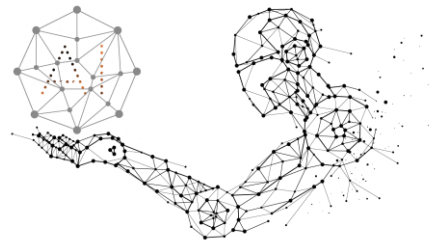
Applied Machine Learning

Chapter 7- End-to-End Machine Learning Project

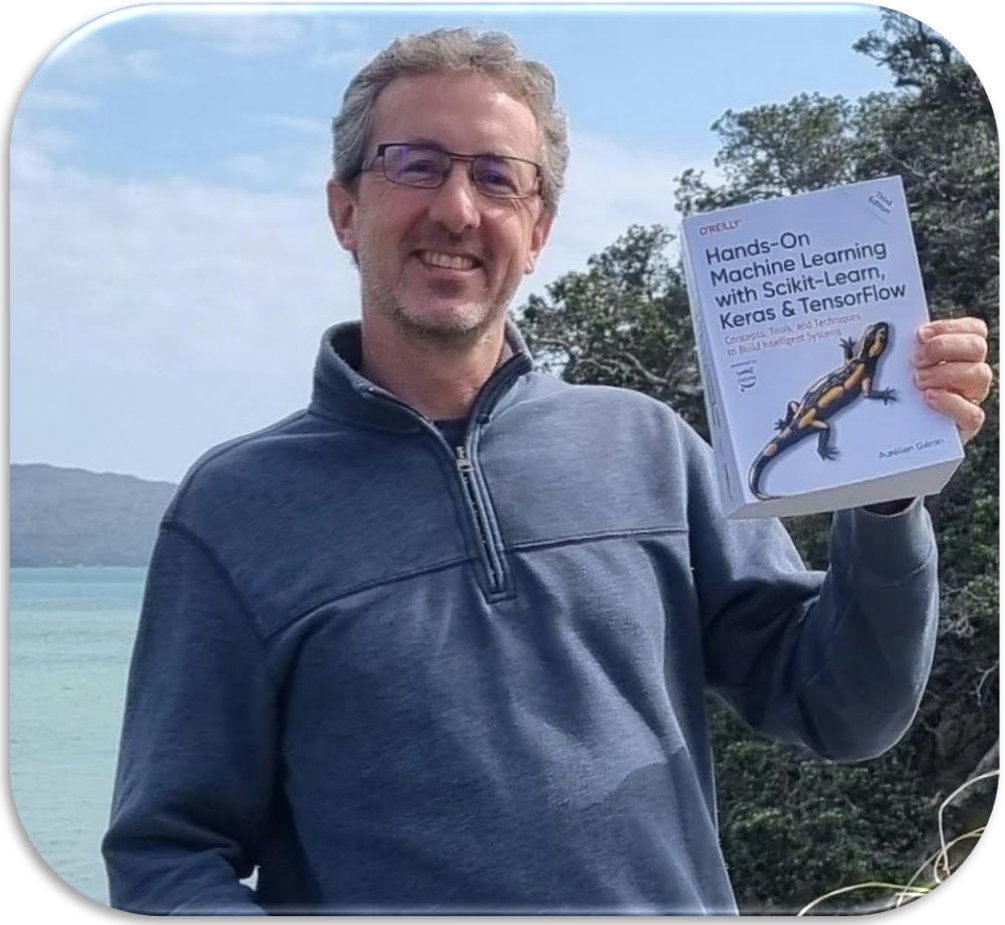
دانشگاه تربیت مدرس
Tarbiat Modares University

Hossein Homaei
Department of Electrical & Computer Engineering



Reference

- A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 3rd ed. O'Reilly Media, 2023.
  - Chapter 2 and Appendix A



All the code examples are available as Jupyter notebooks:
https://github.com/ageron/handson-ml3

Content

- Work through an example project:
  - Predicting house prices
    - Regression problem
    - Dataset: California Housing Prices from StatLib repository
      - This dataset is based on data from the 1990 California census.

Main Steps in ML Projects

1. Look at the big picture
2. Get the data
3. Explore and visualize the data to gain insights
4. Prepare the data for machine learning algorithms
5. Explore many different models, shortlist the best ones, and train the models
6. Fine-tune your models and combine them into a great solution
7. Present your solution
8. Launch, monitor, and maintain your system

- You can adapt this checklist to your needs

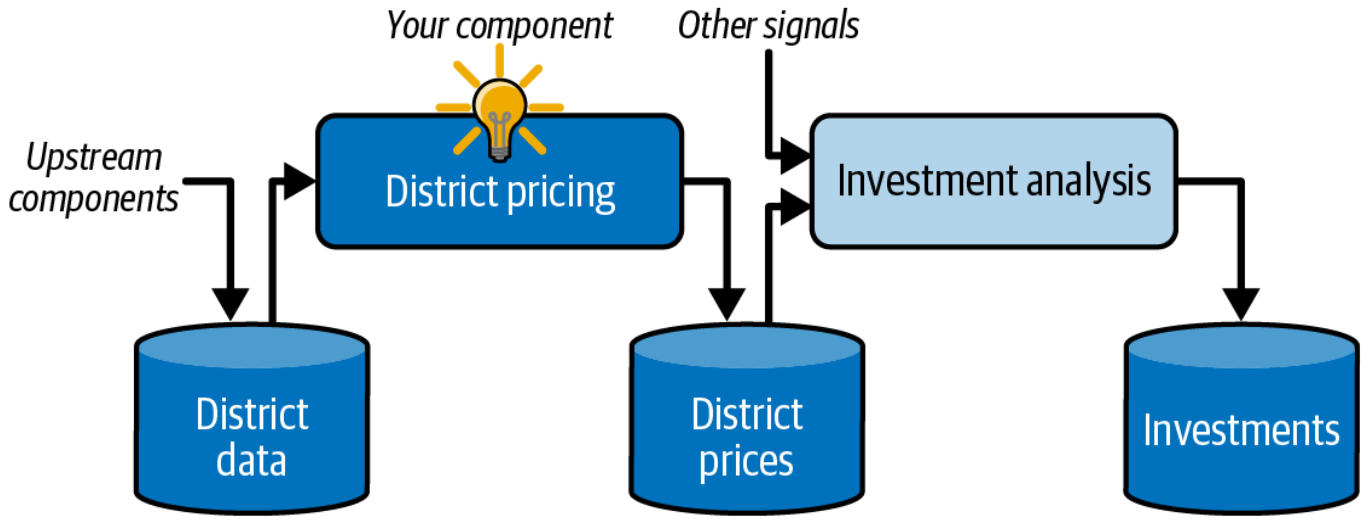
Main Steps in ML Projects

1. **Look at the big picture**
2. Get the data
3. Explore and visualize the data to gain insights
4. Prepare the data for machine learning algorithms
5. Explore many different models, shortlist the best ones, and train the models
6. Fine-tune your models and combine them into a great solution
7. Present your solution
8. Launch, monitor, and maintain your system

Look at the Big Picture

Understanding the Problem

1. What is the business objective?
   - Predict a district's median housing price
2. How will your solution be used?
   - Output will be fed to another machine learning system
     - Is it worth investing in a given area?



Solution Overview

3. What are the current solutions (if any)?
   - Manually estimated by experts using complex rules:
     - Costly and time-consuming
     - Their estimates are not great (30% error)
   - May give you insights on how to solve the problem and measure the performance

4. How should you frame this problem (supervised/unsupervised, online/offline, etc.)?
   - Supervised
     - Training with labeled examples
   - Regression
     - Predict a value
   - Batch learning
     - Do not need to adjust to changing data rapidly
     - Data is small enough to fit in memory

Overview of the Evaluation

5. How should performance be measured?
   - A typical performance measure for regression problems is the Root Mean Square Error (RMSE) cost function (ℓ_2 norm or Euclidean distance):

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

     - X: the matrix containing all the feature values
     - m: number of instances in the dataset
     - $x^{(i)}$: vector of all the feature values of the ith instance in the dataset
     - $y^{(i)}$: the label (the desired output value) for the ith instance
     - h: prediction function, also called a hypothesis.
       - The system get an instance's feature vector $x^{(i)}$ and outputs a predicted value ($h(x^{(i)})$ or $\hat{y}^{(i)}$)
   - Mean Absolute Error (MAE) (ℓ_1 norm or Manhattan distance): $\frac{1}{m}\sum_{i=1}^{m}\left|h(x^{(i)}) - y^{(i)}\right|$

Review of the Assumptions

6. List and verify the assumptions
   - The district prices that your system outputs are going to be fed into a downstream machine learning system.
   - What happens if the downstream system converts the prices into categories (e.g. “cheap”, “medium”, or “expensive”) and then uses those categories instead of the prices themselves?
     - Getting the price perfectly right is not important; your system just needs to get the category right.
     - The classification task, not a regression task.
   - You don’t want to find this out after working on a regression system for months.

Get the Data

Main Steps in ML Projects

1. Look at the big picture
2. **Get the data**
3. Explore and visualize the data to gain insights
4. Prepare the data for machine learning algorithms
5. Explore many different models, shortlist the best ones, and train the models
6. Fine-tune your models and combine them into a great solution
7. Present your solution
8. Launch, monitor, and maintain your system

Data Gathering

- Download available dataset
  - Typically, available in common data stores like relational DB, files, documents or spread across multiple stores.
  - The comma-separated values (CSV) files are very common
  - Be sure that you are authorized to use the data
  - Use `pandas.read_csv()` in Python to get a DataFrame

- Acquire data and Build your own dataset
  - Randomly select representative samples from the population
  - Check the quality, e.g. Balanced or not?
  - Format
  - We do not explore this topic here!

Quick Look at the Data

- Take a quick look at the data structure
  - Looking at the top five rows to get a sense from data
    - `head()` method in Pandas

	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

… Quick Look at the Data

- Get a quick description of the data
  - `info()` method in Pandas
  - Number of rows
  - Each attribute's type

207 districts are missing this feature

```
<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)
```

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

… Quick Look at the Data

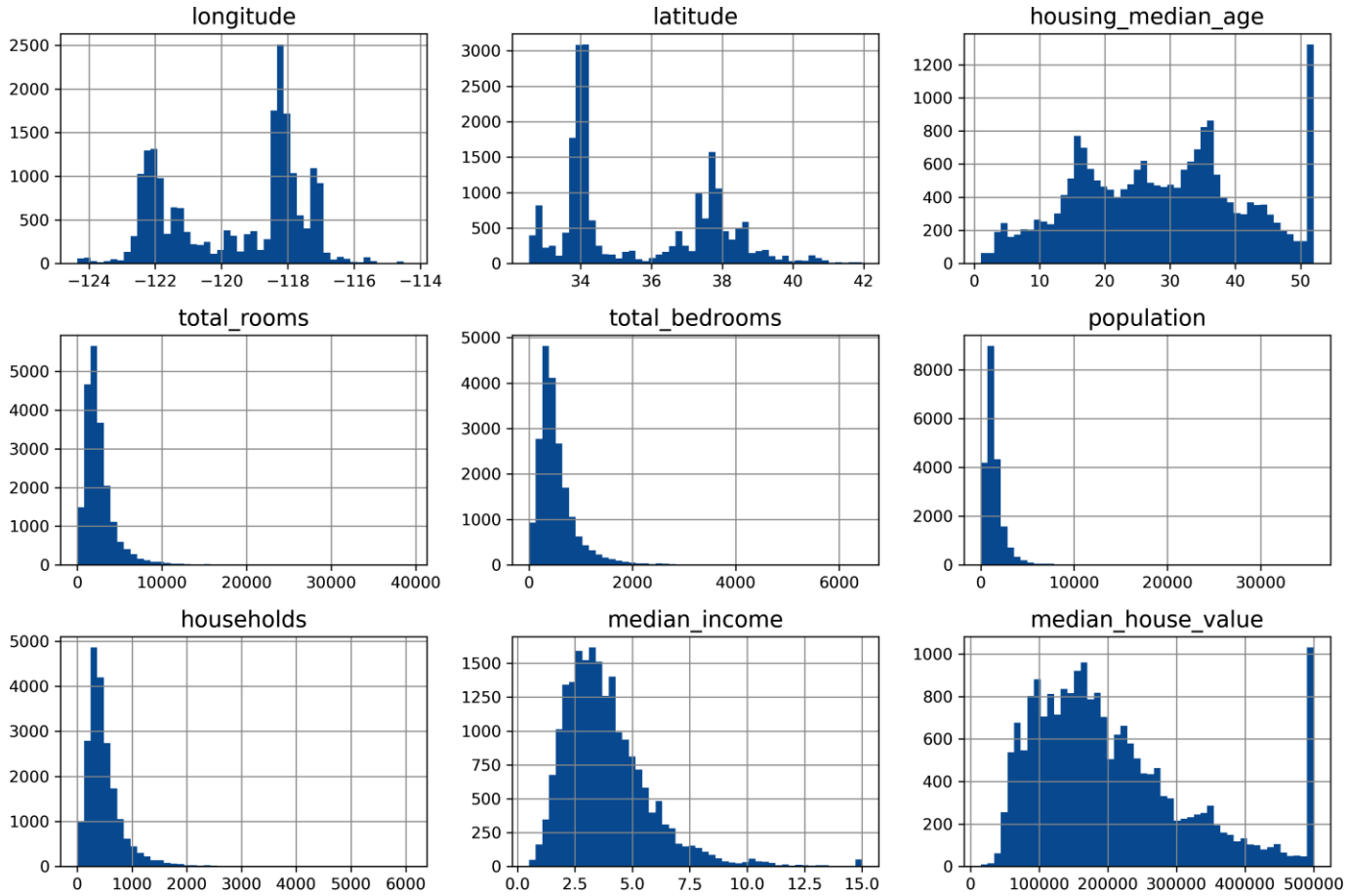
- Summary of the numerical attributes
  - `describe()` method in Pandas

	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

… Quick Look at the Data

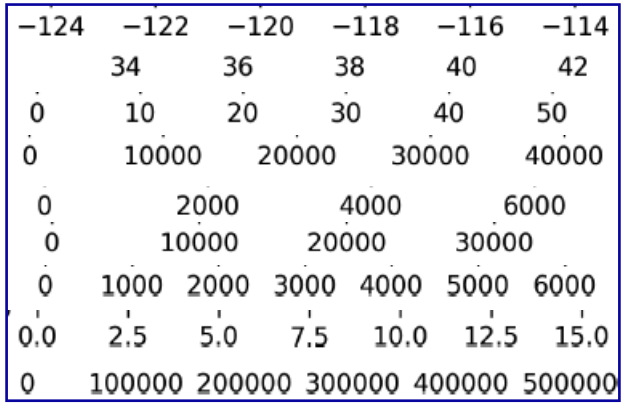
- Plot a histogram for each numerical attribute and try to understand the data

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

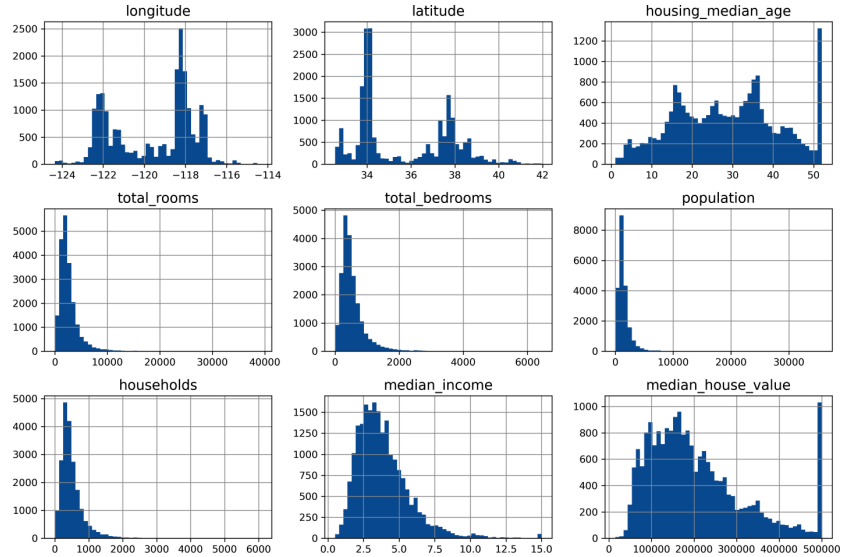


… Quick Look at the Data

- The attributes have very different scales
  - We should use feature scaling techniques
    - Discuss later!

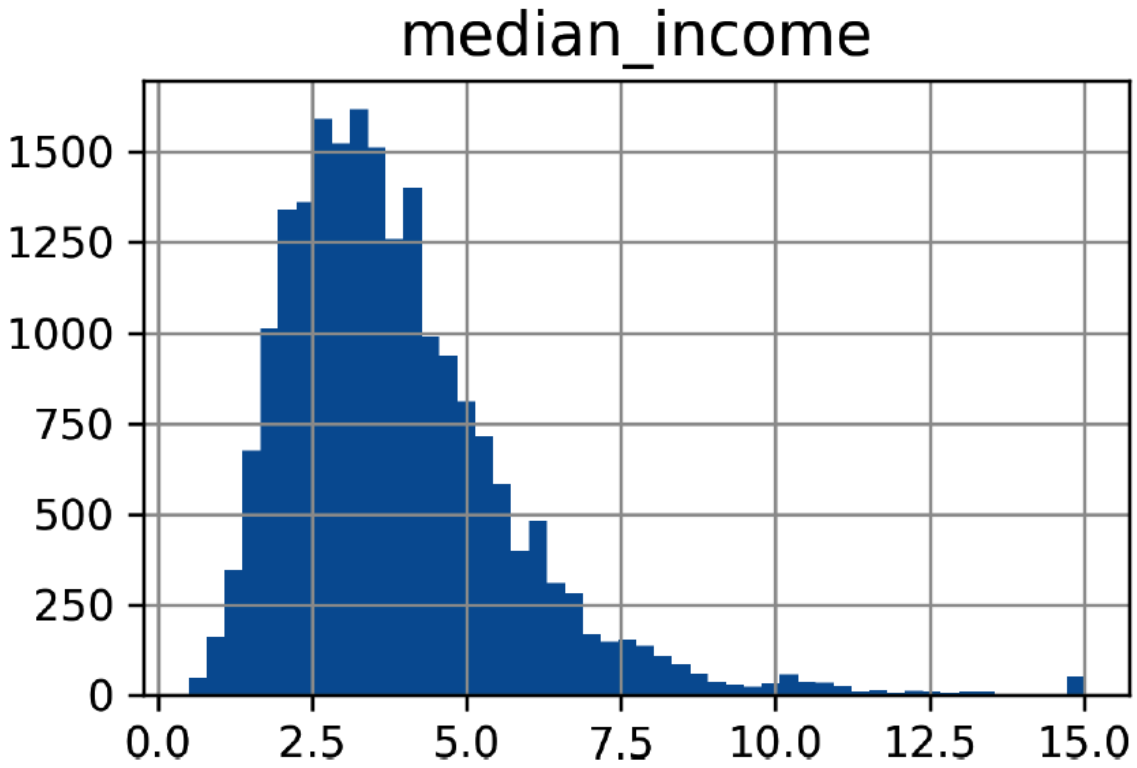


- Many histograms are skewed right
  - This may make it a bit harder for some ML algorithms to detect patterns
    - Try to transform these attributes to have more symmetrical and bell-shaped distributions



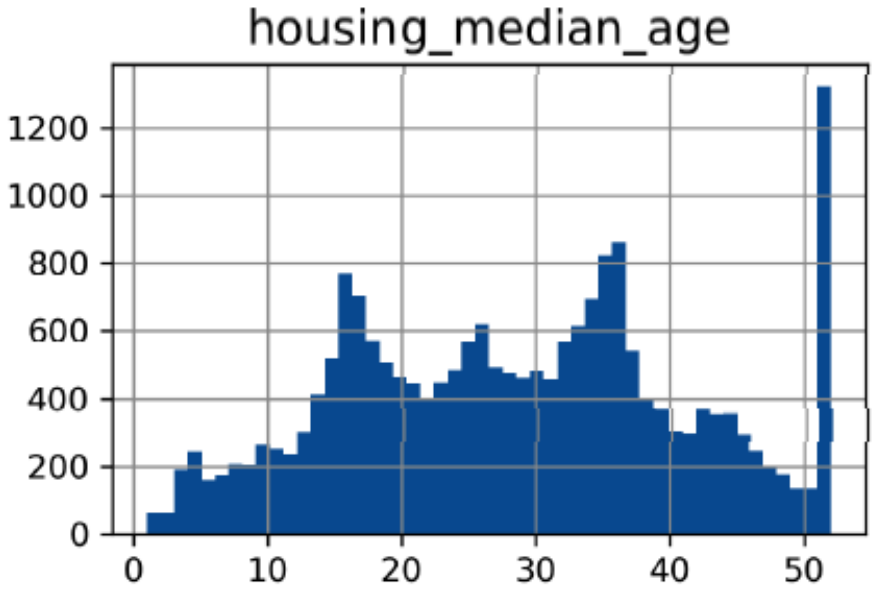
… Quick Look at the Data

- The median income attribute does not look like it is expressed in US dollars.
  - Checking with the team that collected the data
    - They told you that data has been scaled (divide by 10000) and capped at 15 and 0.5 for higher and lower median incomes
  - Working with preprocessed attributes is common in ML, and it is not necessarily a problem, but you should try to understand how the data was computed.

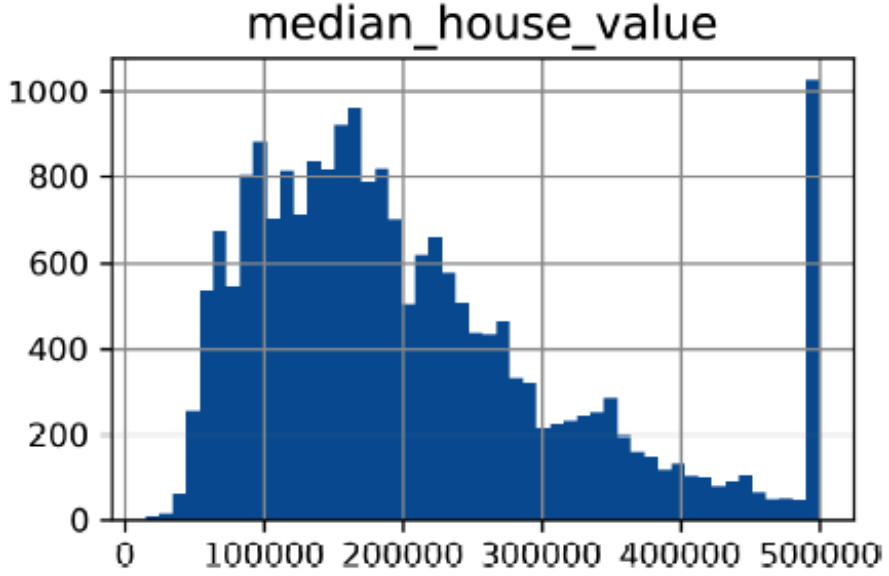


… Quick Look at the Data

- Capped at 50 years
  - Hopefully, no problem!



- Capped at $500,000
  - May be a serious problem since it is your target attribute



… Quick Look at the Data

- Median house value problem
  - The ML algorithm may learn that prices never go beyond the limit
  - Check with your client if this is a problem or not
  - If they tell you that they need precise predictions even beyond $500,000, then you have two options:
    - Collect proper labels for the districts whose labels were capped.
    - Remove those districts from the dataset

Creating the Test Set

- Theoretically simple: pick some instances randomly
  - Typically, 20% of the dataset (less if your dataset is very large)

```
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
```

Works, but not perfect!

… Creating the Test Set

- If you run the program again, it will generate a different test set
  - Over time, you (or your ML algorithms) will see the whole dataset
    - Not good! ➔ data snooping bias
    - When you estimate the generalization error using the test set, your estimate will be too optimistic.
    - Prone to overfitting
  - Solution: Use the same test set every time
    - Save the test set on the first run and then load it in subsequent runs.
    - Set the random number generator's seed: `np.random.seed(42)`
      - It always generates the same shuffled indices
  - Problem: you cannot work with updated dataset
    - We need a stable train/test split even after updating the dataset

… Creating the Test Set

- Solution: use each instance's identifier to decide whether it should go in the test set
  - Assumption: Instances have unique and immutable identifiers
  - Practical implementation:
    - compute a hash of each instance's identifier
    - If the hash value of an instance ≤ 20% of the maximum hash value, put it in the 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")
```

… Creating the Test Set

- Note: If you use the row index as a unique identifier, you need to make sure that new data gets appended to the end of the dataset and that no row ever gets deleted.
  - If it is not possible, then you can try to use the most stable features to build a unique identifier.
    - Example: District's latitude and longitude are guaranteed to be stable

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

… Creating the Test Set

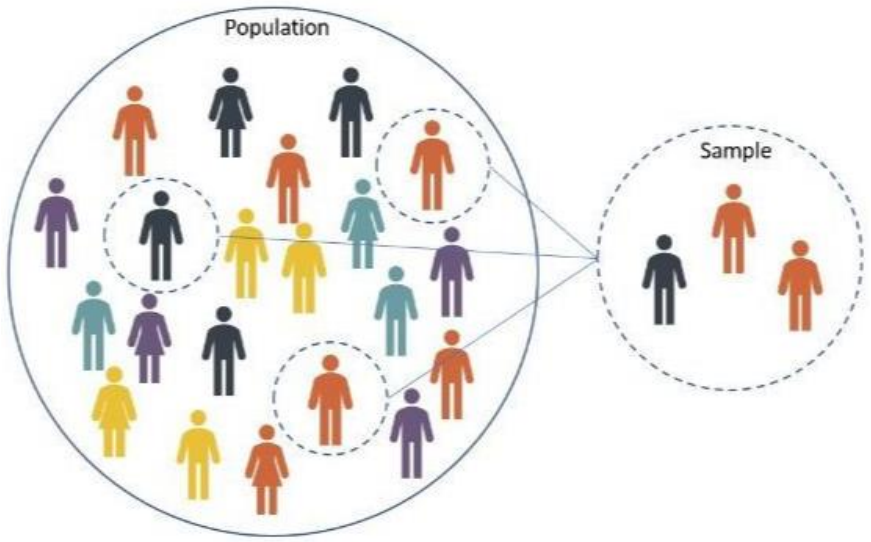
- Built-in scikit-learn function: `train_test_split()`
  - pick some instances randomly

```
from sklearn.model_selection import train_test_split

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

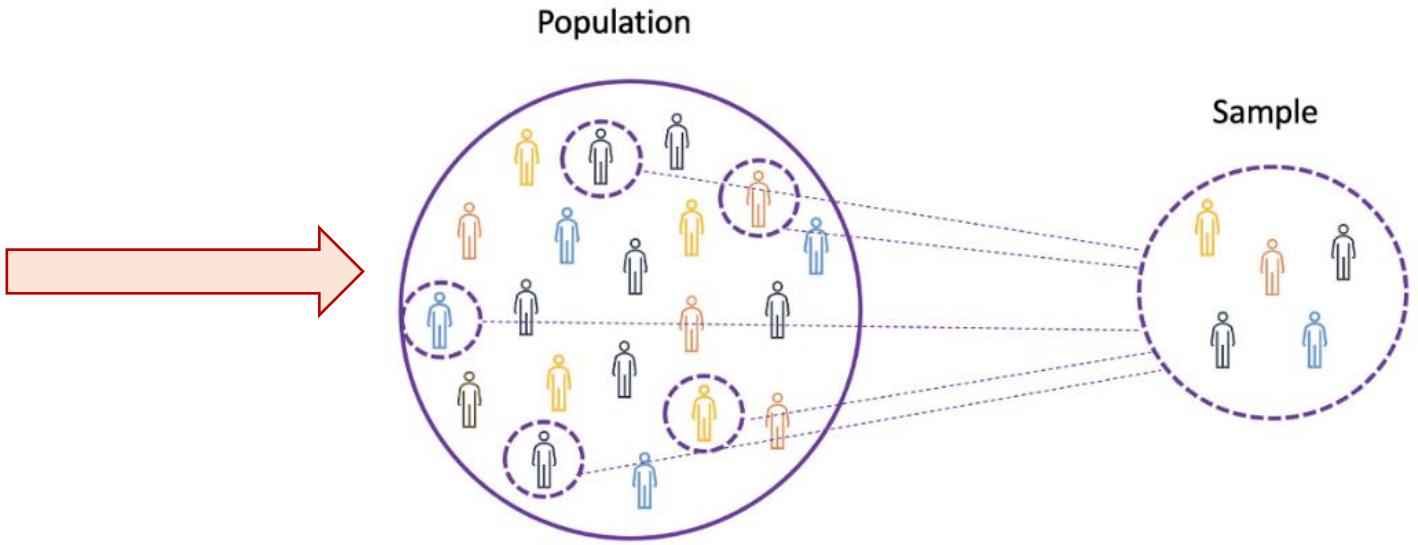
… Creating the Test Set

- Stratified sampling
  - The dataset is divided into homogeneous subgroups called strata
    - The right number of instances is sampled from each stratum to guarantee that the test set is representative of the overall population
    - Prevent sampling bias
    - Example:
      - Select 1000 people to ask them a few questions.
      - Selected people should be representative of the whole population, concerning the questions they want to ask.
      - E.g. 51.1%female and 48.9% male in the US
      - A well-conducted survey in the US would try to maintain this ratio, as the answers may vary across genders
        - 511 females and 489 males

… Creating the Test Set

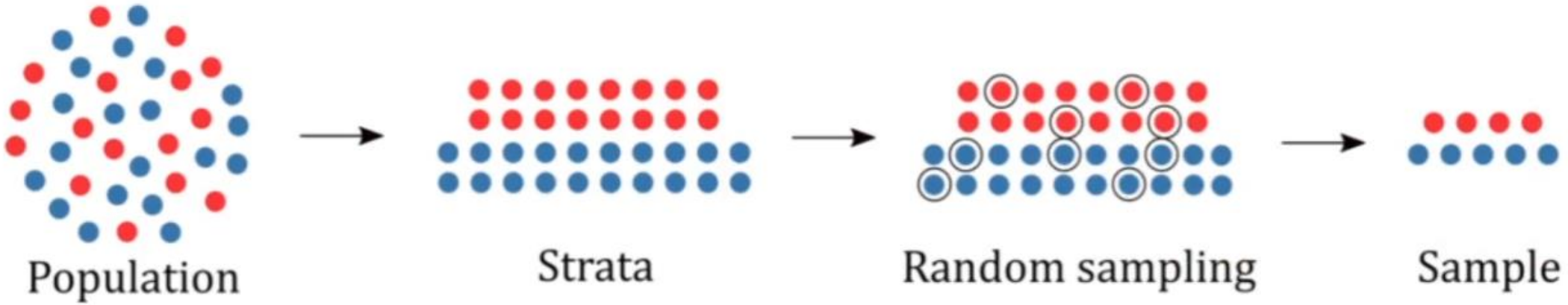


Random sampling



Stratified sampling

… Creating the Test Set



… Creating the Test Set

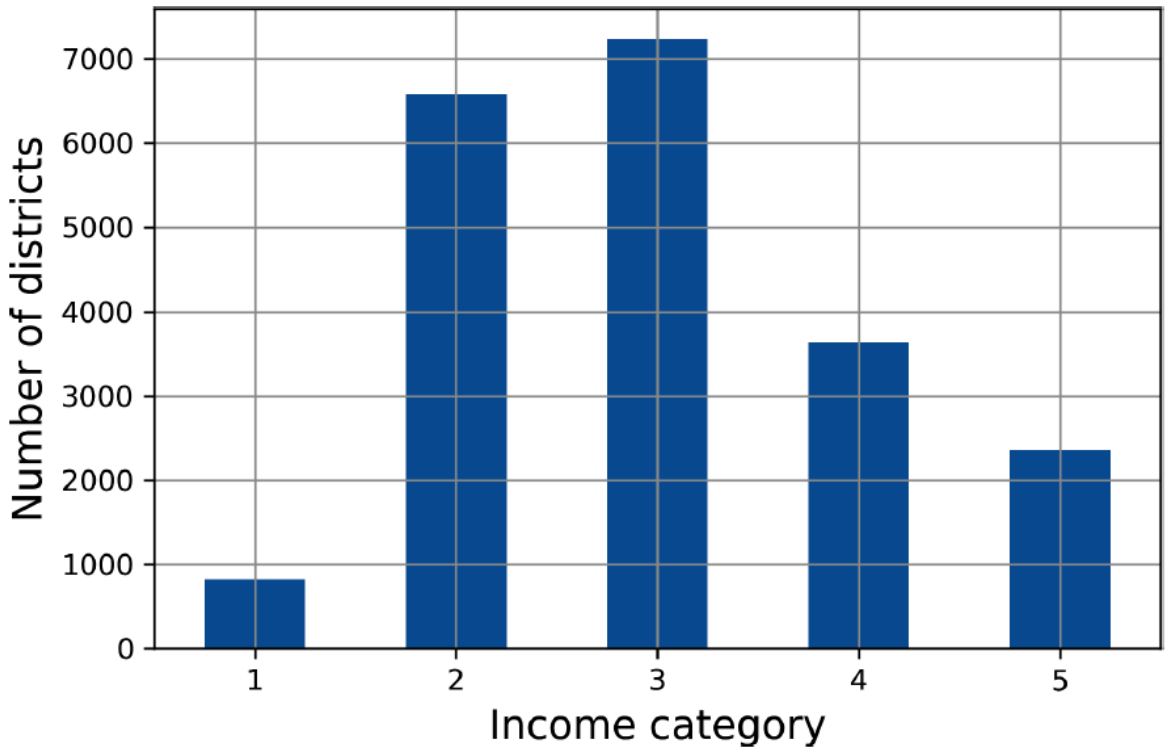
- … Stratified sampling
  - Experts told you that the median income is a very important attribute to predict median housing prices.
  - You should ensure that the test set is representative of the various categories of incomes
  - The median income is a continuous numerical attribute ➔ We need to create an income category attribute
  - Note: You should not have too many strata, and each stratum should be large enough

```
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])
```

… Creating the Test Set

- … Stratified sampling

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



… Creating the Test Set

- … Stratified sampling

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

```
>>> 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
```

Explore & Visualize the Data

Main Steps in ML Projects

1. Look at the big picture
2. Get the data
3. **Explore and visualize the data to gain insights**
4. Prepare the data for machine learning algorithms
5. Explore many different models, shortlist the best ones, and train the models
6. Fine-tune your models and combine them into a great solution
7. Present your solution
8. Launch, monitor, and maintain your system

Notes

- Put the test set aside and explore only the training set
- If the training set is very large, you can sample an exploration set to make manipulations easy and fast during the exploration phase
- It is recommended to work with a copy of the original data so you can revert to it afterwards

Visualizing Geographical Data

- Scatter plot of all the districts to visualize the data

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

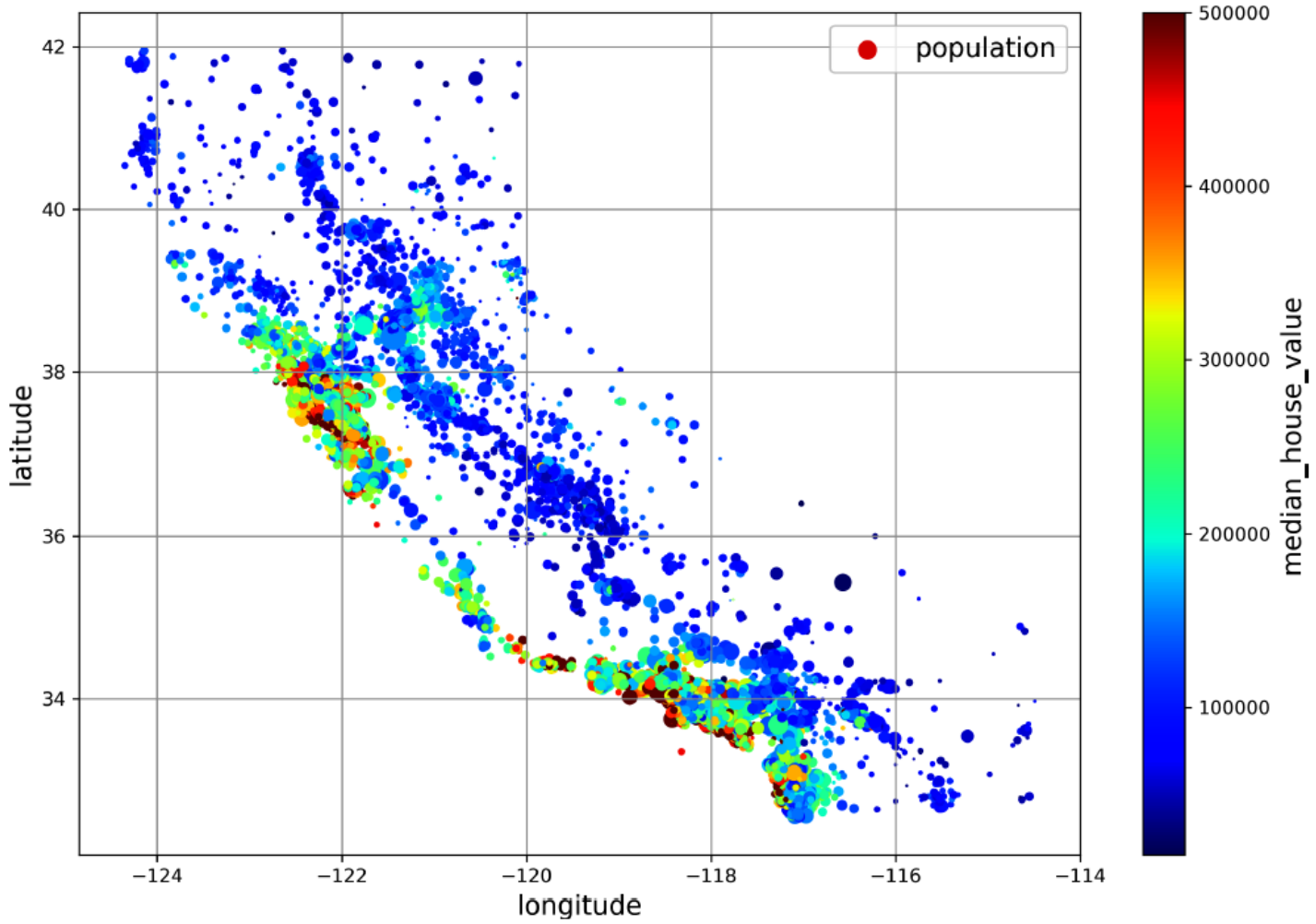
Radius of each circle

Color of each circle

Color map
Default = jet (ranges from blue (low values) to red (high prices))

… Visualizing Geographical Data

- This image tells you that the housing prices are very much related to the location (e.g., close to the ocean) and to the population density



Correlations

- Standard (Pearson) correlation coefficient between every pair of attributes

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

```
>>> 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
```

Strong positive correlation

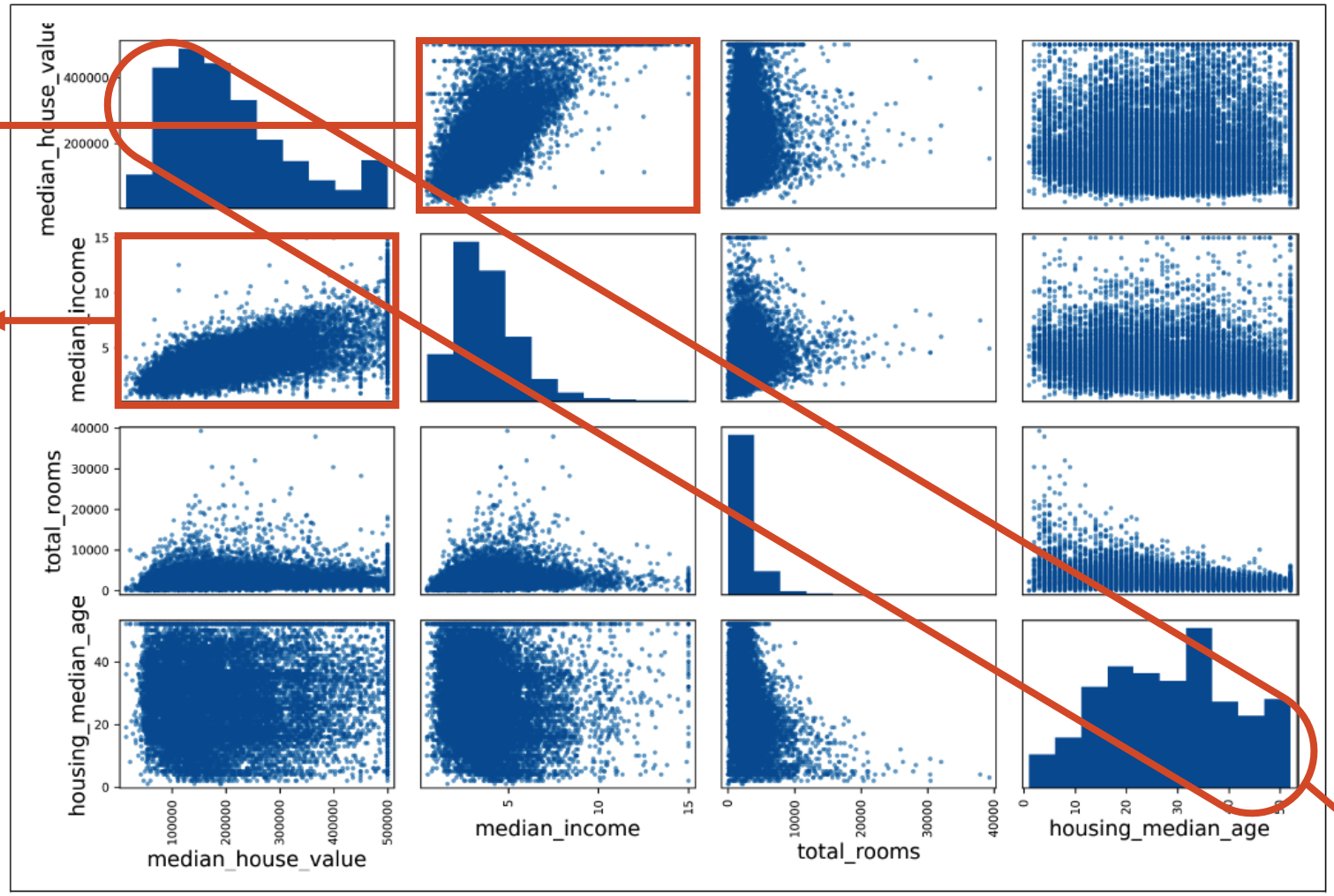
… Correlations

- Alternative method to visually check the correlation: Pandas `scatter_matrix()`
  - There are 11 numerical attributes ➔ We get $11^2 = 121$ plots
    - Not fit on a page
    - So, focus on a few promising attributes that seem most correlated with the target (median housing value)

```
from pandas.plotting import scatter_matrix

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

… Correlations

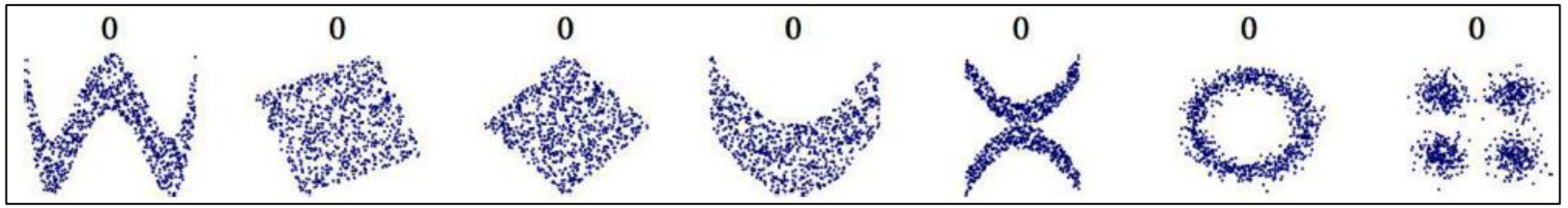
The most promising attribute for prediction



histogram of each numerical attribute's values

… Correlations

- Note: The correlation coefficient only measures linear correlations



Examples of nonlinear relationships

Attribute Combinations

- Create new meaningful features by combining raw attributes
  - For example, the total number of rooms in a district is not very useful if you don't know how many households there are. What you really want is the number of rooms per household.

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

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

```
median_house_value    1.000000
median_income         0.688380
rooms_per_house       0.143663
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
people_per_house     -0.038224
longitude            -0.050859
latitude             -0.139584
bedrooms_ratio       -0.256397
```

Prepare Data for ML Algorithms

Main Steps in ML Projects

1. Look at the big picture
2. Get the data
3. Explore and visualize the data to gain insights
4. **Prepare the data for machine learning algorithms**
5. Explore many different models, shortlist the best ones, and train the models
6. Fine-tune your models and combine them into a great solution
7. Present your solution
8. Launch, monitor, and maintain your system

Notes

- Use the training set
  - Separate the predictors and the labels
  - Work with a copy
    - Note: `drop()` creates a copy

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

Clean the Data

- Most ML algorithms cannot work with missing features
- Fixing methods?
  - Deleting rows
    - Get rid of the corresponding districts
    - Pandas DataFrame's `dropna()` method

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

  - Deleting columns
    - Get rid of the whole attribute
    - Pandas DataFrame's `drop()` method

```
housing.drop("total_bedrooms", axis=1)
```

  - Imputation: Set the missing values to some value
    - Zero, mean, median, most frequent,…
    - Pandas DataFrame's `fillna()` method

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

… Clean the Data

- Use Scikit-Learn's transformers (imputers)
  - Benefit: make it possible to simply reuse for any feature and other updated datasets, validation set, and test set.
  - Usage
    - Create a SimpleImputer instance
    - Specify your strategy (e.g. median)
    - Fit the imputer instance to the training data
      - It may require preprocessing
        - Example: Since median can only be computed on numerical attributes, create a copy of the data with only the numerical attributes.

```
from sklearn.impute import SimpleImputer

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

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

⇒ Trained imputer

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

    - Transform the training set by replacing missing values with the learned medians (based on the trained median values)

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

… Clean the Data

- … Use Scikit-Learn's imputer
  - … Usage
    - Other strategies for SimpleImputer
      - `strategy="mean“`
      - `strategy="most_frequent”`
      - `strategy="constant", fill_value=…`

      } Support non-numerical data
    - Other Imputers (for numerical features only)
      - `KNNImputer`
        - Replaces each missing value with the mean of the k-nearest neighbors' values for that feature. (The distance is based on all the available features)
      - `IterativeImputer`
        - Trains a regression model per feature to predict the missing values based on all the other available features. It then trains the model again on the updated data and repeats the process several times, improving the models and the replacement values at each iteration.

… Clean the Data

- … Use Scikit-Learn's imputer
  - Transformers (often) output NumPy arrays
    - X has neither column names nor index
      - We should wrap X in a DataFrame and recover the column names and index from

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

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

Handling Categorical Attributes

- Most ML algorithms prefer to work with numbers
- Convert categories from text to numbers ➔ Encode
  - Example: Use Scikit-Learn's OrdinalEncoder

…Handling Categorical Attributes

- Scikit-Learn's OrdinalEncoder
  - Specify the column that you want to be encoded
  - Create an instance of OrdinalEncoder
  - Fit and transform data

```
housing_cat = housing[["ocean_proximity"]]
```

```
from sklearn.preprocessing import OrdinalEncoder

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

  - Results

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

housing_cat

ocean_proximity
NEAR BAY
<1H OCEAN
INLAND
INLAND
NEAR OCEAN
INLAND
<1H OCEAN
NEAR BAY

housing_cat_encoded

```
array([[3.],
       [0.],
       [1.],
       [1.],
       [4.],
       [1.],
       [0.],
       [3.]])
```

… Handling Categorical Attributes

- Problem: ML algorithms assume that two nearby values are more similar than two distant values.
  - Fine in some cases
    - e.g., for ordered categories such as “bad”, “average”, “good”, and “excellent”
  - Generally, not
  - Solution: One-hot encoding
    - Create one binary attribute per category
      - One attribute equal to 1 when the category is "<1H OCEAN" (and 0 otherwise),
      - Another attribute equal to 1 when the category is "INLAND" (and 0 otherwise),
      - …
    - Only one attribute will be equal to 1 (hot), while the others will be 0 (cold)
    - The new attributes are sometimes called dummy attributes

… Handling Categorical Attributes

- One-hot encoding using Scikit-Learn

```
from sklearn.preprocessing import OneHotEncoder

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

  - By default, the output of a `OneHotEncoder` is a SciPy sparse matrix
    - Very efficient representation for matrices that contain mostly zeros
    - It only stores the nonzero values and their positions
    - If you want to convert it to a NumPy array,
      - Call the `toarray()`, OR
      - Set `sparse=False` when creating the OneHotEncoder

```
housing_cat_1hot.toarray()
```

```
cat_encoder = OneHotEncoder(sparse=False)
```

… Handling Categorical Attributes

- One-hot encoder input/output features

```
>>> cat_encoder.feature_names_in_
array(['ocean_proximity'], dtype=object)
>>> cat_encoder.get_feature_names_out()
array(['ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAND',
       'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
       'ocean_proximity_NEAR OCEAN'], dtype=object)
```

Feature Scaling

- Problem description
  - Most of the ML algorithms don't perform well when the input numerical attributes have very different scales.
  - The models will be biased toward ignoring the features having small ranges
  - Example: housing data
    - $6 <$ total number of rooms < 39320
    - $0 <$ median incomes < 15
    - Without any scaling, most models will be biased toward ignoring the median income and focusing more on the number of rooms

… Feature Scaling

- Methods
  - Min-max scaling (normalization)
  - Standardization

… Feature Scaling

- … Methods
  - Min-max scaling (normalization)
    - The values are shifted and rescaled to end up ranging from 0 to 1
    - Subtracting the min value and dividing by the difference between the min and the max
    - Affected by outlier
      - Example: Suppose a district has a median income 100 (by mistake). Min-max scaling maps this outlier to 1 and crushes all the other values down to 0–0.15

$$x' = a + \frac{(x - \min(X))(b - a)}{\max(X) - \min(X)}$$

$$\min(X) \leq x \leq \max(X) \Longrightarrow a \leq x' \leq b$$

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)}$$

$$\min(X) \leq x \leq \max(X) \Longrightarrow 0 \leq x' \leq 1$$

… Feature Scaling

- … Methods
  - Standardization
    - Subtracting the mean value and dividing by the standard deviation
    - Standardized values have a zero mean and standard deviation equal to 1
    - Does not restrict values to a specific range, but hopefully, most of them will be in the range of -2 to +2 for normally distributed data
    - Less affected by outlier

$$x' = \frac{x - \mu}{\sigma}$$

$$\begin{cases} mean = \mu \\ std = \sigma \end{cases} \Rightarrow \begin{cases} mean = 0 \\ std = 1 \end{cases}$$

… Feature Scaling

- Scikit-Learn transformers for scaling
  - Min-max scaling
    - `feature_range` hyperparameter lets you change the range if you don't want 0–1
      - E.g. neural networks work best with zero-mean inputs, so a range of –1 to 1 is preferable

```
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)
```

… Feature Scaling

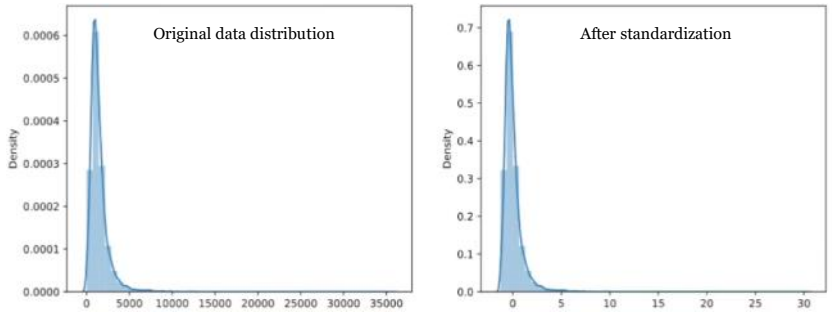
- … Scikit-Learn transformers for scaling
  - Standardization

```
from sklearn.preprocessing import StandardScaler

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

- To scale a sparse matrix, you can use `StandardScaler(with_mean=False)`
  - It will only divide the data by the standard deviation, without subtracting the mean (as this would break sparsity)

Feature Transformation

- Problem
  - Scaling with previous methods does not change the shape of distributions
  - Features with heavy (long) tail distribution do not fit well into ML algorithms
  - We are interested in Normal distributions

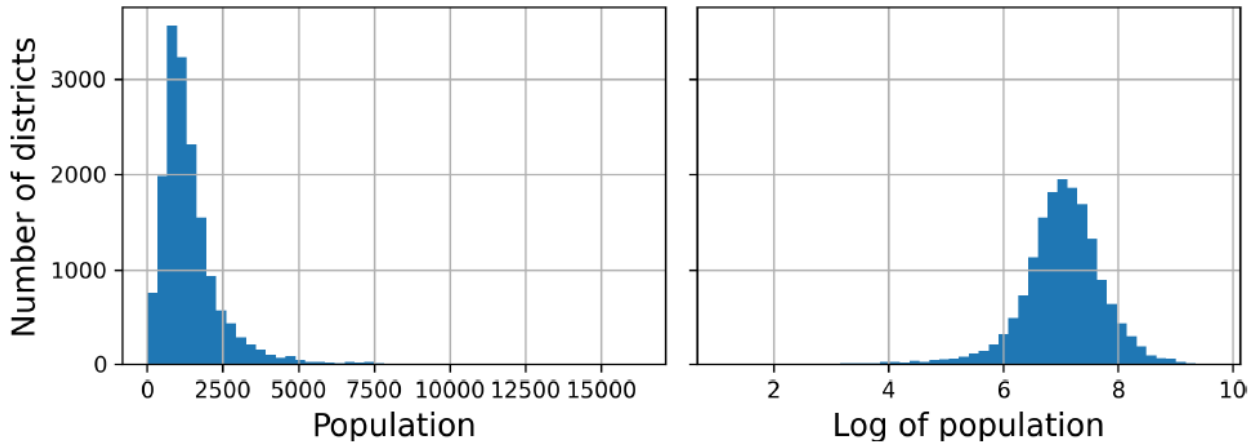


- Solution
  - Before scaling, shrink the heavy tail and make it roughly symmetric if possible
  - How? …

… Feature Transformation

- … Solution
  - **Before scaling**, replace each value with…
    - its square root
      - or raise the feature to a power between 0 and 1
    - its logarithm

(For positive features with a heavy tail to the right)



… Feature Transformation

- … Solution
  - Bucketizing: Chopping the feature distribution into roughly equal-sized buckets, and replacing each feature value with the index of the bucket it belongs to
    - Like we did to create the `income_cat` feature in the "stratified sampling" section
    - When a feature has a multimodal distribution, it is better to treat the bucket IDs as categories (rather than numerical values)
      - Encode the bucket indices by `OneHotEncoder`.
      - Benefit: The regression model learn different rules for different ranges more easily
      - Example: median house age

… Feature Transformation

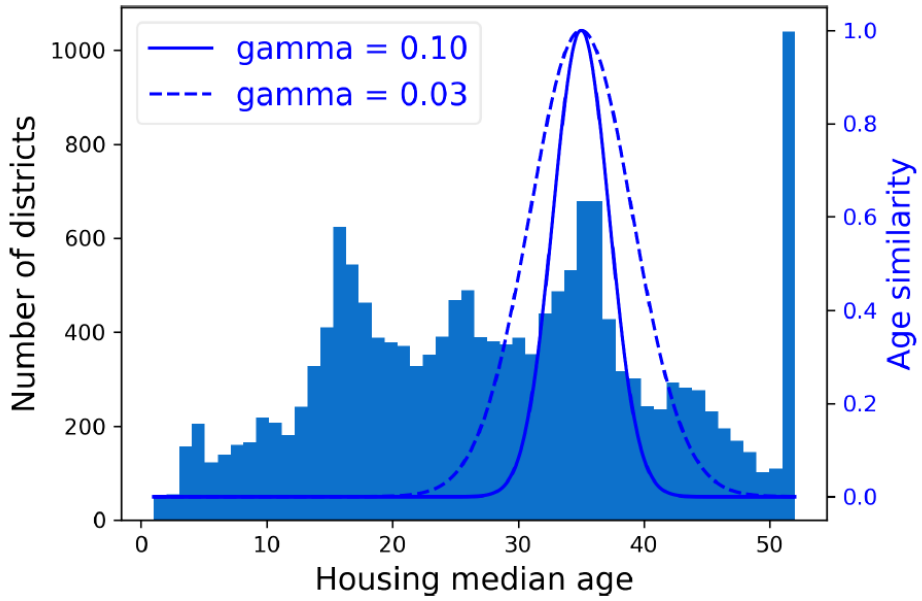
- … Solution
  - Add new features for multimodal distributions
    - Add a feature for each of the modes (at least the main ones), representing the similarity between the values and the modes
    - How do these features help us?
      - For example, perhaps houses built around 35 years ago have a special style that fell out of fashion, and therefore they're cheaper than their age alone would suggest
    - Similarity measure?
      - Typically, computed using a radial basis function (RBF)
        - RBF: Any function that depends only on the distance between the input value and a fixed point
        - The most commonly used RBF is the Gaussian RBF
          - Formula: $y = e^{-\gamma(x-C)^2}$
            - C is the fixed point
            - γ determines how quickly the similarity measure decays as x moves away from the fixed point
              - Hyperparameter that specifies the shape

… Feature Transformation

- … Solution
  - … Add new features for multimodal distributions
    - Example- housing median age
      - The Gaussian RBF similarity between the housing age x and 35 is given by the equation $e^{-\gamma(x-35)^2}$

```
from sklearn.metrics.pairwise import rbf_kernel

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

Another similarity feature can also be defined for $age = 17$



Target Scaling

- Description: In addition to the input features, the target values may also need to be transformed.
  - Example: If the target distribution has a heavy tail, replace the target with its logarithm.

- Problem: The regression model will predict the transformed value
  - Example: The regression model will predict the transformed log of the median house value, not the median house value itself.

- Solution: Revert the predicted value
  - Example: Compute the exponential of the model's prediction

… Target Scaling

- Implementation:
  - Most of Scikit-Learn's transformers have an `inverse_transform()` method

```
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)
```

… Target Scaling

- … Implementation:
  - Simpler option: `TransformedTargetRegressor`

```
from sklearn.compose import TransformedTargetRegressor

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

Custom Transformers

- Problem
  - Scikit-Learn provides many useful transformers, but you may need your owns
- Solution-1
  - Write your own function which
    - Takes a NumPy array as input
    - Outputs the transformed array
  - Example: Custom transformer to transform features with heavy-tailed distributions by replacing them with their logarithm

… Custom Transformers

- Solution-2
  - For transformations that don't require any training (e.g., mean), you can use the Scikit-Learn's `FunctionTransformer`
  - Example: Replacing feature values with their logarithms

```
from sklearn.preprocessing import FunctionTransformer

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

Optional argument
e.g. if you plan to use your transformer in a TransformedTargetRegressor

… Custom Transformers

- … Solution-2
  - … `FunctionTransformer`
    - Functions with hyperparameters
      - Use Kw_args and pass a dictionary of the hyperparameters
      - Example-1

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

      - Example-2

Location of 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"]])
```

Measure the geographic similarity between each district and San Francisco
(Euclidean distance)

… Custom Transformers

- … Solution-2
  - … `FunctionTransformer`
    - Can be used to combine features
      - Example- divide the value of the 1st column by the value of the 2nd column

```
>>> ratio_transformer = FunctionTransformer(lambda X: X[:, [0]] / X[:, [1]])
>>> ratio_transformer.transform(np.array([[1., 2.], [3., 4.]]))
array([[0.5 ],
       [0.75]])
```

… Custom Transformers

- Solution-3
  - For transformation that requires training, we should write a custom class that implements `fit()`, `transform()`, and `fit_transform()` methods
  - Implementation notes
    - The `fit_transform()` method is not required to be implemented if we use the `TransformerMixin` as a base class
      - It simply calls the `fit()` and `transform()` methods sequentially
    - The `fit()` method must return self
    - The `fit()` method must take two arguments **X** and **y**
      - Even when we do not use **y**, we need the y=None argument

… Custom Transformers

- … Solution-3

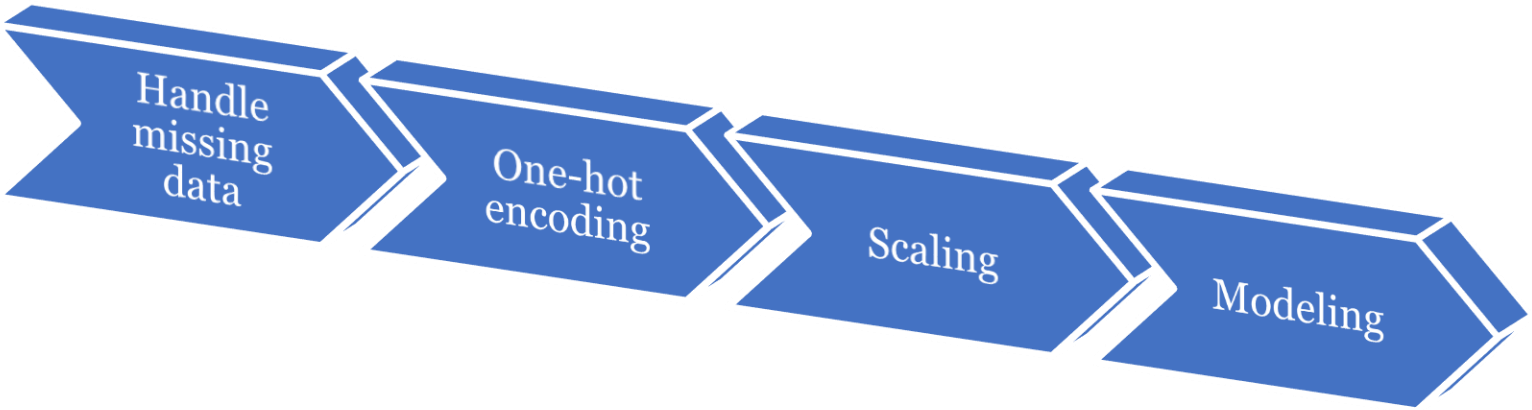
If you add `BaseEstimator` as a base class and avoid using `*args` and `**kwargs` in the constructor, You will get two extra methods: `get_params()` and `set_params()`. These will be useful for hyperparameter tuning

If you add `TransformerMixin` as a base class, You will get the default `fit_transform()` method (`fit()` + `transform()`)

```python
from sklearn.base import BaseEstimator, TransformerMixin
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)]
```

Transformation Pipelines

- Problem
  - many data transformation steps need to be executed in the right order
- Solution
  - Make a function that calls the transformers sequentially



- Implementation
  - Use Scikit-Learn's `Pipeline` or `ColumnTransformer` classes

… Transformation Pipelines

- `Pipeline`- Example: Impute then scale the input features

```
from sklearn.pipeline import Pipeline

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

The `Pipeline` constructor

List of name/estimator pairs

It can be anything you like
Useful for hyperparameter tuning (discuss later)

must be transformers (must implement `fit()` and `transform()` methods)
Exception: implementing the `transform()` method is not essential for the last estimator.
(The last one can be any estimator that implements `fit()` e.g. predictor)

- If you don't want to name the transformers, you can use the make_pipeline() instead

```
from sklearn.pipeline import make_pipeline
num_pipeline = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
```

… Transformation Pipelines

- … `Pipeline`
  - Result of calling the pipeline's `fit_transform()` method
    - `fit_transform()` is called sequentially on all the transformers
    - Passing the output of each call as the parameter to the next call

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

  - Result of calling the pipeline's `fit()/predict()` method:
    - `fit_transform()` is called sequentially on all the transformers
      - Except the final estimator, for which it just calls the `fit()/predict()` method

… Transformation Pipelines

- `ColumnTransformer`
  - Apply appropriate transformations to each column separately
    - Useful when we want to make different transformations on the various columns using a single transformer
    - Example- handle categorical and numerical columns separately

… Transformation Pipelines

- … `ColumnTransformer`
  - … Example: Apply num_pipeline (the one we just defined) to the numerical attributes and cat_pipeline to the categorical attribute

Not convenient to name the attributes
Use function `make_column_selector()`

```
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),
])
```

Set 000 … 0 for unknown (not seen before) categories

Can also use `make_column_transformer()`

… Transformation Pipelines

- … `ColumnTransformer`

```
from sklearn.compose import make_column_selector, make_column_transformer

preprocessing = make_column_transformer(
    (num_pipeline, make_column_selector(dtype_include=np.number)),
    (cat_pipeline, make_column_selector(dtype_include=object)),
)
```

  - Transformation

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

Transformation Pipelines (Example)

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

def ratio_name(function_transformer, feature_names_in):
    return ["ratio"]  # feature names out

def ratio_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=ratio_name),
        StandardScaler())

log_pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    FunctionTransformer(np.log, feature_names_out="one-to-one"),
    StandardScaler())
cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
default_num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                                     StandardScaler())
preprocessing = ColumnTransformer([
        ("bedrooms", ratio_pipeline(), ["total_bedrooms", "total_rooms"]),
        ("rooms_per_house", ratio_pipeline(), ["total_rooms", "households"]),
        ("people_per_house", ratio_pipeline(), ["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
```

Callable: A function that takes two positional arguments:
1. This FunctionTransformer (`self`)
2. An array of input feature names (`input_features`)

and returns an array of output feature names

Determines the list of feature names that will be returned by the `get_feature_names_out()` method. The `get_feature_names_out()` method is only defined if `feature_names_out` is not `None` (default).

The output feature names will equal the input feature names.

… Transformation Pipelines (Example)

- Transformation

```
>>> housing_prepared = preprocessing.fit_transform(housing)
>>> housing_prepared.shape
(16512, 24)
>>> preprocessing.get_feature_names_out()
array(['bedrooms__ratio', 'rooms_per_house__ratio',
       'people_per_house__ratio', 'log__total_bedrooms',
       'log__total_rooms', 'log__population', 'log__households',
       'log__median_income', 'geo__Cluster 0 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)
```

estimator name ___ category name

Select and Train a Model

Main Steps in ML Projects

1. Look at the big picture
2. Get the data
3. Explore and visualize the data to gain insights
4. Prepare the data for machine learning algorithms
5. **Explore many different models, shortlist the best ones, and train the models**
6. Fine-tune your models and combine them into a great solution
7. Present your solution
8. Launch, monitor, and maintain your system

Strat with a Basic Model

- Train a very basic linear regression model to get started

```
from sklearn.linear_model import LinearRegression

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

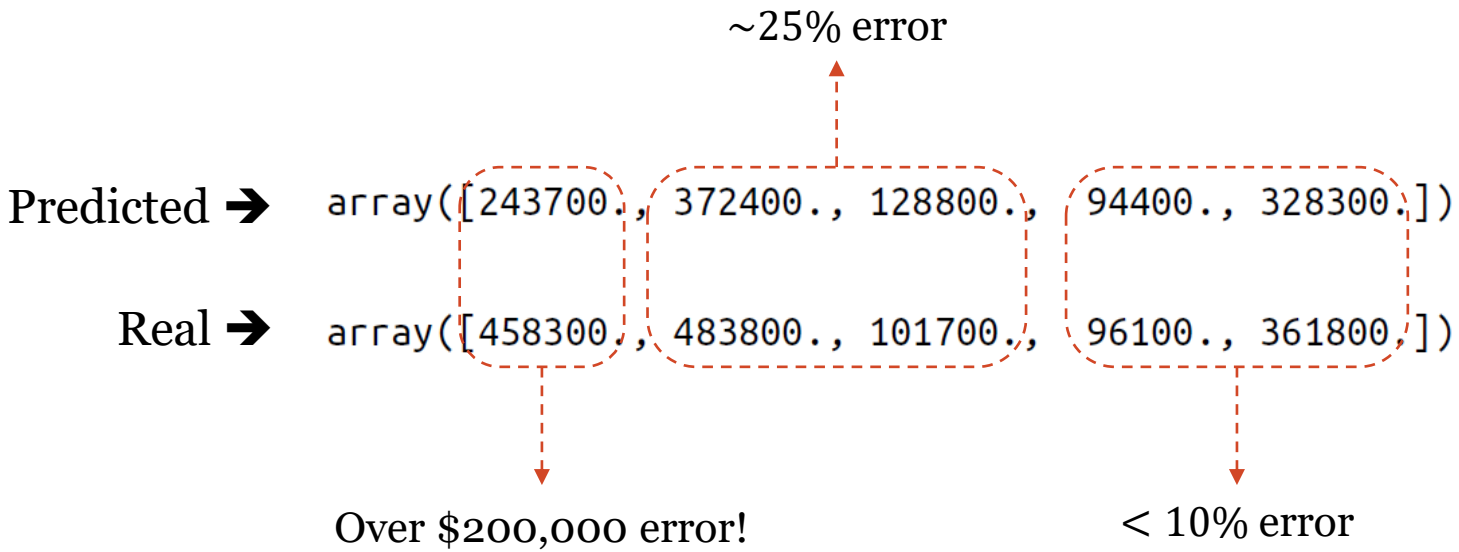
Not fit_transform()

Initial Evaluation

- Simple initial evaluation
  - Looking at the first five 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.])
>>> housing_labels.iloc[:5].values
array([458300., 483800., 101700.,  96100., 361800.])
```

… Initial Evaluation



Evaluation on the Training Data

- Evaluation on the whole training data
  - Remember that you chose to use the RMSE as your performance measure

```
from sklearn.metrics import root_mean_squared_error
lin_rmse = root_mean_squared_error(housing_labels, housing_predictions)
```

```
>>> lin_rmse
68687.89176589991
```

$120,000 < median_housing_values < $265,000 (for most districts)

Prediction error of $68,600 is not satisfying

Not a great score

Model underfitting the training data

… Evaluation on the Training Data

- Model underfitting on the training data
  - What does it mean?
    - The features do not provide enough information to make good predictions, OR
    - The model is not powerful enough
  - What should we do?
    - We decide to test a more complex model capable of finding complex nonlinear relationships ➔ DecisionTreeRegressor (discuss later)

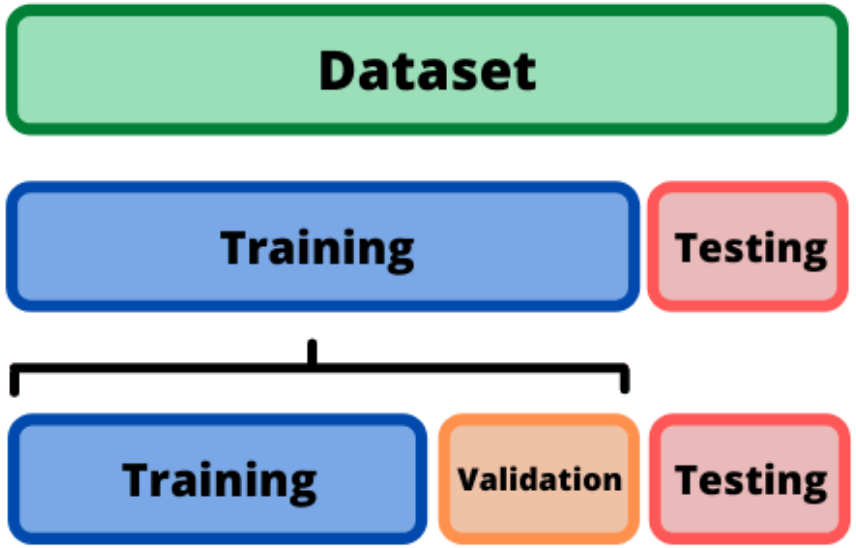
```
from sklearn.tree import DecisionTreeRegressor

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

housing_predictions = tree_reg.predict(housing)
tree_rmse = root_mean_squared_error(housing_labels, housing_predictions)
```

```
>>> tree_rmse
0.0
```

→ Likely to be overfitted.

… Evaluation on the Training Data

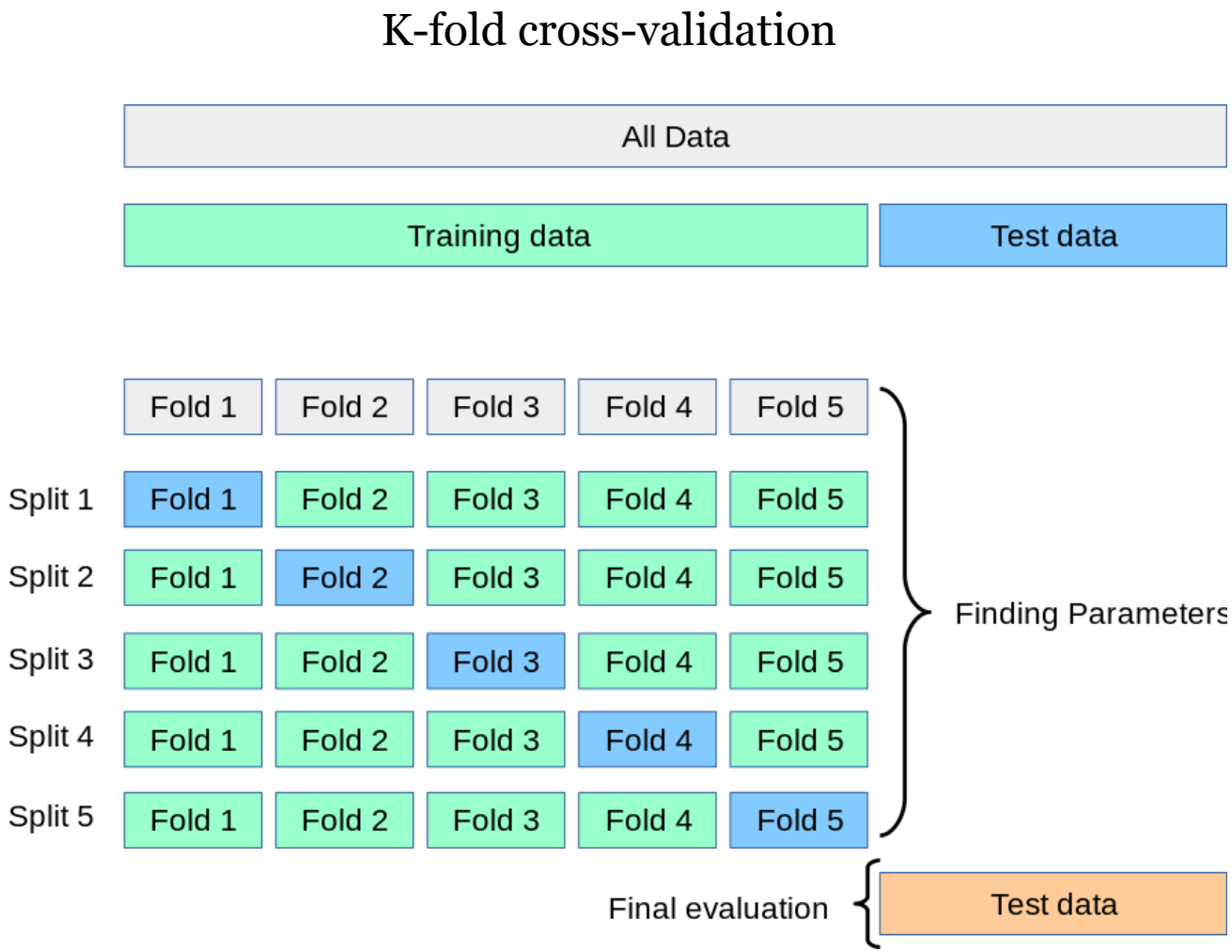
- Model overfitting on the training data
  - How can we be sure about overfitting?
    - Use validation (development or dev) set
    - Why we do not use the test set?
      - Prevent data snooping: We don't want to touch the test set until we are ready to launch a model we are confident about
  - Implementation
    - Use `train_test_split()` on the training set
  - Good, but not perfect!



… Evaluation on the Training Data

- Problems
  - Splitting the training set will reduce the number of samples ➔ May result in insufficient data and inaccurate score
  - A significant portion of the data which contain an important pattern may fall into the validation set
    - We will not train the pattern
- Solution
  - Use k-fold cross-validation
    - We would be more confident

Cross-Validation (CV)



… Cross-Validation

- K-fold CV
  - The training set is split into k smaller nonoverlapping sets, called folds
  - The following procedure is followed for each of the k folds
    - A model is trained using $k - 1$ of the folds as training data
    - The resulting model is validated on the remaining part of the data
      - i.e., it is used as a test set to compute a performance measure
  - The result is an array containing the k evaluation scores
    - The performance measure is the average of the scores

  - k is usually set to 10
    - Experimentally shown to be a good choice!
    - For large datasets, 3 fold is usually sufficient

… Cross-Validation

- Scikit-Learn implementation

```
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)
```

Negative

Expect a utility function (greater is better) rather than a cost function (lower is better)

… Cross-Validation

- Example- Results for `DecisionTreeRegressor`

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

  - The model performs almost as poorly as the linear regression model!
  - The training error is low (actually zero) while the validation error is high
    - Overfitting occurs

… Cross-Validation

- Example- `RandomForestRegressor`

```
from sklearn.ensemble import RandomForestRegressor

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

```
>>> 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
```

… Cross-Validation

Decision tree

```
>>> 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
```

Random forest

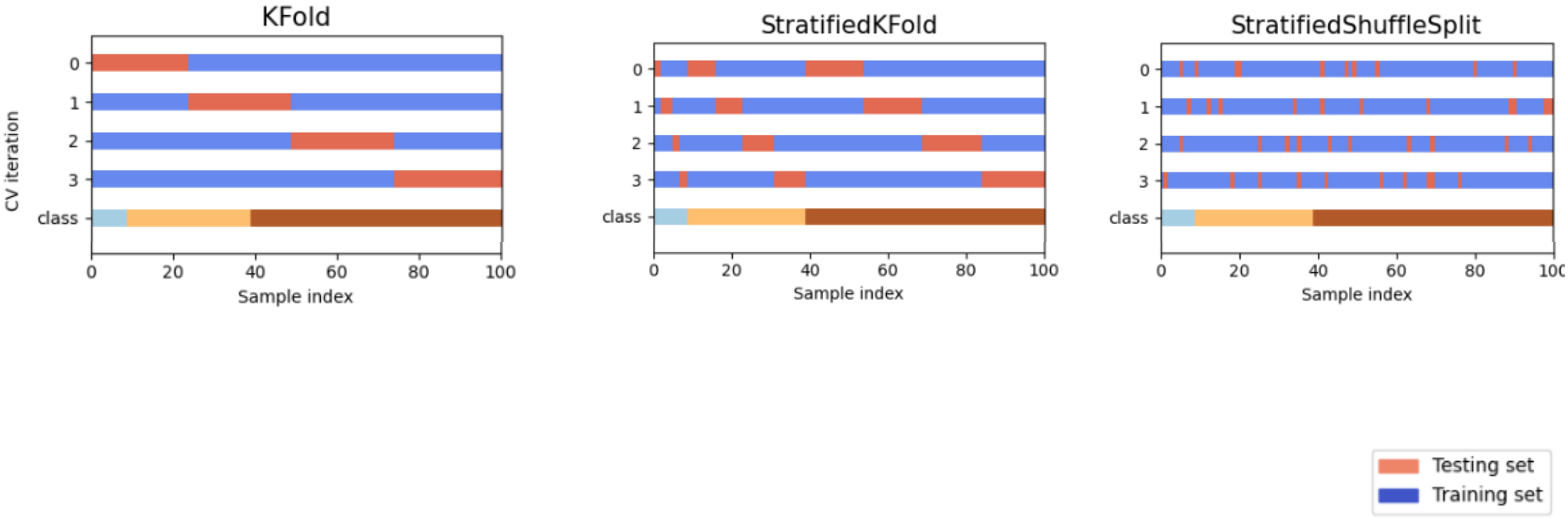
```
>>> pd.Series(forest_rmses).describe()
count       10.000000
mean     47019.561281
std       1033.957120
min      45458.112527
25%      46464.031184
50%      46967.596354
75%      47325.694987
max      49243.765795
dtype: float64
```

Better

… Cross-Validation

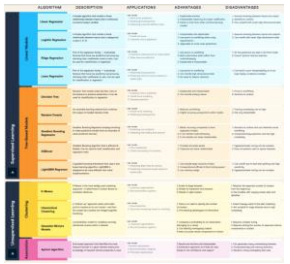
- Other types of CV
  - Repeated K-Fold
    - Repeats K-Fold n times, producing different splits in each repetition
  - Leave One Out (LOO)
    - Each learning set is created by taking all the samples except one, the test set being the sample left out. → useful for datasets with few samples
  - Leave P Out (LPO)
    - Creates all possible training/test sets by removing p samples from the complete set
  - Stratified
    - Each set contains approximately the same percentage of samples of each target class as the complete set
  - Stratified Shuffle
    - The stratified CV that shuffles the data

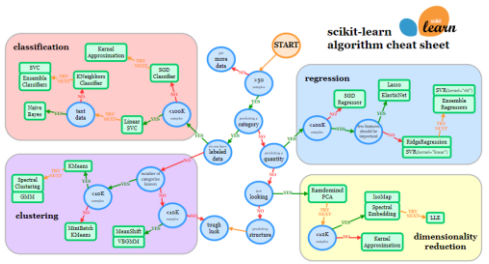
… Cross-Validation



Sum-up

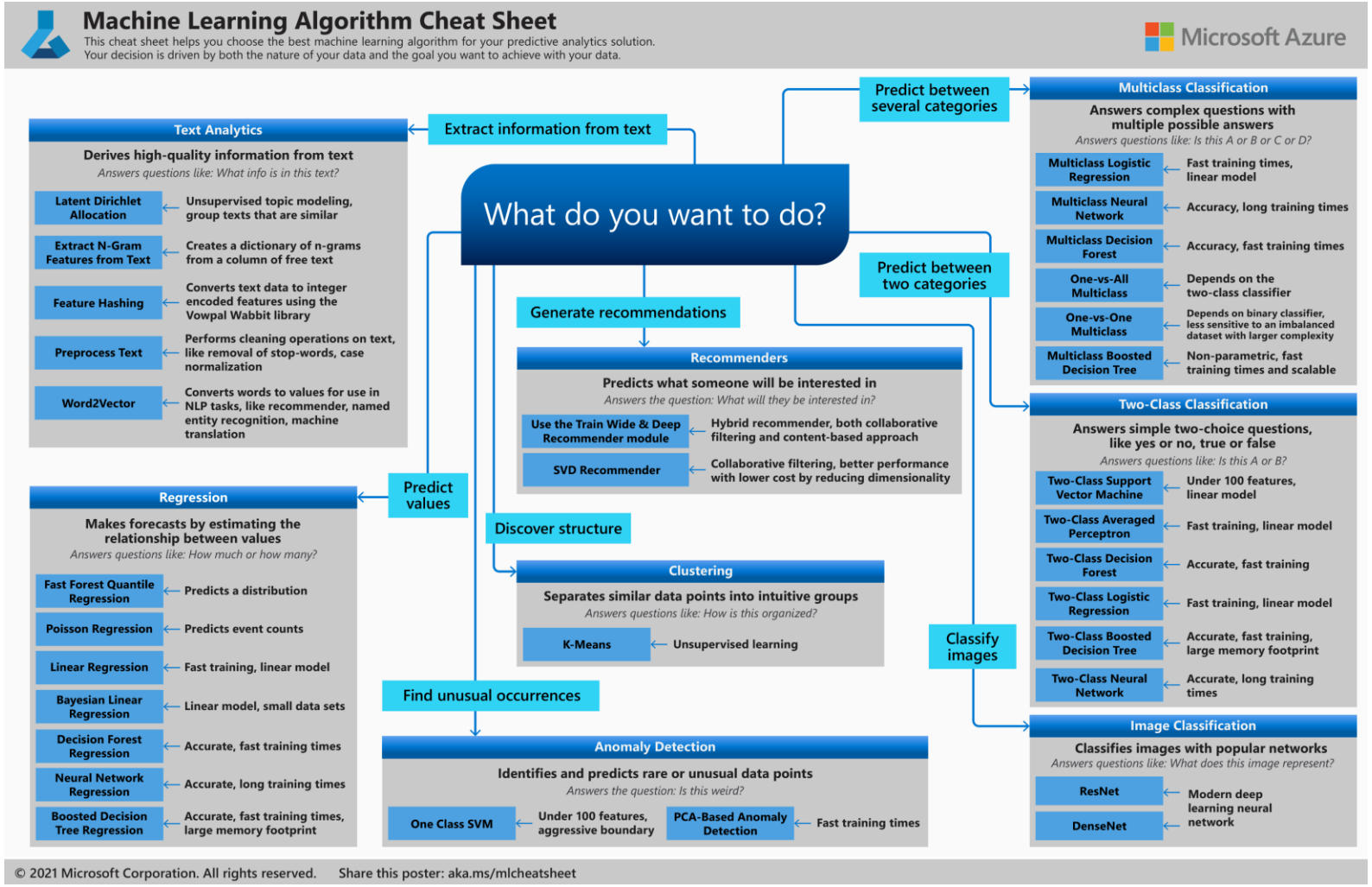
- Goal: Make a shortlist of **two to five** promising ML models
  1. Try out many models from various categories of ML
     - Search for
       - Machine learning algorithms cheat sheet
       - Choosing the right estimator in scikit-learn
       - ...



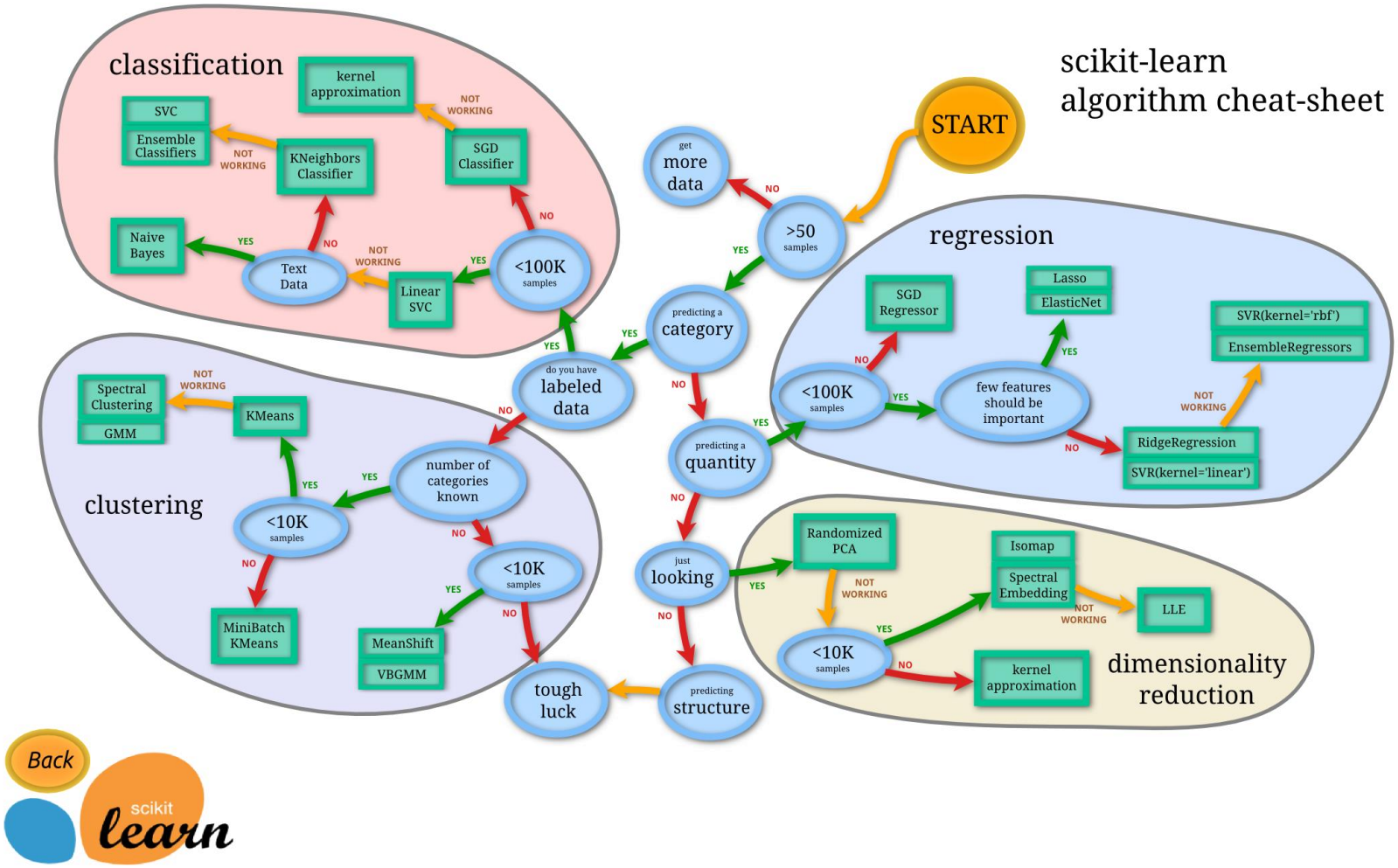


  2. Train and cross-validate them
     - Without spending too much time tweaking the hyperparameters
  3. Choose the best candidates

… Sum-up


Machine Learning Algorithm Cheat Sheet
This cheat sheet helps you choose the best machine learning algorithm for your predictive analytics solution.
Your decision is driven by both the nature of your data and the goal you want to achieve with your data.
Microsoft Azure
What do you want to do?
Extract information from text
Text Analytics
Derives high-quality information from text
Answers questions like: What info is in this text?
Latent Dirichlet Allocation
Unsupervised topic modeling, group texts that are similar
Extract N-Gram Features from Text
Creates a dictionary of n-grams from a column of free text
Feature Hashing
Converts text data to integer encoded features using the Vowpal Wabbit library
Preprocess Text
Performs cleaning operations on text, like removal of stop-words, case normalization
Word2Vector
Converts words to values for use in NLP tasks, like recommender, named entity recognition, machine translation
Predict between several categories
Multiclass Classification
Answers complex questions with multiple possible answers
Answers questions like: Is this A or B or C or D?
Multiclass Logistic Regression
Fast training times, linear model
Multiclass Neural Network
Accuracy, long training times
Multiclass Decision Forest
Accuracy, fast training times
One-vs-All Multiclass
Depends on the two-class classifier
One-vs-One Multiclass
Depends on binary classifier, less sensitive to an imbalanced dataset with larger complexity
Multiclass Boosted Decision Tree
Non-parametric, fast training times and scalable
Predict between two categories
Two-Class Classification
Answers simple two-choice questions, like yes or no, true or false
Answers questions like: Is this A or B?
Two-Class Support Vector Machine
Under 100 features, linear model
Two-Class Averaged Perceptron
Fast training, linear model
Two-Class Decision Forest
Accurate, fast training
Two-Class Logistic Regression
Fast training, linear model
Two-Class Boosted Decision Tree
Accurate, fast training, large memory footprint
Two-Class Neural Network
Accurate, long training times
Generate recommendations
Recommenders
Predicts what someone will be interested in
Answers the question: What will they be interested in?
Use the Train Wide & Deep Recommender module
Hybrid recommender, both collaborative filtering and content-based approach
SVD Recommender
Collaborative filtering, better performance with lower cost by reducing dimensionality
Predict values
Regression
Makes forecasts by estimating the relationship between values
Answers questions like: How much or how many?
Fast Forest Quantile Regression
Predicts a distribution
Poisson Regression
Predicts event counts
Linear Regression
Fast training, linear model
Bayesian Linear Regression
Linear model, small data sets
Decision Forest Regression
Accurate, fast training times
Neural Network Regression
Accurate, long training times
Boosted Decision Tree Regression
Accurate, fast training times, large memory footprint
Discover structure
Clustering
Separates similar data points into intuitive groups
Answers questions like: How is this organized?
K-Means
Unsupervised learning
Classify images
Image Classification
Classifies images with popular networks
Answers questions like: What does this image represent?
ResNet
Modern deep learning neural network
DenseNet
Find unusual occurrences
Anomaly Detection
Identifies and predicts rare or unusual data points
Answers the question: Is this weird?
One Class SVM
Under 100 features, aggressive boundary
PCA-Based Anomaly Detection
Fast training times
© 2021 Microsoft Corporation. All rights reserved.
Share this poster: aka.ms/mlcheatsheet

… Sum-up


scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
Text Data
NO
NOT WORKING
Linear SVC
YES
<100K samples
NO
get more data
NO
>50 samples
YES
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
NO
<100K samples
YES
few features should be important
YES
NOT WORKING
NO
RidgeRegression
SVR(kernel='linear')
YES
predicting a category
YES
NO
do you have labeled data
YES
NO
predicting a quantity
YES
NO
clustering
Spectral Clustering
GMM
NOT WORKING
KMeans
YES
number of categories known
YES
NO
<10K samples
NO
MiniBatch KMeans
<10K samples
YES
NO
MeanShift
VBGMM
just looking
YES
NO
Randomized PCA
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
dimensionality reduction
tough luck
predicting structure
Back
scikit learn

Fine-Tune Your Model

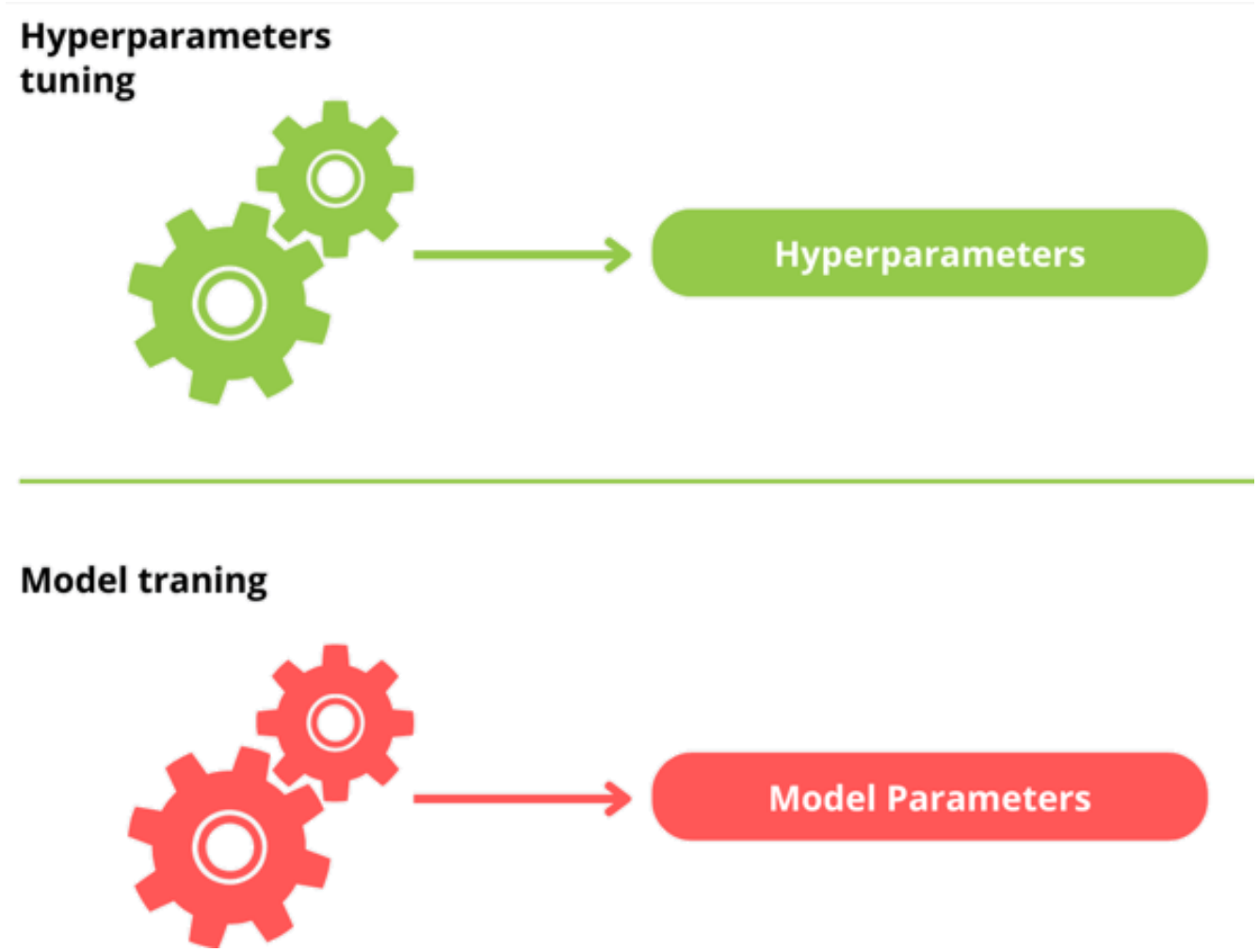
Main Steps in ML Projects

1. Look at the big picture
2. Get the data
3. Explore and visualize the data to gain insights
4. Prepare the data for machine learning algorithms
5. Explore many different models, shortlist the best ones, and train the models
6. **Fine-tune your models and combine them into a great solution**
7. Present your solution
8. Launch, monitor, and maintain your system

Introduction

- Fine-tune the hyperparameters on the promising models you found in the previous step
  - Find the best hyperparameters for each model
- Compare the fine-tuned models
- Choose the best model or combine some of them to get the best result

Hyperparameters vs Parameters



Grid Search

- Description
  - Specify the hyperparameters you want to tune
  - For each hyperparameter, specify the values you want to be tested
  - Use cross-validation to evaluate all the possible combinations of hyperparameter values

… Grid Search

- Implementation: Scikit-Learn's `GridSearchCV`
  - Search for the best combination of hyperparameter values for the `RandomForestRegressor`

```
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)
```

Evaluate 3 × 3 = 9 combinations

+

Evaluate 2 × 3 = 6 combinations

15 combinations

Double underscores
Scikit-Learn splits the string at these points and looks for estimator names in the nested pipeline.

n_clusters and max_features

15 × 3 = 45
rounds of training

… Grid Search

- … Implementation
  - All evaluation scores are available using `cv_results_`
    - This is a dictionary ➔ Wrap it in a DataFrame

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

… Grid Search

- … Implementation
  - Get the best hyperparameters

```
>>> grid_search.best_params_
{'preprocessing__geo__n_clusters': 15, 'random_forest__max_features': 6}
```

  - Extra note
    - If `GridSearchCV` is initialized with refit=True (which is the default), then once it finds the best estimator using cross-validation, it retrains it on the whole training set.

Randomized Search

- Description
  - Evaluates a fixed number of combinations, selecting a random value for each hyperparameter at every iteration

- Used when the hyperparameter search space is large
  - Example
    - Some of your hyperparameters are continuous
    - Discrete hyperparameters with many possible values

… Randomized Search

- Implementation: Scikit-Learn's `RandomizedSearchCV`

```
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)
```

Ensemble Methods

- Another way to fine-tune your system is to try to combine the models that perform best

- The group (or “ensemble”) will often perform better than the best individual model, especially if the individual models make very different types of errors
  - Example: Random Forests (RF) perform better than the individual decision trees they rely on

- We can also combine various models
  - Example: combine the fine-tuned KNN and RF
    - Final prediction = mean of the predictions of these two models

Analyzing the Best Models

- Find the relative importance of each feature
  - You may want to try dropping some of the less useful features

Make it more readable

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

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

Analyzing the Best Models

- Find the relative importance of each feature
  - You may want to try dropping some of the less useful features

```
[(0.18694559869103852, 'log__median_income'),
 (0.0748194905715524, 'cat__ocean_proximity_INLAND'),
 (0.06926417748515576, 'bedrooms__ratio'),
 (0.05446998753775219, 'rooms_per_house__ratio'),
 (0.05262301809680712, 'people_per_house__ratio'),
 (0.03819415873915732, 'geo__Cluster 0 similarity'),
 [...]
 (0.00015061247730531558, 'cat__ocean_proximity_NEAR BAY'),
 (7.301686597099842e-05, 'cat__ocean_proximity_ISLAND')]
```

only one ocean_proximity category is really useful

… Analyzing the Best Models

- Ensure that your model works well on all categories
  - The model may work very well on some categories but poorly on the others ➔ We get a good score on average, but it is deceptive
  - Example: Does our final model work well on all categories of districts, whether they're rural or urban, rich or poor, northern or southern,…
    - The model may predict well for the rich districts but awful for the poor ones
  - How to test?
    - Create subsets of the validation set for each category and test

- What should we do if it is the case?
  - Do not deploy the model until the issue is solved
    - At least it should not be used to make predictions for that category

Evaluate on the Test Set

- Not new!

```
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 = root_mean_squared_error(y_test, final_predictions)
print(final_rmse)  # prints 41424.40026462184
```

- What if it is just 0.1% better than the model currently in production?
  - We should specify how precise this estimate is
    - Compute the 95% confidence interval

… Evaluate on the Test Set

- Compute the confidence interval using `scipy.stats.t.interval()`
  - $\bar{X} \pm t_{\alpha/2} \times \frac{s}{\sqrt{n}}$
    - To calculate $t_{\alpha/2}$ we need to know the degrees of freedom and the confidence level
    - $\frac{s}{\sqrt{n}}$ is called the standard error

```
>>> 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])
```

Others

Main Steps in ML Projects

1. Look at the big picture
2. Get the data
3. Explore and visualize the data to gain insights
4. Prepare the data for machine learning algorithms
5. Explore many different models, shortlist the best ones, and train the models
6. Fine-tune your models and combine them into a great solution
7. **Present your solution**
8. **Launch, monitor, and maintain your system**

Presentation

- Ensure your key findings are communicated through
  - Clear and beautiful visualizations
  - Easy-to-remember statements
    - Example: “the median income is the number one predictor of housing prices”
- Explain why your solution achieves the business objective
- Highlight
  - What you have learned
  - What worked and what did not
  - Assumptions were made
  - Your system’s limitations

… Presentation

- Compare with current solutions
  - In this California housing example, the final performance of the system is not much better than the experts' price estimates (30% error)
  - It may still be a good idea to launch it
    - It frees up some time for the experts so they can work on more productive tasks

You got approval to launch!

Launch

- Get your solution ready for production
  - Polish the code
  - Write documentation and tests

… Launch

- Save the best model you trained
  - Use the `joblib.dump()` method

```
import joblib

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

… Launch

- Transfer the file to your production environment and load it
  - Use the `joblib.load()` method
  - Note: We must first import (copy) any custom classes and functions the model relies on, then load the model

```
import joblib
[...]  # import KMeans, BaseEstimator, TransformerMixin, rbf_kernel, etc.

def column_ratio(X): [...]
def ratio_name(function_transformer, feature_names_in): [...]
class ClusterSimilarity(BaseEstimator, TransformerMixin): [...]

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

new_data = [...]  # some new districts to make predictions for
predictions = final_model_reloaded.predict(new_data)
```

… Launch

- Make an interface for the final user
  - Develop your website/CMD App/GUI App/web service/…
  - Connect the frontend to the code

Monitoring

- Collect fresh data regularly and label it
- Write a script to train the model and fine-tune the hyperparameters automatically
  - Retrain your models on a regular basis on fresh data
- Write another script that will evaluate both the new model and the previous model on the updated test set
  - The script should test the performance of your model on various subsets of the test set, such as poor or rich districts

… Monitoring

- Evaluate the model's input data quality
  - Sometimes performance will degrade because of a poor-quality signal
    - e.g., a malfunctioning sensor sending random values, or another team's output becoming stale
  - How to evaluate the input automatically?
    - We should trace the evidence
    - Examples
      - More and more inputs are missing a feature
      - The mean or std drifts too far from the training set
      - Categorical feature starts containing new categories

Maintenance

- Keep backups of
  - Every model you create
  - Every version of your datasets