



CIS5560 Term Project Tutorial



Authors: Krithy Nanaiah Atrangada; Neha Shashidhara Guli; Anupam Sahay

Instructor: [Jongwook Woo](#)

Date: 05/16/2018

Lab Tutorial

katrang (katrang@calstatela.edu)

sneha (sneha@calstatela.edu)

asahay (asahay@calstatela.edu)

05/16/2018

San Francisco Bay Area Bike Share Analysis On Data Bricks in Spark Machine Learning

Objectives

List what your objectives are. In this hands-on lab, you will learn how to:

- Get data manually
- Create Spark cluster
- Train NLP system
- SQL commands to perform the analysis.

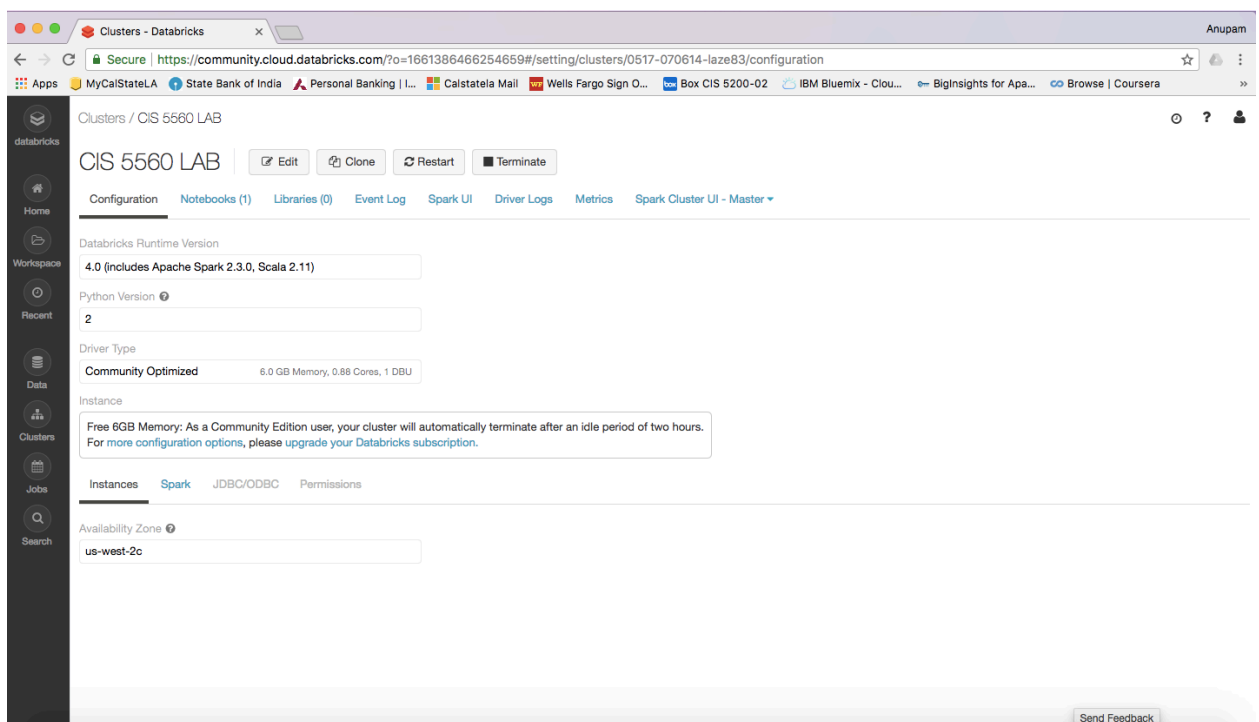
- Writing PySpark codes to develop a predictive model.
- Predicting total number of trips on a certain day using Decision Tree Regression, Random Forest Regression, and Gradient Boosting Regression.
- Visualization

Platform Spec

- Data Bricks PySpark
- Databricks Runtime Version: 4.0(Incl. Apache Spark 2.3.0, Scala 2.11)
- Execution: Single Node
- Memory: 6GB Capacity

Step 1: Creating a Cluster in Data Bricks

1. This step is to create a cluster for the execution of the codes.



Properties: -

- Python Version: 2
- Driver Type: Community Optimized
- Availability Zone: us-west-2c

Step 2: Loading the Data Set in the Data Bricks

Table: trip_csv

id	duration	start_date	start_station_name	start_station_id	end_date	end_station_name	end_station_id	bike_id	subscription_type	zip_code
4576	63	8/29/2013 14:13	South Van Ness at Market	66	8/29/2013 14:14	South Van Ness at Market	66	520	Subscriber	94127
4607	70	8/29/2013 14:42	San Jose City Hall	10	8/29/2013 14:43	San Jose City Hall	10	661	Subscriber	95138
4130	71	8/29/2013 10:16	Mountain View City Hall	27	8/29/2013 10:17	Mountain View City Hall	27	48	Subscriber	97214
4251	77	8/29/2013 11:29	San Jose City Hall	10	8/29/2013 11:30	San Jose City Hall	10	26	Subscriber	95060
4299	83	8/29/2013 12:02	South Van Ness at Market	66	8/29/2013 12:04	Market at 10th	67	319	Subscriber	94103
4927	103	8/29/2013 18:54	Golden Gate at Polk	59	8/29/2013 18:56	Golden Gate at Polk	59	527	Subscriber	94109
4500	109	8/29/2013 13:25	Santa Clara at Almaden	4	8/29/2013 13:27	Adobe on Almaden	5	679	Subscriber	95112
4563	111	8/29/2013 14:02	San Salvador at 1st	8	8/29/2013 14:04	San Salvador at 1st	8	687	Subscriber	95112
4760	113	8/29/2013 17:01	South Van Ness at Market	66	8/29/2013 17:03	South Van Ness at Market	66	553	Subscriber	94103
4258	114	8/29/2013 11:33	San Jose City Hall	10	8/29/2013 11:35	MLK Library	11	107	Subscriber	95060
4549	125	8/29/2013 13:52	Spear at Folsom	49	8/29/2013 13:55	Embarcadero at Bryant	54	368	Subscriber	94109
4498	126	8/29/2013 13:23	San Pedro Square	6	8/29/2013 13:25	Santa Clara at Almaden	4	26	Subscriber	95112
4965	129	8/29/2013 19:32	Mountain View Caltrain Station	28	8/29/2013 19:35	Mountain View Caltrain Station	28	140	Subscriber	94041
4557	130	8/29/2013 13:57	2nd at South Park	64	8/29/2013 13:59	2nd at South Park	64	371	Subscriber	94122

Send Feedback

Table: station_csv

city	string	null
installation_date	string	null

Sample Data:

id	name	lat	long	dock_count	city	installation_date
2	San Jose Diridon Caltrain Station	37.329732	-121.90178200000001	27	San Jose	8/6/2013
3	San Jose Civic Center	37.330698	-121.888979	15	San Jose	8/5/2013
4	Santa Clara at Almaden	37.333988	-121.894902	11	San Jose	8/6/2013
5	Adobe on Almaden	37.331415	-121.8932	19	San Jose	8/5/2013
6	San Pedro Square	37.336721000000004	-121.894074	15	San Jose	8/7/2013
7	Paseo de San Antonio	37.333798	-121.88694299999999	15	San Jose	8/7/2013
8	San Salvador at 1st	37.330165	-121.88583100000001	15	San Jose	8/5/2013
9	Japantown	37.348742	-121.89471499999999	15	San Jose	8/5/2013
10	San Jose City Hall	37.337391	-121.886995	15	San Jose	8/6/2013
11	MLK Library	37.335885	-121.88566000000002	19	San Jose	8/6/2013
12	SJSU 4th at San Carlos	37.332808	-121.88389999999999	19	San Jose	8/7/2013
13	St James Park	37.339301	-121.88993700000002	15	San Jose	8/6/2013
14	Arena Green / SAP Center	37.332692	-121.900884	19	San Jose	8/5/2013
16	SJSU - San Salvador at 9th	37.333954999999996	-121.877349	15	San Jose	8/7/2013
21	Franklin at Maple	37.481758	-122.226904	15	Redwood City	8/12/2013
22	Redwood City Caltrain Station	37.486078000000006	-122.23288999999999	25	Redwood City	8/15/2013
23	San Mateo County Center	37.487615999999996	-122.229951	15	Redwood City	8/15/2013
24	Redwood City Public Library	37.484219	-122.227424	15	Redwood City	8/12/2013
25	Stanford in Redwood City	37.48537	-122.20328799999999	15	Redwood City	8/12/2013
26	Redwood City Medical Center	37.487682	-122.223492	15	Redwood City	8/12/2013

Send Feedback

Table: weather_csv

min_visibility_miles	string	null

Sample Data:

date	max_temperature_f	mean_temperature_f	min_temperature_f	max_dew_point_f	mean_dew_point_f	min_dew_point_f	max_humidity	mean_humidity	min_humidity	max...
8/29/2013	74.0	68.0	61.0	61.0	58.0	56.0	93.0	75.0	57.0	30.
8/30/2013	78.0	69.0	60.0	61.0	58.0	56.0	90.0	70.0	50.0	30.
8/31/2013	71.0	64.0	57.0	57.0	56.0	54.0	93.0	75.0	57.0	30.
9/1/2013	74.0	66.0	58.0	60.0	56.0	53.0	87.0	68.0	49.0	29.
9/2/2013	75.0	69.0	62.0	61.0	60.0	58.0	93.0	77.0	61.0	29.
9/3/2013	73.0	67.0	60.0	59.0	56.0	51.0	84.0	65.0	46.0	30.
9/4/2013	74.0	68.0	61.0	59.0	57.0	56.0	90.0	72.0	53.0	30.
9/5/2013	72.0	66.0	60.0	57.0	56.0	54.0	90.0	74.0	57.0	30.
9/6/2013	85.0	71.0	56.0	57.0	51.0	45.0	86.0	58.0	29.0	30.
9/7/2013	88.0	73.0	58.0	64.0	54.0	46.0	86.0	59.0	31.0	29.
9/8/2013	74.0	65.0	56.0	58.0	54.0	52.0	86.0	70.0	53.0	29.
9/9/2013	76.0	66.0	55.0	58.0	55.0	52.0	90.0	70.0	50.0	29.
9/10/2013	74.0	66.0	57.0	59.0	56.0	54.0	93.0	73.0	53.0	29.
9/11/2013	74.0	68.0	62.0	57.0	55.0	54.0	78.0	68.0	57.0	30.
9/12/2013	71.0	65.0	59.0	58.0	57.0	55.0	84.0	73.0	61.0	29.
9/13/2013	66.0	62.0	57.0	55.0	54.0	54.0	93.0	80.0	67.0	29.
9/14/2013	66.0	62.0	57.0	55.0	54.0	53.0	87.0	77.0	67.0	29.
9/15/2013	73.0	66.0	58.0	59.0	55.0	52.0	90.0	72.0	53.0	29.
9/16/2013	71.0	65.0	59.0	58.0	55.0	53.0	90.0	74.0	57.0	29.
9/17/2013	68.0	63.0	57.0	55.0	53.0	50.0	86.0	72.0	58.0	29.

Send Feedback

Create Table - Databricks

Create New Table

Data source

Upload File S3 DBFS Spark Data Sources

Upload to DBFS

/FileStore/tables/ (optional) Select

File

status.csv

2 GB

Cancel upload

Send Feedback

STEPS:

- First, we click on Data on left hand side.
- Then we upload the file from the local disk on to the data bricks
- Once the file is uploaded click on create table with UI.

- Select the cluster which we created. Then click on Preview Table
- In Specify Table attributes check first row is header. Then click on create table.

Table: status_csv

docks_available	time
string	null
string	null

Sample Data:

station_id	bikes_available	docks_available	time
2	2	25	2013/08/29 12:06:01
2	2	25	2013/08/29 12:07:01
2	2	25	2013/08/29 12:08:01
2	2	25	2013/08/29 12:09:01
2	2	25	2013/08/29 12:10:01
2	2	25	2013/08/29 12:11:01
2	2	25	2013/08/29 12:12:01
2	2	25	2013/08/29 12:13:01
2	2	25	2013/08/29 12:15:01
2	2	25	2013/08/29 12:16:02
2	2	25	2013/08/29 12:18:01
2	2	25	2013/08/29 12:19:01
2	2	25	2013/08/29 12:20:01
2	2	25	2013/08/29 12:21:01
2	2	25	2013/08/29 12:22:01
2	2	25	2013/08/29 12:23:01
2	2	25	2013/08/29 12:25:01
2	2	25	2013/08/29 12:26:01
2	2	25	2013/08/29 12:27:04
2	2	25	2013/08/29 12:29:01

Step 3: Train Natural Language Processing

This step explains the codes which are used in the Data Bricks for execution.

```
%fs ls /FileStore/tables/status.csv
```

```
%fs ls /FileStore/tables/weather.csv
```

```
%fs ls /FileStore/tables/trip.csv
```

```
%fs ls /FileStore/tables/staion.csv
```

- In this step we load the data in the data bricks file system.
- We then display the file. In the data bricks to check if all the files are loaded from the source.

The screenshot shows a Databricks notebook interface. The top bar indicates the notebook is attached to 'CIS 5560 LAB'. The code cell contains the following Python code:

```
1 status_data = spark.read.csv("dbfs:/FileStore/tables/status.csv", inferSchema=True, header=True)
2 display(status_data)
```

The output shows a Spark job completion message and a preview of the data from the 'status_data' DataFrame. The DataFrame has columns: station_id, bikes_available, docks_available, and time. The preview shows 10 rows of data.

station_id	bikes_available	docks_available	time
2	2	25	2013/08/29 12:06:01
2	2	25	2013/08/29 12:07:01
2	2	25	2013/08/29 12:08:01
2	2	25	2013/08/29 12:09:01
2	2	25	2013/08/29 12:10:01
2	2	25	2013/08/29 12:11:01
2	2	25	2013/08/29 12:12:01
2	2	25	2013/08/29 12:13:01
2	2	25	2013/08/29 12:14:01
2	2	25	2013/08/29 12:15:01

Below the table, it says 'Showing the first 1000 rows.' and 'Command took 5.63 minutes -- by asahay@calstatela.edu at 5/17/2018, 12:11:51 AM on CIS 5560 LAB'. The bottom cell shows the start of another code block for reading 'weather_data'.

- When all the data is loaded on the data bricks, we then check if any of the tables contains any null values.
- We check for the null values for all the dataset.

from pyspark.sql.functions import isnan, count, when

status_data.select([count(when(isnan(c), c)).alias(c) for c in status_data.columns]).show()

from pyspark.sql.functions import isnan, count, when

trip_data.select([count(when(isnan(c), c)).alias(c) for c in trip_data.columns]).show()

- We then check for the max,min,mean,median for the trip_data.

from pyspark.sql.functions import mean, min, max, stddev

trip_data.select([mean('duration'), min('duration'), max('duration'), stddev('duration')]).show()

CIS PROJECT 5560 (FINAL VERSION) (Python)

Attached: CIS 5560 LAB

Cmd 12

```
1 from pyspark.sql.functions import isnan, count, when
2 status_data.select((count(when(isnan(c), c)).alias(c) for c in status_data.columns)).show()
```

↳ (1) Spark Jobs

station_id	bikes_available	docks_available	time
0	0	0	0

Command took 4.73 minutes -- by asahay@calstatela.edu at 5/17/2018, 12:17:41 AM on CIS 5560 LAB

Cmd 13

```
1 from pyspark.sql.functions import isnan, count, when
2 trip_data.select((count(when(isnan(c), c)).alias(c) for c in trip_data.columns)).show()
```

↳ (1) Spark Jobs

id	duration	start_date	start_station_name	start_station_id	end_date	end_station_name	end_station_id	bike_id	subscription_type	zip_code
0	0	0	0	0	0	0	0	0	0	0

Command took 11.57 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:41 AM on CIS 5560 LAB

Cmd 14

```
1 from pyspark.sql.functions import mean, min, max, stddev
2 trip_data.select([mean('duration'), min('duration'), max('duration'), stddev('duration')]).show()
```

↳ (1) Spark Jobs

Send Feedback

- We then create a new duration in which we convert all the seconds to minutes and we drop the old columns.
- We then again check for the max, min, mean, median for the trip_data.

CIS PROJECT 5560 (FINAL VERSION) (Python)

Attached: CIS 5560 LAB

Cmd 15

```
1 #dropping old column..
2 trip_data = trip_data.drop(trip_data.duration)
```

↳ trip_data: pyspark.sql.dataframe.DataFrame = [id: integer, start_date: string ... 9 more fields]

Command took 9.15 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:42 AM on CIS 5560 LAB

Cmd 17

```
1 trip_data.show()
```

↳ (1) Spark Jobs

id	start_date	start_station_name	start_station_id	end_date	end_station_name	end_station_id	bike_id	subscription_type	zip_code	duration_new
4576	8/29/2013 14:13	South Van Ness at...	66	8/29/2013 14:14	South Van Ness at...	66	520	Subscriber	94127	1.85
4607	8/29/2013 14:42	San Jose City Hall	10	8/29/2013 14:43	San Jose City Hall	10	661	Subscriber	95138	1.1666666666666667
4138	8/29/2013 10:16	Mountain View Cit...	27	8/29/2013 10:17	Mountain View Cit...	27	48	Subscriber	97214	1.1833333333333333
4251	8/29/2013 11:29	San Jose City Hall	10	8/29/2013 11:30	San Jose City Hall	10	26	Subscriber	95060	1.2833333333333333
4298	8/29/2013 12:02	South Van Ness at...	66	8/29/2013 12:04	Market at 10th	67	319	Subscriber	94103	1.3833333333333333
4927	8/29/2013 18:54	Golden Gate at Polk	59	8/29/2013 18:56	Golden Gate at Polk	59	527	Subscriber	94109	1.7166666666666666
4508	8/29/2013 13:25	San Jose City Hall	10	8/29/2013 13:27	Adobe on Almaden	5	679	Subscriber	95112	1.8166666666666667
4563	8/29/2013 14:02	San Salvador at 1st	8	8/29/2013 14:04	San Salvador at 1st	8	687	Subscriber	95112	1.85
4769	8/29/2013 17:01	South Van Ness at...	66	8/29/2013 17:03	South Van Ness at...	66	553	Subscriber	94103	1.8833333333333333
4258	8/29/2013 11:33	San Jose City Hall	10	8/29/2013 11:35	MLK Library	11	107	Subscriber	95060	1.9
4549	8/29/2013 13:52	Spear at Folsom	49	8/29/2013 13:55	Embarcadero at Br...	54	368	Subscriber	94109	2.0833333333333335
4498	8/29/2013 13:23	San Pedro Square	6	8/29/2013 13:25	San Jose City Hall	10	26	Subscriber	95112	2.1
4965	8/29/2013 19:32	Mountain View Cal...	28	8/29/2013 19:35	Mountain View Cal...	28	140	Subscriber	94041	2.15
4557	8/29/2013 13:57	2nd at South Park	64	8/29/2013 13:59	2nd at South Park	64	371	Subscriber	94122	2.1666666666666665
4386	8/29/2013 12:31	Clay at Battery	41	8/29/2013 12:33	Beale at Market	56	503	Subscriber	94109	2.2333333333333334
4749	8/29/2013 16:57	Post at Kearney	47	8/29/2013 16:59	Post at Kearney	47	408	Subscriber	94117	2.3
4242	8/29/2013 11:25	San Jose City Hall	10	8/29/2013 11:27	San Jose City Hall	10	26	Subscriber	95060	2.35
4329	8/29/2013 12:11	Market at 10th	67	8/29/2013 12:14	Market at 10th	67	319	Subscriber	94103	2.3833333333333333
4607	8/29/2013 14:13	South Van Ness at...	66	8/29/2013 14:14	South Van Ness at...	66	520	Subscriber	94127	1.85

Send Feedback

- Then we convert it into data time so that it can be manipulated easily.

- We then look for the distinct trip and we sort them in ascending order

The screenshot shows a Databricks notebook titled "CIS PROJECT 5560 (FINAL VERSION)". The code cell contains the following Python code:

```
1 train = train.sort('date', ascending = True)
```

The output shows the command took 0.14 seconds. Below it, a second code cell shows the command:

```
1 train.show()
```

The output displays a table with two columns: 'date' and 'trips'. The data is sorted by date in ascending order.

date	trips
2013-08-29	742
2013-08-30	699
2013-08-31	628
2013-09-01	683
2013-09-02	652
2013-09-03	592
2013-09-04	599
2013-09-05	671
2013-09-06	810
2013-09-07	788
2013-09-08	692
2013-09-09	765
2013-09-10	892
2013-09-11	876
2013-09-12	933
2013-09-13	969
2013-09-14	690
2013-09-15	613
2013-09-16	912
2013-09-17	1866

- We do similar transformation for weather data and we look out for distinct zip codes and weather events which will affect our prediction.

weather_data.select('zip_code').distinct().show()

weather_data.select('events').distinct().show()

The screenshot shows a Databricks notebook titled "CIS PROJECT 5560 (FINAL VERSION)". The code cell contains the following Python code:

```
1 weather_data = weather_data.filter(weather_data['zip_code'] == 94107)
```

The output shows the command took 1.52 seconds. Below it, a second code cell shows the command:

```
1 weather_data.select('events').distinct().show()
```

The output displays a table with one column: 'events'. The data shows distinct weather events for the specified zip code.

events
null
Fog
Rain
Fog-Rain
Rain

- We import US Federal Holiday Calendar in our data and we treat the holidays/weekends as 0 and the working days/business days as 1.
- We make True=1 and False=0. This is required to find out the total number of holidays during the time span.
- We again convert all these parameters in the format of “date”, “month”, and “year”.

```

1 #Find all of the holidays during our time span
2 from pandas.tseries.holiday import USFederalHolidayCalendar
3 from pandas.tseries.offsets import CustomBusinessDay
4 train = train1.toPandas()
5 calendar = USFederalHolidayCalendar()
6 holidays = calendar.holidays(start=train.date.min(), end=train.date.max())
7
8 us_bd = CustomBusinessDay(calendar=USFederalHolidayCalendar())
9 business_days = pd.DatetimeIndex(start=train.date.min(), end=train.date.max(), freq=us_bd)
10
11 business_days = pd.to_datetime(business_days, format='%Y/%m/%d').date
12 holidays = pd.to_datetime(holidays, format='%Y/%m/%d').date
13
14 #A 'business_day' or 'holiday' is a date within either of the respected lists.
15 train['business_day'] = train.date.isin(business_days)
16 train['holiday'] = train.date.isin(holidays)
17
18 #Convert True to 1 and False to 0
19 train.business_day = train.business_day.map(lambda x: 1 if x == True else 0)
20 train.holiday = train.holiday.map(lambda x: 1 if x == True else 0)
21 train['year'] = pd.to_datetime(train['date']).dt.year
22 train['month'] = pd.to_datetime(train['date']).dt.month
23 train['weekday'] = pd.to_datetime(train['date']).dt.weekday
24 labels = train.trips
25 train = train.drop(['date'], 1)
26 train = sqlContext.createDataFrame(train)

```

(1) Spark Jobs
 train: pyspark.sql.dataframe.DataFrame = [trips: long, max_temperature_f: double ... 31 more fields]
 Command took 0.69 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:54 AM on CIS 5560 LAB

- We split the data as train and test in the ratio of 0.8, 0.2 respectively.

train,test = final_data.randomSplit([0.8,0.2])

```
Cmd 56
1 #split data
2 train,test = final_data.randomSplit([0.8,0.2])

train: pyspark.sql.dataframe.DataFrame = [features: udt, target: long]
test: pyspark.sql.dataframe.DataFrame = [features: udt, target: long]
Command took 0.84 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:56 AM on CIS 5560 LAB

Cmd 57
1 from pyspark.ml.regression import (DecisionTreeRegressor)
2 dTree = DecisionTreeRegressor(labelCol='target', featuresCol='features')

Command took 0.88 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:56 AM on CIS 5560 LAB

Cmd 58
1 dtc_model = dTree.fit(train)
2 dtc_pred = dtc_model.transform(test)

@ Spark Jobs
dtc_pred: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
Command took 2.75 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:57 AM on CIS 5560 LAB
```

Step 4: Model 1

- After the preparation of the data we use our first algorithm to build our model.
- We use Decision Tree Regression Algorithm.
- We find out the accuracy on the test data and the root mean square error.

```
Cmd 59
1 from pyspark.ml.evaluation import RegressionEvaluator
2 # Select (prediction, true label) and compute test error
3 evaluator = RegressionEvaluator(
4     labelCol="target", predictionCol="prediction", metricName="r2")
5 accuracy = evaluator.evaluate(dtc_pred)
6 print("Accuracy on test data = %g" % (accuracy * 100))
7
8 evaluator1 = RegressionEvaluator(
9     labelCol="target", predictionCol="prediction", metricName="rmse")
10 rmse = evaluator1.evaluate(dtc_pred)
11 print("RMSE = %g" % (rmse))

@ Spark Jobs
Accuracy on test data = 88.897
RMSE = 176.905
Command took 0.89 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:57 AM on CIS 5560 LAB

Cmd 60
1 from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
2 paramGrid = ParamGridBuilder().addGrid(dTree.maxDepth, [2,3,4,5,6]).build()
3 crossval = CrossValidator(estimator=dTree,
4     estimatorParamMaps=paramGrid,
5     evaluator=evaluator,
6     numFolds=3) # use 3+ folds in practice
7
8 crossval_rmse = CrossValidator(estimator=dTree,
9     estimatorParamMaps=paramGrid,
10    evaluator=evaluator1,
11    numFolds=3)
12 # Run cross-validation, and choose the best set of parameters.
13 cvModel = crossval.fit(final_data)
```

- In this we can see that the accuracy on the test data is 80.097 and the RMSE is 176.905.
- After using the cross- validation where K =3 the accuracy is improved and the RMSE is reduced.

The screenshot shows a Databricks notebook titled "CIS PROJECT 5560 (FINAL VERSION)" in a Python environment. The notebook interface includes a sidebar with navigation options like Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area displays two code cells. The first cell, labeled "Cmd 61", contains Python code for cross-validation using a decision tree estimator. The output of this cell shows the model accuracy and RMSE. The second cell, labeled "Cmd 62", contains code to import a Random Forest Regressor.

```

7
8 crossval_rmse = CrossValidator(estimator=dtree,
9                               estimatorParamMaps=paramGrid,
10                              evaluator=evaluator1,
11                              numFolds=3)
12 # Run cross-validation, and choose the best set of parameters.
13 cvModel = crossval.fit(final_data)
14 cvModel_rmse = crossval_rmse.fit(final_data)

Command took 58.81 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:58 AM on CIS 5560 LAB

Cmd 61
1 cvPredictions = cvModel.transform(final_data)
2 accuracy = evaluator.evaluate(cvPredictions)
3 print("Model Accuracy with Cross Validation: ", accuracy*100)
4 cvPredictions1 = cvModel_rmse.transform(final_data)
5 rmse = evaluator1.evaluate(cvPredictions1)
6 print("RMSE: ", rmse)
7

(2) Spark Jobs
▶ cvPredictions: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
▶ cvPredictions1: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
('Model Accuracy with Cross Validation: ', 90.29542430757549)
('RMSE: ', 147.73967344750508)

Command took 1.14 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:58 AM on CIS 5560 LAB

Cmd 62
1 from pyspark.ml.regression import RandomForestRegressor
2 rfr = RandomForestRegressor(labelCol='target', featuresCol='features')

```

```

('Model Accuracy with Cross Validation: ', 90.29542430757549)
('RMSE: ', 147.73967344750508)

```

Step 5: Model 2

- We use Random Forest Regression Algorithm.
- We find out the accuracy on the test data and the root mean square error.
- The accuracy on the test data is 84.45 and the RMSE is 156.331
- After using the cross- validation where K =3 the accuracy is improved and the RMSE is reduced significantly.

The screenshot shows a Databricks notebook titled "CIS PROJECT 5560 (FINAL VERSION)" in Python. The interface includes a top navigation bar with the user's name "Anupam" and a sidebar with navigation icons. The main area displays a code cell (Cmd 63) with the following Python code:

```
1 rfr_model = rfr.fit(train)
2 rfr_pred = rfr_model.transform(test)
3 evaluator = RegressionEvaluator(
4     labelCol="target", predictionCol="prediction", metricName="r2")
5 accuracy = evaluator.evaluate(rfr_pred)
6 print("Accuracy on test data = %g" % (accuracy * 100))
7
8 evaluator1 = RegressionEvaluator(
9     labelCol="target", predictionCol="prediction", metricName="rmse")
10 rmse = evaluator1.evaluate(rfr_pred)
11 print("RMSE = %g" % (rmse))
```

The output of the code cell shows the results of the RFR model training and evaluation:

```
> (1) Spark Jobs
> rfr_pred: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
Accuracy on test data = 84.4573
RMSE = 156.331
Command took 9.05 seconds -- by asahay@calstatela.edu at 5/17/2018, 12:17:59 AM on CIS 5560 LAB
```

After the cross-validation process: -

The screenshot shows the same Databricks notebook, but now displaying a code cell (Cmd 64) that performs cross-validation. The code is as follows:

```
1 paramGrid = ParamGridBuilder().addGrid(rfr.maxDepth, [2,3,4,5,6]).build()
2 crossval = CrossValidator(estimator=rfr,
3     estimatorParamMaps=paramGrid,
4     evaluator=evaluator,
5     numFolds=3) # use 3+ folds in practice
6
7 crossval_rmse = CrossValidator(estimator=rfr,
8     estimatorParamMaps=paramGrid,
9     evaluator=evaluator1,
10    numFolds=3)
11
12 # Run cross-validation, and choose the best set of parameters.
13 cvModel = crossval.fit(final_data)
14 cvModel_rmse = crossval_rmse.fit(final_data)
15
16 cvPredictions = cvModel.transform(final_data)
17 accuracy = evaluator.evaluate(cvPredictions)
18 print ("Model Accuracy with Cross Validation: ", accuracy*100)
19 cvPredictions1 = cvModel_rmse.transform(final_data)
20 rmse = evaluator1.evaluate(cvPredictions1)
21 print("RMSE: ", rmse)
```

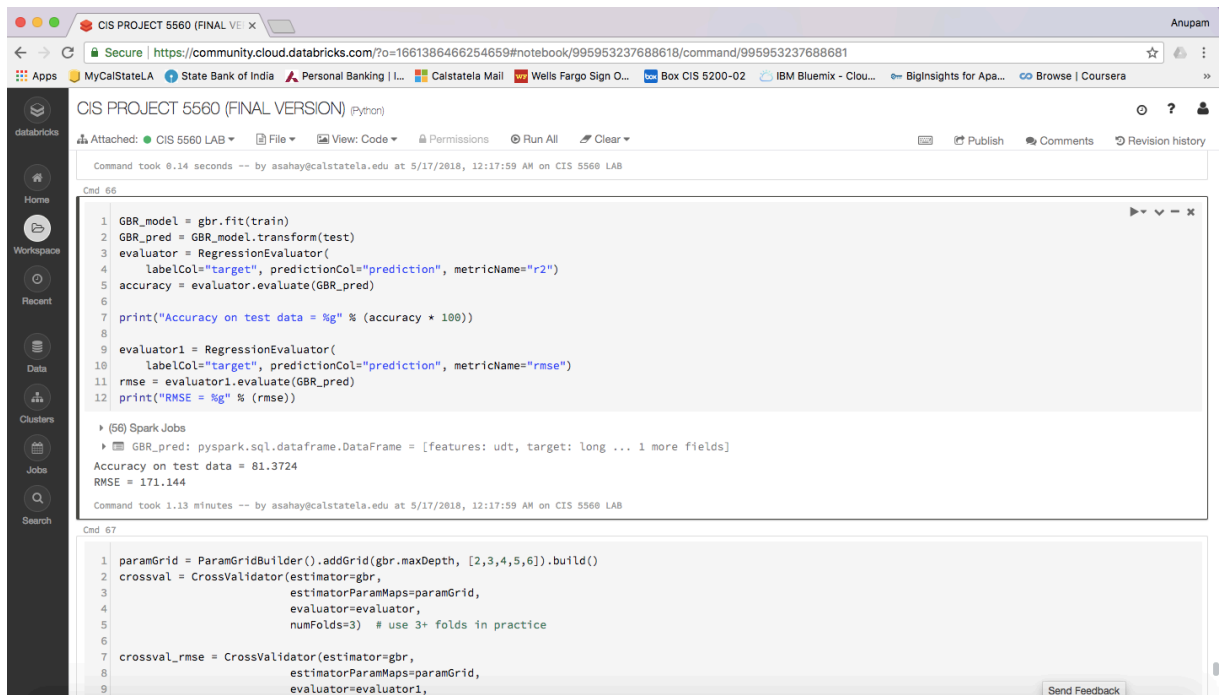
The output of the code cell shows the results of the cross-validation process:

```
> (80) Spark Jobs
> cvPredictions: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
> cvPredictions1: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
('Model Accuracy with Cross Validation: ', 94.80949987238908)
('RMSE: ', 92.21243504764885)
Command took 1.68 minutes -- by asahay@calstatela.edu at 5/17/2018, 12:17:59 AM on CIS 5560 LAB
```

('Model Accuracy with Cross Validation: ', 94.80949987238908)
('RMSE: ', 92.21243504764885)

Step 6: Model 3

- We use Gradient Boosting Regression Algorithm.
- We find out the accuracy on the test data and the root mean square error.
- The accuracy on the test data is 81.3724 and the RMSE is 171.144
- After using the cross- validation where K =3 the accuracy is improved and the RMSE is reduced significantly.



The screenshot shows a Databricks notebook titled "CIS PROJECT 5560 (FINAL VERSION)" in Python. The notebook interface includes a sidebar with navigation options like Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area displays two code cells. The first cell (Cmd 66) contains Python code for training a Gradient Boosting Regression (GBR) model and evaluating its performance on test data. The output shows an accuracy of 81.3724 and an RMSE of 171.144. The second cell (Cmd 67) contains Python code for performing cross-validation using a ParamGridBuilder and CrossValidator. The output shows the results of the cross-validation process, including the accuracy and RMSE for the cross-validated model.

```
1 GBR_model = gbr.fit(train)
2 GBR_pred = GBR_model.transform(test)
3 evaluator = RegressionEvaluator(
4     labelCol="target", predictionCol="prediction", metricName="r2")
5 accuracy = evaluator.evaluate(GBR_pred)
6
7 print("Accuracy on test data = %g" % (accuracy * 100))
8
9 evaluator1 = RegressionEvaluator(
10     labelCol="target", predictionCol="prediction", metricName="rmse")
11 rmse = evaluator1.evaluate(GBR_pred)
12 print("RMSE = %g" % (rmse))

> (56) Spark Jobs
> GBR_pred: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
Accuracy on test data = 81.3724
RMSE = 171.144
Command took 1.13 minutes -- by asahay@calstatela.edu at 5/17/2018, 12:17:59 AM on CIS 5560 LAB
```

```
1 paramGrid = ParamGridBuilder().addGrid(gbr.maxDepth, [2,3,4,5,6]).build()
2 crossval = CrossValidator(estimator=gbr,
3     estimatorParamMaps=paramGrid,
4     evaluator=evaluator,
5     numFolds=3) # use 3+ folds in practice
6
7 crossval_rmse = CrossValidator(estimator=gbr,
8     estimatorParamMaps=paramGrid,
9     evaluator=evaluator1,
```

After implementation of the cross-validation process: -

```
('Model Accuracy with Cross Validation: ', 94.32795648189204)
('RMSE: ', 96.39503484547339)
```

```
1 paramGrid = ParamGridBuilder().addGrid(gbr.maxDepth, [2,3,4,5,6]).build()
2 crossval = CrossValidator(estimator=gbr,
3                             estimatorParamMaps=paramGrid,
4                             evaluator=evaluator,
5                             numFolds=3) # use 3+ folds in practice
6
7 crossval_rmse = CrossValidator(estimator=gbr,
8                                estimatorParamMaps=paramGrid,
9                                evaluator=evaluator1,
10                               numFolds=3)
11 # Run cross-validation, and choose the best set of parameters.
12 cvModel = crossval.fit(final_data)
13 cvModel_rmse = crossval_rmse.fit(final_data)
14
15 cvPredictions = cvModel.transform(final_data)
16 accuracy = evaluator.evaluate(cvPredictions)
17 print ("Model Accuracy with Cross Validation: ", accuracy*100)
18 cvPredictions1 = cvModel_rmse.transform(final_data)
19 rmse = evaluator1.evaluate(cvPredictions1)
20 print("RMSE: ", rmse)
```

Command took 1.13 minutes -- by asahay@calstatela.edu at 5/17/2018, 12:17:59 AM on CIS 5560 LAB

(59) Spark Jobs

- cvPredictions: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]
- cvPredictions1: pyspark.sql.dataframe.DataFrame = [features: udt, target: long ... 1 more fields]

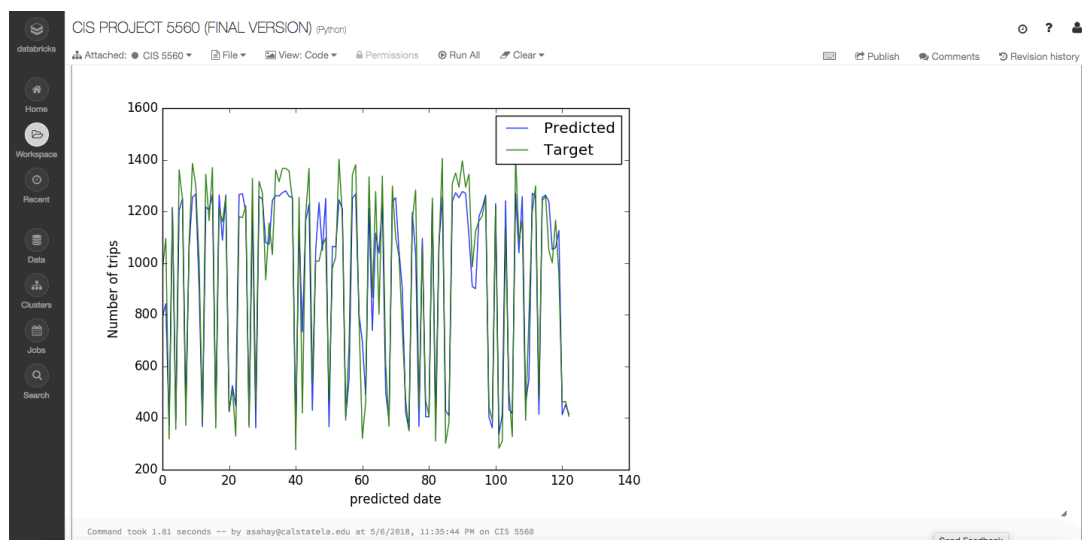
('Model Accuracy with Cross Validation: ', 94.32795648189204)

('RMSE: ', 96.39503484547339)

Command took 13.63 minutes -- by asahay@calstatela.edu at 5/17/2018, 12:18:00 AM on CIS 5560 LAB

- From above three models we can find out that the Random forest regression is the best model for the prediction.
- We make the visualization of the model by using the graphs.

Step 7: Visualization (Graph)



- In this visualization we are showing the target number of trips which takes place on a particular day, and number of trips predicted by using Random Forest Regression Algorithm.
- These models give a prediction of number of trips. We do think that we have made a good model.

References: -

1. <https://forums.databricks.com/>
2. <https://stackoverflow.com/questions/tagged/databricks>