



Advanced Methods in Data Science and Big Data Analytics Lab Guide

Copyright

Copyright ©2015 EMC Corporation. All Rights Reserved. Published in the USA. EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license. The trademarks, logos, and service marks (collectively "Trademarks") appearing in this publication are the property of EMC Corporation and other parties. Nothing contained in this publication should be construed as granting any license or right to use any Trademark without the prior written permission of the party that owns the Trademark.

EMC, EMC² AccessAnywhere Access Logix, AdvantEdge, AlphaStor, AppSync ApplicationXtender, ArchiveXtender, Atmos, Authentica, Authentic Problems, Automated Resource Manager, AutoStart, AutoSwap, AVALONidm, Avamar, Bus-Tech, Captiva, Catalog Solution, C-Clip, Celerra, Celerra Replicator, Centera, CenterStage, CentraStar, EMC CertTracker, CIO Connect, ClaimPack, ClaimsEditor, Claralert, cLARIiON, ClientPak, CloudArray, Codebook Correlation Technology, Common Information Model, Compuset, Compute Anywhere, Configuration Intelligence, Configuresoft, Connectrix, Constellation Computing, EMC ControlCenter, CopyCross, CopyPoint, CX, DataBridge, Data Protection Suite, Data Protection Advisor, DBClassify, DD Boost, Dantz, DatabaseXtender, Data Domain, Direct Matrix Architecture, DiskXtender, DiskXtender 2000, DLS ECO, Document Sciences, Documentum, DR Anywhere, ECS, eInput, E-Lab, Elastic Cloud Storage, EmailXaminer, EmailXtender, EMC Centera, EMC ControlCenter, EMC LifeLine, EMCTV, Enginuity, EPFM, eRoom, Event Explorer, FAST, FarPoint, FirstPass, FLARE, FormWare, Geosynchrony, Global File Virtualization, Graphic Visualization, Greenplum, HighRoad, HomeBase, Illuminator, InfoArchive, InfoMover, Infoscapes, Infra, InputAccel, InputAccel Express, Invista, Ionix, ISIS, Kazeon, EMC LifeLine, Mainframe Appliance for Storage, Mainframe Data Library, Max Retriever, MCx, MediaStor, Metro, MetroPoint, MirrorView, Multi-Band Deduplication, Navisphere, Netstorage, NetWorker, nLayers, EMC OnCourse, OnAlert, OpenScale, Petrocloud, PixTools, Powerlink, PowerPath, PowerSnap, ProSphere, ProtectEverywhere, ProtectPoint, EMC Proven, EMC Proven Professional, QuickScan, RAPIDPath, EMC RecoverPoint, Rainfinity, RepliCare, RepliStor, ResourcePak, Retrospect, RSA, the RSA logo, SafeLine, SAN Advisor, SAN Copy, SAN Manager, ScaleIO Smarts, EMC Snap, SnapImage, SnapSure, SnapView, SourceOne, SRDF, EMC Storage Administrator, StorageScope, SupportMate, SymmAPI, SymmEnabler, Symmetrix, Symmetrix DMX, Symmetrix VMAX, TimeFinder, TwinStrata, UltraFlex, UltraPoint, UltraScale, Unisphere, Universal Data Consistency, Vblock, Velocity, Viewlets, ViPR, Virtual Matrix, Virtual Matrix Architecture, Virtual Provisioning, Virtualize Everything, Compromise Nothing, Virtuent, VMAX, VMAXe, VNX, VNXe, Voyence, VPLEX, VSAM-Assist, VSAM I/O PLUS, VSET, VSPEX, Watch4net, WebXtender, xPression, xPresso, Xtrem, XtremCache, XtremSF, XtremSW, XtremIO, YottaYotta, Zero-Friction Enterprise Storage.

Revision Date: April 2015
Revision Number: MR-1CP-ETAAMUSD 1.0

Document Revision History

Rev #	File Name	Date
1.0	Specialist_Lab_Guide.docx	2/25/15

Table of Contents

COPYRIGHT	2
DOCUMENT REVISION HISTORY	3
LAB ENVIRONMENT OVERVIEW	7
INSTALLED SOFTWARE AND TOOLS.....	7
LOCAL DIRECTORY STRUCTURE	8
HDFS DIRECTORY STRUCTURE.....	9
LAB EXERCISE 1: EXECUTING A HADOOP JOB	11
WORKFLOW OVERVIEW	12
LAB INSTRUCTIONS	13
LAB EXERCISE 2: WORKING WITH HDFS.....	21
WORKFLOW OVERVIEW	22
LAB INSTRUCTIONS	23
LAB EXERCISE 3: EXECUTING MAPREDUCE JOBS WITH APACHE PIG	27
WORKFLOW OVERVIEW	28
LAB INSTRUCTIONS	29
LAB EXERCISE 4: EXECUTING MAPREDUCE JOBS WITH APACHE HIVE	35
WORKFLOW OVERVIEW	36
LAB INSTRUCTIONS	37
LAB EXERCISE 5: STORING AND ACCESSING DATA IN APACHE HBASE	43
WORKFLOW OVERVIEW	44
LAB INSTRUCTIONS	45
LAB EXERCISE 6: SPARK	53
SPARK - WORKFLOW OVERVIEW	54
LAB INSTRUCTIONS	55
LAB EXERCISE 7: BASIC TEXT PROCESSING WITH NLTK	57
WORKFLOW OVERVIEW	58
LAB INSTRUCTIONS	59
LAB EXERCISE 8: TFIDF.....	65
WORKFLOW OVERVIEW	66
LAB INSTRUCTIONS	67
LAB EXERCISE 9: PART-OF-SPEECH TAGGING	73
WORKFLOW OVERVIEW	74
LAB INSTRUCTIONS	75

LAB EXERCISE 10: VISUALIZING SOCIAL NETWORK DATA	83
Workflow Overview	84
Lab Instructions	85
LAB EXERCISE 11: ANALYZING THE HUBWAY DATA	95
Workflow Overview	96
Lab Instructions	97
LAB EXERCISE 12: SIMULATION	105
Workflow Overview	106
Lab Instructions	107
LAB EXERCISE 13: RANDOM FOREST	113
Random Forests - Workflow Overview	114
Lab Instructions	115
FINAL LAB EXERCISE ON BIG DATA ANALYTICS	123
Case Study Background and Problem Definition	124
Suggested Workflow	127

This PAGE intentionally left blank.

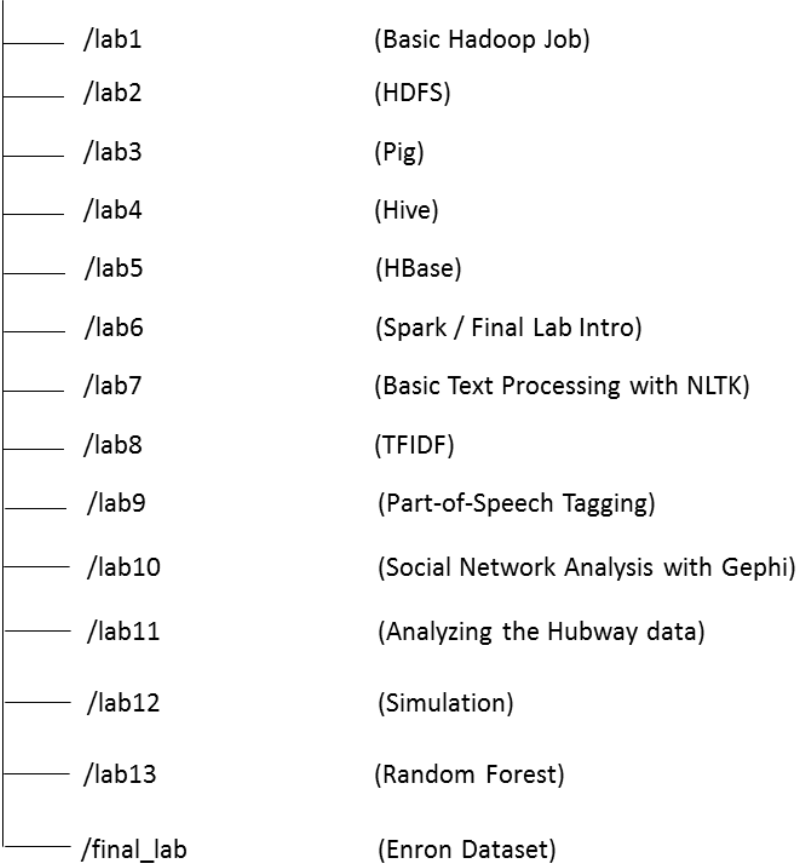
Lab Environment Overview

Installed Software and Tools

Software	Link
Apache Ant	http://ant.apache.org/
Apache Hadoop	http://hadoop.apache.org/
Apache HBase	http://hbase.apache.org/
Apache Hive	https://hive.apache.org/
Apache Mahout	https://mahout.apache.org/
Apache Maven	http://maven.apache.org/
Apache Pig	http://pig.apache.org/
Apache Spark	https://spark.apache.org/
Apache Sqoop	http://sqoop.apache.org/
Eclipse	https://www.eclipse.org/
Gephi	https://gephi.github.io/
Greenplum Database	http://www.pivotal.io/big-data/pivotal-greenplum-database
Java - openjdk	https://www.java.com/en/
Python	https://www.python.org/
R	http://www.r-project.org/
RStudio Server	http://www.rstudio.com/

Local Directory Structure

/home/gpadmin/Labs

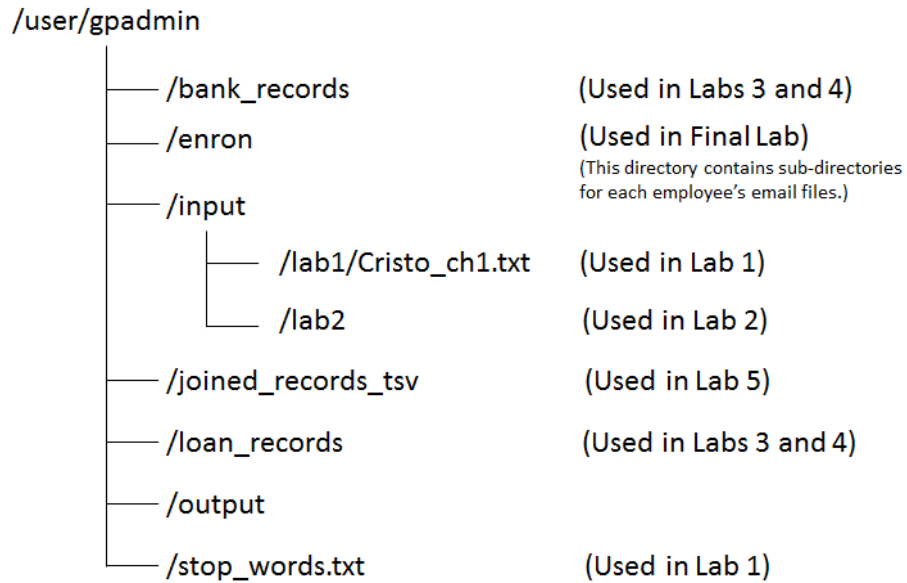


/lab1	(Basic Hadoop Job)
/lab2	(HDFS)
/lab3	(Pig)
/lab4	(Hive)
/lab5	(HBase)
/lab6	(Spark / Final Lab Intro)
/lab7	(Basic Text Processing with NLTK)
/lab8	(TFIDF)
/lab9	(Part-of-Speech Tagging)
/lab10	(Social Network Analysis with Gephi)
/lab11	(Analyzing the Hubway data)
/lab12	(Simulation)
/lab13	(Random Forest)
/final_lab	(Enron Dataset)

The directory, **/home/gpadmin/Solutions**, has sub-directories corresponding to each lab directory above. These sub-directories contain scripts and source code that may be used to aid the completion of each lab.

HDFS Directory Structure

This is the state of the environment's Hadoop Distributed File System (HDFS) before any labs are completed. Many of the empty directories will be populated during completion of the labs.



This PAGE intentionally left blank.

Lab Exercise 1: Executing a Hadoop Job

Purpose:	<p>This lab introduces the tools used to create and run a Hadoop job. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Use Eclipse to package the MapReduce Java code• Submit a Hadoop job and view its output.• Utilize the provided user interfaces to examine the status of the Hadoop cluster and the submitted jobs
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• View and edit Java code using Eclipse• Submit a Hadoop job to perform a word count• Modify the provided word count job<ul style="list-style-type: none">○ Add a combiner○ Specify the number of reducers○ Change the output filename
References:	

Workflow Overview

- 1 • Open Eclipse and Import the WordCount Project
- 2 • Review the WordCount Java Code
- 3 • Package the Code Into a JAR File
- 4 • Submit the Hadoop Job
- 5 • View the Output
- 6 • Exclude Certain Words from the Word Count
- 7 • Use a Combiner Function
- 8 • Specify the Number of Reduce Tasks
- 9 • Change the Output Filename
- 10 • Run the New Hadoop Job and View Output
- 11 • View the Hadoop ResourceManager UI
- 12 • View the NameNode UI

Lab Instructions

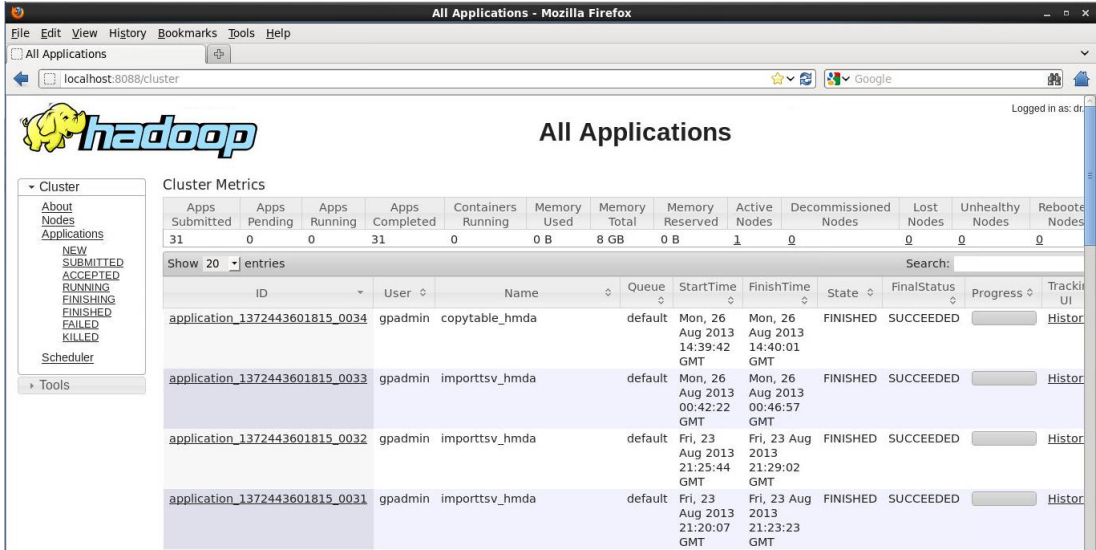
Step	Action
1	<p><u>Open Eclipse and Import the Provided WordCount Project</u></p> <ol style="list-style-type: none"> Open Eclipse from the desktop Go to File -> Import... Under the General folder, select Existing Projects into Workspace, and click Next. Using the Browse button for the root directory, select ~/gpadmin/Labs/lab1/WordCount and click OK. Click Finish on the Import window <p>Now the WordCount project can be viewed and edited through Eclipse.</p> <p>Note: The final Java files are located at: ~/Solutions/lab1.</p>
2	<p><u>Review the WordCount Java Code</u></p> <p>The WordCount project is already coded to count the number of occurrences of each word in any input TXT file. The project includes a Mapper class (<i>WordCountMapper.java</i>), a Reducer class (<i>WordCountReducer.java</i>) and a driver class (<i>WordCount.java</i>). To view these files in Eclipse using the Package Explorer:</p> <ol style="list-style-type: none"> Double click on the WordCount project Double click on src Double click on (default package) Double click on WordCount.java or any of the other Java files Look over the files and analyze the design of the word count code. <p>Several portions of the code are commented out. These sections will be modified later in this exercise. One modification enables the use of the MyTextOutputFormat.java file that also appears in this project.</p>
3	<p><u>Package the Code Into a JAR File</u></p> <p>In order to be run as a Hadoop job, the project must first be packaged as a JAR file:</p> <ol style="list-style-type: none"> In Eclipse's Package Explorer, right-click on the project's src folder and click on Export... Under the Java folder, select JAR File, and click Next. Under the heading Select the export destination, click Browse. In the new window, go to the directory ~/gpadmin/Labs/lab1, enter wc.jar as the name for the JAR file at the top of the window, and click OK. Click Finish on the JAR export window.

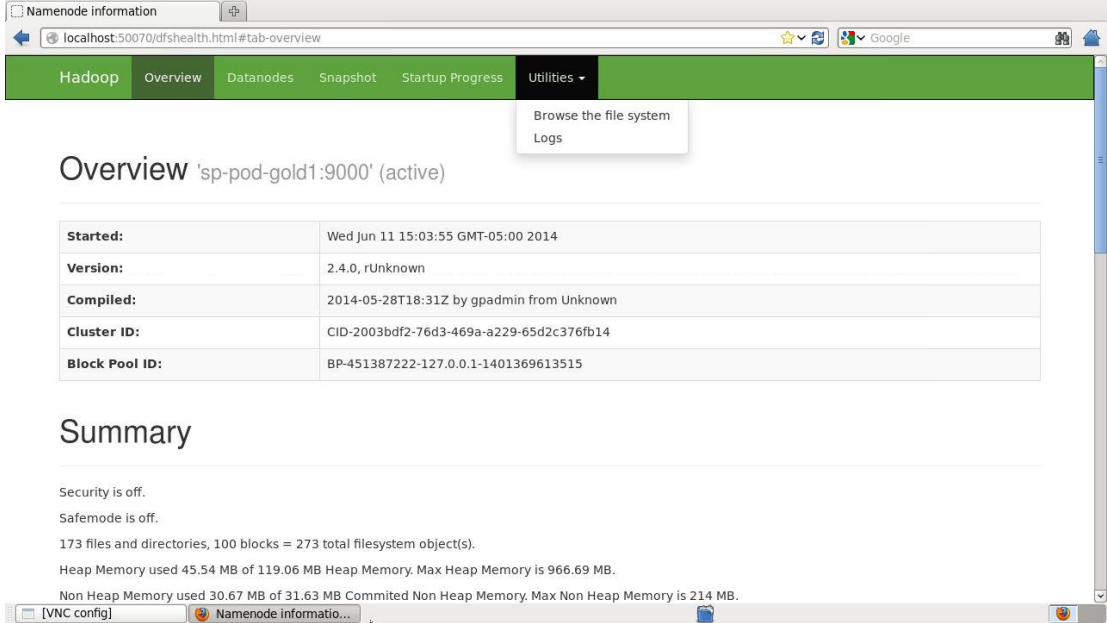
Step	Action
4	<p><u>Submit the Hadoop Job</u></p> <p>The following commands will verify that the HDFS directory structure is properly defined and then submit the job:</p> <ol style="list-style-type: none"> Open a Terminal from the desktop. To display the top of the HDFS directory run: <code>hdfs dfs -ls</code> Verify that the input and output HDFS directories exist. Verify that <i>Cristo_ch1.txt</i> exists under the input/lab1 directory <code>hdfs dfs -ls input/lab1</code> Verify that output/WCoutput does <u>not</u> exist: <code>hdfs dfs -ls output</code> Change the current directory to the location of the wc.jar file <code>cd ~/Labs/lab1</code> To submit the job, enter: <code>hadoop jar wc.jar WordCount input/lab1 output/WCoutput</code> <p>The general command for submitting a Hadoop job is:</p> <pre>\$ hadoop jar <JAR file> <driver class> <HDFS input directory> <HDFS output directory></pre> <p>To prevent errors, it is important that the specified HDFS output directory does not already exist.</p> <p>Hadoop will process all files in the provided input directory unless the filename begins with the underscore character (_).</p>
5	<p><u>View the Output</u></p> <p>Review the displayed results of the job's execution in the terminal. How many map tasks and reduce tasks were used by the job?</p> <p>Map tasks: _____ Reduce tasks: _____</p> <p>The reduce task's output is stored in a file named part-r-00000 within the specified output directory. Use the HDFS <code>-cat</code> command to view the output:</p> <pre>hdfs dfs -cat output/WCoutput/part-r-00000 more</pre> <p>How many times did the word "inquired" appear? _____</p>

Step	Action
6	<p data-bbox="293 283 837 315"><u>Exclude Certain Words from the Word Count</u></p> <p data-bbox="293 352 1404 491">The WordCount code will be modified to ignore certain stop words such as <i>it, I, a, what</i>, etc. A list of stop words is located in the HDFS location <code>/user/gpadmin/stop_words.txt</code>. Hadoop's distributed cache system allows the user to store any file on all of the data nodes at the start of the job so that the map tasks can have quick access to the file.</p> <p data-bbox="293 531 1409 598">The following updates to the code will specify the stop word file to use and to incorporate the file into the mapper logic. Using Eclipse:</p> <ol style="list-style-type: none"> <li data-bbox="342 638 833 669">To the driver class (WordCount.java): <p data-bbox="391 709 1133 741">Add the following line to the top of the file, under the imports:</p> <pre data-bbox="391 743 743 774">import java.net.URI;</pre> <p data-bbox="391 814 1219 846">Under the comment "Add a file to each Mapper's cache" add the line:</p> <pre data-bbox="391 848 1409 879">job.addCacheFile(new URI("/user/gpadmin/stop_words.txt"));</pre> <li data-bbox="342 919 948 951">To the mapper class (WordCountMapper.java): <p data-bbox="391 991 1333 1022">Add the following line under the "Open up the Mapper's cached file" comment:</p> <pre data-bbox="391 1024 1198 1056">private File f = new File("./stop_words.txt");</pre> <p data-bbox="391 1096 1365 1163">Add the following lines under the "Store each stop word into an ArrayList<String>" comment:</p> <pre data-bbox="391 1165 1409 1423">String currentStopWord; BufferedReader br = new BufferedReader(new FileReader(f)); ArrayList<String> stopWords = new ArrayList<String>(); while((currentStopWord = br.readLine()) != null) { stopWords.add(currentStopWord); } br.close();</pre> <li data-bbox="342 1463 1382 1562">Near the end of WordCountMapper.java, place the following <code>if</code> statement around the existing <code>word.set()</code> and <code>context.write()</code> lines to only count the words not in the list of stop words. This piece of code should look like: <pre data-bbox="391 1602 1263 1766">if(!stopWords.contains(currentWord.toLowerCase())) { word.set(currentWord); context.write(word, one); }</pre>

Step	Action
7	<p><u>Use a Combiner Function</u></p> <p>As is often the case, the combiner functionality is identical to the reducer functionality. So, it is not necessary to write a new Java class. It is sufficient to specify that the existing reducer class is to be used as the combiner class. So, add the following line to the driver class (WordCount.java) under the comment “Add a combiner function”:</p> <pre>job.setCombinerClass(WordCountReducer.class);</pre>
8	<p><u>Specify the Number of Reduce Tasks</u></p> <p>In the <i>WordCount</i> project’s driver class, to specify that the Hadoop job will use three reduce tasks, add the following line to the driver class (WordCount.java) under the comment “Specify the number of Reducers that the job will use”:</p> <pre>job.setNumReduceTasks(3);</pre>
9	<p><u>Change the Output Filename</u></p> <p>In the first Hadoop job we ran, the output file was given the default name part-r-00000. This filename means that the output file was the first output partition from a reducer. In this step we will use a custom OutputFormat class to change the name of the output file. The custom class has already been created for you and is called MyTextOutputFormat. To use the MyTextOutputFormat class, add the following lines to the driver class WordCount.java under the comment “Change the name of the output file”:</p> <pre>job.getConfiguration().set("mapreduce.output.basename", "wordcount_output.txt"); job.setOutputFormatClass(MyTextOutputFormat.class);</pre> <p>The first line will change the base name of the output file to wordcount_output.txt. The second line invokes our MyTextOutputFormat class. The MyTextOutputFormat class will also add a unique identifier to the front of the filename in case we had multiple output files produced.</p> <p>By default, Hadoop uses the TextOutputFormat class. Our MyTextOutputFormat class makes small changes to the TextOutputFormat class.</p>

Step	Action
10	<p><u>Run the New Hadoop Job and View Output</u></p> <p>With the modifications made in steps 6 through 9,</p> <ol style="list-style-type: none"> Re-package the <i>WordCount</i> source code into a new JAR file, <code>wc_new.jar</code>. Submit the Hadoop job. View the output. <p>Note: The directory, to store the job's output data, must not exist before the job is submitted, otherwise, the job will fail.</p> <p>How many Mappers and Reducers were used this time? Write your answers below.</p> <p>Mappers: _____ Reducers: _____</p>

Step	Action
11	<p><u>View the Hadoop ResourceManager UI</u></p> <p>Hadoop comes with a ResourceManager web UI that provides various details about the Hadoop cluster and the progress of ongoing jobs or the history of completed jobs.</p> <p>To access the ResourceManager UI:</p> <ol style="list-style-type: none"> Open the Firefox Web Browser. Go to the address localhost:8088/cluster <p>The Hadoop ResourceManager UI should look like this:</p>  <p>In the center of the screen are all past and current Hadoop jobs that have been run on your cluster. You can click on a job ID's link for further job details. The list of links on the left allows you to view different categories of jobs such as jobs that are currently running, jobs that have finished, jobs that were killed, etc.</p>

Step	Action
12	<p><u>View the NameNode UI</u></p> <p>Hadoop also provides a NameNode web UI, which allows you to view the status of the cluster's NameNode and explore the file system of HDFS. Exploring HDFS is much easier with this UI than trudging through the HDFS directories in the terminal.</p> <p>To access the NameNode UI:</p> <ol style="list-style-type: none"> Open the Firefox Web Browser, if not already open Enter the address localhost:50070. <p>The NameNode UI home screen should appear similar to the following:</p>  <p>In order to browse your HDFS, click on the link that says Browse the file system under the Utilities drop-down tab.</p>

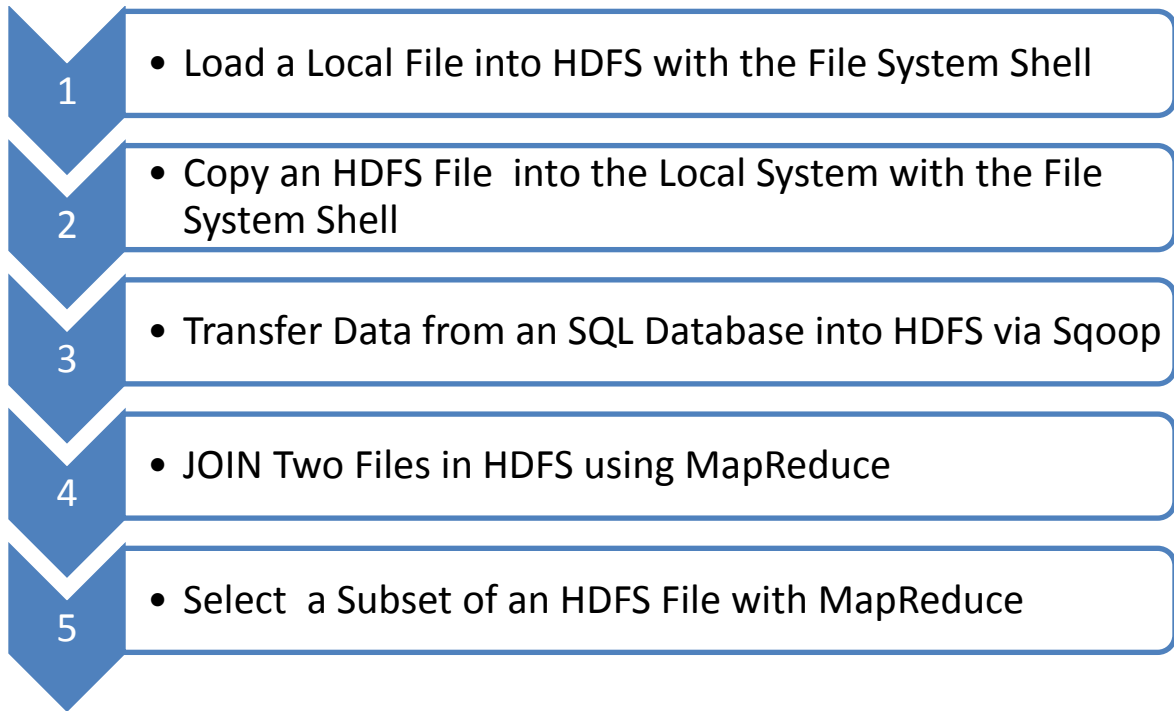
End of Lab Exercise

This PAGE intentionally left blank.

Lab Exercise 2: Working with HDFS

Purpose:	<p>This lab demonstrates different ways a user can interact with the Hadoop Distributed File System (HDFS). After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Use different HDFS shell commands to manipulate data• Import data from a relational database into HDFS• Finish and run Hadoop jobs that will manipulate data in HDFS
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Use the HDFS shell command <code>-put</code> to load a file into HDFS from your local machine• Use the HDFS shell command <code>-get</code> to copy a file from HDFS onto your local machine• Use Apache Sqoop to import structured data from a Greenplum database into HDFS• Run a Hadoop job that will JOIN two tables of data in HDFS• Run a Hadoop job that will SELECT a certain subset of data from a file in HDFS• Direct a Hadoop job to send its output to a local CSV file
References:	

Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Load a Local File into HDFS with the File System Shell</u></p> <p>Lab 1 used an HDFS file, stop_word.txt, with Hadoop's distributed cache system, to exclude commonly used words from being counted. A copy of this file exists in the local file system at ~/Labs/lab1. This step will illustrate how to move a file from the local file system to HDFS and into a new HDFS directory.</p> <p>From an open Terminal window:</p> <ol style="list-style-type: none">Verify that the stop_words.txt exists in the local file system <pre>cd ~/Labs/lab1 ls</pre>Make a new HDFS directory <pre>hdfs dfs -mkdir stopwords</pre>Move a file from the local file system to HDFS with the <code>-put</code> shell command <pre>hdfs dfs -put stop_words.txt stopwords</pre>Check that the file was properly transferred <pre>hdfs dfs -ls stopwords hdfs dfs -cat stopwords/stop_words.txt more</pre>
2	<p><u>Copy an HDFS File into the Local System with the File System Shell</u></p> <p>In Lab 1, an HDFS folder, WOutput, was created to store the part-r-00000 file which was the output from a submitted Hadoop job. In this step, part-r-00000 will be copied from HDFS to the local file system with the <code>-get</code> shell command.</p> <ol style="list-style-type: none">In a Terminal window, enter: <pre>cd ls Desktop</pre>If part-r-00000 does not already exist, copy part-r-00000 to the local file system <pre>hdfs dfs -get output/WOutput/part-r-00000 Desktop</pre>To verify that part-r-00000 was successfully transferred, go the CentOS desktop or enter: <pre>ls Desktop</pre>

Step	Action																				
3	<p><u>Transfer Data from an SQL Database into HDFS via Sqoop</u></p> <p>Apache Sqoop is a tool designed to efficiently import and export large amounts of data between HDFS and relational database management systems (RDBMS). In this step, Sqoop is used to import data into HDFS from a Greenplum PostgreSQL database.</p> <p>For the Sqoop tool, <code>import</code>, the corresponding arguments and their descriptions are:</p> <table border="1"> <thead> <tr> <th>Argument</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>--driver <class-name></code></td><td>Manually specify JDBC driver class to use</td></tr> <tr> <td><code>--connect <jdbc-uri></code></td><td>Specify JDBC connect string</td></tr> <tr> <td><code>--username <username></code></td><td>Set authentication username</td></tr> <tr> <td><code>--password <password></code></td><td>Set authentication password</td></tr> <tr> <td><code>--table <table-name></code></td><td>Table to read</td></tr> <tr> <td><code>--split-by <column-name></code></td><td>Column of the table used to split work units</td></tr> <tr> <td><code>--target-dir <dir></code></td><td>HDFS destination directory</td></tr> <tr> <td><code>--fields-terminated-by <char></code></td><td>Sets the field separator character</td></tr> <tr> <td><code>--num-mappers <n></code></td><td>Use n map tasks to import in parallel</td></tr> </tbody> </table> <p>Two database tables: <i>meteorites_characteristics</i> and <i>meteorites_locations</i> will be imported into HDFS directories of the same table names. Both tables are located in the database named <i>postgres</i>. After substituting 1) the IP address for the VM, 2) the provided password, and 3) an appropriate table name, run the following command for each table:</p> <pre>sqoop import --driver org.postgresql.Driver \ --connect jdbc:postgresql://<IP address>/postgres \ --username gpadmin \ --password <provided password> \ --table <table name> \ --split-by place \ --target-dir /user/gpadmin/input/lab2/<table name> \ --fields-terminated-by "\t" \ --num-mappers 1</pre> <p>Note: The “\” is a line continuation identifier. The “\t” instructs Sqoop to build HDFS files with tab-separated columns from the imported tables.</p> <p>How many reducers were run? ____ Why? _____</p> <p>Verify that the files have been imported into HDFS by executing the following commands</p> <pre>hdfs dfs -ls input/lab2/meteorites_characteristics hdfs dfs -ls input/lab2/meteorites_locations</pre>	Argument	Description	<code>--driver <class-name></code>	Manually specify JDBC driver class to use	<code>--connect <jdbc-uri></code>	Specify JDBC connect string	<code>--username <username></code>	Set authentication username	<code>--password <password></code>	Set authentication password	<code>--table <table-name></code>	Table to read	<code>--split-by <column-name></code>	Column of the table used to split work units	<code>--target-dir <dir></code>	HDFS destination directory	<code>--fields-terminated-by <char></code>	Sets the field separator character	<code>--num-mappers <n></code>	Use n map tasks to import in parallel
Argument	Description																				
<code>--driver <class-name></code>	Manually specify JDBC driver class to use																				
<code>--connect <jdbc-uri></code>	Specify JDBC connect string																				
<code>--username <username></code>	Set authentication username																				
<code>--password <password></code>	Set authentication password																				
<code>--table <table-name></code>	Table to read																				
<code>--split-by <column-name></code>	Column of the table used to split work units																				
<code>--target-dir <dir></code>	HDFS destination directory																				
<code>--fields-terminated-by <char></code>	Sets the field separator character																				
<code>--num-mappers <n></code>	Use n map tasks to import in parallel																				

Step	Action
4	<p data-bbox="289 289 787 321"><u>JOIN Two Files in HDFS using MapReduce</u></p> <p data-bbox="289 373 1396 485">Using the two tables you imported into HDFS in the previous step (<i>meteorites_characteristics</i> and <i>meteorites_locations</i>), we are going to run a Hadoop job that will JOIN the two tables based on the column <i>place</i>.</p> <ol data-bbox="337 520 1404 730" style="list-style-type: none"> From <code>~/Labs/lab2</code>, import the project folder <i>JoinMeteorite</i> into Eclipse Review the comments throughout the project's three class files. Use the following code below to tell the Mapper how to perform the JOIN when there is a match between a row in <i>meteorites_characteristics</i> and a row in <i>meteorites_locations</i>. Copy the following bolded code into the <i>if</i> statement in the <i>JoinMeteoriteMapper.java</i> file: <pre data-bbox="386 758 1414 1150"> if(charPlace.equalsIgnoreCase(locPlace)) { year = tokenizer.nextToken(); coordinates1 = tokenizer.nextToken(); coordinates2 = tokenizer.nextToken(); mw = new MeteoriteWritable(type, mass, year, coordinates1, coordinates2); place.set(charPlace); context.write(place, mw); br.close(); return; // Since matches are one-to-one we can end the searching once there is a match } </pre> <p data-bbox="289 1192 1396 1297">This code creates an instance of our custom-made Writable object, <i>MeteoriteWritable</i>, with the appropriate information about the two rows we want to JOIN so that the new row can be sent to the final output.</p> <p data-bbox="289 1325 1065 1356">Note: A final version of the project is located in <code>~/Solutions/lab2</code>.</p> <p data-bbox="289 1392 1414 1549">The <i>JoinMeteorite</i> job needs to access both the <i>meteorites_characteristics</i> and <i>meteorites_locations</i> tables in HDFS when the job is run. Since the job's driver class caches the <i>meteorites_locations</i> table for each Mapper to access, we need to run this job with the HDFS directory location of the <i>meteorites_characteristics</i> table as the given input.</p> <ol data-bbox="337 1585 901 1654" style="list-style-type: none"> Build the <i>joinMeteorite.jar</i> file using Eclipse Submit the Hadoop job: <pre data-bbox="386 1682 1243 1745"> hadoop jar joinMeteorite.jar JoinMeteorite \ input/lab2/meteorites_characteristics output/join </pre> <ol data-bbox="337 1780 820 1812" style="list-style-type: none"> View the contents of the joined files: <pre data-bbox="386 1839 1174 1871"> hdfs dfs -cat output/join/part-m-00000 more </pre>

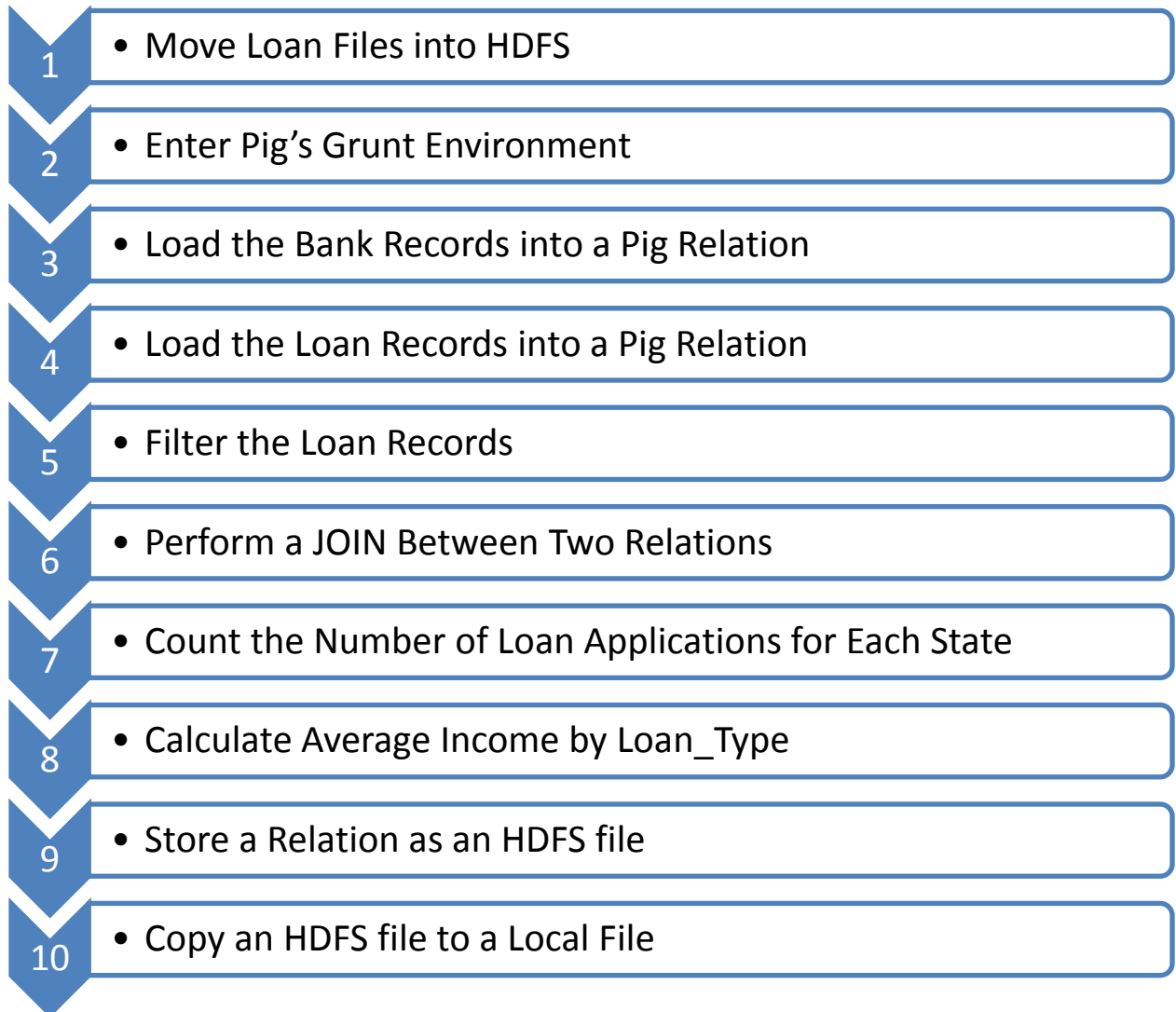
Step	Action
5	<p data-bbox="289 289 878 321"><u>Select a Subset of an HDFS File with MapReduce</u></p> <p data-bbox="289 373 1414 485">With the output from step 4, this step will use a Hadoop job to select those records where the mass of a meteorite is greater than 5,000,000 grams. Also, the selected records will be written into a local CSV file as well as HDFS.</p> <ol data-bbox="337 520 1252 625" style="list-style-type: none"> From <code>~/Labs/lab2</code>, import the project folder <i>SelectMeteorite</i> into Eclipse. Review the comments throughout the project's three class files. Add the following code to <code>SelectMeteoriteMapper.java</code>: <pre data-bbox="386 653 1357 814"> if(floatMass > 5000000) { MeteoriteWritable mw = new MeteoriteWritable(type, stringMass, year, coordinates1, coordinates2); context.write(new Text(place), mw); } </pre> At the very end of the driver class (after the method <code>Job.waitForCompletion()</code> is called), tell the job to copy its output onto your local machine in the form of a CSV file. To accomplish this task, copy the following line of code into the <code>SelectMeteorite.java</code> file: <pre data-bbox="386 1031 1382 1094"> fs.copyToLocalFile(new Path(args[1]+"/part-m-00000"), new Path("heavy_meteorites.csv")); </pre> Build the <code>selectMeteorite.jar</code> file Run: <pre data-bbox="386 1230 1398 1293"> hadoop jar selectMeteorite.jar SelectMeteorite output/join output/select </pre> Run the Hadoop job, make sure that a local CSV file was created, and view the output. <p data-bbox="289 1356 1295 1388">Did the output only include meteorites with mass greater than 5,000,000 g? _____</p> <p data-bbox="289 1423 1052 1455">How many rows of data were selected? _____</p>

End of Lab Exercise

Lab Exercise 3: Executing MapReduce Jobs with Apache Pig

Purpose:	<p>This lab demonstrates how to use Apache Pig to run MapReduce jobs and analyze large datasets. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Understand the basic data flow concepts of a Pig Latin program.• Understand the use of Pig's operators such as <code>LOAD</code>, <code>STORE</code>, <code>DUMP</code>, <code>DESCRIBE</code>, <code>FILTER</code>, <code>FOREACH</code>, <code>ILLUSTRATE</code>, <code>JOIN</code>, and <code>GROUP</code>.• Use Pig's Interactive and batch mode to run Pig jobs.
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Use Pig's <i>Grunt</i> shell to run Pig Latin programs.• Load data from HDFS into your Pig environment.• Filter out data from a relation.• Create categorical variables in a Pig relation.• Perform a <code>JOIN</code> between two datasets in Pig.• Save data from Pig into a local CSV file
References:	

Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Move Loan Files into HDFS</u></p> <p>Two files are located in the local directory, ~/Labs/lab3</p> <ul style="list-style-type: none">• 2011HMDAIInstitutionRecords.txt contains tab-separated data on various banks and institutions that approved or rejected loan applications in 2011.• 2011HMDALAR.csv contains comma-separated data on over 800,000 mortgage loan applications from 2011 corresponding to the banks in the first file. <p>From a terminal window, use the following commands to add the two files into HDFS:</p> <pre>hdfs dfs -put ~/Labs/lab3/2011HMDAIInstitutionRecords.txt bank_records</pre> <pre>hdfs dfs -put ~/Labs/lab3/2011HMDALAR.csv loan_records</pre>
2	<p><u>Enter Pig's Grunt Environment</u></p> <p>From a terminal window, enter:</p> <pre>pig</pre> <p>You should be at the grunt> prompt.</p>

Step	Action
3	<p><u>Load the Bank Records into a Pig Relation</u></p> <p>a. As a single line, enter the following command to load the text file into a Pig relation:</p> <pre>bank_records = LOAD 'bank_records/2011HMDAInstitutionRecords.txt' AS (year:chararray, bankID:chararray, name:chararray, address:chararray, city:chararray, state:chararray, zipCode:chararray);</pre> <p>b. View the structure of the relation:</p> <pre>DESCRIBE bank_records;</pre> <p>c. Examine a sample record:</p> <pre>ILLUSTRATE bank_records;</pre> <p>d. Count the number of records in the bank_records relation:</p> <pre>num = FOREACH (GROUP bank_records ALL) GENERATE COUNT(bank_records); DUMP num;</pre> <p>How many records are in bank_records? _____</p> <p>The (GROUP bank_records ALL) groups all of the tuples in bank_records into a single group. The FOREACH...GENERATE... then counts the number of occurrences of the single group in the bank_records relation.</p>
4	<p><u>Load the Loan Records into a Pig Relation</u></p> <p>a. As a single line, enter the following command to load the text file into a Pig relation:</p> <pre>loan_records = LOAD 'loan_records/2011HMDALAR.csv' USING PigStorage(',') AS (year:chararray, bankID:chararray, loan_type:int, property_type:int, loan_purpose:int, occupancy:int, loan_amount:int, action_taken:int, state_code:int, county_code:int,applicant_income:int);</pre> <p>b. Repeating the similar substeps in Step 3, verify that the LOAD of loan_records was properly performed and determine the number of loan records.</p> <p>How many records are in loan_records? _____</p>

Step	Action
5	<p><u>Filter the Loan Records</u></p> <p>For the purpose of an analysis, only the loans that meet the following criteria will be examined where:</p> <ul style="list-style-type: none"> • occupancy is a 1 or 2 (1 denotes owner-occupied / 2 denotes not owner-occupied) • action_taken is a 1, 3, or 6 (1 and 6 denote approved, 3 denotes rejected) <p>A full explanation of these codes can be found at http://www.ffiec.gov/hmdarawdata/FORMATS/2011HMDACodeSheet.pdf.</p> <p>Enter the following FILTER statement into the <i>Grunt</i> shell to remove rows where action_taken does not equal 1, 3 or 6 and occupancy equals 3:</p> <pre>filtered_loans = FILTER loan_records BY occupancy IN (1,2) AND action_taken IN (1,3,6);</pre>
6	<p><u>Perform a JOIN Between Two Relations</u></p> <p>In this step, based on the bankIDs in each relation, the filtered loan records will be joined with the bank records to build one relation.</p> <p>a. In case there are loans that do not have a matching bank, perform a left outer join by entering:</p> <pre>joined = JOIN filtered_loans BY bankID LEFT OUTER, bank_records BY bankID;</pre> <p>b. Since bankID and year appear in both joined relations, there is no need to pull these values twice. Thus, using the order designations of the fields, define a relation with only the distinct fields. So, the first 11 fields will be used from filtered_loans and the last 5 fields from bank_records. This new relation can be defined by entering:</p> <pre>clean_joined = FOREACH joined GENERATE \$0,\$1,\$2,\$3,\$4,\$5,\$6,\$7,\$8,\$9,\$10,\$13,\$14,\$15,\$16,\$17;</pre>

Step	Action
7	<p data-bbox="297 289 953 321"><u>Count the Number of Loan Applications for Each State</u></p> <p data-bbox="297 373 1328 485">Now that the bank and loan datasets have been filtered and joined, some simple data exploration will be conducted. First, count the number of loans by US state and sort in descending order based on the count.</p> <pre data-bbox="393 533 1339 831">grouped_by_state = GROUP clean_joined BY state; state_count = FOREACH grouped_by_state GENERATE group, COUNT(\$1); sorted_count = ORDER state_count BY \$1 DESC; dump sorted_count;</pre> <p data-bbox="297 919 1411 951">Which state had the most loan applications? _____ With how many applications? _____</p> <p data-bbox="297 1003 1422 1108">Note: In the <code>grouped_by_state</code> relation, the first column <code>group</code> represents the name of the state while the second column will be the count of every item in the tuple's bag. The bag is represented by the expression <code>\$1</code> because it is the second column in the relation.</p>

Step	Action
8	<p><u>Calculate Average Income by Loan Type</u></p> <p>Next, the average applicant income will be calculated for the four loan types:</p> <ul style="list-style-type: none"> 1 -- Conventional (any loan other than FHA, VA, FSA, or RHS loans) 2 -- FHA-insured (Federal Housing Administration) 3 -- VA-guaranteed (Veterans Administration) 4 -- FSA/RHS (Farm Service Agency or Rural Housing Service) <p>Note: The applicant income is expressed in thousands of US dollars.</p> <p>To obtain the respective average incomes, run the following three Pig Latin statements:</p> <pre>grp_by_loantype = GROUP clean_joined BY loan_type;</pre> <pre>income_avg = FOREACH grp_by_loantype GENERATE group, AVG(clean_joined.applicant_income);</pre> <pre>dump income_avg;</pre> <p>Note: To refer to a field within a bag, you must use the expression <i>b.f</i>, where <i>b</i> is the name of the bag, and <i>f</i> is the name of the field inside the bag as in <code>clean_joined.applicant_income</code>.</p>
9	<p><u>Store a Relation as an HDFS file</u></p> <p>Once the necessary data is in the proper structure, it is often necessary to persist the data as an HDFS file. This step will illustrate how a PiggyBank function can be used to store the results in a CSV format.</p> <ul style="list-style-type: none"> a. Enter the following statement to register the PiggyBank jar file for use in the Pig environment: <pre>REGISTER /home/gpadmin/Software/pig- 0.12.1/contrib/piggybank/java/piggybank.jar;</pre> b. Use the following <code>STORE</code> command with the PiggyBank function <code>CSVExcelStorage()</code>, write the relation, <code>clean_joined</code>, as a CSV file into HDFS: <pre>STORE clean_joined INTO '2011HMDA_joined' USING org.apache.pig.piggybank.storage .CSVExcelStorage();</pre>

Step	Action
10	<p data-bbox="297 289 688 321"><u>Copy an HDFS File to a Local File</u></p> <p data-bbox="345 359 1401 474">Once the data is in HDFS, the HDFS file can be copied to a local file with <code>hdfs dfs -get...</code> as seen in Lab 2. Alternatively, the <code>copyToLocal</code> command could be used to copy the HDFS to a local file by executing the following statement at the grunt prompt.</p> <pre data-bbox="394 520 1094 590">copyToLocal 2011HMDA_joined/part-r-00000 2011HMDA_joined.csv;</pre> <p data-bbox="297 636 1354 667">The Pig environment can be closed by entering the <code>quit</code> command at the grunt prompt.</p>

End of Lab Exercise

Lab Exercise 4: Executing MapReduce Jobs with Apache Hive

Purpose:	<p>This lab demonstrates how to use Apache Hive to analyze large datasets with a language similar to SQL. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Create tables in a Hive environment• Understand the basic concepts of HiveQL• Write queries on the tables created in Hive
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Create and populate external tables in Hive• Identify rows in a table with null values• Perform a Multi-table INSERT• Run a query with a user-defined function• Perform an INNER JOIN along with a LEFT OUTER JOIN• Run a query with Hive's <code>GROUP BY</code> statement• Export a Hive table to a local CSV file
References:	<p>When you create a Hive table, by default Hive will store and manage the table and its corresponding data files in its warehouse directory /user/hive/warehouse in HDFS. This means that when you load data into a Hive-managed table, the data file will be <i>moved</i> (as opposed to <i>copied</i>) into the warehouse. If you drop the table, the data file will be lost forever.</p> <p>As an alternative, we can create an <i>external</i> table, telling Hive to refer to data that is stored outside the warehouse directory. This way, other processes can continue to use the same data file(s). Also, if the external table is dropped, nothing will happen to the actual data files, only the table's metadata will be deleted. Therefore, we will use external tables to store the mortgage data since we may want to use the data in other processes.</p>

Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Enter the Hive Shell</u></p> <p>In this lab, the same home loan mortgage datasets that were used in the Pig Lab Exercise 3 will be used. These two datasets are located in the directory <code>~/Labs/lab4</code>.</p> <ul style="list-style-type: none"> • 2011HMDAInstitutionRecords.txt contains tab-separated data on various banks and institutions that approved or rejected loan applications in 2011. • 2011HMDALAR.csv contains comma-separated data on over 800,000 mortgage loan applications from 2011 corresponding to the banks in the first file. <p>Note: In the previous lab, these files were copied into HDFS directories <i>bank_records</i> and <i>loan_records</i>, respectively. So, these directories should already be created and contain the datasets. If not, run step 1 of lab exercise 3.</p> <p>Open the Hive shell by typing <code>hive</code> into a terminal:</p> <pre>hive</pre> <p>The <code>hive></code> prompt should appear.</p>
2	<p><u>Create Hive Tables</u></p> <p>Enter the following two Hive statements into the Hive shell to create external tables for the bank and loan records:</p> <pre>CREATE EXTERNAL TABLE bank_records (year STRING, bankID STRING, name STRING, address STRING, city STRING, state STRING, zipCode STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/user/gpadmin/bank_records'; CREATE EXTERNAL TABLE loan_records (year STRING, bankID STRING, loan_type INT, property_type INT, loan_purpose INT, occupancy INT, loan_amount INT, action_taken INT, state_code INT, county_code INT, applicant_income INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/user/gpadmin/loan_records';</pre>

Step	Action
3	<p><u>Identify Rows with Null Values</u></p> <p>We will identify rows in <code>loan_records</code> where <code>applicant_income</code> is null. Using Hive's aggregate <code>COUNT()</code> function, query the table <code>loan_records</code> to find out how many rows have a null value for <code>applicant_income</code>. Enter the following code into your Hive shell:</p> <pre>SELECT COUNT(*) FROM loan_records WHERE applicant_income is null;</pre> <p>How many rows in <code>loan_records</code> have a null value for <code>applicant_income</code>? _____</p>
4	<p><u>Split the Loan Records into Two Tables</u></p> <p>This step will split the <code>loan_records</code> table into two tables based on whether or not the <code>applicant_income</code> is null.</p> <ol style="list-style-type: none"> Create two tables of the same structure as <code>loan_records</code>: <pre>CREATE TABLE income_provided LIKE loan_records; CREATE TABLE income_missing LIKE loan_records;</pre> Both <code>income_provided</code> and <code>income_missing</code> are tables managed by Hive, as opposed to being external tables. Enter the following code into the Hive shell to view the directories that were created for each table in the Hive warehouse: <pre>dfs -ls /user/hive/warehouse;</pre> Split the <code>loan_records</code> into the appropriate Hive table by running the following: <pre>INSERT OVERWRITE TABLE income_provided SELECT * FROM loan_records WHERE applicant_income is not null;</pre> <p>What was the number of jobs run? _____ What was the time taken? _____</p> <pre>INSERT OVERWRITE TABLE income_missing SELECT * FROM loan_records WHERE applicant_income is null;</pre> <p>What was the number of jobs run? _____ What was the time taken? _____</p>

Step	Action
5	<p><u>Use a Multi-table INSERT</u></p> <p>The previous step will now be completed using multi-table INSERT which will require only one pass through the source table, <code>loan_records</code>. Run:</p> <pre>FROM loan_records INSERT OVERWRITE TABLE income_provided SELECT * WHERE applicant_income is not null INSERT OVERWRITE TABLE income_missing SELECT * WHERE applicant_income is null;</pre> <p>What was the number of jobs run? _____ What was the time taken? _____</p> <p>In step 4, two separate INSERT statements were executed. So, two complete passes through <code>loan_records</code> had to be completed. For repeatedly processing a very large table, a multi-table INSERT may be quite beneficial in terms of time savings.</p>
6	<p><u>Establish a User-Defined Function</u></p> <p>In this step, we will run a query with the user-defined function (UDF) <code>getRange()</code> to display the range, the maximum value minus the minimum value, of any column that contains integers.</p> <p>To use a UDF in Hive, we must first package the function's Java code into a JAR file and register the file with Hive.</p> <ol style="list-style-type: none"> In Eclipse, import the project <code>UDAF_range</code> located in the directory <code>~/Labs/lab4</code>. Package the source code of this project into a JAR file named <code>Range.jar</code> and export the file into directory <code>/home/gpadmin</code>. To register the JAR file with Hive, enter the following code into the Hive shell: <pre>ADD JAR Range.jar;</pre> Enter the following code into your Hive shell in order to give an alias to the function's class name, which enables us to call the function in a query: <pre>CREATE TEMPORARY FUNCTION getRange AS 'com.hadoopbook.hive.Range';</pre> <p>As the keyword <code>TEMPORARY</code> implies, UDFs are defined only for the duration of the current Hive session.</p>

Step	Action
7	<p><u>Query with a User-Defined Function</u></p> <p>The UDF <code>getRange()</code> is now available, just like any Hive built-in function. Use the built-in functions <code>MAX()</code> and <code>MIN()</code> along with the UDF <code>getRange()</code> to calculate the maximum, minimum and range values of the column <code>loan_amount</code> from the table <code>loan_records</code>. Enter the following query into your Hive shell:</p> <pre>SELECT 'Minimum', MIN(loan_amount) FROM loan_records UNION ALL SELECT 'Maximum', MAX(loan_amount) FROM loan_records UNION ALL SELECT 'Range', GETRANGE(loan_amount) FROM loan_records;</pre> <p>Enter the resulting values:</p> <p>Maximum: _____ Minimum: _____ Range: _____</p>
8	<p><u>Perform an INNER JOIN</u></p> <p>In this step, we will run a query that performs an <code>INNER JOIN</code> between the <code>loan_records</code> and <code>bank_records</code> tables based on the column <code>respondentID</code>.</p> <p>In order to perform the <code>INNER JOIN</code>, enter the following code into your Hive shell:</p> <pre>SELECT l.*, b.name, b.address, b.city, b.state, b.zipCode FROM loan_records l JOIN bank_records b ON (l.bankID = b.bankID) LIMIT 10;</pre> <p>Note: The use of the keyword <code>LIMIT</code> reduces query output to the specified number of rows.</p> <p>As is the case with Apache Pig, notice how much easier it is to write a <code>JOIN</code> in Hive than it is to code out a <code>JOIN</code> in a Java MapReduce job.</p>

Step	Action
9	<p><u>Create a New Table and Use a LEFT OUTER JOIN</u></p> <p>In this step, we will perform a LEFT OUTER JOIN between the tables <code>loan_records</code> and <code>bank_records</code> and store the output into a new table using the <code>CREATE TABLE...AS SELECT</code> operation.</p> <p>a. Enter:</p> <pre>CREATE TABLE joined_records AS SELECT l.*, b.name, b.address, b.city, b.state, b.zipCode FROM loan_records l LEFT OUTER JOIN bank_records b ON (l.bankID = b.bankID);</pre> <p>b. To verify the table creation, enter the following command to list all of the tables that currently exist in the Hive environment (both managed and external tables):</p> <pre>SHOW TABLES;</pre>
10	<p><u>Identify Cities with the Highest Number of Loan Applications</u></p> <p>In this step, Hive's GROUP BY statement is used to query the table <code>joined_records</code> (created in step 9) and provide a sorted list of the top 10 cities with the highest number of loan applications.</p> <pre>SELECT city, state, COUNT(*) AS qty FROM joined_records GROUP BY city, state ORDER BY qty DESC LIMIT 10;</pre> <p>List the top 3 cities and their loan counts:</p> <hr/> <hr/> <hr/>

Step	Action
11	<p data-bbox="300 289 760 321"><u>Export a Hive Table to a Local CSV File</u></p> <p data-bbox="300 373 1404 489">In this step, the <code>joined_records</code> will be dropped and rebuilt to store the table as a comma delimited file in order to move the file to the local file system as a CSV file in the next step.</p> <ol style="list-style-type: none"> <li data-bbox="349 525 673 556">Drop the existing table: <pre data-bbox="396 583 849 615">DROP TABLE joined_records;</pre> <li data-bbox="349 651 1128 682">To ensure that the created table is not a compressed file, run: <pre data-bbox="396 709 1024 741">set hive.exec.compress.output=false;</pre> <li data-bbox="349 777 1177 808">Rebuild the table specifying the fields are to be comma delimited: <pre data-bbox="396 846 1307 1108">CREATE TABLE joined_records ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' AS SELECT l.*, b.name, b.address, b.city, b.state, b.zipCode FROM loan_records l LEFT OUTER JOIN bank_records b ON (l.bankID = b.bankID);</pre> <li data-bbox="349 1144 1161 1176">Determine the HDFS file for the Hive table, <code>joined_records</code>: <pre data-bbox="396 1213 1166 1245">dfs -ls /user/hive/warehouse/joined_records;</pre> <li data-bbox="349 1281 803 1312">Export the HDFS file to a local file: <pre data-bbox="396 1350 1323 1423">dfs -get /user/hive/warehouse/joined_records/000000_0 joined_records.csv;</pre> <li data-bbox="349 1459 730 1491">To exit the Hive shell, enter: <pre data-bbox="396 1518 483 1549">quit;</pre>

End of Lab Exercise

Lab Exercise 5: Storing and Accessing Data in Apache HBase

Purpose:	<p>This lab demonstrates how to use Apache HBase to store and retrieve massive amounts of data in real-time. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Understand the basic concepts of HBase• Create, populate, and drop tables in HBase• Retrieve data from tables in HBase in real-time
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Create a table in HBase• Bulk import data into a table• Query the data stored in a table• Add new versions of a table cell• Delete certain versions of a table cell• Dynamically add new columns to a pre-existing table• Export a table out of HBase• Drop a table in HBase
References:	

Workflow Overview



Lab Instructions

1	<p>Create an HBase Table</p> <p>This lab exercise uses the same home mortgage data that was used in exercises 3 and 4. However, for this lab, a tab-separated data file of the loan data already joined with the bank data is provided in directory <code>~/Labs/lab5</code>. The file is named 2011HMDA_joined_tsv.txt.</p> <p>In order to enter data into HBase, each data record must contain a row key. All data in HBase is accessed via this unique row key, which is automatically sorted by HBase. Therefore, your table's key design should lend itself to how the data is going to be read. In the file 2011HMDA_joined_tsv.txt, an extra column has been added to serve as the row key for the table. The row key is a composite key which consists of the respondent ID and a unique identifying number. Therefore, all rows with the same respondentID will be stored next to each other in HBase and lends to quicker access when querying for loans belonging to a particular respondentID.</p> <ol style="list-style-type: none">Open a terminal windowCopy the HMDA data into the HDFS directory named joined_records_tsv: <pre>hdfs dfs -put ~/Labs/lab5/2011HMDA_joined_tsv.txt joined_records_tsv</pre>Open the HBase shell by entering <code>hbase shell</code> into your terminal.In order to create a table for the joined HMDA data, enter the following code into your HBase shell: <pre>create 'hmda', {NAME => 'loan', VERSIONS => 2}, \ {NAME => 'bank', VERSIONS => 2}, \ {SPLITS => ['0000013347_1136035', '0000080028_1312871', \ '0000707056_1490981']}</pre> <p>This command will create an HBase table called <i>hmda</i> that contains two column families: <i>loan</i> and <i>bank</i>. The VERSIONS option specifies the number of versions that will be saved for each cell in the table. The SPLITS option specifies how the table will be divided based on the row portion of the key. In this example, the table will be split into four parts, called regions. Rows less than '0000013347_1136035' will be added to the first region; rows from '0000013347_1136035' to less than '0000080028_1312871' will be added to the second region, and likewise for the remaining splits.</p>

2	<p><u>Populate an HBase Table from an HDFS file</u></p> <ol style="list-style-type: none"> Enter <code>exit</code> to leave the HBase shell. An HBase table is split into regions, and each region is split into column families that are stored as separate files in HDFS. To see how the 'hmda' table is actually being stored, run: <pre>hdfs dfs -ls -R /hbase/data/default/hmda</pre> From the terminal prompt, import the HMDA data from Step 1 into the HBase table using HBase's <code>importtsv</code> command by entering the following: <pre>hbase org.apache.hadoop.hbase.mapreduce.ImportTsv\ -Dimporttsv.columns=loan:year,bank:respondentID,\ loan:type_conventional,loan:type_FHAInsured,\ loan:type_VAguaranteed,loan:type_FSAorRHS,\ loan:property_single_family,\ loan:property_manufactured_housing,\ loan:property_multifamily,\ loan:purpose_homePurchase,loan:purpose_homeImprovement,\ loan:purpose_refinancing,loan:isOccupied,\ loan:loan_amount,loan:isApproved,loan:state_code,\ loan:county_code,loan:applicant_income,\ bank:respondent_name,bank:respondent_address,\ bank:respondent_city,bank:respondent_state,\ bank:respondent_zipCode,HBASE_ROW_KEY 'hmda'\ 'joined_records_tsv'</pre> <p>This command adds different columns to the two column families we already created for the <i>hmda</i> table (<i>loan</i> and <i>bank</i>) and populates them with data from our 2011HMDA_joined_tsv.txt file.</p> <p>Note: The backwards slashes at the ends of lines are line continuation markers.</p>

3	<p><u>Retrieve Data with the get and scan Commands</u></p> <p>a. At the terminal prompt, enter: <code>hbase shell</code></p> <p>b. Use the <code>get</code> command to retrieve the data for row, 0000014564_1167896:</p> <pre>get 'hmda', '0000014564_1167896'</pre> <p>What is the <i>bank:respondent_name</i> for this row? _____</p> <p>c. Using <code>scan</code>, retrieve the data starting at row 0000014564_1167896, but ending before 0000014564_1167898:</p> <pre>scan 'hmda', {STARTROW => '0000014564_1167896', STOPROW => '0000014564_1167898'}</pre> <p>How many rows were returned? _____</p> <p>d. Using the <code>PrefixFilter()</code> function under the <code>scan FILTER</code> argument, retrieve the respondent ID and loan amount for all rows of data where the bank's respondent ID is 0000014564. <code>PrefixFilter()</code> checks if the row begins with the given argument – in this case 0000014564. To accomplish this, copy the following code into your HBase shell:</p> <pre>scan 'hmda', {COLUMNS => ['bank:respondentID', \ 'loan:loan_amount'], \ FILTER => "PrefixFilter('0000014564')"} </pre>

4	<p><u>Add New Versions of a Table Cell</u></p> <p>By default, when a user performs a <code>scan</code> or <code>get</code> operation, only the most recent versions of each cell are displayed; that is, the cells with the largest timestamps are displayed.</p> <p>a. For row, 0000014564_1167896, update the respondent's name:</p> <pre>put 'hmda', '0000014564_1167896', 'bank:respondent_name', 'Big Bank'</pre> <p>b. View the columns for row, 0000014564_1167896:</p> <pre>get 'hmda', '0000014564_1167896'</pre> <p>Is the timestamp for <code>bank:respondent_name</code>, the same as the timestamp for the other columns? _____</p> <p>c. To see both versions of the row's <code>bank:respondent_name</code>, run</p> <pre>get 'hmda', '0000014564_1167896', {COLUMN => 'bank:respondent_name', VERSIONS => 2}</pre>
5	<p><u>Determine Timestamp for January 1, 2020 00:00:00</u></p> <p>At the hbase shell prompt, enter:</p> <pre>import java.text.SimpleDateFormat import java.text.ParsePosition SimpleDateFormat.new(\ "yy/MM/dd HH:mm:ss").parse("20/01/01 00:00:00",\ ParsePosition.new(0)).getTime()</pre> <p>Write down this timestamp value: _____</p>

6	<p><u>Use Put with a Specified Timestamp Value</u></p> <p>a. Use the <code>put</code> command to now change the respondent name of the row with row key 0000014564_1167896 (same as before) to “Trustworthy Bank” and specify the timestamp for January 1, 2020 12:00 a.m.</p> <pre>put 'hmda', '0000014564_1167896', 'bank:respondent_name', 'Trustworthy Bank', SimpleDateFormat.new(\ "yy/MM/dd HH:mm:ss").parse("20/01/01 00:00:00", \ ParsePosition.new(0)).getTime()</pre> <p>b. View the columns for this row:</p> <pre>get 'hmda', '0000014564_1167896'</pre> <p>c. The creation of the <i>hmda</i> table specified to use only the latest two versions of each cell for both column families. Therefore, the following <code>get</code> command only displays “Trustworthy Bank” and “Big Bank” corresponding to the previous puts.</p> <pre>get 'hmda', '0000014564_1167896', \ {COLUMN => 'bank:respondent_name', VERSIONS => 3}</pre> <pre>get 'hmda', '0000014564_1167896', \ {RAW => TRUE, COLUMN => 'bank:respondent_name'}</pre> <p>d. However, the original respondent name is still in the database. Enter the following into the HBase shell to view the original respondent name for the row with row key 0000014564_1167896. Be sure to replace the <code><timestamp></code> argument with the timestamp of the original data. Do not include any quotes around the timestamp value.</p> <pre>get 'hmda', '0000014564_1167896', \ {COLUMN => 'bank:respondent_name', \ TIMESTAMP => <timestamp>}</pre>

7	<p><u>Delete Table Cells</u></p> <p>In this lab, we are going to delete the newest <code>respondent_name</code> cell (“Trustworthy Bank”) that we added to the row with row key <code>0000014564_1167896</code>. In HBase, if a particular version of a cell is deleted, all older versions of that cell will also be deleted.</p> <p>a. Examine the available <code>respondent_names</code> for the row of interest:</p> <pre>get 'hmda', '0000014564_1167896', {COLUMN => 'bank:respondent_name', VERSIONS => 2}</pre> <p>b. Delete the updated <code>respondent_names</code> with the timestamp for 2020:</p> <pre>delete 'hmda', '0000014564_1167896', 'bank:respondent_name', 1577854800000</pre> <p>c. Re-examine the available <code>respondent_names</code> for the row of interest:</p> <pre>get 'hmda', '0000014564_1167896', {COLUMN => 'bank:respondent_name', VERSIONS => 2}</pre>
8	<p><u>Add a New Column to a Table</u></p> <p>Recall that a column consists of a column family and a column qualifier. A new column qualifier, called <i>eSignature</i>, will be added to the <i>loan</i> column family for the <i>hmda</i> table. This column will have a value of either 1 or 0, representing whether or not the loan was signed with an e-signature. This column qualifier is not added until it is necessary to write something to the table for the first time.</p> <p>a. To add the new column, enter:</p> <pre>put 'hmda', '0000014564_1167896', 'loan:eSignature', '1'</pre> <p>b. To verify that the column has been added, enter:</p> <pre>get 'hmda', '0000014564_1167896'</pre>

9	<p><u>Build an HBase Table from an Existing HBase Table</u></p> <p>In this step, a subset of the <i>hmda</i> table will be added to a new table called <i>hmda_subset</i>, which will contain all rows corresponding to the respondent ID of 0000014564.</p> <ol style="list-style-type: none"> Create <i>hmda_subset</i>: <pre>create 'hmda_subset', {NAME => 'loan', VERSIONS => 2}, \ {NAME => 'bank', VERSIONS => 2}, \ {SPLITS => ['0000014564_0000200']}</pre> Exit the HBase shell, by entering <code>quit</code> Using HBase's <code>CopyTable</code> function, populate the <i>hmda_subset</i> table with rows from <i>hmda</i> where the respondent ID is 0000014564. At the terminal prompt, enter: <pre>hbase org.apache.hadoop.hbase.mapreduce.CopyTable \ --startrow=0000014564_0000000 \ --stoprow=0000014564_9999999 \ --versions=1 \ --new.name=hmda_subset 'hmda'</pre> <p>This is an example of how MapReduce can be applied to an HBase table</p>
10	<p><u>Write an HBase table to a Local File</u></p> <ol style="list-style-type: none"> From the terminal prompt, run an HBase scan, but write the results to a local file: <pre>echo "scan 'hmda_subset'" hbase shell > hbase_table.txt</pre> View the contents of the file <pre>cat hbase_table.txt more</pre> <p>Use CTRL-C to end the viewing of the file</p>

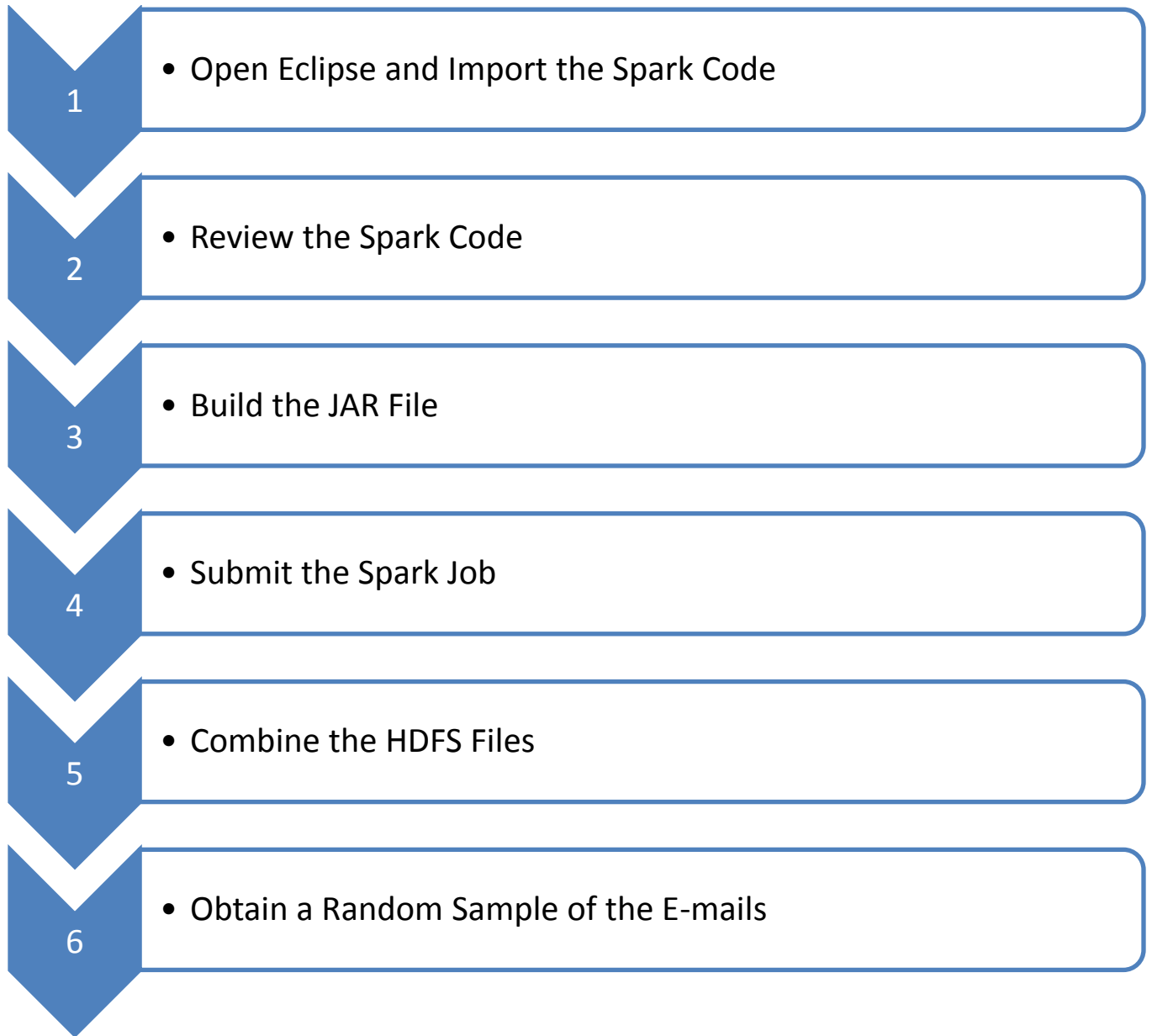
11	<p><u>Drop an HBase Table</u></p> <p>In order to drop/delete an entire table in HBase, you must first <i>disable</i> the table. After the table has been disabled, you can then successfully drop/delete the table.</p> <ol style="list-style-type: none">Enter the HBase shell.Disable the <i>hmda_subset</i> table with the following command: <code>disable 'hmda_subset'</code>Drop the <i>hmda_subset</i> table with the following command: <code>drop 'hmda_subset'</code>Exit the HBase shell. <code>quit</code>

End of Lab Exercise

Lab Exercise 6: Spark

Purpose:	<p>This lab is designed to:</p> <ul style="list-style-type: none">• Familiarize the student with the use of Spark• Prepare a dataset for use in the final lab exercise
Tasks:	<p>Tasks you will complete in this lab include:</p> <ul style="list-style-type: none">• Build and submit a Spark JAR file to process the Enron e-mail dataset for the final lab exercise.• Use Python to obtain a random sample of the e-mails
References:	<p>The final lab exercise in this lab guide.</p> <p>Enron dataset: https://www.cs.cmu.edu/~enron/</p>

Spark - Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Open Eclipse and Import the Spark Code</u></p> <p>This exercise will combine, into a single file, over a 100,000 separate e-mails that were disclosed during the federal government's investigation of Enron. Each row of the resulting file will correspond to exactly one Enron e-mail file. To reduce the number of duplicate e-mails, only the contents of the Enron employees Sent folders are used in this exercise. The emails are already loaded into an HDFS directory named enron.</p> <p>This step will import the provided Eclipse project that contains the Spark code to combine the individual e-mails into one file.</p> <ol style="list-style-type: none"> Open Eclipse. Go to File > Import..., click on Existing Projects into Workspace, and using the Browse button for the root directory, select the directory ~/gpadmin/Labs/lab6/Enron_Spark and click OK. Click Finish.
2	<p><u>Review the Spark Code</u></p> <p>Using Package Explorer in Eclipse:</p> <ol style="list-style-type: none"> Double click on the Enron_Spark project Double click on src Double click on com.emc.final_lab Double click on ConvertEmailsToRows.java Examine the code. <p>In which directory will the output of the Spark job be written? _____</p> <p>In which file system? _____</p>
3	<p><u>Build the JAR File</u></p> <p>In Eclipse's Package Explorer,</p> <ol style="list-style-type: none"> Right-click on the project's src folder and click on Export... Under the Java folder, select JAR File, and click Next. Under the heading Select the export destination, click Browse. In the new window, go to the directory ~/gpadmin/Labs/lab6, enter enron_spark.jar as the name for the JAR file at the top of the window, and click OK. Click Finish on the JAR export window.

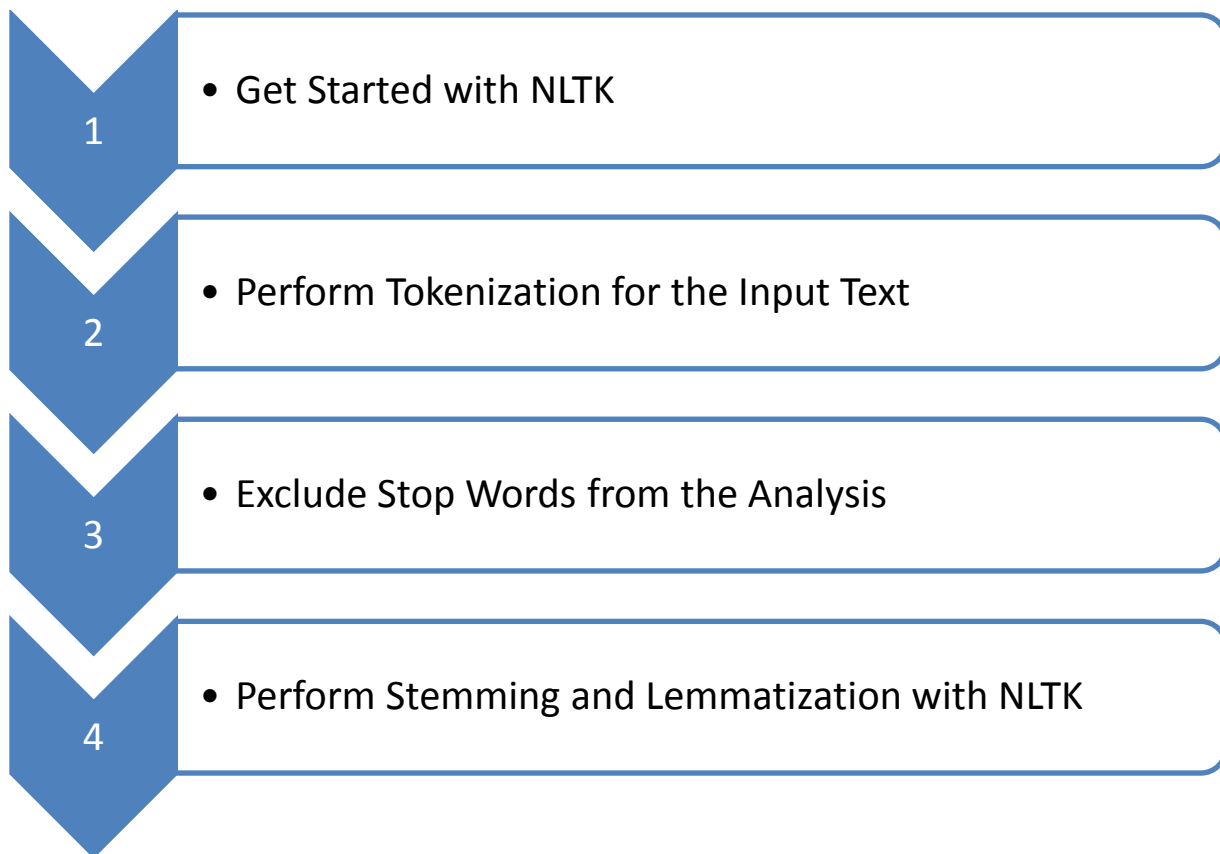
Step	Action
4	<p><u>Submit the Spark Job</u></p> <p>From a terminal prompt,</p> <ol style="list-style-type: none"> Change the current directory to the location of the JAR file. To submit the Spark job, enter: <pre>spark-submit --class com.emc.final_lab.ConvertEmailsToRows enron_spark.jar</pre>
5	<p><u>Combine the HDFS Files</u></p> <p>From a terminal prompt,</p> <ol style="list-style-type: none"> Examine the Spark job output: <pre>hdfs dfs -ls email_output</pre> <ol style="list-style-type: none"> Copy the HDFS files to the local directory: <pre>hdfs dfs -get email_output/part-00000 hdfs dfs -get email_output/part-00001</pre> <ol style="list-style-type: none"> Concatenate the two output files into one file by entering the following commands: <pre>cat part-00001 >> part-00000 mv part-00000 email_output.txt rm part-00001</pre> <p>Record the location of email_output.txt. _____</p>
6	<p><u>Obtain a Random Sample of the E-mails</u></p> <p>For development/testing purposes in the final lab exercise, extract a small random sample of the e-mail records. In the ~/Labs/lab6 directory, a Python script, sample.py is provided to extract 1,000 random e-mails. Run the following command from the same directory as email_output.txt:</p> <pre>python ~/Labs/lab6/sample.py</pre> <p>The file sample.txt will contain the 1,000 random records.</p>

End of Lab Exercise

Lab Exercise 7: Basic Text Processing with NLTK

Purpose:	<p>This lab introduces you to natural language processing using Python NLTK. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Understand basic concepts of text processing• Perform basic tasks with Python NLTK
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Perform tokenization• Remove stop words• Perform stemming and lemmatization
References:	<ul style="list-style-type: none">• http://www.nltk.org

Workflow Overview



Lab Instructions

Step	Action
1	<p>Get Started with NLTK</p> <p>a. Open a terminal and change the directory:</p> <pre>cd ~/Labs/lab7</pre> <p>b. Enter the python interactive command-line:</p> <pre>python</pre> <p>Note: The following Python code can be found in the file ~/Solutions/lab7/basic_nlp_solution.py.</p> <p>c. The NLTK package has already been installed in the system. To load the package, enter the following command:</p> <pre>>>> import nltk</pre>

Step	Action
2	<p data-bbox="289 289 769 321"><u>Perform Tokenization for the Input Text</u></p> <p data-bbox="289 357 1403 472">Tokenization, also called tokenizing, is the NLP task of separating words from the body of text. After the tokenization, raw text is converted into collections of tokens, where each token is generally a word.</p> <ol style="list-style-type: none"> <li data-bbox="337 508 1411 609">The <code>word_tokenize()</code> function uses NLTK's recommended word tokenizer to tokenize words in a given sentence. The following code shows how to use this function to tokenize a string called <code>text</code>: <pre data-bbox="378 646 1395 808">>>> from nltk import word_tokenize >>> text = "Sue, Can you give me the conf. call number and pass code?" >>> tokens1 = word_tokenize(text) >>> tokens1</pre> <li data-bbox="337 879 1354 945">The <code>wordpunct_tokenize()</code> function divides a text into sequences of tokens based on whitespaces and punctuation. Enter the following command: <pre data-bbox="378 982 1062 1081">>>> from nltk import wordpunct_tokenize >>> tokens2 = wordpunct_tokenize(text) >>> tokens2</pre> <li data-bbox="337 1152 1395 1218">The <code>tokenize</code> method of <code>SpaceTokenizer</code> class divides a string into substrings by treating any single space character as a separator. Enter the following command: <pre data-bbox="378 1255 1167 1354">>>> from nltk import SpaceTokenizer >>> tokens3 = SpaceTokenizer().tokenize(text) >>> tokens3</pre> <p data-bbox="383 1392 1362 1423">Compare these three outputs. What are the differences between each function?</p> <hr data-bbox="383 1522 1255 1528"/> <li data-bbox="337 1600 1282 1631">Now try the <code>split()</code> method of the <code>text</code>. Enter the following command: <pre data-bbox="394 1669 675 1696">>>> text.split()</pre> <p data-bbox="383 1734 1190 1766">Which method above has the same result as the <code>split()</code> method?</p> <hr data-bbox="383 1833 1255 1839"/>

Step	Action
3	<p data-bbox="289 287 756 321"><u>Exclude Stop Words from the Analysis</u></p> <p data-bbox="289 352 1414 493">Stop words are words that occur very frequently in the text. In English, for example, it is common to remove words such as <i>the</i>, <i>a</i>, <i>of</i>, <i>and</i>, <i>to</i>, and <i>also</i> before further processing to reduce the high dimensionality in the text. NLTK includes a corpus of stop words built from the Snowball's stop lists: http://snowball.tartarus.org/.</p> <ol style="list-style-type: none"> <li data-bbox="337 531 1386 600">Import the <code>stopwords</code> corpus and print out the English stop words by entering the following commands: <pre data-bbox="383 636 1029 701">>>> from nltk.corpus import stopwords >>> stopwords.words('english')</pre> <li data-bbox="337 737 1313 770">Execute the command below to print out the number of stop words in English: <pre data-bbox="383 804 922 835">len(stopwords.words('english'))</pre> <p data-bbox="289 905 1380 938">How many English stop words are stored in the <code>stopwords</code> corpus? _____</p> <li data-bbox="337 1014 1333 1047">Change the parameter from 'english' to 'french' or 'german'. What is the result? <li data-bbox="337 1098 1425 1251">In Lab 6, you generated a text file (<i>email_output.txt</i>) from your Spark job. Each line in that file corresponds to an email from the Enron corpus. Next, let's remove stop words on some of the emails. The following function removes stop words from a list of tokens, and only keeps those tokens consisting of letters. <pre data-bbox="383 1274 1256 1535">import re def remove_stopwords(tokens, stopWords): cleaned = [] for t in tokens: if re.match("[a-zA-Z]+\$", t) and t not in stopWords: cleaned.append(t) return cleaned</pre>

Step	Action
3 Cont.	<p>e. The following function retrieves the first several lines from the text file, extracts the email content, removes the stop words, and plots the word frequencies:</p> <pre data-bbox="381 384 1360 1108"> from nltk import FreqDist def clean_file(fname, maxlines): sw = frozenset(stopwords.words('english')) fd = FreqDist() with open(fname) as raw: for i, line in enumerate(raw): if i == maxlines: break if line.strip() == "": continue segs = [x.strip()[1:-1] for x in line.split('\t')] text = ' '.join([segs[5], segs[6]]) text = text.lower() tokens = text.split() print '-'*20 print 'Email text (original):\n', tokens tokens = remove_stopwords(tokens, sw) print '\nEmail text (removed stopwords):\n', tokens for t in tokens: fd[t] += 1 fd.plot(20) </pre> <p>f. Now call the function:</p> <pre data-bbox="381 1234 1377 1266"> clean_file(fname='../lab6/email_output.txt', maxlines=10) </pre> <p>Compare the result before and after the stop words removal. Examine the plot. Write down the 20 most common words from the first 10 email messages after removing the stop words.</p> <hr data-bbox="381 1507 1253 1516"/>

Step	Action
4	<p data-bbox="289 289 893 321"><u>Perform Stemming and Lemmatization with NLTK</u></p> <p data-bbox="289 357 1430 527">In text preprocessing, we often want to reduce inflections or variant forms to the base form as a way to reduce the high dimensionality in the text. For example, we may want to map words such as <i>play</i>, <i>plays</i>, <i>played</i>, and <i>playing</i> to the same term. Lemmatization and stemming are two different techniques that reduce the number of dimensions and reduce inflections or variant forms to the base form.</p> <p data-bbox="337 569 1406 705">a. The PorterStemmer class implements the Porter stemming algorithm, which removes and replaces word suffixes to arrive at a common root form (stem) of the word with a set of heuristics. The following code defines a function that converts a list of tokens to their stems.</p> <pre data-bbox="381 743 1062 873"> from nltk.stem import PorterStemmer def stemming(tokens): ps = PorterStemmer() return [ps.stem(t) for t in tokens] </pre> <p data-bbox="337 919 1430 1056">b. The WordNetLemmatizer class uses WordNet as the dictionary to reduce a word to its base form. It returns the input word unchanged, if the word cannot be found in WordNet. The following code defines a function that performs lemmatization on a list of tokens.</p> <pre data-bbox="381 1094 1170 1224"> from nltk.stem import WordNetLemmatizer def lemmatization(tokens): wnl = WordNetLemmatizer() return [wnl.lemmatize(t) for t in tokens] </pre> <p data-bbox="337 1266 1360 1329">c. The following function performs lemmatization and stemming on a single email found at a specific line number.</p> <pre data-bbox="381 1367 1414 1854"> def simplify_words(fname, lineNum): with open(fname) as raw: for i, line in enumerate(raw): if i == lineNum: if line.strip() == "": continue segs = [x.strip()[1:-1] for x in line.split('\t')] # store email subject and content together text = ' '.join([segs[5], segs[6]]) tokens = text.lower().split() print 'Email text (original):\n', tokens print 'Lemmatizing:', lemmatization(tokens) print 'Stemming:', stemming(tokens) break </pre>

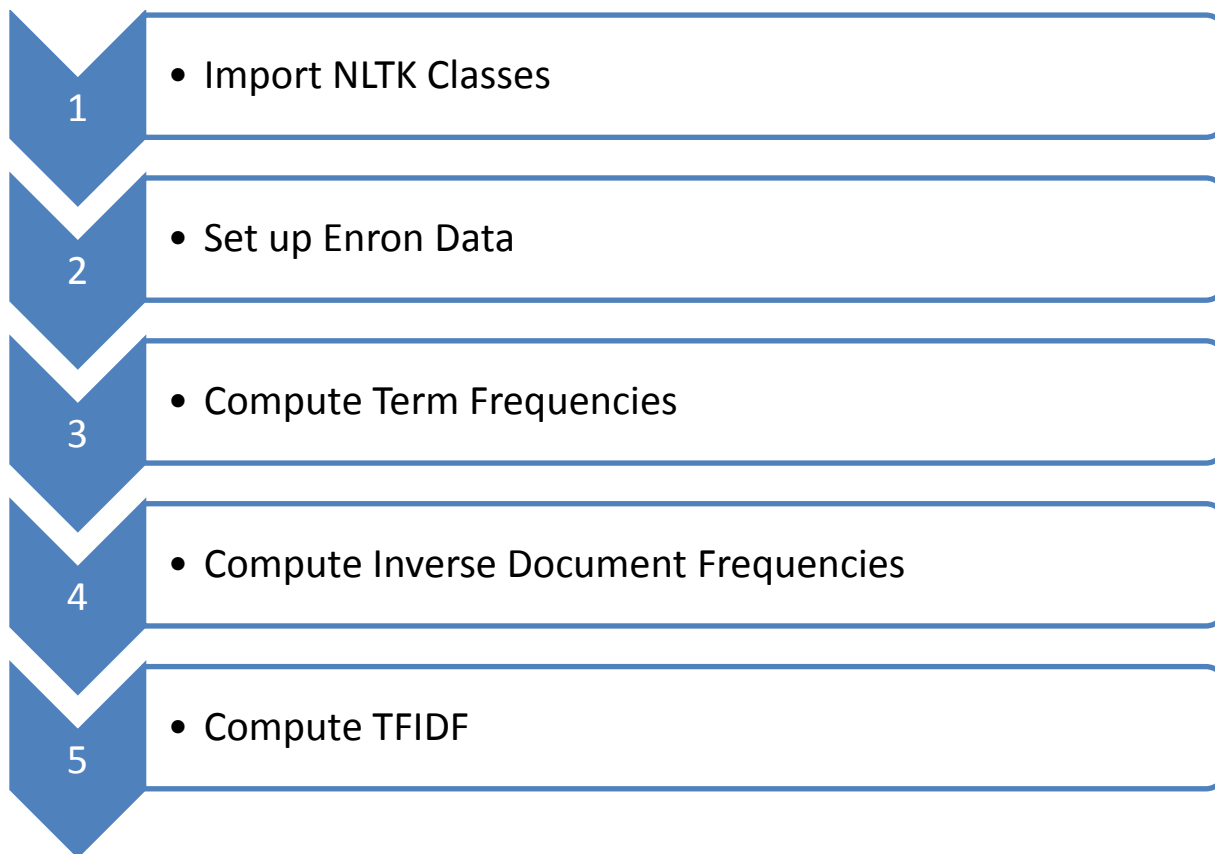
Step	Action
4 Cont.	<p>d. Now call the function to perform lemmatization and stemming on the 3rd email in <i>email_output.txt</i>:</p> <pre>simplify_words(fname='../lab6/email_output.txt', lineNum=3)</pre> <p>Compare the results from stemming and lemmatization. Are the stems actual words?</p> <p>What is the difference between stemming and lemmatization?</p>

End of Lab Exercises

Lab Exercise 8: TFIDF

Purpose:	<p>This lab demonstrates how to compute Term Frequency-Inverse Document Frequency (TFIDF) over textual data. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Understand notions of TFIDF• Compute TFIDF over any text collections
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Compute term frequencies• Compute inverse document frequencies• Compute TFIDF
References:	<ul style="list-style-type: none">• https://nltk.googlecode.com/svn/trunk/doc/api/nltk.text.TextCollection-class.html

Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Import NLTK Classes</u></p> <p>Note: All of the following Python code can be found in the file <code>~/Solutions/lab8/tfidf_solution.py</code>.</p> <p>Term Frequency-Inverse Document Frequency (TFIDF, also known as TF-IDF), is a numerical statistic which reflects how important a word is to a document in a collection or corpus. NLTK defines a <code>TextCollection</code> class which provides functions to compute the TF-IDF. Here we experiment the <code>TextCollection</code> over the Enron corpus.</p> <p>This lab needs to import two NLTK classes <code>nltk.text.Text</code> and <code>nltk.text.TextCollection</code> for computing TFIDF scores. The regular expression (<code>re</code>) class is needed for removing stop words.</p> <pre>from nltk.corpus import stopwords from nltk.text import Text, TextCollection import re</pre>

Step	Action
2	<p data-bbox="289 289 511 321"><u>Set up Enron Data</u></p> <p data-bbox="334 352 1409 457">a. The following code creates a function <code>create_texts()</code> that returns a list of <code>nltk.text.Text</code> objects from a corpus. Note that this function calls the <code>remove_stopwords()</code> function created in the previous lab.</p> <pre data-bbox="380 495 1398 1184"> def create_texts(fname, maxlines): # English stop words sw = frozenset(stopwords.words('english')) texts = [] with open(fname) as raw: for i, line in enumerate(raw): if i == maxlines: break # Skip all empty lines if line.strip() == "": continue # Need to remove the leading and ending double quotes segs = [x.strip()[1:-1] for x in line.split('\t')] # store email subject and content together text = ' '.join([segs[5], segs[6]]) text = text.lower() tokens = text.split() tokens = remove_stopwords(tokens, sw) texts.append(Text(tokens)) return texts </pre> <p data-bbox="334 1224 1393 1287">b. The following code calls the <code>create_texts()</code> function to build a list of <code>Text</code> objects over the top 1000 lines of the Enron email corpus.</p> <pre data-bbox="380 1325 1414 1388"> enronTexts = create_texts(fname='../lab6/email_output.txt', maxlines=1000) </pre> <p data-bbox="380 1428 1409 1491">Note: You may increase the value of <code>maxlines</code> to perform TFIDF over a larger subset of the email corpus.</p>

Step	Action
3	<p data-bbox="289 289 625 321"><u>Compute Term Frequencies</u></p> <p data-bbox="337 352 1328 384">a. Set up a TextCollection object over the list of Texts created in the previous step:</p> <pre data-bbox="381 422 1010 453">enronTC = TextCollection(enronTexts)</pre> <p data-bbox="337 485 1369 552">b. Compute the term frequency of the word “business” in the 2nd email (Note: Python index starts at 0).</p> <pre data-bbox="381 590 1027 621">enronTC.tf('business', enronTexts[1])</pre> <p data-bbox="381 659 1341 690">What is the term frequency of the word “business”? _____</p> <p data-bbox="337 728 1369 760">c. Similarly, compute the term frequencies of words “meetings,” “energy,” and “golf.”</p> <pre data-bbox="381 798 1027 894">enronTC.tf('meetings', enronTexts[1]) enronTC.tf('energy', enronTexts[1]) enronTC.tf('golf', enronTexts[1])</pre> <p data-bbox="381 932 971 963">Write down the term frequencies of these words.</p> <p data-bbox="381 1001 1320 1033">Term frequency of “meeting:” _____</p> <p data-bbox="381 1071 1304 1102">Term frequency of “energy:” _____</p> <p data-bbox="381 1140 1266 1171">Term frequency of “golf:” _____</p> <p data-bbox="381 1209 1390 1241">Which word of the four has the highest term frequency in the 2nd email? _____</p>

Step	Action
4	<p data-bbox="289 289 787 321"><u>Compute Inverse Document Frequencies</u></p> <p data-bbox="337 352 1123 384">a. Compute the inverse document frequency of word “business:”</p> <pre data-bbox="381 420 782 451">enronTC.idf('business')</pre> <p data-bbox="381 487 1370 518">What is the inverse document frequency of this word? _____</p> <p data-bbox="337 554 1393 627">b. Similarly, compute the inverse document frequencies of words “meetings,” “energy,” and “golf.”</p> <pre data-bbox="381 663 782 758">enronTC.idf('meetings') enronTC.idf('energy') enronTC.idf('golf')</pre> <p data-bbox="381 793 1127 825">Write down the inverse document frequencies of these words.</p> <p data-bbox="381 861 1401 892">Inverse document frequency of “meeting:” _____</p> <p data-bbox="381 928 1401 959">Inverse document frequency of “energy:” _____</p> <p data-bbox="381 995 1401 1026">Inverse document frequency of “golf:” _____</p> <p data-bbox="381 1062 1401 1094">Which word of the four has the highest inverse document frequency? _____</p>

Step	Action
5	<p><u>Compute Term Frequency-Inverse Document Frequency</u></p> <p>a. Compute the TFIDF of word “business” over the 2nd email</p> <pre data-bbox="381 422 1097 453">enronTC.tf_idf('business', enronTexts[1])</pre> <p>What is the TFIDF of this word? _____</p> <p>b. Similarly, compute the TFIDF of words “meetings,” “energy,” and “golf.”</p> <pre data-bbox="381 627 1097 722">enronTC.tf_idf('meetings', enronTexts[1]) enronTC.tf_idf('energy', enronTexts[1]) enronTC.tf_idf('golf', enronTexts[1])</pre> <p>Write down the TFIDF of these words.</p> <p>TFIDF of “meeting:” _____</p> <p>TFIDF of “energy:” _____</p> <p>TFIDF of “golf:” _____</p> <p>Which word of the four has the highest TFIDF score over the 2nd email? _____</p>

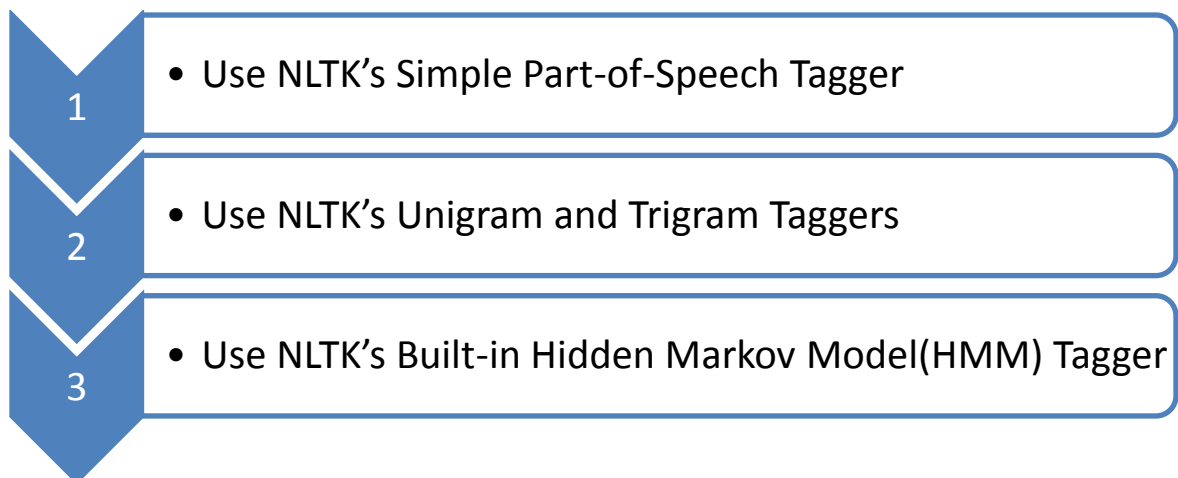
End of Lab Exercises

This PAGE intentionally left blank.

Lab Exercise 9: Part-of-Speech Tagging

Purpose:	This lab introduces part-of-speech (POS) tagging using NLTK's built-in functions.
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Perform POS tagging with a:<ul style="list-style-type: none">○ unigram tagger○ trigram tagger○ HMM tagger• Compare the performances of these taggers with measures such as accuracy and confusion matrix
References:	<ul style="list-style-type: none">• http://nltk.org• http://www.nltk.org/book/• http://www.americannationalcorpus.org/oanc/penn.html

Workflow Overview



Lab Instructions

Step	Action
1	<p>Use NLTK's Simple Part-of-Speech Tagger</p> <p>Note: All of the following Python code can be found in the file <code>~/Solutions/lab9/PosTagging_solution.py</code>.</p> <p>A part-of-speech tagger, or POS tagger, processes a sequence of words and attaches a part of speech tag to each word.</p> <ol style="list-style-type: none">Import <code>nltk</code> by entering the following command line: <pre>>>> import nltk</pre>NLTK's built-in <code>pos_tag()</code> function uses naïve Bayes classifier to perform part-of-speech tagging on a given list of tokens. The following command imports <code>pos_tag</code> and <code>word_tokenize</code>: <pre>>>> from nltk import pos_tag, word_tokenize</pre>The following commands tokenize a sentence into words and perform POS tagging on these words: <pre>>>> tokens = nltk.word_tokenize("They became what they beheld.") >>> nltk.pos_tag(tokens)</pre><p>What is the tag following the word "beheld"? _____</p>Make up a sentence of your own and run the POS-tagger on this sentence.Find out the difference between "VBD" and "VBP" by entering the following command lines: <pre>>>> nltk.help.upenn_tagset('VBD') >>> nltk.help.upenn_tagset('VBP')</pre> <p>Note: You can find the UPenn POS tagset in the reference.</p>

Step	Action
2	<p data-bbox="289 283 792 319"><u>Use NLTK's Unigram and Trigram Taggers</u></p> <p data-bbox="289 352 1429 430">The Brown corpus is the first million-word electronic corpus of English. This corpus contains text from 500 sources, which have been categorized by genre, such as <i>news</i>, <i>editorial</i> and so on.</p> <p data-bbox="337 466 974 499">a. Import the Brown corpus and the unigram tagger:</p> <pre data-bbox="381 529 1047 604">>>> from nltk.corpus import brown >>> from nltk.tag import UnigramTagger</pre> <p data-bbox="337 646 1230 680">b. Prepare the news category in the Brown corpus for training and testing:</p> <pre data-bbox="381 709 1291 823">>>> brown_news_tagged = brown.tagged_sents(categories='news') >>> brown_news_text = brown.sents(categories='news')</pre> <p data-bbox="337 865 1279 898">c. Train the unigram tagger on the first 4000 documents in the news category:</p> <pre data-bbox="381 928 1377 961">>>> tagger = nltk.UnigramTagger(brown_news_tagged[:4000])</pre> <p data-bbox="337 1003 1214 1037">d. Test the trained tagger on the 4,001st document in the Brown corpus.</p> <pre data-bbox="381 1066 1026 1100">>>> tagger.tag(brown_news_text[4000])</pre> <p data-bbox="337 1150 1344 1264">Note: The index in Python starts from 0, not 1. So <code>brown_news_tagged[:4000]</code> retrieves the first 4,000 documents (indexed from 0 to 3999), and <code>brown_news_text[4000]</code> returns the 4,001st document (at index 4000).</p> <p data-bbox="337 1297 1279 1331">Note: Words that the tagger has not seen during training receive a tag of <i>None</i>.</p> <p data-bbox="337 1365 1351 1442">e. You can view the list of categories in the Brown corpus by executing the following command lines.</p> <pre data-bbox="381 1472 766 1505">>>> brown.categories()</pre> <p data-bbox="289 1539 1058 1575">Try the unigram tagger on a different category instead of “news.”</p>

Step	Action
2 Cont.	<p>The Trigram tagger is similar to the unigram tagger, except that it finds the most likely tag for each word based on the current word, and the previous two tags. It is called a "trigram" tagger because it uses these three pieces of information. When training, it can look up the preceding two tags directly. When run on new data, it works through the sentence from left to right and uses the two tags that it generated for the preceding words.</p> <p>f. Train the trigram tagger on the first 4000 documents in the news category:</p> <pre data-bbox="383 583 1382 615">>>> tagger = nltk.TrigramTagger(brown_news_tagged[:4000])</pre> <p>g. Test the trained trigram tagger on the 4,001st document in the Brown corpus.</p> <pre data-bbox="383 716 1029 747">>>> tagger.tag(brown_news_text[4000])</pre> <p>Review the results of both UnigramTagger and TrigramTagger. Explain why there are more NONE tags appearing in the TrigramTagger than in the UnigramTagger.</p> <hr data-bbox="383 953 1429 957"/> <hr data-bbox="383 1020 1429 1024"/> <hr data-bbox="383 1089 1429 1094"/> <p>h. Next, we will split the Brown corpus "news" category data into 80/20 training/testing set:</p> <pre data-bbox="383 1236 1101 1367">>>> totallen = len(brown_news_tagged) >>> sep = int(0.8*totallen) >>> brown_train = brown_news_tagged[:sep] >>> brown_test = brown_news_tagged[sep:]</pre> <p>i. The UnigramTagger class provides the evaluate() function to compute the accuracy. Execute the following commands to train the unigram tagger over the training set and evaluate over the testing set:</p> <pre data-bbox="383 1562 1398 1625">>>> unigram_tagger = nltk.UnigramTagger(brown_train) >>> print 'Accuracy:', unigram_tagger.evaluate(brown_test)</pre> <p>What is the accuracy? _____</p>

Step	Action
2 Cont.	<p>j. Train and evaluate the trigram tagger:</p> <pre data-bbox="381 352 1380 415">>>> trigramTagger = nltk.TrigramTagger(brown_train) >>> print 'Accuracy:', trigramTagger.evaluate(brown_test)</pre> <p>What is the accuracy? _____</p> <p>Compare the performance of unigram and trigram taggers. Which one has a higher accuracy? What could be the reason?</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>k. The following code defines the displayConfusionMatrix() function. This function runs a tagger on a given test set of sentences, compares its results with the gold standard of the tagged sentences, and displays the confusion matrix of the top 10 corrected classified tags.</p> <pre data-bbox="381 1054 1364 1579">from collections import deque from nltk.metrics import ConfusionMatrix from itertools import izip def displayConfusionMatrix(tagger, sents, tagged_sents, n=10, sort_by_count=True): ref = deque() tagged = deque() for sent, gold_tagged_sent in izip(sents, tagged_sents): result_tagged_sent = tagger.tag(sent) for r_word_pos, g_word_pos in izip(result_tagged_sent, gold_tagged_sent): ref.append(g_word_pos[1]) tagged.append(r_word_pos[1]) cm = ConfusionMatrix(list(ref), list(tagged)) print cm.pp(truncate=n, sort_by_count=sort_by_count)</pre>

Step	Action
2 Cont.	<p data-bbox="337 289 1274 321">l. Use the following code to display a confusion matrix of the unigram tagger.</p> <pre data-bbox="383 352 1198 449">>>> brown_test_untagged = brown_news_text[sep:] >>> displayConfusionMatrix(unigram_tagger, brown_test_untagged, brown_test)</pre> <p data-bbox="289 520 1256 552">How many times tag NN is correctly classified as NN? _____</p> <p data-bbox="289 588 1256 619">How many times tag NN is misclassified as JJ? _____</p> <p data-bbox="289 655 946 686">Write down the top 10 tags that are correctly classified.</p> <p data-bbox="289 743 1364 890"> _____ _____ _____ </p> <p data-bbox="337 926 1263 957">m. Use the following code to display a confusion matrix of the trigram tagger.</p> <pre data-bbox="383 989 1094 1056">>>> displayConfusionMatrix(trigramTagger, brown_test_untagged, brown_test)</pre> <p data-bbox="289 1127 1256 1159">How many times tag NN is correctly classified as NN? _____</p> <p data-bbox="289 1194 1256 1226">How many times tag NN is misclassified as JJ? _____</p> <p data-bbox="289 1262 946 1293">Write down the top 10 tags that are correctly classified.</p> <p data-bbox="289 1350 1364 1497"> _____ _____ _____ </p> <p data-bbox="337 1533 1429 1602">n. Split the “news” category of the Brown corpus into 90/10 training/test data. Compute the accuracy and display the confusion matrix for both the unigram and trigram taggers.</p>

Step	Action
3	<p data-bbox="289 283 974 315"><u>Use NLTK's Built-in Hidden Markov Model(HMM) Tagger</u></p> <p data-bbox="289 352 1372 420">The HMM tagger uses a hidden Markov model to find the most likely tag sequence for each sentence.</p> <p data-bbox="337 462 1015 493">a. Import the HMM tagger and the Lidstone smoothing:</p> <pre data-bbox="381 525 1312 588">>>> from nltk.tag.hmm import HiddenMarkovModelTrainer >>> from nltk.probability import LidstoneProbDist</pre> <p data-bbox="381 634 1425 829">The Lidstone smoothing is parameterized by a real number gamma, ranging from 0 to 1. The Lidstone estimate approximates the probability of a sample with count c from an experiment with N outcomes and B bins as $(c + \text{gamma}) / (N + B \cdot \text{gamma})$. This is equivalent to adding gamma to the count for each bin, and taking the maximum likelihood estimate of the resulting frequency distribution.</p> <p data-bbox="337 877 1364 993">b. We will reuse the training and testing set from the previous step for an 80/20 split. Execute the following command lines to process the data in order to meet HMM tagger's input format:</p> <pre data-bbox="381 1029 977 1255">>>> tag_set = set() >>> symbol_set = set() >>> for sentence in brown_train: ... for word, tag in sentence: ... symbol_set.add(word) ... tag_set.add(tag)</pre> <p data-bbox="337 1291 1128 1323">c. Create the HMM tagger by executing the following commands:</p> <pre data-bbox="381 1360 1412 1486">>>> trainer = HiddenMarkovModelTrainer(list(tag_set), list(symbol_set)) >>> hmm_tagger = trainer.train_supervised(brown_train, estimator=lambda fd, bins: LidstoneProbDist(fd, 0.1, bins))</pre> <p data-bbox="337 1528 1237 1560">d. Compute HMM tagger's accuracy by executing the following command:</p> <pre data-bbox="381 1596 1328 1621">>>> print 'Accuracy:', hmm_tagger.evaluate(brown_test)</pre> <p data-bbox="289 1696 1263 1728">What is the accuracy? _____</p>

<p>3 Cont.</p>	<p>e. Call the <code>displayConfusionMatrix()</code> function from the previous step to display a confusion matrix of the HMM tagger.</p> <pre>>>> displayConfusionMatrix(hmm_tagger, brown_test_untagged, brown_test)</pre> <p>How many times tag NN is correctly classified as NN? _____</p> <p>How many times tag NN is misclassified as JJ? _____</p> <p>Write down the top 10 tags that are correctly classified.</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>Compare performances of the unigram, trigram and HMM taggers. Which one performs the best? Explain.</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>_____</p> <p>f. Split the “news” category of the Brown corpus into 90/10 training/test data. Compute the accuracy and display the confusion matrix for the HMM tagger.</p> <p>Note: It may take a while to compute the result. Be patient.</p>
--------------------	---

End of Lab Exercise

This PAGE intentionally left blank.


Lab Exercise 10: Visualizing Social Network Data

Purpose:	<p>This lab introduces you to social network analysis using Gephi. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Import datasets into Gephi• Layout the network• Find communities• Calculate metrics of the network• Export the graph
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Import the Facebook Dataset into Gephi• Understand the Gephi Environment• Layout the graph• Reset the properties of the Layout• Reset the properties of the nodes• Find communities in this dataset with Gephi• Calculate metrics with Gephi• Filter the graph• Preview the graph• Save the project
References:	<ul style="list-style-type: none">• Gephi: http://gephi.github.com

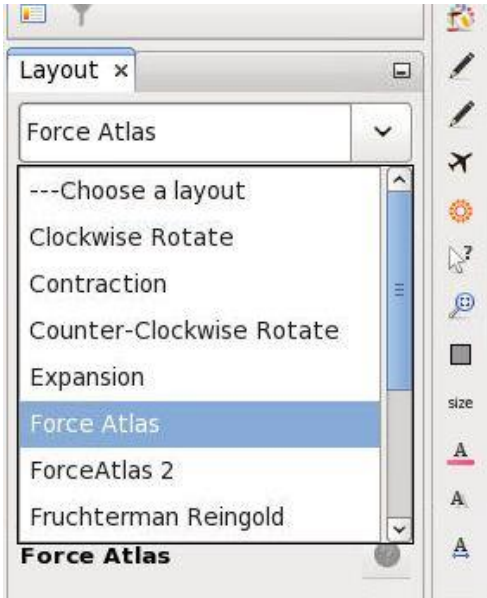
Workflow Overview

- 1 • Import the Facebook Dataset into Gephi
- 2 • Explore the Gephi Interface
- 3 • Layout the Graph
- 4 • Configure the Layout Properties
- 5 • Configure the Node Properties
- 6 • Find Communities in the Dataset
7. • Calculate Metrics with Gephi
- 8 • Filter the Graph
- 9 • Preview the Graph
- 10 • Export the Graph and Save the Project

Lab Instructions




Step	Action
1	<p><u>Import the Facebook Dataset into Gephi</u></p> <p>During this lab, we'll use a Facebook dataset in the GML format. In this file, each node represents a person in the Facebook network and each edge connects two persons if they are friends.</p> <ol style="list-style-type: none"> Double-click on the Gephi icon on the desktop Click "File" → "Open" in the Gephi menu bar, and open the fbdata.gml file located in the directory ~/Labs/lab10. <p>The import report window will display.</p> <p>How many nodes do you see in this network? _____</p> <p>How many edges? _____</p> <ol style="list-style-type: none"> Select the Graph Type as Undirected. Click on the OK button to view the generated graph. <p>Note: If you do not see the created graph, go to the menu bar, click on the Window tab and select graph.</p>
2	<p><u>Explore the Gephi Interface</u></p> <p>Note: You can skip this step if you are familiar with the Gephi GUI.</p> <p>In Gephi, there are three interfaces that you can switch between via the three tabs at the top of the screen:</p> 

Step	Action
2 Cont.	<p>By default we are in the Overview tab. This is the primary tab where you can manipulate, calculate statistics about, and run different functions on your graph.</p> <ul style="list-style-type: none"> In the top-left box (left image below), you can rank and partition the nodes and edges on the graph. The bottom-left box (middle image below) allows you to display different layouts of your graph. The bottom-right box (right image below) allows you to calculate statistics about and run filters on the data in your graph. The results of these statistics appear as a new column under the Data Laboratory tab. <div data-bbox="290 669 1421 1163"> </div> <p>a. Click on the Data Laboratory tab. The Data Laboratory tab is where you can view and manipulate the actual data that the graph is representing:</p> <div data-bbox="362 1276 1346 1740"> </div> <p>This interface allows you to import data, export data, search for certain records, add columns, merge columns, add edges, add nodes, and much more.</p>




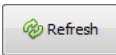
Step	Action
2 Cont.	<p>b. Switch to the Preview tab. Click the Refresh button on the bottom left to preview your graph.</p> <p>The Preview settings tab on the left side of the window allows you to modify settings for nodes, node labels, edges, edge arrows, and edge labels. It also allows you to export the graph as a PDF, PNG, or SVG image.</p>
3	<p><u>Layout the Graph</u></p> <p>The layout algorithm sets the graph shape.</p> <ol style="list-style-type: none"> Select the Overview tab Locate the Layout window on the left panel. Select Force Atlas and leave the default properties unchanged. Click the Run button. When you see that the layout of the graph stops changing, click the Stop button. You may select other layout algorithms instead of Force Atlas and review the results.  <p>The screenshot shows a software window titled 'Layout x'. It features a dropdown menu currently displaying 'Force Atlas'. Below the dropdown, a list of layout algorithms is visible: '---Choose a layout', 'Clockwise Rotate', 'Contraction', 'Counter-Clockwise Rotate', 'Expansion', 'Force Atlas' (which is highlighted with a blue background), 'ForceAtlas 2', and 'Fruchterman Reingold'. To the right of the list is a vertical scrollbar. Below the list, the text 'Force Atlas' is displayed again. On the far right, a vertical toolbar contains several icons, including a pencil, an eraser, a selection tool, a zoom tool, and a 'size' label.</p>

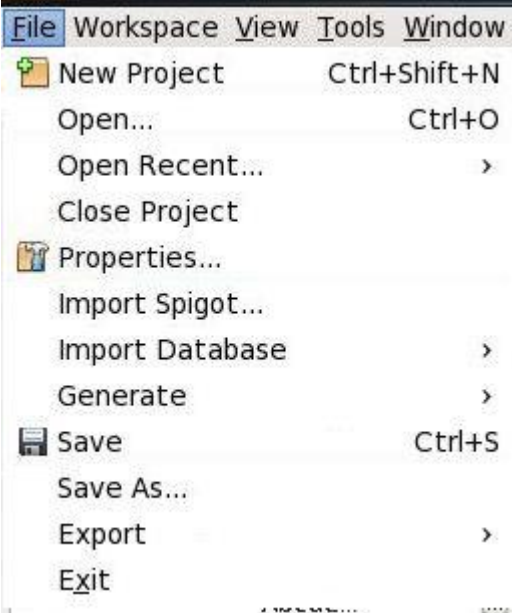
Step	Action
4	<p data-bbox="282 285 672 317"><u>Configure the Layout Properties</u></p> <p data-bbox="282 352 1386 426">When a layout algorithm (such as Force Atlas) is chosen, you may customize its layout Properties to make an aesthetically pleasing representation. In the Properties of Force Atlas,</p> <ol data-bbox="331 462 1403 657" style="list-style-type: none"> Change the “Repulsion strength” from 200.0 to 1000.0 to expand the graph and click Run. Change the “Repulsion strength” from 1000.0 to 10000.0. Click Run to view the result. You should see an expanded graph. Set an appropriate “Repulsion strength” value to help you to view the graph. <p data-bbox="282 735 1099 766">Note: You could use your mouse to move and scale the visualization.</p> <ul data-bbox="360 802 846 919" style="list-style-type: none"> • Zoom: Mouse wheel • Pan: Right click drag and drop • Move nodes: Left click drag and drop

Step	Action
5	<p data-bbox="282 285 656 317"><u>Configure the Node Properties</u></p> <p data-bbox="282 352 1382 426">The Ranking panel lets you configure colors and sizes of the nodes to help identify them and provide a better view of the graph.</p> <ol data-bbox="331 464 1192 657" style="list-style-type: none"> Select the Ranking window near the top left of the Gephi GUI. Select the Degree as the rank parameter. Click on the Apply button. Move your mouse over the gradient component on the color bar. Left-click to create a triangle that defines new colors for your nodes. <p data-bbox="378 667 1118 699">Note: To remove a triangle from the color bar, drag it upward.</p> <div data-bbox="544 798 1166 1104" data-label="Image"> <p>The screenshot shows the 'Ranking' panel in Gephi. It features a 'Color' section with a horizontal gradient bar transitioning from yellow to red. Below the bar are several black triangles representing node color mappings. A 'Range' slider is positioned below the color bar, with numerical markers at 0 and 64. At the bottom of the panel, there is a 'Spline...' button, an infinity symbol, and an 'Apply' button with a green play icon.</p> </div> <ol data-bbox="331 1142 1250 1215" style="list-style-type: none"> Click the Apply button again to view the change to the graph. Select the diamond icon in the toolbar to map node sizes to their Degree. <div data-bbox="532 1285 1175 1698" data-label="Image"> <p>The screenshot shows the 'Nodes' panel in Gephi. At the top, there are tabs for 'Nodes' and 'Edges', along with several icons including a color wheel, a red heart, and two 'A' icons. A dropdown menu is set to 'Degree'. Below this, there are input fields for 'Min size' (set to 1) and 'Max size' (set to 16), each with up and down arrow buttons. A 'Range' slider is located below these fields, with markers at 0 and 64. At the bottom, there is a 'Spline...' button, an infinity symbol, and an 'Apply' button with a green play icon.</p> </div> <ol data-bbox="331 1736 1105 1768" style="list-style-type: none"> Set the Min size to 3 and the Max size to 20, then click Apply.

Step	Action
5 Cont.	<p>i. Display the ranking value of each node by clicking on the Result List button  located at the bottom left of the Ranking window, then click Apply.</p> <p>List the first and third most connected node in the graph.</p> <p>_____</p> <p>How many links does Dunstan have? _____</p>
6	<p><u>Find Communities in the Dataset</u></p> <p>Next, we will use Gephi 's built-in algorithms to find and colorize communities.</p> <ol style="list-style-type: none"> Locate the Statistics window on your right panel. Click the Run button that corresponds to modularity. Leave the default parameters unchanged and click the OK button. View the Modularity Report in the pop-up, and click Close. <p>What is the modularity of this network? _____</p> <p>How many communities do you see? _____</p> <ol style="list-style-type: none"> Locate the Partition window on your top left panel. The Partition window allows you to colorize communities using parameters such as modularity. Click the Refresh button . Select the Modularity class as the partition parameter and click Apply. <p>What is the value that corresponds to the largest community? _____</p> <ol style="list-style-type: none"> Click the Show Pie button  and write down the three largest communities' id number. <ol style="list-style-type: none"> _____ _____ _____

Step	Action										
7	<p><u>Calculate Metrics with Gephi</u></p> <p>The Statistic module can calculate some interesting values. Locate the Statistic module on your right panel:</p> <p>a. Click on the corresponding Run buttons in order to answer the following questions.</p> <p>What is the Network Diameter value? _____</p> <p>What is the Average Degree of the graph? _____</p> <p>What is the Graph Density value? _____</p> <p>b. Click on the Run button that corresponds to “PageRank” (do not change default parameters). Review the resulting PageRank distribution.</p> <p>c. Locate the Ranking window on your top left panel, select the PageRank as the rank parameter and click the Apply button.</p> <p>List the top three nodes with the highest ranking value and find the location of these three nodes.</p> <p>1. _____</p> <p>2. _____</p> <p>3. _____</p> <p>Note: Metrics Available:</p> <table border="0"> <tr> <td>- Average Degree</td><td>- Network Diameter</td></tr> <tr> <td>- Graph Density</td><td>- HITS</td></tr> <tr> <td>- Community Detection (Modularity)</td><td>- PageRank</td></tr> <tr> <td>- Connected Component</td><td>- Clustering Coefficient</td></tr> <tr> <td>- Eigenvector Centrality</td><td>- Path Length</td></tr> </table>	- Average Degree	- Network Diameter	- Graph Density	- HITS	- Community Detection (Modularity)	- PageRank	- Connected Component	- Clustering Coefficient	- Eigenvector Centrality	- Path Length
- Average Degree	- Network Diameter										
- Graph Density	- HITS										
- Community Detection (Modularity)	- PageRank										
- Connected Component	- Clustering Coefficient										
- Eigenvector Centrality	- Path Length										

Step	Action
8	<p><u>Filter the Graph</u></p> <p>We can create filters to hide certain nodes and edges on the network.</p> <ol style="list-style-type: none"> Locate the Filters module on your right panel. Find Degree Range under the Topology category.  <ol style="list-style-type: none"> Double click on the item Degree Range. A range slider and the chart that represents the data will show at the bottom right of the panel. Move the slider to set its lower bound to 8.  <ol style="list-style-type: none"> Click the Filter button. <p>Nodes with a degree less than 8 are now hidden. How many nodes and edges do you see in the current graph? _____</p>
9	<p><u>Preview the Graph</u></p> <ol style="list-style-type: none"> Select the Preview tab on the top.  <ol style="list-style-type: none"> Click the Refresh button  on the bottom left. A preview graph will be displayed in the Preview window on the right. In the preset settings under the Node Labels, check the Show Labels box. Click the Refresh button again to update the graph. In the preset settings, experiment with different configurations of the graph.

Step	Action
10	<p><u>Export the Graph and Save the Project</u></p> <ol style="list-style-type: none"> Go to File -> Export and select the SVG/PDF/PNG file. Save the graph as a PDF format. You can also save the entire project into a .gephi file by clicking File -> Save.  <p>The screenshot shows the 'File' menu in the Gephi application. The menu items are: New Project (Ctrl+Shift+N), Open... (Ctrl+O), Open Recent... (with a submenu arrow), Close Project, Properties... (with a folder icon), Import Spigot..., Import Database (with a submenu arrow), Generate (with a submenu arrow), Save (Ctrl+S), Save As..., Export (with a submenu arrow), and Exit. The 'File' menu is highlighted with a blue border.</p>

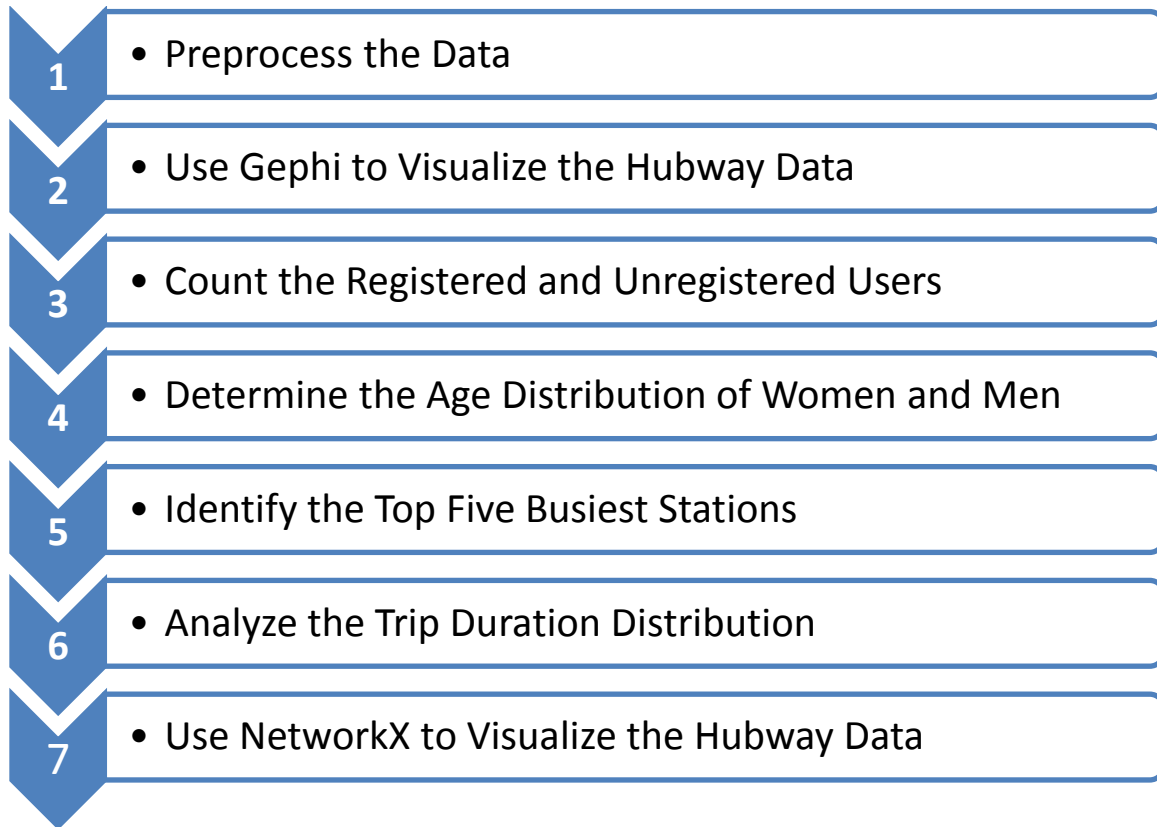
End of Lab Exercise

This PAGE intentionally left blank.

Lab Exercise 11: Analyzing the Hubway Data

Purpose:	<p>This lab introduces you to applying social network analysis to a real-world dataset, with tools such as Python, Gephi and NetworkX. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Learn how to preprocess a dataset• Visualize the dataset in Gephi• Write Python codes to compute various metrics• Use NetworkX to visualize the dataset
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Preprocess the Hubway data• Use Gephi to visualize the Hubway data• Compute the registered and unregistered users• Compute the age distribution among women and men• Compute the top five hottest stations• Analyze the distribution of duration for each trip• Use NetworkX to visualize the Hubway data
References:	<ul style="list-style-type: none">• Gephi: http://gephi.github.com• NetworkX: http://networkx.lanl.gov/• Hubway: http://www.thehubway.com/• Hubway Data Challenge: http://hubwaydatachallenge.org/

Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Preprocess the Data</u></p> <p>Hubway is a bicycle sharing system in the city of Boston, Massachusetts, USA. The Hubway dataset was obtained from the Hubway Data Challenge website (http://hubwaydatachallenge.org/). The dataset is stored in CSV files. We will use the Python CSV module to process the data.</p> <p>a. In a terminal window, change the directory and enter the Python environment:</p> <pre>\$ cd ~/Labs/lab11 \$ python</pre> <p>b. Import the CSV module by entering the following command:</p> <pre>>>> import csv</pre> <p>Note: All of the following Python code can be found in the file <code>~/Solutions/lab11/hubway_solution.py</code>.</p> <p>c. Create the CSV file reader:</p> <pre>>>> reader= csv.DictReader(open('trips.csv'), skipinitialspace=True)</pre> <p>d. Print the first row of the trips.csv file by entering the following command:</p> <pre>>>> reader.fieldnames</pre> <p>e. Print the first column of the second row by entering the following commands:</p> <pre>>>> nextrow = reader.next() >>> nextrow['status']</pre> <p>f. In order to use Gephi to visualize this dataset, we need to reformat the dataset to meet Gephi's input requirement. Enter the following command:</p> <pre>>>> import prepare_hubwaydata</pre> <p>How many edges are in this dataset? _____</p> <p>g. Review the newly generated <i>hubwaynetwork.gml</i> file and explain what “weight” value means in the edge parameter.</p>

Step	Action
2	<p><u>Use Gephi to Visualize the Hubway Data</u></p> <p>a. Inside Gephi, import <i>hubwaynetowrk.gml</i> as a directed graph. In this graph, each node is a Hubway station and each edge is a trip made between two stations.</p> <p>How many stations are in this dataset? _____</p> <p>How many edges are in this dataset? _____</p> <p>b. Compute the graph density of this network: _____</p> <p>c. Compute the modularity of this network: _____</p> <p>d. How many communities can be found in the Modularity Report? _____</p> <p>e. List the ID numbers of the two largest communities.</p> <p>1. _____</p> <p>2. _____</p> <p>f. List the three highest ranked stations (use <i>Degree</i> as rank parameter).</p> <p>1. _____</p> <p>2. _____</p> <p>3. _____</p> <p>g. Choose a graph layout.</p> <p>h. Filter the graph.</p> <p>i. Save the project as Hubway.gephi.</p>

Step	Action
3	<p><u>Count the Registered and Unregistered Users</u></p> <p>Users are either registered or unregistered. In this step, we will compute the number of each type of users and plot the results.</p> <p>a. To compute the number of registered users, enter the following commands:</p> <pre data-bbox="370 527 1421 1052">>>> import csv >>> >>> def CountUser(path): ... tripsReader = csv.DictReader(open(path), skipinitialspace=True) ... reg_user_count = 0 ... unreg_user_count = 0 ... for row in tripsReader: ... if row['subscription_type'] == 'Registered': ... reg_user_count += 1 ... else: ... unreg_user_count += 1 ... print 'registered user is %d' %reg_user_count ... >>> TRIPS_FILE = 'trips.csv' >>> CountUser(TRIPS_FILE)</pre> <p>How many registered users are there? _____</p> <p>What is the ratio of registered users to total users? _____</p> <p>b. Change the above code to calculate the number of unregistered users.</p> <p>How many unregistered users are there? _____</p> <p>Note: The total number of users is equal to the sum of registered users and unregistered users.</p> <p>c. Plot the result by entering the following command:</p> <pre data-bbox="381 1541 769 1572">>>> import plotreguser</pre> <p>Review the generated graph. What is the ratio of unregistered users to whole users?</p> <p>_____</p>

Step	Action
4	<p><u>Determine the Age Distribution of Women and Men</u></p> <p>Within the dataset, the users' age has been captured. In this step, we want to know the age distribution among both male and female users.</p> <p>a. Compute the maximum and minimum age within the dataset by entering the following commands:</p> <pre> >>> import csv >>> from datetime import date >>> def CalculateUserAge(path): ... current_year = date.today().year ... min_age = 100 ... max_age = 0 ... tripsReader = csv.DictReader(open(path), skipinitialspace=True) ... for row in tripsReader: ... if row['subscription_type'] == 'Registered' and row['birth_date'] != '': ... age = current_year - int(row['birth_date']) ... if age < min_age: ... min_age = age ... if age > max_age: ... max_age = age ... print "Max age is {0}.\nMin age is {1}.\n".format(max_age, min_age) >>> >>> TRIPS_FILE = 'trips.csv' >>> CalculateUserAge(TRIPS_FILE) </pre> <p>Write down the age of the oldest and youngest person in the dataset.</p> <p>Oldest: _____ Youngest: _____</p> <p>b. Referring to Step 1 and Step 3, change the code in step 3 to calculate the number of male and female users.</p> <p>How many male users are there? _____</p> <p>How many female users are there? _____</p> <p>c. Plot the results by entering the following command:</p> <pre> >>> import plotuser </pre> <p>What is the largest age group? _____</p>

Step	Action
5	<p><u>Identify the Top Five Busiest Stations</u></p> <p>The dataset includes a list of stations. From the data, we want to calculate the number of visits to each station.</p> <p>a. To compute the frequency of visits to each station in the dataset, enter the following commands:</p> <pre> >>> import csv >>> from collections import defaultdict >>> def CalculateVisStation(trips_path, stations_path): ... start_counts = defaultdict(int) ... end_counts = defaultdict(int) ... tripsReader = csv.DictReader(open(trips_path), skipinitialspace=True) ... total_count = 0 ... for row in tripsReader: ... start_station = row['start_station'] ... end_station = row['end_station'] ... if start_station != '' and end_station != '': ... total_count += 1 ... start_counts[start_station] += 1 ... end_counts[end_station] += 1 ... stationsReader = csv.DictReader(open(stations_path), skipinitialspace=True) ... station_start_counts = [] ... station_end_counts = [] ... station_count = 0 ... for row in stationsReader: ... sid = row['id'] ... sname = row['name'] ... if sid != '' and sname != '': ... station_count += 1 ... station_start_counts.append((... start_counts[sid], sname)) ... station_end_counts.append((... end_counts[sid], sname)) ... print 'Station name\t Number of trips starting from\n' ... for tuple in station_start_counts: ... print '{0}\t{1}\n'.format(tuple[1], tuple[0]) >>> >>> TRIPS_FILE = 'trips.csv' >>> STATIONS_FILE = 'stations.csv' >>> CalculateVisStation(TRIPS_FILE, STATIONS_FILE) </pre> <p>Review the printed results and explain the meaning of the numbers after each station.</p>

Step	Action
5 Cont.	<p>List the five busiest stations with the highest visited frequency.</p> <ol style="list-style-type: none"> 1. _____ 2. _____ 3. _____ 4. _____ 5. _____ <p>b. Plot the results by entering the following command:</p> <pre>>>> import plotheadstation</pre>

Step	Action
6	<p data-bbox="289 289 753 319"><u>Analyze the Trip Duration Distribution</u></p> <p data-bbox="289 352 1409 466">There are several stations that passengers travel to and from. Captured in the dataset is the amount of time it took to travel from one station to another. We will calculate the distribution of the duration of each trip.</p> <p data-bbox="347 499 1429 575">a. To calculate the number of trips where the duration is less than 6 hours but larger than 4 hours, execute the following commands:</p> <pre data-bbox="480 604 1393 1138"> >>> import csv >>> >>> def CalculateDurationFreq(path): ... count = 0 ... tripsReader = csv.DictReader(open(path), skipinitialspace=True) ... for row in tripsReader: ... if row['duration'] != '': ... hrs = int(row['duration']) / 60 ... if hrs >= 4 and hrs < 6: ... count += 1 ... print 'The number of trips less than 6 hours but more than 4 hours is: %d ' % count >>> >>> TRIPS_FILE = 'trips.csv' >>> CalculateDurationFreq(TRIPS_FILE) </pre> <p data-bbox="321 1209 1338 1239">b. Compute the number of trips where the duration is between 2 hours and 4 hours.</p> <p data-bbox="321 1348 987 1377">c. Plot the results by entering the following command:</p> <pre data-bbox="370 1423 773 1453"> >>> import plotduration </pre>

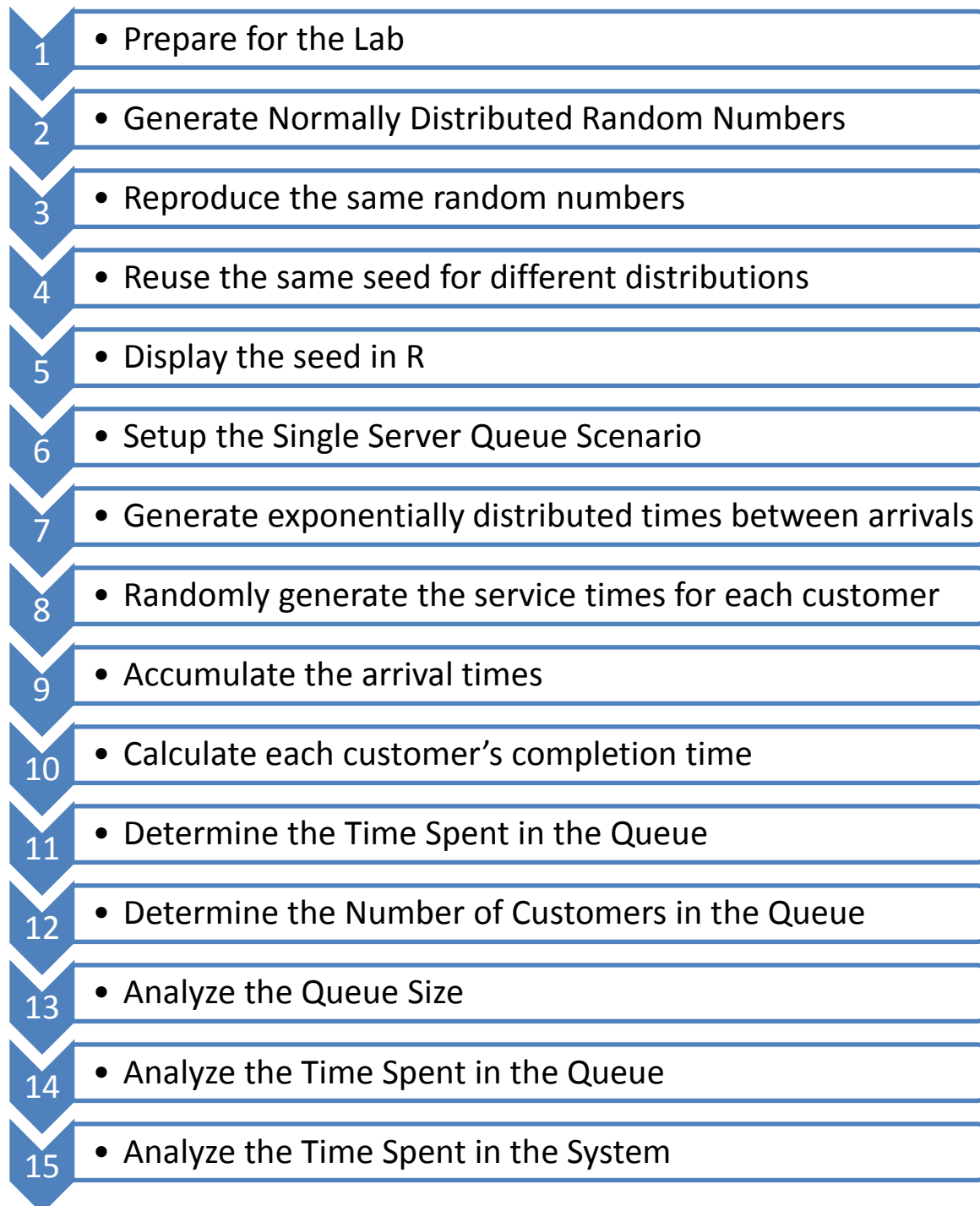
Step	Action
7	<p><u>Use NetworkX to Visualize the Hubway Data</u></p> <p>NetworkX is a Python package for the creation, manipulation and study of the structure, dynamics and functions of complex networks.</p> <p>a. Add all stations, as nodes, into the graph by entering the following commands:</p> <pre data-bbox="380 562 1256 890">>>> import csv >>> import networkx as nx >>> from collections import defaultdict >>> import matplotlib.pyplot as plt >>> from create_networkx_graph import CreateGraph >>> TRIPS_FILE = 'trips.csv' >>> STATIONS_FILE = 'stations.csv' >>> graph = CreateGraph(TRIPS_FILE, STATIONS_FILE) >>> graph.number_of_nodes() >>> graph.number_of_edges()</pre> <p>How many stations are there? _____</p> <p>How many actual trips were made? _____</p> <p>How many distinct routes are there between stations? _____</p> <p>b. Plot the hubway data network by entering the following commands:</p> <pre data-bbox="380 1283 699 1346">>>> nx.draw(graph) >>> plt.show()</pre>

End of Lab Exercise

Lab Exercise 12: Simulation

Purpose:	<p>The lab introduces random number generation in R. After some basic R random number generation commands are illustrated, the simulation of a single server queuing scenario is examined.</p> <ul style="list-style-type: none">• Generate random numbers in R• Understand the use of specifying the seed• Evaluate the simulation results
Tasks:	<p>Tasks you will complete in this lab include:</p> <ul style="list-style-type: none">• Generate normally distributed random numbers• Generate exponentially distributed arrival times• Generate uniformly distributed service times• Simulate the use of an ATM with a queue• Analyze the distributions of the queue size and the time spent in the queue
References:	

Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Prepare for the Lab</u></p> <p>Note: All of the following R code can be found in the file <code>~/Solutions/lab12/simulation.R</code></p> <ol style="list-style-type: none"> Open the Firefox Web Browser. Enter localhost:8787 to access the RStudio login. Enter the provided credentials.
2	<p><u>Generate Normally Distributed Random Numbers</u></p> <ol style="list-style-type: none"> Specify the random number generator seed and generate 10,000 normally distributed random numbers with a mean of 10 and a standard deviation of 5. <pre>set.seed(654655) v1 <- rnorm(10000, 10, 5)</pre> Examine the generated random numbers with a histogram and a normal Q-Q plot. <pre>hist(v1) qqnorm(v1)</pre> <p>Do the random numbers appear normally distributed? _____</p> Using the same mean and standard deviation, but a different seed, generate 10,000 normally distributed random numbers. <pre>set.seed(54635756) v2 <- rnorm(10000, 10, 5)</pre> Examine the relationship between these two sets, v1 and v2, of random values <pre>plot(v1, v2) sum(v1==v2) cor(v1, v2)</pre> <p>What is the relationship between v1 and v2? _____</p>

Step	Action
3	<p><u>Reproduce the same random numbers</u></p> <p>Reusing the same seed, compare the two sets of random numbers generated from the same normal distribution:</p> <pre>set.seed(654655) v3 <- rnorm(10000,10,5) set.seed(654655) v4 <- rnorm(10000,10,5) sum(v3==v4)</pre> <p>Is v4 identical to v3? _____ Why? _____</p>
4	<p><u>Reuse the same seed for different distributions</u></p> <p>a. With the same seed, generate two sets of normally distributed random numbers: one set with a mean of 10 and standard deviation of 5 and the second set with a mean of 0 and standard deviation of 2.</p> <pre>set.seed(654655) v5 <- rnorm(10000,10,5) set.seed(654655) v6 <- rnorm(10000,0,2)</pre> <p>b. Examine the relationship between these two sets, v5 and v6, of random values.</p> <pre>sum(v5==v6) hist(v5) hist(v6) plot(v5,v6) cor(v5,v6)</pre> <p>What is the relationship between v5 and v6? _____</p>
5	<p><u>Display the seed in R</u></p> <p>To display the contents of the current seed, display the contents of:</p> <pre>.Random.seed</pre> <p>Assuming the system defaults have not been changed, the first value of the vector, .Random.seed, is a 403 which denotes the Mersenne Twister random number generator is being used in R. The second value of the vector denotes the current position in the 624 elements of the vector. After the 624 value are used, another 624 values will be generated by the Mersenne Twister algorithm. More information on generating random numbers in R can be obtained by entering ?Random in the RStudio console.</p>

Step	Action
6	<p><u>Setup the Single Server Queue Scenario</u></p> <p>The following steps will simulate a single server queue scenario that may be experienced with an Automated Teller Machine (ATM). Based on assumed distributions of customer arrival times as well as how long customers use the ATM, a simulation is to be conducted to examine how often the ATM is in use, how often a customer has to wait, and so on. In R, set up the inputs to the simulation model.</p> <pre># Time between customer arrivals are exponentially # distributed with the following # customer arrival rate (customer arrivals per hour) arrival_rate = 10 # the time to perform the service is uniformly distributed # between 1 and 6 minutes converted to hours to keep units # aligned with the arrival_rate min_service_time = 1/60 max_service_time = 6/60 # number of customers for the simulation num_of_custs <- 50000</pre> <p>At the beginning of the simulation, the simulation assumes that the ATM is not in use and there is no one in the queue waiting to use the ATM.</p>
7	<p><u>Generate exponentially distributed times between arrivals</u></p> <p>a. Generate the time (in hours) between arrivals for the customers:</p> <pre>set.seed(238947123) arrivals <- rexp(num_of_custs, arrival_rate)</pre> <p>b. View the distribution of the arrival times in minutes:</p> <pre>hist(arrivals*60)</pre>

Step	Action
8	<p><u>Randomly generate the service times for each customer</u></p> <p>a. Generate the uniformly distributed times that each customer uses the ATM.</p> <pre>set.seed(34556463) serv_times <- runif(num_of_custs,min=min_service_time, max=max_service_time)</pre> <p>b. View the distribution of the service times</p> <pre>hist(serv_times)</pre> <p>Do the randomly generated numbers appear uniformly distributed? _____</p> <p>c. Consider this histogram:</p> <pre>hist(serv_times, breaks=((2:12)*.5)/60)</pre> <p>Do the randomly generated numbers now appear uniformly distributed? _____</p> <p>Why do the two histograms provide seemingly different results? _____</p> <p>_____</p>
9	<p><u>Accumulate the arrival times</u></p> <p>Previously, the times between customer arrivals were simulated. In this step, the arrival times are accumulated to determine each customer's arrival time from the start of the simulation at time zero. Using the cumsum() function, run:</p> <pre>cumul_arrivals <- cumsum(arrivals) arrivals[1:5] cumul_arrivals[1:4]</pre> <p>How was the value of cumul_arrivals[3] obtained? _____</p> <p>_____</p>

Step	Action
10	<p><u>Calculate each customer's completion time</u></p> <p>In this step, a customer's completion time is the time since the start of the simulation that the customer finishes using the ATM. A customer's completion time will be determined by the customer's cumulative arrival time, time in the queue, and service time. In R, run:</p> <pre># build a vector to store the completion time of each customer compl_time <- vector(mode="numeric", length = num_of_custs) # Add first customer to the system compl_time[1] <- cumul_arrivals[1] + serv_times[1] for (i in 2:num_of_custs) { compl_time[i] <- max(cumul_arrivals[i], compl_time[i-1]) + serv_times[i] }</pre> <p>The calculation in the For loop is based on that if customer X arrives <u>after</u> the previous customer finishes with the ATM, customer X will have a completion time of customer X's cumulative arrival time plus customer X's service time. If customer X arrives <u>before</u> the previous customer finishes with the ATM, customer X will not start using the ATM until the previous customer finishes with the ATM; so the customer X's completion time will be the previous customer's completion time plus customer X's service time.</p>
11	<p><u>Determine the Time Spent in the Queue</u></p> <p>To calculate the time each customer spent in the queue, run:</p> <pre>time_in_queue <- compl_time - (cumul_arrivals + serv_times)</pre>
12	<p><u>Determine the Number of Customers in the Queue</u></p> <p>To determine the size of the queue at the time each customer arrived, run:</p> <pre>queue_size <- vector(mode="integer", length = num_of_custs) queue_size[1] <- 0 for (i in 2:num_of_custs) { queue_size[i] = sum(compl_time[1:i-1]>cumul_arrivals[i]) }</pre> <p>In other words, count the number of customers who had a completion time after each customer arrived at the ATM.</p> <p>Note: This step may take a few minutes to run.</p>

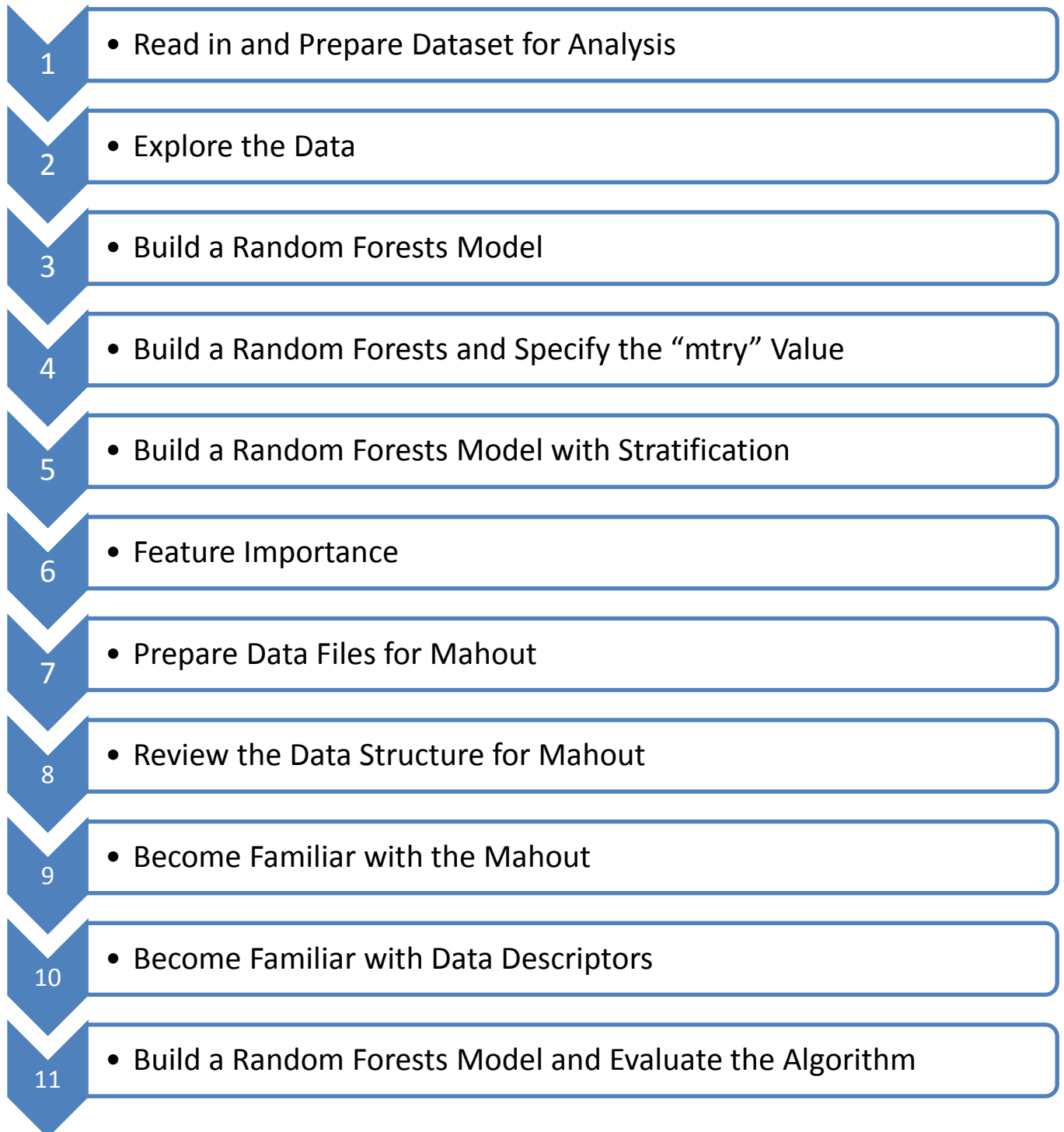
Step	Action
13	<p><u>Analyze the Queue Size</u></p> <p>a. Summarize the size of the queue for the simulated customer:</p> <pre>summary(queue_size)</pre> <p>b. View a histogram of the queue:</p> <pre>hist(queue_size, breaks=0:15, right=FALSE)</pre> <p>c. Determine the probability that a customer will not have to wait in the queue:</p> <pre>sum(queue_size==0)/num_of_custs</pre>
14	<p><u>Analyze the Time Spent in the Queue</u></p> <p>a. Summarize and visualize the time spent in the queue in minutes:</p> <pre>summary(time_in_queue*60) hist(time_in_queue*60)</pre> <p>b. Summarize and visualize the non-zero time spent in the queue in minutes:</p> <pre>summary(time_in_queue[time_in_queue>0]*60) hist(time_in_queue[time_in_queue>0]*60)</pre>
15	<p><u>Analyze the Time Spent in the System</u></p> <p>The term, system, refers to the queue and the ATM. So, the time spent in the system is the time spent in the queue plus using the ATM.</p> <p>a. Summarize and visualize the time spent in the system in minutes:</p> <pre>summary((time_in_queue+serv_times)*60) hist((time_in_queue+serv_times)*60) plot(density((time_in_queue+serv_times)*60))</pre> <p>b. Determine the probability a customer will spend more than 10 minutes in the system:</p> <pre>sum(((time_in_queue+serv_times)*60) > 10)/num_of_custs</pre>

End of Lab Exercise

Lab Exercise 13: Random Forest

Purpose:	<p>This lab is designed to familiarize you and give you practice with Random Forests. After completing the tasks in this lab you should be able to:</p> <ul style="list-style-type: none">• Use R to build a Random Forests model and interpret the results• Use Mahout to build a Random Forests model on data in HDFS and interpret the results
Tasks:	<p>Tasks you will complete in this lab include:</p> <ul style="list-style-type: none">• Use R to train a forest on a dataset, then estimate the OOB error rate of the model• Explore tunable parameters and their impact on the OOB error• Use Random Forests to identify the important features in a dataset• Use Mahout to build a Random Forest model
References:	<ul style="list-style-type: none">• Nursery database reference: https://archive.ics.uci.edu/ml/datasets/Nursery• R randomForest package reference: http://cran.r-project.org/web/packages/randomForest• Mahout API web page: https://builds.apache.org/job/mahout-quality/javadoc/

Random Forests - Workflow Overview



Lab Instructions

Step	Action
1	<p><u>Read in and Prepare Dataset for Analysis</u></p> <p>In this lab, nursery school application data is used to build a Random Forests model to predict the approval assessment assigned to an application. The models are built using R and Apache Mahout.</p> <p>a. The (<i>nursery.data-header.csv</i>) dataset for this lab is located in the ~/Labs/lab13 directory. Read <i>nursery.name.txt</i> in the ~/Labs/lab13 directory for more information about this dataset.</p> <p>Read data into R dataframe</p> <pre>setwd("~/Labs/lab13") nurseryData <- read.csv("./nursery.data-header.csv", as.is=TRUE)</pre> <p>The “randomForests” package in R requires that all categorical features be factors. Convert all features to factors:</p> <pre>nurseryData[sapply(nurseryData, is.character)] <- lapply(nurseryData[sapply(nurseryData, is.character)], as.factor)</pre>
2	<p><u>Explore the Data</u></p> <p>a. The data frame “nurseryData” contains 8 independent variables and 1 dependent variable. Let’s take a brief look at the shape of the data :</p> <pre>summary(nurseryData)</pre> <p>What stands out with regards to the dependent variable “application_rank”?</p> <hr/> <p>The distribution of the classifications within the dependent variable is skewed towards two class labels: “recommend” and “very_recom.” To simplify the analysis these two class labels will be combined into one.</p> <pre>levels(nurseryData\$application_rank)[match("recommend", levels(nurseryData\$application_rank))] <- "very_recom"</pre> <p>Verify the consolidation.</p> <pre>summary(nurseryData)</pre>

Step	Action
3	<p><u>Build a Random Forests Model</u></p> <p>Using R, build a Random Forests model</p> <p>a. Load the required libraries</p> <pre>library(tree) library(caret) library(rpart) library(rpart.plot) library(randomForest)</pre> <p>b. Setting a seed will allow for reproducible results</p> <pre>set.seed(1260)</pre> <p>c. Build a forest of 100 trees with the default number of variables at each split</p> <pre>fit <- randomForest(application_rank ~ ., data=nurseryData, ntree=100)</pre> <p>d. Examine meta-data and performance of the model we just created.</p> <pre>print(fit)</pre> <p>What is the OOB Error Rate? _____</p> <p>What was the # of variables tried at each split? _____</p> <p>e. Examine structure of “fit.”</p> <pre>summary(fit)</pre> <p>Observe that the variable “mtry” is a vector of length 1. This variable is used to specify the number of variables to randomly sample at each split in the tree. The default is the square root of the number of features in the data set.</p> <p>f. Before we define a plotting configuration we need to set the default</p> <pre>old.par <- par(mar=c(0,0,0,0))</pre>

Step	Action
3 Cont'd	<p>g. Plot Out of Bag Error Estimates for model</p> <pre>par(mar=c(4,4,4,4)) plot(fit, main=paste("Error Rate vs. # Trees (mtry =",fit\$mtry,""), type="l", col.main="black",lwd=2,lty=1) legend(60, 0.25, colnames(fit\$serr.rate),col=1:4, cex=0.8, fill=1:5, lwd=1, bty="n")</pre> <p>What would contribute to the error rate of “very_recom” that makes it much worse than the other class labels?</p> <p>_____</p>
4	<p><u>Build a Random Forests and Specify the “mtry” Value</u></p> <p>a. Specify that 3 variables should be considered at each split</p> <pre>set.seed(1260)</pre> <pre>fit2 <- randomForest(application_rank ~ ., data=nurseryData, ntree=100, mtry=3)</pre> <p>b. Examine meta-data and performance of the model we just created.</p> <pre>print(fit2)</pre> <p>What is the OOB Estimate of Error Rate? _____</p> <p>What was the # of variables tried at each split? _____</p> <p>c. Plot the Out of Bag Error Estimates for the model</p> <pre>par(mar=c(4,4,4,4)) plot(fit2, main=paste("Error Rate vs. # Trees (mtry =",fit2\$mtry,""), type="l", col.main="black",lwd=2,lty=1) legend(60, 0.25, colnames(fit2\$serr.rate),col=1:4, cex=0.8, fill=1:5, lwd=1, bty="n")</pre> <p>How did the change to “mtry” impact the OOB error rate?</p> <p>_____</p>

Step	Action
5	<p><u>Build a Random Forests Model with Stratification</u></p> <p>a. Stratify the data to ensure that each class label in “application_ranking” is equally represented in each tree. Here we also introduce the "importance" parameter in the building of the model. This will allow us to look at which features had the biggest influence on the building of the model.</p> <pre>set.seed(1260)</pre> <pre>par(mar=c(4,4,4,4))</pre> <pre>fit3 <- randomForest(application_rank ~ ., data=nurseryData, importance=TRUE, ntree=100, mtry=3, sampsize = c('not_recom' = 4320, 'priority' = 4266, 'very_recom' = 330, 'spec_prior' = 4044))</pre> <pre>print(fit3)</pre> <p>What impact did stratification have on the OOB error rate?</p> <hr/> <p>b. Plot the Out of Bag Error Estimates for model</p> <pre>par(mar=c(4,4,4,4))</pre> <pre>plot(fit3, main=paste("Error Rate vs. # Trees (mtry =", fit3\$mtry, ")", type="l", col.main="black", lwd=2, lty=1)</pre> <pre>legend(60, 0.25, colnames(fit3\$err.rate), col=1:4, cex=0.8, fill=1:5, lwd=1, bty="n")</pre>
6	<p><u>Feature Importance</u></p> <p>a. Explore the “importance” matrix of the Random Forests we just built</p> <pre>fit3\$importance</pre> <p>The column “MeanDecreaseAccuracy” along with “MeanDecreaseGini” are tools to help identify the features that have the biggest overall impact on the creation of the forests.</p>

Step	Action
<p>6</p> <p>Cont'd</p>	<p>b. Create a bar plot of feature importance</p> <p>Type 1 will source "Mean Decrease in Accuracy"</p> <p>Type 2 will source "Mean Decrease in Gini"</p> <pre>fit3.tmp <- importance(fit3, type = 1);</pre> <pre>fit3.imp <- fit3.tmp[order(fit3.tmp[, 1], decreasing = TRUE),]; par(mar = c(6, 6, 6, 6))</pre> <pre>barplot(fit3.imp, col = "lightblue", names.arg = names(fit3.imp), yaxt = "n", cex.names = 1, las=2, ylab="Mean Decrease Accuracy", main="Importance Rank of Predictors (mtry=3), Stratified Data");</pre> <p>The plot demonstrates that “health”, “has_nurs” and “parents” have the biggest overall impact on the building of the forests.</p>

Step	Action
7	<p><u>Prepare Data Files for Mahout</u></p> <p>Using the nursery data, you will now create a Random Forests model using Mahout. The Mahout implementation of Random Forests requires that the data be partitioned into training and testing sets. For this exercise the data has been pre-partitioned for you.</p> <p>a. The training (<i>nursery.test.csv</i>) and testing (<i>nursery.train.csv</i>) datasets for this lab are located in the ~/Labs/lab13 directory. Read <i>nursery.name.txt</i> in the ~/Labs/lab13 directory for more information about this dataset. Put these datasets into the HDFS by executing the following commands:</p> <pre>hdfs dfs -mkdir datasets hdfs dfs -put ~/Labs/lab13/nursery.train.csv datasets hdfs dfs -put ~/Labs/lab13/nursery.test.csv datasets</pre> <p>b. Confirm your execution by issuing the following command:</p> <pre>hdfs dfs -ls datasets</pre>
8	<p><u>Review the Data Structure for Mahout</u></p> <p>There are two kinds of Mahout data types: numerical and categorical. The numerical data can be an integer or a floating point number. The categorical data should be any string without blank space (' ') and comma (',') characters.</p> <p>For each entity of the dataset, the data attributes are divided by only one blank space (' ') or one comma (','), and each data entity is one line in the dataset file. Any missing data can be represented by a dash ('-') character. Do not use a blank space for missing data.</p> <p>The testing dataset should have the same data structure as the training dataset. If the class label attribute of the testing dataset is unknown, replace it with a dash ('-') character.</p>

Step	Action
9	<p><u>Become Familiar with the Mahout</u></p> <p>One way of running Random Forest Mahout jobs is through the terminal. There are three different commands that must be run in order to submit a Random Forest Mahout job:</p> <ul style="list-style-type: none"> • Create data descriptor • Build the model • Test the model <p>We have already created an executable Python script that will run these commands for you. The script is called random_forest.py and it is located in the directory ~/Labs/lab13.</p> <p>Execute the commands below to learn more about this program:</p> <pre>cd ~/Labs/lab13 python random_forest.py -h</pre> <p>What options are available? _____</p> <p>_____</p>
10	<p><u>Become Familiar with Data Descriptors</u></p> <p>The data descriptor is used to describe the data type of the dataset. There are four kinds of data types in the descriptor option string:</p> <p>N: Numerical data C: Categorical data I: Ignored data L: Label</p> <p>Numbers can also be used in the descriptor string, for example “3 N I 2 C L”, which means: the first 3 attributes of the dataset are numerical data, the fourth attribute should be ignored by the algorithm, the fifth and sixth attributes are categorical data and the last attribute is the class label.</p>

Step	Action
11	<p><u>Build a Random Forests Model with Mahout and Evaluate the Algorithm</u></p> <p>Execute the following command from the terminal window to build a forest with 10 trees using 3 variables at each node:</p> <pre>Python random_forest.py -tr datasets/nursery.train.csv -ts datasets/nursery.test.csv -o randomforest -d 8 C L -m 3 -t 10</pre> <p>What is the meaning of the descriptors in the above command?</p> <p>_____</p> <p>What is the accuracy? _____</p>

End of Lab Exercise

Final Lab Exercise on Big Data Analytics

Purpose:	This lab allows students to apply analytical methods and tools to a big data problem with unstructured data.
Tasks:	<p>Tasks you will complete in this lab exercise include:</p> <ul style="list-style-type: none">• Explore a large, provided dataset and prepare the data for analysis• Assess data quality and clean the data when necessary• Conduct model selection as well as code, execute, and score the model• Use Gephi to visualize the data• Create a narrative summary of your findings, using the methods covered in this course
References:	Enron dataset: https://www.cs.cmu.edu/~enron/
Working directory:	The directory <code>/home/gpadmin/Labs/final_lab</code> in your lab environment will be your working directory for the final lab exercise.

Case Study Background and Problem Definition

Scenario	<p>The audit firm FPC would like to investigate a large energy and services corporation to ensure they are adhering to all regulatory laws. Corporation shareholders, along with regulatory enforcement agencies, have become troubled with the company's confusing quarterly financial statements. This has created concern in the financial community and, as a result, FPC has been hired as an independent auditing firm to aid in the investigation of possible wrong-doing in the company.</p> <p>Regulatory agencies have given FPC the initial task of analyzing 126,616 subpoenaed emails from company employees to evaluate if regulatory laws have been deliberately broken. In an effort to relieve FPC from having to read every email, the agencies want FPC to run various text and social network analyses on the corpus of emails, organized by sender. There is a directory for each selected company employee and each directory contains the email files sent by the employee. Each email is a text file that contains the date on which it was sent, the sender, the recipient(s), the subject line, and the body of the email.</p> <p>The agencies and FPC realize that although this initial email analysis may be imprecise, they feel there is much that can be learned from it. They believe that analyzing the emails could provide an overview of the company's structure, activity, and key actors. More specifically, they hope to develop a detailed timeline of company activity, determine who has been communicating with whom, what groups may be responsible for illegal activity, and conclude whether or not the emails contain elements of illicit activity that would warrant further review.</p>
-----------------	---

Issues to Address	<p>A number of issues came up during the kick-off meeting for the project:</p> <ol style="list-style-type: none"> 1. Dealing with 126,616 different email files may be cumbersome and inefficient for certain analytical environments. A preliminary task should be to consolidate all of these emails into one large file that can be easily stored and transferred between different environments. 2. The corpus of emails does not include which group or position each person worked for. Text analyses must be applied to classify each employee into a particular group in the company. 3. Some emails were sent on behalf of an employee by their assistant or secretary. As a result, the assistant's email address appeared on the "From:" line in the email instead of the employee's address. FPC wants to make sure that the proper employee is investigated, not his or her assistant. 4. Some emails might be duplicated in other email files if they were replied to or forwarded. Keeping track of every employee who interacted with a particular email, when they did so, and how frequently the email was forwarded throughout the company would indicate the importance of a particular email. 5. A method needs to be developed that can identify emails that discuss illicit activity. Proper text and sentiment analysis must be run that will flag emails containing certain key phrases like "cover up", "jail", "don't tell", "illegal", "prison", "against the law", etc. 6. It is important to know when certain actions were taken by corporation employees. If any emails are found to indicate illicit activity, the date and time of the email should be recorded. With this information, FPC could build a detailed timeline of when significant actions were taken, which would aid further investigation. 7. The regulatory agencies want FPC to summarize the common lines of communication between employees in the corporation. FPC needs to consider what types of analyses would provide an informative summary of who talks to whom in the company, as well as who are the most influential people in the company.
--------------------------	--

Developing an Analytic Plan	<ol style="list-style-type: none"> 1. What type of environment best suits fast access and the types of analysis you'll want to run? 2. Perform preliminary exploration of the data to evaluate the data quality and determine if data cleaning is necessary. 3. Develop preprocessing scripts you might need to add more structure to the data. Can you consolidate the corpus of 126,616 files into a single file? Are there certain file types you'll need to perform proper analysis? 4. What sort of information will most help the stakeholders - the regulatory agencies - in a potential investigation? Make sure your results are understandable to your stakeholders. 5. Consider which text analysis model and keywords you will use to classify emails as illicit, non-illicit, personal, business, etc. 6. Consider which text analysis model and keywords you will use to classify employees into different groups, for example, "Sales", "Finance", "Research", "Development", "Marketing", etc. 7. Decide how you would present your findings as to whether or not illicit activity is occurring in the company and what groups/people are responsible for it. 8. Consider how you might perform a social network analysis to identify the most influential employees and lines of communication among employees. 9. Decide how you would present your findings from the social network analysis. 10. Determine if you want to present a timeline in your final findings. Can your results paint a chronological picture of illicit activity in the company, including who was involved and when each action occurred? 11. Using training and testing datasets, find ways to examine how robust your models are and tune them accordingly.
--	--

Suggested Workflow

Checkpoint 1	<p>In Lab 6, you have created a Spark job to generate a text file (email_output.txt) from the Enron corpus (https://www.cs.cmu.edu/~enron/). Each row is an email with the following fields tab-delimited:</p> <ul style="list-style-type: none">• Email file path• Message-ID• Timestamp• From• To (all users)• Subject• Body of entire e-mail <p>Copy this email_output.txt file from Lab 6 to your final lab working directory.</p> <p>Categorize each user by functional area (Sales, Finance, etc.). There are many ways to do this. A naïve method is as follows:</p> <ul style="list-style-type: none">• Go through subject line and body to find key terms to categorize the e-mail – Term Frequency Inverse Document Frequency (TFIDF)• Categorize each e-mail, mapping its words to functional areas. For example, you can use the WordNet in NLTK to compute the semantic similarity of each term in the email and a functional area as the other term.• Categorize user based on % of categorized e-mails• Identify the top three influential users in each functional area <p>Categorize each e-mail as illicit or not illicit.</p> <ul style="list-style-type: none">• Come up your own illicit terms• Categorize emails based on the how often these illicit terms appear in them
Checkpoint 2	<p>Make a Social Graph of the users:</p> <ul style="list-style-type: none">• Let each node represent a user• Let the edges represent the strength of the relationships (frequency of e-mails)• Make the graph directed -- each edge indicates the direction of the e-mail• Apply a graph layout algorithm <p style="text-align: center;">< Continued on the next page ></p>

<p>Checkpoint 2</p> <p>Cont.</p>	<p>Identify the influential nodes:</p> <ul style="list-style-type: none"> • Rank nodes with centrality measures such as in-degree, betweenness, closeness, and PageRank • Identify the most influential node with each of the centrality measures <p>Communities:</p> <ul style="list-style-type: none"> • Partition the network into different communities using modularity • Identify the subnetwork where each member node has an in-degree ≥ 50 <p>Apply preview settings if necessary.</p> <p>Export your social graph into PDF or PNG files.</p>
--	--

End of Lab Exercise