Artificial Intelligence and Machine Learning for Connected Systems

Class 3 & 4 - End to end project

Naresh Modina

Last class

- Refresh on basic concepts
  - Statistic and probability
  - Python
  - Numpy
  - Pandas
  - Scikit-learn
  - Matplotlib

End-to-End Machine Learning Project

- Look at the big picture
- Get the data
- Explore and visualize the data to gain insights
- Prepare the data for Machine Learning algorithms
- Select a model and train it
- Fine-tune your model
- Present your solution
- Launch, monitor, and maintain your system

End-to-End Machine Learning Project

Working with Real Data

- To learn about Machine Learning
  - It is best to experiment with real-world data
    - Not artificial datasets
- There are thousands of open datasets available
  - Ranging across all sorts of domains

A few places you can look to get 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

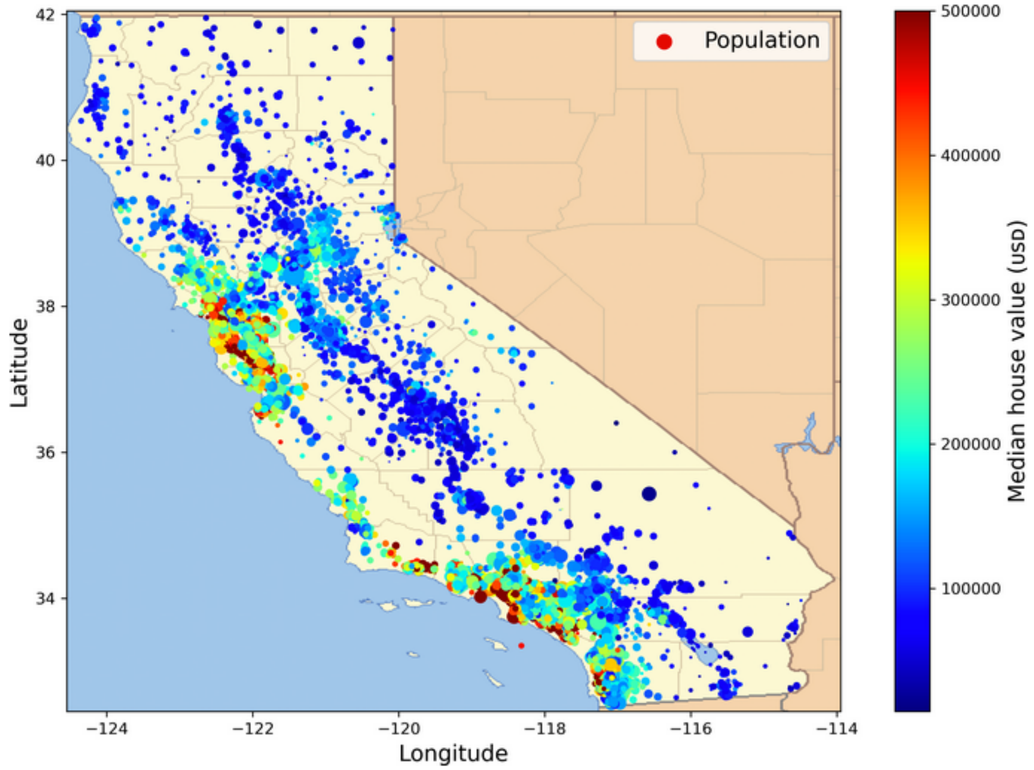
- Other pages listing many popular open data repositories
  - Wikipedia's list of Machine Learning datasets
  - Quora.com
  - The datasets subreddit

California Housing Prices

- Dataset from the StatLib repository
- Based on data from the 1990 California census
- It has many qualities for learning
- Added a categorical attribute
- Removed a few features

Building a model of housing prices in the state

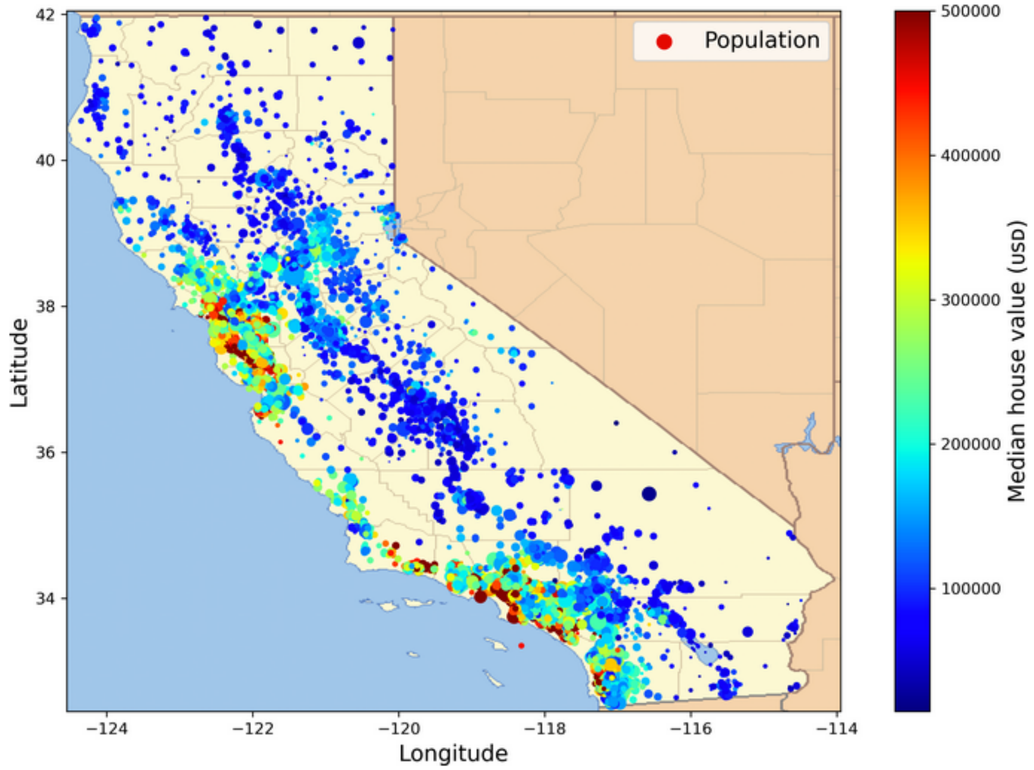
- It includes metrics
  - Population
  - Median income
  - Median housing price
    - Each block group in California
    - The smallest geographical unitT
    - We will it "districts”



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Building a model of housing prices in the state

- The model should learn from this data
  - Be able to predict the median housing price in any district
    - Given all the other metrics



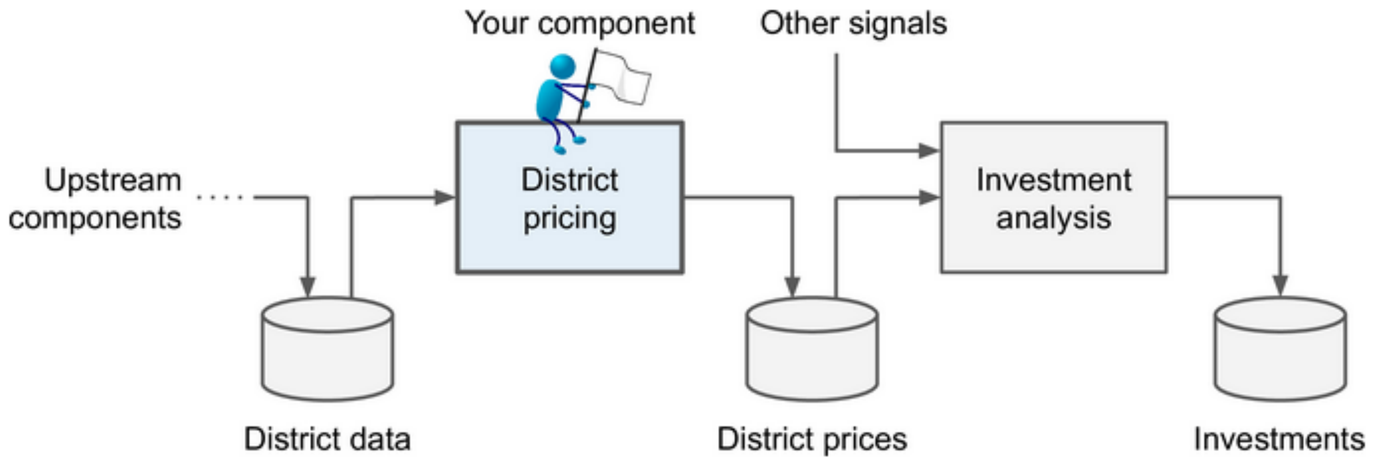
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Frame the Problem

- What is exactly the business objective is?
  - Building a model is probably not the end goal
  - How do we expect to use and benefit from this model?
  - It will determine how we frame the problem
    - Algorithms we will select
    - Performance measure we will use to evaluate your model
    - How much effort we will spend tweaking it

A Machine Learning pipeline

- Imagine that the model output will be fed to another Machine Learning system
  - Along with many other signals
  - This downstream system will determine whether it is worth investing
  - In a given area or not



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Pipelines

- A sequence of data processing components
- Each component is self-contained and typically run asynchronously
  - Pulls in a large amount of data
  - Processes the data it
  - Store the result in another data store

Pipelines

- The system simple to grasp
- The architecture quite robust
  - If a component breaks down
    - The downstream components can often continue to run normally
      - Using the last output from the broken component

How the estimation was done?

- What the current solution looks like?
  - Might give you a reference for performance
  - Insights on how to solve the problem
  - Manually estimation by experts
    - This is costly and time-consuming
    - Usually their estimates are not great

Understanding the problem

- Determine what kind of training supervision the model will need
  - Supervised, unsupervised, semi-supervised, self-supervised, or Reinforcement Learning
- What kind of task?
  - Classification
  - Regression
  - Something else
- Should you use batch learning or online learning techniques

In our case

- A typical supervised learning task
  - The model can be trained with labeled examples
    - Each instance comes with the expected output
      - The district’s median housing price

In our case

- A typical regression task
  - The model will predict a value
    - A multiple regression problem
      - Use multiple features to make a prediction
    - A univariate regression problem
      - Predicting a single value for each district

In our case

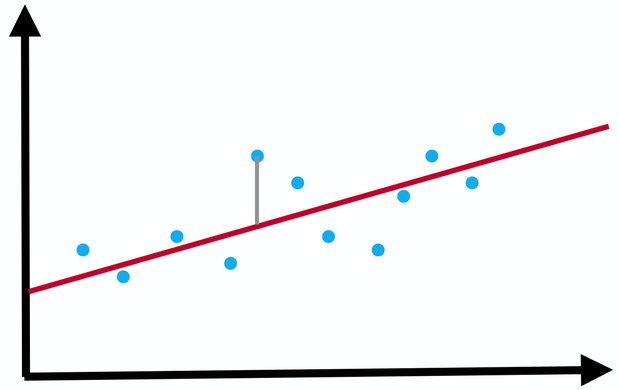
- Batch learning
  - No continuous flow of data coming into the system
  - No particular need to adjust to changing data rapidly
  - The data is small enough to fit in memory

Select a Performance Measure

- A typical performance measure for regression problems
  - Root Mean Square Error — RMSE
    - How much error the system makes in its predictions
    - A higher weight given to large errors

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"



Some notations

Notation	Meaning
m	The number of instances in the dataset
x(i)	The i-th instance of the vector of all the feature values*
y(i)	The i-th instance of the vector of labels

* Excluding the labels

- Example
  - The first district in the dataset
  - Ignoring the other features for now

Features	Value
Longitude	–118.29°
Latitude	33.91°
Inhabitants	1,416
Median income	$38,372
Median house value	$156,400

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

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Some notations

- X —> matrix containing all the feature values of all instances in the dataset
  - There is one row per instance
  - The i-th row is equal to the transpose of x(i) —> $(x(i))^T$
- In our previous example

$$\mathbf{X} = \begin{pmatrix} \left(\mathbf{x}^{(1)}\right)^T \\ \left(\mathbf{x}^{(2)}\right)^T \\ \vdots \\ \left(\mathbf{x}^{(1999)}\right)^T \\ \left(\mathbf{x}^{(2000)}\right)^T \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Some notations

Notation	Meaning
h	system's prediction function, aka a hypothesis
$\hat{y}(i)$	The predicted value for the i-th instance

- $\hat{y}(i) = h(x(i))$
- Example
  - The system predicts that the median housing price in the first district is $158,400
  - $\hat{y}(1) = h(x(1)) = 158,400$
  - The prediction error for this district is $\hat{y}(1) - y(1) = 2,000$

Performance measure

- The RMSE is generally the preferred performance measure for regression tasks
- In some contexts you may prefer to use another function
  - There are many outlier districts
    - Consider using the Mean Absolute Error — MAE
      - The average absolute deviation

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Check the Assumptions

- List and verify the assumptions that have been made so far
  - Help you catch serious issues early on

Check the Assumptions — example

- Assuming district prices is the system output
  - To be fed into a downstream Machine Learning system
  - If the downstream system converts the prices into categories
    - “cheap”, “medium” or “expensive”
    - Getting the price perfectly right is not important at all
    - The system just needs to get the category right
      - A classification task and not a regression task

Getting ready

First steps

- Log in into your google account
- Once logged in and you try to run the code
  - A security warning telling you that this notebook was not authored by Google
    - A malicious person could create a notebook that tries to trick you into
      - Entering your Google credentials so they can access your personal data.
      - Before you run a notebook
        - Always make sure you trust its author
        - Double-check what each code cell will do before running it

Creating a new code cell

- Selecting Insert Code cell from the menu
- Click the + Code button in the toolbar
- Hover your mouse over the bottom of a cell until you see
  - + Code and + Text appear
    - Click on + Code

Creating a new code cell

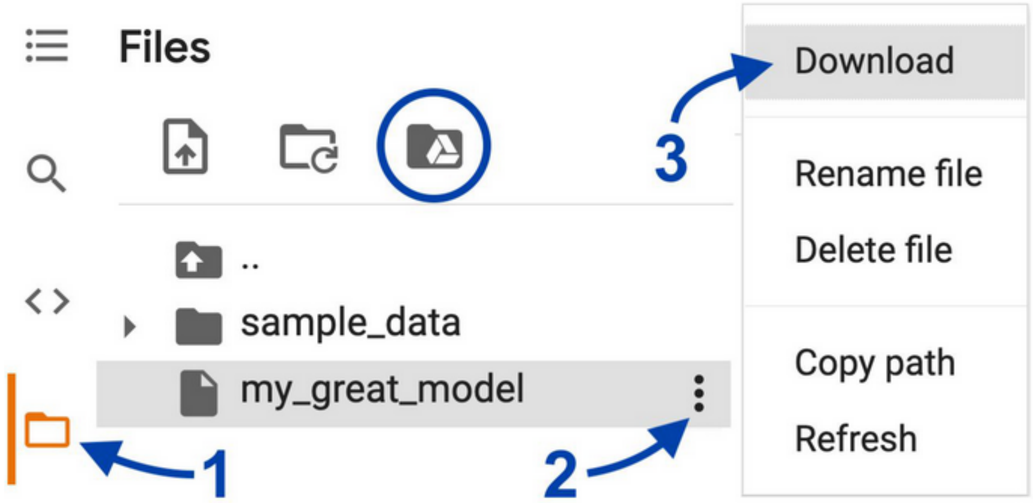
- In the new code cell
  - Type some Python code
    - print("Hello World")
  - To run the code
  - Press Shift-Enter to run this code
  - Click on the ▷ button on the left side of the cell)

Saving Your Code Changes and Your Data

- The changes to a Colab notebook
  - Persist for as long as you keep your browser tab open
  - Make sure you save a copy of the notebook to your Google Drive
    - Selecting File
      - Save a copy in Drive
  - Download the notebook to your computer
    - Selecting File > Download > Download .ipynb
- Google Colab is meant only for interactive use
  - Cannot let the notebooks run unattended for a long period of time
    - The Runtime will be shutdown and all of its data will be lost

Saving the data

- When the notebook generates data that you care about
  - Download this data before the Runtime shuts down
    - (1) click on the Files icon
    - (2) find the file you want to download
      - click on the vertical dots
    - (3) click Download



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Mount your Google Drive on the Runtime

- Allowing the notebook to read and write files directly to Google Drive
  - Like local directory
    - Your Google Drive will be mounted at
      - /content/drive/MyDrive
- To backup a data file
  - Copy it to this directory by running
    - !cp /content/my_great_model /content/drive/MyDrive
- Any command starting with a bang (!) is treated as a shell command
  - Colab Runtimes run on Linux — Ubuntu

The Power and Danger of Interactivity

- Jupyter notebooks are interactive
  - Can run each cell one by one
  - Stop at any point
  - Insert a cell and play with the code
  - Go back and run the same cell again

The Power and Danger of Interactivity

- Problems
  - Very easy to run cells in the wrong order
  - Forget to run a cell
    - If this happens
      - The subsequent code cells are likely to fail
  - When you find a weird error
    - Try restarting the Runtime
    - Selecting Runtime > Restart runtime from the menu

Getting and understanding the data

Download the Data

- Data would be available in a relational database or some other common data store
  - Spread across multiple tables/documents/files
- Accessing the data
  - Get your credentials and access authorizations
  - Familiarize yourself with the data schema

Download the Data

- In our project
  - Download a single compressed file, housing.tgz
    - Contains a comma-separated values (CSV) file: housing.csv
- Write a function to download and decompress the data
  - Useful when the data changes regularly
    - Need to install the dataset on multiple machines

Loading housing data

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

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.
        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()
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Take a Quick Look at the Data Structure

- Let's take a look at the top five rows of 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
3	-122.25	37.85	52.0	5.6431	NEAR BAY	341300.0
4	-122.25	37.85	52.0	3.8462	NEAR BAY	342200.0

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Get a quick description of the data

```
>>> 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
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

ocean_proximity feature

- How many districts on each category?

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Visualizing basic statistics

```
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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

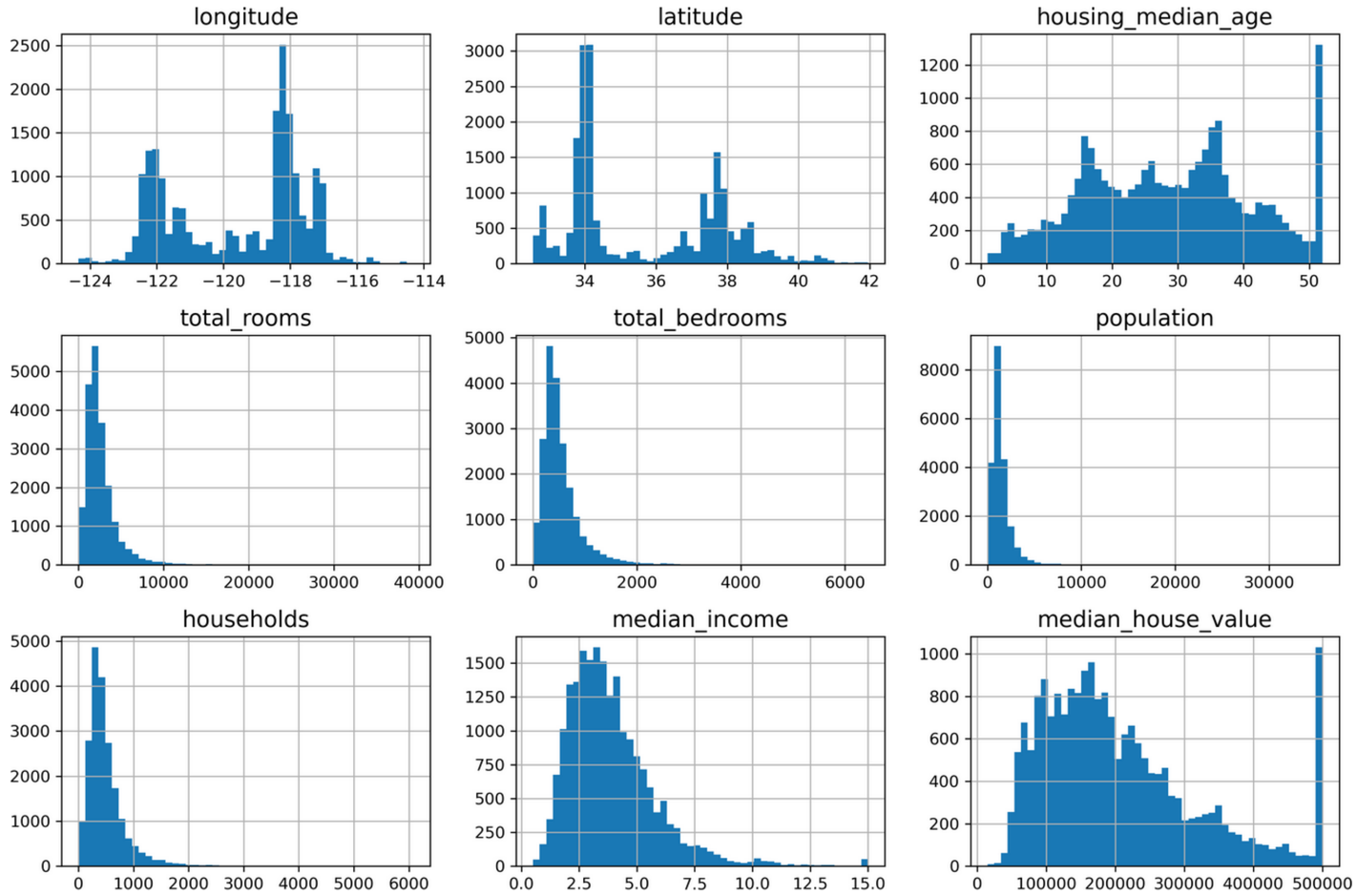
Ploting a histogram

- A quick way to get a feel of the type of data you are dealing
  - For each numerical attribute

```
import matplotlib.pyplot as plt

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

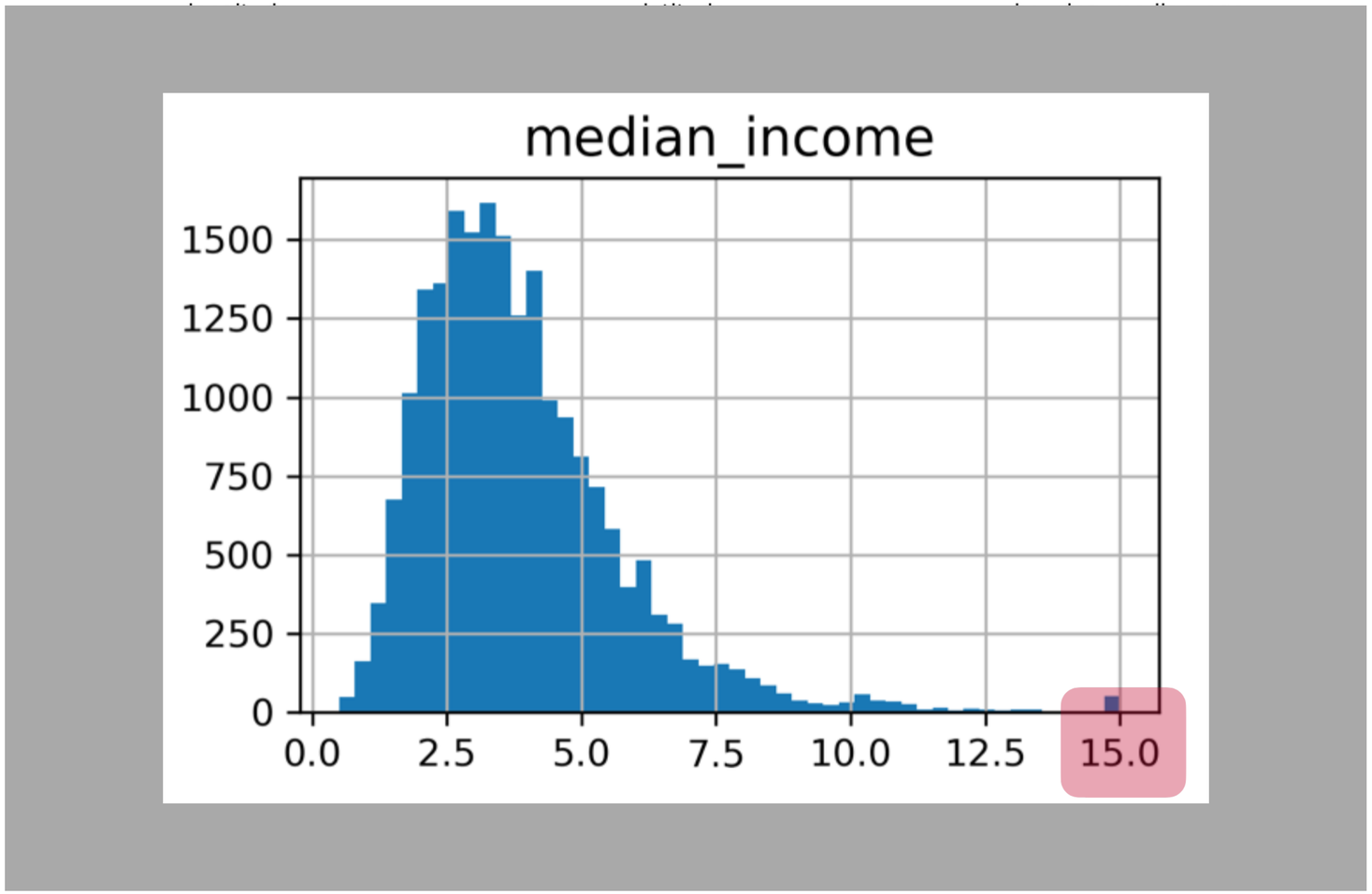
source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Histogram of numerical attributes



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Histogram of numerical attributes



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Preprocessed attributes

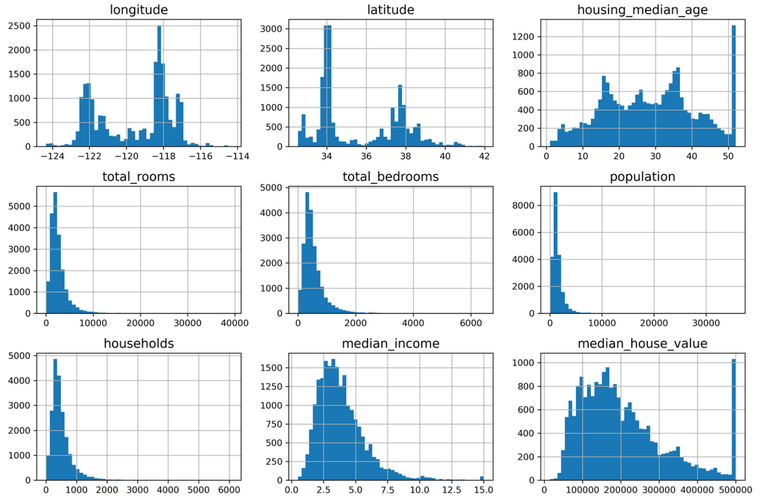
- Common in Machine Learning
  - Important to understand how the data was computed
- Other features capped
  - The housing median age
  - The median house value
    - Might be a serious problem since it is your target attribute (your labels)
    - The Machine Learning algorithms may learn that prices
      - Never go beyond that limit
      - If we need precise predictions even beyond $500,000?

Possible solutions

- Collect proper labels for the districts whose labels were capped
- Remove those districts from
  - The training set
  - Test set
  - The system should not be evaluated poorly if it predicts values beyond $500,000

Data skewness

- Many histograms are skewed right
  - Extend much farther to the right of the median than to the left
- This may make it a bit harder for some ML algorithms to detect patterns
  - Try transforming these attributes
    - More symmetrical
      - Bell-shaped distribution



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating a test set

Creating a Test Set

- Create a test set and put it aside
  - Never look at it
- Data snooping bias
  - Our brain is an amazing pattern detection system
    - It is highly prone to overfitting
    - The estimation of the generalization error using the test set
      - Too optimistic and the system will not perform as well as expected
- Pick some instances randomly
  - Typically 20% of the dataset
  - Less if your dataset is very large

Creating a Test Set

```
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]
```

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

What is the first problem?

- Running the program more than once
  - It will generate a different test set
  - The ML algorithms will get to see the whole dataset
    - We want to avoid it
- Possible solutions
  - Save the test set on the first run and then load it in subsequent runs
  - Set the random number generator’s seed to a fixed value
    - np.random.permutation() will always generate the same shuffled indices

Still a problem

- The next time you fetch an updated dataset
- Use each instance’s identifier to decide
  - It should go in the test set
  - Ensure a stable train/test split even after updating the dataset
  - Assuming instances have a unique and immutable identifier

Still a problem — example

- Compute a hash of each instance's identifier
  - Put that instance in the test set
    - If the hash is lower than or equal to 20% of the maximum hash value
  - The new test set will contain 20% of the new instances
    - It will not contain any instance that was previously in the training set

Creating a test set

```
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]
```

```
housing_with_id = housing.reset_index()  # adds an `index` column
train_set, test_set = split_data_with_id_hash(housing_with_id, 0.2, "index")
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Still a problem

- When you use the row index as a unique identifier
  - Make sure that
    - New data gets appended to the end of the dataset
    - No row ever gets deleted
- If it is not possible
  - Use the most stable features to build a unique identifier
    - A district's latitude and longitude
      - Combine them into an ID

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_data_with_id_hash(housing_with_id, 0.2, "id")
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Spliting the dataset with Scikit-learn

- **train_test_split()**
  - The same thing as the function **shuffle_and_split_data()**
  - Additional features
    - A **random_state** parameter
      - Allows you to set the random generator seed
    - Pass multiple datasets with an identical number of rows
      - It will split them on the same indices
      - Useful when you have a separate DataFrame for labels

Spliting the dataset with Scikit-learn

```
from sklearn.model_selection import train_test_split

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Sampling bias

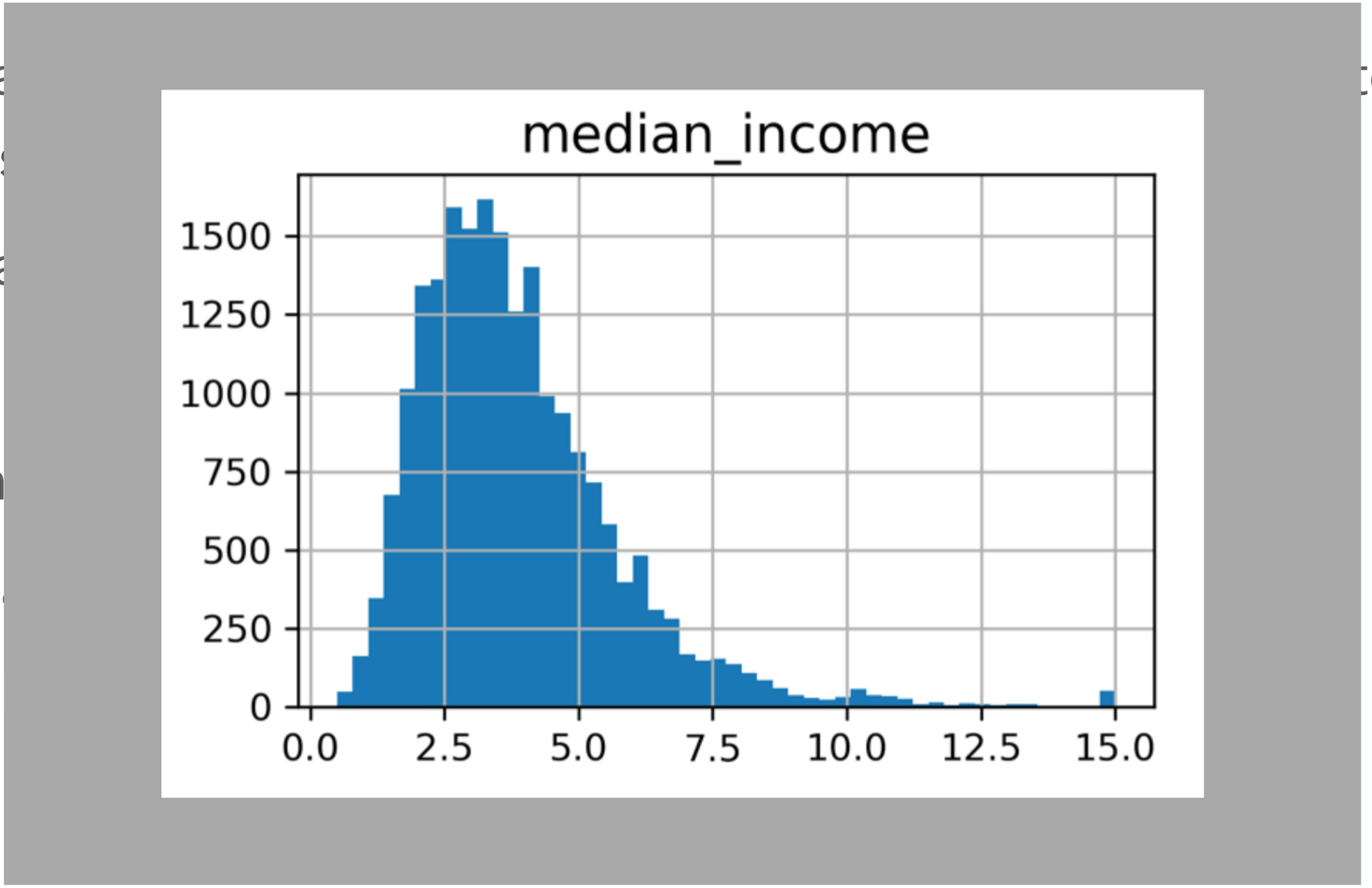
- Purely random sampling methods
  - This is generally fine if your dataset is large enough
    - Especially relative to the number of attributes
  - Otherwise
    - Risk of introducing a significant sampling bias

Sampling bias

- In a survey with 1,000 in France
- Stratified sampling
  - Ensure that these 1,000 people are representative of the whole population
    - 54.1% women and 45.9% men
      - Try to maintain this ratio in the sample: 541 female and 459 male

Median income feature

- Consider tha … to predict median hou…
  - Ensure tha…
    - Various …
  - Median in…
    - Create …
    - Take a l…



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Median income categories

- Dividing into categories

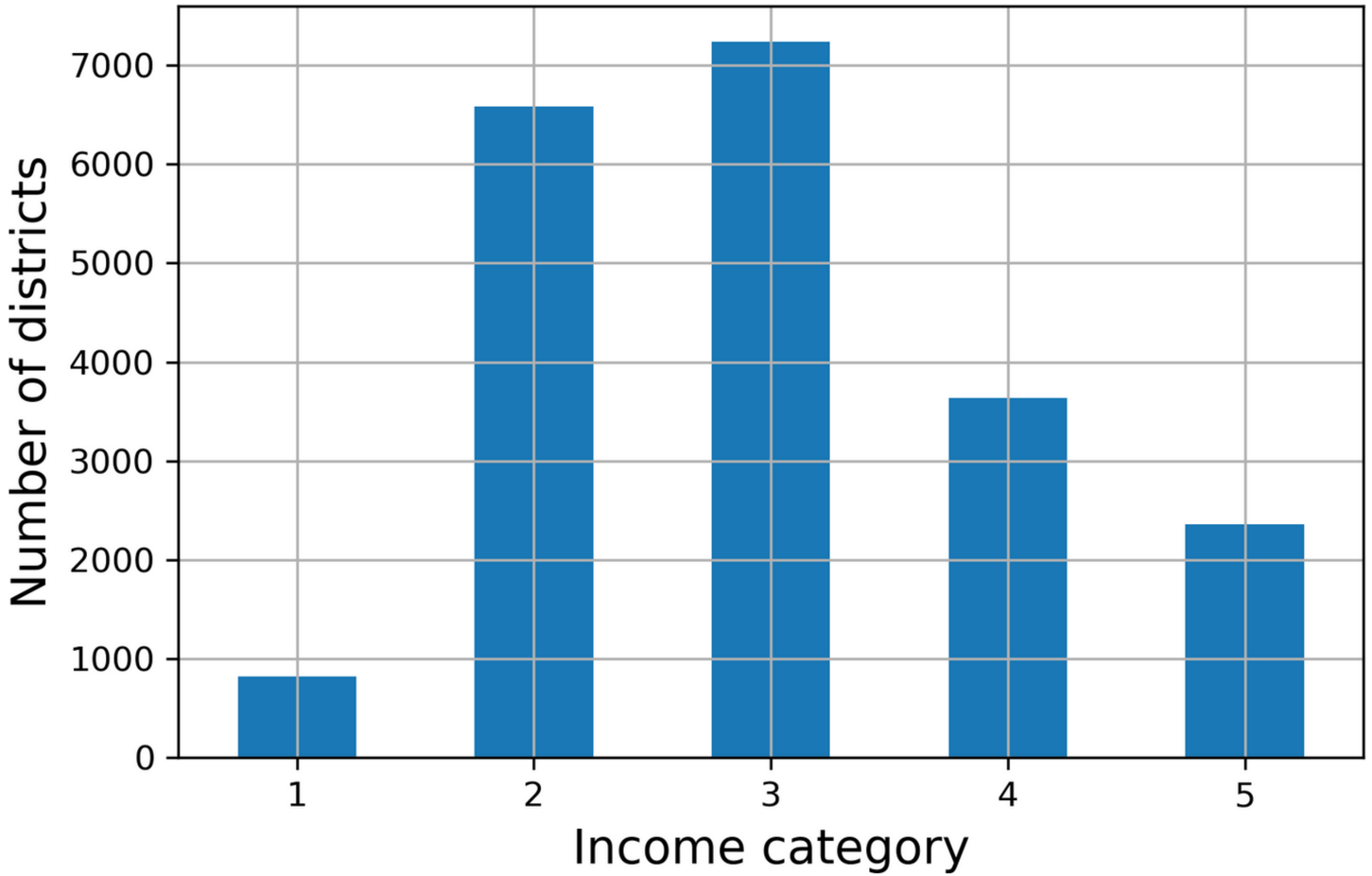
```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

- Plotting the bar graph

```
housing["income_cat"].value_counts().sort_index().plot.bar(rot=0, grid=True)
plt.xlabel("Income category")
plt.ylabel("Number of districts")
save_fig("housing_income_cat_bar_plot")  # extra code
plt.show()
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Ploting the histogram



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Stratified sampling — income category

- Scikit-Learn provides a number of splitter classes
  - In the **sklearn.model_selection** package
    - Implement various strategies to split the dataset
    - Into a **train** set and a **test** set
  - Each splitter has a **split()** method
    - Returns an iterator over different train/test splits of the same data
      - Yields the **train** and **test** indices
        - Not the data itself
    - Multiple splits can be useful
      - Better estimate the performance of your model

Stratified sampling — income category

```
from sklearn.model_selection import StratifiedShuffleSplit

splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
strat_splits = []
for train_index, test_index in splitter.split(housing, housing["income_cat"]):
    strat_train_set_n = housing.loc[train_index]
    strat_test_set_n = housing.loc[test_index]
    strat_splits.append([strat_train_set_n, strat_test_set_n])
```

- For now, we can just use the first split:

```
strat_train_set, strat_test_set = strat_splits[0]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Stratified sampling — income category

- A shorter way to get a single split using the **train_test_split()** function

```
strat_train_set, strat_test_set = train_test_split(
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

- Take a look at the income category proportions in the test set

```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

```
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Sampling bias comparison

- Compare the income category proportions in the full dataset
  - The test set generated using stratified sampling
    - Income category proportions almost identical

Income Category	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
1	3.98	4.00	4.24	0.36	6.45
2	31.88	31.88	30.74	-0.02	-3.59
3	35.06	35.05	34.52	-0.01	-1.53
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Sampling bias comparison

```
def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)

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

compare_props = pd.DataFrame({
    "Overall %": income_cat_proportions(housing),
    "Stratified %": income_cat_proportions(strat_test_set),
    "Random %": income_cat_proportions(test_set),
}).sort_index()
compare_props.index.name = "Income Category"
compare_props["Strat. Error %"] = (compare_props["Stratified %"] /
                                   compare_props["Overall %"] - 1)
compare_props["Rand. Error %"] = (compare_props["Random %"] /
                                  compare_props["Overall %"] - 1)
(compare_props * 100).round(2)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Reverting the data back to its original state

- Now we won’t use the income_cat column anymore

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Taking a deeper look

Discover and Visualize the Data to Gain Insights

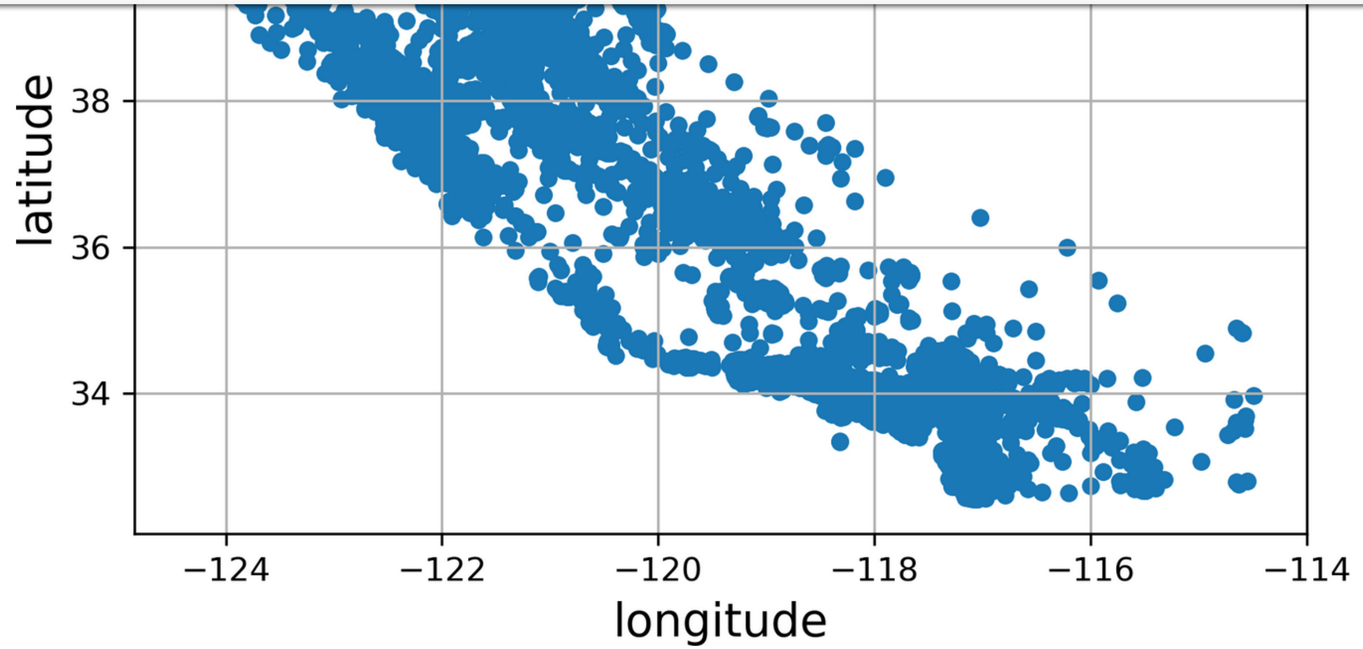
- Take a deeper look at the data
- Make sure you have put the test set
  - We will only exploring the training set
- For very large training seta
  - Sample an exploration set
    - Make manipulations easy and fast during the exploration phase
- Make a copy of the original so you can revert to it afterwards

```
housing = strat_train_set.copy()
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

A geographical scatterplot of the data

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
save_fig("bad_visualization_plot")  # extra code
plt.show()
```

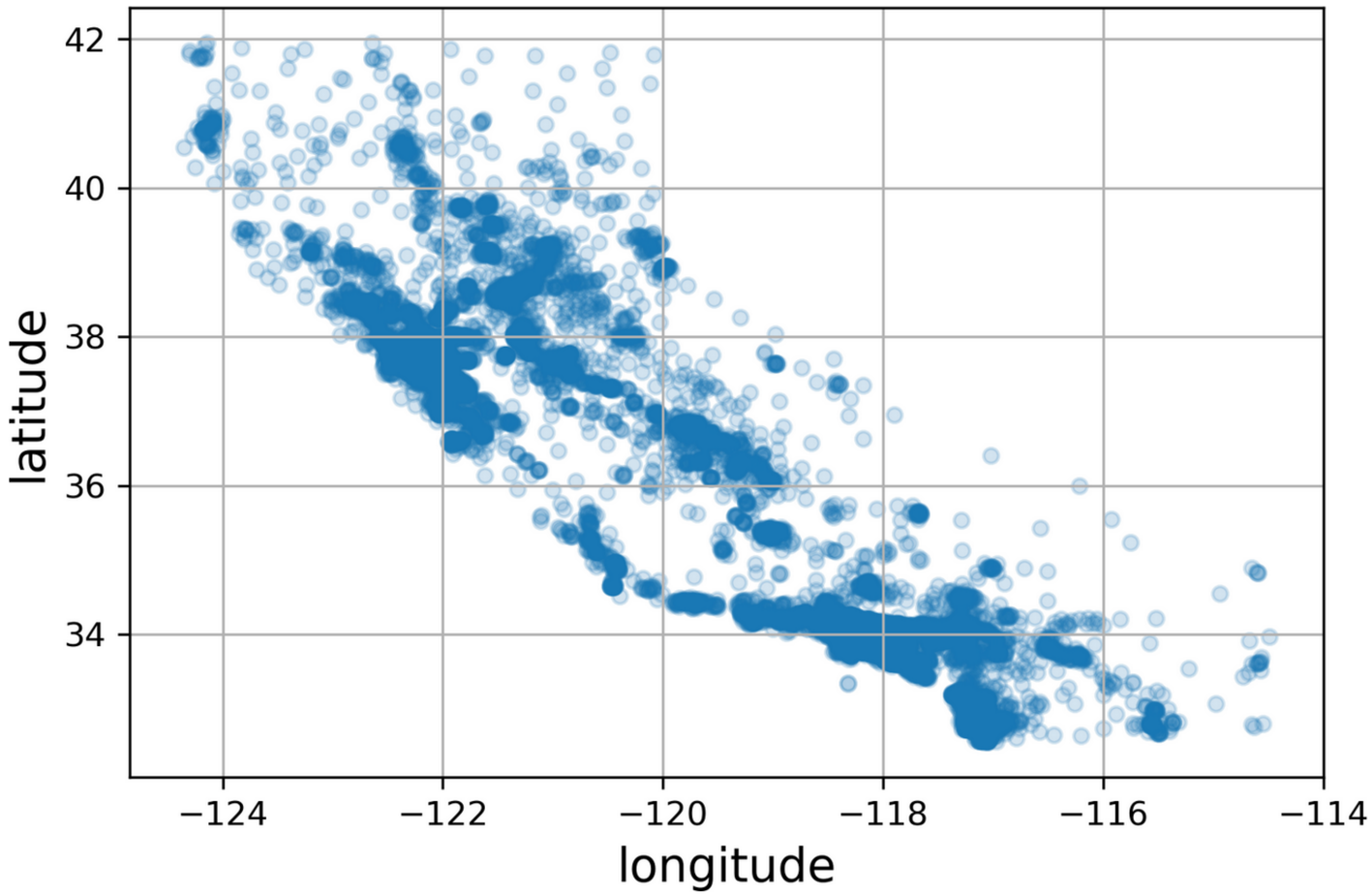


source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

A better visualization

- Highlightin
- Setting the
  - Makes it
    density c

a high



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

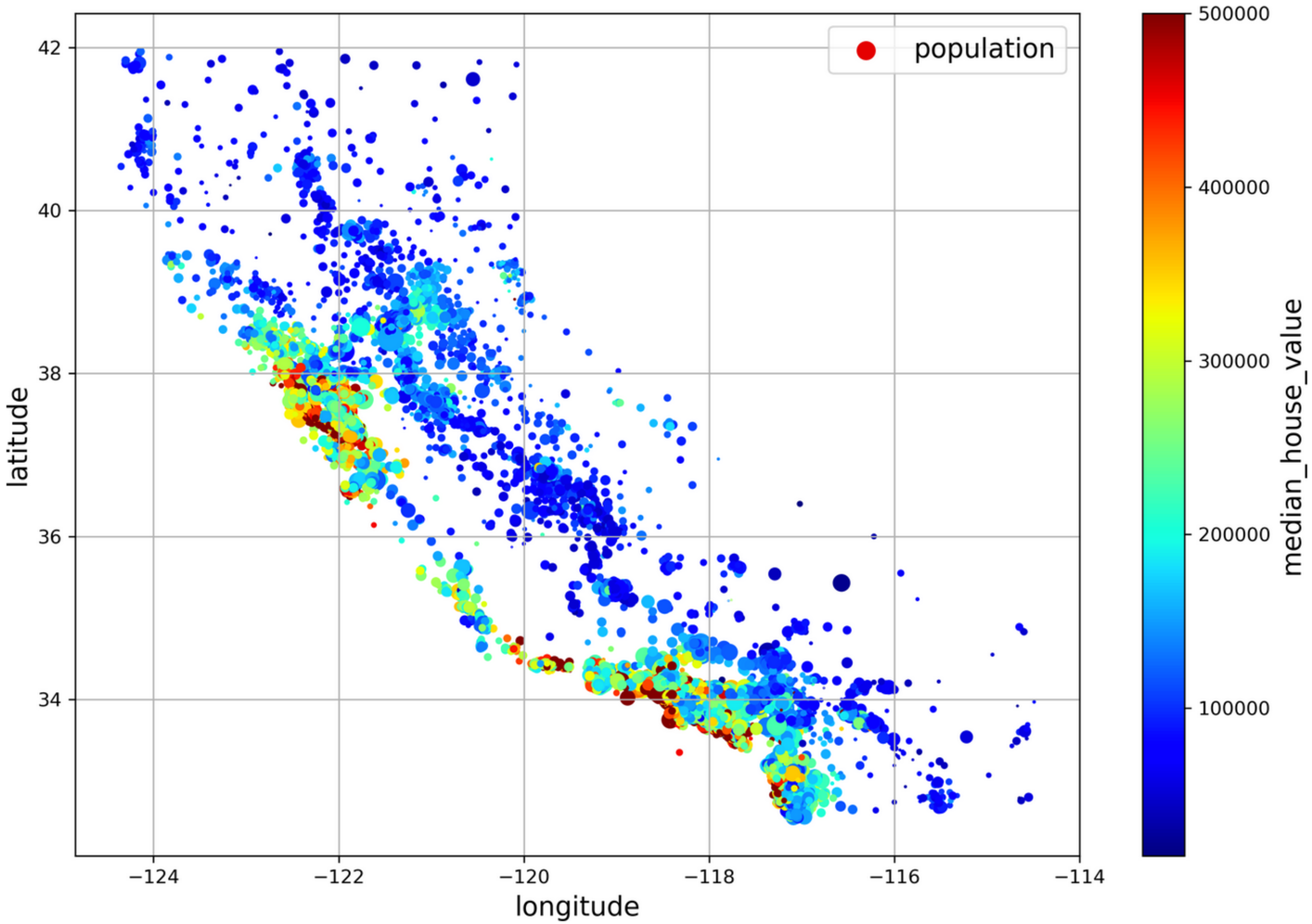
Looking at the housing prices

- The radius of each circle represents the district's population (option s)
- Color represents the price (option c)
- Use a predefined color map (option cmap) called jet
  - which ranges from blue (low values) to red (high 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))
save_fig("housing_prices_scatterplot")  # extra code
plt.show()
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

A geographical scatterplot of the data



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Insights about the dataset

- A clustering algorithm should be useful for detecting the main cluster
  - Adding new features that measure the proximity to the cluster centers
- The ocean proximity attribute may be useful
  - Northern California the housing
    - Prices in coastal districts are not too high
      - It is not a simple rule

Looking for correlations

- With a dataset not too large
  - We can easily compute the standard correlation coefficient
    - Pearson's r
    - Between every pair of attributes using the **corr()** method
    - The correlation coefficient ranges from –1 to 1
      - Close to 1 —> strong positive correlation
      - Close to –1 —> strong negative correlation

```
corr_matrix = housing.corr()
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Looking for correlations

```
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
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Checking for correlation II

- Use the Pandas scatter_matrix() function
  - Plots every numerical attribute against every other numerical attribute
    - There are 11 numerical attributes
      - 121 plots
    - We focus on a few promising attributes
      - Seem most correlated with the median housing value

Checking for correlation II

- This scatter matrix plots every numerical attribute
  - Against every other numerical attribute
    - A histogram of each numerical attribute's values on the main diagonal

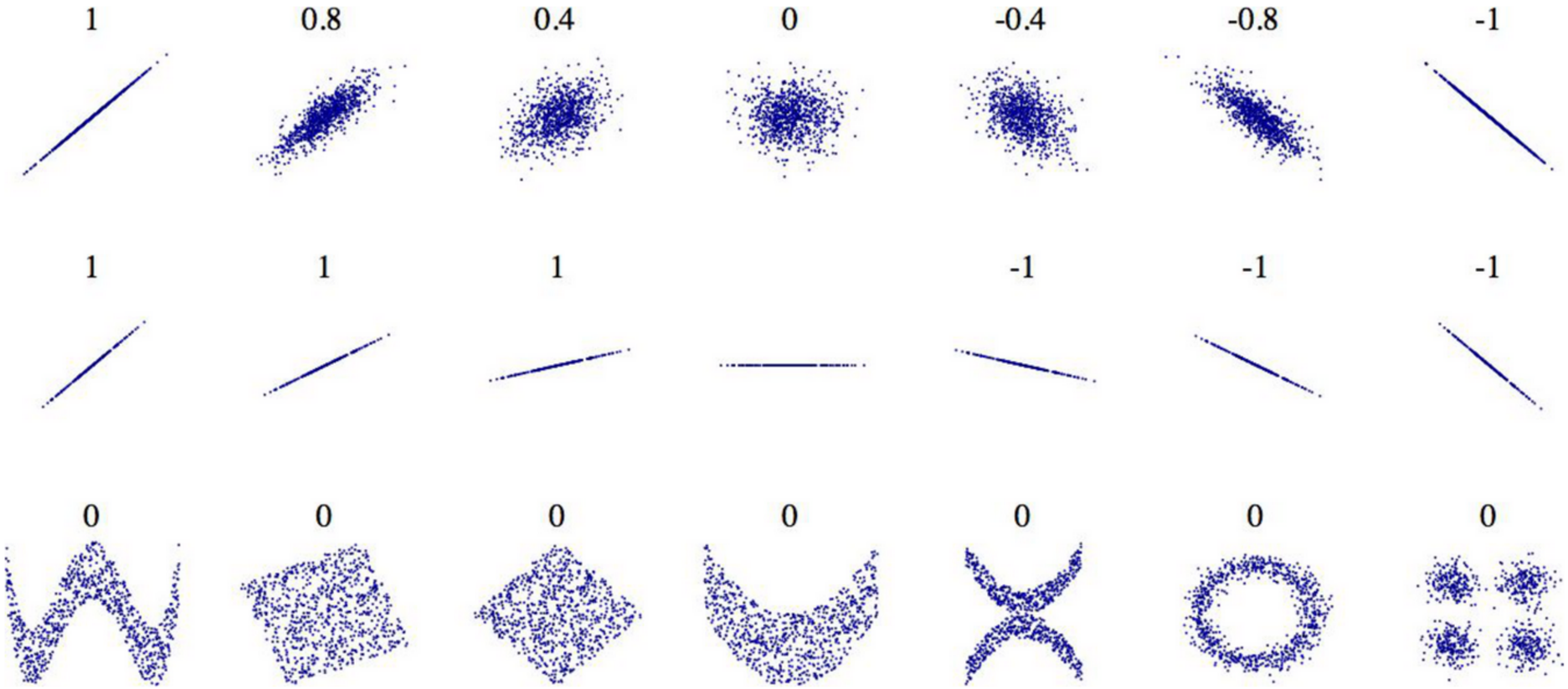
```
from pandas.plotting import scatter_matrix

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Correlation coefficient

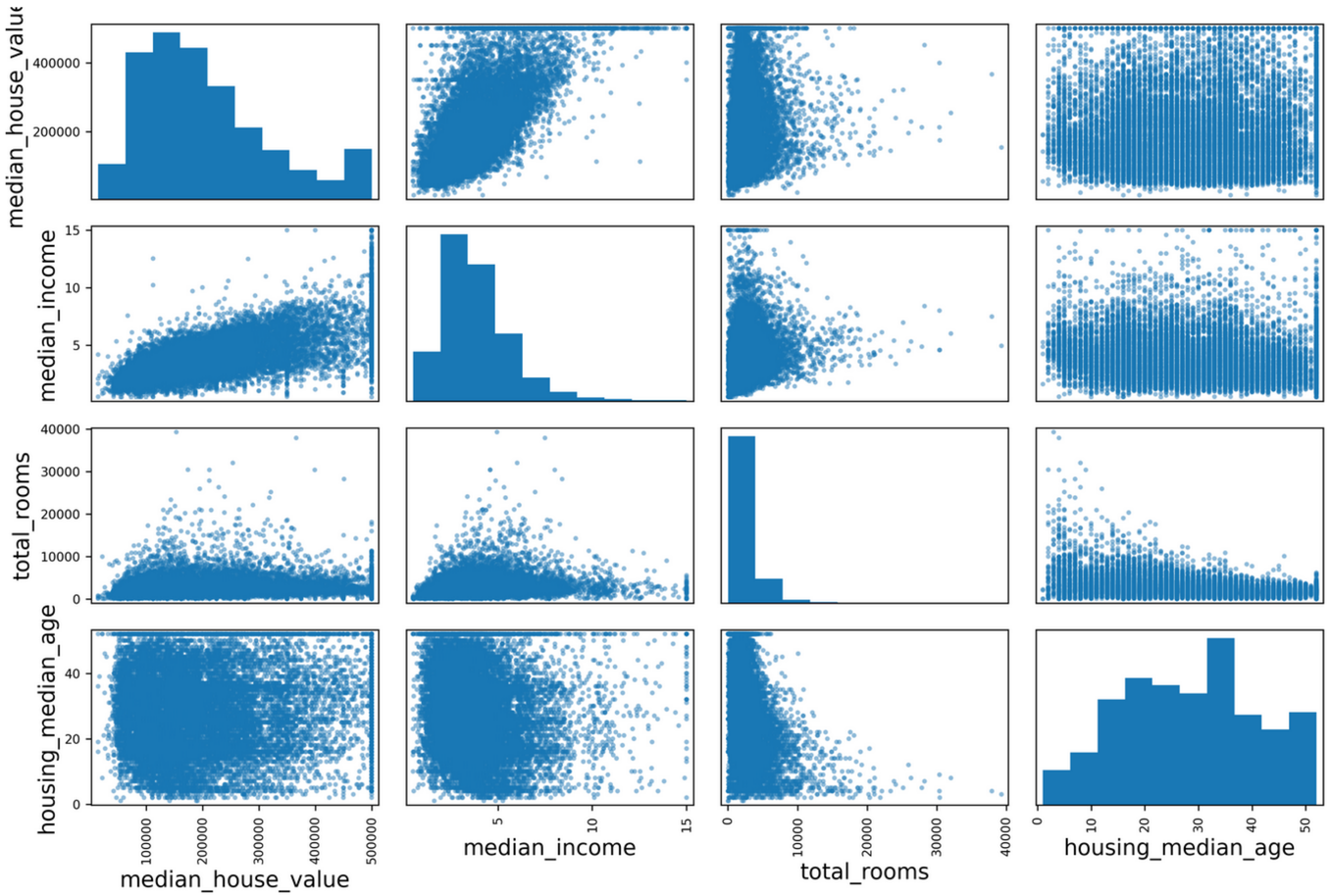
- The correlation coefficient only measures linear correlations
  - x goes up —> y generally goes up/down
- It may completely miss out on nonlinear relationships
  - x approaches 0 —> y generally goes up

Standard correlation coefficient of various datasets



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

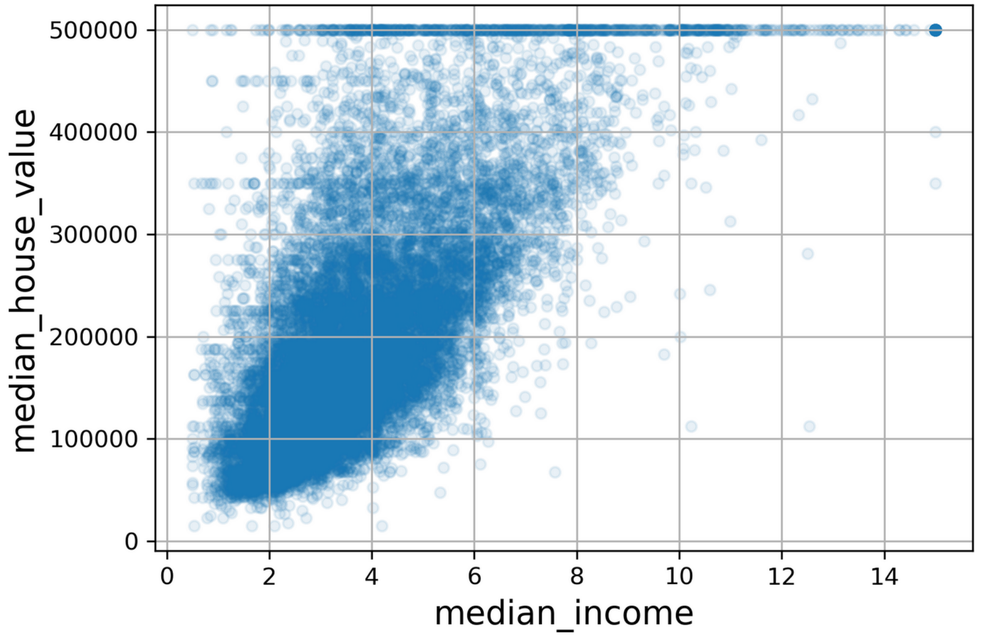
Checking for correlation II



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Zoom in median income

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



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Experimenting with Attribute Combinations

- Try out various attribute combinations
  - Not very useful
    - The total number of rooms in a district
      - If you don’t know how many households there are
        - What you really want is the number of rooms per household
    - The total number of bedrooms
      - Probably want to compare it to the number of rooms
- The population per household
  - Seems like an interesting attribute combination to look at

Attribute combinations

```
housing["rooms_per_house"] =
housing["bedrooms_ratio"] =
housing["people_per_house"]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Correlations — attribute combinations

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Attribute combinations

- The new bedrooms_ratio attribute
  - Much more correlated with the median house value
  - The total number of rooms or bedroom
  - Apparently houses with a lower bedroom/room ratio tend to be more expensive
- The number of rooms per household is also more informative
  - The total number of rooms in a district

Attribute combinations

- This round of exploration does not have to be absolutely thorough
  - It is an iterative process
    - Prototype up and running
      - Analyze its output to gain more insights and come back to this exploration step

Prepare the Data for Machine Learning Algorithms

Prepare the Data for Machine Learning Algorithms

- Writing functions for this purpose
  - Allow you to reproduce these transformations easily on any dataset
    - The next time you get a fresh dataset
  - Gradually build a library of transformation functions that you can reuse in future projects
  - Use these functions in your live system to transform the new data before feeding it to your algorithms
  - Easily try various transformations and see which combination of transformations works best

Using the original sets

- Revert to a clean training set
  - Copying strat_train_set once again
  - Let’s also separate the predictors and the labels
    - Not necessarily the same transformations will be applied
      - The predictors and the target values

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Data Cleaning

- Most Machine Learning algorithms cannot work with missing features
  - Create a few functions to take care of them
- The total_bedrooms attribute has some missing values
  - We have three options
    - Get rid of the corresponding districts
    - Get rid of the whole attribute
    - Imputation
      - Setting the missing values to some value
        - Zero, the mean, the median, etc.

Dealing with missing values

- You can accomplish these easily using DataFrame's methods
  - **dropna()**, **drop()**, **fillna()**

```
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)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Dealing with missing values

- We will use option 3
  - The least destructive
    - Use a handy Scikit-Learn class : SimpleImputer
      - Stores the median value of each feature
        - Make it possible to impute missing values not only on the training set
        - On the validation set, the test set, and any new data fed to the model

```
from sklearn.impute import SimpleImputer

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Dealing with missing values

- The median can only be computed on numerical attributes
  - Need to create a copy of the data with only the numerical attributes

```
housing_num = housing.select_dtypes(include=[np.number])
```

- Now you can fit the imputer instance to the training data
  - Using the **fit()** method:

```
imputer.fit(housing_num)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Dealing with missing values

- The imputer has simply computed the median of each attribute
  - Stored the result in its statistics_ instance variable
- Only the total_bedrooms attribute had missing values
- We cannot be sure that there won't be any missing values
  - In new data after the system goes live, so it is safer to apply the imputer to all the numerical attributes:

```
>>> imputer.statistics_
array([-118.51 , 34.26 , 29. , 2125. , 434. , 1167. , 408. , 3.
>>> housing_num.median().values
array([-118.51 , 34.26 , 29. , 2125. , 434. , 1167. , 408. , 3.
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Dealing with missing values

- Use this “trained” imputer to transform
  - The **training** set by replacing missing values with the learned medians

```
X = imputer.transform(housing_num)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

- Missing values can also be replaced with
  - The mean value —> strategy=“mean”
  - Most frequent value —> strategy=“most_frequent”
  - Constant value —> strategy=“constant", fill_value= <value>

More powerful imputers in sklearn.impute

- *KNNImputer*
  - Replaces each missing value
    - The mean of the k nearest neighbors' values for that feature
    - The distance is based on all the available features

More powerful imputers in sklearn.impute

- *IterativeImputer*
  - Trains a regression model per feature
    - Predict the missing values based on all the other available features
    - It then trains the model again on the updated data
    - Repeats the process several times
      - Improving the models and the replacement values at each iteration

Scikit-learn transformers

- Scikit-Learn transformers output **NumPy** arrays
  - Sometimes SciPy sparse matrices
    - Even when they are fed Pandas DataFrames as input
- So the output of **imputer.transform(housing_num)** is a **NumPy** array
  - X has neither column names nor index
    - It's not too hard to wrap X in a DataFrame
      - Recover the column names and index from **housing_num**:

```
housing_tr.loc[null_rows_idx].head()
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Outliers

Now let's drop some outliers:

```
[ ]  from sklearn.ensemble import IsolationForest

     isolation_forest = IsolationForest(random_state=42)
     outlier_pred = isolation_forest.fit_predict(X)
```

```
     outlier_pred
```

```
array([-1,  1,  1, ...,  1,  1,  1])
```

```
[ ]  #housing = housing.iloc[outlier_pred == 1]
     #housing_labels = housing_labels.iloc[outlier_pred == 1]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Handling Text and Categorical Attributes

- In our dataset there is just one
  - The **ocean_proximity** attribute

```
>>> 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
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Categorical atribute

- It’s not arbitrary text
  - There are a limited number of possible values
    - Represents a category —> a categorical attribute
- Most Machine Learning algorithms prefer to work with numbers
- Converting these categories from text to numbers

```
from sklearn.preprocessing import OrdinalEncoder

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

encoded values in housing_cat_encoded

- 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.]])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Getting the list of categories

- Using the categories_ instance variable
  - A list containing a 1D array of categories for each categorical attribute
- A list containing a single array
  - One categorical attribute

```
ordinal_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

One issue with this representation

- ML algorithms will assume that two nearby values
  - More similar than two distant values
    - This may be fine in some cases
      - For ordered categories such as “bad,” “average,” “good,” and “excellent”
      - Not the case for the **ocean_proximity** column

```
ordinal_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

One-hot encoding

- Solution: one-hot encoding
  - Create one binary attribute per category
    - One attribute equal to 1 when the category is “<1H OCEAN”
      - Zero otherwise
    - Only one attribute will be equal to 1 (hot)
      - The others will be 0 (cold)

```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Sparce matrix

- Very efficient representation for matrices that contain mostly zeroes
  - Internally it only stores the non-zero values and their positions
  - When a categorical attribute has hundreds or thousands of categories
    - One-hot-encoding it results in a very large matrix full of zeros
      - Except for a single 1 per row.
      - Sparse matrix is exactly what we need
- The output of a OneHotEncoder is a SciPy sparse matrix

```
>>> housing_cat_1hot
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
 with 16512 stored elements in Compressed Sparse Row format>
```

```
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.]])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

One-hot encoding

- Set **sparse=False** when creating the OneHotEncoder
  - The **transform()** method will return a regular (dense) NumPy array directly

```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

```
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.]])
```

  - As with the OrdinalEncoder
    - Get the list of categories using the encoder's **categories_** instance variable

A large number of possible categories

- One-hot encoding will result in a large number of input features
  - May slow down training and degrade performance
- Replace the categorical input
  - Useful numerical features related to the categories
    - Replace the ocean_proximity feature
      - The distance to the ocean
    - Replace a country code
      - The country's population and GDP per capita

Feature Scaling and Transformation

Feature Scaling and Transformation

- Machine Learning algorithms don’t perform well
  - When the input numerical attributes have very different scales
    - With few exceptions
- One of the most important transformations
  - Apply to your data is feature scaling

Feature Scaling and Transformation

- In our case —> Without any scaling
  - Most models will be biased towards
    - Ignoring the median income
    - Focusing more on the number of rooms.
- There are two common ways to get all attributes to have the same scale
  - Min-max scaling and standardization.

Min-max scaling

- Also known as normalization is the simplest
  - For each attribute
    - Values are shifted and rescaled
      - Ranging from 0 to 1
- Procedure

```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_num)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Scaling data

- As with all estimators
  - It is important to fit the scalers to the training data only
- Once you have a trained scaler
  - Can use it to **transform()** any other set
    - Validation set, the test set, and new data

Scaling data

- The training set values will always be scaled to the specified range
  - New data contains outliers
    - Scaled outside the range
      - To avoid this
      - Set the **clip** hyperparameter to **True**

Standardization

- Subtracts the mean value
  - Standardized values have a zero mean
- Divides the result by the standard deviation
  - Standardized values have a standard deviation equal to 1
- Does not bound values to a specific range
- Much less affected by outliers

```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Standardization

- If you want to scale a sparse matrix
  - Without converting it to a dense matrix
  - Use a StandardScaler
    - Its **with_mean** hyperparameter set to **False**
    - It will only divide the data by the standard deviation
      - Without subtracting the mean
        - This would break sparsity

Feature with a heavy tail distribution

- When values far from the mean are not exponentially rare
- Min-max scaling and standardization
  - Squash most values into a small range
    - Machine Learning models generally don’t like this at all
- Before you scale the feature
  - Transform it to shrink the heavy tail
  - If possible to make the distribution roughly symmetrical

Feature with a heavy tail distribution

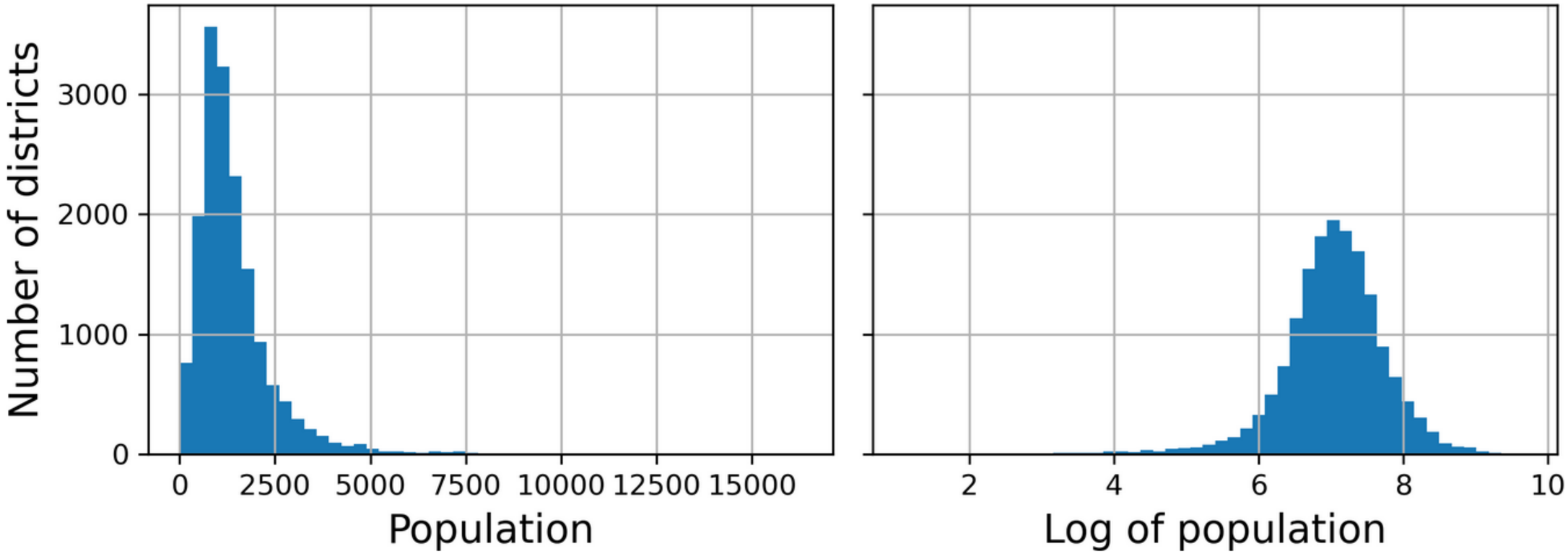
- For positive features with a heavy tail to the right
  - Replace the feature with its square root
  - Raise the feature to a power between 0 and 1
  - When the feature has a really long and heavy tail
    - Power law distribution
      - Replacing the feature with its logarithm may help

Show the histogram of populations and log

```
housing["population"].hist(ax=axs[0], bins=50)
housing["population"].apply(np.log).hist(ax=axs[1], bins=50)
axs[0].set_xlabel("Population")
axs[1].set_xlabel("Log of population")
axs[0].set_ylabel("Number of districts")
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Feature population



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

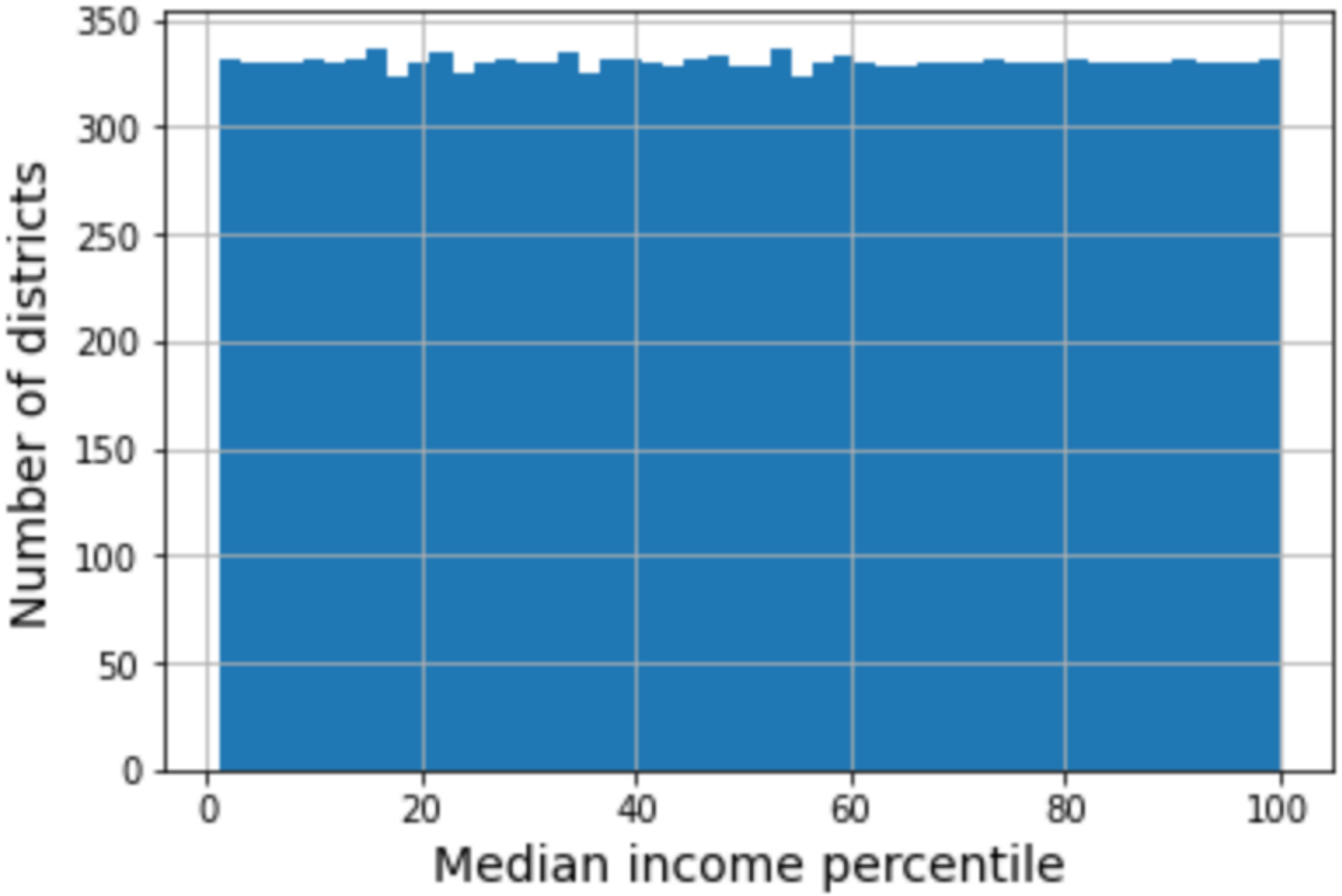
Bucketizing the feature

- Another approach to handle heavy tailed features
  - Chopping its distribution into roughly equal size buckets
    - Replacing each feature value with the index of the bucket it belongs to
- Example
  - Replace each value with its percentile

Bucketizing the feature

- Bucketizing with equal-sized buckets
  - Results in a feature with an almost uniform distribution
  - There’s no need for further scaling
    - Divide by the number of buckets to force the values to the 0–1 range

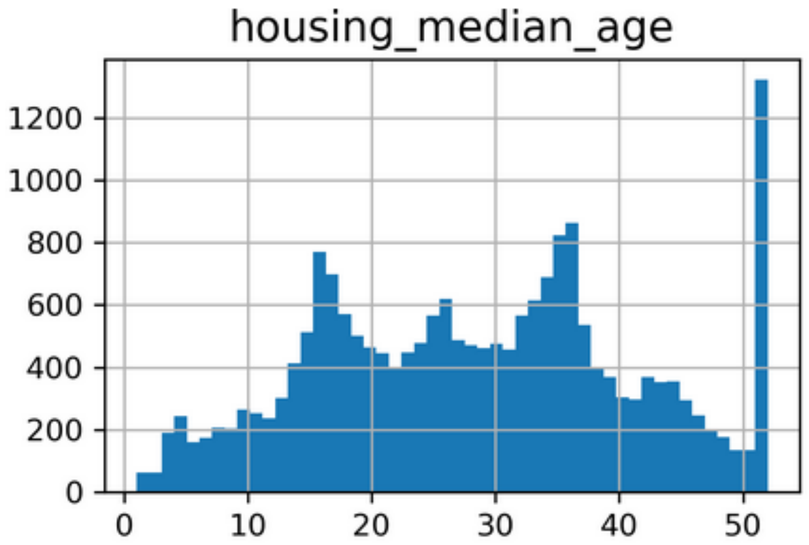
Replace each value with its percentile



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Bucketizing the feature

- Feature with a multimodal distribution
  - Two or more clear peaks
    - The **housing_median_age**
- Use the bucket ids as categories
  - Rather than as numerical values
  - The bucket indices must be encoded
    - For example using a OneHotEncoder



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Transforming multimodal distributions

- Add a feature for each of the modes
  - Representing the similarity between the housing median age and that particular mode
- The similarity measure is typically computed using a Radial Basis Function (RBF)
  - Any function which depends only on the distance between the input value and a fixed point

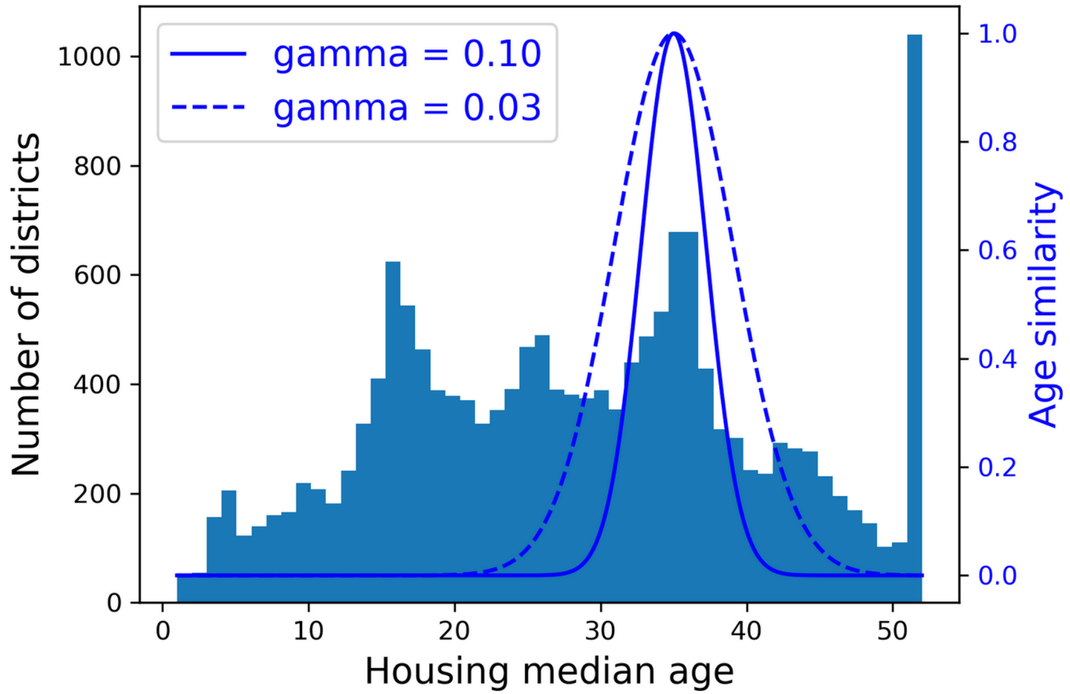
Transforming multimodal distributions

- Gaussian RBF
  - Output value decays exponentially as the input value moves away from the fixed point
  - Example
    - The Gaussian RBF similarity between the housing age x and 35
    - The hyperparameter gamma determines
      - How quickly the similarity measure decays as x moves away from 35.

$$e^{-\gamma x}$$

Transforming multimodal distributions

```
from sklearn.metrics.pairwise import rbf_kernel

age_simil_35 = rbf_kernel(housing[["housing_median_age"]], [[35]], gamma=0.1)
```



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Scaling and transforming the target

- If the target distribution has a heavy tail
  - May replace the target with its logarithm
  - The regression model will now predict the log of the median house value
  - Compute the exponential of the model's prediction
- Most of Scikit-Learn's transformers have an **inverse_transform()** method
  - Makes it easy to compute the inverse of their transformations

Scaling and transforming the target

```
from sklearn.linear_model import LinearRegression

target_scaler = StandardScaler()
scaled_labels = target_scaler.fit_transform(housing_labels.to_frame())

model = LinearRegression()
model.fit(housing[["median_income"]], scaled_labels)
some_new_data = housing[["median_income"]].iloc[:5]  # pretend this is new data

scaled_predictions = model.predict(some_new_data)
predictions = target_scaler.inverse_transform(scaled_predictions)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Construct a TransformedTargetRegressor

- Give it the regression model and the label transformer
- Fit it on the training set
  - Using the original unscaled labels
- It will automatically
  - Use the transformer to scale the labels
  - Train the regression model on the resulting scaled labels
- When we make a prediction
  - It will call the regression model's **predict()** method
  - Use the scaler's **inverse_transform()** method to produce the predictions

Construct a TransformedTargetRegressor

```
from sklearn.compose import TransformedTargetRegressor

model = TransformedTargetRegressor(LinearRegression(),
                                   transformer=StandardScaler())
model.fit(housing[["median_income"]], housing_labels)
predictions = model.predict(some_new_data)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Custom Transformers

- Write your own transformers
  - Cleanup operations
  - Combining specific attributes
- For transformations that don’t require any training
  - Just write a function that takes a NumPy array as input
  - Outputs the transformed array

```
from sklearn.preprocessing import FunctionTransformer

log_transformer = FunctionTransformer(np.log, inverse_func=np.exp)
log_pop = log_transformer.transform(housing[["population"]])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Custom Transformers

- Your transformation function can take hyperparameters
  - Additional arguments
- Example
  - Creating a transformer that computes the Gaussian RBF similarity measure

```
rbf_transformer = FunctionTransformer(rbf_kernel,
                                      kw_args=dict(Y=[[35.]], gamma=0.1))
age_simil_35 = rbf_transformer.transform(housing[["housing_median_age"]])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Note

- The rbf_kernel() does not treat the features separately
  - If you pass it an array with 2 features
    - It will measure the 2D distance (Euclidean) to measure similarity
- Example
  - Add a feature that will measure the geographic similarity
    - Between each district and San Francisco:

```
sf_coords = 37.7749, -122.41
sf_transformer = FunctionTransformer(rbf_kernel,
                                     kw_args=dict(Y=[sf_coords], gamma=0.1))
sf_simil = sf_transformer.transform(housing[["latitude", "longitude"]])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Trainable transformer

- Learning some parameters in the **fit()** method
- Using them later in the **transform()** method
  - Need to write a custom class
- It needs three methods
  - **fit()** —> return self
  - **transform()**
  - **fit_transform()**
    - Adding **TransformerMixin** as a base class

Custom tranformer

```python
from sklearn.cluster import KMeans

class ClusterSimilarity(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=10, gamma=1.0, random_state=None):
        self.n_clusters = n_clusters
        self.gamma = gamma
        self.random_state = random_state

    def fit(self, X, y=None, sample_weight=None):
        self.kmeans_ = KMeans(self.n_clusters, random_state=self.random_state)
        self.kmeans_.fit(X, sample_weight=sample_weight)
        return self  # always return self!

    def transform(self, X):
        return rbf_kernel(X, self.kmeans_.cluster_centers_, gamma=self.gamma)

    def get_feature_names_out(self, names=None):
        return [f"Cluster {i} similarity" for i in range(self.n_clusters)]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

K-Means

- A clustering algorithm that locates clusters in the data
  - The number of cluster —> n_clusters hyperparameter
- After training
  - The cluster centers are available via the cluster_centers_ attribute
- The fit() method of KMeans supports an optional argument sample_weight
  - Specify the relative weights of the samples

Custom tranformer

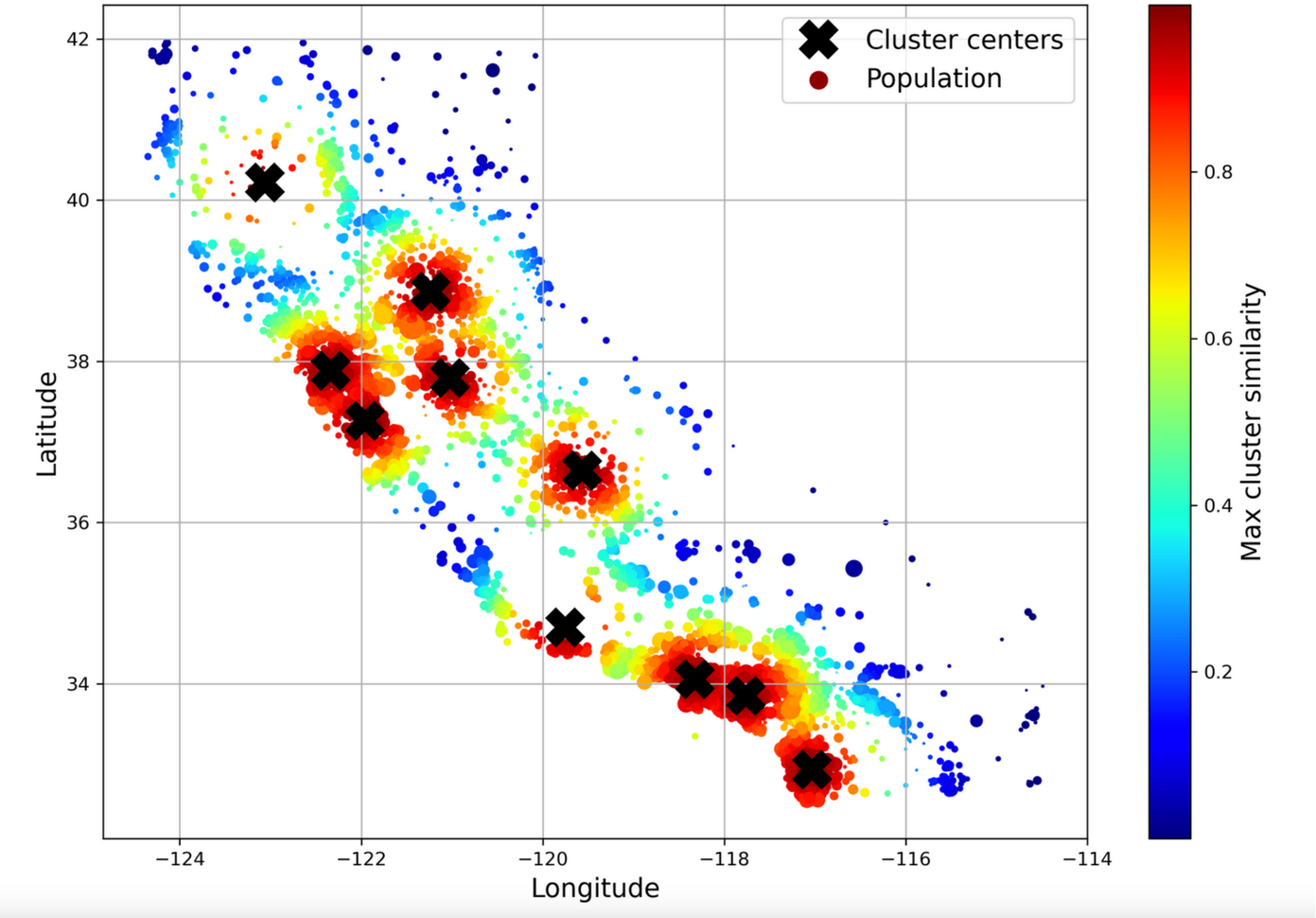
```
cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
similarities = cluster_simil.fit_transform(housing[["latitude", "longitude"]],
                                           sample_weight=housing_labels)
```

```
similarities[:3].round(2)
```

```
array([[0.  , 0.14, 0.  , 0.  , 0.  , 0.08, 0.  , 0.99, 0.  , 0.6 ],
       [0.63, 0.  , 0.99, 0.  , 0.  , 0.  , 0.04, 0.  , 0.11, 0.  ],
       [0.  , 0.29, 0.  , 0.  , 0.01, 0.44, 0.  , 0.7 , 0.  , 0.3 ]])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Cluster found by K-Means



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Transformation Pipelines

- The Pipeline class helps with sequences of transformations

```
from sklearn.pipeline import Pipeline

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Pipelines

- If you don’t want to bother naming the transformers
  - Use the **make_pipeline()** function
  - It just takes transformers as positional arguments
  - It creates a Pipeline using the names of the transformers’ classes, in lower case
    - Without underscores (e.g., “simpleimputer”)

Pipeline example

```
from sklearn.pipeline import make_pipeline

num_pipeline = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
```

```
from sklearn import set_config

set_config(display='diagram')

num_pipeline
```

```
Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median')),
                ('standardscaler', StandardScaler())])
```

```
housing_num_prepared = num_pipeline.fit_transform(housing_num)
housing_num_prepared[:2].round(2)
```

```
array([[-1.42,  1.01,  1.86,  0.31,  1.37,  0.14,  1.39, -0.94],
       [ 0.6 , -0.7 ,  0.91, -0.31, -0.44, -0.69, -0.37,  1.17]])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Pipeline example

- The pipeline exposes the same methods as the final estimator

```
from sklearn.pipeline import Pipeline

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Viewing the transformation

```
housing_num_prepared = num_pipeline.fit_transform(housing_num)
housing_num_prepared[:2].round(2)
```

```
array([[-1.42,  1.01,  1.86,  0.31,  1.37,  0.14,  1.39, -0.94],
       [ 0.6 , -0.7 ,  0.91, -0.31, -0.44, -0.69, -0.37,  1.17]])
```

- Using the pipeline's get_feature_names_out() method:

```
df_housing_num_prepared = pd.DataFrame(
    housing_num_prepared, columns=num_pipeline.get_feature_names_out(),
    index=housing_num.index)
```

```
df_housing_num_prepared.head(2)  # extra code
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
13096	-1.423037	1.013606	1.861119	0.311912	1.368167	0.137460	1.394812	-0.936491
14973	0.596394	-0.702103	0.907630	-0.308620	-0.435925	-0.693771	-0.373485	1.171942

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Pipelines

- Pipelines support indexing
  - **pipeline[1]** returns the second estimator in the pipeline
  - **pipeline[:-1]** returns a Pipeline object containing all but the last estimator
  - You can also access the estimators via the steps attribute
    - A list of name/estimator pairs
    - Via the name_steps dictionary attribute
      - Maps the names to the estimators
    - For example **num_pipeline["simpleimputer"]** returns the estimator named "simpleimputer"

Handling all columns

- A single transformer capable of handling all columns
  - Applying the appropriate transformations to each column
    - Use a **ColumnTransformer**
- Example
  - A ColumnTransformer will apply
    - **num_pipeline** —> numerical attributes
    - **cat_pipeline** —> categorical attribute

ColumnTransformer

```
from sklearn.compose import ColumnTransformer

num_attribs = ["longitude", "latitude", "housing_median_age", "total_rooms",
               "total_bedrooms", "population", "households", "median_income"]
cat_attribs = ["ocean_proximity"]

cat_pipeline = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OneHotEncoder(handle_unknown="ignore"))

preprocessing = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", cat_pipeline, cat_attribs),
])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

make_column_selector()

- Scikit-Learn provides a make_column_selector() function
  - Returns a selector function
    - Automatically select all the features of a given type
    - It can be passed to the ColumnTransformer
      - Instead of column names or indices
- We can use make_column_transformer()
  - Don’t care about naming the transformers,
  - Chooses the names for us

Creating a single pipeline

- Missing values will be imputed
  - Numerical features — > replacing them with the median
  - Categorical features —> placed by the most frequent category
- The categorical feature will be one-hot encoded
- A few ratio features will be computed and added
  - bedrooms_ratio
  - rooms_per_house
  - people_per_house

Creating a single pipeline

- 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
- All numerical features will be standardized

Creating a single pipeline I

```
def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

def ratio_pipeline(name=None):
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio,
                            feature_names_out=[name]),
        StandardScaler())

log_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                             FunctionTransformer(np.log),
                             StandardScaler())
cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
default_num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                                     StandardScaler())
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Creating a single pipeline II

```
preprocessing = ColumnTransformer([
        ("bedrooms_ratio", ratio_pipeline("bedrooms_ratio"),
                           ["total_bedrooms", "total_rooms"]),
        ("rooms_per_house", ratio_pipeline("rooms_per_house"),
                            ["total_rooms", "households"]),
        ("people_per_house", ratio_pipeline("people_per_house"),
                             ["population", "households"]),
        ("log", log_pipeline, ["total_bedrooms", "total_rooms",
                               "population", "households", "median_income"]),
        ("geo", cluster_simil, ["latitude", "longitude"]),
        ("cat", cat_pipeline, make_column_selector(dtype_include=object)),
    ],
    remainder=default_num_pipeline)  # one column remaining: housing_median_age
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Running the pipeline

```
housing_prepared = preprocessing.fit_transform(housing)
housing_prepared.shape
```

```
(16512, 24)
```

```
preprocessing.get_feature_names_out()
```

```
array(['bedrooms_ratio__bedrooms_ratio',
       'rooms_per_house__rooms_per_house',
       'people_per_house__people_per_house', 'log__total_bedrooms',
       'log__total_rooms', 'log__population', 'log__households',
       'log__median_income', 'geo__Cluster 0 similarity',
       'geo__Cluster 1 similarity', 'geo__Cluster 2 similarity',
       'geo__Cluster 3 similarity', 'geo__Cluster 4 similarity',
       'geo__Cluster 5 similarity', 'geo__Cluster 6 similarity',
       'geo__Cluster 7 similarity', 'geo__Cluster 8 similarity',
       'geo__Cluster 9 similarity', 'cat__ocean_proximity_<1H OCEAN',
       'cat__ocean_proximity_INLAND', 'cat__ocean_proximity_ISLAND',
       'cat__ocean_proximity_NEAR BAY', 'cat__ocean_proximity_NEAR OCEAN',
       'remainder__housing_median_age'], dtype=object)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Until now

- Framed the problem
- Explored the data
- Sampled a training set and a test set
- Wrote a preprocessing pipeline
  - Automatically clean up and prepare your data for ML algorithms

- We are ready to select and train a Machine Learning model !

Select and Train a Model

Training and Evaluating on the Training Set

- We will start
  - Training a very basic Linear Regression model

```
from sklearn.linear_model import LinearRegression

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Testing linear regretion

- Let's try it out on the training set
  - Looking at the first 5 predictions and comparing them to the labels

```
[ ]  housing_predictions = lin_reg.predict(housing)
     housing_predictions[:5].round(-2)  # -2 = rounded to the nearest hundred
```

```
array([243700., 372400., 128800.,  94400., 328300.])
```

Compare against the actual values:

```
housing_labels.iloc[:5].values
```

```
array([458300., 483800., 101700.,  96100., 361800.])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Performance results

```
error_ratios = housing_predictions[:5].round(-2) / housing_labels.iloc[:5].values - 1
print(", ".join([f"{100 * ratio:.1f}%" for ratio in error_ratios]))
```

```
-46.8%, -23.0%, 26.6%, -1.8%, -9.3%
```

```
from sklearn.metrics import mean_squared_error

lin_rmse = mean_squared_error(housing_labels, housing_predictions,
                              squared=False)
lin_rmse
```

```
68687.89176589991
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Poor results — Underfitting problem

- Most districts' **median_housing_values** range
  - Between $120,000 and $265,000
  - A typical prediction error of $68,628 is really not very satisfying
- This is an example of a model underfitting the training data

Poor results — Underfitting problem

- The main ways to fix underfitting
  - Select a more powerful model
  - Reduce the constraints on the model
- This model is not regularized
  - Rules out the last option

Another try

- DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor

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

```
[ ]  housing_predictions = tree_reg.predict(housing)
     tree_rmse = mean_squared_error(housing_labels, housing_predictions,
                                    squared=False)
     tree_rmse
```

```
0.0
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Another try

- DecisionTreeRegressor
  - More likely that the model has badly overfit the data
- We don’t want to touch the test set
  - Until you are ready to launch a model
- Use part of the training set for training
  - And part of it for model validation

Cross-validation

- Use the **train_test_split()** function
  - Split the **training** set into a smaller **training** set and a **validation** set
  - Train your models against the smaller **training** set
  - Evaluate them against the **validation** set
- Use Scikit-Learn's K-fold cross-validation feature

```
[ ]  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)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Cross-validation results

- The result is an array containing the 10 evaluation scores

```
>>> 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
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Overfitting problem

- Computing the same metric for the Linear Regression model
  - Mean RMSE = 69,858
  - Standard deviation = 4,182
  - Decision Tree model seems to perform very slightly better
    - Due to severe overfitting
      - Training error is low —> Zero
      - Validation error is high

Yet another try

- RandomForestRegressor
  - Random Forests work by training many Decision Trees
    - On random subsets of the features
      - Averaging out their predictions
  - Capable of boosting the performance of the underlying model

```
[ ]  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)
```

Cross-validation results

- The result is an array containing the 10 evaluation scores

```
>>> 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
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

There’s still quite a lot of overfitting going on

- Training a RandomForest
  - Measuring the RMSE on the training set —> 17,474
- Possible solutions
  - Simplify the model
  - Constrain the model —> ex. regularize it
  - Get a lot more training data

There’s still quite a lot of overfitting going on

- Before you dive much deeper into Random Forests
  - Try out many other models
    - From various categories of Machine Learning algorithms
    - Without spending too much time tweaking the hyperparameters
  - The goal is to shortlist a few promising models
    - Two to five

Fine-Tune Your Model

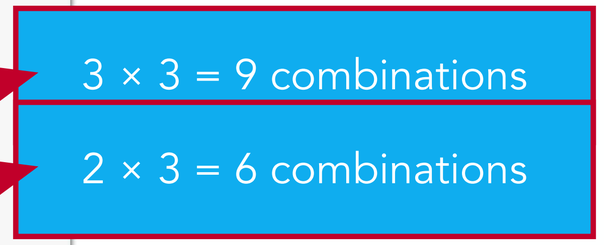
- Basic option
  - Fiddle with the hyperparameters manually
    - Until you find a great combination of hyperparameter values
  - This would be very tedious work
    - May not have time to explore many combinations

Grid Search

- Use Scikit-Learn's GridSearchCV class to search for us
- Specify
  - Which hyperparameters we want it to be tested
  - What values to try out
- It will use cross-validation to evaluate
  - All the possible combinations of hyperparameter values

GridSearch example

```
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)
```



- Full list of hyperparameters available for running

```
print(str(full_pipeline.get_params().keys())[:1000] + "...")
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

The best model

```
grid_search.best_params_
```

```
{'preprocessing__geo__n_clusters': 15, 'random_forest__max_features': 6}
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

- In this example, the best model is obtained by setting
  - **n_clusters** to 15
  - **max_features** to 8

- Since 15 is the maximum value that was evaluated for **n_clusters**
  - We should probably try searching again with higher values
    - The score may continue to improve

The best estimator

- The best estimator can be accessed using
  - **grid_search.best_estimator_**
- If GridSearchCV is initialized with refit=True (default)
  - Once it finds the best estimator using cross-validation
    - It retrains it on the whole training set
- The evaluation scores are available using
  - grid_search.cv_results_
    - Wrap it in a DataFrame

Visualizing GridSearch results

	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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Randomized Search

- The grid search approach is fine when you are exploring relatively few combinations
- RandomizedSearchCV is often preferable
  - Hyperparameter search space is large
- It evaluates a fixed number of combinations
  - Selecting a random value for each hyperparameter at every iteration

Randomized Search

- Several benefits
  - For continuous hyperparameters
    - Or discrete with many possible values
      - With 1,000 iterations
        - It will explore 1,000 different values for each of these hyperparameters
  - When a hyperparameter does not actually make much difference
    - It will not make any difference.

Randomized Search

- For each hyperparameter must provide
  - A list of possible values
  - A probability distribution

```
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)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Randomized Search — results

	n_clusters	max_features	split0	split1	split2	mean_test_rmse
1	45	9	41287	42150	42627	42021
8	32	7	41690	42542	43224	42485
0	41	16	42223	42959	43321	42834
5	42	4	41818	43094	43817	42910
2	23	8	42264	42996	43830	43030

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Ensemble Methods

- Another way to fine-tune your system
  - Try to combine the models that perform best
- The group will often perform better than the best individual model
  - Especially if the individual models make very different types of errors

Analyze the Best Models and Their Errors

- RandomForestRegressor can indicate the relative importance of each attribute for making accurate predictions

```
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, 0.04, 0.01, 0.  ,
       0.01, 0.01, 0.01, 0.01, 0.01, 0.  , 0.01, 0.01, 0.01, 0.  , 0.01,
       0.01, 0.01, 0.01, 0.01, 0.  , 0.  , 0.02, 0.01, 0.01, 0.01, 0.02,
       0.01, 0.  , 0.02, 0.03, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01,
       0.01, 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01, 0.  , 0.07,
       0.  , 0.  , 0.  , 0.01])
```

```
sorted(zip(feature_importances,
           final_model["preprocessing"].get_feature_names_out()),
           reverse=True)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Most important features — RFR

```
[(0.18694559869103852, 'log__median_income'),
 (0.0748194905715524, 'cat__ocean_proximity_INLAND'),
 (0.06926417748515576, 'bedrooms_ratio__bedrooms_ratio'),
 (0.05446998753775219, 'rooms_per_house__rooms_per_house'),
 (0.05262301809680712, 'people_per_house__people_per_house'),
 (0.03819415873915732, 'geo__Cluster 0 similarity'),
 (0.02879263999929514, 'geo__Cluster 28 similarity'),
 (0.023530192521380392, 'geo__Cluster 24 similarity'),
 (0.020544786346378206, 'geo__Cluster 27 similarity'),
 (0.019873052631077512, 'geo__Cluster 43 similarity'),
 (0.018597511022930273, 'geo__Cluster 34 similarity'),
 (0.017409085415656868, 'geo__Cluster 37 similarity'),
 (0.015546519677632162, 'geo__Cluster 20 similarity'),
 (0.014230331127504292, 'geo__Cluster 17 similarity'),
 (0.0141032216204026, 'geo__Cluster 39 similarity'),
 (0.014065768027447325, 'geo__Cluster 9 similarity'),
 (0.01354220782825315, 'geo__Cluster 4 similarity'),
 (0.01348963625822907, 'geo__Cluster 3 similarity'),
 (0.01338319626383868, 'geo__Cluster 38 similarity'),
 (0.012240533790212824, 'geo__Cluster 31 similarity'),
 (0.012089046542256785, 'geo__Cluster 7 similarity'),
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Dropping less useful features

- Apparently only one ocean_proximity category is really useful
  - IN_LAND
  - We could try dropping the others
- The **sklearn.feature_selection.SelectFromModel** transformer
  - Automatically drop the least useful features for you
  - When you fit it
    - It trains a model —> typically a random forest
    - It looks at its **feature_importances_ attribute**
    - Selects the most useful features
    - It just drops the other features
      - When we call **transform()**

Learning from specific errors

- Looking at the specific errors that our system makes
  - Try to understand why it makes them
  - What could fix the problem
    - Adding extra features
    - Getting rid of uninformative ones, cleaning up outliers, etc.

Testing the model with specific categories

- It is also important to ensure that our model
  - Performs well on all categories of districts
    - Rural or urban
    - Rich or poor
    - North or South
    - Minority or not
- Creating subsets of the validation set for each category

Evaluate Your System on the Test Set

- After ensuring at least an acceptable performance of our system
  - We are ready to evaluate the final model on the test set

```
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)
```

```
41424.40026462184
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Estimate precision

- Calculate the precision of our estimate
  - We can compute a 95% confidence interval for the generalization error
    - Using **scipy.stats.t.interval()**

```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

```
array([39275.40861216, 43467.27680583])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Hyperparameter tuning

- When we do a lot of hyperparameter tuning
  - The performance will usually be slightly worse than what you measured using cross-validation
  - It is not the case in this example
    - The test RMSE is lower than the validation RMSE
  - When it happens
    - Resist the temptation to tweak the hyperparameters

Present our solution

- Highlighting what you have learned
- What worked and what did not
- What assumptions were made
- The system’s limitations
- Document everything
- Create nice presentations
- Clear visualizations and easy-to-remember statements
  - “the median income is the number one predictor of housing prices”

Some tips — saving stuff

- Every model we experiment
  - Come back easily to any model we want
- The cross-validation scores
  - Maybe the actual predictions on the validation set
    - Allow us to easily compare scores across model types
    - Compare the types of errors they make

Launch, Monitor, and Maintain our System

Deploying your model

- The most basic way
  - Save the best model you trained
  - Transfer the file to your production environment
  - Load it
  - To save the model

```
import joblib

joblib.dump(final_model, "my_california_housing_model.pkl")
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Deploying to production

```
[ ]  import joblib

     # extra code – excluded for conciseness
     from sklearn.cluster import KMeans
     from sklearn.base import BaseEstimator, TransformerMixin
     from sklearn.metrics.pairwise import rbf_kernel

     def column_ratio(X):
         return X[:, [0]] / X[:, [1]]

     #class ClusterSimilarity(BaseEstimator, TransformerMixin):
     #    [...]

     final_model_reloaded = joblib.load("my_california_housing_model.pkl")

     new_data = housing.iloc[:5]  # pretend these are new districts
     predictions = final_model_reloaded.predict(new_data)
```

```
[ ]  predictions

     array([442737.15, 457566.06, 105965.  ,  98462.  , 332992.01])
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Monitoring the model

- Write monitoring code to check the system's live performance
  - At regular intervals and trigger alerts when it drops
- System performance may drop
  - Very quickly if a component breaks in your infrastructure
  - Very slowly
    - Can easily go unnoticed for a long time
      - Quite common because of model rot

Monitoring the model

- How can we monitor our system
  - Can be inferred from downstream metrics
  - Example
    - The model is part of a recommender system

Monitoring system

- Some times we might need human analysis to determine the model’s performance
  - Example
    - Suppose we've trained an image classification model
    - Detect several product defects on a production line
    - One solution is to send to human raters a sample
      - All the pictures that the model classified
        - Especially pictures that the model wasn’t so sure about
- Develop all the relevant processes to define what to do in case of failures
  - How to prepare for them

Update the dataset and retrain your model

- Automate the whole process as much as possible
  - Collect fresh data regularly and label it
  - Write a script to train the model and fine-tune the hyperparameters automatically
    - The frequency will depend each problem
  - Write a script to
    - Evaluate the new model and the previous model on the updated test set
    - Deploy the model to production
      - If the performance has not decreased
    - Test the performance of your model on various subsets of the test set
      - Poor or rich districts, rural or urban districts, etc

Input data quality

- Make sure we evaluate the model's input data quality
  - Sometimes performance will degrade slightly because of a poor-quality signal
    - A malfunctioning sensor sending random values
    - Another team's output becoming stale
    - Example
      - Trigger an alert if more and more inputs are missing a featurel
      - If its mean or standard deviation drifts too far from the training set
      - A categorical feature starts containing new categories.

Backups

- Make sure we
  - Keep backups of every model we create
  - Have the process and tools in place
    - Roll back to a previous model quickly
      - In case the new model starts failing badly for some reason
- Backups also makes it possible to easily compare new models with previous ones
- Keep backups of every version of your datasets
  - Roll back to a previous dataset if the new one ever gets corrupted
  - Allows you to evaluate any model against any previous dataset

Summary

Frame the Problem and Look at the Big Picture

- Define the objective in business terms.
- How will your solution be used?
- What are the current solutions/workarounds (if any)?
- How should you frame this problem (supervised/unsupervised, online/offline, etc.)?
- How should performance be measured?
- Is the performance measure aligned with the business objective?

Frame the Problem and Look at the Big Picture

- What would be the minimum performance needed to reach the business objective?
- What are comparable problems? Can you reuse experience or tools?
- Is human expertise available?
- How would you solve the problem manually?
- List the assumptions you (or others) have made so far
- Verify assumptions if possible

Get the Data

- List the data you need and how much you need
- Find and document where you can get that data
- Check how much space it will take
- Check legal obligations, and get authorization if necessary
- Get access authorizations
- Create a workspace (with enough storage space)

Get the Data

- Convert the data to a format you can easily manipulate
  - Without changing the data itself
- Ensure sensitive information is deleted or protected (e.g., anonymized)
- Check the size and type of data
  - Time series, sample, geographical, etc.
- Sample a test set
  - Put it aside, and never look at it

Explore the Data

- Create a copy of the data for exploration (sampling it down to a manageable size if necessary).
- Create a Jupyter notebook to keep a record of your data exploration.
- Study each attribute and its characteristics:
  - Name
  - Type (categorical, int/float, bounded/unbounded, text, structured, etc.)
  - % of missing values
  - Noisiness and type of noise (stochastic, outliers, rounding errors, etc.)
  - Usefulness for the task
  - Type of distribution (Gaussian, uniform, logarithmic, etc.)

Explore the Data

- For supervised learning tasks, identify the target attribute(s).
- Visualize the data.
- Study the correlations between attributes.
- Study how you would solve the problem manually.
- Identify the promising transformations you may want to apply.
- Identify extra data that would be useful (go back to “Get the Data”).
- Document what you have learned

Prepare the Data

- Work on copies of the data (keep the original dataset intact)
- Write functions for all data transformations you apply, for five reasons
  - So you can easily prepare the data the next time you get a fresh dataset
  - So you can apply these transformations in future projects
  - To clean and prepare the test set
  - To clean and prepare new data instances once your solution is live
  - To make it easy to treat your preparation choices as hyperparameters

Prepare the data

- Data cleaning:
  - Fix or remove outliers (optional)
  - Fill in missing values (e.g., with zero, mean, median…) or drop their rows (or columns)
- Feature selection (optional)
  - Drop the attributes that provide no useful information for the task
- Feature engineering, where appropriate
  - Discretize continuous features
  - Decompose features (e.g., categorical, date/time, etc.)
  - Add promising transformations of features (e.g., log(x), sqrt(x), x2, etc.)
  - Aggregate features into promising new features.
- Feature scaling
  - Standardize or normalize features

Notes

- If the data is huge
  - May want to sample smaller training sets
    - Train many different models in a reasonable time
    - Be aware that this penalizes complex models such as large neural nets or Random Forests
- Once again, try to automate these steps as much as possible

Shortlist Promising Models

- Train many quick-and-dirty models from different categories
  - Linear, naive Bayes, SVM, Random Forest, neural net, etc.
  - Using standard parameters
- Measure and compare their performance
  - For each model
    - Use N-fold cross-validation
      - Compute the mean and standard deviation
        - Performance measure on the N folds
- Analyze the most significant variables for each algorithm

Shortlist Promising Models

- Analyze the types of errors the models make
  - What data would a human have used to avoid these errors?
- Perform a quick round of feature selection and engineering
- Perform one or two more quick iterations of the five previous steps
- Shortlist the top three to five most promising models
  - Preferring models that make different types of errors

Fine-Tune the System

- Fine-tune the hyperparameters using cross-validation:
- Treat your data transformation choices as hyperparameters
  - Especially when you are not sure about them
  - If you’re not sure whether to replace missing values
    - With zeros or with the median value, or to just drop the rows
- Unless there are very few hyperparameter values to explore
  - Prefer random search over grid search
  - If training is very long, you may prefer a Bayesian optimization approach

Fine-Tune the System

- Try Ensemble methods
  - Combining your best models will often produce better performance than running them individually
- Once you are confident about your final model
  - Measure its performance on the test set to estimate the generalization error
- Remember
  - Don’t tweak your model after measuring the generalization error
    - You would just start overfitting the test set

Present your solution

- Document what you have done
- Create a nice presentation
- Make sure you highlight the big picture first
- Explain why your solution achieves the business objective
- Don’t forget to present interesting points you noticed along the way
- Describe what worked and what did not
- List your assumptions and your system’s limitations
- Ensure your key findings are communicated through beautiful visualizations or easy-to-remember statements

Launch

- Get your solution ready for production
  - Plug into production data inputs, write unit tests, etc.
- Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops
- Beware of slow degradation: models tend to "rot" as data evolves
- Measuring performance may require a human pipeline
- Also monitor your inputs' quality
- Retrain your models on a regular basis on fresh data

Artificial Intelligence and Machine Learning for Connected Systems

Class 3 & 4 - End to end project

Naresh Modina