

Data Cleaning

Master efficient workflows for cleaning real-world, messy data.

View Certificate

100% complete! Congrats!



Courses Discussions

Lessons

Tutorial Exercise

1

Handling Missing Values

Drop missing values, or fill them in with an automated workflow.



2

Scaling and Normalization

Transform numeric variables to have helpful properties.



3

Parsing Dates

Help Python recognize dates as composed of day, month, and year.



4

Character Encodings

Avoid UnicodeDecodeErrors when loading CSV files.



5

Inconsistent Data Entry

Efficiently fix typos in your data.



Builds on

[Pandas](#)

Hours to earn certificate

4 (estimated)

Cost

No cost, like all Kaggle Learn Courses

Instructor

[Rachael Tatman](#)



```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,  
# THEN FEEL FREE TO DELETE THIS CELL.  
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON  
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR  
# NOTEBOOK.  
import kagglehub  
aparnashastry_building_permit_applications_data_path = kagglehub.dataset_download('aparr  
  
print('Data source import complete.')
```

This notebook is an exercise in the [Data Cleaning](#) course. You can reference the tutorial at [this link](#).

In this exercise, you'll apply what you learned in the **Handling missing values** tutorial.

✓ Setup

The questions below will give you feedback on your work. Run the following cell to set up the feedback system.

```
from learntools.core import binder  
binder.bind(globals())  
from learntools.data_cleaning.ex1 import *  
print("Setup Complete")
```

```
🔄 /usr/local/lib/python3.10/dist-packages/learntools/data_cleaning/ex1.py:6: DtypeWa  
sf_permits = pd.read_csv("../input/building-permit-applications-data/Building_Pe  
<ipython-input-1-b6529dcc52c3>:3: DeprecationWarning: `product` is deprecated as o  
from learntools.data_cleaning.ex1 import *  
/usr/local/lib/python3.10/dist-packages/learntools/data_cleaning/ex1.py:69: Future  
_expected = sf_permits.fillna(method='bfill', axis=0).fillna(0)  
Setup Complete
```

✓ 1) Take a first look at the data

Run the next code cell to load in the libraries and dataset you'll use to complete the exercise.

```
# modules we'll use  
import pandas as pd  
import numpy as np  
  
# read in all our data  
sf_permits = pd.read_csv("../input/building-permit-applications-data/Building_Permits.
```

```
# set seed for reproducibility
np.random.seed(0)
```

```
<ipython-input-3-fa725f1adc1c>:6: DtypeWarning: Columns (22,32) have mixed types.
sf_permits = pd.read_csv("../input/building-permit-applications-data/Building_Pe
```

Use the code cell below to print the first five rows of the `sf_permits` DataFrame.

```
# TODO: Your code here!
sf_permits.head()
```

```
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1458: RuntimeW
has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1458: RuntimeW
has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
```

	Permit Number	Permit Type	Permit Type Definition	Permit Creation Date	Block	Lot	Street Number	Street Number Suffix	Street Name
0	201505065519	4	sign - erect	05/06/2015	0326	023	140	NaN	Ellis
1	201604195146	4	sign - erect	04/19/2016	0306	007	440	NaN	Geary
2	201605278609	3	additions alterations or repairs	05/27/2016	0595	203	1647	NaN	Pacific
3	201611072166	8	otc alterations permit	11/07/2016	0156	011	1230	NaN	Pacific
4	201611283529	6	demolitions	11/28/2016	0342	001	950	NaN	Market

5 rows x 43 columns

Does the dataset have any missing values? Once you have an answer, run the code cell below to get credit for your work.

```
# Check your answer (Run this code cell to receive credit!)
q1.check()
```

Correct:

The first five rows of the data does show that several columns have missing values. You can see this in the "Street Number Suffix", "Proposed Construction Type" and "Site Permit" columns, among others.

```
# Line below will give you a hint
#q1.hint()
```

Hint: Use `sf_permits.head()` to view the first five rows of the data.

✓ 2) How many missing data points do we have?

What percentage of the values in the dataset are missing? Your answer should be a number between 0 and 100. (If 1/4 of the values in the dataset are missing, the answer is 25.)

```
# TODO: Your code here!
total = np.product(sf_permits.shape)
error = sf_permits.isnull().sum().sum()
percent_missing = error / total * 100

# Check your answer
q2.check()
```

Correct

```
# Lines below will give you a hint or solution code
#q2.hint()
#q2.solution()
```

✓ 3) Figure out why the data is missing

Look at the columns **"Street Number Suffix"** and **"Zipcode"** from the [San Francisco Building Permits dataset](#). Both of these contain missing values.

- Which, if either, are missing because they don't exist?
- Which, if either, are missing because they weren't recorded?

Once you have an answer, run the code cell below.

```
# Check your answer (Run this code cell to receive credit!)
q3.check()
```

Correct:

If a value in the "Street Number Suffix" column is missing, it is likely because it does not exist. If a value in the "Zipcode" column is missing, it was not recorded.

```
# Line below will give you a hint
#q3.hint()
```

✓ 4) Drop missing values: rows

If you removed all of the rows of `sf_permits` with missing values, how many rows are left?

Note: Do not change the value of `sf_permits` when checking this.

```
# TODO: Your code here!
drop_rows = sf_permits.dropna()

print(drop_rows.head)
```

Once you have an answer, run the code cell below.

```
# Check your answer (Run this code cell to receive credit!)
q4.check()
```

Correct:

There are no rows remaining in the dataset!

```
# Line below will give you a hint
#q4.hint()
```

✓ 5) Drop missing values: columns

Now try removing all the columns with empty values.

- Create a new DataFrame called `sf_permits_with_na_dropped` that has all of the columns with empty values removed.
- How many columns were removed from the original `sf_permits` DataFrame? Use this number to set the value of the `dropped_columns` variable below.

```
# TODO: Your code here
sf_permits_with_na_dropped = sf_permits.dropna(axis=1)

dropped_columns = sf_permits.shape[1] - sf_permits_with_na_dropped.shape[1]

print("Awal %d, hilang %d" % (sf_permits.shape[1], dropped_columns))

# Check your answer
q5.check()
```

Awal 43, hilang 31

Correct

```
# Lines below will give you a hint or solution code
#q5.hint()
#q5.solution()
```

✓ 6) Fill in missing values automatically

Try replacing all the NaN's in the `sf_permits` data with the one that comes directly after it and then replacing any remaining NaN's with 0. Set the result to a new DataFrame `sf_permits_with_na_imputed`.

```
# TODO: Your code here
sf_permits_with_na_imputed = sf_permits.fillna(method='bfill', axis=0).fillna(0)

# Check your answer
q6.check()
```

```
<ipython-input-34-260846a1370e>:2: FutureWarning: DataFrame.fillna with 'method' i
sf_permits_with_na_imputed = sf_permits.fillna(method='bfill', axis=0).fillna(0)
Correct
```

```
# Lines below will give you a hint or solution code
#q6.hint()
#q6.solution()
```

Hint: Use the `.fillna()` method twice.

✓ More practice

If you're looking for more practice handling missing values:

- Check out [this notebook](#) on handling missing values using scikit-learn's imputer.
- Look back at the "Zipcode" column in the `sf_permits` dataset, which has some missing values. How would you go about figuring out what the actual zipcode of each address should be? (You might try using another dataset. You can search for datasets about San Fransisco on the [Datasets listing](#).)

Keep going

In the next lesson, learn how to [apply scaling and normalization](#) to transform your data.

Have questions or comments? Visit the [course discussion forum](#) to chat with other learners.

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,  
# THEN FEEL FREE TO DELETE THIS CELL.  
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON  
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR  
# NOTEBOOK.  
import kagglehub  
kemical_kickstarter_projects_path = kagglehub.dataset_download('kemical/kickstarter-proj  
  
print('Data source import complete.')
```

This notebook is an exercise in the [Data Cleaning](#) course. You can reference the tutorial at [this link](#).

In this exercise, you'll apply what you learned in the **Scaling and normalization** tutorial.

✓ Setup

The questions below will give you feedback on your work. Run the following cell to set up the feedback system.

```
from learntools.core import binder  
binder.bind(globals())  
from learntools.data_cleaning.ex2 import *  
print("Setup Complete")
```

 Setup Complete

✓ Get our environment set up

To practice scaling and normalization, we're going to use a [dataset of Kickstarter campaigns](#). (Kickstarter is a website where people can ask people to invest in various projects and concept products.)

The next code cell loads in the libraries and dataset we'll be using.

```
# modules we'll use  
import pandas as pd  
import numpy as np  
  
# for Box-Cox Transformation  
from scipy import stats  
  
# for min_max scaling
```



```

from mlxtend.preprocessing import minmax_scaling

# plotting modules
import seaborn as sns
import matplotlib.pyplot as plt

# read in all our data
kickstarters_2017 = pd.read_csv("../input/kickstarter-projects/ks-projects-201801.csv")

# set seed for reproducibility
np.random.seed(0)

```

Let's start by scaling the goals of each campaign, which is how much money they were asking for. After scaling, all values lie between 0 and 1.

```

# select the usd_goal_real column
original_data = pd.DataFrame(kickstarters_2017.usd_goal_real)

# scale the goals from 0 to 1
scaled_data = minmax_scaling(original_data, columns=['usd_goal_real'])

print('Original data\nPreview:\n', original_data.head())
print('Minimum value:', float(original_data.min()),
      '\nMaximum value:', float(original_data.max()))
print('_'*30)

print('\nScaled data\nPreview:\n', scaled_data.head())
print('Minimum value:', float(scaled_data.min()),
      '\nMaximum value:', float(scaled_data.max()))

```



Original data

Preview:

	usd_goal_real
0	1533.95
1	30000.00
2	45000.00
3	5000.00
4	19500.00

Minimum value: 0.01
Maximum value: 166361390.71

Scaled data

Preview:

	usd_goal_real
0	0.000009
1	0.000180
2	0.000270
3	0.000030
4	0.000117

Minimum value: 0.0
Maximum value: 1.0

<ipython-input-3-b05f42876f40>:8: FutureWarning: Calling float on a single element
 print('Minimum value:', float(original_data.min()),

```
<ipython-input-3-b05f42876f40>:9: FutureWarning: Calling float on a single element
'\nMaximum value:', float(original_data.max()))
<ipython-input-3-b05f42876f40>:13: FutureWarning: Calling float on a single element
print('Minimum value:', float(scaled_data.min())),
<ipython-input-3-b05f42876f40>:14: FutureWarning: Calling float on a single element
'\nMaximum value:', float(scaled_data.max()))
```

✓ 1) Practice scaling

We just scaled the "usd_goal_real" column. What about the "goal" column?

Begin by running the code cell below to create a DataFrame `original_goal_data` containing the "goal" column.

```
# select the usd_goal_real column
original_goal_data = pd.DataFrame(kickstarters_2017.goal)
```

Use `original_goal_data` to create a new DataFrame `scaled_goal_data` with values scaled between 0 and 1. You must use the `minmax_scaling()` function.

```
# TODO: Your code here
scaled_goal_data = minmax_scaling(original_goal_data, columns=['goal'])

# Check your answer
q1.check()
```

Correct

```
# Lines below will give you a hint or solution code
#q1.hint()
#q1.solution()
```

✓ 2) Practice normalization

Now you'll practice normalization. We begin by normalizing the amount of money pledged to each campaign.

```
# get the index of all positive pledges (Box-Cox only takes positive values)
index_of_positive_pledges = kickstarters_2017.usd_pledged_real > 0

# get only positive pledges (using their indexes)
positive_pledges = kickstarters_2017.usd_pledged_real.loc[index_of_positive_pledges]

# normalize the pledges (w/ Box-Cox)
normalized_pledges = pd.Series(stats.boxcox(positive_pledges)[0],
```

```

name='usd_pledged_real', index=positive_pledges.index)

print('Original data\nPreview:\n', positive_pledges.head())
print('Minimum value:', float(positive_pledges.min()),
      '\nMaximum value:', float(positive_pledges.max()))
print('_'*30)

print('\nNormalized data\nPreview:\n', normalized_pledges.head())
print('Minimum value:', float(normalized_pledges.min()),
      '\nMaximum value:', float(normalized_pledges.max()))

```

```

Original data
Preview:
  1    2421.0
  2    220.0
  3     1.0
  4   1283.0
  5  52375.0
Name: usd_pledged_real, dtype: float64
Minimum value: 0.45
Maximum value: 20338986.27

```

```

Normalized data
Preview:
  1    10.165143
  2     6.468598
  3     0.000000
  4     9.129277
  5    15.836853
Name: usd_pledged_real, dtype: float64
Minimum value: -0.7779954120722627
Maximum value: 30.69054040635253

```

The values have changed significantly with normalization!

In the next code cell, you'll take a look at the distribution of the normalized data, where it should now resemble a normal distribution.

```

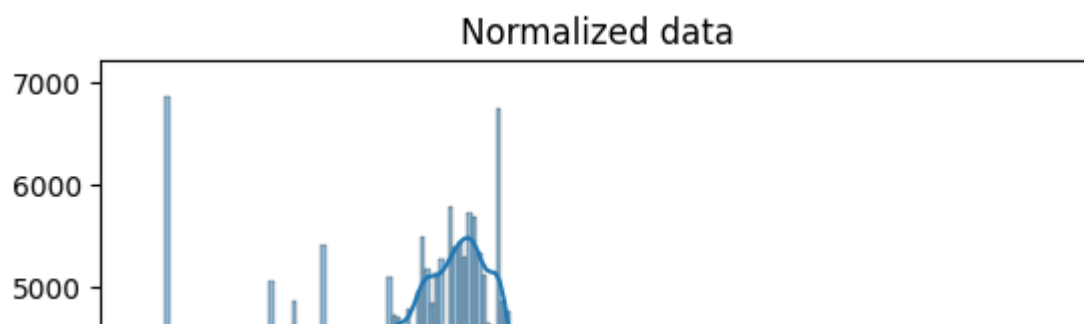
# plot normalized data
ax = sns.histplot(normalized_pledges, kde=True)
ax.set_title("Normalized data")
plt.show()

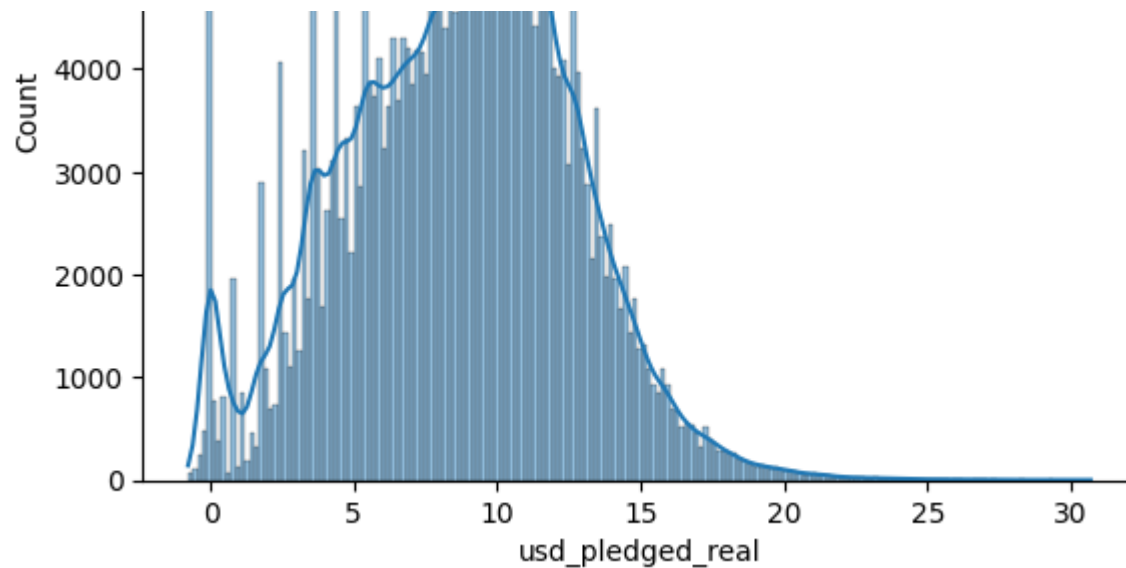
```

```

/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: u
with pd.option_context('mode.use_inf_as_na', True):

```





We used the "usd_pledged_real" column. Follow the same process to normalize the "pledged" column.

```
# normalize the pledges (w/ Box-Cox)
normalized_pledges = pd.Series(stats.boxcox(positive_pledges)[0],
                               name='pledged', index=positive_pledges.index)

print('Original data\nPreview:\n', positive_pledges.head())
print('Minimum value:', float(positive_pledges.min()),
      '\nMaximum value:', float(positive_pledges.max()))
print('_'*30)

print('\nNormalized data\nPreview:\n', normalized_pledges.head())
print('Minimum value:', float(normalized_pledges.min()),
      '\nMaximum value:', float(normalized_pledges.max()))

# plot normalized data
ax = sns.histplot(normalized_pledges, kde=True)
ax.set_title("Normalized data")
plt.show()
```

How does the normalized "usd_pledged_real" column look different from when we normalized the "pledged" column? Or, do they look mostly the same?

Once you have an answer, run the code cell below.

```
# Check your answer (Run this code cell to receive credit!)
q2.check()
```

```
# Line below will give you a hint
#q2.hint()
```

✓ (Optional) More practice

Try finding a new dataset and pretend you're preparing to perform a [regression analysis](#).

[These datasets are a good start!](#)

Pick three or four variables and decide if you need to normalize or scale any of them and, if you think you should, practice applying the correct technique.

Keep going

In the next lesson, learn how to [parse dates](#) in a dataset.

Have questions or comments? Visit the [course discussion forum](#) to chat with other learners.

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,  
# THEN FEEL FREE TO DELETE THIS CELL.  
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON  
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR  
# NOTEBOOK.  
import kagglehub  
organizations_smithsonian_volcanic_eruptions_path = kagglehub.dataset_download('organiza  
organizations_usgs_earthquake_database_path = kagglehub.dataset_download('organizations/  
  
print('Data source import complete.')
```

This notebook is an exercise in the [Data Cleaning](#) course. You can reference the tutorial at [this link](#).

In this exercise, you'll apply what you learned in the **Parsing dates** tutorial.

✓ Setup

The questions below will give you feedback on your work. Run the following cell to set up the feedback system.

```
from learntools.core import binder  
binder.bind(globals())  
from learntools.data_cleaning.ex3 import *  
print("Setup Complete")
```

↔ Setup Complete

✓ Get our environment set up

The first thing we'll need to do is load in the libraries and dataset we'll be using. We'll be working with a dataset containing information on earthquakes that occurred between 1965 and 2016.

```
# modules we'll use  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import datetime  
  
# read in our data  
earthquakes = pd.read_csv("../input/earthquake-database/database.csv")
```

```
# set seed for reproducibility
np.random.seed(0)
```

✓ 1) Check the data type of our date column

You'll be working with the "Date" column from the `earthquakes` dataframe. Investigate this column now: does it look like it contains dates? What is the dtype of the column?

```
# TODO: Your code here!
print(earthquakes['Date'])
```

```
0      01/02/1965
1      01/04/1965
2      01/05/1965
3      01/08/1965
4      01/09/1965
...
23407   12/28/2016
23408   12/28/2016
23409   12/28/2016
23410   12/29/2016
23411   12/30/2016
Name: Date, Length: 23412, dtype: object
```

Once you have answered the question above, run the code cell below to get credit for your work.

```
# Check your answer (Run this code cell to receive credit!)
q1.check()
```



Correct:

The "Date" column in the earthquakes DataFrame does have dates. The dtype is "object".

```
# Line below will give you a hint
#q1.hint()
```



Hint: Use `earthquakes['Date'].head()` to check that the column contains dates and verify that it has dtype "object". You can also use `earthquakes['Date'].dtype` to verify the dtype.

✓ 2) Convert our date columns to datetime

Most of the entries in the "Date" column follow the same format: "month/day/four-digit year". However, the entry at index 3378 follows a completely different pattern. Run the code cell below to see this.

```
earthquakes[3378:3383]
```

```
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1458: RuntimeWarning:
has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning:
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning:
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1458: RuntimeWarning:
has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning:
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning:
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
```

	Date	Time	Latitude	Longitude	Type
3378	1975-02-23T02:58:41.000Z	1975-02-23T02:58:41.000Z	8.017	124.075	Earthquake
3379	02/23/1975	03:53:36	-21.727	-71.356	Earthquake
3380	02/23/1975	07:34:11	-10.879	166.667	Earthquake
3381	02/25/1975	05:20:05	-7.388	149.798	Earthquake
3382	02/26/1975	04:48:55	85.047	97.969	Earthquake

5 rows × 21 columns

This does appear to be an issue with data entry: ideally, all entries in the column have the same format. We can get an idea of how widespread this issue is by checking the length of each entry in the "Date" column.

```
date_lengths = earthquakes.Date.str.len()
date_lengths.value_counts()
```

```
Date
10    23409
24         3
Name: count, dtype: int64
```

Looks like there are two more rows that has a date in a different format. Run the code cell below to obtain the indices corresponding to those rows and print the data.

```
indices = np.where([date_lengths == 24])[1]
print('Indices with corrupted data:', indices)
earthquakes.loc[indices]
```

```
Indices with corrupted data: [ 3378  7512 20650]
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1458: RuntimeWarning:
```



```

has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1458: RuntimeW
has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:1459: RuntimeW
has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()

```

	Date	Time	Latitude	Longitude	
3378	1975-02-23T02:58:41.000Z	1975-02-23T02:58:41.000Z	8.017	124.075	Earthq
7512	1985-04-28T02:53:41.530Z	1985-04-28T02:53:41.530Z	-32.998	-71.766	Earthq
20650	2011-03-13T02:23:34.520Z	2011-03-13T02:23:34.520Z	36.344	142.344	Earthq

3 rows × 21 columns

Given all of this information, it's your turn to create a new column "date_parsed" in the earthquakes dataset that has correctly parsed dates in it.

Note: When completing this problem, you are allowed to (but are not required to) amend the entries in the "Date" and "Time" columns. Do not remove any rows from the dataset.

```

# TODO: Your code here
earthquakes.loc[3378, "Date"] = "02/23/1975"
earthquakes.loc[7512, "Date"] = "04/28/1985"
earthquakes.loc[20650, "Date"] = "03/13/2011"

earthquakes['date_parsed'] = pd.to_datetime(earthquakes['Date'], format="%m/%d/%Y")

# Check your answer
q2.check()

```

Correct

```

# Lines below will give you a hint or solution code
#q2.hint()
#q2.solution()

```

3) Select the day of the month

Create a Pandas Series `day_of_month_earthquakes` containing the day of the month from the "date_parsed" column

the date_parsed column:

```
# try to get the day of the month from the date column
day_of_month_earthquakes = earthquakes['date_parsed'].dt.day

# Check your answer
q3.check()
```

Correct

```
# Lines below will give you a hint or solution code
#q3.hint()
#q3.solution()
```

✓ 4) Plot the day of the month to check the date parsing

Plot the days of the month from your earthquake dataset.

```
# TODO: Your code here!
print(day_of_month_earthquakes.isnull().sum())

# Tidak ada NaN/Null jadi tidak perlu dibersihkan

sns.distplot(day_of_month_earthquakes)
```

0

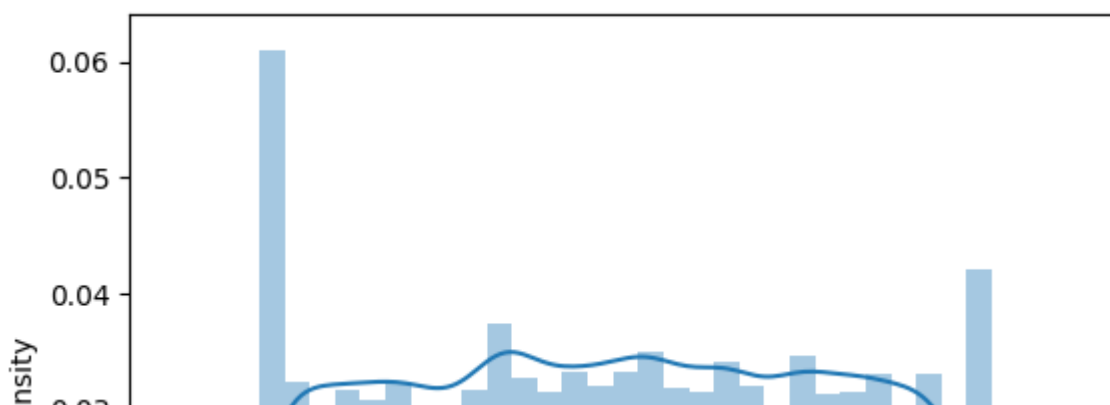
<ipython-input-26-55ba03761a25>:4: UserWarning:

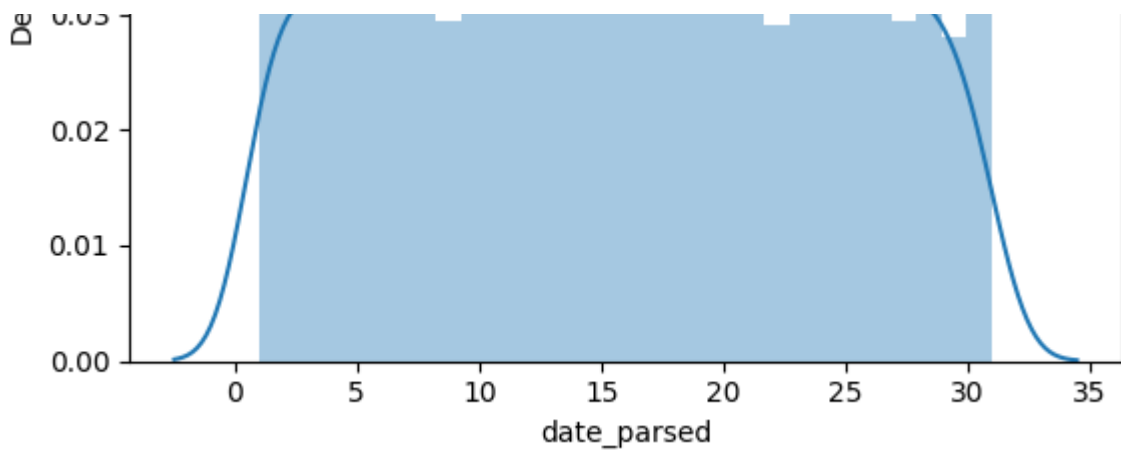
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
sns.distplot(day_of_month_earthquakes)
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: u
  with pd.option_context('mode.use_inf_as_na', True):
<Axes: xlabel='date_parsed', ylabel='Density'>
```





Does the graph make sense to you?

```
# Check your answer (Run this code cell to receive credit!)
q4.check()
```

Correct:

The graph should make sense: it shows a relatively even distribution in days of the month, which is what we would expect.

```
# Line below will give you a hint
#q4.hint()
```

✓ (Optional) Bonus Challenge

For an extra challenge, you'll work with a [Smithsonian dataset](#) that documents Earth's volcanoes and their eruptive history over the past 10,000 years

Run the next code cell to load the data.

```
volcanos = pd.read_csv("../input/volcanic-eruptions/database.csv")
```

Try parsing the column "Last Known Eruption" from the `volcanos` dataframe. This column contains a mixture of text ("Unknown") and years both before the common era (BCE, also known as BC) and in the common era (CE, also known as AD).

```
volcanos['Last Known Eruption'].sample(5)
```

```
764      Unknown
1069     1996 CE
34       1855 CE
489     2016 CE
9        1302 CE
Name: Last Known Eruption, dtype: object
```

✓ (Optional) More practice

If you're interested in graphing time series, [check out this tutorial](#).

You can also look into passing columns that you know have dates in them the `parse_dates` argument in `read_csv`. (The documentation [is here](#).) Do note that this method can be very slow, but depending on your needs it may sometimes be handy to use.

Keep going

In the next lesson, learn how to [work with character encodings](#).

Have questions or comments? Visit the [course discussion forum](#) to chat with other learners.

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,  
# THEN FEEL FREE TO DELETE THIS CELL.  
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON  
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR  
# NOTEBOOK.  
import kagglehub  
kwullum_fatal_police_shootings_in_the_us_path = kagglehub.dataset_download('kwullum/fatal-police-shootings-in-the-us')  
  
print('Data source import complete.')
```

This notebook is an exercise in the [Data Cleaning](#) course. You can reference the tutorial at [this link](#).

In this exercise, you'll apply what you learned in the **Character encodings** tutorial.

✓ Setup

The questions below will give you feedback on your work. Run the following cell to set up the feedback system.

```
from learntools.core import binder  
binder.bind(globals())  
from learntools.data_cleaning.ex4 import *  
print("Setup Complete")
```

🔄 Setup Complete

✓ Get our environment set up

The first thing we'll need to do is load in the libraries we'll be using.

```
# modules we'll use  
import pandas as pd  
import numpy as np  
  
# helpful character encoding module  
import charset_normalizer  
  
# set seed for reproducibility  
np.random.seed(0)
```

✓ 1) What are encodings?

You're working with a dataset composed of bytes. Run the code cell below to print a sample entry.

```
sample_entry = b'\xa7A\xa6n'
print(sample_entry)
print('data type:', type(sample_entry))
```

```
↗ b'\xa7A\xa6n'
  data type: <class 'bytes'>
```

You notice that it doesn't use the standard UTF-8 encoding.

Use the next code cell to create a variable `new_entry` that changes the encoding from "big5-tw" to "utf-8". `new_entry` should have the bytes datatype.

```
new_entry = sample_entry.decode("big5-tw").encode("utf-8")

# Check your answer
q1.check()
```

```
↗ Correct
```

```
# Lines below will give you a hint or solution code
#q1.hint()
#q1.solution()
```

2) Reading in files with encoding problems

Use the code cell below to read in this file at path `"../input/fatal-police-shootings-in-the-us/PoliceKillingsUS.csv"`.

Figure out what the correct encoding should be and read in the file to a DataFrame `police_killings`.

```
# TODO: Load in the DataFrame correctly.
with open("../input/fatal-police-shootings-in-the-us/PoliceKillingsUS.csv", 'rb') as r:
    result = charset_normalizer.detect(rawdata.read(30000))

print(result)

police_killings = pd.read_csv("../input/fatal-police-shootings-in-the-us/PoliceKillingsUS.csv")

# Check your answer
q2.check()
```

→▼

```
{'encoding': 'windows-1250', 'language': 'English', 'confidence': 1.0}
```

Correct

Feel free to use any additional code cells for supplemental work. To get credit for finishing this question, you'll need to run `q2.check()` and get a result of **Correct**.

```
# (Optional) Use this code cell for any additional work.  
# Saya langsung pakai cell atas
```

```
# Lines below will give you a hint or solution code  
#q2.hint()  
#q2.solution()
```

✓ 3) Saving your files with UTF-8 encoding

Save a version of the police killings dataset to CSV with UTF-8 encoding. Your answer will be marked correct after saving this file.

Note: When using the `to_csv()` method, supply only the name of the file (e.g., `"my_file.csv"`). This saves the file at the filepath `"/kaggle/working/my_file.csv"`.

```
# TODO: Save the police killings dataset to CSV  
police_killings.my_file.csv()
```

```
# Check your answer  
q3.check()
```

```
# Lines below will give you a hint or solution code  
q3.hint()  
#q3.solution()
```

Hint: Use `.to_csv()`.

✓ (Optional) More practice

Check out [this dataset of files in different character encodings](#). Can you read in all the files with their original encodings and then save them out as UTF-8 files?

If you have a file that's in UTF-8 but has just a couple of weird-looking characters in it, you can try out the [ftfy module](#) and see if it helps.

Keep going

In the final lesson, learn how to [clean up inconsistent text entries](#) in your dataset.

Have questions or comments? Visit the [course discussion forum](#) to chat with other learners.


```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,  
# THEN FEEL FREE TO DELETE THIS CELL.  
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON  
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR  
# NOTEBOOK.  
import kagglehub  
alexisbcook_pakistan_intellectual_capital_path = kagglehub.dataset_download('alexisbcook  
  
print('Data source import complete.')
```

This notebook is an exercise in the [Data Cleaning](#) course. You can reference the tutorial at [this link](#).

In this exercise, you'll apply what you learned in the **Inconsistent data entry** tutorial.

✓ Setup

The questions below will give you feedback on your work. Run the following cell to set up the feedback system.

```
from learntools.core import binder  
binder.bind(globals())  
from learntools.data_cleaning.ex5 import *  
print("Setup Complete")
```



Setup Complete

/usr/local/lib/python3.10/dist-packages/fuzzywuzzy/fuzz.py:11: UserWarning: Using
warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein

✓ Get our environment set up

The first thing we'll need to do is load in the libraries and dataset we'll be using. We use the same dataset from the tutorial.

```
# modules we'll use  
import pandas as pd  
import numpy as np  
  
# helpful modules  
import fuzzywuzzy  
from fuzzywuzzy import process  
import charset_normalizer
```

```
# read in all our data
professors = pd.read_csv("../input/pakistan-intellectual-capital/pakistan_intellectual_c

# set seed for reproducibility
np.random.seed(0)
```

Next, we'll redo all of the work that we did in the tutorial.

```
# convert to lower case
professors['Country'] = professors['Country'].str.lower()
# remove trailing white spaces
professors['Country'] = professors['Country'].str.strip()

# get the top 10 closest matches to "south korea"
countries = professors['Country'].unique()
matches = fuzzywuzzy.process.extract("south korea", countries, limit=10, scorer=fuzzyw

def replace_matches_in_column(df, column, string_to_match, min_ratio = 47):
    # get a list of unique strings
    strings = df[column].unique()

    # get the top 10 closest matches to our input string
    matches = fuzzywuzzy.process.extract(string_to_match, strings,
                                         limit=10, scorer=fuzzywuzzy.fuzz.token_sort_r

    # only get matches with a ratio > 90
    close_matches = [matches[0] for matches in matches if matches[1] >= min_ratio]

    # get the rows of all the close matches in our dataframe
    rows_with_matches = df[column].isin(close_matches)

    # replace all rows with close matches with the input matches
    df.loc[rows_with_matches, column] = string_to_match

    # let us know the function's done
    print("All done!")


replace_matches_in_column(df=professors, column='Country', string_to_match="south kore
countries = professors['Country'].unique()
```

 All done!

✓ 1) Examine another column

Write code below to take a look at all the unique values in the "Graduated from" column.

```
# TODO: Your code here
print(professors['Graduated from'].unique()[0:10]) # Hanya 10 pertama
```

 ['Asian Institute of Technology']

```
'Balochistan University of Information Technology, Engineering and Management Sci  
'University of Balochistan' "Sardar Bahadur Khan Women's University"  
'SRH Hochschule Heidelberg'  
'Institute of Business Administration,Karachi' 'DUET,Karachi'  
'University of Turbat' 'University of Vienna' 'Monash University']
```

Do you notice any inconsistencies in the data? Can any of the inconsistencies in the data be fixed by removing white spaces at the beginning and end of cells?

Once you have answered these questions, run the code cell below to get credit for your work.

```
# Check your answer (Run this code cell to receive credit!)  
q1.check()
```

Correct:

There are inconsistencies that can be fixed by removing white spaces at the beginning and end of cells. For instance, "University of Central Florida" and " University of Central Florida" both appear in the column.

```
# Line below will give you a hint  
#q1.hint()
```

✓ 2) Do some text pre-processing

Convert every entry in the "Graduated from" column in the `professors` DataFrame to remove white spaces at the beginning and end of cells.

```
# TODO: Your code here  
professors['Graduated from'] = professors['Graduated from'].str.strip()  
  
# Check your answer  
q2.check()
```

Correct

```
# Lines below will give you a hint or solution code  
#q2.hint()  
#q2.solution()
```

✓ 3) Continue working with countries

In the tutorial, we focused on cleaning up inconsistencies in the "Country" column. Run the code cell below to view the list of unique values that we ended with.

```
# get all the unique values in the 'City' column
```

```
# get all the unique values in the "City" column
countries = professors['Country'].unique()

# sort them alphabetically and then take a closer look
countries.sort()
countries

array(['australia', 'austria', 'canada', 'china', 'finland', 'france',
      'germany', 'greece', 'hongkong', 'ireland', 'italy', 'japan',
      'macau', 'malaysia', 'mauritius', 'netherlands', 'new zealand',
      'norway', 'pakistan', 'portugal', 'russian federation',
      'saudi arabia', 'scotland', 'singapore', 'south korea', 'spain',
      'sweden', 'thailand', 'turkey', 'uk', 'urbana', 'usa', 'usofa'],
      dtype=object)
```

Take another look at the "Country" column and see if there's any more data cleaning we need to do.

It looks like 'usa' and 'usofa' should be the same country. Correct the "Country" column in the dataframe to replace 'usofa' with 'usa'.

Use the most recent version of the DataFrame (with the whitespaces at the beginning and end of cells removed) from question 2.

```
# TODO: Your code here!
country_list_fuzzy = fuzzywuzzy.process.extract("usa", countries, limit=10, scorer=fuz
print(country_list_fuzzy)

# replace_matches_in_column(df=professors, column="Country", string_to_match="usa", mi
replace_matches_in_column(professors, "Country", "usa", 70)

# Check your answer
q3.check()
```

```
[('usa', 100), ('usofa', 75), ('austria', 60), ('australia', 50), ('spain', 50), (
All done!
```

```
Incorrect: Incorrect value for professors
```

```
# Lines below will give you a hint or solution code
q3.hint()
#q3.solution()
```

Hint: Use the `replace_matches_in_column()` function defined above.

Solution:

```
matches = fuzzywuzzy.process.extract("usa", countries, limit=10, scorer=fuzzywuzzy
replace_matches_in_column(df=professors, column='Country', string_to_match="usa",
```

✓ **Congratulations!**

Congratulations for completing the **Data Cleaning** course on Kaggle Learn!

To practice your new skills, you're encouraged to download and investigate some of [Kaggle's Datasets](#).

Have questions or comments? Visit the [course discussion forum](#) to chat with other learners.