Final Project

"Analysis of Residential Electricity Usage in California"

Due: Tuesday, November 22, 6:30 p.m.

Total Points: 100

Submission: Upload and Presentation

Task List: See the following table

Step	Descriptor	Description
1	IoT Device Data Message Use Case and Simulator	Create a simulated use-case that utilizes a basic network of IoT devices. Generate device data messages for 10,000 messages using an IoT device data message simulator. **Note: As a result, save 10,000 messages in a file and attach the file to the assignment. 50 Points.**
2	IoT Device Data Message Processing/Analysis	Run analysis of data provided by IoT device data simulator (step 1) in the cloud using one of the Big Data processing or analytics techniques learned in class (e.g. modified versions of lab 8 or lab 9). Note: As a result, save a representational sampling of your output in a file and attach the file to the assignment. 30 Points.
3	Presentation of Results from IoT Device Data Message Simulation (Step 1) and Processing/Analysis (Step 2)	Either provide a written description of steps 1 and 2, or present a 5-minute (1 slide) presentation during the last class. **Note: Sign up and present your 5-minute (1 slide) summary at the final class or attach a file with a written description of items/steps 1 and 2 (above) to this assignment. 20 Points.**

Step 1: IoT Device Message Data Use Case and Message Data Simulation/Provision

IoT Device Message Data Use Case

The IoT device message data use case presented here is based on a network of 3 IoT devices used to measure (1) electricity usage (from smart meter), (2) outside temperature at customer premise (from temperature sensor), and (3) outside relative humidity at customer premise (from humidity sensor). The IoT device message data (time-series data) covers a period of approximately 1.5 years (about 576 days). The data used for this project originated from the California Statewide Pricing Pilot which used a representative sample of customer data from a variety of different climate areas found in California.

The following overview presented in figure 1 shows the data frame structure established for each of the 2,494 residential customers as a source to provide the data from the 3 IoT devices:

Data Frame – Customer 1

Customer ID	Date	Time	Temp	Hum	Electricity Usage
A01PMA 00102	Day 1	12 AM	T	H	U
A01PMA 00102	Day 1	1 AM	T	H	U
…	…	…	…	…	…
A01PMA 00102	Day 1	11 PM	T	H	U
A01PMA 00102	Day 2	12 AM	T	H	U
…	…	…	…	…	…
A01PMA 00102	Day 576	12 AM	T	H	U
…	…	…	…	…	…
A01PMA 00102	Day 576	11 PM	T	H	U

Data Frame – Customer 2,494

Customer ID	Date	Time	Temp	Hum	Electricity Usage
R08PSL 20703	Day 1	12 AM	T	H	U
R08PSL 20703	Day 1	1 AM	T	H	U
…	…	…	…	…	…
R08PSL 20703	Day 1	11 PM	T	H	U
R08PSL 20703	Day 2	12 AM	T	H	U
…	…	…	…	…	…
R08PSL 20703	Day 576	12 AM	T	H	U
…	…	…	…	…	…
R08PSL 20703	Day 576	11 PM	T	H	U

Figure 1: Overview of 2,494 Data Frames for 2,494 Residential Customers

As shown in figure 1, the data frame for each customer is composed of the following variables:

1. Customer ID
2. Date, Time
3. Hourly Temperature
4. Hourly Humidity
5. Hourly Electricity Usage.

While the first four variables (customer ID, date and time, hourly temperature, hourly humidity) represent the input variables, the hourly electricity usage reflects the dependent output variable. The following figure 2 illustrates it:

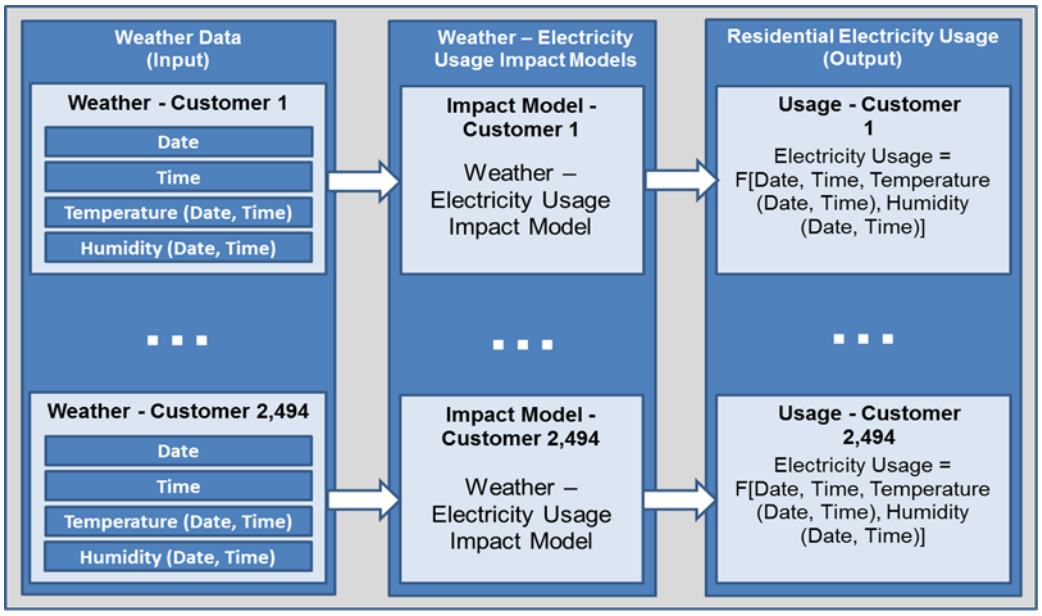

Figure 2: Weather – Residential Electricity Usage Input – Output Model

Given the stated IoT problem, the following assumption can be made:

- For each residential customer, we have 3 sensors which deliver data, i.e. (1) smart meter (measures and delivers hourly electricity usage data), (2) temperature sensor (measures and delivers hourly temperature data), (3) humidity sensor (measures and delivers hourly humidity data).
- For each customer and point in time t_0, the measurement set is composed of {Electricity Usage (t_0);Temperature(t_0);Humidity(t_0)} coming from a set of 3 IoT devices composed of {smart meter; temperature sensor; humidity sensor}.

- Assuming hourly data from each of the 3 sensors (electricity usage, temperature, humidity) available for 576 days (about 1.5 years of hourly data) we will need 13,824 data records (576 days times 24 hours per day) per sensor. Given 3 sensors we will need 3 files (1 file per sensor) with 13,824 data records in JSON format each. Note: We will select one customer for whom 12,432 data records will be available (i.e. hourly data for usage, temperature and humidity over a period of 518 days).

IoT Device Message Data Provision/Simulation

Assuming one customer for this project, the data source will be 1 data frame (CSV File) as shown in figure 1. The data stored in the data frame will be converted into 3 JSON files each of which will hold 12,432 JSON compliant data records (for usage, temperature, relative humidity). The following diagram illustrates the data conversion from the data frame originally fed by the 3 IoT devices (smart meter, temperature and humidity sensors) as the source with actual data via a Python script which converts the data into the 3 necessary JSON files with 12,432 data records each:

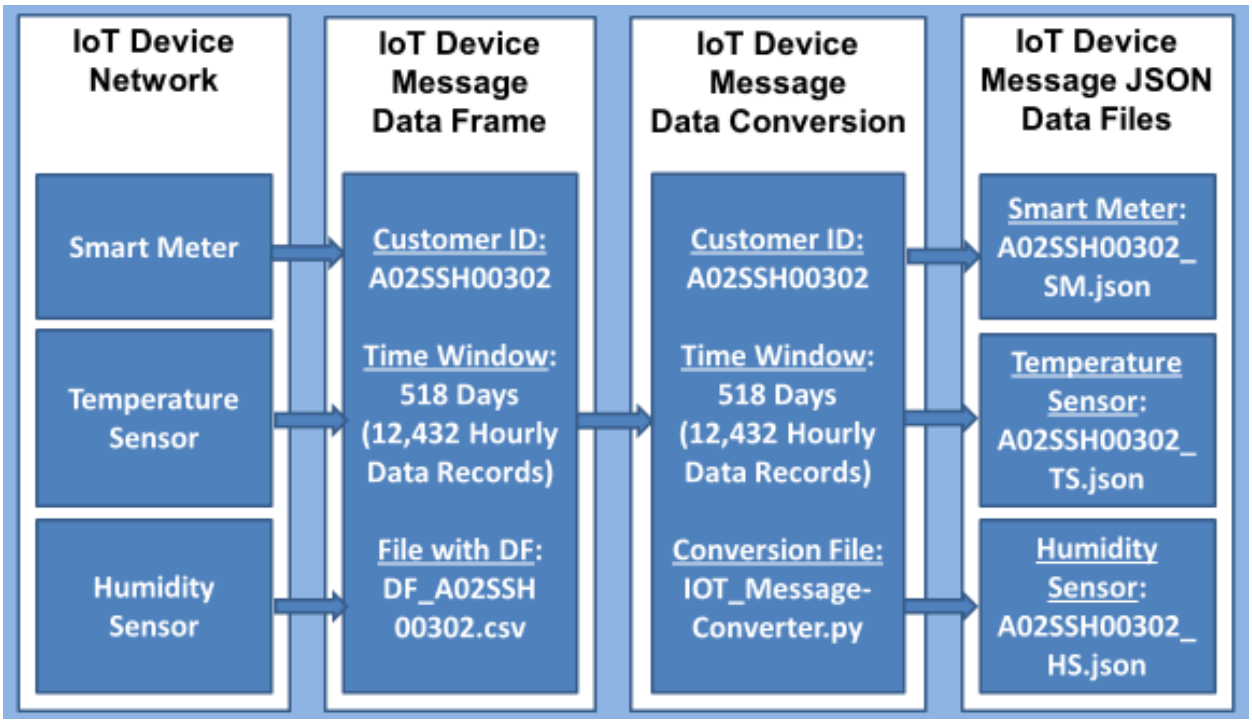

Figure 3: Data Conversion Process for 1 of 2,494 Residential Customers

As shown above, the outcome of the data simulation and conversion process is a triple of JSON files each of which is holding 12,432 hourly data records in JSON format to be used in the next step to stream data and apply some simple analysis to it.

The source code of the IoT device message data conversion file “IOT_Message-Converter.py” is displayed in the following:

IoT Device Message Data Converter (Python)

```
#!/usr/bin/python

import sys
import datetime
import random
import string
import csv

# Set number of simulated messages to generate
# if len(sys.argv) > 1:
#   numMsgs = int(sys.argv[1])
# else:
#   numMsgs = 1

csvfile = file('DF_A02SSH00302.csv','r')
csvreader = csv.reader(csvfile)

# Fixed values
smartmeterIDStr = "SM000543218"
formatStr1 = "urn:com:smartmeter:powerusageinkW"
formatStr2 = "urn:com:tempsensor:temperatureinF"
formatStr3 = "urn:com:humsensor:relativehumidityin%"

iotmsg_header = """\
{
  "smartmeterID": "%s",
  "customerID": "%s",
"""


iotmsg_eventTime = """\
  "eventTime": "%sZ",
"""


iotmsg_payload ="""\
  "payload": {
     "format": "%s",
"""
```

IoT Device Message Data Converter (Python)

```
iotmsg_data1 ="""\
     "data": {
       "Power": %s
     }
   }
}"""

iotmsg_data2 ="""\
     "data": {
       "Temperature": %s
     }
   }
}"""

iotmsg_data3 ="""\
     "data": {
       "Humidity": %s
     }
   }
}"""

##### Generate JSON output:
files_opened = 0

dataElementDelimiter = ","
for row in csvreader:

  # CustId,1
  # Date  ,2
  # Hour  ,3
  # Rhum  ,4
  # Temp  ,5
  # Kw    ,6

customerIDStr = row[1]

  if (row[2] == "DATE"):  # Header line skipping
    continue

  if ( (row[1] != "DATE") and (files_opened != 1)):
    powr_out = file(row[1] + '_SM.json','w')
    temp_out = file(row[1] + '_TS.json','w')
    rhum_out = file(row[1] + '_HS.json','w')
    powr_out.write("[")
    temp_out.write("[")
    rhum_out.write("[")
    files_opened = 1
```

IoT Device Message Data Converter (Python)

```
date_s  = row[2].split(" ")
  date_sp = date_s[0].split("/")
  year    = date_sp[2]
  month   = date_sp[0]
  day     = date_sp[1]

  hour    = row[3]

  if (int(month) < 10):
    month = "0" + month

  if (int(day) < 10):
    day = "0" + day

  if (int(hour) < 10):
    hour = "0" + hour

  if (int(hour) == 24):
    hour = "00"

  datestr = year + "-" + month + "-" + day + "T" + hour + ":" + "00:00.000000"

powr_out.write( iotmsg_header % (smartmeterIDStr, customerIDStr) )
  temp_out.write( iotmsg_header % (smartmeterIDStr, customerIDStr) )
  rhum_out.write( iotmsg_header % (smartmeterIDStr, customerIDStr) )
  powr_out.write( iotmsg_eventTime % (datestr)                    )
  temp_out.write( iotmsg_eventTime % (datestr)                    )
  rhum_out.write( iotmsg_eventTime % (datestr)                    )
  powr_out.write( iotmsg_payload % (formatStr1)                   )
  temp_out.write( iotmsg_payload % (formatStr2)                   )
  rhum_out.write( iotmsg_payload % (formatStr3)                   )
  powr_out.write( iotmsg_data1 % (row[6]) + dataElementDelimiter   )
  temp_out.write( iotmsg_data2 % (row[5]) + dataElementDelimiter   )
  rhum_out.write( iotmsg_data3 % (row[4]) + dataElementDelimiter   )
```

IoT Device Message Data Converter (Python)

```
  # print iotmsg_header % (smartmeterIDStr, customerIDStr)
  # print iotmsg_eventTime % (datestr)
  # print iotmsg_payload % (formatStr)
  # print iotmsg_data % (row[6]) + dataElementDelimiter

# for counter in range(0, numMsgs):
#
#   print iotmsg_header % (smartmeterIDStr, customerIDStr)
#
#   today = datetime.datetime.today()
#   datestr = today.isoformat()
#   print iotmsg_eventTime % (datestr)
#
#   print iotmsg_payload % (formatStr)
#
#   # Generate a random floating point number
#   randPower = random.uniform(0.0, 0.3) + 0.01
#   if counter == numMsgs - 1:
#     dataElementDelimiter = ""
#   print iotmsg_data % (randPower) + dataElementDelimiter

powr_out.write( "]" )
temp_out.write( "]" )
rhum_out.write( "]" )
```

Applying the above Python file “IOT_Message-Converter.py” the following JSON files are generated from the data frame file “DF_A02SSH00302”:

IoT Device Message Data File from Smart Meter Data (JSON): A02SSH00302_SM.json (12,432 Records)


```
[{
  "smartmeterID": "SM000543218",
  "customerID": "A02SSH00302",
  "eventTime": "2003-05-02T01:00:00.000000Z",
  "payload": {
     "format": "urn:com:smartmeter:powerusageinkW",
     "data": {
       "Power": 0.52
     }
   }
},

…

 {
  "smartmeterID": "SM000543218",
  "customerID": "A02SSH00302",
  "eventTime": "2004-09-30T00:00:00.000000Z",
  "payload": {
     "format": "urn:com:smartmeter:powerusageinkW",
     "data": {
       "Power": 1.88
     }
   }
},]
```

IoT Device Message Data File from Temperature Sensor Data (JSON): A02SSH00302_TS.json (12,432 Records)


```
[{
  "smartmeterID": "SM000543218",
  "customerID": "A02SSH00302",
  "eventTime": "2003-05-02T01:00:00.000000Z",
  "payload": {
     "format": "urn:com:tempsensor:temperatureinF",
     "data": {
       "Temperature": 54
     }
   }
},{


…

{
  "smartmeterID": "SM000543218",
  "customerID": "A02SSH00302",
  "eventTime": "2004-09-30T00:00:00.000000Z",
  "payload": {
     "format": "urn:com:tempsensor:temperatureinF",
     "data": {
       "Temperature": 65
     }
   }
},]
```

IoT Device Message Data File from Humidity Sensor Data (JSON): A02SSH00302_HS.json (12,432 Records)


```
[{
  "smartmeterID": "SM000543218",
  "customerID": "A02SSH00302",
  "eventTime": "2003-05-02T01:00:00.000000Z",
  "payload": {
     "format": "urn:com:humsensor:relativehumidityin%",
     "data": {
       "Humidity": 83
     }
   }
},


…

{
  "smartmeterID": "SM000543218",
  "customerID": "A02SSH00302",
  "eventTime": "2004-09-30T00:00:00.000000Z",
  "payload": {
     "format": "urn:com:humsensor:relativehumidityin%",
     "data": {
       "Humidity": 73
     }
   }
},]
```

Step 2: IoT Device Message Data Processing & Analysis

IoT Device Message Data Processing and Analysis - Overview

The following configuration of terminal windows serves to run the IoT device message data processing and analysis:

Terminal Window 1	Terminal Window 2	Terminal Window 3
Working Directory: /home/ec2-user/kafka_2.11-0.10.1.0	Working Directory: /home/ec2-user/	Working Directory: /home/ec2-user/
Steps performed: ➢ Step 1: Stop kafka ➢ Step 2: Stop zookeeper ➢ Step 3: Start zookeeper ➢ Step 4: Start kafka	Steps performed: ➢ Step 5: Run kafka streaming and analysis files to stream IoT device message data from 3 IoT devices (smart meter, temperature and humidity sensors) once data is sent by them (Listening if IoT device message is available and if so stream, analyze and display)	Steps performed: ➢ Step 6: Run JSON files with data from 3 chosen IoT devices (smart meter, temperature sensor, humidity sensor) to send 12,432 IoT messages each and run simple analysis of the JSON data.

There are 3 scenarios leveraging the following IoT device message simulator and streaming files including the directory where those files are located:

Scenario	IoT Device Message Simulator File (JSON)	IoT Device Message Processor/Analyzer File (Python)	File Directory (for Python and JSON Files)
Smart Meter Device	A02SSH00302_SM.json	kafka-direct-iotmsg-power-meter.py	/home/ec2-user
Temperature Sensor Device	A02SSH00302_TS.json	kafka-direct-iotmsg-temp-sensor.py	/home/ec2-user
Humidity Sensor Device	A02SSH00302_HS.json	kafka-direct-iotmsg-hum-sensor.py	/home/ec2-user

The following figure 4 illustrates the approach for processing and analyzing the 12,432 JSON hourly records coming from each of the 3 IoT devices (smart meter, temperature sensor, humidity sensor):

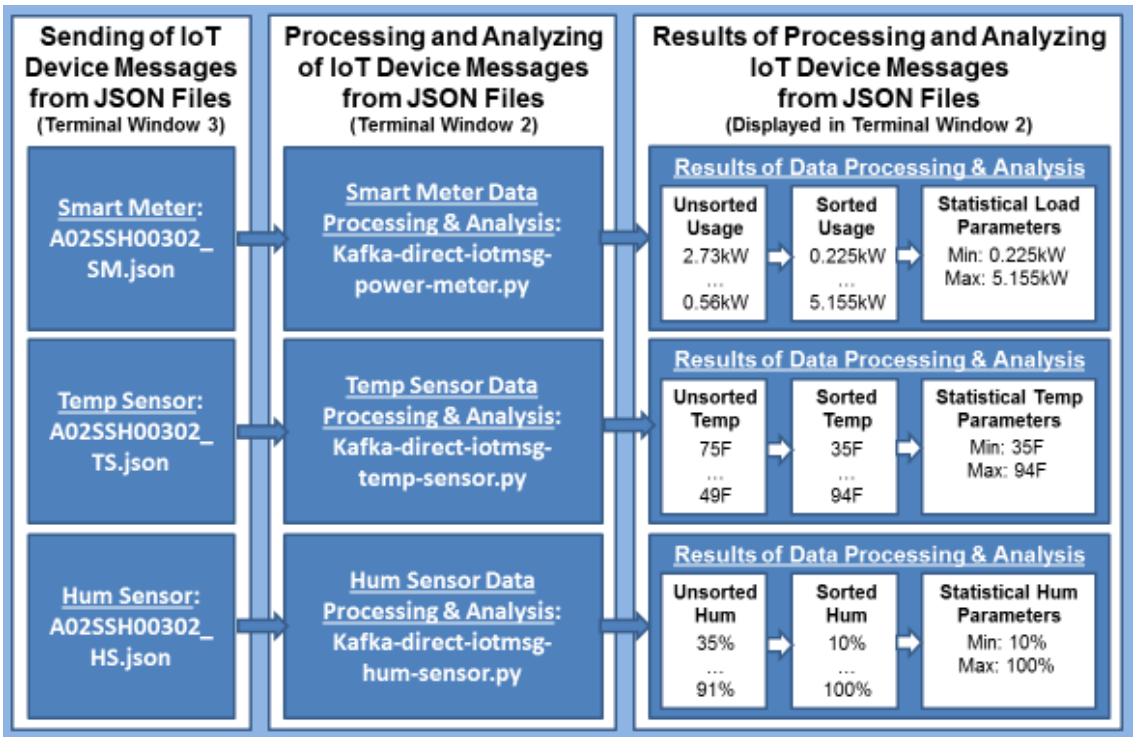

Figure 4: Data Processing & Analysis for 1 of 2,494 Residential Customers

Scenario 1: (1) Send 12,432 IoT Smart Meter Device Messages (Electricity Usage) sourced from JSON file (A02SSH00302_SM.json); (2) Process and analyze 12,432 IoT Smart Meter Device Messages (Electricity Usage) with IoT Device Message Processor and Analyzer (kafka-direct-iotmsg-power-meter.py)

The command line approach for scenario 1 in terms of terminal window configuration and command sequence for each terminal window is summarized in the following tables:

Terminal Window 1
(Working directory: /home/ec2-user/kafka_2.11-0.10.1.0)

Open terminal window 1 and perform the following sequence of commands to clean (stopping zookeeper and kafka) and start zookeeper and kafka:

```
$ cd kafka_2.11-0.10.1.0

$ bin/kafka-server-stop.sh
$ bin/zookeeper-server-stop.sh
$ pkill -9 java
$ pkill -9 python

$ export KAFKA_HEAP_OPTS="-Xmx128M -Xms64M"
$ bin/zookeeper-server-start.sh config/zookeeper.properties > \
>/tmp/zookeeper.log 2>&1 & <ENTER>

$ export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"
$ bin/kafka-server-start.sh config/server.properties > \
>  /tmp/kafka.log 2>&1 & <ENTER>
```

The result of executing the above sequence will be a clean restart of zookeeper and kafka.

Terminal Window 2
(Working directory: /home/ec2-user)

Open terminal window 2 and perform the following sequence of commands to stream IoT device message data using the Big Data processing file “kafka-direct-iotmsg-power-meter.py”:

```
$ export KAFKA_HEAP_OPTS="-Xmx64M -Xms48M"
$ spark-submit \
>  --jars spark-streaming-kafka-0-8-assembly_2.11-2.0.0-preview.jar \
> ./kafka-direct-iotmsg-power-meter.py localhost:9092 iotmsgs <ENTER>
```

The result of executing the above sequence will be a continuous sequence of time stamps displayed in terminal window 2 until stopped with <CTRL C>. Terminal window 2 serves to display the data streaming and analysis results.

Terminal Window 3
(Working directory: /home/ec2-user)

Open terminal window 3 and perform the following sequence of commands to send actual IoT smart meter device message data stored in JSON file “A02SSH00302_SM.json”:

```
$ cd ${HOME}
$ export KAFKA_HEAP_OPTS="-Xmx64M -Xms48M"
$ cat A02SSH00302_SM.json | ./kafka_2.11-0.10.1.0/bin/kafka-console-producer.sh \
>  --broker-list localhost:9092 \
>  --topic iotmsgs <ENTER>
```

The result of executing the above sequence will be 12,432 actual IoT smart meter device messages sent by the IoT smart meter device message holder “A02SSH00302_SM.json” to be streamed and displayed by the IoT device message data streaming and analyzer file “kafka-direct-iotmsg-power-meter.py” in terminal window 2.

Note that the execution of the above command sequence in terminal window 3 completes by concluding with the normal prompt “$” when finished. The streaming of the IoT device messages is displayed in terminal window 2 which serves as the data streaming display.

The results of processing and analyzing the 12,432 IoT smart meter device messages (hourly electricity usage data from 518 consecutive days) in terminal window 2 are displayed in the following:

Time: 2016-11-21 07:40:04

Time: 2016-11-21 07:40:06

"Power":0.52

"Power":0.54

"Power":0.465

"Power":0.57

"Power":0.47

"Power":0.495

"Power":0.575

"Power":0.47

"Power":0.97

"Power":1.915

"Power":2.0475

"Power":1.9225

"Power":1.93

"Power":2.1475

"Power":1.4775

"Power":0.5875

"Power":0.4

"Power":1.18

"Power":0.84

"Power":0.45

"Power":0.91

"Power":1.315

"Power":0.825

"Power":0.525

"Power":0.545

"Power":0.51

"Power":0.515

"Power":0.495

"Power":0.59

"Power":0.465

"Power":0.63

"Power":0.475

"Power":0.6025

"Power":2.045

"Power":2.175

"Power":2.15

"Power":2.41

"Power":2.0725

"Power":2.255

"Power":0.835

"Power":0.84

"Power":0.885

"Power":1.575

"Power":0.695

"Power":1.025

"Power":0.955

"Power":0.625

"Power":0.72

"Power":0.3975

…

"Power":0.9

"Power":1.1

"Power":1.035

"Power":0.735

"Power":0.595

Time: 2016-11-21 07:40:06

Time: 2016-11-21 07:40:06

0.225

0.225

0.23

0.23

0.23

0.23

0.235
0.235
0.24
0.24
0.24
0.245
0.245
0.245
0.245
0.245
0.245
0.25
0.25
0.25
0.25
0.25
0.25
0.25
0.25
0.25
0.25
0.25
0.255
0.255

0.255

0.255

0.26

0.26

0.26

0.26

0.26

0.265

0.265

0.265

…

3.43

3.45

3.4675

3.4875

3.49

3.565

3.575

3.59

3.6

3.6025

3.605

3.7

3.75

3.76

3.765

3.775

3.82

3.835

3.885

3.935

3.95

3.95

3.985

4.06

4.065

4.085

4.085

4.13

4.205

4.22

4.235

4.255

4.54

4.565

4.58

4.69

4.735

4.795

5.155

Note that the first sequence of hourly electricity usage data displayed above is unsorted while the second sequence of hourly usage data shows a sorted sequence in ascending order. As a result of the sorting of the electricity usage data of 518 consecutive days, the identified range of electricity usage varies between a minimum of 0.225kW and a maximum of 5.155kW. This represents a wide spread of hourly electricity usage which implies that flagging of larger hourly electricity usage values would represent a meaningful option for advanced real-time streaming analysis as the hourly electricity usage values would come in.

Scenario 2: (1) Send 12,432 IoT Temperature Sensor Device Messages (Outside Temperature at Electricity Customer Premise) sourced from JSON file (A02SSH00302_TS.json); (2) Process and analyze 12,432 IoT Temperature Sensor Device Messages (Temperature in Fahrenheit) with IoT Device Message Processor and Analyzer (kafka-direct-iotmsg-temp-sensor.py)

The command line approach for scenario 2 in terms of terminal window configuration and command sequence for each terminal window is summarized in the following tables:

Terminal Window 1 (Working directory: /home/ec2-user/kafka_2.11-0.10.1.0)

Open terminal window 1 and perform the following sequence of commands to clean (stopping zookeeper and kafka) and start zookeeper and kafka:
 	
$ cd kafka_2.11-0.10.1.0
 	
$ bin/kafka-server-stop.sh
$ bin/zookeeper-server-stop.sh
$ pkill -9 java
$ pkill -9 python
 	
$ export KAFKA_HEAP_OPTS="-Xmx128M -Xms64M"
$ bin/zookeeper-server-start.sh config/zookeeper.properties > \
>/tmp/zookeeper.log 2>&1 & <ENTER>
 	
$ export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"
$ bin/kafka-server-start.sh config/server.properties > \
> /tmp/kafka.log 2>&1 & <ENTER>
 	
The result of executing the above sequence will be a clean restart of zookeeper and kafka.

Terminal Window 2 (Working directory: /home/ec2-user)

Open terminal window 2 and perform the following sequence of commands to stream IoT device message data using the Big Data processing file “kafka-direct-iotmsg-temp-sensor.py”:
 	
$ export KAFKA_HEAP_OPTS="-Xmx64M -Xms48M"
$ spark-submit \
> --jars spark-streaming-kafka-0-8-assembly_2.11-2.0.0-preview.jar \
> ./kafka-direct-iotmsg-temp-sensor.py localhost:9092 iotmsgs <ENTER>
 	
The result of executing the above sequence will be a continuous sequence of time stamps displayed in terminal window 2 until stopped with <CTRL C>. Terminal window 2 serves to display the data streaming and analysis results.

Terminal Window 3 (Working directory: /home/ec2-user)

Open terminal window 3 and perform the following sequence of commands to send actual IoT temperature sensor device message data stored in JSON file “A02SSH00302_TS.json”:
 	
$ cd ${HOME}
$ export KAFKA_HEAP_OPTS="-Xmx64M -Xms48M"
$ cat A02SSH00302_TS.json | ./kafka_2.11-0.10.1.0/bin/kafka-console-producer.sh \
> --broker-list localhost:9092 \
> --topic iotmsgs <ENTER>
 	
The result of executing the above sequence will be 12,432 actual IoT temperature sensor device messages sent by the IoT temperature sensor device message holder “A02SSH00302_TS.json” to be streamed, analyzed and displayed by the IoT device message data streaming and analyzer file “kafka-direct-iotmsg-temp-sensor.py” in terminal window 2.
 	
Note that the execution of the above command sequence in terminal window 3 completes by concluding with the normal prompt “$” when finished. The streaming of the IoT device messages is displayed in terminal window 2 which serves as the data streaming display.

The results of processing and analyzing the 12,432 IoT temperature sensor device messages (hourly outside temperature data at customer premise from 518 consecutive days) in terminal window 2 are displayed in the following:

Time: 2016-11-21 06:24:06

Time: 2016-11-21 06:24:08

"Temperature":54
"Temperature":55
"Temperature":55
"Temperature":53
"Temperature":53
"Temperature":54
"Temperature":56
"Temperature":58
"Temperature":60
"Temperature":61
"Temperature":64
"Temperature":63
"Temperature":63
"Temperature":64
"Temperature":63
"Temperature":62
"Temperature":61
"Temperature":59
"Temperature":58
"Temperature":58
"Temperature":58
"Temperature":58
"Temperature":58
"Temperature":58
"Temperature":58

"Temperature":58
"Temperature":57
"Temperature":56
"Temperature":54
"Temperature":55
"Temperature":55
"Temperature":56
"Temperature":56
"Temperature":59
"Temperature":59
"Temperature":58
"Temperature":60
"Temperature":62
"Temperature":61
"Temperature":61
"Temperature":61
"Temperature":60
"Temperature":59
"Temperature":58
"Temperature":58
"Temperature":58
"Temperature":57
"Temperature":57
"Temperature":57
"Temperature":57

"Temperature":57

……

"Temperature":61

"Temperature":65

"Temperature":66

"Temperature":67

"Temperature":66

"Temperature":61

"Temperature":55

"Temperature":54

"Temperature":54

"Temperature":53

"Temperature":51

Time: 2016-11-21 06:24:08

Time: 2016-11-21 06:24:08

35

37

38

38

38

39

39

39
39
40
40
40
40
40
40
40
40
40
40
41
41
41
41
41
41
41
41
41
41
41
41
41

42
42
42
42
42
42
42
42
42
42
42
42
42
42
42
42
42
42
42
43
43
43
43
43
43
43

43
43
43
43
43
43
43
43
43
43
43
43
43
43
43
44
44
44
44
44
44
44
44
44
….

89
89
89
90
90
90
90
90
91
91
91
91
91
91
91
91
91
92
92
93
93
94

Time: 2016-11-21 06:24:10

Note that the first sequence of hourly temperature data displayed above is unsorted while the second sequence of temperature data shows a sorted sequence in ascending order. Resulting from the sorting of the temperature data of 518 consecutive days, the identified range of outside temperature at the customer premise varies between a minimum of 35^0 Fahrenheit and a maximum of 94^0 Fahrenheit. This represents a wide spread in outside temperature at the customer premise which implies that flagging of smaller and larger temperature values would represent a meaningful option for advanced real-time streaming analysis as the hourly temperature values would come in and advanced correlation analysis related to the corresponding electricity usage would be applied.

Scenario 3: (1) Send 12,432 IoT Humidity Sensor Device Messages (Outside Humidity at Electricity Customer Premise) sourced from JSON file (A02SSH00302_HS.json); (2) Process and analyze 12,432 IoT Humidity Sensor Device Messages (Relative Humidity in %) with IoT Device Message Processor and Analyzer (kafka-direct-iotmsg-hum-sensor.py)

The command line approach for scenario 2 in terms of terminal window configuration and command sequence for each terminal window is summarized in the following tables:

Terminal Window 1 (Working directory: /home/ec2-user/kafka_2.11-0.10.1.0)

Open terminal window 1 and perform the following sequence of commands to clean (stopping zookeeper and kafka) and start zookeeper and kafka:
 	
$ cd kafka_2.11-0.10.1.0
 	
$ bin/kafka-server-stop.sh
$ bin/zookeeper-server-stop.sh
$ pkill -9 java
$ pkill -9 python
 	
$ export KAFKA_HEAP_OPTS="-Xmx128M -Xms64M"
$ bin/zookeeper-server-start.sh config/zookeeper.properties > \
>/tmp/zookeeper.log 2>&1 & <ENTER>
 	
$ export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"
$ bin/kafka-server-start.sh config/server.properties > \
> /tmp/kafka.log 2>&1 & <ENTER>
 	
The result of executing the above sequence will be a clean restart of zookeeper and kafka.

Terminal Window 2 (Working directory: /home/ec2-user)

Open terminal window 2 and perform the following sequence of commands to stream IoT device message data using the Big Data processing file “kafka-direct-iotmsg-hum-sensor.py”:
 	
$ export KAFKA_HEAP_OPTS="-Xmx64M -Xms48M"
$ spark-submit \
> --jars spark-streaming-kafka-0-8-assembly_2.11-2.0.0-preview.jar \
> ./kafka-direct-iotmsg-hum-sensor.py localhost:9092 iotmsgs <ENTER>
 	
The result of executing the above sequence will be a continuous sequence of time stamps displayed in terminal window 2 until stopped with <CTRL C>. Terminal window 2 serves to display the data streaming and analysis results.

Terminal Window 3 (Working directory: /home/ec2-user)

Open terminal window 3 and perform the following sequence of commands to send actual IoT humidity sensor device message data stored in JSON file “A02SSH00302_HS.json”:
 	
$ cd ${HOME}
$ export KAFKA_HEAP_OPTS="-Xmx64M -Xms48M"
$ cat A02SSH00302_HS.json | ./kafka_2.11-0.10.1.0/bin/kafka-console-producer.sh \
> --broker-list localhost:9092 \
> --topic iotmsgs <ENTER>
 	
The result of executing the above sequence will be 12,432 actual IoT humidity sensor device messages sent by the IoT humidity sensor device message holder “A02SSH00302_HS.json” to be streamed and displayed by the IoT device message data streaming and analyzer file “kafka-direct-iotmsg-hum-sensor.py” in terminal window 2.
 	
Note that the execution of the above command sequence in terminal window 3 completes by concluding with the normal prompt “$” when finished. The streaming of the IoT device messages is displayed in terminal window 2 which serves as the data streaming display.

The results of processing and analyzing the 12,432 IoT humidity sensor device messages (hourly outside humidity data at customer premise from 518 consecutive days) in terminal window 2 are displayed in the following:

Time: 2016-11-21 07:11:26

"Humidity":83

"Humidity":77

"Humidity":75

"Humidity":77

"Humidity":80
"Humidity":77
"Humidity":75
"Humidity":70
"Humidity":62
"Humidity":60
"Humidity":54
"Humidity":56
"Humidity":56
"Humidity":54
"Humidity":56
"Humidity":60
"Humidity":60
"Humidity":67
"Humidity":70
"Humidity":72
"Humidity":72
"Humidity":75
"Humidity":75
"Humidity":72
"Humidity":72
"Humidity":72
"Humidity":80
"Humidity":87
"Humidity":96

"Humidity":93
"Humidity":96
"Humidity":96
"Humidity":96
"Humidity":90
"Humidity":90
"Humidity":93
"Humidity":93
"Humidity":81
"Humidity":81
"Humidity":81
"Humidity":81
"Humidity":81
"Humidity":87
"Humidity":90
"Humidity":90
"Humidity":93
"Humidity":96
"Humidity":93
"Humidity":93
"Humidity":93
"Humidity":93
"Humidity":93
"Humidity":90
"Humidity":93

"Humidity":93
"Humidity":81
"Humidity":75
"Humidity":70
"Humidity":65
"Humidity":63
"Humidity":63
"Humidity":70
"Humidity":65
"Humidity":67
"Humidity":70
"Humidity":72
"Humidity":78
"Humidity":80
"Humidity":83
"Humidity":90
"Humidity":90
"Humidity":86
"Humidity":93
"Humidity":93
"Humidity":93
"Humidity":93
"Humidity":90
"Humidity":90
"Humidity":83

"Humidity":72

"Humidity":70

"Humidity":65

"Humidity":63

…

"Humidity":53

"Humidity":57

"Humidity":59

"Humidity":59

"Humidity":61

"Humidity":70

"Humidity":70

"Humidity":70

"Humidity":73

"Humidity":73

"Humidity":73

Time: 2016-11-21 07:11:26

Total Count of Msgs: 12432

Time: 2016-11-21 07:11:26

10

10

10
10
11
11
11
11
11
11
11
11
12
12
12
12
12
12
12
12
12
12
12
13
13
13
13

13
13
13
13
13
13
13
13
14
14
14
14
14
14
14
14
…
23
23
23
23
23
23
23
23

24

24

24

24

24

24

24

24

24

24

24

24

24

24

24

24

24

24

24

…

97

97

97

97

97

97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97

97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97

97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97

97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97
97

97
97
97
97
97
97
97
97
97
…
100
100
100
100
100
100
100
100
100
100
100
100
100
100
100

100

100

100

Note that the first sequence of hourly relative humidity data displayed above is unsorted while the second sequence of humidity data shows a sorted sequence in ascending order. As a result of the sorting of the hourly relative humidity data of 518 consecutive days, the identified range of outside relative humidity at the customer premise varies between a minimum of 10% and a maximum of 100%. This represents a wide spread in outside relative humidity at the customer premise which implies that flagging of smaller and larger humidity values might represent a meaningful option for advanced real-time streaming analysis as the hourly humidity values would come in and advanced correlation analysis related to the corresponding electricity usage could be applied.

IoT Device Message Data Processing and Analysis – Regression Model

In addition to real-time streaming analysis of time-series data as discussed in steps 1 and 2 (zookeeper and kafka), there is also the option to apply batch analysis to time series data in order to develop a regression model that describes the correlation between the incoming independent weather variables temperature and humidity and their impact on residential electricity usage as the dependent output variable. Figure 2 “Weather – Residentiial Electricity Usage Input – Output Model” shown in step 1 has illustrated the approach.

To do so, the data frame for a customer must be imported into R, or in this case SparkR so that the necessary regression model can be identified. There are several types of regression models which can be used to find the best possible model fit. The following figures illustrate a performance comparison between two regression models, i.e. linear and polynomial for one of the 2,494 customers included in the data set.

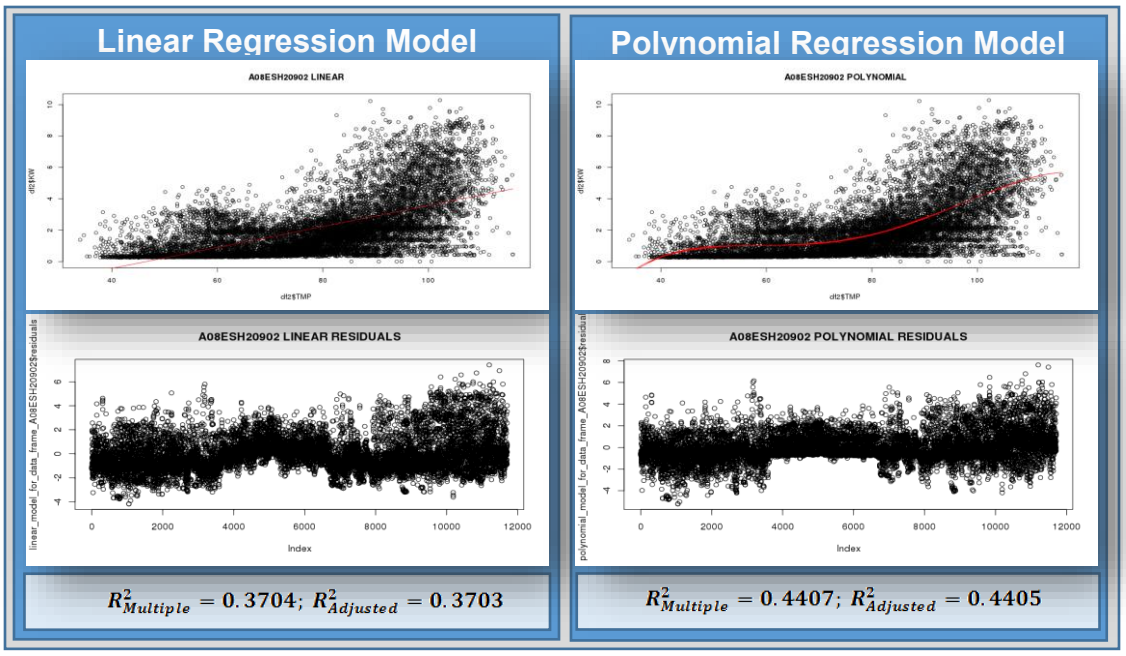

Figure 5: Linear and Polynomial Regression Modeling – Comparison for SCE Customer

The following figure 6 illustrates the detailed mathematical description for each of the above 2 regression models, i.e.

Linear Regression Model

$$Usage_{kW} = 0.0666825 \times Temp_F - 3.1035150$$

Regression Statistics - Coefficients

No.	Variable	Linear Coefficient	Likelihood (Probability) of Insignificance	Significance Code
1	Intercept	-3.1035150	< 2e-16 (very small)	***
2	$Temp_F$	0.0666825	< 2e-16 (very mall)	***

$$R^2_{Multiple} = 0.3704;\ R^2_{Adjusted} = 0.3703$$

Polynomial Regression Model

$$Usage_{kW} = -31.22 + 1.891 \times Temp_F - 0.04039 \times Temp_F^2 + 0.0003679 \times Temp_F^3 - 1.177 \times 10^{-6} \times Temp_F^4$$

Regression Statistics - Coefficients

No.	Variable	Linear Coefficient	Likelihood (Probability) of Insignificance	Significance Code
1	Intercept	-31.22	< 2e-16 (very small)	***
2	$Temp_F$	1.891	< 2e-16 (very small)	***
3	$Temp_F^2$	-0.04039	< 2e-16 (very small)	***
4	$Temp_F^3$	0.0003679	< 2e-16 (very small)	***
5	$Temp_F^4$	-1.177e-06	< 2e-16 (very small)	***

$$R^2_{Multiple} = 0.4407;\ R^2_{Adjusted} = 0.4405$$

Figure 6: Linear and Polynomial Regression Modeling – Regression Statistics for SCE Customer

Based on the above regression modeling, the following conclusions can be drawn:

- Linear multivariable regression is not well suited for modeling daily electricity load curves.
- Polynomial multivariable regression does not make for sufficient model fit either to describe the daily electricity load behavior.
- The quality of fit of linear and polynomial regression models for describing the behavior of electricity loads very much depends on the specific customer.
- Electricity usage has more significant impact from temperature than from humidity. Note: Temperature and humidity are not completely independent input variable.
- To get a better reflection of dynamic customer behavior in electricity usage, discrete time series models must be applied.

IoT Device Message Data Processing and Analysis – Summary

Whether streaming or batch analytics applied to time series data, both approaches have their benefits and are needed. While streaming analytics are normally more basic in mathematical nature (e.g. sorting, filtering, counting, identification of simple statistical parameters), they are more challenging in terms of developing the necessary software programs to run the needed analysis to achieve real-time results. Batch analytics on the other hand are more advanced in terms of the mathematics applied (e.g. regression modeling) while the software tools used to do so (e.g. R) are more user-friendly and, therefore, less demanding to deploy. As this report has shown, a good approach is to run data processing and analysis in real-time using zookeeper and kafka first (for streaming analytics to apply flagging or actionable triggers), and then apply more advanced data science tools such as R afterwards when time-series data batches are available to find more advanced regression models that describe the systems behavior inherent in the input and output time-series data collected.