MATH2405 TP1, 2022

Assignment 1

Timm Rahrt [redacted]

Setup

```
# Un-commend if the Webscraping API package "RSocrata" is not installed
#install.packages("RSocrata")

library(RSocrata) # API for web scraping if a website doesn't contain an HTML table
library(readr) # Useful for importing data
library(foreign) # Useful for importing SPSS, SAS, STATA etc. data files
library(dplyr) # Useful for pipe operator and data manipulation
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(knitr) # Useful for creating nice tables
library(magrittr) # So we can pipe assign "%<>%"
```

Data Description

The dataset “complete_data” stems from the Connecticut Open Data” website, which hosts various datasets across fields like Government, Business and Education. The original dataset provides sales information about real estate objects from 2001 to 2021 and can be found here (https://data.ct.gov/Housing-and-Development/Real-Estate-Sales-2001-2021-GL/5mzw-sjtu/data_preview) (Ct Data, 2024). It consists of 14 variables and over 1 million observations. This assessment uses a sample of this data, containing 10 variables and 10000 randomly selected observations. The variables in the dataset are:

- `serialnumber` : Unique identifier of the real estate object
- `listyear` : Year when the objects were listed
- `daterecorded` : Date the property was added to the dataset
- `town` : Town where the property is located
- `address` : Exact location of the real estate

- `assessedvalue` : Assessed value of the property
- `saleamount` : Amount the property was sold for
- `salesratio` : Ratio of the sale, calculated as (saleamount / assessedvalue)
- `propertytype` : Type of the property (e.g. Residential, Vacant Land)
- `residentialtype` : Type of the residence (e.g. Single Family, Two Family)

This dataset can be used to visualise trends in property values in Connecticut.

Import Data

This section imports the data from the `url` into R. Since the websites data table isn't stored in HTML tables, we use a different package for webscraping then `rvest`. `RSocrate` enables webscraping data using an API endpoint, which is provided under the websites 'Export' tab. The package extracts a JSON file and stores this as a dataframe in R. First, we have to install the package by using `install.packages()` and activate it with `library()` (Github.com, 2024).

Next, we store the API endpoint in a variable called `url` and save the whole dataset into a second variable "complete_data" by calling the `read.socrata()` function. To generate a sample dataset with 10000 randomly selected rows, we require to set a seed and assign a random attribute to ensure reproducibility (rdocumentation.org, 2024). The next code section uses the dpylr pipe operator `%>%` to apply the `slice_sample()` function on the complete dataset. `slice_sample()` allows us to subset randomly selected rows of a dataset with all attributes, the output will be saved to a new variable `rows_df` (Dplyr.tidyverse.org, 2024). Finally, we create our sample dataframe, which will be achieved by slicing the randomly selected rows, so we keep all rows but just variables at position 1 to 8 and 12 to 13. The result will be saved to a new variable `df`, which represents our sample dataset. We call the `head()` function on `df` with an additional attribute of `10`, to display the first ten rows and all columns.

```
# Import the whole dataset
url <- "https://data.ct.gov/resource/5mzw-sjtu.json"
complete_data <- read.socrata(url)
# Select random rows once
set.seed(123)
rows_df <- complete_data %>% slice_sample(n = 10000)
# Slice population dataset to sample
df <- rows_df[, c(1:8,12:13)]
head(df,10)
```

	serialnumber <chr>	listyear <chr>	daterecorded <dttm>	town <chr>	address <chr>	a <chr>
1	30410	2003	2004-07-13	East Lyme	18 WOODLAND RD	1
2	150308	2015	2016-06-16	Cheshire	293 GREENWOOD DR	2
3	100617	2010	2011-08-04	Danbury	136 PEMBROKE RD #11-106	1
4	70721	2007	2008-08-29	Stratford	623B ONONDAGA LN	2
5	1200107	2012	2013-06-11	Weston	23 COVENANT LANE	4
6	190018	2019	2019-10-21	Berlin	319 NEW BRITAIN ROAD UNIT 311	6

7	140124	2014	2015-01-07	Southbury	39 D HERITAGE VILLAGE	5
8	200406	2020	2021-03-22	Enfield	25 OLD KING ST	1
9	130351	2013	2014-05-23	Wallingford	169 SOUTH MAIN ST	3
10	70186	2007	2008-05-30	New London	11 THOMPSON CT	7

1-10 of 10 rows | 1-7 of 11 columns

Inspect dataset and variables

After importing our dataset, we inspect `df` for a deeper understanding of the data. First, we call `dim()` to check the dimension. The function outputs a vector of the total number of observations (10000) and variables (10).

```
dim(df)
```

```
## [1] 10000    10
```

We apply `sapply()` to understand the type of all variables, as this determines what analysis method we can later apply on which variable. `sapply()` iterates through a sliced version of `df`, which returns zero observations but all variables, and applies the `typeof()` function on them (rdocumentation.org, 2024). Most of the variables are from type `character`, just `daterecorded` is a double. Alternatively, `summary()` also returns all variable names, their class, length and it reveals some basic statistics for the numeric variables.

```
sapply(df[0,],typeof)
```

```
##    serialnumber        listyear    daterecorded            town         address
##     "character"     "character"        "double"     "character"     "character"
##   assessedvalue      saleamount      salesratio    propertytype residentialtype
##     "character"     "character"     "character"     "character"     "character"
```

```
summary(df)
```

```
##  serialnumber         listyear          daterecorded
##  Length:10000       Length:10000       Min.   :2001-10-01 00:00:00.00
##  Class :character   Class :character   1st Qu.:2005-10-26 00:00:00.00
##  Mode  :character   Mode  :character   Median :2012-02-08 12:00:00.00
##                                        Mean   :2012-02-13 04:00:02.16
##                                        3rd Qu.:2018-01-24 06:00:00.00
##                                        Max.   :2022-09-30 00:00:00.00
##      town             address          assessedvalue       saleamount
##  Length:10000       Length:10000       Length:10000       Length:10000
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##   salesratio        propertytype       residentialtype
##  Length:10000       Length:10000       Length:10000
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
```

The below code outputs the sum of all `NA` in our character variables by iterating the anonymous function `sum(is.na())` on all variables of `df` (campus.datacamp.com, 2024). Here, `is.na()` takes a variable as an input and outputs a TRUE for all `NA` and FALSE if the value contains information. `sum()` coerces the output of `is.na()` into numeric values where FALSE changes to 0 and TRUE into 1. Therefore, the `sum` of `is.na` represents all missing values in this variable.

```
#sapply to check all character variables for missing data
sapply(df,function (x) sum(is.na(x)))
```

```
##    serialnumber        listyear    daterecorded            town         address
##               0               0               0               0               1
##   assessedvalue      saleamount      salesratio    propertytype residentialtype
##               0               0               0            3566            3680
```

To replace all NAs with an updated `Property Type`, `Residential Type` and `Address` variable, we pipe the `mutate()` function on `df`, overwrite the variable name and call an `ifelse` function. `ifelse(is.na(variable))` replaces the values in the variable if `is.na()` returns `TRUE` and skips the value if the output is `FALSE` (rdocumentation.org, 2024). We do this for all three variables and check, if we were successful.

```
# Replace all NA's with an updated df$variable
df %<>%
  mutate(propertytype = ifelse(is.na(propertytype), "Unkown", propertytype))
df %<>%
  mutate(residentialtype = ifelse(is.na(residentialtype), "Unkown", residentialtyp
e))
df %<>%
  mutate(address = ifelse(is.na(address), "Unkown", address))
# Test if we were successful
sapply(df,function(x) sum(is.na(x)))
```

```
##    serialnumber        listyear    daterecorded            town         address
##               0               0               0               0               0
##   assessedvalue      saleamount      salesratio    propertytype residentialtype
##               0               0               0               0               0
```

Next, we convert variable types with the `as.` function, to ensure proper statistical analysis on numeric variables later. We access columns with the `$` operator, to change each variable individually. Since the variables `year` , `propertytype` and `residentialtype` consist of the same 21, 11 and 6 values, we change their type from `character` to `factor` .

```
df$listyear <- as.factor(df$listyear)
df$daterecorded <- as.Date(df$daterecorded)
df$assessedvalue <- as.numeric(df$assessedvalue)
df$saleamount <- as.numeric(df$saleamount)
df$salesratio <- as.numeric(df$salesratio)
df$propertytype <- as.factor(df$propertytype)
df$residentialtype <- as.factor(df$residentialtype)
```

```
cat("Levels for the column `listyear`:")
```

```
## Levels for the column `listyear`:
```

```
levels(df$listyear)
```

```
##  [1] "2001" "2002" "2003" "2004" "2005" "2006" "2007" "2008" "2009" "2010"
## [11] "2011" "2012" "2013" "2014" "2015" "2016" "2017" "2018" "2019" "2020"
## [21] "2021"
```

```
cat("Levels for the column `propertytype`:")
```

```
## Levels for the column `propertytype`:
```

```
levels(df$propertytype)
```

```
##  [1] "Apartments"    "Commercial"    "Condo"         "Four Family"
##  [5] "Industrial"    "Residential"   "Single Family" "Three Family"
##  [9] "Two Family"    "Unkown"        "Vacant Land"
```

```
cat("Levels for the column `residentialtype`:")
```

```
## Levels for the column `residentialtype`:
```

```
levels(df$residentialtype)
```

```
## [1] "Condo"         "Four Family"   "Single Family" "Three Family"
## [5] "Two Family"    "Unkown"
```

Finally, to understand each variable more easily, we display their names by calling `names()` and assign updated column names to a new variable `column_names`. Moreover, we call `colnames()` on `df` and input the `column_names` variable, to change all variable names. Following that, `str()` displays a bunch of useful information about a dataset, like its class, total observations, total variables, all updated variable names with their class and couple of observations.

```
names(df)
```

```
##  [1] "serialnumber"    "listyear"        "daterecorded"    "town"
##  [5] "address"         "assessedvalue"   "saleamount"      "salesratio"
##  [9] "propertytype"    "residentialtype"
```

```
# Rename column names
column_names <- c("Serial Number","List Year","Date Recorded","Town","Address","Asse
ssed Value","Sale Amount","Sale Ratio","Property Type","Residential Type")
colnames(df) <- column_names
str(df)
```

```
## 'data.frame':    10000 obs. of  10 variables:
##  $ Serial Number   : chr  "30410" "150308" "100617" "70721" ...
##  $ List Year       : Factor w/ 21 levels "2001","2002",..: 3 15 10 7 12 19 14 20
13 7 ...
##  $ Date Recorded   : Date, format: "2004-07-12" "2016-06-15" ...
##  $ Town            : chr  "East Lyme" "Cheshire" "Danbury" "Stratford" ...
##  $ Address         : chr  "18 WOODLAND RD" "293 GREENWOOD DR" "136 PEMBROKE RD #1
1-106" "623B ONONDAGA LN" ...
##  $ Assessed Value  : num  196700 264350 149500 205170 443900 ...
##  $ Sale Amount     : num  412000 350000 165000 197000 735000 ...
##  $ Sale Ratio      : num  0.477 0.755 0.906 1.041 0.604 ...
##  $ Property Type   : Factor w/ 11 levels "Apartments","Commercial",..: 10 7 3 3 7
3 3 6 7 7 ...
##  $ Residential Type: Factor w/ 6 levels "Condo","Four Family",..: 6 3 1 1 3 1 1 3
3 3 ...
```

Tidy data

In this next chapter, we will conform the data to the tidy principle formulated by Wickham & Grolemund (2016):

1. “Each variable in our dataset must form its own column
2. All 10000 observation must form their own rows
3. Each cell represents one value”

By using the `glimpse()` function on `df`, we extract an overview about the first two tidy principles. All variables are representing an individual column, which contains one row per observation. Since this function doesn’t show us all existing 10000 observations, we use `vapply` to iterate the `is.list` function through `df` and check if any lists hide in the cells, `logical(1)` ensures the output is a logical value (rpubs.com, 2024). Since the function returns `FALSE`, all values stand alone. We see below, `df` doesn’t contain lists, therefore all three rules of Wickham are met and no reshaping is required.

```
# Inspect df
glimpse(df)
```

```
## Rows: 10,000
## Columns: 10
## $ `Serial Number`    <chr> "30410", "150308", "100617", "70721", "1200107", "1…
## $ `List Year`        <fct> 2003, 2015, 2010, 2007, 2012, 2019, 2014, 2020, 201…
## $ `Date Recorded`    <date> 2004-07-12, 2016-06-15, 2011-08-03, 2008-08-28, 20…
## $ Town               <chr> "East Lyme", "Cheshire", "Danbury", "Stratford", "W…
## $ Address            <chr> "18 WOODLAND RD", "293 GREENWOOD DR", "136 PEMBROKE…
## $ `Assessed Value`   <dbl> 196700, 264350, 149500, 205170, 443900, 64500, 5655…
## $ `Sale Amount`      <dbl> 412000, 350000, 165000, 197000, 735000, 112000, 550…
## $ `Sale Ratio`       <dbl> 0.47742718, 0.75528571, 0.90606061, 1.04147208, 0.6…
## $ `Property Type`    <fct> Unkown, Single Family, Condo, Condo, Single Family,…
## $ `Residential Type` <fct> Unkown, Single Family, Condo, Condo, Single Family,…
```

```
vapply(df, is.list, logical(1))
```

```
##    Serial Number        List Year    Date Recorded             Town
##            FALSE             FALSE            FALSE            FALSE
##          Address   Assessed Value      Sale Amount       Sale Ratio
##            FALSE             FALSE            FALSE            FALSE
##    Property Type Residential Type
##            FALSE             FALSE
```

To ensure accurate summary statistics, we check if `0` values are hidden inside the dataset. We pipe `filter()` through `df` to extract all rows with `0` values in the columns `Assessed Value`, `Sale Amount` and `Sale Ratio`, the output are 5 rows.

```
# Check if missing values/ zero values exist
df %>%
  filter(`Assessed Value` == 0 & `Sale Amount` == 0 & `Sale Ratio` == 0)
```

Address <chr>	Assessed Value <dbl>	Sale Amount <dbl>	Sale Ratio <dbl>	Property Type <fct>	Resid <fct>
Unkown	0	0	0	Unkown	Unko
52 STEVENS ST	0	0	0	Unkown	Unko
140 BOLTON AVE	0	0	0	Unkown	Unko
928A HERITAGE VILLAGE	0	0	0	Unkown	Unko
576 ZION ST	0	0	0	Unkown	Unko

5 rows | 5-10 of 10 columns

Next, we calculate the mean of the variables excluding `0` values and save the result to a new variable. The last chunk overwrites the variable and replaces all `0` values with the variable mean. Lastly, we check if we were successful.

```
# Calculate the mean, exclude all `0` values
mean_assessed_value <- mean(df$`Assessed Value`[df$`Assessed Value` != 0])
mean_sale_amount <- mean(df$`Sale Amount`[df$`Sale Amount` != 0])
mean_sale_ratio <- mean(df$`Sale Ratio`[df$`Sale Ratio` != 0])

# Replace with `0` value in df
df %<>%
  mutate(`Assessed Value` = replace(`Assessed Value`,`Assessed Value` == 0, mean_ass
essed_value),
         `Sale Amount` = replace(`Sale Amount`, `Sale Amount` == 0, mean_sale_amoun
t),
         `Sale Ratio` = replace(`Sale Ratio`, `Sale Ratio` == 0, mean_sale_ratio))

# Check if it worked
df %>%
  filter(`Assessed Value` == 0 & `Sale Amount` == 0 & `Sale Ratio` == 0)
```

0 rows | 1-8 of 10 columns

The below takes `df` , groups it by the six `Residential Types` and performs the `summarise()` function to display newly created variables which represent the Mean, Median, Min, Max and Standard Deviation of `Sale Ratio`. `summarise()` doesn't display any other variable then the grouped and the statistical variables (rdocumentation, 2024).

```
# Summary statistics
summary_df <- df %>%
  group_by(`Residential Type`) %>%
  summarise(Mean_Sale_Ratio = mean(`Sale Ratio`),
            Median_Sale_Ratio = median(`Sale Ratio`),
            Min_Sale_Ratio = min(`Sale Ratio`),
            Max_Sale_Ratio = max(`Sale Ratio`),
            Sd_Sale_Ratio = max(`Sale Ratio`))
summary_df
```

Residential Type <fct>	**Mean_Sale_Ratio** <dbl>	**Median_Sale_Ratio** <dbl>	**Min_Sale_Ratio** <dbl>	**Max_Sale_Ratio** <dbl>
Condo	0.8300368	0.6698500	0.003902564	40.871385
Four Family	0.9835488	0.7737071	0.333200000	3.716034
Single Family	0.8940004	0.6830000	0.001822218	149.187500
Three Family	1.6803657	0.6116940	0.136573585	84.655895
Two Family	1.0024729	0.6964250	0.195571942	16.128000
Unkown	5.0518326	0.5019560	0.000101010	8803.000000

6 rows

Statistical summary of all other values:

```
df %>%
  summary()
```

```
##  Serial Number        List Year    Date Recorded            Town
##  Length:10000       2004   : 798   Min.   :2001-09-30   Length:10000
##  Class :character   2020   : 656   1st Qu.:2005-10-25   Class :character
##  Mode  :character   2005   : 572   Median :2012-02-07   Mode  :character
##                     2019   : 564   Mean   :2012-02-12
##                     2002   : 562   3rd Qu.:2018-01-23
##                     2003   : 560   Max.   :2022-09-29
##                     (Other):6288
##    Address           Assessed Value      Sale Amount          Sale Ratio
##  Length:10000       Min.   :      20   Min.   :       10   Min.   :   0.000
##  Class :character   1st Qu.:   90220   1st Qu.:   143375   1st Qu.:   0.489
##  Mode  :character   Median :  142875   Median :   234000   Median :   0.625
##                     Mean   :  274184   Mean   :   391401   Mean   :   2.432
##                     3rd Qu.:  233428   3rd Qu.:   375000   3rd Qu.:   0.796
##                     Max.   :58387880   Max.   :152384149   Max.   :8803.000
##
##        Property Type       Residential Type
##  Single Family:3817   Condo        :1262
##  Unkown       :3566   Four Family  :  24
##  Residential  :1068   Single Family:4573
##  Condo        :1046   Three Family : 155
##  Two Family   : 252   Two Family   : 306
##  Three Family : 117   Unkown       :3680
##  (Other)      : 134
```

Create a list

The below list is a numeric representation of the variable `Residential Type`. The `levels()` function extracts the levels of the factor, `factor()` converts them to a factor, `as.numeric()` to numerical values, `as.list()` to a list and `setNames()` assigns the levels as names to each element in the list (GeeksforGeeks, 2020). `df_list` saves the numeric output:

1. Represents: `Condo`
2. Represents: `Four Family`
3. Represents: `Single Family`
4. Represents: `Three Family`
5. Represents: `Two Family`
6. Represents: `Unknown`

```
df_list <- setNames(as.list(as.numeric(factor(levels(df$`Residential Type`)))),level
s(df$`Residential Type`))
df_list
```

```
## $Condo
## [1] 1
##
## $`Four Family`
## [1] 2
##
## $`Single Family`
## [1] 3
##
## $`Three Family`
## [1] 4
##
## $`Two Family`
## [1] 5
##
## $Unkown
## [1] 6
```

Join the list

Below, we join `df_list` with `df` . First, we create a new dataframe `df2` which contains two variables, `Residential Type` from `df_list` and a new `Residential_Type_Num` variable, which returns the numerical values of the factor. `unlist()` converts the list into a vector and the attribute `check.names` ensures the variable names are kept unchanged (rdocumentation, 2024). We pipe `df2` with a `left_join()` on `df` by the key `Residential Type` , the result will be an output of the whole left table `df` plus an additional column `Residential_Type_Num` of `df2` , where `Residential_Type` of `df` and `df2` match. We display 10 observations this with `head()` .

```
df2 <- data.frame(`Residential Type` = names(df_list), Residential_Type_Num = unlis
t(df_list), check.names = FALSE)
df %<>%
  left_join(df2, by = "Residential Type")
head(df,10)
```

Assessed Value <dbl>	Sale Amount <dbl>	Sale Ratio <dbl>	Property Type <fct>	Residential Type <chr>	Resident
196700	412000	0.4774272	Unkown	Unkown	
264350	350000	0.7552857	Single Family	Single Family	
149500	165000	0.9060606	Condo	Condo	
205170	197000	1.0414721	Condo	Condo	
443900	735000	0.6039456	Single Family	Single Family	
64500	112000	0.5759000	Condo	Condo	
56550	55000	1.0281818	Condo	Condo	

	111070	235000	0.4726000	Residential	Single Family
	360400	572500	0.6295197	Single Family	Single Family
	73150	199500	0.3666667	Single Family	Single Family

1-10 of 10 rows | 7-12 of 12 columns

Subsetting (10 observations)

We subset `df` into 10 observations with all columns. The result will be assigned to a variable `subset_df`. To convert the subsetted dataframe to a matrix, we call `as.matrix()`. Next, we use `str()` to inspect the small dataframe closely.

```
subset_df <- df[1:10,]
subset_df <- as.matrix(subset_df)
str(subset_df)
```

```
##  chr [1:10, 1:11] "30410" "150308" "100617" "70721" "1200107" "190018" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:10] "1" "2" "3" "4" ...
##   ..$ : chr [1:11] "Serial Number" "List Year" "Date Recorded" "Town" ...
```

What differentiates a matrix from a dataframe is that a matrix can just contain a single data type. As the matrix above contains character variables, it coerces all numeric variables into characters and adds double quotation marks to its, previously numeric, values. A type conversion from character into numerical is impossible (carpentries-incubator.github.io, 2024).

Subsetting (first and last variable)

This section extracts the first and last variable of the `df` and saves the result in a new `RData` file. We pipe the dataframe through `select()`, which takes variable names or their indexing position as an input. To display the first variable, we use `1` and `cnol(df)` as our attributes, as this returns the first and last columns of a dataframe, as `ncol` takes a dataframe, counts through all variables and outputs a number. The result will be saved to a new variable `first_last_var_df`.

```
first_last_var_df <- df %>% select(1,ncol(df))
head(first_last_var_df)
```

	Serial Number <chr>	Residential_Type_Num <dbl>
1	30410	6
2	150308	3
3	100617	1
4	70721	1

5	1200107	3
6	190018	1

6 rows

To save the variable as an `RData` file, we call the `saveRDS`, specify the R object we want to save and how we name the file. Here, we save the object “First and Last Variable of dataframe df” as an RData file in the current working directory, which is callable by using `getwd()`, our file will be in the same folder (Grolemund, 2024).

```
saveRDS(first_last_var_df, file = "First and Last Variable of dataframe df.RData")
# Check the working directory
getwd()
```

```
## [1] "/Users/timmrahrt/Desktop/RMIT/Data Wrangling/Assessments/First Assessment/Dr
aft"
```

References

- Connecticut Open Data. (2024). Connecticut Open Data. [online] Available at: https://data/ct.gov (https://data/ct.gov) [Accessed 05 Jul. 2024].
- GitHub. (2024). RSocrata. [online] Available at: https://github.com/Chicago/RSocrata (https://github.com/Chicago/RSocrata) [Accessed 06 Jul. 2024].
- Rdocumentation.org. 2024. set.seed function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/simEd/versions/2.0.1/topics/set.seed (https://www.rdocumentation.org/packages/simEd/versions/2.0.1/topics/set.seed) [Accessed 06 Jul. 2024].
- Dplyr.tidyverse.org. (2024). Subset rows using their positions – slice. [online] Available at: https://dplyr.tidyverse.org/reference/slice.html# (https://dplyr.tidyverse.org/reference/slice.html#):~:text=slice_sample()%20randomly%20selects%20rows [Accessed 06 Jul. 2024].
- Rdocumentation.org. 2024. Sapply function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/memisc/versions/0.99.31.7/topics/Sapply (https://www.rdocumentation.org/packages/memisc/versions/0.99.31.7/topics/Sapply) [Accessed 07 Jul. 2024].
- Campus.datacamp.com. 2024. lapply and anonymous functions | R. [online] Available at: https://campus.datacamp.com/courses/intermediate-r/chapter-4-the-apply-family?ex=4 (https://campus.datacamp.com/courses/intermediate-r/chapter-4-the-apply-family?ex=4) [Accessed 11 Jul. 2024].
- Rdocumentation.org. 2024. ifelse function | R Documentation. [online] Available at: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/ifelse (https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/ifelse). [Accessed 11 Jul. 2024].
- Wickham, H. and Grolemund, G. (2016). R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. 1st Edition. O’Reilly Media.
- Rpubs.com. 2024. RPubs - lapply, sapply, and vapply. [online] Available at: https://rpubs.com/GilbertTeklevchiev/1113536 (https://rpubs.com/GilbertTeklevchiev/1113536)

[Accessed 12 Jul. 2024].

- Rdocumentation.org. 2024. summarise function | R Documentation. [online] Available at: https://www.rdocumentation.org/packages/dplyr/versions/0.7.8/topics/summarise (https://www.rdocumentation.org/packages/dplyr/versions/0.7.8/topics/summarise). [Accessed 12 Jul 2024].
- GeeksforGeeks. 2020. Convert Factor to Numeric and Numeric to Factor in R Programming. [online] Available at: https://www.geeksforgeeks.org/convert-factor-to-numeric-and-numeric-to-factor-in-r-programming/ (https://www.geeksforgeeks.org/convert-factor-to-numeric-and-numeric-to-factor-in-r-programming/). [Accessed 12 Jul 2024].
- Rdocumentation.org. 2024. unlist function - RDocumentation. [online] Available at: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/unlist (https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/unlist) [Accessed 15 Jul. 2024].
- Carpentries-incubator.github.io. 2024. Dataframes, Matrices, and Lists – Library Carpentry: Introduction to R and litsearchr. [online] Available at: https://carpentries-incubator.github.io/lc-litsearchr/07-data-frames-matrices-and-lists/index.html# (https://carpentries-incubator.github.io/lc-litsearchr/07-data-frames-matrices-and-lists/index.html#):~:text=The%20main%20difference%20is%20that. [Accessed 15 Jul. 2024].
- Grolemund, G. 2024. D Loading and Saving Data in R | Hands-On Programming with R. [online] rstudio-education.github.io. Available at: https://rstudio-education.github.io/hopr/dataio.html# (https://rstudio-education.github.io/hopr/dataio.html#) [Accessed 17 Jul. 2024].