Introduction to Data Acquisition using Tidyverse

Estimated Time Needed: **30 min**

Welcome!

In this section, you will learn how to approach data acquisition in various ways, and obtain necessary information about the dataset. By the end of this lab, you will successfully load the data into Jupyter Notebook, and gain some fundamental insights via the Tidyverse library.

Table of Contents:

1. Data Acquisition

There are various formats for a dataset like .csv, .json, .xlsx, etc. The dataset can be stored in different places, on your local machine or sometimes online. In this section, you will learn how to load a dataset into our Jupyter Notebook.

In our case, the Airline Dataset is an online source, and it is in CSV (comma separated value) format. Let's use a smaller subset of the original dataset as an example to practice data reading.

- data source: https://dax-cdn.cdn.appdomain.cloud/dax-airline/1.0.1/lax_to_jfk.tar.gz
- data type: csv

The Tidyverse library is a useful tool that enables us to read various datasets into a data frame; our Jupyter notebook platforms have a built-in Tidyverse library so that all we need to do is import Tidyverse without installing.

However, if you decide to run this on your local machine, you can use the below line of code to install Tidyverse before loading.

[ ]:
```
# Uncomment to install tidyverse if running locally
# install.packages("tidyverse")
```

[ ]:
```
# Load tidyverse
library(tidyverse)
```

The Tidyverse library has a suite of packages that all work together well, namely:

- tidyr: helps create tidy data
- dplyr: manipulates and transforms data
- readr: easily read different types of datasets
- purrr: functional programming toolkit
- ggplot2: create plots and visualizations

In this notebook, we will mainly use functions from readr and dplyr. However, don't worry about memorizing which function is from which library, just know that all are included in Tidyverse.

Read Data

We use the function `readr::read_csv()` to read csv files. The `::` tells you which package the function is from, so `read_csv()` is from library `readr` (which is automatically loaded with Tidyverse). There are a few parts to the function to go over:

- `file` (or the first parameter): this is file path along with quotation marks, so that `read_csv()` will read the file into a data frame from that address. The file path can be either an URL or your local file address.
- `col_names` : by deafult this is set to `col_names = TRUE` . If this is TRUE then the first row is set as the headers (which is correct in this dataset).
- `col_types` : used to specify what types columns are. By default, `read_csv()` will guess the type of the columns but if there are columns you want to specify you can do so as well.
  - Posible types you could use are `col_logical()` , `col_integer()` , `col_number()` , `col_character()` . You can look at the documentation of `readr::cols` for more.

You can assign the loaded dataset to any variable name, here it is `sub_airline` .

[ ]:
```
# url where the data is located
url <- "https://dax-cdn.cdn.appdomain.cloud/dax-airline/1.0.1/lax_to_jfk.tar.gz"
# download the file
download.file(url, destfile = "lax_to_jfk.tar.gz")
# untar the file so we can get the csv only
untar("lax_to_jfk.tar.gz", tar = "internal")
# read_csv only
sub_airline <- read_csv("lax_to_jfk/lax_to_jfk.csv",
                    col_types = cols(
                     'DivDistance' = col_number(),
                     'DivArrDelay' = col_number()
                     ))
```

After reading the dataset, there are a few functions you can use to get some initial information about the dataframe:

- `head(dataframe, n)` : returns the first *n* rows of the dataframe, if *n* is not specified, then by default the first 6 rows (not including the column headers) are returned
- `tail(dataframe, n)` : returns the last *n* rows of the dataframe, if *n* is not specified, then by default the last 6 rows are returned.

- `colnames(dataframe)` : retrieves the column names of the dataframe
- `dim(dataframe)` : retrieves the dimension (number of rows and columns) of the dataframe

[ ]:
```
# show the first n = 3 rows
head(sub_airline, 3)
```

Or, you can choose to not input `n`, then it will show the first or the last 6 rows as default.

[ ]:
```
# show the first 6 rows
head(sub_airline)
```

[ ]:
```
# show the last 6 rows
tail(sub_airline)
```

Question #1:

Check the last 10 rows of data frame "sub_airline".

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
tail(sub_airline, 10)
```

Preprocess Data

Throughout this course, we are going to be focusing on flights from LAX to JFK and predict the possibility of Amy's flight delays. So we also exclude flights that got cancelled or diverted. The previous example uses the final subset dataset `sub_airline`. However, there was some preprocessing done on the original dataset to get there. You can find the full dataset here: https://developer.ibm.com/technologies/artificial-intelligence/data/airline/.

From this original dataset, we will show you how we got the `sub_airline` dataset. After downloading the original data with `download.file()` and reading the file with `read_csv()`:

1. Firstly, we use `dplyr::filter()` to filter out all flights from `LAX` to `JFK`, and filter out the cancelled and diverted flights.
2. Then, we use `dplyr::select()` to select the columns by a pre-determined headers list. The original dataset contains more than 100 columns, so we are only going to use a few of the columns such as `Month`, `DaysOfWeek`, `FlightDate`, `Reporting_Airline`, and so on.

The below code is to show you how to subset a big dataset. The code is here for your reference. Feel free to play around with it and modify the columns. Remember **not** to run the below code in Skills Network Labs. The data size too big and it is better run to run locally so that Skills Network Labs does not freeze or lag.

```
# THIS CELL TAKES LONGER TO RUN
# PLEASE RUN THIS LOCALLY, OR IT MAY FREEZE YOUR SKILLS NETWORK LABS KERNEL
# Download 2 million dataset

# url where the data is located
url <- "https://dax-cdn.cdn.appdomain.cloud/dax-airline/1.0.1/airline_2m.tar.gz"

# download the file
download.file(url, destfile = "airline_2m.tar.gz")

# untar the file so we can get the csv only
untar("airline_2m.tar.gz")

# read_csv only
airlines <- read_csv("airline_2m.csv",
                     col_types = cols(
                      'DivDistance' = col_number(),
                      'Div1Airport' = col_character(),
                      'Div1AirportID' = col_character(),
                      'Div1AirportSeqID' = col_character(),
                      'Div1WheelsOn' = col_character(),
                      'Div1TotalGTime' = col_number(),
                      'Div1LongestGTime' = col_number(),
                      'DivReachedDest' = col_number(),
                      'DivActualElapsedTime' = col_number(),
                      'DivArrDelay' = col_number(),
                      'Div1WheelsOff'= col_character(),
                      'Div1TailNum' = col_character(),
                      'Div2Airport' = col_character(),
                      'Div2AirportID' = col_character(),
                      'Div2AirportSeqID' = col_character(),
                      'Div2WheelsOn' = col_character(),
                      'Div2TotalGTime' = col_number(),
                      'Div2LongestGTime' = col_number(),
                      'Div2WheelsOff' = col_character(),
                      'Div2TailNum' = col_character()
                      ))
# We are going to be focusing on flights from  LAX to JFK and we will exclude
# flights that got cancelled or diverted
# we are also going to get only useful columns
sub_airline <- airlines %>%
  filter(Origin == "LAX", Dest == "JFK",
         Cancelled != 1, Diverted != 1) %>%
  select(Month, DayOfWeek, FlightDate,
         Reporting_Airline, Origin, Dest,
         CRSDepTime, CRSArrTime, DepTime,
         ArrTime, ArrDelay, ArrDelayMinutes,
         CarrierDelay, WeatherDelay, NASDelay,
         SecurityDelay, LateAircraftDelay, DepDelay,
         DepDelayMinutes, DivDistance, DivArrDelay)
```

Print and recheck our subset airline data using `dim()` on `sub_airline`.

This subset dataframe should include 2855 rows and 21 columns.

[ ]:
```
# Check dimensions of the dataset
dim(sub_airline)
```

Question #2:

Find the name of the columns of the dataframe

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.
```r colnames(sub_airline) ```

Save Dataset

Correspondingly, "readr" enables us to save the dataset to csv by using the `write_csv()` method, you can add the file path and name along with quotation marks in the brackets.

For example, if you wanted to save the dataframe sub_airline as lax_to_jfk.csv to your local machine in the current directory, you can use the syntax below:

[ ]:
```
write_csv(sub_airline, "lax_to_jfk.csv")
```

Read/Save Other Data Formats

We can also read and save other file formats, we can use similar functions to **`read_csv()`** and **`write_csv()`** for other data formats, the functions are listed in the following table:

Data Format	Read	Save
csv	`read_csv()`	`write_csv()`
tsv	`read_tsv()`	`write_tsv()`
delimiter	`read_delim()`	`write_delim()`
...	...	...

You can find more functions from the "readr" documentation here.

2. Basic Insights of the Dataset

After reading data into a dataframe, it is time for us to explore the dataset a little more.
There are several ways to obtain essential insights of the data to help us better understand our dataset.

Data Types

Data has a variety of types.

The main types stored in R are: `numeric`, `integer`, `complex`, `logical`, `character`. In order to better learn about each attribute, it is always good for us to know the data type of each column.

Let's check by using `sapply()` to apply `typeof` to every column in the dataframe `sub_airline`:

[ ]:
```
sapply(sub_airline, typeof)
```

Numeric and Integer Types

Decimal values are called `numeric` in R. It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.

[ ]:
```
x = 10.5       # assign a decimal value
class(x)       # print the class name of x, which should be numeric
```

Furthermore, even if we assign an integer to a variable k, it is still being saved as a numeric value.

[ ]:
```
k = 1          # assign an integer value
class(x)       # print the class name of x, which should be numeric
```

In order to create an integer variable in R, we invoke the integer function. We can be assured that y is indeed an integer by applying the is.integer function.

[ ]:
```
y = as.integer(3)        # assign a integer value
class(y)                 # print the class name of y, which should be integer
```

Complex Type

A complex value in R is defined via the pure imaginary value *i* (`0i` in R).

[ ]:
```
z = 0i
class(z)
```

Logical Type

A logical value is often created via comparison between variables, such as True, False.

[ ]:
```
logical_values = c(TRUE, T, FALSE, F)
class(logical_values)
```

Character Type

A character object is used to represent string values in R.

[ ]:
```
class('this is a character')
```

Dplyr for Data Wrangling and Transformation

The dplyr package is very useful to transform data and get basic insights from the dataset. The most important functions are:

- `select()`
- `filter()`
- `summarize()`
- `arrange()`
- `mutate()`
- `group_by()`

Pipe

Before we dive deeper into using some of the functions of dyplr, we first introduce the **pipe** operator `%>%`. The pipe operator allows us to chain together dplyr data wrangling functions.
```

An advantage of using pipe is that it replaces having messy nested functions that can be difficult to interpret like:

```
summarize(group_by(filter(sub_airline, Month == 1), Reporting_Airline), avg_carrier_delay = mean(CarrierDelay, na.rm = TRUE))
```

The same code can be written using pipe and makes it much easier to understand:

```
sub_airline %>%
  filter(Month == 1) %>%
  group_by(Reporting_Airline) %>%
  summarize(avg_carrier_delay = mean(CarrierDelay, na.rm = TRUE))
```

When interpreting the above lines of code, the pipe operator can be read as "then". The `%>%` operator allows us to go from one step in dplyr to the next easily, so the above example can clearly be interpreted as:

- **`filter`** our data frame for month 1 *then*
- **`group_by`** the reporting airline types *then*
- **`summarize`** the average `CarrierDelay` for each airline. The final output is the average `CarrierDelay` for each `Reporting_Airline` in month 1.

Summarize

Now, let's go over some examples using basic dplyr functions with pipe. If we would like to get a statistical summary of a column, such as count, column mean value, column standard deviation, etc., we can use the `summarize()` method (can also use `summarise()`).

This method will provide various summary statistics, excluding `NA` (Not Available) values.

[ ]:
```
# group_by / summarize workflow example
sub_airline %>%
  group_by(Reporting_Airline) %>%
  summarize(avg_carrier_delay = mean(CarrierDelay, na.rm = TRUE)) # use mean value
```

The statistical metrics can tell the data scientist if there are mathematical issues that may exist in a particular column, such as extreme outliers and large deviations. The data scientist may have to address these issues later. The above example used `mean()` to return the mean value of arrival delay, you may also use `sd` for standard deviation, `median`, etc.

[ ]:
```
# group_by / summarise workflow example
sub_airline %>%
  group_by(Reporting_Airline) %>%
  summarize(sd_carrier_delay = sd(CarrierDelay, na.rm = TRUE)) # use standard deviation
```

Question #3:

Using `sub_airline`, get the mean of `ArrDelay` for each `Reporting_Airline`. In other words, group by `Reporting_Airline` and summarize the mean of `ArrDelay` of each reporting airline. Remember to use `na.rm = TRUE`.

▼Click here for the solution.

```
sub_airline %>%
  group_by(Reporting_Airline) %>%
  summarize(airline_Delay = mean(ArrDelay, na.rm = TRUE))
```

Glimpse

Another method you can use to check your dataset is:

```
glimpse()
```

It provide a concise summary of your DataFrame.

[ ]:
```
# look at the info of airline dataset
glimpse(sub_airline)
```

Here we are able to see the information of our dataframe, it prints out column names, column types, and first few values for brief data previews. It also shows us this subset dataset has 2,855 rows and 21 columns.

This subset dataset `sub_airline` ("lax_to_jfk.csv") will be used throughout the course.

Excellent! You have just completed the Introduction Notebook!

About the Authors:

This notebook was written by Yiwen Li and Gabriela De Queiroz.

Yiwen Li has approximately three year experiences in big tech industry. Currently, she is a developer advocate, a data scientist, a product manager at IBM, where she designs and develops data science solutions and Machine Learning models to solve real world problems. She has delivered talks this year in JupyterCon, PyCon, Pyjamas, CrowdCast.ai, Global AI on Tour 2020 and Belpy 2021 with hundreds of attendants per talk. Gabriela de Queiroz is a Sr. Engineering & Data Science Manager at IBM where she manages and leads a team of developers working on Data & AI Open Source projects. She works to democratize AI by building tools and launching new open source projects. She is the founder of AI Inclusive, a global organization that is helping increase the representation and participation of gender minorities in Artificial Intelligence. She is also the founder of R-Ladies, a worldwide organization for promoting diversity in the R community with more than 190 chapters in 50+ countries. She has worked in several startups and where she built teams, developed statistical models, and employed a variety of techniques to derive insights and drive data-centric decisions


Simple
0
1
R | I...
Initialized (additional servers needed)
Mem: 231.50 / 6144.00 ...
Mode: Comma...
Ln 1, Co...
English (Americ...
DA0151EN-Review-Introduction-to-data-acquisition-using-tidyverse.ip...