Cleaning and Transforming Data with `pandas` and `dplyr` Libraries

Alier Reng

Saturday, September 30, 2023

Table of contents

1 Harnessing the Power of `pandas` and `dplyr` Packages

In this tutorial, we will demonstrate how to clean and transform a customer call dataset with `pandas` and `dplyr`. We obtained this dataset from Alex the Analyst's GitHub. And his tutorial video is located here!

In this revised version of the tutorial, we will incorporate improvement suggestions by Andrea Dalseno. While there are multiple ways to accomplish this task, it is always best practice to use robust approaches.

1.1 Cleaning and Transforming Customer Call Dataset

We will first transform this data with `pandas` and then with `dplyr` in the next section. We will use `pandas` method chaining.

1.2 Loading the Required Packages

Since we will be using both `R` and `Python`, we will load the `reticulate` library.

```
# Libraries
library(reticulate)
library(tidyverse)
```

Loading `Python` Libraries

```
import pandas as pd
import numpy as np
from janitor import clean_names
import re
```

1.3 Importing the dataset

In this section of the updated tutorial, we will first create a list of potential `NaN` values in case we encounter others that do not appear in our current dataset.

```
# Create a list of potential NaN values
# -------------------------------------
nan_strings = ['', '#N/A', '#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN',
]

# Loading the dataset
# -------------------
customer_raw = (
  pd.read_excel(
    '../00_data/Customer Call List.xlsx',
    na_values=nan_strings
  )
  # Clean columns names
  .clean_names()
)
```

2 Wrangling Customer Data with pandas

Now let's kick things off with the **mighty pandas** to accomplish our task. Our objective in this project is to create a working customer list. In other words, we only need to retain customers who have consented to being contacted and have a working phone number.

```
# Adjusting pandas column display option
pd.set_option("display.max_columns", None)

# Make labels - updated using Andrea's suggestion
labels = {'Y': 'Yes','YES':'Yes', 'YE':'Yes', 'N': 'No', 'NO':'No'}

# Define a function to clean and format phone numbers
def clean_phone_number(phone):
    # Convert the value to a string, and then remove non-alphanumeric characters
    #phone = re.sub(r'[^a-zA-Z0-9]', '', str(phone))
    phone = re.sub(r'\D', '', str(phone))

    # Check if the phone number has 10 digits
    if len(phone) == 10:
        # Format the phone number as xxx-xxx-xxxx
        phone = f'{phone[:3]}-{phone[3:6]}-{phone[6:]}'
    else:
        # Handle other formats or invalid phone numbers
        phone = np.nan

    return phone

# Define a function to clean and transform the address column
def clean_address(df):
    df[['street_address', 'state', 'zip_code']] = df['address'].str.split(',', n=2, expand
    return df
```

```
# Clean and transform the data
# ----------------------------
customer_df = (
  customer_raw
  # Clean and transform column values
  .assign(
    last_name=lambda x: x['last_name'].str.strip('/|...|_').str.strip(' '),
    paying_customer=lambda x: x['paying_customer'].replace(labels),
    do_not_contact=lambda x: x['do_not_contact'].replace(labels),
    phone_number=lambda x: x['phone_number'].apply(clean_phone_number)
  )
  # Split address column into: Street Address, State, and Zip Code
  .pipe(clean_address)
  # Delete unwanted column
  .drop(columns=['not_useful_column', 'address'])
  .query('~(do_not_contact == "Yes" | do_not_contact.isna()) & ~phone_number.isna()')
  .rename(columns={'customerid': 'customer_id'})
  .reset_index(drop=True)
)

# Inspecting the first 5 rows
customer_df.head()
```

```
   customer_id first_name last_name  phone_number paying_customer  \
0         1001      Frodo   Baggins  123-545-5421             Yes
1         1005        Jon      Snow  876-678-3469             Yes
2         1008   Sherlock    Holmes  876-678-3469              No
3         1010      Peter    Parker  123-545-5421             Yes
4         1013        Don    Draper  123-543-2345             Yes

  do_not_contact     street_address     state zip_code
0             No    123 Shire Lane      Shire     None
1             No  123 Dragons Road       None     None
2             No      98 Clue Drive      None     None
3             No  25th Main Street   New York     None
4             No  2039 Main Street       None     None
```

2.1 Regenerating the Same Results Using a Revised Function

```
# Revised version
# Define a function to clean last name
def clean_last_name_revised(name):
  if pd.isna(name):
    return ''
  # Remove non alphabetic characters but keeps spaces ' and -
  name = re.sub(r"[^A-Za-z\-\s']", '', name).strip()
  name = re.sub(r"\s+", " ", name)
  return name
```

```
# Clean and transform the data
# ----------------------------
customer_final = (
  customer_raw
  # Clean and transform column values
  .assign(
    last_name=lambda x: x['last_name'].apply(clean_last_name_revised),
    paying_customer=lambda x: x['paying_customer'].replace(labels),
    do_not_contact=lambda x: x['do_not_contact'].replace(labels),
    phone_number=lambda x: x['phone_number'].apply(clean_phone_number)
  )
  # Split address column into: Street Address, State, and Zip Code
  .pipe(clean_address)
  # Delete unwanted column
  .drop(columns=['not_useful_column', 'address'])
  .query('~(do_not_contact == "Yes" | do_not_contact.isna()) & ~phone_number.isna()')
  .rename(columns={'customerid': 'customer_id'})
  .reset_index(drop=True)
)

# Inspecting the first 5 rows
customer_df.head()
```

```
   customer_id first_name last_name  phone_number paying_customer  \
0         1001      Frodo   Baggins  123-545-5421             Yes
1         1005        Jon      Snow  876-678-3469             Yes
2         1008   Sherlock    Holmes  876-678-3469              No
3         1010      Peter    Parker  123-545-5421             Yes
4         1013        Don    Draper  123-543-2345             Yes

  do_not_contact     street_address      state zip_code
0             No     123 Shire Lane      Shire     None
1             No  123 Dragons Road       None     None
2             No      98 Clue Drive       None     None
3             No  25th Main Street   New York     None
4             No  2039 Main Street       None     None
```

2.2 Converting Code into a Function

Now, let's convert our code into a function and write a Python module to manage our customer call data cleaning and transformation.

```
def tweak_customer_call_data(df):

  # Define a function to clean and format phone numbers
  def clean_phone_number(phone):
    # Convert the value to a string, and then remove non-alphanumeric characters
    phone = re.sub(r'\D', '', str(phone))

    # Check if the phone number has 10 digits
    if len(phone) == 10:
        # Format the phone number as xxx-xxx-xxxx
        phone = f'{phone[:3]}-{phone[3:6]}-{phone[6:]}'
    else:
        # Handle other formats or invalid phone numbers
        phone = np.nan

    return phone


  # Define a function to clean last name
  def clean_last_name_revised(name):
    if pd.isna(name):
      return ''
    # Remove non alphabetic characters but keeps spaces ' and -
    name = re.sub(r"[^A-Za-z\-\s']", '', name).strip()
    name = re.sub(r"\s+", " ", name)
    return name

  # Clean and transform the data
  # ----------------------------
  return(
    customer_raw
    # Clean and transform column values
    .assign(
      last_name=lambda x: x['last_name'].apply(clean_last_name_revised),
      paying_customer=lambda x: x['paying_customer'].replace(labels),
      do_not_contact=lambda x: x['do_not_contact'].replace(labels),
      phone_number=lambda x: x['phone_number'].apply(clean_phone_number)
    )
    # Split address column into: Street Address, State, and Zip Code
    .pipe(clean_address)
    # Delete unwanted column
```

```
        .drop(columns=['not_useful_column', 'address'])
        .query('~(do_not_contact == "Yes" | do_not_contact.isna()) & ~phone_number.isna()')
        .rename(columns={'customerid': 'customer_id'})
        .reset_index(drop=True)
    )
```

2.3 Testing the Function

```
df = tweak_customer_call_data(customer_raw)
df.head()
```

```
   customer_id first_name last_name  phone_number paying_customer  \
0         1001      Frodo   Baggins  123-545-5421             Yes
1         1005        Jon      Snow  876-678-3469             Yes
2         1008   Sherlock    Holmes  876-678-3469              No
3         1010      Peter    Parker  123-545-5421             Yes
4         1013        Don    Draper  123-543-2345             Yes

  do_not_contact    street_address      state zip_code
0             No     123 Shire Lane      Shire     None
1             No  123 Dragons Road       None     None
2             No      98 Clue Drive       None     None
3             No  25th Main Street   New York     None
4             No  2039 Main Street       None     None
```

2.4 Importing Our New Module

Here we used `ChatGPT` to add a `docstring` to our function.

```
# Load the Module
import custopy as cy

# Test the module
customer = cy.tweak_customer_call_data(customer_raw)
```

3 Replicating the Same Task in R

Now let's turn to `dplyr` to accomplish the same task. Our objective in this project is to create a working customer list. In other words, we only need to retain customers who have consented to being contacted and have a working phone number.

```r
# Cleaning and transforming customer call dataset with dplyr
# convert a pandas DataFrame into R dataframe
pattern <- "[^A-Za-z\\-\\s']"
phone_pattern <- "[a-zA-Z\\-\\|/]"
customer_tbl <- py$customer_raw |>

  # You can include or exclude columns using the select() function.
  select(-not_useful_column) |>

  # Tidy column values
  mutate(
    last_name  = str_remove_all(last_name, pattern) |> str_trim(),
    phone_number = as.numeric(str_remove_all(phone_number, phone_pattern)),
    phone_number = str_c(str_sub(phone_number, 1, 3), "-",
                         str_sub(phone_number, 4, 6), "-",
                         str_sub(phone_number, 7, 10)
                   )
  ) |>

  # Separate address column into street address, state, and zip code
  separate_wider_delim(
    address,
    delim = ",",
    names = c("street_address", "state", "zip_code"),
    too_few = "align_start"
  ) |>

  # Modify column values
  mutate(
    paying_customer = case_when(
      paying_customer == "Y" ~ "Yes",
      paying_customer == "N" ~ "No",
      TRUE ~ paying_customer
    )
  ) |>
```

```
# Alternative method
mutate(
  do_not_contact = case_when(
    str_detect(do_not_contact, "Y") ~ "Yes",
    str_detect(do_not_contact, "N") ~ "No",
    TRUE ~ do_not_contact
  )
) |>

# Remove unwanted rows
filter(
  do_not_contact != "Yes" & !is.na(phone_number)
) |>

# Rename a column
rename(customer_id = customerid)
```

4 Closing Remarks

In this revised tutorial, we have incorporated Andrea's suggestions for best practices and code robustness. Additionally, we have made modifications to the R section of the code to reflect the same improvements as in the pandas section.We hope you will find this tutorial beneficial, and if you do, please leave us a comment and follow us @tongakuot on LinkedIn, GitHub, and YouTube.

Happy Coding!