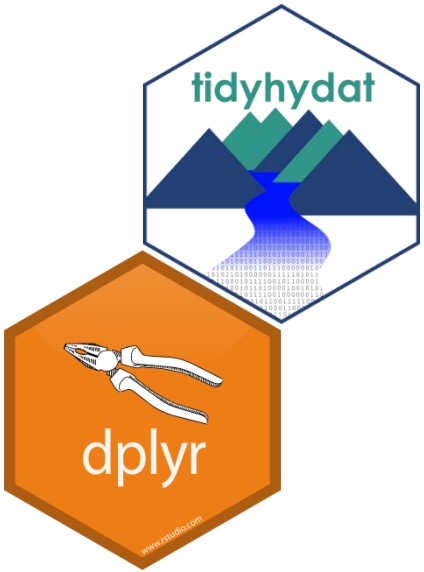
Transform Data



tidyhydat functions

Import and tidy Water Survey of Canada data

```
hy_*(STATION_NUMBER, PROV_TERR_STATE_LOC, ...)
```

```
realtime_*(STATION_NUMBER, PROV_TERR_STATE_LOC, ...)
```

For this exercise use: `hy_stn_data_range()`

hy_stn_data_range()

Your turn

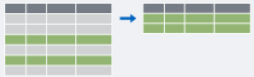
- What does `hy_stn_data_range()` tell us?
- Create an object from `hy_stn_data_range()` called `data_range`

tidyhydat: Data Ranges

```r
data_range <- hy_stn_data_range()
```

```
# A tibble: 11,854 x 6
   STATION_NUMBER DATA_TYPE SED_DATA_TYPE YEAR_FROM YEAR_TO RECORD_LENGTH
   <chr>          <chr>     <chr>             <int>   <int>         <int>
 1 01AA002        Q         NA                 1967    1977            11
 2 01AD001        Q         NA                 1918    1997            80
 3 01AD002        Q         NA                 1926    2014            89
 4 01AD003        H         NA                 2011    2016             6
 5 01AD003        Q         NA                 1951    2016            66
 6 01AD004        H         NA                 1980    2016            32
 7 01AD004        Q         NA                 1968    1979            12
 8 01AD005        H         NA                 1966    1974             9
 9 01AD008        H         NA                 1972    1974             3
10 01AD009        H         NA                 1973    1982            10
# ... with 11,844 more rows
```

dplyr: Data manipulation

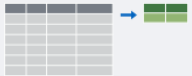


Extract cases with **filter()**

Extract variables with **select()**

Arrange cases, with **arrange()**.

Make new variables, with **mutate()**.



Make tables of summaries with **summarise()**.

along with **group_by()**

filter()

filter()

Extract rows that meet logical criteria.

```
filter(.data, ... )
```

data frame to transform

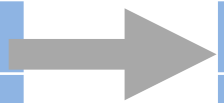
one or more logical tests (filter returns each row for which the test is TRUE)

filter()

Extract rows that meet logical criteria.

```
filter(data_range, DATA_TYPE == "Q")
```

STATION_NUMBER	DATA_TYPE	…
01AA002	Q	
01AD001	Q	
01AD002	Q	
01AD003	H	
01AD003	Q	
01AD004	H	



STATION_NUMBER	DATA_TYPE	…
01AA002	Q	
01AD001	Q	
01AD002	Q	
01AD003	Q	

filter()

Extract rows that meet logical criteria.

```
filter(data_range, DATA_TYPE == "Q")
```

= sets
(returns nothing)
== tests if equal
(returns TRUE or FALSE)

Logical tests

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Your turn

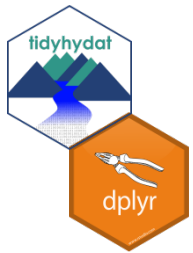
Try using logical operators to show the following stations that :

1. Began operation before and including the year 1900
2. Are not discharge stations (i.e. "Q")
3. Have more than 100 years of data

```
filter(data_range, YEAR_FROM <= 1900)
```

```
filter(data_range, DATA_TYPE != "Q")
```

```
filter(data_range, RECORD_LENGTH > 100)
```



Two common mistakes

1. Use = instead of ==

```
filter(data_range, DATA_TYPE = "Q")
filter(data_range, DATA_TYPE == "Q")
```

2. Forgetting quotes

```
filter(data_range, DATA_TYPE == Q)
filter(data_range, DATA_TYPE == "Q")
```

filter()

Extract rows that meet *every* logical criteria.

additional arguments must also be TRUE

```
filter(data_range, DATA_TYPE == "Q", YEAR_FROM > 1920)
```

STATION_NUMBER	DATA_TYPE	YEAR_FROM
01AA002	Q	1967
01AD001	Q	1918
01AD002	Q	1926
01AD003	H	2011
01AD003	Q	1951
01AD004	H	1980

STATION_NUMBER	DATA_TYPE	YEAR_FROM
01AA002	Q	1967
01AD002	Q	1926
01AD003	Q	1951

Boolean operators

?base::Logic

a & b	and
a | b	or
!a	not

Your turn

Use filter to:

1. Find stations that are level ("H") and have more than 80 years record
2. Find stations that start before 1890 *or* after 2016
3. These stations in one data.frame 08KA009, 05JA005, 06AC006 (hint look at %in%)

```r
filter(data_range, DATA_TYPE == "H" & RECORD_LENGTH > 80)
```

```r
filter(data_range, YEAR_FROM < 1890 | YEAR_FROM > 2016)
```

```r
filter(data_range, STATION_NUMBER %in% c("08KA009","05JA005","06AC006"))
```

Two more common mistakes

3. Collapsing multiple tests into one

```
filter(data_range, 1960 < YEAR_FROM < 1980)
filter(data_range, 1960 < YEAR_FROM, YEAR_FROM < 1980)
```

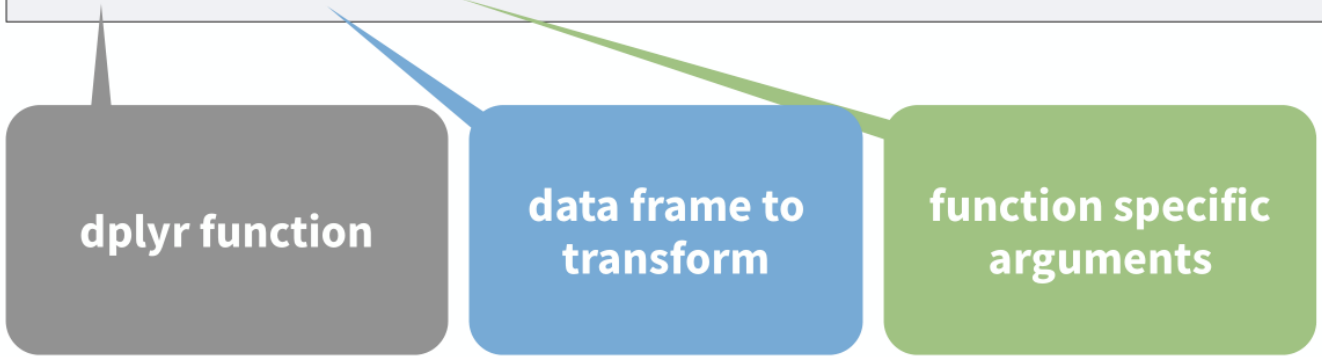
4. Stringing together many tests (when you could use %in%)

```
filter(data_range,
STATION_NUMBER == "08KA009" | STATION_NUMBER == "05JA005")
filter(data_range, STATION_NUMBER %in% c("08KA009","05JA005"))
```

dplyr common syntax

Each function takes a data frame / tibble as its first argument and returns a data frame / tibble.

```
filter(.data, ...)
```



pull()

pull()

Pull out a single variable

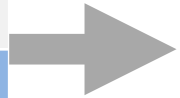
```
pull(.data, ...)
```

pull()

Pull out a single variable

```
pull(data_range, STATION_NUMBER)
```

STATION_NUMBER	DATA_TYPE	YEAR_FROM
01AA002	Q	1967
01AD001	Q	1918
01AD002	Q	1926
01AD003	H	2011
01AD003	Q	1951
01AD004	H	1980



STATION_NUMBER
01AA002
01AD001
01AD002
01AD003
01AD003
01AD004

Your turn

1. Extract YEAR_FROM and STATION_NUMBER from data_range into distinctly named objects
2. Inspect the created objects

```
year_from_vector <- pull(data_range, YEAR_FROM)
```

```
stns <- pull(data_range, STATION_NUMBER)
```

Pipe %>%

Multistep Operations

Consider the following:

Filter data for only rows that have record length greater than 100 then **extract** the station number.

Option 1: Use intermediate variables

```
rows_gt_100 <- filter(data_range, RECORD_LENGTH > 100)
pull(rows_gt_100, STATION_NUMBER)
```

Multistep Operations

Consider the following:

Filter data for only rows that have record length greater than 100 then **extract** the station number.

Option 2: Do it all on one line

```
pull(filter(data_range, RECORD_LENGTH > 100), STATION_NUMBER)
```

The pipe operator %>%

Passes result on left into first argument of function on right.

```
data_range %>% filter(_______, DATA_TYPE == "Q")
```

These do the same thing. Try it:

```
filter(data_range, DATA_TYPE == "Q")
data_range %>% filter(DATA_TYPE == "Q")
```

Multistep Operations

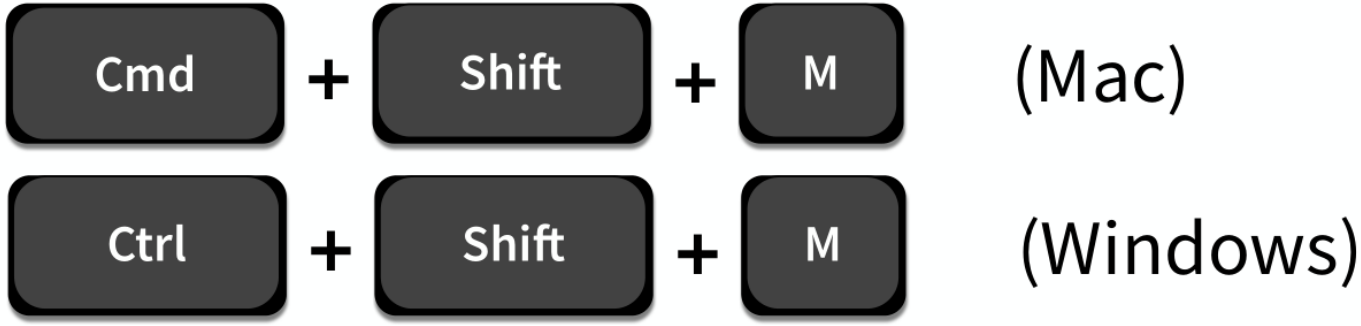
Consider the following:

Filter data for only rows that have record length greater than 100 then **extract** the station number.

Option 3: Use the pipe

```
data_range %>%
   filter(RECORD_LENGTH > 100) %>%
   pull(STATION_NUMBER)
```

Shortcut to type %>%



mutate()

mutate()

Adds columns. Convert cms to ft per second

```
hy_monthly_flows("08MF005") %>% mutate(Value_cfs = Value = 0.0167)
```

....	Sum_stat	Value
	MAX	578
	MEAN	485
	MIN	399
	TOTAL	15038
	MAX	2070
	MEAN	1150

....	Sum_stat	Value	Value_cfs
	MAX	578	9.65
	MEAN	485	8.10
	MIN	399	6.66
	TOTAL	15038	251.13
	MAX	2070	34.57
	MEAN	1150	19.20

mutate()

Can also use functions in mutate:

ifelse() – Create a test then return separate values if the test is TRUE or FALSE

```
x <- 1:10
ifelse(x > 5, "large", "small")
```

mutate()

Can also use functions in mutate

```
hy_monthly_flows("08MF005") %>% mutate(Value_cfs = Value * 0.0167)
```

....	Sum_stat	Value
	MAX	578
	MEAN	485
	MIN	399
	TOTAL	15038
	MAX	2070
	MEAN	1150

....	Sum_stat	Value	Value_cfs
	MAX	578	9.65
	MEAN	485	8.10
	MIN	399	6.66
	TOTAL	15038	251.13
	MAX	2070	34.57
	MEAN	1150	19.20

Your turn

Create another variable from hy_monthly_flows() called *category* that categorises the *Value* into two groups:

- greater than 500
- below and including 500

```
hy_monthly_flows("08MF005") %>%
   mutate(category = ifelse(Value > 500, "Large", "Small"))
```

summarise()

Need some data to work with

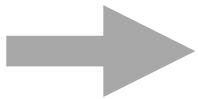
```
annual_gt_100 <- hy_stn_data_range() %>%
    filter(RECORD_LENGTH > 100, DATA_TYPE == "Q") %>%
    pull(STATION_NUMBER) %>%
    hy_annual_stats()
```

summarise()

Compute table of summaries

```
annual_gt_100 %>%
    summarise(mean_value = mean(Value))
```

STATION_NUMBER	Parameter	Year	Sum_stat	Value	...
02CA001	Flow	1860	MEAN	2250	
02HA003	Flow	1860	MEAN	6760	
02CA001	Flow	1860	MIN	NA	
02HA003	Flow	1860	MIN	NA	



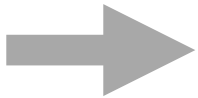
mean_value
1222.599

summarise()

Compute table of summaries

```
annual_gt_100 %>%
   summarise(mean_value = mean(Value, na.rm = TRUE),
             min_value = min(Value, na.rm = TRUE))
```

STATION_NUMBER	Parameter	Year	Sum_stat	Value	…
02CA001	Flow	1860	MEAN	2250	
02HA003	Flow	1860	MEAN	6760	
02CA001	Flow	1860	MIN	NA	
02HA003	Flow	1860	MIN	NA	



mean_value	min_value
1222.599	0

Summary functions

Take a vector as input, return a single value as output.

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

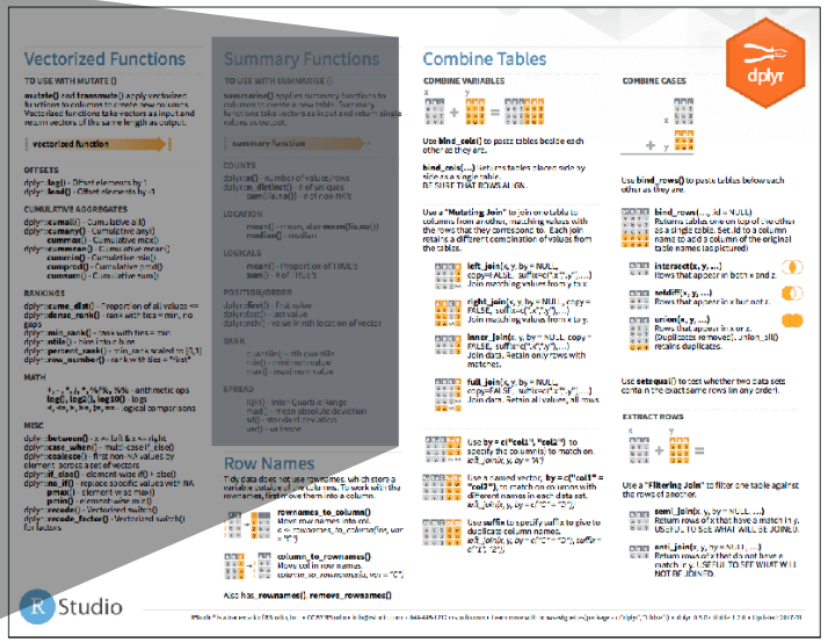
dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - mean absolute deviation
sd() - standard deviation
var() - variance



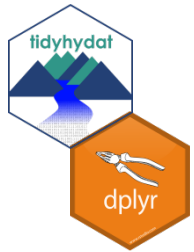
dplyr

Your turn

Use summarise() to compute three statistics about the data:

- The first (minimum) year in the dataset
- The median annual flow (Value) in the dataset
- The number of stations represented in the data (Hint: use cheatsheet)

```
annual_gt_100 %>%
    summarise(min_year = min(Year, na.rm = TRUE),
              median_value = median(Value, na.rm = TRUE),
              num_stations = n_distinct(STATION_NUMBER))
```



Your turn

- Compute the summaries as before but only the mean annual flow (i.e. where Sum_stat is MEAN)

```
annual_gt_100 %>%
  filter(Sum_stat == "MEAN") %>%
  summarise(min_year = min(Year, na.rm = TRUE),
            median_value = median(Value, na.rm = TRUE),
            num_stations = n_distinct(STATION_NUMBER))
```

Grouping Cases

group_by()

Groups cases by common values of one or more columns.

```
annual_gt_100 %>%
    group_by(Year)
```

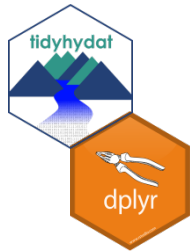
```
# A tibble: 18,957 x 7
# Groups:   Year [158]
   STATION_NUMBER Parameter  Year Sum_stat Value Date       Symbol
   <chr>          <chr>     <int> <chr>    <dbl> <date>     <chr>
 1 02CA001        Flow       1860 MEAN      2250 NA         NA
 2 02HA003        Flow       1860 MEAN      6760 NA         NA
 3 02CA001        Flow       1860 MIN         NA NA         NA
 4 02HA003        Flow       1860 MIN         NA NA         NA
 5 02CA001        Flow       1860 MAX         NA NA         NA
 6 02HA003        Flow       1860 MAX         NA NA         NA
 7 02CA001        Flow       1861 MEAN      2280 NA         NA
```

group_by()

Groups cases by common values of one or more columns.

```
annual_gt_100 %>%
    group_by(Year) %>%
    summarise(num_stations = n_distinct(STATION_NUMBER))
```

Year	num_stations
1860	2
1861	2
1862	2
…	…



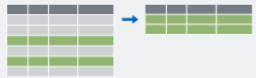
```
# A tibble: 18,957 x 7
# Groups:   STATION_NUMBER [54]
   STATION_NUMBER Parameter  Year Sum_stat Value Date       Symbol
   <chr>          <chr>     <int> <chr>    <dbl> <date>     <chr>
 1 02CA001        Flow       1860 MEAN      2250 NA         NA
 2 02HA003        Flow       1860 MEAN      6760 NA         NA
 3 02CA001        Flow       1860 MIN         NA NA         NA
 4 02HA003        Flow       1860 MIN         NA NA         NA
 5 02CA001        Flow       1860 MAX         NA NA         NA
 6 02HA003        Flow       1860 MAX         NA NA         NA
 7 02CA001        Flow       1861 MEAN      2280 NA         NA
 8 02HA003        Flow       1861 MEAN      6750 NA         NA
 9 02CA001        Flow       1861 MIN         NA NA         NA
10 02HA003        Flow       1861 MIN         NA NA         NA
# ... with 18,947 more rows
```

Your turn

- Compute the mean annual flow of stations with more than 100 years of record grouped by station number

```
annual_gt_100 %>%
  filter(Sum_stat == "MEAN") %>%
  group_by(STATION_NUMBER) %>%
  summarise(mean_mad = mean(Value, na.rm = TRUE))
```

dplyr: Data manipulation



Extract cases with **filter()**

Extract variables with **select()**

Arrange cases, with **arrange()**.

Make new variables, with **mutate()**.

Make tables of summaries with **summarise()**.

along with **group_by()**

select()

Select variables by name

```
annual_gt_100 %>%
   select(STATION_NUMBER, Value)
```

arrange()

Arrange rows by variables

```
annual_gt_100 %>%
   arrange(Value)
```

Challenge

Extract daily flow information from all active stations on Prince Edward Island.

Hint

Use tidyhydat function `hy_stations()` and `hy_daily_flows()`

Challenge

```
pe_stns <- hy_stations(prov_terr_state_loc = "PE") %>%
    filter(HYD_STATUS == "ACTIVE") %>%
    pull(STATION_NUMBER)
pe_flows <- hy_daily_flows(pe_stns)
```

1. Simple

They do one thing, and they do it well

filter() - extract **cases**
arrange() - reorder **cases**
group_by() - group **cases**
select() - extract **variables**
mutate() - create new **variables**
summarise() - summarise **variables** / create **cases**

2. Composable

They can be combined with other functions for multi-step operations

```
gapminder %>%
  filter(year == 2007) %>%
  arrange(desc(lifeExp))
```

Each dplyr function takes a tibble as its first argument and returns a tibble. As a result, you can directly pipe the output of one function into the next.

3. Smart

They can use R objects as input.

```
years <- 2001:2011
gapminder %>%
  filter(year %in% years)
```

Found in .data, no need for $

Found in global environment