Housing

April 26, 2021

1 Chapter 2 - End-to-End Machine Learning Project

1.1 Get the data

1.1.1 Download housing Data

[1]:
```
import os
import tarfile
import urllib

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):
    os.makedirs(housing_path, exist_ok=True)
    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()
```

1.1.2 Load the data

[2]:
```
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()
```

1.1.3 Visualize Data structure

[3]:
```
housing.head()
```

[3]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

1.1.4 Visualize Data description

[4]:
```
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
```

1.1.5 Feature count

[5]:
```
housing["ocean_proximity"].value_counts()
```

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

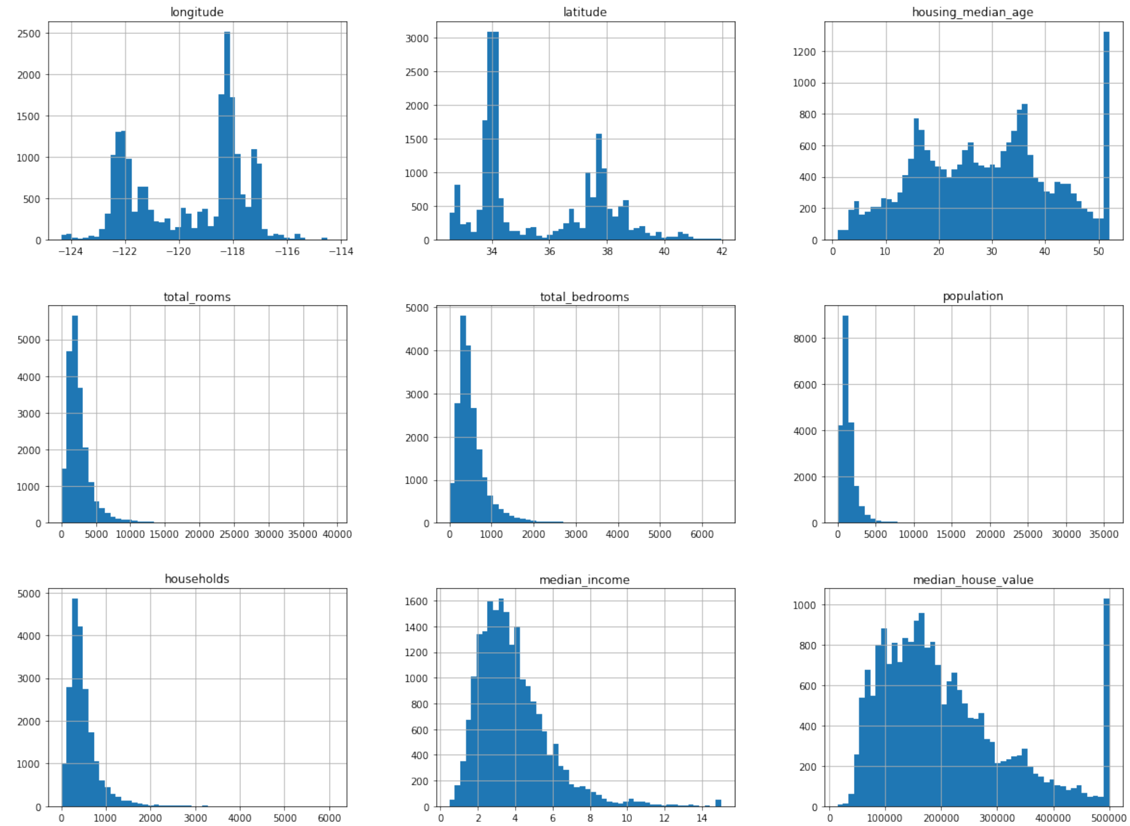
1.1.6 Summary of the numerical attributes

[6]:
```python
housing.describe()
```

[6]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

1.1.7 Plotting histogram

[7]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



1.2 Creating a test set

[8]:
```
import numpy as np

def split_train_test(data, test_ratio):
    np.random.seed(42)
    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)
```

We can use Scikit-Learn functions to split datasets into multiple subsets

[9]:
```
from sklearn.model_selection import train_test_split

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

1.2.1 Creating a stable test set, using instance identifier

[10]:
```
from zlib import crc32

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

def split_test_set_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]

housing_with_id = housing.reset_index()

# housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]

train_set, test_set = split_test_set_by_id(housing_with_id, 0.2, "index")
```

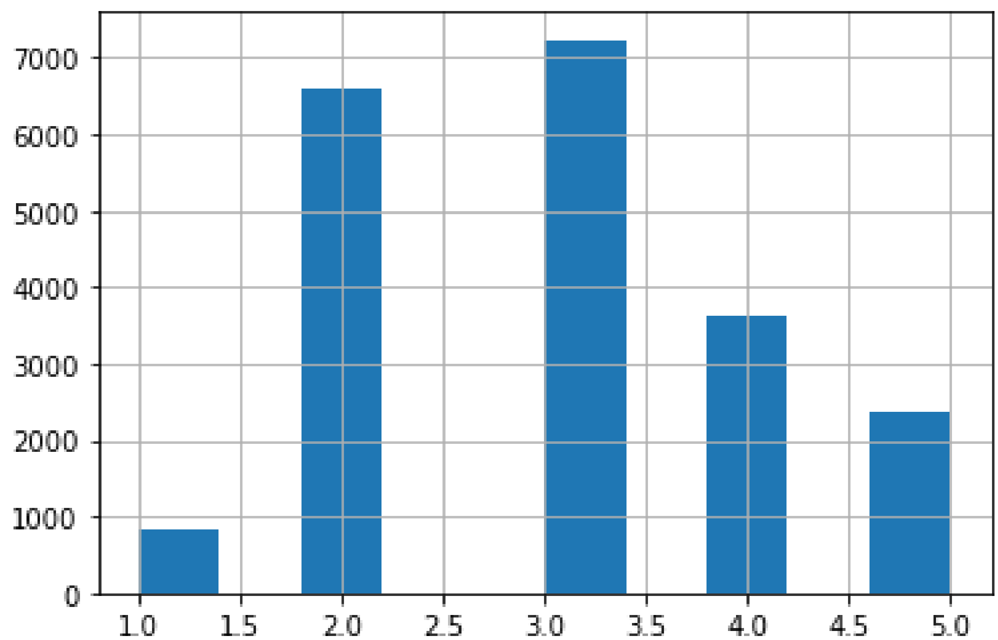
Using Scikit-Learn to generate test set

[11]:
```
from sklearn.model_selection import train_test_split

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

Creating income category attribute with five categories

[12]:
```
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"].hist()
```

[12]:
```
<AxesSubplot:>
```



Doing stratified sampling

[13]:
```python
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]
```

Looking at the income category proportions in the test set

[14]:
```python
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

[14]:
```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

Removing the income_cat attribute so the data is back to its original state

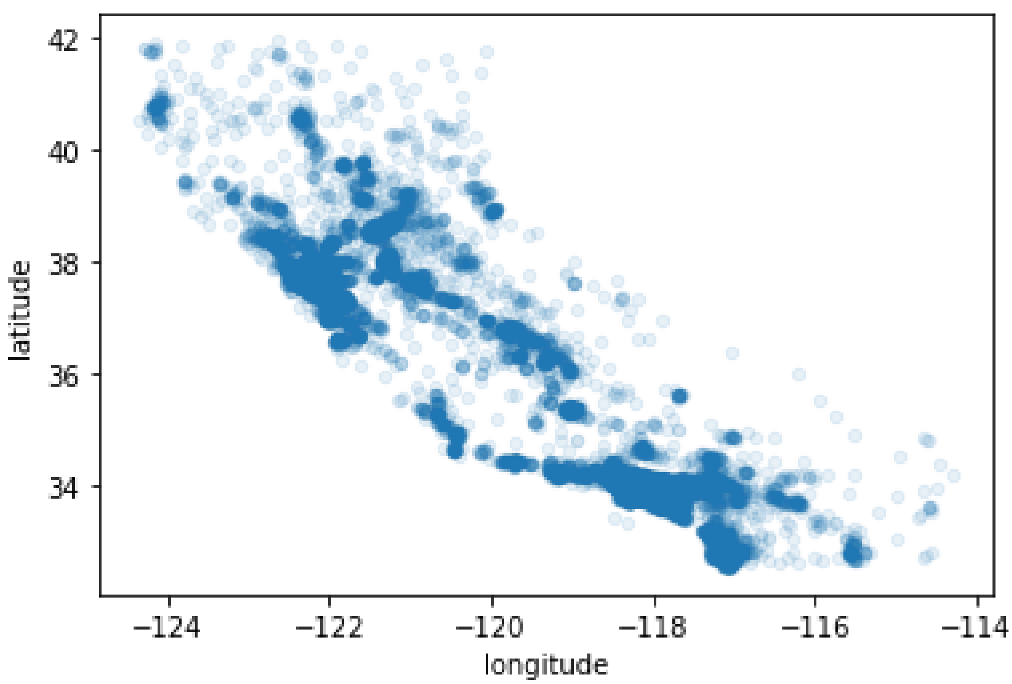
[15]:
```python
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

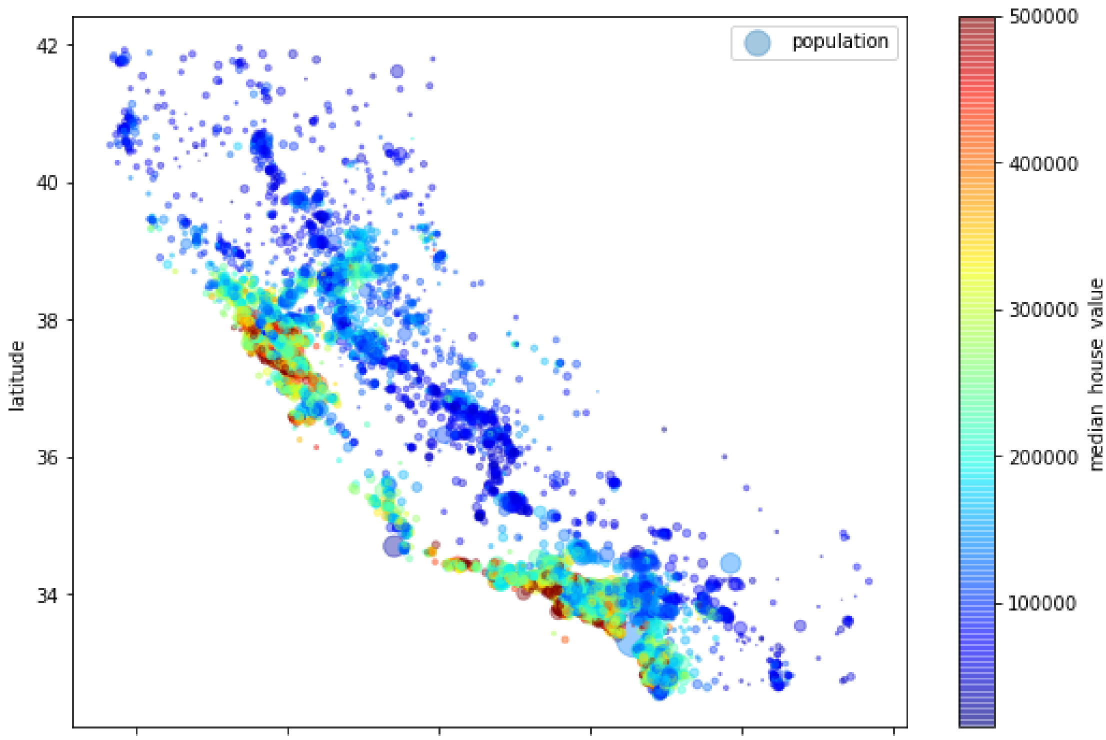
1.3 Discover and Visualize the Data to Gain Insights

1.3.1 Visualizing Geographical Data

[16]:
```
housing = strat_train_set.copy()

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

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,
)

plt.legend()
```

[16]:
```
<matplotlib.legend.Legend at 0x7f6fcc5e3190>
```





1.3.2 Looking for Correlations

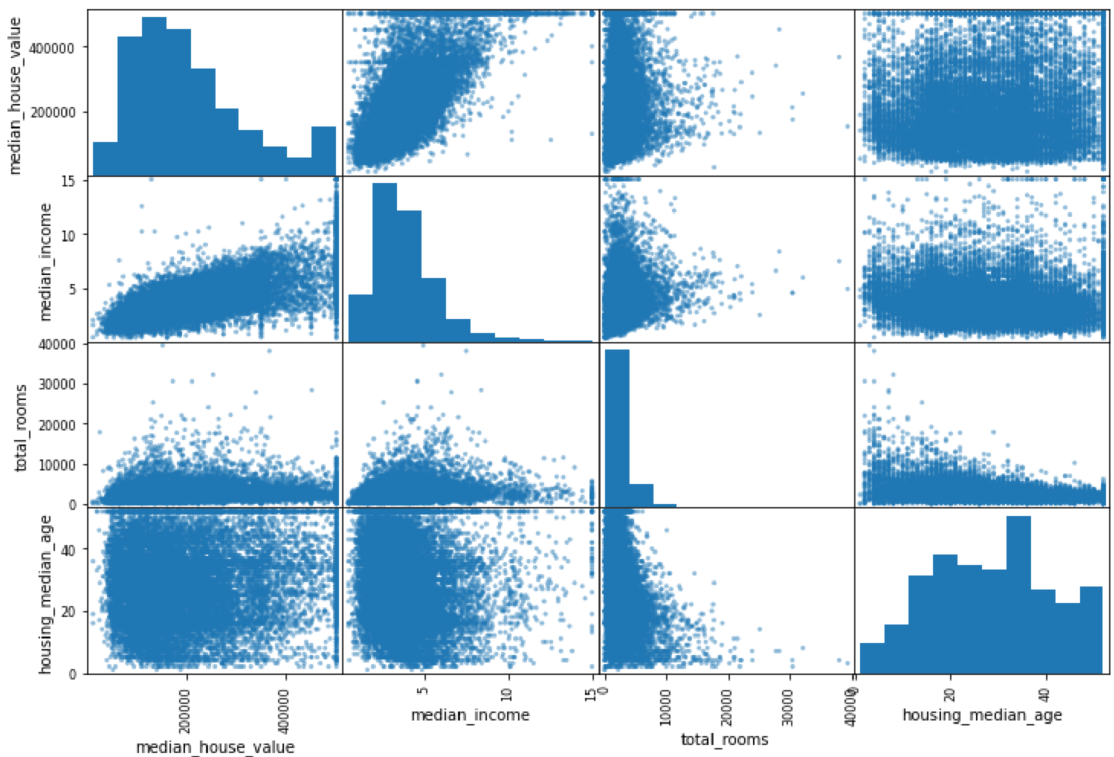
[17]:
```
corr_matrix = housing.corr()

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

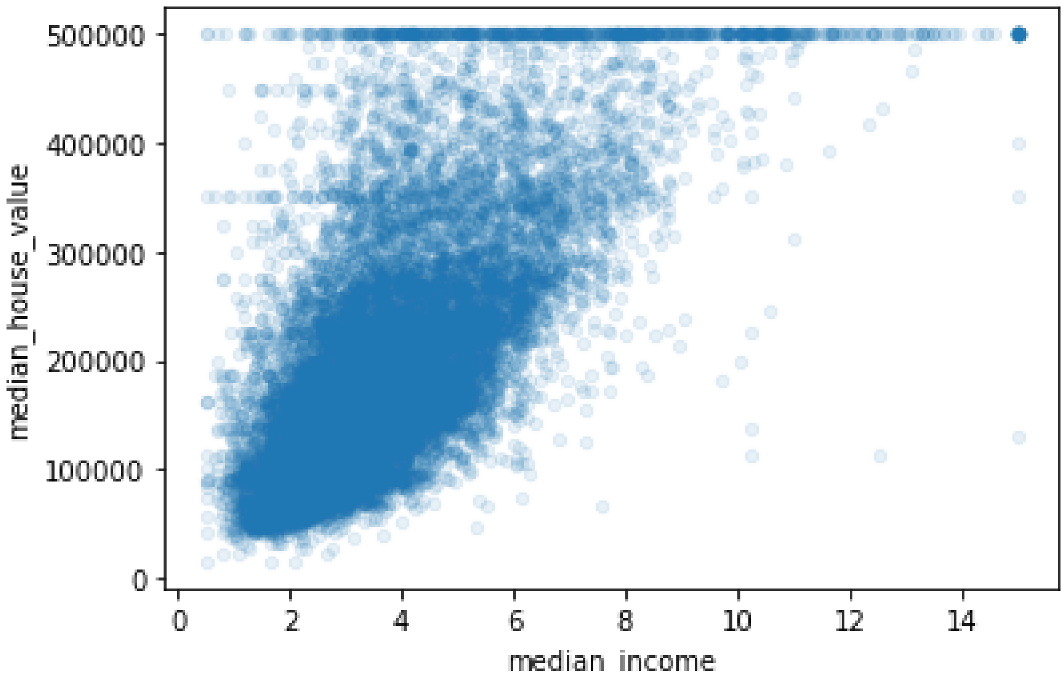
[17]:
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

[18]:
```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",␣
 ↪"housing_median_age"]
```

```python
scatter_matrix(housing[attributes], figsize=(12, 8))

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

[18]: <AxesSubplot:xlabel='median_income', ylabel='median_house_value'>





1.3.3 Experimenting with Attribute Combinations

[19]:
```
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)
```

[19]:
```
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
```

1.4 Prepare the Data for Machine Learning Algorithms

[20]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

1.4.1 Data Cleaning

[21]:
```
median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median, inplace=True)
```

Data cleaning with Scikit-Learn

[22]:
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)

print(imputer.statistics_)
print(housing_num.median().values)

X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns, index=housing_num.
 ↪index)
```

```
[-118.51     34.26      29.      2119.5      433.      1164.       408.
    3.5409]
[-118.51     34.26      29.      2119.5      433.      1164.       408.
    3.5409]
```

1.4.2 Handling Text and Categorical Attributes

[23]:
```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

[23]:

	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
```

Converting categories from text to numbers

[24]:
```python
from sklearn.preprocessing import OrdinalEncoder

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

```
[[0.]
 [0.]
 [4.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]]
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
      dtype=object)]
```

Creating a one-hot encoding

[25]:
```python
from sklearn.preprocessing import OneHotEncoder

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

[25]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

1.4.3 Custom Transformers

[26]:
```python
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
```

```
        return self
    def transform(self, X):
        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)
```

1.4.4 Transformation Pipelines

[27]:
```
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)
```

A single transformer able to handle all columns, applying the appropriate transformations to each column

[28]:
```
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)
```

1.5 Select and Train a Model

1.5.1 Training and Evaluating on the Training Set with Linear Regression model

[29]:
```
from sklearn.linear_model import LinearRegression

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

some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)
print("Predictions:", lin_reg.predict(some_data_prepared))
print("Labels:", list(some_labels))


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
```

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

[29]: 68628.19819848923

Training a DecisionTreeRegressor

[30]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)

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

[30]: 0.0

1.5.2 Better Evaluation Using Cross-Validation

[31]:
```
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(), end='\n \n')

display_scores(tree_rmse_scores)

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: [69327.01708558 65486.39211857 71358.25563341 69091.37509104
 70570.20267046 75529.94622521 69895.20650652 70660.14247357
 75843.74719231 68905.17669382]
Mean: 70666.74616904806
Standard deviation: 2928.322738055112

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: 2731.674001798348
```

Training a RandomForestRegressor

[32]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)


scores = cross_val_score(forest_reg, housing_prepared, housing_labels,␣
 ↪scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-scores)

display_scores(forest_rmse_scores)
```

```
Scores: [49557.6095063  47584.54435547 49605.349788   52325.13724488
 49586.9889247  53154.87424699 48800.48987508 47880.32844243
 52958.68645964 50046.17489414]
Mean: 50150.018373763225
Standard deviation: 1902.0697041387534
```

1.6 Fine-Tune Your Model

1.6.1 Grid Search

[33]:
```python
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
]

forest_reg = RandomForestRegressor()
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)

print(grid_search.best_params_)

print(grid_search.best_estimator_)

cvres = grid_search.cv_results_

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

```
{'max_features': 6, 'n_estimators': 30}
RandomForestRegressor(max_features=6, n_estimators=30)
63433.40391736115 {'max_features': 2, 'n_estimators': 3}
56049.06443637957 {'max_features': 2, 'n_estimators': 10}
52824.848527310685 {'max_features': 2, 'n_estimators': 30}
60924.41328448018 {'max_features': 4, 'n_estimators': 3}
52713.650694157855 {'max_features': 4, 'n_estimators': 10}
50660.92190603788 {'max_features': 4, 'n_estimators': 30}
59604.01184459288 {'max_features': 6, 'n_estimators': 3}
52347.604952708156 {'max_features': 6, 'n_estimators': 10}
49923.3473574243 {'max_features': 6, 'n_estimators': 30}
59308.345962472304 {'max_features': 8, 'n_estimators': 3}
52320.77872780119 {'max_features': 8, 'n_estimators': 10}
50080.73594153239 {'max_features': 8, 'n_estimators': 30}
62160.41351492645 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
```

```
54391.4645181866 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
60269.48857946438 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52791.4337224519 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
59188.03690511952 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
52193.83170447224 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

1.6.2 Analyze the Best Models and Their Errors

[34]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
print(feature_importances)

extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
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)
```

```
[7.55720671e-02 6.39878625e-02 4.24072059e-02 1.82928273e-02
 1.68924417e-02 1.75601900e-02 1.66881781e-02 3.03268232e-01
 6.31565549e-02 1.08958622e-01 8.44196144e-02 8.53515062e-03
 1.73063945e-01 8.08024120e-05 2.96250425e-03 4.15380176e-03]
```

[34]:
```
[(0.303268232301214, 'median_income'),
 (0.1730639450304893, 'INLAND'),
 (0.10895862174634888, 'pop_per_hhold'),
 (0.0844196144263057, 'bedrooms_per_room'),
 (0.07557206707255014, 'longitude'),
 (0.06398786252477989, 'latitude'),
 (0.06315655490931624, 'rooms_per_hhold'),
 (0.04240720593117474, 'housing_median_age'),
 (0.01829282732311651, 'total_rooms'),
 (0.017560189966804522, 'population'),
 (0.01689244166020893, 'total_bedrooms'),
 (0.01668817806453196, 'households'),
 (0.008535150622100876, '<1H OCEAN'),
 (0.0041538017589390725, 'NEAR OCEAN'),
 (0.0029625042500806965, 'NEAR BAY'),
 (8.080241203860085e-05, 'ISLAND')]
```

1.6.3 Evaluate Your System on the Test Set

[35]:
```
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()
```

```python
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)
print(final_rmse)
```

```
48760.26530172545
```