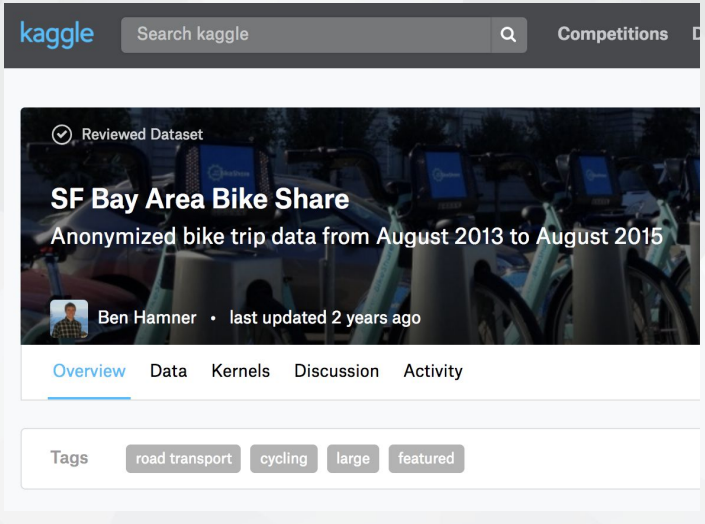
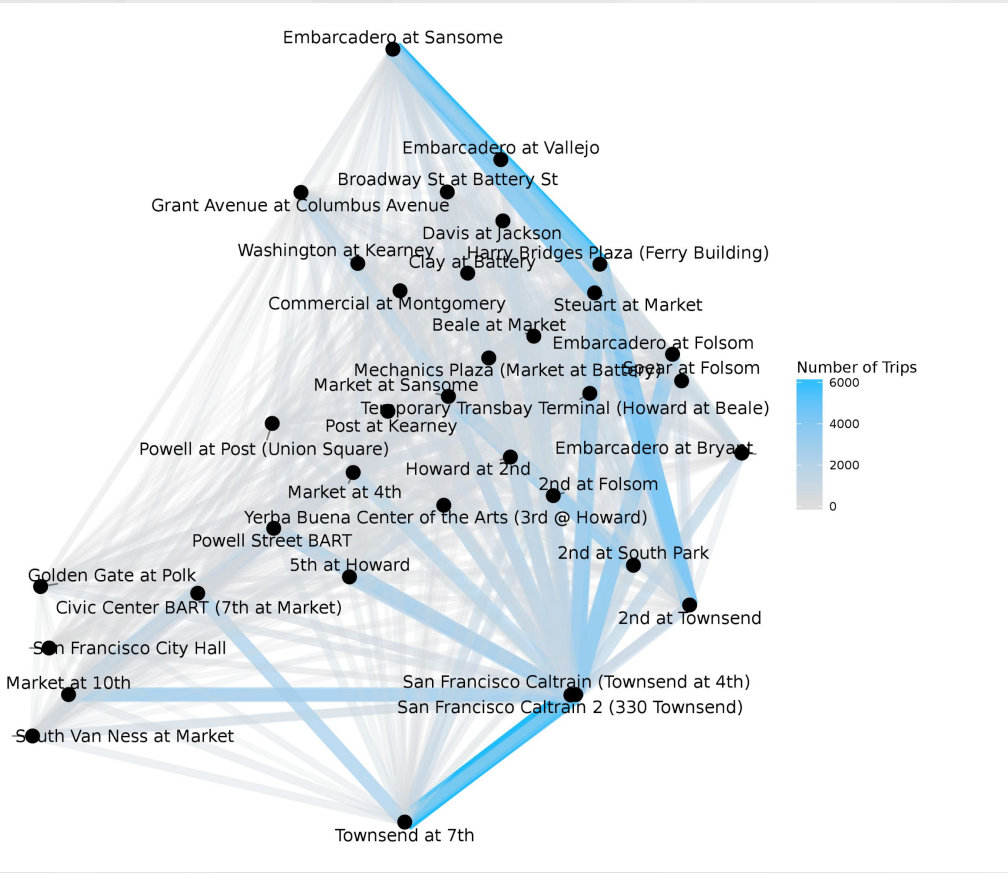
BikeShare

Jade Yun, Si Chen, Yimei Chen, Yu Tian

Data



Data Size: 2.71GB
Format: CSV
Time Window: 2013/8 - 2015/8 (2 yrs)



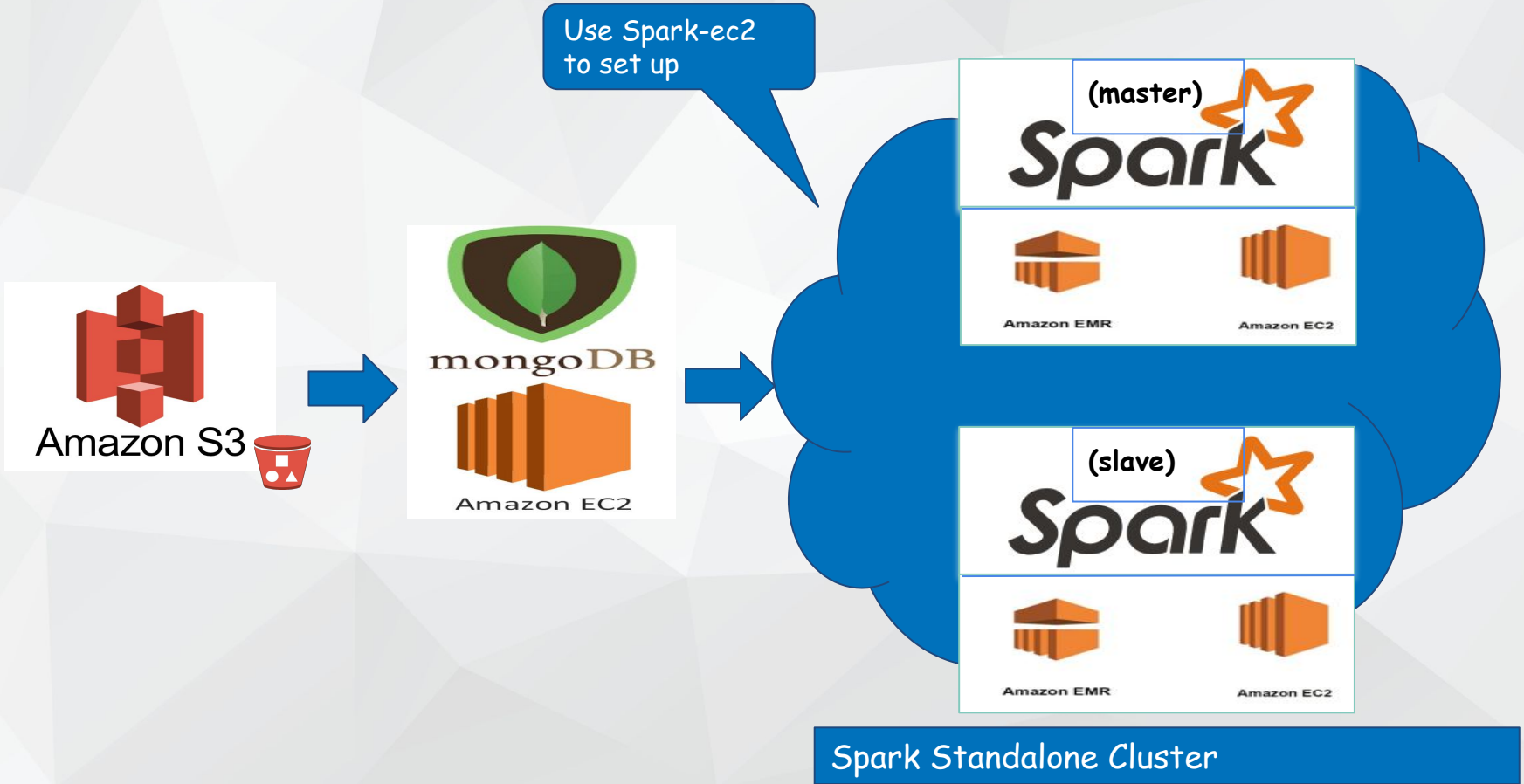
Why choose this data?

- Provide convenience to big city commute
- A lot of insights can be generated
- Interesting topic: smart city

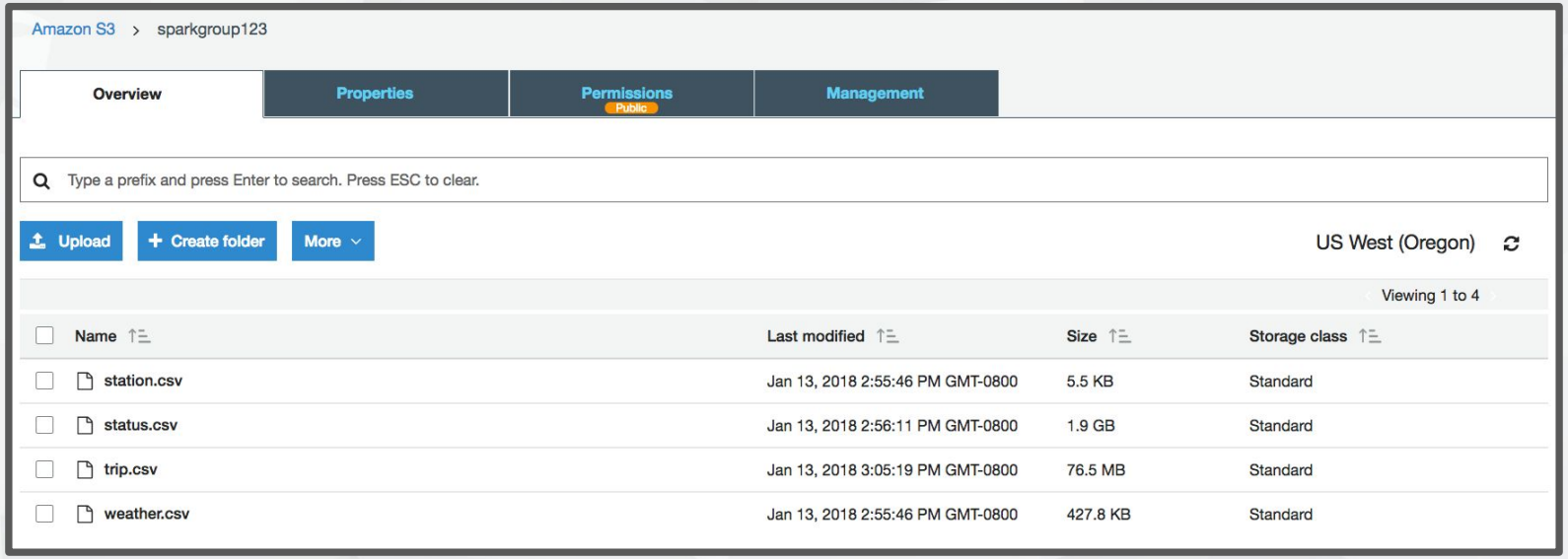
Goal?

Predicting daily demand for each station

The Pipeline We Use



1. S3 - Upload Data



For larger data Build an EC2 to download data and use command line to upload to S3

2. MongoDB Server (t2.medium)

1. Download data from S3

```
[ec2-user@ip-172-31-23-234 ~]$ aws s3 cp s3://sparkgroup123/status.csv .
download: s3://sparkgroup123/status.csv to ./status.csv
[ec2-user@ip-172-31-23-234 ~]$ aws s3 cp s3://sparkgroup123/trip.csv .
download: s3://sparkgroup123/trip.csv to ./trip.csv
[ec2-user@ip-172-31-23-234 ~]$ aws s3 cp s3://sparkgroup123/weather.csv .
download: s3://sparkgroup123/weather.csv to ./weather.csv
[ec2-user@ip-172-31-23-234 ~]$
```

Model	vCPU	CPU Credits / hour	Mem (GiB)
t2.medium	2	24	4

2. Load data into MongoDB

```
[ec2-user@ip-172-31-23-234 ~]$ mongoimport --db bikeshare --collection status --file status.csv
--type csv --headerline
2018-01-15T02:58:06.481+0000    connected to: localhost
2018-01-15T02:58:09.480+0000    [........................] bikeshare.status    9.26MB/1.85GB (0.
5%)
2018-01-15T02:58:12.480+0000    [........................] bikeshare.status    18.1MB/1.85GB (1.
0%)
2018-01-15T02:58:15.483+0000    [........................] bikeshare.status    27.2MB/1.85GB (1.
4%)
2018-01-15T02:58:18.480+0000    [........................] bikeshare.status    36.2MB/1.85GB (1.
9%)
2018-01-15T02:58:21.480+0000    [........................] bikeshare.status     45.0MB/1.85GB (2
.4%)
2018-01-15T02:58:24.480+0000    [........................] bikeshare.status     54.0MB/1.85GB (2
.8%)
2018-01-15T02:58:27.481+0000    [........................] bikeshare.status     63.3MB/1.85GB (3
.3%)
```

3. Processing & Query

```
> db.station.findOne()
{
        "_id" : ObjectId("5a5d461db2c867959cfd1198"),
        "id" : 3,
        "name" : "San Jose Civic Center",
        "lat" : 37.330698,
        "long" : -121.888979,
        "dock_count" : 15,
        "city" : "San Jose",
        "installation_date" : "8/5/2013"
}
> db.station.count()
70
> db.station.update({},{$rename:{'id':'station_id'}},{multi:true})
WriteResult({ "nMatched" : 70, "nUpserted" : 0, "nModified" : 70 })
```

What we learnt from the pipeline

Connect from my laptop to that mongodb to check data

t2.micro/t2.small: the importing status.csv (the big data) quits in the middle!!!

```
[ec2-user@ip-172-31-27-156 ~]$ mongoimport --db bikeshare --collection status --file status.csv --type csv --headerline
2018-01-17T20:51:12.861+0000    connected to: localhost
2018-01-17T20:51:15.861+0000    [........................] bikeshare.status     6.27MB/1.85GB (0.3%)
2018-01-17T20:51:18.873+0000    [........................] bikeshare.status     12.2MB/1.85GB (0.6%)
2018-01-17T20:51:21.874+0000    [........................] bikeshare.status     18.0MB/1.85GB (0.9%)
2018-01-17T20:51:24.864+0000    [........................] bikeshare.status     23.7MB/1.85GB (1.2%)
2018-01-17T20:51:27.861+0000    [........................] bikeshare.status     29.6MB/1.85GB (1.6%)
2018-01-17T20:51:30.863+0000    [........................] bikeshare.status     35.4MB/1.85GB (1.9%)
2018-01-17T20:51:33.868+0000    [........................] bikeshare.status     41.2MB/1.85GB (2.2%)
2018-01-17T20:51:36.861+0000    [........................] bikeshare.status     46.9MB/1.85GB (2.5%)
2018-01-17T20:51:39.861+0000    [........................] bikeshare.status     52.5MB/1.85GB (2.8%)
2018-01-17T20:51:42.864+0000    [........................] bikeshare.status     58.6MB/1.85GB (3.1%)
2018-01-17T20:51:45.861+0000    [........................] bikeshare.status     64.7MB/1.85GB (3.4%)
2018-01-17T20:51:48.865+0000    [........................] bikeshare.status     70.6MB/1.85GB (3.7%)
2018-01-17T20:51:51.861+0000    [........................] bikeshare.status     76.5MB/1.85GB (4.0%)
2018-01-17T20:51:54.867+0000    [#.......................] bikeshare.status     82.6MB/1.85GB (4.4%)
2018-01-17T20:51:57.861+0000    [#.......................] bikeshare.status     88.5MB/1.85GB (4.7%)
2018-01-17T20:52:00.864+0000    [#.......................] bikeshare.status     94.5MB/1.85GB (5.0%)
2018-01-17T20:52:03.868+0000    [#.......................] bikeshare.status     100MB/1.85GB (5.3%)
2018-01-17T20:52:06.870+0000    [#.......................] bikeshare.status     107MB/1.85GB (5.6%)
2018-01-17T20:52:09.861+0000    [#.......................] bikeshare.status     112MB/1.85GB (5.9%)
2018-01-17T20:52:12.869+0000    [#.......................] bikeshare.status     118MB/1.85GB (6.2%)
2018-01-17T20:52:15.861+0000    [#.......................] bikeshare.status     124MB/1.85GB (6.5%)
2018-01-17T20:52:18.861+0000    [#.......................] bikeshare.status     130MB/1.85GB (6.8%)
2018-01-17T20:52:21.862+0000    [#.......................] bikeshare.status     135MB/1.85GB (7.1%)
2018-01-17T20:52:22.759+0000    [#.......................] bikeshare.status     137MB/1.85GB (7.2%)
2018-01-17T20:52:22.760+0000    Failed: lost connection to server
2018-01-17T20:52:22.760+0000    imported 5281000 documents
```

3. Spark - Load Data from MongoDB

```
weather= spark.read.format('com.mongodb.spark.sql.DefaultSource').option('uri','mongodb://54.202.150.222/bikeshare.weath
trip= spark.read.format('com.mongodb.spark.sql.DefaultSource').option('uri','mongodb://54.202.150.222/bikeshare.trip').
station= spark.read.format('com.mongodb.spark.sql.DefaultSource').option('uri','mongodb://54.202.150.222/bikeshare.stat
status= spark.read.format('com.mongodb.spark.sql.DefaultSource').option('uri','mongodb://54.202.150.222/bikeshare.statu
```

```
station=station.drop('_id')
station.printSchema()
```

```
root
 |-- city: string (nullable = true)
 |-- dock_count: integer (nullable = true)
 |-- installation_date: string (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- name: string (nullable = true)
 |-- station_id: integer (nullable = true)
```

```
status=status.drop('_id')
status.printSchema()
```

```
root
 |-- bikes_available: integer (nullable = true)
 |-- docks_available: integer (nullable = true)
 |-- station_id: integer (nullable = true)
 |-- time: string (nullable = true)
```

```
weather=weather.drop('_id')
weather.printSchema()
```

```
root
 |-- cloud_cover: double (nullable = true)
 |-- date: string (nullable = true)
 |-- events: string (nullable = true)
 |-- max_dew_point_f: string (nullable = true)
 |-- max_gust_speed_mph: string (nullable = true)
 |-- max_humidity: string (nullable = true)
 |-- max_sea_level_pressure_inches: double (nullable = true)
 |-- max_temperature_f: double (nullable = true)
 |-- max_visibility_miles: string (nullable = true)
 |-- max_wind_Speed_mph: double (nullable = true)
 |-- mean_dew_point_f: string (nullable = true)
 |-- mean_humidity: string (nullable = true)
 |-- mean_sea_level_pressure_inches: double (nullable = true)
 |-- mean_temperature_f: double (nullable = true)
 |-- mean_visibility_miles: string (nullable = true)
 |-- mean_wind_speed_mph: double (nullable = true)
 |-- min_dew_point_f: string (nullable = true)
 |-- min_humidity: string (nullable = true)
 |-- min_sea_level_pressure_inches: double (nullable = true)
 |-- min_temperature_f: double (nullable = true)
 |-- min_visibility_miles: string (nullable = true)
 |-- precipitation_inches: string (nullable = true)
 |-- wind_dir_degrees: double (nullable = true)
 |-- zip_code: integer (nullable = true)
```

```
trip=trip.drop('_id')
trip.printSchema()
```

```
root
 |-- bike_id: integer (nullable = true)
 |-- duration: integer (nullable = true)
 |-- end_date: string (nullable = true)
 |-- end_station_id: integer (nullable = true)
 |-- end_station_name: string (nullable = true)
 |-- start_date: string (nullable = true)
 |-- start_station_id: integer (nullable = true)
 |-- start_station_name: string (nullable = true)
 |-- subscription_type: string (nullable = true)
 |-- trip_id: integer (nullable = true)
 |-- zip_code: string (nullable = true)
```

3. Spark - Load Data from MongoDB

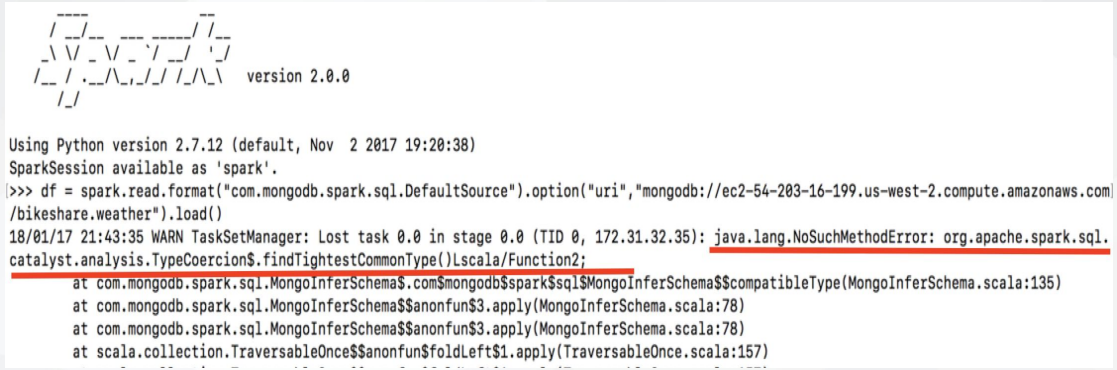
API Name	Memory	vCPUs	Instance Storage	Network Performance
m1.small	1.7 GiB	1 vCPUs	160 GiB HDD + 900MB swap	Low
m1.medium	3.75 GiB	1 vCPUs	410 GiB HDD	Moderate
m1.large	7.5 GiB	2 vCPUs	840 GiB (2 * 420 GiB HDD)	Moderate

What we learnt from the pipeline

Set up mongo-spark-connect

Make sure the version of the mongo-spark-connector is consistent with the spark version on the cluster!!

Failing this will result in the error when run the spark code.



Tip: to find the version of spark

```
$ spark-submit --version
```

Spark version	connector	note
2.2.x	`org.mongodb.spark:mongo-spark-connector_2.11:2.2.0`	Usually on the Mac development machine
2.0.x	`org.mongodb.spark:mongo-spark-connector_2.11:2.0.0`	One the spark cluster set up by spark-ec2

Feature Selection

Station	Trips	Weather
Features: Station ID, Station Name, latitude, longitude, number of Bikes available, city, install date	Features: id, duration, start date, start station name, end date, end station name, bike id, subscription type, zip code, Number of Trip	Features: temperature, dew point, humidity, sea level pressure, visibility, wind speed, gust speed, precipitation, cloud cover, weather condition, wind direction, zip code, date

- Join
- Generate New Features

Final Dataframe
Features: Holiday , Day of Week, Station ID, Station Name, latitude, longitude, number of Bikes available,duration, start date, start station name, end date, end station name, bike id, subscription type, zip code, temperature, dew point, humidity, sea level pressure, visibility, wind speed, gust speed, precipitation, cloud cover, weather condition, wind direction, Number of Trips (label)

Label

```
tripRaw['start_date'] = tripRaw['start_date'].dt.floor('d')
```

```
tripAgg = tripRaw.groupby(['start_date','zip_code', 'start_station_name'],
                          as_index=False)['id'].count()
```

```
tripAgg=tripAgg.rename(columns={'id': "num_trips"})
```

```
tripAgg.head()
```

	start_date	zip_code	start_station_name	num_trips
0	2013-08-29	10003	California Ave Caltrain Station	2
1	2013-08-29	10003	University and Emerson	2
2	2013-08-29	10009	Commercial at Montgomery	1
3	2013-08-29	10009	Mechanics Plaza (Market at Battery)	1
4	2013-08-29	10009	South Van Ness at Market	1

5. Processing

```
trip.show(5)
```

```
+--------------------+-------+--------+--------------+--------------+--------------------+----+--------------+-----
-----------+----------+-----------------+--------+
|                 _id|bike_id|duration|      end_date|end_station_id|    end_station_name|  id|    start_date|start
_station_id|  start_station_name|subscription_type|zip_code|
+--------------------+-------+--------+--------------+--------------+--------------------+----+--------------+-----
-----------+----------+-----------------+--------+
|[5a5d4642b2c86795...|    520|      63|8/29/2013 14:14|            66|South Van Ness at...|4576|8/29/2013 14:13|
          66|South Van Ness at...|       Subscriber|   94127|
|[5a5d4642b2c86795...|    661|      70|8/29/2013 14:43|            10|  San Jose City Hall|4607|8/29/2013 14:42|
          10|  San Jose City Hall|       Subscriber|   95138|
|[5a5d4642b2c86795...|     48|      71|8/29/2013 10:17|            27|Mountain View Cit...|4130|8/29/2013 10:16|
          27|Mountain View Cit...|       Subscriber|   97214|
|[5a5d4642b2c86795...|     26|      77|8/29/2013 11:30|            10|  San Jose City Hall|4251|8/29/2013 11:29|
          10|  San Jose City Hall|       Subscriber|   95060|
|[5a5d4642b2c86795...|    527|     103|8/29/2013 18:56|            59| Golden Gate at Polk|4927|8/29/2013 18:54|
          59| Golden Gate at Polk|       Subscriber|   94109|
+--------------------+-------+--------+--------------+--------------+--------------------+----+--------------+-----
-----------+----------+-----------------+--------+
only showing top 5 rows
```

```
df = targetDf.drop('_id','unnamed', 'date','installation_date', 'city')
```

```
df = df.na.drop()
```

```
#converting strings to numeric values
from pyspark.ml.feature import StringIndexer

def indexStringColumns(df, cols):
    #variable newdf will be updated several times
    newdf = df

    for c in cols:
        #For each given colum, fits StringIndexerModel.
        si = StringIndexer(inputCol=c, outputCol=c+"-num")
        sm = si.fit(newdf)
        #Creates a DataFame by putting the transformed values in the new colum with suffix "-num"
        #and then drops the original columns.
        #and drop the "-num" suffix.
        newdf = sm.transform(newdf).drop(c)
        newdf = newdf.withColumnRenamed(c+"-num", c)
    return newdf

dfnumeric = indexStringColumns(df,['start_station_name','events', 'dock_count'])
```

```
# Merging the data with Vector Assembler.
from pyspark.ml.feature import VectorAssembler
input_cols = ['zip_code','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_sea_level_pressure_inches', 'mean_sea_level_pressure_inches',
        'min_sea_level_pressure_inches', 'max_visibility_miles','mean_visibility_miles',
        'min_visibility_miles', 'max_wind_Speed_mph', 'mean_wind_speed_mph',
        'max_gust_speed_mph','precipitation_inches1','cloud_cover','wind_dir_degrees',
        'station_id', 'lat','long','holidays','day_of_week','start_station_name',
        'events','dock_count']

#VectorAssembler takes a number of collumn names(inputCols) and output column name (outputCol)
#and transforms a DataFrame to assemble the values in inputCols into one single vector with outputCol.
va = VectorAssembler(outputCol="features", inputCols=input_cols)
#lpoints - labeled data.
df_final = va.transform(dfnumeric).select("features", "label")
```

```
dfSets = df_final.randomSplit([0.8, 0.2], 1)
dfTrain = dfSets[0].cache()
dfTest = dfSets[1].cache()
```

Schema — Incorrect schema prevents model from fitting

Null Value — If data contains null value, model won't fit

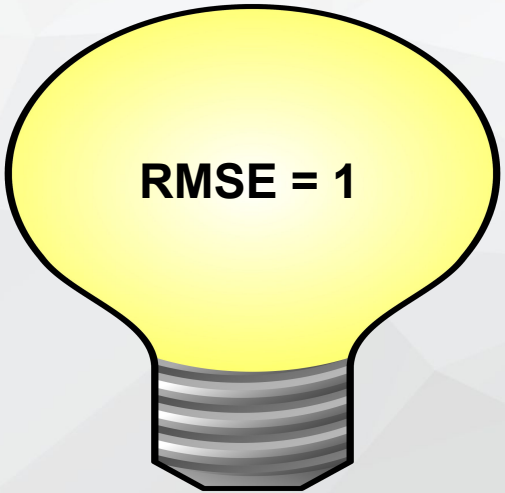
5. Modeling & Results

```
dfSets = df_final.randomSplit([0.8, 0.2], 1)
dfTrain = dfSets[0].cache()
dfTest = dfSets[1].cache()
```

```
from pyspark.ml.regression import RandomForestRegressor
rf = RandomForestRegressor(maxDepth=20, maxBins=70)
rfmodel = rf.fit(dfTrain)
```

```
from pyspark.ml.evaluation import RegressionEvaluator
rfpredicts = rfmodel.transform(dfTest)
evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(rfpredicts)
print("RMSE = %d" % rmse)
```

```
RMSE = 1
```




```
rfpredicts.show(7)
```

```
+--------------------+-----+----------+
|            features|label|prediction|
+--------------------+-----+----------+
|[94041.0,56.0,49....|    1|       1.6|
|[94041.0,58.0,48....|    1|      1.55|
|[94041.0,60.0,51....|    1|       1.9|
|[94041.0,61.0,50....|    1|       1.4|
|[94041.0,61.0,56....|    2|       1.8|
|[94041.0,63.0,51....|    1|       1.8|
|[94041.0,63.0,56....|    2|       1.8|
+--------------------+-----+----------+
only showing top 7 rows
```

6. Performance Evaluation - modeling.py pipeline

Read in data and create spark dataframe

Numericalize string to numbers

Assemble data into vector and split into training and test set

Fit the model and report the result

- time spark-submit modeling.py > output.txt
  final dataframe schema, baseline rmse

- real: wall clock time
- user: CPU time spent in user-mode code (outside the kernel) within the process
- Sys: CPU time spent in the kernel within the process
- Sys + user:how much actual CPU time the process used

6. Performance (local)

After time spark-submit modeling.py > output.txt
and finger crossed...

```
real    1m16.128s
user    1m42.300s
sys     0m2.709s
```

6. Spark-EC2 Cluster - 2 slaves (m1.large)

Spark Master at spark://ip-172-31-22-51.us-west-2.compute.internal:7077

URL: spark://ip-172-31-22-51.us-west-2.compute.internal:7077
REST URL: spark://ip-172-31-22-51.us-west-2.compute.internal:6066 *(cluster mode)*
Alive Workers: 2
Cores in use: 4 Total, 0 Used
Memory in use: 12.1 GB Total, 0.0 B Used
Applications: 0 Running, 2 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20180118212347-172.31.26.153-52060	172.31.26.153:52060	ALIVE	2 (0 Used)	6.1 GB (0.0 B Used)
worker-20180118212347-172.31.26.73-47901	172.31.26.73:47901	ALIVE	2 (0 Used)	6.1 GB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20180118220930-0001	sparkml	4	5.8 GB	2018/01/18 22:09:30	root	FINISHED	2.0 min

Spark Worker at 172.31.26.153:52060

ID: worker-20180118212347-172.31.26.153-52060
Master URL: spark://ip-172-31-22-51.us-west-2.compute.internal:7077
Cores: 2 (0 Used)
Memory: 6.1 GB (0.0 B Used)

Back to Master

Running Executors (0)

ExecutorID	Cores	State	Memory	Job Details	Logs

Finished Executors (3)

ExecutorID	Cores	State	Memory	Job Details	Logs
0	2	KILLED	5.8 GB	**ID:** app-20180118215852-0000 **Name:** sparkml **User:** root	stdout stderr

Spark-EC2

```
real    2m10.093s
user    1m5.436s
sys     0m5.008s
```

6. Performance Comparisons

Spark-EC2

```
real    2m10.093s
user    1m5.436s
sys     0m5.008s
```

Local

```
real    1m16.128s
user    1m42.300s
sys     0m2.709s
```

Time saved on AWS: 32 s

6. Lesson learnt

1. Data size might change
2. Machine selection decision at different stage should be flexible
3. Never stuck in the idea of “big data” and ignore cost efficiency

The End