012-data-wrangling-with-pandas

April 25, 2022

Preparing Mexico Data

[2]:
```
import pandas as pd
from IPython.display import VimeoVideo
```

1 Import

The first part of any data science project is preparing your data, which means making sure its in the right place and format for you to conduct your analysis. The first step of any data preparation is importing your raw data and cleaning it.

If you look in the `small-data` directory on your machine, you'll see that the data for this project comes in three CSV files: `mexico-real-estate-1.csv`, `mexico-real-estate-2.csv`, and `mexico-real-estate-3.csv`.

[3]:
```
VimeoVideo("656321516", h="e85e3bf248", width=600)
```

[3]:
```
<IPython.lib.display.VimeoVideo at 0x7f21449643a0>
```

Task 1.2.1: Read these three files into three separate DataFrames named `df1`, `df2`, and `df3`, respectively.

- What's a DataFrame?
- What's a CSV file?
- Read a CSV file into a DataFrame using pandas.

[4]:
```
df1 = pd.read_csv('data/mexico-real-estate-1.csv')
df2 = pd.read_csv('data/mexico-real-estate-2.csv')
df3 = pd.read_csv('data/mexico-real-estate-3.csv')
```

1.1 Clean `df1`

Now that you have your three DataFrames, it's time to inspect them to see if they need any cleaning. Let's look at them one-by-one.

[5]:
```
VimeoVideo("656320563", h="a6841fed28", width=600)
```

[5]:
```
<IPython.lib.display.VimeoVideo at 0x7f2144964070>
```

Task 1.2.2: Inspect `df1` by looking at its `shape` attribute. Then use the `info` method to see the data types and number of missing values for each column. Finally, use the `head` method to determine to look at the first five rows of your dataset.

- Inspect a DataFrame using the `shape`, `info`, and `head` in pandas.

[6]:
```
df1.shape
```

[6]:
```
(700, 6)
```

[7]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   property_type  700 non-null    object 
 1   state          700 non-null    object 
 2   lat            583 non-null    float64
 3   lon            583 non-null    float64
 4   area_m2        700 non-null    float64
 5   price_usd      700 non-null    object 
dtypes: float64(3), object(3)
memory usage: 32.9+ KB
```

[9]:
```
df1.head()
```

[9]:

	property_type	state	lat	lon	area_m2	price_usd
0	house	Estado de México	19.560181	-99.233528	150.0	$67,965.56
1	house	Nuevo León	25.688436	-100.198807	186.0	$63,223.78
2	apartment	Guerrero	16.767704	-99.764383	82.0	$84,298.37
3	apartment	Guerrero	16.829782	-99.911012	150.0	$94,308.80
4	house	Veracruz de Ignacio de la Llave	NaN	NaN	175.0	$94,835.67

It looks like there are a couple of problems in this DataFrame that you need to solve. First, there are many rows with `NaN` values in the `"lat"` and `"lon"` columns. Second, the data type for the `"price_usd"` column is `object` when it should be `float`.

[10]:
```
VimeoVideo("656316512", h="33eb5cb26e", width=600)
```

[10]:
```
<IPython.lib.display.VimeoVideo at 0x7f21448fafd0>
```

Task 1.2.3: Clean `df1` by dropping rows with `NaN` values. Then remove the `"$"` and `","` characters from `"price_usd"` and recast the values in the column as floats.

- What's a data type?
- Drop rows with missing values from a DataFrame using pandas.
- Replace string characters in a column using pandas.
- Recast a column as a different data type in pandas.

[12]:
```
df1.dropna(inplace = True)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 583 entries, 0 to 699
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   property_type  583 non-null    object
 1   state          583 non-null    object
 2   lat            583 non-null    float64
 3   lon            583 non-null    float64
 4   area_m2        583 non-null    float64
 5   price_usd      583 non-null    object
dtypes: float64(3), object(3)
memory usage: 31.9+ KB
```

[22]:
```
df1['price_usd'] = (df1['price_usd']
                    .str.replace('$', '', regex = False)
                    .str.replace(',', '')
                    .astype('float'))
```

[23]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 583 entries, 0 to 699
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   property_type  583 non-null    object
 1   state          583 non-null    object
 2   lat            583 non-null    float64
 3   lon            583 non-null    float64
 4   area_m2        583 non-null    float64
 5   price_usd      583 non-null    float64
dtypes: float64(4), object(2)
memory usage: 31.9+ KB
```

[ ]:

1.2 Clean df2

Now it's time to tackle `df2`. Take a moment to inspect it using the same commands you used before. You'll notice that it has the same issue of `NaN` values, but there's a new problem, too: The home prices are in Mexican pesos (`"price_mxn"`), not US dollars (`"price_usd"`). If we want to compare all the home prices in this dataset, they all need to be in the same currency.

[24]:
```
VimeoVideo("656315668", h="c9bd116aca", width=600)
```

[24]:
```
<IPython.lib.display.VimeoVideo at 0x7f21448fa460>
```

Task 1.2.4: First, drop rows with `NaN` values in `df2`. Next, use the `"price_mxn"` column to create a new column named `"price_usd"`. (Keep in mind that, when this data was collected in 2014, a dollar cost 19 pesos.) Finally, drop the `"price_mxn"` from the DataFrame.

- Drop rows with missing values from a DataFrame using pandas.
- Create new columns derived from existing columns in a DataFrame using pandas.
- Drop a column from a DataFrame using pandas.

[25]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   property_type  700 non-null    object
 1   state          700 non-null    object
 2   lat            571 non-null    float64
 3   lon            571 non-null    float64
 4   area_m2        700 non-null    float64
 5   price_mxn      700 non-null    float64
dtypes: float64(4), object(2)
memory usage: 32.9+ KB
```

[29]:
```
df2.dropna(inplace = True)
df2['price_usd'] = (df2['price_mxn'] / 19).round(2)
df2.drop(columns = ['price_mxn'], inplace = True)
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 571 entries, 0 to 699
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   property_type  571 non-null    object
 1   state          571 non-null    object
```

```
 2   lat             571 non-null    float64
 3   lon             571 non-null    float64
 4   area_m2         571 non-null    float64
 5   price_usd       571 non-null    float64
dtypes: float64(4), object(2)
memory usage: 31.2+ KB
```

[ ]:

[ ]:

[ ]:

1.3 Clean df3

Great work! We're now on the final DataFrame. Use the same `shape`, `info` and `head` commands to inspect the `df3`. Do you see any familiar issues?

You'll notice that we still have `NaN` values, but there are two new problems:

1. Instead of separate `"lat"` and `"lon"` columns, there's a single `"lat-lon"` column.
2. Instead of a `"state"` column, there's a `"place_with_parent_names"` column.

We need the resolve these problems so that `df3` has the same columns in the same format as `df1` and `df2`.

[30]:
```
VimeoVideo("656314718", h="8d1127a93f", width=600)
```

[30]:
```
<IPython.lib.display.VimeoVideo at 0x7f21448faa30>
```

Task 1.2.5: Drop rows with `NaN` values in `df3`. Then use the `split` method to create two new columns from `"lat-lon"` named `"lat"` and `"lon"`, respectively.

- Drop rows with missing values from a DataFrame using pandas.
- Split the strings in one column to create another using pandas.

[34]:
```
df3.dropna(inplace = True)
df3[['lat', 'lon']] = df3['lat-lon'].str.split(',', expand = True)
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 582 entries, 0 to 699
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   property_type           582 non-null    object
 1   place_with_parent_names 582 non-null    object
 2   lat-lon                 582 non-null    object
 3   area_m2                 582 non-null    float64
 4   price_usd               582 non-null    float64
```

```
 5   lat                       582 non-null    object
 6   lon                       582 non-null    object
dtypes: float64(2), object(5)
memory usage: 36.4+ KB
```

[35]:
```
VimeoVideo("656314050", h="13f6a677fd", width=600)
```

[35]:
```
<IPython.lib.display.VimeoVideo at 0x7f214320dfa0>
```

Task 1.2.6: Use the `split` method again, this time to extract the state for every house. (Note that the state name always appears after `"México|"` in each string.) Use this information to create a `"state"` column. Finally, drop the `"place_with_parent_names"` and `"lat-lon"` columns from the DataFrame.

- Split the strings in one column to create another using pandas.
- Drop a column from a DataFrame using pandas.

[40]:
```
df3['state'] = df3['place_with_parent_names'].str.split('|', expand = True)[2]
df3.drop(columns = ['lat-lon', 'place_with_parent_names'], inplace = True)
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 582 entries, 0 to 699
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   property_type  582 non-null    object
 1   area_m2        582 non-null    float64
 2   price_usd      582 non-null    float64
 3   lat            582 non-null    object
 4   lon            582 non-null    object
 5   state          582 non-null    object
dtypes: float64(2), object(4)
memory usage: 31.8+ KB
```

1.4 Concatenate DataFrames

Great work! You have three clean DataFrames, and now it's time to combine them into a single DataFrame so that you can conduct your analysis.

[41]:
```
VimeoVideo("656313395", h="ccadbc2689", width=600)
```

[41]:
```
<IPython.lib.display.VimeoVideo at 0x7f214320d910>
```

Task 1.2.7: Use `pd.concat` to concatenate `df1`, `df2`, `df3` as new DataFrame named `df`. Your new DataFrame should have 1,736 rows and 6 columns:`"property_type"`, `"state"`, `"lat"`, `"lon"`, `"area_m2"`, `"price_usd"`, and `"price_per_m2"`.

- Concatenate two or more DataFrames using pandas.

[42]:
```
df = pd.concat([df1, df2, df3])
print(df.shape)
df.head()
```

```
(1736, 6)
```

[42]:

	property_type	state	lat	lon	area_m2	price_usd
0	house	Estado de México	19.560181	-99.233528	150.0	67965.56
1	house	Nuevo León	25.688436	-100.198807	186.0	63223.78
2	apartment	Guerrero	16.767704	-99.764383	82.0	84298.37
3	apartment	Guerrero	16.829782	-99.911012	150.0	94308.80
5	house	Yucatán	21.052583	-89.538639	205.0	105191.37

1.5 Save `df`

The data is clean and in a single DataFrame, and now you need to save it as a CSV file so that you can examine it in your exploratory data analysis.

[43]:
```
VimeoVideo("656312464", h="81ee04de15", width=600)
```

[43]:
```
<IPython.lib.display.VimeoVideo at 0x7f21448470a0>
```

Task 1.2.8: Save `df` as a CSV file using the `to_csv` method. The file path should be `"./data/mexico-real-estate-clean.csv"`. Be sure to set the `index` argument to `False`.

- What's a CSV file?
- Save a DataFrame as a CSV file using pandas.

[45]:
```
df.to_csv('data/mexico-real-estate-clean.csv', index = False)
```
