Hands-on Machine Learning

End-to-End Machine Learning Project

End-to-End Machine Learning Project

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for machine learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.

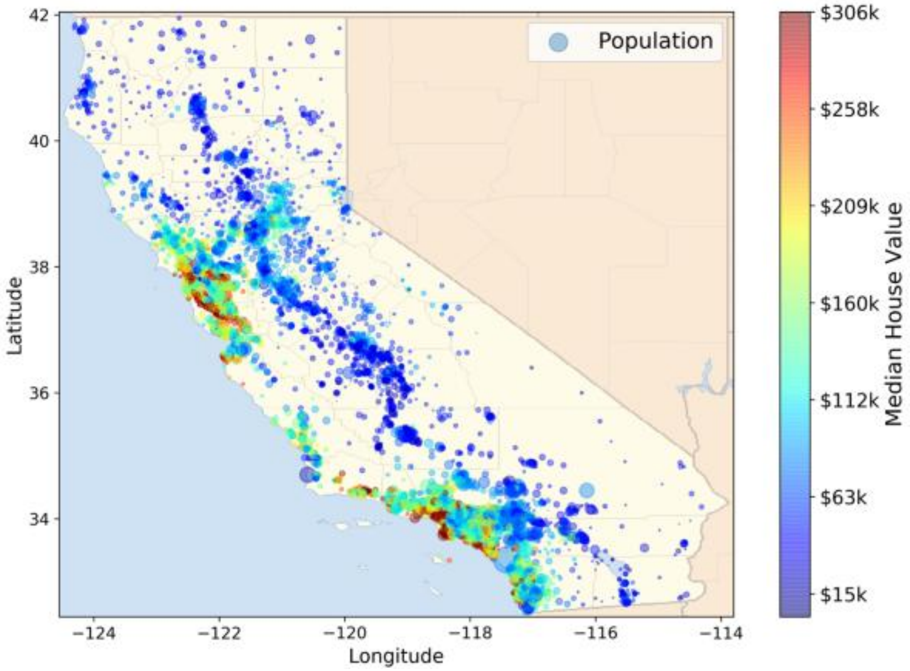
1.

Look at the big picture

California Housing Prices Dataset

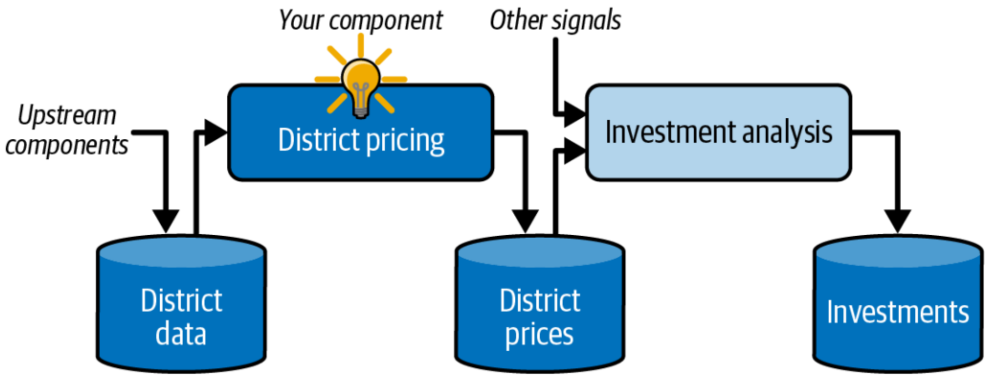
- Task: use California census data to build a model of housing prices in the state
  - metrics such as the population, median income, and median housing price for each district

- Goal: predict the median housing price in any district, given all the other metrics.



Frame the Problem

- What is the business objective?
  - How does the company expect to use and benefit from this model?
- Knowing the objective will determine how you frame the problem:
  - Which algorithms and performance measures you will select?
  - How much effort you will spend tweaking it?



Frame the Problem

- What is the current solution?
  - Prices are estimated manually by experts: costly, time-consuming, and 20% off
- Frame the problem:
  - Is it supervised, unsupervised, or reinforcement learning?
  - Is it a classification task, a regression task, or something else?
  - Should you use batch learning or online learning techniques?
- Check the assumptions:
  - List and verify the assumptions that have been made so far
  - This can help you catch serious issues early on

Select a Performance Measure

- *Root Mean Square Error (RMSE):* gives an idea of how much error the system makes in its predictions, with a higher weight for large errors.

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

- *Mean Absolute Error (MAE):*

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

- ℓ_k *norm* of a vector $\boldsymbol{v}$ containing n elements: $\left(|v_0|^k + |v_1|^k + \cdots + |v_n|^k\right)^{\frac{1}{k}}$

2.
Get the Data

Download the Data

```python
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.tgz"
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

Take a Quick Look at the Data Structure

```
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Check 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
```

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

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

Check the Data

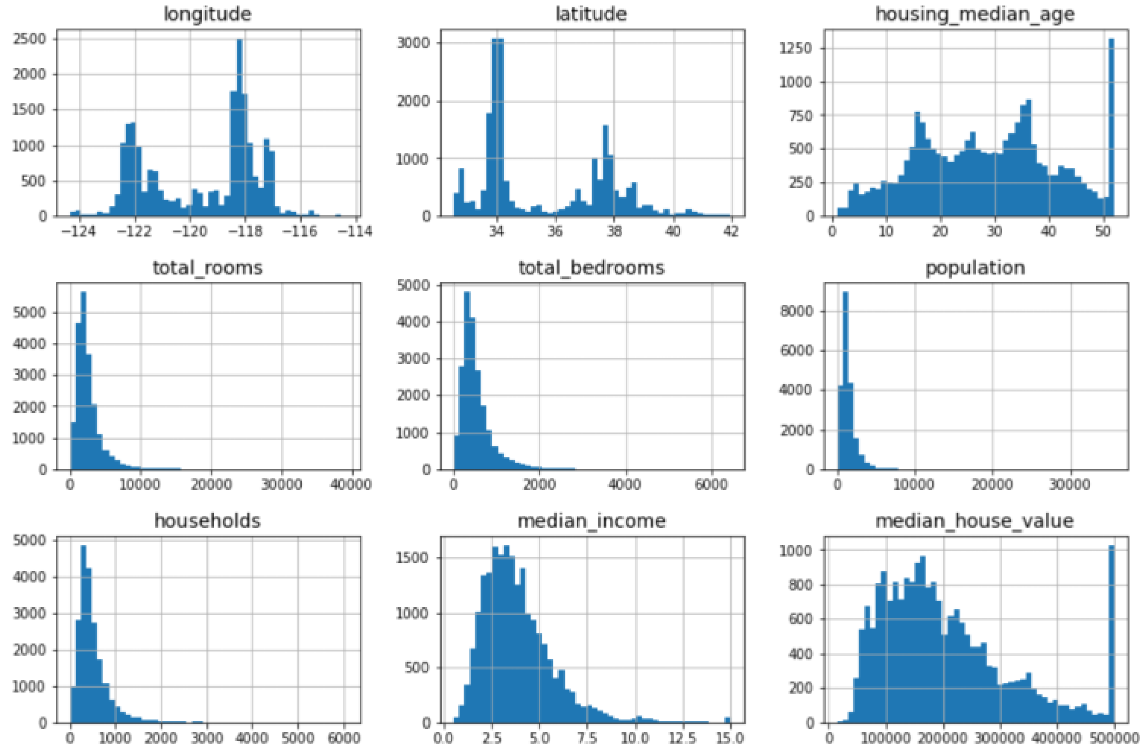
- The `describe()` method shows a summary of the *numerical* attributes.

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

Check the Data: Histogram

```python
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(12, 8))
plt.show()
```



Create a Test Set

- Creating a test set is theoretically simple; pick some instances randomly, typically 20% of the dataset (or less if your dataset is very large).

```python
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]
```

```python
train_set, test_set = shuffle_and_split_data(housing, 0.2)
```

```python
np.random.seed(42)
```

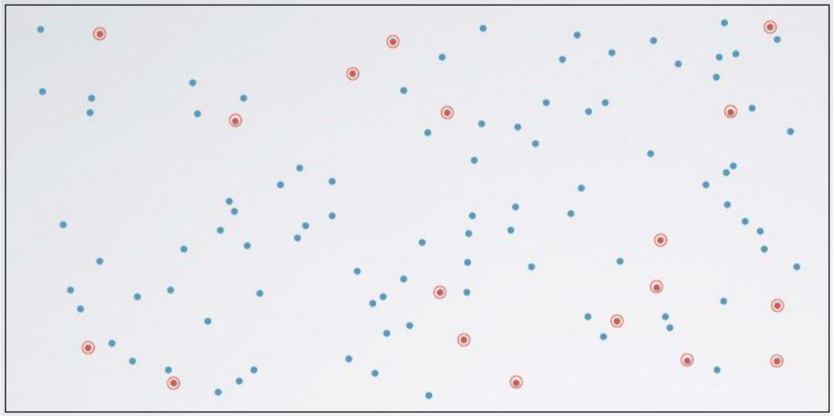
Create a Test Set

- Scikit-Learn provides a few functions to split datasets into multiple subsets in various ways.

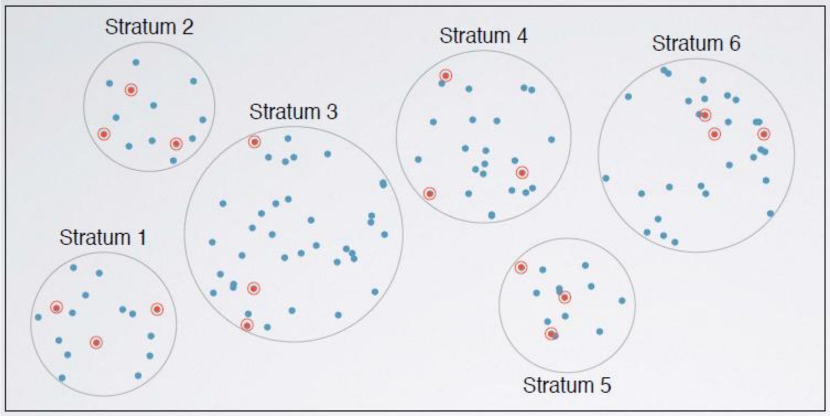
```
from sklearn.model_selection import train_test_split

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

- Using purely random sampling methods is generally fine if your dataset is large enough (relative to the number of attributes)
- If it is not, you run the risk of introducing a significant sampling bias.
- *Stratified sampling*: population is divided into homogeneous subgroups (*strata*), and the right number of instances are sampled from each stratum to guarantee that the test set is representative of the overall population.

Methods of Sampling



Simple Random Sampling



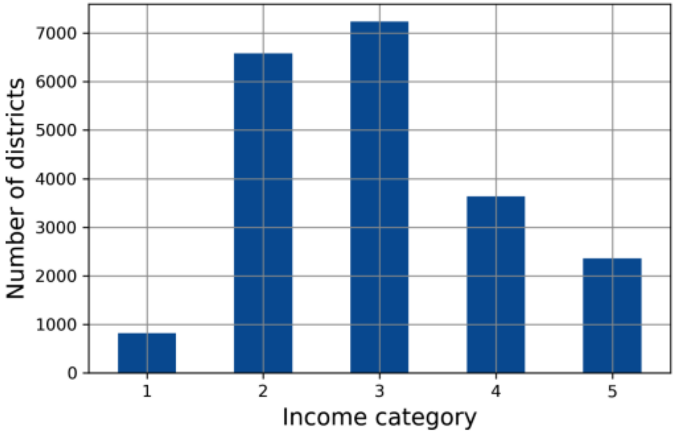
Stratified Sampling

Create a Test Set: Stratified Sampling

- Assume that the median income is an important attribute to predict median housing prices and you want to ensure that the test set is representative of the various categories of incomes in the whole dataset.
- `median_income` is numerical and you should make it categorical.

```
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])
```

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



Create a Test Set: Stratified Sampling

- The `split()` method yields the training and test *indices*, not the data.
- Having multiple splits is useful for cross-validation.

```
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.iloc[train_index]
    strat_test_set_n = housing.iloc[test_index]
    strat_splits.append([strat_train_set_n, strat_test_set_n])
```

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

- A shorter way to get a single split:

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

3.
Discover and Visualize the Data to Gain Insights

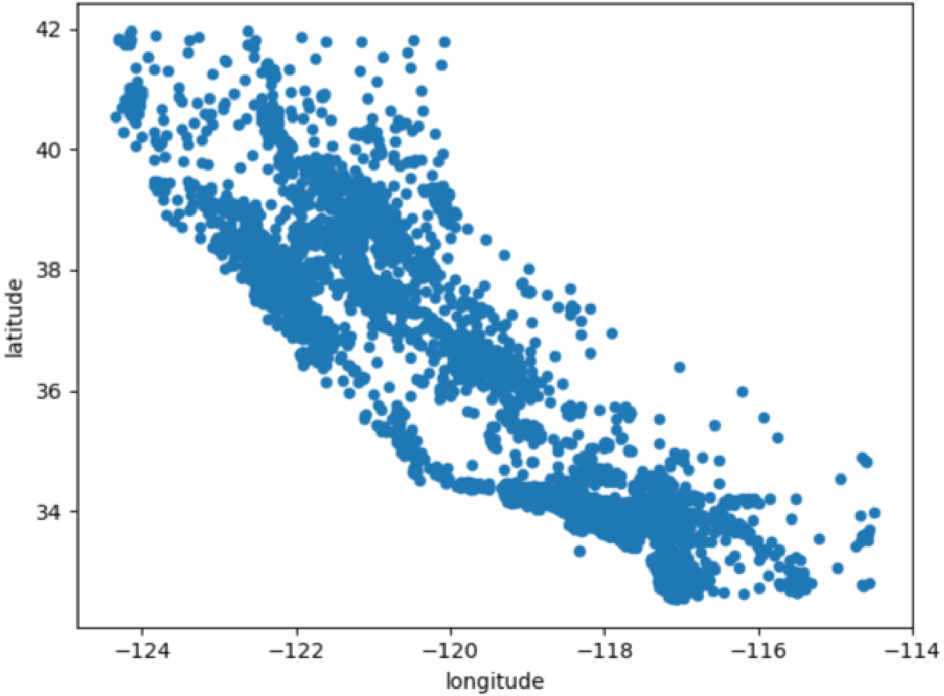
Exploring Training Set

- ➢ Make sure you have put the test set aside and you are only exploring the training set.
- ➢ If the training set is very large, you may want to sample an exploration set, to make manipulations easy and fast.
- ➢ Create a copy so that you can play with it without harming the training set:

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

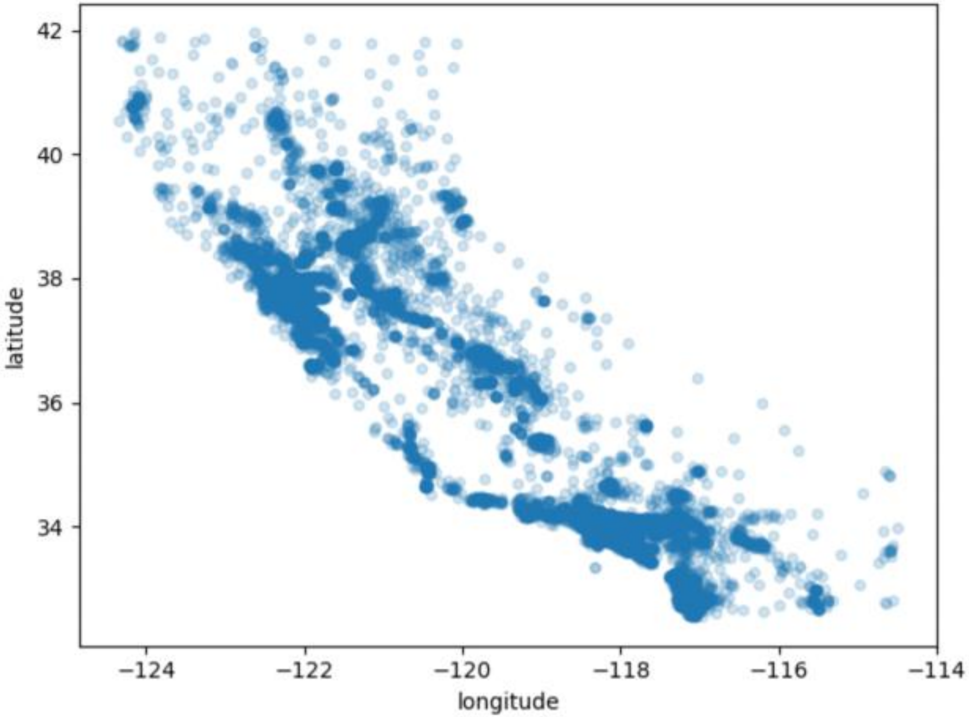
Visualizing Geographical Data

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



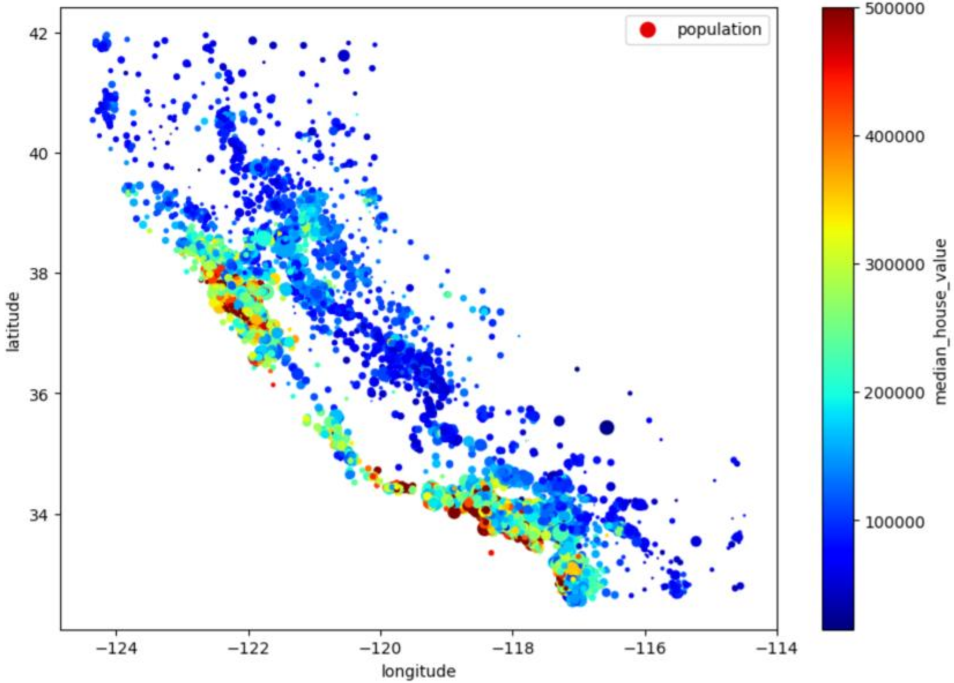
Visualizing Geographical Data

```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.2)
save_fig("better_visualization_plot")  # extra code
plt.show()
```



Visualizing Geographical Data

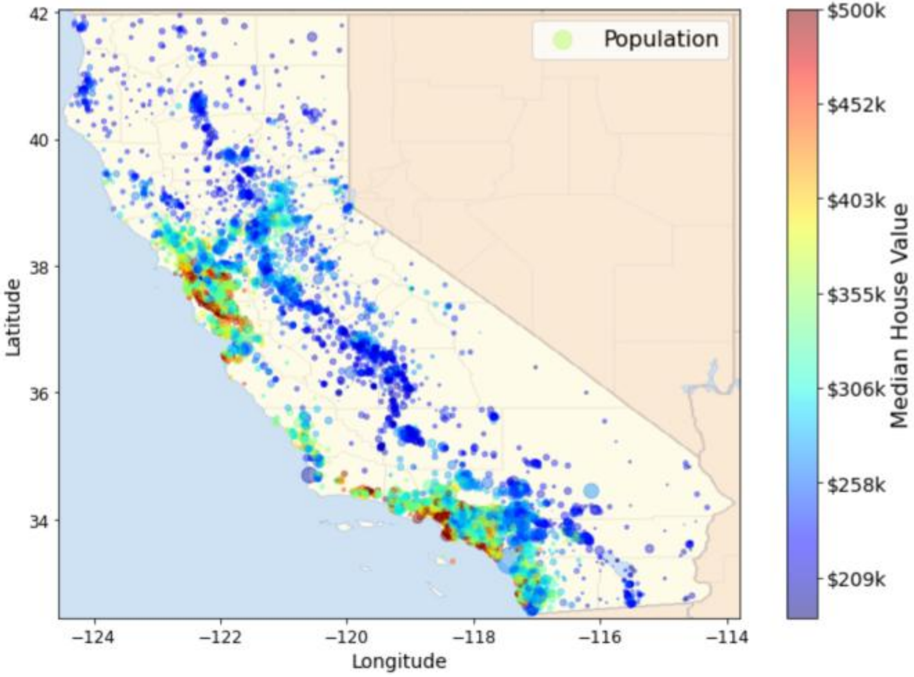
```python
housing.plot(kind="scatter",
             x="longitude", y="latitude",
             s=housing["population"] / 100,
             label="population",
             c="median_house_value",
             cmap="jet",
             colorbar=True,
             legend=True,
             sharex=False,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



Visualizing Geographical Data

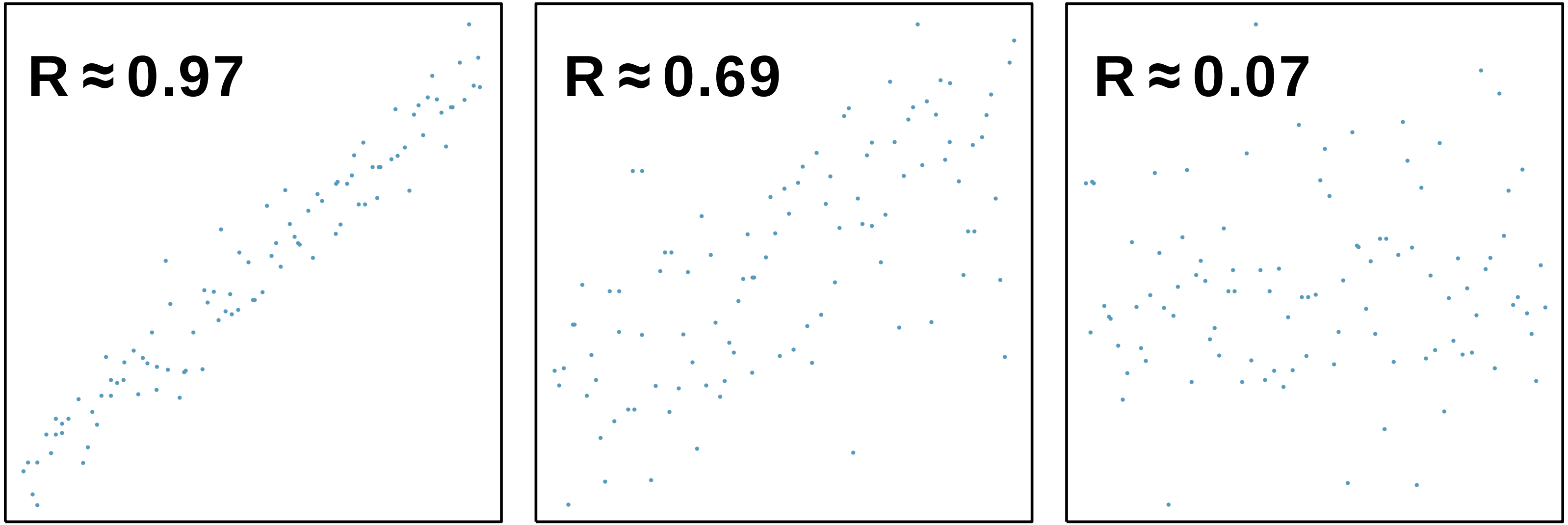
```
import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                  s=housing['population']/100, label="Population",
                  c="median_house_value", cmap=plt.get_cmap("jet"),
                  colorbar=False, alpha=0.4)
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cbar = plt.colorbar(ticks=tick_values/prices.max())
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
save_fig("california_housing_prices_plot")
plt.show()
```

```
Saving figure california_housing_prices_plot
```

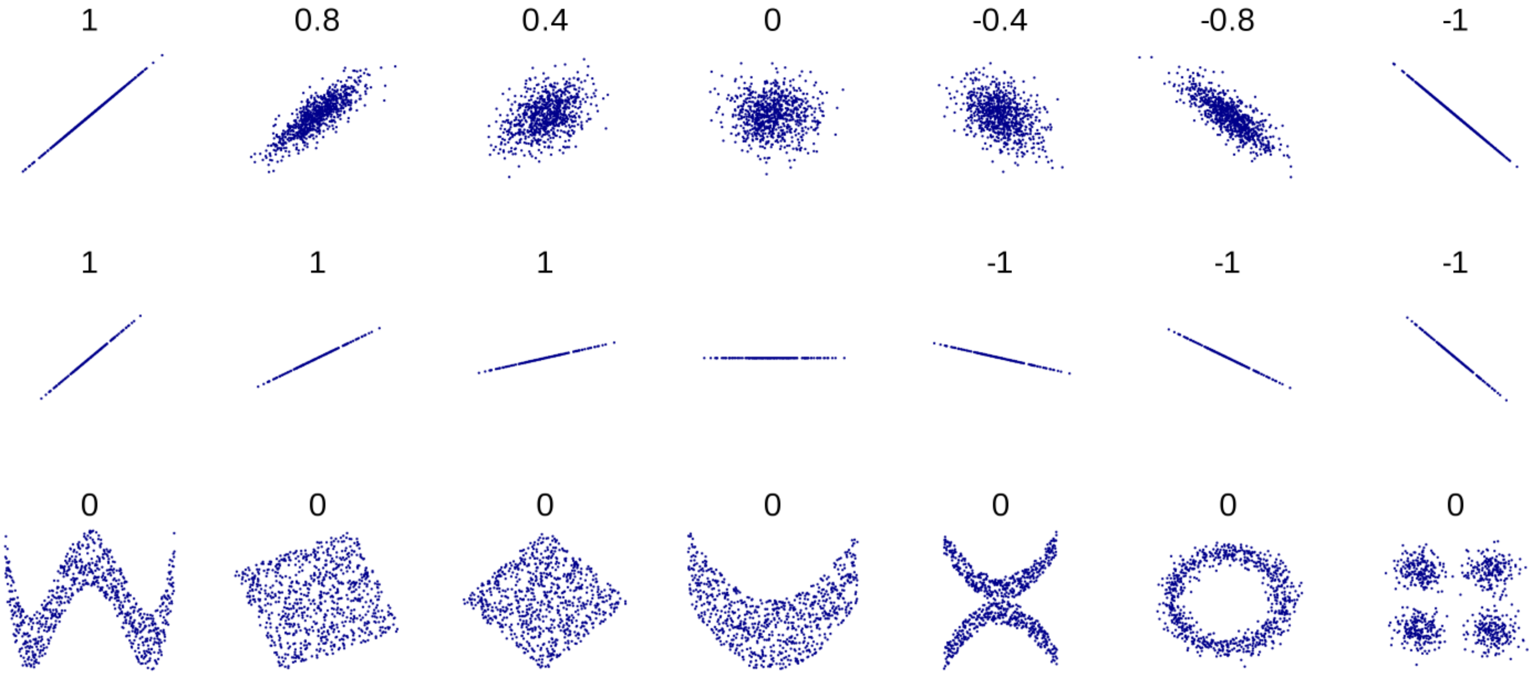


Correlation Coefficient

- Describes the strength of the linear association between two variables and is denoted as R or ρ.



Correlation Coefficient



Looking for Correlations

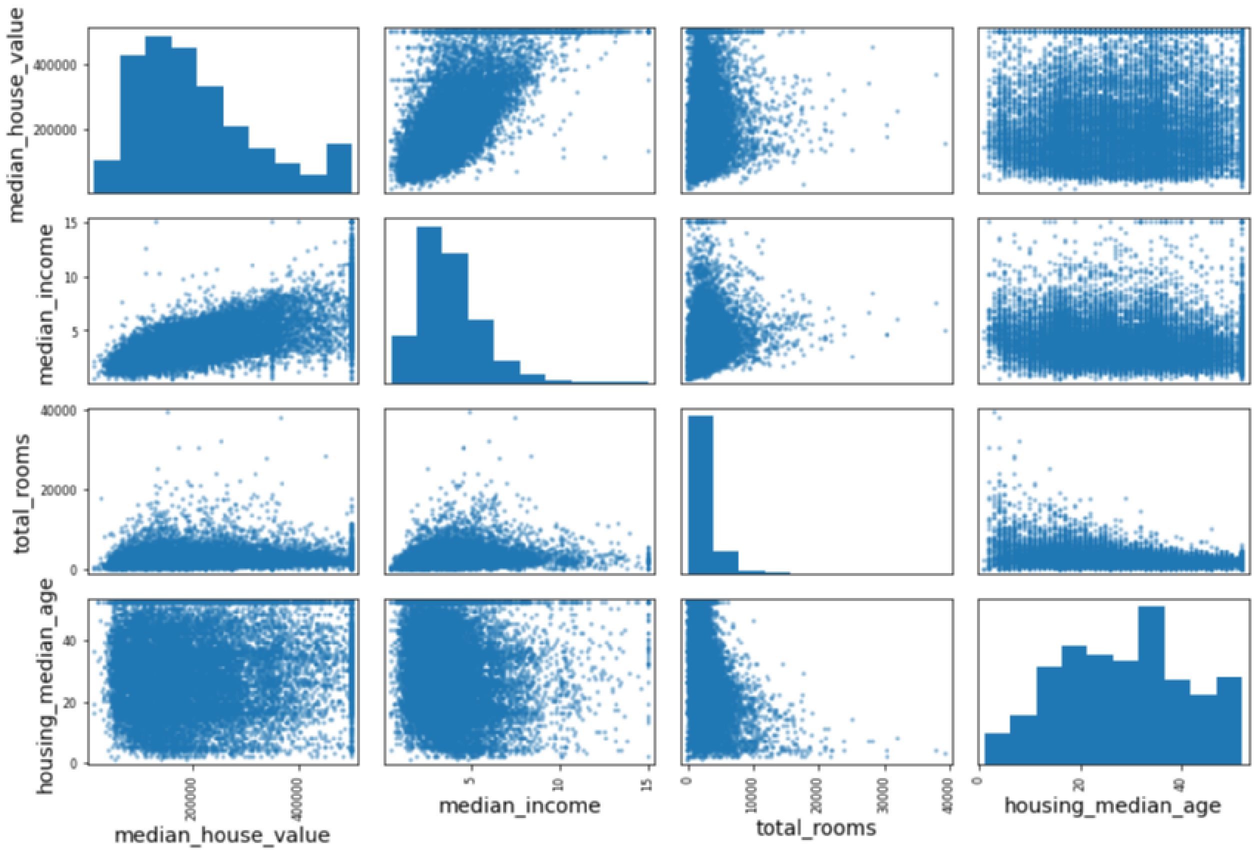
```
corr_matrix = housing.corr()
```

```
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
```

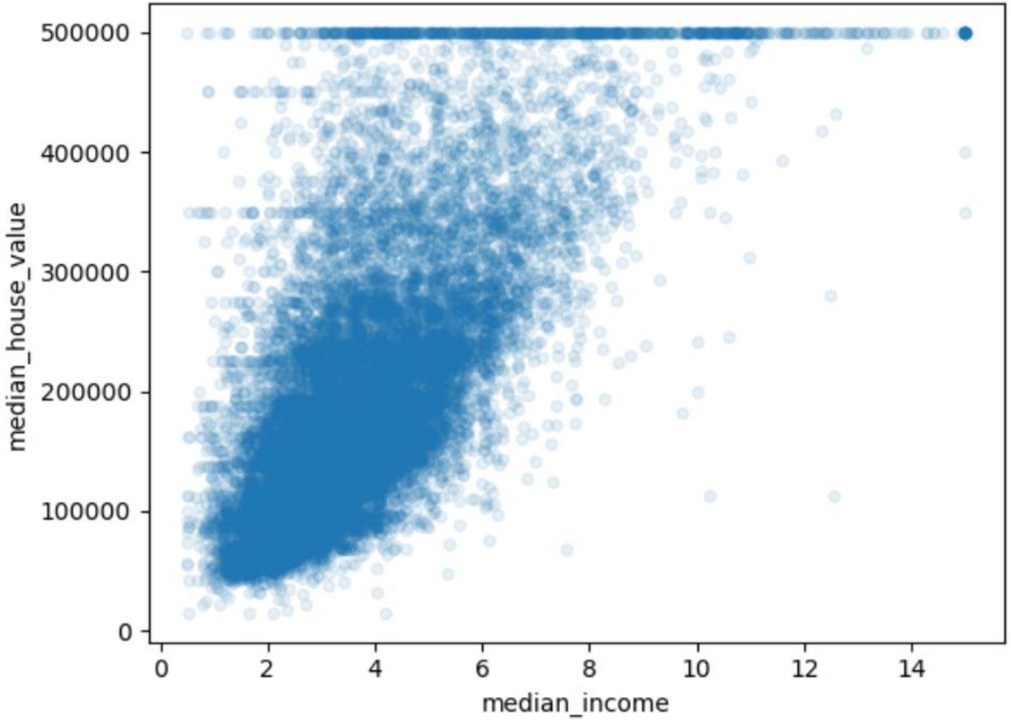
```
from pandas.plotting import scatter_matrix

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

Looking for Correlations



Looking for Correlations

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



Attribute Combinations

- Try out various attribute combinations.
- Example: the total number of rooms in a district is not very useful if you don’t know how many households there are.
  - The number of rooms per household is more informative.
- Create new attributes:

```
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"]
```

Attribute Combinations

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

```
median_house_value    1.000000
median_income         0.688380
rooms_per_house       0.143663
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
people_per_house     -0.038224
longitude            -0.050859
latitude             -0.139584
bedrooms_ratio       -0.256397
Name: median_house_value, dtype: float64
```