Branch: master

Find file | Copy path

ml / machine_learning / end_to_end_project.ipynb

rajtilak82 Imputer

73229aa on Apr 9

3 contributors

Download | History

1.09 MB

# End-to-end Machine Learning project

The best way to learning any programming language or new concept is to do hands-on on that. Let's start with buidling machine learning model

## Problem Statement

Welcome to Machine Learning Housing Corp.! Your task is to predict median house values in Californian districts, given a number of features from these districts.

## Dataset

Dataset is based on data from the 1990 California census. It is located at `datasets/housing/housing.csv`

# Get the data

In [1]:

```
import pandas as pd
import os

HOUSING_PATH = 'datasets/housing/'
def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housin
g.csv")
    return pd.read_csv(csv_path)
```

In [2]:

```
housing = load_housing_data()
housing.head()
```

Out[2]:

| | longitude | latitude | housing_median_age | total_rooms |
|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41.0 | 880.0 |
| 1 | -122.22 | 37.86 | 21.0 | 7099.0 |
| 2 | -122.24 | 37.85 | 52.0 | 1467.0 |
| 3 | -122.25 | 37.85 | 52.0 | 1274.0 |
| 4 | -122.25 | 37.85 | 52.0 | 1627.0 |

Each row represents one district. There are 10 attributes:

longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, median_house_value and ocean_proximity

In [3]:

```
# The info() method is useful to get a quick descr
iption of the data
# in particular the total number of rows,
# and each attribute's type and number of non-null
values

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
```

There are 20,640 instances in the dataset.

Notice that the total_bedrooms attribute has only 20,433 non null values, meaning that 207 districts are missing this feature.

All attributes are numerical, except the ocean_proximity field. Its type is object, so it could hold any kind of Python object, but since you loaded this data from a CSV file you know that it must be a text attribute.

When you looked at the top five rows, you probably noticed that the values in that column were repetitive, which means that it is probably a categorical attribute.

In [4]:

```
# Find out what categories exist
# and how many districts belong to each category b
y using the value_counts() method

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

Out[4]:

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
```

```
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [5]:

```
# Let's look at the other fields.
# The describe() method shows a summary of the num
erical attributes

housing.describe()
```

Out[5]:

| | longitude | latitude | housing_median_age |
|---|---|---|---|
| count | 20640.000000 | 20640.000000 | 20640.000000 |
| mean | -119.569704 | 35.631861 | 28.639486 |
| std | 2.003532 | 2.135952 | 12.585558 |
| min | -124.350000 | 32.540000 | 1.000000 |
| 25% | -121.800000 | 33.930000 | 18.000000 |
| 50% | -118.490000 | 34.260000 | 29.000000 |
| 75% | -118.010000 | 37.710000 | 37.000000 |
| max | -114.310000 | 41.950000 | 52.000000 |

The count, mean, min and max rows are self-explanatory.

Note the count of total_bedrooms is 20,433, not 20,640. It means that null values are ignored

**std** rows shows the standard deviation (which measures how dispersed the values are)

**25%, 50%, 75%** shows the corresponding percentiles

**Points to Note**

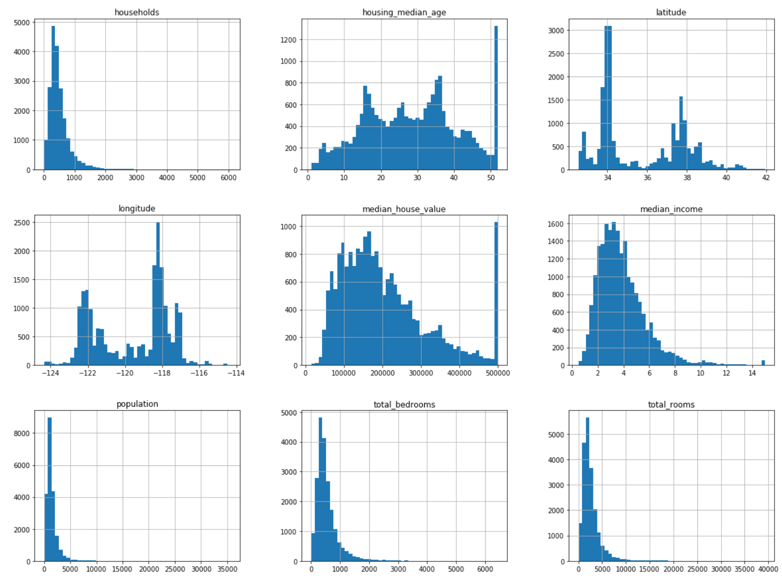
1. **25th percentile is called 1st quartile** - 25% of the districts have a housing_median_age lower than 18.
2. **50th percentile is called median** - 50% of the districts have a housing_median_age lower than 29.
3. **75th percentile is called 3rd quartile** - 75% of the districts have a housing_median_age lower than 37.

# Go back to slide - Plot histogram

In [6]:

```
# Let's plot a histogram to get the feel of type o
```

```
# Let's plot a histogram to get the feel of type o
f data we are dealing with
# We can plot histogram only for numerical attribu
tres

%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```




Refer to slide [Things to Note in Histogram]

In [7]:

```
# To make this notebook's output identical at ever
y run

import numpy as np

np.random.seed(42)
```

In [8]:

```
# For illustration only. Sklearn has train_test_sp
lit()

def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(d
ata))
    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[tes
t_indices]

train_set, test_set = split_train_test(housing, 0.
2)
print(len(train_set), "train +", len(test_set), "t
est")
```

```
est")
```

16512 train + 4128 test

In [9]:

```
# With unique and immutable identifier

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_co
lumn, hash=hashlib.md5):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_c
heck(id_, test_ratio, hash))
    return data.loc[~in_test_set], data.loc[in_tes
t_set]

housing_with_id = housing.reset_index()   # adds a
n `index` column
train_set, test_set = split_train_test_by_id(housi
ng_with_id, 0.2, "index")

print(len(train_set), "train +", len(test_set), "t
est")
```

16362 train + 4278 test

In [10]:

```
test_set.head()
```

Out[10]:

| | index | longitude | latitude | housing_median_age | total_ |
|---|---|---|---|---|---|
| 4 | 4 | -122.25 | 37.85 | 52.0 | 1627. |
| 5 | 5 | -122.25 | 37.85 | 52.0 | 919.0 |
| 11 | 11 | -122.26 | 37.85 | 52.0 | 3503. |
| 20 | 20 | -122.27 | 37.85 | 40.0 | 751.0 |
| 23 | 23 | -122.27 | 37.84 | 52.0 | 1688. |

In [11]:

```
# Combining latitude and longitude into an ID

housing_with_id["id"] = housing["longitude"] * 100
0 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housi
ng_with_id, 0.2, "id")

print(len(train_set), "train +", len(test_set), "t
```

```
est")

test_set.head()
```

16267 train + 4373 test

Out[11]:

| | index | longitude | latitude | housing_median_age | total_ |
|---|---|---|---|---|---|
| 8 | 8 | -122.26 | 37.84 | 42.0 | 2555. |
| 10 | 10 | -122.26 | 37.85 | 52.0 | 2202. |
| 11 | 11 | -122.26 | 37.85 | 52.0 | 3503. |
| 12 | 12 | -122.26 | 37.85 | 52.0 | 2491. |
| 13 | 13 | -122.26 | 37.84 | 52.0 | 696.0 |

In [12]:

```
np.random.seed(1)
np.random.permutation(4)
```

Out[12]:

```
array([3, 2, 0, 1])
```

In [13]:

```
from sklearn.model_selection import train_test_spl
it

help(train_test_split)
```

```
Help on function train_test_split in module sklear
n.model_selection._split:

train_test_split(*arrays, **options)
    Split arrays or matrices into random train and
test subsets

    Quick utility that wraps input validation and
    ``next(ShuffleSplit().split(X, y))`` and appli
cation to input data
    into a single call for splitting (and optional
ly subsampling) data in a
    oneliner.

    Read more in the :ref:`User Guide <cross_valid
ation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same len
gth / shape[0]
        Allowed inputs are lists, numpy arrays, sc
ipy-sparse
        matrices or pandas dataframes.
```

```
        matrices or pandas dataframes.

    test_size : float, int or None, optional (defa
ult=None)
        If float, should be between 0.0 and 1.0 an
d represent the proportion
        of the dataset to include in the test spli
t. If int, represents the
        absolute number of test samples. If None,
the value is set to the
        complement of the train size. If ``train_s
ize`` is also None, it will
        be set to 0.25.

    train_size : float, int, or None, (default=Non
e)
        If float, should be between 0.0 and 1.0 an
d represent the
        proportion of the dataset to include in th
e train split. If
        int, represents the absolute number of tra
in samples. If None,
        the value is automatically set to the comp
lement of the test size.

    random_state : int, RandomState instance or No
ne, optional (default=None)
        If int, random_state is the seed used by t
he random number generator;
        If RandomState instance, random_state is t
he random number generator;
        If None, the random number generator is th
e RandomState instance used
        by `np.random`.

    shuffle : boolean, optional (default=True)
        Whether or not to shuffle the data before
splitting. If shuffle=False
        then stratify must be None.

    stratify : array-like or None (default=None)
        If not None, data is split in a stratified
fashion, using this as
        the class labels.

    Returns
    -------
    splitting : list, length=2 * len(arrays)
        List containing train-test split of input
s.

        .. versionadded:: 0.16
            If the input is sparse, the output wil
l be a
            ``scipy.sparse.csr_matrix``. Else, out
put type is the same as the
            input type.
```

```
    Examples
    --------
    >>> import numpy as np
    >>> from sklearn.model_selection import train_
test_split
    >>> X, y = np.arange(10).reshape((5, 2)), rang
e(5)
    >>> X
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7],
           [8, 9]])
    >>> list(y)
    [0, 1, 2, 3, 4]

    >>> X_train, X_test, y_train, y_test = train_t
est_split(
    ...     X, y, test_size=0.33, random_state=42)
    ...
    >>> X_train
    array([[4, 5],
           [0, 1],
           [6, 7]])
    >>> y_train
    [2, 0, 3]
    >>> X_test
    array([[2, 3],
           [8, 9]])
    >>> y_test
    [1, 4]

    >>> train_test_split(y, shuffle=False)
    [[0, 1, 2], [3, 4]]
```

In [14]:

```
# With sklearn train_test_split

from sklearn.model_selection import train_test_spl
it

train_set, test_set = train_test_split(housing, te
st_size=0.2, random_state=42)

print(len(train_set), "train +", len(test_set), "t
est")

test_set.head()
```

16512 train + 4128 test

Out[14]:

| | longitude | latitude | housing_median_age | total_roc |
|---|---|---|---|---|
| 20046 | -119.01 | 36.06 | 25.0 | 1505.0 |
| 3024 | -119.46 | 35.14 | 30.0 | 2943.0 |

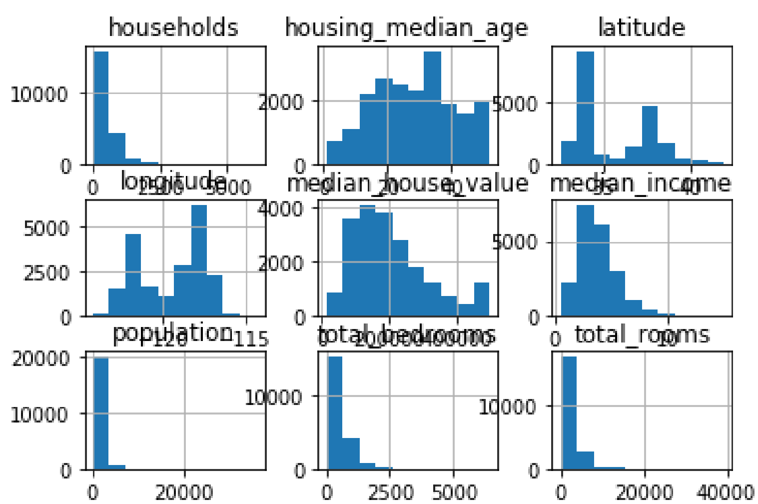
| | | | | |
|---|---|---|---|---|
| 15663 | -122.44 | 37.80 | 52.0 | 3830.0 |
| 20484 | -118.72 | 34.28 | 17.0 | 3051.0 |
| 9814 | -121.93 | 36.62 | 34.0 | 2351.0 |

In [15]:

```
housing.hist()
```

Out[15]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C4CFFAC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C49EFDC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C4B727C8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C4D39EC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C4C1B508>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C447B288>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C4B69308>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C4982F08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C4982FC8>]],
      dtype=object)
```

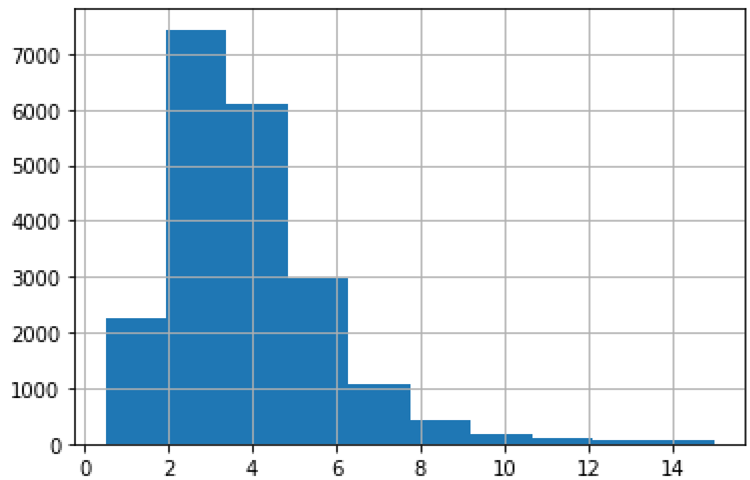



In [16]:

```
# Create a histrogram of median income

housing["median_income"].hist()
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2c4b5b608>
```




In [17]:

```
np.ceil(1.1)
```

Out[17]:

```
2.0
```

In [18]:

```
# Divide by 1.5 to limit the number of income cate
gories
# Round up using ceil to have discrete categories
housing["income_cat"] = np.ceil(housing["median_in
come"] / 1.5)
```

In [19]:

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

Out[19]:

```
3.0     7236
2.0     6581
4.0     3639
5.0     1423
1.0      822
6.0      532
7.0      189
8.0      105
9.0       50
11.0      49
10.0      14
Name: income_cat, dtype: int64
```

In [20]:

```
# Label those above 5 as 5
housing["income_cat"].where(housing["income_cat"]
< 5, 5.0, inplace=True)
```

In [21]:

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

Out[21]:

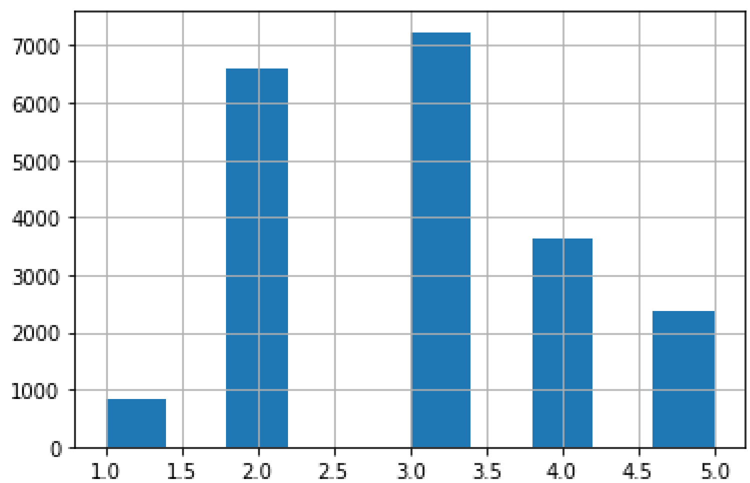
```
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0     822
Name: income_cat, dtype: int64
```

In [22]:

```
housing["income_cat"].hist()
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2c4a
aea08>
```




In [23]:

```
# Stratified Sampling using Scikit-learn's Stratif
iedShuffleSplit Class

from sklearn.model_selection import StratifiedShuf
fleSplit

split = StratifiedShuffleSplit(n_splits=1, test_si
ze=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]
```

In [24]:

```
# Income category proportion in test set generated
with stratified sampling
strat_test_set["income_cat"].value_counts() / len(
strat_test_set)
```

Out[24]:

```
3.0    0.350533
```

```
2.0    0.318798
4.0    0.176357
5.0    0.114583
1.0    0.039729
Name: income_cat, dtype: float64
```

In [25]:

```
# Income category proportion in full dataset

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

Out[25]:

```
3.0    0.350581
2.0    0.318847
4.0    0.176308
5.0    0.114438
1.0    0.039826
Name: income_cat, dtype: float64
```

In [26]:

```
# Let’s compare income category proportion in Stra
tified Sampling and Random Sampling

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

train_set, test_set = train_test_split(housing, te
st_size=0.2, random_state=42)

compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_tes
t_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_prop
s["Random"] / compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_pro
ps["Stratified"] / compare_props["Overall"] - 100
```

In [27]:

```
compare_props
```

Out[27]:

| | Overall | Stratified | Random | Rand. %error | Strat. %error |
|---|---|---|---|---|---|
| 1.0 | 0.039826 | 0.039729 | 0.040213 | 0.973236 | -0.243309 |
| 2.0 | 0.318847 | 0.318798 | 0.324370 | 1.732260 | -0.015195 |
| 3.0 | 0.350581 | 0.350533 | 0.358527 | 2.266446 | -0.013820 |

| | | | | | |
|---|---|---|---|---|---|
| 4.0 | 0.176308 | 0.176357 | 0.167393 | -5.056334 | 0.027480 |
| 5.0 | 0.114438 | 0.114583 | 0.109496 | -4.318374 | 0.127011 |

In [28]:

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

# Discover and visualize the data to gain insights

In [29]:

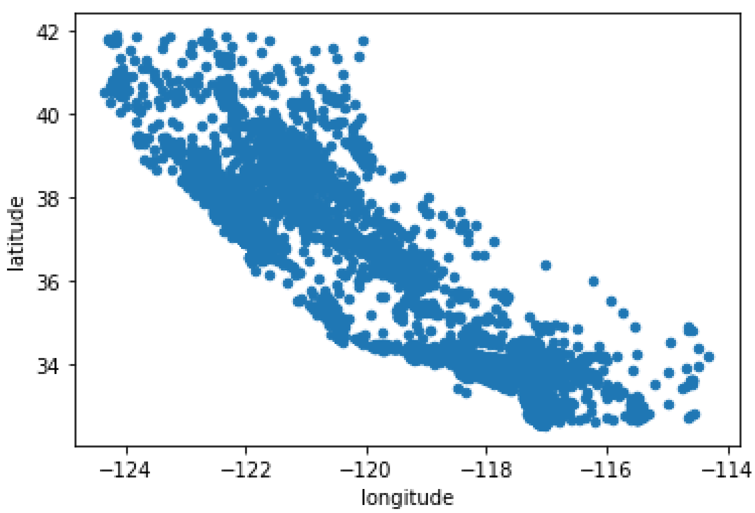
```
housing = strat_train_set.copy()
```

In [30]:

```
housing.plot(kind="scatter", x="longitude", y="lat
itude")
```

Out[30]:

<matplotlib.axes._subplots.AxesSubplot at 0x1d2c49
bcbc8>

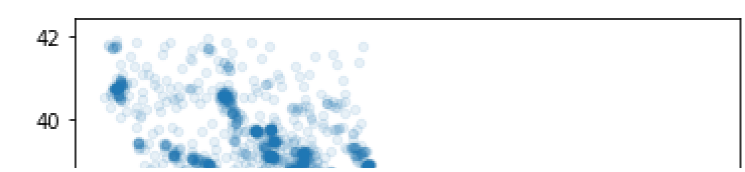


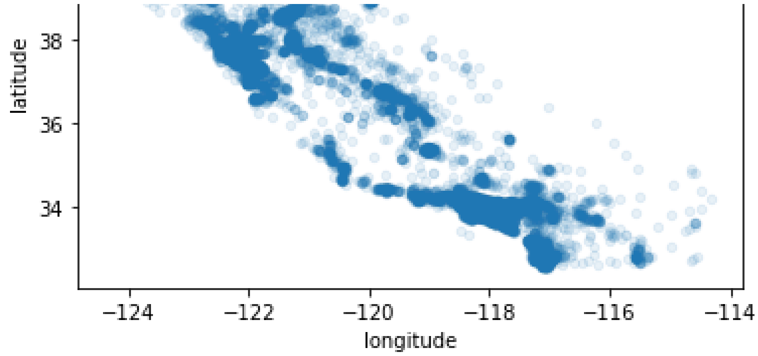

In [31]:

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

Out[31]:

<matplotlib.axes._subplots.AxesSubplot at 0x1d2c48
9a048>





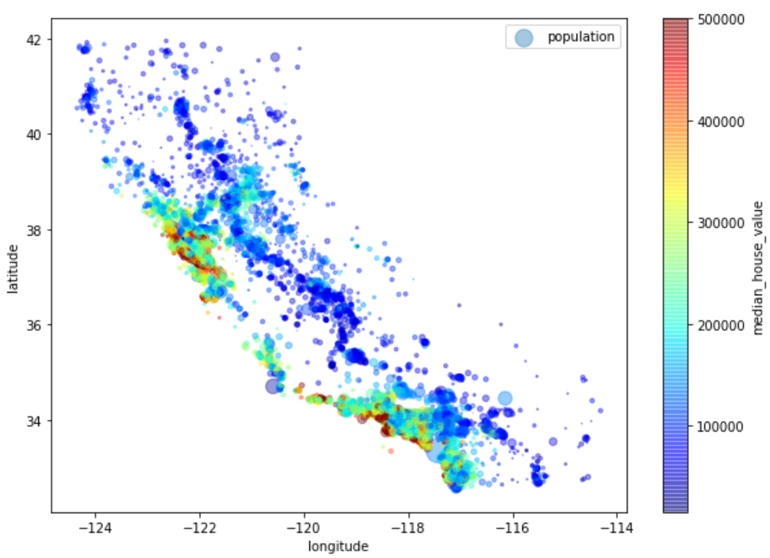

The argument `sharex=False` fixes a display bug (the x-axis values and legend were not displayed). This is a temporary fix (see: https://github.com/pandas-dev/pandas/issues/10611 (https://github.com/pandas-dev/pandas/issues/10611)). Thanks to Wilmer Arellano for pointing it out.

In [32]:

```
housing.plot(kind="scatter", x="longitude", y="lat
itude", alpha=0.4,
    s=housing["population"]/100, label="populatio
n", figsize=(10,7),
    c="median_house_value", cmap=plt.get_cmap("je
t"), colorbar=True,
    sharex=False)
plt.legend()
```

Out[32]:

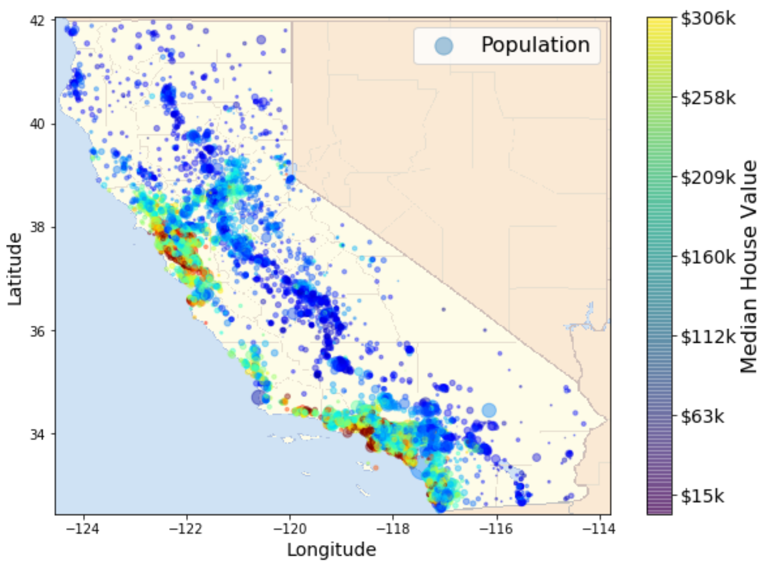
```
<matplotlib.legend.Legend at 0x1d2c4743448>
```




In [33]:

```
import matplotlib.image as mpimg
california_img=mpimg.imread('images/end_to_end_pro
ject/california.png')
ax = housing.plot(kind="scatter", x="longitude", y
="latitude", figsize=(10,7),
                       s=housing['population']/100
  label="Population"
```

```
, label="Population",
                           c="median_house_value", cma
p=plt.get_cmap("jet"),
                           colorbar=False, alpha=0.4,
                          )
plt.imshow(california_img, extent=[-124.55, -113.8
0, 32.45, 42.05], alpha=0.5)
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()
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) fo
r v in tick_values], fontsize=14)
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
plt.show()
```




In [34]:

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

Out[34]:

| | longitude | latitude | housing_media |
|---|---|---|---|
| longitude | 1.000000 | -0.924478 | -0.105848 |
| latitude | -0.924478 | 1.000000 | 0.005766 |
| housing_median_age | -0.105848 | 0.005766 | 1.000000 |
| total_rooms | 0.048871 | -0.039184 | -0.364509 |
| total_bedrooms | 0.076598 | -0.072419 | -0.325047 |
| population | 0.108030 | -0.115222 | -0.298710 |
| households | 0.063070 | -0.077647 | -0.306428 |

| | | | |
|---|---|---|---|
| median_income | -0.019583 | -0.075205 | -0.111360 |
| median_house_value | -0.047432 | -0.142724 | 0.114110 |

In [35]:

```
corr_matrix["median_house_value"]
```

Out[35]:

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

In [36]:

```
corr_matrix["median_house_value"].sort_values(asce
nding=False)
```

Out[36]:

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

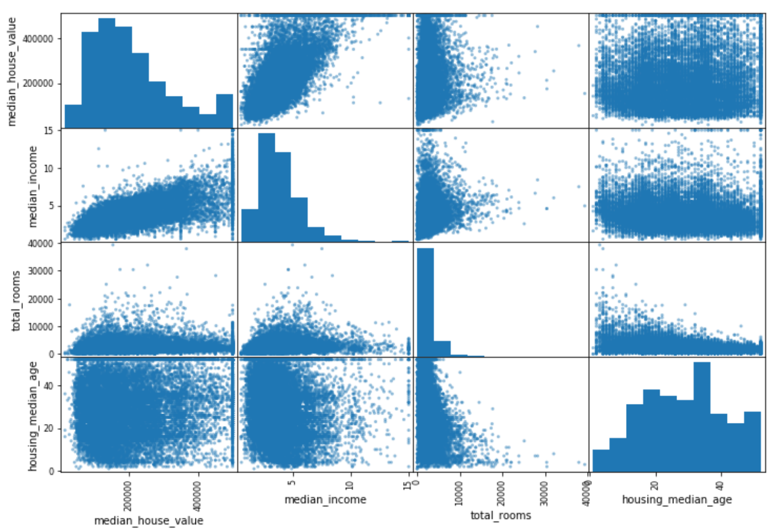
In [37]:

```
# from pandas.tools.plotting import scatter_matrix
# For older versions of Pandas

from pandas.plotting import scatter_matrix

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

Out[37]:

```
array([[<matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C4793EC8>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C59C0448>,
        <matplotlib.axes._subplots.AxesSubplot obj
```

```
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C58473C8>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C587F808>],
       [<matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C58B8948>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C58F2A48>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C592BB08>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C59D2C48>],
       [<matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C59DD808>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C74479C8>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C74ACF88>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C8D29048>],
       [<matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C8D63148>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C8D9B248>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C8DD5388>,
        <matplotlib.axes._subplots.AxesSubplot obj
ect at 0x000001D2C941F488>]],
      dtype=object)
```

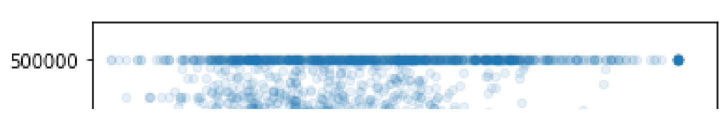


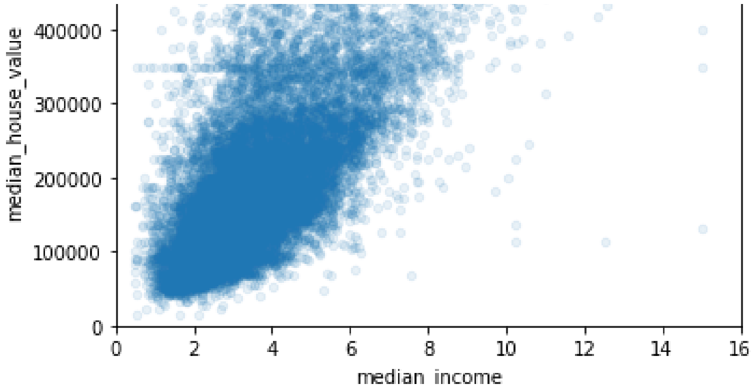

In [38]:

```
housing.plot(kind="scatter", x="median_income", y=
"median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
```

Out[38]:

```
[0, 16, 0, 550000]
```






In [39]:

```
# Experimenting with Attribute Combinations

housing["rooms_per_household"] = housing["total_ro
oms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedr
ooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["popul
ation"]/housing["households"]

housing.head(20)
```

Out[39]:

| | longitude | latitude | housing_median_age | total_roo |
|---|---|---|---|---|
| 17606 | -121.89 | 37.29 | 38.0 | 1568.0 |
| 18632 | -121.93 | 37.05 | 14.0 | 679.0 |
| 14650 | -117.20 | 32.77 | 31.0 | 1952.0 |
| 3230 | -119.61 | 36.31 | 25.0 | 1847.0 |
| 3555 | -118.59 | 34.23 | 17.0 | 6592.0 |
| 19480 | -120.97 | 37.66 | 24.0 | 2930.0 |
| 8879 | -118.50 | 34.04 | 52.0 | 2233.0 |
| 13685 | -117.24 | 34.15 | 26.0 | 2041.0 |
| 4937 | -118.26 | 33.99 | 47.0 | 1865.0 |
| 4861 | -118.28 | 34.02 | 29.0 | 515.0 |
| 16365 | -121.31 | 38.02 | 24.0 | 4157.0 |
| 19684 | -121.62 | 39.14 | 41.0 | 2183.0 |
| 19234 | -122.69 | 38.51 | 18.0 | 3364.0 |
| 13956 | -117.06 | 34.17 | 21.0 | 2520.0 |
| 2390 | -119.46 | 36.91 | 12.0 | 2980.0 |
| 11176 | -117.96 | 33.83 | 30.0 | 2838.0 |
| 15614 | -122.41 | 37.81 | 25.0 | 1178.0 |

| | | | | |
|---|---|---|---|---|
| 2953 | -119.02 | 35.35 | 42.0 | 1239.0 |
| 13209 | -117.72 | 34.05 | 8.0 | 1841.0 |
| 6569 | -118.15 | 34.20 | 46.0 | 1505.0 |

In [40]:

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

Out[40]:

| | **longitude** | **latitude** | **housing_r** |
|---|---|---|---|
| longitude | 1.000000 | -0.924478 | -0.105848 |
| latitude | -0.924478 | 1.000000 | 0.005766 |
| housing_median_age | -0.105848 | 0.005766 | 1.000000 |
| total_rooms | 0.048871 | -0.039184 | -0.364509 |
| total_bedrooms | 0.076598 | -0.072419 | -0.325047 |
| population | 0.108030 | -0.115222 | -0.298710 |
| households | 0.063070 | -0.077647 | -0.306428 |
| median_income | -0.019583 | -0.075205 | -0.111360 |
| median_house_value | -0.047432 | -0.142724 | 0.114110 |
| rooms_per_household | -0.028345 | 0.107621 | -0.147186 |
| bedrooms_per_room | 0.095603 | -0.116884 | 0.136788 |
| population_per_household | -0.000410 | 0.005420 | 0.015031 |

In [41]:

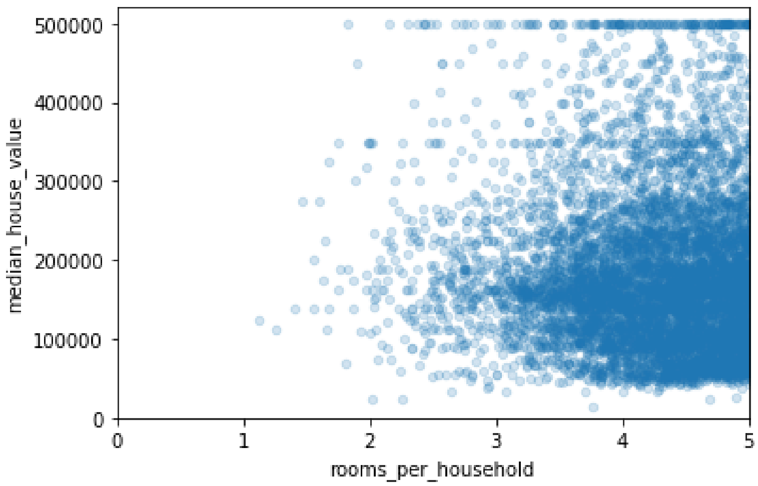
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(asce
nding=False)
```

Out[41]:

```
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
```

In [42]:

```
housing.plot(kind="scatter", x="rooms_per_househol
d", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```




In [43]:

```
housing.describe()
```

Out[43]:

| | longitude | latitude | housing_median_age |
|---|---|---|---|
| count | 16512.000000 | 16512.000000 | 16512.000000 |
| mean | -119.575834 | 35.639577 | 28.653101 |
| std | 2.001860 | 2.138058 | 12.574726 |
| min | -124.350000 | 32.540000 | 1.000000 |
| 25% | -121.800000 | 33.940000 | 18.000000 |
| 50% | -118.510000 | 34.260000 | 29.000000 |
| 75% | -118.010000 | 37.720000 | 37.000000 |
| max | -114.310000 | 41.950000 | 52.000000 |

# Prepare the data for Machine Learning algorithms

In [44]:

```
# Let's revert to a clean training set

housing = strat_train_set.drop("median_house_valu
e", axis=1) # drop labels for training set
housing_labels = strat_train_set["median_house_val
```

```
housing_labels = strat_train_set["median_house_val
ue"].copy()

# Note drop() creates a copy of the data and does
 not affect strat_train_set
```

In [45]:

```
isn = housing.isnull()
isn.any(axis=1)
```

Out[45]:

```
17606    False
18632    False
14650    False
3230     False
3555     False
         ...
6563     False
12053    False
13908    False
11159    False
15775    False
Length: 16512, dtype: bool
```

In [46]:

```
# Let’s experiment with sample dataset for data cl
eaning

sample_incomplete_rows = housing[housing.isnull().
any(axis=1)].head(100)
sample_incomplete_rows
```

Out[46]:

| | longitude | latitude | housing_median_age | total_roc |
|---|---|---|---|---|
| 4629 | -118.30 | 34.07 | 18.0 | 3759.0 |
| 6068 | -117.86 | 34.01 | 16.0 | 4632.0 |
| 17923 | -121.97 | 37.35 | 30.0 | 1955.0 |
| 13656 | -117.30 | 34.05 | 6.0 | 2155.0 |
| 19252 | -122.79 | 38.48 | 7.0 | 6837.0 |
| ... | ... | ... | ... | ... |
| 14986 | -117.03 | 32.73 | 34.0 | 2061.0 |
| 4186 | -118.23 | 34.13 | 48.0 | 1308.0 |
| 16105 | -122.50 | 37.75 | 44.0 | 1819.0 |
| 7668 | -118.08 | 33.92 | 38.0 | 1335.0 |
| 14386 | -117.23 | 32.75 | 5.0 | 1824.0 |

100 rows × 9 columns

In [47]:

```
# Option one
# dropna() - drops the missing values

sample_incomplete_rows.dropna(subset=["total_bedro
oms"])
```

Out[47]:

| | longitude | latitude | housing_median_age | total_rooms | t |
|---|---|---|---|---|---|

In [ ]:

```

```

In [48]:

```
# Option two
# drop() - drops the attribute

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

Out[48]:

| | longitude | latitude | housing_median_age | total_roo |
|---|---|---|---|---|
| 4629 | -118.30 | 34.07 | 18.0 | 3759.0 |
| 6068 | -117.86 | 34.01 | 16.0 | 4632.0 |
| 17923 | -121.97 | 37.35 | 30.0 | 1955.0 |
| 13656 | -117.30 | 34.05 | 6.0 | 2155.0 |
| 19252 | -122.79 | 38.48 | 7.0 | 6837.0 |
| ... | ... | ... | ... | ... |
| 14986 | -117.03 | 32.73 | 34.0 | 2061.0 |
| 4186 | -118.23 | 34.13 | 48.0 | 1308.0 |
| 16105 | -122.50 | 37.75 | 44.0 | 1819.0 |
| 7668 | -118.08 | 33.92 | 38.0 | 1335.0 |
| 14386 | -117.23 | 32.75 | 5.0 | 1824.0 |

100 rows × 8 columns

In [49]:

```
# Option three
# fillna() - sets the missing values
# Let’s fill the missing values with the median
```

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

Out[49]:

| | longitude | latitude | housing_median_age | total_roc |
|---|---|---|---|---|
| 4629 | -118.30 | 34.07 | 18.0 | 3759.0 |
| 6068 | -117.86 | 34.01 | 16.0 | 4632.0 |
| 17923 | -121.97 | 37.35 | 30.0 | 1955.0 |
| 13656 | -117.30 | 34.05 | 6.0 | 2155.0 |
| 19252 | -122.79 | 38.48 | 7.0 | 6837.0 |
| ... | ... | ... | ... | ... |
| 14986 | -117.03 | 32.73 | 34.0 | 2061.0 |
| 4186 | -118.23 | 34.13 | 48.0 | 1308.0 |
| 16105 | -122.50 | 37.75 | 44.0 | 1819.0 |
| 7668 | -118.08 | 33.92 | 38.0 | 1335.0 |
| 14386 | -117.23 | 32.75 | 5.0 | 1824.0 |

100 rows × 9 columns

In [52]:

```
# Let's use Scikit-Learn Imputer class to fill mis
sing values

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
```

In [53]:

```
# Remove the text attribute because median can onl
y be calculated on numerical attributes

housing_num = housing.drop('ocean_proximity', axis
=1)
```

In [54]:

```
# Fit the imputer instance to the training data

imputer.fit(housing_num)
```

Out[54]:

```
SimpleImputer(add_indicator=False, copy=True, fill
_value=None,
```

```
                 missing_values=nan, strategy='mean',
verbose=0)
```

In [55]:

```
imputer.statistics_
```

Out[55]:

```
array([-119.57583394,   35.63957728,   28.6531007
8, 2622.7283188 ,
        534.97389018, 1419.7908188 ,  497.0603803
3,    3.87558937])
```

Transform the training set:

In [56]:

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

Out[56]:

```
array([[-121.89  ,   37.29  ,   38.    , ...,  71
0.    ,  339.    ,
          2.7042],
       [-121.93  ,   37.05  ,   14.    , ...,  30
6.    ,  113.    ,
          6.4214],
       [-117.2   ,   32.77  ,   31.    , ...,  93
6.    ,  462.    ,
          2.8621],
       ...,
       [-116.4   ,   34.09  ,    9.    , ..., 209
8.    ,  765.    ,
          3.2723],
       [-118.01  ,   33.82  ,   31.    , ..., 135
6.    ,  356.    ,
          4.0625],
       [-122.45  ,   37.77  ,   52.    , ..., 126
9.    ,  639.    ,
          3.575 ]])
```

In [57]:

```
housing_tr = pd.DataFrame(X, columns=housing_num.c
olumns)
housing_tr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16512 entries, 0 to 16511
Data columns (total 8 columns):
longitude             16512 non-null float64
latitude              16512 non-null float64
housing_median_age    16512 non-null float64
total_rooms           16512 non-null float64
total_bedrooms        16512 non-null float64
population            16512 non-null float64
households            16512 non-null float64
median_income         16512 non-null float64
```

```
median_income         16512 non-null float64
dtypes: float64(8)
memory usage: 1.0 MB
```

Now let's preprocess the categorical input feature, `ocean_proximity`:

In [58]:

```
# Convert ocean_proximity to numbers

housing_cat = housing['ocean_proximity']
housing_cat.head(10)
```

Out[58]:

```
17606     <1H OCEAN
18632     <1H OCEAN
14650    NEAR OCEAN
3230         INLAND
3555      <1H OCEAN
19480        INLAND
8879      <1H OCEAN
13685        INLAND
4937      <1H OCEAN
4861      <1H OCEAN
Name: ocean_proximity, dtype: object
```

In [59]:

```
# Pandas factorize() example

df = pd.DataFrame({
        'A':['type1','type3','type3', 'type2', 'ty
pe0']
    })
df['A'].factorize()
```

Out[59]:

```
(array([0, 1, 1, 2, 3], dtype=int64),
 Index(['type1', 'type3', 'type2', 'type0'], dtype
='object'))
```

In [60]:

```
# Convert ocean_proximity to numbers
# Use Pandas factorize()


housing_cat_encoded, housing_categories = housing_
cat.factorize()
housing_cat_encoded[:10]
```

Out[60]:

```
array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0], dtype=int64)
```

In [61]:

```
# Check encoding classes
```

```
# check encoding classes

housing_categories
```

Out[61]:

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

In [62]:

```
# We can convert each categorical value to a one-h
ot vector using a `OneHotEncoder`
# Note that fit_transform() expects a 2D array
# but housing_cat_encoded is a 1D array, so we nee
d to reshape it

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_c
at_encoded.reshape(-1,1))
housing_cat_1hot
```

Out[62]:

```
<16512x5 sparse matrix of type '<class 'numpy.floa
t64'>'
        with 16512 stored elements in Compressed S
parse Row format>
```

In [63]:

```
# The OneHotEncoder returns a sparse array by defa
ult, but we can convert it to a dense array if nee
ded

housing_cat_1hot.toarray()
```

Out[63]:

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

In [64]:

```
# Just run this cell, or copy it to your code, do
 not try to understand it (yet).
# Definition of the CategoricalEncoder class, copi
ed from PR #9151.

from sklearn.base import BaseEstimator, Transforme
rMixin
from sklearn.utils import check_array
from sklearn.preprocessing import LabelEncoder
from scipy import sparse
```

```
from scipy import sparse

class CategoricalEncoder(BaseEstimator, Transforme
rMixin):
    """Encode categorical features as a numeric ar
ray.
    The input to this transformer should be a matr
ix of integers or strings,
    denoting the values taken on by categorical (d
iscrete) features.
    The features can be encoded using a one-hot ak
a one-of-K scheme
    (``encoding='onehot'``, the default) or conver
ted to ordinal integers
    (``encoding='ordinal'``).
    This encoding is needed for feeding categorica
l data to many scikit-learn
    estimators, notably linear models and SVMs wit
h the standard kernels.
    Read more in the :ref:`User Guide <preprocessi
ng_categorical_features>`.
    Parameters
    ----------
    encoding : str, 'onehot', 'onehot-dense' or 'o
rdinal'
        The type of encoding to use (default is 'o
nehot'):
        - 'onehot': encode the features using a on
e-hot aka one-of-K scheme
          (or also called 'dummy' encoding). This
 creates a binary column for
          each category and returns a sparse matri
x.
        - 'onehot-dense': the same as 'onehot' but
returns a dense array
          instead of a sparse matrix.
        - 'ordinal': encode the features as ordina
l integers. This results in
          a single column of integers (0 to n_cate
gories - 1) per feature.
    categories : 'auto' or a list of lists/arrays
 of values.
        Categories (unique values) per feature:
        - 'auto' : Determine categories automatica
lly from the training data.
        - list : ``categories[i]`` holds the categ
ories expected in the ith
          column. The passed categories are sorted
before encoding the data
          (used categories can be found in the ``c
ategories_`` attribute).
    dtype : number type, default np.float64
        Desired dtype of output.
    handle_unknown : 'error' (default) or 'ignore'
        Whether to raise an error or ignore if a u
nknown categorical feature is
        present during transform (default is to ra
ise). When this is parameter
```

```
        is set to 'ignore' and an unknown category
is encountered during
        transform, the resulting one-hot encoded c
olumns for this feature
        will be all zeros.
        Ignoring unknown categories is not support
ed for
        ``encoding='ordinal'``.
    Attributes
    ----------
    categories_ : list of arrays
        The categories of each feature determined
 during fitting. When
        categories were specified manually, this h
olds the sorted categories
        (in order corresponding with output of `tr
ansform`).
    Examples
    --------
    Given a dataset with three features and two sa
mples, we let the encoder
    find the maximum value per feature and transfo
rm the data to a binary
    one-hot encoding.
    >>> from sklearn.preprocessing import Categori
calEncoder
    >>> enc = CategoricalEncoder(handle_unknown='i
gnore')
    >>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1],
 [1, 0, 2]])
    ... # doctest: +ELLIPSIS
    CategoricalEncoder(categories='auto', dtype=
<... 'numpy.float64'>,
              encoding='onehot', handle_unknown='i
gnore')
    >>> enc.transform([[0, 1, 1], [1, 0, 4]]).toar
ray()
    array([[ 1.,  0.,  0.,  1.,  0.,  0.,  1.,
 0.,  0.],
           [ 0.,  1.,  1.,  0.,  0.,  0.,  0.,
 0.,  0.]])
    See also
    --------
    sklearn.preprocessing.OneHotEncoder : performs
a one-hot encoding of
      integer ordinal features. The ``OneHotEncode
r assumes`` that input
      features take on values in the range ``[0, m
ax(feature)]`` instead of
      using the unique values.
    sklearn.feature_extraction.DictVectorizer : pe
rforms a one-hot encoding of
      dictionary items (also handles string-valued
features).
    sklearn.feature_extraction.FeatureHasher : per
forms an approximate one-hot
      encoding of dictionary items or strings.
    """
```

```
    def __init__(self, encoding='onehot', categori
es='auto', dtype=np.float64,
                 handle_unknown='error'):
        self.encoding = encoding
        self.categories = categories
        self.dtype = dtype
        self.handle_unknown = handle_unknown

    def fit(self, X, y=None):
        """Fit the CategoricalEncoder to X.
        Parameters
        ----------
        X : array-like, shape [n_samples, n_featur
e]
            The data to determine the categories o
f each feature.
        Returns
        -------
        self
        """

        if self.encoding not in ['onehot', 'onehot
-dense', 'ordinal']:
            template = ("encoding should be either
'onehot', 'onehot-dense' "
                        "or 'ordinal', got %s")
            raise ValueError(template % self.handl
e_unknown)

        if self.handle_unknown not in ['error', 'i
gnore']:
            template = ("handle_unknown should be
 either 'error' or "
                        "'ignore', got %s")
            raise ValueError(template % self.handl
e_unknown)

        if self.encoding == 'ordinal' and self.han
dle_unknown == 'ignore':
            raise ValueError("handle_unknown='igno
re' is not supported for"
                             " encoding='ordinal'"
)

        X = check_array(X, dtype=np.object, accept
_sparse='csc', copy=True)
        n_samples, n_features = X.shape

        self._label_encoders_ = [LabelEncoder() fo
r _ in range(n_features)]

        for i in range(n_features):
            le = self._label_encoders_[i]
            Xi = X[:, i]
            if self.categories == 'auto':
                le.fit(Xi)
            else:
```

```
            else:
                valid_mask = np.in1d(Xi, self.cate
gories[i])
                if not np.all(valid_mask):
                    if self.handle_unknown == 'err
or':
                        diff = np.unique(Xi[~valid
_mask])
                        msg = ("Found unknown cate
gories {0} in column {1}"
                               " during fit".forma
t(diff, i))
                        raise ValueError(msg)
                le.classes_ = np.array(np.sort(sel
f.categories[i]))

        self.categories_ = [le.classes_ for le in 
self._label_encoders_]

        return self

    def transform(self, X):
        """Transform X using one-hot encoding.
        Parameters
        ----------
        X : array-like, shape [n_samples, n_featur
es]
            The data to encode.
        Returns
        -------
        X_out : sparse matrix or a 2-d array
            Transformed input.
        """
        X = check_array(X, accept_sparse='csc', dt
ype=np.object, copy=True)
        n_samples, n_features = X.shape
        X_int = np.zeros_like(X, dtype=np.int)
        X_mask = np.ones_like(X, dtype=np.bool)

        for i in range(n_features):
            valid_mask = np.in1d(X[:, i], self.cat
egories_[i])

            if not np.all(valid_mask):
                if self.handle_unknown == 'error':
                    diff = np.unique(X[~valid_mask
, i])
                    msg = ("Found unknown categori
es {0} in column {1}"
                           " during transform".for
mat(diff, i))
                    raise ValueError(msg)
                else:
                    # Set the problematic rows to
 an acceptable value and
                    # continue `The rows are marke
d `X_mask` and will be
                    # removed later.
```

```
                    X_mask[:, i] = valid_mask
                    X[:, i][~valid_mask] = self.ca
tegories_[i][0]
            X_int[:, i] = self._label_encoders_[i]
.transform(X[:, i])

        if self.encoding == 'ordinal':
            return X_int.astype(self.dtype, copy=F
alse)

        mask = X_mask.ravel()
        n_values = [cats.shape[0] for cats in self
.categories_]
        n_values = np.array([0] + n_values)
        indices = np.cumsum(n_values)

        column_indices = (X_int + indices[:-1]).ra
vel()[mask]
        row_indices = np.repeat(np.arange(n_sample
s, dtype=np.int32),
                                n_features)[mask]
        data = np.ones(n_samples * n_features)[mas
k]

        out = sparse.csc_matrix((data, (row_indice
s, column_indices)),
                                shape=(n_samples,
indices[-1]),
                                dtype=self.dtype).
tocsr()
        if self.encoding == 'onehot-dense':
            return out.toarray()
        else:
            return out
```

In [65]:

```
# The CategoricalEncoder expects a 2D array contai
ning one or more categorical input features.
# We need to reshape `housing_cat` to a 2D array:

cat_encoder = CategoricalEncoder(encoding="onehot-
dense")
housing_cat_reshaped = housing_cat.values.reshape(
-1, 1)
housing_cat_1hot = cat_encoder.fit_transform(housi
ng_cat_reshaped)
housing_cat_1hot
```

Out[65]:

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

In [66]:

```
cat_encoder.categories_
```

Out[66]:

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

Let's create a custom transformer to add extra attributes:

In [67]:

```
from sklearn.base import BaseEstimator, Transforme
rMixin

# column index
rooms_ix, bedrooms_ix, population_ix, household_ix
= 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, Trans
formerMixin):
    def __init__(self, add_bedrooms_per_room = Tru
e): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_
per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    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, p
opulation_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, p
opulation_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_
per_room=False)
housing_extra_attribs = attr_adder.transform(housi
ng.values)
housing_extra_attribs = pd.DataFrame(housing_extra
_attribs, columns=list(housing.columns)+["rooms_pe
r_household", "population_per_household"])
housing_extra_attribs.head()
```

Out[67]:

| | longitude | latitude | housing_median_age | total_rooms |
|---|---|---|---|---|
| 0 | -121.89 | 37.29 | 38 | 1568 |

| | | | | |
|---|---|---|---|---|
| 1 | -121.93 | 37.05 | 14 | 679 |
| 2 | -117.2 | 32.77 | 31 | 1952 |
| 3 | -119.61 | 36.31 | 25 | 1847 |
| 4 | -118.59 | 34.23 | 17 | 6592 |

# Go to slide Custom Transformers - Summary

In [68]:

```
# Feature Scaling - Min-max Scaling - Example
# Creating DataFrame first

s1 = pd.Series([1, 2, 3, 4, 5, 6], index=(range(6
)))
s2 = pd.Series([10, 9, 8, 7, 6, 5], index=(range(6
)))
df = pd.DataFrame(s1, columns=['s1'])
df['s2'] = s2
df
```

Out[68]:

| | s1 | s2 |
|---|---|---|
| 0 | 1 | 10 |
| 1 | 2 | 9 |
| 2 | 3 | 8 |
| 3 | 4 | 7 |
| 4 | 5 | 6 |
| 5 | 6 | 5 |

In [71]:

```
# Use Scikit-Learn minmax_scaling

from mlxtend.preprocessing import minmax_scaling
minmax_scaling(df, columns=['s1', 's2'])
```

Out[71]:

| | s1 | s2 |
|---|---|---|
| 0 | 0.0 | 1.0 |
| 1 | 0.2 | 0.8 |
| 2 | 0.4 | 0.6 |
| 3 | 0.6 | 0.4 |

| | | |
|---|---|---|
| 4 | 0.8 | 0.2 |
| 5 | 1.0 | 0.0 |

In [74]:

```
# Now let's build a pipeline for preprocessing the
numerical attributes:

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="media
n")),
        ('attribs_adder', CombinedAttributesAdder
()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housin
g_num)
```

In [75]:

```
housing_num_tr
```

Out[75]:

```
array([[-1.15604281,  0.77194962,  0.74333089,
..., -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 ,
...,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186,
..., -0.46531516,
        -0.09240499,  0.4222004 ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222,
...,  0.3469342 ,
        -0.03055414, -0.52177644],
       [ 0.78221312, -0.85106801,  0.18664186,
...,  0.02499488,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,  1.85670895,
..., -0.22852947,
        -0.09586294,  0.10180567]])
```