File Edit View Run Kernel Tabs Settings Help

011-tabular-and-tidy-data.ip × 012-data-wrangling-with-pa ×

Code Python 3 (ipykernel)

Filter files by name

/ ··· / ds-curriculum / 010-housing-in-mexico /

Name	Last Modified
data	16 hours ago
images	16 hours ago
011-tabular-a...	7 minutes ago
012-data-wra...	16 hours ago
013-explorat...	16 hours ago
014-size-or-l...	16 hours ago
015-assignm...	16 hours ago

[1]:
```
# WELCOME THE FIRST LESSON OF PROJECT 1!
#
# If you haven't watched the virtual machine orientation video yet,
# 👆 Click the ▶ button above to get started.

from IPython.display import VimeoVideo

VimeoVideo("656189257", h="3298dbabb7", width=600)
```

[1]:

1.1 Organizing Tabular Data in Python

What's Tabular Data?

[2]:
```
VimeoVideo("645390762", h="578b5a9e19", width=600)
```

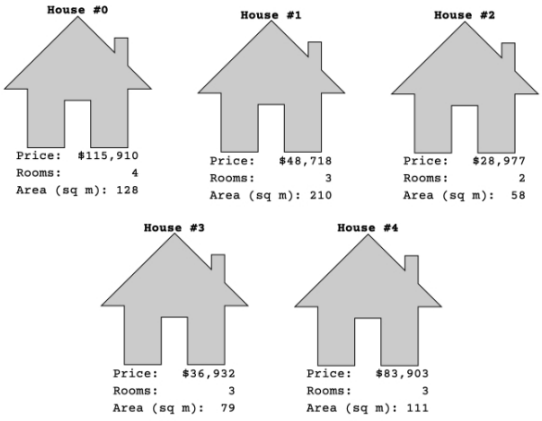
[2]:

Frequent Question: When I try to run this cell 👆 I get:

> `NameError: name 'VimeoVideo' is not defined`

Tip: `NameError` usually means that you're asking Python to use a tool that you haven't imported yet. So make sure you run the first cell of this notebook, the one that contains `from IPython.display import VimeoVideo`.

Information can come in many forms, and part of a data scientist's job is making sure that information is organized in a way that's conducive to analysis. Take for example these five houses from the Mexico real estate dataset we'll use in this project:



One common way to organize this information is in a **table**, which is a group of **cells** organized into **rows** and **columns**:

house	price	rooms	area
0	$115,910	4	128
1	$48,718	3	210
2	$28,977	2	58
3	$36,932	3	79
4	$83,903	3	111

[32]: `houses_columnwise = {`

Cell Tags

Add Tag +

Raw NBConvert Format

Slide Type

Advanced Tools

When working with this sort of **tabular data**, it's important to organize row and columns following the principles of "**tidy data**." What does that mean in the case of our dataset?

1. Each row corresponds to a single house in our dataset. We'll call each of these houses an **observation**.
2. Each column corresponds to a characteristic of each house. We'll call these **features**.
3. Each cell contains only one **value**.

So whenever you encounter a new dataset, make sure your data is "tidy." `NOTE: need to add activity here`

Tabular Data and Python Data Structures

Working with Lists

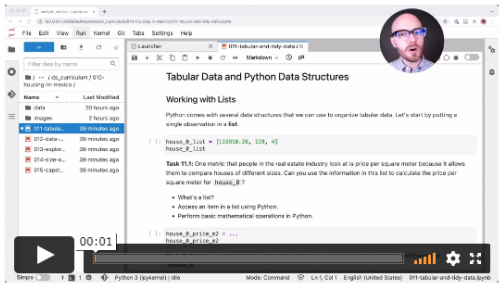
Python comes with several data structures that we can use to organize tabular data. Let's start by putting a single observation in a **list**.

[3]:
```
house_0_list = [115910.26, 128, 4]
house_0_list
```

[3]:
```
[115910.26, 128, 4]
```

[4]:
```
VimeoVideo("645390786", h="da0bb831d1", width=600)
```

[4]:



Task 1.1.1: One metric that people in the real estate industry look at is price per square meter because it allows them to compare houses of different sizes. Can you use the information in this list to calculate the price per square meter for `house_0`?

- What's a list?
- Access an item in a list using Python.
- Perform basic mathematical operations in Python.

[6]:
```
house_0_price_m2 = [115910.26, 128, 4]
house_0_price_m2 = house_0_price_m2[0]/house_0_price_m2[1]
house_0_price_m2
```

[6]:
```
905.54890625
```

[7]:
```
VimeoVideo("645390797", h="86c579a9cb", width=600)
```

[7]:

Task 1.1.2: Next, use the `append` method to add the price per square meter to the end of the end of `house_0`.

- Append an item to a list in Python.

[11]:
```

house_0_list.remove(house_0_price_m2)
house_0_list
```

[11]:
```
[115910.26, 128, 4, 905.54890625]
```

Now that you can work with data for a single house, let's think about how to organize the whole dataset. One option would be to create a list for each observation and then put those together in another list. This is called a **nested list**.

[14]:
```
houses_nested_list = [
    [115910.26, 128.0, 4.0],
    [48718.17, 210.0, 3.0],
    [28977.56, 58.0, 2.0],
    [36932.27, 79.0, 3.0],
    [83903.51, 111.0, 3.0],
```

```
]

houses_nested_list
```

[14]:
```
[[115910.26, 128.0, 4.0],
 [48718.17, 210.0, 3.0],
 [28977.56, 58.0, 2.0],
 [36932.27, 79.0, 3.0],
 [83903.51, 111.0, 3.0]]
```

[19]:
```
# [list_item.append(list_item[0]/list_item[1])  for list_item in houses_nested_list]
# houses_nested_list
```

Now that we have more observations, it doesn't make sense to calculate the price per square meter for each house one-by-one. Instead, we can automate this repetitive task using a `for` loop.

[16]:
```
VimeoVideo("645390807", h="4536120cf5", width=600)
```

[16]:

Task 1.1.3: Append the price per square meter to each observation in `houses_nested_list` using a `for` loop.

- What's a for loop?
- Write a for loop in Python.

[18]:
```
for house in houses_nested_list:
    house_price = house[0] / house[1]
    house.append(house_price)
houses_nested_list
```

[18]:
```
[[115910.26, 128.0, 4.0, 905.54890625],
 [48718.17, 210.0, 3.0, 231.9912857142857],
 [28977.56, 58.0, 2.0, 499.61310344827587],
 [36932.27, 79.0, 3.0, 467.4970886075949],
 [83903.51, 111.0, 3.0, 755.8874774774774]]
```

Working with Dictionaries

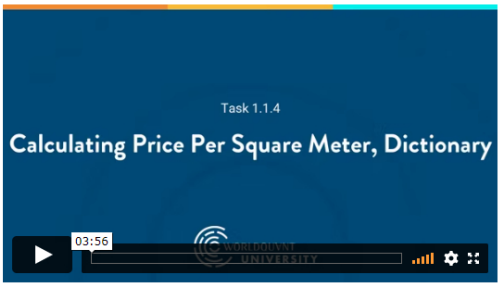
Lists are a good way to organize data, but one drawback is that we can only represent values. Why is that a problem? For example, someone looking at `[115910.26, 128.0, 4]` wouldn't know which values corresponded to price, area, etc. A better option might be a **dictionary**, where each value is associated with a key. Here's what `house_0` looks like as a dictionary instead of a list.

[20]:
```
house_0_dict = {
    "price_aprox_usd": 115910.26,
    "surface_covered_in_m2": 128,
    "rooms": 4,
}

house_0_dict
```

[20]:
```
{'price_aprox_usd': 115910.26, 'surface_covered_in_m2': 128, 'rooms': 4}
```

[21]:
```
VimeoVideo("645390821", h="884613d46b", width=600)
```

[21]:



Task 1.1.4: Calculate the price per square meter for `house_0` and add it to the dictionary under the key `"price_per_m2"`.

- What's a dictionary?
- Access an item in a dictionary in Python.

[22]:
```
house_0_dict["price_per_m2"] = house_0_dict['price_aprox_usd'] / house_0_dict['surface_covered_in_m2']
house_0_dict
```

[22]:
```
{'price_aprox_usd': 115910.26,
 'surface_covered_in_m2': 128,
 'rooms': 4,
 'price_per_m2': 905.54890625}
```

If we wanted to combine all our observations together, the best way would be to create a list of dictionaries.

[25]:
```
houses_rowwise = [
    {
        "price_aprox_usd": 115910.26,
        "surface_covered_in_m2": 128,
        "rooms": 4,
    },
    {
        "price_aprox_usd": 48718.17,
        "surface_covered_in_m2": 210,
        "rooms": 3,
    },
    {
        "price_aprox_usd": 28977.56,
        "surface_covered_in_m2": 58,
        "rooms": 2,
    },
    {
        "price_aprox_usd": 36932.27,
        "surface_covered_in_m2": 79,
        "rooms": 3,
    },
    {
        "price_aprox_usd": 83903.51,
        "surface_covered_in_m2": 111,
        "rooms": 3,
    },
```

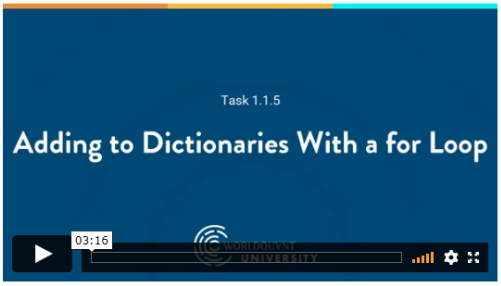
```
]

houses_rowwise
```

[25]:
```
[{'price_aprox_usd': 115910.26, 'surface_covered_in_m2': 128, 'rooms': 4},
 {'price_aprox_usd': 48718.17, 'surface_covered_in_m2': 210, 'rooms': 3},
 {'price_aprox_usd': 28977.56, 'surface_covered_in_m2': 58, 'rooms': 2},
 {'price_aprox_usd': 36932.27, 'surface_covered_in_m2': 79, 'rooms': 3},
 {'price_aprox_usd': 83903.51, 'surface_covered_in_m2': 111, 'rooms': 3}]
```

This way of storing data is so popular, it has its own name: **JSON**. We'll learn more about it later in the course. For now, let's build another for loop, but this time, we'll add a add the price per square meter to each dictionary.

[26]:
```
VimeoVideo("645390833", h="0d3963c0d0", width=600)
```

[26]:



Task 1.1.5: Using a `for` loop, calculate the price per square meter and store the result under a `"price_per_m2"` key for each observation in `houses_rowwise`.

- What's JSON?
- Write a for loop in Python.

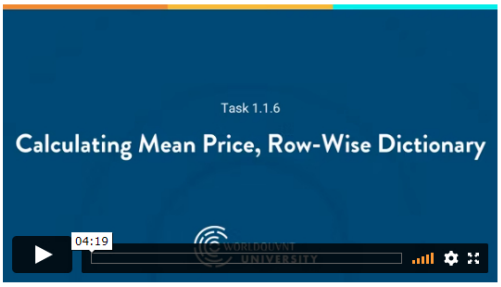
[29]:
```
for house in houses_rowwise:
    house["price_per_m2"]  = house['price_aprox_usd'] / house['surface_covered_in_m2']
houses_rowwise
```

[29]:
```
[{'price_aprox_usd': 115910.26,
  'surface_covered_in_m2': 128,
  'rooms': 4,
  'price_per_m2': 905.54890625},
 {'price_aprox_usd': 48718.17,
  'surface_covered_in_m2': 210,
  'rooms': 3,
  'price_per_m2': 231.9912857142857},
 {'price_aprox_usd': 28977.56,
  'surface_covered_in_m2': 58,
  'rooms': 2,
  'price_per_m2': 499.61310344827587},
 {'price_aprox_usd': 36932.27,
  'surface_covered_in_m2': 79,
  'rooms': 3,
  'price_per_m2': 467.4970886075949},
 {'price_aprox_usd': 83903.51,
  'surface_covered_in_m2': 111,
  'rooms': 3,
  'price_per_m2': 755.8874774774774}]
```

JSON is a great way to organize data, but it does have some downsides. Note that each dictionary represents a single house or, if we think about it as tabular data, a row in our dataset. This means that it's pretty easy to do row-wise calculations (like we did with price per square meter), but column-wise calculations are more complicated. For instance, what if we wanted to know the mean house price for our dataset? First we'd need to collect the price for each house in a list and then calculate mean.

[30]:
```
VimeoVideo("645390848", h="889a3cfb33", width=600)
```

[30]:



Task 1.1.6: To calculate the mean price for `houses_rowwise` by completing the code below.

- Write a for loop in Python.
- Append an item to a list in Python.

[31]:
```
house_prices = []
for house in houses_rowwise:
    house_prices.append(house["price_aprox_usd"])
mean_house_price = sum(house_prices) / len(house_prices)

mean_house_price
```

[31]:
```
62888.35399999999
```

One way to make this sort of calculation easier is to organize our data by features instead of observations. We'll still use dictionaries and lists, but we'll implement them a slightly differently.

[32]:
```
houses_columnwise = {
    "price_aprox_usd": [115910.26, 48718.17, 28977.56, 36932.27, 83903.51],
    "surface_covered_in_m2": [128.0, 210.0, 58.0, 79.0, 111.0],
    "rooms": [4.0, 3.0, 2.0, 3.0, 3.0],
}

houses_columnwise
```

[32]:
```
{'price_aprox_usd': [115910.26, 48718.17, 28977.56, 36932.27, 83903.51],
 'surface_covered_in_m2': [128.0, 210.0, 58.0, 79.0, 111.0],
 'rooms': [4.0, 3.0, 2.0, 3.0, 3.0]}
```

[33]:
```
VimeoVideo("645390869", h="ef4d49bf66", width=600)
```

[33]:

Task 1.1.7: Calculate the mean house price in `houses_columnwise`

- Perform common aggregation tasks on a list in Python.

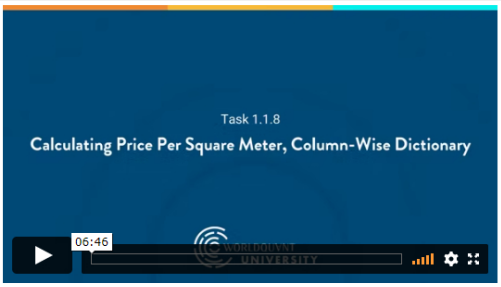
[34]:
```
mean_house_price = sum(houses_columnwise['price_aprox_usd']) / len(houses_columnwise['price_aprox_usd'])

mean_house_price
```

[34]:
```
62888.35399999999
```

Of course, when we organize our data according to columns / features, row-wise operations become more difficult.

[35]:
```
VimeoVideo("645396267", h="66eda35f00", width=600)
```

[35]:



Task 1.1.8: Create a `"price_per_m2"` column in `houses_columnwise`?

- Add a a key-value pair to a dictionary in Python.
- Zip two lists together in Python.
- Write a for loop in Python.

[42]:
```
houses_columnwise['price_per_m2'] = []
for price, m2 in zip(houses_columnwise['price_aprox_usd'], houses_columnwise['surface_covered_in_m2']):
    houses_columnwise['price_per_m2'].append(price/m2)
houses_columnwise
```

[42]:
```
{'price_aprox_usd': [115910.26, 48718.17, 28977.56, 36932.27, 83903.51],
 'surface_covered_in_m2': [128.0, 210.0, 58.0, 79.0, 111.0],
 'rooms': [4.0, 3.0, 2.0, 3.0, 3.0],
 'price_per_m2': [905.54890625,
  231.9912857142857,
  499.61310344827587,
  467.4970886075949,
  755.8874774774774]}
```

Tabular Data and pandas DataFrames

[ ]:
```
VimeoVideo("645396345", h="ba25c25741", width=600)
```

While you've shown that you can wrangle data using lists and dictionaries, it's not as intuitive as working with, say, a spreadsheet. Fortunately, there are lots of libraries for Python that make it an even better tool for tabular data – way better than spreadsheet applications like Microsoft Excel or Google Sheets! One of the best known data science libraries is **pandas**, which allows you to organize data into **DataFrames**.

Let's import pandas and then create a DataFrame from `houses_columnwise`.

[43]:
```
import pandas as pd

data = {
    "price_aprox_usd": [115910.26, 48718.17, 28977.56, 36932.27, 83903.51],
    "surface_covered_in_m2": [128.0, 210.0, 58.0, 79.0, 111.0],
    "rooms": [4.0, 3.0, 2.0, 3.0, 3.0],
}

df_houses = pd.DataFrame(data)

df_houses
```

[43]:

	price_aprox_usd	surface_covered_in_m2	rooms
0	115910.26	128.0	4.0
1	48718.17	210.0	3.0
2	28977.56	58.0	2.0
3	36932.27	79.0	3.0
4	83903.51	111.0	3.0

Excellent work! You've mastered the concept of **tabular data**, understand the principles behind **tidy data**, and used **lists** and **dictionaries** to organize and augment our Mexico housing dataset. Next, we'll use these skills on the entire dataset — with over 150,000 observations — to better understand the real estate market in the country.
