Data Science for Operational Researchers Using R Online

6. Relational data with dplyr

Prof. Jim Duggan,
School of Computer Science
University of Galway.

https://github.com/JimDuggan/explore_or

Day 2 Overview

- Relational data with dplyr
- Statistical transformation - ggplot2
- Tibble manipulation with tidyr and purrr
- OR Examples
- Project work

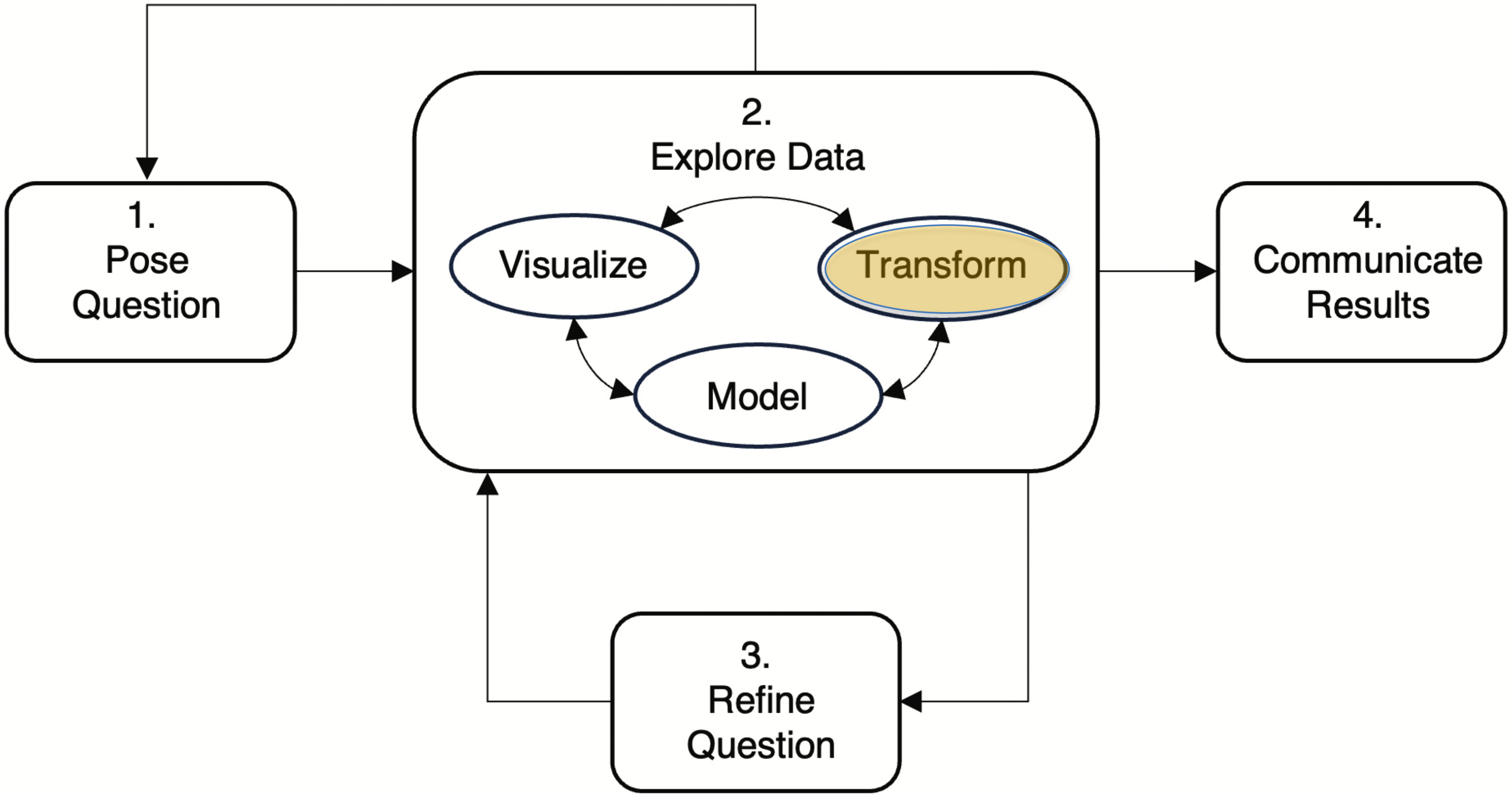
It's rare that a data analysis involves only a single table of data. Typically you have many tables of data, and you must combine them to answer the questions you are interested in.

— Hadley Wickham and Garrett Grolemund (Wickham and Grolemund, 2016)

Overview

- Relational data - overview
- Mutating joins
- Filtering joins
- Case Study
- Challenge: nycflights13

1. Overview – relational data



Leveraging data relationships

- When exploring data, there may be more than one tibble of interest, and these tibbles may share common information
- For example, we may want to explore associations between:
  - flight delays and weather events, and these information sources may be contained in two different tibbles.
  - weather data and wind energy generated, in order to build energy prediction models.

Relational concepts

- A relational model organizes data into one or more tables of columns and rows, where a unique key identifies each row.
- A key can be a single column value, known as a primary key, for example, a unique identifier for a person.
- For example, for the tibble aimsir17::stations, the primary key is the variable station, as this value is unique for each row, and therefore can be used to identify a single observation.
- A primary key from one table can also be a column in another table, and if this is the case, it is termed a foreign key in that table.

Order contains one product, product can be part of zero or more orders

Primary key

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

Primary key *Foreign key*

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

Some observations

1. To simplify the example, we limit the number of product types for each order to one. Therefore, we have what is called a one-to-many relationship, where an order is for one product, but a product can be part of many orders.
2. There is an in-built inconsistency in the data, specifically, one of the orders (“OR-4”) contains a product (“PR-4”) that is not present in the product table. Normally, in a real-world database this would not be allowed, and through a process known as referential integrity, the database management system would enforce a constraint that a foreign key in one table would have to already be defined as a primary key in another table.
3. We use (2) to show a range of joins available in dplyr.

Creating the data - products

```
products <- tibble(ProductID=c("PR-1","PR-2","PR-3","PR-9"),
                   Type=c("Computer","Tablet","Phone","Headphones"))
products
#> # A tibble: 4 x 2
#>   ProductID Type
#>   <chr>     <chr>
#> 1 PR-1      Computer
#> 2 PR-2      Tablet
#> 3 PR-3      Phone
#> 4 PR-9      Headphones
```

Creating the data - orders

```
orders <- tibble(OrderID=c("OR-1","OR-2","OR-3","OR-4","OR-5"),
                 Quantity=c(1,2,1,2,3),
                 ProductID=c("PR-1","PR-2","PR-3","PR-4","PR-1"))
orders
#> # A tibble: 5 x 3
#>   OrderID Quantity ProductID
#>   <chr>      <dbl> <chr>
#> 1 OR-1           1 PR-1
#> 2 OR-2           2 PR-2
#> 3 OR-3           1 PR-3
#> 4 OR-4           2 PR-4
#> 5 OR-5           3 PR-1
```

2. Mutating joins

- The first category of joins we explore is known as a mutating join, as these result in a new tibble with additional columns.
- Here are dplyr functions that perform mutating joins:
  - inner_join(),
  - left_join(),
  - full_join().

inner_join(x,y)

- The inner_join() function joins observations that appear in both tables, based on a common key, which need to be present in both tables. It takes the following arguments, and returns an object of the same type as x.
  - x and y, a pair of tibbles or data frames to be joined.
  - by, which is a character vector of variables to join by. In cases where the key column name is different, a named vector can be used, for example, by = c("key_x" = "key_y").

Primary key

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

Primary key · Foreign key

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

```
i_j <- dplyr::inner_join(orders,products,by="ProductID")
i_j
#> # A tibble: 4 x 4
#>   OrderID Quantity ProductID Type
#>   <chr>      <dbl> <chr>     <chr>
#> 1 OR-1           1 PR-1      Computer
#> 2 OR-2           2 PR-2      Tablet
#> 3 OR-3           1 PR-3      Phone
#> 4 OR-5           3 PR-1      Computer
```

left_join(x,y)

- A left join will keep all observations in the tibble x, even if there is no match in tibble y.
- This is a widely used function, given that it maintains all the observations in x.
- We can now show two examples based on the tibbles orders and products.

Primary key

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

Primary key / *Foreign key*

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

```
l_j1 <- dplyr::left_join(orders,products,by="ProductID")
l_j1
#> # A tibble: 5 x 4
#>   OrderID Quantity ProductID Type
#>   <chr>      <dbl> <chr>     <chr>
#> 1 OR-1           1 PR-1      Computer
#> 2 OR-2           2 PR-2      Tablet
#> 3 OR-3           1 PR-3      Phone
#> 4 OR-4           2 PR-4      <NA>
#> 5 OR-5           3 PR-1      Computer
```

Primary key ←→

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

Primary key ←→ *Foreign key* ←- - →

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

```
l_j3 <- dplyr::left_join(products,orders,by="ProductID")
l_j3
#> # A tibble: 5 x 4
#>   ProductID Type       OrderID Quantity
#>   <chr>     <chr>      <chr>      <dbl>
#> 1 PR-1      Computer   OR-1           1
#> 2 PR-1      Computer   OR-5           3
#> 3 PR-2      Tablet     OR-2           2
#> 4 PR-3      Phone      OR-3           1
#> 5 PR-9      Headphones <NA>          NA
```

Primary key

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

Primary key | Foreign key

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

```
l_j2 <- dplyr::left_join(orders,products,by="ProductID",keep=TRUE)
l_j2
#> # A tibble: 5 x 5
#>   OrderID Quantity ProductID.x ProductID.y Type
#>   <chr>      <dbl> <chr>       <chr>       <chr>
#> 1 OR-1           1 PR-1        PR-1        Computer
#> 2 OR-2           2 PR-2        PR-2        Tablet
#> 3 OR-3           1 PR-3        PR-3        Phone
#> 4 OR-4           2 PR-4        <NA>        <NA>
#> 5 OR-5           3 PR-1        PR-1        Computer
```

full_join()

A full join keeps all observations in both x and y.

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

Primary key: ProductID

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

Primary key: OrderID; Foreign key: ProductID

```
f_j1 <- dplyr::full_join(products,orders,by="ProductID")
f_j1
#> # A tibble: 6 x 4
#>   ProductID Type       OrderID Quantity
#>   <chr>     <chr>      <chr>      <dbl>
#> 1 PR-1      Computer   OR-1           1
#> 2 PR-1      Computer   OR-5           3
#> 3 PR-2      Tablet     OR-2           2
#> 4 PR-3      Phone      OR-3           1
#> 5 PR-9      Headphones <NA>          NA
#> 6 PR-4      <NA>       OR-4           2
```

3. Filtering joins

- There are another set of joins that can be used to filter observations in tibble x, based on their relationship with values in another table.
- As the name suggests, these are filtering joins, and perform a similar task to the regular filter() function, except that information from two tables is used.

semi_join(x,y)

- This function will keep all the observations in x that have a matching column in y.

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

```
s_j1 <- dplyr::semi_join(orders,products,by="ProductID")
s_j1
#> # A tibble: 4 x 3
#>   OrderID Quantity ProductID
#>   <chr>      <dbl> <chr>
#> 1 OR-1           1 PR-1
#> 2 OR-2           2 PR-2
#> 3 OR-3           1 PR-3
#> 4 OR-5           3 PR-1
```

anti_join(x,y)

- This filtering function will keep all the observations in x that do not have a matching column in y.

ProductID	Type
PR-1	Computer
PR-2	Tablet
PR-3	Phone
PR-9	Headphones

OrderID	Quantity	ProductID
OR-1	1	PR-1
OR-2	2	PR-2
OR-3	1	PR-3
OR-4	2	PR-4
OR-5	3	PR-1

```
a_j1 <- dplyr::anti_join(orders,products,by="ProductID")
a_j1
#> # A tibble: 1 x 3
#>   OrderID Quantity ProductID
#>   <chr>      <dbl> <chr>
#> 1 OR-4           2 PR-4
```

Mini-case: joining weather and energy data

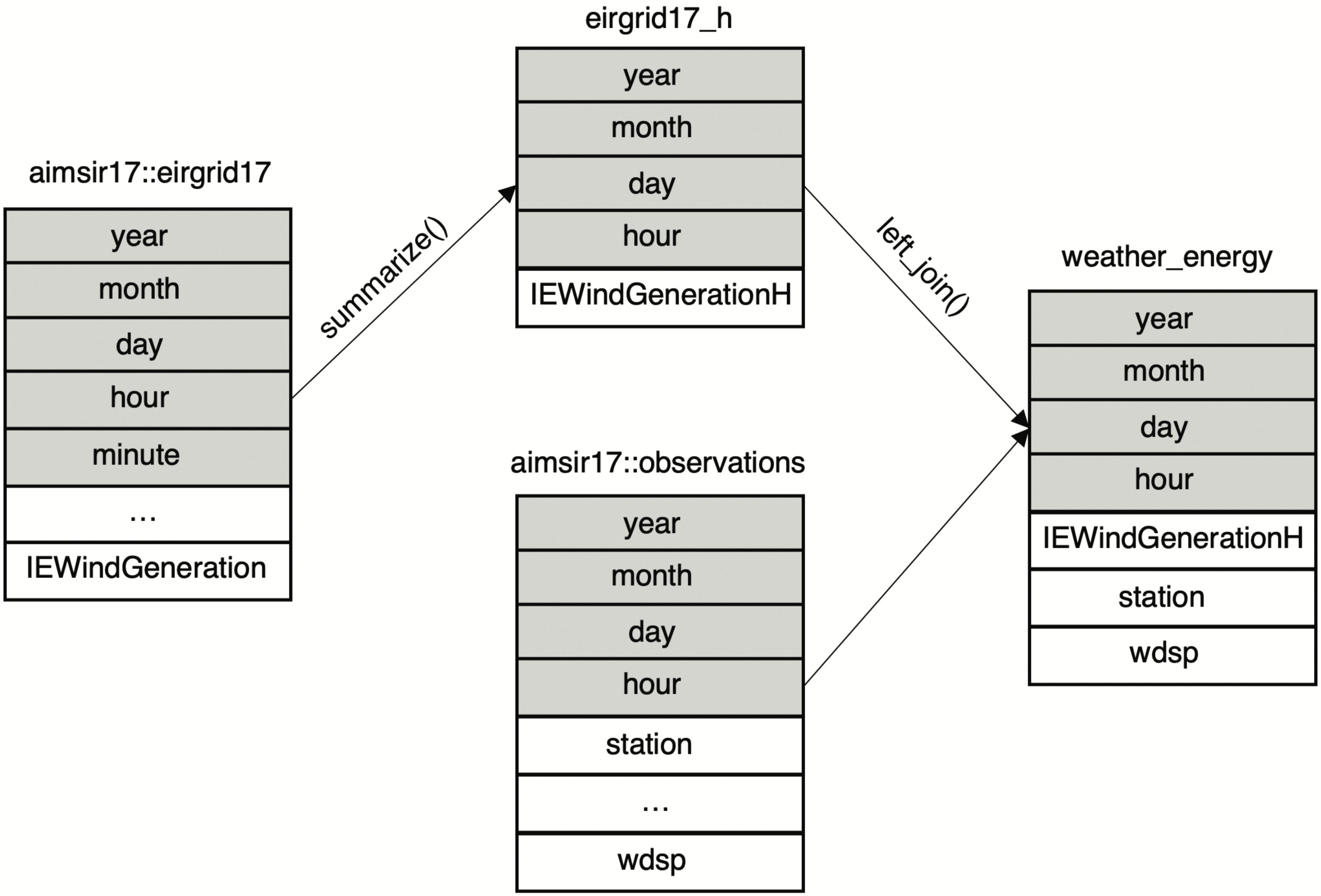
- The aim of this example is to show how to combine related tibbles using dplyr, in order to explore new relationships.
- We use existing data from the package aimsir17, and in particular two tibbles:
  - observations, which contains hourly information from 2017 for 25 weather stations across Ireland, including the rainfall, temperature, humidity, mean sea level atmospheric pressure, wind speed, and wind direction.
  - eirgrid17, which holds information, recorded four times per hour, on energy demand and supply from the Irish Grid in 2017, including overall demand, energy generated, and wind energy generated.

eirgrid17

year	month	day	hour	minute	date	NIGeneration	NIDemand	NIWindAvailability	NIWindGeneration	IEGeneration	IEDemand	IEWindAvailability	IEWindGeneration
2017	1	1	0	0	2017-01-01 00:00:00	889.005	775.931	175.065	198.202	3288.57	2921.44	1064.79	1044.72
2017	1	1	0	15	2017-01-01 00:15:00	922.234	770.233	182.866	207.765	3282.12	2884.19	965.60	957.74
2017	1	1	0	30	2017-01-01 00:30:00	908.122	761.186	169.796	193.103	3224.27	2806.38	915.35	900.46
2017	1	1	0	45	2017-01-01 00:45:00	918.802	742.718	167.501	190.757	3171.27	2718.77	895.38	870.81
2017	1	1	1	0	2017-01-01 01:00:00	882.441	749.238	174.094	195.790	3190.28	2682.91	1028.03	998.31
2017	1	1	1	15	2017-01-01 01:15:00	848.863	742.455	189.922	212.956	3184.67	2649.87	1144.17	1119.12
2017	1	1	1	30	2017-01-01 01:30:00	842.778	726.472	222.139	244.569	3100.66	2578.31	1080.40	1056.76
2017	1	1	1	45	2017-01-01 01:45:00	808.910	709.353	233.321	245.045	3125.78	2555.87	1184.10	1165.61
2017	1	1	2	0	2017-01-01 02:00:00	797.183	697.191	281.727	298.625	3106.37	2498.73	1300.97	1275.92
2017	1	1	2	15	2017-01-01 02:15:00	754.976	684.101	259.120	285.191	3078.77	2441.95	1362.96	1337.95
2017	1	1	2	30	2017-01-01 02:30:00	795.311	668.003	272.766	306.300	2984.55	2400.41	1293.22	1266.24
2017	1	1	2	45	2017-01-01 02:45:00	780.488	651.126	261.728	290.701	2947.35	2365.22	1283.57	1257.01
2017	1	1	3	0	2017-01-01 03:00:00	777.013	640.060	286.211	308.586	2919.70	2358.54	1338.51	1314.98
2017	1	1	3	15	2017-01-01 03:15:00	780.512	628.341	266.933	293.743	2850.68	2306.28	1278.03	1253.87

observations

station	year	month	day	hour	date	rain	temp	rhum	msl	wdsp	wddir
ATHENRY	2017	1	1	0	2017-01-01 00:00:00	0.0	5.2	89	1021.9	8	320
ATHENRY	2017	1	1	1	2017-01-01 01:00:00	0.0	4.7	89	1022.0	9	320
ATHENRY	2017	1	1	2	2017-01-01 02:00:00	0.0	4.2	90	1022.1	8	320
ATHENRY	2017	1	1	3	2017-01-01 03:00:00	0.1	3.5	87	1022.5	9	330
ATHENRY	2017	1	1	4	2017-01-01 04:00:00	0.1	3.2	89	1022.7	8	330
ATHENRY	2017	1	1	5	2017-01-01 05:00:00	0.0	2.1	91	1023.3	8	330
ATHENRY	2017	1	1	6	2017-01-01 06:00:00	0.0	2.0	89	1023.5	7	330
ATHENRY	2017	1	1	7	2017-01-01 07:00:00	0.0	1.7	89	1024.4	7	340
ATHENRY	2017	1	1	8	2017-01-01 08:00:00	0.0	1.0	91	1025.0	7	330
ATHENRY	2017	1	1	9	2017-01-01 09:00:00	0.0	1.1	91	1026.1	8	330
ATHENRY	2017	1	1	10	2017-01-01 10:00:00	0.0	3.0	84	1026.8	9	320
ATHENRY	2017	1	1	11	2017-01-01 11:00:00	0.0	4.3	78	1027.0	12	350
ATHENRY	2017	1	1	12	2017-01-01 12:00:00	0.0	5.1	75	1027.4	11	360
ATHENRY	2017	1	1	13	2017-01-01 13:00:00	0.0	5.5	72	1027.4	12	360
ATHENRY	2017	1	1	14	2017-01-01 14:00:00	0.0	5.9	72	1027.6	11	360


eirgrid17_h
year
month
day
hour
IEWindGenerationH
aimsir17::eirgrid17
year
month
day
hour
minute
…
IEWindGeneration
summarize()
left_join()
weather_energy
year
month
day
hour
IEWindGenerationH
station
wdsp
aimsir17::observations
year
month
day
hour
station
…
wdsp

Create hourly summaries…

```
eirgrid17_h <- eirgrid17 %>%
                  dplyr::group_by(year,month,day,hour) %>%
                  dplyr::summarize(IEWindGenerationH=
                                    mean(IEWindGeneration,
                                         na.rm=T)) %>%
                  dplyr::ungroup()
dplyr::slice(eirgrid17_h,1:4)
#> # A tibble: 4 x 5
#>    year month   day  hour IEWindGenerationH
#>   <dbl> <dbl> <int> <int>             <dbl>
#> 1  2017     1     1     0              943.
#> 2  2017     1     1     1             1085.
#> 3  2017     1     1     2             1284.
#> 4  2017     1     1     3             1254.
```

Use left_join(x,y)

```
obs1 <- observations %>%
         dplyr::filter(!is.na(wdsp))

weather_energy <- dplyr::left_join(eirgrid17_h,
                                    obs1,
                                    by=c("year",
                                         "month",
                                         "day",
                                         "hour")) %>%
                  dplyr::select(year,month,day,hour,
                                IEWindGenerationH,
                                station,
                                wdsp)
```

Joined data

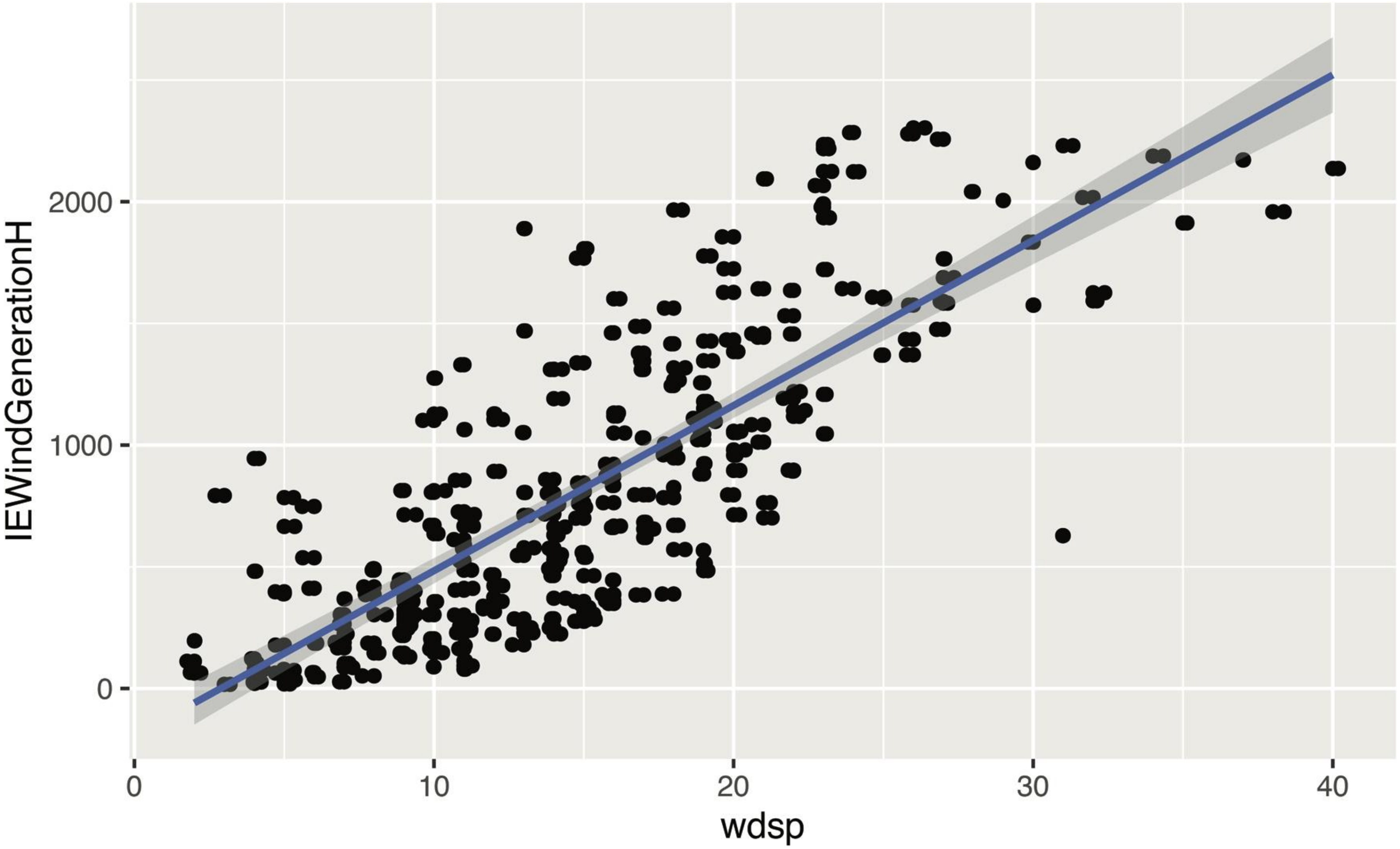
```
weather_energy
#> # A tibble: 201,430 x 7
#>      year month   day  hour IEWindGenerationH station        wdsp
#>     <dbl> <dbl> <int> <int>             <dbl> <chr>         <dbl>
#>  1  2017     1     1     0              943. ATHENRY           8
#>  2  2017     1     1     0              943. BALLYHAISE        5
#>  3  2017     1     1     0              943. BELMULLET        13
#>  4  2017     1     1     0              943. CASEMENT          8
#>  5  2017     1     1     0              943. CLAREMORRIS       8
#>  6  2017     1     1     0              943. CORK AIRPORT     11
#>  7  2017     1     1     0              943. DUBLIN AIRPORT   12
#>  8  2017     1     1     0              943. DUNSANY           6
#>  9  2017     1     1     0              943. FINNER           12
#> 10  2017     1     1     0              943. GURTEEN           7
#> # ... with 201,420 more rows
```

Extract sample data

```
set.seed(100)
obs_sample <- weather_energy %>%
               dplyr::filter(station %in% c("MACE HEAD")) %>%
               dplyr::sample_n(300)

ggplot(obs_sample,aes(x=wdsp,y=IEWindGenerationH))+
  geom_point()+
  geom_jitter()+
  geom_smooth(method="lm")
```

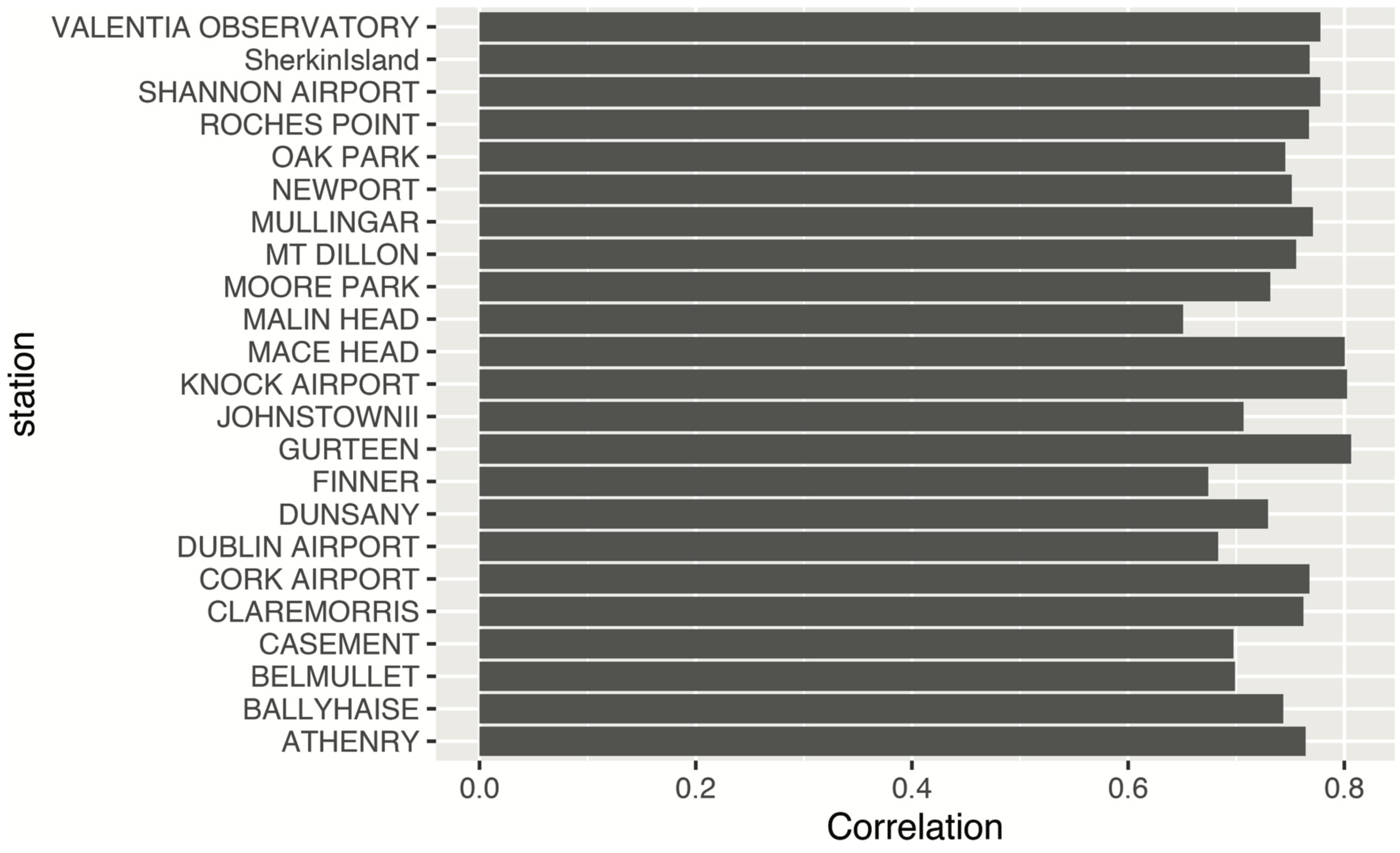
Visualise Plot



Comparing Stations

```
corr_sum <- weather_energy %>%
            dplyr::group_by(station) %>%
            dplyr::summarize(Correlation=cor(wdsp,
                                             IEWindGenerationH)) %>%
            dplyr::arrange(desc(Correlation))
corr_sum
#> # A tibble: 23 x 2
#>    station                Correlation
#>    <chr>                        <dbl>
#>  1 GURTEEN                      0.806
#>  2 KNOCK AIRPORT                0.802
#>  3 MACE HEAD                    0.800
#>  4 VALENTIA OBSERVATORY         0.778
#>  5 SHANNON AIRPORT              0.778
#>  6 MULLINGAR                    0.771
#>  7 SherkinIsland                0.768
#>  8 CORK AIRPORT                 0.768
#>  9 ROCHES POINT                 0.767
#> 10 ATHENRY                      0.764
#> # ... with 13 more rows
```

Showing correlations



Challenge

1. Based on the package `nycflights13`, which can be downloaded from CRAN, generate the following tibble based on the first three records from the tibble `flights`, and the airline name from `airlines`.

```
first_3a
#> # A tibble: 3 x 5
#>   time_hour           origin dest  carrier name
#>   <dttm>              <chr>  <chr> <chr>   <chr>
#> 1 2013-01-01 05:00:00 EWR    IAH   UA      United Air Lines Inc.
#> 2 2013-01-01 05:00:00 LGA    IAH   UA      United Air Lines Inc.
#> 3 2013-01-01 05:00:00 JFK    MIA   AA      American Airlines Inc.
```