Data Science and Big Data Analytics

Lab Guide

Copyright

Revision Date: February 2012
Revision Number: MR-1CP-DSBDA .1.2

Document Revision History

Rev #	File Name	Date
1	DSBDA Integrated Lab Guide 12-01-11 alpha.doc	12-01-11
2	DSBDA Integrated Lab Guide 12-07-11 alpha03.doc (incorporating the review updates by Noelle and Rashmi)	12-07-11
4	DSBDA Integrated Lab Guide 12-16-11 alpha04.doc (incorporating the review updates by Noelle)	12-16-11
5	Lab workflow included for labs 3 to 12 Beta version	1-4-12
6	GA Version	1-18-12
7	Made VILT-related edits. Added a disclaimer that applies to the VILT.	4-9-12

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK.

Lab Exercise 1: Introduction to Data Environment

Purpose:	The first lab introduces the *Analytics Lab Environment* you will be working on throughout the course. After completing the tasks in this lab you should able to: • Authenticate and access the Virtual Machine (VM) assigned to you for all of your lab exercises • Use SQL and Meta commands in PSQL to navigate through the data sets • Create subsets of the *data*, using *table joins and filters* to analyze subsequent lab exercises

Tasks:	Tasks you will complete in this lab exercise include: • Exploring databases and datasets • Using PSQL statements and Meta commands. • Creating subsets of data for use in subsequent lab exercises

References:	References used throughout the labs are located in your ***Student Resource Guide Appendix***. See the Appendix for: • PSQL Commands – Quick Reference • PSQL Meta Commands – Quick Reference • Surviving LINUX – Quick Reference • R – Quick Reference

1.1 Accessing Lab Environment

Note: The lab access is available only in the Instructor Led Training (ILT) program. **For those learning the content via the Video ILT (DVD), the labs are provided only as a reference and are recorded for demonstration purposes only.**

Step	Action
1	**Accessing from a Public ISP:** **1.** From any Internet connection, open a suitable browser (Internet Explorer strongly recommended) at **https://vdc.emc.com** 2. Your user name and password details are provided by your instructor. **Accessing the LAB** 1. Refer to the instruction sheet for the access details to your assigned windows host - called your "Front-End" (fe) host - and follow the instructions to open a desktop session to your "fe". 2. The access to the "Front-End" (fe) of the Lab is now established. 3. Refer to the instruction sheet for the IP address of the "Back-End" (be) that hosts the databases and the RStudio environment 4. RStudio may be accessed through the "safari" browser available as a desktop icon on your "fe" host. For RStudio, direct "safari" browser on "fe" to the URL **http://<be_host_IP_Address>:8787/** 5. Utilities such as "PuTTy", WinSCP and PGadmin III are also available on the "fe" to access and update contents in the "be". Some lab exercises will require that you log in directly on the "be" via a terminal session; you may use "PuTTy" to start a terminal session. 6. Follow the lab guide for additional instructions that may be associated with individual labs.

1.2 Database Environment – Retail Data

Step	Action
1	Log in to the "be" host using "PuTTy" on your "fe" host. Specify "gpadmin" as the username, and the appropriate password (to be supplied by your instructor). Currently you are logged in as **GPADMIN** and you have administrative access to the *Greenplum Database Environment*, in which you will be working. You must first verify if the database up and running. 1. Type: **`gpstate`** 2. Review the output; you should be able to see that the database is active with the following output. Please note that because of the large output size we only show selected lines and that your configuration details may slightly differ from the one below. **`[INFO]:-Starting gpstate with args:`** **`[INFO]:-local Greenplum Version: 'postgres (Greenplum Database) 4.1.1.1 build 1'`** **`[INFO]:-Obtaining Segment details from master...`** **`[INFO]:-Gathering data from segments...`** **`[INFO]:-Greenplum instance status summary`** **`[INFO]:----------------------------------------`** **`[INFO]:-   Master instance                 = Active`** **`[INFO]:-   Master standby                  = No master standby configured`** … **`[INFO]:-   Total primary segments                          = 2`** **`[INFO]:-   Total primary segment valid (at master)         = 2`** **`[INFO]:-   Total primary segment failures (at master)      = 0`** … **`[INFO]:-   Mirrors not configured on this array`** **`[INFO]:-----------------------------------------------------`**

Step	Action
2	Now you're ready to open a PSQL session and check all available databases. Refer to the *PSQL Commands – Quick Reference*, *l*ocated in your ***Student Resource Guide Appendix,*** for the PSQL meta commands. **Note:** PSQL meta commands start with a backslash (\). To review all available meta commands type backslash and question mark (\?). To review all available databases in your environment: 1. Type: **`psql`** This will open a new PSQL session to the default database. 2. Next type: **`\l`** Notice a list of databases and record databases named "training*". **Note:** Another way of listing all available databases (without opening a PSQL session) is to call PSQL executable with parameter (-l): **`psql -l`**
3	**<u>Connect to the training1 database:</u>** 1. At the PSQL prompt type : **`\c training1`** at the OS level prompt type: **`psql training1`** To see the schemas you have in this database: 2. Type: **`\dn`** • You should see "ddemo" schema, listed. • You should also ensure that this schema is included in the search path. 3. Execute your first PSQL command, type: **`SET search_path TO ddemo, public;`** **Note:** PSQL commands are terminated with a semi-colon- ";"

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4</td><td>You can now view the tables in this database.

1. Type: \dt
2. Record the number of tables in the database: _____
3. Locate the table, “customers_dim”.
4. Review the column descriptions for this table:
5. Type: \d+ customers_dim
6. Record the column descriptions, their types and column name(s) by which the table is distributed (aka: the distribution key):

<table><tr><th>Column Descriptions</th><th>Type</th><th>Distribution Key Column(s)</th></tr><tr><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>5</td><td><u>Analyze the gender distribution of the customer base:</u>

1. To locate the number of males and females type:

SELECT gender,count(*) FROM customers_dim GROUP BY gender;

2. Record the number of female customers: _____
3. Record the number of male customers: _____
4. Record the total number of customers: _____</td></tr>
</table>

Step	Action
6	1. Using PSQL, generate a report on the average spending by gender, Type: `SELECT` `  c.gender` `, AVG(o.item_price) AS avg_price` `FROM` `  ddemo.order_lineitems AS o` `JOIN` `  ddemo.customers_dim AS c` `  ON o.customer_id = c.customer_id` `GROUP BY c.gender` `;` **Note:** You can find this code in the LAB01 directory. This script can be executed using the following command from the OS prompt: 2. To exit the PSQL environment, use the following meta command, type: `\q` You are now at the OS prompt. 3. To execute the SQL script type: `psql –d training1 -f lab1p1step6.sql` **Note 1:** In the *psql* command above option "-d" specifies the database name to connect to ("training1"). This is equivalent to specifying *dbname* as the first **non-option argument** on the command line. As a convention we have used the option "-d" throughout this document. However *dbname* can be specified without option "-d" as long as it is the first argument of the *psql* command. **Note 2:** This query may take some time to execute as it is processing a million rows of data. 4. Record the average expenditures by gender: Male : _____ Female:_____

Step	Action
7	Use the script, “lab1p1step7”, with the appropriate modifications to list the top five product categories ordered by men and women.

	Men	***Women***
1		
2		
3		
4		
5		

1.3 Database Environment-Census Data

Step	Action
1	Follow the steps detailed in, Lab 1 - Data Set 1, to connect to and inspect another database “training2”.
2	Record the tables in database (Schema – Public)“training2”
3	Describe the type of data in the database.
4	Record the number of rows in each table.
5	**Data Preparation & Cleanup – 1:** (Scenario) You realize that the Intern who loaded the “housing” data has copied records into the table twice. Each different row is represented by a unique combination of “serialno“ and “state” columns. 1. Execute the following code: `SELECT` `  SUM(c) AS total_records` `, SUM(CASE WHEN c>1 THEN c-1 ELSE 0 END) AS total_dupes` `, COUNT(*) AS total_uniques` `FROM (` `  SELECT` `  COUNT(*) AS c` `  FROM` `  housing` `  GROUP BY` `  serialno` `  , state` `) AS dupes` `;` **Note:** This code is also available at, **/home/gpadmin/LAB01/countdupes.sql**, 2. Record the total number of records in the table: _____ 3. Record the total number of duplicate records: _____ 4. Record the total number of unique records: _____

Step	Action
6	**Data Preparation & Cleanup – 2:** To prepare and clean the data you need to create a "housing_nodupes" table. Make sure that you are in the PSQL environment if you have previously exited to the OS command line. 1. Check to see if a table already exists with the name ("housing_nodupes"). Type `\dt` Note: the command \dt will list all tables in the database. \dt public.* will list all tables in the public schema. 2. If this table already exists execute the following SQL statement: `DROP TABLE IF EXISTS housing_nodupes;` 3. Execute the following SQL statement: `CREATE TABLE housing_nodupes AS` `SELECT DISTINCT ON` `(serialno, state) *` `FROM` `housing` `DISTRIBUTED BY (serialno, state)` `;` **Note:** This code is also available at, **/home/gpadmin/LAB01/lab1p2step6.sql** 4. Repeat the queries in Step 5 (previous step) to ensure that there are no duplicate records in the housing_nodupes table.

Step	Action
7	**Basic Analytics Using the "Housing" Data:** 1. Execute the following SQL statement to calculate correlation between household income and number of rooms: `SELECT` `  corr(hinc, rooms)` `FROM` `  housing_nodupes` `WHERE` `  state = 25` `;` 2. Record your result: 3. Execute the following SQL statement calculate the R-squared of the regression line of household income and number of rooms:: `SELECT` `  regr_r2(hinc, rooms)` `FROM` `  housing_nodupes` `WHERE` `  state = 25` `;` 4. Record your result:

Step	Action
8	**Prepare "Housing" Data for Subsequent Analytic Exercises:** You need to prepare data from the, "housing_nodupes" and "persons" tables, for subsequent analysis with "R" in the next module. 1. Run the following commands and SQL query to move (pipe) the results into a text file **Note:** Use the meta commands to render your output to a file and remove the white spaces (formatting) `\a` `\o lab1_01.txt` `SELECT` `  serialno` `, hinc` `, rooms` `FROM` `  housing_nodupes` `WHERE` `  hinc > 0` `  AND state = 25` `;` **Note:** The SQL query is also available at the following location: /home/gpadmin/LAB01/lab1p2step8.sql 2. Alternatively you can execute the following command from the OS prompt: `psql -d training2 –f lab1p2step8.sql` Now, your data is ready for the lab exercise in the next module. 3. Remove the summary line at the end of the output file lab1_01.txt

Step	Action
9	**Prepare "Persons" Data for Subsequent Analytic Exercises:** Prepare a summary table with the number of people by race and by education level. **Note:** Use the following Races: White, Black, American Indian/Alaska Native, Asian, Hawaiian /Pacific Islander, and Others. `(white) White,` `(black) Black,` `(aian) American_Indian_Alaska_native,` `(asian) Asian,` `(nhpi) Hawaii_pacific_islander,` `(other) Others` Use the following Education Levels:

01. No schooling completed 02. Nursery school to 4th grade 03. 5th grade or 6th grade 04. 7th grade or 8th grade 05. 9th grade	06. 10th grade 07. 11th grade 08. 12th grade, no diploma 09. High school graduate 10. Some college, but less than 1 year	11. One or more years of college, no degree 12. Associate degree 13. Bachelor's degree 14. Master's degree 15. Professional degree 16. Doctorate degree

1. Create a table with columns for Races and rows for Educational Level. (The cells denote the number of "persons" for each category.) Prepare a text file with headers to use in the next module. SQL code necessary for this task is presented below:

```
\a
\o lab1_02.txt
SELECT
  educ AS Education_Level
  , SUM(white) AS White
  , SUM(black) AS Black
  , SUM(aian) AS American_Indian_Alaska_Native
  , SUM(asian) AS Asian
  , SUM(nhpi) AS Hawaii_Pacific_Islander
  , SUM(other) AS Others
FROM
  persons
WHERE
  age > 17
  AND educ > 0
GROUP BY educ
ORDER BY educ
;
```

Step	Action
10	The code in step 9 is also available at the following location: /home/gpadmin/LAB01/lab1p2step9.sql Execute the following command from the OS prompt: **`psql -d training2 –f lab1p2step9.sql`** Remove the last “summary” line as you did in Step 8 and prepare the file “lab1_02.txt” for the lab exercise in the next module.

End of Lab Exercise

Lab Exercise 2: Introduction to R

Purpose:	This lab introduces you to the use of the R statistical package within the Data Science and Big Data Analytics environment. After completing the tasks in this lab you should able to: • Read data sets into R, save them, and examine the contents

Tasks:	Tasks you will complete in this lab include: • Invoke the R environment and examine the R workspace • Read tables created in Lab 1 into the R statistical package • Examine, manipulate and save data sets • Exit the R environment

References:	References used in this lab are located in your ***Student Resource Guide Appendix***. See the Appendix for: • R Commands – Quick Reference • Surviving LINUX – Quick Reference

Step	Action
1	**Invoke the R Environment:** Logon to RStudio environment. 1. The RStudio is accessed through the "safari" browser available as a desktop icon on your "fe" host. 2. To start an RStudio session, start "safari" on the "fe" host and direct it to the URL http://<*IP_Address_of_your_be_host*>:8787/ 3. RStudio access details are as follows: User-id : gpadmin Password : <supplied by your instructor> With a successful login to the back-end, you should see the standard RStudio four-panel display. 1. Verify that you see the following text in the lower left-hand pane: `R version 2.13.1 (2011-04-13)` `Copyright (C) 2011 The R Foundation for Statistical Computing` `ISBN 3-900051-07-0` `Platform: i386-pc-mingw32/i386 (32-bit)` `--- <snip> ---` `Type 'demo()' for some demos, 'help()' for on-line help, or` `'help.start()' for an HTML browser interface to help.` `Type 'q()' to quit R` `>`
2	**Examine the Workspace:** Type the following command into the R command panel, and hit [ENTER] `ls()` You should see the following: `character(0)` **Note:** R is telling you that you have nothing in your workspace.

Step	Action
3	**Getting Familiar with R** 1. Click each tab in each panel. What happens? 2. Type the following commands into the R command panel `help()` `help.start()` `demo()` `demo(graphics)` `Hit esc to exit out of the demo`
4	**Read-in the Lab1 Script** 1. Now, in the script window, open the script called "Module3Lab1.R". (Click on "File", "Open File" and Navigate to directory LAB02 and click on file "Module3Lab1.R".) 2. All the commands we will be executing in this lab are contained in this script. In order to execute a command, do the following: o Position your cursor inside the line that represents the command you wish to execute. o Either click on the "Run" button, or hit [CTRL-Enter]. You can execute many commands at once by selecting a sequence of commands and then issuing the "Run" command. 3. The command will be executed in the command pane. If the command produces graphical output, it will appear in the graphic frame. Note that you can expand this panel by clicking on the "expand window" box. In some instances, this will show more information that has been hidden because of the size of the panel. • The (*Module3Lab1.R*) file is divided into sections. Each section corresponds to a step in this lab. By selecting an individual line or lines, you can click "**Run**…" and the command(s) will be executed in the R panel. 1. On the 1st line in Section 1, put your cursor on the line containing the word ls(). 2. Click **Run.** The ls() command will execute in the command window and show you the contents of your workspace.

Step	Action
5	**Working with R:** Load the .txt files you created in the first lab. Load the first file, **lab1_01.txt** 1. Set the working directory to LAB01 where we have stored the data. On the console window type: `setwd("~/LAB01")` 2. Select the line and press <ctl>**Enter**: `lab1 <- table.read("lab1_01.txt", sep="|", header=TRUE)` • If correct, R will simply return you to the command prompt ("> "). 3. Now load the second .txt file, **lab1_02.txt**, by modifying the command (using the line of code in the RStudio command panel) you just entered. (Use the up/down, left/right arrow buttons to move from and within lines; change each occurrence of "lab1" to "lab2".) The command should read: `lab2 <- read.table("lab1_02.txt", sep="|", header=TRUE)` 4. When you have completed the edits, make sure that your cursor is within the line, press **Enter**. **Note:** R supports copy and paste, as well as up and down arrows for moving to previous commands, left and right arrows to move within/between lines and home/end to move to the beginning or end of a line.
6	**Verify the Contents of the Tables:** It is always a good idea to look at the data to make sure that everything works. You can use the `head()` command to print out the first 6 lines of a table or the, `tail()` command to print out the last 6 lines of the table. 1. Select and run the command: `head(lab1, n=10)` Record the value of the 10th line here: _____ 2. Now do the same for the lab2 table, but use the `tail(lab2, n=10)` command instead. 3. Record the value of the 1st line here: _____

Step	Action
7	**Manipulating Data Tables (data frames) in R:** Examine the contents of the table in more detail. 1.Execute the following command: `summary(lab1)` Ignore the values for the *hinc* and *rooms* columns for now. The *serialnoid* field represents a unique identifier (it's the household identifier) from the Postgres database. You no longer need it and it will interfere with some of the procedures you want to run against this data set, so create a copy of the lab1 table without that column. 2. Select and run: `nlab1 <- lab1[,2:3]` This uses a feature of R that allows us to refer to rows and columns in a dataframe as if they were entries in a matrix. A blank entry in a row or column position means "use all available." This statement says: use all the rows in the table, but only use columns 2 and 3 You could have used the following for the same effect (Note that the following code is not part of the script you can see in the source file *Module3lab1.R)*: `hinc <- lab1$hinc` `rooms <- lab1$rooms` `nlab11 <- data.frame(hinc, rooms)` You're taking advantage of R behavior that names the columns after the name of the variable. You could have used the following for the same effect: `nlab11 <- data.frame(lab1$hinc, lab1$rooms)` `names(nlab11) = c("hinc", "rooms")`

Step	Action
7 Cont.	3. The dim(<table>) has the nice property of telling us how many rows exist in the table. Execute the following commands: **dim(nlab1)** **typeof(nlab1)** **class(nlab1)** Each of these commands tells us something about this particular object. You may not use these often, but they can be useful when R complains that it doesn't like something about the object that you just used.
8	**Continue to Investigate Your Data:** 1. Select and execute the following commands: **summary(nlab1)** **cor(nlab1)** The summary function for data frames prints out summary statistics. 2. Compare the median and the mean. What does it mean if the mean is less than the median? _____ 3. How about the mean greater than the median? _____ 4. Does the min and max value for the quartiles make sense to you? Here again you have a chance to do further cleaning of your data sets, but postpone this until you've finished the next few lessons. 5. How do the values returned by the cor() function differ from the results obtained in lab 1? _____
9	**Save the Data Sets:** 1. Execute the following commands: **rm(lab1)** **lab1 <- nlab1** **save(lab1, lab2, file="Labs.Rdata")** **rm(lab1, lab2)** **ls() # make sure they're not in the workspace**

Step	Action
10	**Examine Your Data:** 1. Experiment with some of the examples used in the lecture portion of this lesson. Using the same selection techniques that you used earlier, run each line in the file. • Some commands don't print their results. If this is the case, type in the value of the variable you created in the command window. If the variable was named "x", you can type "x". You can also type "print(x)" which will do the same thing. 2. Experiment with R functions that identify the class and data type of a particular variable, type: **`typeof(x), class(x), attributes(x), names(x), dim(x)`** 3. Which ones work on which kind of data types? _____ 4. Type these values into the RStudio command panel. _____ 5. Typing all these commands for each variable is tedious. Alternatively, we will write a function *`tellme`* that takes a variable as an argument and performs `typeof, class, names and str` on that variable. Select and run the lines beginning with **`"tellme <- function(x)`** **`{extending`** through the right curly brace. 6. Now execute the following command **`tellme`** You should see the definition of the function that you just entered! This is because R doesn't interpret a plain **tellme** as a function, but rather as an object to be printed out. The default print function for a function is to print its definition. You can try this with any other R function. Type **mean** and inspect the results. 7. Try tellme **`()`** with a series of variables. 8. Which commands actually list something? _____ 9. How might you get the other commands to list their return value? [Hint: try `print()`]
11	**Exit R:** 1. Execute the following command: **`q()`** 2. R will ask you if you want to save your workspace. Answer **`"no".`**

End of Lab Exercise

Lab Exercise 3: Basic Statistics, Visualization, and Hypothesis Tests

Purpose:	The lab introduces you to the analysis of data using the R statistical package within the Data Science and Big Data Analytics environment. After completing the tasks in this lab you should able to: • Perform summary (descriptive) statistics on the data sets • Create basic visualizations using R both to support investigation of the data as well as exploration of the data • Create plot visualizations of the data using a graphics package • Test a hypothesis about the data

Tasks:	Tasks you will complete in this lab include: • Reload data sets into the R statistical package • Perform summary statistics on the data • Remove outliers from the data • Plot the data using R • Plot the data using lattice and ggplot • Test a hypothesis about the data

References:	References used in this lab are located in your ***Student Resource Guide Appendix***. See the Appendix for: • R Commands – Quick Reference • Surviving LINUX – Quick Reference

Part 1 – Basic Statistics and Visualization Using R

Workflow Overview

1. Prepare working environment for the Lab and load data files
2. Obtain summary statistics for Household Income and visualize data
3. Obtain summary statistics for number of rooms and visualize data
4. Remove Outliers
5. Stratify Variable – Household Income and plot the results
6. Plot Histogram and Distributions
7. Compute Correlation between income and number of rooms
8. Create a Boxplot – Distribution of income as a factor of number of rooms
9. Exit R

LAB Instructions

Step	Action
1	**Prepare working environment for the Lab and load data files** 1. Set the working directory to LAB01 where we have stored the data. On the console window type: `setwd("~/LAB01")` 2. In the script window, open the script called "Module3Lab2.R". (Click on "File", "Open File" and Navigate to directory LAB03 and click on file "Module3Lab2.R"). Start R and Read the Data Set Back Into Your Workspace: 3. Execute the following commands from the script window: `options(digits=3)` `options(width=68)` `ls()` `load(file="Labs.Rdata")` `ls()` `rm(lab2)` `ds <- lab1` `colnames(ds) <- c("income", "rooms")`
2	**Obtain summary statistics for Household Income and visualize data:** 1. Execute the following commands from the script window: `summary(ds$income)` `range(ds$income)` `sd(ds$income)` `var(ds$income)` `plot(density(ds$income))  # left skewed` 2. What is the mean? _____ 3. What is the median? _____ 4. What is the standard deviation? _____

Step	Action
3	**Obtain summary statistics for Number of rooms and visualize data:** Execute the following commands from the script window: `summary(ds$rooms)` `range(ds$rooms)` `sd(ds$rooms)` `plot(as.factor(ds$rooms))` What is the mean? What is the median? What is the standard deviation?
4	**Remove Outliers** In a previous lab, you recorded the range of income. You observed that the minimum household income is 4, and the maximum is 1,620,560. 1. Does this make sense to you? Why? *_____ 2. What happens if you throw out the top and bottom 10%? Execute the following line from the script window `(m <- mean(ds$income, trim=0.10) )` 3. How does this compare to the previous mean of this variable? 4. Execute the following commands from the script window: `ds <- subset(ds, ds$income >= 10000 & ds$income < 1000000)` `summary(ds)` `quantile(ds$income, seq(from=0, to=1, length=11))` 5. How do these values vary from the values in the original data set? 6. Do they make more sense? 7. Which data set would you prefer to use? _____ *We might consider the high and low value as outliers, and get rid of them. On the other hand, as we will discover, income is best described via a lognormal distribution, and hence these values are in the extreme ends +- 3 sds from the mean.

Step	Action
5	**Stratify Variable – Household Income and plot the results:** Stratify breaks that occur close to U.S. Guidelines for Poverty, Median Income, Wealth, and Rich (> $250k @ year) 1. Execute the following code (listed under comment heading "step 5" in the script file): `breaks <- c(0, 23000, 52000, 82000, 250000, 999999)` `labels <- c("Poverty", "LowerMid", "UpperMid", "Wealthy", "Rich")` `wealth <- cut(ds$income, breaks, labels)` `# add wealth as a column to ds` `ds <- cbind(ds, wealth)` `# show the 1st few lines.` `head(ds)` 2. Continue to execute the remaining part of the code in Step 5 `wt <- table(wealth)` `percent <- wt/sum(wt)*100` `wt <- rbind(wt, percent)` `wt` `plot(wt)` 3. Take another look at the relationship between wealth and income. Execute the following lines: `# take another look -- wealth by rooms` `nt <- table(wealth, ds$rooms)` `print(nt)` `plot(nt)  # nice mosaic plot` 4. Execute this code from the script file. These lines will remove the variables wealth, breaks and labels, and then save the variables data set and write into a file named "Census.Rdata". `rm(wealth,breaks,labels)` `save(ds, wt, nt, file="Census.Rdata")`

Step	Action
6	**Plot Histogram and Distributions:** Problem: How do you represent income given the range of values? 1. Select and execute the code under Step 6 Histograms and distributions in the script file. **library(MASS)** **with(ds, {** ** hist(income, main="Distribution of Household Income", freq=FALSE)** ** lines(density(income), lty=2, lwd=2)** # line type (lty) 2 is dashed ** xvals = seq(from=min(income), to=max(income), length=100)** ** param = fitdistr(income, "lognormal")** ** lines(xvals, dlnorm(xvals, meanlog=param$estimate[1], sdlog=param$estimate[2]), col="blue")** **})** 2. Now try the same thing with log10(income) **logincome = log10(ds$income)** **hist(logincome, main="Distribution of Household Income", freq=FALSE)** # line type lty(2) is a dashed line **lines(density(logincome), lty=2, lwd=2)** **xvals = seq(from=min(logincome), to=max(logincome), length=100)** **param = fitdistr(logincome, "normal")** **lines(xvals, dnorm(xvals, param$estimate[1], param$estimate[2]), lwd=2, col="blue")**

Step	Action
7	**Compute Correlation between income and number of rooms:** 1. You need to consider your hypothesis. • Your hypothesis is that the number of rooms in a house is predicted by household income (the rich can buy bigger houses), e.g. *lm(rooms ~ income)* • Therefore, our null hypothesis: no correlation between income and number of rooms. • Alternate hypothesis: there is a correlation between income and the number of rooms. 4. Execute the following code (listed after the comment line "Step7 in the script file). `with(ds, cor(income, rooms))` `with(ds, cor(log(income), rooms))) # this will give a better correlation` 5. For comparison, correlate rooms with a completely unrelated variable. `n = length(ds$income)` `with(ds, cor(runif(n), rooms))`
8	**Create a Boxplot - Distribution of income as a factor of number of rooms:** 1. Select and execute the code (Listed after the comment line "Step 8") in the script window. 2. Plot the distribution of income as a factor of # of rooms. 'log="y"' plots income on log scale. We will suppress the outlier points and let the whiskers cover the full range of the data. `boxplot(income ~ as.factor(rooms), data=ds, range=0, outline=F, log="y", xlab="# rooms", ylab="Income")` 3. Plot the # of rooms as a function of wealth level. `boxplot(rooms ~ wealth, data = ds, main="Room by Wealth", Xlab="Category", ylab="# rooms")` `# we'll keep the outlier points in this one`

Step	Action
9	**Exit R:** 1. Type the following command into the RStudio command window: `q()` 2. R will ask you if you want to save your workspace. Answer "**no**."

End of Lab Exercise

Part 2 – Graphics Package Plots and Hypothesis Tests

Workflow Overview

1. • Define problem – Analysis of Variance (ANOVA)
2. • Generate the Data
3. • Examine the Data
4. • Plot and determine how purchase size varies within the three groups
5. • Use lm() to do the ANOVA
6. • Use Tukey's test to check all the differences of means
7. • Use the lattice package for density plot
8. • Plot the Logarithms of the Data
9. • Use ggplot() package
10. • Generate the example data to perform a Hypothesis Test with manual calculations
11. •Create a function to calculate the pooled variance, which is used in the Student's t statistic
12. • Examine the Data
13. • Calculate the t statistic for Student's t-test
14. • Calculate the degrees of freedom
15. • Compute the area under the curve
16. • Perform Student's t-test directly and compare the results

Lab Instructions

Step	Action
1	**Define problem - Analysis of Variance (ANOVA):** Suppose we are evaluating our marketing department's incentive campaign that is trying to increase the amount of money that customers spend when they visit our online site. We ran a short experiment, where visitors to our site randomly received one of two incentive offers or got no offer at all.
2	**Generate the Data:** `offers = sample(c("noffer", "offer1", "offer2"), size=500, replace=T)` `purchasesize = ifelse(offers=="noffer", rlnorm(500, meanlog=log(25)), ifelse(offers=="offer1", rlnorm(500, meanlog=log(50)), rlnorm(500, meanlog=log(55))))` `offertest = data.frame(offer=offers, purchase_amt=purchasesize)`
3	**Examine the Data:** `summary(offertest)` The following command does the equivalent of the SQL command "SELECT avg(purchase_amt) FROM offertest GROUP BY offer", `aggregate(x=offertest$purchase_amt, by=list(offertest$offer), FUN="mean")`
4	**Plot and determine how purchase size varies within the three groups:** 1. The 'log="y"' argument plots the y axis on the log scale. Does it appear that making offers increases purchase amount? `boxplot(purchase_amt ~ as.factor(offers), data=offertest, log="y")`

Step	Action
5	**Use lm() to do the ANOVA:** 1. Execute the following commands: `model = lm(log10(purchase_amt) ~ as.factor(offers), data=offertest)` `summary(model)` 2. What is the p-value on the F-stat? Can we reject the null hypothesis? _____ The intercept of the model is the mean value of log10(purchase_amt | no offer), (call it m0) and the other coefficients are: mean(log10(purchase_amt |offer1)) – m0, and mean(log10(purchase_amt |offer2)) – m0, respectively. 3. What are the p-values on those coefficients? _____ 4. Can we reject the null hypotheses that the mean purchase amount for offer1 was different from that of no offer, and similarly for offer2 vs. no offer? _____
6	**Use Tukey's test to check all the differences of means:** 1. Execute the following command: `TukeyHSD(aov(model))` 1. Did offer1 and offer2 increase purchase size to different amounts (to the p<0.05 significance level)? _____ 2. What would you recommend to the marketing department, based on these results? _____

Step	Action
7	**Use the lattice package for density plot:** For this course, you are only expected to become familiar with the base graphics capabilities of R; however, there are other graphics packages available for R that makes certain kinds of visualizations easier to produce. If you continue to use R in the future, it will be helpful to be aware of these alternatives to base graphics. The lattice package makes it easy to split data into different groups to highlight the differences between the groups. Here, we split the purchase_amt data by offer, and plot the three offer-specific purchase_amt densityplots on the same graph. `library(lattice)` `densityplot(~ purchase_amt, group=offers, data=offertest, auto.key=T)`
8	**Plot the Logarithms of the Data:** 1. Because the data is so left-skewed, we may want to plot the logarithms of the data to more clearly see the differences in the distributions, and the different locations of the modes. `densityplot(~ log10(purchase_amt), group=offers, data=offertest, auto.key=T)` 2. Also try the plots: `densityplot(~purchase_amt | offers, data=offertest)` `densityplot(~log10(purchase_amt) | offers, data=offertest)` 3. Which style of graph do you find more helpful?

Step	Action
10	**Use ggplot() package:** The ggplot2 package is based on a theory of the "algebra of graphs". The syntax is rather complex, but ggplot excels at assembling rich composite graphs that use a variety of different graphic techniques. Here, we show how to produce a variation of the scatterplot + box-and-whisker plot that we saw earlier in the course, to plot the distributions of purchase amounts against offer. 1. Execute the following commands: `library(ggplot2)` `ggplot(data=offertest, aes(x=as.factor(offers), y=purchase_amt)) + geom_point(position="jitter", alpha=0.2) + geom_boxplot(alpha=0.1, outlier.size=0) + scale_y_log10()` The function geom_point() plots scatterplots. The function geom_boxplot() plots box-and-whisker plots; outlier.size()=0 removes the outlier points beyond the whiskers that normally would be plotted. The function scale_y_log10() plots the y axis on a log10 scale. 2. You need to plot at least one geom_xxx to get a graph. Try adding and removing the different terms of the graphing command to create simpler scatterplots or box-and-whisker plots, with and without log scaling. 3. Here's how you would create the densityplots that you created in lattice. Execute the following commands. `ggplot(data=offertest) + geom_density(aes(x=purchase_amt, colour=as.factor(offers)))` `ggplot(data=offertest) + geom_density(aes(x=purchase_amt, colour=as.factor(offers))) + scale_x_log10()`

Step	Action
11	**Generate the example data to perform a Hypothesis Test with manual calculations:** Hopefully, you won't have to do this too often. Most statistical packages have functions that calculate a test statistic and evaluate it against the proper distribution, for the most common hypothesis tests. On occasion, you may need to calculate the p-values yourself. For our example, we will calculate the Student's t-test for difference of means (unlike Welch's test, Student's t-test assumes identical variances), under the alternative hypothesis that the means are not equal. 1. Select and execute the following commands: `x = rnorm(10) # distribution centered at 0` `y = rnorm(10,2) # distribution centered at 2`
12	**Create a function to calculate the pooled variance, which is used in the Student's t statistic:** 1. Select and execute the following commands. This will create a function named *pooled.var.* `pooled.var = function(x, y) {` `  nx = length(x)` `  ny = length(y)` `  stdx = sd(x)` `  stdy = sd(y)` `  num = (nx-1)*stdx^2 + (ny-1)*stdy^2` `  denom = nx+ny-2  # degrees of freedom` `  (num/denom) * (1/nx + 1/ny)` `}`
13	**Examine the Data:** Select and execute the following commands: `mx = mean(x)` `my = mean(y)` `mx - my` `pooled.var(x,y)`

Step	Action
14	**Calculate the t statistic for Student's t-test:** 1. Select and execute the following commands: `tstat = (mx - my)/sqrt(pooled.var(x,y))` `tstat`
15	**Calculate the degrees of freedom:** Under the null hypothesis, the t statistic is distributed in a Student's distribution with nx+ny-2 degrees of freedom. Calculate the degrees of freedom for our problem. Select and execute the following commands: `dof = length(x) + length(y) – 2` `dof`
16	**Compute the area under the curve:** The function pt(x, dof) gives the area under the curve from *-Inf* to *x* for the Student's distribution with dof degrees of freedom. Since in this case we have a negative tstat, pt(tstat, dof) will give us the area under the left tail. 1. Select and execute the following commands: `tailarea = pt(tstat, dof)` 2. Since our null hypothesis is that m1 <> m2, we need the area under both tails. `pvalue = 2*tailarea` 3. Are the means different (to the p<0.05 significance level)? _____
17	**Perform Student's t-test directly and compare the results:** 1. Execute the following command: `t.test(x, y, var.equal=T)` 2. Does t.test() give the same results? _____

End of Lab Exercise

Lab Exercise 4: K-means Clustering

Purpose:	This lab is designed to investigate and practice K-means Clustering. After completing the tasks in this lab you should able to: • Use R functions to create K-means Clustering models • Use ODBC connection to the database and execute SQL statements and load datasets from the database in an R environment • Visualize the effectiveness of the K-means Clustering algorithm using graphic capabilities in R • Use MADlib functions for K-means clustering

Tasks:	Tasks you will complete in this lab include: • Use the R –Studio environment to code K-means Clustering models • Use the ODBC connection in the R environment to create the average household income from the census database as test data for K-means Clustering • Use R graphics functions to visualize the effectiveness of the K-means Clustering algorithm • Use MADlib functions for K-means clustering

References:	References used in this lab are located in your ***Student Resource Guide Appendix.*** http://www.statmethods.net/advstats/cluster.html (originally from Everitt & Hothorn).

Workflow overview

1. Set the Working Directory
2. Establish the ODBC Connection
3. Open Connections to ODBC Database
4. Get Data from the Database
5. Read in the Data for Modeling
6. Execute the Model
7. Review the Output
8. Plot the Results
9. Find the Appropriate Number of Clusters
10. Close Connections to ODBC Database
11. Perform K-means Clustering Using In-database Analytics Method

Lab Instructions

Step	Action
1	Log in with GPADMIN credentials onto R-Studio.
2	**Set the Working Directory:** 1. Set working directory to ~/LAB04/, execute the command: `setwd("~/LAB04")` • (Or use the "Tools" option in the tool bar in the RStudio environment.)
3	**Establish the ODBC Connection:** Load the RODBC package, type in: `library('RODBC')`
4	**Open Connections to ODBC Database:** Before connecting to the ODBC database make sure the file, /etc/odbc.ini, is set to point to database, "training2". 1. If not, edit the line that starts with "Database =" within the file /etc/odbc.ini, to point to "training2". 2. Ensure the username(uid) and password (pwd) are provided correctly in the following command – note, you need to supply the correct password in the "pwd" field below: `ch <- odbcConnect("Greenplum",uid="gpadmin", case="postgresql",pwd="password_of_the_gpadmin_user")`

Step	Action
5	**Get Data from the Database:** 1. Before creating the table, “income_state”, you must first delete the table, if it already exists. Type in: `sqlDrop(ch,"income_state")` 2. Use the sqlQuery command to create the table, “income_state” : `> sqlQuery(ch,` `"CREATE TABLE income_state AS` `SELECT` `f.name AS state` `, round(avg(h.hinc),0) AS income` `FROM` `housing AS h` `JOIN` `fips AS f` `ON` `h.state = f.code` `WHERE` `h.hinc > 0` `GROUP BY` `f.name` `DISTRIBUTED BY (income);` `" )` **Note:** This code creates the table, “income_state”, in database “training2”. 3. Inspect this table using the following command: `sqlColumns(ch,"income_state")` 4. Review the output on the console. **Note:** The SQL Query is available for you to copy and paste into the working directory File name: mod4lab4.sql.

Step	Action
6	**Read in the Data for Modeling:** You need the data to be read in as a "matrix". 1. Execute the following statement to read in the database table "income_state". Use the sqlFetch command. The "rownames" attribute ensures the data is rendered as a matrix and the row names are taken from the column "state". `income <- as.matrix(sqlFetch(ch,"income_state",` `rownames="state"))` `> summary(income)` 2. Review the results of "income" on the console window. 3. Ensure that in the "data" window the variable "income" is represented as a 52x1 integer matrix.
7	**Execute the Model:** 1. Sort the data "income" before the modeling process. This will make it easier to understand the results and in visualizing. `income <- sort(income)` The K-Means function, provided by the *cluster* package, is used as follows: `kmeans(x, centers, iter.max = 10, nstart = 1, algorithm =` `c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"))` where the arguments are: • **x:** A numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns). • **centers:** Either the number of clusters or a set of initial (distinct) cluster centers. If a number, a random set of (distinct) rows in x is chosen as the initial centers. • **iter.max:** The maximum number of iterations allowed. • **nstart:** If *centers* is a number, *nstart* gives the number of random sets that should be chosen. • **algorithm:** The algorithm to be used. It should be one of the following "Hartigan-Wong", "Lloyd", "Forgy" or "MacQueen". If no algorithm is specified, the algorithm of Hartigan and Wong is used by default. 2. Cluster the data into 3 groups (centers = 3) and also specify the number of random sets to start with as, 15. `> # Fit the k-means cluster with 3 initial cluster centers` `> km <- kmeans (income,3,15)`

Step	Action
8	**Review the Output:** 1. Use the following command to display the fitted model on the console: `> km` 2. What are the cluster means? 3. What are the available components in the model? 4. How many data points cluster into each group? The output from the model provides the following: • **cluster** A vector of integers (from 1:k) indicating the cluster to which each point is allocated. • **centers** A matrix of cluster centers. • **withinss** The within-cluster sum of squares for each cluster. • **totss** The total within-cluster sum of squares. • **tot.withinss** Total within-cluster sum of squares, that is, sum(withinss). • **betweenss** The between-cluster sum of squares. • **size** The number of points in each cluster.
9	**Plot the Results:** 1. Now plot the results using the following commands: `> # plot clusters` `> plot(income, col = km$cluster)` `> # plot centers` `> points(km$centers, col = 1:3, pch = 8)` 2. Review the output on the graphic window.
10	**Find the Appropriate Number of Clusters:** 1. Plot the within-group-sum of squares and look for an "elbow" of the plot. The elbow (if you can find one) tells you what the appropriate number of clusters probably is. (Adapted from http://www.statmethods.net/advstats/cluster.html (originally from Everitt & Hothorn)). `wss <- numeric(15)` `> for (i in 1:15) wss[i] <- sum(kmeans(income,` `centers=i)$withinss)` `> plot(1:15, wss, type="b", xlab="Number of Clusters",` `ylab="Within groups sum of squares")` 2. Review the output on the graphic window. Is there an elbow to the plot? 3. Repeat the modeling with a few values around the elbow (or 4 and 5 centers if there is no elbow) and review the results.

Step	Action
11	**Close Connections to ODBC Database:** Use the following command: `odbcClose(ch)` The R Code for this exercise is available at /home/gpadmin/LAB04/kmeans1.R

Step	Action
12	**Perform K-means Clustering Using In-database Analytics Method:** 1. In-database analytics using "MADlib" function calls will be detailed in module 5 later in this course. However this step provides an overview of the flexibility to "cluster" the data points within the database environment. Review the code provided below: `DROP TABLE IF EXISTS myschema.data;` `CREATE TABLE myschema.data (` `pid INT` `, position FLOAT8[])` `DISTRIBUTED BY (pid);` `INSERT INTO myschema.data (pid,position[1])` `SELECT` `h.state` `, round(avg(h.hinc),0)` `FROM` `housing AS h` `WHERE` `h.hinc > 0` `GROUP BY` `h.state` `;` `SET SEARCH_PATH to madlib,public,myschema;` `SELECT madlib.kmeans('myschema.data', 3, 1, 'mytestrun', 'myschema');` `SELECT * FROM myschema.kmeans_out_centroids_mytestrun;` `SELECT * FROM myschema.kmeans_out_points_mytestrun;` 2. The first part of the code is similar to the one created in step 5 of this lab. The K-means function is called by: `SELECT madlib.kmeans( 'input_table', k,` `'goodness', 'run_id', 'output_schema');` The centroid locations are stored in kmeans_out_centroids_(*run_id*): The cluster assignments for each input data point are stored in kmeans_out_points_(*run_id*): In the example above input_table is myshema.data, number of clusters is 3 goodness is 1 run_id is mytestrun and output schema is myschema. The code is as follows: `SELECT madlib.kmeans('myschema.data', 3, 1, 'mytestrun', 'myschema');`

Step	Action
12 Cont.	3. The outputs are available in tables kmeans_out_centroids_madlib.mytestrun kmeans_out_points_madlib.mytestrun in schema myschema. The code below will output the results: `SELECT * FROM myschema.kmeans_out_centroids_mytestrun;` `SELECT * FROM myschema.kmeans_out_points_mytestrun;` 4. Review the results and compare them with the results generated with R in step 7.
13	The code is available /home/gpadmin/LAB04/kmeansmadlib.sql You can execute this code with the following command in the appropriate directory: `psql –d training2 –f kmeansmadlib.sql`

End of Lab Exercise

Lab Exercise 5: Association Rules

Purpose:	This lab is designed to investigate and practice Association Rules. After completing the tasks in this lab you should able to: • Use R functions for Association Rule based models

Tasks:	Tasks you will complete in this lab include: • Use the R –Studio environment to code Association Rule models • Apply constraints in the Market Basket Analysis methods such as minimum thresholds on support and confidence measures that can be used to select interesting rules from the set of all possible rules • Use R graphics "arules" to execute and inspect the models and the effect of the various thresholds

References:	• The groceries data set - provided for arules by Michael Hahsler, Kurt Hornik and Thomas Reutterer. http://rss.acs.unt.edu/Rdoc/library/arules/html/Groceries.html o Michael Hahsler, Kurt Hornik, and Thomas Reutterer (2006) Implications of probabilistic data modeling for mining association rules. In M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nuernberger, and W. Gaul, editors, *From Data and Information Analysis to Knowledge Engineering, Studies in Classification, Data Analysis, and Knowledge Organization*, pages 598–605. Springer-Verlag.

Workflow Overview

1. Set the Working Directory and install the “arules” package
2. Read in the Data for Modeling
3. Review Transaction data
4. Plot Transactions
5. Mine the Association Rules
6. Read in Groceries dataset
7. Mine the Rules for the Groceries Data
8. Extract the Rules in which the Confidence Value is >0.8 and high lift

LAB Instructions

Step	Action
1	Log in with GPADMIN credentials on to R-Studio.
2	**Set the Working Directory and install the "arules" package:** To understand Market Basket Analysis and the R package "arules," use a simple set of transaction lists of "book-purchases". 1. Set the working directory to ~/LAB05/ by executing the command: `setwd("~/LAB05")` • (Or using the "Tools" option in the tool bar in the RStudio environment.) 2. Load the package and the required libraries: `> # Load libraries` `> library('arules')`

Step	Action
3	**Read in the Data for Modeling:** • **Transaction List** is a special data type function in the "arules" package. 1. Read the data in as a Transaction List using the following statement for the states data, "MBAdata.csv". `> #read in the csv file as a transaction data` `> txn <- read.transactions ("MBAdata.csv",rm.duplicates = FALSE,format="single",sep=",",cols=c(1,2))` The arguments for the **read.transaction functions** are detailed below: • **file** the file name. • **format** a character string indicating the format of the data set. One of "basket" or "single", can be abbreviated. • **Sep** a character string specifying how fields are separated in the data file, or NULL (default). For basket format, this can be a regular expression; otherwise, a single character must be given. The default corresponds to white space separators. • **Cols** For the 'single' format, cols is a numeric vector of length two giving the numbers of the columns (fields) with the transaction and item ids, respectively. For the 'basket' format, cols can be a numeric scalar giving the number of the column (field) with the transaction ids. If cols = NULL • **rm.duplicates** a logical value specifying if duplicate items should be removed from the transactions.
4	**Review Transaction data:** 1. First inspect the transaction data `>txn@transactionInfo` `>txn@itemInfo` 2. Review the results on the console
5	**Plot Transactions:** 1. Use the "image" function that shows a visual representation of the transaction set in which the rows are individual transactions (identified by transaction ids) and the dark squares are items contained in each transaction. `> image(txn)` 2. Review the output in the graphics window

Step	Action
6	**Mine the Association Rules:** The **"apriori" function**, provided by the *arulesr* package, is used as follows: `rules <- apriori(File, parameter = list(supp = 0.5, conf = 0.9, target = "rules"))` where the arguments are: • **data** object of class transactions or any data structure which can be coerced into transactions (for example, a binary matrix or data.frame). • **parameter** named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 5. 1. Read in the statement for the transaction data: `> #mine association rules` `> basket_rules <-` `apriori(txn,parameter=list(sup=0.5,conf=0.9,target="rules"))` 2. Review the output on the console. The number of rules generated can be seen in the output and is represented as follows: `writing ... [1 rule(s)] done [0.00s]` 3. Inspect the rule using the following statement: `> inspect(basket_rules)` 4. Review the output. 5. State the generated rule and the support, confidence and the lift thresholds for the rule

Step	Action
7	**Read in Groceries dataset** Use the standard data set, "Groceries" available with the "arules" package. • The Groceries data set contains 1 month (30 days) of real-world point-of-sale transaction data from a typical local grocery outlet. The data set contains 9835 transactions and the items are aggregated to 169 categories. 1. Read in the data set and inspect the item information `> #Read in Groceries data` `> data(Groceries)` `> Groceries@itemInfo`
8	**Mine the Rules for the Groceries Data:** `> #mine rules` `> rules <- apriori(Groceries, parameter=list(support=0.001, confidence=0.5))` • Note the values used for the parameter list. 1. How many rules are generated?
9	**Extract the Rules in which the Confidence Value is >0.8 and high lift:** 1. Execute the following commands: `> subrules <- rules[quality(rules)$confidence > 0.8]` `> inspect(subrules)` 2. Review the results. 3. How many sub-rules did you extract? • These rules are more valuable for the business. 4. Extract the top three rules with high threshold for the parameter "lift". `> #Extract the top three rules with high lift` `> rules_high_lift <- head(sort(rules, by="lift"), 3)` `> inspect(rules_high_lift)` 5. List the rules and the value of the parameters associated with these rules: • The R Code for this exercise is available at home/gpadmin/LAB05/mba.R

End of Lab Exercise

Lab Exercise 6: Linear Regression

Purpose:	This lab is designed to investigate and practice the Linear Regression method. After completing the tasks in this lab you should able to: • Use R functions for Linear Regression (Ordinary Least Squares – OLS) • Predict the dependent variables based on the model • Investigate different statistical parameter tests that measure the effectiveness of the model

Tasks:	Tasks you will complete in this lab include: • Use the R –Studio environment to code OLS models • Review the methodology to validate the model and predict the dependent variable for a set of given independent variables • Use R graphics functions to visualize the results generated with the model

References:	References used in this lab are located in your ***Student Resource Guide Appendix.***

Workflow Overview

1. Set Working directory
2. Use random number generators to create data for the OLS Model
3. Generate the OLS model using R function “lm”
4. Print and visualize the results and review the plots generated
5. Generate Summary Outputs
6. Introduce a slight non-linearity and test the model
7. Perform In-database Analysis of Linear Regression

LAB Instructions

Step	Action
1	Log in with GPADMIN credentials on to R-Studio.
2	**Set Working directory** Set the working directory to ~/LAB06/ by executing the command: `setwd("~/LAB06")` • (Or using the "Tools" option in the tool bar in the RStudio environment).
3	**Use random number generators to create data for the OLS Model :** 1. Run the "runif" function in R which generates random deviates within the specified minimum and maximum range. `x <- runif(100, 0, 10)` This generates 100 random values for "x" in the range 0 to 10. 2. Create the dependent variable "y" with the "beta" values as 5 and 6 and the "sigma" = 1 (generated with the "rnorm" function, random generation for the normal distribution with mean =0 and SD= 1.) `y <- 5 + 6*x + rnorm(100)` 3. Plot it `> plot(x,y)` 4. Review the results in the graphics window

Step	Action
4	**Generate the OLS model using R function "lm":** An OLS Model is generated with an R function call "lm". You can learn about "lm" with the following command on the console: `?lm` 1. Generate an OLS Model using the following command: `d <- lm(y ~ x)` 2. Use the following command to display the structure of the object "d" created with the function call "lm" `str(d)` 3. You can see the details of the fitted model. What are the values of the coefficients (Beta) in the model?

Step	Action
5	**Print and visualize the results and review the plots generated** 1. Get the compact results of the model with the following command: `print(d)` 2. Visualize the model with the command: `par(mfrow=c(2,2))` `plot(d)` The explanation of the plots are as follows: **Residuals vs. Fitted**: you want to make sure that the errors are evenly distributed over the entire range of fitted values; if the errors are markedly larger (or biased either positively or negatively) in some range of the data, this is evidence that this model may not be entirely appropriate for the problem. **Q-Q plot**: tests whether or not the errors are in fact distributed approximately normally (as the model formulation assumes). If they are, the Q-Q plot will be along the x=y line. If they aren't, the model may still be adequate, but perhaps a more robust modeling method is suitable. Also, the usual diagnostics (R-squared, t-tests for significance) will not be valid. **Scale-Location**: a similar idea to Residuals v. Fitted; you want to make sure that the variance (or its stand-in, "scale") is approximately constant over the range of fitted values. **Residuals vs. Leverage**: used for identifying potential outliers and "influential" points. Points that are far from the centroid of the scatterplot in the x direction (high leverage) are influential, in the sense that they may have disproportionate influence on the fit (that doesn't mean they are wrong, necessarily). Points that are far from the centroid in the y direction (large residuals) are potential outliers. 3. Here are some examples of plots that may be a little more intuitive, Type in the following: `> ypred <- predict(d)` `> par(mfrow=c(1,1))` `> plot(y,y, type="l", xlab="true y", ylab="predicted y")` `> points(y, ypred)` Review the results in the graphics window. The plot of predicted vs. true outcome can be seen there. The plot should be near the x=y line. Where it does not run along the x=y line indicates where the model tends to over-predict or under-predict. You can also use this plot to identify ranges where the errors are especially large. This information is similar to the Residuals vs. Fitted plot, but perhaps is more intuitive to the layperson. Note: The “predict” function requires the variables to be named exactly as in the fitted model.

Step	Action
6	**Generate Summary Outputs:** 1. For more detailed results type: `d1 <- summary(d)` `print(d1)` Read the explanations given below from the summary output and note the values from the output on the console for each statistic detailed: **coefficients :** the estimated value of each coefficient, along with the standard error. coefficient +/- 2*std.error is useful as a quick measure of confidence interval around the estimate. **t-value**: coefficient/std.error, or how tight an estimate this is (compared to 0). If the "true" coefficient is zero (meaning this variable has no effect on the outcome), t-value is "small". **Pr(>|t|):** the probability of observing this t-value if the coefficient is actually zero. You want this probability to be small. How small depends on the significance desired. Standard significances are given by the significance codes. So, for example "***" means that the probability that this coefficient is really zero is negligible. **R-squared:** A goodness of fit measure: the square of the correlation between predicted response and the true response. You want it close to 1. Adjusted R-squared compensates for the fact that having more parameters tends to increase R-squared. Since we only have one variable here, the two are the same. **F-statistic and p-value**. Used to determine if this model is actually doing better than just guessing the mean value of y as the prediction (the "null model"). If the linear model is really just estimating the same as the null model, then the F-statistic should be about 1. The p-value is the probability of seeing an F-statistic this large, if the true value is 1. Obviously, you want this value to be very small. 2. Type in the following command: `> cat("OLS gave slope of ", d1$coefficients[2,1],` `"and an R-sqr of ", d1$r.squared, "\n")` 3. Note the result you see on the console in the space below:

Step	Action
7	**Introduce a slight non-linearity and test the model:** 1. Create a "training" data set First the training set in which we will introduce a slight non-linearity. `> x1 <- runif(100)` `> # introduce a slight nonlinearity` `> y1 = 5 + 6*x1 + 0.1*x1*x1 + rnorm(100)` 2. Generate the model `> m <- lm(y1 ~ x1)` 3. Create the test data set `> x2 <- runif(100)` `> y2 = 5 + 6*x2 + 0.1*x2*x2 + rnorm(100)` 4. Repeat steps 5 and 6 for model "m" and compare what you observe with the "ideal" in the earlier steps. 5. What's the R^2 on the model? Notice (from the ypred v. y plot) that the model tends to under-predict for higher values of y. 6. Use the predict function on the test data we generated: `y2pred <- predict(m,data.frame(x2))` 7. compare y2pred with the true outcomes y2 8. What did you observe from the comparison? • The code for this part of the lab is available at the following location: home/gpadmin/LAB06/ols.R

Step	Action
8	**Perform In-database Analysis of Linear Regression:** To illustrate an in-database execution of linear regression we make use of with approximately 64,000 rows of data aggregated from the census data. The following columns are generated for zip code and sex(male/female) tuples: zip code, mean age in the zip code, mean number of years of education, mean level of employment (Categorical level values were converted to a range from 1 - 3, three being a "high status" job, 1 being "low status" mean household employment in the zip code This data is stored in table “zeta” in the database “training2”. 1. Explore the table with Meta commands or with the pgadmin utility 2. Review the MADlib documentation at http://doc.madlib.net/v0.2beta/group__grp__linreg.html and the following code: `DROP TABLE IF EXISTS zeta1;` `CREATE TABLE zeta1 (` `  depvar FLOAT8` `  , indepvar FLOAT8[])` `DISTRIBUTED BY (depvar)` `;` `INSERT INTO zeta1 (` `  depvar` `, indepvar[1]` `, indepvar[2]` `, indepvar[3]` `, indepvar[4]` `)` `SELECT` `  ln(meanhouseholdincome + 1)` `, 1` `, CASE` `  WHEN sex = 'M'  THEN 0` `  WHEN sex = 'F' THEN 1` `  END AS sex` `, meanage` `, meanemployment` `FROM` `  zeta` `;`

Step	Action
8 Cont.	**SET SEARCH_PATH to madlib,public,myschema;** **SELECT (linregr(depvar,indepvar)).r2 FROM zeta1;** **SELECT (linregr(depvar,indepvar)).coef FROM zeta1;** **SELECT (linregr(depvar,indepvar)).std_err FROM zeta1;** **SELECT (linregr(depvar,indepvar)).t_stats FROM zeta1;** **SELECT (linregr(depvar,indepvar)).p_values FROM zeta1;** We are predicting mean household income with drivers age, sex, and years of employment: Notice that we take the log of income (refer to the discussions in the student resource guide) The code is available at **/home/gpadmin/LAB06/madliblinear.sql** 1. You can execute the code with the following command at the command prompt: **psql -d training2 –f madliblinear.sql** 2. Review the results and note your observations below:

End of Lab Exercise

Lab Exercise 7: Logistic Regression

Purpose:	This lab is designed to investigate and practice the Logistic Regression method. After completing the tasks in this lab you should able to: • Use R functions for Logistic Regression – *also known as Logit*) • Predict the dependent variables based on the model • Investigate different statistical parameter tests that measure the effectiveness of the model

Tasks:	Tasks you will complete in this lab include: • Use R –Studio environment to code Logit models • Review the methodology to validate the model and predict the dependent variable for a set of given independent variables • Use R graphics functions to visualize the results generated with the model

References:	References used in this lab are located in your ***Student Resource Guide Appendix.***

Workflow Overview

1. Set the Working Directory
2. Define the problem and review input data
3. Read in and Examine the Data
4. Build and Review logistic regression Model
5. Review and interpret the coefficients
6. Visualize the Model Using the Plot Function
7. Use relevel Function to re-level the Price factor with value 30 as the base reference
8. Plot the ROC Curve
9. Predict Outcome given Age and Income
10. Predict outcome for a sequence of Age values at price 30 and income at its mean
11. Predict outcome for a sequence of income at price 30 and Age at its mean
12. Use Logistic regression as a classifier

LAB Instructions

Step	Action
1	Log in with GPADMIN credentials on to R-Studio.
2	**Set the Working Directory** Set the working directory to ~/LAB07/ by executing the command: `setwd("~/LAB07")` • (Or using the "Tools" option in the tool bar in the RStudio environment).
3	**Define the problem and review input data** **Logistic Regression**, also known as ***Logit***, is typically used in models where the dependent variables have a binary outcome (True/False, which is coded with 1/0). You model the log odds of the outcome as a linear combination of predictor variables). Marketing Survey Data In this lab you use hypothetical marketing survey data in which customers: • Responded to the question: ○ Would you choose a product based on a "pricing" factor (three "Price" ranges 10, 20 and 30)? • Response options: ○ "1" for "yes" and "0" for "no" • The survey also collected information such as "Age" and "Income" of the respondent. Business Need The marketing campaign team wants to send special offers to those respondents with the highest probability of purchase. This data file "survey.csv" is available in the folder ~\LAB07\survey.csv. 1. Review the survey.csv file. 2. How many responses to the survey does the file contain? 3. What is the main purpose of building this model?

Step	Action
4	**Read in and Examine the Data:** 1. The first step in the modeling process is to examine the data and determine if there are any outliers. To do this you must read in the survey data, use the following command: `Mydata <- read.csv("survey.csv",header=TRUE,sep=",")` 2. With the following command, explore the data further: `> table(Mydata$MYDEPV)` `> with(Mydata, table(Price,MYDEPV))` `> summary(Mydata$Age)` `> cor.mat <- cor(Mydata[,-1])` `> cor.mat` 3. Review the results on the console **Note:** The general rule **is not** to include variables in your model that are too highly correlated with other predictors. For example, including two variables that are correlated by 0.85 in your model may prevent the true contribution of each variable from being identified by the statistical algorithm. Confirm that the variables in our survey do not fall in this category.

Step	Action
5	**Build and Review Logistic Regression Model:** 1. Use the "glm" function for logit modeling. Type in the following command: `mylogit <- glm(MYDEPV ~ Income + Age + as.factor(Price) , data=Mydata,family=binomial(link="logit"), na.action=na.pass)` 2. Review the model by typing the "summary" and "plot" functions: `summary (mylogit)` Review the results of the summary command, for the fitted model, on the console. Results you should see: • The first line provides the model you specified. • Next, you should see the **deviance residuals**, which provide the measure of the model fit. • The next part of the output shows the **coefficients,** their standard errors, the **z-statistic** (sometimes called a Wald z-statistic), and the associated **p-values.** • Both **Income** and **Age** are statistically significant, as are the two terms for **Price**. • The **logistic regression coefficients** show the change in the **log odds** of the outcome for a one unit increase in the predictor variable. • Residual deviance: analogous to the Residual Sum of Squares of a linear model; that is, it is related to the "total error" of the fit. It is twice the negative log likelihood of the model. • Null deviance: the deviance associated with the "null model" -- that is the model that returns just the global probability of TRUE for every x. The quantity 1 - (Residual deviance/Null deviance) is sometimes called "pseudo-R-squared"; you use it to evaluate goodness of fit in the same way that R-sqr is used for linear models. The interpretation of the results are as follows: 1. Review the "Estimate" column. For every one unit change in **Income**, the log odds of Purchase (versus no-Purchase) increases by 0.12876. 2. Record the number that describes how much one unit increase in **Age** increases the log odds of purchase: The indicator variables for **Price** are interpreted differently. **For** example, Purchase decision at a **Price** of 20, compared with a **Price** of **10,** decreases the log odds of admission by 0.74418 3. Record the log odds at Price point 30 compared to Price point 10 below: The summary then shows the table of coefficients that are "fit indices", including the null and deviance residuals and the AIC.

Step	Action
6	**Review the results and interpret the coefficients** 1. Use the "confint" function to obtain the confidence intervals of the coefficient estimates: `confint(mylogit)` 2. Review the results on the console. • You can also exponentiate the coefficients and interpret them as odds-ratios. • To get the exponentiated coefficients, use (**exp( )**) • The object you want to exponentiate is called coefficients and it is part of mylogit (**mylogit$coefficients**). `exp(mylogit$coefficients)` You can observe that for every unit change in income, the odd-ratio of Purchase increases by a multiplicative factor of 1.137 (and remember a multiplicative factor of 1 corresponds to no change). This is actually a bit more intuitive than the log odds explanation you reviewed in the previous step. Observe that that Age does not appear to be a very strong factor in this model, and the price factor of 30 has a stronger effect than a price factor of 20.
7	**Visualize the Model Using the Plot Function:** `plot(mylogit)` You should see multiple plots generated on the graphics window.

Step	Action
8	**Use relevel Function to re-level the Price factor with value 30 as the base reference**. In the original model that we fitted with the function call: mylogit <- glm(MYDEPV ~ Income + Age + as.factor(Price) , data= Mydata,family=binomial(link="logit"), na.action=na.pass) we obtained the results shown below: **Coefficients:** **Estimate Std. Error z value Pr(>|z|)** **(Intercept) -6.02116 0.53244 -11.309 < 2e-16 ***** **Income 0.12876 0.00923 13.950 < 2e-16 ***** **Age 0.03506 0.01179 2.974 0.00294 **** **as.factor(Price)20 -0.74418 0.26439 -2.815 0.00488 **** **as.factor(Price)30 -2.21028 0.31108 -7.105 1.2e-12 *** **---** What does this tell us? The odds of MYDEPV decreases when price changes from 10 to 20 and decreases even more when we go from 10 to 30. 1. Now let's use 30 as the reference price, instead of 10. Type in the following: **Mydata$pricefactor = relevel(as.factor(Mydata$Price), "30")**

Step	Action
8 Cont.	Fit the Model Again (mylogit2) and Display the Summary: `mylogit2 = glm(MYDEPV ~ Income + Age + pricefactor , data= Mydata,family=binomial(link="logit"), na.action=na.pass)` `summary(mylogit2)` You will see the results as follows: `Coefficients:` `Estimate Std. Error z value Pr(>|z|)` `(Intercept) -8.23144 0.66180 -12.438 < 2e-16 ***` `Income 0.12876 0.00923 13.950 < 2e-16 ***` `Age 0.03506 0.01179 2.974 0.00294 **` `pricefactor10 2.21028 0.31108 7.105 1.20e-12 ***` `pricefactor20 1.46610 0.29943 4.896 9.76e-07 ***` `---` Notice that the intercept has changed (because we changed the reference situation), but the coefficients for Income and Age are the same. The new model tells us that the odds of MYDEPV increase when price decreases from 30 to 10, and less so price decreases from 30 to 20.

Step	Action
9	**Plot the ROC Curve:** 1. Make sure you have the package ROCR installed and the library included `install.packages("ROCR")` `library(ROCR)` 2. First get all the probability scores on the training data `pred = predict(mylogit, type="response")` 3. Every classifier evaluation using ROCR starts with creating a prediction object. This function is used to transform the input data (which can be in vector, matrix, data frame, or list form) into a standardized format. We create the prediction object needed for ROCR as follows: `predObj = prediction(pred, Mydata$MYDEPV)` 4. All kinds of predictor evaluations are performed using the function "performance". Read and understand the parameters of the function with `?performance` 5. We now create the ROC curve object and the AUC object with performance function `rocObj = performance(predObj, measure="tpr", x.measure="fpr")` `# creates ROC curve obj` `aucObj = performance(predObj, measure="auc")  # auc object` 6. Extract the value of AUC and display on the console: `auc = aucObj@y.values[[1]]` `auc` 7. What is the value of AUC? 8. We will plot the ROC curve now `plot(rocObj, main = paste("Area under the curve:", auc))` 9. Review the curve on the plot window. Review the discussions on ROC in the student resources guide. Record your observations below:

Step	Action
10	**Predict Outcome given Age and Income:** 1. Use the "predict" function to predict the probability of the purchase outcome given **Age** and **Income.** Start with predicting the probability of the purchase decision at different **Price** points (10, 20, and 30). Create a "data frame" called "newdata1" using the following commands: `Price <- c(10,20,30)` `Age <- c(mean(Mydata$Age))` `Income <- c(mean(Mydata$Income))` `newdata1 <- data.frame(Income,Age,Price)` `newdata1` You are predicting with **Income** and **Age** both set at their mean value and **Price** at 10, 20 and 30. **Note:** The values of the data frame "newdata1" displayed on the console. The predict function requires the variables to be named exactly as in the fitted model. 2. Create the fourth variable "PurchaseP". `newdata1$PurchaseP <- predict` `(mylogit,newdata=newdata1,type="response")` `newdata1` 3. What is your observation on the probability of purchase at different **Price** levels?

Step	Action
11	**Predict outcome for a sequence of Age values at price 30 and income at its mean:** 1. Keep the **Price** at 30, **Income** at its mean value and select a sequence of values for **Age** starting at a minimum age, incrementing by 2 until the maximum age in our dataset: `newdata2 <-` `data.frame(Age=seq(min(Mydata$Age),max(Mydata$Age),2),` `Income=mean(Mydata$Income),Price=30)` `newdata2$AgeP<-` `predict(mylogit,newdata=newdata2,type="response")` `cbind(newdata2$Age,newdata2$AgeP)` Newdata2$AgeP stores the predicted variables and you just display the sequence for **Age** you generated and the corresponding probability of the purchase decision using the "cbind" function shown above. 2. Plot and visualize how the "purchase" probability varies with Age: `plot(newdata2$Age,newdata2$AgeP)`
12	**Predict outcome for a sequence of income at price 30 and Age at its mean:** 1. Using the same methodology, create a data frame newdata3 with the following characteristics: • **Income** is a sequence from 20 to 90 in steps of 10 • **Age** is the mean value for the dataset Mydata • **Price** point at 30 2. Predict **newdata3$IncomeP** and display the Income sequence along with the predicted probabilities. 3. Plot the results.

Step	Action
13	**Use Logistic regression as a classifier:** Recall the problem statement in Step 3, the marketing campaign team wants to send special offers to those respondents with the highest probability of purchase. They have established a threshold of 0.5 and they want to target customers whose probability of purchase are greater than 0.5. **Note:** We are assuming that age and income are uniformly distributed in our customer base, and the price factors of our products are also uniformly distributed. Typically in order to run a scenario like this you should understand the demographic distribution of the customers (and the price distribution of the products). 1. You want an idea of how many offers will be sent out, using this threshold, so you test it on a 'random' set of data. First, generate this random set using "runif" functions: `newdata4 <- data.frame (` `Age=` `round(runif(10,min(Mydata$Age),max(Mydata$Age))),` `Income=round(runif(10,min(Mydata$Income),max(Mydata$Income))),` `Price = round((runif(10,10,30)/10))*10)` `newdata4$Prob <-` `predict(mylogit,newdata=newdata4,type="response")` `newdata4` 2. How many samples in your random selection qualify for special offers? The R Code for this exercise is available at home/gpadmin/LAB07/logit.R

End of Lab Exercise

Lab Exercise 8: Naïve Bayesian Classifier

Purpose:	This lab is designed to investigate and practice the Naïve Bayesian Classifier analytic technique. After completing the tasks in this lab you should be able to: • Use R functions for Naïve Bayesian Classification • Apply the requirements for generating appropriate training data • Validate the effectiveness of the Naïve Bayesian Classifier with the big data

Tasks:	Tasks you will complete in this lab include: • Use R –Studio environment to code the Naïve Bayesian Classifier • Use the ODBC connection to the "census" database to create a training data set for Naïve Bayesian Classifier from the big data • Use the Naïve Bayesian Classifier program and evaluate how well it predicts the results using the training data and then compare the results with original data

References:	References used in this lab are located in your ***Student Resource Guide Appendix.***

Part 1 – Building Naïve Bayesian Classifier

Workflow Overview

1. Set working directory and review training and test data
2. Install and load library “e1071”
3. Read in and review data
4. Build the Naïve Bayesian classifier Model from First Principles
5. Predict the Results
6. Execute the Naïve Bayesian Classifier with e1071 package
7. Predict the Outcome of “Enrolls” with the Testdata
8. Review results

LAB Instructions

Step	Action
1	Log in with GPADMIN credentials on to R-Studio.
2	**Set working directory and review training and test data** 1. Set the working directory using the following command: `> setwd("~/LAB08")` • The **"sample1.csv"** file in this directory represents the data worked with in the instructor led training session. The file has a header row, followed by 14 rows of training data. • The **testing data** on which you will predict the results should be appended after the **training data**. The data set should read: `Age,Income,Jobstaisfaction,Desire,Enrolls ←-------Header` `<=30,High,No,Fair,No` `<=30,High,No,Excellent,No` `31 to 40,High,No,Fair,Yes` `>40,Medium,No,Fair,Yes` `>40,Low,Yes,Fair,Yes` `>40,Low,Yes,Excellent,No` `31 to 40,Low,Yes,Excellent,Yes` `<=30,Medium,No,Fair,No` `<=30,Low,Yes,Fair,Yes` `>40,Medium,Yes,Fair,Yes` `<=30,Medium,Yes,Excellent,Yes` `31 to 40,Medium,No,Excellent,Yes` `31 to 40,High,Yes,Fair,Yes` `>40,Medium,No,Excellent,No` `<=30,Medium,Yes,Fair, ←---------testing data`
3	**Install and load library "e1071"** Execute the following command to install the required packages and load the libraries: `> install.packages("e1071")` `> library("e1071")`

Step	Action
4	**Read in and review data** 1. Execute the following to read in the data. `> # read the data into a table from the file` `> sample <- read.table("sample1.csv",header=TRUE,sep=",")` `> # we will now define the data frames to use the NB classifier` `> # we will now define the data frames to use the NB classifier` `> traindata <- as.data.frame(sample[1:14,])` `> testdata <- as.data.frame(sample[15,])` You now have two data frame objects "**traindata**" and "**testdata**" for running the NB Classifier. 2. Execute the following command to display the data frames, to ensure they are loaded properly. `> #Display data frames` `> traindata` `> testdata` 3. Review the output on the console window.

Step	Action
5	**Build the Naïve Bayesian classifier Model from First Principles:** 1. The first step in building the model is the computation of prior probabilities. The independent variables here are the “Age”, “Income”, “Jobsatisfaction” and “Desire”. The dependent variable is “Enrolls” Compute the prior probabilities of enrollment, P(no), P(yes) first, the counts : `> tprior <- table(traindata$Enrolls)` then, normalize over the total number of instances to get the probabilities `> tprior <- tprior/sum(tprior)` `> tprior` Review the results of prior probabilities on the console 2. Compute the summaries that you need to create a Bayes model: P(A|b), b={no, yes} First, count up "no" and "yes" by Age: `> ageCounts <-table(traindata[,c("Enrolls", "Age")])` 3. Then, normalize by the total number of "no" and "yes" each to get the conditional probabilities `> ageCounts <- ageCounts/rowSums(ageCounts)` Display the results on the console and review the conditional probabilities `> ageCounts` 4. Do the same for the other variables. `> incomeCounts <- table(traindata[,c("Enrolls", "Income")])` `> incomeCounts <- incomeCounts/rowSums(incomeCounts)` `> jsCounts <- table(traindata[,c("Enrolls", "Jobsatisfaction")])` `> jsCounts<-jsCounts/rowSums(jsCounts)` `> desireCounts <- table(traindata[,c("Enrolls", "Desire")])` `> desireCounts <- desireCounts/rowSums(desireCounts)`

Step	Action
6	**Predict the Results:** 1. Use the Naïve Bayesian Classifier formula to compute product of P(A|b), for b={no, yes}. The maximum of the two is the "predicted" result of the dependent variable. In the test data we need to predict the "Enrolls" given the for Age<=30, Income = Medium, Jobsatisfaction = yes and Desire = Fair `> pyes <-` `ageCounts["Yes","<=30"]*` `incomeCounts["Yes","Medium"]*` `jsCounts["Yes","Yes"]*` `desireCounts["Yes","Fair"]*` `tprior["Yes"]` followed by `> pno <-` `ageCounts["No","<=30"]*` `incomeCounts["No","Medium"]*` `jsCounts["No","Yes"]*` `desireCounts["No","Fair"]*` `tprior["No"]` 2. The prediction will be max(pyes,pno). `> pyes` `> pno` `> max(pyes,pno)` 3. What is the predicted result for "Enrolls"?

Step	Action
7	**Execute the Naïve Bayesian Classifier with e1071 package:** The Naïve Bayes function computes the conditional a-posterior probabilities of a categorical class variable given independent categorical predictor variables using the Bayes rule. The usage takes the form of naiveBayes(formula, data,…) where the arguments are defined as follows: • **formula** A formula of the form class ~ x1 + x2 + ... Interactions are not allowed. • **data** Either a data frame of factors or a contingency table. • You are modeling for attribute " Enrolls". 1. Use the following commands to execute the model and display the results. `> # use the NB classifier` `> model <- naiveBayes(Enrolls ~.,traindata)` `> # display model` `> model` 2. Review the results on the console and compare these results to the **apriori probabilities** you manually computed earlier in step 5.
8	**Predict the Outcome of "Enrolls" with the Testdata:** 1. To use the predict function, type in the following: `> # predict with testdata` `> results <- predict (model,testdata)` `> # display results` `> results` 2. Review the results (Prediction for "Enrolls") on the console.

Step	Action
9	**Review results** 1. Look at P(age=31-40 | Enrolls = No). You will observe a zero probability. Is this a problem? 2. Build another NB model, with Laplace smoothing model2 = naiveBayes(Enrolls ~.,traindata, laplace=0.01) 3. Compare the probabilities here with those of the first model Note down your observations in the space provided below:

End of Lab Exercise

Part 2 – Naïve Bayesian Classifier – Census data

Workflow Overview

1. Define the Problem *(Translating to an Analytics Question)*
2. Establish the ODBC Connection
3. Open Connections to ODBC Database
4. Build the Training Dataset and the Test Dataset from the Database
5. Extract the first 10000 records for the training data set and the remaining 10 for the test
6. Execute the NB Classifier
7. Validate the Effectiveness of the NB Classifier with a Confusion Matrix
8. Execute NB Classifier with MADlib Function Calls Within the Database

LAB Instructions

Step	Action
1	**Define the Problem *(Translating to an Analytics Question):*** 1. Use "Persons" table in the Census dataset to categorize Age, Gender, Educational qualifications and Annual Income from each record as follows: ○ 3 Age categories: >20 and ≤ 30 , >30 and ≤45 and >45 ○ 2 Gender categories: M and F ○ 3 Educational Qualifications categories: >14 (Professional/Phd), >12 and ≤14 (College) and <12 (others) – ○ 3 Annual Income categories: >10000 and ≤ 50000, >50000 and ≤ 80000 and > 80000 2. Build an appropriate "training" dataset, which will be a subset of the "categorized" table with four columns age, gender, education and income. 3. Predict the annual income category a person will belong to given the Age, Gender and educational qualifications, using the Naïve Bayesian Classifier.
2	**Establish the ODBC Connection:** 1. Load the RODBC package, using the following command: `library('RODBC')`
3	**Open Connections to ODBC Database:** 1. Before connecting to the ODBC database make sure the file, /etc/odbc.ini is properly set to point to database "training2". 2. If not, edit the line that starts with, "Database =" within the file /etc/odbc.ini to point to "training2" 3. Ensure the username(uid) and password (pwd) are provided correctly in the following command : `ch <- odbcConnect("Greenplum",uid="gpadmin", case="postgresql",pwd="password_of_the_gpadmin_user")`

Step	Action
4	**Build the Training Dataset and the Test Dataset from the Database:** 1. Drop the table, NBtrain, from the database, use the following command `sqlDrop(ch,"NBtrain")` 2. Execute a SQL query using the sqlQuery command, creating the table, NBtrain, selecting 10010 random records and categorizing the variables in the categories we defined in Step1 of Part2 of this lab. Use the code below. **Note:** The code is available in the working directory "NBTrain.sql". To execute, you can copy and paste it into your R script window. `> sqlQuery(ch, "` `CREATE TABLE NBtrain (` `age VARCHAR(8)` `, sex VARCHAR(8)` `, educ VARCHAR(8)` `, income VARCHAR(8)` `) DISTRIBUTED BY (age)` `;` `INSERT INTO NBtrain` `SELECT` `t1.age` `, t1.sex` `, t1.educ` `, t1.income` `FROM (` `SELECT` `CASE` `WHEN age BETWEEN 20 AND 30 THEN '20-30'` `WHEN age BETWEEN 31 AND 45 THEN '31-45'` `WHEN age > 45 THEN 'GT 45'` `ELSE 'unknown age'` `END AS age` `, CASE` `WHEN sex = 1 THEN 'M'` `WHEN sex = 2 THEN 'F'` `ELSE 'unknown sex'` `END AS sex` `, CASE` `WHEN educ >14 THEN 'Prof/Phd'` `WHEN educ BETWEEN 12 AND 14 THEN 'College'` `WHEN educ <12 THEN 'Others'` `ELSE 'unknown educ'` `END AS educ`

Step	Action
4 Cont.	`, CASE` `WHEN inctot BETWEEN 10000 AND 50000 THEN '10-50K'` `WHEN inctot BETWEEN 50000+1 AND 80000 THEN '50-80K'` `WHEN inctot > 80000 THEN 'GT 80K'` `ELSE 'unknown i'` `END AS income` `FROM` `persons` `) AS t1` `WHERE` `not (t1.age like 'unk%' or t1.sex like 'unk%' or t1.educ like 'unk%' or t1.income like 'unk%')` `ORDER BY RANDOM ()` `LIMIT 10010` `;` `")`
5	**Extract the first 10000 records for the training data set and the remaining 10 for the test** 1. Use the sqlFetch command for reading data into an R data frame. `NBtrain <- (sqlFetch(ch,"NBtrain"))` 2. Extract the training dataset `> NBtrain1 <- NBtrain[1:10000,]` 3. Extract the test dataset `> NBtest <- NBtrain[10001:10010,]` 4. Close the ODBC channel `> odbcClose(ch)`

Step	Action
6	**Execute the NB Classifier:** 1. Run the model as you did in Part 1. Use the following command: `# model` `model <- naiveBayes(income ~.,NBtrain1)` `model` 2. Review the results of the model on the console. 3. Run the predict function using the following command: `# predict with testdata` `results <- predict (model,NBtest[1:10,-1])` `results` 4. Record the results of the predict function: • 1 • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9 • 10 5. Use the parameter "type" with a value "raw" and you can see how the scores are very close to 0 or 1 rather than looking like realistic probabilities `> results1 <- predict (model,NBtest[1:10,-1],type="raw")` `> results1` Note down your observations below:

Step	Action
7	**Validate the Effectiveness of the NB Classifier with a Confusion Matrix:** 1. Compare the results of the previous step with the actual data that you have in NBtest. Build a confusion matrix for the predictions vs. actual values: `> conf <- table(actual=NBtest[1:10,4],predicted=results)` `> conf` 2. The diagonals of "conf" give the count of correctly classified instances by class. The off-diagonals tell how many instances of each class are mis-classified. What class does the model predict best? 3. What % of data did the NB Classifier predicted correctly? You can calculate this as the sum of the diagonal elements of the confusion matrix normalized by the total number of test cases `> accuracy <- sum(diag(conf))/sum(conf)` `> accuracy` 4. Record any other observations below: • The R code and the data are available at /home/gpadmin/LAB08/NBcoderev.R

Step	Action
8	**Execute NB Classifier with MADlib Function Calls Within the Database:** In this step we will execute a SQL query that uses MADlib function calls for the NB classifier model and assignment of class labels. 1. Log into the VM assigned to you with "gpadmin" credentials 2. Navigate to /home/gpadmin/LAB08/ 3. Review the file NBmadlib.sql in this directory 4. The first part of the code categorizes the data and prepares a table that can be used with MADlib functions for NB classifier: `DROP TABLE IF EXISTS myschema.NBmdlib;` `CREATE TABLE myschema.NBmdlib (` `  attr INTEGER[]` `  , class INTEGER` `) DISTRIBUTED BY (class)` `;` `INSERT INTO myschema.NBmdlib (attr[1],attr[2],attr[3],class)` `SELECT` `  t1.age` `, t1.sex` `, t1.educ` `, t1.income` `FROM (` `  SELECT` `  CASE` `  WHEN age BETWEEN 20 AND 30 THEN 1` `  WHEN age BETWEEN 31 AND 45 THEN 2` `  WHEN age > 45 THEN 3` `  ELSE 0` `  END AS age` `  , CASE` `  WHEN sex = 1  THEN 1` `  WHEN sex = 2 THEN 2` `  ELSE 0` `  END AS sex` `  , CASE` `  WHEN educ > 14 THEN 1` `  WHEN educ BETWEEN 12 AND 14 THEN 2` `  WHEN educ < 12 THEN 3` `  ELSE 0` `  END AS educ` `  , CASE` `  WHEN inctot BETWEEN 10000 AND 50000 THEN 1` `  WHEN inctot BETWEEN 50000+1 AND 80000 THEN 2` `  WHEN inctot > 80000 THEN 3` `  ELSE 0` `  END AS income`

Step	Action
8 Cont.	

```
FROM
    persons
) AS t1
WHERE
  not (income = 0 OR age = 0 OR sex = 0)
   ;
```

5. The second part of the code builds the classifier model. Note here we have all the eligible rows in the table taken in for the training data set. Please review NB classifier documentation for the MADlib function at http://doc.madlib.net/v0.2beta/group__grp__bayes.html

```
DROP TABLE IF EXISTS myschema.nb_feature_probs;
DROP TABLE IF EXISTS myschema.nb_class_priors;

SELECT madlib.create_nb_prepared_data_tables(
  'myschema.NBmdlib', 'class'
, 'attr', 3, 'myschema.nb_feature_probs'
, 'myschema.nb_class_priors');

SELECT * FROM myschema.nb_feature_probs;
SELECT * FROM myschema.nb_class_priors;
```

6. Step 5 will take some time to execute. We now prepare some data to score the model. We essentially use the same code in step 4 but select 10 random records.

```
DROP IF EXISTS TABLE myschema.NBmdlib_test;
CREATE TABLE myschema.NBmdlib_test (
  id SERIAL
, attr INTEGER[]
, original_data INTEGER
) DISTRIBUTED BY (id)
;
INSERT INTO myschema.NBmdlib_test(
  attr[1]
, attr[2]
, attr[3]
, original_data
)
SELECT
  t1.age
, t1.sex
, t1.educ
, t1.income
FROM (
  SELECT
    CASE
      WHEN age BETWEEN 20 AND 30 THEN 1
      WHEN age BETWEEN 31 AND 45 THEN 2
      WHEN age > 45 THEN 3
      ELSE 0
    END AS age
```

Step	Action
8 Cont.	

```
, CASE
      WHEN sex = 1  THEN 1
      WHEN sex = 2 THEN 2
      ELSE 0
    END AS sex
  , CASE
      WHEN educ >14 THEN 1
      WHEN educ BETWEEN 12 AND 14 THEN 2
      WHEN educ <12 THEN 3
      ELSE 0
    END AS educ

, CASE
      WHEN inctot BETWEEN 10000 AND 50000 THEN 1
      WHEN inctot BETWEEN 50000+1 AND 80000 THEN 2
      WHEN inctot > 80000  THEN 3
      ELSE 0
    END AS income
  FROM
    persons
) AS t1
WHERE
   NOT (income = 0 OR age = 0 OR sex = 0)
ORDER BY RANDOM ()
LIMIT 10
;
```

7. We run the SQL code that predicts the classes for the test data we created in the previous step:

```
SELECT * from myschema.NBmdlib_test ORDER BY id;

DROP TABLE IF EXISTS myschema.nb_classify_view_fast;
DROP TABLE IF EXISTS myschema.nb_probs_view_fast;

SELECT madlib.create_nb_classify_view (
  'myschema.nb_feature_probs'
, 'myschema.nb_class_priors'
, 'myschema.NBmdlib_test'
, 'id', 'attr', 3
, 'myschema.nb_classify_view_fast'
);
```

Step	Action
8 Cont.	**SELECT * FROM myschema.nb_classify_view_fast ORDER BY key;** **SELECT madlib.create_nb_probs_view (** **'myschema.nb_feature_probs'** **, 'myschema.nb_class_priors'** **, 'myschema.NBmdlib_test'** **, 'id', 'attr', 3** **, 'myschema.nb_probs_view_fast');** **SELECT * FROM myschema.nb_probs_view_fast ORDER BY key,class;** All these code are available at /home/gpadmin/LAB08/NBmadlib.sql You can execute them with the following command **psql –d training2 –f NBmadlib.sql** Record your observations below:

End of Lab Exercise

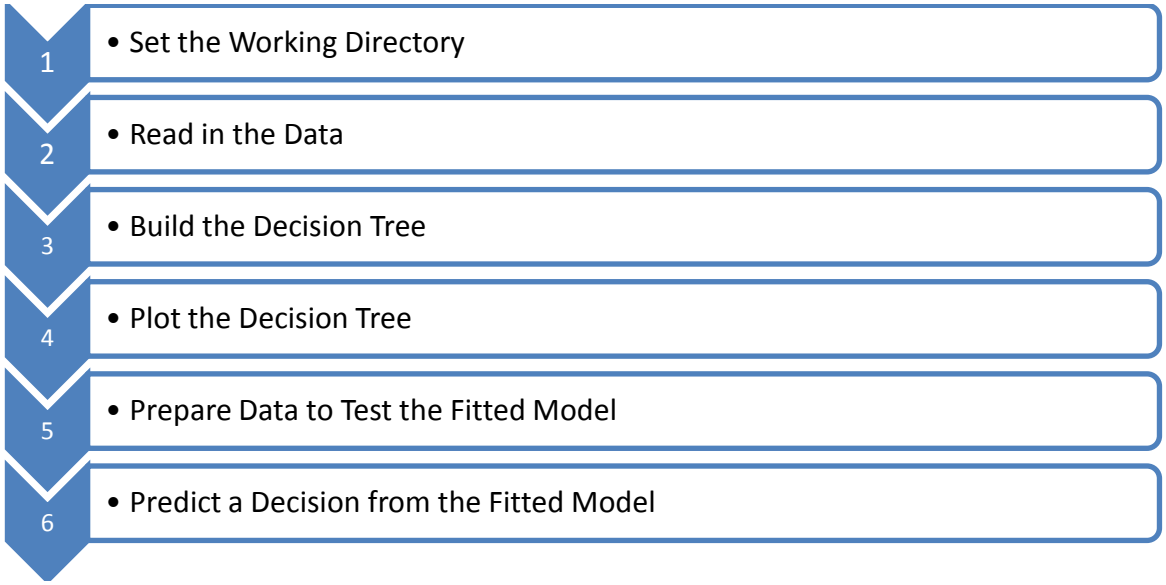
Lab Exercise 9: Decision Trees

Purpose:	This lab is designed to investigate and practice Decision Tree (DT) models covered in the course work. After completing the tasks in this lab you should able to: • Use R functions for Decision Tree models • Predict the outcome of an attribute based on the model

Tasks:	Tasks you will complete in this lab exercise include: • Use the R –Studio environment to code Decision Tree Models • Build a Decision Tree Model based on data whose schema is composed of attributes • Predict the outcome of one attribute based on the model

References:	References used in this lab are located in your ***Student Resource Guide Appendix***.

Workflow Overview



LAB Instructions

Step	Action
1	Log in with GPADMIN credentials on to R-Studio.
2	**Set the Working Directory:** 1. Execute the command: `setwd("~/LAB09")` (Or use the "Tools" option in the tool bar in the RStudio environment.) 2. Load the package rpart.plot and the associated libraries. If prompted for the location to download select any integer representing a location nearest to you. `> install.packages("rpart.plot")` `> library("rpart")` `> library("rpart.plot")`
3	**Read in the Data:** • Use a data table with columns for data attributes : Play, Outlook, Temperature, Humidity and Windy • A Decision Tree allows for predicting the values of the attribute Play, given that you know the values for attributes like Outlook, Humidity and Windy. 1. Read in the data from the "Dtdata.csv" file in the working directory and display the contents: `> #Read the data` `> play_decision <- read.table("DTdata.csv",header=TRUE,sep=",")` `> play_decision` 2. How many observations did you read in? 3. How many variables (attributes) did you read in? 4. Use the command "summary" for a detailed list of the table object you read in `summary(play_decision)` 5. Review the results. (The Summary is located in the console window.)

Step	Action
4	**Build the Decision Tree:** Use the "rpart" package in R for classification by Decision Trees. The RPart Programs build classification or regression models of a very general structure using a two stage procedure; the resulting models can be represented as binary trees. 1. Use the following rpart commands to grow a Decision Tree: `rpart (formula, data=, method=, control=)` • **formula** is in the format: outcome ~ predictor1+predictor2+predictor3+ect. • **data=** specifies the dataframe • **method=** "class" for a classification tree "anova" for a regression tree • **control=** optional parameters for controlling tree growth. For example, control=rpart.control(minsplit=30, cp=0.001) requires that the minimum number of observations in a node be 30 before attempting a split and that a split must decrease the overall lack of fit by a factor of 0.001 (cost complexity factor) before being attempted. The "Play" attribute is the outcome that will be predicted. 2. Use the command: `> fit <- rpart(Play ~ Outlook + Temperature + Humidity + Wind, method="class", data=play_decision,` `+ control=rpart.control(minsplit=1))` 3. You can now display "fit" and review the results: `> summary(fit)` Note that the leaf nodes information includes both the class label and the class probabilities (P(no), P(yes))
5	**Plot the Decision Tree:** 1. Review the arguments for rpart.plot function. Type in: `> ?rpart.plot` We will use the arguments "type" and "extra" in our plot. 2. Type in the following : `> rpart.plot(fit, type=4, extra=1)` 3. Review the Decision Tree plot on the graphics window.

Step	Action
6	**Prepare Data to Test the Fitted Model:** You must use "fit" for a new set of data to create predictions from the DT: Play Decision Outlook Temperature Humidity Wind ? rainy mild high FALSE 1. "newdata" is a data frame object and can be built for our test data. Type in the following statement: `newdata <- data.frame(Outlook="rainy",Temperature="mild",Humidity="high",Wind=FALSE)` 2. Review the "newdata" displaying the dataframe `> newdata` 3. The data displayed as follows: `Outlook Temperature Humidity Wind` `1 rainy mild high FALSE`

7	**Predict a Decision from the Fitted Model:** The "predict" function is used to generate predictions from a fitted rpart object. • "type" is a character string denoting the type of the predicted value • Use both "prob" and "class" to predict from a Decision Tree model `predict(object, newdata = list(),` `type = c("vector", "prob", "class", "matrix"))` 1. The **type="prob"** gives the class probabilities for the prediction for newdata Type in `> predict(fit,newdata=newdata,type="prob")` 2. Repeat the prediction with type="class" `> predict(fit,newdata=newdata,type="class")` Review the results. 3. What is the prediction for the "newdata"?
8	The code is available at /home/gpadmin/LAB09/DT.R

End of Lab Exercise

Lab Exercise 10: Time Series Analysis with ARIMA

Purpose:	This lab is designed to investigate and practice Time Series Analysis with ARIMA models (Box-Jenkins-methodology). After completing the tasks in this lab you should able to: • Use R functions for ARIMA models • Apply the requirements for generating appropriate training data • Validate the effectiveness of the ARIMA models

Tasks:	Tasks you will complete in this lab include: • Use the R –Studio environment to code ARIMA models • Use the ODBC connection to the database to create the weekly sales data from the retail database • Prepare the data (sorting and rendering the data as a Time series) • Generate a model and evaluate how well it predicts the results and compare the results with original data

References:	References used in this lab are located in your ***Student Resource Guide Appendix.***

Workflow Overview

1. Set the Working Directory
2. Establish the ODBC Connection
3. Open Connections to ODBC Database
4. Get Data from the Database
5. Review, Update, and Prepare DataFrame ”msales” File for ARIMA Modeling
6. Convert “sales” into Time Series Type Data
7. Plot the Time Series
8. Analyze the ACF and PACF
9. Difference the Data to Make it Stationary
10. Plot ACF and PACF for the Differenced Data
11. Fit the ARIMA Model
12. Generate Predictions
13. Compare predicted values with actual values

LAB Instructions

Step	Action
1	Log in with GPADMIN credentials on to R-Studio.
2	**Set the working directory** Set working directory to ~/LAB10/ , execute the command: `setwd("~/LAB10")` • (Or using the "Tools" option in the tool bar in the RStudio environment.)
3	**Establish the ODBC Connection:** Load the RODBC package using the following command: `library('RODBC')`
4	**Open Connections to ODBC Database:** 1. Before connecting to the ODBC database make sure the file, /etc/odbc.ini, is properly set to point to database "training1". 2. If not, edit the line that starts with, "Database =" within the file /etc/odbc.ini, to point to, "training1". 3. Ensure the username(uid) and password (pwd) are provided correctly in the following command: `ch <- odbcConnect("Greenplum",uid="gpadmin", case="postgresql",pwd="password_of_the_gpadmin_user")`

Step	Action
5	**Get Data from the Database:** 1. Drop the table, weekly_sales, from the schema ddemo: `sqlDrop(ch,"ddemo.weekly_sales")` 2. Execute an SQL query using the sqlQuery command, creating a table, weekly_sales, in which the weekly sales are grouped first by year and the week within a year: `sqlQuery(ch,` `"CREATE TABLE` `ddemo.weekly_sales (` `sale INTEGER,` `Y1 INTEGER,` `W1 INTEGER )` `DISTRIBUTED BY (sale);` `INSERT INTO ddemo.weekly_sales` `SELECT` `SUM((o.item_price*o.item_quantity)) as sale ,` `EXTRACT(YEAR FROM o.order_datetime) as y1,` `CASE` `WHEN (EXTRACT(WEEK FROM o.order_datetime) = 53) THEN 52` `ELSE` `EXTRACT(WEEK FROM o.order_datetime)` `END w1` `FROM` `ddemo.order_lineitems o` `GROUP BY` `y1,w1` `;` `"` • **)Note:** Note the use of the **EXTRACT** function to obtain the year and the week within the year from the order_datetime field • The sales number is accumulated for each week • The ISO Standard for numbering weeks within a year may lead to a year containing 52 or 53 weeks. • In order to work with Time Series data you need a consistent "periodicity" with the data. You must accumulate the vales for week 53 with that of week 52 in the same year. We use the CASE statement to designate the week 53 as 52 and cumulate the sales amount for week 53 into week 52 of the same year. 3. Get the results from the table into data frame msales. Execute the command: `msales <- (sqlFetch(ch,"ddemo.weekly_sales"))` 4. Close the ODBC channel. `odbcClose(ch)`

Step	Action
6	**Review, Update and Prepare DataFrame "sales" for ARIMA Modeling:** 1. Sort the data in the order of Year and Week: Use the R function "order" : `attach(msales)` `msales <- msales[order(y1,w1),]` `detach(msales)` 2. Extract 300 values from column 1 of "msales" for modeling and 12 values to compare with the predictions done by the model. Store them in two different vectors: "sales" and "csales". Use the following command: `>sales <- c(rep(0,300))` `>csales <- c(rep(0,12))` `>sales[1:300] <- msales[1:300,1]` `>csales[1:12] <- msales[301:312,1]`
7	**Convert "sales" into Time Series Type Data:** Convert the "sales" into a time series. • This "transformation" is required for most of the time–series functions, since a time series contains more information than the values themself, namely information about dates and frequencies at which the time series has been recorded. `> sales <- ts(sales,start=2005,frequency=52)`
8	**Plot the Time Series:** 1. Plot the Time Series using the following command: `plot(sales,type="l")` 2. Review the plot of the Time Series. 3. Identify the seasonality features in the graph. 4. Is the data Seasonal (Do you see patterns that repeat at a particular frequency)? 5. Is the data stationary? 6. Is there a trend to the data?

Step	Action
9	**Analyze the ACF and PACF :** The next step in analyzing time series is to examine the **autocorrelations (ACF)** and **partial autocorrelations (PACF).** R provides the functions acf( ) and pacf( ) for computing and plotting of ACF and PACF. 1. Use the **parameter function "par"** to set the plot window to display both the ACF and PACF plots. Plot ACF and PACF on the same graph using the command: `> par(mfrow=c(2,1))` `> acf(sales)` `> pacf(sales)` 2. Using the plot generated in the plot window: • Does the ACF tail off quickly? • What does the ACF indicate with respect to stationarity of the data? • What will you do to make the data stationary?
10	**Difference the Data to Make it Stationary:** To difference the data and make it stationary you should use the "diff" function in R. The "diff" function takes the pair of each observation and differences it from the one previous to it. 1. Run the following code: `> #Difference the series and plot it` `> sales1 <- diff(sales)` `> m <- length(sales1)` `> par(mfrow=c(1,1))` `> plot(1:m,sales1,type="l")` 1. Do you see any trend to the data now? 2. Is seasonality still shown in the data? 3. How does the trend of oscillating spikes change as you move from left to right on this plot?

Step	Action
11	**Plot ACF and PACF for the Differenced Data:** 1. Run the following code: `> #Plot ACF and PACF on the same graph` `> par(mfrow=c(2,1))` `> acf(sales1)` `> pacf(sales1)` 2. Do you see the ACF tailing off quickly?

Step	Action
12	**Fit the ARIMA Model:** Once you configure the data and review the seasonality elements, you are ready to fit an ARIMA model. Selecting the correct model to fit an ARIMA model is a bit of an art. Algorithms are used to select the correct parameters. • Use the data configuration and the principle of "parsimony" to fit a basic model • Use the statement ARIMA • Use the time series • Use "sales" and not the "diff(sales)" that we computed in step 10. ARIMA will automatically invoke the "diff" function based on the parameters specified. • You need to specify the order (p,d,q) where "p" is the order of AR , "q" order of MA and "d" is the number of differences. • Fit (1,1,0) so the model will difference once, use AR and MA parameters as 1 and 0. • Use a seasonal statement since the data seems to have a seasonal component to it as you see spikes every 52 weeks. • In the seasonal statement, you have a "list" where you put in the "order" and a (time) "period" which will be "52". • Use "include.mean = false" - R will force a mean and continue any trend seen in the past for the model. This automatically defaults to "True". You turn the automatic default off with this statement. • **Note:** You can also experiment with not using this statement during this lab. 1. Type in the following code: `sales.fit <- arima (sales,` `order=c(1,1,0),` `seasonal = list(order=c(1,1,0),period=52),` `include.mean=FALSE)` `> sales.fit` 2. Review the output displayed. 3. Record the coefficients for the AR term and the seasonal AR terms and the standard errors. 4. What is your observation on the standard errors compared to the coefficients? • **Note:** The model gives you the "log likelihoods" that provide important input on model selection.

Step	Action
13	**Generate Predictions:** 1. Use the fitted model "sales.fit" and the "predict" statement for 12 periods ahead: `sales.predict <- predict (sales.fit, n.ahead=12)` • **Note:** You can see the predictions by typing "sales.fit". 2. Plot the predictions using plot statements: `> par(mfrow=c(1,1))` `> plot (sales,xlim=c(2009.50,2011))` `> lines (sales.predict$pred,col="blue")` `> lines (sales.predict$pred+2*sales.predict$se,col="red")` `> lines (sales.predict$pred-2*sales.predict$se,col="red")` • **Note:** The first plot statement plots the original "sales" data • EMC has restricted the "xlim" value to zoom in on the values just prior to the predicted values • The "blue" line indicates the mean of the predictions • The "red" lines denote the upper and lower bounds of the prediction
14	**Compare predicted values with actual values** 1. Compare the predicted values with the actual values and plot the actual Values Predicted as a barplot. Use the following code: `> #comparing the predictions with the actual values` `> par(mfrow=c(1,1))` `> forbar <- matrix(x,ncol=12,byrow=TRUE)` `> colnames(forbar)<- asales[1:12,3]` `> rownames(forbar)<- c("Actual","Predicted")` `> barplot(forbar,beside=TRUE,` `+ main="Actual Vs Predicted",` `+ ylab="Weekly Sales",` `+ col=rainbow(2),` `+ xlab="Weeks 2010")` 2. Review the plot. • **Note:** The complete code is available in the folder /home/gpadmin/LAB10/arima.R

End of Lab Exercise

Lab Exercise 11: Hadoop, HDFS and MapReduce

Purpose:	This lab introduces the *Hadoop and MapReduce environment that* you will be working on for the next lab. After completing the tasks in this lab you should able to: • Get help on the various Hadoop commands • Observe a MapReduce job in action • Query various Hadoop servers regarding status

Tasks:	Tasks you'll be completing in this lab include: • Run Hadoop and Hadoop fs and collect help information • Run a shell script to perform a word count activity • Run a MapReduce job to produce similar output • Investigate the UI for MapReduce/HDFS components to track system behavior

References:	References used in this lab are located in your ***Student Resource Guide.*** See the Guide for: • Hadoop Commands • HDFS Commands

Workflow Overview

1. Check your Hadoop Environment
2. Get Help
3. Load data into HDFS
4. Execute MapReduce
5. Check on MapReduce and HDFS Progress

Step	Action
1	**Check your Hadoop Environment:** 1. Start a terminal session to your "be" host – you may use PuTTy on your "fe" host for this. Log in with username "gpadmin" and the gpadmin password supplied by your instructor. Make sure you are connected to the right directory. Execute the command: `cd ~/LAB11` 2. Execute the command: `printenv | grep '^H'` 3. Are there any Hadoop variables defined? Which ones? 4. Execute the command: `hadoop fs –ls` 5. What do you see? 6. Execute the command: `ls` This should be different
3	**Get Help:** 1. Execute the command: `hadoop –help 2>&1 | tee Hadoop.hlp` 2. Execute the command: `hadoop fs –help | tee HDFS.hlp` 3. Now list the files by executing the commands: `clear && more Hadoop.hlp` `clear && more HDFS.hlp` These files are also contained in your Student Resource Guide. These commands can be run from your command shell whenever you need them. 4. Now we create an alias so we don't have to type "hadoop fs" every time `alias hdfs="hadoop fs"`

Step	Action
4	**Load data into HDFS:** In this step, we will be loading an input data file into HDFS that we will be using in later activities. 1. Execute the following commands. **hdfs -copyFromLocal speech.txt input/speech.txt** **hdfs -ls input** The file *speech.txt* is now in HDFS.
5	**Execute MapReduce:** In this step, you will run a MapReduce job and observe its output. This job is identical to the example job discussed as part of the lecture. 1. First, execute the command **time ./wf.sh speech.txt** 2. How long did it take to produce its output? _____ 3. List the file "MRwordcount.sh" by executing **more MRwordcount.sh** This should be identical to the file used as an example in the lecture. 4. Execute the following command: **time ./MRWordCount.sh** 5. What do you see? _____ 6. How long did it take to execute this script? _____ 7. Is this command slower than the Unix script file? Why do you think that is? _____ 8. Looking at the output The output of the MapReduce job is stored in a subdirectory of the output directory in HDFS. This directory is named "d" followed by a string of numbers (and is listed in the output of the MapReduce command). You can see the content of the directory by executing the following command: **hdfs -cat output/d*/part-***

Step	Action
6	**Check on MapReduce and HDFS Progress:** In this step, we will look at some of the status information about MapReduce and Hadoop. First we look at the administrative interfaces for the JobTracker and the TaskTracker components of the MapReduce framework, and then NameNode User Interface (UI) for HDFS. 0.1. Create a shortcut on your desktop for each UI that you will be investigating. For each of the following steps, right-click on your Windows desktop and select New>>Shortcut. Enter the following URL: http://<IP-ADDRESS-OF-SERVER>":"<PORT_NUMBER>/ For the JobTracker, PORT_NUMBER will be 50030. IP-ADDRESS-OF-SERVER is the IP address for your server that you received at the beginning of the course. Click "Next." Name the shortcut "JobTracker" and click "Finish" 0.2. Do the same for the NameNode (port number 50070) and for the TaskTracker (port # 50060). 1. Click on the desktop icon labeled "Jobtracker". This will bring up the UI for the JobTracker MapReduce node. 2. Examine the output. Do you see anything similar to what you saw in the output of the script you just ran? Anything different? 3. Click on the desktop icon labeled "TaskTracker" This will bring up the UI for the MapReduce TaskTracker node. 4. Examine the output. Do you see anything similar to what you saw in the output of the script you just ran? Anything different? Now we look at the UI for the NameNode node in HDFS. 5. Click on the desktop icon labeled "NameNode" This invokes the UI for the NameNode node in an HDFS implementation. Examine the output. If you are strictly working as an analyst, you may never need to look at the administrative interface to these components of the Hadoop framework. On the other hand, if things aren't working out as you might have expected, you can use these interfaces to take a deeper look "under the hood" at the mechanics of Hadoop.

End of Lab Exercise

Lab 12: In-database Analytics

Purpose:	This lab is designed to familiarize you and give you practice with the in-database analytics methods covered in lessons three and four of Module 5 After completing the tasks in this lab you should able to: • Use window functions • Implement user defined aggregates and user defined functions • Use ordered aggregates • Use Regular Expressions (Regex) in SQL for text filtering • Use MADlib functions and plot results from MADlib function outputs

Tasks:	Tasks you'll be completing in this lab include: • Process Clickstream analysis data using window functions, User defined functions, User defined aggregates and regular expressions • Compute median household income using ordered aggregates • Use MADlib functions for logistic regression and direct output to plot the results

References:	Student resource guide http://doc.madlib.net/v0.2beta/group__grp__logreg.html

Part 1 – In-database analysis of Click-Stream data

Workflow Overview

1. Define problem – Clickstream Analysis
2. Validate how the window function works
3. Use Regular expression
4. Develop a final query assuming the user defined aggregate is available
5. Define data type
6. Develop the user defined aggregate “clickpath”
7. Create PREFUNC window_exclusion
8. Create the FINALFUNC clickpath_final – Boolean evaluator
9. Create SFUNC clickpath_transition – the aggregator
10. Put them all together

LAB Instructions

Step	Action
1	**Define problem - Clickstream Analysis** Problem Definition: A users' click-stream is defined as the aggregate of all activity a user has through a website via their clicks, derived through the web logs. This has become an important view of the data, as it enables insights into typical paths a user takes to navigate a website of interest. Analysis of click-streams can help to improve the usability of the websites, identify hacking attempts on the website, etc. In this lab, we will be constructing and analyzing click-streams from pre-processed weblog data. You are provided with data in a database table called "clicks". The table is defined as follows: TABLE clicks(user_id BIGINT, timestamp BIGINT, page_type VARCHAR) Where userid : user session number, page_type : identification of the page visited timestamp : the time of the visit. We want to determine which users: – Start at the home page, – **then** Click on an auction, – **then** View **at least one** help page – **then** Place a bid **In this lab you will connect to "module5indb" database and all the data tables required for this lab are available in this database.**

Step	Action
2	**Validate how the window function works** Key in the following code and test how the windows function works: `SELECT` `  sid` `, page_type` `, time` `, count(*) OVER (prefix) AS seq_length` `, count(*) OVER (PARTITION BY sid) AS max_seq_length` `FROM` `  clicks` `WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)` `LIMIT 50` `;` The SELECT statement selects from table "clicks", Session_id, Page_type, and the time stamp. Two standard "count" aggregate functions (which return the count of all records), are also included in the SELECT statement. The first one is defined as "sequence length" and the second one is defined as maximum sequence length. The first "count" aggregate is cumulated over a window defined as "prefix"; "prefix" is partitioned by variable "session id" and ordered by "time" (in a ascending order). For example if "session_id" = "1" had 10 different clicks at different times, your output for seq_length will be the sequence number of the clicks in the ascending order of time in session_ id = "1". If there are 10 clicks that session you will have 10 rows in the output. The second aggregate is cumulated over the partition defined by session id, the second "count" aggregate in our example will be 10 as there are 10 clicks for "session_id" =1. Execute the code and observe the results. We have limited the output to 50 rows.

Step	Action
3	**Use Regular expression** Check through the window defined as “prefix” and determine if the user went through a particular sequence of “page_types”. We want to know if the user (defined by the session_id): a) Starts at the home page b) Then clicks on an auction c) Then views at least one help page d) Then places a bid Define an aggregate that will step through the window “prefix” and look at the page types at every record in the window. If we call our pages with notation S,A,H,B we are looking for a sequence in regular expression terms “^SAH+B”. (defined with a variable “pattern”) Extract the first character of page_type (use upper case) and build a sequence of the page_type characters and compare this with our regular expression string “^SAB+H”. The code to perform the above mentioned tasks: `SELECT` `  sid` `, page_type` `, time` `, count(*) OVER (prefix) AS seq_length` `, count(*) OVER (PARTITION BY sid) AS  max_seq_length` `, upper(substring(page_type for 1)) AS mystring` `, '^SAH+B' AS pattern` `FROM` `  clicks` `WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)` `LIMIT 50` `;` Review the output.

Step	Action
4	**Develop a final query assuming the user defined aggregate is available** The output of the column is the first character of "page id". As you step through each time stamp of the "preview" window, aggregate the first characters at each pass. This aggregated character set is compared with the "pattern" "^SAH+B". **Write a user defined aggregate** that will accumulate the text string on each step it traverses in the window and return a Boolean value "true" or "false" based on the match with the pattern. Call this function "clickpath" and the arguments for this function are • the upper cased first character of the page_type and • the regular expression "pattern" `clickpath(upper(substring(page_type for 1)), '^SAH+B' )` This function should work as an aggregate over the window "prefix" accumulating the first character and determining the Boolean value of match. Our final query code (assuming clickpath works the way it is intended) will be: `SELECT` `  sid` `FROM (` `  SELECT` `  sid` `  , page_type` `  , time` `  , clickpath(upper(substring(page_type for 1)),'^SAH+B'` `  ) OVER (prefix) AS match` `  , count(*) OVER (prefix) AS seq_length` `  , count(*) OVER (PARTITION BY sid) AS max_seq_length` `  FROM` `  clicks` `  WINDOW prefix AS (PARTITION BY sid ORDER BY time ASC)` `  ) AS subq` `  WHERE` `  seq_length = max_seq_length` `  AND match = true` `  ;`

Step	Action
5	**Define data type** Define a composite data type that you will use with the aggregation function. Our composite data type will consists of • the sequence we are aggregating and • a regular expression "pattern" (which does not change) that we will use for comparison. Create data type with the following code: `DROP TYPE IF EXISTS clickstream_state CASCADE;` `CREATE TYPE clickstream_state AS (` `  sequence VARCHAR` `, pattern VARCHAR` `);`
6	**Develop the user defined aggregate "clickpath"** There are two major functions of "clickpath" • It should aggregate the characters (transition function that aggregates) • It should compare and return a Boolean function (the final function that returns the Boolean value) Key in the following code: `DROP AGGREGATE IF EXISTS clickpath(` `  /* Symbol */ CHAR` `, /* regex */ TEXT` `  );` `CREATE AGGREGATE clickpath(` `  /* Symbol */ CHAR` `, /* regex */ TEXT)` `(` `  STYPE = clickstream_state,` `  SFUNC = clickpath_transition,` `  FINALFUNC = clickpath_final,` `  PREFUNC = window_exclusion` `);` **Note:** The STYPE is the data type we defined in step 5. We have to now create two functions: • clickpath_transition (the aggregator) and • clickpath_final (the Boolean evaluator) Notice that we also defined a PREFUNC, a function required to enable the function clickpath to be called as a window function.

Step	Action
7	**Create PREFUNC window_exclusion:** `CREATE OR REPLACE FUNCTION window_exclusion(clickstream_state, clickstream_state) RETURNS clickstream_state AS $$ BEGIN RAISE EXCEPTION 'aggregate may only be called from a window function'; END; $$ LANGUAGE PLPGSQL STRICT;`
8	**Create the FINALFUNC clickpath_final - Boolean evaluator:** The Boolean evaluator is the simpler of the two remaining functions. `CREATE OR REPLACE FUNCTION clickpath_final(state clickstream_state) RETURNS BOOLEAN AS $$ SELECT $1.sequence ~ $1.pattern; $$ LANGUAGE SQL STRICT;` The sequence and the pattern are matched and the Boolean value is returned. $1 refers to the first and the only argument in the function call. Recall the composite data type we created has both sequence and pattern.

Step	Action
9	**Create SFUNC clickpath_transition – the aggregator** The next and the last function to define is the aggregator. This function has three arguments. • The "state" which aggregates with every step, • The "symbol", the character we read in from the current row • The pattern to match When you step into a new window, the "state" will be NULL and it will take in the first character. As we step through each row within the window the aggregation will be carried out. Code the function as follows: `CREATE OR REPLACE FUNCTION clickpath_transition(` `  state clickstream_state, symbol CHAR(1), pattern VARCHAR)` `RETURNS clickstream_state AS $$` `    SELECT CASE` `        WHEN $1 IS NULL THEN ($2, $3)::clickstream_state` `        ELSE ($1.sequence \|\| $2, $3)::clickstream_state` `    END;` `$$ LANGUAGE SQL CALLED ON NULL INPUT;`
10	**Put them all together**: 1. Start with the definition of data type (step 5) 2. Code the three functions SFUNC, FINALFUNC and PREFUNC (Steps 8,9,7) 3. Complete the user defined aggregate (step6) 4. Run the query (step4) The segments of this code are available in **/home/gpadmin/LAB12/clickstream_step*.sql** (* represents the steps in the document). If you have not completed the code as you reviewed the Lab, compile them in to one file and execute the query with the following command: `psql  -d module5indb –f your_code.sql`

End of Lab Exercise

Part 2 – In-database computation of Median with Ordered Aggregates

Workflow Overview

1. Define Problem
2. Compute the median household income using ordered aggregates
3. Expand the code in step 2 to report median income by “state”

LAB Instructions

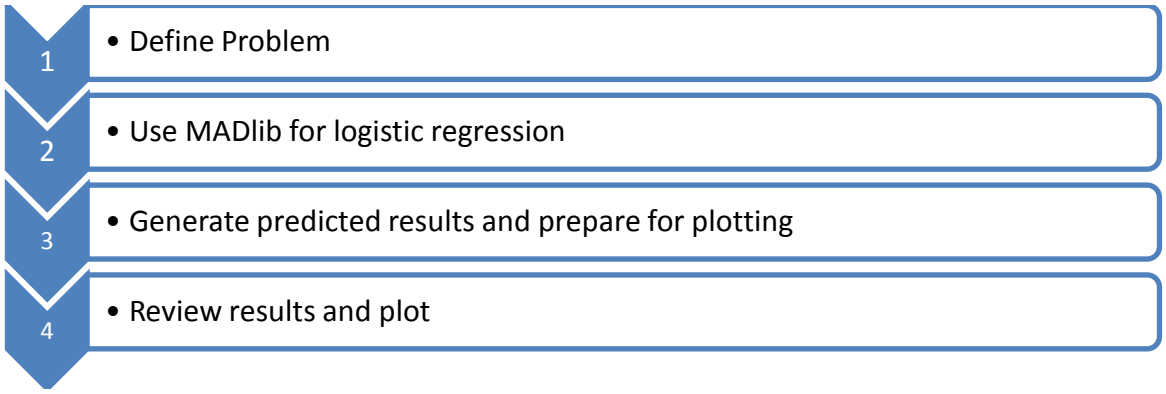
Step	Action
1	**Define Problem:** Use the housing table in training2 database (census) to compute the median household income for each state.
2	**Compute the median household income using ordered aggregates** Use ordered aggregates for the computation of median household income. Code suggestion: `SELECT` `  ( arr[ length/2 + 1 ] + arr[ (length + 1)/2 ] ) / 2.0 AS median_income` `FROM(` `  SELECT` `  array_agg(hinc ORDER BY hinc) AS arr` `  , count(*) AS length` `  FROM` `  housing` `  ) AS q` What is the overall median household income in the US?

Step	Action
3	**Expand the code in step 2 to report median income by "state":** Execute the following code: `SELECT` `  f.name` `, ( arr[ length/2 + 1 ] + arr[ (length + 1)/2 ] ) / 2.0 AS median_income` `FROM(` `  SELECT` `  state AS s` `  , array_agg(hinc ORDER BY hinc) AS  arr` `  , count(*) AS length` `  FROM` `  housing` `  GROUP BY` `  state` `  ) AS q` `JOIN` `  fips f` `ON` `  s = f.code` `ORDER BY` `  f.name` `;` What is the median income of Massachusetts and Alaska? The code in step 4 is available at /home/gpadmin/LAB12/median.sql

End of Lab Exercise

Part 3: Logistic Regression with MADlib

Workflow Overview



Step	Action
1	**Define Problem:** In this exercise you will use the MADlib function for logistic regression and generate the model and plot the predicted results. Synthetic data is available in the table "artificiallogreg" in database "module5indb"
2	**Use MADlib for logistic regression** Execute the following code to generate the model and store the results in a table "logr_coef" `DROP TABLE IF EXISTS logr_coef;` `CREATE TABLE logr_coef AS` `  SELECT 0::INT AS bla` `, NULL::FLOAT8[] AS coef` `DISTRIBUTED BY (bla)` `;` `UPDATE logr_coef SET coef = (SELECT coef FROM madlib.logregr('artificiallogreg', 'y', 'x', 20, 'irls', 0.001) AS coef)` `;`
3	**Generate predicted results and prepare for plotting** Generate the predicted results; organize them in ascending order of value of x. Pipe the results using meta commands "\o" to a file called "graphics.txt" that we can use to plot in the next step: `\a` `\o graphics.txt` `SELECT` `  DISTINCT rank::FLOAT8/total_count AS x` `, count::FLOAT8/total_true AS y` `FROM (` `  SELECT` `  y` `  , rank() OVER (ORDER BY prediction DESC)` `  , count(*) OVER () total_count` `  , count(*) FILTER (WHERE y = TRUE) OVER (ORDER BY prediction DESC)` `  , count(*) FILTER (WHERE y = TRUE) OVER () AS total_true`

Step	Action
3 cont'd	`FROM (` `SELECT` `r.*` `, 1. / (1. + exp(-dotProduct(r.x, c.coef))) AS prediction` `FROM` `artificiallogreg AS r` `CROSS JOIN` `logr_coef as c` `) q` `) p` `;` `\o`
4	**Review results and plot** Review the results in the file "graphics.txt". You can use "GNUplot" or "R" or even EXCEL to plot the results. This task is left un-scripted as a student exercise.
5	The code in step 2 and 3 above is available in file /home/gpadmin/LAB12/logistic.sql

End of Lab Exercise

Final Lab Exercise on Big Data Analytics

Purpose:	This lab allows students to apply what they have learned from the analytical methods and tools to a big data problem using the Analytics Lab Environment.

Tasks:	Tasks you will complete in this lab exercise include: • Explore the big data set provided and prepare the data for analysis • Assess data quality, outliers and training sets • Conduct model selection, code, execute and score the model • Use R and PSQL statements during your analysis of big data • Create a narrative summary of your findings, using the methods covered earlier in this module

References:	References used throughout the labs are located in your ***Student Resource Guide Appendix***. See the Appendix for: • http://www.ffiec.gov/hmda/
Working directory:	The directory /home/gpadmin/FINAL_LAB in your lab environment will be your working directory for the final lab exercise. Following files are pre-loaded in this lab: Analyst.ppt – Analyst presentation template Sponsor.ppt – Sponsor presentation template *.asc – encrypted files with suggested code for the solution. The decrypting of these files are performed with the following command at the $ prompt in the FINAL_LAB directory: `gpg -o *.* -d *.*.asc` (*.* represents the filename with extension name) You will be prompted for a passphrase. Your instructor will provide the pass phrase

Case Study Background and Problem Definition

Scenario	A financial planning company, FPC would like to expand the set of services they offer by creating an online site for loan advice. Potential home loan borrowers can enter information about their personal finances and the kind of home loan they want, and the site will return the probability of getting such a loan, along with some general advice about how to increase their likelihood of success. For example, the advice could be: "Increase the down payment so as to decrease the loan amount by X dollars"; or "Consider a home in the price range Y"; "Are you eligible for a particular type of loan?", or "Can you add a co-signer to the loan?". The company hopes this online service will be a lead-in for customers to come to FPC for more focused, personal financial planning to achieve their life goals. FPC would also consider partnering with a real estate broker to showcase houses to potential homebuyers. FPC realizes that the customers are looking for fast responses and the online-service must provide an answer within 45 seconds. FPC plans to enter into a service level agreement with the partner websites such as those managed by the real estate brokers. Ideally, the model behind this advice site can give reasonable, grounded predictions. Of course, the site cannot ask applicants to fill out an entire loan application and the sensitive data it contains, such as credit scores, employment history, or existing debt. The FPC project stakeholders want to stick to basic, easy to enter information such as applicant income, loan type, loan size, and the location of the property (ZIP code). They recognize that a model with only that information can only give general advice, rather than truly precise predictions. We have a set of data that can support this approach, and allow making predictions based on the information above.

Issues to Address	A number of issues came up during the kick-off meeting for the project: 1. Should there be one big model, or separate models for different types of loans? 2. Someone in the group wondered if personal demographic information (sex, gender, and ethnicity) would improve the prediction. The others are hesitant about the idea of asking such questions on the site, but agreed to explore whether knowing that information would improve the model. 3. Someone else offered the opinion that giving the users raw probabilities would not be meaningful to them. She suggested that the model should set thresholds, and deliver qualitative messages instead, such as the following: • If the model reports that the probability of getting a loan were greater than 75%, then the system would send the user a message such as: "Congratulations! You have a very good chance of getting your loan!" • For probability less than 50%: "Sorry. Looks like the chances aren't so good," with a link to FPC's advice page. • For probability between 50-75%: "Your chances aren't the strongest. Come talk to us about developing a plan to improve your chances of getting financing." This work led the group to a metric for measuring model performance. Of the people who score > 75%, do more than 75% of them actually get a loan? Likewise for people who score less than 50%, how many of them are actually get loan? Also, how many people in the general population get each message? For instance, does the entire population score more than 75%?
Data Scientist Goals	Your goal, as the data scientist on this project, is to answer the following questions: 1. Would it be more effective to develop different models or one model? Why? If different models, focus on a single one for the initial study. 2. Should we ask for personal demographic information, or can we build a good enough model without it? 3. How accurate is the model, in terms of the thresholds that the stakeholders set in their discussion (75% and 50%)? What is the coverage of the threshold regimes? 4. Provide suggestions for the kind of general advice FPC can put on their advice page.

Considerations for developing an Analytic plan	• Consider the scope of the data you will need to include in the analysis, and filters you may need to set to construct the data set for your analysis. • Consider the types of models best suited to perform the analysis needed for the new website engine. Does this scenario represent a classification, clustering, or prediction problem? • Examine the distribution of data, such as loan data for home improvement, home purchase, and refinancing loans, to identify the influences on how you will select and create the model • Look at creating several models and compare them in terms of ROC/AUC, or other performance metrics. • Find ways to examine how robust the model is with the help of a confusion matrix or similar diagnostic technique. • Give thought to how you would portray this information to business stakeholders as well as an analytical audience. • Consider the Service Level Targets that FPC can offer to their end users when they score the model with their inputs • Consider Service Level Targets that you can provide to FPC in terms of computational resources required for model generation and validation. • Provide some suggestions for the kind of general advice FPC can put on their advice page, based on the results from your modeling exercise. For instance, mention the types of things an applicant can do to increase their likelihood of success when applying for a loan on the website.

Suggested Workflow and Checkpoints for the Lab

Suggested Workflow

Checkpoint 1

1. The data for this lab is the housing loan database assembled by federal agencies pursuant to the Home Mortgage Disclosure Act (HMDA). This database identifies the census tract location of almost every housing loan and housing loan application made in the United States each year. The data provided for analysis in this lab is an extract for the year 2010. The data is organized in three database tables larDB1,larDB2,larDB3 (in database "hmdalab") for different states as follows:

larDB1	larDB2	larDB3
AK	AL	CT
AZ	AR	DC
CA	CO	DE
HI	GA	FL
ID	IA	MA
MN	IL	MD
MT	IN	ME
ND	KS	Na
NM	KY	NC
NV	LA	NH
OR	MI	NJ
SD	MO	NY
UT	MS	PA
WA	NE	RI
WI	OH	SC
WY	OK	VA
	PR	VT
	TN	
	TX	
	WV	

2. The tables provide the HMDA Loan Application Registration (lar) details and they have the following structure:

```
As_of_Year INTEGER,
Respondent_Id VARCHAR(10),
Agency_Code VARCHAR(1),
Loan_Type INTEGER,
Property_Type VARCHAR(1),
Loan_Purpose INTEGER,
Occupancy INTEGER,
Loan_Amount_inK INTEGER,
Preapproval VARCHAR(1),
Action_Type INTEGER,
MSAMD VARCHAR(5),
State_Code VARCHAR(2),
County_Code VARCHAR(3),
Census_Tract_Number VARCHAR(7),
Applicant_Ethnicity VARCHAR(1),
Co_Applicant_Ethnicity VARCHAR(1),
Applicant_Race_1 VARCHAR(1),
Applicant_Race_2 VARCHAR(1),
Applicant_Race_3 VARCHAR(1),
Applicant_Race_4 VARCHAR(1),
Applicant_Race_5 VARCHAR(1),
Co_Applicant_Race_1 VARCHAR(1),
Co_Applicant_Race_2 VARCHAR(1),
Co_Applicant_Race_3 VARCHAR(1),
Co_Applicant_Race_4 VARCHAR(1),
Co_Applicant_Race_5 VARCHAR(1),
Applicant_Sex INTEGER,
Co_Applicant_Sex INTEGER,
Applicant_Income_inK VARCHAR(4),
Purchase_Type VARCHAR(1),
Denial_Reason_1 VARCHAR(1),
Denial_Reason_2 VARCHAR(1),
Denial_Reason_3 VARCHAR(1),
Rate_Spread VARCHAR(5),
HOEPA_Status VARCHAR(1),
Lien_Status VARCHAR(1),
Edit_Status VARCHAR(1),
Sequence_Number VARCHAR(7),
Population VARCHAR(8),
Minority_Population_pct VARCHAR(6),
HUD_Median_Family_Income VARCHAR(8),
Tract_To_MSAMD_Income_pct VARCHAR(6),
Number_of_Owner_occupied_units VARCHAR(8),
Number_of_1_to_4_Family_units VARCHAR(8),
Application_Date_Indicator INTEGER);
```

3. All the required codes for the modeling exercise are made available in different tables as detailed below:

Table name	variable defined
action	**Action_Type**
counties	**County_Code**
ethnicity	**Applicant_Ethnicity**
fips	**State_Code**
inst	Institution Record format
lienstatus	**Lien_Status**
loanpurpose	**Loan_Purpose**
loantype	**Loan_Type**
msamd	MSAMD office format
preapproval	**Preapproval**
race	**Applicant_Race_1**
sex	**Applicant_Sex**

Property type is not coded in a table, but has code definitions as follows:

1: 1 to 4 family
2: Manufactured housing
3: Multi-family

Occupancy = 1 indicates owner occupied housing (our focus of analysis)

4. For your analysis you are required to select
    a. A single state
    b. Occupancy = 1
    c. Property_Type = 1
    d. Action_Type <= 4
5. Extract data from the "lar" table (with the conditions in step 4) and create a table with the following variables:

```
Loan_Type VARCHAR(20),
Loan_Purpose VARCHAR(25),
Loan_Amount_inK INTEGER,
Preapproval VARCHAR(25),
Action_Type VARCHAR(25),
County_Name VARCHAR(50),
```

	Applicant_Ethnicity VARCHAR(25), Co_Applicant_Ethnicity VARCHAR(1), **Applicant_Race_1 VARCHAR(25),** **Applicant_Sex VARCHAR(25),** Applicant_Income_inK VARCHAR(4), Rate_Spread VARCHAR(5), HOEPA_Status VARCHAR(1), **Lien_Status VARCHAR(25),** Minority_Population_pct VARCHAR(6), HUD_Median_Family_Income VARCHAR(8), Tract_To_MSAMD_Income_pct VARCHAR(6), Number_of_Owner_occupied_units VARCHAR(8) The highlighted variables must be expanded to the values corresponding to the codes in the "lar" table.
Suggested Workflow **Checkpoint 2**	1. You can read the table created in Checkpoint 1 in your RStudio environment. Ensure the database "hmdalab" is specified in the "odbc.ini" file. When you read in the table with sqlFetch make sure that as.is=T is specified so that you have control in specifying the variables you want to treat as factors. 2. In R, use the factor function to specify those variables highlighted in step 5. 3. Convert the income, rate_spread and other variables to Numeric. Check for "NA" in the data 4. Use Table Function to tabulate values (number of records) with specific value combinations. You can also plot these data to begin exploring the variables. For instance, compare action_type vs. Rate_spread. 5. Remove records without income info (income="NA") 6. Code appropriate "releveling" for variables to explain the models in subsequent analysis 7. Define HUD_Median_Family_Income as numeric. Check for NAs 8. Remove rows with nulls (NA) in Tract_To_MSAMD_Income_pct, Minority_Population_pct, Tract_To_MSAMD_Income_pct 9. Visualize the variables. You should visualize all (or many, at least) of them. Check whether the loan amount has any odd, multi-modal distribution. This may suggest to us that we might want to build separate models for the different loan purposes. 10. Look for spikes in the outputs. You may need to develop one model with all the data below that spike and develop a separate model with the data beyond that spike. 11. Optionally eliminate some of the very small loans that trail out on the right. Subset data based on only those "action_types" you need for your model

Suggested Workflow **Checkpoint 3**	1. Finalize your dependent variable (Hint: approved implied by action_type==originated) and the drivers. First model with all drivers 2. Use the Log of monetary variables 3. Model with 10% of data (small set) Hint use: `prob1data$gp = runif(dim(prob1data)[1])` `smallset = subset(prob1data, prob1data$gp < 0.1) # 10% of data` 4. Execute the chosen model with the smallset 5. Analyze the results 6. Drop/change variables (such as including or excluding personal demographic information) and repeat steps 3,4 and 5 7. Predict using different models (with and without personal demographic info) 8. Plot the ROC curves for all the models predicted 9. Compare the AUC for the different models
Suggested Workflow **Final Presentation**	1. Look at the original questions you had when you started. Can you answer the questions we started with? 2. Examine the probability thresholds suggested by marketing. Do the evaluations on the "hold out" set. 3. Use a confusion matrix to determine the probability of approval in each bin (high, medium and low probability) 4. Use the presentation template to compile your results and develop your narrative summary 5. Present your findings to the class.

End of Lab Exercise