Station Monthly Temperature-based Needs

Ad-Hoc Assignment Technical Report

Robert Duc Bui

2022-06-27

Prerequisites

To gather and process the data necessary, we will be using the following R packages:

- `tidyverse`: meta-package containing many subpackages containing useful data tools. Specifically we will be using `dplyr` for joins and other cleaning steps, and `stringr` for string manipulation.
- `airportr`: open-source R package to retrieve airport information.
- `rnoaa`: open-source R package to retrieve NOAA metadata.

```
library(tidyverse)
library(stringr)
library(airportr)
library(rnoaa)
```

Data Sources

We will be using the following data sources:

- Internal list of line stations where data is needed. 58 lines of IATA airport codes.
- Airport coordinates (latitude/longitude), ICAO code conversion: `airportr` from CRAN/github. Data sourced from OpenFlights Airport Database made available under the Open Database License.
- NOAA weather stations participating in Global Historical Climatology Network daily reporting (GHCN-d) metadata: `ropensci/rnoaa` from CRAN/github. Data sourced from the NOAA NCEI's open API.
- Monthly normal temperature data by station, recorded from 2006-2020: compressed archive of .csv files directly downloaded from NOAA page for U.S. Climate Normals.

Methodology

Step 1: Importing data

The following script imports our data from aforementioned sources.

- Internal airport list: reading in `input_vector.csv`, extracted from internal staffing documents.
- Airport metadata: `airportr::airports()` function call to retrieve data frame of all airports from database. Filtered for US-based airports only. Selecting for IATA code, ICAO code, Latitude, and Longitude.
- NOAA metadata: `rnoaa::ghcnd_stations()` function call to retrieve data frame of all NOAA weather stations participating in GHCN reporting program for US climate normals.
- Monthly normal temps from 2006-2020: reading in `mly-normal-allall.csv`, downloaded from NOAA site.

```
# Input vector - pasted from staffing spreadsheet
input_vector <- read_csv("input_vector.csv")

# Parsing airport data to get latitude and longitude
df_inputairports <- airports %>%
  filter(`Country Code (Alpha-2)` == "US") %>%
  right_join(input_vector, by = c("IATA" = "iata_code")) %>%
  select(IATA,ICAO,
         airport_lat = Latitude,
         airport_lon = Longitude) %>%
  arrange(IATA)

# rnoaa package helper function - downloads metadata about all GHCN stations from NOAA API
df_ghcnd <- ghcnd_stations()

# Reading raw temperature data file for all recorded stations from NOAA
mly_data <- read_csv("data_noaa/by_variable/mly-normal-allall.csv") %>%
  select(GHCN_ID, month, `MLY-TMIN-NORMAL`) %>%
  filter(str_starts(GHCN_ID, "US"))
```

Step 2: Station data cleaning and joining

Next, using the monthly normals dataset station ID as primary keys, we perform a SQL join with the station metadata, only keeping stations with monthly normal temp data. From this joined table we keep the following variables:

- NOAA internal reporting ID
- Station coordinates (longitude/latitude)
- Month (1-12)
- Monthly lower normalized temperature

```
# Parsing NOAA station metadata for existing dataset only
df_allstations <- df_ghcnd %>%
  filter(element == "TMIN") %>%
  right_join(mly_data, by = c("id"="GHCN_ID")) %>%
  select(
    id,
    station_lat = latitude,
    station_lon = longitude,
    month,
    monthly_norm_min = `MLY-TMIN-NORMAL`
  )
```

Step 3: Finding nearest weather station to each airport

To determine which weather station is closest to each airport, we will be calculating the distance between each airport and station by their latitudes and longitudes. We could use either Haversine (half-versine) distance or Euclidean distance - here, to account for the curvature of the Earth, we will be using Haversine distance.

The Haversine distance formula, given r as the radius of earth, d as the distance between two points, ϕ_1, ϕ_2 as latitudes, and λ_1, λ_2 as longitudes, is as follows:

$$haversin(\frac{d}{r}) = haversin(\phi_2 - \phi_1) + cos(\phi_1)cos(\phi_2)haversin(\lambda_2 - \lambda_1)$$

Otherwise, to find the distance d, we can perform the following calculation:

$$a = sin^2(\frac{\phi_2 - \phi_1}{2}) + cos(\phi_1)cos(\phi_2)sin^2(\frac{\lambda_2 - \lambda_1}{2})$$

$$d = r.(2.atan2(\sqrt{a}, \sqrt{1-a}))$$

We define a custom function to do so as follows:

```
# Custom function to derive haversine distance between lat-lon points
haversine <- function(lat1,lon1,lat2,lon2){
  # Converting lat/lon pairs from degrees to radian
  lat1 <- lat1*pi/180
  lon1 <- lon1*pi/180
  lat2 <- lat2*pi/180
  lon2 <- lon2*pi/180

  dlon <- lon2 - lon1
  dlat <- lat2 - lat1
  a <- (sin(dlat/2))^2 + cos(lat1) * cos(lat2) * (sin(dlon/2))^2
  a <- pmin(a, 1)
  c <- 2 * atan2(sqrt(a), sqrt(1-a))
  d <- 3961 * c
  return(d)
}
```

Then we use `expand.grid()` to create a matrix of all possible combinations of one IATA airport code and one NOAA weather station. Left-joining twice with airport metadata and station metadata containing latitudes and longitudes, we can now call our previously defined `haversine()` function on the coordinates, and derive the pairwise distances.

```
# Find nearest NOAA station by haversine distance (assuming Earth radius of 3961 miles)
df_neareststations <- expand.grid(df_inputairports$IATA,
            df_allstations$id %>% unique()) %>%
  rename(IATA = Var1,
         id = Var2) %>%
  left_join(df_inputairports) %>%
  left_join(df_allstations %>% select(id,station_lat,station_lon) %>% unique()) %>%
  mutate(
    # haversine distance in miles
```

```
    haversine = haversine(airport_lon,airport_lat,station_lon,station_lat)
  ) %>%
  group_by(IATA) %>%
  slice_min(order_by = haversine)
```

We can then perform one final filter to keep only the smallest distance pairs for each IATA code. Note that distance here is measured in miles. The 58-row table is presented below, sorted by furthest to nearest distance.

Most airports appear to have a weather station less than 5 miles away, with two exceptions being EGE (Eagle County Regional) and HOB (Lea County Regional), both being smaller stations unlikely to have a dedicated NOAA station on-site. We can possibly consider this distance to be satisfactory for our calculation purposes.

```
# Distance calculation results, ranked by furthest to nearest
df_neareststations %>%
  select(IATA,NOAA_GHCN_id = id,haversine) %>%
  arrange(desc(haversine))
```

```
## # A tibble: 58 x 3
## # Groups:   IATA [58]
##    IATA  NOAA_GHCN_id haversine
##    <chr> <chr>            <dbl>
##  1 EGE   USW00093073      9.10
##  2 HOB   USC00294026      5.95
##  3 HHH   USW00093831      3.18
##  4 PQI   USC00176937      2.64
##  5 HDN   USC00053867      2.55
##  6 JAC   USC00486428      1.85
##  7 SCE   USC00368449      1.52
##  8 GRB   USC00473268      1.22
##  9 SHV   USW00013957      1.07
## 10 MAF   USW00023023      0.906
## # ... with 48 more rows
```

Step 4: Calculating Deicing/Heatwatch Months

We now perform a left-join between the table derived in the previous step and the full temperature dataset, only keeping the normalized lower temps. We end up with a 696-row table of 58 airports times 12 months of lower temp data.

Using 35 degrees Fahrenheit as our upper bound for temperature, we can use a simple `ifelse()` call to determine whether each month at each station would require deicing staffing.

```
df_deice <- df_neareststations %>%
  select(IATA,id) %>%
  left_join(df_allstations) %>%
  select(IATA,id,month,monthly_norm_min) %>%
  mutate(month = month %>% as.numeric()) %>%
  mutate(deicing_needed = ifelse(monthly_norm_min <= 35,1,0)) %>%
  select(IATA,month,deicing_needed)

df_deice
```

```
## # A tibble: 696 x 3
## # Groups:   IATA [58]
##    IATA  month deicing_needed
##    <chr> <dbl>          <dbl>
##  1 ABE       1              1
##  2 ABE       2              1
##  3 ABE       3              1
##  4 ABE       4              0
##  5 ABE       5              0
##  6 ABE       6              0
##  7 ABE       7              0
##  8 ABE       8              0
##  9 ABE       9              0
## 10 ABE      10              0
## # ... with 686 more rows
```

We cannot simply take the min or max of months that deicing is needed to determine the starting and ending months, since winter in the Northern hemisphere carries from one year to the other. Instead, we shift the month variable so that months would count from the beginning of summer solstice instead (June in the Northern hemisphere) by using the modulo operator. In essence, we are shifting the year up six months. We call this variable `solstice_month`, for months since the previous summer solstice.

$$solstice\_month = ((month - 6) \mod 12) + 1$$

After taking the min and max of `solstice_month` for each airport, we convert this back to the nominal month by reversing the modulo operation. The formula for this conversion is below:

$$month = ((solstice\_month + 4) \mod 12) + 1$$

```
# Solstice month conversion + parsing min/max to find begin and end month for deicing
df_deice <- df_deice %>%
  mutate(solstice_month = ((month-6)%%12)+1) %>%
  group_by(IATA,deicing_needed) %>%
```

```
  summarise(
    deice_begin = min(solstice_month),
    deice_end = max(solstice_month)
  ) %>%
  filter(deicing_needed == 1) %>%
  mutate(
    deice_begin = ((deice_begin+4)%%12)+1,
    deice_end   = ((deice_end  +4)%%12)+1
  )
```

Finally, to account for airports where deicing is not needed, i.e. station temperature does not dip below 35F at any month, we perform a left join between the input list of airports and the derived tables above. Airports with no de-icing need will have NA values for begin and end months.

```
# Output table
output_table <- input_vector %>%
  rename(IATA = iata_code) %>%
  left_join(df_deice) %>%
  select(IATA,deice_begin,deice_end)
```

Step 5: Export & Display

We export the results to a .csv file, from which we can copy the results into our internal staffing spreadsheet.

```
# Save output to .csv
write_csv(output_table,
          file = "output_table.csv")
```

We can also print the results of the table here:

```
library(kableExtra)
output_table %>%
  kable(format = 'latex',
        booktabs = T,
        longtable = T) %>%
  kable_styling(position = "center") %>%
  kable_styling(latex_options = c("repeat_header"))
```

IATA	deice_begin	deice_end
ABE	11	3
ABQ	11	2
AVL	12	2
AVP	11	3
BHM	1	1
BIH	11	3
BNA	12	2
BTV	11	3
BUF	11	3
CHA	1	1
CRP	NA	NA
DAY	11	3
EGE	10	5
ERI	12	3
EYW	NA	NA
FAR	11	4
FLG	10	5
FSD	11	3
GRB	11	4
GSP	1	2
HDN	10	4
HHH	NA	NA
HOB	12	2
ILM	NA	NA
ITO	NA	NA
JAC	10	5
JAN	NA	NA
LEX	12	2
LFT	NA	NA
LNK	11	3

(continued)

IATA	deice_begin	deice_end
MAF	12	1
MDT	12	3
MEM	11	3
MFR	12	2
MKE	11	3
MRY	NA	NA
MTJ	11	3
ONT	NA	NA
ORF	1	1
PIA	11	3
PQI	11	4
PSP	NA	NA
PVD	12	3
PWM	11	3
RIC	12	2
ROC	11	3
SAF	11	4
SAV	NA	NA
SBA	NA	NA
SBN	11	3
SCE	11	3
SDF	12	2
SHV	NA	NA
SRQ	NA	NA
STS	NA	NA
SYR	11	3
TYS	12	2
XNA	12	2

References

- Chamberlain, Scott, and Daniel Hocking. "NOAA Weather Data from R." Accessed June 27, 2022. https://docs.ropensci.org/rnoaa/.
- ———. "Ropensci/Rnoaa: R Interface to Many NOAA Data APIs." Accessed June 27, 2022. https://github.com/ropensci/rnoaa. Chamberlain, Scott, Daniel Hocking, Brooke Anderson, Maëlle Salmon, Adam Erickson, Nicholas Potter, Joseph Stachelek, et al. Rnoaa: "NOAA" Weather Data from R (version 1.3.8), 2021. https://CRAN.R-project.org/package=rnoaa.
- NOAA. "U.S. Climate Normals." National Centers for Environmental Information (NCEI), February 25, 2021. http://www.ncei.noaa.gov/products/land-based-station/us-climate-normals.
- Open Knowledge Foundation. "Open Data Commons Open Database License (ODbL) v1.0 — Open Data Commons: Legal Tools for Open Data." Accessed June 27, 2022. https://opendatacommons.org/licenses/odbl/1-0/.
- Patokallio, Jani. "OpenFlights: Airport and Airline Data." Accessed June 27, 2022. https://openflights.org/data.html.

- Shkolnik, Dmitry. Airportr: Convenience Tools for Working with Airport Data (version 0.1.3), 2019. https://CRAN.R-project.org/package=airportr.
- ———. "Convenience Tools for Working with Airport Data." Accessed June 27, 2022. https://dshkol.github.io/airportr/index.html.