4.3.4

Handling Additional Dirty Data

Now that you know how to handle both missing and duplicated data, it's time to turn your attention to other common problems with datasets. Sometimes, datasets have additional types of dirty data. These include typos, misplaced data, mixed data types, and incorrect symbols or punctuation. You need to fix all these problems when preparing data for an analysis.

Let's begin by examining how to clean some common problems with text data.

Identify Messy Text Data

Although a data analyst might dream of working with large datasets that are chock-full of numbers, the reality is that data often contains text fields, as well. Names, emails, comments, symbols, and other text values might prove relevant to an analysis. So, we need tools and techniques to clean and prepare any messy text. **Messy text** can include misspellings, capitalization problems, extra or missing spaces, invalid symbols and punctuation, and many more issues.

Rather than address all the possible text problems, we'll focus on the specific example of how currency symbols, like the United States dollar sign (`$`), influence numbers. When currency symbols are present—for example, in $0.53—Pandas treats the numeric values as strings, so they can't undergo mathematical calculations.

DEEP DIVE ▲

DEEP DIVE

The addition of a currency symbol changes the data type of a number from either an `int` or a `float` to a `string`. A string in Python consists of a series of characters that includes letters, numbers, and special characters. When a DataFrame gets created, Pandas considers any data that contains a text element, such as a letter or a dollar sign (`$`), to be a string. In effect, that

data becomes text. And, the Pandas numerical functions, like `sum` and `mean`, can't process strings.

Luckily, we can apply techniques to remove the currency symbols from any problematic text.

REWIND

In Python, the `int` data type refers to an integer. This is a whole number that can be either positive or negative. The `float` data type refers to a floating point number. This is a numeric value, either positive or negative, that contains a decimal point.

Data pertaining to sales or finance often includes currency symbols. This information can be helpful, but to use it effectively in an analysis, we need to do some work. Specifically, we need to separate the currency symbols from the numerical values. Then, we can use the numbers, calculations, and statistics that are relevant to the analysis.

Consider a dataset that lists prices in United States dollar (USD) currency, as the following code shows:

```
import pandas as pd

prices = pd.DataFrame({
    "price_usd": ["$0.53", "$0.22", "0.34"]
})

prices
```

Notice that the preceding code creates the `prices` DataFrame with three values in the "prices_usd" column. The first two values include a dollar sign (`$`).

Running the preceding code produces the output that the following image shows:

```
import pandas as pd

prices = pd.DataFrame({
    "price_usd": ["$0.53", "$0.22", "0.34"]
})

prices
```

	price_usd
0	$0.53
1	$0.22
2	0.34

In the preceding image, notice that the first two values contain a dollar sign (`$`).

By using the `dtypes` function, we can confirm the data type that Pandas assigns to the DataFrame elements that have a currency symbol, as the following code shows:

```
prices.dtypes
```

Running the preceding code produces the output that the following image shows:

```
prices.dtypes
```

```
price_usd     object
dtype: object
```

In the preceding image, notice that the `price_usd` Series is an `object`. This means that each of the three values in the "prices_usd" column is a Python `object`. The output also shows that the `dtype` is `object`.

The Pandas `object` is the equivalent of a Python `string`. This is a text element that, as we learned earlier, doesn't work with mathematical calculations. So, we'll need to convert the data type to a numerical data type. First, we'll remove the dollar signs from the text data. Then, we'll convert the text data type to a numerical data type.

Remove Symbols from Text Data

To remove the dollar sign, we'll use a Pandas trick: calling the `str.replace` function to replace each instance of the "$" currency symbol string with an empty string, which is represented by two quotation marks with no spaces between them (`""`).

IMPORTANT

For this step, we should replace the string only in the column that we want to fix—by using `loc` or `iloc` to specify that column. In our example, we have only one column ("prices_usd"), but as a best practice, we'll still use `loc` or `iloc`.

The following code replaces the dollar signs with empty strings:

```
prices.loc[:, "price_usd"] = prices.loc[:, "price_usd"].str.replace("$", "")
prices
prices.dtypes
```

Running the preceding code produces the output that the following image shows:

```
prices.loc[:, "price_usd"] = prices.loc[:, "price_usd"].str.replace("$", "")
prices
```

	price_usd
0	0.53
1	0.22
2	0.34

```
prices.dtypes
```

```
price_usd    object
dtype: object
```

In the preceding image, notice that the values in the "price_usd" column no longer have the dollar signs, but their data type is still a Python `object`.

So, we still need to fix the data type—and we'll get to that—but first, let's more closely examine `str.replace`, because it's a bit complex.

The `str.replace` function accepts two arguments. The first is the string that we need to search for. In this case, it's "$", but it can be any text string that you want to fix. The `str.replace` function replaces this argument with the second argument—in this case, an empty string.

Convert a Text Data Type to a Numerical Data Type

Removing the currency symbols from the data still left us with string representations of our numbers. Now, we can use a Pandas function named `astype` to convert the strings to numbers. The `astype` function accepts an argument for the numerical type that we want to convert the data to. The two most-common conversion arguments are `float` (for numbers with decimals) and `int` (for whole numbers).

Because datasets often deal with currencies or prices, the `float` tends to be the numerical type of choice. So, let's use `astype` to convert our price data to floating point numbers, as the following code shows (note that we use `loc` to specify the column that we want to convert):

```
prices.loc[:, "price_usd"] = prices.loc[:, "price_usd"].astype("float")
prices.dtypes
```

Running the preceding code produces the output that the following image shows:

In the preceding image, notice that the data type of the values in the "price_usd" column is now `float64`.

In an analysis, we can now use the values in the "price_usd" column for any numerical calculation that we want.

Now that you've learned how to handle additional dirty data, you'll next have the opportunity to put your new skills together to prepare a dataset for analysis on your own.