Lecture 2
Data Pre-processing

How many of you use ChatGPT / other LLMs ?

Well don’t ask everything to ChatGPT

- Some of our recent research work …

1. ChatGPT in the Classroom: An Analysis of Its Strengths and Weaknesses for Solving Undergraduate Computer Science Questions Link

2. ‘It’s not like Jarvis, but it’s pretty close!’ - Examining ChatGPT’s Usage among Undergraduate Students in Computer Science   Link

Working with Real Data

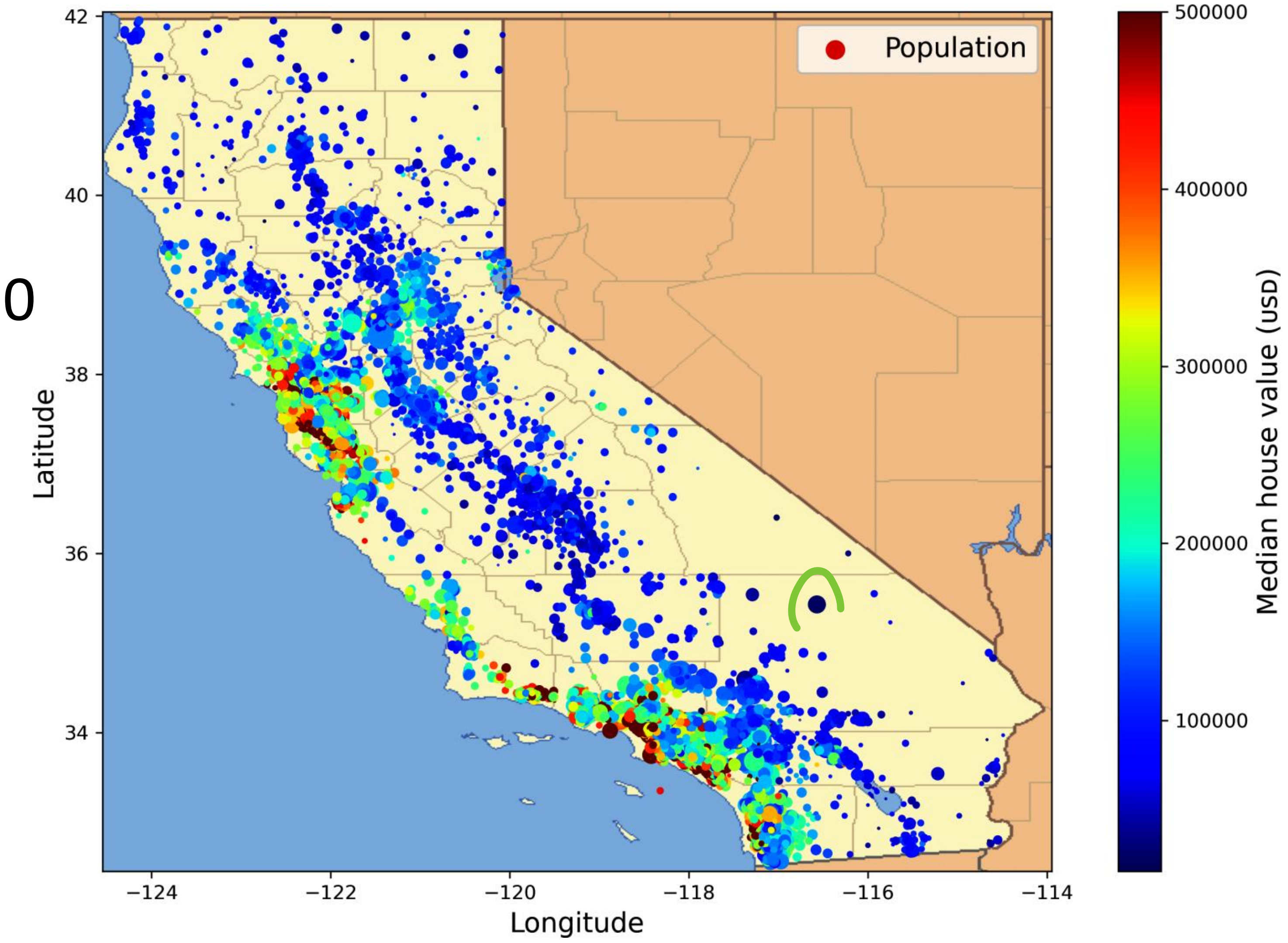
- Popular Data Sets:
- OpenML.org
- —Kaggle.com
- —PapersWithCode.com
- —UC Irvine Machine Learning Repository
- —Amazon's AWS datasets
- —TensorFlow datasets
- Meta portals (they list open data repositories):
- —DataPortals.org
- —OpenDataMonitor.eu
- Other pages listing many popular open data repositories: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

End to End ML Project - Major Steps

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.

StatLib Data

- California Housing prices dataset (StatLib repo)
- Data based on California 1990 census



Look at the Big Picture

- The task is to use California census data to build a model of housing prices in the state
- This data includes metrics such as the population, median income, and median housing price for each block group in California
- Your model should learn from this data and be able to predict the median housing price in any district, given all the other metrics
- FRAME THE PROBLEM:

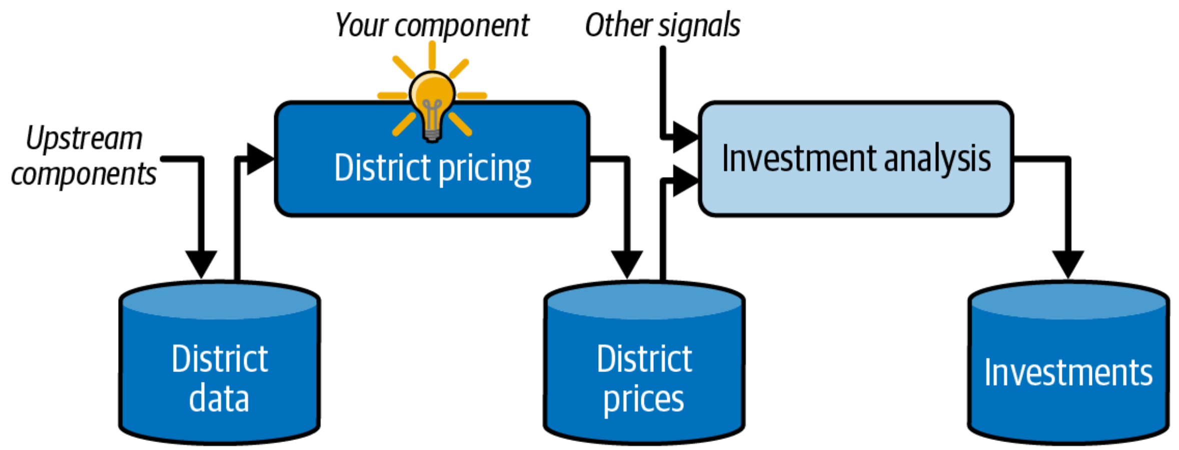
- Is building a model the end goal ?

- Will be fed into another system to analyse investment

- Current solution ? A: It is manual

It is costly and time consuming

- Data Pipeline
- Check the Assumptions



Performance Measure

- Root Mean Square Error:

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

- m is the number of instances in the dataset you are measuring the RMSE on.
  - — For example, if you are evaluating the RMSE on a validation set of 2,000 districts, then $m = 2{,}000$.
- $\mathbf{x}^{(i)}$ is a vector of all the feature values (excluding the label) of the i^{th} instance in the dataset, and $y^{(i)}$ is its label (the desired output value for that instance).
  - — For example, if the first district in the dataset is located at longitude –118.29°, latitude 33.91°, and it has 1,416 inhabitants with a median income of \$38,372, and the median house value is \$156,400 (ignoring other features for now), then:

$$\mathbf{x}^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1{,}416 \\ 38{,}372 \end{pmatrix}$$

and:

$$y^{(1)} = 156{,}400$$

Performance Measure

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

$$\hat{y} = h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

- h is your system's prediction function, also called a *hypothesis*. When your system is given an instance's feature vector $\mathbf{x}^{(i)}$, it outputs a predicted value $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$ for that instance ($\hat{y}$ is pronounced "y-hat").
  - — For example, if your system predicts that the median housing price in the first district is \$158,400, then $\hat{y}^{(1)} = h(\mathbf{x}^{(1)}) = 158{,}400$. The prediction error for this district is $\hat{y}^{(1)} - y^{(1)} = 2{,}000$.
- RMSE($\mathbf{X}$,h) is the cost function measured on the set of examples using your hypothesis h.

Performance Measure

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

- Mean Absolute Error:

- Both the RMSE and the MAE are ways to measure the distance between two vectors: the vector of predictions and the vector of target values. Various distance measures, or *norms*, are possible:
- Computing the root of a sum of squares (RMSE) corresponds to the *Euclidean norm*: this is the notion of distance we are all familiar with. It is also called the ℓ_2 *norm*, noted $\| \cdot \|_2$ (or just $\| \cdot \|$).
- Computing the sum of absolutes (MAE) corresponds to the ℓ_1 *norm*, noted $\| \cdot \|_1$. This is sometimes called the *Manhattan norm* because it measures the distance between two points in a city if you can only travel along orthogonal city blocks – e.g. Grid like formation

- • More generally, the ℓ_k *norm* of a vector **v** containing *n* elements is defined as $\|\mathbf{v}\|_k = (|v_1|^k + |v_2|^k + \ldots + |v_n|^k)^{1/k}$.
- The higher the norm index, the more it focuses on large values and neglects small ones. This is why the RMSE is more sensitive to outliers than the MAE. But when outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.

Get the Data

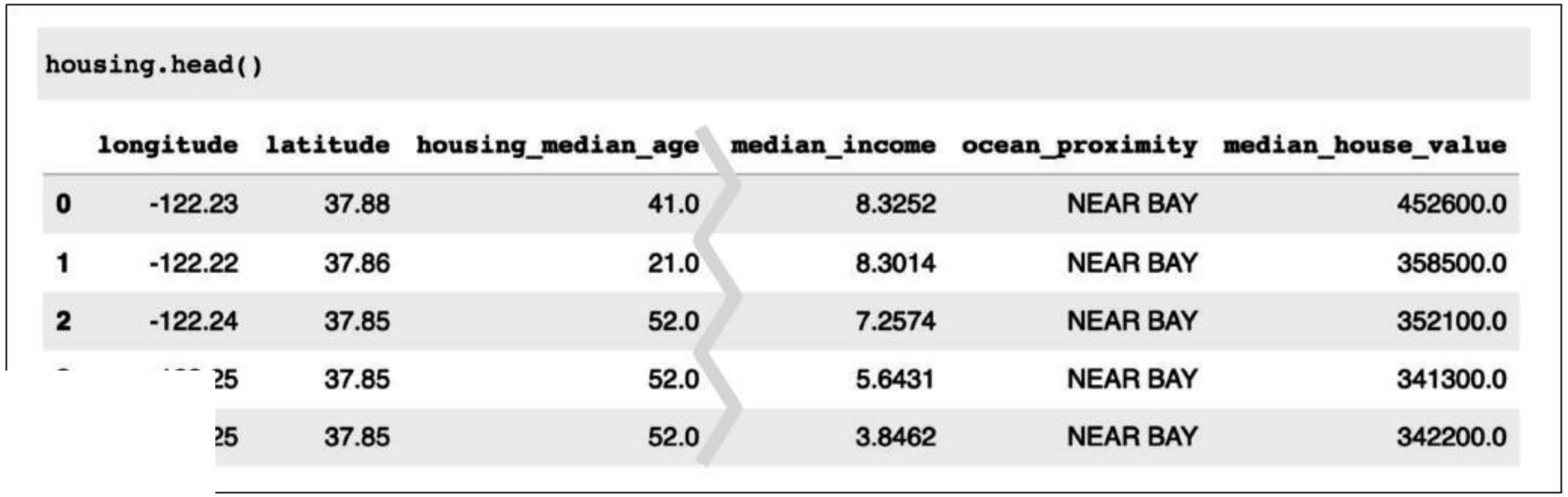
- Running via Google Colab
- Data and notebooks: https://homl.info/colab3 \
- There will be a tutorial after this lecture by TAs on how to use Google Colab / Jupyter Notebook / some python Libraries

```python
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request
```

- Loading the data

```python
def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

Explore the Data



```
housing.head()
```

	longitude	latitude	housing_median_age	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41.0	8.3252	NEAR BAY	452600.0
1	-122.22	37.86	21.0	8.3014	NEAR BAY	358500.0
2	-122.24	37.85	52.0	7.2574	NEAR BAY	352100.0
	25	37.85	52.0	5.6431	NEAR BAY	341300.0
	25	37.85	52.0	3.8462	NEAR BAY	342200.0

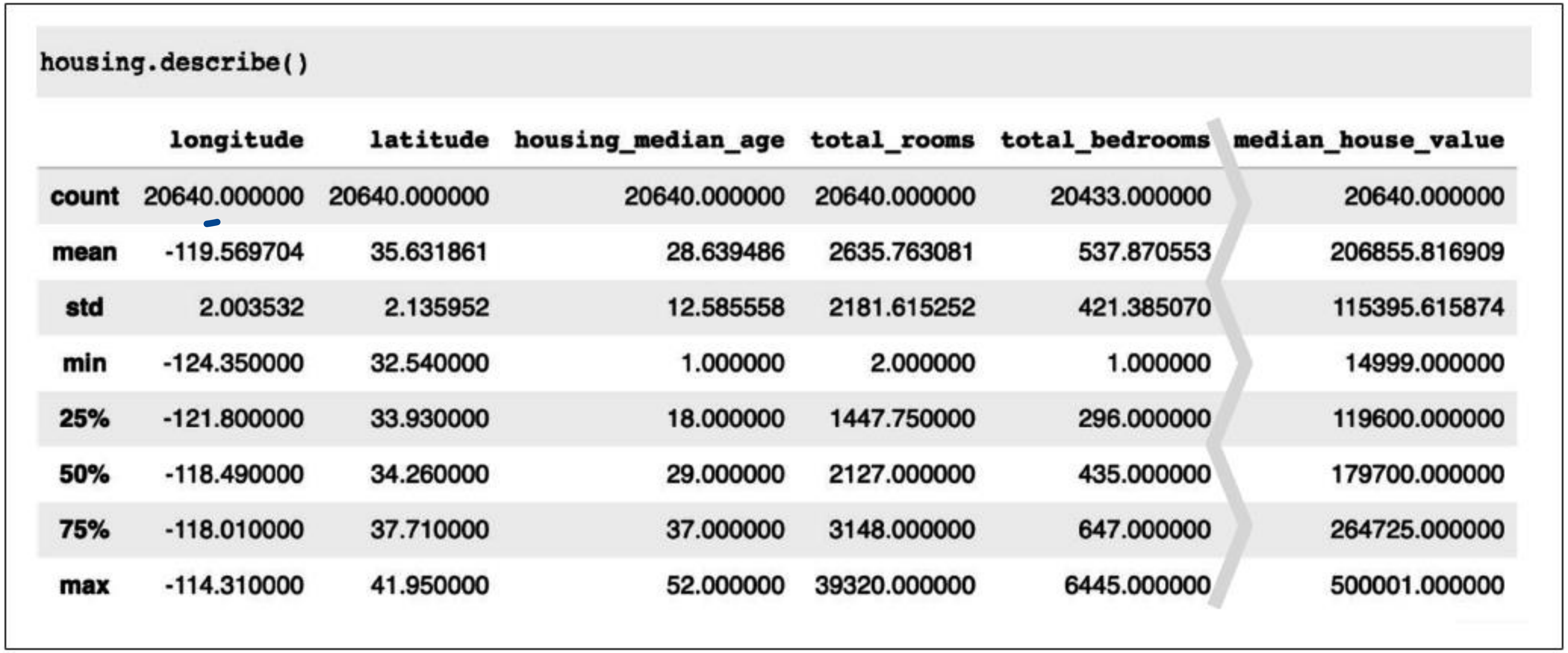
```
>>> housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

- 20,640 data instances
- 20,433 non-null values – 207 districts are values.

```
>>> housing["ocean_proximity"].value_counts()
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

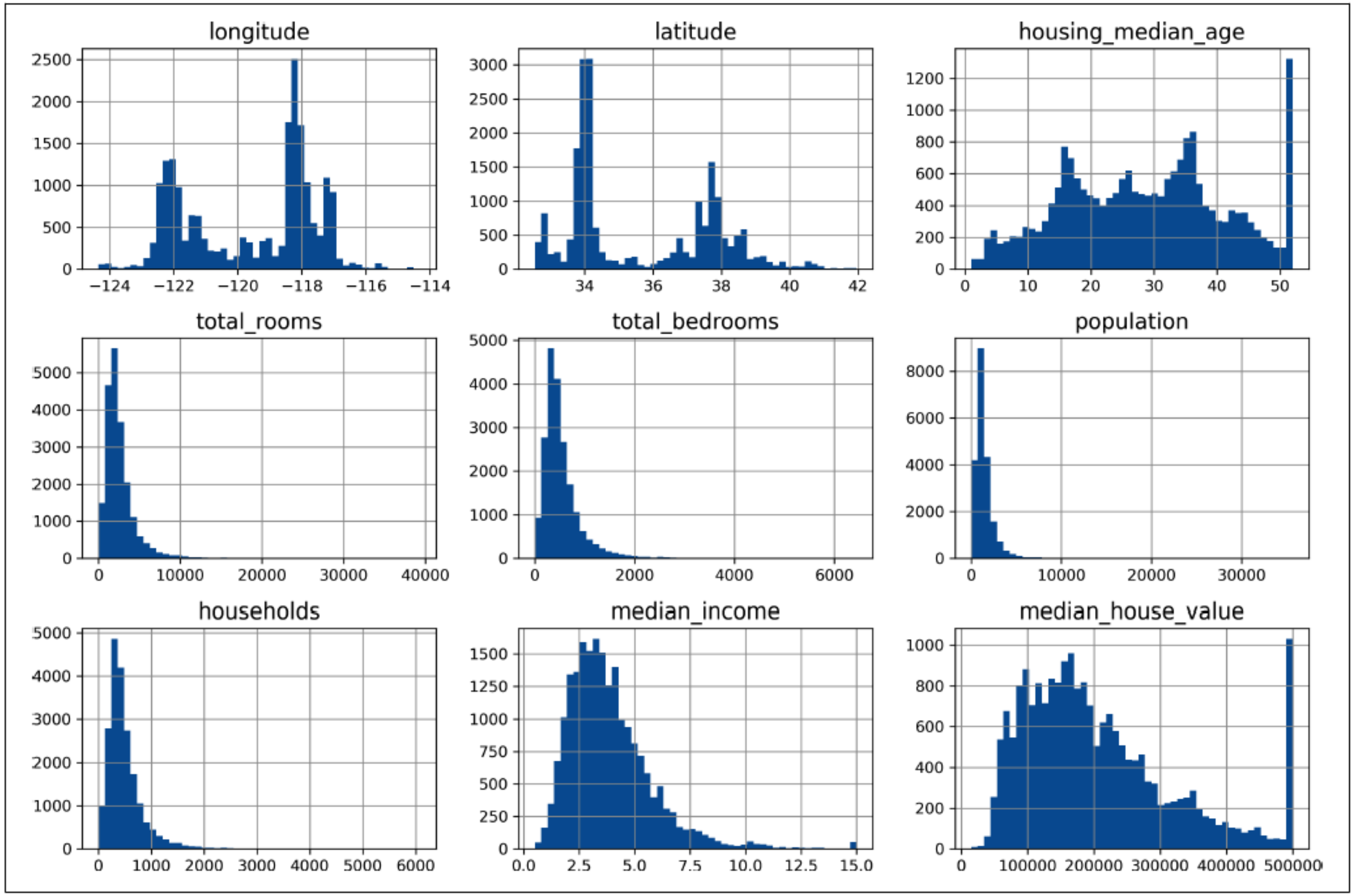
Explore the Data



```
housing.describe()
```

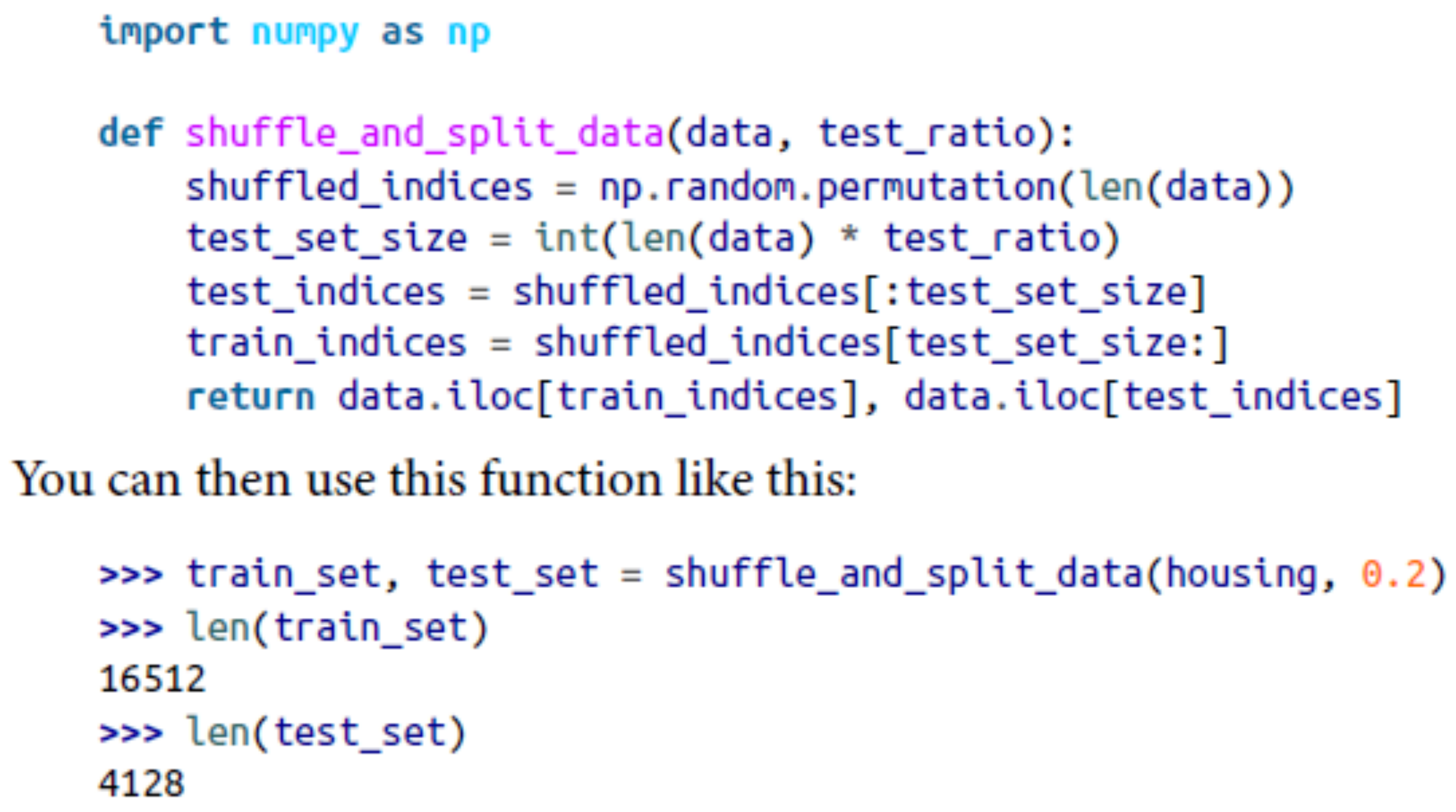
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	500001.000000

```
import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(12, 8))
plt.show()
```



Create a Test Set

- Roughly 20% split (or less if data is larger)



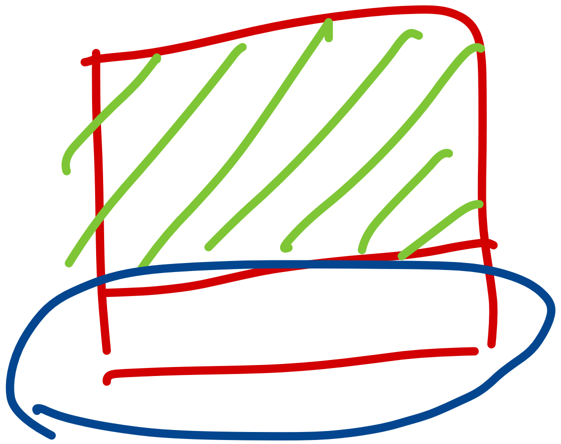
```
import numpy as np

def shuffle_and_split_data(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

You can then use this function like this:

```
>>> train_set, test_set = shuffle_and_split_data(housing, 0.2)
>>> len(train_set)
16512
>>> len(test_set)
4128
```

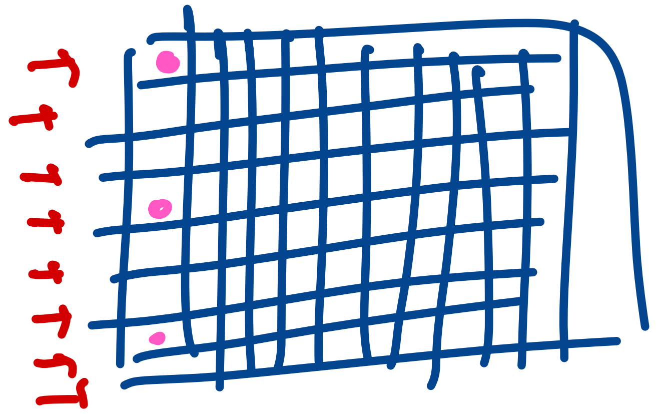
Test Data

- How can you ensure, across various runs of the code, that your model does not see the test data ?



untouched.

Unique Identifier



```python
from zlib import crc32

def is_id_in_test_set(identifier, test_ratio):
    return crc32(np.int64(identifier)) < test_ratio * 2**32

def split_data_with_id_hash(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: is_id_in_test_set(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

- Have a look at train_test_split() in Scikit-Learn

Tagging

```python
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

10000

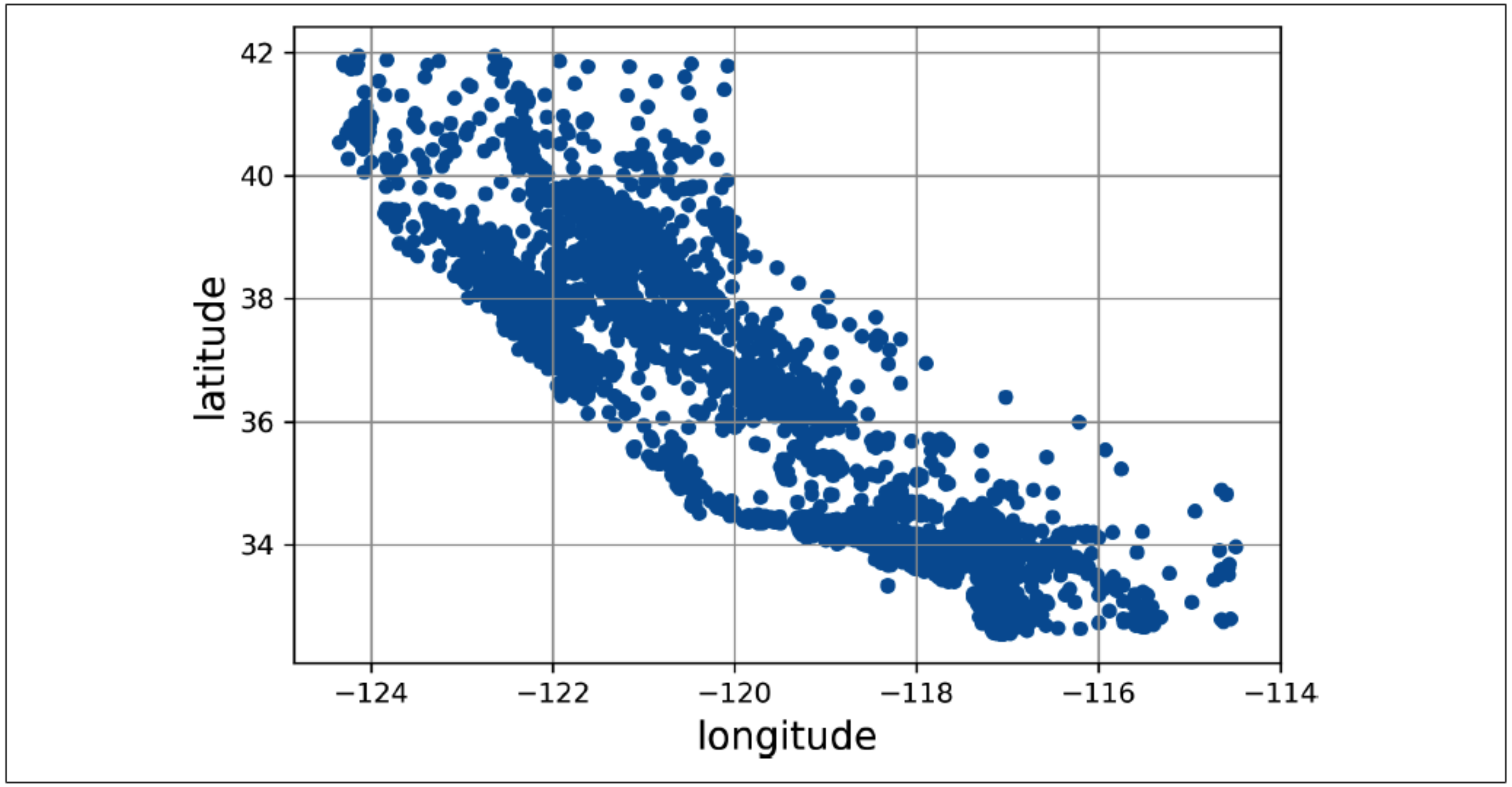
110

1000

Seeding.

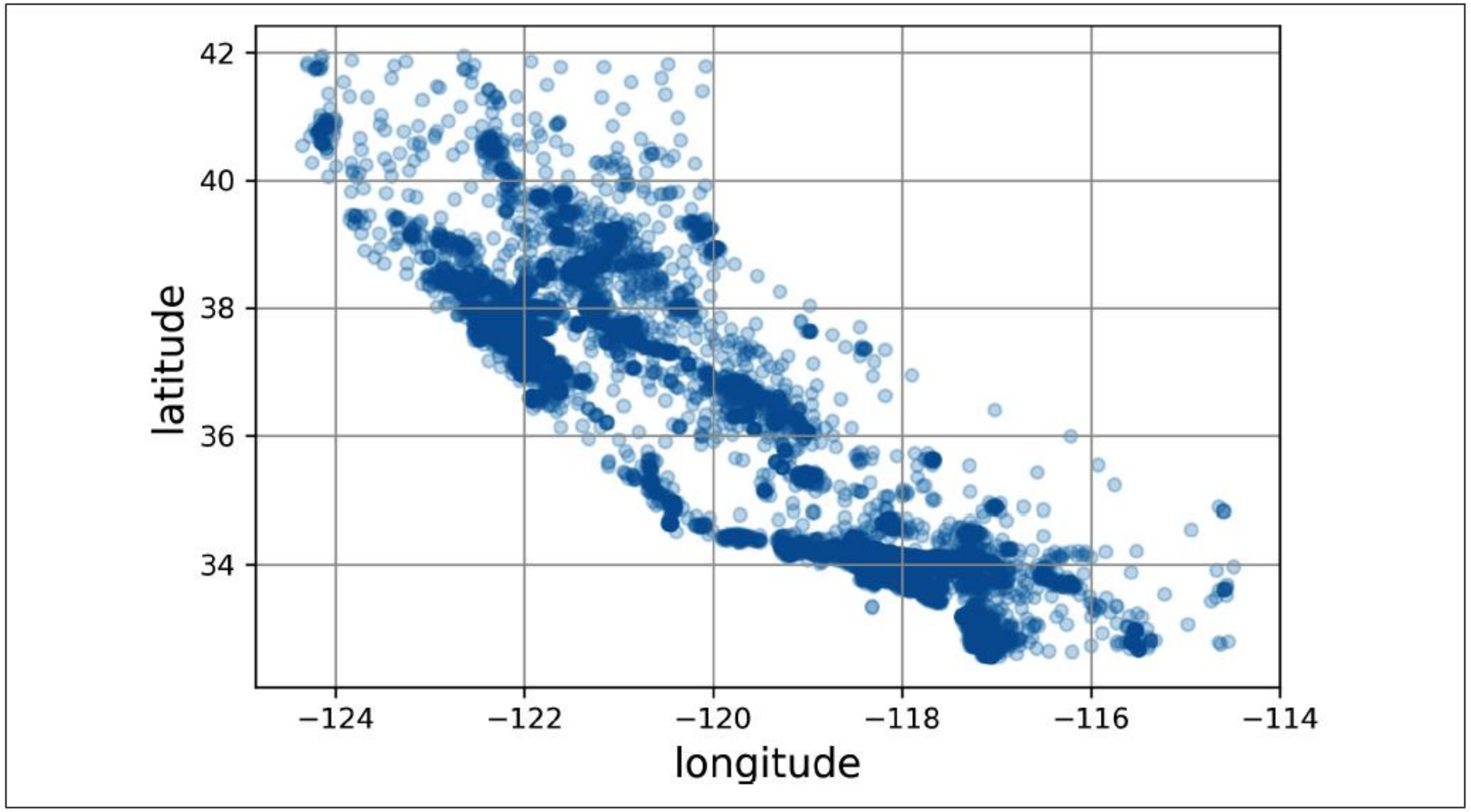
Visualise the Data

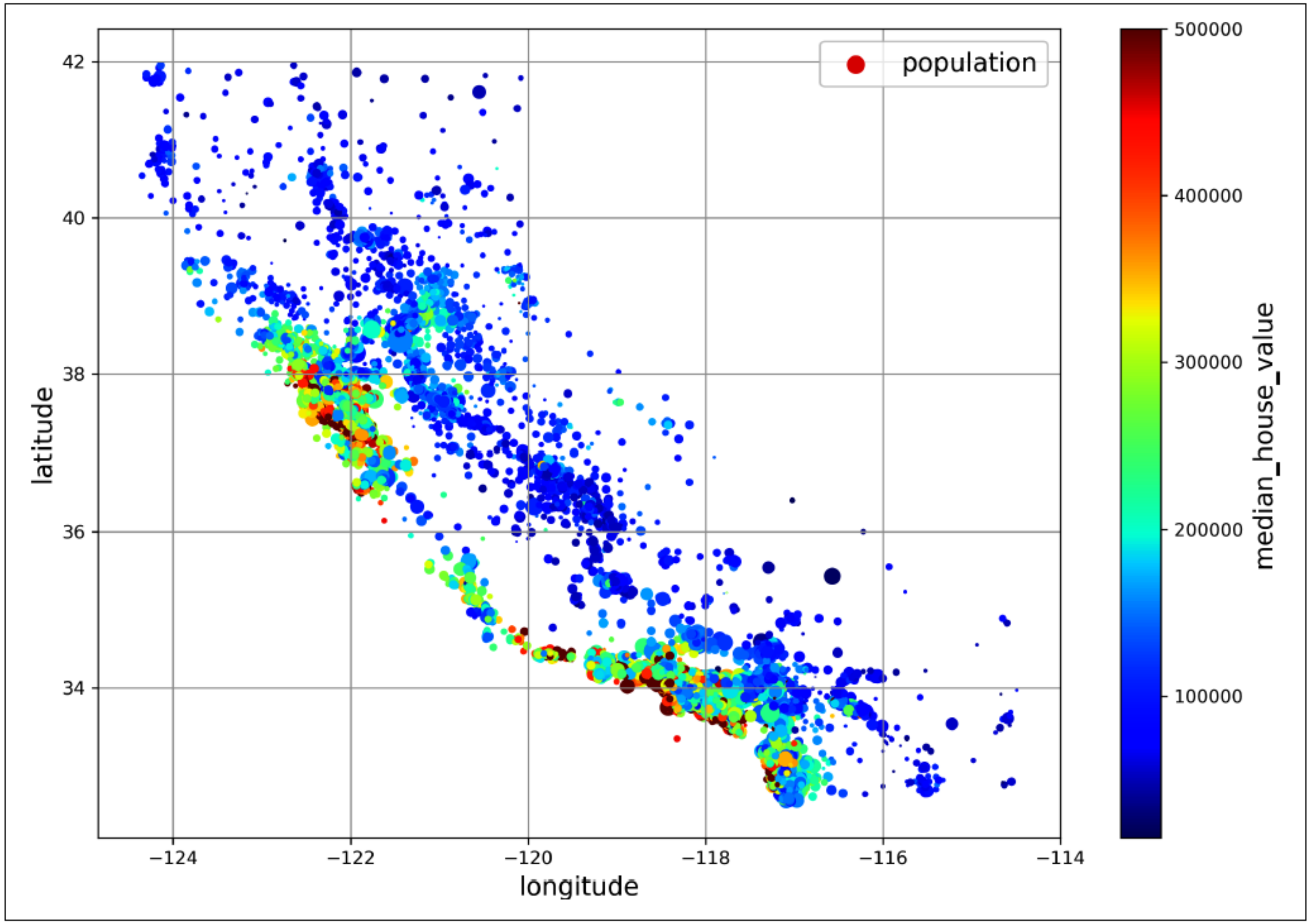
```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
plt.show()
```



Visualise the Data

```
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()
```




population
latitude
longitude
median_house_value
42
40
38
36
34
−124
−122
−120
−118
−116
−114
500000
400000
300000
200000
100000

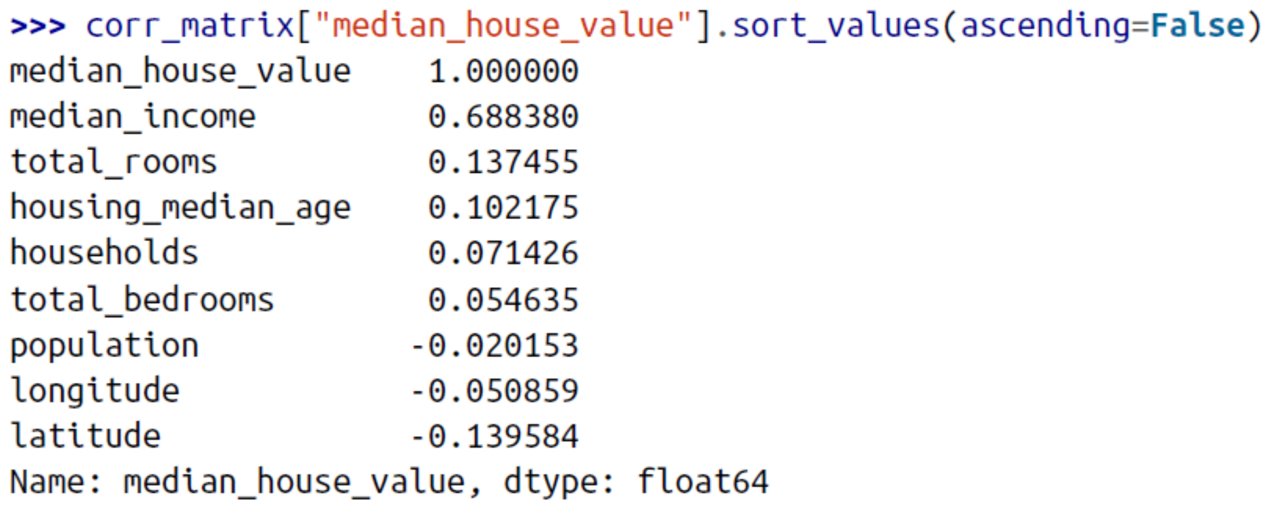
Look for Correlations

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where

- n is sample size
- x_i, y_i are the individual sample points indexed with *i*
- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$.

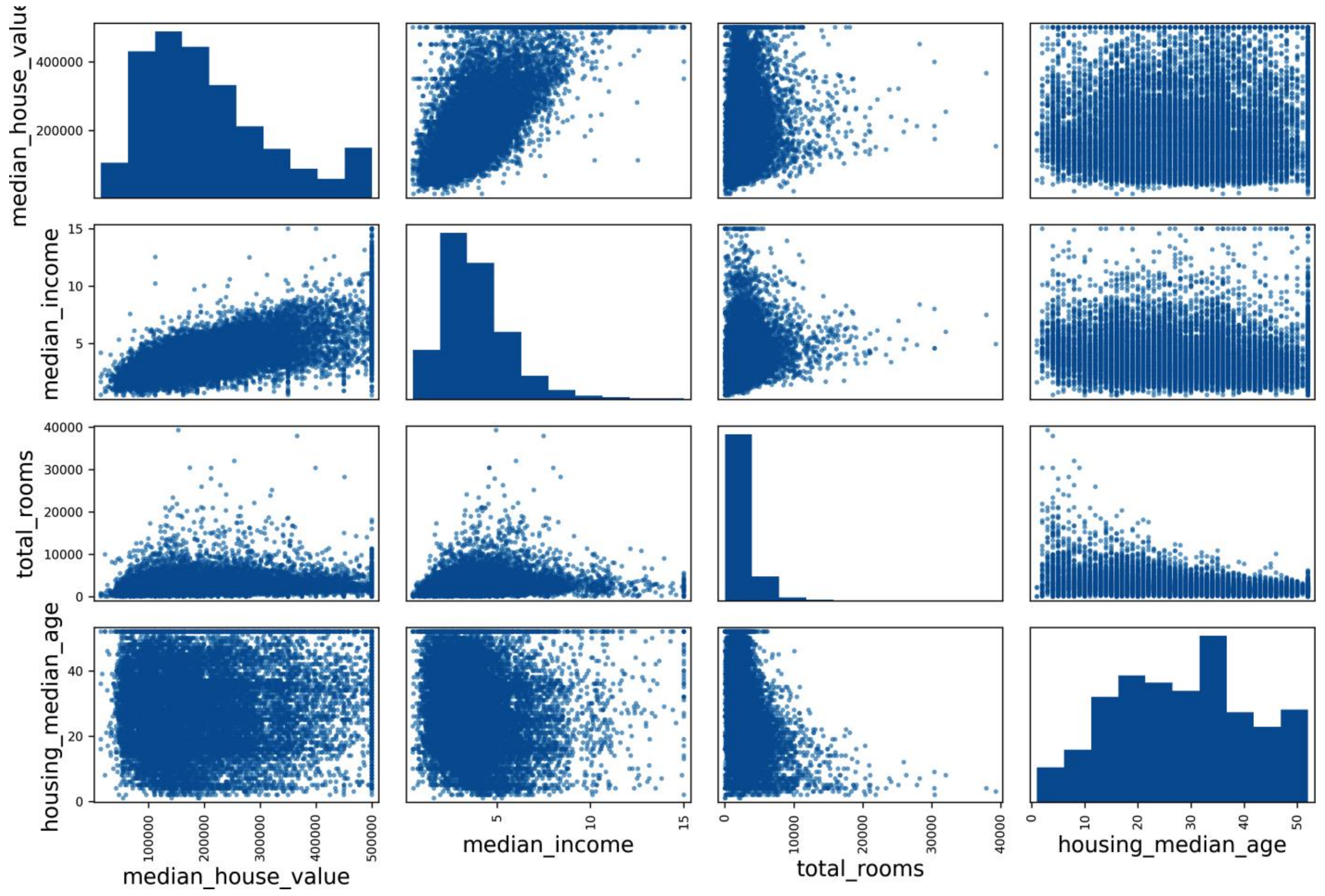
- Standard correlation coefficient (Pearson's *r)*
- The correlation coefficient ranges from –1 to 1.
- When it is close to 1, it means that there is a strong positive correlation; for example, the median house value tends to go up when the median income goes up.
- When the coefficient is close to –1, it means that there is a strong negative correlation; you can see a small negative correlation between the latitude and the median house value



```
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
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
Name: median_house_value, dtype: float64
```

Correlations

```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```

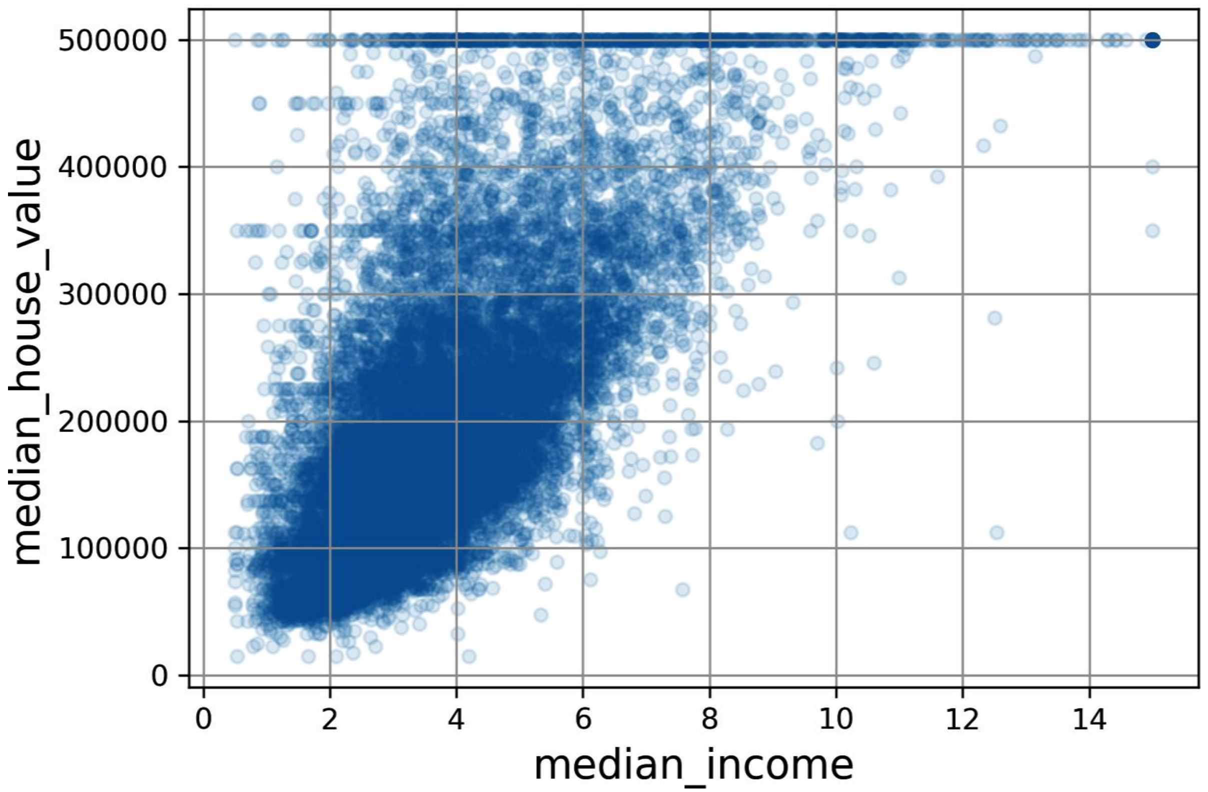


- This scatter matrix plots every numerical attribute against every other numerical attribute, plus a histogram of each numerical attribute’s values on the main diagonal (top left to bottom right)
- The main diagonal would be full of straight lines if Pandas plotted each variable against itself, which would not be very useful. So instead, the Pandas displays a histogram of each attribute (other options are available; see the Pandas documentation formore details).

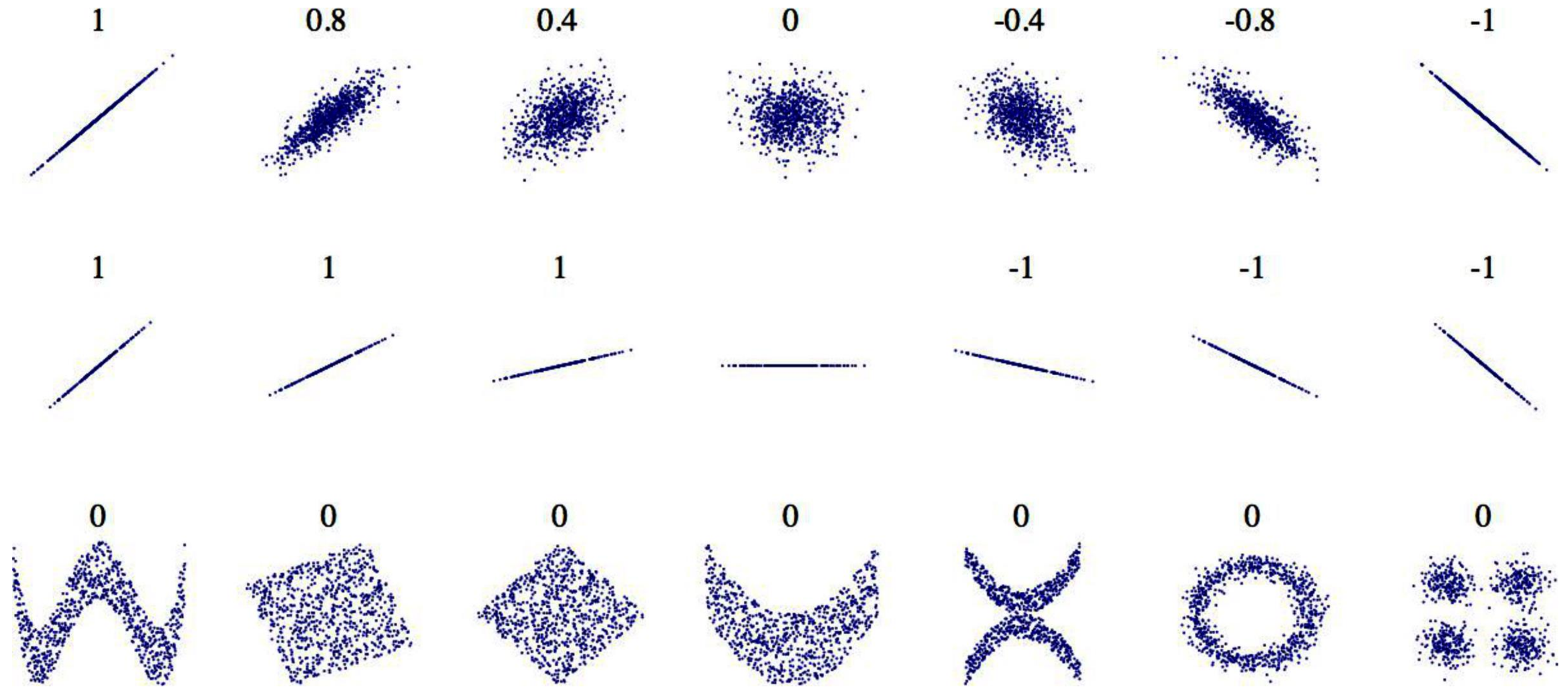
Correlations

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1, grid=True)
plt.show()
```

- Median income seems to be promising correlation
- The correlation is indeed quite strong; you can clearly see the upward trend, and the points are not too dispersed
- Second, the price cap you noticed earlier is clearly visible as a horizontal line at $500,000.
- Less obvious straight lines: a horizontal line around $450,000, another around $350,000, perhaps one around $280,000, and



Standard Correlation Coefficients for Various Datasets



More Correlations …

```
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"]
```

```
>>> 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
```

Data Preparation

- Structured data in machine learning consists of rows and columns.
- Data preparation is a required step in each machine learning project.
- The routineness of machine learning algorithms means the majority of effort on each project is spent on data preparation.
- “Data quality is one of the most important problems in data management, since dirty data often leads to inaccurate data analytics results and incorrect business decisions.”
- “it has been stated that up to 80% of data analysis is spent on the process of cleaning and preparing data. However, being a prerequisite to the rest of the data analysis workflow (visualization, modeling, reporting), it's essential that you become fluent and efficient in data wrangling techniques.”
- Step 1: clean the data: (e.g. total bedroom field – as attributes are missing)
- 1. Get rid of the corresponding districts.
- 2. Get rid of the whole attribute.
- 3. Set the missing values to some value (zero, the mean, the median, etc.). This is called *imputation*.
- Text and Categorial attributes / imputation.
- Homework: what are some good imputation approaches

Data Cleaning

- Using statistics to detect noisy data and identify outliers
- Identifying columns that have the same value or no variance and removing them
- Identifying duplicate rows of data and removing them.
- Marking empty values as missing
- Imputing missing values using statistics or a learned model

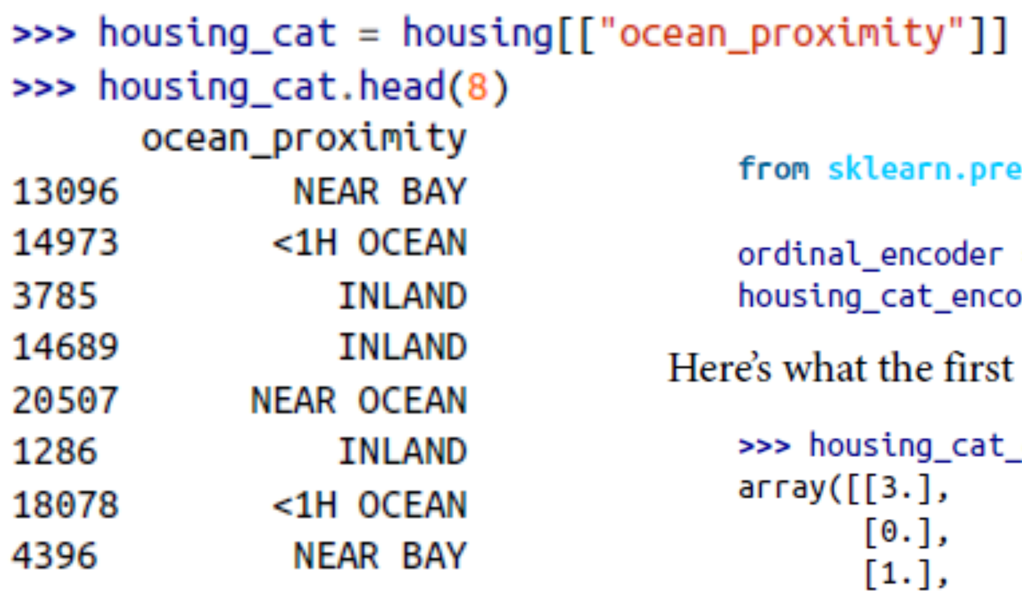
total_bedrooms attribute

1. Get rid of the corresponding districts.
2. Get rid of the whole attribute.
3. Set the missing values to some value (zero, the mean, the median, etc.). This is called *imputation*.

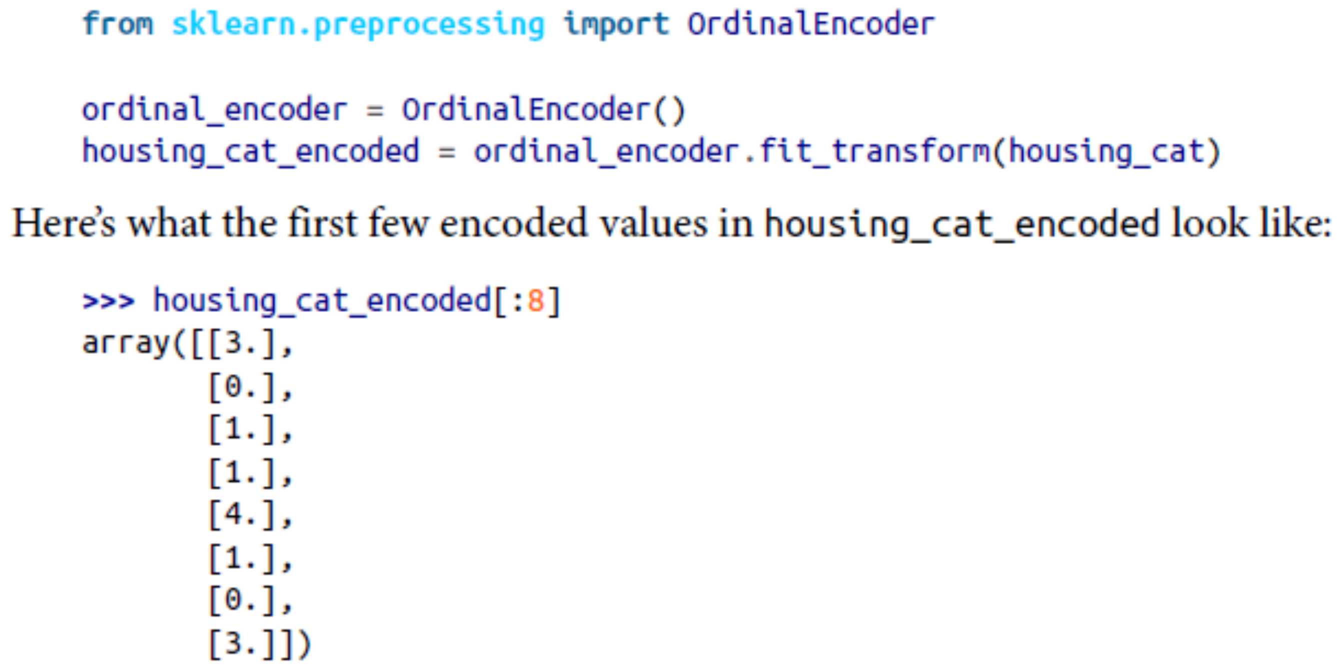
```
housing.dropna(subset=["total_bedrooms"], inplace=True)  # option 1

housing.drop("total_bedrooms", axis=1)  # option 2

median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

Converting Text to Numbers



```
>>> housing_cat = housing[["ocean_proximity"]]
>>> housing_cat.head(8)
      ocean_proximity
13096        NEAR BAY
14973       <1H OCEAN
3785           INLAND
14689          INLAND
20507      NEAR OCEAN
1286           INLAND
18078       <1H OCEAN
4396         NEAR BAY
```



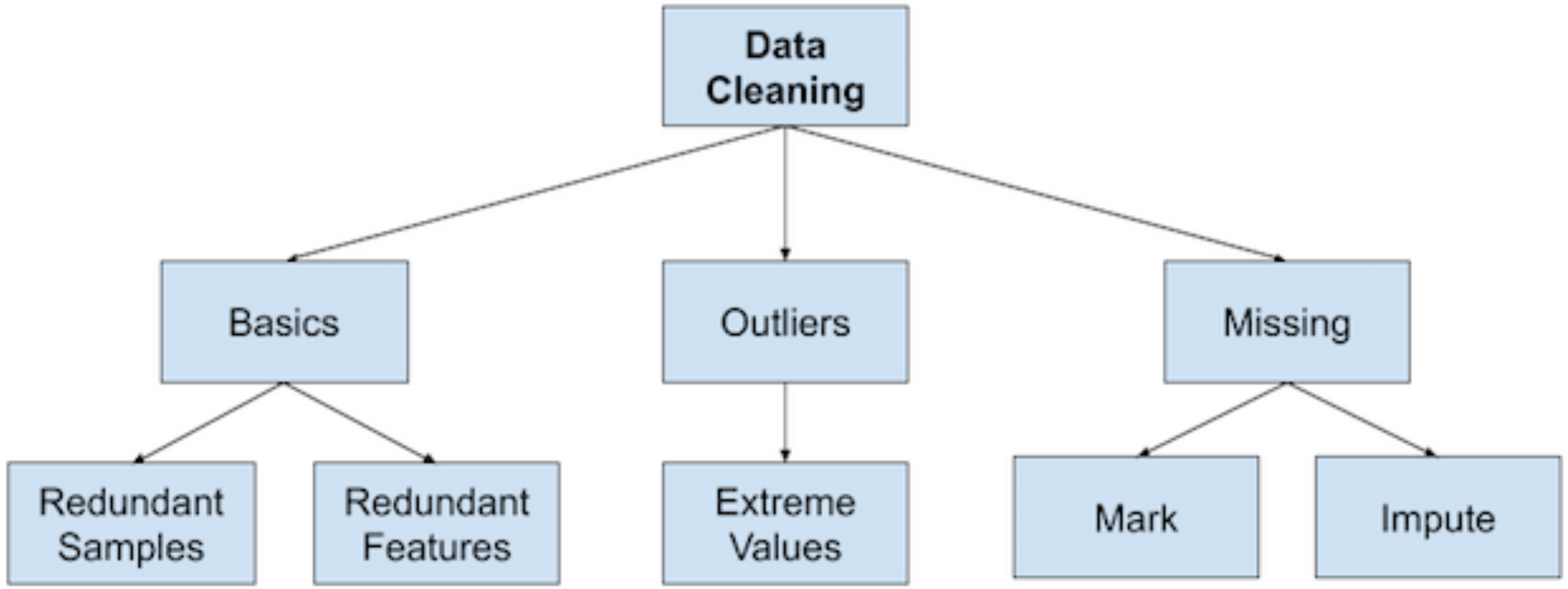
```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
```

Here's what the first few encoded values in housing_cat_encoded look like:

```
>>> housing_cat_encoded[:8]
array([[3.],
       [0.],
       [1.],
       [1.],
       [4.],
       [1.],
       [0.],
       [3.]])
```

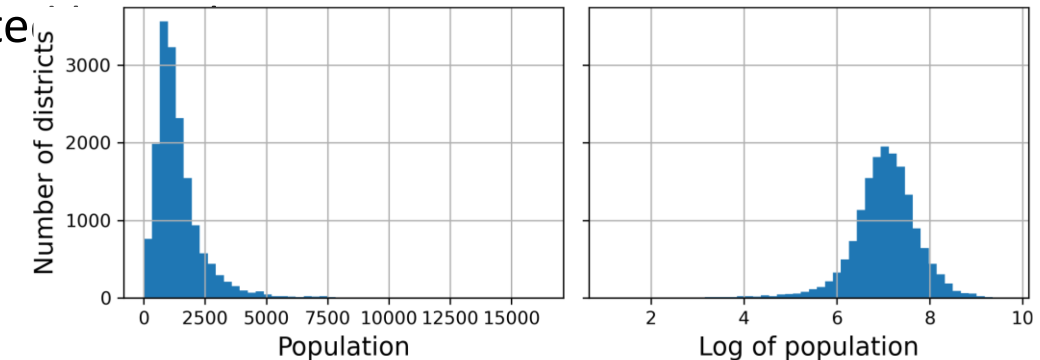
Data Cleaning

Overview of Data Cleaning



Feature Scaling and Transformation

- ML algorithms don't perform well when input numerical attributes have very different scales.
- This is the case for the housing data: the total number of rooms ranges from about 6 to 39,320, while the median incomes only range from 0 to 15.
- Min-max scaling normalization – each attribute is shifted and rescaled such that they end up between 0 -1 .
- Scikit-Learn provides a transformer called MinMaxScaler for this. It has a feature_range hyperparameter that lets you change the range if, for some reason, you don't want 0–1 (e.g., neural networks work best with zero-mean inputs, so a range of –1 to 1 is preferable) – affected by outliers.
- Standardization is different: first it subtracts the mean value (so standardized values have a zero mean), then it divides the result by the standard deviation (so standardized values have a standard deviation equal to 1). Less affecte

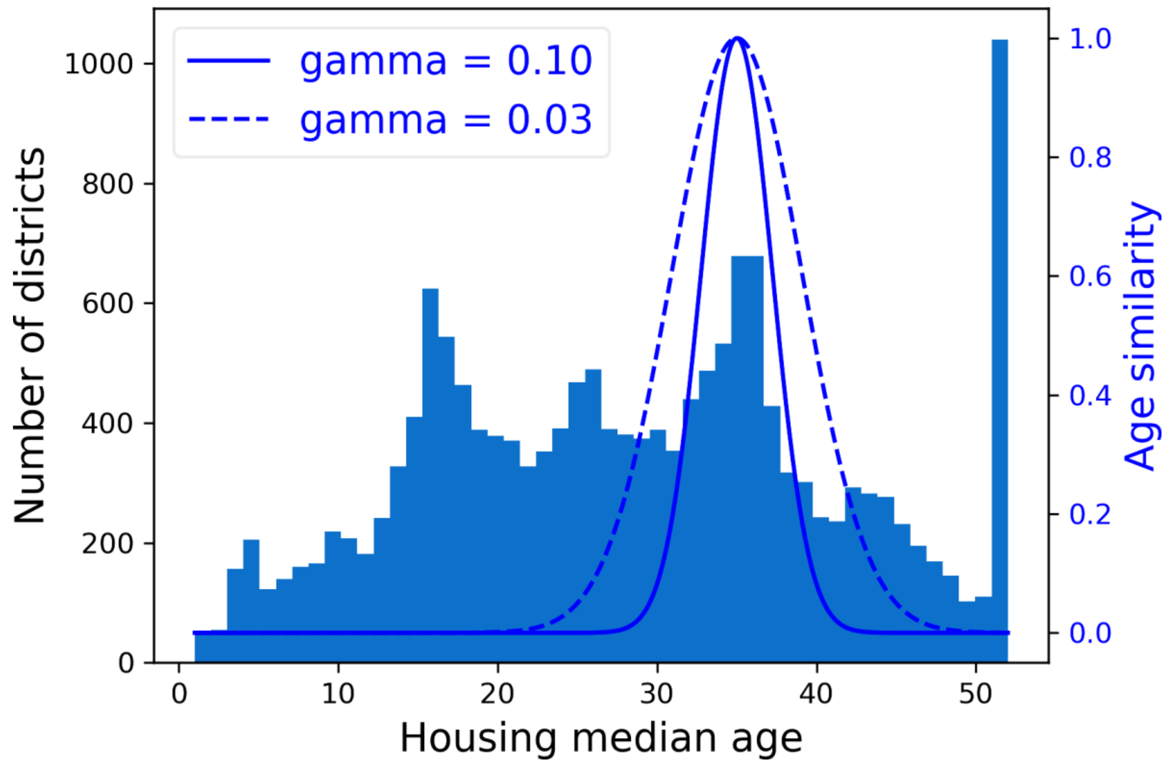


Multimodal Distribution

- Two or more clear peaks
- Either create buckets of the data
- Add a feature for the main modes
- Radial basis function – Gaussian RBF

$\exp(-\gamma(x - 35)^2)$

The hyperparameter γ (gamma) determines how quickly the similarity measure decays as x moves away from 35.



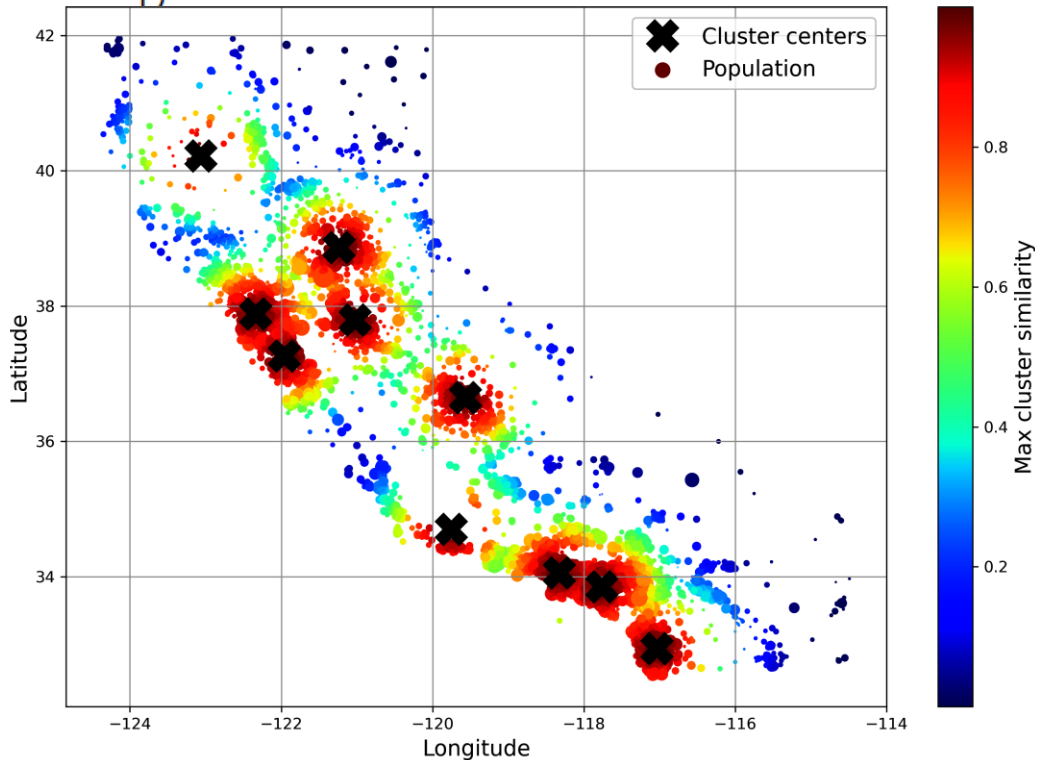
Transformation Pipeline

- Missing values in numerical features will be imputed by replacing them with the median, as most ML algorithms don't expect missing values. In categorical features, missing values will be replaced by the most frequent category.
- The categorical feature will be one-hot encoded, as most ML algorithms only accept numerical inputs.
- A few ratio features will be computed and added: `bedrooms_ratio`, `rooms_per_house`, and `people_per_house`. Hopefully these will better correlate with the median house value, and thereby help the ML models.
- A few cluster similarity features will also be added. These will likely be more useful to the model than latitude and longitude.
- Features with a long tail will be replaced by their logarithm, as most models prefer features with roughly uniform or Gaussian distributions.
- All numerical features will be standardized, as most ML algorithms prefer when all features have roughly the same scale.

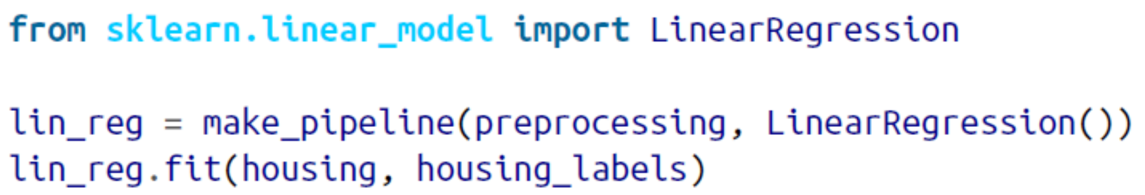
Transformation Pipeline

- First imputations
- Scaling of features
- Transform the data
- Ex. clusterSimilarity Transform

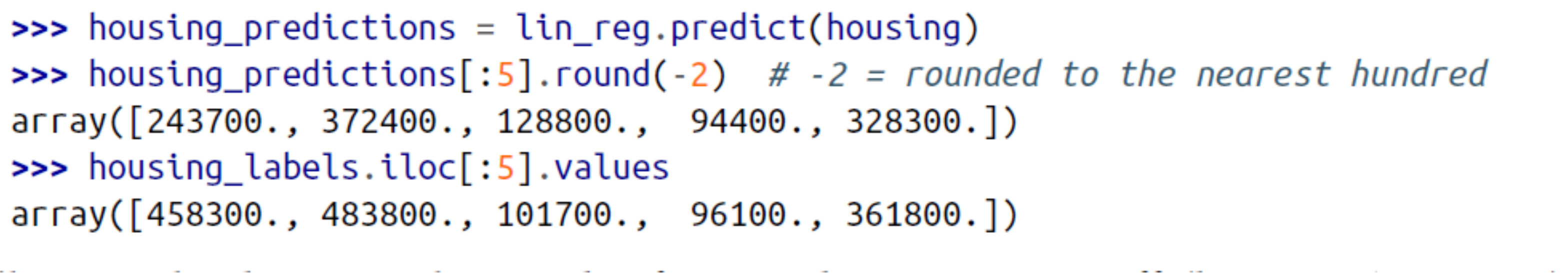
```
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline([
    ("impute", SimpleImputer(strategy="median")),
    ("standardize", StandardScaler()),
])
```



Model Selection & Training



```
from sklearn.linear_model import LinearRegression

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



```
>>> 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.])
```

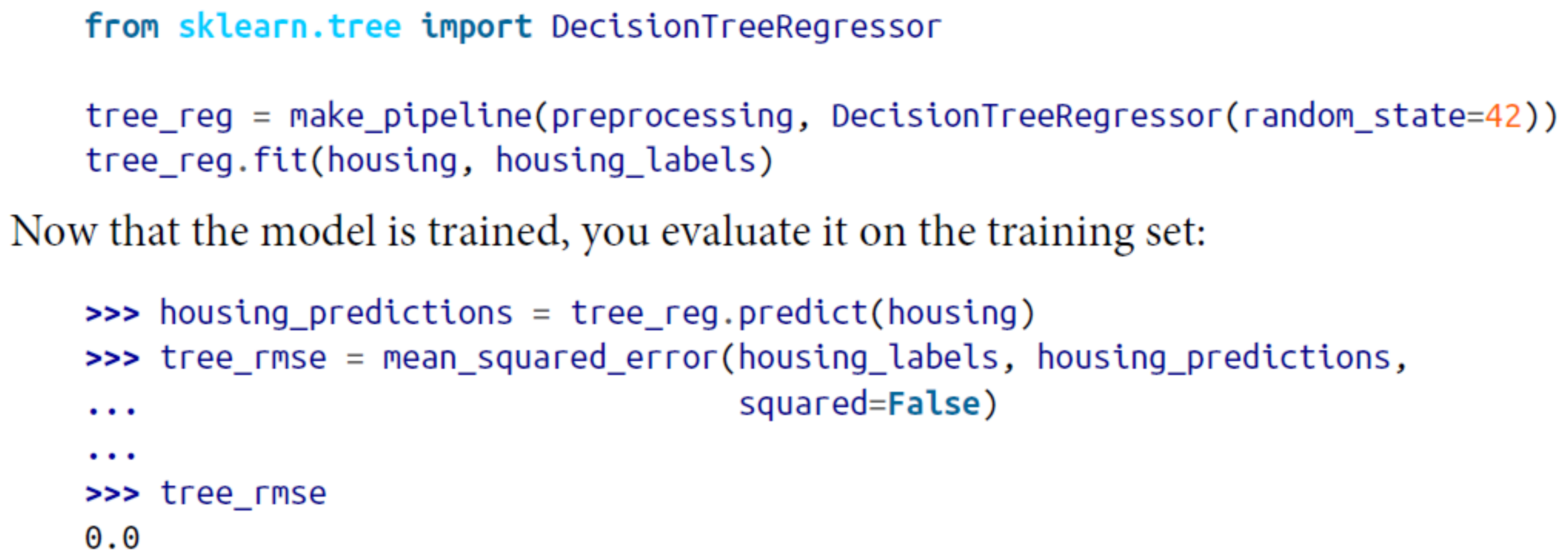
How good is this model ?

Model Selection & Training

```python
from sklearn.linear_model import LinearRegression

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

```python
>>> 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.])
```

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



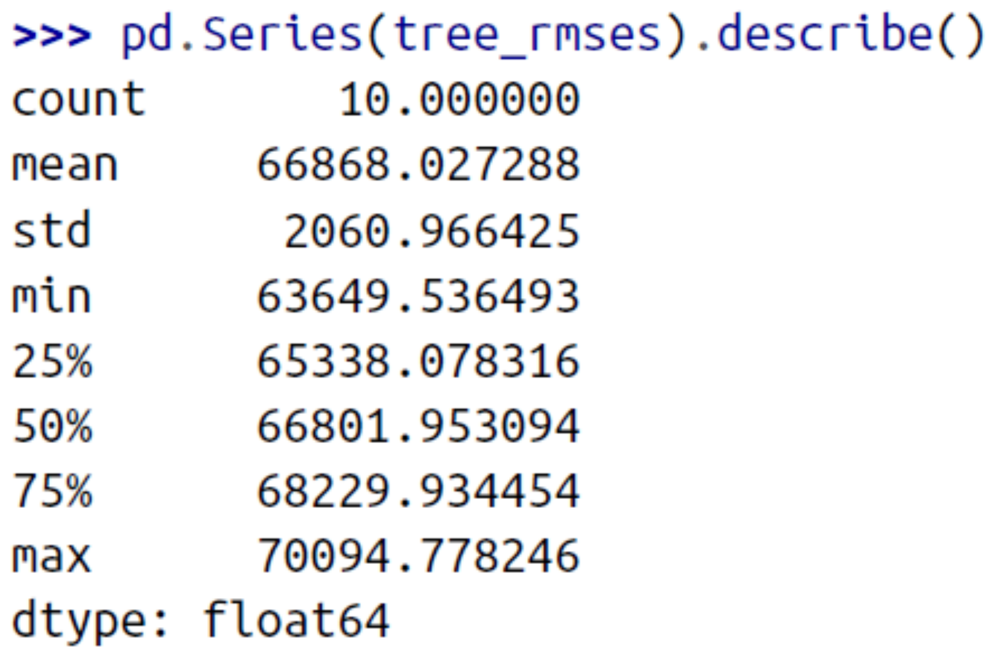
```python
from sklearn.tree import DecisionTreeRegressor

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
```

- How is this model ?

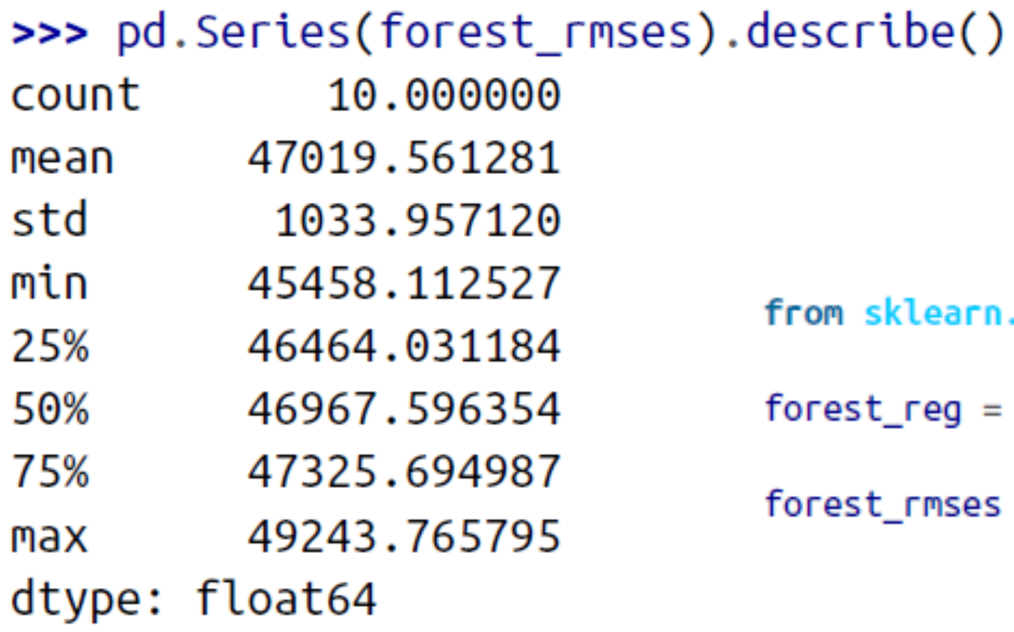
Cross-Validation



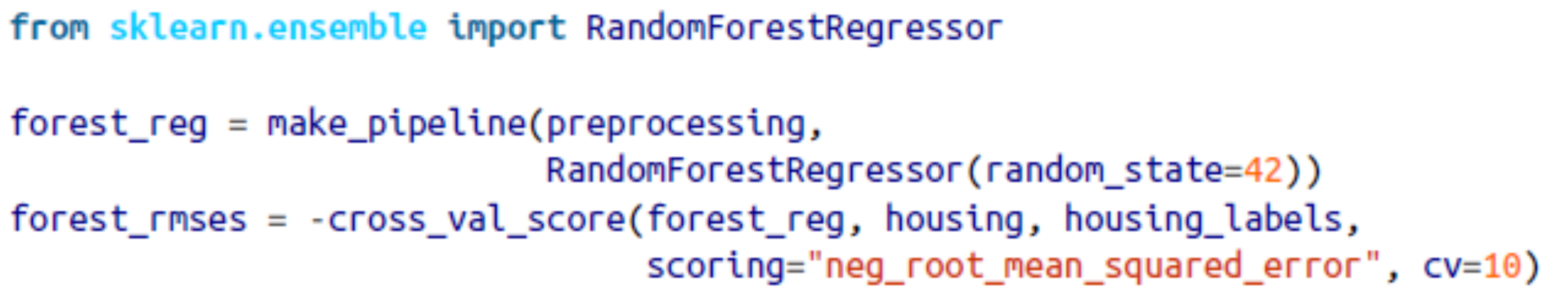
```
>>> pd.Series(tree_rmses).describe()
count       10.000000
mean     66868.027288
std       2060.966425
min      63649.536493
25%      65338.078316
50%      66801.953094
75%      68229.934454
max      70094.778246
dtype: float64
```

- The decision tree has an RMSE of about 66,868, with a standard deviation of about 2,061

- RandomForestRegressor – train on many decision trees.



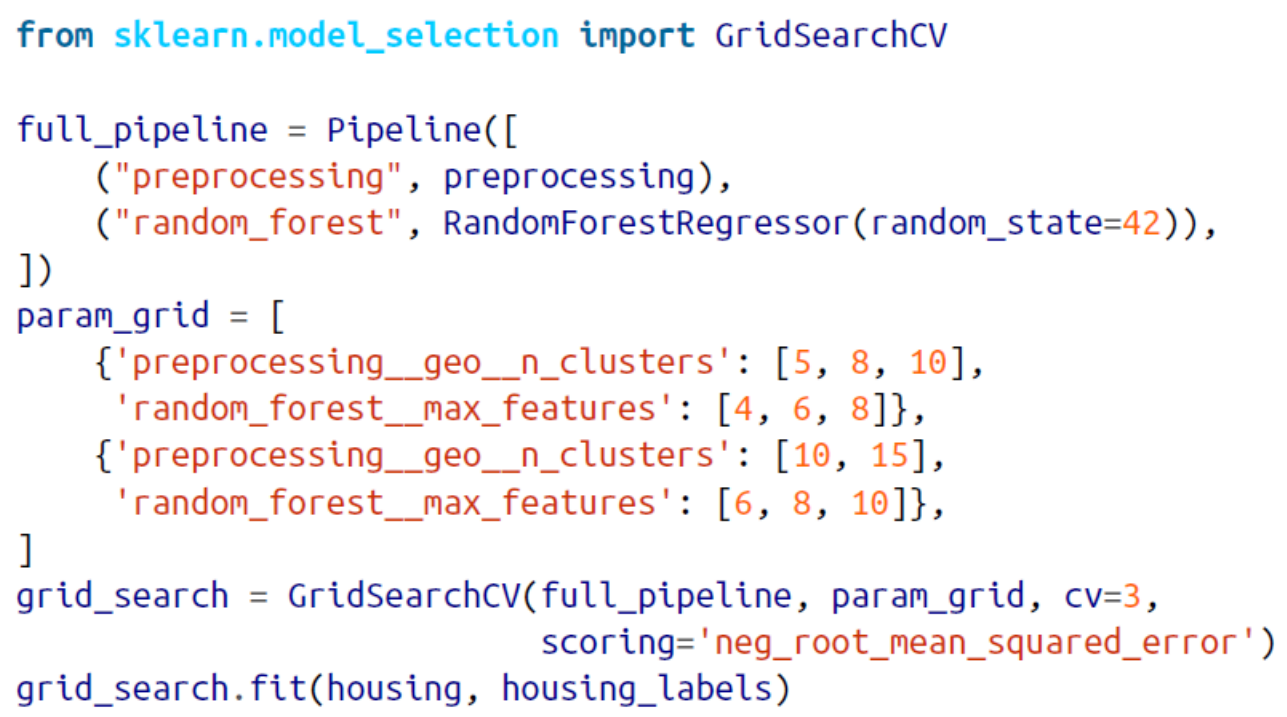
```
>>> pd.Series(forest_rmses).describe()
count       10.000000
mean     47019.561281
std       1033.957120
min      45458.112527
25%      46464.031184
50%      46967.596354
75%      47325.694987
max      49243.765795
dtype: float64
```



```python
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)
```

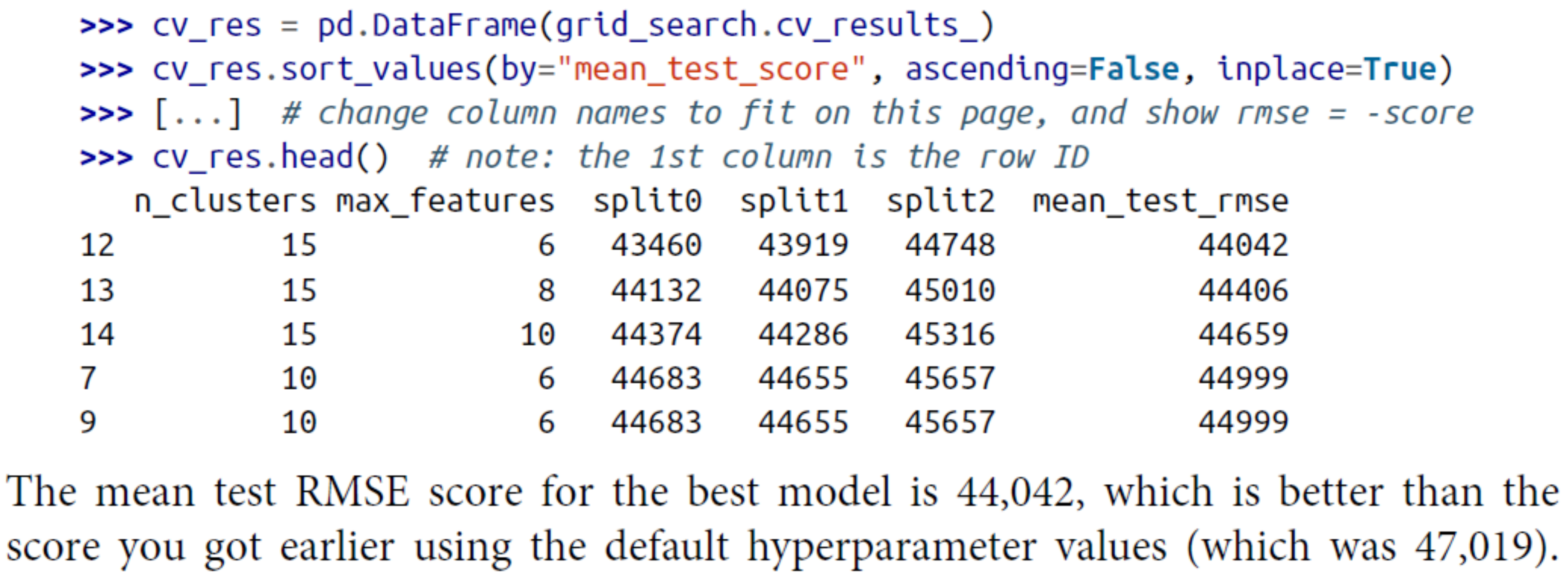
Hyperparameter Tuning

- Grid Search: GridSearchCV
- You need to do is tell it which hyperparameters you want it to experiment with and what values to try out, and it will use cross-validation to evaluate all the possible combinations



```python
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])
param_grid = [
    {'preprocessing__geo__n_clusters': [5, 8, 10],
     'random_forest__max_features': [4, 6, 8]},
    {'preprocessing__geo__n_clusters': [10, 15],
     'random_forest__max_features': [6, 8, 10]},
]
grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(housing, housing_labels)
```

Hyperparameter Tuning



```
>>> cv_res = pd.DataFrame(grid_search.cv_results_)
>>> cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
>>> [...]  # change column names to fit on this page, and show rmse = -score
>>> cv_res.head()  # note: the 1st column is the row ID
   n_clusters max_features  split0  split1  split2  mean_test_rmse
12         15            6   43460   43919   44748           44042
13         15            8   44132   44075   45010           44406
14         15           10   44374   44286   45316           44659
7          10            6   44683   44655   45657           44999
9          10            6   44683   44655   45657           44999
```

The mean test RMSE score for the best model is 44,042, which is better than the score you got earlier using the default hyperparameter values (which was 47,019).

Randomized Search

- GridSearch is ok for a few combinations
- RandomizedSearch is good for a large parameter space
- It evaluates a fixed number of combinations, selecting a random value for each hyperparameter at every iteration
- If some of your hyperparameters are continuous (or discrete but with many possible values), and you let randomized search run for, say, 1,000 iterations, then it will explore 1,000 different values for each of these hyperparameters, whereas grid search would only explore the few values you listed for each one.
- Suppose a hyperparameter does not actually make much difference, but you don’t know it yet. If it has 10 possible values and you add it to your grid search, then training will take 10 times longer. But if you add it to a random search, it will not make any difference.
- If there are 6 hyperparameters to explore, each with 10 possible values, then grid search offers no other choice than training the model a million times, whereas random search can always run for any number of iterations you choose.

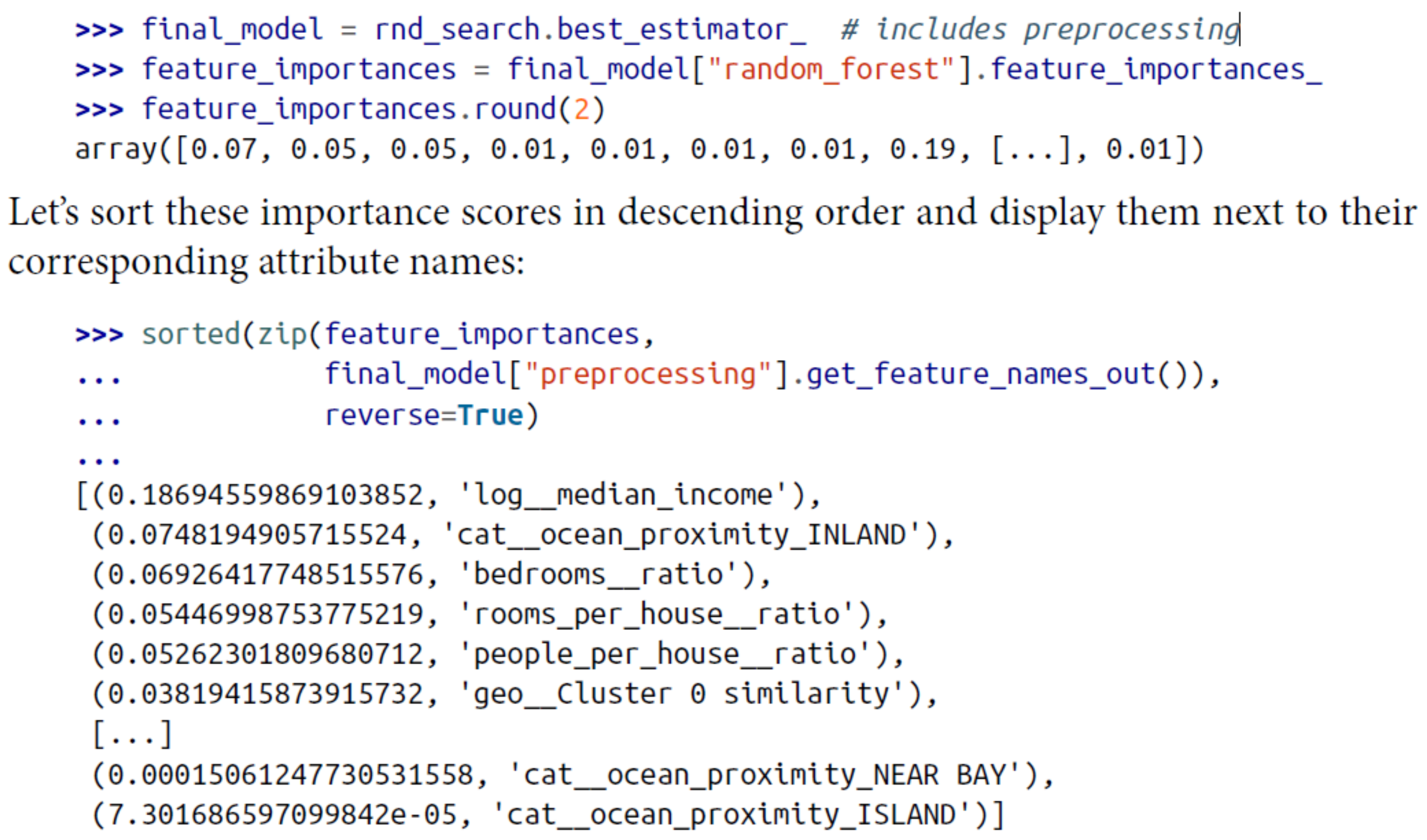
Randomized Search

For each hyperparameter, you must provide either a list of possible values, or a probability distribution:

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {'preprocessing__geo__n_clusters': randint(low=3, high=50),
                  'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
    full_pipeline, param_distributions=param_distribs, n_iter=10, cv=3,
    scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
```

Feature Importance

- Drop less important features

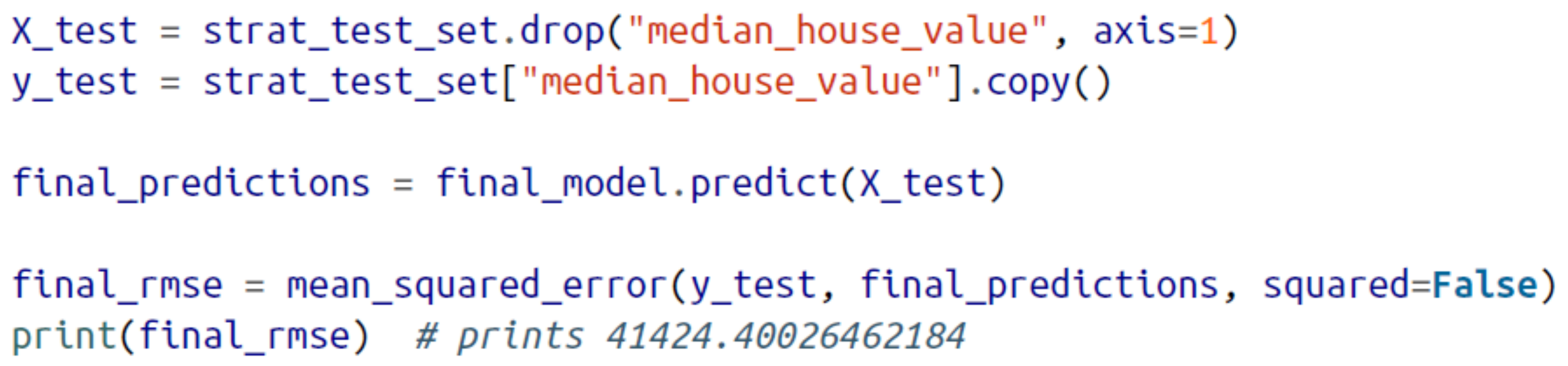


```
>>> final_model = rnd_search.best_estimator_  # includes preprocessing
>>> feature_importances = final_model["random_forest"].feature_importances_
>>> feature_importances.round(2)
array([0.07, 0.05, 0.05, 0.01, 0.01, 0.01, 0.01, 0.19, [...], 0.01])
```

Let's sort these importance scores in descending order and display them next to their corresponding attribute names:

```
>>> sorted(zip(feature_importances,
...            final_model["preprocessing"].get_feature_names_out()),
...            reverse=True)
...
[(0.18694559869103852, 'log__median_income'),
 (0.0748194905715524, 'cat__ocean_proximity_INLAND'),
 (0.06926417748515576, 'bedrooms__ratio'),
 (0.05446998753775219, 'rooms_per_house__ratio'),
 (0.05262301809680712, 'people_per_house__ratio'),
 (0.03819415873915732, 'geo__Cluster 0 similarity'),
 [...]
 (0.00015061247730531558, 'cat__ocean_proximity_NEAR BAY'),
 (7.301686597099842e-05, 'cat__ocean_proximity_ISLAND')]
```

Evaluate on Test Set

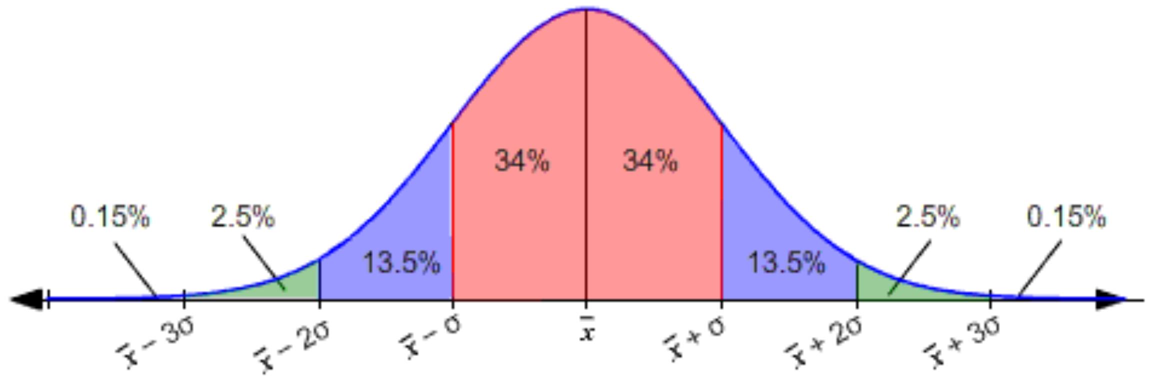


```python
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

final_predictions = final_model.predict(X_test)

final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
print(final_rmse)  # prints 41424.40026462184
```

In this California housing example, the final performance of the system is not much better than the experts’ price estimates, which were often off by 30%, but it may still be a good idea to launch it, especially if this frees up some time for the experts so they can work on more interesting and productive tasks.

Additional Data Pre-processing Steps

- Can consider removing variables with zero variance (each row for that column has the same value)
- Can remove columns of data that have low variance
- Identify rows that contain duplicate data
- Outlier identification and removal:
- Is the data outside 3 standard deviations ?
- Inter-quartile range methods
- Automatic outlier removal (LocalOutlierFactor Class)
- Marking and removing missing data
- Statistical Imputation
- KNN Imputation
- Iterative imputation


$\bar{x}-3\sigma$
$\bar{x}-2\sigma$
$\bar{x}-\sigma$
$\bar{x}$
$\bar{x}+\sigma$
$\bar{x}+2\sigma$
$\bar{x}+3\sigma$