A new framework of conducting parametric analysis on EnergyPlus models via R eSim 2020 Conference

Hongyuan Jia[1] , Adrian Chong[2]
[1]SinBerBEST Program, Berkeley Education Alliance for Research in Singapore, Singapore, 138602, Singapore
[2]Department of Building, School of Design and Environment, National University of Singapore, 4 Architecture Drive, Singapore, 117566, Singapore

Abstract

Parametric analysis using EnergyPlus has been a powerful approach to enable investigation of environmental and energy performance for different design and retrofit design alternatives. However, the absence of streamlined workflow integrating model editing, simulation and output extracting raises problems in the productivity of parametric analysis. This paper presents a newly developed framework for conducting parametric analysis using EnergyPlus via the programming language R package called "eplusr". The proposed framework, with a data-centric design philosophy, focuses on not only EnergyPlus model editing but also parametric simulation output data extraction and analysis. It provides rich-featured interfaces to query and modify models programmatically and a simple yet flexible and extensible prototype of conducting parametric simulations and collecting all results in one go. The paper discusses the philosophy behind the framework, its architecture and core capabilities, and uses simple tasks to demonstrate the streamlined workflow of conducting parametric analysis in R.

Introduction

Nowadays, EnergyPlus(Crawley et al., 2001) has been widely used in building performance assessment and building regulation compliance. Automating simulation tasks has been approved to be an extremely useful way not only for developing and analyzing building energy models, but also for conducting large-scale parametric analyses (Roth et al., 2018).

It is almost essential to use scripting or other techniques for conducting parametric analyses, either for design and retrofit optimization, model input calibration, uncertainty analysis. There are several existing EnergyPlus-based parametric simulation tools (Philip, 2020; Guglielmetti et al., 2011; Big Ladder Software, 2020) that are tailored for different purposes. eppy (Philip, 2020) is a library for manipulating EnergyPlus models programmatically via python programming language. It parses EnergyPlus IDF files into a python object and provides low-level programmatic access to EnergyPlus inputs. Moreover, eppy also provides additional fields for some specific EnergyPlus objects, e.g. building surface areas and zone volumes. jEplus (Yi, 2020) is a software written in Java to perform complex parametric analysis on multiple design parameters. It contains tools to create and manage simulation jobs and to collect results. Modelkit (Big Ladder Software, 2020) is a free and open-source framework for parametric modelling using EnergyPlus. It is capable of automating the generation and management of EnergyPlus models via its templates and scripting tools.

Tools described above are mostly focusing on creating and managing parametric simulations. However, the output is not always friendly in format for applying these methods, which makes data pre-processing an essential but time-consuming and laborious process for any data-driven analytics There is an absence of streamlined workflow integrating model editing, simulation and output extracting, which could raise problems in the productivity of parametric analysis.

In this paper, we introduce a new parametric simulation framework called eplusr (Jia, 2020) using the statistical programming language R (R Core Team, 2019). eplusr is different from existing tools because of its data-centric design philosophy. The main goal of this framework is to provide a rich-featured R toolbox that contains a set of abstractions to help modify EnergyPlus simulation inputs in a programmatic way and provides a simple yet flexible and extensible prototype of conducting parametric simulations.

Software architecture

eplusr is developed using the R language (R Core Team, 2019) and has been published on CRAN (The Comprehensive R Archive Network) (Jia, 2020). The source code is hosted publicly in a version-controlled repository via GitHub. eplusr is released under the MIT license and runs on Linux, macOS and Windows operating systems. It can be easily installed via a single command as shown in Listing 1.

Listing 1

```
install.packages("eplusr")
```

eplusr adopts the Object-Oriented Programming (OOP) programming paradigm (Wikipedia, 2020).

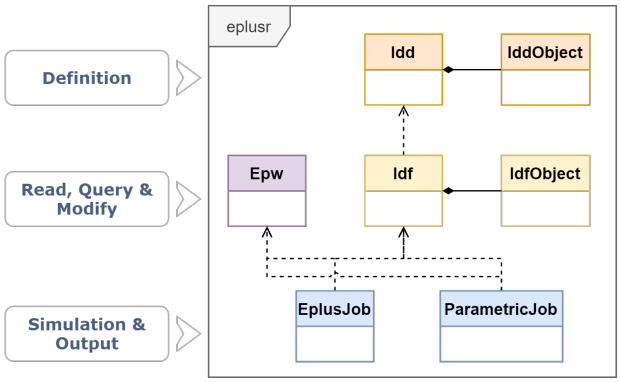

Figure 1: Class structure of the libaray, including (a) Idd and IddObject for data schema, (b) Idf and IdfObject for model modification, (c) Epw for weather file modification, (d) EplusJob for single simulation, and (e) ParametricJob for parametric simulations.

As shown in Fig. 1, it introduces 6 main classes that can be further classified into three categories based on their different focuses:

1. Data schema
2. Weather and model modification
3. Simulation and output extraction

Model data schema

EnergyPlus uses a text input file format named IDF (Input Data File) as a model file, and each version of an IDF file has a corresponding version of an IDD (Input Data Dictionary) file who works as a data format schema. In eplusr, `Idd` class and `IddObject` class provides parsing and data-storing functionality of an IDD file. These data will be used for subsequent parsing and data verification of all IDFs of that version. This makes eplusr be able to handle various versions of IDF files at the same time easily.

For most of the time, users do not need to directly interact with `Idd` and `IddObject` objects, but instead simply using `Idf` and `IdfObject` objects to modify an IDF file.

Model modification

eplusr uses `Idf` class to present a whole IDF file and `IdfObject` class to present a single object in IDF. Both `Idf` and `IdfObject` classes contain around thirty member functions for low-level programmatic access to and modification of the IDF data. Every modification on the IDF data will be verified to make sure the result complies with underlying IDD.

`Idf` class provides rich-featured interfaces to modify models via various scope, including scope of a single object, scope of grouped objects level, and scope of all objects in a certain class.

Listing 2

```
model$set(
  `Supply Fan 1` = list(Fan_Total_Efficiency = 0.9),

  .("Metal Decking", "Metal Roofing") :=
      list(Roughness = "Smooth", Thickness = 0.003),

  Lights := list(Watts_per_Zone_Floor_Area = 7.0)
)
```

Listing 2 demonstrates the flexible value modifying scopes provided by `Idf$set()` method. `model` here is an `Idf` object. In Line 2, the total efficiency of a `Fan:VariableVolume` object named `Supply Fan 1` will be changed to 0.9; In line 4-5, the roughness and thickness of two materials named `Metal Decking` and `Metal Roofing` will be changed; While in line 7, the lighting power density of all lights will be changed to 7.0 W/m^2.

Besides of `Idf$set()`, `Idf$update()` method provides an interface that can achieve the same results but with a `data.frame` input. `data.frame` is an R representation of a table where each column contains values of one variable. In Listing 3, Line 1, 4 and 8 extract object data into a `data.table`s (Dowle et al., 2019), an extension of `data.frame` extremely optimized for speed. Line 2, 5, 6 and 9 are used to modify certain field values. Line 11-13 are then call `Idf$update()` method to update object data.

Users can perform any modifications to that data and then update corresponding object values by feeding the updated data to `Idf$update()` method.

Listing 3

```
fan <- model$to_table("Supply Fan 1")
fan[field == "Fan Total Efficiency", value := "0.9"]

mat <- model$to_table(c("Metal Decking", "Metal Roofing"))
mat[field == "Roughness", value := "Smooth"]
mat[field == "Thickness", value := "0.003"]

lit <- model$to_table(class = "Lights")
lit[field == "Watts per Zone Floor Area", value := "7.0"]

model$update(fan)
model$update(mat)
model$update(lit)
```

There are several other methods in `Idf` class that can duplicate, rename, delete existing objects and also add new ones.

Weather modification

The weather data format used in EnergyPlus is called EPW (EnergyPlus Weather), a simple, text-based format with comma-separated data. eplusr introduces the `Epw` class to directly extract both meta data and core weather data of an EPW file. Moreover, `Epw` class is capable of creating new and modifying existing EPW files, which is usually an essential process when doing model parameter calibration. An example of utilizing the `Epw` class is the *epwshiftr*[1] R package which can be used to adapt EPW files to incorporate climate change predictions using the morphing method.

[1] Link: `https://cran.r-project.org/package=epwshiftr`

Simulation and output extraction

eplusr uses `EplusJob` class and `ParametricJob` class to represent a single EnergyPlus simulation job and multiple parametric jobs, respectively. Both classes have methods to collect various types of simulation outputs.

Listing 4

```
job <- model$run()

job$report_data(
  key_value = "heat pump cooling mode ahu1",
  name = "cooling coil total cooling rate", case = "example",
  all = TRUE, simulation_days = 1, hour = 11, minute = 0
)

job$tabular_data(
  report_name = "ComponentSizingSummary",
  report_for = "entire facility",
  table_name = "airterminal:singleduct:uncontrolled",
  column_name = "user-specified maximum air flow rate",
  row_name = "space5-1 direct air"
)
```

Design philosophy

eplusr keeps a data-centric design philosophy. Everything can be exposed.

This design philosophy is reflected in the design data structure and the return format results extraction

- Everything is stored as tables
- Every results is returned in a tidy-format

Table data structure

Under the hook, eplusr uses a SQL-like structure to store both IDF and IDD data in various tables. As shown in Fig. 2. Data of an IDF is stored in three tables, i.e. object, value, and reference while data of an IDD is stored in four tables, i.e. group, class, field and reference. With this data structure, to modify an EnergyPlus model in eplusr is equal to change the data in those `Idf` tables accordingly, in the context of specific `Idd` data. In this sense, all methods in `Idf` and `Idd` classes are just interface wrappers which expose the modification process in a more user-friendly manner.

All the data are masked from end users but can be easily extracted when needed.

Tidy-format outputs

When extracting simulation results, instead of directly reading and manipulating the CSV and HTML files, eplusr uses EnergyPlus SQL output to extract results of `Output:Variable`, `Output:Meter*`, and `Output:Table` objects specified in the IDF. The main benefit of this approach is that it makes sure eplusr always return the simulation results in a *Tidy Data* format.

Tidy data format was first proposed by Wickham (2014), which is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types.

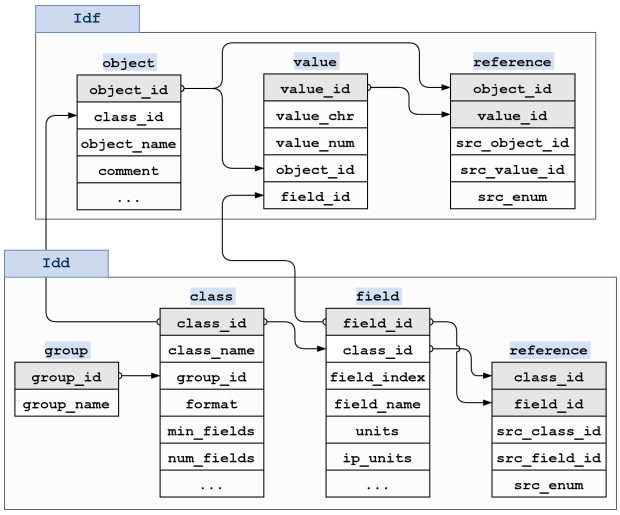

Figure 2: Data structure of an Idf and Idd object. The Rational Database structure follows the idea of database normalization where each variable is expressed in only one place, avoiding any data redundancy. The hierarchy structure of the IDF data schema is retained through various tables. Data integrity is maintained via relations among table variables. Each RD has a reference table to store the referencing relations among various field values.

In tidy data:

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

Tidy data makes it easy for an analyst or a computer to extract needed variables because it provides a standard way of structuring a dataset. Tidy data is particularly well suited for vectorized programming languages like R, because the layout ensures that values of different variables from the same observation are always paired.

However, EnergyPlus CSV output does not present data in a tidy way. Taking Table 1 an example. We are facing one main problems when working with this type of EnergyPlus CSV output, i.e. column headers contain values, not only variable names.

In EnergyPlus CSV output, column headers are composed in format *Key value* (`MEETING_ROOM_1`) + *Variable name* (`Zone Total Internal Gain Rate`) + *Units* (`W`) + *Reporting frequency* (`Hourly`). This format often leads to additional data cleaning efforts. For instance, to get all outputs for variable `Zone Total Internal Latent Gain Rate`, users have to write their methods of splitting column headers into different parts or subsetting column using regular expressions.

When using eplusr, the same results in Table 1 are always presented in a tidy format, as shown in Table 2. It transforms the original column head-

Table 1: Example of EnergyPlus CSV output

Date/Time	ROOM1:Zone Total Internal Latent Gain Rate [W](Hourly)
04/21 01:00:00	16.99
04/21 02:00:00	17.07
04/21 03:00:00	17.06
04/21 04:00:00	17.03
04/21 05:00:00	16.99
04/21 06:00:00	16.96
04/21 07:00:00	16.22
04/21 08:00:00	14.04
04/21 09:00:00	14.04
04/21 10:00:00	14.04

ers from EnergyPlus CSV outputs into four separated columns, i.e. `key_value`, `name`, `units` and `reporting_frequency`. Despite those changes, eplusr will also add an additional column named `case`, which is by default the IDF file name without extension. `case` column can be quite useful and serve as an indicator to separate each simulation results when extracting outputs from multiple parametric simulations. Moreover, instead of presenting the simulated date and time as strings shown in Table @ref(tab: mess-csv), eplusr will calculate a proper year value for each run period and compose a series of `DateTimeClasses` values in the `datetime` column. This makes it quite straightforward to apply further time-series analysis on the simulation results.

In summary, giving data in a tidy format has several advantages in R, as it will make sure the data is always fit for directly handling by the *tidyverse* (Wickham et al., 2019) package ecosystem, which is a language for solving data science challenges with R code.

Listing 5 shows how easy this data format can be fitted into the tidyverse workflow.

Listing 5

```
library(tidyverse)
result %>%
  filter(name == "Zone Total Internal Latent Gain Rate") %>%
  mutate(date = as.Date(datetime)) %>%
  group_by(case, key_value, date) %>%
  summarize(value = mean(value)) %>%
  ggplot() +
  geom_line(aes(date, value, color = key_value))
```

The `%>%` is the pipe operator which conveys an object forward into the function on the right hand side. We first filter the data to only include output variable `Zone Total Internal Latent Gate Rate`, and we then create a new `date` column, calculate the daily mean latent loads by the IDF and zone names, and we finally draw a line plot to show the latent load variations with each zone having a separate color.

The parametric simulation framework

eplusr provides a class `ParametricJob` to do parametric simulations. It is simple to use but is also

Table 2: Tidy format of EnergyPlus output

case	datetime	key_value	name	units	reporting_frequency	value
model	2014-04-21 01:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	16.99
model	2014-04-21 02:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	17.07
model	2014-04-21 03:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	17.06
model	2014-04-21 04:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	17.03
model	2014-04-21 05:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	16.99
model	2014-04-21 06:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	16.96
model	2014-04-21 07:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	16.22
model	2014-04-21 08:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	14.04
model	2014-04-21 09:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	14.04
model	2014-04-21 10:00:00	ROOM1	Zone Total Internal Latent Gain Rate	W	Hourly	14.04

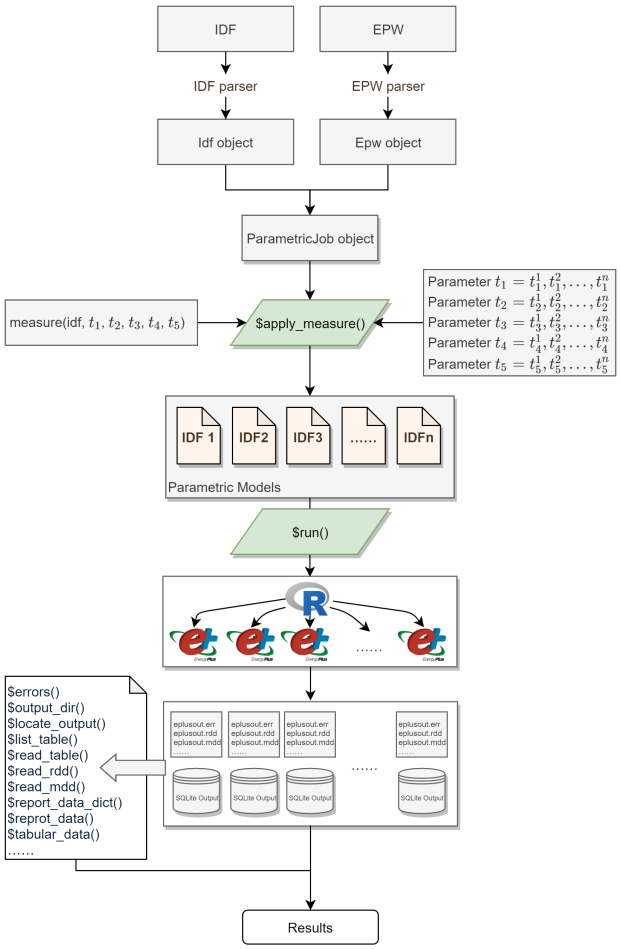

Figure 3: Workflow of a parametric simulation using eplusr

quite flexible and extensible. It takes full advantages of eplusr model editing and result collecting functionalities. The overall process of the framework is shown in Fig. 3 and can be summarized as follows:

1. Initialize a `ParametricJob` object. Creating a parametric job in eplusr is just a simple call of the constructor `param_job()` function. It takes an IDF file as the *seed* and an EPW file as the *weather* to be used, as shown in Listing 6.

Listing 6

```
param <- param_job(idf = model, epw = path_epw)
```

2. Construct a measure function. Here, the concept of measure in eplusr is inspired by "measures" in OpenStudio (Guglielmetti et al., 2011). Basically, a measure is a function that takes an `Idf` object and any other arguments as input, and returns a modified `Idf` object as output. This approach makes it easy for users to access to all the model-editing API (Application Programming Interface) that eplusr provides. Listing 7 shows a simple measure example that modifies air change rate (ACH) of infiltration objects for all zones.

Listing 7

```
set_infil_rate <- function (idf, infil_rate) {
  idf$set(`ZoneInfiltration:DesignFlowRate` :=
      list(air_changes_per_hour = infil_rate)
  )
  idf
}
```

3. Create parametric models. The method `ParametricJob$apply_measure()` allows to apply a measure with specified parameter values to an `Idf` and create parametric models for later simulations. How the parameter values will be generated is all left to the users, which makes it possible to leverage all the statistical methods and libraries that R provides. In Listing 8, we create 30 parametric models with ACH changing from 0.1 to 3.0 with a 0.1 step.

Listing 8

```
param$apply_measure(set_infil_rate, seq(0.1, 3, by = 0.1))
```

4. Run simulations and collect results. Calling `ParametricJob$run()` method will run all parametric simulations in parallel and put each simulation outputs in a separate folder. After the simulations complete, all results from different models can be retrieved in one step using all the data collecting APIs, including `ParametricJob$report_data()`, `ParametricJob$tabular_data()`, and `ParametricJob$read_table()`. As described above, the tables returned will always be presented in a tidy-format, which makes it easy to apply any further analyses in the R ecosystem. In Listing 8, we collect the annual total site energy of all simulations.

Listing 9

```
param$run()

param$tabular_data(
    table_name = "Site and Source Energy",
    column_name = "Total Energy",
    row_name = "Total Site Energy"
)
```

Examples of extension

One good example of extending the parametric framework in eplusr is the epluspar (Jia and Chong, 2020) R package, which provides new classes for conducting parametric analysis on EnergyPlus models, including sensitivity analysis using Morris method (Morris, 1991) and Bayesian calibration using the method proposed by Chong et al. (2017). All the new classes introduced in epluspar package are based on the `ParametricJob` class. The main differences lie in applying specific statistical methods for sampling parameter values when calling `ParametricJob$apply_measure()`.

Conclusion

Parametric analysis using EnergyPlus has been a powerful approach to enable investigation of environmental and energy performance for different design and retrofit design alternatives. However, the absence of streamlined workflow integrating model editing, simulation and output extracting raises problems in the productivity of parametric analysis. This paper presents a newly developed framework for conducting parametric analysis using EnergyPlus via the programming language R package called "eplusr".

The proposed framework, with a data-centric design philosophy, focuses on not only EnergyPlus model editing but also parametric simulation output data extraction and analysis. It provides rich-featured interfaces to query and modify models programmatically.

The parametric prototype developed provides a set of abstractions to ease the process of parametric model generation, design alternative evaluation, and large parametric simulation management. It is capable of defining various analyses using any algorithms available in R. The flexibility and extensibility of the parametric simulation prototype in this framework are demonstrated by its easy adoption to perform multi-objective optimization and Bayesian calibration.

Acknowledgements

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. It was funded through a grant to the Berkeley Education Alliance for Research in Singapore (BEARS) for the Singapore-Berkeley Building Efficiency and Sustainability in the Tropics (SinBerBEST) Program. BEARS has been established by the University of California, Berkeley as a center for intellectual excellence in research and education in Singapore.

References

Big Ladder Software (2020, March). Modelkit/-Params Framework.

Chong, A., K. P. Lam, M. Pozzi, and J. Yang (2017, November). Bayesian calibration of building energy models with large datasets. *Energy and Buildings 154*, 343–355.

Crawley, D. B., L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, R. K. Strand, R. J. Liesen, D. E. Fisher, M. J. Witte, and J. Glazer (2001, April). EnergyPlus: Creating a new-generation building energy simulation program. *Energy and Buildings 33*(4), 319–331.

Dowle, M., A. Srinivasan, J. Gorecki, M. Chirico, P. Stetsenko, T. Short, S. Lianoglou, E. Antonyan, M. Bonsch, H. Parsonage, S. Ritchie, K. Ren, X. Tan, R. Saporta, O. Seiskari, X. Dong, M. Lang, W. Iwasaki, S. Wenchel, K. Broman, T. Schmidt, D. Arenburg, E. Smith, F. Cocquemas, M. Gomez, P. Chataignon, D. Groves, D. Possenriede, F. Parages, D. Toth, M. Yaramaz-David, A. Perumal, J. Sams, M. Morgan, M. Quinn, @javrucebo, @marc-outins, R. Storey, M. Saraswat, M. Jacob, M. Schubmehl, and D. Vaughan (2019, December). Data.table: Extension of 'data.frame'.

National Renewable Energy Lab. (NREL), Golden, CO (United States) (2011, December). *OpenStudio: An Open Source Integrated Analysis Platform; Preprint*.

Jia, H. (2020, February). Eplusr: A toolkit for using EnergyPlus in R.

Jia, H. and A. Chong (2020, March). Epluspar: Conduct parametric analysis on EnergyPlus models in R.

Morris, M. D. (1991, May). Factorial Sampling Plans for Preliminary Computational Experiments. *Technometrics 33*(2), 161–174.

Philip, S. (2020, March). Eppy: Scripting language for E+, Energyplus.

R Core Team (2019). R: A language and environment for statistical computing.

Roth, A., J. Bull, S. Criswell, P. Ellis, J. Glazer, D. Goldwasser, N. Kruis, A. Parker, S. Philip, and D. Reddy (2018, September). Scripting frameworks for enhancing energyplus modeling productivity. In *Proceedings of SimBuild 2018*, pp. 8.

Wickham, H. (2014, September). Tidy Data. *Journal of Statistical Software 59*(1), 1–23.

Wickham, H., M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Pedersen, E. Miller, S. Bache, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani (2019, November). Welcome to the Tidyverse. *Journal of Open Source Software 4*(43), 1686.

Wikipedia (2020, February). Object-oriented programming.

Yi, Z. (2020, March). jEplus.