End_to_end_Machine_Learning_project

July 5, 2021

Chapter 2 – End-to-end Machine Learning project

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.

This notebook contains all the sample code and solutions to the exercices in chapter 2.

1 Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn 0.20.

[ ]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)
```

```
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

2 Get the data

[ ]:
```
import os
import tarfile
import urllib.request

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

[ ]:
```
fetch_housing_data()
```

[ ]:
```
import pandas as pd

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

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

[ ]:

	longitude	latitude	…	median_house_value	ocean_proximity
0	-122.23	37.88	…	452600.0	NEAR BAY
1	-122.22	37.86	…	358500.0	NEAR BAY
2	-122.24	37.85	…	352100.0	NEAR BAY
3	-122.25	37.85	…	341300.0	NEAR BAY
4	-122.25	37.85	…	342200.0	NEAR BAY

[5 rows x 10 columns]

[ ]:
```
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()
```

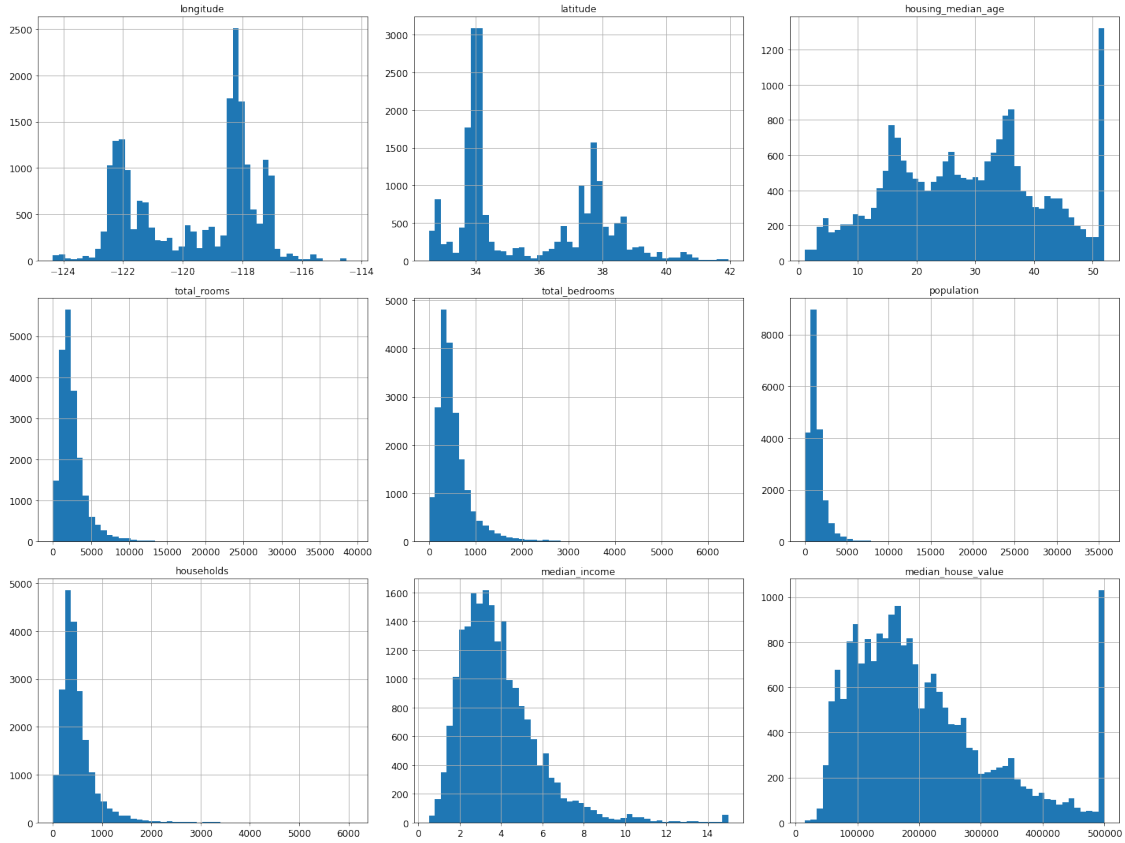
[ ]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

[ ]:
```
housing.describe()
```

[ ]:
```
          longitude      latitude  …  median_income  median_house_value
count  20640.000000  20640.000000  …   20640.000000        20640.000000
mean    -119.569704     35.631861  …       3.870671       206855.816909
std        2.003532      2.135952  …       1.899822       115395.615874
min     -124.350000     32.540000  …       0.499900        14999.000000
25%     -121.800000     33.930000  …       2.563400       119600.000000
50%     -118.490000     34.260000  …       3.534800       179700.000000
75%     -118.010000     37.710000  …       4.743250       264725.000000
max     -114.310000     41.950000  …      15.000100       500001.000000

[8 rows x 9 columns]
```

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```

```
Saving figure attribute_histogram_plots
```



[ ]:
```
# to make this notebook's output identical at every run
np.random.seed(42)
```

[ ]:
```
import numpy as np

# For illustration only. Sklearn has train_test_split()
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

[ ]:
```
train_set, test_set = split_train_test(housing, 0.2)
len(train_set)
```

[ ]:
```
16512
```

[ ]:
```
len(test_set)
```

[ ]:
```
4128
```

[ ]:
```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

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

The implementation of `test_set_check()` above works fine in both Python 2 and Python 3. In earlier releases, the following implementation was proposed, which supported any hash function, but was much slower and did not support Python 2:

[ ]:
```
import hashlib

def test_set_check(identifier, test_ratio, hash=hashlib.md5):
    return hash(np.int64(identifier)).digest()[-1] < 256 * test_ratio
```

If you want an implementation that supports any hash function and is compatible with both Python 2 and Python 3, here is one:

[ ]:
```
def test_set_check(identifier, test_ratio, hash=hashlib.md5):
    return bytearray(hash(np.int64(identifier)).digest())[-1] < 256 * test_ratio
```

[ ]:
```
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")
```

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

[ ]:
```
test_set.head()
```

[ ]:
```
    index  longitude  latitude  …  median_house_value  ocean_proximity
id
8       8    -122.26     37.84  …            226700.0         NEAR BAY
-122222.16
10     10    -122.26     37.85  …            281500.0         NEAR BAY
-122222.15
11     11    -122.26     37.85  …            241800.0         NEAR BAY
-122222.15
12     12    -122.26     37.85  …            213500.0         NEAR BAY
-122222.15
13     13    -122.26     37.84  …            191300.0         NEAR BAY
-122222.16
```

```
[5 rows x 12 columns]
```

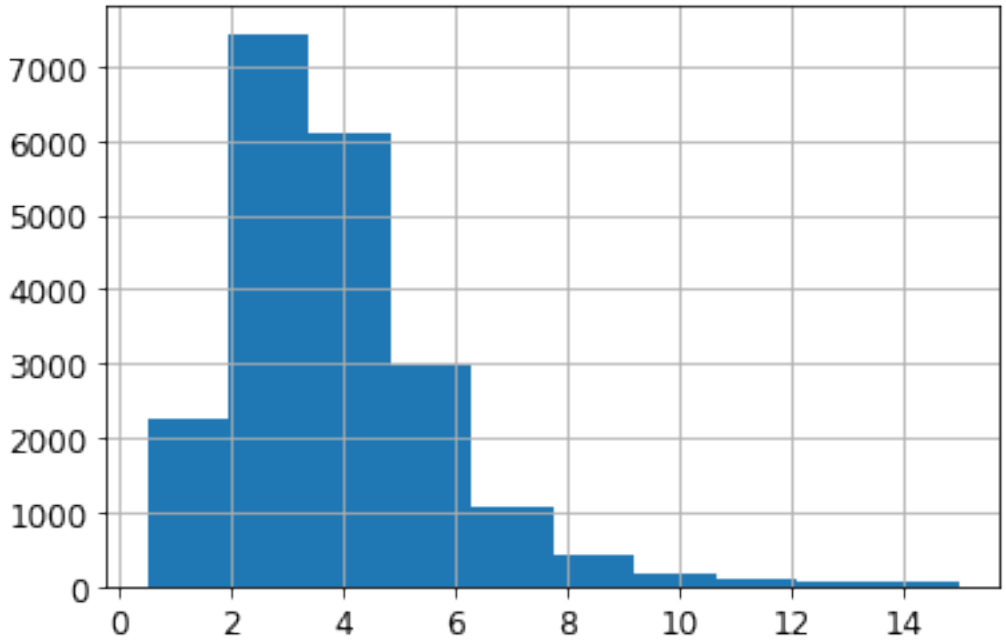
[ ]:
```
from sklearn.model_selection import train_test_split

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

[ ]:
```
test_set.head()
```

[ ]:
```
       longitude  latitude  …  median_house_value  ocean_proximity
20046    -119.01     36.06  …             47700.0           INLAND
3024     -119.46     35.14  …             45800.0           INLAND
15663    -122.44     37.80  …            500001.0         NEAR BAY
20484    -118.72     34.28  …            218600.0        <1H OCEAN
9814     -121.93     36.62  …            278000.0       NEAR OCEAN

[5 rows x 10 columns]
```

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

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f16cda990>
```



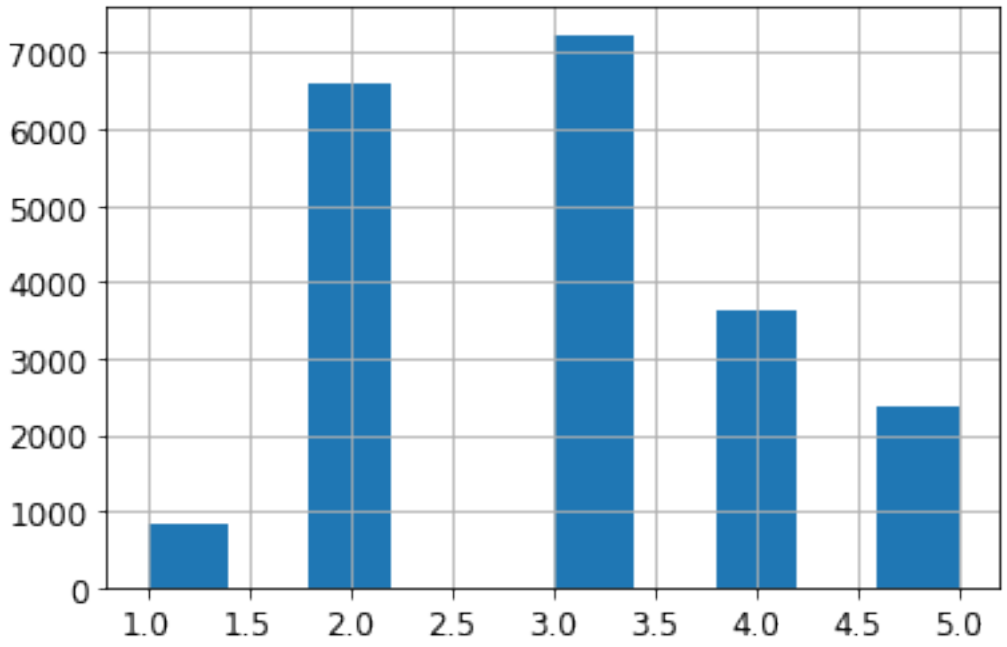
[ ]:
```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

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

[ ]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

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

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f15ea0b50>
```



[ ]:
```
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]
```

[ ]:
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

[ ]:
```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

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

[ ]:
```
3    0.350581
2    0.318847
4    0.176308
5    0.114438
1    0.039826
Name: income_cat, dtype: float64
```

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

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

compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_test_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_props["Random"] /␣
 ↪compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] /␣
 ↪compare_props["Overall"] - 100
```

[ ]:
```
compare_props
```

[ ]:

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

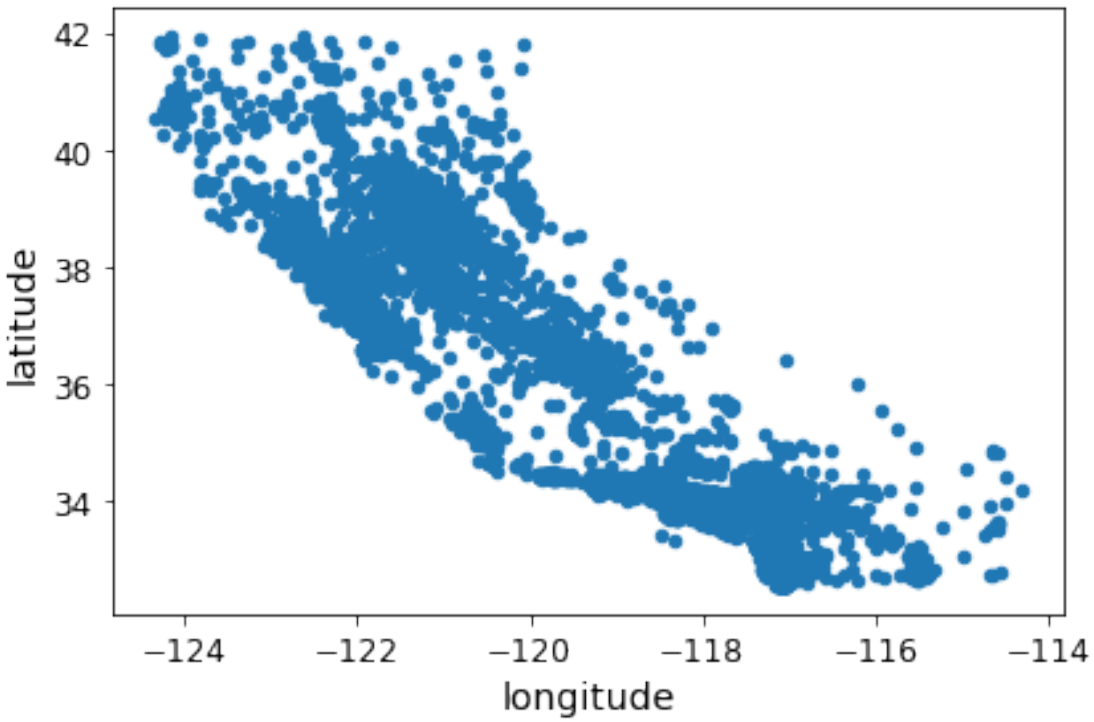
[ ]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

3 Discover and visualize the data to gain insights

[ ]:
```
housing = strat_train_set.copy()
```

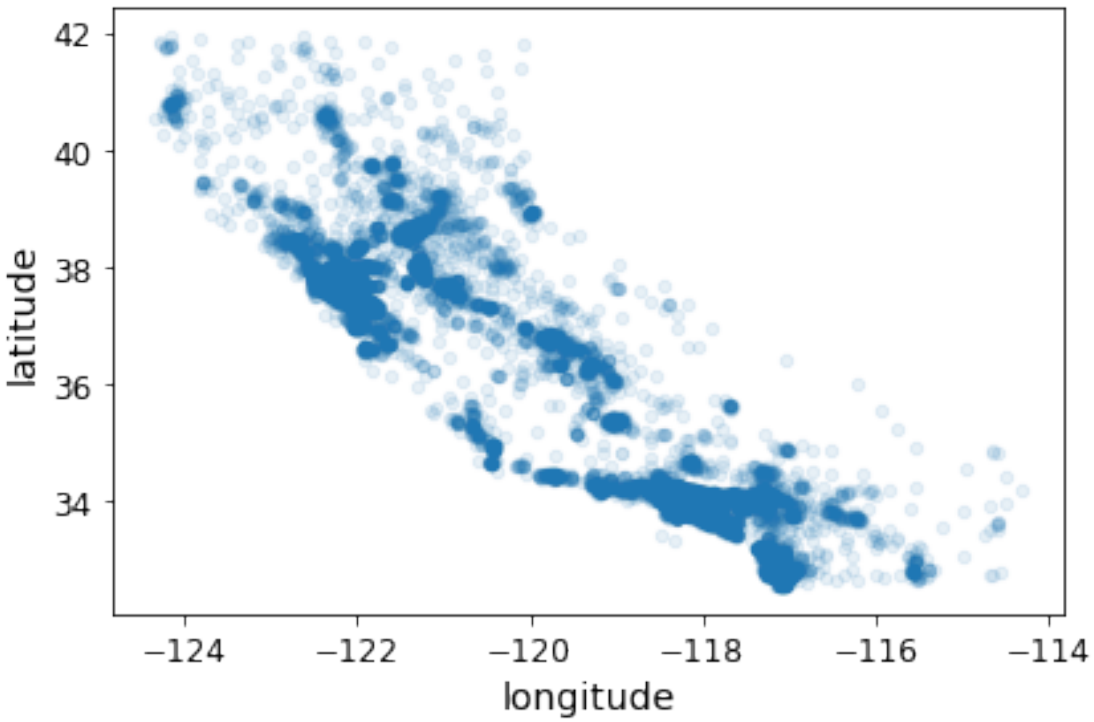
[ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```



[ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
save_fig("better_visualization_plot")
```

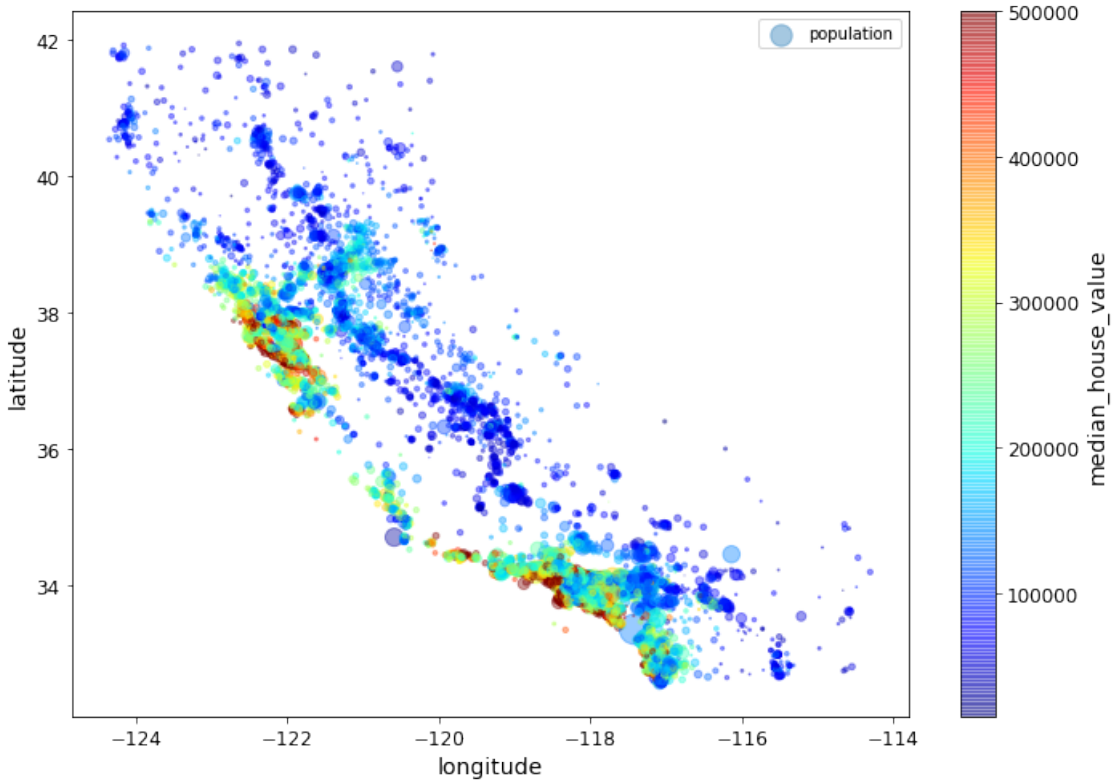
```
Saving figure better_visualization_plot
```



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). Thanks to Wilmer Arellano for pointing it out.

[ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
             sharex=False)
plt.legend()
save_fig("housing_prices_scatterplot")
```

```
Saving figure housing_prices_scatterplot
```



[ ]:
```
# Download the California image
images_path = os.path.join(PROJECT_ROOT_DIR, "images", "end_to_end_project")
os.makedirs(images_path, exist_ok=True)
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
filename = "california.png"
print("Downloading", filename)
url = DOWNLOAD_ROOT + "images/end_to_end_project/" + filename
urllib.request.urlretrieve(url, os.path.join(images_path, filename))
```

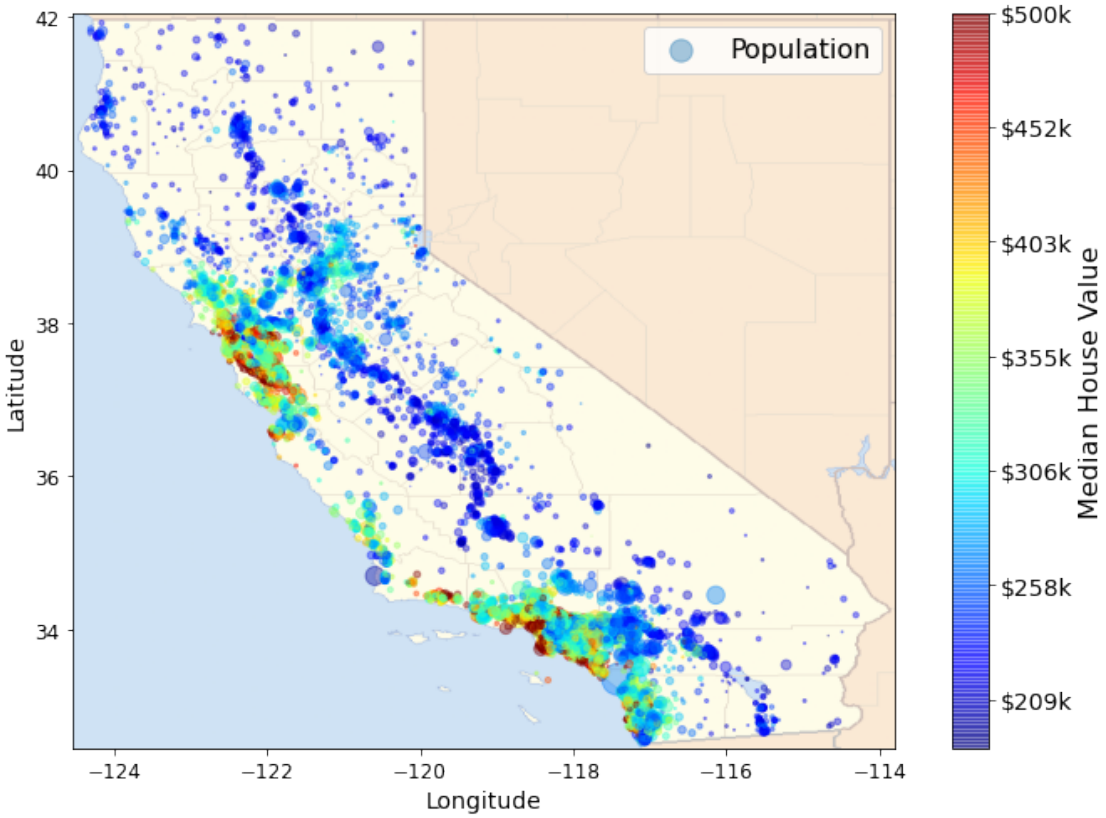
```
Downloading california.png
```

[ ]:
```
('./images/end_to_end_project/california.png',
 <http.client.HTTPMessage at 0x7f3f19db8090>)
```

[ ]:
```
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
```



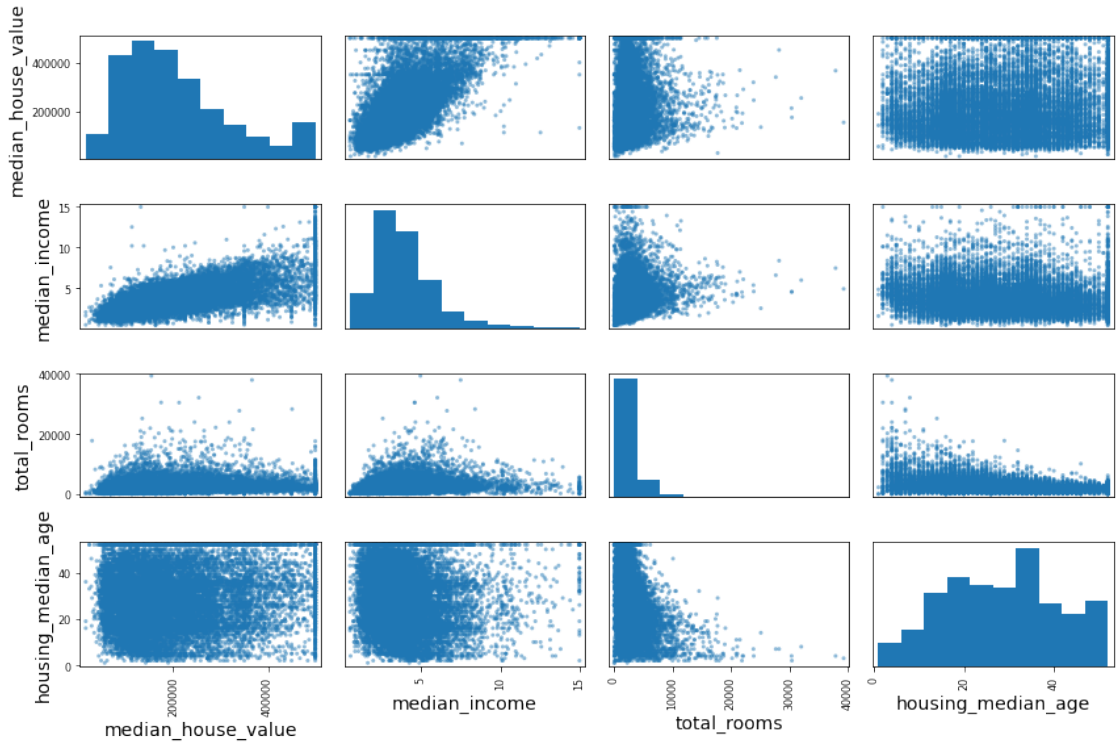
[ ]:
```
corr_matrix = housing.corr()
```

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

[ ]:
```
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
```

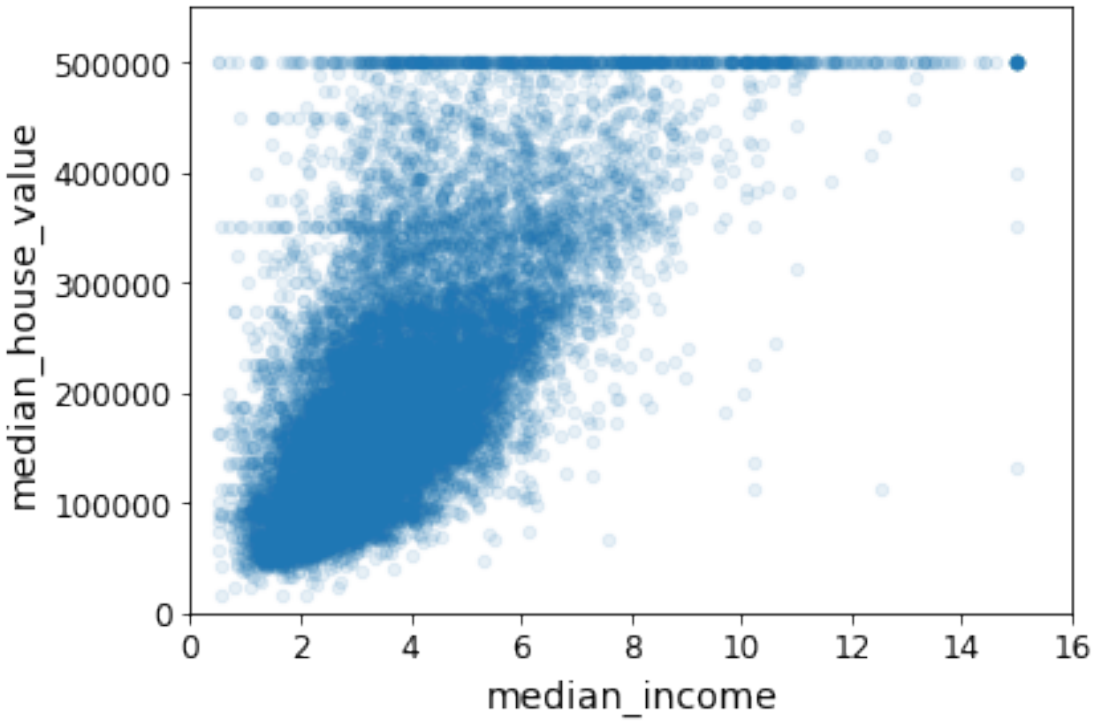
[ ]:
```
# from pandas.tools.plotting import scatter_matrix # For older versions of␣
↪Pandas
from pandas.plotting import scatter_matrix

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

```
Saving figure scatter_matrix_plot
```



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

```
Saving figure income_vs_house_value_scatterplot
```



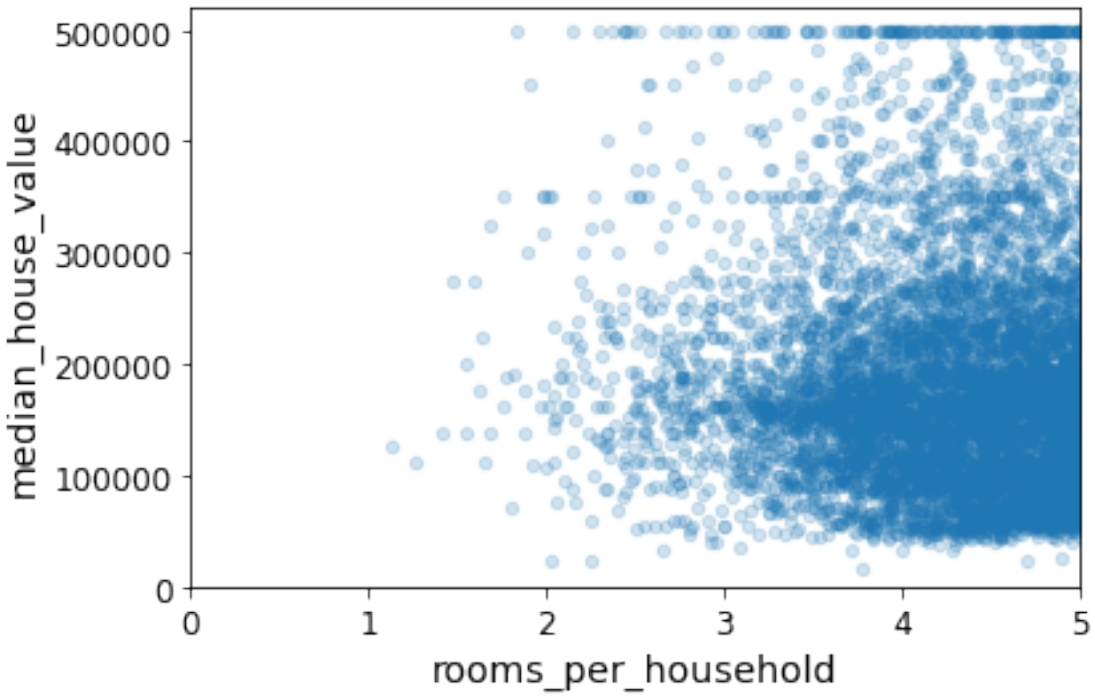
[ ]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

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

[ ]:
```
median_house_value          1.000000
median_income               0.687160
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
```

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



[ ]:
```
housing.describe()
```

[ ]:
```
          longitude      latitude  …  bedrooms_per_room
population_per_household
count  16512.000000  16512.000000  …       16354.000000
16512.000000
mean    -119.575834     35.639577  …           0.212878
3.096437
std        2.001860      2.138058  …           0.057379
11.584826
min     -124.350000     32.540000  …           0.100000
0.692308
25%     -121.800000     33.940000  …           0.175304
2.431287
50%     -118.510000     34.260000  …           0.203031
```

```
2.817653
75%     -118.010000     37.720000  …           0.239831
3.281420
max     -114.310000     41.950000  …           1.000000
1243.333333

[8 rows x 12 columns]
```

4 Prepare the data for Machine Learning algorithms

[ ]:
```
housing = strat_train_set.drop("median_house_value", axis=1) # drop labels for␣
↪training set
housing_labels = strat_train_set["median_house_value"].copy()
```

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

[ ]:
```
       longitude  latitude  …  median_income  ocean_proximity
4629     -118.30     34.07  …         2.2708        <1H OCEAN
6068     -117.86     34.01  …         5.1762        <1H OCEAN
17923    -121.97     37.35  …         4.6328        <1H OCEAN
13656    -117.30     34.05  …         1.6675           INLAND
19252    -122.79     38.48  …         3.1662        <1H OCEAN

[5 rows x 9 columns]
```

[ ]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1
```

[ ]:
```
Empty DataFrame
Columns: [longitude, latitude, housing_median_age, total_rooms, total_bedrooms,
population, households, median_income, ocean_proximity]
Index: []
```

[ ]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2
```

[ ]:
```
       longitude  latitude  …  median_income  ocean_proximity
4629     -118.30     34.07  …         2.2708        <1H OCEAN
6068     -117.86     34.01  …         5.1762        <1H OCEAN
17923    -121.97     37.35  …         4.6328        <1H OCEAN
13656    -117.30     34.05  …         1.6675           INLAND
19252    -122.79     38.48  …         3.1662        <1H OCEAN

[5 rows x 8 columns]
```

[ ]:
```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3
```

```
[ ]: sample_incomplete_rows
```

```
[ ]:        longitude  latitude  …  median_income  ocean_proximity
     4629     -118.30     34.07  …          2.2708        <1H OCEAN
     6068     -117.86     34.01  …          5.1762        <1H OCEAN
     17923    -121.97     37.35  …          4.6328        <1H OCEAN
     13656    -117.30     34.05  …          1.6675           INLAND
     19252    -122.79     38.48  …          3.1662        <1H OCEAN

     [5 rows x 9 columns]
```

```
[ ]: from sklearn.impute import SimpleImputer
     imputer = SimpleImputer(strategy="median")
```

Remove the text attribute because median can only be calculated on numerical attributes:

```
[ ]: housing_num = housing.drop("ocean_proximity", axis=1)
     # alternatively: housing_num = housing.select_dtypes(include=[np.number])
```

```
[ ]: imputer.fit(housing_num)
```

```
[ ]: SimpleImputer(add_indicator=False, copy=True, fill_value=None,
                   missing_values=nan, strategy='median', verbose=0)
```

```
[ ]: imputer.statistics_
```

```
[ ]: array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
            408.    ,    3.5409])
```

Check that this is the same as manually computing the median of each attribute:

```
[ ]: housing_num.median().values
```

```
[ ]: array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
            408.    ,    3.5409])
```

Transform the training set:

```
[ ]: X = imputer.transform(housing_num)
```

```
[ ]: housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                               index=housing.index)
```

```
[ ]: housing_tr.loc[sample_incomplete_rows.index.values]
```

```
[ ]:        longitude  latitude  …  households  median_income
     4629     -118.30     34.07  …      1462.0         2.2708
     6068     -117.86     34.01  …       727.0         5.1762
     17923    -121.97     37.35  …       386.0         4.6328
```

```
13656    -117.30     34.05  …       391.0         1.6675
19252    -122.79     38.48  …      1405.0         3.1662

[5 rows x 8 columns]
```

[ ]:
```
imputer.strategy
```

[ ]:
```
'median'
```

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

[ ]:
```
housing_tr.head()
```

[ ]:
```
       longitude  latitude  …  households  median_income
17606    -121.89     37.29  …       339.0         2.7042
18632    -121.93     37.05  …       113.0         6.4214
14650    -117.20     32.77  …       462.0         2.8621
3230     -119.61     36.31  …       353.0         1.8839
3555     -118.59     34.23  …      1463.0         3.0347

[5 rows x 8 columns]
```

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

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

[ ]:
```
      ocean_proximity
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
```

[ ]:
```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

[ ]:
```
array([[0.],
       [0.],
```

```
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

[ ]:
```
ordinal_encoder.categories_
```

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

[ ]:
```
from sklearn.preprocessing import OneHotEncoder

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

[ ]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

By default, the `OneHotEncoder` class returns a sparse array, but we can convert it to a dense array if needed by calling the `toarray()` method:

[ ]:
```
housing_cat_1hot.toarray()
```

[ ]:
```
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.]])
```

Alternatively, you can set `sparse=False` when creating the `OneHotEncoder`:

[ ]:
```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

[ ]:
```
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.]])
```

[ ]:
```
cat_encoder.categories_
```

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

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

[ ]:
```
from sklearn.base import BaseEstimator, TransformerMixin

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

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # 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):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_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]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Note that I hard coded the indices (3, 4, 5, 6) for concision and clarity in the book, but it would be much cleaner to get them dynamically, like this:

[ ]:
```
col_names = "total_rooms", "total_bedrooms", "population", "households"
rooms_ix, bedrooms_ix, population_ix, households_ix = [
    housing.columns.get_loc(c) for c in col_names] # get the column indices
```

Also, `housing_extra_attribs` is a NumPy array, we've lost the column names (unfortunately, that's a problem with Scikit-Learn). To recover a `DataFrame`, you could run this:

[ ]:
```
housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household",␣
 ↪"population_per_household"],
    index=housing.index)
housing_extra_attribs.head()
```

[ ]:
```
       longitude latitude  … rooms_per_household population_per_household
17606    -121.89    37.29  …              4.62537                    2.0944
```

```
18632   -121.93    37.05  …            6.00885                  2.70796
14650    -117.2    32.77  …            4.22511                  2.02597
3230    -119.61    36.31  …            5.23229                  4.13598
3555    -118.59    34.23  …            4.50581                  3.04785

[5 rows x 11 columns]
```

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="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

[ ]:
```
housing_num_tr
```

[ ]:
```
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]])
```

[ ]:
```
from sklearn.compose import ColumnTransformer

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)
```

[ ]:
```
housing_prepared
```

[ ]:
```
array([[-1.15604281,  0.77194962,  0.74333089, …,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , …,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, …,  0.        ,
         0.        ,  1.        ],
       …,
       [ 1.58648943, -0.72478134, -1.56295222, …,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, …,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, …,  0.        ,
         1.        ,  0.        ]])
```

[ ]:
```
housing_prepared.shape
```

[ ]:
```
(16512, 16)
```

For reference, here is the old solution based on a `DataFrameSelector` transformer (to just select a subset of the Pandas `DataFrame` columns), and a `FeatureUnion`:

[ ]:
```
from sklearn.base import BaseEstimator, TransformerMixin

# Create a class to select numerical or categorical columns
class OldDataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

Now let's join all these components into a big pipeline that will preprocess both the numerical and the categorical features:

[ ]:
```
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

old_num_pipeline = Pipeline([
        ('selector', OldDataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

old_cat_pipeline = Pipeline([
```

```
        ('selector', OldDataFrameSelector(cat_attribs)),
        ('cat_encoder', OneHotEncoder(sparse=False)),
    ])
```

[ ]:
```
from sklearn.pipeline import FeatureUnion

old_full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", old_num_pipeline),
        ("cat_pipeline", old_cat_pipeline),
    ])
```

[ ]:
```
old_housing_prepared = old_full_pipeline.fit_transform(housing)
old_housing_prepared
```

[ ]:
```
array([[-1.15604281,  0.77194962,  0.74333089, …,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , …,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, …,  0.        ,
         0.        ,  1.        ],
       …,
       [ 1.58648943, -0.72478134, -1.56295222, …,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, …,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, …,  0.        ,
         1.        ,  0.        ]])
```

The result is the same as with the `ColumnTransformer`:

[ ]:
```
np.allclose(housing_prepared, old_housing_prepared)
```

[ ]:
```
True
```

5 Select and train a model

[ ]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

[ ]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[ ]:
```
# let's try the full preprocessing pipeline on a few training instances
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)
```

```
print("Predictions:", lin_reg.predict(some_data_prepared))
```

```
Predictions: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

Compare against the actual values:

[ ]:
```
print("Labels:", list(some_labels))
```

```
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

[ ]:
```
some_data_prepared
```

[ ]:
```
array([[-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
        -0.63621141, -0.42069842, -0.61493744, -0.31205452, -0.08649871,
         0.15531753,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
        -0.99833135, -1.02222705,  1.33645936,  0.21768338, -0.03353391,
        -0.83628902,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, -0.31365989, -0.15334458,
        -0.43363936, -0.0933178 , -0.5320456 , -0.46531516, -0.09240499,
         0.4222004 ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [-0.01706767,  0.31357576, -0.29052016, -0.36276217, -0.39675594,
         0.03604096, -0.38343559, -1.04556555, -0.07966124,  0.08973561,
        -0.19645314,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.49247384, -0.65929936, -0.92673619,  1.85619316,  2.41221109,
         2.72415407,  2.57097492, -0.44143679, -0.35783383, -0.00419445,
         0.2699277 ,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ]])
```

[ ]:
```
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
```

[ ]:
```
68628.19819848923
```

Note: since Scikit-Learn 0.22, you can get the RMSE directly by calling the `mean_squared_error()` function with `squared=False`.

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

```
lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

[ ]: 49439.89599001897

[ ]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

[ ]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

[ ]:
```
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

6 Fine-tune your model

[ ]:
```
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)
tree_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: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

[ ]:
```
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: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798344
```

Note: we specify `n_estimators=100` to be future-proof since the default value is going to change to 100 in Scikit-Learn 0.22 (for simplicity, this is not shown in the book).

[ ]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

[ ]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

[ ]:
```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

[ ]:
```
18603.515021376355
```

[ ]:
```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

[ ]:
```
scores = cross_val_score(lin_reg, housing_prepared, housing_labels,␣
 ↪scoring="neg_mean_squared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

[ ]:
```
count       10.000000
mean     69052.461363
std       2879.437224
min      64969.630564
25%      67136.363758
50%      68156.372635
75%      70982.369487
max      74739.570526
dtype: float64
```

[ ]:
```
from sklearn.svm import SVR

svm_reg = SVR(kernel="linear")
svm_reg.fit(housing_prepared, housing_labels)
housing_predictions = svm_reg.predict(housing_prepared)
svm_mse = mean_squared_error(housing_labels, housing_predictions)
svm_rmse = np.sqrt(svm_mse)
svm_rmse
```

[ ]:
```
111094.6308539982
```

[ ]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

[ ]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
```

```
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators=100, n_jobs=None,
                                       oob_score=False, random_state=42,
                                       verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_squared_error', verbose=0)
```

The best hyperparameter combination found:

[ ]:
```
grid_search.best_params_
```

[ ]:
```
{'max_features': 8, 'n_estimators': 30}
```

[ ]:
```
grid_search.best_estimator_
```

[ ]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=8, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=30, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

Let's look at the score of each hyperparameter combination tested during the grid search:

[ ]:
```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
63669.11631261028 {'max_features': 2, 'n_estimators': 3}
55627.099719926795 {'max_features': 2, 'n_estimators': 10}
53384.57275149205 {'max_features': 2, 'n_estimators': 30}
60965.950449450494 {'max_features': 4, 'n_estimators': 3}
52741.04704299915 {'max_features': 4, 'n_estimators': 10}
50377.40461678399 {'max_features': 4, 'n_estimators': 30}
58663.93866579625 {'max_features': 6, 'n_estimators': 3}
52006.19873526564 {'max_features': 6, 'n_estimators': 10}
50146.51167415009 {'max_features': 6, 'n_estimators': 30}
57869.25276169646 {'max_features': 8, 'n_estimators': 3}
51711.127883959234 {'max_features': 8, 'n_estimators': 10}
49682.273345071546 {'max_features': 8, 'n_estimators': 30}
62895.06951262424 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.176157539405 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.40652318466 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
```

```
52724.9822587892 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.5691951261 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.495668875716 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

[ ]:
```
pd.DataFrame(grid_search.cv_results_)
```

[ ]:

	mean_fit_time	std_fit_time	…	mean_train_score	std_train_score
0	0.075469	0.002116	…	-1.105559e+09	2.220402e+07
1	0.252199	0.004846	…	-5.818785e+08	7.345821e+06
2	0.718751	0.023780	…	-4.394734e+08	2.966320e+06
3	0.124153	0.008283	…	-9.848396e+08	4.084607e+07
4	0.385951	0.010452	…	-5.163863e+08	1.542862e+07
5	1.149166	0.020713	…	-3.879289e+08	8.571233e+06
6	0.157751	0.004441	…	-9.023976e+08	2.591445e+07
7	0.522872	0.013427	…	-5.013349e+08	3.100456e+06
8	1.577083	0.014839	…	-3.841296e+08	3.617057e+06
9	0.196157	0.002068	…	-8.883545e+08	2.750227e+07
10	0.662111	0.009898	…	-4.923911e+08	1.459294e+07
11	2.047319	0.011783	…	-3.810330e+08	4.871017e+06
12	0.109765	0.001124	…	0.000000e+00	0.000000e+00
13	0.373219	0.007022	…	-6.056027e-01	1.181156e+00
14	0.147413	0.001658	…	-1.214568e+01	2.429136e+01
15	0.484351	0.011663	…	-5.272080e+00	8.093117e+00
16	0.181339	0.004151	…	0.000000e+00	0.000000e+00
17	0.606996	0.010032	…	-3.028238e-03	6.056477e-03

[18 rows x 23 columns]

[ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5,␣
 ↪scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

[ ]:
```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
```

```
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs=None,
oob_score=Fals…
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_features':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7f3f221f4210>,
                                       'n_estimators':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7f3f221f4850>},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

[ ]:
```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49150.70756927707 {'max_features': 7, 'n_estimators': 180}
51389.889203389284 {'max_features': 5, 'n_estimators': 15}
50796.155224308866 {'max_features': 3, 'n_estimators': 72}
50835.13360315349 {'max_features': 5, 'n_estimators': 21}
49280.9449827171 {'max_features': 7, 'n_estimators': 122}
50774.90662363929 {'max_features': 3, 'n_estimators': 75}
50682.78888164288 {'max_features': 3, 'n_estimators': 88}
49608.99608105296 {'max_features': 5, 'n_estimators': 100}
50473.61930350219 {'max_features': 3, 'n_estimators': 150}
64429.84143294435 {'max_features': 5, 'n_estimators': 2}
```

[ ]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

[ ]:
```
array([7.33442355e-02, 6.29090705e-02, 4.11437985e-02, 1.46726854e-02,
       1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,
       5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,
       1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03])
```

[ ]:
```
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
#cat_encoder = cat_pipeline.named_steps["cat_encoder"] # old solution
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
```

```
sorted(zip(feature_importances, attributes), reverse=True)
```

[ ]:
```
[(0.36615898061813423, 'median_income'),
 (0.16478099356159054, 'INLAND'),
 (0.10879295677551575, 'pop_per_hhold'),
 (0.07334423551601243, 'longitude'),
 (0.06290907048262032, 'latitude'),
 (0.056419179181954014, 'rooms_per_hhold'),
 (0.053351077347675815, 'bedrooms_per_room'),
 (0.04114379847872964, 'housing_median_age'),
 (0.014874280890402769, 'population'),
 (0.014672685420543239, 'total_rooms'),
 (0.014257599323407808, 'households'),
 (0.014106483453584104, 'total_bedrooms'),
 (0.010311488326303788, '<1H OCEAN'),
 (0.0028564746373201584, 'NEAR OCEAN'),
 (0.0019604155994780706, 'NEAR BAY'),
 (6.0280386727366e-05, 'ISLAND')]
```

[ ]:
```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

[ ]:
```
final_rmse
```

[ ]:
```
47730.22690385927
```

We can compute a 95% confidence interval for the test RMSE:

[ ]:
```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

[ ]:
```
array([45685.10470776, 49691.25001878])
```

We could compute the interval manually like this:

[ ]:
```
m = len(squared_errors)
mean = squared_errors.mean()
tscore = stats.t.ppf((1 + confidence) / 2, df=m - 1)
tmargin = tscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - tmargin), np.sqrt(mean + tmargin)
```

[ ]:
```
(45685.10470776, 49691.25001877858)
```

Alternatively, we could use a z-scores rather than t-scores:

[ ]:
```
zscore = stats.norm.ppf((1 + confidence) / 2)
zmargin = zscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - zmargin), np.sqrt(mean + zmargin)
```

[ ]:
```
(45685.717918136455, 49690.68623889413)
```

7 Extra material

7.1 A full pipeline with both preparation and prediction

[ ]:
```
full_pipeline_with_predictor = Pipeline([
        ("preparation", full_pipeline),
        ("linear", LinearRegression())
    ])

full_pipeline_with_predictor.fit(housing, housing_labels)
full_pipeline_with_predictor.predict(some_data)
```

[ ]:
```
array([210644.60459286, 317768.80697211, 210956.43331178,  59218.98886849,
       189747.55849879])
```

7.2 Model persistence using joblib

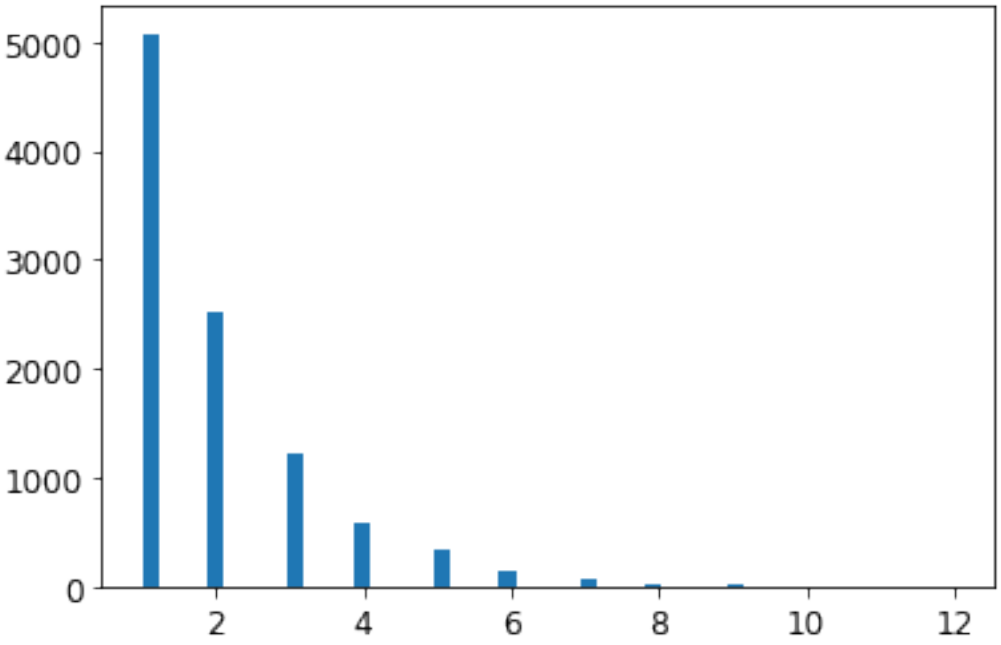
[ ]:
```
my_model = full_pipeline_with_predictor
```

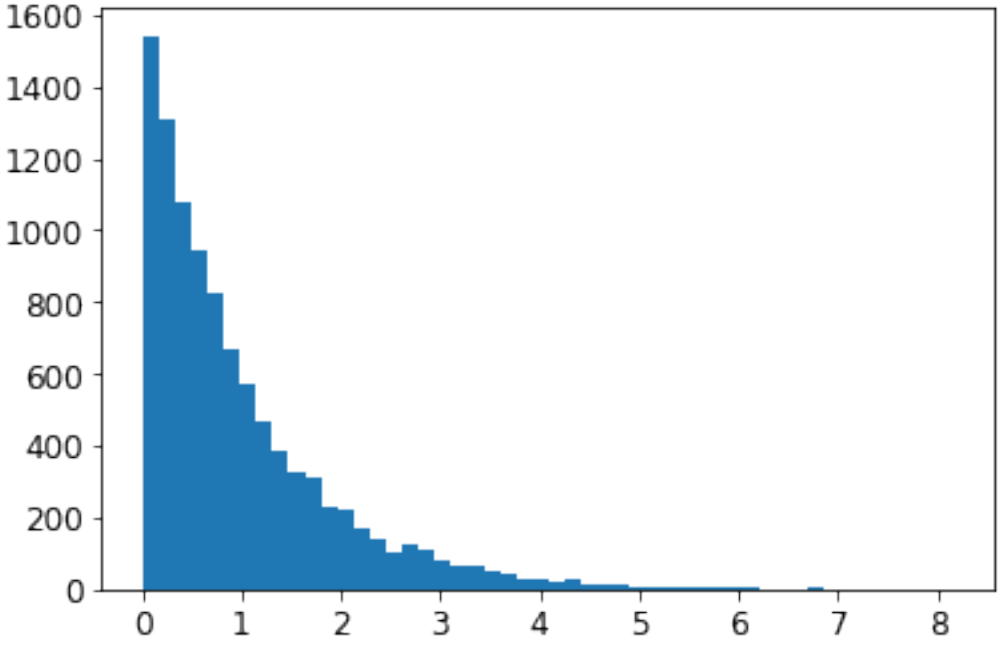
[ ]:
```
import joblib
joblib.dump(my_model, "my_model.pkl") # DIFF
#...
my_model_loaded = joblib.load("my_model.pkl") # DIFF
```

7.3 Example SciPy distributions for `RandomizedSearchCV`

[ ]:
```
from scipy.stats import geom, expon
geom_distrib=geom(0.5).rvs(10000, random_state=42)
expon_distrib=expon(scale=1).rvs(10000, random_state=42)
plt.hist(geom_distrib, bins=50)
plt.show()
plt.hist(expon_distrib, bins=50)
```

```
plt.show()
```


5000
4000
3000
2000
1000
0
2
4
6
8
10
12


1600
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7
8

8 Exercise solutions

8.1 1.

Question: Try a Support Vector Machine regressor (`sklearn.svm.SVR`), with various hyperparameters such as `kernel="linear"` (with various values for the `C` hyperparameter) or `kernel="rbf"` (with various values for the `C` and `gamma` hyperparameters). Don't worry about what these hyperparameters mean for now. How does the best `SVR` predictor perform?

Warning: the following cell may take close to 30 minutes to run, or more depending on your hardware.

[ ]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
        {'kernel': ['linear'], 'C': [10., 30., 100., 300., 1000., 3000., 10000.
 ↪, 30000.0]},
        {'kernel': ['rbf'], 'C': [1.0, 3.0, 10., 30., 100., 300., 1000.0],
         'gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0]},
    ]

svm_reg = SVR()
grid_search = GridSearchCV(svm_reg, param_grid, cv=5,␣
 ↪scoring='neg_mean_squared_error', verbose=2)
grid_search.fit(housing_prepared, housing_labels)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
[CV] C=10.0, kernel=linear …

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] … C=10.0, kernel=linear, total=  10.7s
[CV] C=10.0, kernel=linear …

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   10.7s remaining:    0.0s

[CV] … C=10.0, kernel=linear, total=  10.1s
[CV] C=10.0, kernel=linear …
[CV] … C=10.0, kernel=linear, total=   9.8s
[CV] C=10.0, kernel=linear …
[CV] … C=10.0, kernel=linear, total=   9.9s
[CV] C=10.0, kernel=linear …
[CV] … C=10.0, kernel=linear, total=   9.8s
[CV] C=30.0, kernel=linear …
[CV] … C=30.0, kernel=linear, total=   9.8s
[CV] C=30.0, kernel=linear …
[CV] … C=30.0, kernel=linear, total=   9.7s
[CV] C=30.0, kernel=linear …
[CV] … C=30.0, kernel=linear, total=  10.1s
[CV] C=30.0, kernel=linear …
[CV] … C=30.0, kernel=linear, total=  10.0s
```

```
[CV] C=30.0, kernel=linear …
[CV] … C=30.0, kernel=linear, total=   9.6s
[CV] C=100.0, kernel=linear …
[CV] … C=100.0, kernel=linear, total=   9.7s
[CV] C=100.0, kernel=linear …
[CV] … C=100.0, kernel=linear, total=   9.7s
[CV] C=100.0, kernel=linear …
[CV] … C=100.0, kernel=linear, total=   9.8s
[CV] C=100.0, kernel=linear …
[CV] … C=100.0, kernel=linear, total=   9.7s
[CV] C=100.0, kernel=linear …
[CV] … C=100.0, kernel=linear, total=   9.5s
[CV] C=300.0, kernel=linear …
[CV] … C=300.0, kernel=linear, total=   9.8s
[CV] C=300.0, kernel=linear …
[CV] … C=300.0, kernel=linear, total=   9.7s
[CV] C=300.0, kernel=linear …
[CV] … C=300.0, kernel=linear, total=  10.0s
[CV] C=300.0, kernel=linear …
[CV] … C=300.0, kernel=linear, total=   9.9s
[CV] C=300.0, kernel=linear …
[CV] … C=300.0, kernel=linear, total=   9.7s
[CV] C=1000.0, kernel=linear …
[CV] … C=1000.0, kernel=linear, total=  10.2s
[CV] C=1000.0, kernel=linear …
[CV] … C=1000.0, kernel=linear, total=  10.3s
[CV] C=1000.0, kernel=linear …
[CV] … C=1000.0, kernel=linear, total=  10.2s
[CV] C=1000.0, kernel=linear …
[CV] … C=1000.0, kernel=linear, total=  10.3s
[CV] C=1000.0, kernel=linear …
[CV] … C=1000.0, kernel=linear, total=  10.1s
[CV] C=3000.0, kernel=linear …
[CV] … C=3000.0, kernel=linear, total=  11.1s
[CV] C=3000.0, kernel=linear …
[CV] … C=3000.0, kernel=linear, total=  10.8s
[CV] C=3000.0, kernel=linear …
[CV] … C=3000.0, kernel=linear, total=  11.3s
[CV] C=3000.0, kernel=linear …
[CV] … C=3000.0, kernel=linear, total=  11.1s
[CV] C=3000.0, kernel=linear …
[CV] … C=3000.0, kernel=linear, total=  10.8s
[CV] C=10000.0, kernel=linear …
[CV] … C=10000.0, kernel=linear, total=  14.8s
[CV] C=10000.0, kernel=linear …
[CV] … C=10000.0, kernel=linear, total=  15.2s
[CV] C=10000.0, kernel=linear …
[CV] … C=10000.0, kernel=linear, total=  15.3s
```

```
[CV] C=10000.0, kernel=linear …
[CV] … C=10000.0, kernel=linear, total=  14.1s
[CV] C=10000.0, kernel=linear …
[CV] … C=10000.0, kernel=linear, total=  13.4s
[CV] C=30000.0, kernel=linear …
[CV] … C=30000.0, kernel=linear, total=  24.5s
[CV] C=30000.0, kernel=linear …
[CV] … C=30000.0, kernel=linear, total=  24.8s
[CV] C=30000.0, kernel=linear …
[CV] … C=30000.0, kernel=linear, total=  26.1s
[CV] C=30000.0, kernel=linear …
[CV] … C=30000.0, kernel=linear, total=  24.8s
[CV] C=30000.0, kernel=linear …
[CV] … C=30000.0, kernel=linear, total=  22.0s
[CV] C=1.0, gamma=0.01, kernel=rbf …
[CV] … C=1.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=1.0, gamma=0.01, kernel=rbf …
[CV] … C=1.0, gamma=0.01, kernel=rbf, total=  17.2s
[CV] C=1.0, gamma=0.01, kernel=rbf …
[CV] … C=1.0, gamma=0.01, kernel=rbf, total=  17.2s
[CV] C=1.0, gamma=0.01, kernel=rbf …
[CV] … C=1.0, gamma=0.01, kernel=rbf, total=  17.2s
[CV] C=1.0, gamma=0.01, kernel=rbf …
[CV] … C=1.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=1.0, gamma=0.03, kernel=rbf …
[CV] … C=1.0, gamma=0.03, kernel=rbf, total=  17.2s
[CV] C=1.0, gamma=0.03, kernel=rbf …
[CV] … C=1.0, gamma=0.03, kernel=rbf, total=  17.6s
[CV] C=1.0, gamma=0.03, kernel=rbf …
[CV] … C=1.0, gamma=0.03, kernel=rbf, total=  17.5s
[CV] C=1.0, gamma=0.03, kernel=rbf …
[CV] … C=1.0, gamma=0.03, kernel=rbf, total=  17.2s
[CV] C=1.0, gamma=0.03, kernel=rbf …
[CV] … C=1.0, gamma=0.03, kernel=rbf, total=  17.2s
[CV] C=1.0, gamma=0.1, kernel=rbf …
[CV] … C=1.0, gamma=0.1, kernel=rbf, total=  16.8s
[CV] C=1.0, gamma=0.1, kernel=rbf …
[CV] … C=1.0, gamma=0.1, kernel=rbf, total=  16.9s
[CV] C=1.0, gamma=0.1, kernel=rbf …
[CV] … C=1.0, gamma=0.1, kernel=rbf, total=  16.9s
[CV] C=1.0, gamma=0.1, kernel=rbf …
[CV] … C=1.0, gamma=0.1, kernel=rbf, total=  17.1s
[CV] C=1.0, gamma=0.1, kernel=rbf …
[CV] … C=1.0, gamma=0.1, kernel=rbf, total=  17.0s
[CV] C=1.0, gamma=0.3, kernel=rbf …
[CV] … C=1.0, gamma=0.3, kernel=rbf, total=  16.4s
[CV] C=1.0, gamma=0.3, kernel=rbf …
[CV] … C=1.0, gamma=0.3, kernel=rbf, total=  16.3s
```

```
[CV] C=1.0, gamma=0.3, kernel=rbf …
[CV] … C=1.0, gamma=0.3, kernel=rbf, total=  16.3s
[CV] C=1.0, gamma=0.3, kernel=rbf …
[CV] … C=1.0, gamma=0.3, kernel=rbf, total=  16.4s
[CV] C=1.0, gamma=0.3, kernel=rbf …
[CV] … C=1.0, gamma=0.3, kernel=rbf, total=  16.4s
[CV] C=1.0, gamma=1.0, kernel=rbf …
[CV] … C=1.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=1.0, gamma=1.0, kernel=rbf …
[CV] … C=1.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=1.0, gamma=1.0, kernel=rbf …
[CV] … C=1.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=1.0, gamma=1.0, kernel=rbf …
[CV] … C=1.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=1.0, gamma=1.0, kernel=rbf …
[CV] … C=1.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=1.0, gamma=3.0, kernel=rbf …
[CV] … C=1.0, gamma=3.0, kernel=rbf, total=  16.6s
[CV] C=1.0, gamma=3.0, kernel=rbf …
[CV] … C=1.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=1.0, gamma=3.0, kernel=rbf …
[CV] … C=1.0, gamma=3.0, kernel=rbf, total=  16.4s
[CV] C=1.0, gamma=3.0, kernel=rbf …
[CV] … C=1.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=1.0, gamma=3.0, kernel=rbf …
[CV] … C=1.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=3.0, gamma=0.01, kernel=rbf …
[CV] … C=3.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=3.0, gamma=0.01, kernel=rbf …
[CV] … C=3.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=3.0, gamma=0.01, kernel=rbf …
[CV] … C=3.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=3.0, gamma=0.01, kernel=rbf …
[CV] … C=3.0, gamma=0.01, kernel=rbf, total=  17.0s
[CV] C=3.0, gamma=0.01, kernel=rbf …
[CV] … C=3.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=3.0, gamma=0.03, kernel=rbf …
[CV] … C=3.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=3.0, gamma=0.03, kernel=rbf …
[CV] … C=3.0, gamma=0.03, kernel=rbf, total=  17.1s
[CV] C=3.0, gamma=0.03, kernel=rbf …
[CV] … C=3.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=3.0, gamma=0.03, kernel=rbf …
[CV] … C=3.0, gamma=0.03, kernel=rbf, total=  17.1s
[CV] C=3.0, gamma=0.03, kernel=rbf …
[CV] … C=3.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=3.0, gamma=0.1, kernel=rbf …
[CV] … C=3.0, gamma=0.1, kernel=rbf, total=  16.8s
```

```
[CV] C=3.0, gamma=0.1, kernel=rbf …
[CV] … C=3.0, gamma=0.1, kernel=rbf, total=  16.9s
[CV] C=3.0, gamma=0.1, kernel=rbf …
[CV] … C=3.0, gamma=0.1, kernel=rbf, total=  16.8s
[CV] C=3.0, gamma=0.1, kernel=rbf …
[CV] … C=3.0, gamma=0.1, kernel=rbf, total=  16.9s
[CV] C=3.0, gamma=0.1, kernel=rbf …
[CV] … C=3.0, gamma=0.1, kernel=rbf, total=  17.0s
[CV] C=3.0, gamma=0.3, kernel=rbf …
[CV] … C=3.0, gamma=0.3, kernel=rbf, total=  16.3s
[CV] C=3.0, gamma=0.3, kernel=rbf …
[CV] … C=3.0, gamma=0.3, kernel=rbf, total=  16.3s
[CV] C=3.0, gamma=0.3, kernel=rbf …
[CV] … C=3.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=3.0, gamma=0.3, kernel=rbf …
[CV] … C=3.0, gamma=0.3, kernel=rbf, total=  16.3s
[CV] C=3.0, gamma=0.3, kernel=rbf …
[CV] … C=3.0, gamma=0.3, kernel=rbf, total=  16.3s
[CV] C=3.0, gamma=1.0, kernel=rbf …
[CV] … C=3.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=3.0, gamma=1.0, kernel=rbf …
[CV] … C=3.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=3.0, gamma=1.0, kernel=rbf …
[CV] … C=3.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=3.0, gamma=1.0, kernel=rbf …
[CV] … C=3.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=3.0, gamma=1.0, kernel=rbf …
[CV] … C=3.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=3.0, gamma=3.0, kernel=rbf …
[CV] … C=3.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=3.0, gamma=3.0, kernel=rbf …
[CV] … C=3.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=3.0, gamma=3.0, kernel=rbf …
[CV] … C=3.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=3.0, gamma=3.0, kernel=rbf …
[CV] … C=3.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=3.0, gamma=3.0, kernel=rbf …
[CV] … C=3.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=10.0, gamma=0.01, kernel=rbf …
[CV] … C=10.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=10.0, gamma=0.01, kernel=rbf …
[CV] … C=10.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=10.0, gamma=0.01, kernel=rbf …
[CV] … C=10.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=10.0, gamma=0.01, kernel=rbf …
[CV] … C=10.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=10.0, gamma=0.01, kernel=rbf …
[CV] … C=10.0, gamma=0.01, kernel=rbf, total=  17.1s
```

```
[CV] C=10.0, gamma=0.03, kernel=rbf …
[CV] … C=10.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=10.0, gamma=0.03, kernel=rbf …
[CV] … C=10.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=10.0, gamma=0.03, kernel=rbf …
[CV] … C=10.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=10.0, gamma=0.03, kernel=rbf …
[CV] … C=10.0, gamma=0.03, kernel=rbf, total=  16.9s
[CV] C=10.0, gamma=0.03, kernel=rbf …
[CV] … C=10.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=10.0, gamma=0.1, kernel=rbf …
[CV] … C=10.0, gamma=0.1, kernel=rbf, total=  16.7s
[CV] C=10.0, gamma=0.1, kernel=rbf …
[CV] … C=10.0, gamma=0.1, kernel=rbf, total=  16.8s
[CV] C=10.0, gamma=0.1, kernel=rbf …
[CV] … C=10.0, gamma=0.1, kernel=rbf, total=  16.8s
[CV] C=10.0, gamma=0.1, kernel=rbf …
[CV] … C=10.0, gamma=0.1, kernel=rbf, total=  16.7s
[CV] C=10.0, gamma=0.1, kernel=rbf …
[CV] … C=10.0, gamma=0.1, kernel=rbf, total=  16.7s
[CV] C=10.0, gamma=0.3, kernel=rbf …
[CV] … C=10.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=10.0, gamma=0.3, kernel=rbf …
[CV] … C=10.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=10.0, gamma=0.3, kernel=rbf …
[CV] … C=10.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=10.0, gamma=0.3, kernel=rbf …
[CV] … C=10.0, gamma=0.3, kernel=rbf, total=  16.1s
[CV] C=10.0, gamma=0.3, kernel=rbf …
[CV] … C=10.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=10.0, gamma=1.0, kernel=rbf …
[CV] … C=10.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=10.0, gamma=1.0, kernel=rbf …
[CV] … C=10.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=10.0, gamma=1.0, kernel=rbf …
[CV] … C=10.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=10.0, gamma=1.0, kernel=rbf …
[CV] … C=10.0, gamma=1.0, kernel=rbf, total=  15.7s
[CV] C=10.0, gamma=1.0, kernel=rbf …
[CV] … C=10.0, gamma=1.0, kernel=rbf, total=  15.5s
[CV] C=10.0, gamma=3.0, kernel=rbf …
[CV] … C=10.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=10.0, gamma=3.0, kernel=rbf …
[CV] … C=10.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=10.0, gamma=3.0, kernel=rbf …
[CV] … C=10.0, gamma=3.0, kernel=rbf, total=  16.1s
[CV] C=10.0, gamma=3.0, kernel=rbf …
[CV] … C=10.0, gamma=3.0, kernel=rbf, total=  16.3s
```

```
[CV] C=10.0, gamma=3.0, kernel=rbf …
[CV] … C=10.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=30.0, gamma=0.01, kernel=rbf …
[CV] … C=30.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=30.0, gamma=0.01, kernel=rbf …
[CV] … C=30.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=30.0, gamma=0.01, kernel=rbf …
[CV] … C=30.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=30.0, gamma=0.01, kernel=rbf …
[CV] … C=30.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=30.0, gamma=0.01, kernel=rbf …
[CV] … C=30.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=30.0, gamma=0.03, kernel=rbf …
[CV] … C=30.0, gamma=0.03, kernel=rbf, total=  16.9s
[CV] C=30.0, gamma=0.03, kernel=rbf …
[CV] … C=30.0, gamma=0.03, kernel=rbf, total=  16.9s
[CV] C=30.0, gamma=0.03, kernel=rbf …
[CV] … C=30.0, gamma=0.03, kernel=rbf, total=  16.9s
[CV] C=30.0, gamma=0.03, kernel=rbf …
[CV] … C=30.0, gamma=0.03, kernel=rbf, total=  16.9s
[CV] C=30.0, gamma=0.03, kernel=rbf …
[CV] … C=30.0, gamma=0.03, kernel=rbf, total=  17.0s
[CV] C=30.0, gamma=0.1, kernel=rbf …
[CV] … C=30.0, gamma=0.1, kernel=rbf, total=  16.7s
[CV] C=30.0, gamma=0.1, kernel=rbf …
[CV] … C=30.0, gamma=0.1, kernel=rbf, total=  16.8s
[CV] C=30.0, gamma=0.1, kernel=rbf …
[CV] … C=30.0, gamma=0.1, kernel=rbf, total=  16.7s
[CV] C=30.0, gamma=0.1, kernel=rbf …
[CV] … C=30.0, gamma=0.1, kernel=rbf, total=  16.6s
[CV] C=30.0, gamma=0.1, kernel=rbf …
[CV] … C=30.0, gamma=0.1, kernel=rbf, total=  16.7s
[CV] C=30.0, gamma=0.3, kernel=rbf …
[CV] … C=30.0, gamma=0.3, kernel=rbf, total=  16.1s
[CV] C=30.0, gamma=0.3, kernel=rbf …
[CV] … C=30.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=30.0, gamma=0.3, kernel=rbf …
[CV] … C=30.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=30.0, gamma=0.3, kernel=rbf …
[CV] … C=30.0, gamma=0.3, kernel=rbf, total=  16.2s
[CV] C=30.0, gamma=0.3, kernel=rbf …
[CV] … C=30.0, gamma=0.3, kernel=rbf, total=  16.1s
[CV] C=30.0, gamma=1.0, kernel=rbf …
[CV] … C=30.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=30.0, gamma=1.0, kernel=rbf …
[CV] … C=30.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=30.0, gamma=1.0, kernel=rbf …
[CV] … C=30.0, gamma=1.0, kernel=rbf, total=  15.6s
```

```
[CV] C=30.0, gamma=1.0, kernel=rbf …
[CV] … C=30.0, gamma=1.0, kernel=rbf, total=  15.5s
[CV] C=30.0, gamma=1.0, kernel=rbf …
[CV] … C=30.0, gamma=1.0, kernel=rbf, total=  15.5s
[CV] C=30.0, gamma=3.0, kernel=rbf …
[CV] … C=30.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=30.0, gamma=3.0, kernel=rbf …
[CV] … C=30.0, gamma=3.0, kernel=rbf, total=  16.4s
[CV] C=30.0, gamma=3.0, kernel=rbf …
[CV] … C=30.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=30.0, gamma=3.0, kernel=rbf …
[CV] … C=30.0, gamma=3.0, kernel=rbf, total=  16.4s
[CV] C=30.0, gamma=3.0, kernel=rbf …
[CV] … C=30.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=100.0, gamma=0.01, kernel=rbf …
[CV] … C=100.0, gamma=0.01, kernel=rbf, total=  17.0s
[CV] C=100.0, gamma=0.01, kernel=rbf …
[CV] … C=100.0, gamma=0.01, kernel=rbf, total=  17.1s
[CV] C=100.0, gamma=0.01, kernel=rbf …
[CV] … C=100.0, gamma=0.01, kernel=rbf, total=  16.9s
[CV] C=100.0, gamma=0.01, kernel=rbf …
[CV] … C=100.0, gamma=0.01, kernel=rbf, total=  17.0s
[CV] C=100.0, gamma=0.01, kernel=rbf …
[CV] … C=100.0, gamma=0.01, kernel=rbf, total=  17.0s
[CV] C=100.0, gamma=0.03, kernel=rbf …
[CV] … C=100.0, gamma=0.03, kernel=rbf, total=  16.7s
[CV] C=100.0, gamma=0.03, kernel=rbf …
[CV] … C=100.0, gamma=0.03, kernel=rbf, total=  16.7s
[CV] C=100.0, gamma=0.03, kernel=rbf …
[CV] … C=100.0, gamma=0.03, kernel=rbf, total=  16.6s
[CV] C=100.0, gamma=0.03, kernel=rbf …
[CV] … C=100.0, gamma=0.03, kernel=rbf, total=  16.6s
[CV] C=100.0, gamma=0.03, kernel=rbf …
[CV] … C=100.0, gamma=0.03, kernel=rbf, total=  16.7s
[CV] C=100.0, gamma=0.1, kernel=rbf …
[CV] … C=100.0, gamma=0.1, kernel=rbf, total=  16.3s
[CV] C=100.0, gamma=0.1, kernel=rbf …
[CV] … C=100.0, gamma=0.1, kernel=rbf, total=  16.3s
[CV] C=100.0, gamma=0.1, kernel=rbf …
[CV] … C=100.0, gamma=0.1, kernel=rbf, total=  16.2s
[CV] C=100.0, gamma=0.1, kernel=rbf …
[CV] … C=100.0, gamma=0.1, kernel=rbf, total=  16.3s
[CV] C=100.0, gamma=0.1, kernel=rbf …
[CV] … C=100.0, gamma=0.1, kernel=rbf, total=  16.2s
[CV] C=100.0, gamma=0.3, kernel=rbf …
[CV] … C=100.0, gamma=0.3, kernel=rbf, total=  16.0s
[CV] C=100.0, gamma=0.3, kernel=rbf …
[CV] … C=100.0, gamma=0.3, kernel=rbf, total=  16.0s
```

```
[CV] C=100.0, gamma=0.3, kernel=rbf …
[CV] … C=100.0, gamma=0.3, kernel=rbf, total=  16.0s
[CV] C=100.0, gamma=0.3, kernel=rbf …
[CV] … C=100.0, gamma=0.3, kernel=rbf, total=  16.0s
[CV] C=100.0, gamma=0.3, kernel=rbf …
[CV] … C=100.0, gamma=0.3, kernel=rbf, total=  16.0s
[CV] C=100.0, gamma=1.0, kernel=rbf …
[CV] … C=100.0, gamma=1.0, kernel=rbf, total=  15.5s
[CV] C=100.0, gamma=1.0, kernel=rbf …
[CV] … C=100.0, gamma=1.0, kernel=rbf, total=  15.5s
[CV] C=100.0, gamma=1.0, kernel=rbf …
[CV] … C=100.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=100.0, gamma=1.0, kernel=rbf …
[CV] … C=100.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=100.0, gamma=1.0, kernel=rbf …
[CV] … C=100.0, gamma=1.0, kernel=rbf, total=  15.6s
[CV] C=100.0, gamma=3.0, kernel=rbf …
[CV] … C=100.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=100.0, gamma=3.0, kernel=rbf …
[CV] … C=100.0, gamma=3.0, kernel=rbf, total=  16.4s
[CV] C=100.0, gamma=3.0, kernel=rbf …
[CV] … C=100.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=100.0, gamma=3.0, kernel=rbf …
[CV] … C=100.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=100.0, gamma=3.0, kernel=rbf …
[CV] … C=100.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=300.0, gamma=0.01, kernel=rbf …
[CV] … C=300.0, gamma=0.01, kernel=rbf, total=  16.7s
[CV] C=300.0, gamma=0.01, kernel=rbf …
[CV] … C=300.0, gamma=0.01, kernel=rbf, total=  16.8s
[CV] C=300.0, gamma=0.01, kernel=rbf …
[CV] … C=300.0, gamma=0.01, kernel=rbf, total=  16.7s
[CV] C=300.0, gamma=0.01, kernel=rbf …
[CV] … C=300.0, gamma=0.01, kernel=rbf, total=  16.8s
[CV] C=300.0, gamma=0.01, kernel=rbf …
[CV] … C=300.0, gamma=0.01, kernel=rbf, total=  16.8s
[CV] C=300.0, gamma=0.03, kernel=rbf …
[CV] … C=300.0, gamma=0.03, kernel=rbf, total=  16.2s
[CV] C=300.0, gamma=0.03, kernel=rbf …
[CV] … C=300.0, gamma=0.03, kernel=rbf, total=  16.2s
[CV] C=300.0, gamma=0.03, kernel=rbf …
[CV] … C=300.0, gamma=0.03, kernel=rbf, total=  16.2s
[CV] C=300.0, gamma=0.03, kernel=rbf …
[CV] … C=300.0, gamma=0.03, kernel=rbf, total=  16.2s
[CV] C=300.0, gamma=0.03, kernel=rbf …
[CV] … C=300.0, gamma=0.03, kernel=rbf, total=  16.4s
[CV] C=300.0, gamma=0.1, kernel=rbf …
[CV] … C=300.0, gamma=0.1, kernel=rbf, total=  15.9s
```

```
[CV] C=300.0, gamma=0.1, kernel=rbf …
[CV] … C=300.0, gamma=0.1, kernel=rbf, total=  15.9s
[CV] C=300.0, gamma=0.1, kernel=rbf …
[CV] … C=300.0, gamma=0.1, kernel=rbf, total=  15.9s
[CV] C=300.0, gamma=0.1, kernel=rbf …
[CV] … C=300.0, gamma=0.1, kernel=rbf, total=  15.9s
[CV] C=300.0, gamma=0.1, kernel=rbf …
[CV] … C=300.0, gamma=0.1, kernel=rbf, total=  15.9s
[CV] C=300.0, gamma=0.3, kernel=rbf …
[CV] … C=300.0, gamma=0.3, kernel=rbf, total=  15.8s
[CV] C=300.0, gamma=0.3, kernel=rbf …
[CV] … C=300.0, gamma=0.3, kernel=rbf, total=  15.7s
[CV] C=300.0, gamma=0.3, kernel=rbf …
[CV] … C=300.0, gamma=0.3, kernel=rbf, total=  15.8s
[CV] C=300.0, gamma=0.3, kernel=rbf …
[CV] … C=300.0, gamma=0.3, kernel=rbf, total=  15.7s
[CV] C=300.0, gamma=0.3, kernel=rbf …
[CV] … C=300.0, gamma=0.3, kernel=rbf, total=  15.7s
[CV] C=300.0, gamma=1.0, kernel=rbf …
[CV] … C=300.0, gamma=1.0, kernel=rbf, total=  15.4s
[CV] C=300.0, gamma=1.0, kernel=rbf …
[CV] … C=300.0, gamma=1.0, kernel=rbf, total=  15.5s
[CV] C=300.0, gamma=1.0, kernel=rbf …
[CV] … C=300.0, gamma=1.0, kernel=rbf, total=  15.4s
[CV] C=300.0, gamma=1.0, kernel=rbf …
[CV] … C=300.0, gamma=1.0, kernel=rbf, total=  15.4s
[CV] C=300.0, gamma=1.0, kernel=rbf …
[CV] … C=300.0, gamma=1.0, kernel=rbf, total=  15.4s
[CV] C=300.0, gamma=3.0, kernel=rbf …
[CV] … C=300.0, gamma=3.0, kernel=rbf, total=  16.1s
[CV] C=300.0, gamma=3.0, kernel=rbf …
[CV] … C=300.0, gamma=3.0, kernel=rbf, total=  16.1s
[CV] C=300.0, gamma=3.0, kernel=rbf …
[CV] … C=300.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=300.0, gamma=3.0, kernel=rbf …
[CV] … C=300.0, gamma=3.0, kernel=rbf, total=  16.2s
[CV] C=300.0, gamma=3.0, kernel=rbf …
[CV] … C=300.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=1000.0, gamma=0.01, kernel=rbf …
[CV] … C=1000.0, gamma=0.01, kernel=rbf, total=  16.1s
[CV] C=1000.0, gamma=0.01, kernel=rbf …
[CV] … C=1000.0, gamma=0.01, kernel=rbf, total=  16.2s
[CV] C=1000.0, gamma=0.01, kernel=rbf …
[CV] … C=1000.0, gamma=0.01, kernel=rbf, total=  16.2s
[CV] C=1000.0, gamma=0.01, kernel=rbf …
[CV] … C=1000.0, gamma=0.01, kernel=rbf, total=  16.1s
[CV] C=1000.0, gamma=0.01, kernel=rbf …
[CV] … C=1000.0, gamma=0.01, kernel=rbf, total=  16.1s
```

```
[CV] C=1000.0, gamma=0.03, kernel=rbf …
[CV] … C=1000.0, gamma=0.03, kernel=rbf, total=  15.7s
[CV] C=1000.0, gamma=0.03, kernel=rbf …
[CV] … C=1000.0, gamma=0.03, kernel=rbf, total=  15.8s
[CV] C=1000.0, gamma=0.03, kernel=rbf …
[CV] … C=1000.0, gamma=0.03, kernel=rbf, total=  15.9s
[CV] C=1000.0, gamma=0.03, kernel=rbf …
[CV] … C=1000.0, gamma=0.03, kernel=rbf, total=  15.8s
[CV] C=1000.0, gamma=0.03, kernel=rbf …
[CV] … C=1000.0, gamma=0.03, kernel=rbf, total=  15.9s
[CV] C=1000.0, gamma=0.1, kernel=rbf …
[CV] … C=1000.0, gamma=0.1, kernel=rbf, total=  15.6s
[CV] C=1000.0, gamma=0.1, kernel=rbf …
[CV] … C=1000.0, gamma=0.1, kernel=rbf, total=  15.6s
[CV] C=1000.0, gamma=0.1, kernel=rbf …
[CV] … C=1000.0, gamma=0.1, kernel=rbf, total=  15.6s
[CV] C=1000.0, gamma=0.1, kernel=rbf …
[CV] … C=1000.0, gamma=0.1, kernel=rbf, total=  15.7s
[CV] C=1000.0, gamma=0.1, kernel=rbf …
[CV] … C=1000.0, gamma=0.1, kernel=rbf, total=  15.8s
[CV] C=1000.0, gamma=0.3, kernel=rbf …
[CV] … C=1000.0, gamma=0.3, kernel=rbf, total=  15.7s
[CV] C=1000.0, gamma=0.3, kernel=rbf …
[CV] … C=1000.0, gamma=0.3, kernel=rbf, total=  15.5s
[CV] C=1000.0, gamma=0.3, kernel=rbf …
[CV] … C=1000.0, gamma=0.3, kernel=rbf, total=  15.6s
[CV] C=1000.0, gamma=0.3, kernel=rbf …
[CV] … C=1000.0, gamma=0.3, kernel=rbf, total=  15.5s
[CV] C=1000.0, gamma=0.3, kernel=rbf …
[CV] … C=1000.0, gamma=0.3, kernel=rbf, total=  15.6s
[CV] C=1000.0, gamma=1.0, kernel=rbf …
[CV] … C=1000.0, gamma=1.0, kernel=rbf, total=  15.5s
[CV] C=1000.0, gamma=1.0, kernel=rbf …
[CV] … C=1000.0, gamma=1.0, kernel=rbf, total=  15.4s
[CV] C=1000.0, gamma=1.0, kernel=rbf …
[CV] … C=1000.0, gamma=1.0, kernel=rbf, total=  15.3s
[CV] C=1000.0, gamma=1.0, kernel=rbf …
[CV] … C=1000.0, gamma=1.0, kernel=rbf, total=  15.3s
[CV] C=1000.0, gamma=1.0, kernel=rbf …
[CV] … C=1000.0, gamma=1.0, kernel=rbf, total=  15.4s
[CV] C=1000.0, gamma=3.0, kernel=rbf …
[CV] … C=1000.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=1000.0, gamma=3.0, kernel=rbf …
[CV] … C=1000.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=1000.0, gamma=3.0, kernel=rbf …
[CV] … C=1000.0, gamma=3.0, kernel=rbf, total=  16.3s
[CV] C=1000.0, gamma=3.0, kernel=rbf …
[CV] … C=1000.0, gamma=3.0, kernel=rbf, total=  16.3s
```

```
[CV] C=1000.0, gamma=3.0, kernel=rbf …
[CV] … C=1000.0, gamma=3.0, kernel=rbf, total=  16.3s

[Parallel(n_jobs=1)]: Done 250 out of 250 | elapsed: 65.5min finished
```

[ ]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='scale', kernel='rbf',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [10.0, 30.0, 100.0, 300.0, 1000.0, 3000.0,
                                10000.0, 30000.0],
                          'kernel': ['linear']},
                         {'C': [1.0, 3.0, 10.0, 30.0, 100.0, 300.0, 1000.0],
                          'gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0],
                          'kernel': ['rbf']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=2)
```

The best model achieves the following score (evaluated using 5-fold cross validation):

[ ]:
```
negative_mse = grid_search.best_score_
rmse = np.sqrt(-negative_mse)
rmse
```

[ ]:
```
70363.84006944533
```

That's much worse than the `RandomForestRegressor`. Let's check the best hyperparameters found:

[ ]:
```
grid_search.best_params_
```

[ ]:
```
{'C': 30000.0, 'kernel': 'linear'}
```

The linear kernel seems better than the RBF kernel. Notice that the value of `C` is the maximum tested value. When this happens you definitely want to launch the grid search again with higher values for `C` (removing the smallest values), because it is likely that higher values of `C` will be better.

8.2 2.

Question: Try replacing `GridSearchCV` with `RandomizedSearchCV`.

Warning: the following cell may take close to 45 minutes to run, or more depending on your hardware.

[ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import expon, reciprocal

# see https://docs.scipy.org/doc/scipy/reference/stats.html
# for `expon()` and `reciprocal()` documentation and more probability␣
↪distribution functions.
```

```
# Note: gamma is ignored when kernel is "linear"
param_distribs = {
        'kernel': ['linear', 'rbf'],
        'C': reciprocal(20, 200000),
        'gamma': expon(scale=1.0),
    }

svm_reg = SVR()
rnd_search = RandomizedSearchCV(svm_reg, param_distributions=param_distribs,
                                n_iter=50, cv=5,␣
↪scoring='neg_mean_squared_error',
                                verbose=2, random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
[CV] C=629.782329591372, gamma=3.010121430917521, kernel=linear …

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV]  C=629.782329591372, gamma=3.010121430917521, kernel=linear, total=   9.8s
[CV] C=629.782329591372, gamma=3.010121430917521, kernel=linear …

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    9.8s remaining:    0.0s

[CV]  C=629.782329591372, gamma=3.010121430917521, kernel=linear, total=  10.1s
[CV] C=629.782329591372, gamma=3.010121430917521, kernel=linear …
[CV]  C=629.782329591372, gamma=3.010121430917521, kernel=linear, total=  10.1s
[CV] C=629.782329591372, gamma=3.010121430917521, kernel=linear …
[CV]  C=629.782329591372, gamma=3.010121430917521, kernel=linear, total=  10.0s
[CV] C=629.782329591372, gamma=3.010121430917521, kernel=linear …
[CV]  C=629.782329591372, gamma=3.010121430917521, kernel=linear, total=   9.9s
[CV] C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf …
[CV]  C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf, total=  19.1s
[CV] C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf …
[CV]  C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf, total=  20.0s
[CV] C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf …
[CV]  C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf, total=  19.7s
[CV] C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf …
[CV]  C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf, total=  19.9s
[CV] C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf …
[CV]  C=26290.206464300216, gamma=0.9084469696321253, kernel=rbf, total=  20.3s
[CV] C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf …
[CV]  C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf, total=  16.5s
[CV] C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf …
[CV]  C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf, total=  16.6s
[CV] C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf …
[CV]  C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf, total=  16.5s
[CV] C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf …
[CV]  C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf, total=  16.6s
```

```
[CV] C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf …
[CV]  C=84.14107900575871, gamma=0.059838768608680676, kernel=rbf, total=  16.8s
[CV] C=432.37884813148855, gamma=0.15416196746656105, kernel=linear ..
[CV]  C=432.37884813148855, gamma=0.15416196746656105, kernel=linear, total=
9.7s
[CV] C=432.37884813148855, gamma=0.15416196746656105, kernel=linear ..
[CV]  C=432.37884813148855, gamma=0.15416196746656105, kernel=linear, total=
9.8s
[CV] C=432.37884813148855, gamma=0.15416196746656105, kernel=linear ..
[CV]  C=432.37884813148855, gamma=0.15416196746656105, kernel=linear, total=
10.2s
[CV] C=432.37884813148855, gamma=0.15416196746656105, kernel=linear ..
[CV]  C=432.37884813148855, gamma=0.15416196746656105, kernel=linear, total=
10.1s
[CV] C=432.37884813148855, gamma=0.15416196746656105, kernel=linear ..
[CV]  C=432.37884813148855, gamma=0.15416196746656105, kernel=linear, total=
10.2s
[CV] C=24.17508294611391, gamma=3.503557475158312, kernel=rbf …
[CV]  C=24.17508294611391, gamma=3.503557475158312, kernel=rbf, total=  16.9s
[CV] C=24.17508294611391, gamma=3.503557475158312, kernel=rbf …
[CV]  C=24.17508294611391, gamma=3.503557475158312, kernel=rbf, total=  16.6s
[CV] C=24.17508294611391, gamma=3.503557475158312, kernel=rbf …
[CV]  C=24.17508294611391, gamma=3.503557475158312, kernel=rbf, total=  16.5s
[CV] C=24.17508294611391, gamma=3.503557475158312, kernel=rbf …
[CV]  C=24.17508294611391, gamma=3.503557475158312, kernel=rbf, total=  16.6s
[CV] C=24.17508294611391, gamma=3.503557475158312, kernel=rbf …
[CV]  C=24.17508294611391, gamma=3.503557475158312, kernel=rbf, total=  16.8s
[CV] C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf …
[CV]  C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf, total=
15.9s
[CV] C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf …
[CV]  C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf, total=
16.0s
[CV] C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf …
[CV]  C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf, total=
16.0s
[CV] C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf …
[CV]  C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf, total=
16.0s
[CV] C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf …
[CV]  C=113564.03940586245, gamma=0.0007790692366582295, kernel=rbf, total=
16.0s
[CV] C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf …
[CV]  C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf, total=  15.8s
[CV] C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf …
[CV]  C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf, total=  15.9s
[CV] C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf …
[CV]  C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf, total=  15.9s
```

```
[CV] C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf …
[CV]  C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf, total=  15.8s
[CV] C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf …
[CV]  C=108.30488238805073, gamma=0.3627537294604771, kernel=rbf, total=  15.9s
[CV] C=21.344953672647435, gamma=0.023332523598323388, kernel=linear .
[CV]  C=21.344953672647435, gamma=0.023332523598323388, kernel=linear, total=
9.7s
[CV] C=21.344953672647435, gamma=0.023332523598323388, kernel=linear .
[CV]  C=21.344953672647435, gamma=0.023332523598323388, kernel=linear, total=
9.8s
[CV] C=21.344953672647435, gamma=0.023332523598323388, kernel=linear .
[CV]  C=21.344953672647435, gamma=0.023332523598323388, kernel=linear, total=
9.8s
[CV] C=21.344953672647435, gamma=0.023332523598323388, kernel=linear .
[CV]  C=21.344953672647435, gamma=0.023332523598323388, kernel=linear, total=
9.7s
[CV] C=21.344953672647435, gamma=0.023332523598323388, kernel=linear .
[CV]  C=21.344953672647435, gamma=0.023332523598323388, kernel=linear, total=
9.7s
[CV] C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf …
[CV]  C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf, total=  15.6s
[CV] C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf …
[CV]  C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf, total=  15.6s
[CV] C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf …
[CV]  C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf, total=  15.6s
[CV] C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf …
[CV]  C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf, total=  15.5s
[CV] C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf …
[CV]  C=5603.270317432516, gamma=0.15023452872733867, kernel=rbf, total=  15.5s
[CV] C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf …
[CV]  C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf, total=  38.2s
[CV] C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf …
[CV]  C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf, total=  41.3s
[CV] C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf …
[CV]  C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf, total=  47.2s
[CV] C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf …
[CV]  C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf, total=  38.8s
[CV] C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf …
[CV]  C=157055.10989448498, gamma=0.26497040005002437, kernel=rbf, total=  43.2s
[CV] C=27652.464358739708, gamma=0.2227358621286903, kernel=linear …
[CV]  C=27652.464358739708, gamma=0.2227358621286903, kernel=linear, total=
22.4s
[CV] C=27652.464358739708, gamma=0.2227358621286903, kernel=linear …
[CV]  C=27652.464358739708, gamma=0.2227358621286903, kernel=linear, total=
23.0s
[CV] C=27652.464358739708, gamma=0.2227358621286903, kernel=linear …
[CV]  C=27652.464358739708, gamma=0.2227358621286903, kernel=linear, total=
24.5s
```

```
[CV] C=27652.464358739708, gamma=0.2227358621286903, kernel=linear …
[CV]  C=27652.464358739708, gamma=0.2227358621286903, kernel=linear, total=
22.3s
[CV] C=27652.464358739708, gamma=0.2227358621286903, kernel=linear …
[CV]  C=27652.464358739708, gamma=0.2227358621286903, kernel=linear, total=
19.5s
[CV] C=171377.39570378003, gamma=0.628789100540856, kernel=linear …
[CV]  C=171377.39570378003, gamma=0.628789100540856, kernel=linear, total=
1.6min
[CV] C=171377.39570378003, gamma=0.628789100540856, kernel=linear …
[CV]  C=171377.39570378003, gamma=0.628789100540856, kernel=linear, total=
1.2min
[CV] C=171377.39570378003, gamma=0.628789100540856, kernel=linear …
[CV]  C=171377.39570378003, gamma=0.628789100540856, kernel=linear, total=
1.6min
[CV] C=171377.39570378003, gamma=0.628789100540856, kernel=linear …
[CV]  C=171377.39570378003, gamma=0.628789100540856, kernel=linear, total=
1.4min
[CV] C=171377.39570378003, gamma=0.628789100540856, kernel=linear …
[CV]  C=171377.39570378003, gamma=0.628789100540856, kernel=linear, total=
1.1min
[CV] C=5385.293820172355, gamma=0.18696125197741642, kernel=linear …
[CV]  C=5385.293820172355, gamma=0.18696125197741642, kernel=linear, total=
12.1s
[CV] C=5385.293820172355, gamma=0.18696125197741642, kernel=linear …
[CV]  C=5385.293820172355, gamma=0.18696125197741642, kernel=linear, total=
12.2s
[CV] C=5385.293820172355, gamma=0.18696125197741642, kernel=linear …
[CV]  C=5385.293820172355, gamma=0.18696125197741642, kernel=linear, total=
12.5s
[CV] C=5385.293820172355, gamma=0.18696125197741642, kernel=linear …
[CV]  C=5385.293820172355, gamma=0.18696125197741642, kernel=linear, total=
12.0s
[CV] C=5385.293820172355, gamma=0.18696125197741642, kernel=linear …
[CV]  C=5385.293820172355, gamma=0.18696125197741642, kernel=linear, total=
12.2s
[CV] C=22.59903216621323, gamma=2.850796878935603, kernel=rbf …
[CV]  C=22.59903216621323, gamma=2.850796878935603, kernel=rbf, total=  16.0s
[CV] C=22.59903216621323, gamma=2.850796878935603, kernel=rbf …
[CV]  C=22.59903216621323, gamma=2.850796878935603, kernel=rbf, total=  16.1s
[CV] C=22.59903216621323, gamma=2.850796878935603, kernel=rbf …
[CV]  C=22.59903216621323, gamma=2.850796878935603, kernel=rbf, total=  16.0s
[CV] C=22.59903216621323, gamma=2.850796878935603, kernel=rbf …
[CV]  C=22.59903216621323, gamma=2.850796878935603, kernel=rbf, total=  16.0s
[CV] C=22.59903216621323, gamma=2.850796878935603, kernel=rbf …
[CV]  C=22.59903216621323, gamma=2.850796878935603, kernel=rbf, total=  16.1s
[CV] C=34246.75194632794, gamma=0.3632878599687583, kernel=linear …
[CV]  C=34246.75194632794, gamma=0.3632878599687583, kernel=linear, total=
```

```
26.4s
[CV] C=34246.75194632794, gamma=0.3632878599687583, kernel=linear …
[CV]  C=34246.75194632794, gamma=0.3632878599687583, kernel=linear, total=
26.0s
[CV] C=34246.75194632794, gamma=0.3632878599687583, kernel=linear …
[CV]  C=34246.75194632794, gamma=0.3632878599687583, kernel=linear, total=
27.5s
[CV] C=34246.75194632794, gamma=0.3632878599687583, kernel=linear …
[CV]  C=34246.75194632794, gamma=0.3632878599687583, kernel=linear, total=
26.4s
[CV] C=34246.75194632794, gamma=0.3632878599687583, kernel=linear …
[CV]  C=34246.75194632794, gamma=0.3632878599687583, kernel=linear, total=
23.4s
[CV] C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf …
[CV]  C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf, total=  15.9s
[CV] C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf …
[CV]  C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf, total=  15.8s
[CV] C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf …
[CV]  C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf, total=  16.1s
[CV] C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf …
[CV]  C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf, total=  15.9s
[CV] C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf …
[CV]  C=167.7278956080511, gamma=0.2757870542258224, kernel=rbf, total=  15.9s
[CV] C=61.54360542501371, gamma=0.6835472281341501, kernel=linear …
[CV]  C=61.54360542501371, gamma=0.6835472281341501, kernel=linear, total=
9.8s
[CV] C=61.54360542501371, gamma=0.6835472281341501, kernel=linear …
[CV]  C=61.54360542501371, gamma=0.6835472281341501, kernel=linear, total=
9.7s
[CV] C=61.54360542501371, gamma=0.6835472281341501, kernel=linear …
[CV]  C=61.54360542501371, gamma=0.6835472281341501, kernel=linear, total=
9.9s
[CV] C=61.54360542501371, gamma=0.6835472281341501, kernel=linear …
[CV]  C=61.54360542501371, gamma=0.6835472281341501, kernel=linear, total=
9.8s
[CV] C=61.54360542501371, gamma=0.6835472281341501, kernel=linear …
[CV]  C=61.54360542501371, gamma=0.6835472281341501, kernel=linear, total=
9.5s
[CV] C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf …
[CV]  C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf, total=  15.9s
[CV] C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf …
[CV]  C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf, total=  15.9s
[CV] C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf …
[CV]  C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf, total=  15.8s
[CV] C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf …
[CV]  C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf, total=  15.9s
[CV] C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf …
[CV]  C=98.73897389920914, gamma=0.4960365360493639, kernel=rbf, total=  15.7s
```

```
[CV] C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf …
[CV]  C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf, total=  15.7s
[CV] C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf …
[CV]  C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf, total=  15.7s
[CV] C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf …
[CV]  C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf, total=  15.8s
[CV] C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf …
[CV]  C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf, total=  15.8s
[CV] C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf …
[CV]  C=8935.505635947808, gamma=0.37354658165762367, kernel=rbf, total=  15.9s
[CV] C=135.76775824842434, gamma=0.838636245624803, kernel=linear …
[CV]  C=135.76775824842434, gamma=0.838636245624803, kernel=linear, total=
9.7s
[CV] C=135.76775824842434, gamma=0.838636245624803, kernel=linear …
[CV]  C=135.76775824842434, gamma=0.838636245624803, kernel=linear, total=
9.8s
[CV] C=135.76775824842434, gamma=0.838636245624803, kernel=linear …
[CV]  C=135.76775824842434, gamma=0.838636245624803, kernel=linear, total=
9.9s
[CV] C=135.76775824842434, gamma=0.838636245624803, kernel=linear …
[CV]  C=135.76775824842434, gamma=0.838636245624803, kernel=linear, total=
10.1s
[CV] C=135.76775824842434, gamma=0.838636245624803, kernel=linear …
[CV]  C=135.76775824842434, gamma=0.838636245624803, kernel=linear, total=
9.7s
[CV] C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf …
[CV]  C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf, total= 3.7min
[CV] C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf …
[CV]  C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf, total= 2.8min
[CV] C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf …
[CV]  C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf, total= 2.6min
[CV] C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf …
[CV]  C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf, total= 3.3min
[CV] C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf …
[CV]  C=151136.20282548846, gamma=1.4922453771381408, kernel=rbf, total= 3.2min
[CV] C=761.4316758498783, gamma=2.6126336514161914, kernel=linear …
[CV]  C=761.4316758498783, gamma=2.6126336514161914, kernel=linear, total=
9.8s
[CV] C=761.4316758498783, gamma=2.6126336514161914, kernel=linear …
[CV]  C=761.4316758498783, gamma=2.6126336514161914, kernel=linear, total=
9.9s
[CV] C=761.4316758498783, gamma=2.6126336514161914, kernel=linear …
[CV]  C=761.4316758498783, gamma=2.6126336514161914, kernel=linear, total=
10.1s
[CV] C=761.4316758498783, gamma=2.6126336514161914, kernel=linear …
[CV]  C=761.4316758498783, gamma=2.6126336514161914, kernel=linear, total=
10.0s
[CV] C=761.4316758498783, gamma=2.6126336514161914, kernel=linear …
```

```
[CV]  C=761.4316758498783, gamma=2.6126336514161914, kernel=linear, total=
10.0s
[CV] C=97392.81883041795, gamma=0.09265545895311562, kernel=linear …
[CV]  C=97392.81883041795, gamma=0.09265545895311562, kernel=linear, total=
53.6s
[CV] C=97392.81883041795, gamma=0.09265545895311562, kernel=linear …
[CV]  C=97392.81883041795, gamma=0.09265545895311562, kernel=linear, total=
51.9s
[CV] C=97392.81883041795, gamma=0.09265545895311562, kernel=linear …
[CV]  C=97392.81883041795, gamma=0.09265545895311562, kernel=linear, total=
1.4min
[CV] C=97392.81883041795, gamma=0.09265545895311562, kernel=linear …
[CV]  C=97392.81883041795, gamma=0.09265545895311562, kernel=linear, total=
54.9s
[CV] C=97392.81883041795, gamma=0.09265545895311562, kernel=linear …
[CV]  C=97392.81883041795, gamma=0.09265545895311562, kernel=linear, total=
44.5s
[CV] C=2423.0759984939164, gamma=3.248614270240346, kernel=linear …
[CV]  C=2423.0759984939164, gamma=3.248614270240346, kernel=linear, total=
11.4s
[CV] C=2423.0759984939164, gamma=3.248614270240346, kernel=linear …
[CV]  C=2423.0759984939164, gamma=3.248614270240346, kernel=linear, total=
11.3s
[CV] C=2423.0759984939164, gamma=3.248614270240346, kernel=linear …
[CV]  C=2423.0759984939164, gamma=3.248614270240346, kernel=linear, total=
10.9s
[CV] C=2423.0759984939164, gamma=3.248614270240346, kernel=linear …
[CV]  C=2423.0759984939164, gamma=3.248614270240346, kernel=linear, total=
11.3s
[CV] C=2423.0759984939164, gamma=3.248614270240346, kernel=linear …
[CV]  C=2423.0759984939164, gamma=3.248614270240346, kernel=linear, total=
10.5s
[CV] C=717.3632997255095, gamma=0.3165604432088257, kernel=linear …
[CV]  C=717.3632997255095, gamma=0.3165604432088257, kernel=linear, total=
10.0s
[CV] C=717.3632997255095, gamma=0.3165604432088257, kernel=linear …
[CV]  C=717.3632997255095, gamma=0.3165604432088257, kernel=linear, total=
9.9s
[CV] C=717.3632997255095, gamma=0.3165604432088257, kernel=linear …
[CV]  C=717.3632997255095, gamma=0.3165604432088257, kernel=linear, total=
10.2s
[CV] C=717.3632997255095, gamma=0.3165604432088257, kernel=linear …
[CV]  C=717.3632997255095, gamma=0.3165604432088257, kernel=linear, total=
10.2s
[CV] C=717.3632997255095, gamma=0.3165604432088257, kernel=linear …
[CV]  C=717.3632997255095, gamma=0.3165604432088257, kernel=linear, total=
9.8s
[CV] C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf …
```

```
[CV]  C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf, total=  16.8s
[CV] C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf …
[CV]  C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf, total=  16.9s
[CV] C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf …
[CV]  C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf, total=  16.8s
[CV] C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf …
[CV]  C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf, total=  16.8s
[CV] C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf …
[CV]  C=4446.667521184072, gamma=3.3597284456608496, kernel=rbf, total=  16.9s
[CV] C=2963.564121207815, gamma=0.15189814782062885, kernel=linear …
[CV]  C=2963.564121207815, gamma=0.15189814782062885, kernel=linear, total=
11.1s
[CV] C=2963.564121207815, gamma=0.15189814782062885, kernel=linear …
[CV]  C=2963.564121207815, gamma=0.15189814782062885, kernel=linear, total=
11.7s
[CV] C=2963.564121207815, gamma=0.15189814782062885, kernel=linear …
[CV]  C=2963.564121207815, gamma=0.15189814782062885, kernel=linear, total=
11.9s
[CV] C=2963.564121207815, gamma=0.15189814782062885, kernel=linear …
[CV]  C=2963.564121207815, gamma=0.15189814782062885, kernel=linear, total=
11.3s
[CV] C=2963.564121207815, gamma=0.15189814782062885, kernel=linear …
[CV]  C=2963.564121207815, gamma=0.15189814782062885, kernel=linear, total=
11.0s
[CV] C=91.64267381686706, gamma=0.01575994483585621, kernel=linear …
[CV]  C=91.64267381686706, gamma=0.01575994483585621, kernel=linear, total=
9.6s
[CV] C=91.64267381686706, gamma=0.01575994483585621, kernel=linear …
[CV]  C=91.64267381686706, gamma=0.01575994483585621, kernel=linear, total=
9.8s
[CV] C=91.64267381686706, gamma=0.01575994483585621, kernel=linear …
[CV]  C=91.64267381686706, gamma=0.01575994483585621, kernel=linear, total=
9.8s
[CV] C=91.64267381686706, gamma=0.01575994483585621, kernel=linear …
[CV]  C=91.64267381686706, gamma=0.01575994483585621, kernel=linear, total=
9.8s
[CV] C=91.64267381686706, gamma=0.01575994483585621, kernel=linear …
[CV]  C=91.64267381686706, gamma=0.01575994483585621, kernel=linear, total=
9.5s
[CV] C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf …
[CV]  C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf, total=  16.8s
[CV] C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf …
[CV]  C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf, total=  16.6s
[CV] C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf …
[CV]  C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf, total=  16.6s
[CV] C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf …
[CV]  C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf, total=  16.6s
[CV] C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf …
```

```
[CV]  C=24547.601975705915, gamma=0.22153944050588595, kernel=rbf, total=  16.6s
[CV] C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf …
[CV]  C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf, total=  16.4s
[CV] C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf …
[CV]  C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf, total=  16.3s
[CV] C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf …
[CV]  C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf, total=  16.4s
[CV] C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf …
[CV]  C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf, total=  16.3s
[CV] C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf …
[CV]  C=22.76927941060928, gamma=0.22169760231351215, kernel=rbf, total=  16.3s
[CV] C=16483.850529752886, gamma=1.4752145260435134, kernel=linear …
[CV]  C=16483.850529752886, gamma=1.4752145260435134, kernel=linear, total=
16.6s
[CV] C=16483.850529752886, gamma=1.4752145260435134, kernel=linear …
[CV]  C=16483.850529752886, gamma=1.4752145260435134, kernel=linear, total=
17.9s
[CV] C=16483.850529752886, gamma=1.4752145260435134, kernel=linear …
[CV]  C=16483.850529752886, gamma=1.4752145260435134, kernel=linear, total=
17.9s
[CV] C=16483.850529752886, gamma=1.4752145260435134, kernel=linear …
[CV]  C=16483.850529752886, gamma=1.4752145260435134, kernel=linear, total=
18.4s
[CV] C=16483.850529752886, gamma=1.4752145260435134, kernel=linear …
[CV]  C=16483.850529752886, gamma=1.4752145260435134, kernel=linear, total=
15.7s
[CV] C=101445.66881340064, gamma=1.052904084582266, kernel=rbf …
[CV]  C=101445.66881340064, gamma=1.052904084582266, kernel=rbf, total= 1.3min
[CV] C=101445.66881340064, gamma=1.052904084582266, kernel=rbf …
[CV]  C=101445.66881340064, gamma=1.052904084582266, kernel=rbf, total= 1.2min
[CV] C=101445.66881340064, gamma=1.052904084582266, kernel=rbf …
[CV]  C=101445.66881340064, gamma=1.052904084582266, kernel=rbf, total= 1.6min
[CV] C=101445.66881340064, gamma=1.052904084582266, kernel=rbf …
[CV]  C=101445.66881340064, gamma=1.052904084582266, kernel=rbf, total= 1.6min
[CV] C=101445.66881340064, gamma=1.052904084582266, kernel=rbf …
[CV]  C=101445.66881340064, gamma=1.052904084582266, kernel=rbf, total= 1.3min
[CV] C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf …
[CV]  C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf, total=  32.0s
[CV] C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf …
[CV]  C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf, total=  32.7s
[CV] C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf …
[CV]  C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf, total=  31.7s
[CV] C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf …
[CV]  C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf, total=  36.4s
[CV] C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf …
[CV]  C=56681.80859029545, gamma=0.9763011917123741, kernel=rbf, total=  33.8s
[CV] C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf …
[CV]  C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf, total=  15.8s
```

```
[CV] C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf …
[CV]  C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf, total=  15.8s
[CV] C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf …
[CV]  C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf, total=  15.9s
[CV] C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf …
[CV]  C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf, total=  15.9s
[CV] C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf …
[CV]  C=48.15822390928914, gamma=0.4633351167983427, kernel=rbf, total=  15.8s
[CV] C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf …
[CV]  C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf, total=  15.5s
[CV] C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf …
[CV]  C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf, total=  15.4s
[CV] C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf …
[CV]  C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf, total=  15.5s
[CV] C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf …
[CV]  C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf, total=  15.5s
[CV] C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf …
[CV]  C=399.7268155705774, gamma=1.3078757839577408, kernel=rbf, total=  15.4s
[CV] C=251.14073886281363, gamma=0.8238105204914145, kernel=linear …
[CV]  C=251.14073886281363, gamma=0.8238105204914145, kernel=linear, total=
9.6s
[CV] C=251.14073886281363, gamma=0.8238105204914145, kernel=linear …
[CV]  C=251.14073886281363, gamma=0.8238105204914145, kernel=linear, total=
9.7s
[CV] C=251.14073886281363, gamma=0.8238105204914145, kernel=linear …
[CV]  C=251.14073886281363, gamma=0.8238105204914145, kernel=linear, total=
9.9s
[CV] C=251.14073886281363, gamma=0.8238105204914145, kernel=linear …
[CV]  C=251.14073886281363, gamma=0.8238105204914145, kernel=linear, total=
10.0s
[CV] C=251.14073886281363, gamma=0.8238105204914145, kernel=linear …
[CV]  C=251.14073886281363, gamma=0.8238105204914145, kernel=linear, total=
9.8s
[CV] C=60.17373642891687, gamma=1.2491263443165994, kernel=linear …
[CV]  C=60.17373642891687, gamma=1.2491263443165994, kernel=linear, total=
9.7s
[CV] C=60.17373642891687, gamma=1.2491263443165994, kernel=linear …
[CV]  C=60.17373642891687, gamma=1.2491263443165994, kernel=linear, total=
9.7s
[CV] C=60.17373642891687, gamma=1.2491263443165994, kernel=linear …
[CV]  C=60.17373642891687, gamma=1.2491263443165994, kernel=linear, total=
9.8s
[CV] C=60.17373642891687, gamma=1.2491263443165994, kernel=linear …
[CV]  C=60.17373642891687, gamma=1.2491263443165994, kernel=linear, total=
9.6s
[CV] C=60.17373642891687, gamma=1.2491263443165994, kernel=linear …
[CV]  C=60.17373642891687, gamma=1.2491263443165994, kernel=linear, total=
9.5s
```

```
[CV] C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf …
[CV]  C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf, total=  16.1s
[CV] C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf …
[CV]  C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf, total=  16.1s
[CV] C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf …
[CV]  C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf, total=  16.2s
[CV] C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf …
[CV]  C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf, total=  16.1s
[CV] C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf …
[CV]  C=15415.161544891856, gamma=0.2691677514619319, kernel=rbf, total=  16.1s
[CV] C=1888.9148509967113, gamma=0.739678838777267, kernel=linear …
[CV]  C=1888.9148509967113, gamma=0.739678838777267, kernel=linear, total=
10.6s
[CV] C=1888.9148509967113, gamma=0.739678838777267, kernel=linear …
[CV]  C=1888.9148509967113, gamma=0.739678838777267, kernel=linear, total=
10.7s
[CV] C=1888.9148509967113, gamma=0.739678838777267, kernel=linear …
[CV]  C=1888.9148509967113, gamma=0.739678838777267, kernel=linear, total=
10.7s
[CV] C=1888.9148509967113, gamma=0.739678838777267, kernel=linear …
[CV]  C=1888.9148509967113, gamma=0.739678838777267, kernel=linear, total=
10.6s
[CV] C=1888.9148509967113, gamma=0.739678838777267, kernel=linear …
[CV]  C=1888.9148509967113, gamma=0.739678838777267, kernel=linear, total=
10.5s
[CV] C=55.53838911232773, gamma=0.578634378499143, kernel=linear …
[CV]  C=55.53838911232773, gamma=0.578634378499143, kernel=linear, total=   9.7s
[CV] C=55.53838911232773, gamma=0.578634378499143, kernel=linear …
[CV]  C=55.53838911232773, gamma=0.578634378499143, kernel=linear, total=   9.7s
[CV] C=55.53838911232773, gamma=0.578634378499143, kernel=linear …
[CV]  C=55.53838911232773, gamma=0.578634378499143, kernel=linear, total=   9.7s
[CV] C=55.53838911232773, gamma=0.578634378499143, kernel=linear …
[CV]  C=55.53838911232773, gamma=0.578634378499143, kernel=linear, total=   9.6s
[CV] C=55.53838911232773, gamma=0.578634378499143, kernel=linear …
[CV]  C=55.53838911232773, gamma=0.578634378499143, kernel=linear, total=   9.5s
[CV] C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf …
[CV]  C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf, total=  15.7s
[CV] C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf …
[CV]  C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf, total=  15.6s
[CV] C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf …
[CV]  C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf, total=  15.5s
[CV] C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf …
[CV]  C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf, total=  15.5s
[CV] C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf …
[CV]  C=26.714480823948186, gamma=1.0117295509275495, kernel=rbf, total=  15.6s
[CV] C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear …
[CV]  C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear, total=
11.9s
```

```
[CV] C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear …
[CV]  C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear, total=
11.5s
[CV] C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear …
[CV]  C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear, total=
11.8s
[CV] C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear …
[CV]  C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear, total=
11.6s
[CV] C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear …
[CV]  C=3582.0552780489566, gamma=1.1891370222133257, kernel=linear, total=
11.1s
[CV] C=198.7004781812736, gamma=0.5282819748826726, kernel=linear …
[CV]  C=198.7004781812736, gamma=0.5282819748826726, kernel=linear, total=
9.5s
[CV] C=198.7004781812736, gamma=0.5282819748826726, kernel=linear …
[CV]  C=198.7004781812736, gamma=0.5282819748826726, kernel=linear, total=
9.6s
[CV] C=198.7004781812736, gamma=0.5282819748826726, kernel=linear …
[CV]  C=198.7004781812736, gamma=0.5282819748826726, kernel=linear, total=
9.8s
[CV] C=198.7004781812736, gamma=0.5282819748826726, kernel=linear …
[CV]  C=198.7004781812736, gamma=0.5282819748826726, kernel=linear, total=
9.8s
[CV] C=198.7004781812736, gamma=0.5282819748826726, kernel=linear …
[CV]  C=198.7004781812736, gamma=0.5282819748826726, kernel=linear, total=
9.6s
[CV] C=129.8000604143307, gamma=2.8621383676481322, kernel=linear …
[CV]  C=129.8000604143307, gamma=2.8621383676481322, kernel=linear, total=
9.7s
[CV] C=129.8000604143307, gamma=2.8621383676481322, kernel=linear …
[CV]  C=129.8000604143307, gamma=2.8621383676481322, kernel=linear, total=
9.5s
[CV] C=129.8000604143307, gamma=2.8621383676481322, kernel=linear …
[CV]  C=129.8000604143307, gamma=2.8621383676481322, kernel=linear, total=
9.8s
[CV] C=129.8000604143307, gamma=2.8621383676481322, kernel=linear …
[CV]  C=129.8000604143307, gamma=2.8621383676481322, kernel=linear, total=
9.7s
[CV] C=129.8000604143307, gamma=2.8621383676481322, kernel=linear …
[CV]  C=129.8000604143307, gamma=2.8621383676481322, kernel=linear, total=
9.5s
[CV] C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf …
[CV]  C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf, total=  15.8s
[CV] C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf …
[CV]  C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf, total=  15.8s
[CV] C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf …
[CV]  C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf, total=  15.9s
```

```
[CV] C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf …
[CV]  C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf, total=  15.8s
[CV] C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf …
[CV]  C=288.4269299593897, gamma=0.17580835850006285, kernel=rbf, total=  15.8s
[CV] C=6287.039489427172, gamma=0.3504567255332862, kernel=linear …
[CV]  C=6287.039489427172, gamma=0.3504567255332862, kernel=linear, total=
12.7s
[CV] C=6287.039489427172, gamma=0.3504567255332862, kernel=linear …
[CV]  C=6287.039489427172, gamma=0.3504567255332862, kernel=linear, total=
12.6s
[CV] C=6287.039489427172, gamma=0.3504567255332862, kernel=linear …
[CV]  C=6287.039489427172, gamma=0.3504567255332862, kernel=linear, total=
13.1s
[CV] C=6287.039489427172, gamma=0.3504567255332862, kernel=linear …
[CV]  C=6287.039489427172, gamma=0.3504567255332862, kernel=linear, total=
12.8s
[CV] C=6287.039489427172, gamma=0.3504567255332862, kernel=linear …
[CV]  C=6287.039489427172, gamma=0.3504567255332862, kernel=linear, total=
12.0s
[CV] C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf …
[CV]  C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf, total=  56.1s
[CV] C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf …
[CV]  C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf, total= 1.1min
[CV] C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf …
[CV]  C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf, total= 1.0min
[CV] C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf …
[CV]  C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf, total= 1.0min
[CV] C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf …
[CV]  C=61217.04421344494, gamma=1.6279689407405564, kernel=rbf, total=  57.2s
[CV] C=926.9787684096649, gamma=2.147979593060577, kernel=rbf …
[CV]  C=926.9787684096649, gamma=2.147979593060577, kernel=rbf, total=  15.7s
[CV] C=926.9787684096649, gamma=2.147979593060577, kernel=rbf …
[CV]  C=926.9787684096649, gamma=2.147979593060577, kernel=rbf, total=  15.6s
[CV] C=926.9787684096649, gamma=2.147979593060577, kernel=rbf …
[CV]  C=926.9787684096649, gamma=2.147979593060577, kernel=rbf, total=  15.6s
[CV] C=926.9787684096649, gamma=2.147979593060577, kernel=rbf …
[CV]  C=926.9787684096649, gamma=2.147979593060577, kernel=rbf, total=  15.6s
[CV] C=926.9787684096649, gamma=2.147979593060577, kernel=rbf …
[CV]  C=926.9787684096649, gamma=2.147979593060577, kernel=rbf, total=  15.6s
[CV] C=33946.157064934, gamma=2.2642426492862313, kernel=linear …
[CV]  C=33946.157064934, gamma=2.2642426492862313, kernel=linear, total=  26.2s
[CV] C=33946.157064934, gamma=2.2642426492862313, kernel=linear …
[CV]  C=33946.157064934, gamma=2.2642426492862313, kernel=linear, total=  25.8s
[CV] C=33946.157064934, gamma=2.2642426492862313, kernel=linear …
[CV]  C=33946.157064934, gamma=2.2642426492862313, kernel=linear, total=  23.6s
[CV] C=33946.157064934, gamma=2.2642426492862313, kernel=linear …
[CV]  C=33946.157064934, gamma=2.2642426492862313, kernel=linear, total=  27.2s
[CV] C=33946.157064934, gamma=2.2642426492862313, kernel=linear …
```

```
[CV]  C=33946.157064934, gamma=2.2642426492862313, kernel=linear, total=  24.2s
[CV] C=84789.82947739525, gamma=0.3176359085304841, kernel=linear …
[CV]  C=84789.82947739525, gamma=0.3176359085304841, kernel=linear, total=
1.1min
[CV] C=84789.82947739525, gamma=0.3176359085304841, kernel=linear …
[CV]  C=84789.82947739525, gamma=0.3176359085304841, kernel=linear, total=
49.3s
[CV] C=84789.82947739525, gamma=0.3176359085304841, kernel=linear …
[CV]  C=84789.82947739525, gamma=0.3176359085304841, kernel=linear, total=
1.2min
[CV] C=84789.82947739525, gamma=0.3176359085304841, kernel=linear …
[CV]  C=84789.82947739525, gamma=0.3176359085304841, kernel=linear, total=
54.1s
[CV] C=84789.82947739525, gamma=0.3176359085304841, kernel=linear …
[CV]  C=84789.82947739525, gamma=0.3176359085304841, kernel=linear, total=
40.9s

[Parallel(n_jobs=1)]: Done 250 out of 250 | elapsed: 99.8min finished
```

[ ]:
```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                                 epsilon=0.1, gamma='scale', kernel='rbf',
                                 max_iter=-1, shrinking=True, tol=0.001,
                                 verbose=False),
                   iid='deprecated', n_iter=50, n_jobs=None,
                   param_distributions={'C':
 <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f3f22192890>,
                                        'gamma':
 <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f3f15ecfdd0>,
                                        'kernel': ['linear', 'rbf']},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=2)
```

The best model achieves the following score (evaluated using 5-fold cross validation):

[ ]:
```
negative_mse = rnd_search.best_score_
rmse = np.sqrt(-negative_mse)
rmse
```

[ ]:
```
54767.960710084146
```

Now this is much closer to the performance of the `RandomForestRegressor` (but not quite there yet). Let's check the best hyperparameters found:

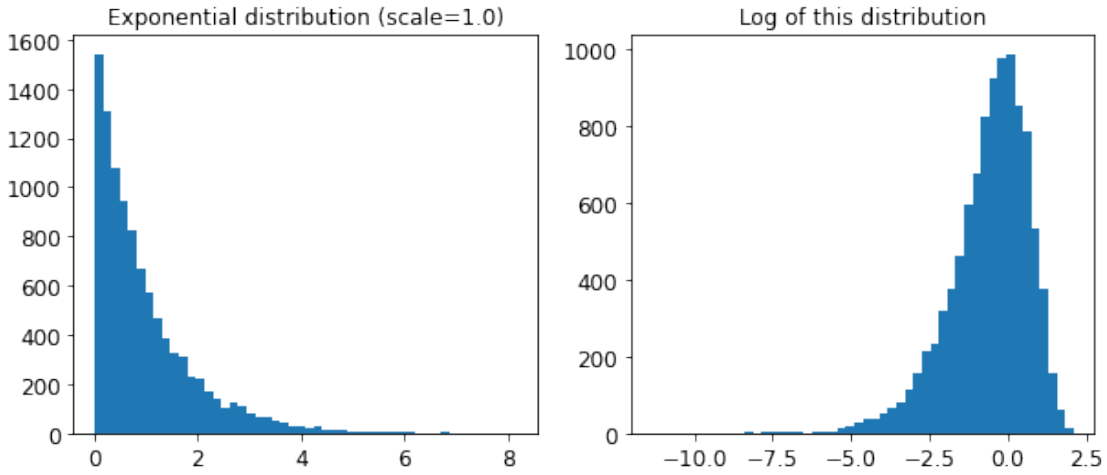
[ ]:
```
rnd_search.best_params_
```

[ ]:
```
{'C': 157055.10989448498, 'gamma': 0.26497040005002437, 'kernel': 'rbf'}
```

This time the search found a good set of hyperparameters for the RBF kernel. Randomized search tends to find better hyperparameters than grid search in the same amount of time.

Let's look at the exponential distribution we used, with `scale=1.0`. Note that some samples are much larger or smaller than 1.0, but when you look at the log of the distribution, you can see that most values are actually concentrated roughly in the range of exp(-2) to exp(+2), which is about 0.1 to 7.4.

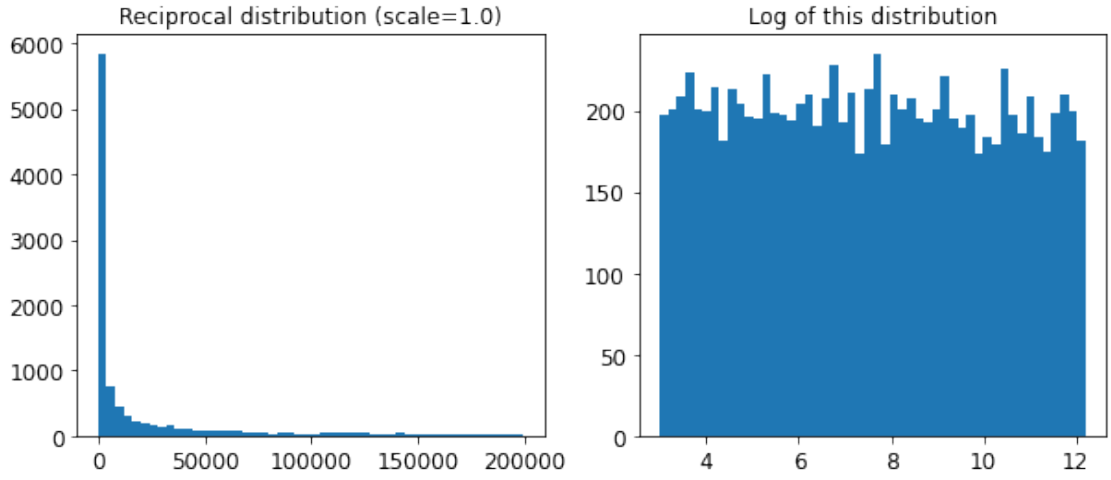
[ ]:
```
expon_distrib = expon(scale=1.)
samples = expon_distrib.rvs(10000, random_state=42)
plt.figure(figsize=(10, 4))
plt.subplot(121)
plt.title("Exponential distribution (scale=1.0)")
plt.hist(samples, bins=50)
plt.subplot(122)
plt.title("Log of this distribution")
plt.hist(np.log(samples), bins=50)
plt.show()
```



The distribution we used for `C` looks quite different: the scale of the samples is picked from a uniform distribution within a given range, which is why the right graph, which represents the log of the samples, looks roughly constant. This distribution is useful when you don't have a clue of what the target scale is:

[ ]:
```
reciprocal_distrib = reciprocal(20, 200000)
samples = reciprocal_distrib.rvs(10000, random_state=42)
plt.figure(figsize=(10, 4))
plt.subplot(121)
plt.title("Reciprocal distribution (scale=1.0)")
plt.hist(samples, bins=50)
plt.subplot(122)
plt.title("Log of this distribution")
```

```
plt.hist(np.log(samples), bins=50)
plt.show()
```



The reciprocal distribution is useful when you have no idea what the scale of the hyperparameter should be (indeed, as you can see on the figure on the right, all scales are equally likely, within the given range), whereas the exponential distribution is best when you know (more or less) what the scale of the hyperparameter should be.

8.3 3.

Question: Try adding a transformer in the preparation pipeline to select only the most important attributes.

[ ]:
```
from sklearn.base import BaseEstimator, TransformerMixin

def indices_of_top_k(arr, k):
    return np.sort(np.argpartition(np.array(arr), -k)[-k:])

class TopFeatureSelector(BaseEstimator, TransformerMixin):
    def __init__(self, feature_importances, k):
        self.feature_importances = feature_importances
        self.k = k
    def fit(self, X, y=None):
        self.feature_indices_ = indices_of_top_k(self.feature_importances, self.
 ↪k)
        return self
    def transform(self, X):
        return X[:, self.feature_indices_]
```

Note: this feature selector assumes that you have already computed the feature importances somehow (for example using a `RandomForestRegressor`). You may be tempted to compute them directly

in the `TopFeatureSelector`'s `fit()` method, however this would likely slow down grid/randomized search since the feature importances would have to be computed for every hyperparameter combination (unless you implement some sort of cache).

Let's define the number of top features we want to keep:

[ ]:
```
k = 5
```

Now let's look for the indices of the top k features:

[ ]:
```
top_k_feature_indices = indices_of_top_k(feature_importances, k)
top_k_feature_indices
```

[ ]:
```
array([ 0,  1,  7,  9, 12])
```

[ ]:
```
np.array(attributes)[top_k_feature_indices]
```

[ ]:
```
array(['longitude', 'latitude', 'median_income', 'pop_per_hhold',
       'INLAND'], dtype='<U18')
```

Let's double check that these are indeed the top k features:

[ ]:
```
sorted(zip(feature_importances, attributes), reverse=True)[:k]
```

[ ]:
```
[(0.36615898061813423, 'median_income'),
 (0.16478099356159054, 'INLAND'),
 (0.10879295677551575, 'pop_per_hhold'),
 (0.07334423551601243, 'longitude'),
 (0.06290907048262032, 'latitude')]
```

Looking good... Now let's create a new pipeline that runs the previously defined preparation pipeline, and adds top k feature selection:

[ ]:
```
preparation_and_feature_selection_pipeline = Pipeline([
    ('preparation', full_pipeline),
    ('feature_selection', TopFeatureSelector(feature_importances, k))
])
```

[ ]:
```
housing_prepared_top_k_features = preparation_and_feature_selection_pipeline.
 ↪fit_transform(housing)
```

Let's look at the features of the first 3 instances:

[ ]:
```
housing_prepared_top_k_features[0:3]
```

[ ]:
```
array([[-1.15604281,  0.77194962, -0.61493744, -0.08649871,  0.        ],
       [-1.17602483,  0.6596948 ,  1.33645936, -0.03353391,  0.        ],
       [ 1.18684903, -1.34218285, -0.5320456 , -0.09240499,  0.        ]])
```

Now let's double check that these are indeed the top k features:

```
[ ]: housing_prepared[0:3, top_k_feature_indices]
```

```
[ ]: array([[-1.15604281,  0.77194962, -0.61493744, -0.08649871,  0.        ],
            [-1.17602483,  0.6596948 ,  1.33645936, -0.03353391,  0.        ],
            [ 1.18684903, -1.34218285, -0.5320456 , -0.09240499,  0.        ]])
```

Works great! :)

8.4 4.

Question: Try creating a single pipeline that does the full data preparation plus the final prediction.

```
[ ]: prepare_select_and_predict_pipeline = Pipeline([
        ('preparation', full_pipeline),
        ('feature_selection', TopFeatureSelector(feature_importances, k)),
        ('svm_reg', SVR(**rnd_search.best_params_))
    ])
```

```
[ ]: prepare_select_and_predict_pipeline.fit(housing, housing_labels)
```

```
[ ]: Pipeline(memory=None,
             steps=[('preparation',
                     ColumnTransformer(n_jobs=None, remainder='drop',
                                       sparse_threshold=0.3,
                                       transformer_weights=None,
                                       transformers=[('num',
                                                      Pipeline(memory=None,
                                                               steps=[('imputer',
    SimpleImputer(add_indicator=False,
     copy=True,
     fill_value=None,
     missing_values=nan,
     strategy='median',
     verbose=0)),
    ('attribs_adder',
    CombinedAttributesAdder(add_…
           1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,
           5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,
           1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03]),
                                        k=5)),
                     ('svm_reg',
                      SVR(C=157055.10989448498, cache_size=200, coef0=0.0, degree=3,
                          epsilon=0.1, gamma=0.26497040005002437, kernel='rbf',
                          max_iter=-1, shrinking=True, tol=0.001, verbose=False))],
             verbose=False)
```

Let's try the full pipeline on a few instances:

[ ]:
```
some_data = housing.iloc[:4]
some_labels = housing_labels.iloc[:4]

print("Predictions:\t", prepare_select_and_predict_pipeline.predict(some_data))
print("Labels:\t\t", list(some_labels))
```

```
Predictions:    [203214.28978849 371846.88152572 173295.65441612  47328.3970888
]
Labels:         [286600.0, 340600.0, 196900.0, 46300.0]
```

Well, the full pipeline seems to work fine. Of course, the predictions are not fantastic: they would be better if we used the best `RandomForestRegressor` that we found earlier, rather than the best `SVR`.

8.5 5.

Question: Automatically explore some preparation options using `GridSearchCV`.

Warning: the following cell may take close to 45 minutes to run, or more depending on your hardware.

[ ]:
```
param_grid = [{
    'preparation__num__imputer__strategy': ['mean', 'median', 'most_frequent'],
    'feature_selection__k': list(range(1, len(feature_importances) + 1))
}]

grid_search_prep = GridSearchCV(prepare_select_and_predict_pipeline,␣
 ↪param_grid, cv=5,
                                scoring='neg_mean_squared_error', verbose=2)
grid_search_prep.fit(housing, housing_labels)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] feature_selection__k=1, preparation__num__imputer__strategy=mean

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV]  feature_selection__k=1, preparation__num__imputer__strategy=mean, total=
12.2s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=mean

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   12.2s remaining:    0.0s

[CV]  feature_selection__k=1, preparation__num__imputer__strategy=mean, total=
12.4s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=mean, total=
12.2s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=mean, total=
12.1s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=mean, total=
```

```
12.1s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=median, total=
12.2s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=median, total=
12.3s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=median, total=
12.3s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=median, total=
12.1s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=median, total=
12.2s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=most_frequent,
total=  12.2s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=most_frequent,
total=  12.3s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=most_frequent,
total=  12.2s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=most_frequent,
total=  12.1s
[CV] feature_selection__k=1, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=1, preparation__num__imputer__strategy=most_frequent,
total=  12.2s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=mean, total=
12.6s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=mean, total=
12.8s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=mean, total=
12.6s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=mean, total=
12.7s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=mean, total=
12.6s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=median, total=
```

```
12.6s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=median, total=
12.8s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=median, total=
12.6s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=median, total=
12.7s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=median, total=
12.7s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=most_frequent,
total=  12.6s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=most_frequent,
total=  13.0s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=most_frequent,
total=  12.6s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=most_frequent,
total=  12.9s
[CV] feature_selection__k=2, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=2, preparation__num__imputer__strategy=most_frequent,
total=  12.7s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=mean, total=
12.9s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=mean, total=
12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=mean, total=
12.7s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=mean, total=
12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=mean, total=
12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=median, total=
12.9s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=median, total=
```

```
12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=median, total=
12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=median, total=
12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=median, total=
12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=most_frequent,
total=  12.9s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=most_frequent,
total=  12.9s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=most_frequent,
total=  12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=most_frequent,
total=  12.8s
[CV] feature_selection__k=3, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=3, preparation__num__imputer__strategy=most_frequent,
total=  12.8s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=mean, total=
14.0s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=mean, total=
13.7s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=mean, total=
13.9s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=mean, total=
13.9s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=mean, total=
13.6s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=median, total=
14.0s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=median, total=
13.6s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=median, total=
```

```
14.0s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=median, total=
13.9s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=median, total=
13.6s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=most_frequent,
total=  14.0s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=most_frequent,
total=  13.6s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=most_frequent,
total=  13.9s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=most_frequent,
total=  13.9s
[CV] feature_selection__k=4, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=4, preparation__num__imputer__strategy=most_frequent,
total=  13.6s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=mean, total=
14.3s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=mean, total=
14.4s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=mean, total=
14.5s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=mean, total=
14.6s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=mean, total=
14.2s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=median, total=
14.3s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=median, total=
14.5s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=median, total=
14.6s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=median, total=
```

```
14.6s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=median, total=
14.2s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=most_frequent,
total=  14.2s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=most_frequent,
total=  14.5s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=most_frequent,
total=  14.5s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=most_frequent,
total=  14.5s
[CV] feature_selection__k=5, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=5, preparation__num__imputer__strategy=most_frequent,
total=  14.2s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=mean, total=
14.6s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=mean, total=
14.9s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=mean, total=
14.6s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=mean, total=
14.4s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=mean, total=
15.1s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=median, total=
14.7s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=median, total=
15.1s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=median, total=
14.8s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=median, total=
14.5s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=median, total=
```

```
15.0s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=most_frequent,
total=  14.6s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=most_frequent,
total=  15.0s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=most_frequent,
total=  14.5s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=most_frequent,
total=  14.5s
[CV] feature_selection__k=6, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=6, preparation__num__imputer__strategy=most_frequent,
total=  15.1s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=mean, total=
16.0s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=mean, total=
15.4s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=mean, total=
16.1s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=mean, total=
15.9s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=mean, total=
15.5s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=median, total=
17.0s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=median, total=
16.0s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=median, total=
15.9s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=median, total=
15.3s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=median, total=
16.0s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=most_frequent,
```

```
total=  17.0s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=most_frequent,
total=  15.6s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=most_frequent,
total=  16.5s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=most_frequent,
total=  15.3s
[CV] feature_selection__k=7, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=7, preparation__num__imputer__strategy=most_frequent,
total=  16.5s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=mean, total=
18.8s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=mean, total=
18.4s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=mean, total=
20.8s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=mean, total=
19.7s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=mean, total=
21.9s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=median, total=
19.1s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=median, total=
18.6s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=median, total=
20.6s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=median, total=
20.5s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=median, total=
19.5s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=most_frequent,
total=  19.2s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=most_frequent,
```

```
total=  18.3s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=most_frequent,
total=  20.0s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=most_frequent,
total=  18.7s
[CV] feature_selection__k=8, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=8, preparation__num__imputer__strategy=most_frequent,
total=  20.3s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=mean, total=
26.7s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=mean, total=
27.4s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=mean, total=
25.4s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=mean, total=
25.7s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=mean, total=
23.8s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=median, total=
26.7s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=median, total=
27.6s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=median, total=
23.1s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=median, total=
26.7s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=median, total=
23.5s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=most_frequent,
total=  27.1s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=most_frequent,
total=  27.2s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=most_frequent,
```

```
total=  27.1s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=most_frequent,
total=  26.6s
[CV] feature_selection__k=9, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=9, preparation__num__imputer__strategy=most_frequent,
total=  25.3s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=mean, total=
27.3s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=mean, total=
29.1s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=mean, total=
35.4s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=mean, total=
31.5s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=mean, total=
29.4s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=median,
total=  27.3s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=median,
total=  31.9s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=median,
total=  29.8s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=median,
total=  30.3s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=10, preparation__num__imputer__strategy=median,
total=  28.5s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=10,
preparation__num__imputer__strategy=most_frequent, total=  28.2s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=10,
preparation__num__imputer__strategy=most_frequent, total=  29.4s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=10,
preparation__num__imputer__strategy=most_frequent, total=  28.1s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=10,
```

```
preparation__num__imputer__strategy=most_frequent, total=  30.1s
[CV] feature_selection__k=10, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=10,
preparation__num__imputer__strategy=most_frequent, total=  32.2s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=mean, total=
36.2s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=mean, total=
32.7s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=mean, total=
33.2s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=mean, total=
35.2s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=mean, total=
34.6s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=median,
total=  30.3s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=median,
total=  30.5s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=median,
total=  30.6s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=median,
total=  36.5s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=11, preparation__num__imputer__strategy=median,
total=  35.6s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=11,
preparation__num__imputer__strategy=most_frequent, total=  38.1s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=11,
preparation__num__imputer__strategy=most_frequent, total=  30.1s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=11,
preparation__num__imputer__strategy=most_frequent, total=  30.6s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=11,
preparation__num__imputer__strategy=most_frequent, total=  33.2s
[CV] feature_selection__k=11, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=11,
```

```
preparation__num__imputer__strategy=most_frequent, total=  39.5s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=mean, total=
37.3s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=mean, total=
36.2s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=mean, total=
36.3s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=mean, total=
36.0s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=mean, total=
37.5s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=median,
total=  33.7s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=median,
total=  35.1s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=median,
total=  38.7s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=median,
total=  37.2s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=12, preparation__num__imputer__strategy=median,
total=  35.5s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=12,
preparation__num__imputer__strategy=most_frequent, total=  33.7s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=12,
preparation__num__imputer__strategy=most_frequent, total=  35.2s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=12,
preparation__num__imputer__strategy=most_frequent, total=  41.3s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=12,
preparation__num__imputer__strategy=most_frequent, total=  34.0s
[CV] feature_selection__k=12, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=12,
preparation__num__imputer__strategy=most_frequent, total=  34.3s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=mean, total=
```

```
44.0s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=mean, total=
39.5s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=mean, total=
43.5s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=mean, total=
40.3s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=mean, total=
33.3s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=median,
total=  36.6s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=median,
total=  43.0s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=median,
total=  45.1s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=median,
total=  44.0s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=13, preparation__num__imputer__strategy=median,
total=  39.4s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=13,
preparation__num__imputer__strategy=most_frequent, total=  37.0s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=13,
preparation__num__imputer__strategy=most_frequent, total=  43.5s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=13,
preparation__num__imputer__strategy=most_frequent, total=  43.6s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=13,
preparation__num__imputer__strategy=most_frequent, total=  43.1s
[CV] feature_selection__k=13, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=13,
preparation__num__imputer__strategy=most_frequent, total=  38.2s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=mean, total=
35.4s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=mean, total=
```

```
41.7s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=mean, total=
43.0s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=mean, total=
43.4s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=mean, total=
38.7s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=median,
total=  41.4s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=median,
total=  43.2s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=median,
total=  43.3s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=median,
total=  42.3s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=14, preparation__num__imputer__strategy=median,
total=  38.8s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=14,
preparation__num__imputer__strategy=most_frequent, total=  43.0s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=14,
preparation__num__imputer__strategy=most_frequent, total=  37.5s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=14,
preparation__num__imputer__strategy=most_frequent, total=  39.2s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=14,
preparation__num__imputer__strategy=most_frequent, total=  43.5s
[CV] feature_selection__k=14, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=14,
preparation__num__imputer__strategy=most_frequent, total=  51.2s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=mean, total=
42.9s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=mean, total=
41.2s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=mean, total=
```

```
43.6s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=mean, total=
35.0s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=mean, total=
40.9s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=median,
total=  37.0s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=median,
total=  42.3s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=median,
total=  44.8s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=median,
total=  46.4s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=15, preparation__num__imputer__strategy=median,
total=  43.3s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=15,
preparation__num__imputer__strategy=most_frequent, total=  45.7s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=15,
preparation__num__imputer__strategy=most_frequent, total=  45.2s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=15,
preparation__num__imputer__strategy=most_frequent, total=  46.3s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=15,
preparation__num__imputer__strategy=most_frequent, total=  38.0s
[CV] feature_selection__k=15, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=15,
preparation__num__imputer__strategy=most_frequent, total=  46.4s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=mean, total=
43.8s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=mean, total=
44.6s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=mean, total=
41.6s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=mean, total=
```

```
44.6s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=mean
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=mean, total=
38.2s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=median,
total=  38.3s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=median,
total=  43.7s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=median,
total=  41.9s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=median,
total=  35.6s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=median
[CV]  feature_selection__k=16, preparation__num__imputer__strategy=median,
total=  42.2s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=16,
preparation__num__imputer__strategy=most_frequent, total=  38.5s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=16,
preparation__num__imputer__strategy=most_frequent, total=  43.4s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=16,
preparation__num__imputer__strategy=most_frequent, total=  41.3s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=16,
preparation__num__imputer__strategy=most_frequent, total=  41.5s
[CV] feature_selection__k=16, preparation__num__imputer__strategy=most_frequent
[CV]  feature_selection__k=16,
preparation__num__imputer__strategy=most_frequent, total=  43.7s

[Parallel(n_jobs=1)]: Done 240 out of 240 | elapsed: 102.3min finished
```

```
[ ]: GridSearchCV(cv=5, error_score=nan,
                  estimator=Pipeline(memory=None,
                                     steps=[('preparation',
                                             ColumnTransformer(n_jobs=None,
                                                               remainder='drop',
                                                               sparse_threshold=0.3,
    transformer_weights=None,
                                                               transformers=[('num',
    Pipeline(memory=None,
      steps=[('imputer',
              SimpleImputer(add_indicator=False,
```

```
                        copy=True,
                        fill_value=None,
                        missing_values=nan,
                        strategy='median',
                        verbose=0)),
          (…
                                          kernel='rbf', max_iter=-1,
                                          shrinking=True, tol=0.001,
                                          verbose=False))],
                               verbose=False),
              iid='deprecated', n_jobs=None,
              param_grid=[{'feature_selection__k': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                   10, 11, 12, 13, 14, 15, 16],
                           'preparation__num__imputer__strategy': ['mean',
                                                                   'median',
'most_frequent']}],
              pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
              scoring='neg_mean_squared_error', verbose=2)
```

[ ]:
```
grid_search_prep.best_params_
```

[ ]:
```
{'feature_selection__k': 15,
 'preparation__num__imputer__strategy': 'most_frequent'}
```

The best imputer strategy is `most_frequent` and apparently almost all features are useful (15 out of 16). The last one (`ISLAND`) seems to just add some noise.

Congratulations! You already know quite a lot about Machine Learning. :)