

STATS 202A – FINAL PROJECT TASK 2

NAME: ANOOSHA SAGAR

STUDENT ID: 605028604

R

OVERVIEW

R is an open source language and environment for statistical computing (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, etc.) and graphics. It is the number 1 choice of data scientists and is supported by a vibrant and talented community of contributors.

FEATURES OF R

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities
- R provides a suite of operators for calculations on arrays, lists, vectors, and matrices
- R has a large, coherent and integrated collection of tools for data analysis
- R has an effective data handling and storage facility
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers

COMMON COMMANDS AND OPERATIONS

VECTORS

Creating vectors

<code>c(1, 2, 3)</code>	Joins elements into a vector
<code>1 : 7</code>	An integer sequence (1 2 3 4 5 6 7)
<code>rep(1:2, times=3)</code>	Repeat a vector (1 2 1 2 1 2)
<code>rep(1:2, each=3)</code>	Repeat elements of a vector (1 1 1 2 2 2)

Sorting a vector x: `sort(x)`

Selection and slicing vectors

<code>x[4]</code>	The fourth element
<code>x[-4]</code>	All but the fourth element
<code>x[2:4]</code>	Elements two to four
<code>x[-(2:4)]</code>	All elements except two to four
<code>x[c(1, 5)]</code>	Elements one and five
<code>x[x == 8]</code>	Elements which are equal to 8
<code>x[x < 3]</code>	All elements which are less than 3
<code>x['apple']</code>	Element with the name 'apple'

PROGRAMMING

For loop

```
for (variable in sequence){  
  Do something  
}
```

While loop

```
while (condition){  
  Do something  
}
```

If statement

```
if (condition){  
  Do something  
} else {  
  Do something different  
}
```

Functions

```
function_name <- function(var){  
  Do something  
  return(new_variable)  
}
```

READING AND WRITING DATA


Input	Output	Description
<code>df <- read.table('file.txt')</code>	<code>write.table(df, 'file.txt')</code>	Read and write a delimited text file
<code>df <- read.csv('file.csv')</code>	<code>write.csv(df, 'file.csv')</code>	Read and write a comma separated value file
<code>load('file.Rdata')</code>	<code>save(df, file='file.Rdata')</code>	Read and write an R data file


MATH FUNCTIONS


<code>log(x)</code>	Natural log	<code>sum(x)</code>	Sum
<code>exp(x)</code>	exp	<code>mean(x)</code>	Mean
<code>max(x)</code>	Largest element	<code>median(x)</code>	Median
<code>min(x)</code>	Smallest element	<code>round(x, n)</code>	Round to n decimal places

MATRICES

`m <- matrix(x, nrow = 3, ncol = 3)` : creates a matrix from x

 `m[2,]` - Select a row

 `m[, 1]` - Select a column

 `m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: $m * x = n$

STATISTICS

<code>lm(x ~ y, data=df)</code>	Linear model
<code>glm(x ~ y, data=df)</code>	Generalized linear model
<code>summary</code>	Get more detailed information out of a model

DISTRIBUTIONS

	Random Variates
Normal	<code>rnorm</code>
Poisson	<code>rpois</code>
Binomial	<code>rbinom</code>
Uniform	<code>runif</code>

PLOTTING

`plot(x)` – plot values of x in order

GETTING HELP

<code>help(topic)</code>	Documentation on topic
<code>help.search("topic")</code>	Search the help system

VARIABLE ASSIGNMENT

`A <- 3`: creates a variable with value 3

`B <- "Hello"`: creates a variable with value "Hello"

NOTES

INDEXING IN R STARTS FROM 1

COMMENTS IN R START WITH #

NULL IS THE NON EXISTENT VALUE IN R WHILE NA IS THE MISSING PLACEHOLDER

RSTUDIO

OVERVIEW

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

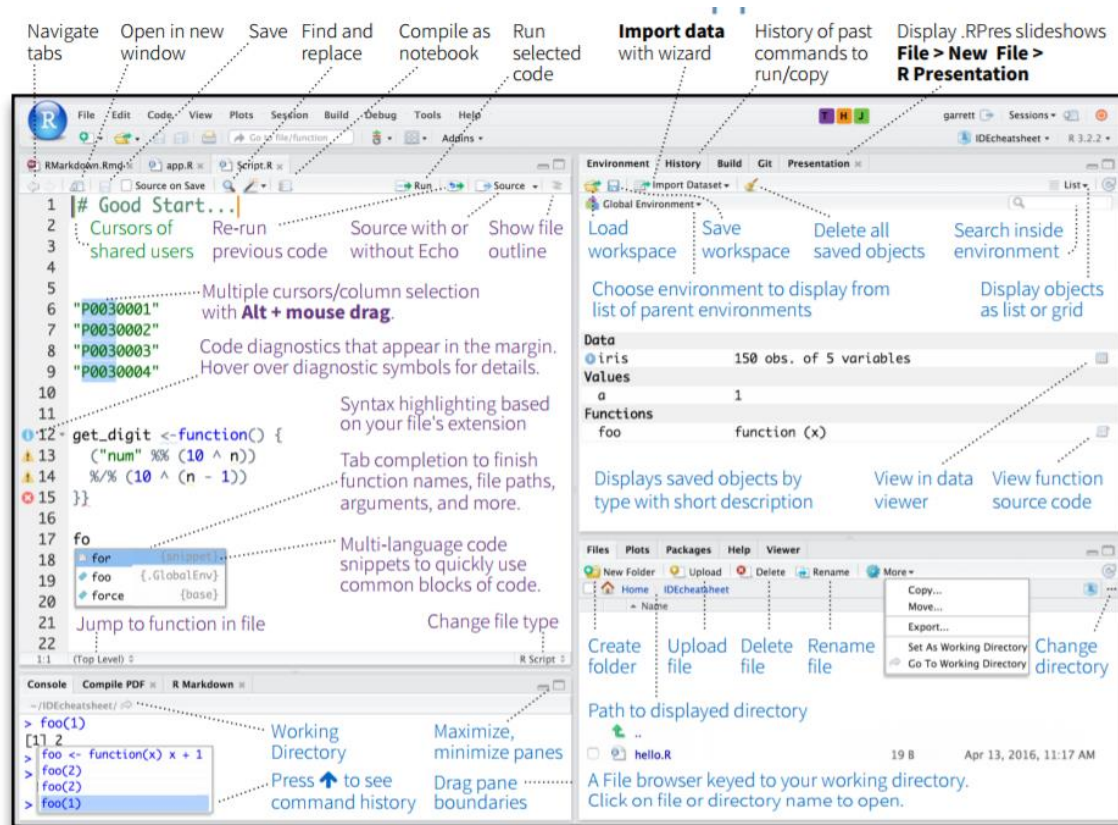
RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to Rstudio Server or Rstudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux).

FEATURES OF RSTUDIO

- Syntax highlighting, code completion, smart indentation
- Execute code directly from the source editor
- Quickly jump to function definitions
- Integrated R help and documentation

- Easily manage multiple working directories using projects
- Workspace browser and data viewer
- Interactive debugger to diagnose and fix errors quickly
- Extensive package development tools
- Authoring with Sweave and R Markdown
- Rstudio runs on most desktops or on a server and is accessed over the web
- Integrated support for Git and Subversion

COMPONENTS OF RSTUDIO



PACKAGE DEVELOPMENT USING RSTUDIO

PACKAGE STRUCTURE

The contents of a package can be stored on disk as a:

- Source – a directory with sub-directories
- Bundle – a single compressed file (.tar.gz)
- Binary – a single compressed file optimized for a specific OS

Or installed into an R library or archived online in a repository

SETUP

The DESCRIPTION file describes our work and sets how our package will work with other packages

```
Package: mypackage
Title: Title of Package
Version: 0.1.0
Authors@R: person("Hadley", "Wickham", email =
  "hadley@me.com", role = c("aut", "cre"))
Description: What the package does (one paragraph)
Depends: R (>= 3.1.0)
License: GPL-2
LazyData: true
Imports:
  dplyr (>= 0.4.0),
  ggvis (>= 0.2)
Suggests:
  knitr (>= 0.1.0)
```

Import packages that your package must have to work. R will install them when it installs your package.

Suggest packages that are not very essential to yours. Users can install them manually, or not, as they like.

WRITE CODE

All the R code in the package goes in R/. Save all code in R/ as scripts

Create a new package project with: `devtools::create("path/to/name")`

Install the created project with: `devtools::install("packagename")`

DOCUMENTATION USING ROXYGEN PACKAGE

roxygen lets us write documentation inline in our .R files with a shorthand syntax

Common roxygen tags

- @examples
- @export
- @param
- @return

```
#' Add together two numbers.
#'
#' @param x A number.
#' @param y A number.
#' @return The sum of \code{x} and \code{y}.
#' @examples
#' add(1, 1)
#' @export
add <- function(x, y) {
  x + y
}
```

Command for creating documentation: `document()`

COMMON KEYBOARD SHORTCUTS

Description	Windows/Linux	Mac
Search command history	Ctrl+↑	Cmd+↑
Navigate command history	↑/↓	↑/↓
Interrupt current command	Esc	Esc
Clear console	Ctrl+L	Ctrl+L
Restart R session	Ctrl+Shift+F10	Cmd+Shift+F10
Run current line/selection	Ctrl+Enter	Cmd+Enter
Source the current file	Ctrl+Shift+S	Cmd+Shift+S
Goto file/function	Ctrl+	Cmd+
Insert <-	Alt+-	Option+-
Insert %>%	Ctrl+Shift+M	Cmd+Shift+M
(Un)comment lines	Ctrl+Shift+C	Cmd+Shift+C
Run from start to current line	Ctrl+Alt+B	Cmd+Option+B
Run from current code section	Ctrl+Alt+T	Cmd+Option+T
Show keyboard shortcuts	Alt+Shift+K	Option+Shift+K

Rcpp

OVERVIEW

The 'Rcpp' package provides R functions as well as C++ classes which offer a seamless integration of R and C++. Many R data types and objects can be mapped back and forth to C++ equivalents which facilitates both writing of new code as well as easier integration of third party libraries. Rcpp sugar gives syntactic sugar such as vectrized C++ expression; Rcpp modules provide easy extensibility using declarations and Rcpp attributes greatly facilitates code integration.

RcppArmadillo connects R with the powerful Armadillo templated C++ library for linear algebra. Armadillo is a C++ linear algebra library aiming towards a good balance between speed and ease of use. Integer, floating point and complex numbers are supported, as well as a subset of trigonometric and statistics functions.

USAGE

EXPORTING C++ FUNCTIONS

```
1 #include <Rcpp.h>
2 using namespace Rcpp;
3
4 // [[Rcpp::export]]
5 void hello()
6 {
7   Rprintf("Hello, World!\n");
8 }
```

Line 1 includes Rcpp.h which contains the definitions used by the Rcpp package. The comment in line 4 is an Rcpp attribute. Attributes are annotations that are added to C++ source files to indicate that C++ functions should be made available as R functions.

Testing from R prompt

```
1 library(Rcpp)
2 sourceCpp("hello.cpp")
3 hello()
```

The sourceCpp function parses a C++ file and looks for functions marked with the Rcpp::export attribute. A shared library is then built and its exported functions are made available as R functions in the specified environment.

BASIC OPERATIONS

Manipulating Armadillo Objects

```
// [[Rcpp::export()]]
List a8 (int n, int r, double v) {
  arma::mat x1 ;
  x1.print() ;
  x1.reshape(n, r) ;
  x1.fill(v) ;
  arma::mat x2(n, r) ;
  x2.fill(v) ;
  arma::mat x3 = x2 ;
  x3.reshape(r, n) ;
  List ret ;
  ret["x1"] = x1 ;
  ret["x2"] = x2 ;
  ret["x3"] = x3 ;
  return(ret) ;
}
```

Indexing Armadillo Objects

```
// [[ Rcpp :: export ()]]
double a10 (arma::mat x, int i, int j) {
  return(x(i, j)) ;
}
```

Element wise addition

```
// [[ Rcpp :: export ()]]
arma::mat a14(arma::mat x) {
  return(x + x) ;
}
Element wise multiplication
// [[ Rcpp :: export ()]]
arma::mat a14(arma::mat x) {
  return(x % x) ;
}
Transpose
// [[ Rcpp :: export ()]]
arma::mat a14(arma::mat x) {
  return(x.t()) ;
}
Matrix inversion
// [[ Rcpp :: export ()]]
arma::mat a14(arma::mat x) {
  return(inv(x * x.t())) ;
}
```

Element wise subtraction

```
// [[ Rcpp :: export ()]]
arma::mat a14(arma::mat x) {
  return(x - x) ;
}
Element wise exponentiation
// [[ Rcpp :: export ()]]
arma::mat a14(arma::mat x) {
  return(exp(x)) ;
}
Matrix multiplication
// [[ Rcpp :: export ()]]
arma::mat a14(arma::mat x) {
  return(x.t() * x) ;
}
```

RPARALLEL

OVERVIEW

The common motivation behind parallel computing is that something is taking too long a time. Some common tasks which can be parallelized are bootstrapping, cross validation, multivariate imputation by chained equations, and fitting multiple regression models.

Packages which can be used for parallel programming in R

1. parallel – used for parallel programming
2. foreach – it creates a hybrid of the standard for loop and the lapply function
3. doParallel – provides a parallel backend for the %dpar% function using the parallel package

SAMPLE PROGRAM FOR UNDERSTANDING PARALLEL PROGRAMMING IN R

```
library(doParallel)
bootstrap_CI <- function(X, B = 10000){
  ## The function returns a 95% confidence interval for the
  ## median of X, computed using the bootstrap
  n <- length(X)
  ## Setup parallel backend to use multiple processors
  cl <- makeCluster(4)
  registerDoParallel(cl)

  ## Start program
  bootstrap_results <- foreach(i = 1:B, .combine = cbind) %dpar% {
    boot_indices <- sample(n, replace = T)
    boot_sample <- X[boot_indices]
    boot_stat <- median(boot_sample)

    ## Combine all of our output into a list
    result <- list(median = boot_stat)
    result
  }
  stopCluster(cl)
  my_results <- as.numeric(unlist(bootstrap_results))
}
```

```

CI_1 <- quantile(my_results, c(0.025, 0.975), names = F)
CI_Low    <- CI_1[1]
CI_high    <- CI_1[2]
CI        <- c(CI_Low, CI_high)
return(CI)
}

```

PYTHON

OVERVIEW

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

BASIC PYTHON

VARIABLES AND DATA TYPES

Variable assignment: `x = 5`

Type conversion: `str()`, `int()`, `float()`, `bool()`

DATA STRUCTURES

Tuple: One dimensional, fixed-length, immutable sequence of Python objects of ANY type.

Create Tuple: `tup1 = 4, 5, 6` or `tup1 = (6,7,8)`

List: One dimensional, variable length, mutable (i.e. contents can be modified) sequence of Python objects of ANY type.

Create List: `List1 = [1, 'a', 3]` or `List1 = list(tup1)`

Dict: Hash maps

Create Dict: `dict1 = {'key1' : 'value1', 2 : [3, 2]}`

Set: A set is an unordered collection of UNIQUE elements

Create Set: `set([3, 6, 3])` or `{3, 6, 3}`

OPERATIONS ON LISTS

Sub setting and slicing

<code>a[1]</code>	Select item at index 1
<code>a[-3]</code>	Select 3rd last item
<code>a[1:3]</code>	Select items at index 1 and 2
<code>a[1:]</code>	Select items after index 0
<code>a[:]</code>	Copy list
<code>a[:3]</code>	Select items before index 3

Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List methods

<code>a.index(b)</code>	Get the index of an item
<code>a.count(b)</code>	Count an item
<code>a.append('!')</code>	Append an item at a time
<code>a.remove('!')</code>	Remove an item
<code>del(a[0:1])</code>	Remove an item
<code>a.reverse()</code>	Reverse the list
<code>a.extend('!')</code>	Append an item
<code>a.pop(-1)</code>	Remove an item
<code>a.insert(0,'!')</code>	Insert an item
<code>a.sort()</code>	Sort the list

NUMPY

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention

```
>>> import numpy as np
```

CREATING NUMPY ARRAYS

```
>>> a = np.array([1,2,3])
```

```
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
```

```
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]], dtype = float)
```

<code>>>> np.zeros((3,4))</code>	Create an array of zeros
<code>>>> np.ones((2,3,4),dtype=np.int16)</code>	Create an array of ones
<code>>>> d = np.arange(10,25,5)</code>	Create an array of evenly spaced values (step value)
<code>>>> np.linspace(0,2,9)</code>	Create an array of evenly spaced values (number of samples)
<code>>>> e = np.full((2,2),7)</code>	Create a constant array

>>> f = np.eye(2)	Create a 2X2 identity matrix
>>> np.random.random((2,2))	Create an array with random values
>>> np.empty((3,2))	Create an empty array

OPERATIONS

Addition	np.add(a, b)
Subtraction	np.subtract(a, b)
Division	a / b
Multiplication	np.multiply(a,b)
Dot product	np.dot(a,b)
Transpose	np.transpose(a)
Changing array shape	a.reshape((size))
Vertical stacking of arrays	np.vstack((a,b))
Horizontal stacking	np.hstack((a, b))
Concatenate arrays	np.concatenate((a,d), axis=0)

MATPLOTLIB

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms

Example

```
plt.axis([0,1000,0,1.2])
plt.xlabel("Number of Iterations --->")
plt.ylabel("Accuracy")
plt.title("Comparison of Testing and Training Accuracy in a 2 Layer Neural Network")
plt.plot(acc_train, 'b--', label="Training Accuracy")
plt.plot(acc_test, label="Testing Accuracy", color='orange')
plt.legend()
plt.grid(True)
plt.show()
```

SCIKIT

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

Linear regression using scikit

```
n = 100
p = 3
X = np.random.rand(n, p)
beta = [1 for i in range(p)]
beta = np.array(beta)
beta = beta.reshape(p, 1)
Y = np.dot(X, beta)
Y = np.add(Y, np.random.rand(n, 1))
regr = Linear_model.LinearRegression()
regr.fit(X, Y)
Py_coeff = regr.coef_
Py_int = regr.intercept_
```

NOTES

PYTHON IS CASE SENSITIVE

INDEXING IN PYTHON STARTS FROM 0

INDENTATION IS USED TO STRUCTURE CODE INTO FUNCTIONS, LOOPS, CONDITIONALS, CLASSES, ETC

COMMENT STARTS WITH #

SQL

OVERVIEW

SQL or Structured Query Language is a domain specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system.

TYPES OF SQL STATEMENTS

DATA DEFINITION LANGUAGE (DDL)

These commands allow a database user to create and restructure database objects, such as creation or deletion of a table.

1. CREATE TABLE – creates a new table
2. ALTER TABLE – modifies a table
3. DROP TABLE – deletes a table
4. CREATE INDEX – creates an index (search key)
5. ALTER INDEX – alters an index
6. DROP INDEX – drops an index
7. CREATE VIEW – creates a view. A view is a virtual table which is a result of a query.
8. DROP VIEW – deletes a view

DATA CONTROL LANGUAGE (DCL)

These commands allow the user to control access to data within the database. These commands are normally used to create objects related to user access and also control the distribution of privileges among users.

1. ALTER PASSWORD
2. GRANT
3. REVOKE
4. CREATE SYNONYM

DATA MANIPULATION LANGUAGE (DML)

These commands are used to manipulate data within objects of a relational database.

SELECT – used to select data from a database	SELECT <i>column1, column2, ...</i> FROM <i>table_name</i> WHERE <i>condition</i> ;
INSERT – inserts new data into a database	INSERT INTO <i>table_name</i> (<i>column1, column2, column3, ...</i>) VALUES (<i>value1, value2, value3, ...</i>);
UPDATE – updates existing data in the database	UPDATE <i>table_name</i> SET <i>column1 = value1, column2 = value2, ...</i> WHERE <i>condition</i> ;
DELETE – deletes data from the database	DELETE FROM <i>table_name</i> WHERE <i>condition</i> ;

DATA ADMINISTRATION COMMANDS

These commands allow the user to perform audits and perform analyses on operations within the database. They can also be used to analyze system performance.

1. START AUDIT
2. STOP AUDIT

TRANSACTIONAL CONTROL COMMANDS

These commands allow the user to manage database transactions

1. COMMIT – saves database transactions
2. ROLLBACK – undoes database transactions
3. SAVEPOINT – creates points within groups of transactions in which to ROLLBACK
4. SET TRANSACTION – place a name on a transaction

COMMON SQL KEYWORDS, CLAUSES, OPERATORS, AND FUNCTIONS

1. DISTINCT

Syntax

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Example

```
SELECT DISTINCT Country FROM Customers;
```

2. ORDER BY – used to sort the result set in ascending or descending order.

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

3. SELECT TOP – used to specify the number of records to return

Version

Syntax

```
SQL      SELECT TOP number|percent column_name(s)
Server/  FROM table_name
MS       WHERE condition;
Access
MySQL    SELECT column_name(s)
         FROM table_name
         WHERE condition
         LIMIT number;
Oracle   SELECT column_name(s)
         FROM table_name
         WHERE ROWNUM <= number;
```

Example

```
SELECT TOP 3 * FROM Customers;
```

```
SELECT * FROM Customers
LIMIT 3;
```

```
SELECT * FROM Customers
WHERE ROWNUM <= 3;
```

4. MIN() – returns the smallest value of the selected column

Syntax

Example

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

5. MAX() – returns the largest value of the selected column

Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

6. COUNT() – returns the number of rows that matches a specified criteria

Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

Example

```
SELECT COUNT(ProductID)
FROM Products;
```

7. AVG() – returns the average value of a numeric column

Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

Example

```
SELECT AVG(Price)
FROM Products;
```

8. SUM() – returns the total sum of a numeric column

Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Example

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

9. UNION

Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Example

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

10. GROUP BY AND HAVING

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Example

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

JOINS

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Types of SQL JOINS

- (INNER) JOIN – returns all records that have matching values in both tables

Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

Example

```
SELECT Orders.OrderID,
Customers.CustomerName
FROM Orders
```

```
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- LEFT (OUTER) JOIN – return all records from the left table, and the matched records from the right table

Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

Example

```
SELECT Customers.CustomerName,
Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

- RIGHT (OUTER) JOIN – return all records from the right table, and the matched records from the left table

Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

Example

```
SELECT Orders.OrderID,
Employees.LastName,
Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

- FULL (OUTER) JOIN – return all records when there is a match in either left or right table

Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```

Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

- SELF JOIN – it is a regular join, but the table is joined with itself

Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Example

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2,A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

UNIX/LINUX

OVERVIEW

- UNIX is an operating system which was first developed in the 1960s, and has been in development ever since.
- The UNIX operating system is made up of three parts: the kernel, the shell, and the programs.
- Versions of UNIX: Sun Solaris, GNU/Linux, and MacOS
- Everything in UNIX is either a file or process.
- All files are grouped together in a grouped directory structure. The top of the hierarchy is traditionally called *root*.

UNIX COMMANDS WITH EXAMPLES

Command	Description	Example
<i>mkdir</i>	creation of a directory	<code>mkdir sampleDir</code>
<i>ls</i>	lists contents of the directory Options: -a: list all files and directories in the directory -l: long listing of files -r: sort in revers -t: sort by modification time	<code>ls</code> <code>ls -a</code> <code>ls -lrt</code> <code>ls sampleDir</code> <code>ls sampleDir -a</code>
<i>cd</i>	change the directory	<code>cd sampleDir</code> <code>cd ..</code> <code>cd .</code>
<i>pwd</i>	displays the path of the current directory	<code>pwd</code>
<i>cp</i>	copies contents of a file to another file	<code>cp ~/readme.txt readme.bak</code> <code>cp ~/readme.txt</code>
<i>mv</i>	move or rename file from file 1 to file 2	<code>mv readme.bak backups/.</code>
<i>rm</i>	delete the specified file	<code>rm readme.txt</code>
<i>rmdir</i>	remove the specified directory	<code>rmdir sampleDir</code>
<i>cat</i>	displays contents of file on the terminal screen	<code>cat readme.txt</code>
<i>less</i>	writes the contents of file onto a separate screen one page at a time	<code>less readme.txt</code>
<i>head</i>	writes the first 10 lines on the file to the terminal screen. The -n option can be used to specify the first n lines to be read	<code>head readme.txt</code> <code>head -2 readme.txt</code>
<i>tail</i>	writes the last 10 lines on the file to the terminal screen. The -n option can be used to specify the last n lines to be read	<code>tail readme.txt</code> <code>tail -6 readme.txt</code>
<i>grep</i>	searches file for specified words or patterns	<code>grep up readme.txt</code> <code>grep -i Up readme.txt</code>
<i>wc</i>	helps find number of words and lines in the file	<code>wc -w readme.txt</code> <code>wc -l readme.txt</code>
<i>man</i>	loads the manual page of the specified command	<code>man ls</code>
<i>whatis</i>	gives a one line description of the command	<code>whatis wc</code>
<i>apropos</i>	gives the commands with the keyword in their manual page header	<code>apropos copy</code>
<i>chmod</i>	used for changing access rights of a file	<code>chmod go-rwx final1</code>
<i>ps</i>	used to see information about processes	<code>ps</code>
<i>jobs</i>	lists all running, backgrounded, and suspended processes	<code>jobs</code>
<i>kill</i>	terminate a process	<code>kill %3</code> <code>kill 2212</code>
<i>df</i>	reports the space left on the system	<code>df</code>
<i>du</i>	outputs the number of kb used by each subdirectory	<code>du</code>
<i>gzip</i>	compress the file	<code>gzip readme.txt</code>
<i>zcat</i>	read the zipped file without uncompressing	<code>zcat readme.txt.gz</code>

GITHUB

OVERVIEW

GitHub is a Web-based Git version control repository hosting service. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

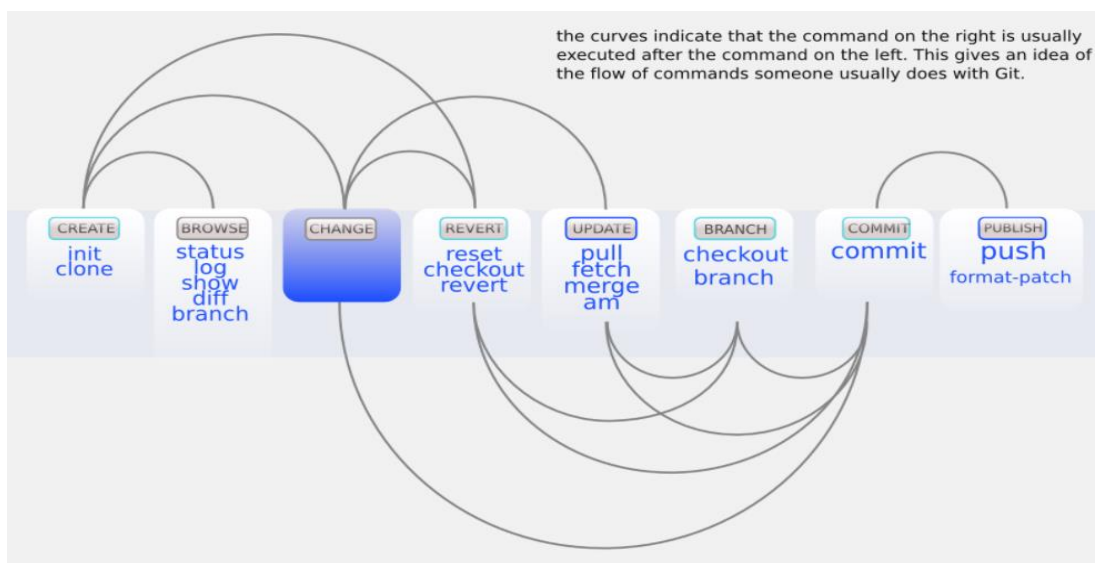
BASIC CONCEPTS

Master	Default development branch
Origin	Default upstream repository
HEAD	Current branch
HEAD^	Parent of HEAD
HEAD~4	The great-great grandparent of HEAD

COMMON GIT COMMANDS

<i>git add</i>	Indicates that a modified file is ready to be added to the repository	<i>git add hello.r</i>
<i>git commit</i>	Adds the modifications to the repository	<i>git commit -m "first commit"</i>
<i>git status</i>	Lists the files that have been changed and any other new additions	<i>git status</i>
<i>git diff</i>	Shows the exact changes made	<i>git diff hello.r</i>
<i>git push</i>	Pushes committed changes to github	<i>git push origin master</i>
<i>git pull</i>	Fetch from and integrate with another repository or a local branch	<i>git pull greek-geek master</i>
<i>git init</i>	Used for creating a new repository	<i>git init</i>
<i>git clone</i>	Downloads a project and its entire version history	<i>git clone <<url>>1</i>

GIT COMMAND SEQUENCE



HOFFMAN2 CLUSTER

OVERVIEW

The Hoffman2 Cluster is a cluster hosting program by IDRE. UCLA's shared Hoffman2 cluster currently consists of 1200+ 64-bit nodes and 13340 cores with an aggregate of over 50 TB of memory. It is the largest and most powerful cluster in the UC system.

If anyone has a large amount of data they want to work with or they must perform some heavy-duty computation which can't be supported by existing computing resources of the person, then it would be a good idea to make use of the Hoffman2 cluster.

GETTING A HOFFMAN2 ACCOUNT

A Hoffman2 account can be obtained on the approval of a faculty sponsor. To obtain the account, the user will have to register with UCLA IDRE.

CONNECTING TO HOFFMAN2

Unix, Linux, Mac Users

Login using the ssh command
`ssh login-id@hoffman2.idre.ucla.edu`

Windows Users

Can be done using SSH clients like putty, cygwin or MobaXterm
A NoMachine client can also be used to access Hoffman2 on Windows

TYPES OF NODES

Login Nodes

When we log onto Hoffman2, we are assigned a 'login node' for maneuvering around and file organization. To ensure load balancing, several login nodes are present which can be used interchangeably. On each login, a user is randomly assigned to one of the login nodes. The name of the login node appears on the shell prompt:

```
[login_id@login4 ~]$
```

A user's user id is limited to 16 sessions per login node.

Compute Nodes

To perform computations on the Hoffman2 cluster, we require a compute node. Computations are not supposed to be performed on the login node. Hoffman2 cluster's compute nodes have different memory sizes. On submitting a job to the scheduler, it is automatically executed on the compute nodes. Compute nodes are also made available for interactive use by using the "qrsh" command.

SAMPLE COMMANDS

SUBMITTING A JOB

`qsub -cwd -N gene_data_lasso -l h_data=64M, h_rt=8:00:00 -m bea myjob.sh`

-cwd: execute the job from the current working directory

-N name: specifies the name of the job

-l resource=value, ...: launch the job meeting the given resource request list.

-m b|e|a|s|n,...: defines or redefines under which circumstances mail is to be sent to the job owner. Options are as follows:

<code>`b'</code>	Mail is sent at the beginning of the job.
<code>`e'</code>	Mail is sent at the end of the job.
<code>`a'</code>	Mail is sent when the job is aborted or rescheduled.
<code>`s'</code>	Mail is sent when the job is suspended.
<code>`n'</code>	No mail is sent.

REQUESTING A COMPUTE NODE FOR INTERACTIVE USE

`qrsh -l h_rt=4:00:00, h_data=12G`

Explanation: h_rt denotes time, h_data denotes amount of memory needed

`qrsh -l h_rt=8:00:00, h_data=4G -pe shared 2`

Explanation: h_rt denotes time, h_data denotes amount of memory needed, -pe shared denotes how many CPU cores are needed

NOTES

EVERY USER OF THE HOFFMAN2 CLUSTER HAS A HOME DIRECTORY WITH 20GB STORAGE FOR GENERAL USE. ADDITIONAL STORAGE OF 2TB IS AVAILABLE IN '/U/SCRATCH' – THIS IS KEPT FOR 7 DAYS

REQUESTING MORE RESOURCES THAN NEEDED WILL DELAY ITS STARTING. IT WILL ALSO DEFEAT THE JOB SCHEDULER'S BACK-FILLING CAPABILITY AND WASTE CLUSTER RESOURCES.

JOBS OR INTERACTIVE SESSIONS USING MORE PROCESSORS OR SIGNIFICANTLY MORE MEMORY THAN WAS RESERVED WITH UGE MAY BE TERMINATED WITHOUT PRIOR NOTICE BY THE SYSTEM ADMINISTRATOR.

TENSORFLOW

OVERVIEW

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

GENERAL PRINCIPLE FOR BUILDING A NEURAL NETWORK

To build a neural network with Tensorflow, you can follow the steps below:

1. Import the modules you need.
2. Define an `add_layer` function to construct layers.
3. Define the variables and initialize them.
4. Create a session to perform the operation.
5. Build the NN, and send data with placeholders and `feed_dict`
6. Train and test the NN, and observe the results with Tensorboard.

COMMANDS AND CONCEPTS

TENSOR

A tensor is a generalization of vectors and matrices to potentially higher dimensions. Internally, tensorflow represents tensors as ndimensional arrays of base datatypes

A `tf.Tensor` has the following properties: - a data type (float32, int32, or string, for example) - a shape

TF.SESSION

It is a tensorflow session to run parts of the graph. Can be created as `sess=tf.Session()`

ADDING A LAYER IN NEURAL NETWORK

```
w1 = tf.get_variable('w1', [784, 500], initializer=tf.random_normal_initializer(stddev=0.3))
b1 = tf.get_variable('b1', [1,], initializer=tf.random_normal_initializer(stddev=0.3))
y1 = tf.nn.relu(tf.matmul(x, w1) + b1)
```

SOME OPTIMIZATION FUNCTIONS PRESENT IN TENSORFLOW

- Gradient descent optimizer
`train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)`
- Adam optimizer
`train_step = tf.train.AdamOptimizer().minimize(cross_entropy)`

TRAINING THE NEURAL NETWORK

```
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
for epoch in range(1000):
    for i in range(int(mnist.train.num_examples / 100)):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    correct = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

SAS

OVERVIEW

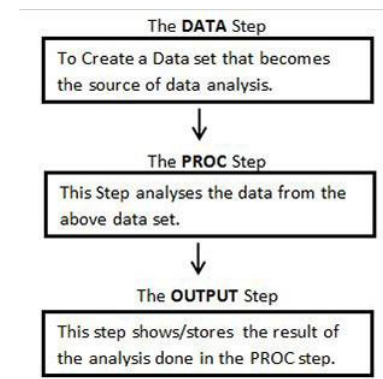
SAS is a software suite developed by SAS Institute for advanced analytics, multivariate analyses, business intelligence, data management, and predictive analytics. SAS provides a graphical point-and-click user interface for non-technical users and more advanced options through the SAS language.

TASKS PERFORMED USING SAS

- Data entry, retrieval, and management
- Report writing and graphics design
- Statistical and mathematical analysis
- Business forecasting and decision support
- Operations research and project management
- Applications development

SAS PROGRAM STRUCTURE

DATA Step	
DATA data_set_name;	#Name the data set.
INPUT var1,var2,var3;	#Define the variables in this data set.
NEW_VAR;	#Create new variables.
LABEL;	#Assign labels to variables.
DATALINES;	#Enter the data.
RUN;	
PROC Step	
PROC procedure_name options;	#The name of the proc.
RUN;	
OUTPUT Step	
PROC PRINT DATA = data_set;	
OPTIONS;	
RUN;	



STATISTICS PROGRAMMING USING SAS

ARITHMETIC MEAN	
PROC MEANS DATA = DATASET; CLASS Variables ; VAR Variables;	PROC MEANS DATA = sashelp.CARS Mean SUM MAXDEC=2; RUN;
STANDARD DEVIATION	
PROC means DATA = dataset STD;	proc means data=CARS1 STD; run;
CORRELATION ANALYSIS	
PROC CORR DATA = dataset options; VAR variable;	proc corr data=cars1 ; VAR horsepower weight ;

	BY make; run;
LINEAR REGRESSION	
PROC REG DATA = dataset; MODEL variable_1 = variable_2;	proc reg data=cars1; model horsepower= weight ; run;
MULTIPLE REGRESSION	
PROC GLM DATA=dataset; MODEL model; RUN;	proc glm data=data; model mpg = length length*length; run;

DATA REPRESENTATION USING SAS

HISTOGRAM	
PROC UNIVARIATE DATA = DATASET; HISTOGRAM variables; RUN;	proc univariate data=sashelp.cars; histogram horsepower / midpoints = 176 to 350 by 50; run;
BAR CHARTS	
PROC SGPLOT DATA = DATASET; VBAR variables; RUN;	proc sgplot data=work.cars1; vbar length ; title 'Lengths of cars'; run;
SCATTER PLOTS	
PROC sgscatter DATA=DATASET; PLOT VARIABLE_1 * VARIABLE_2 / datalabel = VARIABLE group = VARIABLE; RUN;	PROC sgscatter DATA=CARS1; PLOT horsepower*Invoice / datalabel = make group = type grid; title 'Horsepower vs. Invoice for car makers by types'; RUN;
BOXPLOTS	
PROC SGPLOT DATA=DATASET; VBOX VARIABLE / category = VARIABLE; RUN;	PROC SGPLOT DATA=CARS1; VBOX horsepower / category = type; title 'Horsepower of cars by types'; RUN;

NOTES

*COMMENTS START WITH * OR /*...*/*