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

```
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
```

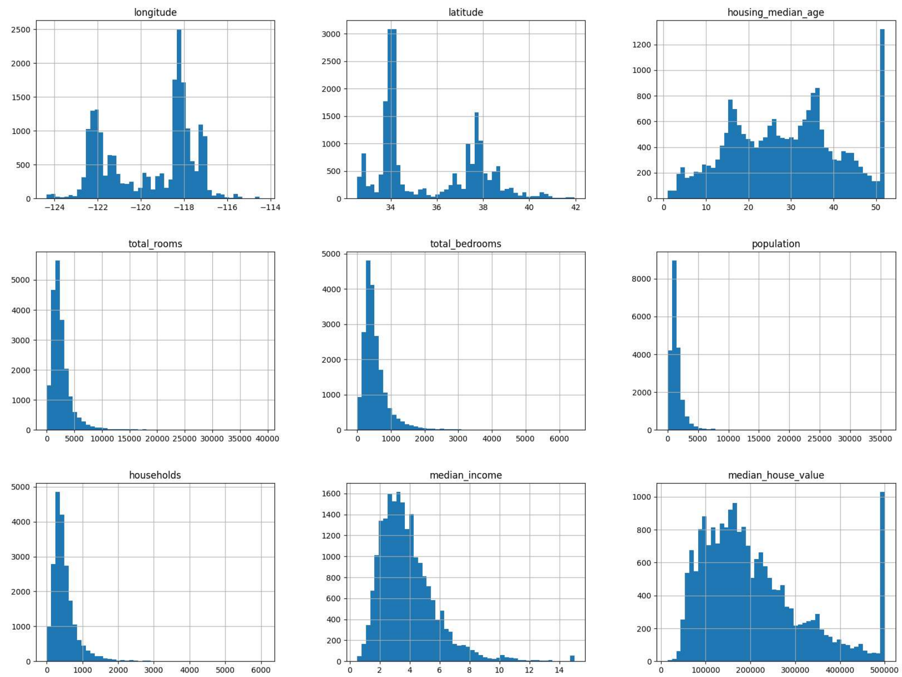
```
housing['ocean_proximity'].value_counts()
```

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

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	2064
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	142
std	2.003532	2.135952	12.585558	2181.615252	421.385070	113
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	78
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	116

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



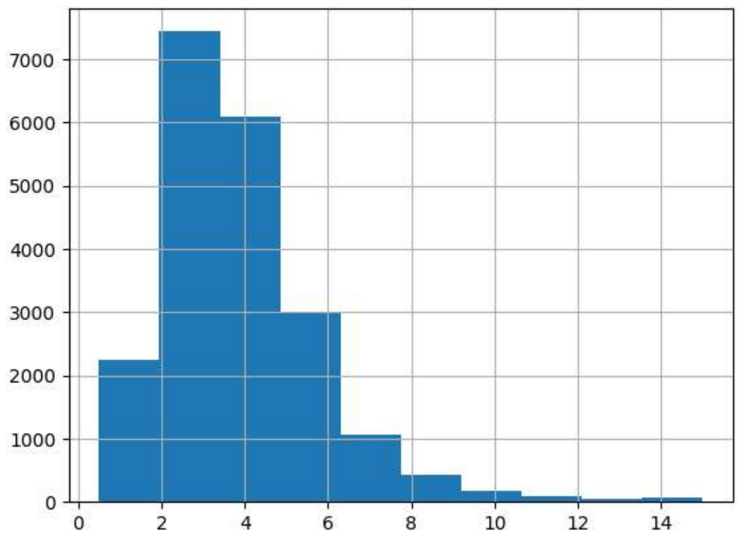

```
from sklearn.model_selection import train_test_split

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

```
test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0

```
housing['median_income'].hist()
plt.show()
```




```
import numpy as np
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()
plt.show()
```

```
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)
```

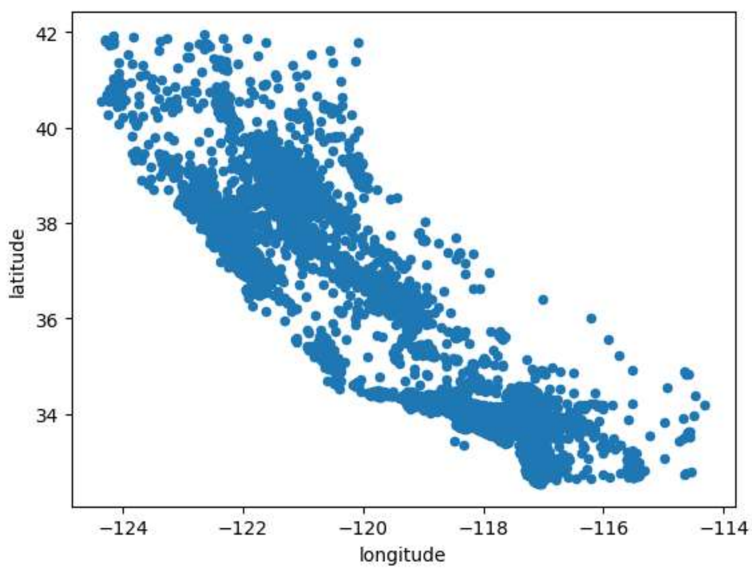
```
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
Name: income_cat, dtype: float64
```

```
print(strat_train_set.columns)
print(strat_test_set.columns)
```

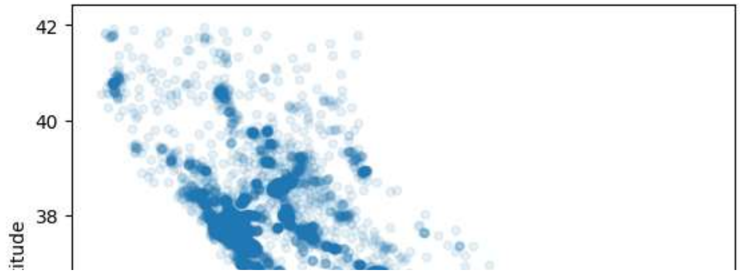
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity', 'income_cat'],
      dtype='object')
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity', 'income_cat'],
      dtype='object')
```

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

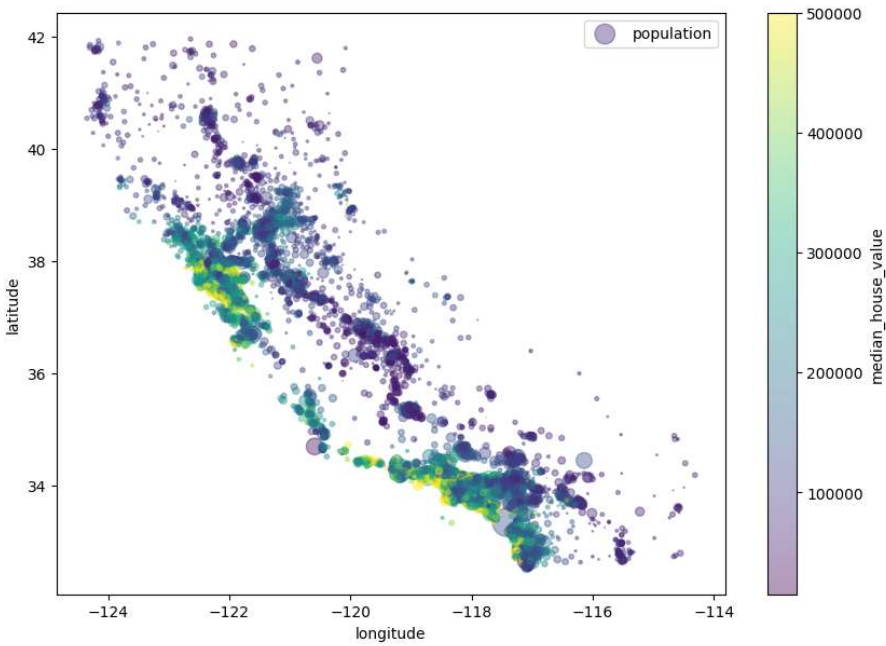
```
housing.plot(kind="scatter", x="longitude", y="latitude")
plt.show()
```




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




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



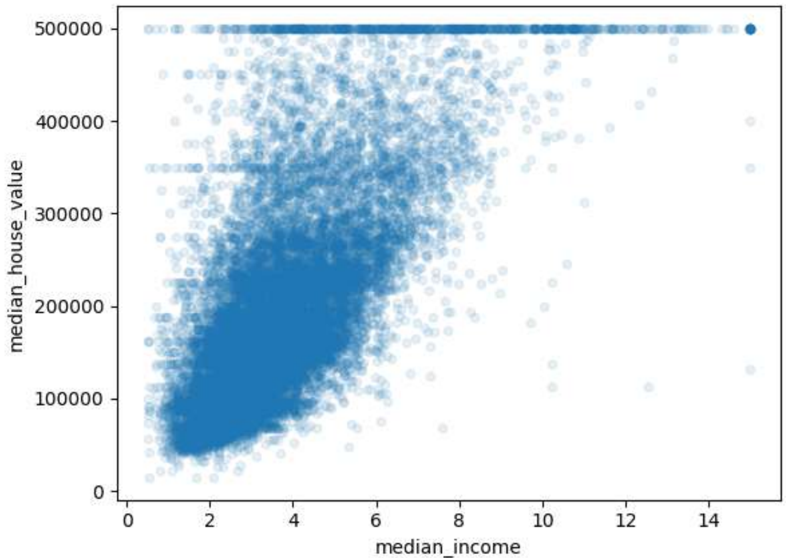

```
# Assuming 'housing' is your DataFrame
numeric_columns = housing.select_dtypes(include=[np.number]).columns.tolist()

# Calculate correlation matrix
corr_matrix = housing[numeric_columns].corr()

# Display correlation with the target variable
corr_matrix['median_house_value'].sort_values(ascending=False)
```

```
median_house_value    1.000000
median_income         0.687151
total_rooms           0.135140
housing_median_age    0.114146
households            0.064590
total_bedrooms        0.047781
population           -0.026882
longitude            -0.047466
latitude             -0.142673
Name: median_house_value, dtype: float64
```

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




```
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"]


# Select only numeric columns for correlation analysis
numeric_columns = housing.select_dtypes(include=[np.number]).columns.tolist()

# Calculate correlation matrix
corr_matrix = housing[numeric_columns].corr()

# Display correlation with the target variable
corr_matrix['median_house_value'].sort_values(ascending=False)
```

```
median_house_value          1.000000
median_income               0.687151
rooms_per_household         0.146255
total_rooms                 0.135140
housing_median_age          0.114146
households                  0.064590
total_bedrooms              0.047781
population_per_household   -0.021991
population                 -0.026882
longitude                  -0.047466
latitude                   -0.142673
bedrooms_per_room          -0.259952
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	housing_median_age	total_rooms	total_bedrooms	po
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	1651
mean	-119.575635	35.639314	28.653404	2622.539789	534.914639	141
std	2.001828	2.137963	12.574819	2138.417080	412.665649	111
min	-124.350000	32.540000	1.000000	6.000000	2.000000	
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	78
50%	-118.510000	34.260000	29.000000	2119.000000	433.000000	116
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	171
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	3568

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

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

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

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

```
imputer.fit(housing_num)
```

```
▾ SimpleImputer
SimpleImputer(strategy='median')
```

```
imputer.statistics_
```

```
array([-118.51   ,   34.26   ,   29.     , 2119.     ,  433.     ,
       1164.     ,  408.     ,    3.54155,    3.     ])
```

```
housing_num.median().values
```

```
<ipython-input-69-8050cbb6f664>:1: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future versi
  housing_num.median().values
array([-118.51   ,   34.26   ,   29.     , 2119.     ,  433.     ,
       1164.     ,  408.     ,    3.54155])
```

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

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

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND

```
from sklearn.preprocessing import OneHotEncoder

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

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` i
  warnings.warn(
array([[0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       ...,
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

```
#CUSTOM TRANSFORMATIONS
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)
```

PIPELINE FOR TRANSFORMATION

```
# TRANSFORMATION PIPELINES
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

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

house_num_tr = num_pipeline.fit_transform(housing_num)


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),
    ])

house_prepared = full_pipeline.fit_transform(housing)
```

```
house_prepared
```

```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.        ,
         0.        ,  0.        ],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ...,  0.        ,
         0.        ,  0.        ],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.        ,
         0.        ,  0.        ],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.        ,
         0.        ,  0.        ]])
```

train a model

```
# Let's train a linear regression model
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
#X_train, y_train = train_data.drop(['median_house_value'],axis=1) , train_data['median_house_value']
X_train_s = scaler.fit_transform(house_prepared)
lin_reg = LinearRegression()
lin_reg= LinearRegression()
lin_reg.fit(X_train_s, housing_labels)
```

```
▾ LinearRegression
LinearRegression()
```

```
#let's measure RSME(root mean squared error) of our model

from sklearn.metrics import mean_squared_error

house_predictions = lin_reg.predict(house_prepared)
lin_mse = mean_squared_error(housing_labels, house_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

```
6.629455738486618e+16
```

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

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
#Let's evatuale on training set
house_predictions = tree_reg.predict(house_prepared)
tree_mse = mean_squared_error(housing_labels, house_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

```
0.0
```

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, house_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

```
# let's see the scores
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [72125.10591877 70703.72251615 67341.33012937 70747.13207239
 69006.2502759  77200.12592331 71592.71814917 73500.26594292
 68421.38251056 71003.12958725]
Mean: 71164.11630257833
Standard deviation: 2649.577918222678
```

```
# let's look for scores for linear regression:
lin_scores = cross_val_score(lin_reg, house_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
# the Decision Tree model is overfitting so badly that it performs worse than the Linear Regression model.
```

```
Scores: [71523.78333874 64044.46774989 67454.97869698 68514.10137273
 66303.62531226 72166.63405138 74464.08841381 68570.11804395
 66063.64175868 69870.86192291]
Mean: 68897.63006613276
Standard deviation: 3002.746127534861
```

```
# let's try Random Forest Regressor
# (Random Forests work by training many Decision Trees on random subsets of the features,
# then averaging out their predictions)
from sklearn.ensemble import RandomForestRegressor

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

```
▾ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
housing_predictions = forest_reg.predict(house_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
print(f"Random Forest RMSE: {forest_rmse}")
```

```
Random Forest RMSE: 18675.224916252282
```

```
from sklearn.model_selection import cross_val_score

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

```
Scores: [51553.65292335 48797.89565614 47005.23947642 52046.73567245
 47700.78025873 51824.08544879 52582.59165129 49949.79025967
 48680.25622229 54019.67674791]
Mean: 50416.070431704204
Standard deviation: 2201.612779754884
```

Fine-Tune the model using randomized search

```
# RANDOMIZED SEARCH
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(house_prepared, housing_labels)
```

```
▸ RandomizedSearchCV
▸ estimator: RandomForestRegressor
  ▸ RandomForestRegressor
```

```
best_forest =rnd_search.best_estimator_
```

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

```
49799.635737761106 {'max_features': 7, 'n_estimators': 180}
52293.114093913726 {'max_features': 5, 'n_estimators': 15}
51327.353255586764 {'max_features': 3, 'n_estimators': 72}
51528.97117998048 {'max_features': 5, 'n_estimators': 21}
49958.42482333546 {'max_features': 7, 'n_estimators': 122}
51270.531241462595 {'max_features': 3, 'n_estimators': 75}
51172.437672640175 {'max_features': 3, 'n_estimators': 88}
50255.14987044715 {'max_features': 5, 'n_estimators': 100}
50894.38729795359 {'max_features': 3, 'n_estimators': 150}
65022.070435017646 {'max_features': 5, 'n_estimators': 2}
```

```
rnd_search.best_estimator_.score(house_prepared, housing_labels)
```

```
0.9749873185793982
```

pipeline for scaling and training

```
from sklearn.feature_selection import SelectFromModel
from sklearn.pipeline import Pipeline
pipeline = Pipeline([("scaler",scaler),
                     ('selector', SelectFromModel(RandomForestRegressor(random_state=42),threshold=0.005)),
                     ("random_forest",best_forest)])
```

```
pipeline.fit(house_prepared, housing_labels)
pipeline.score(house_prepared, housing_labels)
```

```
0.9752809553073623
```