8.3.4

Create the Campaign DataFrame

Two DataFrames are complete, and you have two more to go. Now, you and Britta are ready to dive in and restructure the data to create the campaign DataFrame.

First, you need to make a plan.

To do so, you review the business requirements. For the `campaign_df` DataFrame, you need the following columns:

- The "cf_id" column from the `crowdfunding_info` worksheet.
- A column named "contact_id" that contains the unique "contact_id" values from the `contact_info` worksheet.
- The "company_name" column from the `crowdfunding_info` worksheet.
- The "blurb" column from the `crowdfunding_info` worksheet, renamed as the "description" column.
- The "goal" column from the `crowdfunding_info` worksheet, converted to a floating-point data type.
- The "pledged" column from the `crowdfunding_info` worksheet, converted to a floating-point data type.
- The "outcome" column from the `crowdfunding_info` worksheet.
- The "backers_count" column from the `crowdfunding_info` worksheet.
- The "country" column from the `crowdfunding_info` worksheet.
- The "currency" column from the `crowdfunding_info` worksheet.

- The "launched_at" column from the `crowdfunding_info` worksheet, renamed as the "launch_date" column and converted to a `datetime` format.
- The "deadline" column from the `crowdfunding_info` worksheet, renamed as the "end_date" column and converted to a `datetime` format.
- A "category_id" column that contains numbers matching the unique "category_id" values from the `category_df` DataFrame.
- A "subcategory_id" column that contains numbers matching the unique "subcategory_id" values from the `subcategory_df` DataFrame.

Clean the Campaign Data

Now, it's time to execute the plan.

Inspect
Plan
Execute

First, we'll make a copy of the `crowdfunding_info_df` DataFrame and name it `campaign_df`. This way, we keep the `crowdfunding_info_df` DataFrame intact for future use and eliminate the `SettingWithCopyWarning` when we perform operations.

To do so, run the following code:

```
# Create a copy of the crowdfunding_info_df DataFrame name campaign_df.
campaign_df = crowdfunding_info_df.copy()
campaign_df.head()
```

Next, rename the "blurb", "launched_at", and "deadline" columns in the `campaign_df` DataFrame.

Before moving on, check your knowledge in the following assessment:

HIDE HINT

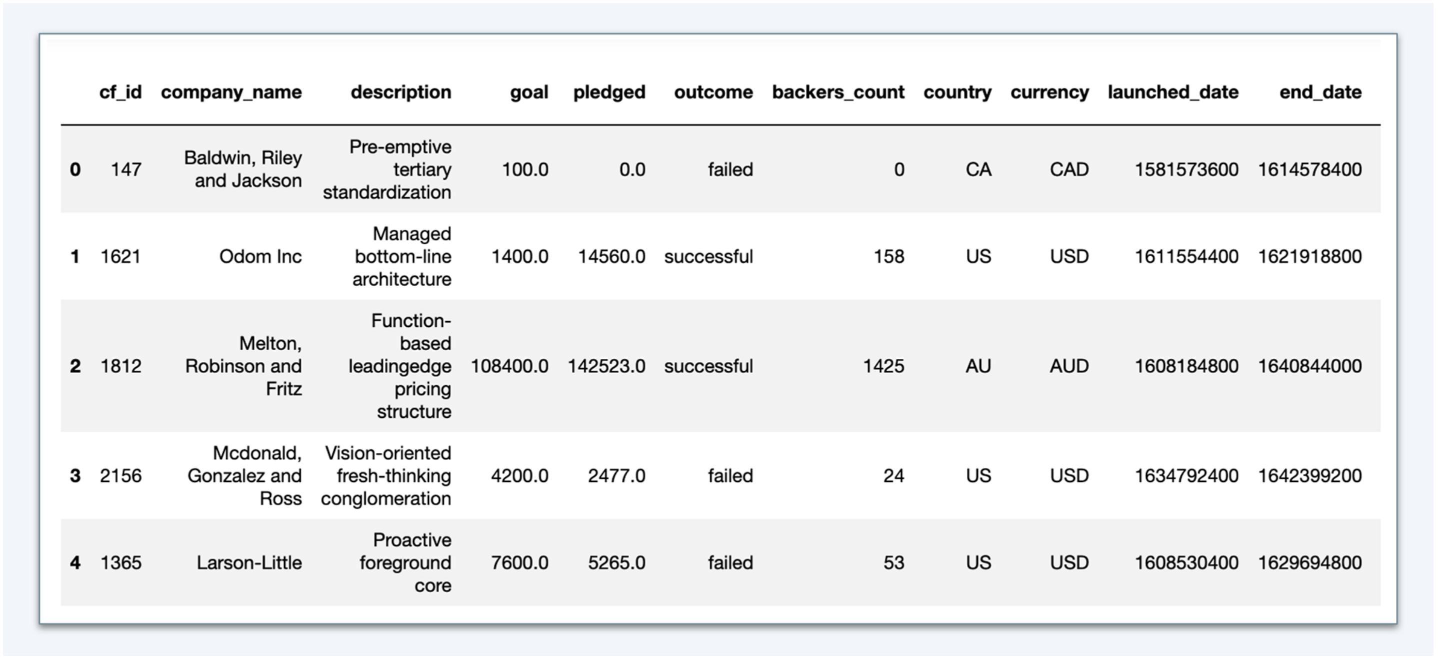
To rename the columns, you can use the following code:

```
.
escription', 'launched_at': 'launched_date', 'deadline': 'end_date'})
```

Then, change the "goal" and "pledged" columns from the `int64` data type to `float` by running the following code:

```
# Convert the goal and pledged columns to a `float` data type.
campaign_df[["goal","pledged"]] = campaign_df[["goal","pledged"]].astype(
campaign_df.head()
```

Next, confirm that your `campaign_df` DataFrame matches the following image. The "blurb", "launched_at", and "deadline" columns are renamed to "description", "launched_date", and "end_date". The "goal" and "pledged" columns are floating-point decimal numbers.



	cf_id	company_name	description	goal	pledged	outcome	backers_count	country	currency	launched_date	end_date
0	147	Baldwin, Riley and Jackson	Pre-emptive tertiary standardization	100.0	0.0	failed	0	CA	CAD	1581573600	1614578400
1	1621	Odom Inc	Managed bottom-line architecture	1400.0	14560.0	successful	158	US	USD	1611554400	1621918800
2	1812	Melton, Robinson and Fritz	Function-based leadingedge pricing structure	108400.0	142523.0	successful	1425	AU	AUD	1608184800	1640844000
3	2156	Mcdonald, Gonzalez and Ross	Vision-oriented fresh-thinking conglomeration	4200.0	2477.0	failed	24	US	USD	1634792400	1642399200
4	1365	Larson-Little	Proactive foreground core	7600.0	5265.0	failed	53	US	USD	1608530400	1629694800

Next, you need to format the "launched_date" and "end_date" columns to a `datetime` format. These dates are currently in epoch time.

REWIND

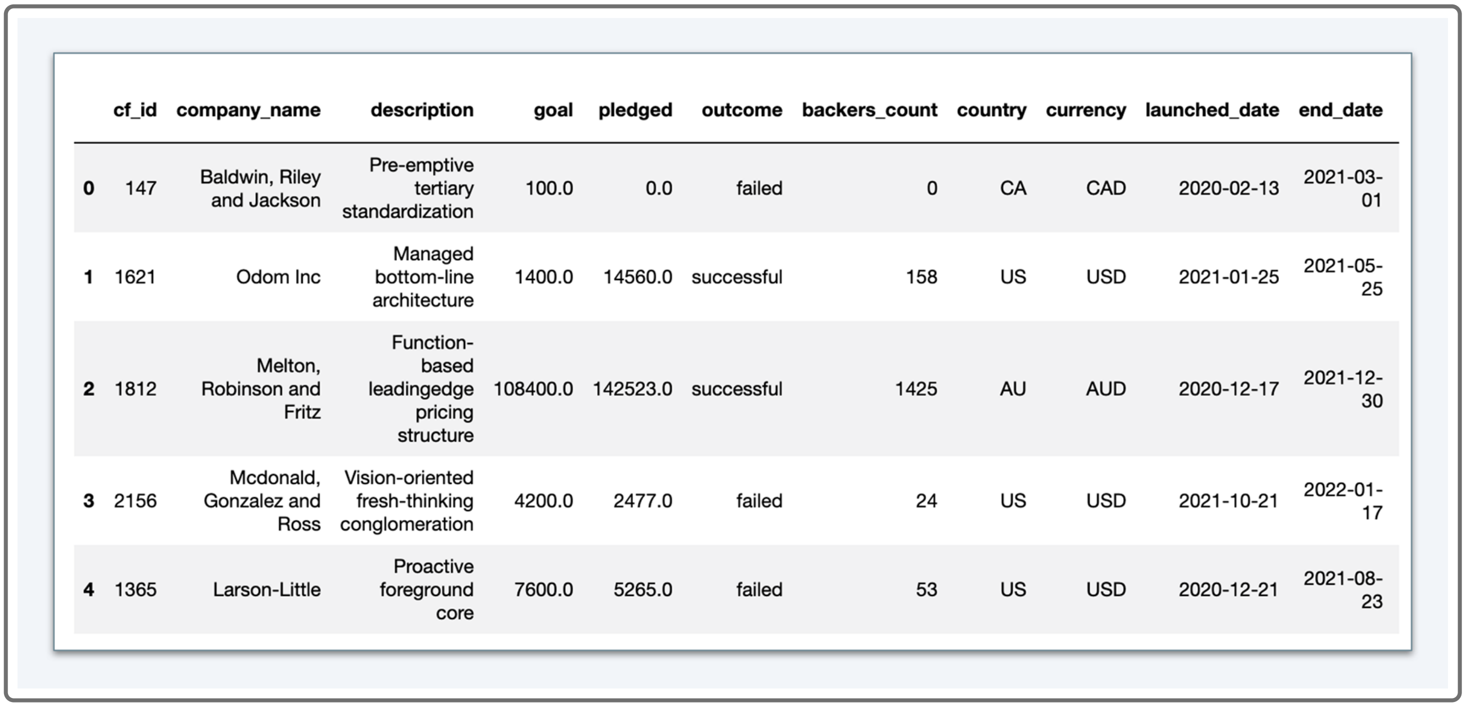
To convert a epoch timestamp to the International Organization for Standardization (ISO) format, or YYYY-MM-DD, we need to use the Python `datetime` module and then convert the time to a string format using, `'%Y-%m-%d'`.

To do so, run the following code:

```
# Format the launched_date and end_date columns to datetime format.
from datetime import datetime as dt
campaign_df["launched_date"] = pd.to_datetime(campaign_df["launched_date"
campaign_df["end_date"] = pd.to_datetime(campaign_df["end_date"], unit='s
campaign_df.head()
```

Let's review the preceding code. First, we import the `datetime` module from the datetime library. Then, we change the data type for the "launched_date" and "end_date" columns to `datetime` by using the `pd.to_datetime` method. And, we convert the format to YYYY-MM-DD by using `.dt.strftime('%Y-%m-%d')`.

The following image shows the output from running the preceding code:



	cf_id	company_name	description	goal	pledged	outcome	backers_count	country	currency	launched_date	end_date
0	147	Baldwin, Riley and Jackson	Pre-emptive tertiary standardization	100.0	0.0	failed	0	CA	CAD	2020-02-13	2021-03-01
1	1621	Odom Inc	Managed bottom-line architecture	1400.0	14560.0	successful	158	US	USD	2021-01-25	2021-05-25
2	1812	Melton, Robinson and Fritz	Function-based leadingedge pricing structure	108400.0	142523.0	successful	1425	AU	AUD	2020-12-17	2021-12-30
3	2156	Mcdonald, Gonzalez and Ross	Vision-oriented fresh-thinking conglomeration	4200.0	2477.0	failed	24	US	USD	2021-10-21	2022-01-17
4	1365	Larson-Little	Proactive foreground core	7600.0	5265.0	failed	53	US	USD	2020-12-21	2021-08-23

Transform the Campaign Data

Next, we want to assign each category and subcategory in the `campaign_df` DataFrame the unique "category_id" and "subcategory_id" number from the `category_df` and `subcategory_df` DataFrame, respectively.

Because the `category_df` and `subcategory_df` DataFrames have the same columns in the `campaign_df` DataFrame, we can merge both of them with the `campaign_df` DataFrame.

Recall how to merge two DataFrames on similar columns.

HIDE HINT

To merge two DataFrames on similar columns, we use the following code:

`df1.merge(df2, on='column',how=left).`

Practice doing this yourself in the following Skill Drill:

SKILL DRILL

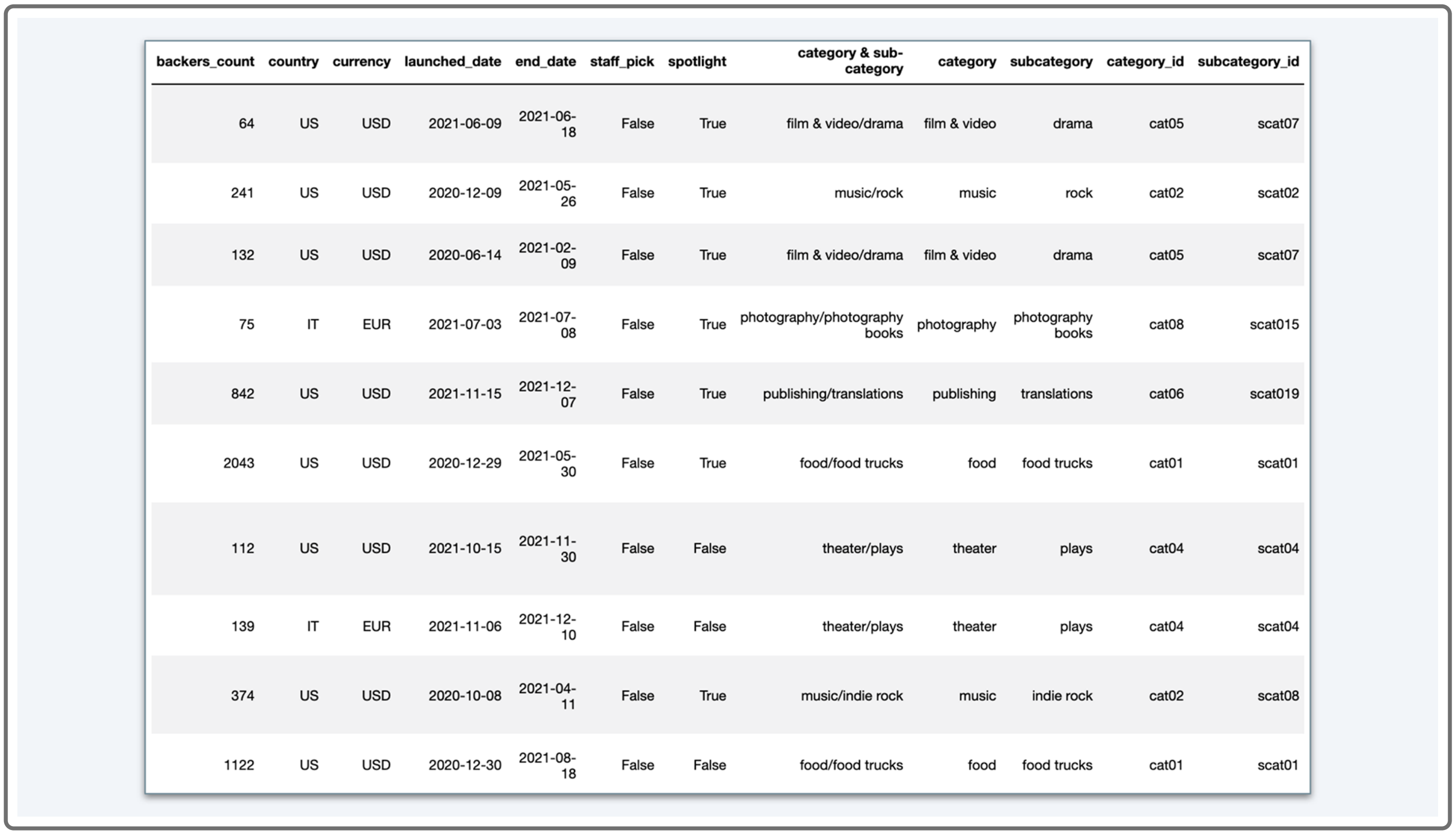
Merge the `category_df` DataFrame with the `campaign_df` DataFrame on the "category" column so that the "category_id" column of the `category_df` DataFrame is the last column in the `campaign_df` DataFrame.

Next, you'll execute this part of the plan.

Instead of creating and running two separate scripts to merge the `category_df` and `subcategory_df` DataFrames with the `campaign_df` DataFrame, run the following single script:

```
# Merge the campaign_df with the category_df on the "category" column and
# the subcategory_df on the "subcategory" column.
campaign_merged_df = campaign_df.merge(category_df, on='category', how='l
campaign_merged_df.tail(10)
```

The following image shows the output from running the preceding code:



backers_count	country	currency	launched_date	end_date	staff_pick	spotlight	category & sub-category	category	subcategory	category_id	subcategory_id
64	US	USD	2021-06-09	2021-06-18	False	True	film & video/drama	film & video	drama	cat05	scat07
241	US	USD	2020-12-09	2021-05-26	False	True	music/rock	music	rock	cat02	scat02
132	US	USD	2020-06-14	2021-02-09	False	True	film & video/drama	film & video	drama	cat05	scat07
75	IT	EUR	2021-07-03	2021-07-08	False	True	photography/photography books	photography	photography books	cat08	scat015
842	US	USD	2021-11-15	2021-12-07	False	True	publishing/translations	publishing	translations	cat06	scat019
2043	US	USD	2020-12-29	2021-05-30	False	True	food/food trucks	food	food trucks	cat01	scat01
112	US	USD	2021-10-15	2021-11-30	False	False	theater/plays	theater	plays	cat04	scat04
139	IT	EUR	2021-11-06	2021-12-10	False	False	theater/plays	theater	plays	cat04	scat04
374	US	USD	2020-10-08	2021-04-11	False	True	music/indie rock	music	indie rock	cat02	scat08
1122	US	USD	2020-12-30	2021-08-18	False	False	food/food trucks	food	food trucks	cat01	scat01

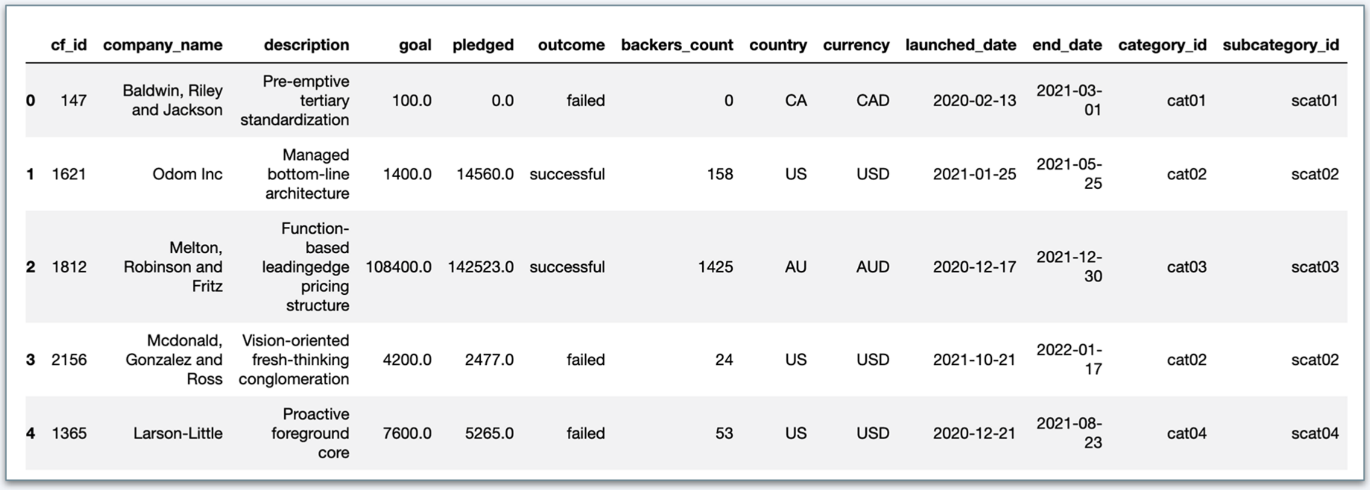
In the preceding image, the "category" and "subcategory" columns in the `campaign_df` DataFrame have been assigned their respective category and subcategory IDs from the `category_df` and `subcategory_df` DataFrames.

Next, let's drop the "category" and "subcategory" columns and any additional unwanted columns.

To do so, run the following code:

```
# Drop unwanted columns.
campaign_cleaned = campaign_merged_df.drop(['staff_pick', 'spotlight', 'c
campaign_cleaned.head()
```

The following image shows the output from running the preceding code:



	cf_id	company_name	description	goal	pledged	outcome	backers_count	country	currency	launched_date	end_date	category_id	subcategory_id
0	147	Baldwin, Riley and Jackson	Pre-emptive tertiary standardization	100.0	0.0	failed	0	CA	CAD	2020-02-13	2021-03-01	cat01	scat01
1	1621	Odom Inc	Managed bottom-line architecture	1400.0	14560.0	successful	158	US	USD	2021-01-25	2021-05-25	cat02	scat02
2	1812	Melton, Robinson and Fritz	Function-based leadingedge pricing structure	108400.0	142523.0	successful	1425	AU	AUD	2020-12-17	2021-12-30	cat03	scat03
3	2156	Mcdonald, Gonzalez and Ross	Vision-oriented fresh-thinking conglomeration	4200.0	2477.0	failed	24	US	USD	2021-10-21	2022-01-17	cat02	scat02
4	1365	Larson-Little	Proactive foreground core	7600.0	5265.0	failed	53	US	USD	2020-12-21	2021-08-23	cat04	scat04

In the preceding image, the `campaign_cleaned` DataFrame is almost complete. It has all the necessary columns, except the "contact_id" column.

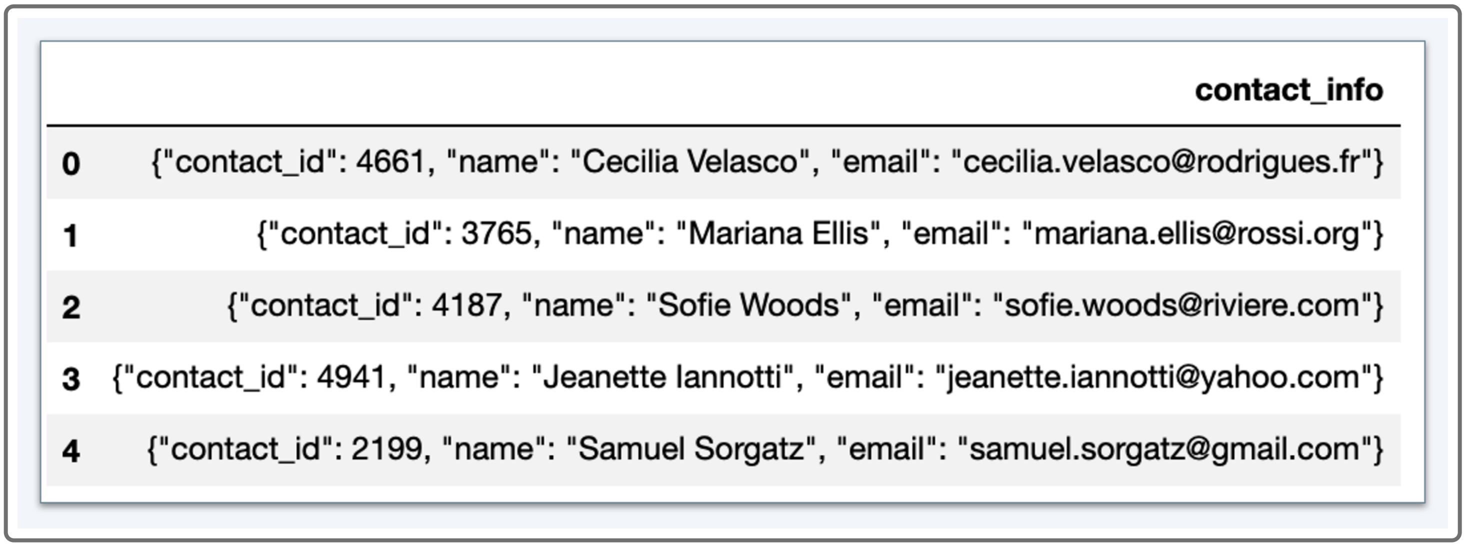
The last step is to add the unique four-digit contact ID number from the `contact_info_df` DataFrame.

Let's inspect the `contact_info_df` DataFrame more closely to find out how to get the unique "contact_id" value from the "contact_info" column in the `contact_info_df` DataFrame.

To do so, run the following code to display the `contact_info_df` DataFrame:

```
# Show the contact_info_df DataFrame.
contact_info_df.head()
```

The following image shows the output from running the preceding code:



```
                                                                           contact_info
0       {"contact_id": 4661, "name": "Cecilia Velasco", "email": "cecilia.velasco@rodrigues.fr"}
1                {"contact_id": 3765, "name": "Mariana Ellis", "email": "mariana.ellis@rossi.org"}
2              {"contact_id": 4187, "name": "Sofie Woods", "email": "sofie.woods@riviere.com"}
3  {"contact_id": 4941, "name": "Jeanette Iannotti", "email": "jeanette.iannotti@yahoo.com"}
4        {"contact_id": 2199, "name": "Samuel Sorgatz", "email": "samuel.sorgatz@gmail.com"}
```

In the preceding image, notice that there is one column name, "contact_info", and each row matches the data in the `contact_info` worksheet from the `crowdfunding.xlsx` file.

REWIND

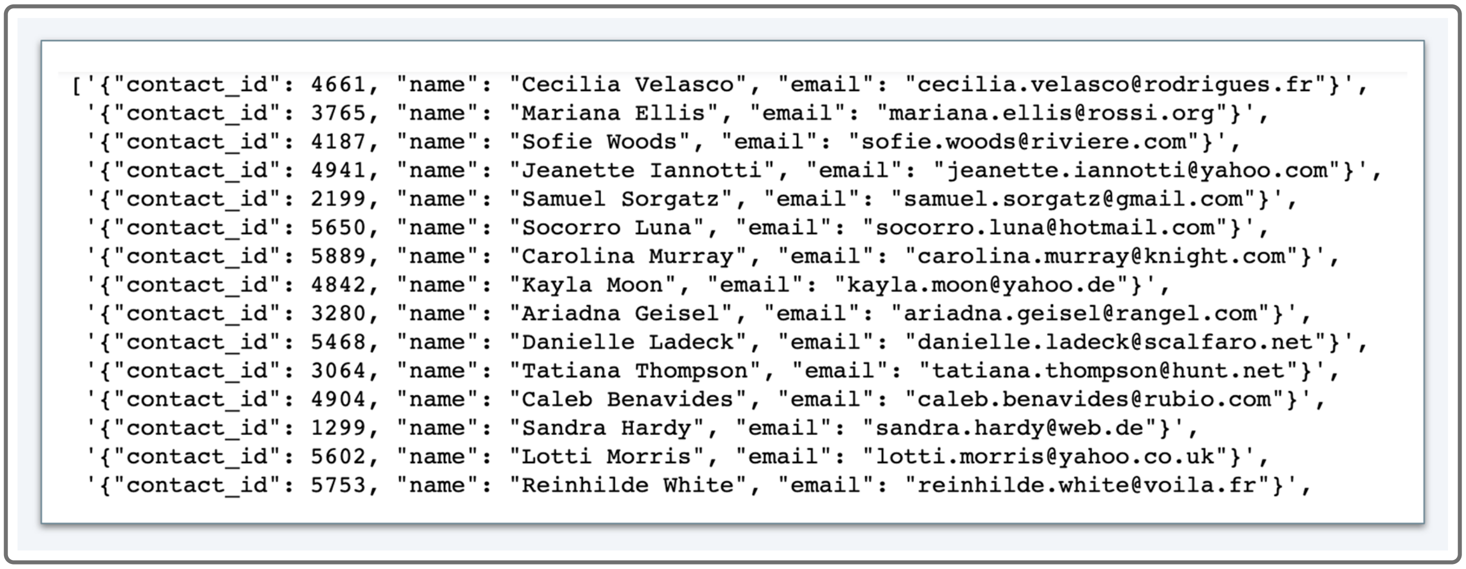
Recall that using the `info` method on the `contact_info_df` DataFrame returned 1,000 non-null rows in one column with the `object` data type. That means that each row is a string.

Next, create a list of the values from the "contact_info" column by running the following code:

```
# Retrieve the data from the "contact_info" column.
contact_info_list = contact_info_df.contact_info.to_list()
```

```
contact_info_list
```

The following image shows the output from running the preceding code:



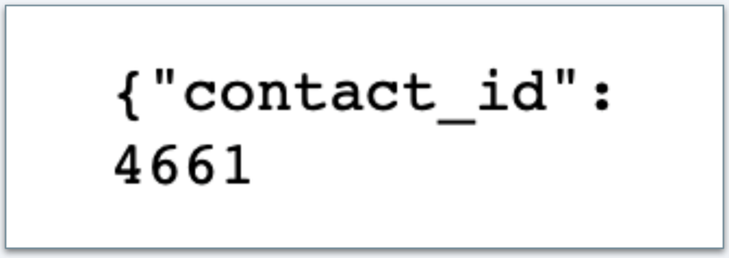
```
['{"contact_id": 4661, "name": "Cecilia Velasco", "email": "cecilia.velasco@rodrigues.fr"}',
 '{"contact_id": 3765, "name": "Mariana Ellis", "email": "mariana.ellis@rossi.org"}',
 '{"contact_id": 4187, "name": "Sofie Woods", "email": "sofie.woods@riviere.com"}',
 '{"contact_id": 4941, "name": "Jeanette Iannotti", "email": "jeanette.iannotti@yahoo.com"}',
 '{"contact_id": 2199, "name": "Samuel Sorgatz", "email": "samuel.sorgatz@gmail.com"}',
 '{"contact_id": 5650, "name": "Socorro Luna", "email": "socorro.luna@hotmail.com"}',
 '{"contact_id": 5889, "name": "Carolina Murray", "email": "carolina.murray@knight.com"}',
 '{"contact_id": 4842, "name": "Kayla Moon", "email": "kayla.moon@yahoo.de"}',
 '{"contact_id": 3280, "name": "Ariadna Geisel", "email": "ariadna.geisel@rangel.com"}',
 '{"contact_id": 5468, "name": "Danielle Ladeck", "email": "danielle.ladeck@scalfaro.net"}',
 '{"contact_id": 3064, "name": "Tatiana Thompson", "email": "tatiana.thompson@hunt.net"}',
 '{"contact_id": 4904, "name": "Caleb Benavides", "email": "caleb.benavides@rubio.com"}',
 '{"contact_id": 1299, "name": "Sandra Hardy", "email": "sandra.hardy@web.de"}',
 '{"contact_id": 5602, "name": "Lotti Morris", "email": "lotti.morris@yahoo.co.uk"}',
 '{"contact_id": 5753, "name": "Reinhilde White", "email": "reinhilde.white@voila.fr"}',
```

By scrolling through the list or observing the preceding image, notice that in each row, a four-digit number follows the `{"contact_id" :` Python string.

We can use list slicing to retrieve the four-digit number. To do so, run the following code:

```
# Pop out the unique identification number using list slicing.
print(contact_info_list[0][:15])
print(contact_info_list[0][15:19])
```

The following image shows the output from running the preceding code:



```
{"contact_id":
4661
```

Next, we'll make a plan for how to add the contact ID numbers to the campaign DataFrame.

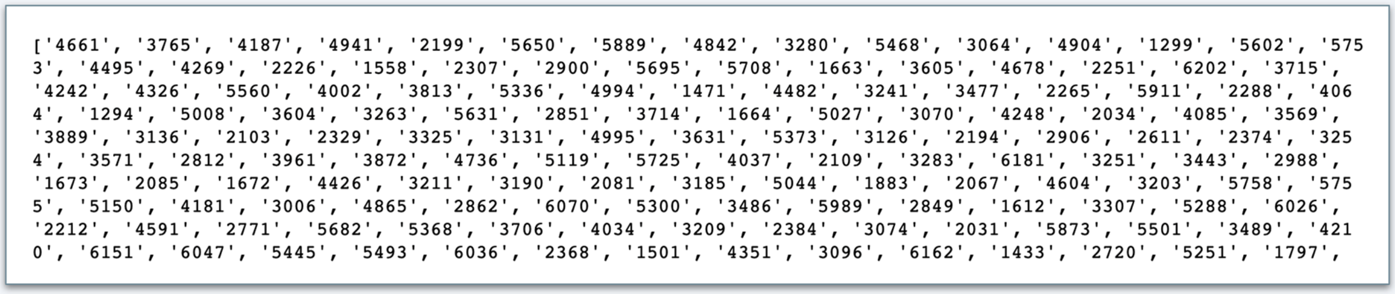
First, we can iterate through `contact_info_list`, get the four-digit contact ID number by using list slicing and add it to a list. Then, we can then add the list as a new column to the `campaign_cleaned` DataFrame. And, we can do this with a list comprehension.

We'll now execute the plan.

Run the following code to get the four-digit contact ID number:

```
# Pop out the unique identification number using list comprehension and s
print([x[15:19] for x in contact_info_list])
```

The following image shows the output from running the preceding code:



```
['4661', '3765', '4187', '4941', '2199', '5650', '5889', '4842', '3280', '5468', '3064', '4904', '1299', '5602', '575
3', '4495', '4269', '2226', '1558', '2307', '2900', '5695', '5708', '1663', '3605', '4678', '2251', '6202', '3715',
'4242', '4326', '5560', '4002', '3813', '5336', '4994', '1471', '4482', '3241', '3477', '2265', '5911', '2288', '406
4', '1294', '5008', '3604', '3263', '5631', '2851', '3714', '1664', '5027', '3070', '4248', '2034', '4085', '3569',
'3889', '3136', '2103', '2329', '3325', '3131', '4995', '3631', '5373', '3126', '2194', '2906', '2611', '2374', '325
4', '3571', '2812', '3961', '3872', '4736', '5119', '5725', '4037', '2109', '3283', '6181', '3251', '3443', '2988',
'1673', '2085', '1672', '4426', '3211', '3190', '2081', '3185', '5044', '1883', '2067', '4604', '3203', '5758', '575
5', '5150', '4181', '3006', '4865', '2862', '6070', '5300', '3486', '5989', '2849', '1612', '3307', '5288', '6026',
'2212', '4591', '2771', '5682', '5368', '3706', '4034', '3209', '2384', '3074', '2031', '5873', '5501', '3489', '421
0', '6151', '6047', '5445', '5493', '6036', '2368', '1501', '4351', '3096', '6162', '1433', '2720', '5251', '1797',
```

In the preceding image, notice that the four-digit number from each row is captured in a list

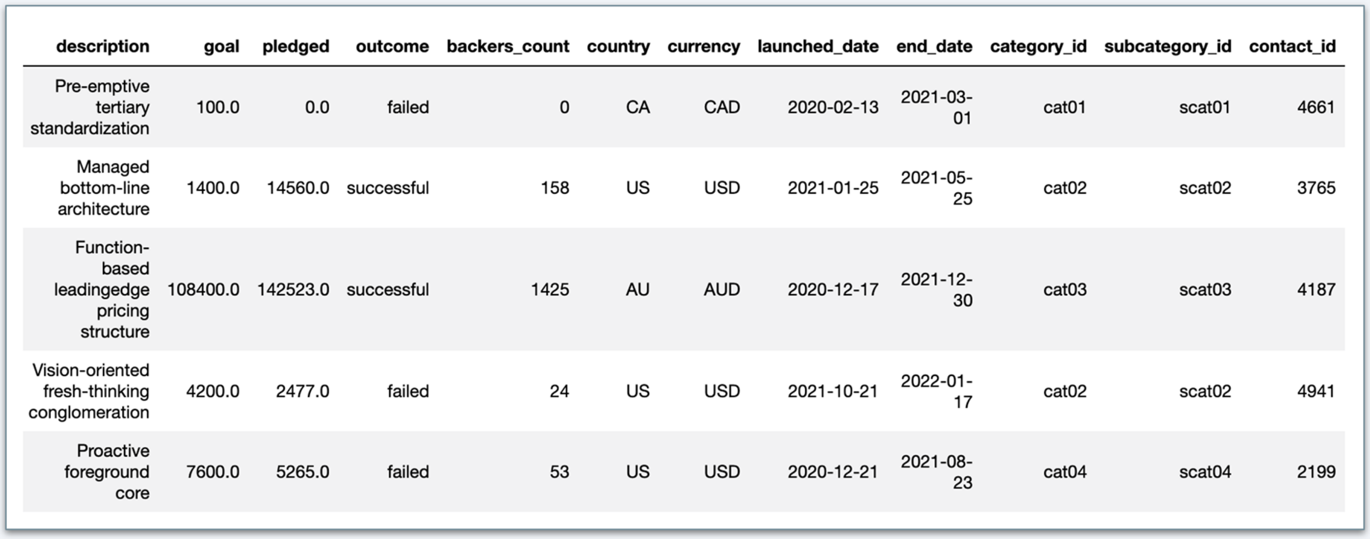
Transform and Clean the Campaign Data

Now, let's modify the list comprehension to get the four-digit numbers directly from the `contact_info_df` DataFrame and then add them as a new column to the `campaign_cleaned` DataFrame.

To do so, run the following code:

```
# Extract the four-digit contact ID number and add it to a new column in
campaign_cleaned["contact_id"] = [x[15:19] for x in contact_info_df["cont
campaign_cleaned.head()
```

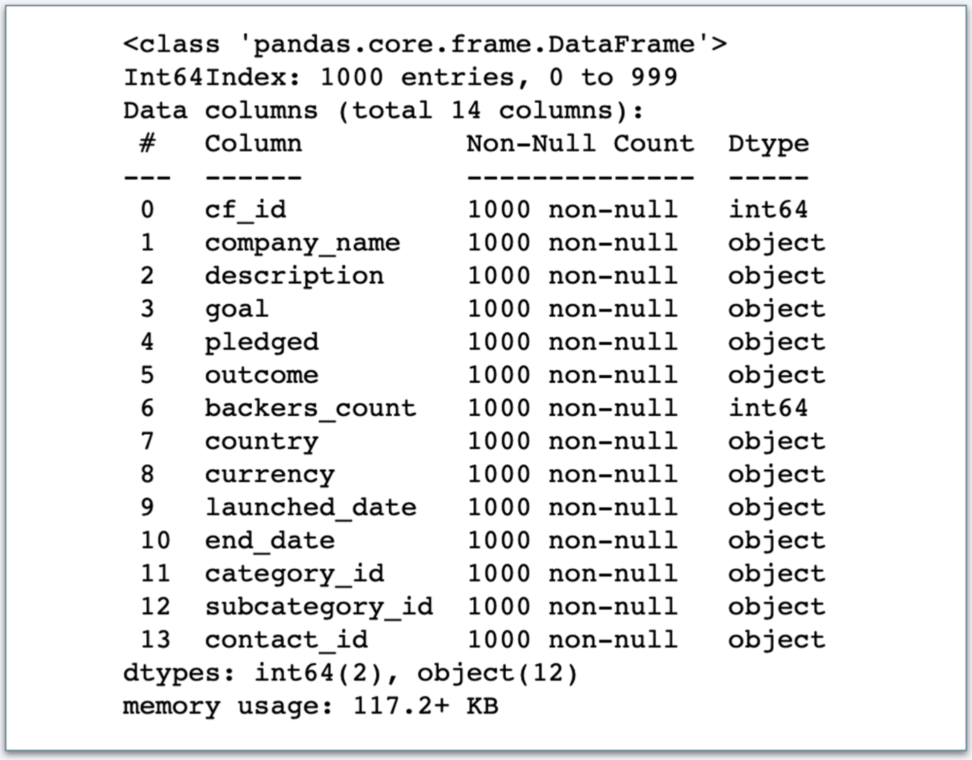
The following image shows the output from running the preceding code, which we can also observe by scrolling to the end of the columns in the `campaign_cleaned` DataFrame:



description	goal	pledged	outcome	backers_count	country	currency	launched_date	end_date	category_id	subcategory_id	contact_id
Pre-emptive tertiary standardization	100.0	0.0	failed	0	CA	CAD	2020-02-13	2021-03-01	cat01	scat01	4661
Managed bottom-line architecture	1400.0	14560.0	successful	158	US	USD	2021-01-25	2021-05-25	cat02	scat02	3765
Function-based leadingedge pricing structure	108400.0	142523.0	successful	1425	AU	AUD	2020-12-17	2021-12-30	cat03	scat03	4187
Vision-oriented fresh-thinking conglomeration	4200.0	2477.0	failed	24	US	USD	2021-10-21	2022-01-17	cat02	scat02	4941
Proactive foreground core	7600.0	5265.0	failed	53	US	USD	2020-12-21	2021-08-23	cat04	scat04	2199

In the preceding image, notice that the four-digit numbers have been added as a new column at the end of the `campaign_cleaned` DataFrame.

If we inspect the `campaign_cleaned` DataFrame by using the `info` method, we find that the "contact_id" column has the `object` data type, as the following image shows:



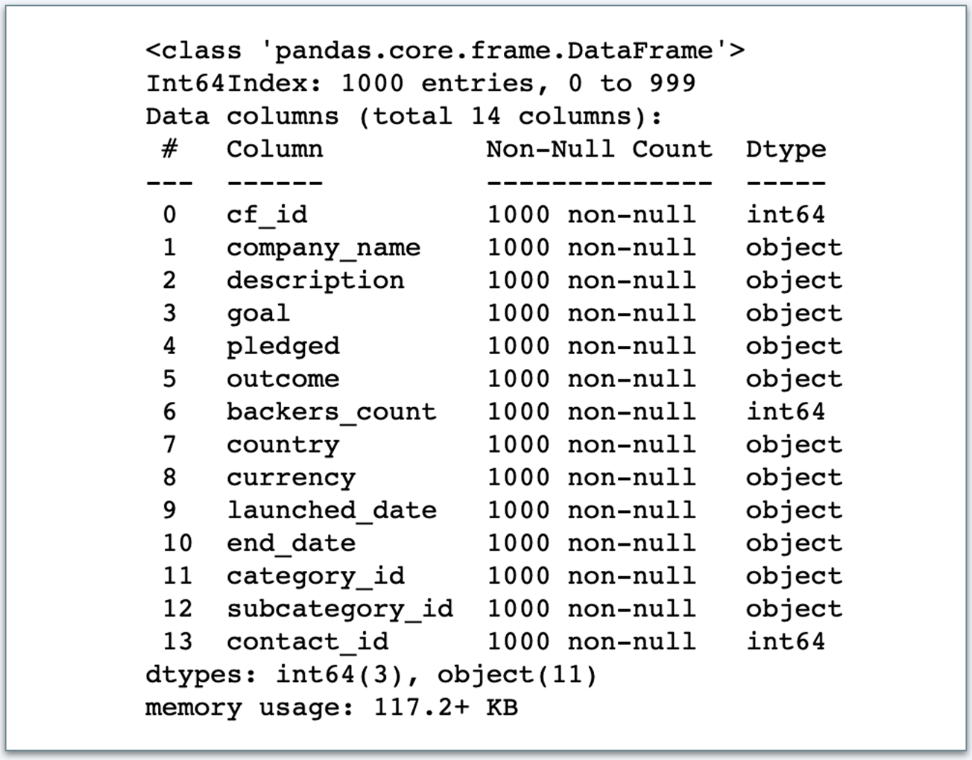
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   cf_id           1000 non-null   int64
 1   company_name    1000 non-null   object
 2   description     1000 non-null   object
 3   goal            1000 non-null   object
 4   pledged         1000 non-null   object
 5   outcome         1000 non-null   object
 6   backers_count   1000 non-null   int64
 7   country         1000 non-null   object
 8   currency        1000 non-null   object
 9   launched_date   1000 non-null   object
 10  end_date        1000 non-null   object
 11  category_id     1000 non-null   object
 12  subcategory_id  1000 non-null   object
 13  contact_id      1000 non-null   object
dtypes: int64(2), object(12)
memory usage: 117.2+ KB
```

We need to convert this "contact_id" column to the `int64` data type before exporting it as CSV.

To do so, run the following code:

```
# Convert the "contact_id" column to an int64 data type.
campaign_cleaned['contact_id'] = pd.to_numeric(campaign_cleaned['contact_
campaign_cleaned.info()
```

The following image shows the output from running the preceding code:



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   cf_id            1000 non-null   int64 
 1   company_name     1000 non-null   object
 2   description      1000 non-null   object
 3   goal             1000 non-null   object
 4   pledged          1000 non-null   object
 5   outcome          1000 non-null   object
 6   backers_count    1000 non-null   int64 
 7   country          1000 non-null   object
 8   currency         1000 non-null   object
 9   launched_date    1000 non-null   object
 10  end_date         1000 non-null   object
 11  category_id      1000 non-null   object
 12  subcategory_id   1000 non-null   object
 13  contact_id       1000 non-null   int64 
dtypes: int64(3), object(11)
memory usage: 117.2+ KB
```

In the preceding image, notice that the "contact_id" column now has the `int64` data type.

The final steps are reorganizing the columns and then exporting the DataFrame as a CSV file. Complete these steps yourself in the following Skill Drill:

SKILL DRILL

Complete the following steps on the `campaign_cleaned` DataFrame:

1. Place the columns in the following order:

   "cf_id", "contact_id", "company_name", "description", "goal", "pledged", "outcome", "backers_count", "country", "currency", "launched_date", "end_date", "category_id","subcategory_id"

2. Export the DataFrame as a CSV named `campaign.csv`, with `encoding='utf8'` and without the index.

Congratulations on transforming the data to create the campaign DataFrame!

ADD/COMMIT/PUSH

Remember to add, commit, and push the Jupyter Notebook file and the `campaign.csv` file to your repository.

Now that we've created the campaign DataFrame, we'll next move on to creating the contacts DataFrame.