Data Cleaning with Python and Pandas

Matt Steele

Resources

- Pandas Documentation
- O’Reilly Learning Platform

Agenda

- Entering your data [lists, dictionaries, series, data frames]
- Reading data sets and cleaning data
- Analyzing data

Recap

- Functions and Arguments
- Variables
- Run Code

Sequence Types

ordered collections of data items of the same type.

Python Documentation - Sequence Types

Tupples	A tuple is a collection which is ordered and unchan
Sets	A set is a collection which is unordered, unchangea
Dictionaries	A dictionary is a collection which is ordered*, chan

Dictionary

A dictionary is a collection which is ordered*, changeable and does not allow duplicates.

```
country_roads = {"Morgantown": 30847, "Charlestown": 45879,
"Reedsville": 603, "Huntington": 44934}

print(country_roads)
```

Data science libraries in Python

Listed below are the major libraries that provide built-in functions, methods, and constants that are important

Storage, M

- Numpy
- Pandas
- Scipy
- StatsMode

Series

Like a dictionary in the standard library, a series from Pandas allows you to store key-value pairs in python.

```
1 import pandas as pd
2
3 series_example = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
4 series_example
```

Data Frame

A **Pandas DataFrame** is a two-dimensional, size-mutable, and heterogeneous tabular data structure. It is composed of:

- **Variables** (or columns), which represent the data types.
- **Observations** (or rows), which represent individual data entries.

Each variable (column) in a DataFrame typically contains data of the same type (e.g., integers, strings). However, different variables can contain different data types.

Step 1: Create Series

- All Series must have the same size to combine
- For this example, we're creating four Series: ye

```
import pandas as pd

# Creating individual Series for each column
year = pd.Series([1977, 1980, 1983], name="Year")

title = pd.Series(["Star Wars", "Empire Strikes Back", "R
name="Title")

length = pd.Series([121, 124, 144], name="Length")

gross = pd.Series([787, 534, 572], name="Gross")
```

Step 2: Merge the Series into a data frame

```
# Creating the DataFrame
starwars_df = pd.DataFrame({
    "Year": year,
    "Title": title,
    "Length": length,
    "Gross": gross
})

# Displaying the DataFrame
print(starwars_df)
```

- After creating the Series, you can merge them
- Each Series becomes a column in the DataFra

Subsetting Variables

used to select and work with specific variables (columns) from a data frame.

```
starwars_df['Title']

mean_length = starwars_df["Length"].mean()

# Displaying the result

print("Mean Length of Movies:", mean_length)
```

Export / Save Data

Once you are done entering your data, you can export it to your working directory. The function without built-in arguments is **write.table( )** but if are saving it as a csv, you are better using **write.csv( )**.

```
# Exporting the DataFrame to a CSV file
starwars_df.to_csv("starwars.csv", index=False)
```

Importing/Reading Data

read_csv	Load delimited data from a file, URL,
read_fwf	Read data in fixed-width column form
read_excel	Read tabular data from an Excel XLS o
read_html	Read all tables found in the given HTM
read_json	Read data from a JSON (JavaScript Ob
read_sas	Read a SAS dataset stored in one of th
read_spss	Read a data file created by SPSS
read_stata	Read a data set from Stata file format
read_xml	Read a table of data from an XML file

Read Non-Proprietary Data

- Non-proprietary files, like CSV, are open formats that can be used and shared across different software platforms.
- CSV (Comma-Separated Values) files store data in a plain text format where values are separated by commas. They are commonly used because they are simple and widely supported.

```
1 import pandas as pd
2
3 reviews = pd.read_csv("customer_reviews.csv")
```

Read Proprietary Data

- Proprietary files, such as SPSS (.sav), are used by specific software systems and often require special tools to open.
- Python can handle proprietary formats like SPSS without needing the original software (SPSS), making it a versatile tool for data analysis.

```
import pandas as pd

demographics = pd.read_spss("demographics.sav")
demographics
```

View Environment

The environment in Python is where all your variables, functions, and imported libraries are stored during your current session.

```
1  %whos
```

Explore a Data Frame

syntax	example	description
.head()	reviews.head()	displays the first 5 rows of the Dat
.dtypes	reviews.dtypes	shows the data type of each colun
.info()	reviews.info()	provides a summary of the DataFr
.shape	reviews.shape	returns the dimensions of the Dat
.columns	reviews.columns	gives you a list of all column name
.describe	reviews.describe()	generates summary statistics for r
.value_counts()	reviews[“Class_Name”].value_counts()	counts the occurrences of each ur

Descriptive Statistics.

syntax	example
.count()	reviews.count()
.first(), .last()	reviews['Age'].fi
.mean(), .median()	reviews['Age'].m
.min(), .max()	reviews['Age'].m
.std(), .var()	reviews['Age'].s
	()

Cleaning Data

- Data in columns and rows are not ordered in the correct way
- Creating values or ignoring missing data
- Units are not correct or are wrong in some way
- Order of magnitude is off
- Outliers and skewing of the dataDplyr function: filter

Filter

allows you to *select rows* in your data frame that meet specific conditions or criteria in a variable

```
#find the mean of rating that people who bought from the General Department

reviews_filter = reviews[reviews["Division_Name"]== "General"]
```

Boolean Operators

boolean operators allow you to build criteria in your code

==	EQUAL
!=	NOT EQUAL
<	LESS THAN
>	GREATER THAN
<=	LESS THAN OR EQUAL
>=	GREATER THAN OR EQUAL

Filter with Boolean

let’s filter the data frame for characters who have blue eyes and were born after 50 BBY

```
reviews_filter_2 = reviews[(reviews["Division_Name"] == "General") &
(reviews["Age"] < 40)]
```

Select

allows you to *keep* or *discard* variables

```
# view

reviews.columns

# select variables

reviews_select = reviews[['Age', 'Title', 'Division_Name',
       'Department_Name', 'Class_Name']]

reviews_select
```

Assign

creates new variables in your data or *change* existing variables by performing calculations or transformations.

```
1 demographics
2
3 demographics = demographics.assign(income = (demographics["income"]/1000))
4 demographics
```

NOTE: if you name your variable as an *existing variable*, it will *overwrite* the existing variable. If you give it a *new name*, it will create a *new variable*

Recode

Transform the values of a variabl

```
# view observation categories

reviews["Recommended_IND"].value_coun

# create a new variable

reviews = reviews.assign(Recommended_
reviews["Recommended_IND"].map({0: "N

```

Recode Data Values

The **.astype** function will allow you to change the data type of a variable.

```
# view types of data values in the dataframe

reviews.dtypes

# Clothing ID should be a string
reviews['Clothing_ID'] = reviews['Clothing_ID'].astype("str")

# Recommended_IND, Division_Name, Department_Name, and Class_Name should be
reviews[['Recommended_IND', 'Division_Name', 'Department_Name', 'Class_Name
reviews[['Recommended_IND', 'Division_Name', 'Department_Name', 'Class_Name
astype("category")

# view the values

reviews.dtypes
```

Rename

Rename the column

```
reviews =
reviews.rename(columns={"Recommended_IND": "Recommended_num",
"Recommended_recode":"Recommended_label"})

reviews
```

Relocate

Move the column location in the data frame

```
reviews.columns

new_order = ['Clothing_ID', 'Age', 'Title',
             'Review_Text', 'Rating', 'Recommended_num', 'Recommended_label
             'Positive_Feedback_Count', 'Division_Name',
             'Department_Name', 'Class_Name']

reviews = reviews[new_order]
reviews
```

Sort

allows you to *sort* variables

```
reviews.sort_values(by="Rating", ascending=False)
```

Group_by & aggregate

the **group_by** function allows you to group

aggregate function allows you to get descr

```
#Groupby

class_rating = reviews[["Rating", "Class_Name"]].g
class_rating


#class_rating = reviews[["Rating", "Class_Name"]].
#class_rating

```

Missing Data

missing data in numeric fields can cause an issue when trying to calculate descriptive statistics

```
# are there missing NA values

number_missing = reviews.isna().sum()
number_missing
```

dropna

removes **all** missing data from data frames or variables

```
reviews.shape

#remove all observations with na

reviews_na = reviews.dropna()

number_missing = reviews_na.isna().sum()
number_missing
```

we can also just drop NAs from a **variable**

```
reviews_na = reviews[reviews['Title'].notna()]
reviews_na

number_missing = reviews_na.isna().sum()
number_missing
```

fillna

you can also recode the NA values for observations with *fillna*

```
reviews["Title"] = reviews["Title"].fillna("None Given")
reviews["Title"]
```

Export: to_csv

the **to_csv** function allows us to export data frames to a csv file once we are done cleaning it up or when we have done some analysis that we want to export

```
# now that we have this date frame cleaned let's save it

# let's export the file

reviews.to_csv("cleaned_reviews.csv")
```