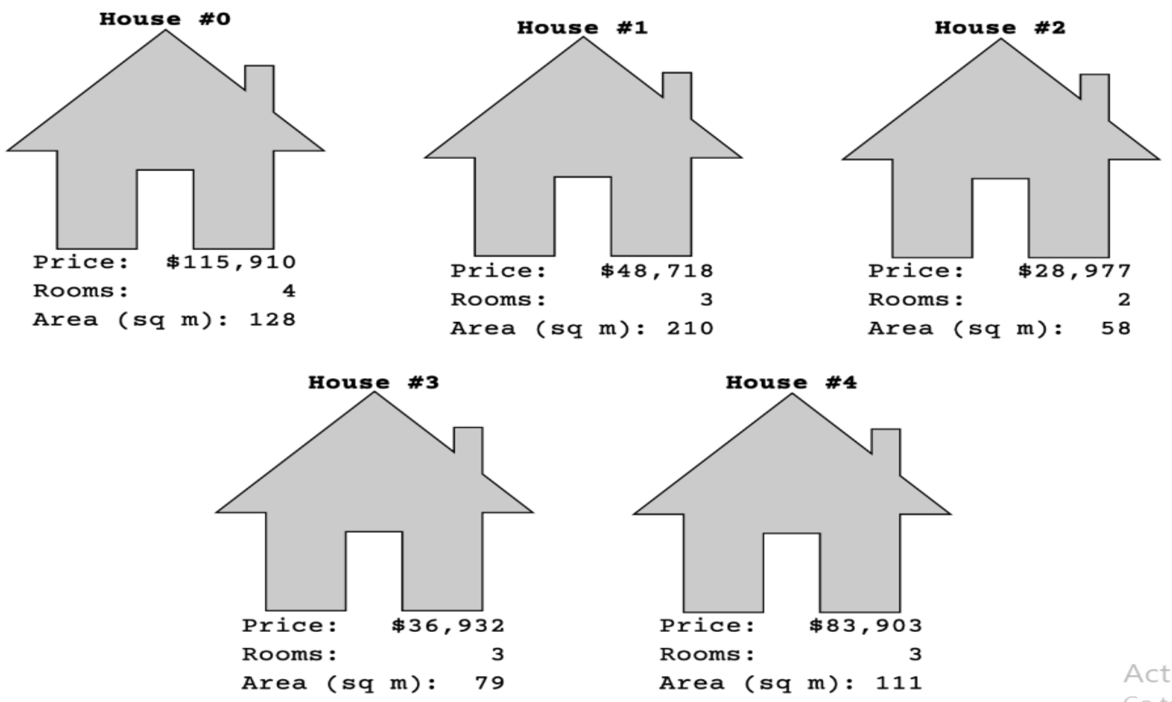
1.1. Organizing Tabular Data in Python

nformation 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

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."

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]:
```python
# Declare variable `house_0_list`
house_0_list = [115910.26, 128, 4]

# Print object type of `house_0_list`
# (We'll learn more about object types in later projects 😉)
print("house_0_list type:", type(house_0_list))

# Print length of `house_0_list`
print("house_0_list length:", len(house_0_list))

# Get output of `house_0_list`
house_0_list
```

```
house_0_list type: <class 'list'>
house_0_list length: 3
```

[3]:
```
[115910.26, 128, 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.](#)

[5]:
```python
# Declare variable `house_0_price_m2`
house_0_price_m2 = house_0_list[0] / house_0_list[1]

# Print object type of `house_0_price_m2`
print("house_0_price_m2 type:", type(house_0_price_m2))

# Get output of `house_0_price_m2`
house_0_price_m2
```

```
house_0_price_m2 type: <class 'float'>
```

[5]:
```
905.54890625
```

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.

```
[7]: # Append price / sq. meter to `house_0_list`
     house_0_list.append(house_0_price_m2)

     # Print object type of `house_0_list`
     print("house_0_list type:", type(house_0_list))

     # Print length of `house_0_list`
     print("house_0_list length:", len(house_0_list))

     # Get output of `house_0_list`
     house_0_list
```

```
house_0_list type: <class 'list'>
house_0_list length: 4
```

```
[7]: [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**.

```
[8]: # Declare variable `houses_nested_list`
     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],
     ]

     # Print `houses_nested_list` type
     print("houses_nested_list type:", type(houses_nested_list))

     # Print `houses_nested_list` length
     print("houses_nested_list length:", len(houses_nested_list))

     # Get output of `houses_nested_list`
     houses_nested_list
```

```
houses_nested_list type: <class 'list'>
houses_nested_list length: 5
```

```
[8]: [[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]]
```

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.

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.

[10]:
```python
# Create for loop to iterate through `houses_nested_list`
for house in houses_nested_list:
    # For each observation, append price / sq. meter
    price_m2 = house[0] / house[1]
    house.append(price_m2)


# Print `houses_nested_list` type
print("houses_nested_list type:", type(houses_nested_list))

# Print `houses_nested_list` length
print("houses_nested_list length:", len(houses_nested_list))

# Get output of `houses_nested_list`
houses_nested_list
```

```
houses_nested_list type: <class 'list'>
houses_nested_list length: 5
```

[10]:
```
[[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.

[11]:
```python
# Declare variable `house_0_dict`
house_0_dict = {
    "price_approx_usd": 115910.26,
    "surface_covered_in_m2": 128,
    "rooms": 4,
}

# Print `house_0_dict` type
print("house_0_dict type:", type(house_0_dict))

# Get output of `house_0_dict`
house_0_dict
```

```
house_0_dict type: <class 'dict'>
```

[11]:
```
{'price_approx_usd': 115910.26, 'surface_covered_in_m2': 128, 'rooms': 4}
```

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.

[13]:
```python
# Add "price_per_m2" key-value pair to `house_0_dict`
house_0_dict["price_per_m2"] = house_0_dict['price_approx_usd'] / house_0_dict['surface_covered_in_m2']

# Get output of `house_0_dict`
house_0_dict
```

[13]:
```
{'price_approx_usd': 115910.26,
 'surface_covered_in_m2': 128,
 'rooms': 4,
 'price_per_m2': 905.54890625}
```

[14]:
```python
# Declare variable `houses_rowwise`
houses_rowwise = [
    {
        "price_approx_usd": 115910.26,
        "surface_covered_in_m2": 128,
        "rooms": 4,
    },
    {
        "price_approx_usd": 48718.17,
        "surface_covered_in_m2": 210,
        "rooms": 3,
    },
    {
        "price_approx_usd": 28977.56,
        "surface_covered_in_m2": 58,
        "rooms": 2,
    },
    {
        "price_approx_usd": 36932.27,
        "surface_covered_in_m2": 79,
        "rooms": 3,
    },
    {
        "price_approx_usd": 83903.51,
        "surface_covered_in_m2": 111,
        "rooms": 3,
    },
]

# Print `houses_rowwise` object type
print("houses_rowwise type:", type(houses_rowwise))

# Print `houses_rowwise` length
print("houses_rowwise length:", len(houses_rowwise))

# Get output of `houses_rowwise`
houses_rowwise
```

```
houses_rowwise type: <class 'list'>
houses_rowwise length: 5
```

[14]:
```
[{'price_approx_usd': 115910.26, 'surface_covered_in_m2': 128, 'rooms': 4},
 {'price_approx_usd': 48718.17, 'surface_covered_in_m2': 210, 'rooms': 3},
 {'price_approx_usd': 28977.56, 'surface_covered_in_m2': 58, 'rooms': 2},
 {'price_approx_usd': 36932.27, 'surface_covered_in_m2': 79, 'rooms': 3},
 {'price_approx_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 the price per square meter to each dictionary.

[16]:
```python
# Create for loop to iterate through `houses_rowwise`
for house in houses_rowwise:
    # For each observation, add "price_per_m2" key-value pair
    house["price_per_m2"] = house['price_approx_usd'] / house['surface_covered_in_m2']

# Print `houses_rowwise` object type
print("houses_rowwise type:", type(houses_rowwise))

# Print `houses_rowwise` length
print("houses_rowwise length:", len(houses_rowwise))

# Get output of `houses_rowwise`
houses_rowwise
```

```
houses_rowwise type: <class 'list'>
houses_rowwise length: 5
```

[16]:
```
[{'price_approx_usd': 115910.26,
  'surface_covered_in_m2': 128,
  'rooms': 4,
  'price_per_m2': 905.54890625},
 {'price_approx_usd': 48718.17,
  'surface_covered_in_m2': 210,
  'rooms': 3,
  'price_per_m2': 231.9912857142857},
 {'price_approx_usd': 28977.56,
  'surface_covered_in_m2': 58,
  'rooms': 2,
  'price_per_m2': 499.61310344827587},
 {'price_approx_usd': 36932.27,
  'surface_covered_in_m2': 79,
  'rooms': 3,
  'price_per_m2': 467.4970886075949},
 {'price_approx_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.

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.]

[19]:
```python
# Declare `house_prices` as empty list
house_prices = []

# Iterate through `houses_rowwise`
for house in houses_rowwise:
    # For each house, append "price_approx_usd" to `house_prices`
    house_prices.append(house['price_approx_usd'])

# Calculate `mean_house_price` using `house_prices`
mean_house_price = sum(house_prices) / len(house_prices)

# Print `mean_house_price` object type
print("mean_house_price type:", type(mean_house_price))

# Get output of `mean_house_price`
mean_house_price
```

```
mean_house_price type: <class 'float'>
```

[19]: `62888.35399999999`

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.

```
[20]: # Declare variable `houses_columnwise`
houses_columnwise = {
    "price_approx_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],
}

# Print `houses_columnwise` object type
print("houses_columnwise type:", type(houses_columnwise))

# Get output of `houses_columnwise`
houses_columnwise
```

```
houses_columnwise type: <class 'dict'>

[20]: {'price_approx_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]}
```

One way to

Task 1.1.7: Calculate the mean house price in houses_columnwise

- [Perform common aggregation tasks on a list in Python.]()

```
[23]: # Calculate `mean_house_price` using `houses_columnwise`
mean_house_price = sum(houses_columnwise['price_approx_usd']) /len(houses_columnwise['price_approx_usd'])

# Print `mean_house_price` object type
print("mean_house_price type:", type(mean_house_price))

# Get output of `mean_house_price`
mean_house_price
```

```
mean_house_price type: <class 'float'>

[23]: 62888.35399999999
```

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

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.]()

```
[26]: price = houses_columnwise['price_approx_usd']
area  = houses_columnwise['surface_covered_in_m2']

price_per_m2 = []
for p,a in zip(price,area):
    price_m2 = p/a
    price_per_m2.append(price_m2)
# Add "price_per_m2" key-value pair for `houses_columnwise`
houses_columnwise["price_per_m2"] =  price_per_m2

# Print `houses_columnwise` object type
print("houses_columnwise type:", type(houses_columnwise))

# Get output of `houses_columnwise`
houses_columnwise
```

```
houses_columnwise type: <class 'dict'>

[26]: {'price_approx_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

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.

[28]:
```
# Import pandas library, aliased as `pd`
import pandas as pd

# Declare variable `df_houses`
df_houses = pd.DataFrame(houses_columnwise)

# Print `df_houses` object type
print("df_houses type:", type(df_houses))

# Print `df_houses` shape
print("df_houses shape:", df_houses.shape)

# Get output of `df_houses`
df_houses
```

```
df_houses type: <class 'pandas.core.frame.DataFrame'>
df_houses shape: (5, 4)
```

[28]:

	price_approx_usd	surface_covered_in_m2	rooms	price_per_m2
0	115910.26	128.0	4.0	905.548906
1	48718.17	210.0	3.0	231.991286
2	28977.56	58.0	2.0	499.613103
3	36932.27	79.0	3.0	467.497089
4	83903.51	111.0	3.0	755.887477

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.

1.2. Preparing Mexico Data

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.

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.

[3]:
```
# Load CSV files into DataFrames
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")

# Print object type and shape for DataFrames
print("df1 type:", type(df1))
print("df1 shape:", df1.shape)
print()
print("df2 type:", type(df2))
print("df2 shape:", df2.shape)
print()
print("df3 type:", type(df3))
print("df3 shape:", df3.shape)
```

```
df1 type: <class 'pandas.core.frame.DataFrame'>
df1 shape: (700, 6)

df2 type: <class 'pandas.core.frame.DataFrame'>
df2 shape: (700, 6)

df3 type: <class 'pandas.core.frame.DataFrame'>
df3 shape: (700, 5)
```

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.

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.

[5]:
```python
# Print df1 shape
df1.shape

# Print df1 info
df1.info()

# Get output of df1 head
df1.head()
```

```
<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
```

[5]:

	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.

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.]()

[7]:
```python
# Drop null values from df1
df1.dropna(inplace=True)

# Clean "price_usd" column in df1
df1["price_usd"] = (df1["price_usd"]
                    .str.replace("$","",regex=False)
                    .str.replace(",","")
                    .astype(float)
                   )

# Print object type, shape, and head
print("df1 type:", type(df1))
print("df1 shape:", df1.shape)
df1.head()
```

```
df1 type: <class 'pandas.core.frame.DataFrame'>
df1 shape: (583, 6)
```

[7]:

	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

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.

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.

[9]:
```python
# Drop null values from df2
df2.dropna(inplace=True)

# Create "price_usd" column for df2 (19 pesos to the dollar in 2014)
df2["price_usd"] = (df2['price_mxn'] / 19 ).round(2)

# Drop "price_mxn" column from df2
df2.drop(columns=['price_mxn'],inplace=True)

# Print object type, shape, and head
print("df2 type:", type(df2))
print("df2 shape:", df2.shape)
df2.head()
```

```
df2 type: <class 'pandas.core.frame.DataFrame'>
df2 shape: (571, 6)
```

[9]:

	property_type	state	lat	lon	area_m2	price_usd
0	apartment	Nuevo León	25.721081	-100.345581	72.0	68421.05
2	house	Morelos	23.634501	-102.552788	360.0	278947.37
6	apartment	Estado de México	19.272040	-99.572013	85.0	65789.47
7	house	San Luis Potosí	22.138882	-100.996510	158.0	111578.95
8	apartment	Distrito Federal	19.394558	-99.129707	65.0	39904.74

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.

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.

[11]:
```
# Drop null values from df3
df3.dropna(inplace = True)

# Create "lat" and "lon" columns for df3
df3[["lat", "lon"]] = df3['lat-lon'].str.split(",", expand=True)
#droping the original columns can be done by
#df3.drop(column=['lat-lon'], inplace=True)

# Print object type, shape, and head
print("df3 type:", type(df3))
print("df3 shape:", df3.shape)
df3.head()
```

```
df3 type: &lt;class 'pandas.core.frame.DataFrame'&gt;
df3 shape: (582, 7)
```

[11]:

	property_type	place_with_parent_names	lat-lon	area_m2	price_usd	lat	lon
0	apartment	|México|Distrito Federal|Gustavo A. Madero|Acu...	19.52589,-99.151703	71.0	48550.59	19.52589	-99.151703
1	house	|México|Estado de México|Toluca|Metepec|	19.2640539,-99.5727534	233.0	168636.73	19.2640539	-99.5727534
2	house	|México|Estado de México|Toluca|Toluca de Lerd...	19.268629,-99.671722	300.0	86932.69	19.268629	-99.671722
4	apartment	|México|Veracruz de Ignacio de la Llave|Veracruz|	19.511938,-96.871956	84.0	68508.67	19.511938	-96.871956
5	house	|México|Jalisco|Guadalajara|	20.689157,-103.366728	175.0	102763.00	20.689157	-103.366728

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.

[18]:
```python
# Create "state" column for df3
#df3["state"] = df3['place_with_parent_names'].str.split("|", expand=True)[2]

# Drop "place_with_parent_names" and "lat-lon" from df3

#df3.drop(columns=['place_with_parent_names',"lat-lon"],inplace=True)
# Print object type, shape, and head
print("df3 type:", type(df3))
print("df3 shape:", df3.shape)
df3.head()
```

```
df3 type: <class 'pandas.core.frame.DataFrame'>
df3 shape: (582, 6)
```

[18]:

	property_type	area_m2	price_usd	lat	lon	state
0	apartment	71.0	48550.59	19.52589	-99.151703	Distrito Federal
1	house	233.0	168636.73	19.2640539	-99.5727534	Estado de México
2	house	300.0	86932.69	19.268629	-99.671722	Estado de México
4	apartment	84.0	68508.67	19.511938	-96.871956	Veracruz de Ignacio de la Llave
5	house	175.0	102763.00	20.689157	-103.366728	Jalisco

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", and "price_usd".

- Concatenate two or more DataFrames using pandas.

[20]:
```python
# Concatenate df1, df2, and df3
df = pd.concat([df1,df2,df3])

# Print object type, shape, and head
print("df type:", type(df))
print("df shape:", df.shape)
df.head()
```

```
df type: <class 'pandas.core.frame.DataFrame'>
df shape: (1736, 6)
```

[20]:

	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

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.

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.]()

[22]:
```
# Save df
df.to_csv("data/mexico-real-estate-clean.csv", index=False)
```

	property_type	state	lat	lon	area_m2	price_usd
1	house	Estado de México	19.560181	-99.233528	150.0	67965.56
2	house	Nuevo León	25.6884355	-100.1988071	186.0	63223.78
3	apartment	Guerrero	16.767704	-99.764383	82.0	84298.37
4	apartment	Guerrero	16.829782	-99.911012	150.0	94308.8
5	house	Yucatán	21.0525830247	-89.5386385918	205.0	105191.37
6	house	Querétaro	20.7163149	-100.4525027	320.0	274034.68
7	house	Morelos	18.8126047	-98.9548261	281.0	151509.56
8	house	Chiapas	16.769737	-93.088928	140.0	79029.72
9	house	Estado de México	19.305407331	-99.646948278	235.0	115937.75
10	house	Morelos	18.804197	-98.932816	117.0	63223.78
11	apartment	Guerrero	16.775165	-99.789939	117.0	157269.15
12	house	Yucatán	21.0483333	-89.6780555	193.0	104607.47
13	house	Estado de México	19.560181	-99.233528	85.0	63238.77
14	house	Tabasco	18.0140820847	-92.896399498	135.0	77994.48
15	apartment	Distrito Federal	19.390748	-99.158695	127.0	131716.2
16	apartment	Yucatán	21.3371507411	-89.3226885796	208.0	203167.1
17	house	Distrito Federal	19.337652	-99.2233268	297.0	264390.77
18	house	Nayarit	21.518174	-104.9074948	173.0	63238.77
19	house	Morelos	18.855343	-99.241142	73.0	36775.16
20	apartment	Puebla	19.0248763	-98.1945109	170.0	173570.3
21	apartment	Distrito Federal	19.403334	-99.157755	129.0	131716.2

1.3. Exploratory Data Analysis

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
from IPython.display import VimeoVideo
```

After importing, the next step in many data science projects is exploratory data analysis (EDA), where you get a feel for your data by summarizing its main characteristics using descriptive statistics and data visualization. A good way to plan your EDA is by looking each column and asking yourself questions what it says about your dataset.

Import Data

Task 1.3.1: Read the CSV file that you created in the last notebook ("../small-data/mexico-real-estate-clean.csv") into a DataFrame named df. Be sure to check that all your columns are the correct data type before you go to the next task.

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

[4]:
```
# Import "data/mexico-real-estate-clean.csv"
df = pd.read_csv("data/mexico-real-estate-clean.csv")
# Print object type, shape, and head
print("df type:", type(df))
print("df shape:", df.shape)
df.head()
```

```
df type: <class 'pandas.core.frame.DataFrame'>
df shape: (1736, 6)
```

[4]:

	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
4	house	Yucatán	21.052583	-89.538639	205.0	105191.37

While there are only two dtypes in our DataFrame (object and float64), there are three categories of data: location, categorical, and numeric. Each of these require a different kind of exploration in our analysis.

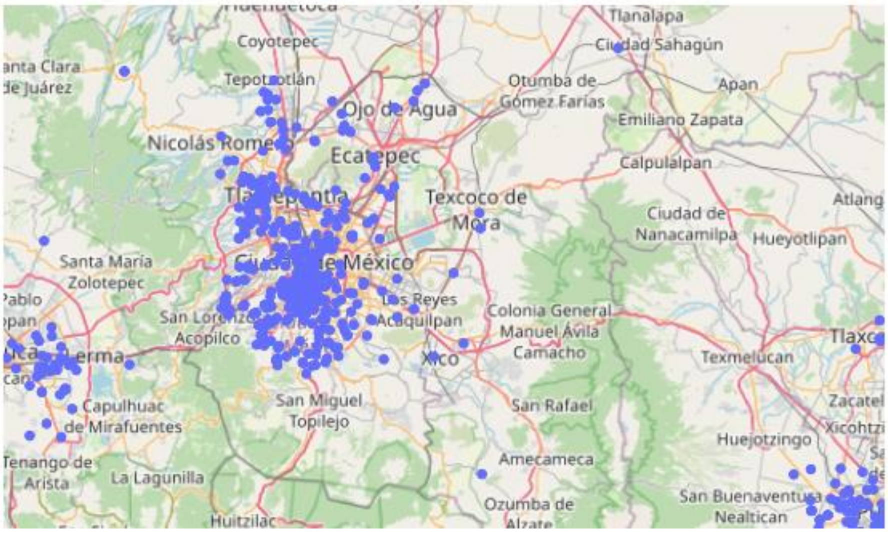
Location Data: "lat" and "lon"

They say that the most important thing in real estate is location, and we can see where where in Mexico our houses are located by using the "lat" and "lon" columns. Since latitude and longitude are based on a coordinate system, a good way to visualize them is to create a scatter plot on top of a map. A great tool for this is the scatter_mapbox from the plotly library.

Task 1.3.2: Add "lat" and "lon" to the code below, and run the code. You'll see a map that's centered on Mexico City, and you can use the "Zoom Out" button in the upper-right corner of the map so that you can see the whole country.

- [What's location data?]()
- [What's a scatter plot?]()

```
[6]: # Use plotly express to create figure
fig = px.scatter_mapbox(
    df,  # Our DataFrame
    lat="lat",
    lon="lon",
    center={"lat": 19.43, "lon": -99.13},  # Map will be centered on Mexico City
    width=600,  # Width of map
    height=600,  # Height of map
    hover_data=["price_usd"],  # Display price when hovering mouse over house
)

# Add mapbox_style to figure layout
fig.update_layout(mapbox_style="open-street-map")

# Show figure
fig.show()
```



Looking at this map, are the houses in our dataset distributed evenly throughout the country, or are there states or regions that are more prevalent? Can you guess where Mexico's biggest cities are based on this distribution?

Categorical Data: "state"

Even though we can get a good idea of which states are most common in our dataset from looking at a map, we can also get the exact count by using the "state" column.

Task 1.3.3: Use the value_counts method on the "state" column to determine the 10 most prevalent states in our dataset.

- [What's categorical data?]()
- [What's a Series?]()
- [Aggregate data in a Series using value_counts in pandas.]()

[8]:
```
# Get value counts of "state" column
df['state'].value_counts().head(10)
```

[8]:
```
Distrito Federal                  303
Estado de México                  179
Yucatán                           171
Morelos                           160
Querétaro                         128
Veracruz de Ignacio de la Llave   117
Puebla                             95
Nuevo León                         83
Jalisco                            60
San Luis Potosí                    55
Name: state, dtype: int64
```

Numerical Data: "area_m2" and "price_usd"

We have a sense for where the houses in our dataset are located, but how much do they cost? How big are they? The best way to answer those questions is looking at descriptive statistics.

Task 1.3.4: Use the describe method to print the mean, standard deviation, and quartiles for the "area_m2" and "price_usd" columns.

- What's numerical data?
- What's a mean?
- What's a standard deviation?
- What are quartiles?
- Print the summary statistics for a DataFrame using pandas.

[11]:
```
# Describe "area_m2", "price_usd" columns
df[['area_m2','price_usd']].describe()
```

[11]:

	area_m2	price_usd
count	1736.000000	1736.000000
mean	170.261521	115331.980766
std	80.594539	65426.173873
min	60.000000	33157.890000
25%	101.750000	65789.470000
50%	156.000000	99262.130000
75%	220.000000	150846.665000
max	385.000000	326733.660000

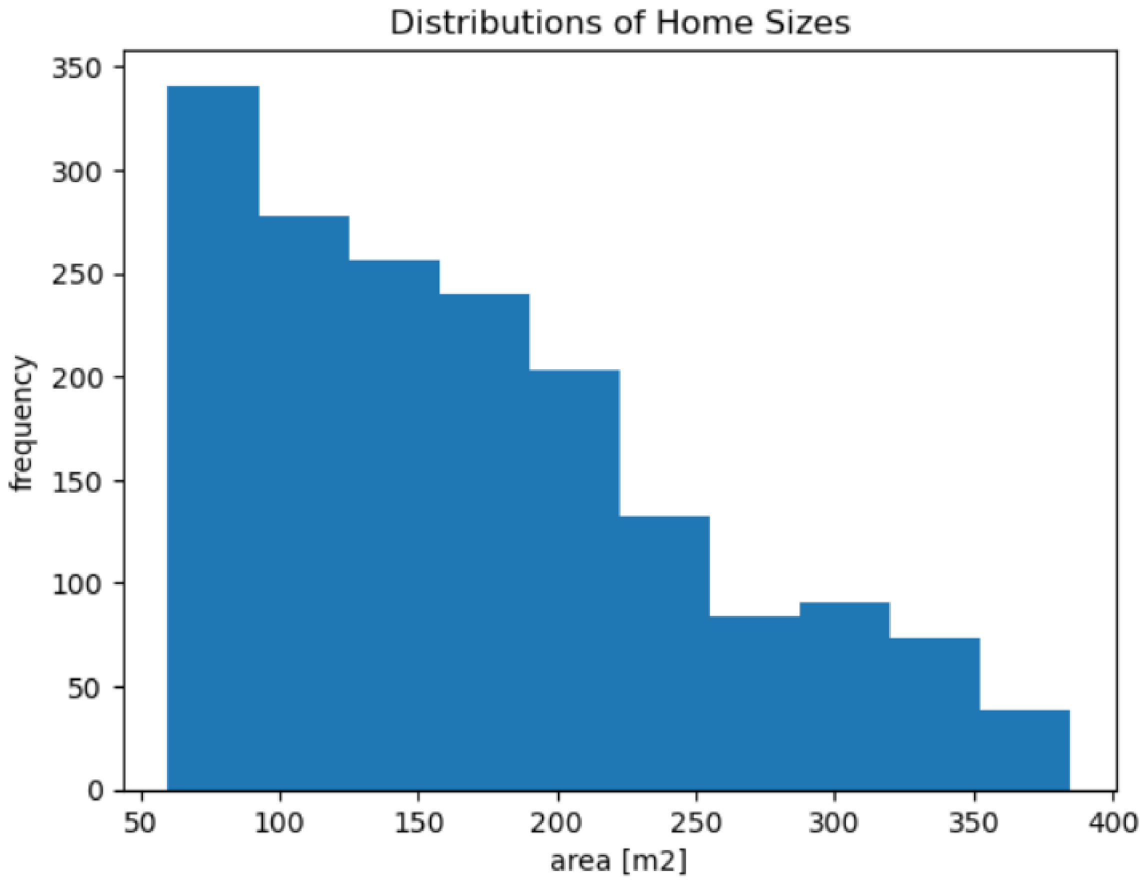
Let's start by looking at "area_m2". It's interesting that the mean is larger than the median (another name for the 50% quartile). Both of these statistics are supposed to give an idea of the "typical" value for the column, so why is there a difference of almost 15 m^2 between them? To answer this question, we need to see how house sizes are distributed in our dataset. Let's look at two ways to visualize the distribution: a histogram and a boxplot.

Task 1.3.5: Create a histogram of "area_m2". Make sure that the x-axis has the label "Area [sq meters]", the y-axis has the label "Frequency", and the plot has the title "Distribution of Home Sizes".

- [What's a histogram?]()
- [Create a histogram using Matplotlib.]()

[16]:
```
# Use Matplotlib to create histogram of "area_m2"
plt.hist(df['area_m2']);

# Add x-axis label
plt.xlabel("area [m2]")

# Add y-axis label
plt.ylabel("frequency")

# Add title
plt.title("Distributions of Home Sizes")
```

[16]: `Text(0.5, 1.0, 'Distributions of Home Sizes')`

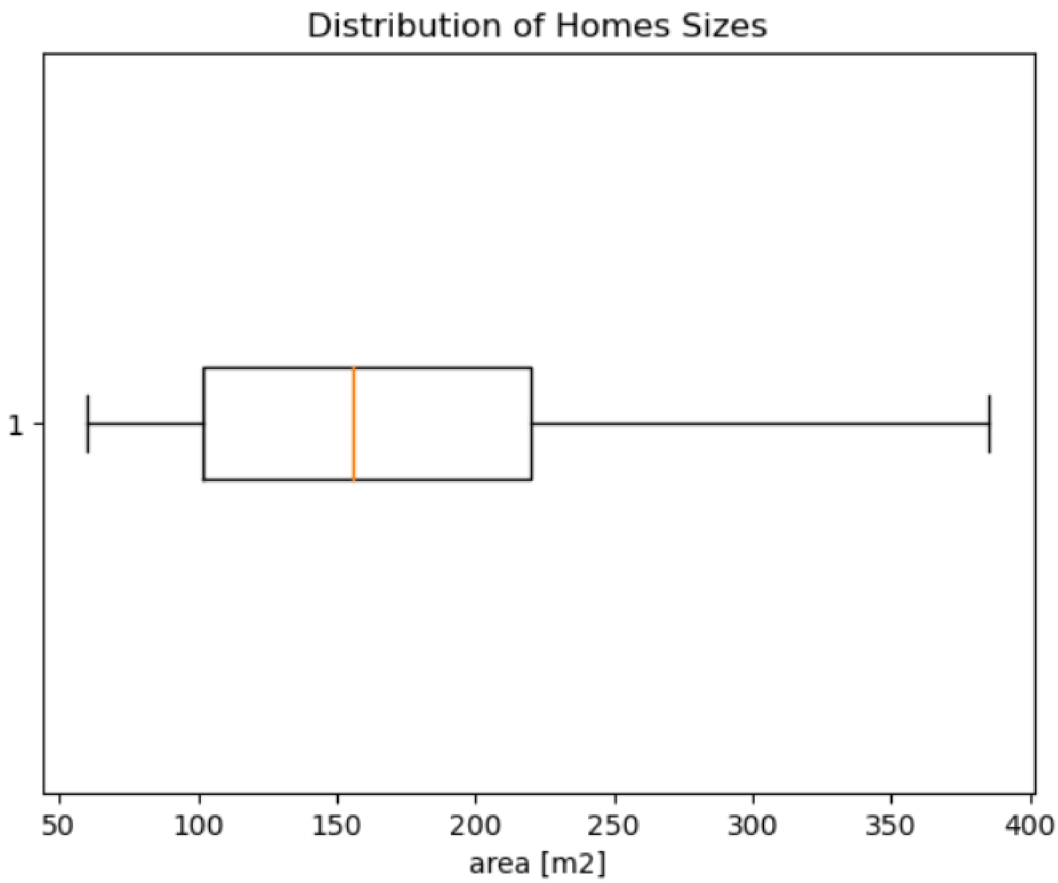


Looking at our histogram, we can see that "area_m2" skews right. In other words, there are more houses at the lower end of the distribution (50–200m^2) than at the higher end (250–400m^2). That explains the difference between the mean and the median.

Task 1.3.6: Create a horizontal boxplot of "area_m2". Make sure that the x-axis has the label "Area [sq meters]" and the plot has the title "Distribution of Home Sizes". How is the distribution and its left skew represented differently here than in your histogram?

- What's a boxplot?
- What's a skewed distribution?
- Create a boxplot using Matplotlib.

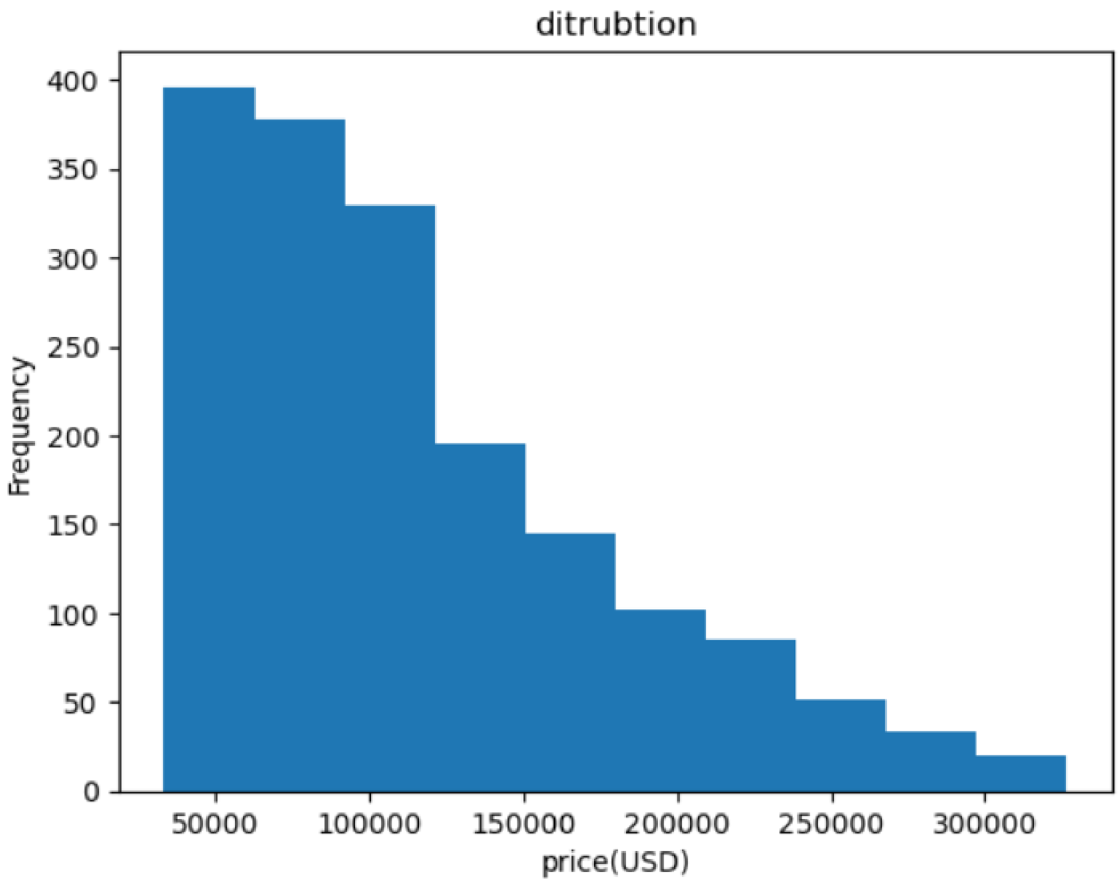
[18]:
```
# Use Matplotlib to create boxplot of "area_m2"
plt.boxplot(df['area_m2'], vert= False)

# Add x-axis label
plt.xlabel("area [m2]")

# Add title
plt.title("Distribution of Homes Sizes");
```



Task 1.3.7: Create a histogram of "price_usd". Make sure that the x-axis has the label "Price [USD]", the y-axis has the label "Frequency", and the plot has the title "Distribution of Home Prices".

- What's a histogram?
- Create a histogram using Matplotlib.

[21]:
```python
# Use Matplotlib to create histogram of "price_usd"
plt.hist(df['price_usd'])

# Add x-axis label
plt.xlabel('price(USD)')

# Add y-axis label
plt.ylabel('Frequency')

# Add title
plt.title('ditrubtion');
```

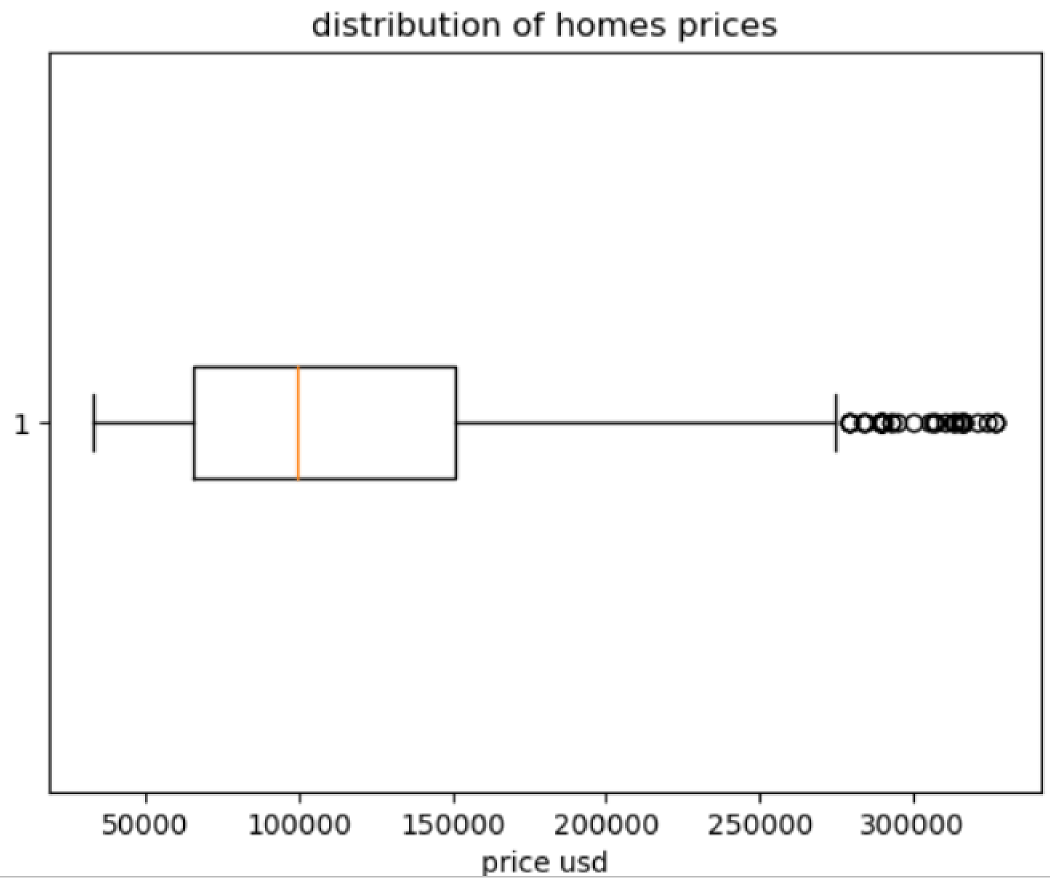


Looks like "price_usd" is even more skewed than "area_m2". What does this bigger skew look like in a boxplot?

Task 1.3.8: Create a horizontal boxplot of "price_usd". Make sure that the x-axis has the label "Price [USD]" and the plot has the title "Distribution of Home Prices".

- [What's a boxplot?]()
- [What's an outlier?]()
- [Create a boxplot using Matplotlib.]()

[24]:
```python
# Use Matplotlib to create boxplot of "price_usd"
plt.boxplot(df['price_usd'], vert = False)

# Add x-label axis
plt.xlabel('price usd')

# Add y-label axis
plt.title('distribution of homes prices');
```



Excellent job! Now that you have a sense of for the dataset, let's move to the next notebook and start answering some research questions about the relationship between house size, price, and location.

1.4. Location or Size: What Influences House Prices in Mexico?

[1]:
```python
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import VimeoVideo
```

Import Data

Task 1.4.1: Read the CSV file that you created in the last notebook ("data/mexico-real-estate-clean.csv") into a DataFrame named df. Be sure to check that all your columns are the correct data type before you go to the next task.

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

[2]:
```python
# Import "data/mexico-real-estate-clean.csv"
df = pd.read_csv("data/mexico-real-estate-clean.csv")

# Print object type, shape, and head
print("df type:", type(df))
print("df shape:", df.shape)
df.head()
```

```
df type: <class 'pandas.core.frame.DataFrame'>
df shape: (1736, 6)
```

[2]:

	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
4	house	Yucatán	21.052583	-89.538639	205.0	105191.37

Research Question 1

Which state has the most expensive real estate market?

Do housing prices vary by state? If so, which are the most expensive states for purchasing a home? During our exploratory data analysis, we used descriptive statistics like mean and median to get an idea of the "typical" house price in Mexico. Now, we need to break that calculation down by state and visualize the results.

We know in which state each house is located thanks to the "state" column. The next step is to divide our dataset into groups (one per state) and calculate the mean house price for each group.

Task 1.4.2: Use the groupby method to create a Series named mean_price_by_state, where the index contains each state in the dataset and the values correspond to the mean house price for that state. Make sure your Series is sorted from highest to lowest mean price.

- What's a Series?
- Aggregate data using the groupby method in pandas.

[4]:
```
# Declare variable `mean_price_by_state`
mean_price_by_state = df.groupby('state')['price_usd'].mean().sort_values(ascending=False)

# Print object type, shape, and head
print("mean_price_by_state type:", type(mean_price_by_state))
print("mean_price_by_state shape:", mean_price_by_state.shape)
mean_price_by_state.head()
```

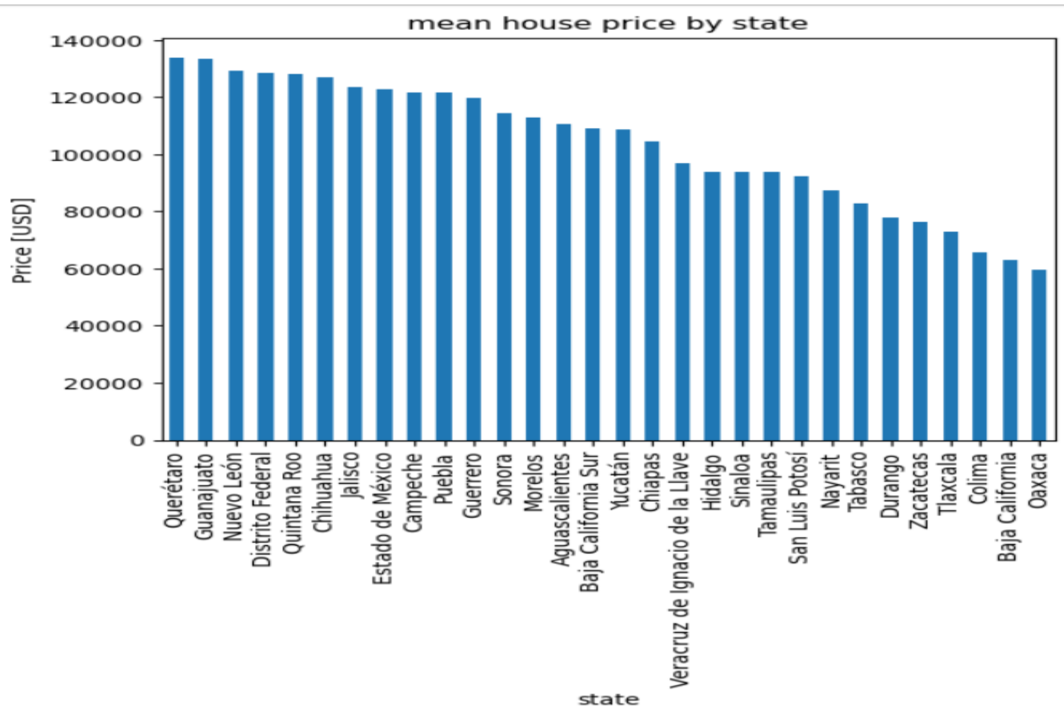
```
mean_price_by_state type: <class 'pandas.core.series.Series'>
mean_price_by_state shape: (30,)
```

[4]:
```
state
Querétaro          133955.913281
Guanajuato         133277.965833
Nuevo León         129221.985663
Distrito Federal   128347.267426
Quintana Roo       128065.416053
Name: price_usd, dtype: float64
```

Task 1.4.3: Use mean_price_by_state to create a bar chart of your results. Make sure the states are sorted from the highest to lowest mean, that you label the x-axis as "State" and the y-axis as "Mean Price [USD]", and give the chart the title "Mean House Price by State".

- Create a bar chart using pandas.

[6]:
```
# Create bar chart from `mean_price_by_state` using pandas
mean_price_by_state.plot(
    kind='bar',
    xlabel='state',
    ylabel='Price [USD]',
    title='mean house price by state'
);
```



It seems odd that Querétaro would be the most expensive real estate market in Mexico when, according to recent GDP numbers, it's not in the top 10 state economies. With all the variations in house sizes across states, a better metric to look at would be price per m^2. In order to do that, we need to create a new column.

Task 1.4.4: Create a new column in df called "price_per_m2". This should be the price for each house divided by it's size.

- Create new columns derived from existing columns in a DataFrame using pandas.

[8]:
```
# Create "price_per_m2" column
df["price_per_m2"] = df['price_usd'] / df['area_m2']

# Print object type, shape, and head
print("df type:", type(df))
print("df shape:", df.shape)
df.head()
```

```
df type: <class 'pandas.core.frame.DataFrame'>
df shape: (1736, 7)
```

[8]:

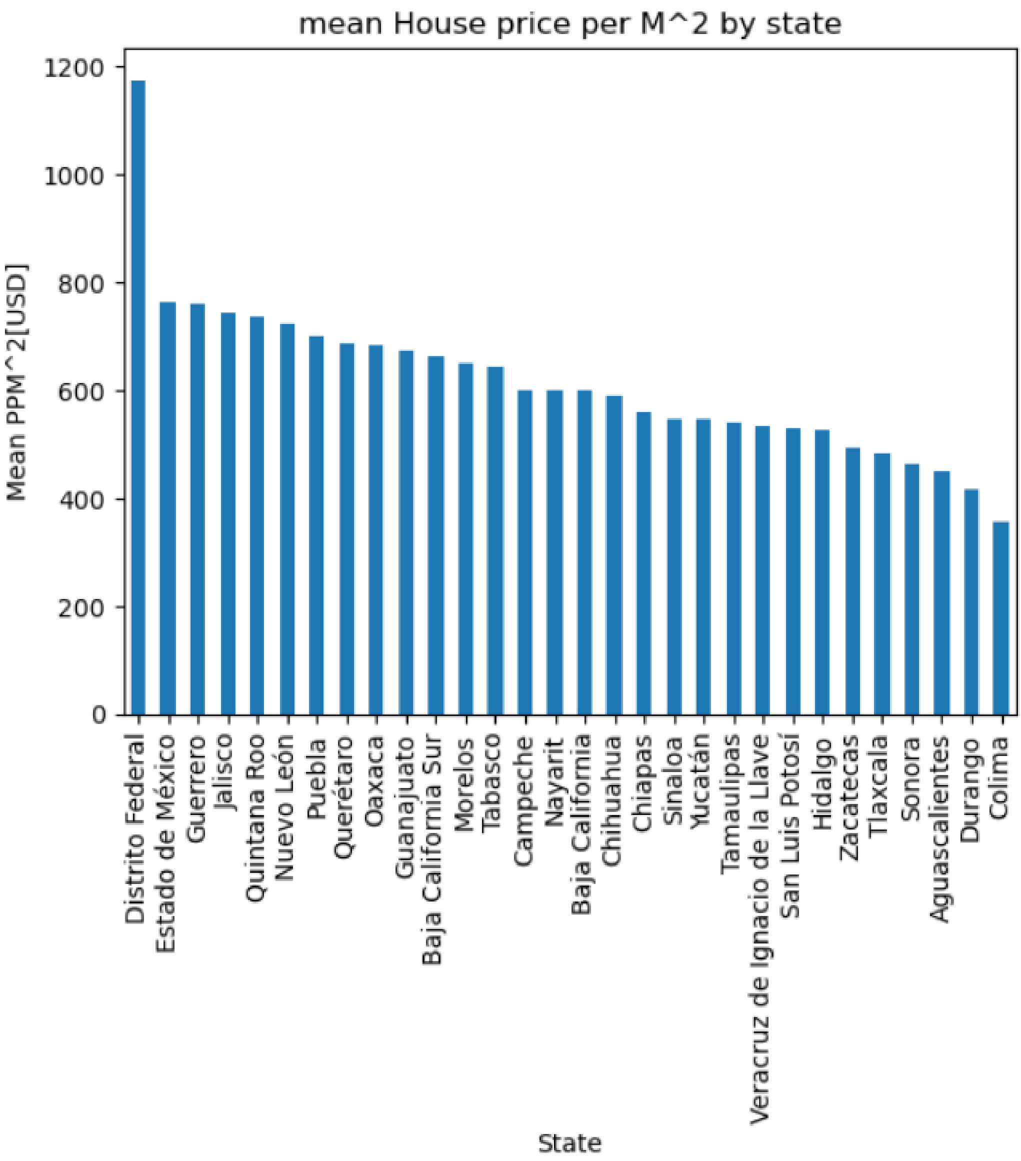
	property_type	state	lat	lon	area_m2	price_usd	price_per_m2
0	house	Estado de México	19.560181	-99.233528	150.0	67965.56	453.103733
1	house	Nuevo León	25.688436	-100.198807	186.0	63223.78	339.912796
2	apartment	Guerrero	16.767704	-99.764383	82.0	84298.37	1028.028902
3	apartment	Guerrero	16.829782	-99.911012	150.0	94308.80	628.725333
4	house	Yucatán	21.052583	-89.538639	205.0	105191.37	513.128634

Let's redo our bar chart from above, but this time with the mean of "price_per_m2" for each state.

Task 1.4.5: First, use the groupby method to create a Series where the index contains each state in the dataset and the values correspond to the mean house price per m^2 for that state. Then use the Series to create a bar chart of your results. Make sure the states are sorted from the highest to lowest mean, that you label the x-axis as "State" and the y-axis as "Mean Price per M^2[USD]", and give the chart the title "Mean House Price per M^2 by State".

- What's a Series?
- Aggregate data using the groupby method in pandas.
- Create a bar chart using pandas.

[10]:
```
# Group `df` by "state", create bar chart of "price_per_m2"
(
    df
    .groupby('state')
    ['price_per_m2'].mean()
    .sort_values(ascending= False)
    .plot(
            kind ="bar",
            xlabel="State",
            ylabel="Mean PPM^2[USD]",
            title="mean House price per M^2 by state"
    )
);
```



Now we see that the capital Mexico City (*Distrito Federal*) is by far the most expensive market. Additionally, many of the top 10 states by GDP are also in the top 10 most expensive real estate markets. So it looks like this bar chart is a more accurate reflection of state real estate markets.

Research Question 2

Is there a relationship between home size and price?

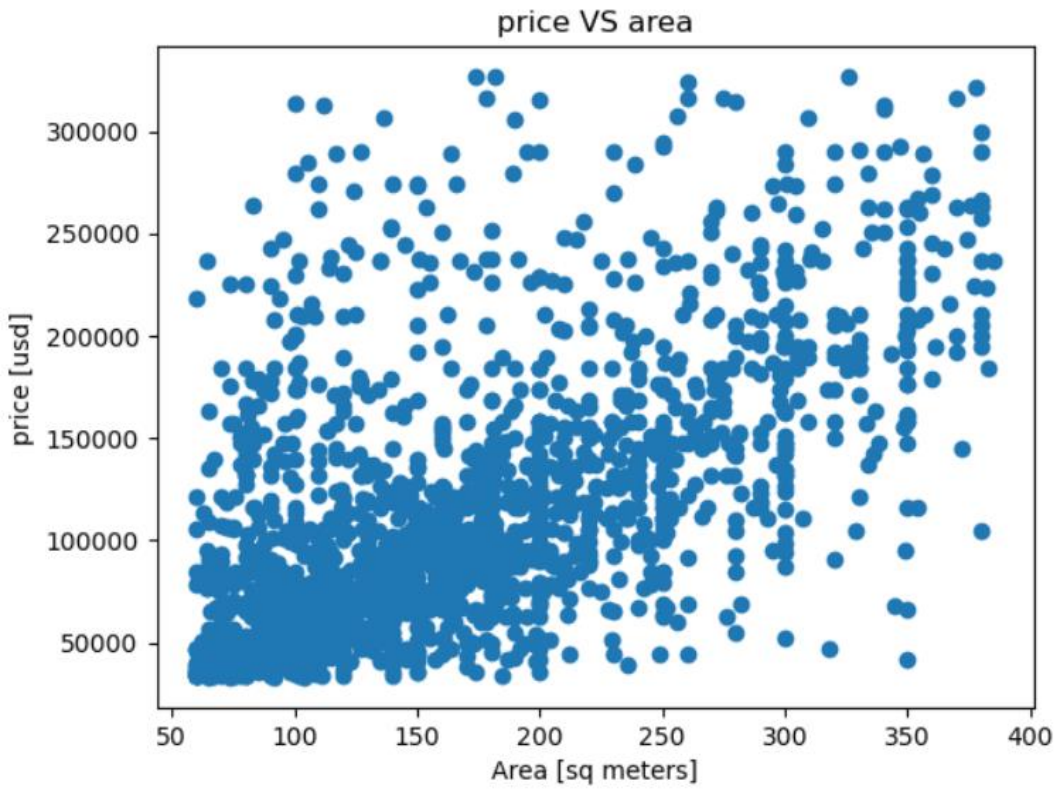
From our previous question, we know that the location of a home affects its price (especially if it's in Mexico City), but what about home size? Does the size of a house influence price?

A scatter plot can be helpful when evaluating the relationship between two columns because it lets you see if two variables are correlated — in this case, if an increase in home size is associated with an increase in price.

Task 1.4.6: Create a scatter plot from `df` that represents price as a function of size. In other words, `"area_m2"` should be on the x-axis, and `"price_usd"` should be on the y-axis. Be sure to use expressive axis labels (`"Area [sq meters]"` and `"Price [USD]"`, respectively).

- What's a scatter plot?
- What's correlation?
- Create a scatter plot using Matplotlib.

[12]:
```
# Create scatter plot of "price_usd" vs "area_m2"
plt.scatter(x=df['area_m2'], y=df['price_usd'])

# Add x-axis label
plt.xlabel('Area [sq meters]')

# Add y-axis label
plt.ylabel('price [usd]')

# Add title
plt.title('price VS area');
```



While there's a good amount of variation, there's definitely a positive correlation — in other words, the bigger the house, the higher the price. But how can we quantify this correlation?

Task 1.4.7: Using the corr method, calculate the Pearson correlation coefficient for "area_m2" and "price_usd".

- [What's a correlation coefficient?]
- [Calculate the correlation coefficient for two Series using pandas.]

[14]:
```
# Calculate correlation of "price_usd" and "area_m2"
p_correlation = df['area_m2'].corr(df['price_usd'])

# Print correlation coefficient
print("Correlation of 'area_m2' and 'price_usd' (all Mexico):", p_correlation)
```

```
Correlation of 'area_m2' and 'price_usd' (all Mexico): 0.5855182453232062
```

The correlation coefficient is over 0.5, so there's a moderate relationship house size and price in Mexico. But does this relationship hold true in every state? Let's look at a couple of states, starting with Morelos.

Task 1.4.8: Create a new DataFrame named df_morelos. It should include all the houses from df that are in the state of Morelos.

- [Subset a DataFrame with a mask using pandas.]

[16]:
```
# Declare variable `df_morelos` by subsetting `df`
df_morelos = df[df['state']=='Morelos']

# Print object type, shape, and head
print("df_morelos type:", type(df_morelos))
print("df_morelos shape:", df_morelos.shape)
df_morelos.head()
```

```
df_morelos type: <class 'pandas.core.frame.DataFrame'>
df_morelos shape: (160, 7)
```

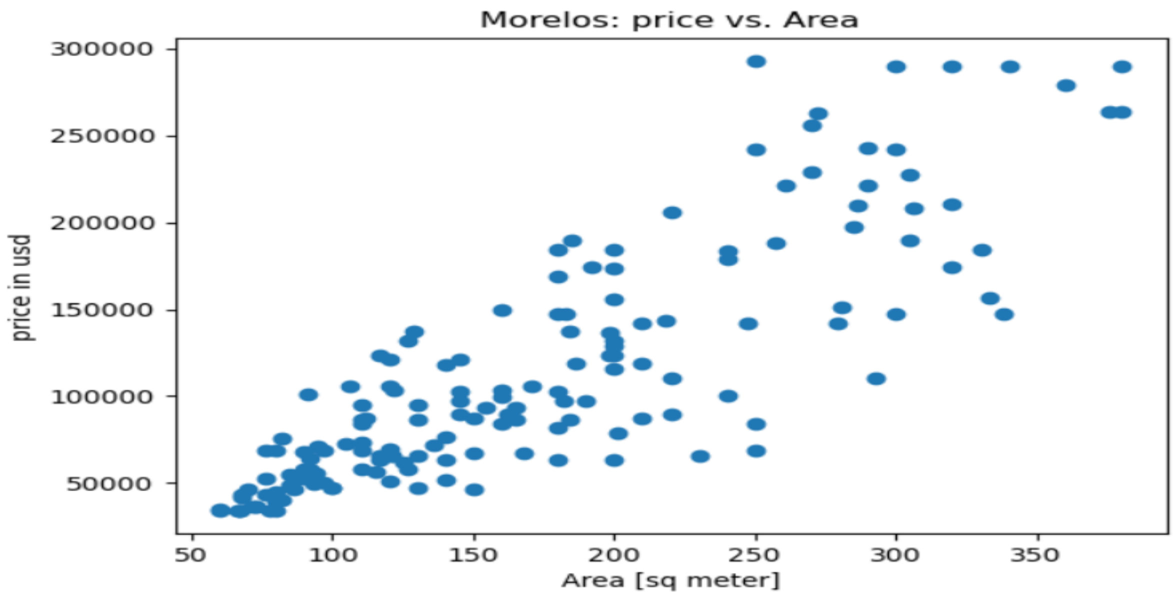
[16]:

	property_type	state	lat	lon	area_m2	price_usd	price_per_m2
6	house	Morelos	18.812605	-98.954826	281.0	151509.56	539.179929
9	house	Morelos	18.804197	-98.932816	117.0	63223.78	540.374188
18	house	Morelos	18.855343	-99.241142	73.0	36775.16	503.769315
49	house	Morelos	18.804197	-98.932816	130.0	65858.10	506.600769
55	house	Morelos	18.960244	-99.212962	305.0	227351.46	745.414623

Task 1.4.9: Using df_morelos, create a scatter plot that shows price vs area. Make sure to use the same axis labels as your last scatter plot. The title should be "Morelos: Price vs. Area".

- What's a scatter plot?
- Create a scatter plot using Matplotlib.

[19]:
```
# Create scatter plot of "price_usd" vs "area_m2" in Morelos
plt.scatter(x=df_morelos['area_m2'], y=df_morelos['price_usd'])
# Add x-axis label
plt.xlabel('Area [sq meter]')

# Add y-axis label
plt.ylabel('price in usd')

# Add title
plt.title('Morelos: price vs. Area');
```



It looks like the correlation is even stronger within Morelos. Let's calculate the correlation coefficient and verify that that's the case.

Task 1.4.10: Using the `corr` method, calculate the Pearson correlation coefficient for "area_m2" and "price_usd" in df_morelos.

- What's a correlation coefficient?
- Calculate the correlation coefficient for two Series using pandas.

[22]:
```
# Calculate correlation of "price_usd" and "area_m2" in `df_morelos`
p_correlation = df_morelos['area_m2'].corr(df_morelos['price_usd'])

# Print correlation coefficient
print("Correlation of 'area_m2' and 'price_usd' (Morelos):", p_correlation)
```

```
Correlation of 'area_m2' and 'price_usd' (Morelos): 0.8498077608713708
```

With a correlation coefficient that high, we can say that there's a strong relationship between house size and price in Morelos.

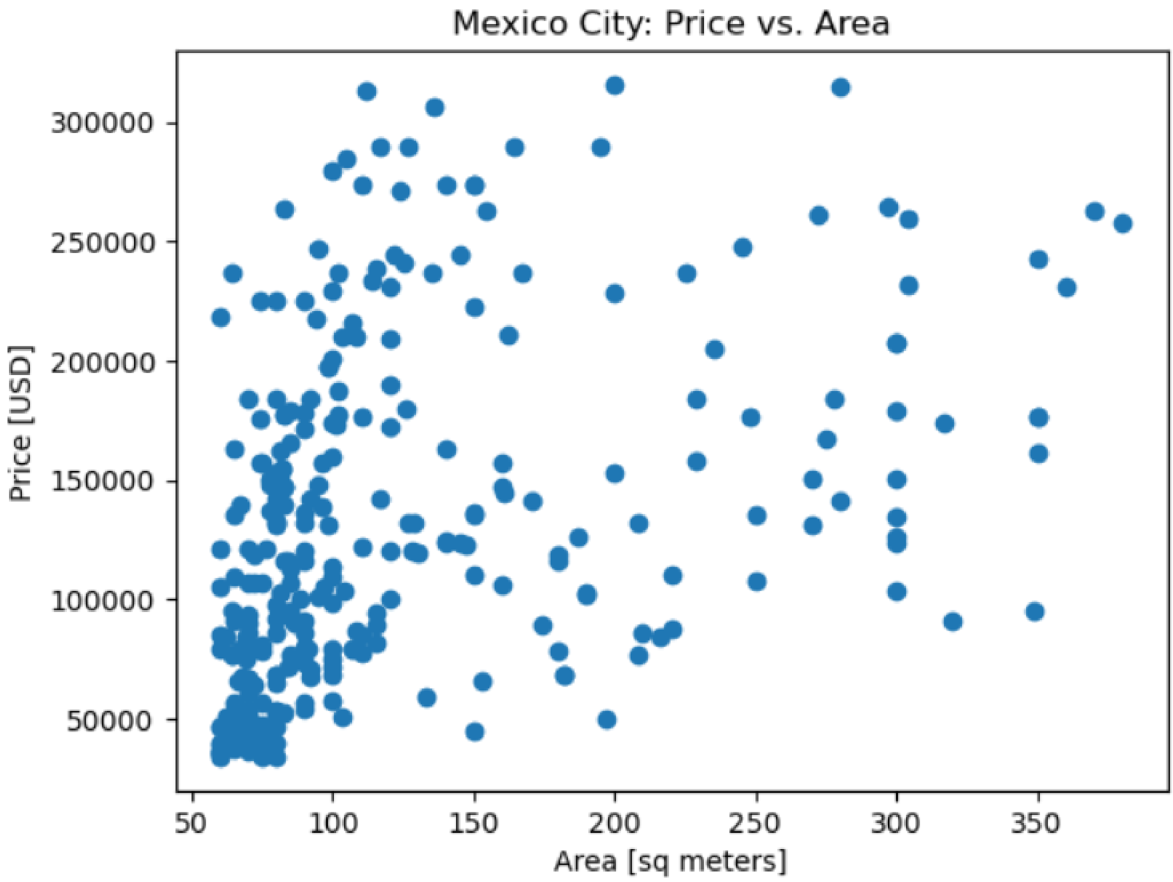
To conclude, let's look at the capital Mexico City (*Distrito Federal*).

Task 1.4.11: First, create a new DataFrame called df_mexico_city that includes all the observations from df that are part of the *Distrito Federal*. Next, create a scatter plot that shows price vs area. Don't forget to label the x- and y-axis and use the title "Mexico City: Price vs. Area". Finally, calculate the correlation coefficient for "area_m2" and "price_usd" in df_mexico_city.

- Calculate the correlation coefficient for two Series using pandas.
- Create a scatter plot using Matplotlib.
- Subset a DataFrame with a mask using pandas.

[25]:
```
# Declare variable `df_mexico_city` by subsetting `df`
df_mexico_city = df[df['state']=='Distrito Federal']

# Print object type and shape
print("df_mexico_city type:", type(df_mexico_city))
print("df_mexico_city shape:", df_mexico_city.shape)

# Create a scatter plot "price_usd" vs "area_m2" in Distrito Federal
plt.scatter(df_mexico_city["area_m2"], df_mexico_city["price_usd"])  # REMOVERHS

# Add x-axis label
plt.xlabel("Area [sq meters]")  # REMOVERHS

# Add y-axis label
plt.ylabel("Price [USD]")  # REMOVERHS

# Add title
plt.title("Mexico City: Price vs. Area")  # REMOVERHS

# Calculate correlation of "price_usd" and "area_m2" in `df_mexico_city`
p_correlation = df_mexico_city['area_m2'].corr(df_mexico_city['price_usd'])

# Print correlation coefficient
print("Correlation of 'area_m2' and 'price_usd' (Mexico City):", p_correlation)
```

```
df_mexico_city type: <class 'pandas.core.frame.DataFrame'>
df_mexico_city shape: (303, 7)
Correlation of 'area_m2' and 'price_usd' (Mexico City): 0.41070392130717887
```



Looking at the scatter plot and correlation coefficient, there's see a weak relationship between size and price. How should we interpret this?

One interpretation is that the relationship we see between size and price in many states doesn't hold true in the country's biggest and most economically powerful urban center because there are other factors that have a larger influence on price. In fact, in the next project, we're going to look at another important Latin American city — Buenos Aires, Argentina — and build a model that predicts housing price by taking much more than size into account.

1.5. Housing in Brazil BR

[1]:
```python
import wqet_grader

wqet_grader.init("Project 1 Assessment")
```

In this assignment, you'll work with a dataset of homes for sale in Brazil. Your goal is to determine if there are regional differences in the real estate market. Also, you will look at southern Brazil to see if there is a relationship between home size and price, similar to what you saw with housing in some states in Mexico.

Import the libraries you'll use in this notebook: Matplotlib, pandas, and plotly. Be sure to import them under the aliases we've used in this project.

[2]:
```python
# Import Matplotlib, pandas, and plotly
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
```

Prepare Data

In this assignment, you'll work with real estate data from Brazil. In the data directory for this project there are two CSV that you need to import and clean, one-by-one.

Import

First, you are going to import and clean the data in data/brasil-real-estate-1.csv.

Task 1.5.1: Import the CSV file data/brasil-real-estate-1.csv into the DataFrame df1.

[3]:
```python
df1 = pd.read_csv("data/brasil-real-estate-1.csv")
df1.head()
```

[3]:

	property_type	place_with_parent_names	region	lat-lon	area_m2	price_usd
0	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.6443051,-35.7088142	110.0	$187,230.85
1	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.6430934,-35.70484	65.0	$81,133.37
2	house	|Brasil|Alagoas|Maceió|	Northeast	-9.6227033,-35.7297953	211.0	$154,465.45
3	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.622837,-35.719556	99.0	$146,013.20
4	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.654955,-35.700227	55.0	$101,416.71

[4]:
```python
wqet_grader.grade("Project 1 Assessment", "Task 1.5.1", df1)
```

Yup. You got it.

Score: 1

Before you move to the next task, take a moment to inspect df1 using the info and head methods. What issues do you see in the data? What cleaning will you need to do before you can conduct your analysis?

```
[5]: print("info of  df1",df1.info)
     print("shape of df1",df1.shape)
     df1.head()
```

```
info of  df1 <bound method DataFrame.info of        property_type                place_with_parent_names     region  \
0          apartment                  |Brasil|Alagoas|Maceió|   Northeast
1          apartment                  |Brasil|Alagoas|Maceió|   Northeast
2              house                  |Brasil|Alagoas|Maceió|   Northeast
3          apartment                  |Brasil|Alagoas|Maceió|   Northeast
4          apartment                  |Brasil|Alagoas|Maceió|   Northeast
...              ...                                      ...         ...
12829      apartment               |Brasil|Pernambuco|Recife|   Northeast
12830      apartment               |Brasil|Pernambuco|Recife|   Northeast
12831      apartment    |Brasil|Pernambuco|Recife|Boa Viagem|   Northeast
12832      apartment    |Brasil|Pernambuco|Recife|Boa Viagem|   Northeast
12833      apartment    |Brasil|Pernambuco|Recife|Boa Viagem|   Northeast

                      lat-lon  area_m2    price_usd
0      -9.6443051,-35.7088142    110.0  $187,230.85
1        -9.6430934,-35.70484     65.0   $81,133.37
2      -9.6227033,-35.7297953    211.0  $154,465.45
3        -9.622837,-35.719556     99.0  $146,013.20
4        -9.654955,-35.700227     55.0  $101,416.71
...                       ...      ...          ...
12829    -8.056418,-34.909309     91.0  $174,748.79
12830   -8.1373477,-34.909181    115.0  $115,459.02
12831   -8.1136717,-34.896252     76.0  $137,302.62
12832                     NaN    130.0  $234,038.56
12833   -8.0578381,-34.882897     99.0  $168,507.77

[12834 rows x 6 columns]>
shape of df1 (12834, 6)
```

[5]:

	property_type	place_with_parent_names	region	lat-lon	area_m2	price_usd
0	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.6443051,-35.7088142	110.0	$187,230.85
1	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.6430934,-35.70484	65.0	$81,133.37
2	house	|Brasil|Alagoas|Maceió|	Northeast	-9.6227033,-35.7297953	211.0	$154,465.45
3	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.622837,-35.719556	99.0	$146,013.20
4	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.654955,-35.700227	55.0	$101,416.71

Task 1.5.2: Drop all rows with NaN values from the DataFrame df1.

```
[6]: df1.dropna(inplace=True)
     print("the new shape is ", df1.info)
```

```
the new shape is  <bound method DataFrame.info of        property_type                place_with_parent_names     region  \
0          apartment                  |Brasil|Alagoas|Maceió|   Northeast
1          apartment                  |Brasil|Alagoas|Maceió|   Northeast
2              house                  |Brasil|Alagoas|Maceió|   Northeast
3          apartment                  |Brasil|Alagoas|Maceió|   Northeast
4          apartment                  |Brasil|Alagoas|Maceió|   Northeast
...              ...                                      ...         ...
12828      apartment               |Brasil|Pernambuco|Recife|   Northeast
12829      apartment               |Brasil|Pernambuco|Recife|   Northeast
12830      apartment               |Brasil|Pernambuco|Recife|   Northeast
12831      apartment    |Brasil|Pernambuco|Recife|Boa Viagem|   Northeast
12833      apartment    |Brasil|Pernambuco|Recife|Boa Viagem|   Northeast

                      lat-lon  area_m2    price_usd
0      -9.6443051,-35.7088142    110.0  $187,230.85
1        -9.6430934,-35.70484     65.0   $81,133.37
2      -9.6227033,-35.7297953    211.0  $154,465.45
3        -9.622837,-35.719556     99.0  $146,013.20
4        -9.654955,-35.700227     55.0  $101,416.71
...                       ...      ...          ...
12828    -8.044497,-34.909519     74.0  $134,182.11
12829    -8.056418,-34.909309     91.0  $174,748.79
12830   -8.1373477,-34.909181    115.0  $115,459.02
12831   -8.1136717,-34.896252     76.0  $137,302.62
12833   -8.0578381,-34.882897     99.0  $168,507.77

[11551 rows x 6 columns]>
```

Act

Task 1.5.3: Use the `"lat-lon"` column to create two separate columns in `df1` : `"lat"` and `"lon"` . Make sure that the data type for these new columns is `float` .

[8]:
```
df1[['lat','lon']]=df1['lat-lon'].str.split(',',2, expand=True)
df1['lat']=df1['lat'].astype(float)
df1['lon']=df1['lon'].astype(float)
df1.head()
```

```
/tmp/ipykernel_163/2096937468.py:1: FutureWarning: In a future version of pandas all arguments of StringMethods.split except for the argument 'pat' wil
l be keyword-only.
  df1[['lat','lon']]=df1['lat-lon'].str.split(',',2, expand=True)
```

[8]:

	property_type	place_with_parent_names	region	lat-lon	area_m2	price_usd	lat	lon
0	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.6443051,-35.7088142	110.0	$187,230.85	-9.644305	-35.708814
1	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.6430934,-35.70484	65.0	$81,133.37	-9.643093	-35.704840
2	house	|Brasil|Alagoas|Maceió|	Northeast	-9.6227033,-35.7297953	211.0	$154,465.45	-9.622703	-35.729795
3	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.622837,-35.719556	99.0	$146,013.20	-9.622837	-35.719556
4	apartment	|Brasil|Alagoas|Maceió|	Northeast	-9.654955,-35.700227	55.0	$101,416.71	-9.654955	-35.700227

Task 1.5.4: Use the `"place_with_parent_names"` column to create a `"state"` column for `df1` . (Note that the state name always appears after `"|Brasil|"` in each string.)

[10]:
```
df1['state'] = df1['place_with_parent_names'].str.split('|').str[2]
```

[11]:
```
wqet_grader.grade("Project 1 Assessment", "Task 1.5.4", df1)
```

Yes! Your hard work is paying off.
Score: 1

Task 1.5.5: Transform the `"price_usd"` column of `df1` so that all values are floating-point numbers instead of strings.

[12]:
```
df1['price_usd']= df1['price_usd'].str.replace(',','', regex=True)
df1['price_usd']= df1['price_usd'].str.replace('$','', regex=True)
df1['price_usd']= df1['price_usd'].astype(float)
```

Task 1.5.5: Transform the `"price_usd"` column of `df1` so that all values are floating-point numbers instead of strings.

[12]:
```
df1['price_usd']= df1['price_usd'].str.replace(',','', regex=True)
df1['price_usd']= df1['price_usd'].str.replace('$','', regex=True)
df1['price_usd']= df1['price_usd'].astype(float)
```


[13]:
```
wqet_grader.grade("Project 1 Assessment", "Task 1.5.5", df1)
```

Wow, you're making great progress.
Score: 1

Task 1.5.6: Drop the `"lat-lon"` and `"place_with_parent_names"` columns from `df1` .

[14]:
```
df1.drop(columns=['lat-lon','place_with_parent_names'], inplace=True)
```

[15]:
```
wqet_grader.grade("Project 1 Assessment", "Task 1.5.6", df1)
```

Awesome work.
Score: 1

Now that you have cleaned `data/brasil-real-estate-1.csv` and created `df1` , you are going to import and clean the data from the second file, `brasil-real-estate-2.csv` .

Task 1.5.7: Import the CSV file `brasil-real-estate-2.csv` into the DataFrame `df2` .

[16]:
```
df2 = pd.read_csv('data/brasil-real-estate-2.csv')
```

Before you jump to the next task, take a look at `df2` using the `info` and `head` methods. What issues do you see in the data? How is it similar or different from `df1` ?

[18]:
```
print('df2 info is', df2.info)
df2.head()
```

```
df2 info is <bound method DataFrame.info of        property_type      state      region        lat        lon  area_m2  \
0          apartment  Pernambuco  Northeast  -8.134204 -34.906326     72.0
1          apartment  Pernambuco  Northeast  -8.126664 -34.903924    136.0
2          apartment  Pernambuco  Northeast  -8.125550 -34.907601     75.0
3          apartment  Pernambuco  Northeast  -8.120249 -34.895920    187.0
4          apartment  Pernambuco  Northeast  -8.142666 -34.906906     80.0
...              ...         ...        ...        ...        ...      ...
12828          house   São Paulo  Southeast -23.587495 -46.559401    250.0
12829      apartment   São Paulo  Southeast -23.522029 -46.189290     55.0
12830      apartment   São Paulo  Southeast -23.526443 -46.529182     57.0
12831          house   Tocantins      North  -8.848399 -48.511164      NaN
12832      apartment   Tocantins      North -10.249091 -48.324286     70.0

        price_brl
0       414222.98
1       848408.53
2       299438.28
3       848408.53
4       464129.36
...           ...
12828  429194.89
12829  252398.80
12830  319400.84
12831  529007.65
12832  289457.01

[12833 rows x 7 columns]>
```


[18]:

	property_type	state	region	lat	lon	area_m2	price_brl
0	apartment	Pernambuco	Northeast	-8.134204	-34.906326	72.0	414222.98
1	apartment	Pernambuco	Northeast	-8.126664	-34.903924	136.0	848408.53
2	apartment	Pernambuco	Northeast	-8.125550	-34.907601	75.0	299438.28
3	apartment	Pernambuco	Northeast	-8.120249	-34.895920	187.0	848408.53
4	apartment	Pernambuco	Northeast	-8.142666	-34.906906	80.0	464129.36

Task 1.5.8: Use the `"price_brl"` column to create a new column named `"price_usd"`. (Keep in mind that, when this data was collected in 2015 and 2016, a US dollar cost 3.19 Brazilian reals.)

[19]:
```
df2['price_usd']=df2['price_brl'] / 3.19
df2.head()
```

[19]:

	property_type	state	region	lat	lon	area_m2	price_brl	price_usd
0	apartment	Pernambuco	Northeast	-8.134204	-34.906326	72.0	414222.98	129850.463950
1	apartment	Pernambuco	Northeast	-8.126664	-34.903924	136.0	848408.53	265958.786834
2	apartment	Pernambuco	Northeast	-8.125550	-34.907601	75.0	299438.28	93867.799373
3	apartment	Pernambuco	Northeast	-8.120249	-34.895920	187.0	848408.53	265958.786834
4	apartment	Pernambuco	Northeast	-8.142666	-34.906906	80.0	464129.36	145495.097179

Task 1.5.9: Drop the `"price_brl"` column from `df2`, as well as any rows that have `NaN` values.

[21]:
```
df2.drop(columns=['price_brl'],inplace=True)
df2=df2.dropna(axis=0)
df2.head()
```

[21]:

	property_type	state	region	lat	lon	area_m2	price_usd
0	apartment	Pernambuco	Northeast	-8.134204	-34.906326	72.0	129850.463950
1	apartment	Pernambuco	Northeast	-8.126664	-34.903924	136.0	265958.786834
2	apartment	Pernambuco	Northeast	-8.125550	-34.907601	75.0	93867.799373
3	apartment	Pernambuco	Northeast	-8.120249	-34.895920	187.0	265958.786834
4	apartment	Pernambuco	Northeast	-8.142666	-34.906906	80.0	145495.097179

[22]:
```
wqet_grader.grade("Project 1 Assessment", "Task 1.5.9", df2)
```

Task 1.5.10: Concatenate `df1` and `df2` to create a new DataFrame named `df`.

```
[23]: df = pd.concat([df1,df2])
      print("df shape:", df.shape)
```

```
df shape: (22844, 7)
```

```
[24]:
      wqet_grader.grade("Project 1 Assessment", "Task 1.5.10", df)
```

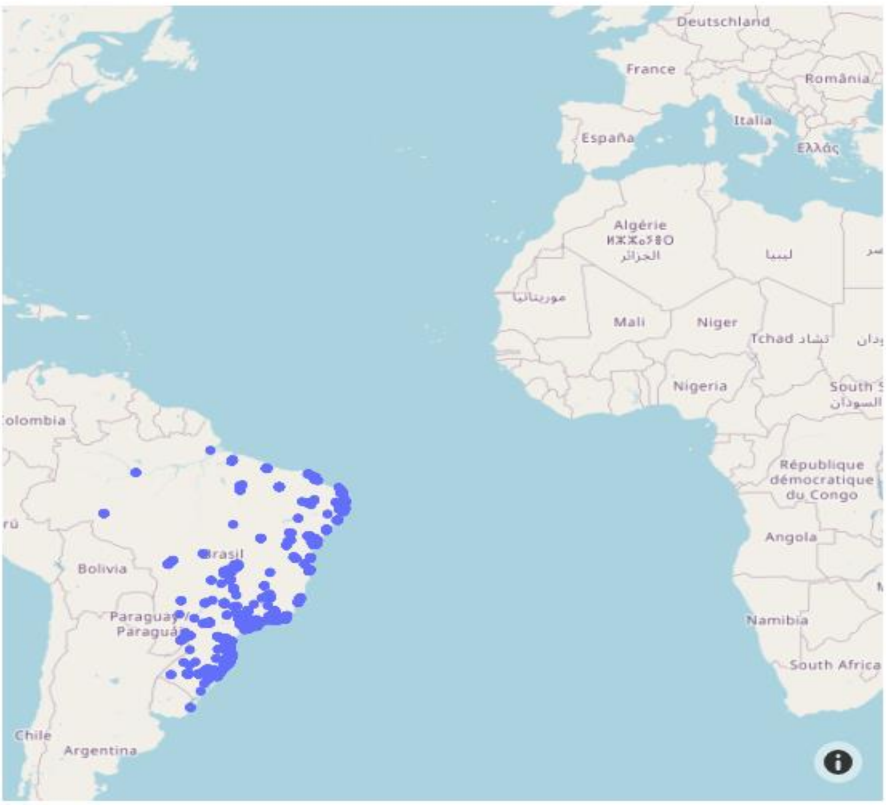
Yes! Your hard work is paying off.
Score: 1

Explore

It's time to start exploring your data. In this section, you'll use your new data visualization skills to learn more about the regional differences in the Brazilian real estate market.

Complete the code below to create a `scatter_mapbox` showing the location of the properties in `df`.

```
[25]: fig = px.scatter_mapbox(
          df,
          lat='lat',
          lon='lon',
          center={"lat": -14.2, "lon": -51.9},  # Map will be centered on Brazil
          width=600,
          height=600,
          hover_data=["price_usd"],  # Display price when hovering mouse over house
      )

      fig.update_layout(mapbox_style="open-street-map")

      fig.show()
```



Task 1.5.11: Use the describe method to create a DataFrame summary_stats with the summary statistics for the "area_m2" and "price_usd" columns.

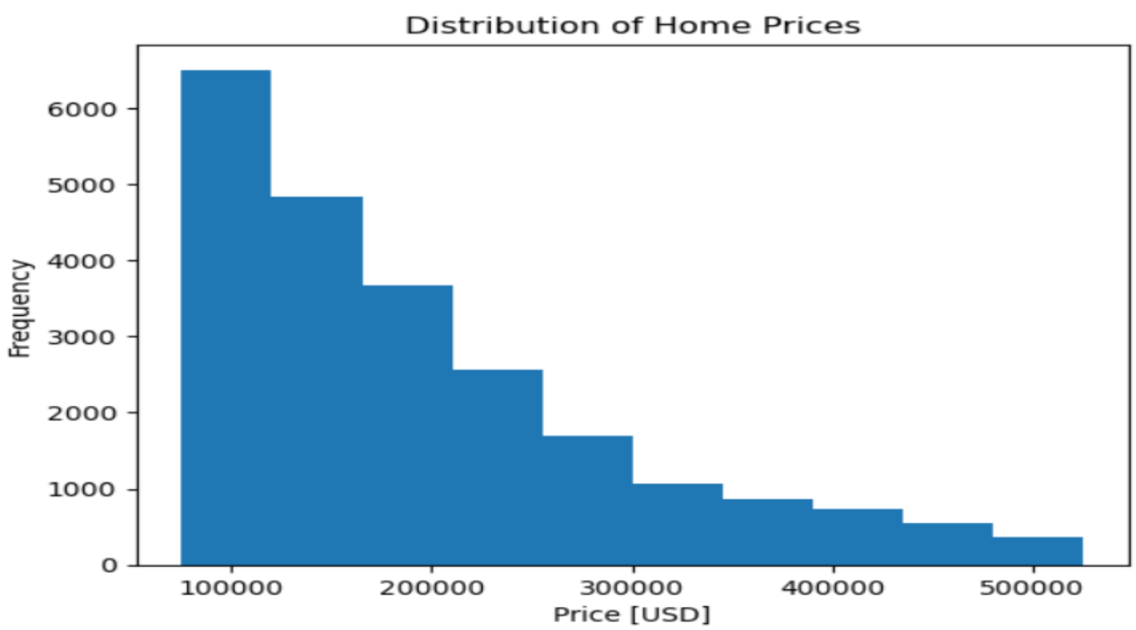
```
[26]: summary_stats = df[['area_m2','price_usd']].describe()
      summary_stats
```

[26]:

	area_m2	price_usd
count	22844.000000	22844.000000
mean	115.020224	194987.315480
std	47.742932	103617.682978
min	53.000000	74892.340000
25%	76.000000	113898.770000
50%	103.000000	165697.555000
75%	142.000000	246900.880878
max	252.000000	525659.717868

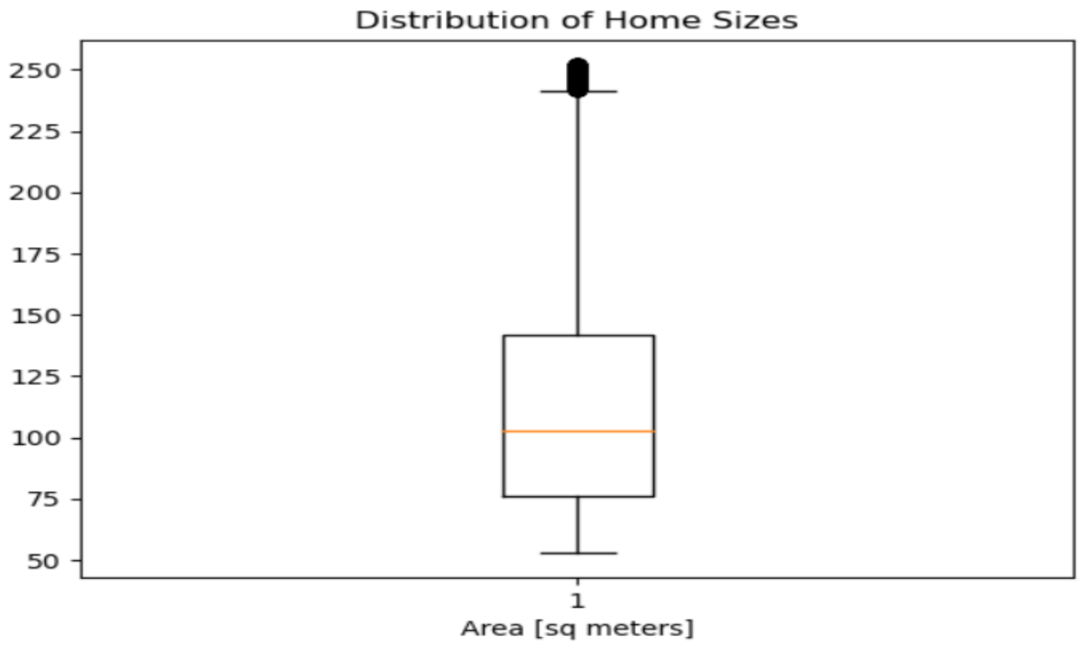
Task 1.5.12: Create a histogram of "price_usd". Make sure that the x-axis has the label "Price [USD]", the y-axis has the label "Frequency", and the plot has the title "Distribution of Home Prices". Use Matplotlib (plt).

```
[28]: # Build histogram
      plt.hist(df['price_usd'], bins=10, rwidth=1)


      # Label axes
      plt.xlabel('Price [USD]')

      plt.ylabel('Frequency')
      # Add title
      plt.title('Distribution of Home Prices')


      # Don't change the code below 👇
      plt.savefig("images/1-5-12.png", dpi=150)
```



Task 1.5.13: Create a horizontal boxplot of "area_m2". Make sure that the x-axis has the label "Area [sq meters]" and the plot has the title "Distribution of Home Sizes". Use Matplotlib (plt).

```
[30]: # Build box plot
plt.boxplot(df['area_m2'])


# Label x-axis
plt.xlabel('Area [sq meters]')

# Add title
plt.title('Distribution of Home Sizes')
# Don't change the code below 👇
plt.savefig("images/1-5-13.png", dpi=150)
```



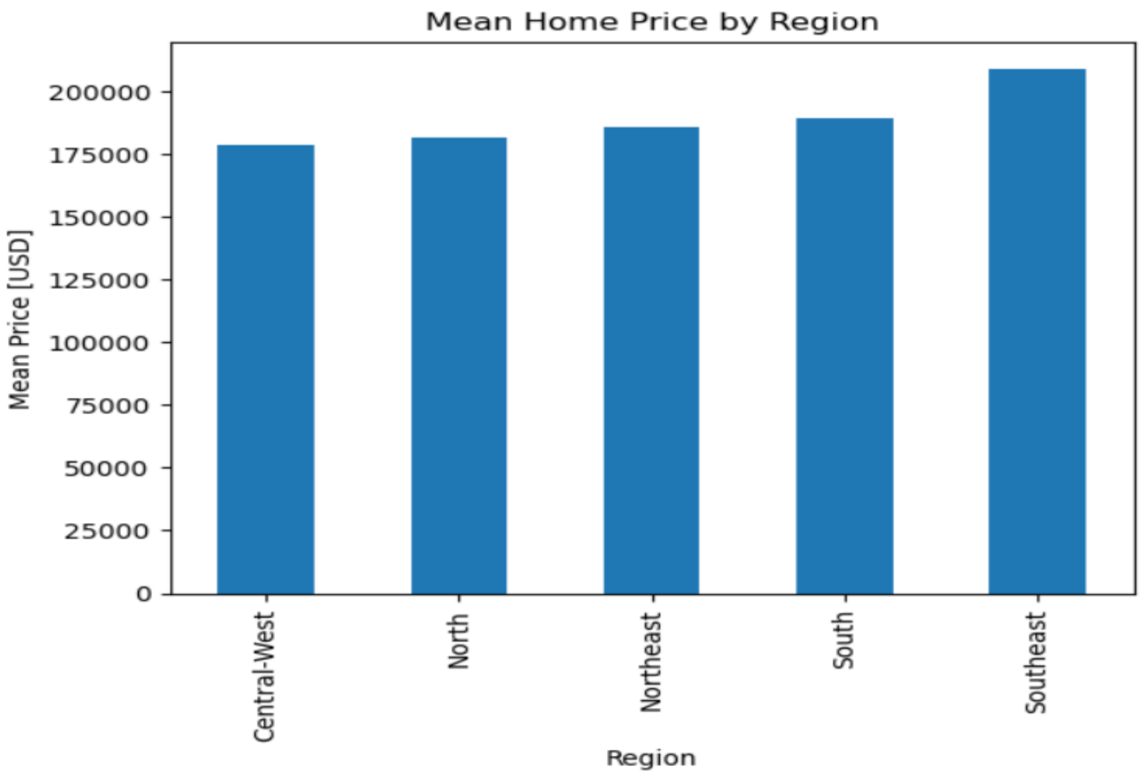
Task 1.5.14: Use the groupby method to create a Series named mean_price_by_region that shows the mean home price in each region in Brazil, sorted from smallest to largest.

```
[32]: mean_price_by_region = df.groupby('region')['price_usd'].mean().sort_values(ascending=True)
mean_price_by_region
```

```
[32]: region
Central-West     178596.283663
North            181308.958207
Northeast        185422.985441
South            189012.345265
Southeast        208996.762778
Name: price_usd, dtype: float64
```

Task 1.5.15: Use mean_price_by_region to create a bar chart. Make sure you label the x-axis as "Region" and the y-axis as "Mean Price [USD]", and give the chart the title "Mean Home Price by Region". Use pandas.

```
[34]: # Build bar chart, label axes, add title
mean_price_by_region.plot(
    kind = 'bar',
    title= 'Mean Home Price by Region',
    xlabel='Region',
    ylabel='Mean Price [USD]'

);

# Don't change the code below 👇
plt.savefig("images/1-5-15.png", dpi=150)
```



You're now going to shift your focus to the southern region of Brazil, and look at the relationship between home size and price.

Task 1.5.16: Create a DataFrame df_south that contains all the homes from df that are in the "South" region.

[36]:
```
df_south = df[df['region']=='South']
df_south.head()
```

[36]:

	property_type	region	area_m2	price_usd	lat	lon	state
9304	apartment	South	127.0	296448.85	-25.455704	-49.292918	Paraná
9305	apartment	South	104.0	219996.25	-25.455704	-49.292918	Paraná
9306	apartment	South	100.0	194210.50	-25.460236	-49.293812	Paraná
9307	apartment	South	77.0	149252.94	-25.460236	-49.293812	Paraná
9308	apartment	South	73.0	144167.75	-25.460236	-49.293812	Paraná

Task 1.5.17: Use the value_counts method to create a Series homes_by_state that contains the number of properties in each state in df_south.

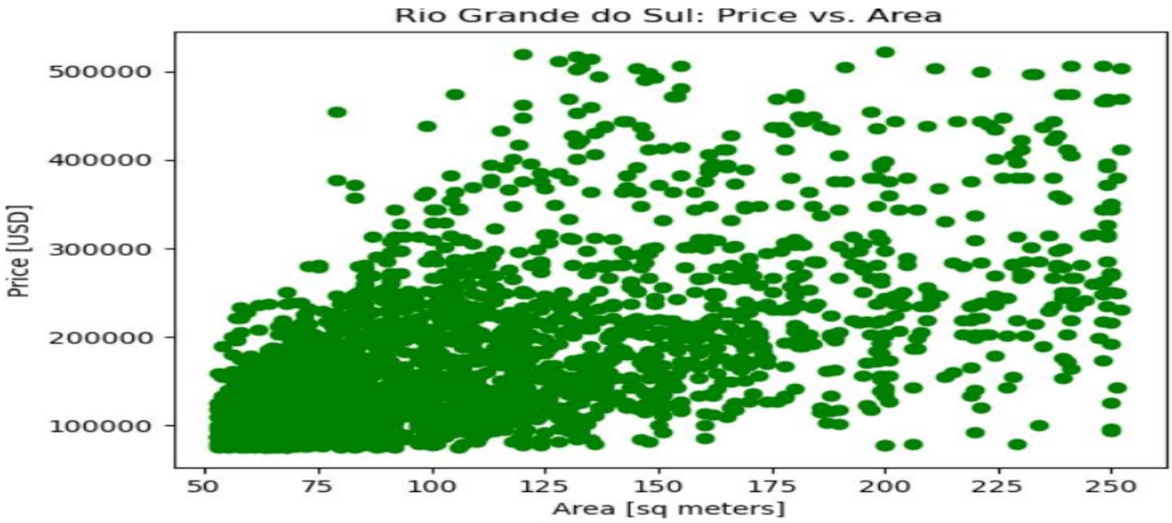
[39]:
```
homes_by_state = df_south['state'].value_counts()
homes_by_state
```

[39]:
```
Rio Grande do Sul     2643
Santa Catarina        2634
Paraná                2544
Name: state, dtype: int64
```

[40]:
```

wqet_grader.grade("Project 1 Assessment", "Task 1.5.17", homes_by_state)
```

Task 1.5.18: Create a scatter plot showing price vs. area for the state in df_south that has the largest number of properties. Be sure to label the x-axis "Area [sq meters]" and the y-axis "Price [USD]"; and use the title "<name of state>: Price vs. Area". Use Matplotlib (plt).

```
[51]: # Subset data
df_south_rgs =df[df['state']=='Rio Grande do Sul']
# Build scatter plot
plt.scatter(df_south_rgs['area_m2'],df_south_rgs['price_usd'], color='g')


# Label axes
plt.xlabel('Area [sq meters]')
plt.ylabel('Price [USD]')


# Add title
plt.title("Rio Grande do Sul: Price vs. Area")

# Don't change the code below 👇
plt.savefig("images/1-5-18.png", dpi=150)
```



Task 1.5.19: Create a dictionary south_states_corr, where the keys are the names of the three states in the "South" region of Brazil, and their associated values are the correlation coefficient between "area_m2" and "price_usd" in that state.

As an example, here's a dictionary with the states and correlation coefficients for the Southeast region. Since you're looking at a different region, the states and coefficients will be different, but the structure of the dictionary will be the same.

{'Espírito Santo': 0.6311332554173303,

'Minas Gerais': 0.5830029036378931,

'Rio de Janeiro': 0.4554077103515366,

'São Paulo': 0.45882050624839366}

```
[64]: south_states_corr = {}

south_states = ['Rio Grande do Sul','Santa Catarina','Paraná']

for state in south_states:
    state_df = df_south[df_south['state']==state]

    correlation = state_df['area_m2'].corr(state_df['price_usd'])

    south_states_corr[state] = correlation

print(south_states_corr)

{'Rio Grande do Sul': 0.5773267433717683, 'Santa Catarina': 0.5068121776366781, 'Paraná': 0.5436659935502659}
```