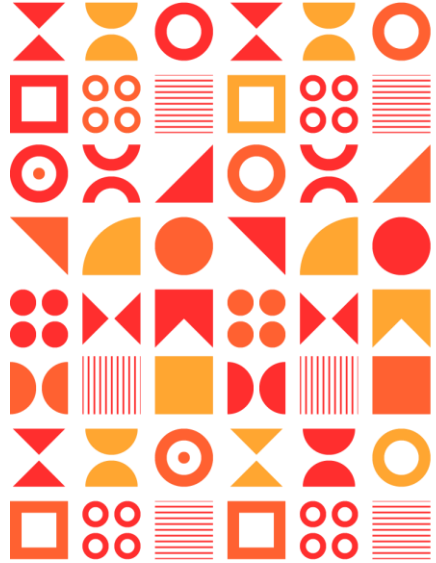

Data Manipulation and Transformation

semicolon

Learning Outcomes

- How Manipulating/Reshape Data Using tidyverse

Packages and data

We will be using the tidyverse package for this class. The data file for this class has been provided in the data folder, it's named mdta.xlsx. It is a data on passengers boarding and alighting at all stations on all lines of the Massachusetts Bay Transportation Authority (MBTA) commuter rail system.

Reading the Data

```
library(readxl)
library(tidyverse)
dta<-read_excel("data/mbta.xlsx",skip = 1,
range = cell_cols(2:60))
View(dta)
```

Gathering the years

The *gather* function in *tidyr* package helps in gathering multiple columns and collapses the columns into key-value pairs as seen below.

```
dta_tidy <- dta %>% gather(`2007-01`:`2011-10`,
                           key = "year", value = "passengers")
View(dta_tidy)
```

Separating year

The *separate* function in *tidyr* turns a single character column into multiple columns as seen below

```
dta_tidy <- dta_tidy %>% separate(year, into = c("year", "month"))
View(dta_tidy)
```

Spreading mode of transportation

The *spread* function helps in spreading a key-value pair across multiple columns

```
dta_tidy <- dta_tidy %>% spread(mode, passengers)
dta_tidy
```

Subsetting the needed columns

We may be interested in certain columns, we can apply our knowledge of subsetting to select the needed columns for our analysis

```
dta_tidy <- dta_tidy %>% .[,c(1:2,6:8)]
dta_tidy
```

Gather rail modes

After successful selecting the columns we are interested in, then we gather columns into a single column using the *gather* function as seen below.

```
dta_tidy <- dta_tidy %>% gather(`Commuter Rail`:`Light Rail`,key="rail_type", value = passengers)
dta_tidy
```

Data transformation using dplyr

For data transformation, we will be using *hflights* data , the dataset contains all flights departing from Houston airports IAH (George Bush Intercontinental) and HOU (Houston Hobby).

```
# install.packages("hflights")
library(hflights)
data(hflights)
View(hflights)
```

Major Variable types in dplyr

- : character vectors, or strings
- : date-times (a date + a time)
- : integers
- : doubles, or real numbers
- : factors
- : dates
- : logical, vectors that contain only TRUE or FALSE

Major dplyr core functions

- filter(): Extracting rows by the rows values
- arrange(): arraning rows
- select(): selecting columns
- mutate(): creating new variables from existing variables
- summarize(): Obtaining summary of variables
- group_by(): convert existing tables into a grouped table
- rename(): renaming columns
- distinct(): finding distinct rows

In dplyr functions, the first argument is always data frame and the returned value is always a data frame as well.

filter() function

The *filter()* is used to choose rows/cases where conditions are true, basically used in subsetting a dataframe based on their row values.

Let’s select all flights of february, 2011.

```
library(hflights)
data("hflights")
filter(hflights, Year == 2011, Month == 2)
```

Year	Month	DayofMonth	DayOfWeek	DepTime	ArrTime
2011	2	1	2	1401	1539
2011	2	2	3	1420	1530
2011	2	3	4	1405	1504
2011	2	4	5	1516	1614
2011	2	5	6	1358	1505
2011	2	6	7	1350	1452
## `fil	ter()`				

filter() function

Following operators work with *filter()* function

Comparison != (not equal),== (equal),>(greater than), >=(greater than or equal), <(less than), <=(less than or equal to)

Boolean & (“and”), | (“or”), ! (“not”)

We can also use syntax such x %in% y, this is telling *filter()* to select every row where x is part of the values of y

filter()

```
fil<-filter(hflights, Dest %in% c("FLL", "IAH"))
View(fil)
```

AirTime	ArrDelay	DepDelay	Origin	Dest	Distance
107	-8	0	IAH	FLL	965
108	-16	-5	IAH	FLL	965
106	-24	-3	IAH	FLL	965
106	-21	-1	IAH	FLL	965
108	-11	6	IAH	FLL	965
111	-3	2	IAH	FLL	965

Using *between* in the filter() function

We can also use *between* to specify the particular range of values we are interested in. It takes the following form *between* (x, left, right) which is equivalen to x >= left & x <= right

Let's filter all flights that covered distance between 224 and 944 miles

```
filter(hflights, between(Distance, 224,944))[,11:16]
```

AirTime	ArrDelay	DepDelay	Origin	Dest	Distance
40	-10	0	IAH	DFW	224
45	-9	1	IAH	DFW	224
48	-8	-8	IAH	DFW	224
39	3	3	IAH	DFW	224
44	-3	5	IAH	DFW	224
45	-7	-1	IAH	DFW	224

Task One

Find all flights that

a. Departed in April, 2011

b. Operated by AA and WN

Using the arrange() function

arrange() function is used to order a dataframe by a set of columns.

Let's arrange the flights data by Year, Month

```
arr_1<-arrange(hflights, Year, Month)
```

Year	Month	DayofMonth	DayOfWeek	DepTime	ArrTime
2011	1	1	6	1400	1500
2011	1	2	7	1401	1501
2011	1	3	1	1352	1502
2011	1	4	2	1403	1513
2011	1	5	3	1405	1507
2011	1	6	4	1359	1503

arrange() in descending order

By dafault arrange() sorts values in ascending order. Use desc() to re-order by a column in descending order.

```
arrange(flights, desc(arr_delay))
```

Year	Month	DayofMonth	DayOfWeek	DepTime	ArrTime
2011	12	12	1	650	808
2011	8	1	1	156	452
2011	11	8	2	721	948
2011	6	21	2	2334	124
2011	5	20	5	858	1027
2011	6	9	4	2029	2243

Using select()function in dplyr

select() can be used to extract variables from a dataframe.

```
select(hflights, Year, Month, FlightNum, AirTime)[1:4,]
```

	Year	Month	FlightNum	AirTime
5424	2011	1	428	40
5425	2011	1	428	45
5426	2011	1	428	48
5427	2011	1	428	39

select() helper functions

select() has various helper functions:

- everything(): selects all variables.
- starts_with("def"): matches names that begin with “def”.
- ends_with("xyz"): matches names that end with “xyz”.
- contains("ijk"): matches names that contain “ijk”.

select() function

select() can be used to rename variables

```
sel_2<-select(hflights, tail_num =TailNum)[1:5,]
```

This will rename TailNum and drop all variables except tail_num.
For renaming, it advisable to use rename()

```
ren_1<-rename(hflights, tail_num = TailNum)
```

	ArrTime	UniqueCarrier	FlightNum	tail_num
5424	1500	AA	428	N576AA
5425	1501	AA	428	N557AA
5426	1502	AA	428	N541AA
5427	1513	AA	428	N403AA
5428	1507	AA	428	N492AA

select()

We can select variables that starts with Dep and Arr

```
select(hflights, starts_with("Dep"), starts_with("Arr"))
```

	DepTime	DepDelay	ArrTime	ArrDelay
5424	1400	0	1500	-10
5425	1401	1	1501	-9
5426	1352	-8	1502	-8
5427	1403	3	1513	3
5428	1405	5	1507	-3

select() Cont'd

- vars <- **c**("Year", "Month", "DayofMonth", "DayofMonth",
  "ArrTime")
  s_1<-**select**(hflights, **one_of**(vars))
  head(s_1)[1:6,]

mutate() Cont’d

mutate() adds new variables using the existing ones, it also preserves existing variables.

```
hflights %>%
  select(ends_with("Delay"), Distance, AirTime) %>%
  mutate(time_gain = ArrDelay - DepDelay,
         speed = Distance / AirTime * 60
)
```

ArrDelay	DepDelay	Distance	AirTime	time_gain	speed
-10	0	224	40	-10	336.0000
-9	1	224	45	-10	298.6667
-8	-8	224	48	0	280.0000
3	3	224	39	0	344.6154
-3	5	224	44	-8	305.4545
-7	-1	224	45	-6	298.6667

summarize()

summarize() function creates one or more scalar variables summarizing the variables of an existing table.

```
summarise(hflights, Delay = sum(DepDelay, na.rm = TRUE))
##     Delay
## 1 2121251
```

summarize() with group_by()

- summarize() with *group_by()* will result in one row in the output for each group

To summarize average delay by day

```
hflights %>%
  group_by(Year, Month, DayofMonth) %>%
  summarise(delay = mean(DepDelay, na.rm = TRUE))
```

summarize() count

For aggregations it is generally a good idea to include a count n(). For example, let's look at the (not cancelled) planes that have the highest average delays:

```
hflights %>%
  group_by(Year, Month, DayofMonth) %>%
  summarise(DepDelay =n())
```

useful functions for `summarize()`

- Measures of location: `mean(x)`, `sum(x)`, `median(x)`.
- Measures of spread: `sd(x)`, `IQR(x)`, `mad(x)`.
- Measures of rank: `min(x)`, `quantile(x, 0.25)`, `max(x)`.
- Measures of position: `first(x)`, `nth(x, 2)`, `last(x)`.
- Counts: `n()`.

End of Session 3

Any Questions ?

semicolon