022-price-and-location

April 10, 2022

Predicting Price with Location

[1]:
```
import warnings

import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import wqet_grader
from IPython.display import VimeoVideo
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.pipeline import Pipeline, 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 this lesson, we're going to build on the work we did in the previous lesson. We're going to create a more complex `wrangle` function, use it to clean more data, and build a model that considers more features when predicting apartment price.

[2]:
```
VimeoVideo("656752925", h="701f3f4081", width=600)
```

[2]:
```
<IPython.lib.display.VimeoVideo at 0x7f11327880d0>
```

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



    return df
```

[4]:
```
VimeoVideo("656752771", h="3a42896eb6", width=600)
```

[4]:
```
<IPython.lib.display.VimeoVideo at 0x7f106e458880>
```

Task 2.2.1: Use your `wrangle` function to create a DataFrame `frame1` from the CSV file `data/buenos-aires-real-estate-1.csv`.

[5]:
```
frame1 = wrangle("data/buenos-aires-real-estate-1.csv")
print(frame1.info())
frame1.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1343 entries, 4 to 8604
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   operation                   1343 non-null   object
 1   property_type               1343 non-null   object
 2   place_with_parent_names     1343 non-null   object
 3   price                       1343 non-null   float64
 4   currency                    1343 non-null   object
 5   price_aprox_local_currency  1343 non-null   float64
 6   price_aprox_usd             1343 non-null   float64
 7   surface_total_in_m2         965 non-null    float64
 8   surface_covered_in_m2       1343 non-null   float64
 9   price_usd_per_m2            927 non-null    float64
 10  price_per_m2                1343 non-null   float64
 11  floor                       379 non-null    float64
 12  rooms                       1078 non-null   float64
 13  expenses                    349 non-null    object
 14  properati_url               1343 non-null   object
 15  lat                         1300 non-null   float64
 16  lon                         1300 non-null   float64
dtypes: float64(11), object(6)
memory usage: 188.9+ KB
None
```

[5]:
```
   operation property_type                       place_with_parent_names  \
4       sell     apartment     |Argentina|Capital Federal|Chacarita|
9       sell     apartment    |Argentina|Capital Federal|Villa Luro|
29      sell     apartment     |Argentina|Capital Federal|Caballito|
40      sell     apartment  |Argentina|Capital Federal|Constitución|
41      sell     apartment          |Argentina|Capital Federal|Once|

       price currency  price_aprox_local_currency  price_aprox_usd  \
4   129000.0      USD                   1955949.6         129000.0
9    87000.0      USD                   1319128.8          87000.0
29  118000.0      USD                   1789163.2         118000.0
40   57000.0      USD                    864256.8          57000.0
41   90000.0      USD                   1364616.0          90000.0

    surface_total_in_m2  surface_covered_in_m2  price_usd_per_m2  \
4                  76.0                   70.0       1697.368421
9                  48.0                   42.0       1812.500000
29                  NaN                   54.0               NaN
40                 42.0                   42.0       1357.142857
41                 57.0                   50.0       1578.947368

    price_per_m2  floor  rooms expenses  \
4    1842.857143    NaN    NaN      NaN
9    2071.428571    NaN    NaN      NaN
29   2185.185185    NaN    2.0      NaN
40   1357.142857    5.0    2.0      364
41   1800.000000    NaN    3.0      450

                                        properati_url        lat        lon
4   http://chacarita.properati.com.ar/10qlv_venta_… -34.584651 -58.454693
9   http://villa-luro.properati.com.ar/12m82_venta… -34.638979 -58.500115
29  http://caballito.properati.com.ar/11wqh_venta_… -34.615847 -58.459957
40  http://constitucion.properati.com.ar/k2f0_vent… -34.625222 -58.382382
41  http://once.properati.com.ar/suwa_venta_depart… -34.610610 -58.412511
```

For our model, we're going to consider apartment location, specifically, latitude and longitude. Looking at the output from `frame1.info()`, we can see that the location information is in a single column where the data type is `object` (pandas term for `str` in this case). In order to build our model, we need latitude and longitude to each be in their own column where the data type is `float`.

[6]:
```
VimeoVideo("656751955", h="e47002428d", width=600)
```

```
[6]: <IPython.lib.display.VimeoVideo at 0x7f111f91d790>
```

Task 2.2.2: Add to the `wrangle` function below so that, in the DataFrame it returns, the `"lat-lon"` column is replaced by separate `"lat"` and `"lon"` columns. Don't forget to also drop the `"lat-lon"` column. Be sure to rerun all the cells above before you continue.

- What's a function?
- Split the strings in one column to create another using pandas.
- Drop a column from a DataFrame using pandas.

```
[7]: # Check your work
assert (
    frame1.shape[0] == 1343
), f"`frame1` should have 1343 rows, not {frame1.shape[0]}."
assert frame1.shape[1] == 17, f"`frame1` should have 17 columns, not {frame1.
 ↪shape[1]}."
```

Now that our `wrangle` function is working, let's use it to clean more data!

```
[8]: VimeoVideo("656751853", h="da40b0a474", width=600)
```

```
[8]: <IPython.lib.display.VimeoVideo at 0x7f111f931820>
```

Task 2.2.3: Use you revised `wrangle` function create a DataFrames `frame2` from the file `data/buenos-aires-real-estate-2.csv`.

```
[9]: frame2 = wrangle("data/buenos-aires-real-estate-2.csv")
```

```
[10]: # Check your work
assert (
    frame2.shape[0] == 1315
), f"`frame1` should have 1315 rows, not {frame2.shape[0]}."
assert frame2.shape[1] == 17, f"`frame1` should have 17 columns, not {frame2.
 ↪shape[1]}."
```

As you can see, using a function is much quicker than cleaning each file individually like we did in the last project. Let's combine our DataFrames so we can use then to train our model.

```
[11]: VimeoVideo("656751405", h="d1f95ab108", width=600)
```

```
[11]: <IPython.lib.display.VimeoVideo at 0x7f111f938cd0>
```

Task 2.2.4: Use `pd.concat` to concatenate `frame1` and `frame2` into a new DataFrame `df`. Make sure you set the `ignore_index` argument to `True`.

- Concatenate two or more DataFrames using pandas.

```
[12]: df = pd.concat([frame1,frame2], ignore_index=True)
print(df.info())
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2658 entries, 0 to 2657
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   operation                   2658 non-null   object 
 1   property_type               2658 non-null   object 
 2   place_with_parent_names     2658 non-null   object 
 3   price                       2658 non-null   float64
 4   currency                    2658 non-null   object 
 5   price_aprox_local_currency  2658 non-null   float64
 6   price_aprox_usd             2658 non-null   float64
 7   surface_total_in_m2         1898 non-null   float64
 8   surface_covered_in_m2       2658 non-null   float64
 9   price_usd_per_m2            1818 non-null   float64
 10  price_per_m2                2658 non-null   float64
 11  floor                       769 non-null    float64
 12  rooms                       2137 non-null   float64
 13  expenses                    688 non-null    object 
 14  properati_url               2658 non-null   object 
 15  lat                         2561 non-null   float64
 16  lon                         2561 non-null   float64
dtypes: float64(11), object(6)
memory usage: 353.1+ KB
None
```

[12]:
```
  operation property_type                      place_with_parent_names     price  \
0      sell     apartment     |Argentina|Capital Federal|Chacarita|  129000.0   
1      sell     apartment    |Argentina|Capital Federal|Villa Luro|   87000.0   
2      sell     apartment     |Argentina|Capital Federal|Caballito|  118000.0   
3      sell     apartment  |Argentina|Capital Federal|Constitución|   57000.0   
4      sell     apartment          |Argentina|Capital Federal|Once|   90000.0   

  currency  price_aprox_local_currency  price_aprox_usd  surface_total_in_m2  \
0      USD                   1955949.6         129000.0                 76.0   
1      USD                   1319128.8          87000.0                 48.0   
2      USD                   1789163.2         118000.0                  NaN   
3      USD                    864256.8          57000.0                 42.0   
4      USD                   1364616.0          90000.0                 57.0   

   surface_covered_in_m2  price_usd_per_m2  price_per_m2  floor  rooms  \
0                   70.0       1697.368421   1842.857143    NaN    NaN   
1                   42.0       1812.500000   2071.428571    NaN    NaN   
2                   54.0               NaN   2185.185185    NaN    2.0   
3                   42.0       1357.142857   1357.142857    5.0    2.0   
4                   50.0       1578.947368   1800.000000    NaN    3.0   
```

```
  expenses                                    properati_url        lat  \
0      NaN  http://chacarita.properati.com.ar/10qlv_venta_… -34.584651
1      NaN  http://villa-luro.properati.com.ar/12m82_venta… -34.638979
2      NaN  http://caballito.properati.com.ar/11wqh_venta_… -34.615847
3      364  http://constitucion.properati.com.ar/k2f0_vent… -34.625222
4      450  http://once.properati.com.ar/suwa_venta_depart… -34.610610

         lon
0 -58.454693
1 -58.500115
2 -58.459957
3 -58.382382
4 -58.412511
```

[13]:
```
# Check your work
assert df.shape == (2658, 17), f"`df` is the wrong size: {df.shape}"
```

1.2 Explore

In the last lesson, we built a simple linear model that predicted apartment price based on one feature, `"surface_covered_in_m2"`. In this lesson, we're building a multiple linear regression model that predicts price based on two features, `"lon"` and `"lat"`. This means that our data visualizations now have to communicate three pieces of information: Longitude, latitude, and price. How can we represent these three attributes on a two-dimensional screen?

One option is to incorporate color into our scatter plot. For example, in the Mapbox scatter plot below, the location of each point represents latitude and longitude, and color represents price.

[14]:
```
VimeoVideo("656751031", h="367be02e14", width=600)
```

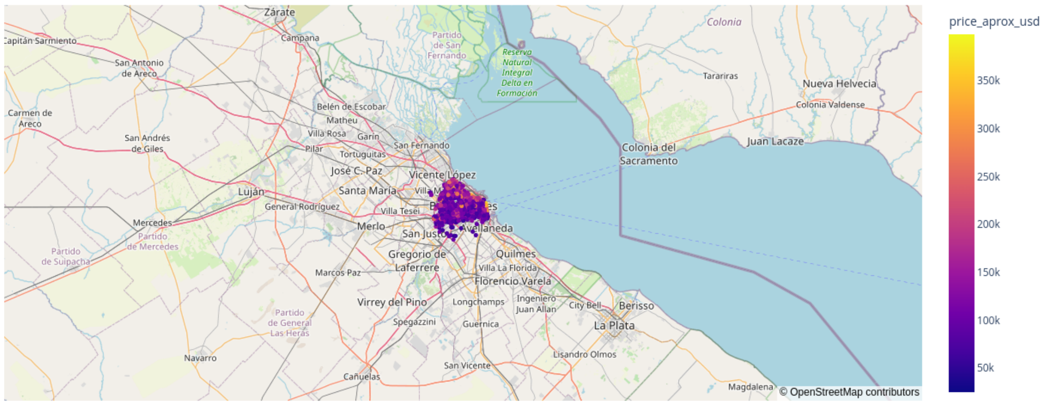
[14]:
```
<IPython.lib.display.VimeoVideo at 0x7f111f9388e0>
```

Task 2.2.5: Complete the code below to create a Mapbox scatter plot that shows the location of the apartments in `df`.

- What's a scatter plot?
- Create a Mapbox scatter plot in plotly express.

[15]:
```
fig = px.scatter_mapbox(
    df,  # Our DataFrame
    lat="lat",
    lon="lon",
    width=600,  # Width of map
    height=600,  # Height of map
    color="price_aprox_usd",
    hover_data=["price_aprox_usd"],  # Display price when hovering mouse over␣
 ↪house
)
```

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

fig.show()
```



Another option is to add a third dimension to our scatter plot. We can plot longitude on the x-axis and latitude on the y-axis (like we do in the map above), and then add a z-axis with price.

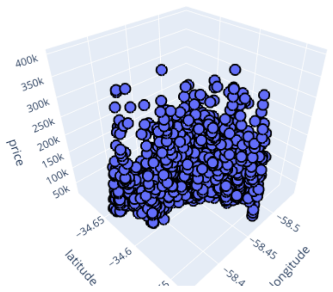
[16]:
```
VimeoVideo("656750669", h="574287f687", width=600)
```

[16]:
```
<IPython.lib.display.VimeoVideo at 0x7f111f938580>
```

Task 2.2.6: Complete the code below to create a 3D scatter plot, with "lon" on the x-axis, "lat" on the y-axis, and "price_aprox_usd" on the z-axis.

- What's a scatter plot?
- Create a 3D scatter plot in plotly express.

[17]:
```
# Create 3D scatter plot
fig = px.scatter_3d(
    df,
    x="lon",
    y="lat",
    z="price_aprox_usd",
    labels={"lon": "longitude", "lat": "latitude", "price_aprox_usd": "price"},
    width=600,
    height=500,
)

# Refine formatting
fig.update_traces(
    marker={"size": 4, "line": {"width": 2, "color": "DarkSlateGrey"}},
    selector={"mode": "markers"},
)

# Display figure
fig.show()
```



Tip: 3D visualizations are often harder for someone to interpret than 2D visualizations. We're using one here because it will help us visualize our model once it's built, but as a rule, it's better to stick with 2D when your communicating with an audience.

In the last lesson, we represented our simple model as a line imposed on a 2D scatter plot.

How do you think we'll represent our multiple linear regression model in the 3D plot we just made?

1.3 Split

Even though we're building a different model, the steps we follow will be the same. Let's separate our features (latitude and longitude) from our target (price).

[18]:
```
VimeoVideo("656750457", h="09f5fe3962", width=600)
```

[18]:
```
<IPython.lib.display.VimeoVideo at 0x7f111ef39a90>
```

Task 2.2.7: Create the feature matrix named `X_train`. It should contain two features: `["lon", "lat"]`.

- What's a feature matrix?
- Subset a DataFrame by selecting one or more columns in pandas.

[19]:
```
features = ["lon", "lat"]
X_train = df[features]
#X_train.head()
```

```
[20]: VimeoVideo("656750323", h="1a82090b9b", width=600)
```

```
[20]: <IPython.lib.display.VimeoVideo at 0x7f111f959880>
```

Task 2.2.8: Create the target vector named `y_train`, which you'll use to train your model. Your target should be `"price_aprox_usd"`. Remember that, in most cases, your target vector should be one-dimensional.

- What's a target vector?
- Select a Series from a DataFrame in pandas.

```
[21]: target = "price_aprox_usd"
      y_train = df[target]
      #y_train.head()
```

2 Build Model

2.1 Baseline

Again, we need to set a baseline so we can evaluate our model's performance. You'll notice that the value of `y_mean` is not exactly the same as it was in the previous lesson. That's because we've added more observations to our training data.

```
[22]: VimeoVideo("656750112", h="1ef669fe2b", width=600)
```

```
[22]: <IPython.lib.display.VimeoVideo at 0x7f111f959580>
```

Task 2.2.9: Calculate the mean of your target vector `y_train` and assign it to the variable `y_mean`.

- Calculate summary statistics for a DataFrame or Series in pandas.

```
[23]: y_mean = y_train.mean()
```

Task 2.2.10: Create a list named `y_pred_baseline` that contains the value of `y_mean` repeated so that it's the same length at `y_train`.

- Calculate the length of a list in Python.

```
[24]: y_pred_baseline = [y_mean] * len(y_train)
      y_pred_baseline[:5]
```

```
[24]: [134732.9734048155,
       134732.9734048155,
       134732.9734048155,
       134732.9734048155,
       134732.9734048155]
```

```
[25]: VimeoVideo("656749994", h="50c71bf4e5", width=600)
```

[25]: <IPython.lib.display.VimeoVideo at 0x7f111ef39970>

Task 2.2.11: Calculate the baseline mean absolute error for your predictions in `y_pred_baseline` as compared to the true targets in `y_train`.

- What's a performance metric?
- What's mean absolute error?
- Calculate the mean absolute error for a list of predictions in scikit-learn.

[26]:
```
mae_baseline = mean_absolute_error(y_train, y_pred_baseline)

print("Mean apt price", round(y_mean, 2))
print("Baseline MAE:", round(mae_baseline, 2))
```

```
Mean apt price 134732.97
Baseline MAE: 45422.75
```

2.2 Iterate

Take a moment to scroll up to the output for `df.info()` and look at the values in the `"Non-Null Count"` column. Because of the math it uses, a linear regression model can't handle observations where there are missing values. Do you see any columns where this will be a problem?

In the last project, we simply dropped rows that contained `NaN` values, but this isn't ideal. Models generally perform better when they have more data to train with, so every row is precious. Instead, we can fill in these missing values using information we get from the whole column — a process called **imputation**. There are many different strategies for imputing missing values, and one of the most common is filling in the missing values with the mean of the column.

In addition to **predictors** like `LinearRegression`, scikit-learn also has **transformers** that help us deal with issues like missing values. Let's see how one works, and then we'll add it to our model.

[27]:
```
VimeoVideo("656748776", h="014f943c46", width=600)
```

[27]: <IPython.lib.display.VimeoVideo at 0x7f111f91d7c0>

Task 2.2.12: Instantiate a `SimpleImputer` named `imputer`.

- What's imputation?
- Instantiate a transformer in scikit-learn.

[28]:
```
imputer = SimpleImputer()
```

[29]:
```
# Check your work
assert isinstance(imputer, SimpleImputer)
```

Just like a predictor, a transformer has a `fit` method. In the case of our `SimpleImputer`, this is the step where it calculates the mean values for each numerical column.

[30]:
```
VimeoVideo("656748659", h="fdaa8d0329", width=600)
```

```
[30]: <IPython.lib.display.VimeoVideo at 0x7f111c58b070>
```

Task 2.2.13: Fit your transformer `imputer` to the feature matrix `X`.

- Fit a transformer to training data in scikit-learn.

```
[31]: imputer.fit(X_train)
```

```
[31]: SimpleImputer()
```

```
[32]: # Check your work
      check_is_fitted(imputer)
```

Here's where transformers diverge from predictors. Instead of using a method like `predict`, we use the `transform` method. This is the step where the transformer fills in the missing values with the means it's calculated.

```
[33]: VimeoVideo("656748527", h="d76e63760c", width=600)
```

```
[33]: <IPython.lib.display.VimeoVideo at 0x7f111c58b8b0>
```

Task 2.2.14: Use your `imputer` to transform the feature matrix `X_train`. Assign the transformed data to the variable `XT_train`.

- Transform data using a transformer in scikit-learn.

```
[34]: XT_train = imputer.transform(X_train)
      pd.DataFrame(XT_train, columns=X_train.columns).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2658 entries, 0 to 2657
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   lon     2658 non-null   float64
 1   lat     2658 non-null   float64
dtypes: float64(2)
memory usage: 41.7 KB
```

```
[35]: # Check your work
      assert XT_train.shape == (2658, 2), f"`XT_train` is the wrong shape: {XT_train.
       ↪shape}"
      assert (
          np.isnan(XT_train).sum() == 0
      ), "Your feature matrix still has `NaN` values. Did you forget to transform is␣
       ↪using `imputer`?"
```

Okay! Our data is free of missing values, and we have a good sense for how predictors work in scikit-learn. However, the truth is you'll rarely do data transformations this way. Why? A model may require multiple transformers, and doing all those transformations one-by-one is slow and likely to lead to errors. Instead, we can combine our transformer and predictor into a single object called a `pipeline`.

```
[36]: VimeoVideo("656748360", h="50b4643a26", width=600)
```

```
[36]: <IPython.lib.display.VimeoVideo at 0x7f1119c1d970>
```

Task 2.2.15: Create a pipeline named `model` that contains a `SimpleImputer` transformer followed by a `LinearRegression` predictor.

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

```
[38]: model = make_pipeline(
          SimpleImputer(),
          LinearRegression()

      )
```

```
[39]: assert isinstance(model, Pipeline), "Did you instantiate your model?"
```

With our pipeline assembled, we use the `fit` method, which will train the transformer, transform the data, then pass the transformed data to the predictor for training, all in one step. Much easier!

```
[40]: VimeoVideo("656748234", h="59ba7958d5", width=600)
```

```
[40]: <IPython.lib.display.VimeoVideo at 0x7f1117584a60>
```

Task 2.2.16: Fit your model to the data, `X_train` and `y_train`.

- Fit a model to training data in scikit-learn.

```
[41]: model.fit(X_train, y_train)
```

```
[41]: Pipeline(steps=[('simpleimputer', SimpleImputer()),
                      ('linearregression', LinearRegression())])
```

```
[42]: # Check your work
      check_is_fitted(model["linearregression"])
```

Success! Let's see how our trained model performs.

2.3 Evaluate

As always, we'll start by evaluating our model's performance on the training data.

```
[43]: VimeoVideo("656748155", h="5672ef44cb", width=600)
```

```
[43]: <IPython.lib.display.VimeoVideo at 0x7f1117584610>
```

Task 2.2.17: Using your model's `predict` method, create a list of predictions for the observations in your feature matrix `X_train`. Name this list `y_pred_training`.

- Generate predictions using a trained model in scikit-learn.

```
[44]: y_pred_training = model.predict(X_train)
```

```
[45]: # Check your work
      assert y_pred_training.shape == (2658,)
```

```
[46]: VimeoVideo("656748205", h="13144556a6", width=600)
```

```
[46]: <IPython.lib.display.VimeoVideo at 0x7f111761ffa0>
```

Task 2.2.18: Calculate the training mean absolute error for your predictions in `y_pred_training` as compared to the true targets in `y_train`.

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

```
[47]: mae_training = mean_absolute_error(y_train, y_pred_training)
      print("Training MAE:", round(mae_training, 2))
```

```
Training MAE: 42962.72
```

It looks like our model performs a little better than the baseline. This suggests that latitude and longitude aren't as strong predictors of price as size is.

Now let's check our test performance. Remember, once we test our model, there's no more iteration allowed.

Task 2.2.19: 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.

```
[48]: 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()
```

```
[48]: 0    136372.324695
      1    168620.352353
      2    130231.628267
      3    102497.549527
      4    123482.077850
      dtype: float64
```

```
[49]: wqet_grader.grade("Project 2 Assessment", "Task 2.2.19", y_pred_test)
```

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

Again, we want our test performance to be about the same as our training performance, but it's OK if it's not quite as good.

3 Communicate Results

Let's take a look at the equation our model has come up with for predicting price based on latitude and longitude. We'll need to expand on our formula to account for both features.

[50]:
```
VimeoVideo("656747630", h="b90db6b373", width=600)
```

[50]:
```
<IPython.lib.display.VimeoVideo at 0x7f111761ff70>
```

Task 2.2.20: 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.

[52]:
```
intercept = model.named_steps["linearregression"].intercept_.round(2)
coefficients = model.named_steps["linearregression"].coef_.round(2)
#intercept
#coefficients
```

[52]:
```
array([196709.42, 765466.58])
```

Task 2.2.21: Complete the code below and run the cell to print the equation that your model has determined for predicting apartment price based on latitude and longitude.

- What's an f-string?

[55]:
```
print(

    f"price = {intercept} + ({coefficients[0]} * longitude) +␣
 ↪({coefficients[1]} * latitude)"
)
```

```
price = 38113587.05 + (196709.42 * longitude) + (765466.58 * latitude)
```

What does this equation tell us? As you move north and west, the predicted apartment price increases.

At the start of the notebook, you thought about how we would represent our linear model in a 3D plot. If you guessed that we would use a plane, you're right!

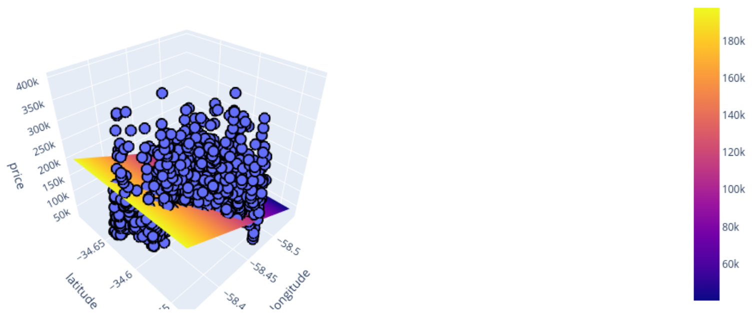
[54]:
```
VimeoVideo("656746928", h="71bfe94764", width=600)
```

[54]:
```
<IPython.lib.display.VimeoVideo at 0x7f1114d25130>
```

Task 2.2.22: Complete the code below to create a 3D scatter plot, with "lon" on the x-axis, "lat" on the y-axis, and "price_aprox_usd" on the z-axis.

- What’s a scatter plot?
- Create a 3D scatter plot in plotly express.

[56]:
```
# Create 3D scatter plot
fig = px.scatter_3d(
    df,
    x="lon",
    y="lat",
    z="price_aprox_usd",
    labels={"lon": "longitude", "lat": "latitude", "price_aprox_usd": "price"},
    width=600,
    height=500,
)

# Create x and y coordinates for model representation
x_plane = np.linspace(df["lon"].min(), df["lon"].max(), 10)
y_plane = np.linspace(df["lat"].min(), df["lat"].max(), 10)
xx, yy = np.meshgrid(x_plane, y_plane)

# Use model to predict z coordinates
z_plane = model.predict(pd.DataFrame({"lon": x_plane, "lat": y_plane}))
zz = np.tile(z_plane, (10, 1))

# Add plane to figure
fig.add_trace(go.Surface(x=xx, y=yy, z=zz))

# Refine formatting
fig.update_traces(
    marker={"size": 4, "line": {"width": 2, "color": "DarkSlateGrey"}},
    selector={"mode": "markers"},
)

# Display figure
fig.show()
```


price
400k
350k
300k
250k
200k
150k
100k
50k
latitude
longitude
180k
160k
140k
120k
100k
80k
60k