NATIONAL ENGINEERING CENTER
University of the Philippines
Diliman, Quezon City

Center for Business Intelligence

3.0 Reading, Manipulating and Writing Data

Eugene Rex L. Jalao, Ph.D.

Associate Professor

Department Industrial Engineering and Operations Research

University of the Philippines Diliman

@thephdataminer

Module 6 of the Business Intelligence and Analytics Certification of UP NEC and the UP Center for Business Intelligence

Outline for this Training

- Introduction to R and R Studio
- Data Types and Operators
  - Case Study on R Scripting
- **Reading, Manipulating and Writing Data**
  - **Case Study on Dataset Analysis with ETL**
- Basic R Programming
  - Case Study: Writing Functions
- Graphics and Plotting
- Deploying R and Dashboard Generation
  - Case Study: Deploying a Simple Dashboard
- Deploying R with C#
  - Case Study: A Simple Standalone GUI For R Apps

Outline for this Session

- Data Frames
- Database Queries
- Reshape Package
- DPLYR Packager

Data Frames

Definition 3.1: Data Frames

- A data frame is a **two dimensional dataset**.
- Tightly coupled **collections of variables** which share many of the properties of matrices.
- Used as the **fundamental data structure** by most of **R**'s modeling scripts

Data Frames

Attributes/Columns/Variables/Features $(p+1)$

Rows/ Instances /Tuples /Objects (n)

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Data Frames

- Calculating Memory Requirements
  - Given a **dataset** with 1,500,000 rows and 120 columns, all of which are numeric data. Roughly, how much memory is required to store this data frame?
  - 1,500, 000 rows × 120 columns × 8 bytes/numeric entity = 1440000000 bytes
  - = 1440000000 bytes/220 bytes/MB
  - = 1,373.29 MB
  - = 1.34 GB

Data Frames

- Restrictions on the contents of a data frame
  - The components must be **vectors** (numeric, character, or logical), factors, numeric matrices, lists, or other data frames.
  - **Numeric vectors**, logicals and factors are included as is, and by default character vectors are coerced to be factors, whose levels are the unique values appearing in the vector.
  - **Vector structures** appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

Data Frames

Example 3.1: Data Frames

- #dataframes
- x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
- y <- c(11.4, 6.6, 4.1, 7.4, 22.7)
- z <- data.frame(x,y)
- z

```
> #dataframes
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> y <- c(11.4, 6.6, 4.1, 7.4, 22.7)
> z <- data.frame(x,y)
> z
     x    y
1 10.4 11.4
2  5.6  6.6
3  3.1  4.1
4  6.4  7.4
5 21.7 22.7
>
```

Data Frames

- To refer to a specific column use the $ notation.
- **Cumbersome**
- Use attach() and detach() to make the components of a data frame temporarily visible as variables under their component name

Data Frames

Example 3.2: Attach and Detach

- `#dataframes using $ symbol`
- `a=z$x+z$y`
- `a`
- `#dataframes using attach and detach`
- `attach(z)`
- `b=x+y`
- `b`
- `detach(z)`

Data Frames

```
> #dataframes using $ symbol
> a=z$x+z$y
> a
[1] 21.8 12.2  7.2 13.8 44.4
> #dataframes using attach and detach
> attach(z)
The following objects are masked _by_ .GlobalEnv:

    x, y

> b=x+y
> b
[1] 21.8 12.2  7.2 13.8 44.4
> detach(z)
> 
```

Data Frames

- Reading Data from Files
  - Large data objects will usually be read as values from **external files** rather than entered during an R session at the keyboard.
  - R input facilities are simple and their requirements are fairly strict and even rather inflexible.
  - Types
    - Read from CSV/Text Files
    - Read from Excel Files
    - Read from Database Files

Data Frames

- Read from CSV Files
  - The sample data can also be in **comma separated values** (CSV) format.
  - Each cell inside such data file is separated by a special character, which usually is a comma, although other characters can be used as well.
  - Reads files from the **current working directory**

➢
```
read.csv(file="filename.csv", head=TRUE,
sep=",")
```

Data Frames

- Copy Data from the Desktop
  - For the following exercises, please **copy the contents** from the Shared Folder given to you to the My Documents -> Work folder
  - Data files to be read need to be in the **current working director**y for ease of access
  - Otherwise, the directory needs to be **explicitly stated**.
  - ➢ `read.csv(file="C:/My Documents/filename.csv", head=TRUE, sep=",")`

Data Frames

Example 3.3: Reading a CSV File

- `#read a CSV file`
- `heisenberg <- read.csv(file="simple.csv",head=TRUE,sep=",")`
- `heisenberg`

```
> heisenberg <- read.csv(file="simple.csv",head=TRUE,sep=",")
> heisenberg
  trial mass velocity
1     A 10.0       12
2     A 11.0       14
3     B  5.0        8
4     B  6.0       10
5     A 10.5       13
6     B  7.0       11
> 
```

Data Frames

- Read from Excel Files
  - We can use the function loadWorkbook from the XLConnect package to read the entire workbook
  - Then load the worksheets with readWorksheet.
  - However, the XLConnect package requires Java to be pre-installed.

Data Frames

Example 3.4: Reading an Excel File

- #Read from Excel
- library(xlsx)
- heisenbergxls = read.xlsx("simple.xlsx", sheetName ="simple")

```
> #Read from Excel
> library(xlsx)
> heisenbergxls = read.xlsx("simple.xlsx", sheetName ="simple")
> heisenbergxls
  trial mass velocity
1     A 10.0       12
2     A 11.0       14
3     B  5.0        8
4     B  6.0       10
5     A 10.5       13
6     B  7.0       11
>
```

Data Frames

- Reading Larger Files
  - An alternative to reading large CSV files
  - Similar to read.csv but **faster**
  - Use the data.table package

Data Frames

Example 3.5: Read from a large file

- #Read Large CSV File
- library("data.table")
- heisenberglargecsv = as.data.frame(fread("largecsv.csv"))

```
> #Read Large CSV File
> library("data.table")
> heisenberglargecsv = as.data.frame(fread("largecsv.csv"))
```

Data Frames

Definition 3.2: Subsetting

- R has powerful **indexing features** for accessing object elements.
- These features can be used to **select and exclude** variables and observations.
- We define here ways to keep or delete variables and observations and to take random samples from a dataset.
- Syntax

```
Dataset[select rows , select columns]
```

Data Frames

Example 3.6: Subsetting

- #Subsetting
- heisenbergaonly = heisenberg[heisenberg$trial=="A",c("trial","mass")]
- heisenbergaonly

```
> #Subsetting
> heisenbergaonly = heisenberg[heisenberg$trial=="A",c("trial","mass")]
> heisenbergaonly
  trial mass
1     A 10.0
2     A 11.0
5     A 10.5
>
```

Data Frames

Definition 3.3: Merging

- To merge two data frames (datasets) horizontally, use the **merge** function.
- In most cases, you join **two data frames** by one or more common key variables (i.e., an inner join).
  - Inner join: merge(x = df1, y = df2, by = “colname")
  - Outer join: merge(x = df1, y = df2, by = “colname", all = TRUE)
  - Left outer: merge(x = df1, y = df2, by = " colname ", all.x = TRUE)
  - Right outer: merge(x = df1, y = df2, by = " colname", all.y = TRUE)

Data Frames

Example 3.7: Merging

```
➢ trial <- c("A","C","D")
➢ cost <- c(11.4, 3.3, 1.1)
➢ trialcost <- data.frame(trial,cost)
➢ trialcost
```

Data Frames

Example 3.7 (Cont.): Merging

- `#merge`
- `innerjoin = merge(x=heisenberg,y=trialcost,by=c("trial"))`
- `outerjoin = merge(x=heisenberg,y=trialcost,by=c("trial"), all= TRUE)`
- `leftjoin = merge(x=heisenberg,y=trialcost,by=c("trial"), all.x=TRUE)`
- `rightjoin = merge(x=heisenberg,y=trialcost,by=c("trial"), all.y=TRUE)`

Data Frames

heisenberg

trial	mass	velocity
A	10.0	12
A	11.0	14
B	5.0	8
B	6.0	10
A	10.5	13
B	7.0	11

trialcost

trial	cost
A	11.4
C	3.3
D	1.1

merge

Inner Join

trial	mass	velocity	cost
A	10.0	12	11.4
A	11.0	14	11.4
A	10.5	13	11.4

Outer Join

trial	mass	velocity	cost
A	10.0	12	11.4
A	11.0	14	11.4
A	10.5	13	11.4
B	5.0	8	NA
B	6.0	10	NA
B	7.0	11	NA
C	NA	NA	3.3
D	NA	NA	1.1

Left Join

trial	mass	velocity	cost
A	10.0	12	11.4
A	11.0	14	11.4
A	10.5	13	11.4
B	5.0	8	NA
B	6.0	10	NA
B	7.0	11	NA

Right Join

trial	mass	velocity	cost
A	10.0	12	11.4
A	11.0	14	11.4
A	10.5	13	11.4
C	NA	NA	3.3
D	NA	NA	1.1

Data Frames

- To sort a data frame in R, use the **order( )** function.
- By default, sorting is ASCENDING. Prepend the sorting variable by a minus sign to indicate DESCENDING order.

Data Frames

Example 3.8: Sorting

```
#Sort by Mass
heisenbergmass = heisenberg[order(heisenberg$mass),]
heisenbergmass
#Sort by Trial then By Mass
heisenbergtrialmass=
heisenberg[order(heisenberg$trial,
heisenberg$mass),]
heisenbergtrialmass
#Sort by Trial then by Mass Descending
heisenbergtrialmassdesc= heisenberg[order(
heisenberg$trial, -heisenberg$mass),]
heisenbergtrialmassdesc
```

Data Frames

```
> #Sort by Mass
> heisenbergmass = heisenberg[order(heisenberg$mass),]
> heisenbergmass
  trial mass velocity
3     B  5.0        8
4     B  6.0       10
6     B  7.0       11
1     A 10.0       12
5     A 10.5       13
2     A 11.0       14
> #Sort by Trial then By Mass
> heisenbergtrialmass= heisenberg[order(heisenberg$trial, heisenberg$mass),]
> heisenbergtrialmass
  trial mass velocity
1     A 10.0       12
5     A 10.5       13
2     A 11.0       14
3     B  5.0        8
4     B  6.0       10
6     B  7.0       11
```

Outline for this Session

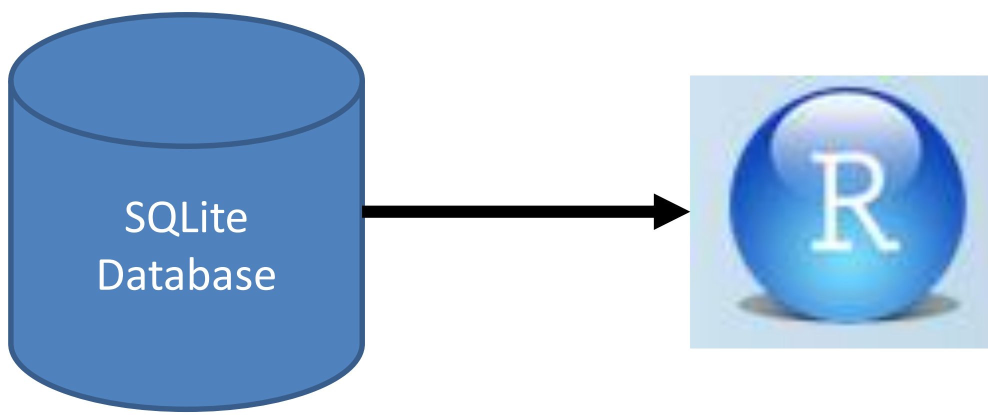
- Data Frames
- **Database Queries**
- Reshape Package
- DPLYR Packager

Database Queries

- Reading from an SQLite Database
- Reading from an MySQL Database

Database Queries

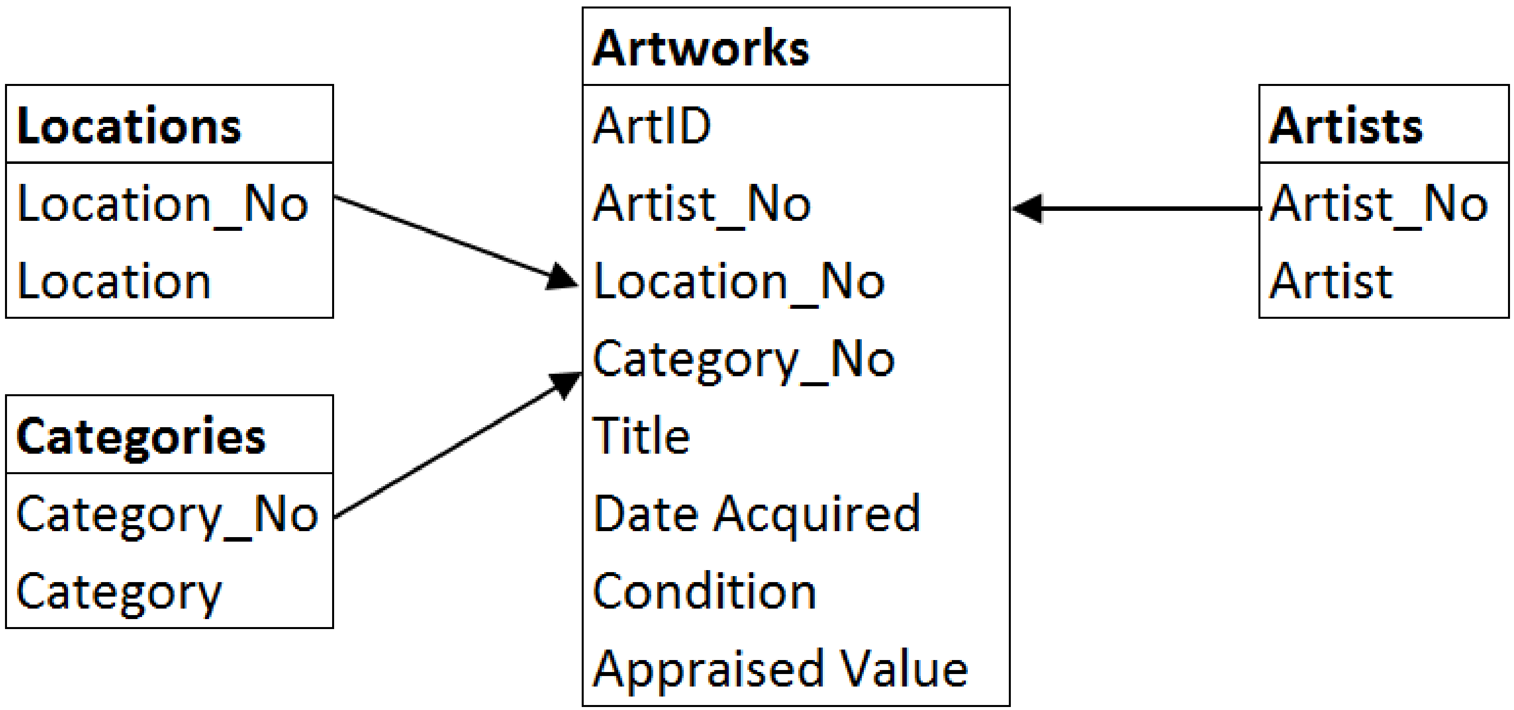
- Reading from an SQLite Database
  - SQLite is an open source database file
  - Connecting to an SQLite Database file is made very easy with the sqlLdf package



Database Queries

Example 3.9: Art Database

- SQLite Database Schema: Art Database



Database Queries

- `library(sqldf)`
- `library(XLConnect)`
- `db <- dbConnect(SQLite(), dbname="Art.sqlite")`
- `rs = dbSendQuery(db, "SELECT * FROM artworks")`
- `data = fetch(rs, n=-1)`
- `dbDisconnect(db)`
- `data`

Database Queries

```
> library(sqldf)
> library(XLConnect)
> db <- dbConnect(SQLite(), dbname="Art.sqlite")
> rs = dbSendQuery(db, "SELECT * FROM artworks")
> data = fetch(rs, n=-1)
> dbDisconnect(db)
[1] TRUE
Warning message:
In .local(conn, ...) : Closing open result set
> data
  ArtID Artist_No Category_No Location_No                Title Date.Acquired
1     1         1           2           1    Red Rock Mountain       3/19/05
2     2         2           2           1            Offerings       5/16/05
3     3         3           3           1       Spring Flowers       3/20/04
4     4         4           3           2       Seeking Shelter       10/8/05
5     5         5           2           1             The Hang       7/16/04
6     6         6           3           1     House Remembered       8/16/04
```

Database Queries

- Reading from an MySQL Database
  - Connecting to MySQL is made very easy with the RMySQL package
  - Once the RMySQL library is installed create a database connection object.

➢ `mydb = dbConnect(MySQL(), user='user', password='password', dbname='database_name', host='host')`

  - To retrieve data from the database we need to save a results set object.

➢ `rs = dbSendQuery(mydb, "select * from some_table")`

Database Queries

- Exporting Data
  - There are numerous methods for exporting **R** objects into other formats.

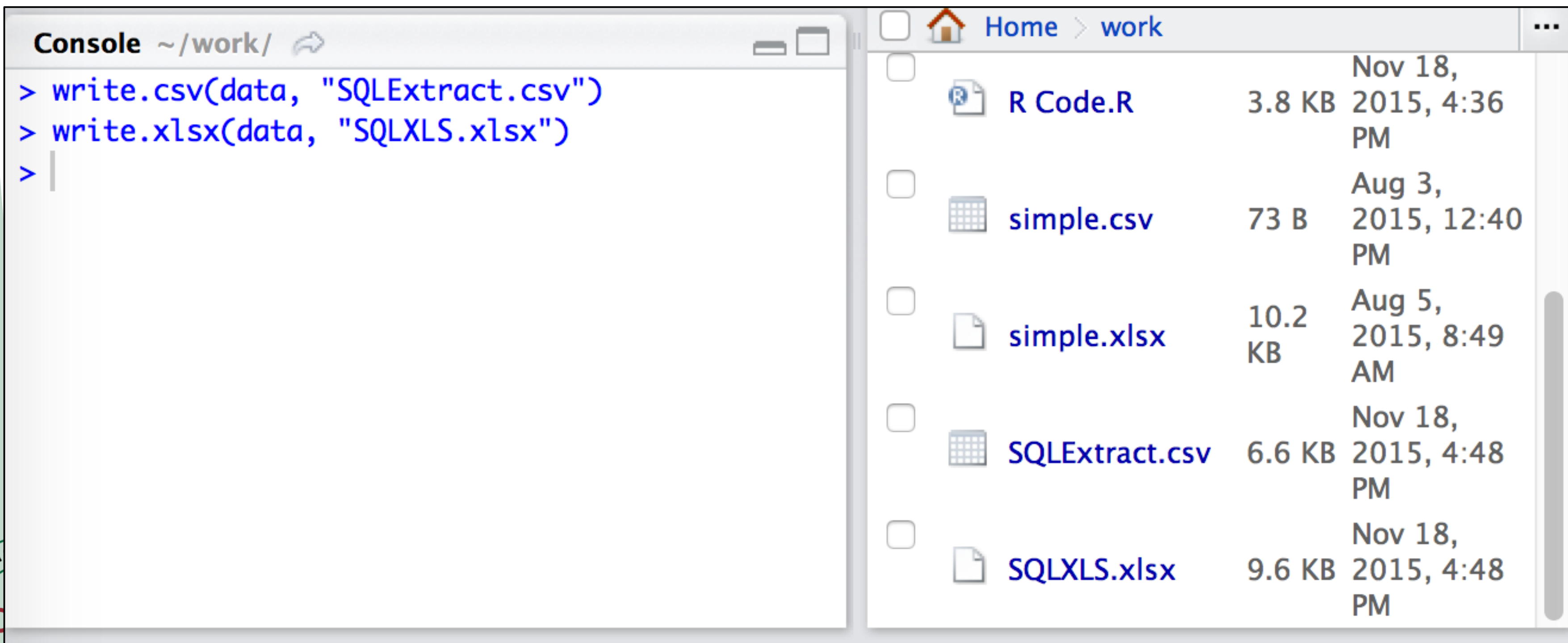
➢ `write.csv(datavariable, “filename.csv")`

➢ `write.xlsx(datavariable, "filename.xlsx")`

Database Queries

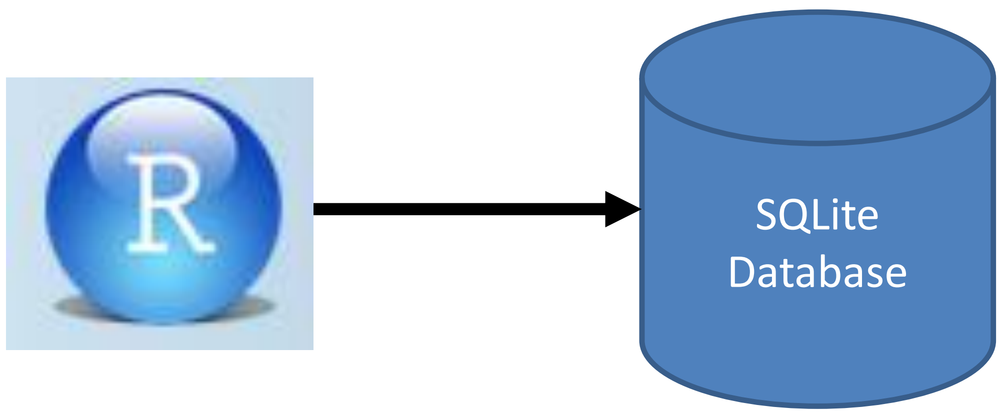
Example 3.10: Exporting Data

- `write.csv(data, “SQLExtract.csv")`
- `write.xlsx(data, “SQLXLS.xlsx")`



Database Queries

- Exporting Data to MySQL
  - Write a local data frame or file to the database.
  - ➢ `dbWriteTable(conn, tablename, value, row.names = NA, overwrite = FALSE, append = FALSE, field.types = NULL)`



Database Queries

Example 3.11: Exporting to an SQLite Database

- `db <- dbConnect(SQLite(), dbname="Art.sqlite")`
- `dbWriteTable(conn = db, name = "artworks2", value =data, row.names = FALSE, append = TRUE)`
- `rs = dbSendQuery(db, "SELECT * FROM artworks2")`
- `datatest = fetch(rs, n=-1)`
- `dbDisconnect(db)`

Outline for this Session

- Data Frames
- Database Queries
- **Reshape Package**
- DPLYR Packager

eljalao@up.edu.ph

The Reshape Package

Definition 3.2: Reshape

- Reshaping data is a **common task in practical data analysis**, and it is usually tedious and unintuitive.
- Data often has **multiple levels of grouping** (nested treatments, split plot designs, or repeated measurements) and typically requires investigation at multiple levels.
- Data reshaping is easiest to define with respect to aggregation.
- Aggregation is a common and familiar task where data is reduced and rearranged into a **smaller, more convenient form**, with a concomitant reduction in the amount of information

The Reshape Package

Definition 3.3: Melt

- Melting is the process of transforming measures/facts into a single column/variable
- Melt by Mass and Velocity Example

dimensions

trial	mass	velocity
A	10.0	12
A	11.0	14
B	5.0	8
B	6.0	10
A	10.5	13
B	7.0	11

facts

Melt

trial	variable	value
A	mass	10.0
A	mass	11.0
B	mass	5.0
B	mass	6.0
A	mass	10.5
B	mass	7.0
A	velocity	12.0
A	velocity	14.0
B	velocity	8.0
B	velocity	10.0
A	velocity	13.0
B	velocity	11.0

The Reshape Package

Definition 3.4: Cast

- Casting is the summarization of a molten dataset into aggregated data
- Get mean of mass and velocity by trial

trial	variable	value
A	mass	10.0
A	mass	11.0
B	mass	5.0
B	mass	6.0
A	mass	10.5
B	mass	7.0
A	velocity	12.0
A	velocity	14.0
B	velocity	8.0
B	velocity	10.0
A	velocity	13.0
B	velocity	11.0

Cast →

trial	mass	velocity
A	10.5	13.000000
B	6.0	9.666667

The Reshape Package

Example 3.12: Reshaping

```
➢ library("reshape2")
➢ heisenberg.m = melt(heisenberg,
  id=c('trial'),
  measure=c('mass','velocity'))
➢ heisenberg.m
➢ heisenberg.c = dcast(heisenberg.m,
  trial ~ variable, mean)
➢ heisenberg.c
```

The Reshape Package

```
> library("reshape2")
> heisenberg.m = melt(heisenberg, id=c('trial'), measure=c('mass','velocity'))
> heisenberg.m
   trial variable value
1      A     mass  10.0
2      A     mass  11.0
3      B     mass   5.0
4      B     mass   6.0
5      A     mass  10.5
6      B     mass   7.0
7      A velocity  12.0
8      A velocity  14.0
9      B velocity   8.0
10     B velocity  10.0
11     A velocity  13.0
12     B velocity  11.0
> heisenberg.c = dcast(heisenberg.m, trial ~ variable, mean)
> heisenberg.c
  trial mass  velocity
1     A 10.5 13.000000
2     B  6.0  9.666667
> 
```

Outline for this Session

- Data Frames
- Database Queries
- Reshape Package
- **DPLYR Packager**

dplyr Package

Definition 3.5: dplyr Package

- dplyr is a new package which provides a set of tools for efficiently manipulating datasets in R.
- dplyr is the next iteration of plyr, focussing on only data frames.
- dplyr is faster, has a more consistent API and should be easier to use.
- Succeeding examples taken from: https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html

dplyr Package

- The **nycflights13** data frame.
  – This dataset contains all 336776 flights that departed from and arrived at New York City in 2013. The data comes from the US Bureau of Transportation Statistics, and is documented in ?nycflights13

dplyr Package

- The `filter()` function
  - Allows you to select a subset of rows in a data frame. The first argument is the name of the data frame.
  - The second and subsequent arguments are the expressions that filter the data frame

dplyr Package

Example 3.13a: Filtering

- Select all flights last January 1, 2013

```
➢ library("dplyr")
➢ flights = read.csv("flights.csv")
➢ filtered = filter(flights, month == 1, day == 1)
➢ View(filtered)
```

dplyr Package

Example 3.13 (Cont.): Filtering

	X	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hou
1	1	2013	1	1	517	515	2	830	819	11	UA	1545	N14228	EWR	IAH	227	1400	5
2	2	2013	1	1	533	529	4	850	830	20	UA	1714	N24211	LGA	IAH	227	1416	5
3	3	2013	1	1	542	540	2	923	850	33	AA	1141	N619AA	JFK	MIA	160	1089	5
4	4	2013	1	1	544	545	-1	1004	1022	-18	B6	725	N804JB	JFK	BQN	183	1576	5
5	5	2013	1	1	554	600	-6	812	837	-25	DL	461	N668DN	LGA	ATL	116	762	6
6	6	2013	1	1	554	558	-4	740	728	12	UA	1696	N39463	EWR	ORD	150	719	5
7	7	2013	1	1	555	600	-5	913	854	19	B6	507	N516JB	EWR	FLL	158	1065	6
8	8	2013	1	1	557	600	-3	709	723	-14	EV	5708	N829AS	LGA	IAD	53	229	6
9	9	2013	1	1	557	600	-3	838	846	-8	B6	79	N593JB	JFK	MCO	140	944	6
10	10	2013	1	1	558	600	-2	753	745	8	AA	301	N3ALAA	LGA	ORD	138	733	6
11	11	2013	1	1	558	600	-2	849	851	-2	B6	49	N793JB	JFK	PBI	149	1028	6
12	12	2013	1	1	558	600	-2	853	856	-3	B6	71	N657JB	JFK	TPA	158	1005	6
13	13	2013	1	1	558	600	-2	924	917	7	UA	194	N29129	JFK	LAX	345	2475	6
14	14	2013	1	1	558	600	-2	923	937	-14	UA	1124	N53441	EWR	SFO	361	2565	6
15	15	2013	1	1	559	600	-1	941	910	31	AA	707	N3DUAA	LGA	DFW	257	1389	6
16	16	2013	1	1	559	559	0	702	706	-4	B6	1806	N708JB	JFK	BOS	44	187	5

dplyr Package

Example 3.13b: Filtering

- Select all flights from January or July 2013.

```
➢ filtered2 = filter(flights, month == 1| month == 7)
➢ View(filtered2)
```

dplyr Package

Example 3.13 (Cont.): Filtering

	X	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
56420	279866	2013	7	31	2243	2245	-2	2348	1	-13	B6
56421	279867	2013	7	31	2245	2245	0	2400	4	-4	B6
56422	279868	2013	7	31	2248	2250	-2	5	14	-9	B6
56423	279869	2013	7	31	2325	2051	154	221	2358	143	B6
56424	279870	2013	7	31	2346	2305	41	38	13	25	B6
56425	279871	2013	7	31	2352	2245	67	49	2359	50	B6
56426	279872	2013	7	31	NA	655	NA	NA	930	NA	AA
56427	279873	2013	7	31	NA	1400	NA	NA	1508	NA	US
56428	279874	2013	7	31	NA	959	NA	NA	1125	NA	UA
56429	279875	2013	7	31	NA	1025	NA	NA	1225	NA	MQ

dplyr Package

- The `slice()` function
  - To select rows by position, use slice():
  - `slice(df, rowindeces)`

dplyr Package

Example 3.14: Slice

- Select first 10 rows of the flights dataset.

```
slice(flights, 1:10)
```

	X	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
1	1	2013	1	1	517	515	2	830	819
2	2	2013	1	1	533	529	4	850	830
3	3	2013	1	1	542	540	2	923	850
4	4	2013	1	1	544	545	-1	1004	1022
5	5	2013	1	1	554	600	-6	812	837
6	6	2013	1	1	554	558	-4	740	728
7	7	2013	1	1	555	600	-5	913	854
8	8	2013	1	1	557	600	-3	709	723
9	9	2013	1	1	557	600	-3	838	846
10	10	2013	1	1	558	600	-2	753	745

dplyr Package

- Use the `arrange()` function to **sort** rows
  - `arrange()` works similarly to `filter()` except that instead of filtering or selecting rows, it **reorders them**.
  - It takes a data frame, and a set of column names (or more complicated expressions) to order by. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

dplyr Package

Example 3.15: Arrange

- Sort by year, then by month and then by day.

```
sorted = arrange(flights, year, month, day)
View(sorted)
```

	X	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	h
1	1	2013	1	1	517	515	2	830	819	11	UA	1545	N14228	EWR	IAH	227	1400	5
2	2	2013	1	1	533	529	4	850	830	20	UA	1714	N24211	LGA	IAH	227	1416	5
3	3	2013	1	1	542	540	2	923	850	33	AA	1141	N619AA	JFK	MIA	160	1089	5
4	4	2013	1	1	544	545	-1	1004	1022	-18	B6	725	N804JB	JFK	BQN	183	1576	5
5	5	2013	1	1	554	600	-6	812	837	-25	DL	461	N668DN	LGA	ATL	116	762	6
6	6	2013	1	1	554	558	-4	740	728	12	UA	1696	N39463	EWR	ORD	150	719	5
7	7	2013	1	1	555	600	-5	913	854	19	B6	507	N516JB	EWR	FLL	158	1065	6
8	8	2013	1	1	557	600	-3	709	723	-14	EV	5708	N829AS	LGA	IAD	53	229	6
9	9	2013	1	1	557	600	-3	838	846	-8	B6	79	N593JB	JFK	MCO	140	944	6
10	10	2013	1	1	558	600	-2	753	745	8	AA	301	N3ALAA	LGA	ORD	138	733	6

dplyr Package

Example 3.15 (Cont.): Arrange

- Sort by descending arrival delay time

```
descsorted = arrange(flights, desc(arr_delay))
View(descsorted)
```

336,776 observations of 20 variables

	X	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance
1	7073	2013	1	9	641	900	1301	1242	1530	1272	HA	51	N384HA	JFK	HNL	640	4983
2	235779	2013	6	15	1432	1935	1137	1607	2120	1127	MQ	3535	N504MQ	JFK	CMH	74	483
3	8240	2013	1	10	1121	1635	1126	1239	1810	1109	MQ	3695	N517MQ	EWR	ORD	111	719
4	327044	2013	9	20	1139	1845	1014	1457	2210	1007	AA	177	N338AA	JFK	SFO	354	2586
5	270377	2013	7	22	845	1600	1005	1044	1815	989	MQ	3075	N665MQ	JFK	CVG	96	589
6	173993	2013	4	10	1100	1900	960	1342	2211	931	DL	2391	N959DL	JFK	TPA	139	1005

dplyr Package

- The `select()` Function to select **certain columns**
  - Selecting Columns from Datasets by Listing Column Names
    - `select(df, col1, col2, col3,…)`
  - Selecting all columns between two columns (inclusive)
    - `select(df, col2:col5)`
  - Excluding all columns between two columns (inclusive) and selecting all others
    - `select(df, -col2:col5)`

dplyr Package

Example 3.16: Select

- Select only Year, Month and Day columns

➢ `selectedcol = select(flights, year, month, day)`

➢ `View(selectedcol)`

	year	month	day
1	2013	1	1
2	2013	1	1
3	2013	1	1
4	2013	1	1
5	2013	1	1
6	2013	1	1

dplyr Package

Example 3.16 (Cont.): Select

- Select all rows except columns on Year, Month and Day

```
➢ selectedcol = select(flights, -(year:day))
➢ View(selectedol)
```

	X	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum
1	1	517	515	2	830	819	11	UA	1545	N14228
2	2	533	529	4	850	830	20	UA	1714	N24211
3	3	542	540	2	923	850	33	AA	1141	N619AA

dplyr Package

- Extract Distinct (unique) Rows
  - Use `distinct()` to find unique values in a table

dplyr Package

Example 3.17: Distinct

- Select unique Origins and Destinations

```
distinct = distinct(select(flights, origin, dest))
View(distinct)
```

	origin	dest
1	EWR	IAH
2	LGA	IAH
3	JFK	MIA
4	JFK	BQN
5	LGA	ATL
6	EWR	ORD
7	EWR	FLL
8	LGA	IAD
9	JFK	MCO
10	LGA	ORD

dplyr Package

- Adding **New Columns** With `mutate()`
  - Besides selecting sets of existing columns, it's often useful to add new columns that are functions of existing columns. This is the job of mutate():
  - Mutate allows you to refer to columns that was just created
  - `mutate(df, newcol1 = f(oldcols), newcol2 = f(newcol1))`

dplyr Package

Example 3.18: Mutate

- Calculate new columns on gain, speed and gain per hour.

```
➢ delayed = mutate(flights,
➢         gain = arr_delay - dep_delay,
➢         speed = distance / air_time * 60,
➢         gain_per_hour = gain / (air_time / 60))
➢ View(delayed)
```

	gain	speed	gain_per_hour
00	9	370.0441	2.3788546
00	16	374.2731	4.2290749
00	31	408.3750	11.6250000
00	-17	516.7213	-5.5737705
00	-19	394.1379	-9.8275862
00	16	287.6000	6.4000000
00	24	404.4304	9.1139241

dplyr Package

- **Summarize** values with summarise()
  - It collapses a data frame to a single row
  - `summarise(df, var = mean(cols))`

dplyr Package

Example 3.19: Summarize

- Calculate mean delay

```
➢ meandelay = summarise(flights,
➢               delay = mean(dep_delay, na.rm = TRUE))
➢ View(meandelay)
```

	delay
1	12.63907

dplyr Package

- dplyr verbs are useful on their own, but they become really powerful when you apply them to **groups of observations** within a dataset.
- In dplyr, this is done by with the `group_by()` function.
- It breaks down a dataset into specified **groups of rows**.
- When the verbs are applied on the resulting object they'll be automatically applied "by group". Most importantly, all this is achieved by using the **same exact syntax** you'd use with an ungrouped object.

dplyr Package

- grouped `select()` is the same as ungrouped select(), except that **grouping variables** are always retained.
- grouped `arrange()` orders first by the grouping variables
- `mutate()` and `filter()` are most useful in conjunction with **window functions** (like `rank()`, or `min(x) == x`).
- `slice()` extracts rows **within each group**.
- `summarise()` can mimic the reshape function as shows in the succeeding examples

dplyr Package

Example 3.20: Grouping

- For each unique plane tail number, count the number of flights it made, average distance travelled and average delay time.

```
➢ by_tailnum <- group_by(flights, tailnum)
➢ delay <- summarise(by_tailnum,
➢   count = n(),
➢   dist = mean(distance, na.rm = TRUE),
➢   delay = mean(arr_delay, na.rm = TRUE))
➢ delay <- filter(delay, count > 20, dist < 2000)
➢ View(delay)
```

dplyr Package

Example 3.20 (Cont.): Grouping

	tailnum	count	dist	delay
1	N0EGMQ	371	676.1887	9.9829545
2	N10156	153	757.9477	12.7172414
3	N102UW	48	535.8750	2.9375000
4	N103US	46	535.1957	-6.9347826
5	N104UW	47	535.2553	1.8043478
6	N10575	289	519.7024	20.6914498
7	N105UW	45	524.8444	-0.2666667

dplyr Package

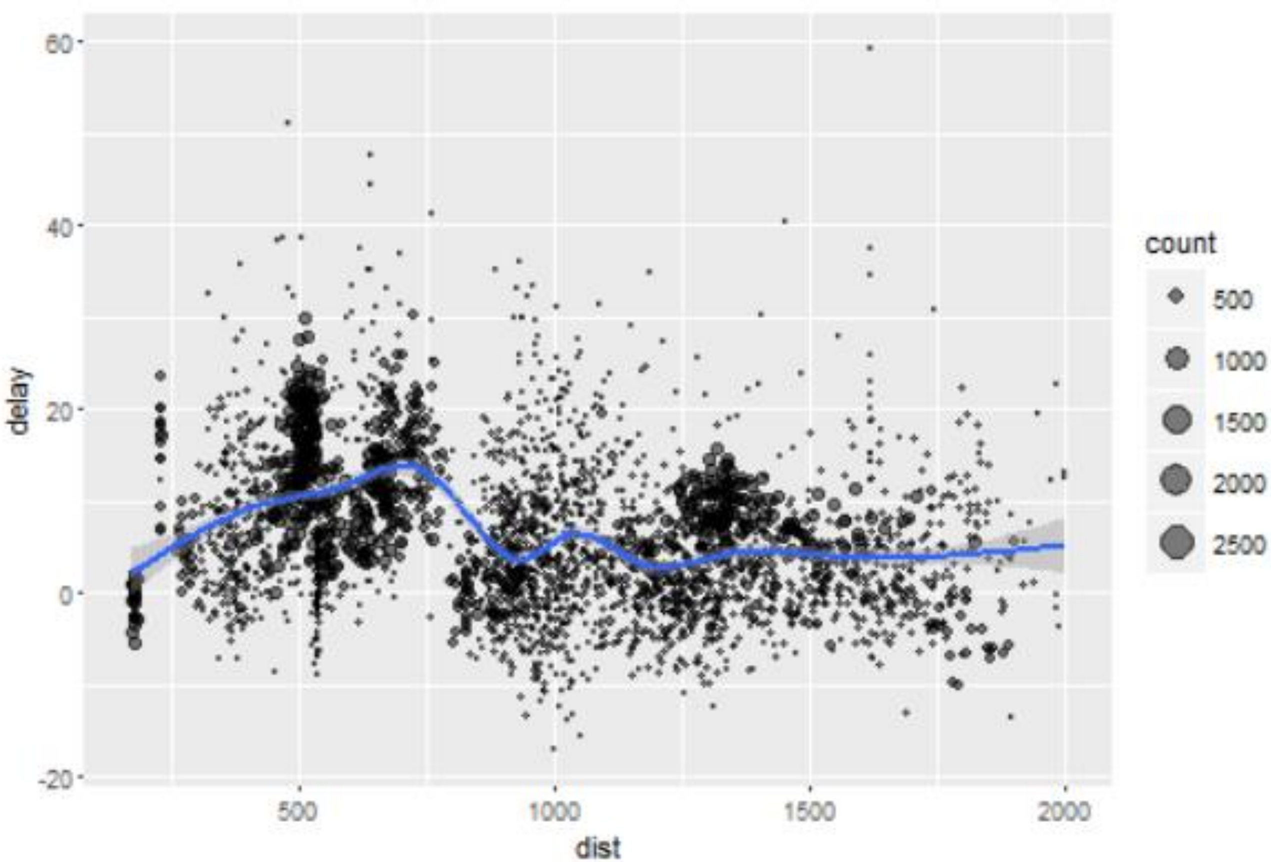
Example 3.21: Grouping with Visualization

- Plot Distance Versus Delay

```
library("ggplot2")
ggplot(delay, aes(dist, delay)) +
  geom_point(aes(size = count), alpha = 1/2) +
  geom_smooth() +
  scale_size_area()
```

dplyr Package

Example 3.21 (Cont.) : Grouping with Visualization



dplyr Package

- Summarize() and aggregate functions **can be used together**, which take a vector of values and return a single number.
- There are many useful examples of such functions in base R like `min()`, `max()`, `mean()`, `sum()`, `sd()`, `median()`, and `IQR()`.
- dplyr provides a handful of others:
  - `n()`: the number of observations in the current group
  - `n_distinct(x)`: the number of unique values in x.
  - `first(x)`, `last(x)` and `nth(x, n)`

dplyr Package

Example 3.22: Summarize and Aggregate

- For each destination, calculate number of planes, flights and average delay

```
destinations <- group_by(flights, dest)
destsummary = summarise(destinations,
          planes = n_distinct(tailnum),
          flights = n(),
          delay = mean(dep_delay, na.rm = TRUE))
View(destsummary)
```

	dest	planes	flights	delay
1	ABQ	108	254	13.740157
2	ACK	58	265	6.456604
3	ALB	172	439	23.620525
4	ANC	6	8	12.875000
5	ATL	1180	17215	12.509824
6	AUS	993	2439	13.025641

dplyr Package

Example 3.22 (Cont.): Summarize and Aggregate

	dest	planes	flights	delay
1	ABQ	108	254	13.740157
2	ACK	58	265	6.456604
3	ALB	172	439	23.620525
4	ANC	6	8	12.875000
5	ATL	1180	17215	12.509824
6	AUS	993	2439	13.025641

dplyr Package

- Chaining
  - The dplyr API is functional in the sense that function calls don't have side-effects.
  - Results must be saved often. This doesn't lead to particularly elegant code, especially if many operations are needed.
  - dplyr provides the %>% operator.
    - `x %>% f(y)` turns into `f(x, y)` so you can use it to rewrite multiple operations that you can read left-to-right, top-to-bottom

dplyr Package

Example 3.23: Chaining

```
delayreport=flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
View(delayreport)
```

	year	month	day	arr	dep
1	2013	1	16	34.24736	24.61287
2	2013	1	31	32.60285	28.65836
3	2013	2	11	36.29009	39.07360
4	2013	2	27	31.25249	37.76327

Case 2

- Case Study on Dataset Analysis with ETL
- You are to build a simple ETL process for the Art Database

References

- http://playingwithr.blogspot.com/2011/05/accessing-mysql-through-r.html
- http://www.inside-r.org/packages/cran/data.table/docs/fread
- http://www.r-tutor.com/r-introduction/data-frame/data-import
- http://www.cyclismo.org/tutorial/R/input.html
- http://www.r-bloggers.com/accessing-mysql-through-r/
- https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html