Filling In Data

Fill in missing data for features that only have a few missing values.

Chapter Goals:

- Find the rows that contain missing values for `'CPI'` and `'Unemployment'`
- Fill in the missing values using previous row values

A. Finding the missing values

We previously noted that both the `'CPI'` and `'Unemployment'` features contain 585 missing values. We'll find the row indexes containing these missing values by first converting the feature columns in the `na_values` boolean DataFrame to integers, i.e. 0 and 1.

We then use the `nonzero` function to find the locations of the 1's, which correspond to the `True` values.

```
import numpy as np  # NumPy library

na_cpi_int = na_values['CPI'].astype(int)
na_indexes_cpi = na_cpi_int.to_numpy().nonzero()[0]
na_une_int = na_values['Unemployment'].astype(int)
na_indexes_une = na_une_int.to_numpy().nonzero()[0]

print(np.array_equal(na_indexes_cpi, na_indexes_une))
```

Obtaining the row indexes of the missing values in 'CPI' and 'Unemployment'. The na_values DataFrame is the same boolean DataFrame used in previous chapters.

The row indexes are located in the `na_indexes_cpi` and `na_indexes_une` NumPy arrays, which you can see contain the exact same row indexes (sorted in ascending order). Now let's take a closer look at the exact rows that contain the missing values.

```
na_indexes = na_indexes_cpi
na_rows = merged_features.iloc[na_indexes]
print(na_rows['Date'].unique())  # missing value weeks


print(merged_features['Date'].unique()[-13:])  # final 13 weeks

print(na_rows.groupby('Store').count()['Date'].unique())
```

Finding the unique weeks corresponding to the missing values. The na_rows DataFrame consists of all the rows containing the missing values.

In the code above, we found that the missing values for `'CPI'` and `'Unemployment'` correspond to only 13 weeks, which are in fact the final 13 weeks of the entire dataset. Furthermore, the final line of code shows that each store contains 13 weeks with missing values.

Since there are 45 stores, and 13 weeks missing per store, that gives us the total of 585 rows with missing values.

B. Filling in the values

The `'CPI'` and `'Unemployment'` features correspond to the national consumer price index and unemployment rate. These values have minimal change on a week-to-week basis, so we can fill in the missing values using the final `'CPI'` and `'Unemployment'` values for the week of `'2018-04-26'` (the final week without a missing value).

```
print(na_indexes[0])  # first missing value row index
print()

first_missing_row = merged_features.iloc[169]
print(first_missing_row[['Date','CPI','Unemployment']])
print()

final_val_row = merged_features.iloc[168]
print(final_val_row[['Date','CPI','Unemployment']])
print()

cpi_final_val = merged_features.at[168, 'CPI']
une_final_val = merged_features.at[168, 'Unemployment']
merged_features.at[169, 'CPI'] = cpi_final_val
merged_features.at[169, 'Unemployment'] = une_final_val

new_row = merged_features.iloc[169]
print(new_row[['Date','CPI','Unemployment']])
print()
```

Filling in the values for the first row containing missing values.

Since the row indexes in `na_indexes` are sorted in ascending order, we can fill in the missing values using a `for` loop through `na_indexes` (more details in the coding exercise at the end of this chapter).

C. Imputation variants

Filling in missing data with substituted values is known as imputation. The method we used for our dataset replaced missing values with values from closely related observations. However, there are many other forms of imputation, such as replacing the feature's missing data with the feature's mean, median, or mode.

There are also more advanced imputation methods such as k-Nearest Neighbors (filling in missing values based on similarity scores from the kNN algorithm) and MICE (applying multiple chained imputations, assuming the missing values are randomly distributed across observations).

In most industry cases these advanced methods are not required, since the data is either perfectly cleaned or the missing values are scarce. Nevertheless, the advanced methods could be useful when dealing with open source datasets, since these tend to be more incomplete.

Time to Code!

In this chapter you'll be completing the `impute_data` function. You'll use final non-missing value to fill in the missing feature values for `'CPI'` and `'Unemployment'` (as demonstrated in the chapter example).

The `na_indexes_cpi` object represents an array containing the row indexes of the `'CPI'` feature's missing values.

Since these indexes are in ascending sorted order, we can iterate through them and set the missing values equal to the previous row's value in `merged_features`.

Create a `for` loop that iterates through `na_indexes_cpi` using variable `i`.

To ensure that we're updating the actual `merged_features` DataFrame, and not just a copy of the `'CPI'` column, we need to use the `at` function.

Inside the `for` loop, set the value at row `i` in the `'CPI'` column of `merged_features` equal to the value at row `i - 1` in the `'CPI'` column. Make sure to use the `at` function for both the setting and getting operations.

We'll now perform the same steps to fill in the missing data for the `'Unemployment'` column.

Repeat the same process for the `'Unemployment'` column using the `na_indexes_une` array.

```
import pandas as pd

# Fill in missing data
def impute_data(merged_features, na_indexes_cpi, na_indexes_une):
    # CODE HERE
    pass
```