Implement a MapReduce application using Hadoop to obtain, programmatically (using Python or your preferred language), the following:

**Q) What is number of open, assigned and closed issues for the following 3 complaint types:**

**Python Script: complaint\_count.py**

1. All noise complaints (See the column “Complaint Type”)
2. Street Condition
3. Illegal Parking

**Final Script Output:** "Count of Noise\_Complaints": 1771120

"Count of Illegal Parking Complaints": 561212

"Count of Street Condition Complaints": 434132

**Sample Script Output:** "Count of Street Condition Complaints": 16

**(1000 Records)** "Count of Noise\_Complaints": 766

"Count of Illegal Parking Complaints": 18

**Q) For the year 2017, for each quarter (Jan-Mar, April-June, July-Sept, Oct-Dec)**

1. **What is the average number of the above complaints per month?**

**Python Script: average\_count.py**

**Final Script Output:**

"Average number of noise complaints per month in Q1 (Jan - Mar)": 27964

"Average number of street complaints per month in Q1 (Jan - Mar)": 8581

"Average number of illegal parking complaints per month in Q1 (Jan - Mar)": 10772

"Average number of noise complaints per month in Q2 (Apr - June)":  43789

"Average number of street complaints per month in Q2 (Apr - June)": 10704

"Average number of illegal parking complaints per month in Q2 (Apr - June)": 12971

"Average number of noise complaints per month in Q3 (July - Sept)":  43990

"Average number of street complaints per month in Q3 (July - Sept)": 6590

"Average number of illegal parking complaints per month in Q3 (July - Sept)": 12784

"Average number of noise complaints per month in Q4 (Oct - Dec)": 33286

"Average number of street complaints per month in Q4 (Oct - Dec)": 5212

"Average number of illegal parking complaints per month in Q4 (Oct - Dec)" : 12179

**Sample Script Output (1000 Records):**

"Average number of noise complaints per month in Q1 (Jan - Mar)": 255

"Average number of street complaints per month in Q1 (Jan - Mar)": 5

"Average number of illegal parking complaints per month in Q1 (Jan - Mar)": 6

"Average number of noise complaints per month in Q2 (Apr - June)": 0

"Average number of street complaints per month in Q2 (Apr - June)": 0

"Average number of illegal parking complaints per month in Q2 (Apr - June)": 0

"Average number of street complaints per month in Q3 (July - Sept)": 0

"Average number of noise complaints per month in Q3 (July - Sept)": 0

"Average number of illegal parking complaints per month in Q3 (July - Sept)": 0

"Average number of street complaints per month in Q4 (Oct - Dec)": 0

"Average number of noise complaints per month in Q4 (Oct - Dec)": 0

"Average number of illegal parking complaints per month in Q4 (Oct - Dec)": 0

1. **Which borough had the highest number of” Illegal Parking” in each quarter?**

**Python Script: illegal\_parking\_count.py**

**Final Script Output:**

"Highest number of illegal parking borough in Q1 (Jan-Mar)": ["BROOKLYN"]

"Highest number of illegal parking borough in Q2 (Apr-June)": ["BROOKLYN"]

"Highest number of illegal parking borough in Q3 (July-Sept)": ["BROOKLYN"]

"Highest number of illegal parking borough in Q4 (Oct-Dec)": ["BROOKLYN"]

**Sample Script Output (1000 Records):**

“Highest number of illegal parking borough in Q1 (Jan-Mar)”: [“QUEENS”]

“Highest number of illegal parking borough in Q2 (Apr-June)”: [“None”]

“Highest number of illegal parking borough in Q3 (July-Sept)”: [“None”]

“Highest number of illegal parking borough in Q4 (Oct-Dec)”: [“None”]

**Q) Provide the code that you have developed to run the proposed problem in Hadoop/HDFS.**

|  |
| --- |
| **complaint\_count.py** |
| **## This script attempts to find the number of open, assigned and closed issues for the following 3 complaint types:**  **## 1. All noise complaints**  **## 2. Street Condition**  **## 3. Illegal Parking**  from mrjob.job import MRJob  class MR311ComplaintCount(MRJob):    def mapper(self, \_, line):  attr\_list = line.split(",")    count\_noise\_complaints = 0 ## Initialize Variables  count\_street\_complaints = 0  count\_parking\_complaints = 0    if (attr\_list[0] == "Unique Key" or attr\_list[1] == "Created Date"): ## Ignore file header  pass  elif 'Noise' in attr\_list[5]: ## Checking complaint type  count\_noise\_complaints += 1  elif 'Street Condition' in attr\_list[5]:  count\_street\_complaints += 1  elif 'Illegal Parking' in attr\_list[5]:  count\_parking\_complaints += 1  else:  pass    yield "Count of Noise\_Complaints", count\_noise\_complaints  yield "Count of Street Condition Complaints", count\_street\_complaints  yield "Count of Illegal Parking Complaints", count\_parking\_complaints    def reducer(self, key, values):  yield key, sum(values)      if \_\_name\_\_ == '\_\_main\_\_':  MR311ComplaintCount.run() |

|  |
| --- |
| **average\_count.py** |
| **## This script attempts to find the average number of different complaint types for each quarter for a given year**  from mrjob.job import MRJob  from datetime import datetime  class MR311ServiceRequests(MRJob):    def mapper(self, \_, line):  q1\_noise = 0 ## Initialize Variables  q1\_parking = 0  q1\_street = 0  q2\_noise = 0  q2\_parking = 0  q2\_street = 0  q3\_noise = 0  q3\_parking = 0  q3\_street = 0  q4\_noise = 0  q4\_parking = 0  q4\_street = 0      attr\_list = line.split(",") ## Splitting each line of the file    formatted\_time = datetime    if attr\_list[1] != "Created Date": ## Formatting created date attribute  attr\_list[1] = attr\_list[1][:-2].strip()  formatted\_time = datetime.strptime(attr\_list[1], "%m/%d/%Y %H:%M:%S")  else:  pass    if (formatted\_time.year == 2017): ## Checking year  if (formatted\_time.month == 1) or (formatted\_time.month == 2) or (formatted\_time.month == 3):  if 'Noise' in attr\_list[5]:  q1\_noise += 1  elif 'Illegal Parking' in attr\_list[5]:  q1\_parking += 1  elif 'Street Condition' in attr\_list[5]:  q1\_street += 1    elif (formatted\_time.month == 4) or (formatted\_time.month == 5) or (formatted\_time.month == 6):  if 'Noise' in attr\_list[5]:  q2\_noise += 1  elif 'Illegal Parking' in attr\_list[5]:  q2\_parking += 1  elif 'Street Condition' in attr\_list[5]:  q2\_street += 1    elif (formatted\_time.month == 7) or (formatted\_time.month == 8) or (formatted\_time.month == 9):  if 'Noise' in attr\_list[5]:  q3\_noise += 1  elif 'Illegal Parking' in attr\_list[5]:  q3\_parking += 1  elif 'Street Condition' in attr\_list[5]:  q3\_street += 1    elif (formatted\_time.month == 10) or (formatted\_time.month == 11) or (formatted\_time.month == 12):  if 'Noise' in attr\_list[5]:  q4\_noise += 1  elif 'Illegal Parking' in attr\_list[5]:  q4\_parking += 1  elif 'Street Condition' in attr\_list[5]:  q4\_street += 1    else:  pass    else:  pass    yield "Average number of noise complaints per month in Q1 (Jan - Mar)", q1\_noise  yield "Average number of illegal parking complaints per month in Q1 (Jan - Mar)", q1\_parking  yield "Average number of street complaints per month in Q1 (Jan - Mar)", q1\_street    yield "Average number of noise complaints per month in Q2 (Apr - June)", q2\_noise  yield "Average number of illegal parking complaints per month in Q2 (Apr - June)", q2\_parking  yield "Average number of street complaints per month in Q2 (Apr - June)", q2\_street    yield "Average number of noise complaints per month in Q3 (July - Sept)", q3\_noise  yield "Average number of illegal parking complaints per month in Q3 (July - Sept)", q3\_parking  yield "Average number of street complaints per month in Q3 (July - Sept)", q3\_street    yield "Average number of noise complaints per month in Q4 (Oct - Dec)", q4\_noise  yield "Average number of illegal parking complaints per month in Q4 (Oct - Dec)", q4\_parking  yield "Average number of street complaints per month in Q4 (Oct - Dec)", q4\_street      def reducer(self, key, values):  yield key, round(sum(values)/3) ## Finding average per month    if \_\_name\_\_ == '\_\_main\_\_':  MR311ServiceRequests.run() |

|  |
| --- |
| **illegal\_parking\_count.py** |
| **## This script attempts to find the number borough with the highest number of illegal parking each quarter for a given year**  from mrjob.job import MRJob  from datetime import datetime  class MR311ServiceRequests(MRJob):    def mapper(self, \_, line):  q1\_parking\_borough = [] ## Initializing empty lists for holding parking borough names  q2\_parking\_borough = []  q3\_parking\_borough = []  q4\_parking\_borough = []    attr\_list = line.split(",") ## Splitting each line of the input file    formatted\_time = datetime    if attr\_list[1] != "Created Date": ## Formatting created date attribute  attr\_list[1] = attr\_list[1][:-2].strip()  formatted\_time = datetime.strptime(attr\_list[1], "%m/%d/%Y %H:%M:%S")  else:  pass    if (formatted\_time.year == 2017): ## Checking year  if (formatted\_time.month == 1) or (formatted\_time.month == 2) or (formatted\_time.month == 3):  if 'Illegal Parking' in attr\_list[5]: ## Checking Parking Borough  q1\_parking\_borough.append(attr\_list[30])    elif (formatted\_time.month == 4) or (formatted\_time.month == 5) or (formatted\_time.month == 6):  if 'Illegal Parking' in attr\_list[5]:  q2\_parking\_borough.append(attr\_list[30])    elif (formatted\_time.month == 7) or (formatted\_time.month == 8) or (formatted\_time.month == 9):  if 'Illegal Parking' in attr\_list[5]:  q3\_parking\_borough.append(attr\_list[30])    elif (formatted\_time.month == 10) or (formatted\_time.month == 11) or (formatted\_time.month == 12):  if 'Illegal Parking' in attr\_list[5]:  q4\_parking\_borough.append(attr\_list[30])  else:  pass    else:  pass    yield("Highest number of illegal parking borough in Q1 (Jan-Mar)",q1\_parking\_borough)  yield("Highest number of illegal parking borough in Q2 (Apr-June)",q2\_parking\_borough)  yield("Highest number of illegal parking borough in Q3 (July-Sept)",q3\_parking\_borough)  yield("Highest number of illegal parking borough in Q4 (Oct-Dec)",q4\_parking\_borough)    def reducer(self, key, values):  try:  values = [x for x in values if x] ## Checking for empty values the list and removing them  yield key, max(values, key=values.count) ## Picking most frequent value in the list  except ValueError: ## Compatible code for Python 2.7  values = ["None"]  yield key, values    #yield key, max(values, key=values.count, default = "None")    if \_\_name\_\_ == '\_\_main\_\_':  MR311ServiceRequests.run() |

**Q) Provide the results obtained (statistics e.g., using plots)**

**Note:** Noise Complaints includes aggregate count of the following Sub types:

**“Noise | Noise - Street/Sidewalk | Noise – Vehicle | Noise – Residential | Noise – Commercial | Noise – Helicopter “**

**Note:**

Q1 – Quarter 1 (Jan – Mar), Q2 – Quarter 2 (Apr – June), Q3 – Quarter 3 (July – Sep), Q4 – Quarter 4 (Oct – Dec)

|  |  |
| --- | --- |
| **Quarter Details (Year 2017)** | **Highest Illegal Parking Borough Name** |
| Q1 (Jan – Mar) | Brooklyn |
| Q2 (Apr – June) | Brooklyn |
| Q3 (July – Sep) | Brooklyn |
| Q4 (Oct – Dec) | Brooklyn |

**Q) Specific question: describe the order in which map and reduce tasks are executed in Hadoop and the duration of map and reduce phases for the different job execution steps**.

|  |  |  |
| --- | --- | --- |
| **Script Name: complaint\_count.py** | **Script Name: average\_count.py** | **Script Name: illegal\_parking\_count.py** |
| **Execution Step:**  **python complaint\_count.py s3://pgarias-bucket-cloud.s3.us-east-2.amazonaws.com/311\_Service\_Requests\_from\_2015\_to\_Present.csv -r emr –output-dir=s3://ranjeeta-hw2-us-east-2/final\_1**  **Log File:**  **aws s3 ls s3://ranjeeta-hw2-us-east-2/final\_1** | **Execution Step:**  **python average\_count.py s3://pgarias-bucket-cloud.s3.us-east-2.amazonaws.com/311\_Service\_Requests\_from\_2015\_to\_Present.csv -r emr –output-dir=s3://ranjeeta-hw2-us-east-2/final\_2**  **Log File:**  **aws s3 ls s3://ranjeeta-hw2-us-east-2/final\_2** | **Execution Step:**  **python illegal\_parking\_count.py s3://pgarias-bucket-cloud.s3.us-east-2.amazonaws.com/311\_Service\_Requests\_from\_2015\_to\_Present.csv -r emr –output-dir=s3://ranjeeta-hw2-us-east-2/final\_3**  **Log File:**  **aws s3 ls s3://ranjeeta-hw2-us-east-2/final\_3** |
| **Order of Map Reduce tasks:**  **For every complaint type, count of number of records is calculated in the mapper. Reducer is simply used to print out the count.** | **Order of Map Reduce tasks:**  **For every complaint type, in the mapper corresponding year, month is checked, and count of records is calculated. Reducer is used to sum the counts and display the average per quarter.** | **Order of Map Reduce tasks:**  **For illegal parking complaint type, in the mapper corresponding year, month is checked, and parking borough is determined. Reducer is used to determine and display the name of the most frequently occurring borough.** |
| **Duration:**  **Data-local map tasks=89**  **Killed map tasks=3**  **Killed reduce tasks=1**  **Launched map tasks=89**  **Launched reduce tasks=10**  **Total megabyte-milliseconds taken by all map tasks=5882479104**  **Total megabyte-milliseconds taken by all reduce tasks=4156360704**  **Total time spent by all map tasks (ms)=3829739**  **Total time spent by all maps in occupied slots (ms)=183827472**  **Total time spent by all reduce tasks (ms)=1352982**  **Total time spent by all reduces in occupied slots (ms)=129886272**  **Total vcore-milliseconds taken by all map tasks=3829739**  **Total vcore-milliseconds taken by all reduce tasks=1352982**  **CPU time spent (ms)=1231710**  **Combine input records=0**  **Combine output records=0**  **Failed Shuffles=0**  **GC time elapsed (ms)=78471**  **Input split bytes=10947**  **Map input records=10420595**  **Map output bytes=1156686045**  **Map output materialized bytes=63417476**  **Map output records=31261785**  **Merged Map outputs=801**  **Physical memory (bytes) snapshot=61739741184**  **Reduce input groups=3**  **Reduce input records=31261785**  **Reduce output records=3**  **Reduce shuffle bytes=63417476**  **Shuffled Maps =801**  **Spilled Records=62523570**  **Total committed heap usage (bytes)=55600218112**  **Virtual memory (bytes) snapshot=336187707392** | **Duration:**  **Data-local map tasks=89**  **Failed map tasks=1**  **Killed map tasks=1**  **Killed reduce tasks=2**  **Launched map tasks=90**  **Launched reduce tasks=10**  **Other local map tasks=1**  **Total megabyte-milliseconds taken by all map tasks=12784375296**  **Total megabyte-milliseconds taken by all reduce tasks=10375268352**  **Total time spent by all map tasks (ms)=8323161**  **Total time spent by all maps in occupied slots (ms)=399511728**  **Total time spent by all reduce tasks (ms)=3377366**  **Total time spent by all reduces in occupied slots (ms)=324227136**  **Total vcore-milliseconds taken by all map tasks=8323161**  **Total vcore-milliseconds taken by all reduce tasks=3377366**  **CPU time spent (ms)=4704540**  **Combine input records=0**  **Combine output records=0**  **Failed Shuffles=0**  **GC time elapsed (ms)=81530**  **Input split bytes=10947**  **Map input records=10420595**  **Map output bytes=8930449915**  **Map output materialized bytes=454214416**  **Map output records=125047140**  **Merged Map outputs=801**  **Physical memory (bytes) snapshot=66615840768**  **Reduce input groups=12**  **Reduce input records=125047140**  **Reduce output records=12**  **Reduce shuffle bytes=454214416**  **Shuffled Maps =801**  **Spilled Records=250094280**  **Total committed heap usage (bytes)=60947431424**  **Virtual memory (bytes) snapshot=336211337216** | **Duration:**  **Data-local map tasks=89**  **Killed map tasks=1**  **Launched map tasks=89**  **Launched reduce tasks=9**  **Total megabyte-milliseconds taken by all map tasks=7794066432**  **Total megabyte-milliseconds taken by all reduce tasks=5608461312**  **Total time spent by all map tasks (ms)=5074262**  **Total time spent by all maps in occupied slots (ms)=243564576**  **Total time spent by all reduce tasks (ms)=1825671**  **Total time spent by all reduces in occupied slots (ms)=175264416**  **Total vcore-milliseconds taken by all map tasks=5074262**  **Total vcore-milliseconds taken by all reduce tasks=1825671**  **CPU time spent (ms)=2262570**  **Combine input records=0**  **Combine output records=0**  **Failed Shuffles=0**  **GC time elapsed (ms)=85484**  **Input split bytes=10947**  **Map input records=10420595**  **Map output bytes=2658633888**  **Map output materialized bytes=135263048**  **Map output records=41682380**  **Merged Map outputs=801**  **Physical memory (bytes) snapshot=65759744000**  **Reduce input groups=4**  **Reduce input records=41682380**  **Reduce output records=4**  **Reduce shuffle bytes=135263048**  **Shuffled Maps =801**  **Spilled Records=83364760**  **Total committed heap usage (bytes)=59907244032**  **Virtual memory (bytes) snapshot=336150122496** |

**Appendix**

|  |  |
| --- | --- |
| **Sample File (Extract)** | **Final File** |
| <https://pgarias-bucket-cloud.s3.us-east-2.amazonaws.com/311_Service_Requests_from_2015_to_Present_head_1000.csv> | <https://pgarias-bucket-cloud.s3.us-east-2.amazonaws.com/311_Service_Requests_from_2015_to_Present.csv> |

**AWS S3 Output Directory:** **s3://ranjeeta-hw2-us-east-2**

|  |
| --- |
| **.mrjob.conf** |
| emr:  aws\_access\_key\_id: AKIAY7FBTDUDJIOVCK6B  aws\_secret\_access\_key: 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  region: us-east-2  subnet: subnet-7c685114  ec2\_key\_pair: hw2\_ids  ec2\_key\_pair\_file: /Users/ranjeeta/.ssh/hw2\_ids.pem  ssh\_tunnel: true  instance\_type: m4.large  master\_instance\_type: m4.large  num\_core\_instances: 5  interpreter: python2.7  bootstrap:  - sudo yum install -y gcc-c++  - sudo pip-2.7 install mrjob |