SAIF

Shanghai Advanced Institute of Finance
上海高级金融学院

量化俱乐部-机器学习-E2E ML Project

2019-08-04

Menu

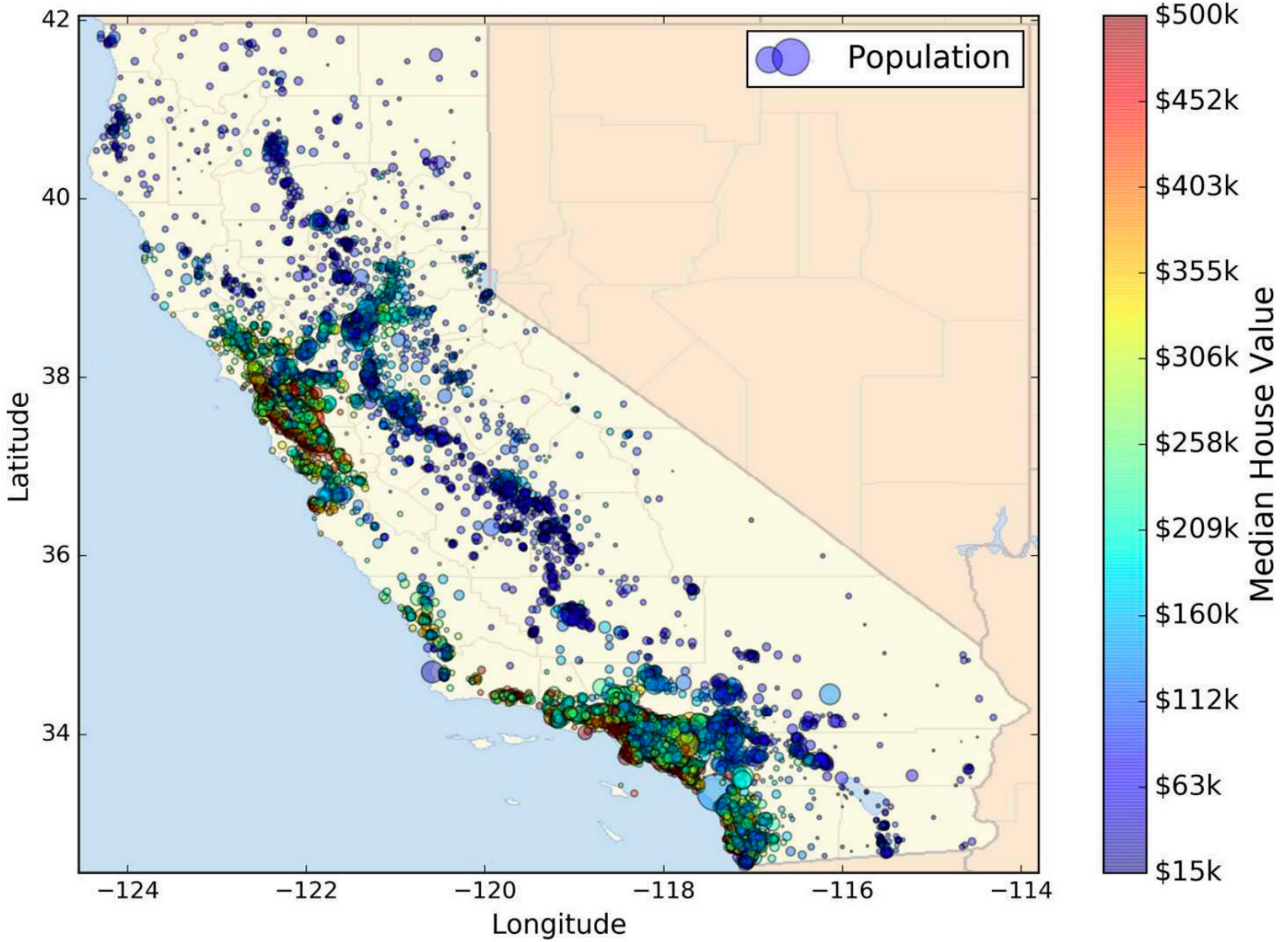
In this session we will go through an example project end to end, pretending to be a recently hired data scientist in a real estate company. Here are the main steps you will go through:

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

Self-Introduction

张骁喆，2010年大连理工大学软件学院本科毕业，高金FMBA 2017PTE。9年计算机工作经验，熟悉开发，测试，运维，项目管理，产品设计各环节。
曾就职eBay，唯品会，2016加入SAP，现担任SAP Cloud部门开发团队主管。
2017开始接触量化投资和python，目前毕业论文研究方向使用机器学习在A股进行量化投资。
量化投资/机器学习咨询培训/项目管理产品管理。

Look at the big picture



Select a Performance Measure

A typical performance measure for regression problems is the Root Mean Square Error (RMSE)

Equation 2-1. Root Mean Square Error (RMSE)

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

We may consider using the Mean Absolute Error as well

Equation 2-2. Mean Absolute Error

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

Get the data

Software installations and libraries&data downloading:

1. Python 3.6.8 and jupyter
2. Download the data

In [2]:
```
def load_housing_data(housing_path):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

In [3]:
```
### 读取初始数据 ###
housing = load_housing_data(r"/Users/i328815/Documents/Notes_HandsOn_ML/datasets")
```

Get the data

Take a look at the data structure:

1. Take a look at the top five rows using the DataFrame’s head() method

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

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
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

Get the data

```
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
```

Get the data

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

Get the data

Let's look at the other fields. The describe() method shows a summary of the numerical attributes

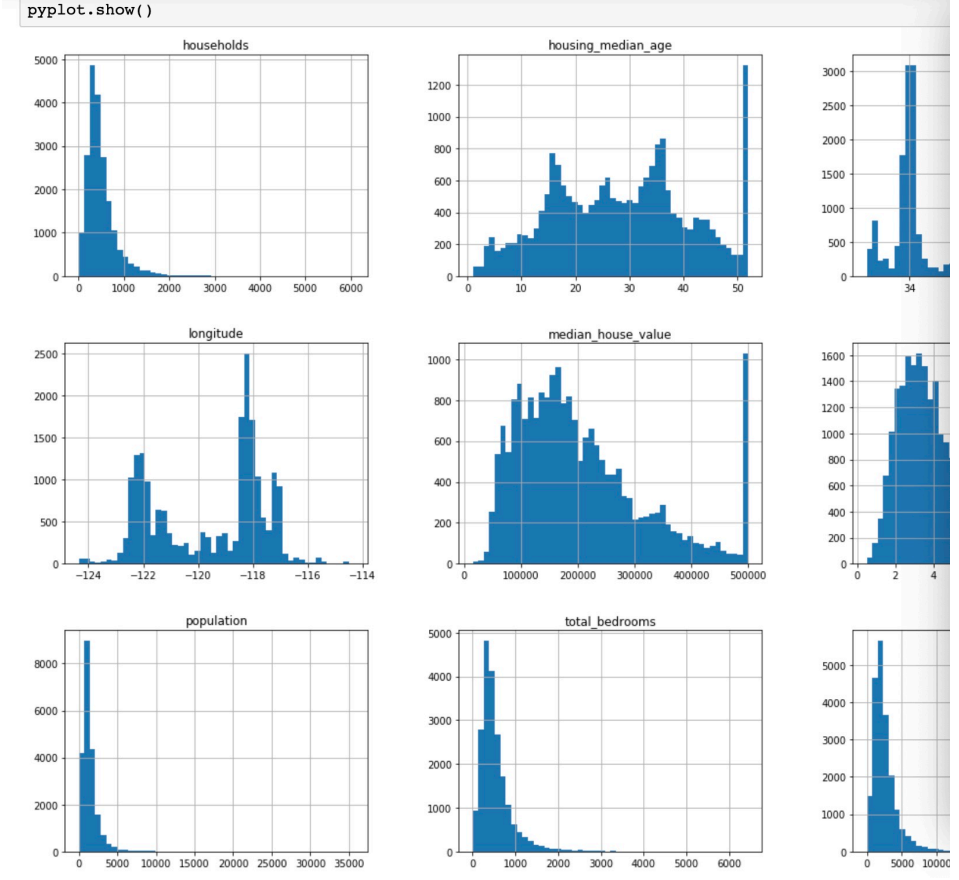
```
#describe() method shows a summary of the numerical attributes
housing.describe()

#展示每一列的直方图
#housing.hist(bins=50, figsize=(20,15))
#pyplot.show()
```

Out[14]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_in
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.0
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.8
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.8
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.4
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.5
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.5
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.7
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.0

Get the data

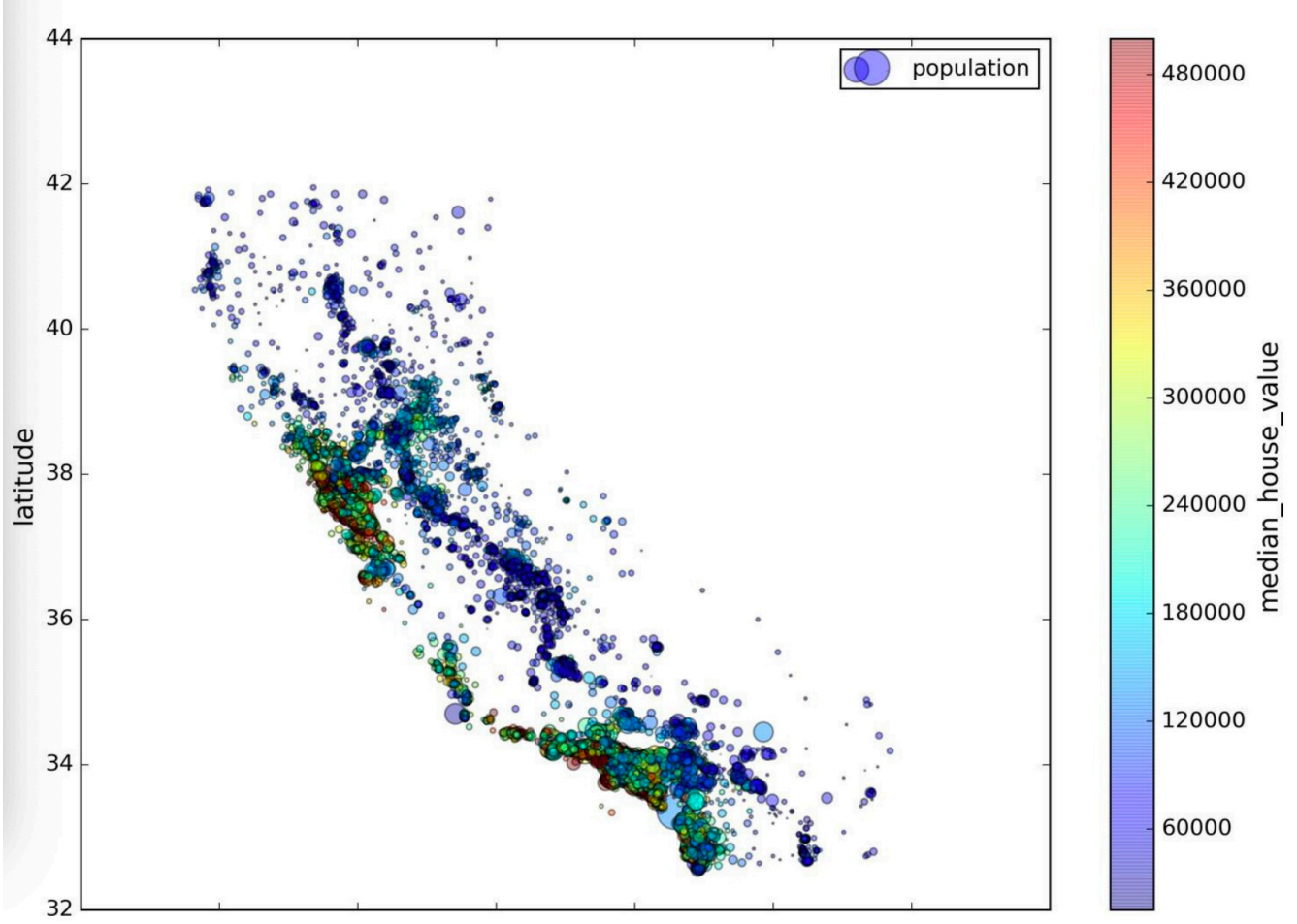
Another quick way to get a feel of the type of data you are dealing with is to plot a histogram for each numerical attribute



Discover and Visualize to Gain Insights

Shanghai Advanced Institute of Finance 上海高级金融学院

Visualizing Geographical Data



AACSB ACCREDITED

Discover and Visualize to Gain Insights

Looking for Correlations:
Since the dataset is not too large, you can easily compute the standard correlation coefficient, let’ s look at how much each attribute correlates with the median house value:

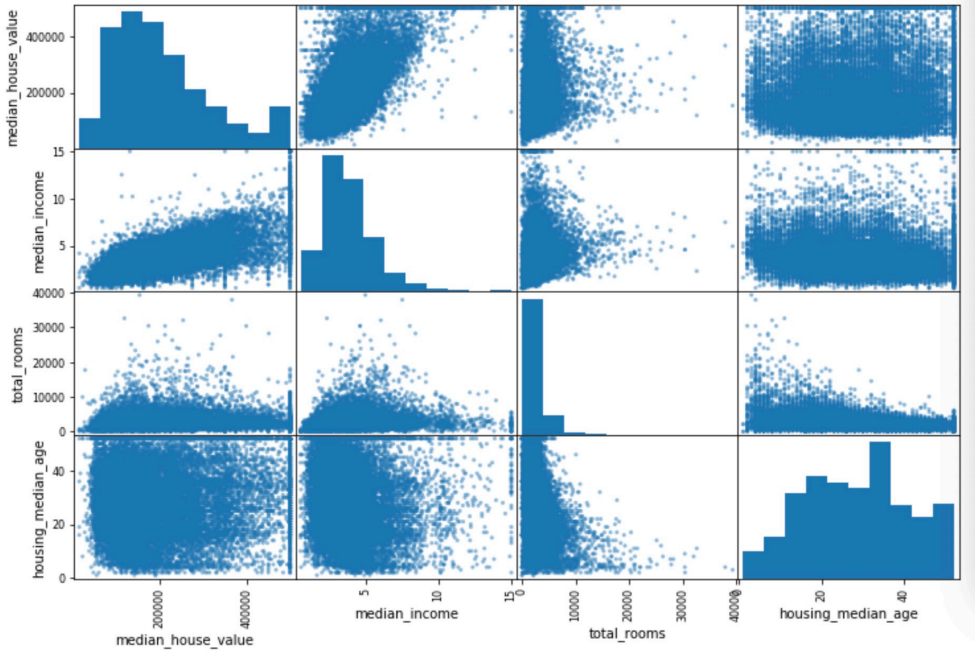
```
In [17]: corr_matrix = housing.corr()
         print(corr_matrix["median_house_value"].sort_values(ascending=False))
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

Discover and Visualize to Gain Insights

Another way to check for correlation between attributes is to use Pandas' scatter_matrix function, which plots every numerical attribute against every other numerical attribute.

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

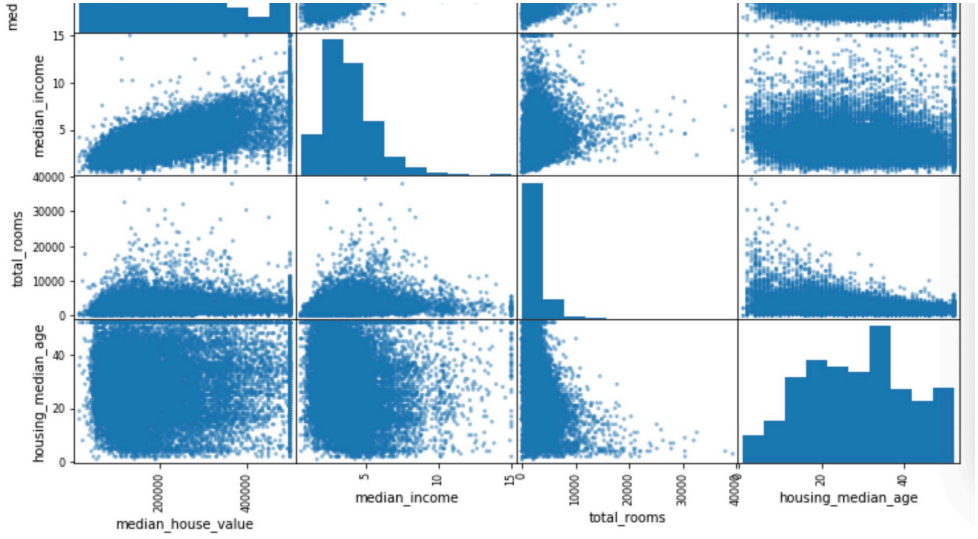


Discover and Visualize to Gain Insights

The most promising attribute to predict the median house value is the median income, so let’ s zoom in on their correlation scatterplot

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

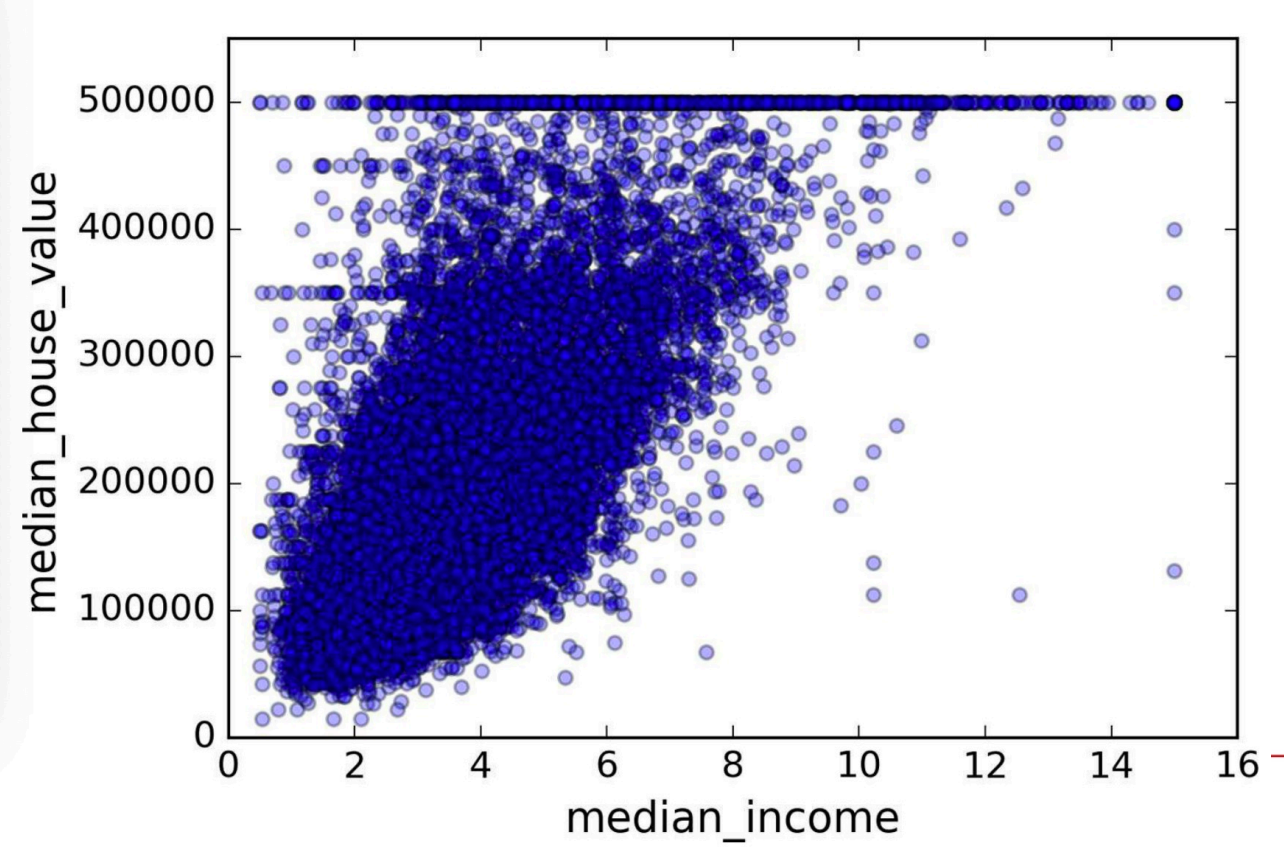
```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
```



Discover and Visualize to Gain Insights

The most promising attribute to predict the median house value is the median income, so let’ s zoom in on their correlation scatterplot

```
#median_income 和 median_house_value 强相关
housing.plot(kind="scatter", x="median_income", y="median_house_value",alpha=0.1)
plt.show()
```



Discover and Visualize to Gain Insights

Shanghai Advanced Institute of Finance 上海高级金融学院

Experimenting with Attribute Combinations

```
In [7]: 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()
        print(corr_matrix["median_house_value"].sort_values(ascending=False))
```

```
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

Prepare the data for ML

Data Cleaning:
Most Machine Learning algorithms cannot work with missing features, so let’ s create a few functions to take care of them. You noticed earlier that the total_bedrooms attribute has some missing values, so let’ s fix this. You have three options:
1. Get rid of the corresponding districts.
2. Get rid of the whole attribute.
3. Set the values to some value (zero, the mean, the median, etc.

Scikit-Learn provides a handy class to take care of missing values: Imputer.
Here is how to use it. First, you need to create an Imputer instance, specifying that you want to replace each attribute’s missing values with the median of that attribute:

In [20]:
```
#数字类型数值缺失的用平均数补上
imputer = SimpleImputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)

# replacing missing values by the learned medians
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
```

Prepare the data for ML

Handling Text and Categorical Attributes

```
#Handling Text and Categorical Attributes
# P117  from text categories to integer then from integer to one-hot ve
encoder_bin = LabelBinarizer()
housing_cat = housing["ocean_proximity"]
housing_cat_1hot = encoder_bin.fit_transform(housing_cat)
print(housing_cat_1hot)
print(type(housing_cat_1hot))
```

```
[[0 0 0 1 0]
 [0 0 0 1 0]
 [0 0 0 1 0]
 ...
 [0 1 0 0 0]
 [0 1 0 0 0]
 [0 1 0 0 0]]
<class 'numpy.ndarray'>
```

Prepare the data for ML

Custom Transformers
Although Scikit-Learn provides many useful transformers, you will need to write your own for tasks such as custom cleanup operations or combining specific attributes. For example, here is a small transformer class that adds the combined attributes we discussed earlier

In [10]:
```
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

Prepare the data for ML

Feature Scaling:

One of the most important transformations you need to apply to your data is feature scaling. With few exceptions, Machine Learning algorithms don't perform well when the input numerical attributes have very different scales. This is the case for the housing data: the total number of rooms ranges from about 6 to 39,320, while the median incomes only range from 0 to 15. Note that scaling the target values is generally not required. There are two common ways to get all attributes to have the same scale: min-max scaling and standardization.

Prepare the data for ML

Transformation Pipelines:

As you can see, there are many data transformation steps that need to be executed in the right order. Fortunately, Scikit-Learn provides the Pipeline class to help with such sequences of transformations

In [11]:
```
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])
```

Select and Train a Model

Training and Evaluating on the Training Set

```
# 准备num的pipline和cat的pipeline
imputer = SimpleImputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
# print(housing_prepared)
# print(housing_prepared.shape)
# print(housing.columns)
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
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))
print("Labels:\t\t", list(some_labels))
```

```
Predictions:     [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
Labels:          [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

Select and Train a Model

Let's measure this regression model's RMSE on the whole training set using Scikit-Learn's mean_squared_error function:

```
>>> 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
```

```
In [23]: from sklearn.metrics import mean_absolute_error

         lin_mae = mean_absolute_error(housing_labels, housing_predictions)
         lin_mae

Out[23]: 49439.89599001897
```

Select and Train a Model

Better Evaluation Using Cross-Validation

Fine-Tune Your Model

Grid Search:

One way to do that would be to fiddle with the hyperparameters manually, until you find a great combination of hyperparameter values. This would be very tedious work, and you may not have time to explore many combinations.

Instead you should get Scikit-Learn's GridSearchCV to search for you. All you need to do is tell it which hyperparameters you want it to experiment with, and what values to try out, and it will evaluate all the possible combinations of hyperparameter values, using cross-validation.

Fine-Tune Your Model

Evaluate Your System on the Test Set:

After tweaking your models for a while, you eventually have a system that performs sufficiently well. Now is the time to evaluate the final model on the test set. There is nothing special about this process; just get the predictors and the labels from your test set, run your full_pipeline to transform the data (call transform(), not fit_transform()!), and evaluate the final model on the test set.

Launch, Monitor, and Maintain Your System

As you can see, much of the work is in the data preparation step, building monitoring tools, setting up human evaluation pipelines, and automating regular model training. The Machine Learning algorithms are also important, of course, but it is probably preferable to be comfortable with the overall process and know three or four algorithms well rather than to spend all your time exploring advanced algorithms and not enough time on the overall process.

SAIF

Shanghai Advanced
Institute of Finance
上海高级金融学院