Week 5

```
#1)
librar y(DBI)
librar y(R SQLite)
2)
test_conn <- dbConn -
ect (RS QLi te: :SQ -
Lite(),
" tes t_d b.s qli te")
3)
file.i nfo ("te st_ -
db.s ql ite ")
4)
test_conn
```

Needed Library's | Create Connection Special File | List Files in Folder | Test connection find where it exists

Week 5 | Basic Operations

Week 5 | Useful Commands

```
This are the main ones
we'll need.
Connec tions
dbConn ect()
dbDisc onn ect()
dbCanC onn ect()
Finding out what is in
the database
dbList Tab les()
dbExis tsT able()
dbList Fie lds()
Fetching data from and
Writing data to the
database
dbRead Table()
dbWrit eTa ble() (note
overwrite and append
options)
dbRemo veT able()
dbGetQ uery()
Contro lling queries and
changes to the database
dbExec ute()
dbBegin()
dbCommit()
dbRoll back()
dbFetch()
```

Week 5 | Read + Delete Table

```
1)
test_conn <- dbConn -
ect (RS QLi te: :SQ -
Lite(), " tes t_d b.s -
qli te")
dbRead Tab le( tes -
t_conn, " vic sch ool -
s")
2)
vv <- dbRead Tab le( -
tes t_conn, " vic sch -
ool s")
vv
3)
dbList Tab les (te st_ -
conn)
```

Week 5 | Read + Delete Table (cont)

```
> dbRemoveTable(test_conn, "-
vicschools")
```

Reading | Storing | Deleting

Week 5 | Rolling

```
By default when SQLite
starts it is in auto-
commit mode:
so that all changes that
are requested are
automa tically made
permanent.
To make a set of
tenative changes enter
commit mode using
the dbBegin() command:
dbBegi n(t est _conn)
Then make a series of
changes to the database.
If you want to keep the
changes go:
dbComm it( tes t_conn)
or if you want to
abandon the changes go:
dbRoll bac k(t est -
_conn)
This abandons all
changes made after the
dbBegin() statement.
After either of these
two calls (dbCommit or
dbRoll back)
the database is back in
auto-c ommit mode
```

Week 5 | Rolling

```
By default when SQLite
starts it is in auto-
commit mode:
so that all changes that
are requested are
automa tically made
permanent.
```

Week 5 | Rolling (cont)

```
> To make a set of tenative
changes enter commit mode
using
the dbBegin() command:
dbBegin(test_conn)
Then make a series of changes
to the database.
If you want to keep the changes
go:
dbCommit(test_conn)
or if you want to abandon the
changes go:
dbRollback(test_conn)
This abandons all changes
made after the dbBegin()
statement.
After either of these two calls
(dbCommit or dbRollback)
the database is back in auto-c-
ommit mode
```

Week 6 | Using SQL select

```
1)
librar y(DBI)
librar y(R SQLite)
test_conn <- dbConn -
ect (RS QLi te: :SQ -
Lite(), " tes t_d b.s -
qli te")
surf <- read.c sv( " -
sur f.c sv")
dbWrit eTa ble (te st_ -
conn, " sur fsh ort ",
surf[1 :10 ,1:8],
overwr ite =TRUE)
2)
dbGetQ uer y(t est -
_conn, " SELECT * FROM
surfsh ort ")
ss <- dbGetQ uer y(t -
est _conn, " SELECT *
FROM surfsh ort ")
3)
```

```
1)
dbList Tab les (te st_ -
conn)
2)
schools <- data.f ram -
e(S chool=
                        -
                        -
c("M ath ematics and
Statis tic s"),
                        -
                      -
Cod e=c ("SM S","S GEE -
S"),
                        -
                      -
Fac ult y=c ("Sc ien -
ce", " Sci enc e")
                        -
                    )
schools
3)
dbWrit eTa ble (te st_ -
conn, " vic sch ool s",
schools, overwr ite -
=TRUE)
dbList Tab les (te st_ -
conn)
4)
file.i nfo ("te st_ -
db.s ql ite ")
5)
dbDisc onn ect (te st_ -
conn)
```

Test whats in database | Define Dataframe in R | Copy into new database table called vicschools | Check db file size | Need to disconnect at end of session

Week 6 | Using SQL select (cont)

> `{sql connection=test_conn}
SELECT * FROM surfshort
`

4)
SELECT marital, gender FROM surfshort

Create n start table | Select n show table n save to var | What tp write in markdown | select specific column

Week 6 | Where Clause

```
1)
SELECT *
FROM surfshort
WHERE Marita l="n eve r"
2)
SELECT *
FROM surfshort
WHERE Marita l<> " nev -
er"
3)
SELECT * FROM surfshort
WHERE Marita l="n eve r"
4)
SELECT Age, Gender
FROM surfshort
WHERE Marita l="n eve r"
AND Qualif ica tio -
n="s cho ol"
5)
SELECT Age, Gender,
Qualif ica tion,
Marital, Marital,
Qualif ication
FROM surfshort
WHERE (Marit al= " nev -
er" AND Qualif ica tio -
n="s cho ol") OR
Marita l="m arr ied "
6)
```

Week 6 | Where Clause (cont)

> SELECT Age, Gender AS Sex, Qualification, Marital, Marital AS MaritalStatus, Qualification
FROM surfshort
WHERE (Marital="never" AND Qualification="school") OR Marital="married"
7)
Common operators we want to use in WHERE clause are:
AND
OR
NOT
and we make comparisons with
=, <>
>, >=, <, <=
LIKE
IN
IS NULL, IS NOT NULL
Here NULL is the way SQL refers to missing data.
8)
SELECT Marital, Age, Qualification
FROM surfshort
WHERE Age IN (34,35,36,45)
ORDER BY Age DESC, Qualification

Select using Conditions | Select not equal | select equal | multiple condition | Mulitple condition v2 pro | rename based on query | Common operators | Order by Ascending

Week 6 | Creating Tables Manipulation

```
1)
CREATE TABLE lecturers (
    fir st_name TEXT,
    las t_name TEXT,
    sta rt_week INTEGER,
    end _week INTEGER,
    school TEXT
)
2)
SELECT * FROM lecturers
3)
INSERT INTO lecturers
(first _name, last_name,
school)
VALUES
("Ri cha rd", " Arn -
old " ,"SM S"),
("Lo uis e","M cMi lla -
n","S MS"),
("Ry an", " Adm ira -
al", " SMS "),
("Jo hn", " Hay woo -
d","S MS")
4)
SELECT * FROM lecturers
5)
UPDATE lecturers
SET start_ week=1,
end_week=6
WHERE first_name = " -
Ric har d"
6)
UPDATE lecturers SET
school ="Ma the matics
and Statis tic s"
7)
DELETE FROM lecturers
WHERE first_ nam e="J -
ohn "
8)
Delete Bunch
```

Week 6 | Creating Tables Manipulation (cont)

> 9)
DROP TABLE lecturers
10)
SELECT Marital, COUNT(*)
FROM SURF
GROUP BY Marital
SELECT Marital, COUNT(*) AS Number, MIN(Age) as AgeMin, MAX(Age) as AgeMax
FROM SURF
WHERE Gender = "female"
GROUP BY Marital

Create table | Insert Data | Insert Data V2 | Checking | Modify | Mulitple Rows at once | Delete | Delete bunch | Delete Table | Counts

Week 6 | Joins

1)
SELECT *
FROM students LEFT JOIN enrolments
ON students.idno=enrolments.idno
ORDER BY idno
2)
SELECT *
FROM enrolments INNER JOIN students
ON students.idno=enrolments.idno
ORDER BY idno
3)
SELECT students.idno, enrolments.idno, "first.name", "last.name", course, grade
FROM students LEFT JOIN enrolments

Week 6 | Joins (cont)

ON students.idno=enrolments.idno
UNION
SELECT students.idno, enrolments.idno, "first.name", "last.name", course, grade
FROM enrolments LEFT JOIN students
ON students.idno=enrolments.idno
ORDER BY students.idno
4)
xx <- data.frame(colour=c("Red","Green","Blue"), height=c("Tall","Tall","Short"))
yy <- data.frame(width=c("wide","narrow"))
dbWriteTable(test_conn, "xx", xx, overwrite=TRUE)
dbWriteTable(test_conn, "yy", yy, overwrite=TRUE)
5)
merge(xx, yy)
merge(students, enrolments, by="idno")
6)
Thus when combining two datasets with merge():
all=FALSE (the default) keeps only matching records (inner join)
all=TRUE keeps all records from both datasets, whether matching or not (full outer join)
all.x=TRUE keeps all records from the first dataset (left join)

Week 6 | Joins (cont)

all.y=TRUE keeps all records from the second dataset (right join)
and
by=NULL does not use a matching key (cross join)
by="xxx" matches on column xxx in both tables
by.x="xxx", by.y="zzz" mathches column xxx in the first table with column zzz in the second. Thus if the matching key has different names in the two tables then the merge() command allows us to specify them separately.

Left Join | Inner Join | Full outer join | Cross Join | Merging in R | Sussy

Week 6 | Subquery

```
1)
We can use a subquery to define and populate a table
CREATE TABLE counts
AS
SELECT idno, COUNT(*) AS ncourses
FROM enrolments
GROUP BY idno
2)
dbRemo veT abl e(t est -_conn, " cou nts ")
3)
CREATE TABLE counts
(idno INTEGER,
  nco urses INTEGER)
4)
```

Week 6 | Subquery (cont)

> INSERT INTO counts (idno, ncourses)
SELECT idno, COUNT(*) AS ncourses
FROM enrolments
GROUP BY idno
5)
SELECT *
FROM counts
WHERE ncourses = (SELECT MAX(ncourses) FROM counts)
6)
SELECT grade, COUNT(*) AS num
FROM enrolments
GROUP BY grade
7)
SELECT grade, COUNT(*) AS num,
    ROUND(COUNT(*)100.0/-(SELECT COUNT(*) FROM enrolments),1) AS pct
FROM enrolments
GROUP BY grade
8)
CREATE TABLE patients (
  PatientID INTEGER,
  FirstName TEXT,
  LastName TEXT,
  DateOfBirth TEXT,
  PRIMARY KEY(PatientID)
)
9)
INSERT INTO patients (PatientID, FirstName, LastName, DateOfBirth)
VALUES
(1121, "Richard", "Arnold", "-1/1/1965"),

Week 6 | Subquery (cont)

> (2155, "Ella", "Li", "6/7/1999"),
(2338, "Gemma", "Watson", "-18/3/2001")
10) Chaning existing
DROP TABLE IF EXISTS simple
CREATE TABLE simple
(name TEXT)
INSERT INTO simple (name)
VALUES
('Richard'),
('John'),
('Louise')
We can rename the table
DROP TABLE IF EXISTS csimple
ALTER TABLE simple RENAME TO csimple
dbListTables(test_conn)
11)
We can insert further rows using a query to
INSERT INTO simple
SELECT * FROM csimple
Though we have to be sure that the column names coming in from csimple match those in simple or the INSERT won't work.
We can add a column to an existing table:
ALTER TABLE simple
ADD COLUMN
first_week INTEGER
UPDATE simple SET first_week = 1 WHERE name = 'Richard'
SELECT * FROM simple

Week 6 | Subquery (cont)

> 12)
Renaming a column is easy too:
ALTER TABLE simple RENAME COLUMN first_week to firstweek
13) date n time
DROP TABLE IF EXISTS dates
CREATE TABLE dates (datestring TEXT)
INSERT INTO dates (datestring) VALUES
('2020-01-01'),
('1977-12-25')
We can output any format we like using SELECT and various conversion functions.
SELECT datestring, strftime("-%d/%m/%Y", datestring) FROM dates

Create table | Remove Table | Create Table n Populate | Use where to find specific | Group by | Convert to percentages Rounded | Create primary key table so only one key per person| insert into new table | renaming |

Week 7 | dyplr n tidyr

```
library(dplyr)
librar y(t idyr)
```

1)
Copying and renaming columns
Copying a data frame in base R
surf.copy <- surf

Week 7 | dyplr n tidyr (cont)

> Copying a data frame in dplyr
surf.copy <- rename(surf)
but with the ability to rename columns as we go
surf.copy <- rename(surf, Sex=Gender, Highest_Qualification=Qualification)
surf[1:2,]
2)
Selecting specific columns
To list just the Age and Income columns in surf in Base R we go
ageinc <- surf[,c("Age","Income")]
ageinc[1:3,]
Age Income
1 15 87
2 40 596
3 38 497
The select() function in dplyr allows us to go
ageinc <- select(surf, Age, Income)
ageinc[1:3,]
3)
We can also omit columns, using the negative sign before the name
noageinc <- select(surf, -Age, -Income)
noageinc[1:3,]
4)

Week 7 | dyplr n tidyr (cont)

> and just like in Base R we can select columns by specifying their numeric locations:
surf[1:3, c(1,6,7)]
5)
base
surf[surf$Gender=="female" & surf$Income>900,]
In dplyr we can use the filter() function to achieve this
filter(surf, Gender=="female" & Income>900)
6)
tidyr
surf[surf$Gender=="female" & surf$Income>900,]
base
surf[which(surf$Gender=="female" & surf$Income>900),]
7) near certain tolerance
filter(starwars, near(height, 170, tol=5))
8)
base
Reordering a data frame
We may want to reorder the rows of a data set by one more more variables. In base R the order() command allows us to to this.
Here are the male high earners:
mhe <- filter(surf, Income>1200, Gender=="male")
mhe

Week 7 | dyplr n tidyr (cont)

> sort(mhe$Age)
order(mhe$Age)
tidy
arrange(mhe, Age)
two or more variables
mhe[order(mhe$Qualification, mhe$Age),]
9)
Creating new columns
In Base R we can create new columns by simply referring to a name that does not yet exist
mhe$AgeSquared <- mhe$Age^2
In dplyr we use the mutate() function - and we can create multiple new columns in one step:
mhe <- mutate(mhe, N=nrow(mhe), AgeSquared=Age^2, AgeCubed=Age^3)
mhe
10)

```
subsurf <- surf %>%
        select(-X) %>%
        rename(Sex=Gender) %>%
        filter(Qualification%in%c("vocational","degre")) %>%
        mutate(AgeSquared=Age^2)
```

In this chain of piped substatements, the pipe sends the output of each substatement to be the first argument of the function in the following substatement. We only specify the second and subsequent arguments.

Week 7 | dyplr n tidyr (cont)

> 11)
Now convert it to a tibble:
mtcars <- as_tibble(mtcars)
We can convert a tibble back to a standard data frame with as.data.frame()
mtcars <- as.data.frame(mtcars)

Week 7 | Extra

```
1)Full join
Keep all entries from A
 and B
  (i.e., keep entries in
A
 that do not have a
match in B
, and keep entries in B
 that do not have a
match in A
).
merge(A, B, all = TRUE,
...)
merge(A, B, all.x =
TRUE, all.y = TRUE, ...)
full_j oin(A, B, by =
id, ...)
Entries in A
 that do not have
matches in B
 will have NAs in fields
from B
, and vice versa.
merge( stu dents,
enrolm ents, by=" id",
all=TRUE)
2)
In base R we can use
reshape()
reshap e(f ail ure -
_data,
```

Week 7 | Extra (cont)

```
>       idvar="Course",
      varying=c("D","E","Withdra-
w"),
      times=c("D","E","Withdr-
aw"),
      timevar="Result",
      v.names=c("Percentage"),
      direction="long")
```

We will do this using the pivot_-longer function, from the tidyr package:

```
library(tidyr)
failure_long <- pivot_longer(fa-
ilure_data,
                    cols = c(D, E,
Withdraw),
                    names_to = "-
Result",
                    values_to = "-
Percentage")
failure_long <- arrange(failure-
_long, Course)
kable(failure_long)
3)
Transforming Data From Long to
Wide Format
reshape(as.data.frame(gdp_l-
ong_2000s),
      timevar="year",
      v.names="gdpPercap",
      direction="wide",
      idvar="country")
gdp_wide <- pivot_wider(gdp_lo-
ng_2000s, names_from = year,
values_from = gdpPercap)
kable(gdp_wide)
```

Week 8 | GGplot

```
1)
Bar charts for catego -
rical variables
barplo t(t abl e(r ugb -
y$p osi tion), xlab="",
ylab="C oun t", las=2)
librar y(g gplot2)
# Two ways to produce
exactly the same bar
chart of player
position.
ggplot (rugby, aes(x =
position)) +
    geo m_bar()
FLIP
ggplot (rugby) +
    geo m_b ar( aes(x =
position)) +
    coo rd_ flip()
LABELS n THEME
ggplot (rugby) +
    geo m_b ar( aes(x =
position, y = (..cou -
nt..) / sum(..c ou -
nt..))) +
    labs(x = " Pos iti -
on", y = " Pro por tio -
n", title= " Dis tri -
butions over positi -
ons ") +
    the me( axi s.title
= elemen t_t ext (si -
ze=20))
2)
BOXPLOT
ggplot (rugby) +
    geo m_b oxp lot -
(aes(x = weight _kg)) +
    lab s(x ="Weight
(kg)") +
    coo rd_ flip()
3)
HISTOGRAM
ggplot (rugby) +
```

Week 8 | GGplot (cont)

```
>  geom_histogram(aes(x =
weight_kg, y = ..density..),
binwidth = 5) +
  labs(x = "Weight (kg)", y = "-
Density")
4)
Frequency polygons and density
plots for numeric variables
ggplot(rugby) +
  geom_freqpoly(aes(x =
weight_kg, y = ..density..),
binwidth = 5) +
  labs(x = "Weight (kg)", y = "-
Density")
ggplot(rugby, aes(x = weight_kg,
y = ..density..)) +
  geom_histogram(binwidth = 5)
+
  labs(x = "Weight (kg)", y = "-
Density") +
  geom_freqpoly(binwidth = 5)
ggplot(rugby) +
  geom_freqpoly(aes(x = weight-
_kg), stat = "density")
5)
SCATTER TWO VARIABLES
plot.settings <- ggplot(rugby,
aes(x = height_cm, y = weight-
_kg)) +
  labs(x = "Height (cm)", y = "-
Weight (kg)") +
  theme_classic()
6)
HEXSCATTER
library(hexbin)
plot.settings +
```

Week 8 | GGplot (cont)

> 7)
our.scatterplot <- plot.settings +
geom_point(p-
osition = "jitter") # Scatterplot of weight versus height.
our.scatterplot +
geom_smooth(method = "lm")
8)
BUNCH OF GRAPHS
ggplot(rugby) +
geom_point(aes(x = height_cm, y = weight_kg), position = "jit-ter") +
facet_wrap(~position)
9)
Sidebyside box
ggplot(rugby) +
geom_boxplot(aes(x = position, y = weight_kg)) +
10)
Summary of Plot Types
Plot type geom type aes options Additional arguments
Bar chart geom_bar x, y, fill position = "fill", position = "dod-ge", stat = "identity"
Histogram geom_histogram x binwidth, bins
Boxplot geom_boxplot x, y boxplot()
Scatterplot geom_point x, y, colour, size, shape position = "-jitter"
Line of best fit overlay, line plot geom_line x, y, colour, linetype size

Week 8 | GGplot (cont)

> Hexagonally binned scatterplot geom_hex x, y binwidth, bins
Bar/column chart geom_col x, y, fill barplot()
labs(x="Position", y="Weight (kg)")
geom_hex()
plot.settings + geom_point()