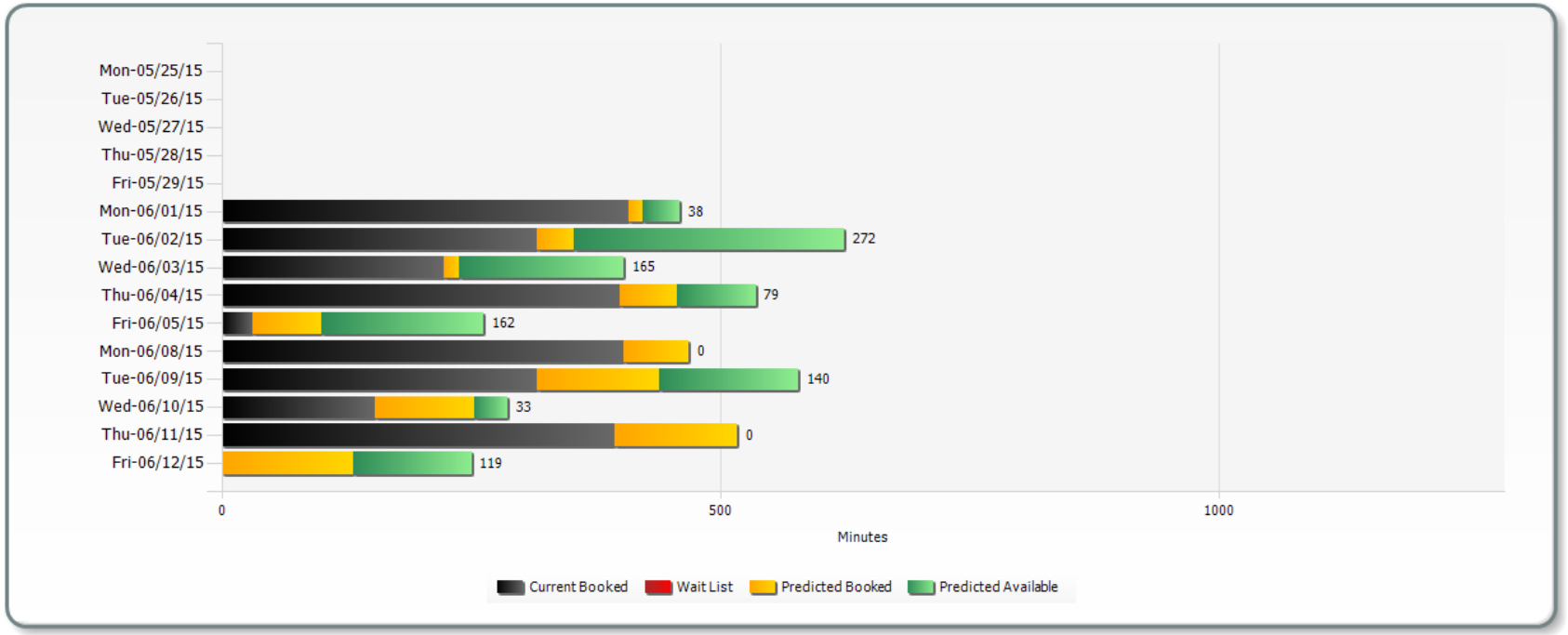
```
> val valuesAndPredsFuture = scaledFutureData.map { point =>
    val prediction = model.predict(point.features)
    (point.label, point.features, prediction)
  }
valuesAndPredsFuture.take(10).foreach({case (v, f, p) =>
     println(s"Features: ${f}, Predicted: ${p}, Actual: ${v}")})
```

```
Features: [-0.24464267654515323,0.04851883110991891,-0.5092994337292287], Predicted: 478.1019349027521, Actual: 546.0
valuesAndPredsFuture: org.apache.spark.rdd.RDD[(Double, org.apache.spark.mllib.linalg.Vector, Double)] = MapPartitionsRDD[9524]
```

```
Command took 0.46s
```

MLLib LinearRegressionSGD predcition: 478.1019349027521

OR Schedule Report (example)



Why the model is working

- Can coordinate waitlist scheduling logistics with physicians and patients within two weeks of the surgery
- Plan staff scheduling and resources so there are less last-minute staffing issues for nursing and anesthesia
- Utilization metrics are showing us where we can maximize our elective surgical schedule and level demand

Thank you.

For more information, please contact denny@databricks.com