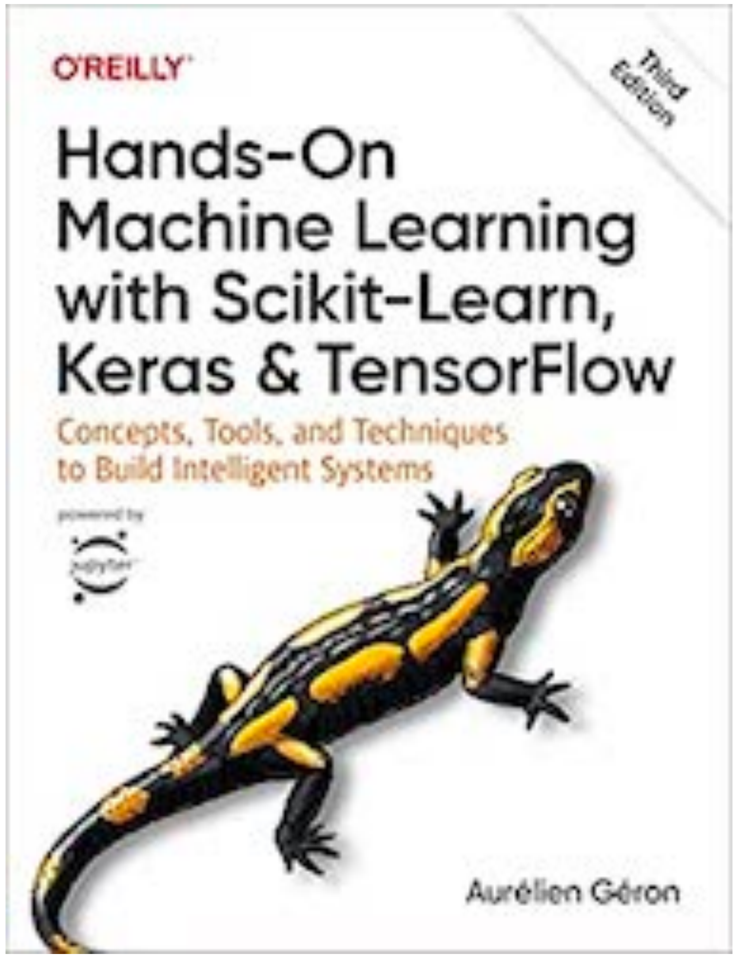
Machine Learning Security

2 End-to-End Machine Learning Project



Made Aug 22, 2023

Steps in an ML Project

1 Look at the big picture

2 Get the data

3 Explore and visualize the data to gain insights

4 Prepare the data for machine learning algorithms

5 Select a model and train it

6 Fine-tune your model

7 Present your solution

8 Launch, monitor, and maintain your system

Getting Real Data

- Popular open data repositories:
  - OpenML.org
  - Kaggle.com
  - PapersWithCode.com
  - UC Irvine Machine Learning Repository
  - Amazon's AWS datasets
  - TensorFlow datasets
- Meta portals (they list open data repositories):
  - DataPortals.org
  - OpenDataMonitor.eu
- Other pages listing many popular open data repositories:
  - Wikipedia's list of machine learning datasets
  - Quora.com
  - The datasets subreddit

1 Look At The Big Picture

Frame the Problem

- The goal is to predict the median housing price from the other metrics in the data, such as number of bedrooms, location, and income in the area.
- The prediction will be used to make investment decisions.
- See the **data pipeline** below

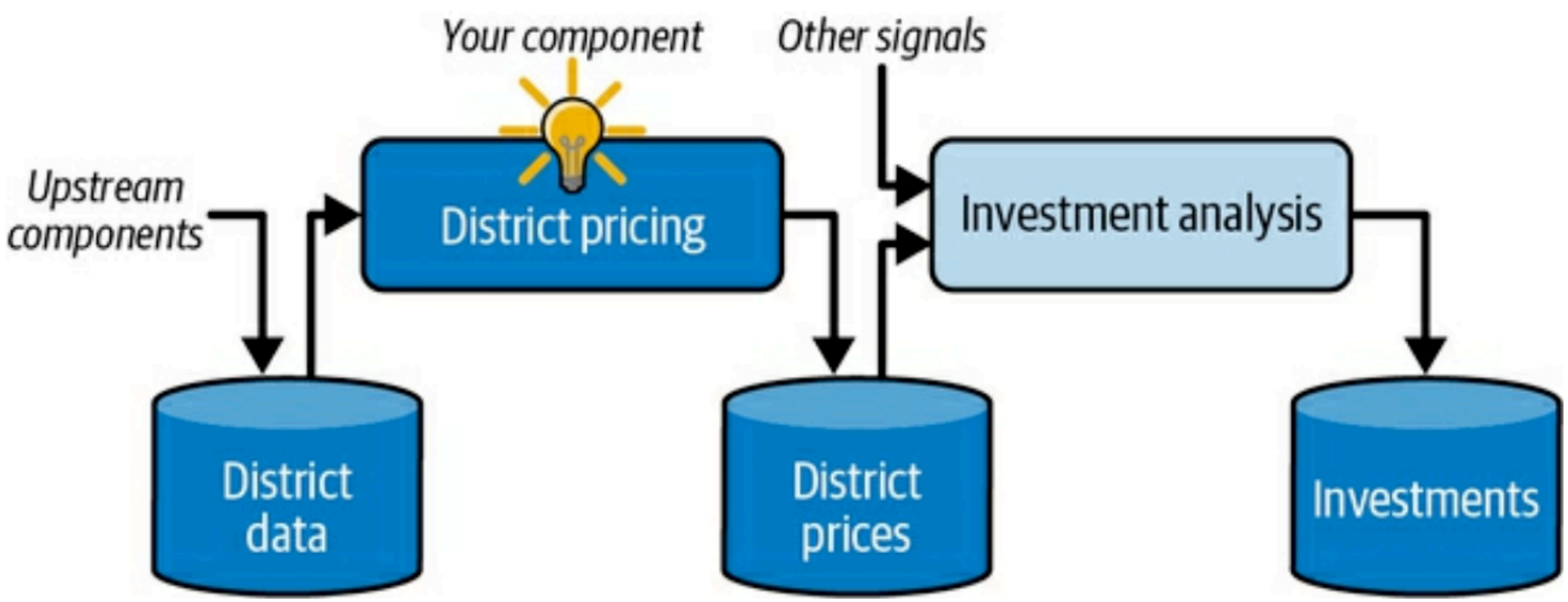


Figure 2-2. A machine learning pipeline for real estate investments

System Design

- Supervised learning
  - Data is labeled
- Regression
  - Model will predict a value
- Batch learning
  - No additional data will be added later

Types of Regression

- **Multiple regression**
  - Uses multiple features to predict a value
- **Univariate regression**
  - Predicts a single value
- **Multivariate regression**
  - Predicts multiple values

Select a Performance Measure

- Root Mean Square Error (RMSE)
  - Adds up the error for each item of data
  - The most commonly used measure for regression tasks

$$\mathrm{RMSE}\left(\mathbf{X}, h\right) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

  - Also called the **Euclidean norm**, or ℓ_2

Select a Performance Measure

- Mean Absolute Error (MAE)
  - Preferred if data has many outliers
  - Also called **Manhattan norm**, or ℓ_1

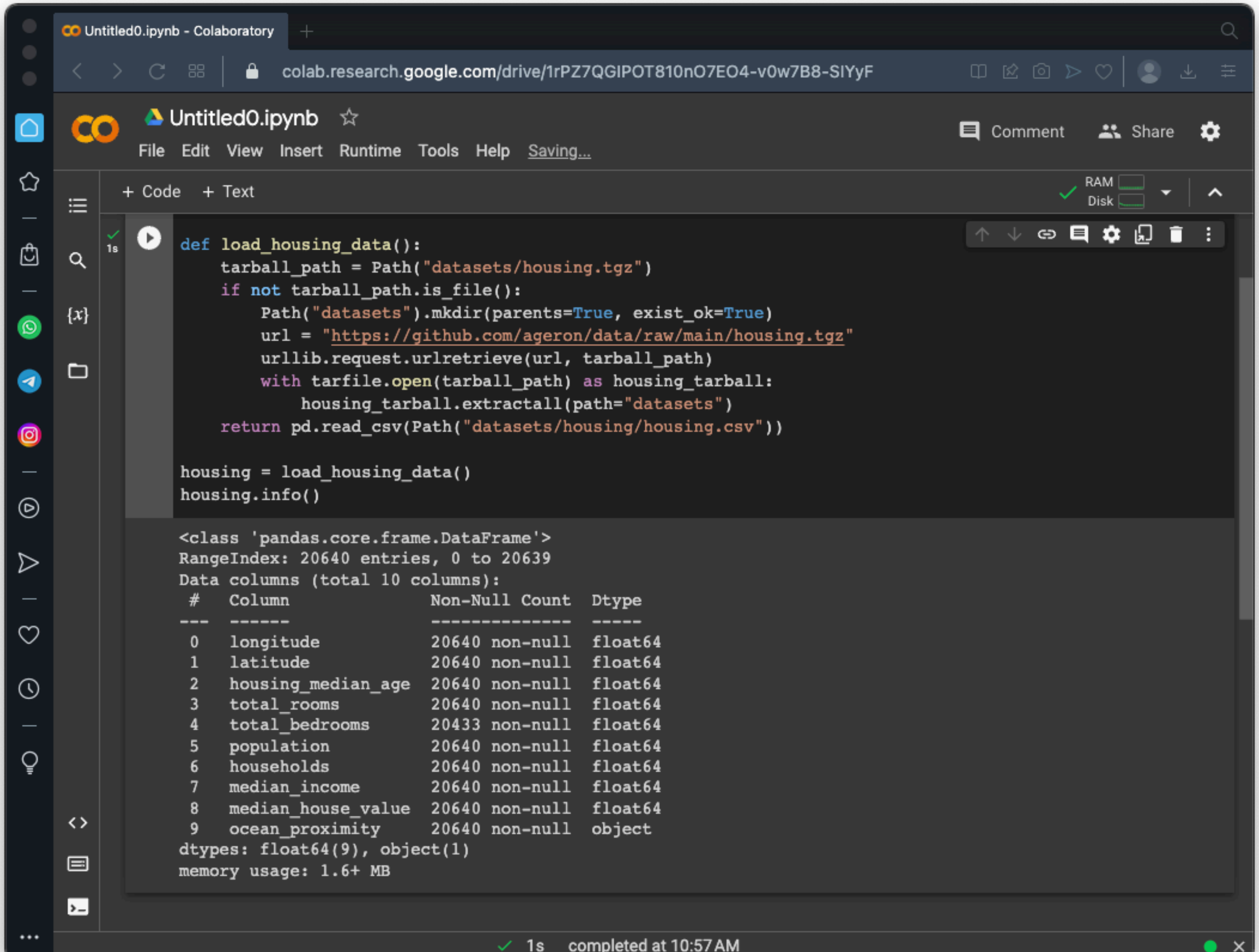
$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^{m} \left| h\left(\mathbf{x}^{(i)}\right) - y^{(i)} \right|$$

Check the Assumptions

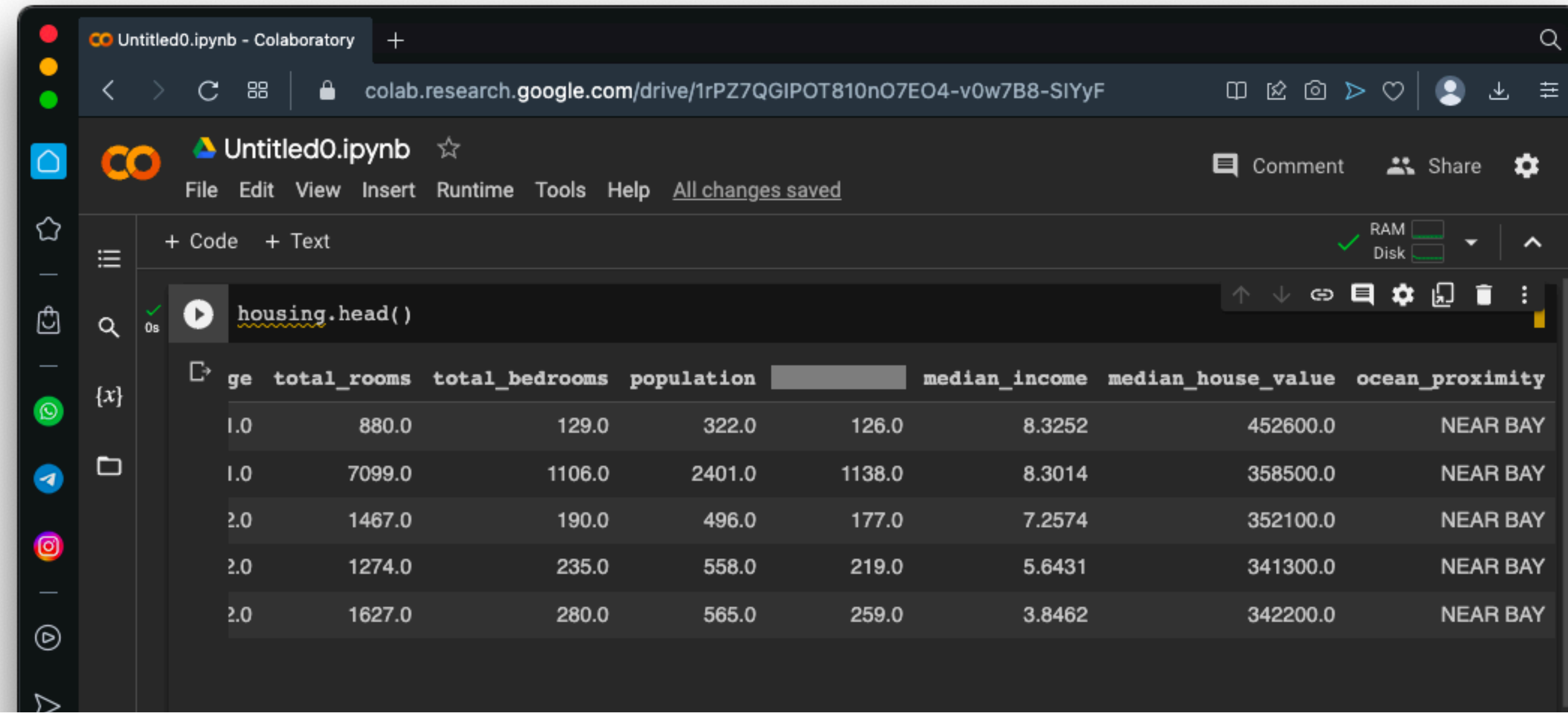
- We're assuming the price will be used as a numerical value
- If the next stage just uses categories, like "cheap", "medium", or "expensive" we should be using classification instead of regression

2 Get The Data

Load Data from Github

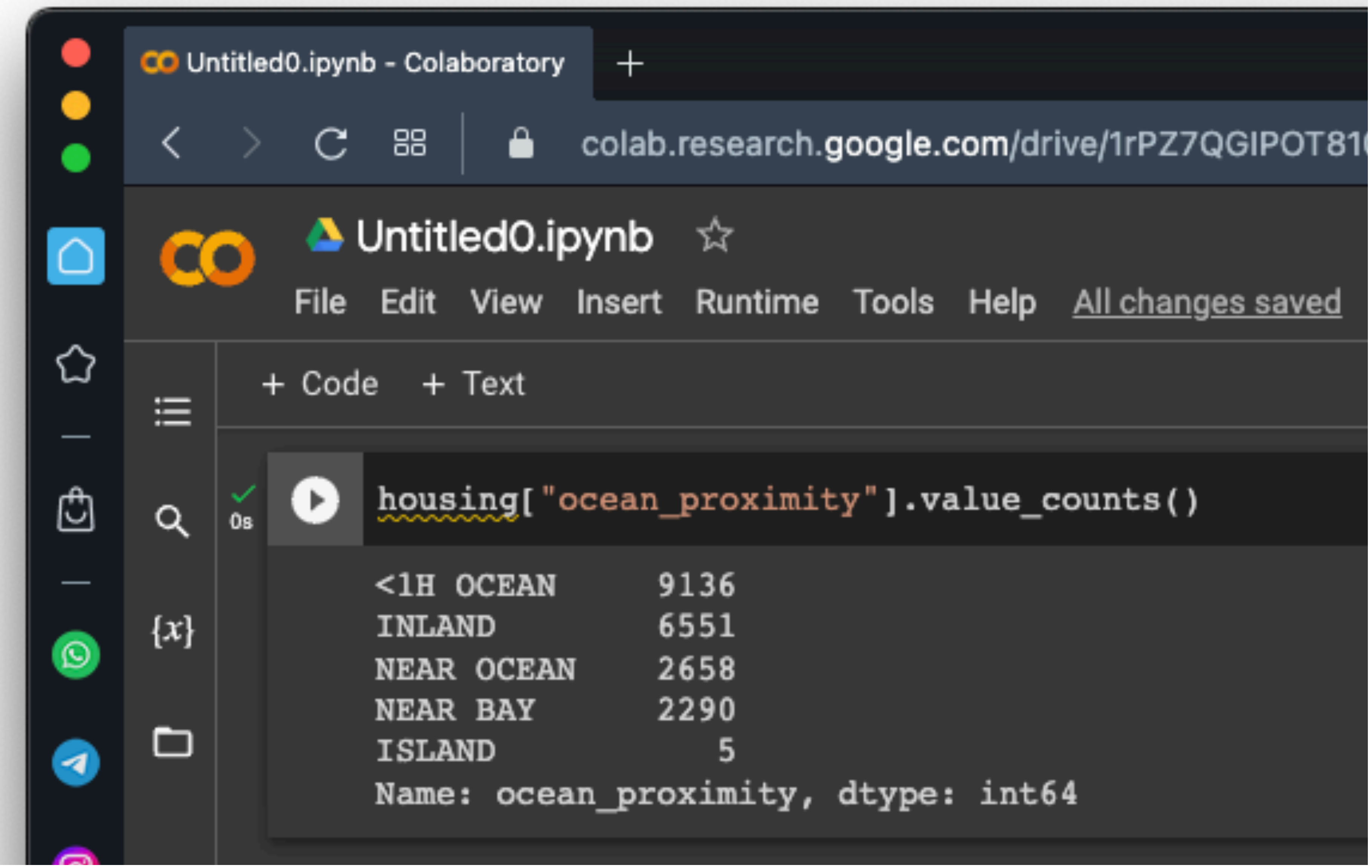


head() Shows First Five Rows

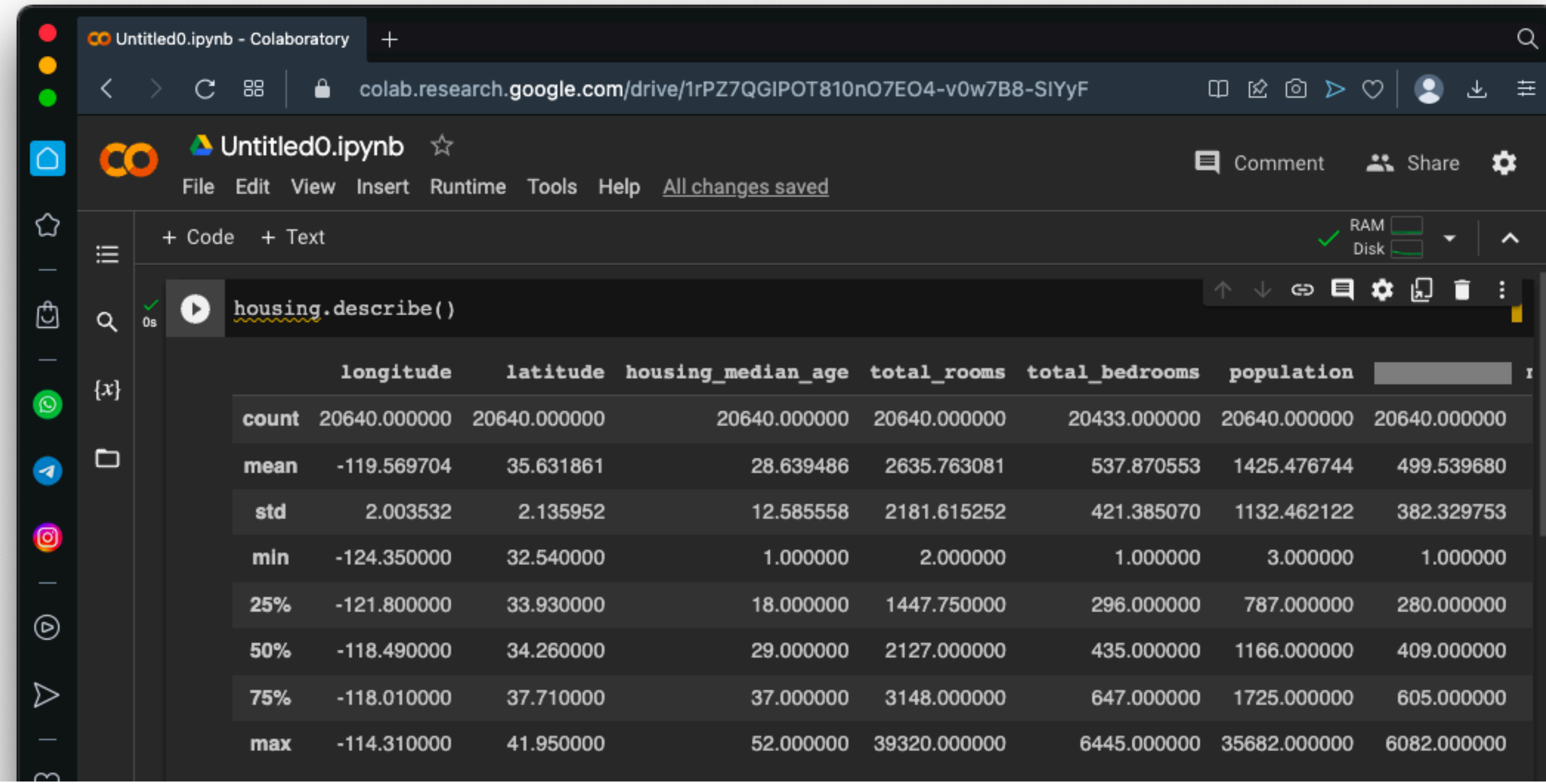


value_counts()

- ocean_proximity is not numeric



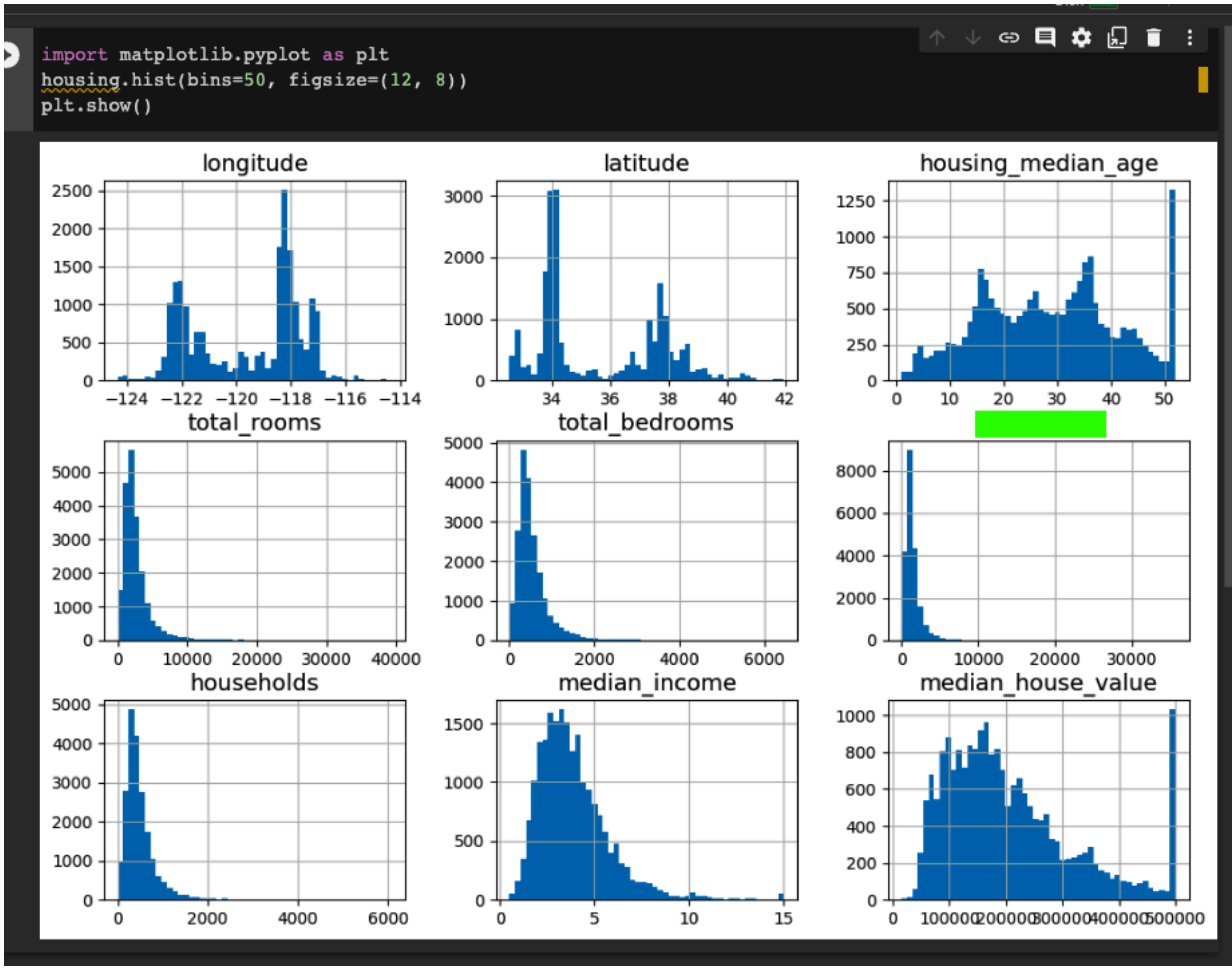
describe() Shows Statistics



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000

Histograms

- Show distribution of numerical attributes



Median Income

- It's not in dollars
- It's been scaled and capped at 15 max and 0.5 min
- Numbers represent roughly tens of thousands of dollars
- Preprocessed attributes are common in ML, this should be OK

Other Capped Values

- Housing median age and median house value were capped
- Median house value is our target, which we want to predict
- It being capped limits the value of our model
- If we want to predict beyond $500,000, there are two options:
  - Collect proper labels for the capped districts
  - Remove those districts from the training and test sets

Scale and Skewing

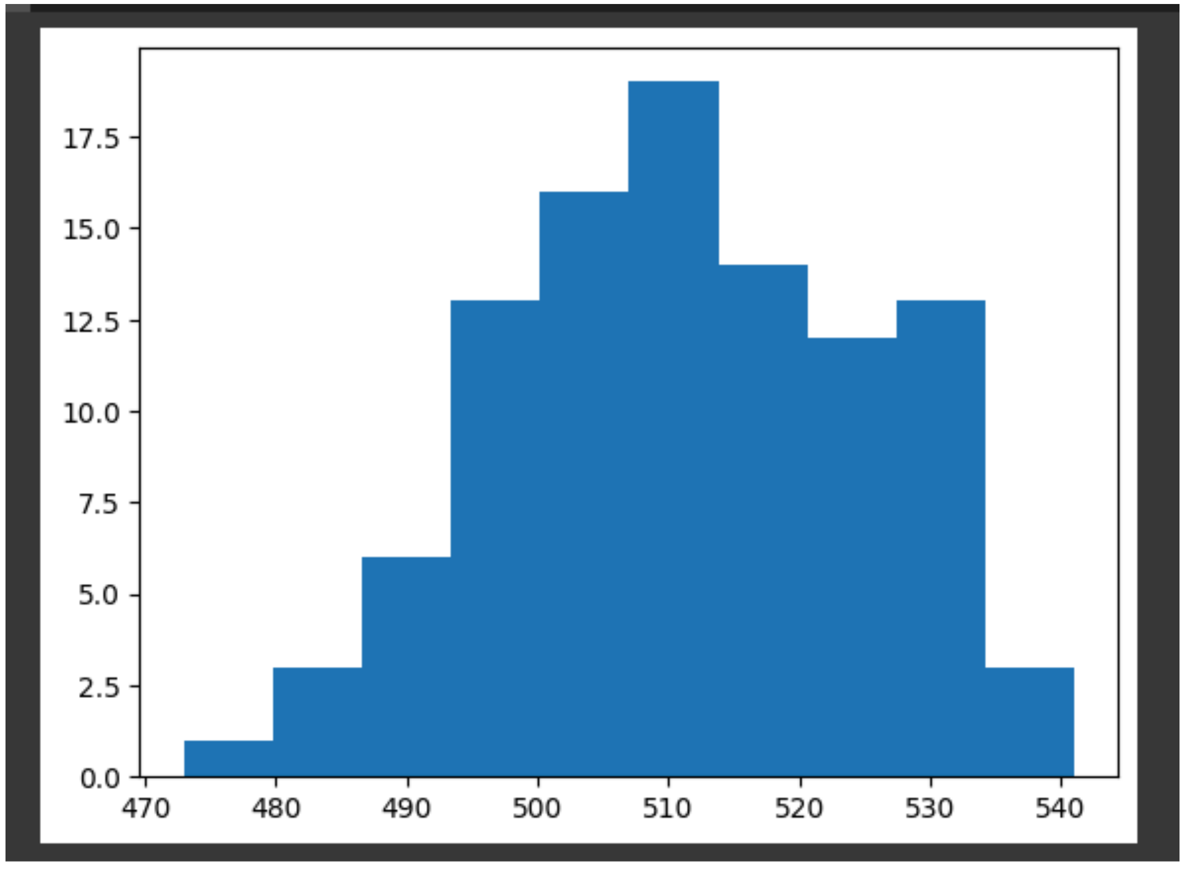
- These attributes have very different scales
  - We'll fix them with **feature scaling**
- Many histograms are **skewed right**
  - They extend more to the right than the left
  - We'll transform them to fix that

Test Sets

- Take 20% of the data and set it aside
- There are two ways to choose the test set
  - Randomly
  - Stratified sampling

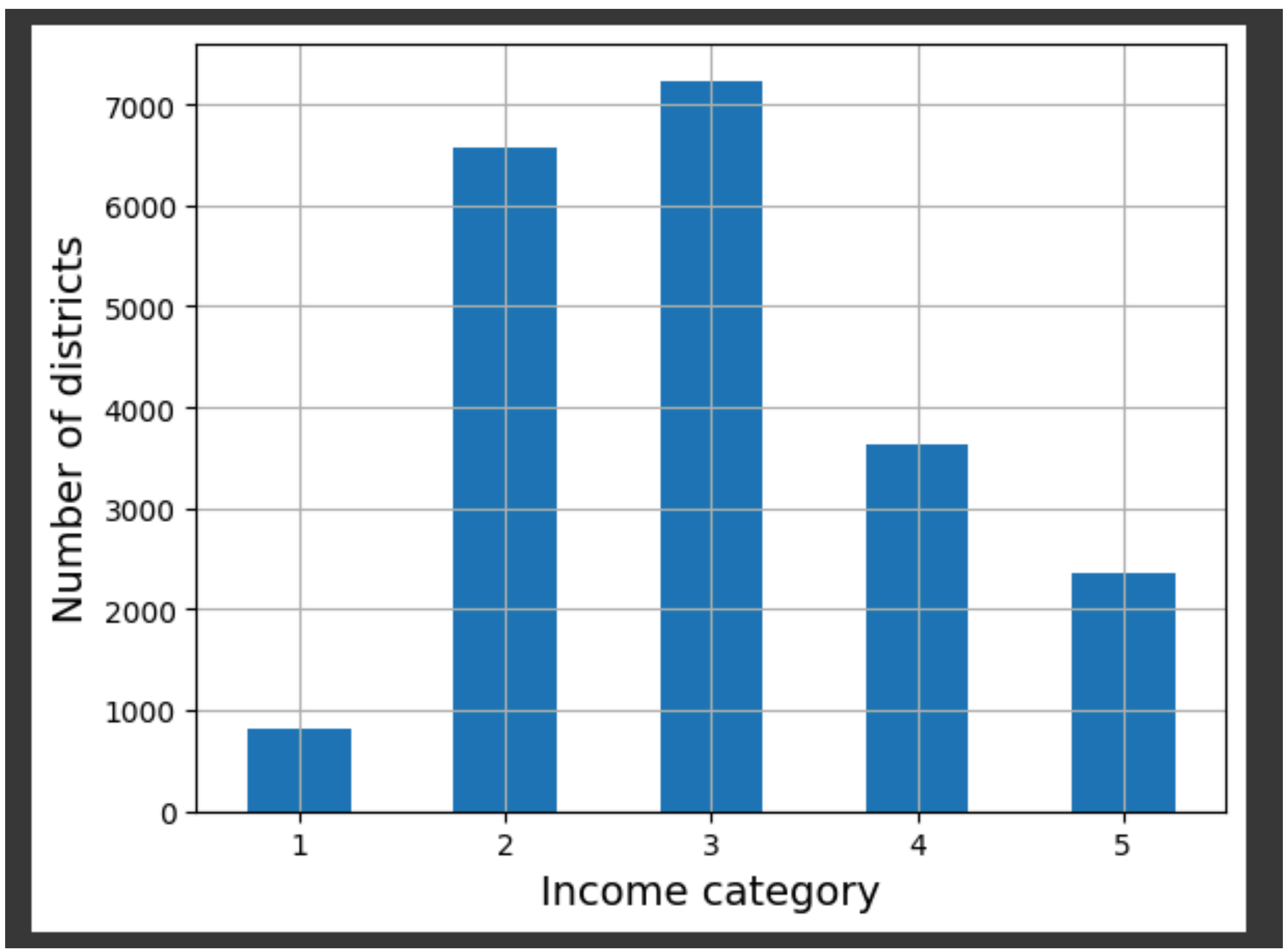
Random Sampling

- Fine for large data sets
- But may introduce sampling bias
- Consider a sample from a population that is 51% female
- A random sample
    - Might contain only 48%
    - or 54% females



Stratified Sampling

- Take the important feature and gather it into categories
- Then sample the correct number from each category
- Training and test sets match now




```
Training set:
3    0.350594
2    0.318859
4    0.176296
5    0.114462
1    0.039789
Name: income_cat,

Test set:
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
```

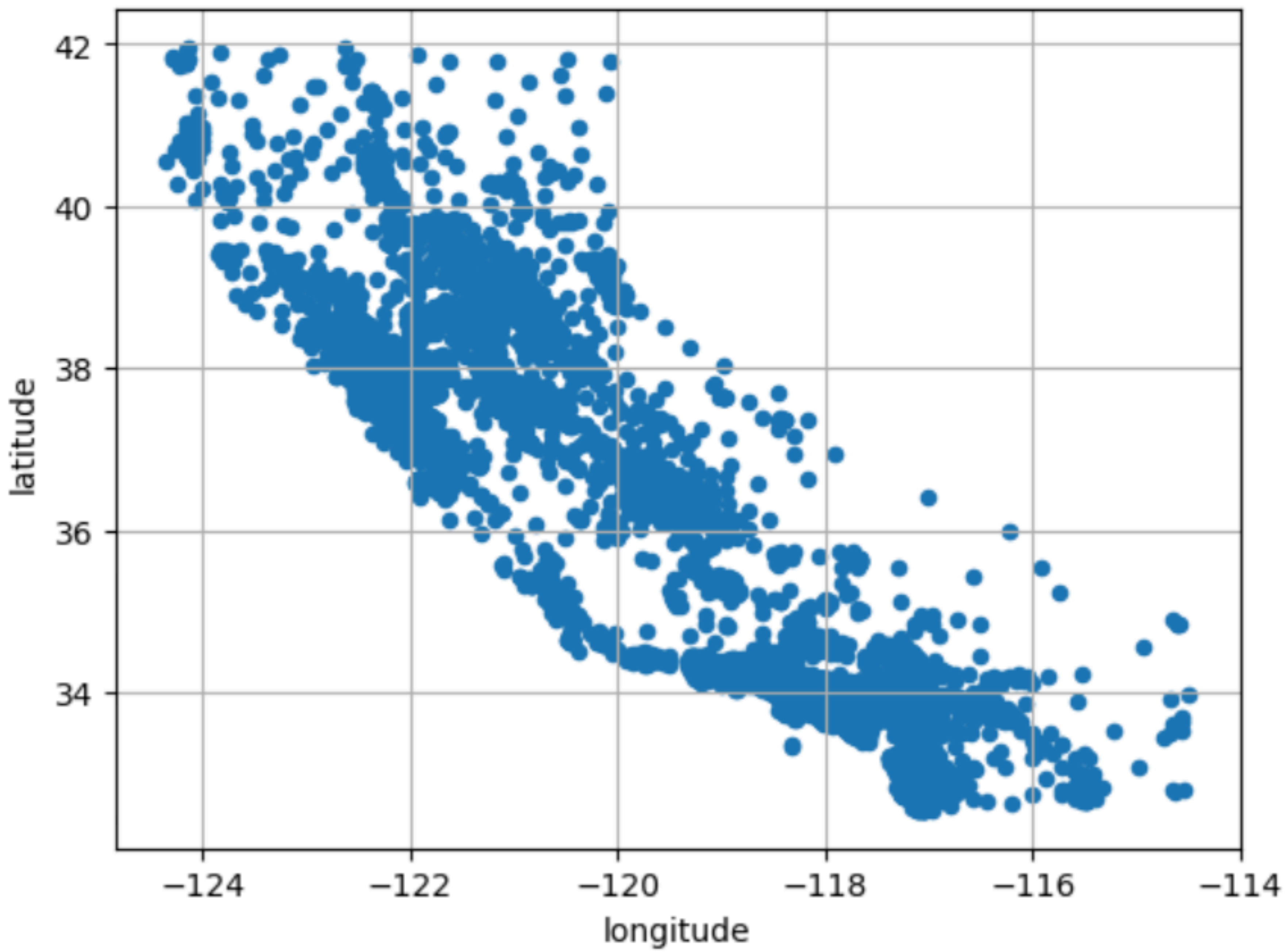
Kahoot!

Ch 2a

3 Explore And Visualize The Data To Gain Insights

Visualizing Geographical Data

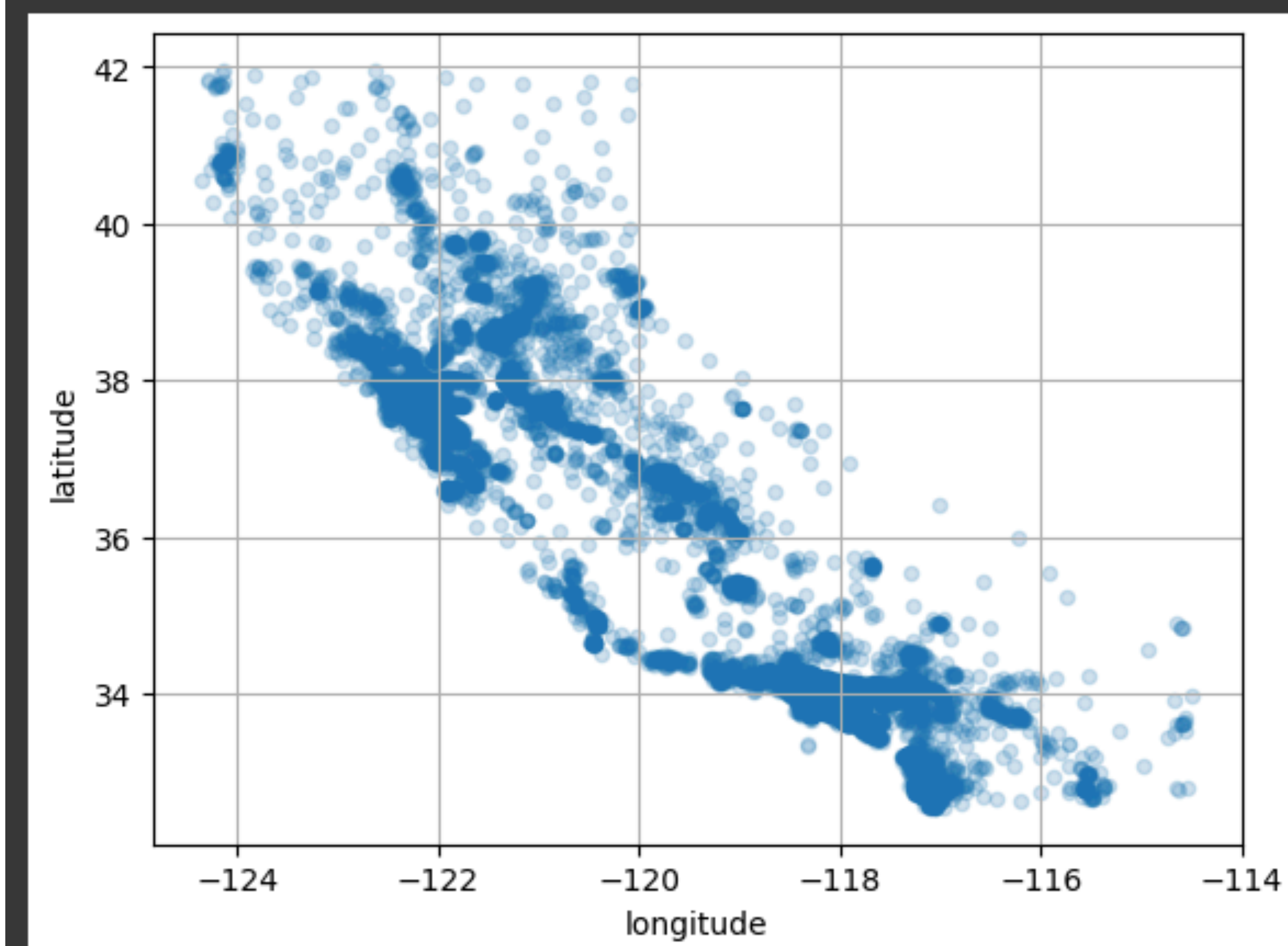
- Scatterplot misses detail as dots cover other dots



Transparency

- Alpha = 0.2 shows more detail in high-density areas

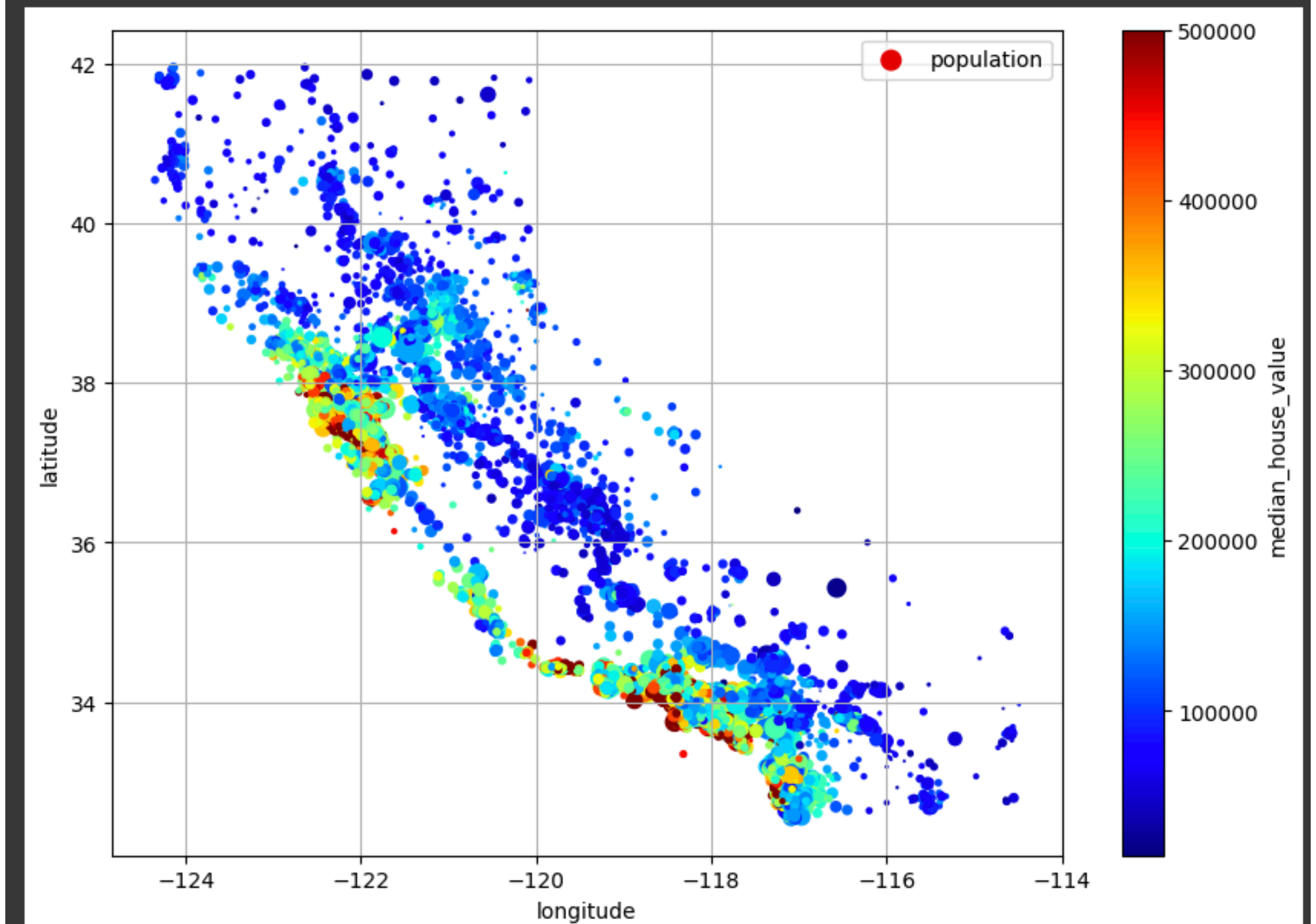
```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True, alpha=0.2)
plt.show()
```



Add Price with Color

- Areas near the ocean and with higher population density have higher prices

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True,
             s=housing["population"] / 100, label="population",
             c="median_house_value", cmap="jet", colorbar=True,
             legend=True, sharex=False, figsize=(10, 7))
plt.show()
```



Correlations

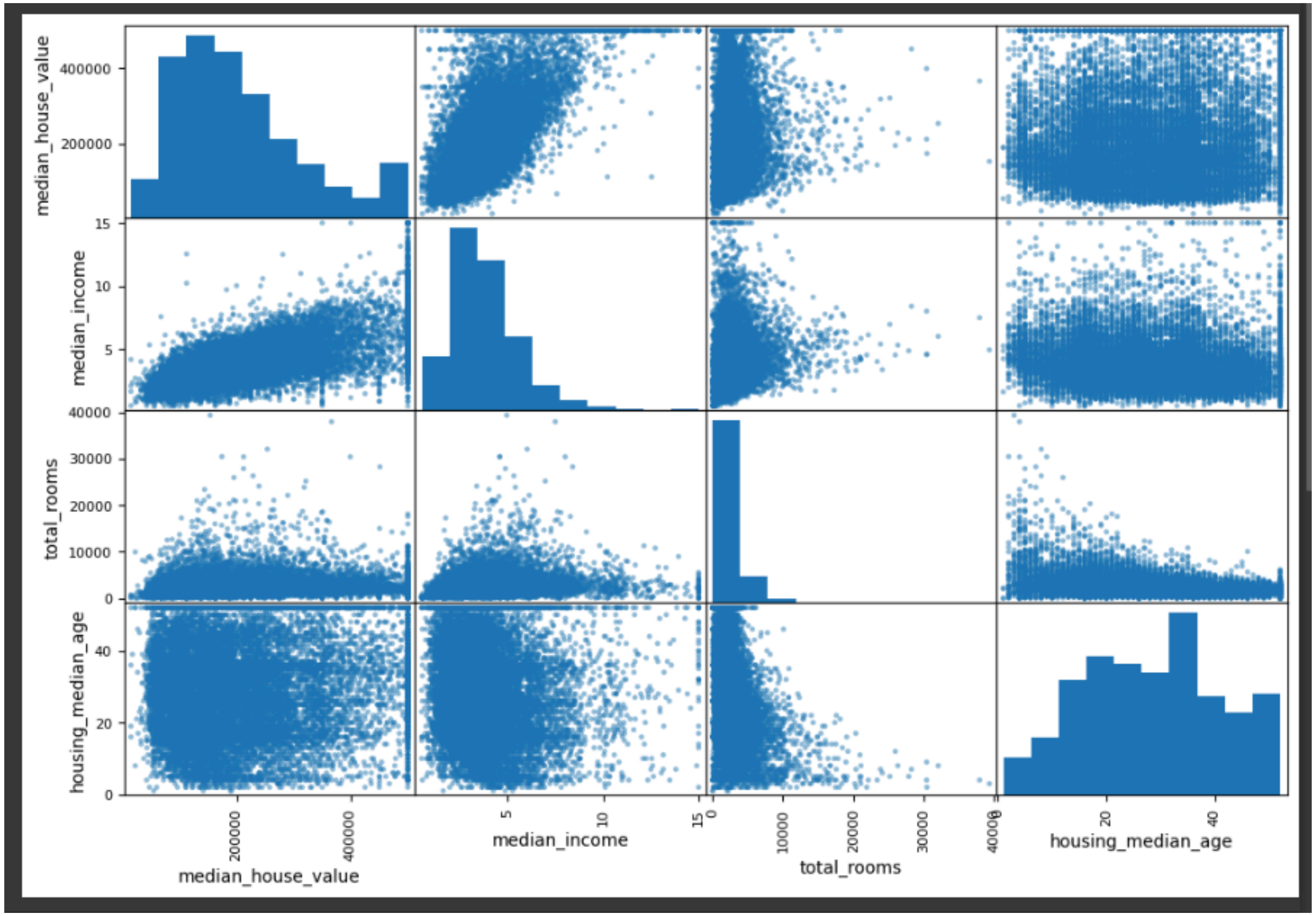
- Strongest correlations with median_house_value:
    - median_income, total_rooms, housing_median_age, latitude

```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
<ipython-input-24-51a0e6bf2eb4>:1: FutureWarning: The default val
  corr_matrix = housing.corr()
median_house_value    1.000000
median_income         0.688380
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
longitude            -0.050859
latitude             -0.139584
```

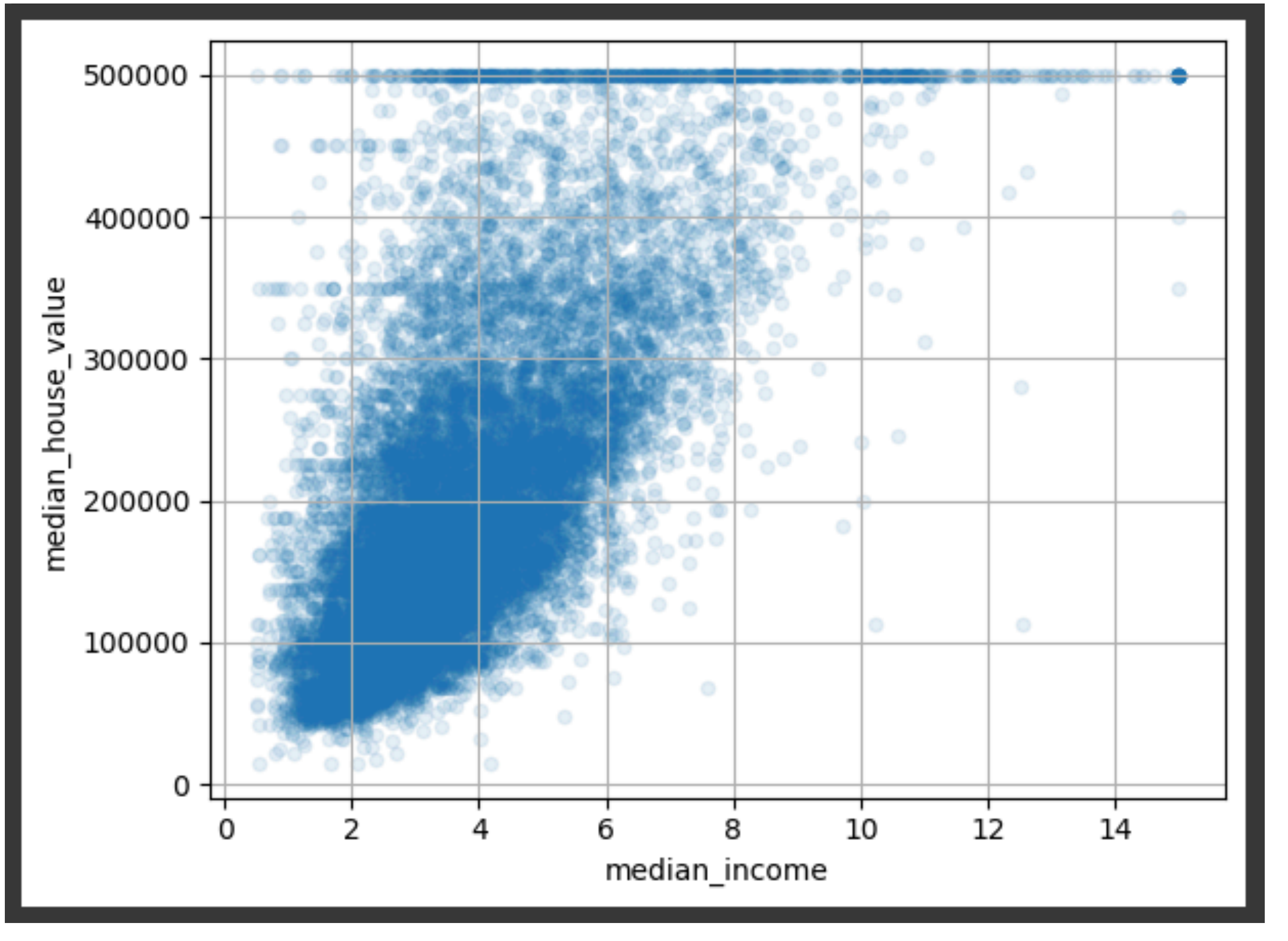
Scatter Matrix

- Strongest relationship is median_income



median_income

- Correlation is strong
- Clusters of points at $500,000. $450,000. and $350,000



Correlation Assumes a Line

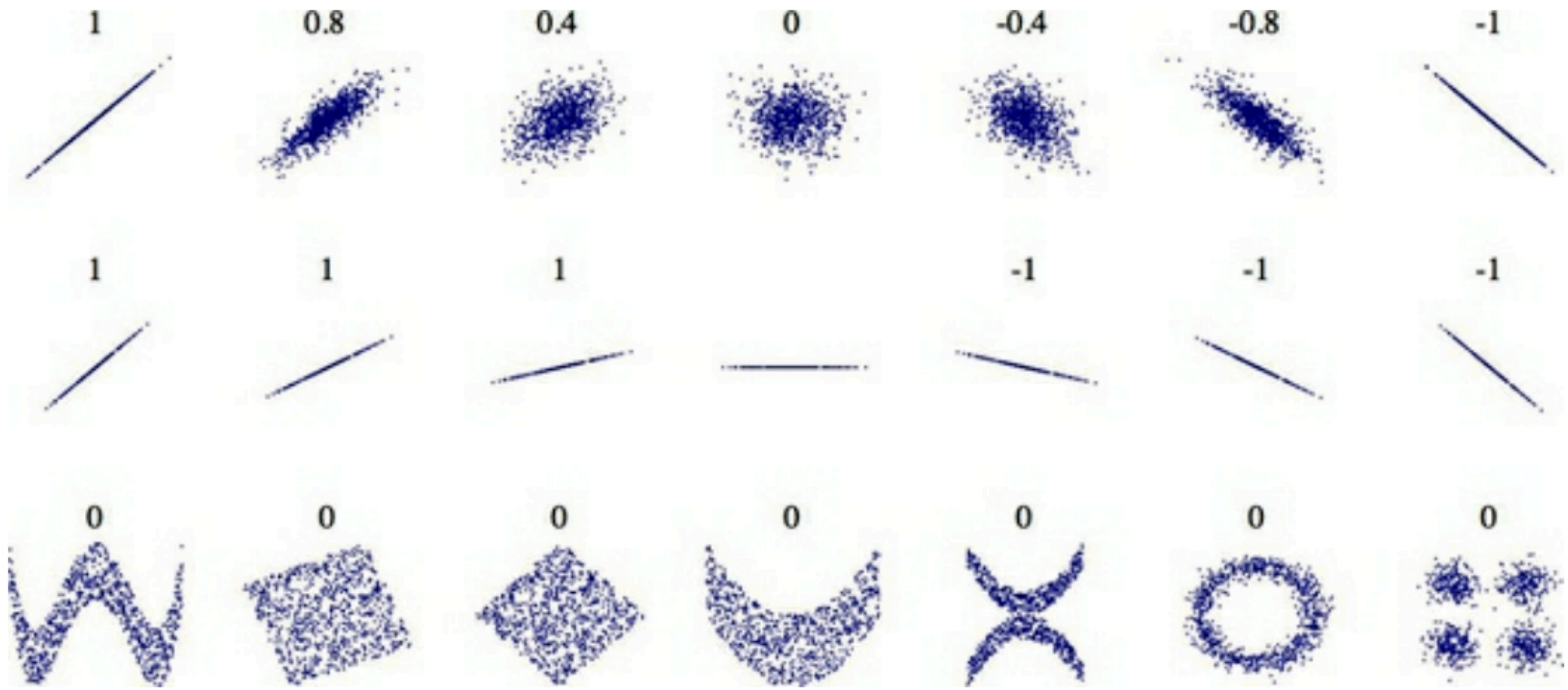


Figure 2-16. Standard correlation coefficient of various datasets (source: Wikipedia; public domain image)

Experiment with Attribute Combinations

```
housing["rooms_per_house"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_ratio"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["people_per_house"] = housing["population"] / housing["households"]
```

And then you look at the correlation matrix again:

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.688380
rooms_per_house       0.143663
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
people_per_house     -0.038224
longitude            -0.050859
latitude             -0.139584
bedrooms_ratio       -0.256397
Name: median_house_value, dtype: float64
```

- bedrooms_ratio has a high correlation

4 Prepare The Data For Machine Learning Algorithms

Clean the data

- Some data is missing the total_bedrooms value.
- Three ways to fix this:
  - Get rid of the corresponding districts.
  - Get rid of the whole attribute.
  - Set the missing values to some value (zero, the mean, the median, etc.). This is called **imputation**.

Handling Text and Categorical Attributes

- ocean_proximity has only a few values
- Replacing them with numbers will make it easier for ML to handle the data
  - But falsely implies that some values are closer to others

```
housing["ocean_proximity"].value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
```

One-Hot Vectors

- A better way to represent such data

```
>>> housing_cat_1hot.toarray()
array([[0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.]])
```

Feature Scaling and Transformation

- Number of rooms ranges from 6 to 39,320
- Median incomes range from 0 to 15
- Models will weight number of rooms far more highly than income
- To prevent this, scale data in one of two ways:
- **min-max scaling**
  - Every value ranges from 0 to 1
  - Or -1 to 1 for neural nets
- **standardization**
  - Subtract the mean, then divide by standard deviation
  - Does not limit the range strictly
  - Less affected by outliers

Heavy Tail

- Values far from the mean are not exponentially rare
- Take square root or log to get closer to a Gaussian
  - Do this before normalization
- Another solution is **bucketizing**
  - Grouping values into ranges

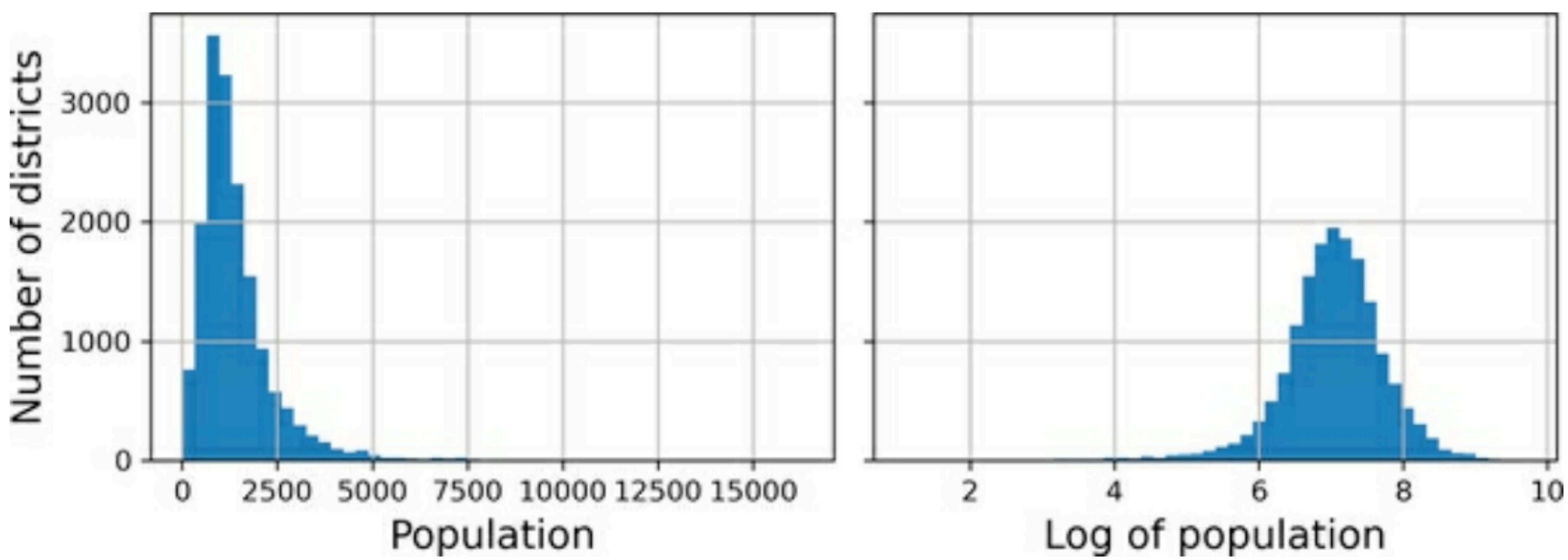


Figure 2-17. Transforming a feature to make it closer to a Gaussian distribution

5 Select A Model And Train It

Linear Regression

```
from sklearn.linear_model import LinearRegression

lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(housing, housing_labels)
```

- The first prediction is off by more than $200,000!

```
>>> housing_predictions = lin_reg.predict(housing)
>>> housing_predictions[:5].round(-2)  # -2 = rounded to the nearest hundred
array([243700., 372400., 128800.,  94400., 328300.])
>>> housing_labels.iloc[:5].values
array([458300., 483800., 101700.,  96100., 361800.])
```

Linear Regression

- The root mean squared error is over $68,000
- The median_housing_values range from $120,000 to $265,000
- Pretty bad predictions

```
>>> from sklearn.metrics import mean_squared_error
>>> lin_rmse = mean_squared_error(housing_labels, housing_predictions,
...                               squared=False)
...
>>> lin_rmse
68687.89176589991
```

DecisionTreeRegressor

- A more powerful model capable of finding complex nonlinear relationships

```python
from sklearn.tree import DecisionTreeRegressor

tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))
tree_reg.fit(housing, housing_labels)
```

Now that the model is trained, you evaluate it on the training set:

```python
>>> housing_predictions = tree_reg.predict(housing)
>>> tree_rmse = mean_squared_error(housing_labels, housing_predictions,
...                                squared=False)
...
>>> tree_rmse
0.0
```

- Zero error suggests overfitting

Better Evaluation Using Cross-Validation

- Splits the training set into 10 subsets called **folds**
- Trains the model 10 times on 9 folds
  - Evaluating each one on the remaining fold

```
from sklearn.model_selection import cross_val_score

tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
                              scoring="neg_root_mean_squared_error", cv=10)
```

- Result is as bad as linear regression

Let's look at the results:

```
>>> pd.Series(tree_rmses).describe()
count       10.000000
mean     66868.027288
std       2060.966425
min      63649.536493
25%      65338.078316
50%      66801.953094
75%      68229.934454
max      70094.778246
dtype: float64
```

RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = make_pipeline(preprocessing,
                           RandomForestRegressor(random_state=42))
forest_rmses = -cross_val_score(forest_reg, housing, housing_labels,
                                scoring="neg_root_mean_squared_error", cv=10)
```

- Results are somewhat better, Error $47,000
- But on the training set, the error is $17,000
- Still a lot of overfitting

```
>>> pd.Series(forest_rmses).describe()
count       10.000000
mean     47019.561281
std       1033.957120
min      45458.112527
25%      46464.031184
50%      46967.596354
75%      47325.694987
max      49243.765795
dtype: float64
```

6 Fine-Tune Your Model

Grid Search

- Scikit-Learn's **GridSearchCSV** class
- Tell it which hyperparameters you want to try, and what values to try
- It will use cross-validation to evaluate them

Randomized Search

- Evaluates a fixed number of random hyperparameter values
- Useful when the hyperparameter search space is large

Ensemble Methods

- Combines several models together

8 Launch, Monitor, And Maintain Your System

Launch, Monitor, and Maintain Your System

- Deploy your trained model as needed
  - Perhaps as a Web app

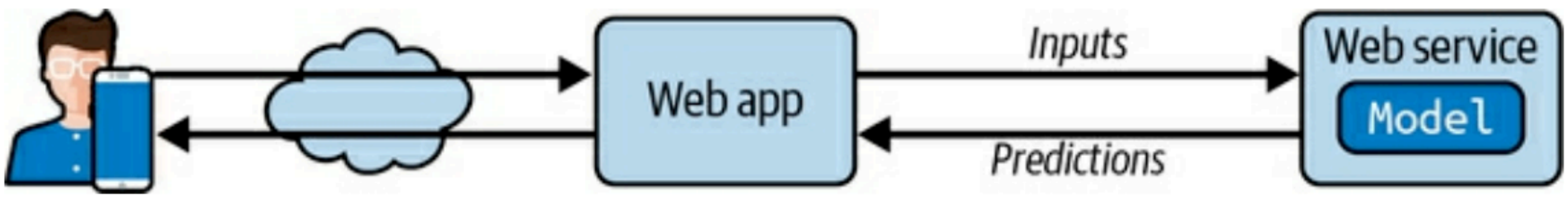


Figure 2-20. A model deployed as a web service and used by a web application

Performance Monitoring

- A component may break, causing performance to drop
- Or it may drop gradually, die to **model rot**
  - The parameters go out of date
- One measure of performance is downstream metrics
  - Number of recommended products sold per day
- Or send human raters sample pictures of products the model classified, to verify them
- It can be a lot of work to set up good performance monitoring

Automatic Updating and Retraining

- Collect fresh data and label it
- Write a script to train the moden and fine-tune the hyperparameters periodically
- Write another script to evaluate both the new model and the previous model on the updated test set
- Evaluate input data quality
- Keep backups of every model
    - Be ready to roll back

Kahoot!

Ch 1b