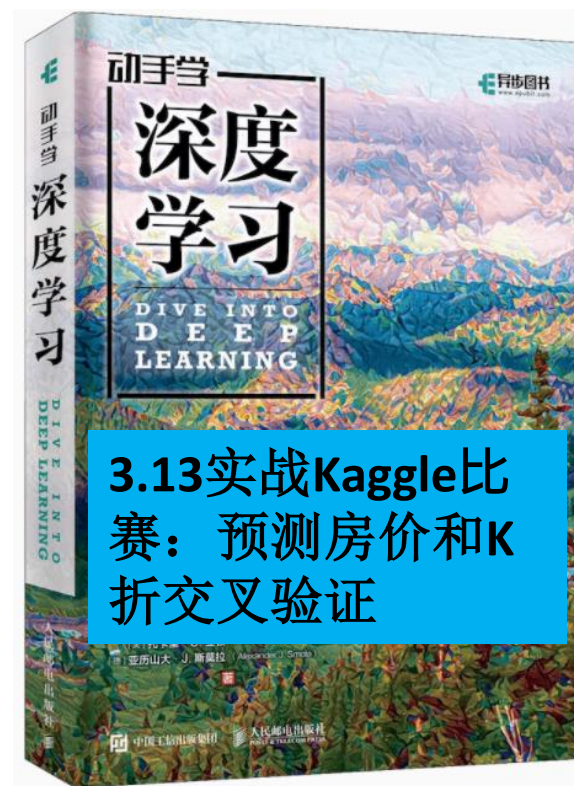
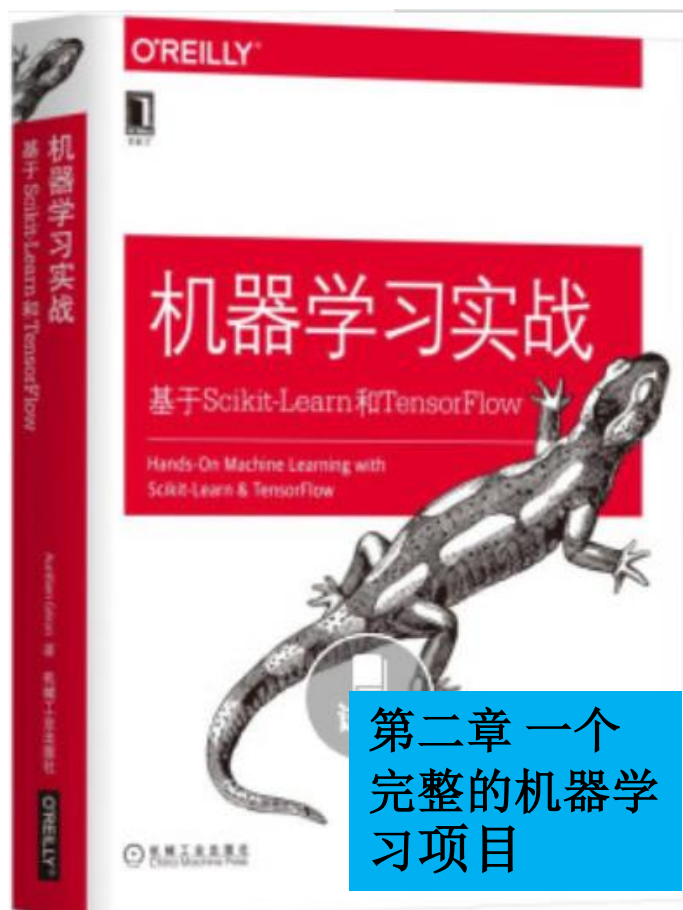


第2章 一个完整的机器学习项目

参考资料



基于MXNet开源深度学习框架
<https://zh.d2l.ai/>

<https://github.com/ageron/handson-ml>

机器学习项目的通用工作流程

- 1 定义问题：软件架构设计、确定评价指标
- 2 获取数据：自动化方式
- 3 研究数据：可视化方式，相关性研究等
- 4 准备数据：数据清理、特征选择及处理
- 5 研究模型：确定评估方法、列出可能的模型并训练，选择最有希望的3~5个模型
- 6 微调模型：寻找最佳超参数，模型融合，评估泛化性能
- 7 展示解决方案：将工作进行文档化总结展示
- 8 启动、监视、维护系统：投入使用

1 定义问题

- 1.1 了解可用的开放数据集
- 1.2 进行业务需求分析
- 1.3 设定问题
- 1.4 选择模型评估指标
- 1.5 检查问题假设是否正确

1.1 了解可用的开放数据集

- 常见的开放数据集：
 - 流行的开源数据仓库：
 - [UC Irvine Machine Learning Repository](#)
 - [Kaggle datasets](#)
 - [Amazon's AWS datasets](#)
 - 准入口（提供开源数据列表）
 - <http://dataportals.org/>
 - <http://opendatamonitor.eu/>
 - <http://quandl.com/>
 - 其它列出流行开源数据仓库的网页：
 - [Wikipedia's list of Machine Learning datasets](#)
 - [Quora.com question](#)
 - [Datasets subreddit](#)

1.1 了解可用的开放数据集

- 本章中用到的数据集：来自StatLib库的加州住房价格的数据集

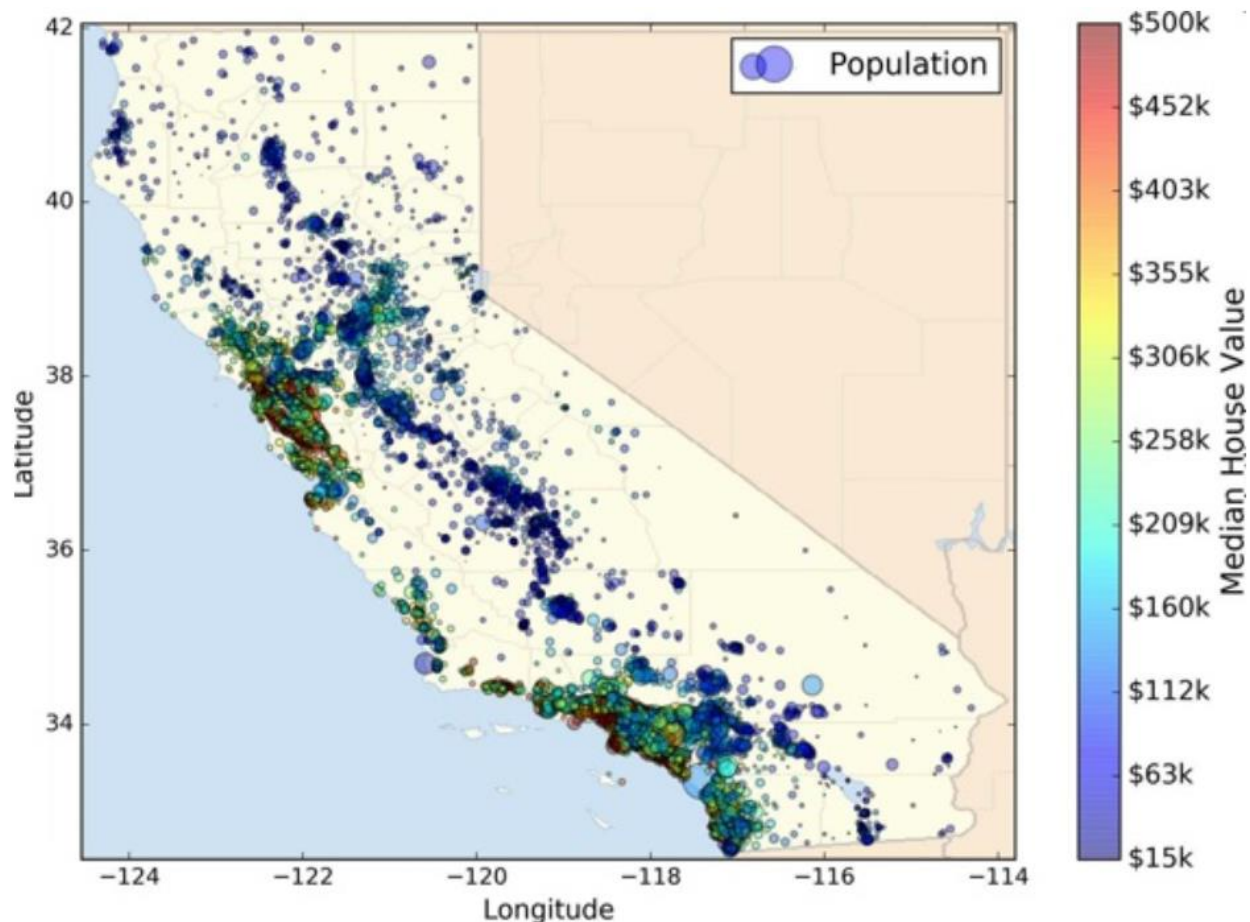


图 2-1 加州房产价格

1.2 进行业务需求分析——What & Why & How?

- 需求分析是：明确客户的真正需求
- 需求分析的结果将决定：（从SE角度）
 - 问题设定（概要设计）
 - 算法选择（详细设计）
 - 模型评估指标（测试）
- 如何进行需求分析？
 - 以问问题的形式
 - Q1：本项目的商业目标是什么？即：公司要如何使用、并从模型受益？
 - Q2：现在有解决方案吗？效果如何？
 - 老板通常会给一个参考性能，以及如何解决问题。

1.2 进行业务需求分析——示例

- 项目概述：利用加州普查数据，建立一个加州房价模型。这个数据包含每个街区组的人口、收入中位数、房价中位数等指标。

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

每行数据代表一个街区，每条数据包含10个属性：经度、纬度、房龄中位数，房间总数，卧室总数，人口数，家庭数，收入中位数，房价中位数，海洋临近度。


```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 20640 entries, 0 to 20639
```

```
Data columns (total 10 columns):
```

longitude	20640	non-null	float64
latitude	20640	non-null	float64
housing_median_age	20640	non-null	float64
total_rooms	20640	non-null	float64
total_bedrooms	20433	non-null	float64
population	20640	non-null	float64
households	20640	non-null	float64
median_income	20640	non-null	float64
median_house_value	20640	non-null	float64
ocean_proximity	20640	non-null	object

符号约定说明

x 标量（整数或实数）

\mathbf{x} : 向量

\mathbf{X} : 矩阵

\mathbf{X} : 张量

第 i 个样本: $\mathbf{x}^{(i)}$

第 i 个样本的第 j 个属性的取值: $\mathbf{x}^{(i)}[j]$

真实值（标签）: \mathbf{y}

预测的值: $\hat{\mathbf{y}}$

- Q1: 本项目的商业目标是什么？即：公司要如何使用、并从模型受益？

- A1:

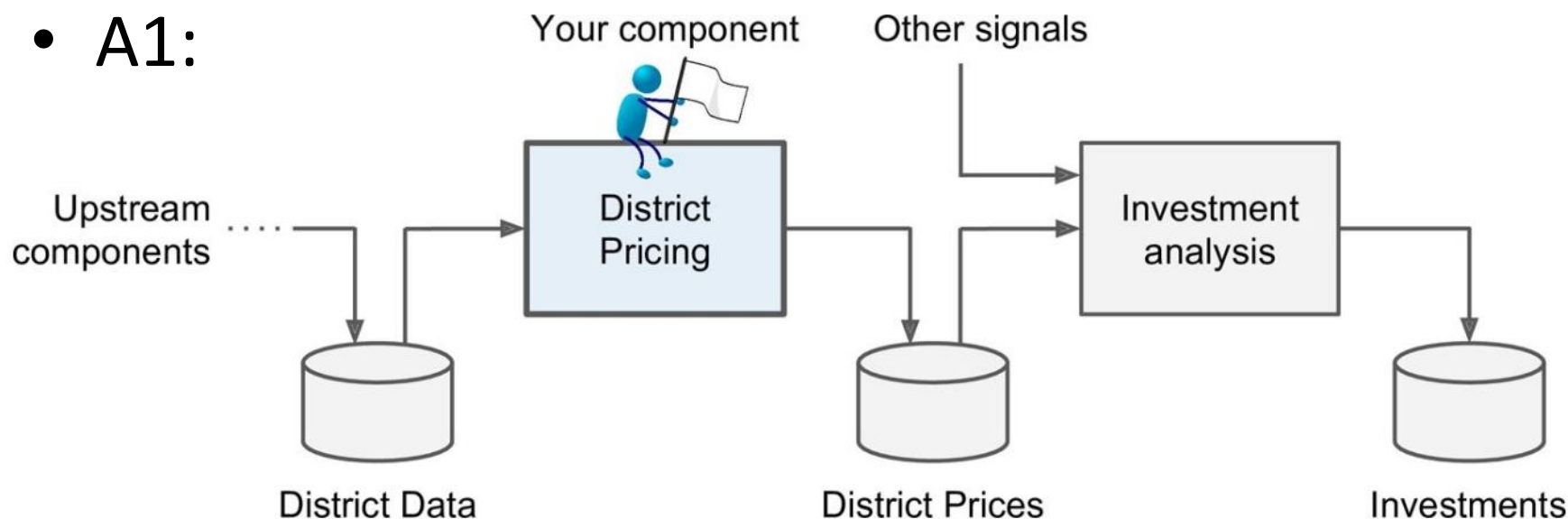


Figure 2-2. A Machine Learning pipeline for real estate investments

输入： 区域数据，每条数据包含每个街区组的人口、收入中位数、房价中位数等指标

输出： 区域价格，该价格被用来决策一个给定的区域是否值得投资

- Q2: 现在有解决方案吗？效果如何？
- A2: 现在街区的房价是靠专家手工估计的，专家队伍收集最新的关于一个区的信息（不包括房价中位数），他们使用复杂的规则进行估计。这种方法费钱费时间，而且估计结果不理想，误差率大概有15%。

1.3 设定问题

- 确定应用场景：
 - 监督学习？ 无监督学习？ 半监督学习？
- 确定任务：
 - 回归？ 分类？ 其他？
- 确定学习类型：（取决于训练数据集是否需要不断在线更新）
 - 批量学习？ 在线学习？

1.3 设定问题——示例

- 回顾：项目需求分析

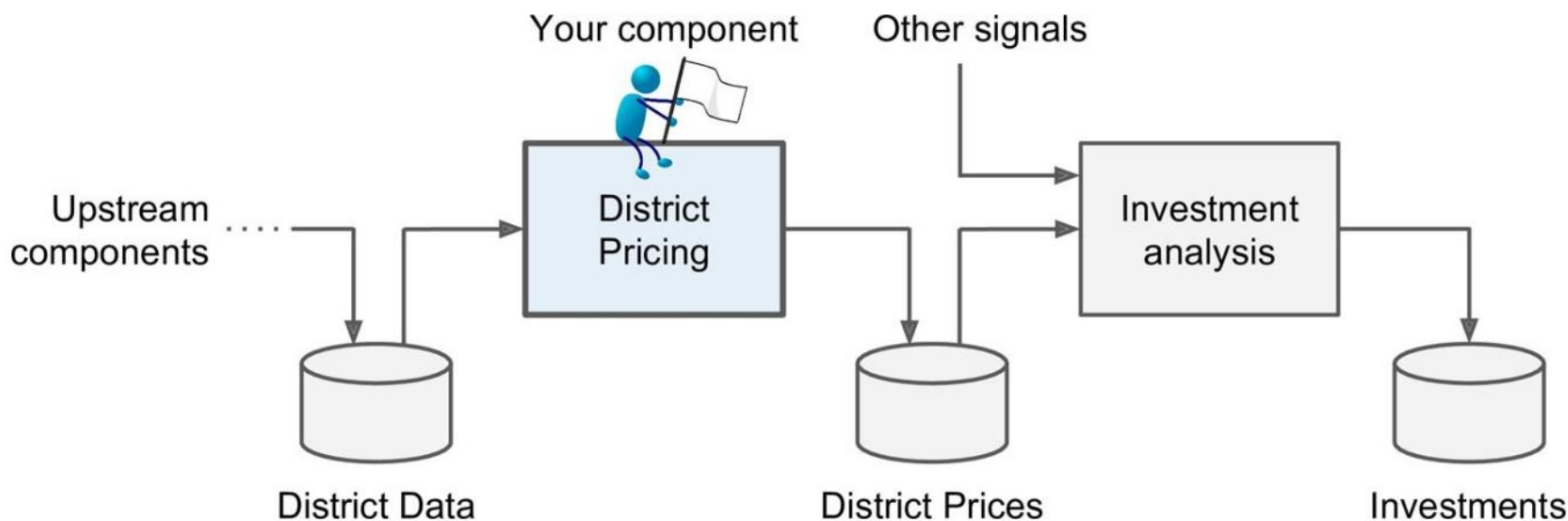


Figure 2-2. A Machine Learning pipeline for real estate investments

输入：区域数据，每条数据包含每个街区组的人口、收入中位数、房价中位数等指标

输出：区域价格，该价格被用来决策一个给定的区域是否值得投资

1.3 设定问题——示例

应用场景：监督学习
任务：回归
学习类型：批量学习

- 回顾：项目需求分析

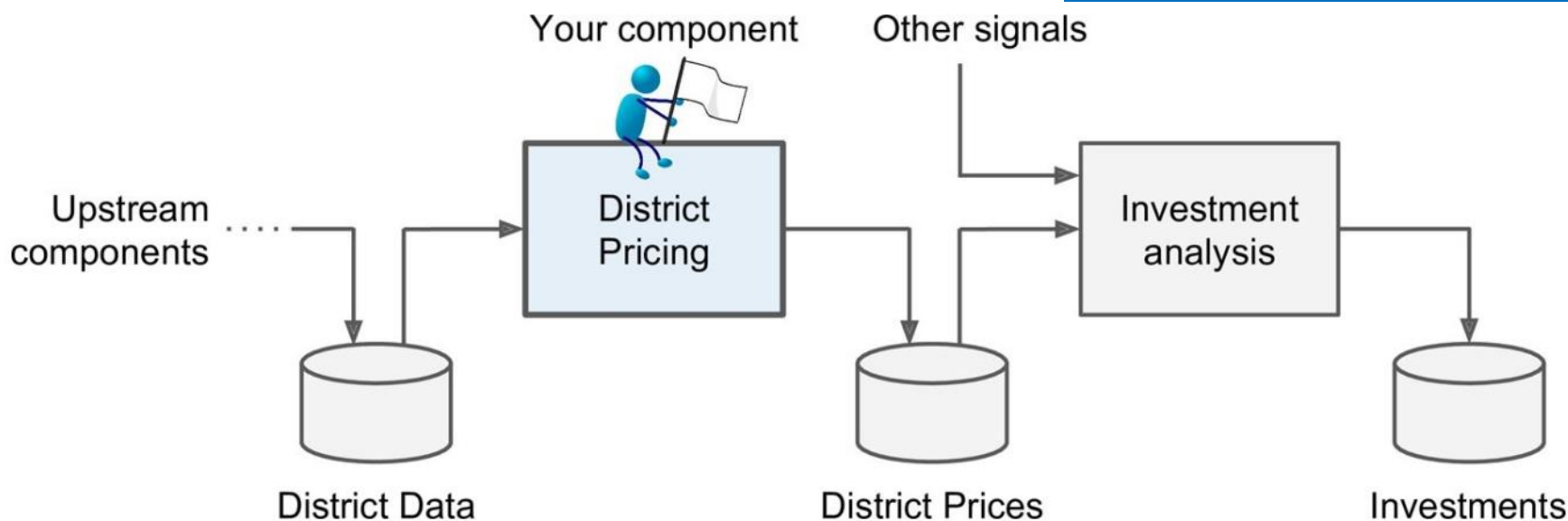


Figure 2-2. A Machine Learning pipeline for real estate investments

输入：区域数据，每条数据包含每个街区组的人口、收入中位数、房价中位数等指标

输出：区域价格，该价格被用来决策一个给定的区域是否值得投资

1.4 选择模型评估指标

- 测量预测值和目标值两个向量距离的方法: **RMSE** 和 **MAE**
- 回归问题的典型指标是均方根误差 (**RMSE**)

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

- 均方根误差测量的是系统预测误差的标准差。例如，RMSE等于50000，意味着，68%的系统预测值位于实际值的50000美元以内，95%的预测值位于实际值的100000美元以内（一个特征通常都符合高斯分布，即满足“68-95-99.7”规则：大约68%的值落在 1σ 内，95%的值落在 2σ 内，99.7%的值落在 3σ 内，这里的 σ 等于50000）。

- **MAE**: 平均绝对误差 (Mean Absolute Error, 也称作平均绝对偏差)

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

1.5 检查问题假设是否正确

- 检查问题设定是否正确
 - 确认模型的输出 正是 下游环节所需要的输入。

2 获取数据

- 2.1 安装Python, Jupyter Notebook, Scikit-Learn, Pandas, Numpy, Matplotlib等
- 2.2 启动Jupyter, 编写"Hello word"测试程序。
- 2.3 编程实现数据的获取、加载和观察
 - 自动获取数据：从指定地址自动获取数据，并解压保存到指定目录下。
 - 加载数据：使用Pandas
 - 快速查看数据结构：

–快速查看数据结构：

- 使用DataFrame的head()方法和info()方法
 - head()方法可查看该数据集的前5行
 - info()方法可以快速查看数据的描述，特别是总行数、每个属性的类型和非空值的数量
 - 使用value_counts()方法查看Object类型的项中都有哪些类别，每个类别中都包含有多少条数据
 - describe()方法展示了数值属性的概括
- 调用hist()方法画出每个数值属性的柱状图

- 每一行都表示一个街区。共有10个属性（截图中可以看到6个）：经度、维度、房屋年龄中位数、总房间数、总卧室数、人口数、家庭数、收入中位数、房屋价值中位数、离大海距离。

```
In [5]: housing = load_housing_data()  
housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

图 2-5 数据集的前五行

- `info()`方法可以快速查看数据的描述，特别是总行数、每个属性的类型和非空值的数量

```
In [6]: housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude                20640 non-null float64
latitude                 20640 non-null float64
housing_median_age       20640 non-null float64
total_rooms              20640 non-null float64
total_bedrooms           20433 non-null float64
population               20640 non-null float64
households               20640 non-null float64
median_income            20640 non-null float64
median_house_value       20640 non-null float64
ocean_proximity          20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

- `info()`方法可以快速查看数据的描述，特别是总行数、每个属性的类型和非空值的数量

```
In [6]: housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 20640 entries, 0 to 20639
```

```
Data columns (total 10 columns):
```

```
longitude
```

```
latitude
```

```
housing_median_age
```

```
total_rooms
```

```
total_bedrooms
```

```
population
```

```
households
```

```
median_income
```

```
median_house_value    20640 non-null float64
```

```
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
```

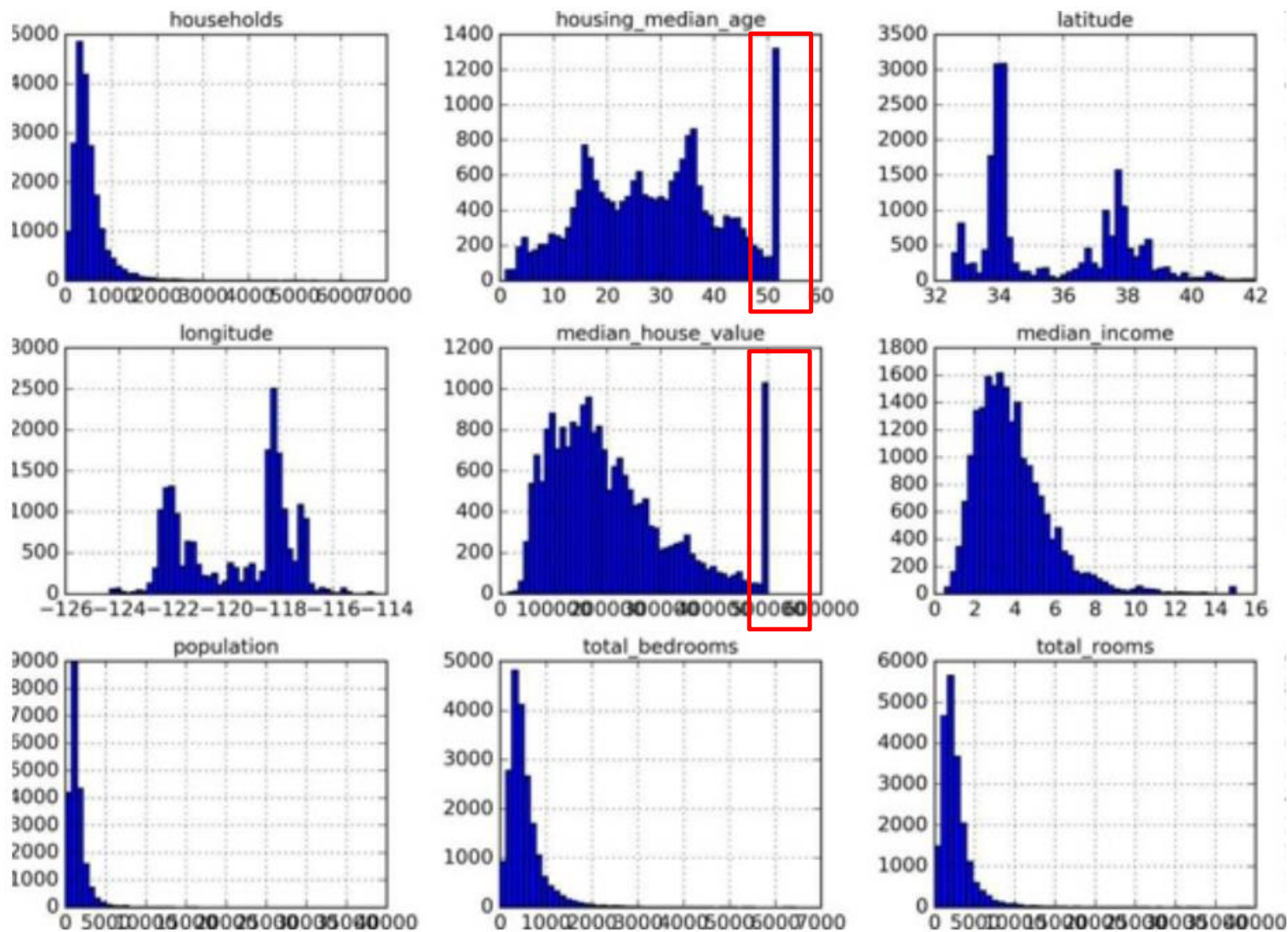


图 2-8 每个数值属性的柱状图

2 获取数据

- 2.4 创建测试集：从完整数据集中随机选择一些实例作为测试集
 - Why？避免数据透视偏差
 - How？
 - 抽样方法的选择：纯随机抽样？分层抽样？
 - 需设置随机数生成器的种子。为什么？
 - 若应用场景中完整数据集总是会更新时，如何创建数据集？
 - 如何判断哪种抽样方法更好？
 - 评判依据：需避免抽样偏差，即测试集中数据的分布规律应与完整数据集是一致的。

- 如何创建测试集？ 避免数据透视偏差
 - 纯随机抽样

```
from sklearn.model_selection import train_test_split  
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

- 分层抽样

- 如何创建测试集？避免数据透视偏差
 - 纯随机抽样

```
from sklearn.model_selection import train_test_split  
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

- 分层抽样

- 增加“收入类别”属性

```
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)  
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

- 根据收入类别，进行分层抽样

```
from sklearn.model_selection import StratifiedShuffleSplit  
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)  
for train_index, test_index in split.split(housing, housing["income_cat"]):  
    strat_train_set = housing.loc[train_index]  
    strat_test_set = housing.loc[test_index]
```

- 创建测试集时，若应用场景中数据集总是更新，那如何保证新的测试数据集中不会出现以前的训练集中的实例呢？

– 利用Hash值

```
import hashlib

def test_set_check(identifier, test_ratio, hash):
    return hash(np.int64(identifier)).digest()[-1] < 256 * test_ratio

def split_train_test_by_id(data, test_ratio, id_column, hash=hashlib.md5):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio, hash))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

– 房价预测问题中，ID值如何选择？

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

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

2.4 创建测试集

— 如何判断哪种抽样方法更好？

- 评判依据：需避免抽样偏差，即测试集中数据的分布规律应与完整数据集是一致的。
- 纯随机抽样 VS. 分层抽样

	Overall	Random	Stratified	Rand. %error	Strat. %error
1.0	0.039826	0.040213	0.039738	0.973236	-0.219137
2.0	0.318847	0.324370	0.318876	1.732260	0.009032
3.0	0.350581	0.358527	0.350618	2.266446	0.010408
4.0	0.176308	0.167393	0.176399	-5.056334	0.051717
5.0	0.114438	0.109496	0.114369	-4.318374	-0.060464

图2-10对比了总数据集、分层采样的测试集、纯随机采样测试集的收入分类比例。可以看到，分层采样测试集的收入分类比例与总数据集几乎相同，而随机采样数据集偏差严重。

图 2-10 分层采样和纯随机采样的样本偏差比较

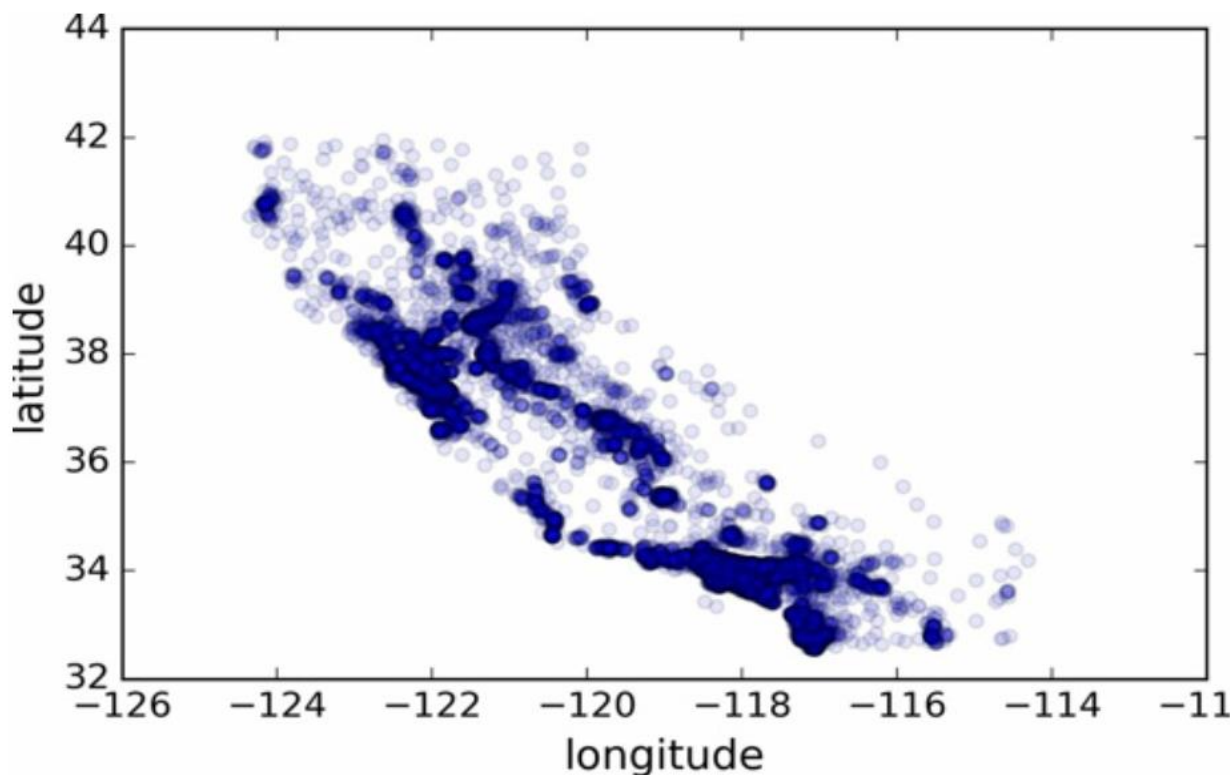
3 研究数据

- 可视化地理数据和房价数据
- 寻找数值类属性之间的相关性
- **Tips:**
 - 只研究训练集。Why?
 - 先备份训练集，再研究它。Why?

3 研究数据

- 可视化地理数据：建立各区域的房产的分布图

```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```



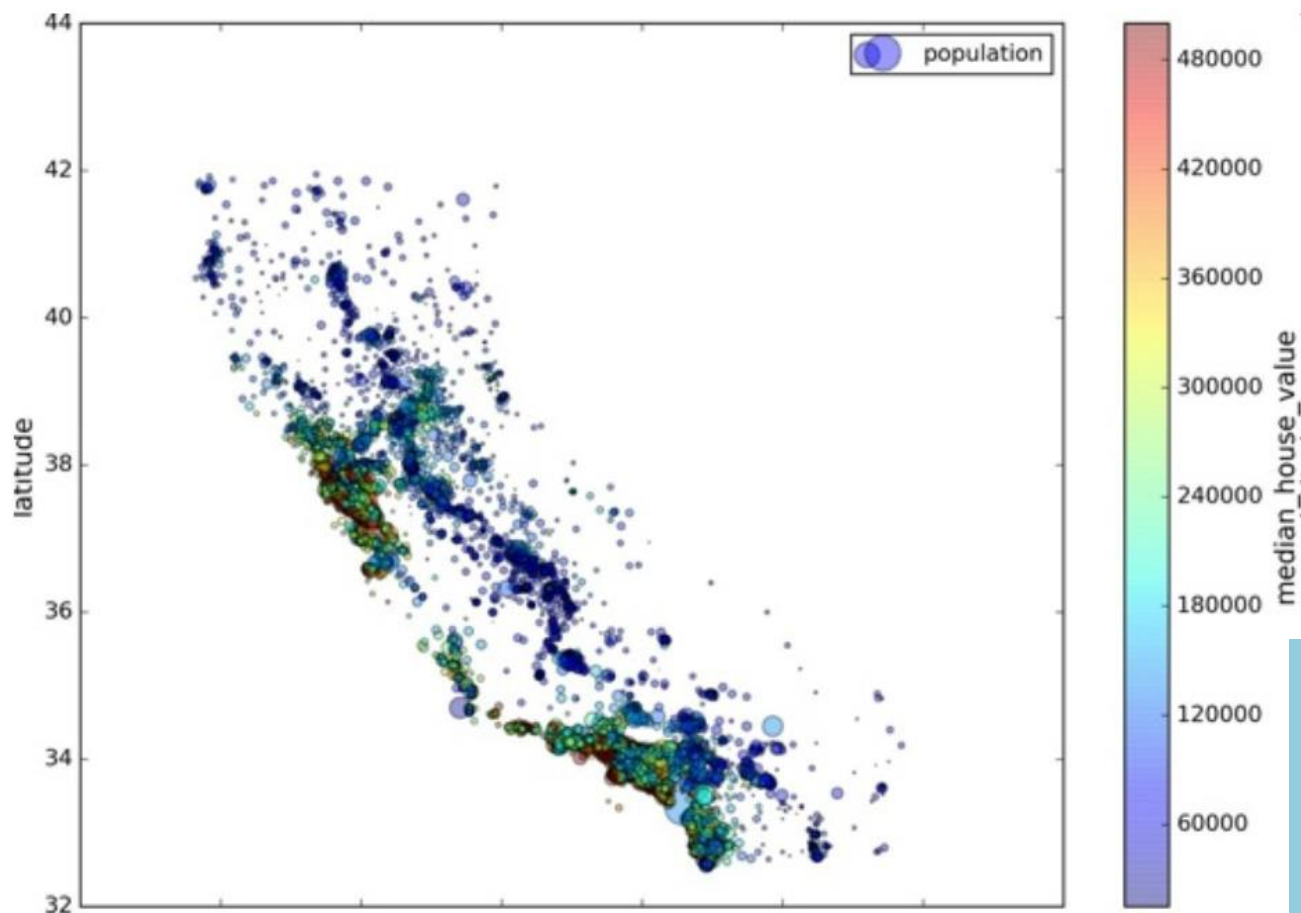
结论：从图中，可以非常清楚地看到高密度区域，比如湾区、洛杉矶和圣迭戈，以及中央谷，特别是从萨克拉门托和弗雷斯诺。

图 2-12 显示高密度区域的散点图

3 研究数据

```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,  
             s=housing["population"]/100, label="population",  
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,  
             )  
plt.legend()
```

- 可视化房价数据



每个圈的半径表示街区的人口（选项**s**），颜色代表价格（选项**c**）。我们用预先定义的名叫**jet**的颜色图（选项**cmap**），它的范围是从蓝色（低价）到红色（高价）

结论：这张图说明房价和位置（比如，靠海）和人口密度联系密切

图 2-13 加州房价

3 研究数据

- 寻找数值类属性之间的相关性：使用corr()方法计算出每对属性间的标准相关系数（standard correlation coefficient，也称作皮尔逊相关系数）

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

每个属性和房价中位数的关联度如下：

```
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.687170
total_rooms          0.135231
housing_median_age    0.114220
households           0.064702
total_bedrooms        0.047865
population            -0.026699
longitude             -0.047279
latitude              -0.142826
Name: median_house_value, dtype: float64
```

相关系数的范围是-1到1。

当接近1时，意味强正相关；例如，当收入中位数增加时，房价中位数也会增加。

当相关系数接近-1时，意味强负相关；你可以看到，纬度和房价中位数有轻微的负相关性（即越往北，房价越可能降低）

3 研究数据

- 寻找数值类属性之间的相关性：使用Pandas的scatter_matrix函数，画出每个数值属性对每个其它数值属性的图

```
from pandas.tools.plotting import scatter_matrix

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

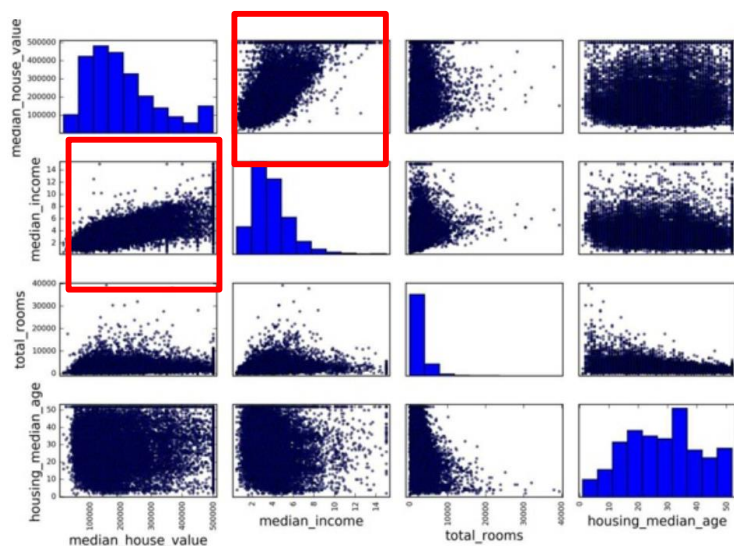


图 2-15 散点矩阵

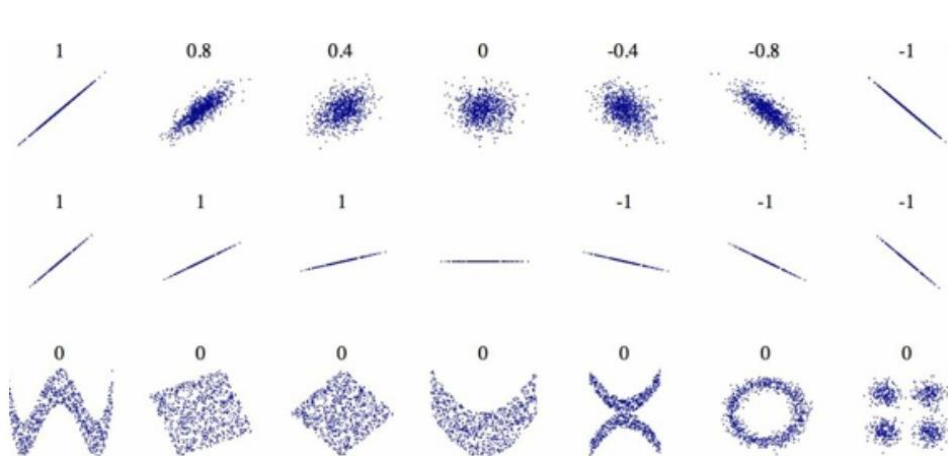
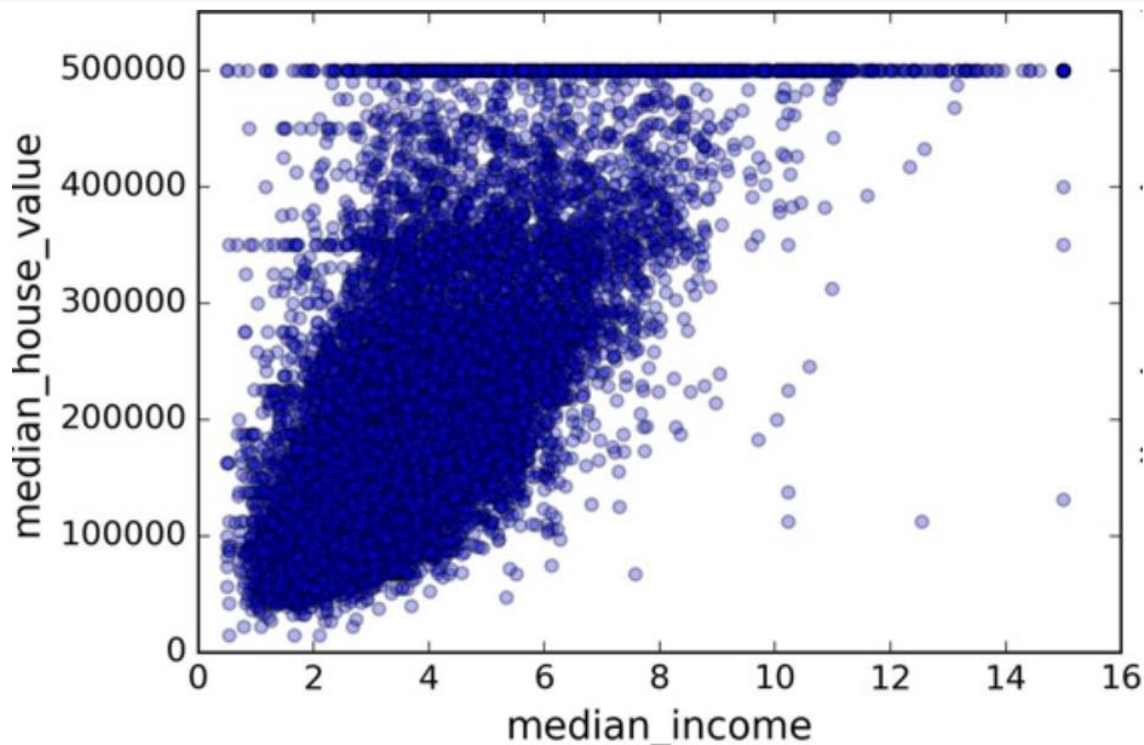


图 2-14 不同数据集的标准相关系数（来源：Wikipedia；公共领域图片）

3 研究数据

- 深入观察：收入中位数与房价中位数之间的相关性

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



结论：

- 1) 相关性非常高；
- 2) 最高价，清晰地呈现为一条位于500000美元的水平线，和一些不太明显的直线：一条位于450000美元的直线，一条位于350000美元的直线，一条在280000美元的线。

3 研究数据

- 试验不同属性的组合，创建新的属性

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

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

median_house_value	1.000000
median_income	0.687170
rooms_per_household	0.199343
total_rooms	0.135231
housing_median_age	0.114220
households	0.064702
total_bedrooms	0.047865
population_per_household	-0.021984
population	-0.026699
longitude	-0.047279
latitude	-0.142826
bedrooms_per_room	-0.260070

Name: median_house_value, dtype: float64

结论：与总房间数或卧室数相比，新的**bedrooms_per_room**属性与房价中位数的关联更强。显然，卧室数/总房间数的比例越低，房价就越高。每户的房间数也比街区的总房间数的更有信息，很明显，房屋越大，房价就越高。

4 准备数据

- 4.1 数据清理：处理缺失值
 - 去掉对应的街区：用DataFrame的dropna()方法
 - 去掉整个属性：用DataFrame的drop()方法
 - 进行赋值（0、平均值、中位数等等）：用DataFrame的fillna()方法 或者Scikit-Learn中的imputer

```
housing.dropna(subset=["total_bedrooms"])      # 选项1
housing.drop("total_bedrooms", axis=1)         # 选项2
median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median)       # 选项3
```

```
from sklearn.preprocessing import Imputer

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

4 准备数据

- 4.2 处理文本和分类属性：将文本类型的属性转换为数字类型
 - 利用转换器LabelEncoder
 - 利用编码器OneHotEncoder
- 4.3 自定义转换器
- 4.4 特征缩放（最重要的转换之一）：让所有的属性有相同的量度
- 4.5 转换流水线：保障数据转换的步骤以正确的顺序来执行

4 准备数据

- 4.4 特征缩放（最重要的转换之一）：让所有的属性有相同的量度
 - 线性函数归一化（Min-Maxscaling）
 - 通过减去最小值，然后再除以最大值与最小值的差值，来进行归一化
 - Scikit-Learn提供了一个转换器MinMaxScaler来实现这个功能，该转换器中的超参数feature_range可以指定缩放后的范围
 - 标准化（standardization）
 - 首先减去平均值（所以标准化值的平均值总是0），然后除以方差，使得到的分布具有单位方差
 - Scikit-Learn提供了一个转换器StandardScaler来进行标准化
- **Tips:**与所有的转换一样，缩放器只能基于训练集进行拟合，而不是面向完整的数据集（包括测试集）

5 研究模型

- 5.1 在训练集上训练

- “三步曲”：

- 选择模型
 - 设计一个损失函数
 - 找到最佳函数（在训练集上进行训练/学习/拟合）

- 在Scikit-Learn中训练一个线性回归模型：

```
from sklearn.linear_model import LinearRegression  
  
lin_reg = LinearRegression()  
lin_reg.fit(housing_prepared, housing_labels)
```

只需要
两步？？

fit函数已经内置了损失函数，并
自动寻找最佳函数来实现拟合

- 5.2 在训练集上评估性能

- 尝试做下预测？

```
>>> some_data = housing.iloc[:5]
>>> some_labels = housing_labels.iloc[:5]
>>> some_data_prepared = full_pipeline.transform(some_data)
>>> print("Predictions:\t", lin_reg.predict(some_data_prepared))
Predictions:      [ 303104.    44800.   308928.   294208.   368704.]
>>> print("Labels:\t\t", list(some_labels))
Labels:      [359400.0, 69700.0, 302100.0, 301300.0, 351900.0]
```


- 使用Scikit-Learn的mean_squared_error函数，在全部训练集上计算改回归模型的RMSE来评估其性能

```
>>> from sklearn.metrics import mean_squared_error
>>> housing_predictions = lin_reg.predict(housing_prepared)
>>> lin_mse = mean_squared_error(housing_labels, housing_predictions)
>>> lin_rmse = np.sqrt(lin_mse)
>>> lin_rmse
68628.413493824875
```

- 换个模型？ 训练一个DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor  
  
tree_reg = DecisionTreeRegressor()  
tree_reg.fit(housing_prepared, housing_labels)
```

```
>>> housing_predictions = tree_reg.predict(housing_prepared)  
>>> tree_mse = mean_squared_error(housing_labels, housing_predictions)  
>>> tree_rmse = np.sqrt(tree_mse)  
>>> tree_rmse  
0.0
```



完美模型？
过拟合？

- 5.3 跨模型的选择更佳的模型：基于验证集
 - 用函数`train_test_split`来分割训练集，得到一个更小的训练集和一个验证集，然后用更小的训练集来训练模型，用验证集来评估。
 - 使用Scikit-Learn的交叉验证功能：K折交叉验证
- （K-fold cross-validation）：它随机地将训练集分成十个不同的子集，成为“折”，然后训练评估决策树模型K次，每次选一个不用的折来做评估，用其它（K-1）个来做训练。

- 使用10折交叉验证来对比决策树和线性回归模型的性能：

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                          scoring="neg_mean_squared_error", cv=10)
rmse_scores = np.sqrt(-scores)
```

```
>>> def display_scores(scores):
...     print("Scores:", scores)
...     print("Mean:", scores.mean())
...     print("Standard deviation:", scores.std())
...
>>> display_scores(tree_rmse_scores)
Scores: [ 74678.4916885   64766.2398337   69632.86942005   69166.67693232
          71486.76507766   73321.65695983   71860.04741226   71086.32691692
          76934.2726093    69060.93319262]
Mean: 71199.4280043
Standard deviation: 3202.70522793
```

```
>>> lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
...                               scoring="neg_mean_squared_error", cv=10)
...
>>> lin_rmse_scores = np.sqrt(-lin_scores)
>>> display_scores(lin_rmse_scores)
Scores: [ 70423.5893262    65804.84913139  66620.84314068  72510.11362141
          66414.74423281  71958.89083606  67624.90198297  67825.36117664
          72512.36533141  68028.11688067]
Mean: 68972.377566
Standard deviation: 2493.98819069
```

- 结论：决策树模型过拟合很严重，它的性能比线性回归模型还差

- 再尝试第三种模型：随机森林模型
RandomForestRegressor

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> forest_reg = RandomForestRegressor()
>>> forest_reg.fit(housing_prepared, housing_labels)
>>> [...]
>>> forest_rmse
22542.396440343684
>>> display_scores(forest_rmse_scores)
Scores: [ 53789.2879722  50256.19806622  52521.55342602  53237.44937943
          52428.82176158  55854.61222549  52158.02291609  50093.66125649
          53240.80406125  52761.50852822]
Mean: 52634.1919593
Standard deviation: 1576.20472269
```

6 微调模型

- 目标：提高模型泛化性能

6.1 选择最佳超参数，找到最好的模型

- 网格搜索：使用Scikit-Learn的GridSearchCV
- 随机搜索：当超参数的搜索空间很大时，最好使用RandomizedSearchCV

6.2 集成模型：将表现最好的模型组合起来

6.3 分析最佳模型和它们的误差：便于搞清为什么会有些误差，以及如何改正问题（添加更多的特征，或相反，去掉没有什么信息的特征，清洗异常值等等）

6.4 用测试集评估系统

- 用测试集评估最后的模型。
- 具体处理过程：从测试集得到预测值和标签，运行`full_pipeline`转换数据（调用`transform()`，而不是`fit_transform()`！），再用测试集评估最终模型

7 展示解决方案

- 文档化你所做的工作
- 创建完美的演示
- 解释为什么你的解决方案达到了业务目标
- 其他需说明之处：
 - 列出你的假设和系统的局限

8 启动、监视、维护系统

- 接入输入数据源，并编写测试
- 编写监控代码
- 自动化地定期用新数据训练模型

谢谢！