Data Wrangling with Regular Expressions

Estimated time needed: **40** minutes

Lab Overview:

In the previous data collection labs, you collected some raw datasets from several different sources. In this lab, you need to perform data wrangling tasks in order to improve data quality.

You will again use regular expressions, along with the `stringr` package (part of `tidyverse`), to clean up the bike-sharing systems data that you previously web scraped from the wiki page:

https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems

Country	City	Name	System	Operator	Launched	Discontinued	Stations	Bicycles	Daily ridership
Albania	Tirana[5]	Ecovolis			March 2011		8	200	
Argentina	Mendoza[6]	Metrobici			2014		2	40	
	San Lorenzo, Santa Fe	Biciudad	Biciudad		27 November 2016		8	80	
	Buenos Aires[7][8]	Ecobici	Serttel Brasil[9]	Bike In Baires Consortium.[10]	2010		400	4000	21917
	Rosario	Mi Bici Tu Bici[11]			2 December 2015		47	480	
Australia	Melbourne[12]	Melbourne Bike Share	PBSC & 8D	Motivate	June 2010	30 November 2019[13]	53	676	
	Brisbane[14][15]	CityCycle	3 Gen. Cyclocity	JCDecaux	September 2010		150	2000	
	Melbourne	oBike	4 Gen. oBike		July 2017	July 2018	dockless	1250	
	Sydney	oBike	4 Gen. oBike		July 2017	July 2018	dockless	1250	
	Sydney	Ofo	4 Gen. Ofo		October 2017		dockless	600	
	Sydney	Reddy Go	Reddy Go		July 2017			2000	
Austria	Vienna	Citybike Wien [16]	3 Gen. Cyclocity	JCDecaux Gewista	June 2003		121	1500	2800[17]
	Burgenland	LEIHRADL nextbike	3 Gen. nextbike		2009		40		
	Lower Austria[18]	LEIHRADL nextbike	3 Gen. nextbike		2009		295	1300	
	Salzburg	nextbike	3 Gen. nextbike		2011				
	Vienna	Viennabike	2 Gen.	Association and city council	April 2002	November 2002	200	1500	
	Vorarlberg		3 Gen. nextbike		2009		14	70	
Bangladesh	Dhaka	JoBike	JoBike		2018		05	300	

One typical challenge of web scraping is that data extracted from HTML pages may contain unnecessary or inconsistently fomatted information.
For example:

- Textual annotations in numeric fields: `1000 (Updated with 1050)`
- Attached reference links: `Bike sharing system [123]`
- Inconsistent data formats: `Yes` and `Y` for the logical value `TRUE` or `2021-04-09` and `Apr 09, 2021` for the same date
- HTML style tags: `<span style="color:blue">Bike sharing system</span>`
- Special characters: ` ` for a white space

Many more such examples of noise may be encountered in real-world scraped data and most of such text related noises could be handled by regular expressions.

To summarize, you will be using `stringr` (part of `tidyverse`) and regular expressions to perform the following data wrangling tasks:

- TASK: Standardize column names for all collected datasets
- TASK: Remove undesired reference links from the scraped bike-sharing systems dataset
- TASK: Extract only the numeric value from undesired text annotations

Let's begin by importing the libraries you will use for these data wrangling tasks.

Please note that the require("tidyverse")" command is commented here as the tidyverse package is already pre-installed in this lab environment. However, if you are executing this lab local R-Studio on your system then install this package first and then load the package.

In [ ]:

```
# Check whether you need to install the `tidyverse` library
# require("tidyverse")
library(tidyverse)
```

TASK: Standardize column names for all collected datasets

In the previous data collection labs, you collected four datasets in csv format:

- `raw_bike_sharing_systems.csv` : A list of active bike-sharing systems across the world
- `raw_cities_weather_forecast.csv` : 5-day weather forecasts for a list of cities, from OpenWeather API
- `raw_worldcities.csv` : A list of major cities' info (such as name, latitude and longitude) across the world
- `raw_seoul_bike_sharing.csv` : Weather information (Temperature, Humidity, Windspeed, Visibility, Dewpoint, Solar radiation, Snowfall, Rainfall), the number of bikes rented per hour, and date information, from Seoul bike-sharing systems

Optional: If you had some difficulties finishing the data collection labs, you may download the datasets directly from the following URLs:

In [ ]:

```
# Download raw_bike_sharing_systems.csv
url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDe
download.file(url, destfile = "raw_bike_sharing_systems.csv")

# Download raw_cities_weather_forecast.csv
url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDe
download.file(url, destfile = "raw_cities_weather_forecast.csv")

# Download raw_worldcities.csv
url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDe
```

```
download.file(url, destfile = "raw_worldcities.csv")

# Download raw_seoul_bike_sharing.csv
url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDe
download.file(url, destfile = "raw_seoul_bike_sharing.csv")
```

To improve dataset readbility by both human and computer systems, we first need to standardize the column names of the datasets above using the following naming convention:

- Column names need to be UPPERCASE
- The word separator needs to be an underscore, such as in `COLUMN_NAME`

You can use the following dataset list and the `names()` function to get and set each of their column names, and convert them according to our defined naming convention.

In [ ]:
```
dataset_list <- c('raw_bike_sharing_systems.csv', 'raw_seoul_bike_sharing.csv',
```

TODO: Write a `for` loop to iterate over the above datasets and convert their column names

In [ ]:
```
for (dataset_name in dataset_list){
    # Read dataset
    dataset <- read_csv(dataset_name)
    # Standardized its columns:

    # Convert all column names to uppercase

    # Replace any white space separators by underscores, using the str_replace_a

    # Save the dataset
    write.csv(dataset, dataset_name, row.names=FALSE)
}
```

TODO: Read the resulting datasets back and check whether their column names follow the naming convention

In [ ]:
```
for (dataset_name in dataset_list){
    # Print a summary for each data set to check whether the column names were c
}
```

Process the web-scraped bike sharing system dataset

By now we have standardized all column names. Next, we will focus on cleaning up the values in the web-scraped bike sharing systems dataset.

In [ ]:
```
# First load the dataset
bike_sharing_df <- read_csv("raw_bike_sharing_systems.csv")
```

In [ ]:
```
# Print its head
head(bike_sharing_df)
```

Even from the first few rows, you can see there is plenty of undesireable embedded textual content, such as the reference link included in `Melbourne[12]`.

In this project, let's only focus on processing the following revelant columns (feel free to process the other columns for more practice):

- `COUNTRY`: Country name
- `CITY`: City name
- `SYSTEM`: Bike-sharing system name
- `BICYCLES`: Total number of bikes in the system

In [ ]:
```
# Select the four columns
sub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLE
```

Let's see the types of the selected columns

In [ ]:
```
sub_bike_sharing_df %>%
    summarize_all(class) %>%
    gather(variable, class)
```

They are all interpreted as character columns, but we expect the `BICYCLES` column to be of numeric type. Let's see why it wasn't loaded as a numeric column - possibly some entries contain characters. Let's create a simple function called `find_character` to check that.

In [ ]:
```
# grepl searches a string for non-digital characters, and returns TRUE or FALSE
# if it finds any non-digital characters, then the bicyle column is not purely n
find_character <- function(strings) grepl("[^0-9]", strings)
```

Let's try to find any elements in the `Bicycles` column containing non-numeric characters.

In [ ]:
```
sub_bike_sharing_df %>%
    select(BICYCLES) %>%
    filter(find_character(BICYCLES)) %>%
    slice(0:10)
```

As you can see, many rows have non-numeric characters, such as `32 (including 6 rollers) [162]` and `1000[253]`. This is actually very common for a table scraped from Wiki when no input validation is enforced.

Later, you will use regular expressions to clean them up.

Next, let's take a look at the other columns, namely `COUNTRY`, `CITY`, and `SYSTEM`, to see if they contain any undesired reference links, such as in `Melbourne[12]`.

In [ ]:
```
# Define a 'reference link' character class,
# `[A-z0-9]` means at least one character
# `\\[` and `\\]` means the character is wrapped by [], such as for [12] or [abc
ref_pattern <- "\\[[A-z0-9]+\\]"
find_reference_pattern <- function(strings) grepl(ref_pattern, strings)
```

In [ ]:
```
# Check whether the COUNTRY column has any reference links
sub_bike_sharing_df %>%
    select(COUNTRY) %>%
    filter(find_reference_pattern(COUNTRY)) %>%
    slice(0:10)
```

Ok, looks like the `COUNTRY` column is clean. Let's check the `CITY` column.

In [ ]:
```
# Check whether the CITY column has any reference links
sub_bike_sharing_df %>%
    select(CITY) %>%
    filter(find_reference_pattern(CITY)) %>%
    slice(0:10)
```

Hmm, looks like the `CITY` column has some reference links to be removed. Next, let's check the `SYSTEM` column.

In [ ]:
```
# Check whether the System column has any reference links
sub_bike_sharing_df %>%
    select(SYSTEM) %>%
    filter(find_reference_pattern(SYSTEM)) %>%
    slice(0:10)
```

So the `SYSTEM` column also has some reference links.

After some preliminary investigations, we identified that the `CITY` and `SYSTEM` columns have some undesired reference links, and the `BICYCLES` column has both reference links and some textual annotations.

Next, you need to use regular expressions to clean up the unexpected reference links and text annotations in numeric values.

TASK: Remove undesired reference links using regular expressions

TODO: Write a custom function using `stringr::str_replace_all` to replace all reference links with an empty character for columns `CITY` and `SYSTEM`

In [ ]:
```
# remove reference link
remove_ref <- function(strings) {
    ref_pattern <- "Define a pattern matching a reference link such as [1]"
    # Replace all matched substrings with a white space using str_replace_all()
    # Trim the reslt if you want
    # return(result)
}
```

TODO: Use the `dplyr::mutate()` function to apply the `remove_ref` function to the `CITY` and `SYSTEM` columns

In [ ]:
```
# sub_bike_sharing_df %>% mutate(column1=remove_ref(column1), ... )
```

TODO: Use the following code to check whether all reference links are removed:

In [ ]:
```
result %>%
    select(CITY, SYSTEM, BICYCLES) %>%
    filter(find_reference_pattern(CITY) | find_reference_pattern(SYSTEM) | find_
```

TASK: Extract the numeric value using regular expressions

TODO: Write a custom function using `stringr::str_extract` to extract the first digital substring match and convert it into numeric type For example, extract the value '32' from `32 (including 6 rollers) [162]`.

In [ ]:
```
# Extract the first number
extract_num <- function(columns){
    # Define a digital pattern
    digitals_pattern <- "Define a pattern matching a digital substring"
    # Find the first match using str_extract
    # Convert the result to numeric using the as.numeric() function
}
```

TODO: Use the `dplyr::mutate()` function to apply `extract_num` on the `BICYCLES` column

In [ ]:
```
# Use the mutate() function on the BICYCLES column
```

TODO: Use the summary function to check the descriptive statistics of the numeric `BICYCLES` column

In [ ]:
```
summary(result$BICYCLES)
```

TODO: Write the cleaned bike-sharing systems dataset into a csv file called `bike_sharing_systems.csv`

In [ ]:
```
# Write dataset to `bike_sharing_systems.csv`
```

References:

If you need to refresh your memory about regular expressions, please refer to this good Regular Expression cheat sheet:

Basic Regular Expressions in R

Next Steps

Great! Now you have cleaned up the bike-sharing system dataset using regular expressions. Next, you will use other `tidyverse` functions to perform data wrangling on the bike-sharing demand dataset.

Authors

Yan Luo

Other Contributors

Jeff Grossman

In [ ]:
```

```