023-price-and-neighborhood

April 13, 2022

Predicting Price with Neighborhood

[1]:
```
import warnings
from glob import glob

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import wqet_grader
from category_encoders import OneHotEncoder
from IPython.display import VimeoVideo
from sklearn.linear_model import LinearRegression, Ridge  # noqa F401
from sklearn.metrics import mean_absolute_error
from sklearn.pipeline import make_pipeline
from sklearn.utils.validation import check_is_fitted

warnings.simplefilter(action="ignore", category=FutureWarning)
wqet_grader.init("Project 2 Assessment")
```

```
<IPython.core.display.HTML object>
```

In the last lesson, we created a model that used location — represented by latitude and longitude — to predict price. In this lesson, we're going to use a different representation for location: neighborhood.

[2]:
```
VimeoVideo("656790491", h="6325554e55", width=600)
```

[2]:
```
<IPython.lib.display.VimeoVideo at 0x7fd3ec0d9a30>
```

1 Prepare Data

1.1 Import

[3]:
```
def wrangle(filepath):
    # Read CSV file
    df = pd.read_csv(filepath)

    # Subset data: Apartments in "Capital Federal", less than 400,000
```

```
    mask_ba = df["place_with_parent_names"].str.contains("Capital Federal")
    mask_apt = df["property_type"] == "apartment"
    mask_price = df["price_aprox_usd"] < 400_000
    df = df[mask_ba & mask_apt & mask_price]

    # Subset data: Remove outliers for "surface_covered_in_m2"
    low, high = df["surface_covered_in_m2"].quantile([0.1, 0.9])
    mask_area = df["surface_covered_in_m2"].between(low, high)
    df = df[mask_area]

    # Split "lat-lon" column
    df[["lat", "lon"]] = df["lat-lon"].str.split(",", expand=True).astype(float)
    df.drop(columns="lat-lon", inplace=True)

    #Extract Neighbourhood
    df["neighborhood"] = df["place_with_parent_names"].str.split("|",␣
↪expand=True)[3]
    df.drop(columns="place_with_parent_names", inplace=True)


    return df
```

In the last lesson, we used our `wrangle` function to import two CSV files as DataFrames. But what if we had hundreds of CSV files to import? Wrangling them one-by-one wouldn't be an option. So let's start with a technique for reading several CSV files into a single DataFrame.

The first step is to gather the names of all the files we want to import. We can do this using pattern matching.

[4]:
```
VimeoVideo("656790237", h="1502e3765a", width=600)
```

[4]:
```
<IPython.lib.display.VimeoVideo at 0x7fd397008040>
```

Task 2.3.1: Use `glob` to create a list that contains the filenames for all the Buenos Aires real estate CSV files in the `data` directory. Assign this list to the variable name `files`.

- Assemble a list of path names that match a pattern in glob.

[5]:
```
files = glob("data/buenos-aires-real-estate-*.csv")
files
```

[5]:
```
['data/buenos-aires-real-estate-2.csv',
 'data/buenos-aires-real-estate-4.csv',
 'data/buenos-aires-real-estate-3.csv',
 'data/buenos-aires-real-estate-1.csv',
 'data/buenos-aires-real-estate-5.csv']
```

[6]:
```
# Check your work
assert len(files) == 5, f"`files` should contain 5 items, not {len(files)}"
```

The next step is to read each of the CSVs in `files` into a DataFrame, and put all of those DataFrames into a list. What's a good way to iterate through `files` so we can do this? A `for` loop!

[7]:
```
VimeoVideo("656789768", h="3b8f3bca0b", width=600)
```

[7]:
```
<IPython.lib.display.VimeoVideo at 0x7fd3ec0d97c0>
```

Task 2.3.2: Use your `wrangle` function in a `for` loop to create a list named `frames`. The list should the cleaned DataFrames created from the CSV filenames your collected in `files`.

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

[8]:
```
frames = []
for file in files:
    df = wrangle(file)
    #print(df.shape)
    frames.append(df)
```

[9]:
```
#len(frames)
#type(frames[0])
frames[0].head()
```

[9]:
```
   operation property_type      price currency  price_aprox_local_currency  \
2       sell     apartment   215000.0      USD                  3259916.00
9       sell     apartment   341550.0      USD                  5178717.72
12      sell     apartment  1386000.0      ARS                  1382153.13
13      sell     apartment   105000.0      USD                  1592052.00
17      sell     apartment    89681.0      USD                  1359779.19

    price_aprox_usd  surface_total_in_m2  surface_covered_in_m2  \
2         215000.00                 40.0                   35.0
9         341550.00                  NaN                   90.0
12         91156.62                 39.0                   33.0
13        105000.00                  NaN                   33.0
17         89681.00                 46.0                   39.0

    price_usd_per_m2  price_per_m2  floor  rooms  expenses  \
2        5375.000000   6142.857143    NaN    1.0    3500.0
9                NaN   3795.000000    8.0    2.0       NaN
12       2337.349231  42000.000000    NaN    NaN       NaN
13               NaN   3181.818182    1.0    1.0       NaN
17       1949.586957   2299.512821    NaN    1.0    1500.0

                                         properati_url        lat        lon  \
2   http://recoleta.properati.com.ar/12j4v_venta_d… -34.588993 -58.400133
```

```
9   http://recoleta.properati.com.ar/100t0_venta_d… -34.588044 -58.398066
12  http://monserrat.properati.com.ar/t05l_venta_d… -34.623320 -58.397461
13  http://belgrano.properati.com.ar/zsd5_venta_de… -34.553897 -58.451939
17  http://villa-del-parque.properati.com.ar/12q2f… -34.628813 -58.472230

        neighborhood
2           Recoleta
9           Recoleta
12         Monserrat
13          Belgrano
17  Villa del Parque
```

[10]:
```
# Check your work
assert len(frames) == 5, f"`frames` should contain 5 items, not {len(frames)}"
assert all(
    [isinstance(frame, pd.DataFrame) for frame in frames]
), "The items in `frames` should all be DataFrames."
```

The final step is to use pandas to combine all the DataFrames in `frames`.

[11]:
```
VimeoVideo("656789700", h="57adef4afe", width=600)
```

[11]: `<IPython.lib.display.VimeoVideo at 0x7fd3ec0f05e0>`

Task 2.3.3: Use `pd.concat` to concatenate the items in `frames` into a single DataFrame `df`. Make sure you set the `ignore_index` argument to `True`.

- Concatenate two or more DataFrames using pandas.

[12]:
```
df = pd.concat(frames, ignore_index=True)
#df.head()
df.shape
```

[12]: `(6582, 17)`

[13]:
```
# Check your work
assert len(df) == 6582, f"`df` is the wrong size: {len(df)}."
```

Excellent work! You can now clean and combine as many CSV files as your computer can handle. You're well on your way to working with big data.

1.2 Explore

Looking through the output from the `df.head()` call above, there's a little bit more cleaning we need to do before we can work with the neighborhood information in this dataset. The good news is that, because we're using a `wrangle` function, we only need to change the function to re-clean all of our CSV files. This is why functions are so useful.

[14]:
```
VimeoVideo("656791659", h="581201dc92", width=600)
```

[14]: <IPython.lib.display.VimeoVideo at 0x7fd3c3d16070>

[15]:
```
df.head()
```

[15]:
```
  operation property_type      price currency  price_aprox_local_currency  \
0      sell     apartment   215000.0      USD                  3259916.00
1      sell     apartment   341550.0      USD                  5178717.72
2      sell     apartment  1386000.0      ARS                  1382153.13
3      sell     apartment   105000.0      USD                  1592052.00
4      sell     apartment    89681.0      USD                  1359779.19

   price_aprox_usd  surface_total_in_m2  surface_covered_in_m2  \
0        215000.00                 40.0                   35.0
1        341550.00                  NaN                   90.0
2         91156.62                 39.0                   33.0
3        105000.00                  NaN                   33.0
4         89681.00                 46.0                   39.0

   price_usd_per_m2  price_per_m2  floor  rooms  expenses  \
0       5375.000000   6142.857143    NaN    1.0    3500.0
1               NaN   3795.000000    8.0    2.0       NaN
2       2337.349231  42000.000000    NaN    NaN       NaN
3               NaN   3181.818182    1.0    1.0       NaN
4       1949.586957   2299.512821    NaN    1.0    1500.0

                                       properati_url        lat        lon  \
0  http://recoleta.properati.com.ar/12j4v_venta_d… -34.588993 -58.400133
1  http://recoleta.properati.com.ar/100t0_venta_d… -34.588044 -58.398066
2  http://monserrat.properati.com.ar/t05l_venta_d… -34.623320 -58.397461
3  http://belgrano.properati.com.ar/zsd5_venta_de… -34.553897 -58.451939
4  http://villa-del-parque.properati.com.ar/12q2f… -34.628813 -58.472230

      neighborhood
0         Recoleta
1         Recoleta
2        Monserrat
3         Belgrano
4  Villa del Parque
```

Task 2.3.4: Modify your `wrangle` function to create a new feature `"neighborhood"`. You can find the neighborhood for each property in the `"place_with_parent_names"` column. For example, a property with the place name `"|Argentina|Capital Federal|Palermo|"` is located in the neighborhood is `"Palermo"`. Also, your function should drop the `"place_with_parent_names"` column.

Be sure to rerun all the cells above before you continue.

- Split the strings in one column to create another using pandas.

[16]:
```
# Check your work
assert df.shape == (6582, 17), f"`df` is the wrong size: {df.shape}."
assert (
    "place_with_parent_names" not in df
), 'Remember to remove the `"place_with_parent_names"` column.'
```

1.3 Split

At this point, you should feel more comfortable with the splitting data, so we're going to condense the whole process down to one task.

[17]:
```
VimeoVideo("656791577", h="0ceb5341f8", width=600)
```

[17]: <IPython.lib.display.VimeoVideo at 0x7fd3c3d16970>

Task 2.3.5: Create your feature matrix `X_train` and target vector `y_train`. `X_train` should contain one feature: `"neighborhood"`. Your target is `"price_aprox_usd"`.

- What's a feature matrix?
- What's a target vector?
- Subset a DataFrame by selecting one or more columns in pandas.
- Select a Series from a DataFrame in pandas.

[18]:
```
target = "price_aprox_usd"
features = ["neighborhood"]
y_train = df[target]
X_train = df[features]
```

[19]:
```
# Check your work
assert X_train.shape == (6582, 1), f"`X_train` is the wrong size: {X_train.
↪shape}."
assert y_train.shape == (6582,), f"`y_train` is the wrong size: {y_train.shape}.
↪"
```

2 Build Model

2.1 Baseline

Let's also condense the code we use to establish our baseline.

[20]:
```
VimeoVideo("656791443", h="120a740cc3", width=600)
```

[20]: <IPython.lib.display.VimeoVideo at 0x7fd3c3abd610>

Task 2.3.6: Calculate the baseline mean absolute error for your model.

- What's a performance metric?
- What's mean absolute error?
- Calculate summary statistics for a DataFrame or Series in pandas.

- Calculate the mean absolute error for a list of predictions in scikit-learn.

[21]:
```
y_mean = y_train.mean()
y_pred_baseline = [y_mean] * len(y_train)
print("Mean apt price:", y_mean)

print("Baseline MAE:", mean_absolute_error(y_train, y_pred_baseline))
```

```
Mean apt price: 132383.83701458527
Baseline MAE: 44860.10834274133
```

The mean apartment price and baseline MAE should be similar but not identical to last lesson. The numbers will change since we're working with more data.

2.2 Iterate

If you try to fit a `LinearRegression` predictor to your training data at this point, you'll get an error that looks like this:

`ValueError: could not convert string to float`

What does this mean? When you fit a linear regression model, you're asking scikit-learn to perform a mathematical operation. The problem is that our training set contains neighborhood information in non-numerical form. In order to create our model we need to **encode** that information so that it's represented numerically. The good news is that there are lots of transformers that can do this. Here, we'll use the one from the Category Encoders library, called a `OneHotEncoder`.

Before we build include this transformer in our pipeline, let's explore how it works.

[22]:
```
VimeoVideo("656792790", h="4097efb40d", width=600)
```

[22]: `<IPython.lib.display.VimeoVideo at 0x7fd3c3abd7f0>`

Task 2.3.7: First, instantiate a `OneHotEncoder` named `ohe`. Make sure to set the `use_cat_names` argument to `True`. Next, fit your transformer to the feature matrix `X_train`. Finally, use your encoder to transform the feature matrix `X_train`, and assign the transformed data to the variable `XT_train`.

- What's one-hot encoding?
- Instantiate a transformer in scikit-learn.
- Fit a transformer to training data in scikit-learn.
- Transform data using a transformer in scikit-learn.

[24]:
```
#Instantiate
ohe = OneHotEncoder(use_cat_names=True)
#Fit
ohe.fit(X_train)
#Transform
XT_train = ohe.transform(X_train)
print(XT_train.shape)
XT_train.head()
```

```
(6582, 57)
```

[24]:
```
   neighborhood_Recoleta  neighborhood_Monserrat  neighborhood_Belgrano  \
0                      1                       0                      0
1                      1                       0                      0
2                      0                       1                      0
3                      0                       0                      1
4                      0                       0                      0

   neighborhood_Villa del Parque  neighborhood_Villa Pueyrredón  \
0                              0                              0
1                              0                              0
2                              0                              0
3                              0                              0
4                              1                              0

   neighborhood_Almagro  neighborhood_Palermo  neighborhood_  \
0                     0                     0              0
1                     0                     0              0
2                     0                     0              0
3                     0                     0              0
4                     0                     0              0

   neighborhood_Tribunales  neighborhood_Balvanera  …  \
0                        0                       0  …
1                        0                       0  …
2                        0                       0  …
3                        0                       0  …
4                        0                       0  …

   neighborhood_Velez Sarsfield  neighborhood_Monte Castro  \
0                             0                          0
1                             0                          0
2                             0                          0
3                             0                          0
4                             0                          0

   neighborhood_Las Cañitas  neighborhood_Constitución  \
0                         0                          0
1                         0                          0
2                         0                          0
3                         0                          0
4                         0                          0

   neighborhood_Parque Avellaneda  neighborhood_Villa Soldati  \
0                               0                           0
1                               0                           0
```

```
2                                0                           0
3                                0                           0
4                                0                           0

   neighborhood_Villa Real  neighborhood_Versalles  neighborhood_Pompeya  \
0                        0                       0                     0
1                        0                       0                     0
2                        0                       0                     0
3                        0                       0                     0
4                        0                       0                     0

   neighborhood_Catalinas
0                       0
1                       0
2                       0
3                       0
4                       0

[5 rows x 57 columns]
```

[25]:
```
# Check your work
assert XT_train.shape == (6582, 57), f"`XT_train` is the wrong shape: {XT_train.
↪shape}"
```

Now that we have an idea for how the `OneHotEncoder` works, let's bring it into our pipeline.

[26]:
```
VimeoVideo("656792622", h="0b9d189e8f", width=600)
```

[26]:
```
<IPython.lib.display.VimeoVideo at 0x7fd3c3aa3be0>
```

Task 2.3.8: Create a pipeline named `model` that contains a `OneHotEncoder` transformer and a `LinearRegression` predictor. Then fit your model to the training data.

- What's a pipeline?
- Create a pipeline in scikit-learn.

[27]:
```
model = make_pipeline(
    OneHotEncoder(use_cat_names=True),
    LinearRegression()
)
model.fit(X_train, y_train)
```

[27]:
```
Pipeline(steps=[('onehotencoder',
                 OneHotEncoder(cols=['neighborhood'], use_cat_names=True)),
                ('linearregression', LinearRegression())])
```

[28]:
```
# Check your work
check_is_fitted(model[-1])
```

Wow, you just built a model with two transformers and a predictor! When you started this course, did you think you'd be able to do something like that?

2.3 Evaluate

Regardless of how you build your model, the evaluation step stays the same. Let's see how our model performs with the training set.

[29]:
```
VimeoVideo("656792525", h="09edc1c3d6", width=600)
```

[29]:
```
<IPython.lib.display.VimeoVideo at 0x7fd3c3aa3c10>
```

Task 2.3.9: First, create a list of predictions for the observations in your feature matrix `X_train`. Name this list `y_pred_training`. Then calculate the training mean absolute error for your predictions in `y_pred_training` as compared to the true targets in `y_train`.

- Generate predictions using a trained model in scikit-learn.
- Calculate the mean absolute error for a list of predictions in scikit-learn.

[32]:
```
y_pred_training = model.predict(X_train)
mae_training = mean_absolute_error(y_train, y_pred_training)
print("Training MAE:", round(mae_training, 2))
```

```
Training MAE: 39330.58
```

Now let's check our test performance.

Task 2.3.10: Run the code below to import your test data `buenos-aires-test-features.csv` into a DataFrame and generate a Series of predictions using your model. Then run the following cell to submit your predictions to the grader.

- What's generalizability?
- Generate predictions using a trained model in scikit-learn.
- Calculate the mean absolute error for a list of predictions in scikit-learn.

[30]:
```
X_test = pd.read_csv("data/buenos-aires-test-features.csv")[features]
y_pred_test = pd.Series(model.predict(X_test))
y_pred_test.head()
```

[30]:
```
0    249440.0
1    161440.0
2     97952.0
3    110688.0
4    126880.0
dtype: float64
```

[31]:
```
wqet_grader.grade("Project 2 Assessment", "Task 2.3.10", y_pred_test)
```

```
<IPython.core.display.HTML object>
```

3 Communicate Results

If we write out the equation for our model, it'll be too big to fit on the screen. That's because, when we used the `OneHotEncoder` to encode the neighborhood data, we created a much wider DataFrame, and each column/feature has it's own coefficient in our model's equation.

This is important to keep in mind for two reasons. First, it means that this is a **high-dimensional** model. Instead of a 2D or 3D plot, we'd need a 58-dimensional plot to represent it, which is impossible! Second, it means that we'll need to extract and represent the information for our equation a little differently than before. Let's start by getting our intercept and coefficient.

[32]:
```
VimeoVideo("656793909", h="fca67856b4", width=600)
```

[32]:
```
<IPython.lib.display.VimeoVideo at 0x7fd312092d00>
```

Task 2.3.11: Extract the intercept and coefficients for your model.

- What's an intercept in a linear model?
- What's a coefficient in a linear model?
- Access an object in a pipeline in scikit-learn.

[35]:
```
intercept = model.named_steps["linearregression"].intercept_
coefficients = model.named_steps["linearregression"].coef_
print("coefficients len:", len(coefficients))
print(coefficients[:5])  # First five coefficients
```

```
coefficients len: 57
[2.33333888e+17 2.33333888e+17 2.33333888e+17 2.33333888e+17
 2.33333888e+17]
```

[36]:
```
# Check your work
assert isinstance(
    intercept, float
), f"`intercept` should be a `float`, not {type(intercept)}."
assert isinstance(
    coefficients, np.ndarray
), f"`coefficients` should be a `float`, not {type(coefficients)}."
assert coefficients.shape == (
    57,
), f"`coefficients` is wrong shape: {coefficients.shape}."
```

We have the values of our coefficients, but how do we know which features they belong to? We'll need to get that information by going into the part of our pipeline that did the encoding.

[37]:
```
VimeoVideo("656793812", h="810161b84e", width=600)
```

[37]:
```
<IPython.lib.display.VimeoVideo at 0x7fd3c2642100>
```

Task 2.3.12: Extract the feature names of your encoded data from the `OneHotEncoder` in your model.

- Access an object in a pipeline in scikit-learn.

[40]:
```
feature_names = model.named_steps["onehotencoder"].get_feature_names()
print("features len:", len(feature_names))
print(feature_names[:5])  # First five feature names
```

```
features len: 57
['neighborhood_Recoleta', 'neighborhood_Monserrat', 'neighborhood_Belgrano',
'neighborhood_Villa del Parque', 'neighborhood_Villa Pueyrredón']
```

[41]:
```
# Check your work
assert isinstance(
    feature_names, list
), f"`features` should be a `list`, not {type(features)}."
assert len(feature_names) == len(
    coefficients
), "You should have the same number of features and coefficients."
```

We have coefficients and feature names, and now we need to put them together. For that, we'll use a Series.

[42]:
```
VimeoVideo("656793718", h="1e2a1e1de8", width=600)
```

[42]: `<IPython.lib.display.VimeoVideo at 0x7fd30682b6a0>`

Task 2.3.13: Create a pandas Series named `feat_imp` where the index is your `features` and the values are your `coefficients`.

- Create a Series in pandas.

[49]:
```
feat_imp = pd.Series(coefficients, index=feature_names)
feat_imp.head()
```

[49]:
```
neighborhood_Recoleta            2.333339e+17
neighborhood_Monserrat           2.333339e+17
neighborhood_Belgrano            2.333339e+17
neighborhood_Villa del Parque    2.333339e+17
neighborhood_Villa Pueyrredón    2.333339e+17
dtype: float64
```

[44]:
```
# Check your work
assert isinstance(
    feat_imp, pd.Series
), f"`feat_imp` should be a `float`, not {type(feat_imp)}."
assert feat_imp.shape == (57,), f"`feat_imp` is wrong shape: {feat_imp.shape}."
assert all(
    a == b for a, b in zip(sorted(feature_names), sorted(feat_imp.index))
), "The index of `feat_imp` should be identical to `features`."
```

To be clear, it's definitely not a good idea to show this long equation to an audience, but let's print it out just to check our work. Since there are so many terms to print, we'll use a `for` loop.

[46]:
```
VimeoVideo("656797021", h="dc90e6dac3", width=600)
```

[46]:
```
<IPython.lib.display.VimeoVideo at 0x7fd3c26edd90>
```

Task 2.3.14: Run the cell below to print the equation that your model has determined for predicting apartment price based on longitude and latitude.

- What's an f-string?

[51]:
```
print(f"price = {intercept.round(2)}")
for f, c in feat_imp.items():
    print(f"+ ({round(c, 2)} * {f})")
```

```
price = -2.3333388787992288e+17
+ (2.333338878801141e+17 * neighborhood_Recoleta)
+ (2.3333388788002086e+17 * neighborhood_Monserrat)
+ (2.3333388788008883e+17 * neighborhood_Belgrano)
+ (2.3333388788002867e+17 * neighborhood_Villa del Parque)
+ (2.3333388788003366e+17 * neighborhood_Villa Pueyrredón)
+ (2.3333388788004448e+17 * neighborhood_Almagro)
+ (2.333338878800881e+17 * neighborhood_Palermo)
+ (2.3333388788002182e+17 * neighborhood_)
+ (2.3333388788003293e+17 * neighborhood_Tribunales)
+ (2.333338878800298e+17 * neighborhood_Balvanera)
+ (2.333338878800971e+17 * neighborhood_Barrio Norte)
+ (2.3333388788003795e+17 * neighborhood_Once)
+ (2.33333887880047e+17 * neighborhood_San Telmo)
+ (2.3333388787999098e+17 * neighborhood_Villa Lugano)
+ (2.3333388788005357e+17 * neighborhood_Coghlan)
+ (2.3333388788003638e+17 * neighborhood_Barracas)
+ (2.3333388788005392e+17 * neighborhood_Villa Urquiza)
+ (2.3333388788004582e+17 * neighborhood_Abasto)
+ (2.3333388788004774e+17 * neighborhood_Villa Crespo)
+ (2.3333388788002093e+17 * neighborhood_Villa Santa Rita)
+ (2.3333388788007994e+17 * neighborhood_Colegiales)
+ (2.3333388788003418e+17 * neighborhood_Paternal)
+ (2.333338878800498e+17 * neighborhood_Caballito)
+ (2.333338878800336e+17 * neighborhood_Parque Chacabuco)
+ (2.333338878800689e+17 * neighborhood_Retiro)
+ (2.3333388788004538e+17 * neighborhood_Villa Devoto)
+ (2.3333388788004157e+17 * neighborhood_Villa Luro)
+ (2.3333388788003066e+17 * neighborhood_San Nicolás)
+ (2.333338878800561e+17 * neighborhood_Saavedra)
+ (2.3333388788003258e+17 * neighborhood_Flores)
+ (2.3333388788003344e+17 * neighborhood_Centro / Microcentro)
+ (2.3333388788002762e+17 * neighborhood_Liniers)
```

```
+ (2.333338878800307e+17 * neighborhood_San Cristobal)
+ (2.3333388788001248e+17 * neighborhood_Boca)
+ (2.3333388788003334e+17 * neighborhood_Congreso)
+ (2.3333388788003498e+17 * neighborhood_Parque Centenario)
+ (2.3333388788000662e+17 * neighborhood_Parque Chas)
+ (2.3333388788008435e+17 * neighborhood_Nuñez)
+ (2.3333388788002525e+17 * neighborhood_Parque Patricios)
+ (2.3333388788003453e+17 * neighborhood_Boedo)
+ (2.333338878800271e+17 * neighborhood_Floresta)
+ (2.3333388788001987e+17 * neighborhood_Mataderos)
+ (2.3333388788017235e+17 * neighborhood_Puerto Madero)
+ (2.333338878800496e+17 * neighborhood_Villa General Mitre)
+ (2.3333388788004048e+17 * neighborhood_Agronomía)
+ (2.3333388788002938e+17 * neighborhood_Villa Ortuzar)
+ (2.333338878800386e+17 * neighborhood_Chacarita)
+ (2.3333388788000752e+17 * neighborhood_Velez Sarsfield)
+ (2.3333388788003786e+17 * neighborhood_Monte Castro)
+ (2.333338878801161e+17 * neighborhood_Las Cañitas)
+ (2.3333388787999862e+17 * neighborhood_Constitución)
+ (2.3333388788000963e+17 * neighborhood_Parque Avellaneda)
+ (2.3333388787996758e+17 * neighborhood_Villa Soldati)
+ (2.3333388788003168e+17 * neighborhood_Villa Real)
+ (2.333338878800357e+17 * neighborhood_Versalles)
+ (2.3333388787998886e+17 * neighborhood_Pompeya)
+ (2.333338878799976e+17 * neighborhood_Catalinas)
```

Warning: In the first lesson for this project, we said that you shouldn't make any changes to your model after you see your test metrics. That's still true. However, we're breaking that rule here so that we can discuss overfitting. In future lessons, you'll learn how to protect against overfitting without checking your test set.

[52]:
```
VimeoVideo("656799309", h="a7130deb64", width=600)
```

[52]:
```
<IPython.lib.display.VimeoVideo at 0x7fd306f80ca0>
```

Task 2.3.15: Scroll up, change the predictor in your model to `Ridge`, and retrain it. Then evaluate the model's training and test performance. Do you still have an overfitting problem? If not, extract the intercept and coefficients again (you'll need to change your code a little bit) and regenerate the model's equation. Does it look different than before?

- What's overfitting?
- What's regularization?
- What's ridge regression?

[53]:
```
# Check your work
assert isinstance(
    model[-1], Ridge
), "Did you retrain your model using a `Ridge` predictor?"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [53], in <cell line: 2>()
      1 # Check your work
----> 2 assert isinstance(
      3     model[-1], Ridge
      4 ), "Did you retrain your model using a `Ridge` predictor?"

AssertionError: Did you retrain your model using a `Ridge` predictor?
```

We're back on track with our model, so let's create a visualization that will help a non-technical audience understand what the most important features for our model in predicting apartment price.

```
[ ]: VimeoVideo("656798530", h="9a9350eff1", width=600)
```

Task 2.3.16: Create a horizontal bar chart that shows the top 15 coefficients for your model, based on their absolute value.

- What's a bar chart?
- Create a bar chart using pandas.

```
[ ]:
```

Looking at this bar chart, we can see that the poshest neighborhoods in Buenos Aires like Puerto Madero and Recoleta increase the predicted price of an apartment, while more working-class neighborhoods like Villa Soldati and Villa Lugano decrease the predicted price.

Just for fun, check out this song by Kevin Johansen about Puerto Madero.
