Understanding the Data

In [ ]:
```
import os
import tarfile
from six.moves import urllib
import pandas as pd

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()
```

upon calling the function `fetch_housing_data()`, it

1. creates datasets/housing directory
2. downloads housing.tgz
3. extracts housing.csv in the directory

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

In [ ]:
```
fetch_housing_data() # get the csv

housing = load_housing_data() # load it
housing.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_incor
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.32
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.30
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.25
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.64
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.84

In [ ]:
```
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
```

In [ ]:

```
# examine the ocean_proximity column

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

Out[ ]:

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

In [ ]:

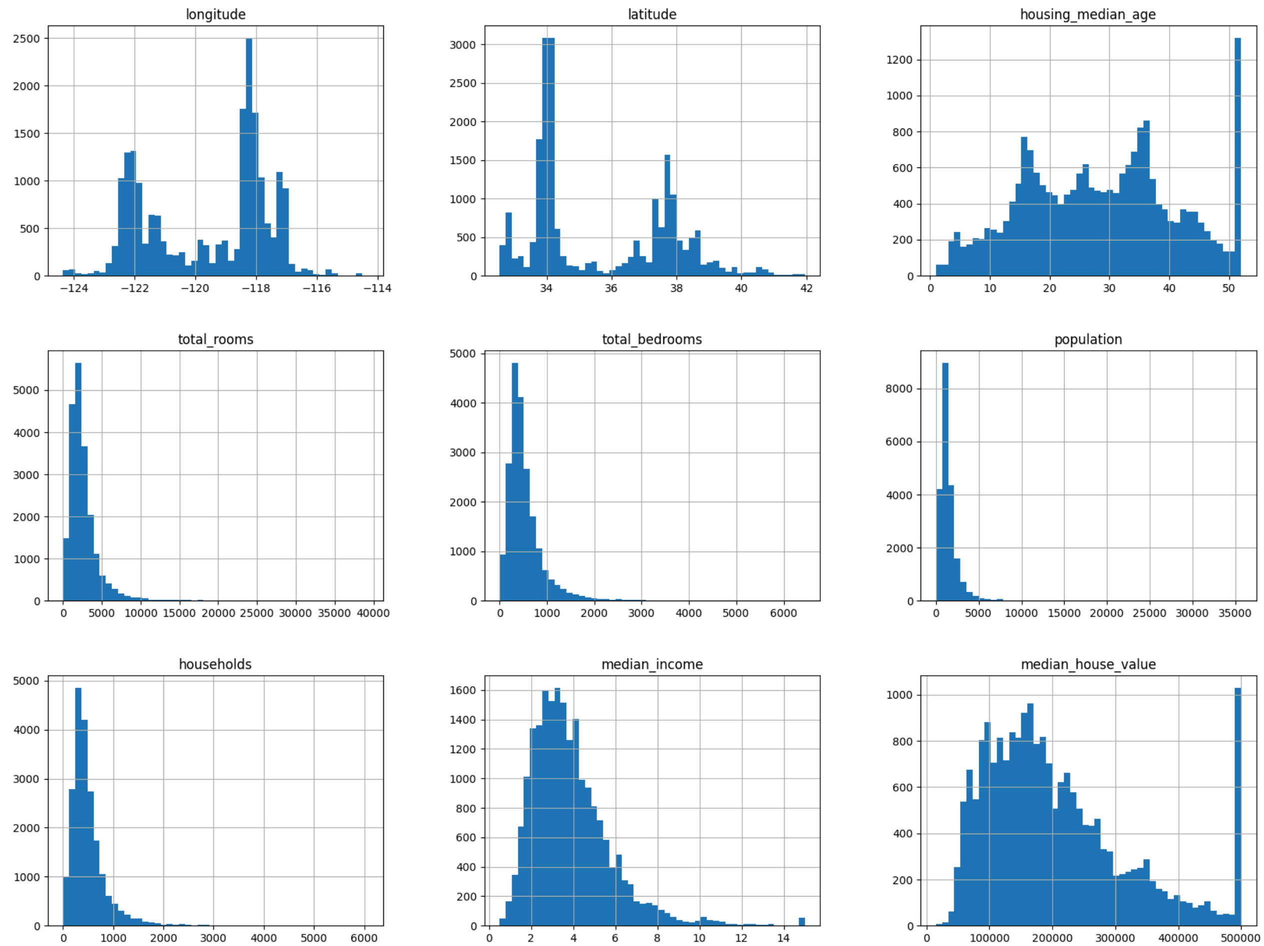
```
housing.describe() # describes the summary of numerical attributes
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househo
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000

In [ ]:

```
housing.hist(bins=50, figsize=(20,15))
__import__("matplotlib").pyplot.show()
```



Vertical Axis $\rightarrow$ Number of entries/instances of the value

Horizontal Axis $\rightarrow$ range of values

In [ ]:
```
housing["median_income"]
```

Out[ ]:
```
0        8.3252
1        8.3014
2        7.2574
3        5.6431
4        3.8462
          ...
20635    1.5603
20636    2.5568
20637    1.7000
20638    1.8672
20639    2.3886
Name: median_income, Length: 20640, dtype: float64
```

You find out that the median_income is represented in ten thousands of US dollars.

eg: 8 means $80,000

Making a Test Set and Training Set

In [ ]:
```
# creating a test set (generally 20% of the dataset or even less if dataset is too large)

import numpy as np

def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices, train_indices = shuffled_indices[:test_set_size], shuffled_indices[test_set_size:
```

```
    return data.iloc[test_indices], data.iloc[train_indices]

test_set, train_set = split_train_test(housing, 0.2)
test_set.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
1121	-121.58	39.79	19.0	2636.0	523.0	1184.0	465.0	
14206	-117.06	32.69	9.0	521.0	111.0	491.0	110.0	
5256	-118.48	34.07	37.0	4042.0	549.0	1318.0	542.0	
15149	-116.90	32.90	19.0	3090.0	552.0	1621.0	520.0	
16915	-122.35	37.56	52.0	1659.0	191.0	519.0	201.0	

Break-proofing the split approach

The next time the code is run again, it will choose different set of indices, over time, the model will get to see the whole dataset which is to be avoided.

Solutions:

1. Save test set on first run and load it in subsequent runs
2. Set a random number generator's seed before calling permutation() so it generates same shuffled indices

But both the solutions will break when fetching an updated dataset.

In [ ]:

```
# Best solution

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

Working of the test_set_check

1. the identifier is converted into a 64-bit integer.
2. The crc32 function returns the hash of the 64-bit identifier and it is masked with the highest value of 32 bit integer which is 0xffffffff to truncate it into a 32-bit integer.
3. the test_ratio is multiplied with 2^{32} to maintain the scale.
4. The hash is checked with the test_ratio

In [ ]:

```
# Add a custom ID column to the dataset since it does not have it

housing
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	
...	...	...	...	...	...	...	...	
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	

20640 rows × 10 columns

In [ ]:

```python
housing_with_id = housing.reset_index() # adds an index column
housing_with_id
```

Out[ ]:

	index	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	r
0	0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	
...	...	...	...	...	...	...	...	...	
20635	20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	
20636	20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	
20637	20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	
20638	20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	
20639	20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	

20640 rows × 11 columns

In [ ]:

```python
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")

test_set
```

Out[ ]:

	index	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	r
2	2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
5	5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	
12	12	-122.26	37.85	52.0	2491.0	474.0	1098.0	468.0	
16	16	-122.27	37.85	52.0	1966.0	347.0	793.0	331.0	
23	23	-122.27	37.84	52.0	1688.0	337.0	853.0	325.0	
...	...	...	...	...	...	...	...	...	
20615	20615	-121.54	39.08	23.0	1076.0	216.0	724.0	197.0	
20617	20617	-121.53	39.06	20.0	561.0	109.0	308.0	114.0	
20622	20622	-121.44	39.00	20.0	755.0	147.0	457.0	157.0	
20626	20626	-121.43	39.18	36.0	1124.0	184.0	504.0	171.0	
20629	20629	-121.39	39.12	28.0	10035.0	1856.0	6912.0	1818.0	

4128 rows × 11 columns

Same thing can be done with Scikit-Learns train_test_split method:

In [ ]:

```
from sklearn.model_selection import train_test_split

train_test, test_set = train_test_split(housing, test_size=0.2, random_state=42) # the integer 42 i
test_set # run it as many times as you want and the result will be the same (unless you change the
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	
...	...	...	...	...	...	...	...	
15362	-117.22	33.36	16.0	3165.0	482.0	1351.0	452.0	
16623	-120.83	35.36	28.0	4323.0	886.0	1650.0	705.0	
18086	-122.05	37.31	25.0	4111.0	538.0	1585.0	568.0	
2144	-119.76	36.77	36.0	2507.0	466.0	1227.0	474.0	
3665	-118.37	34.22	17.0	1787.0	463.0	1671.0	448.0	

4128 rows × 10 columns

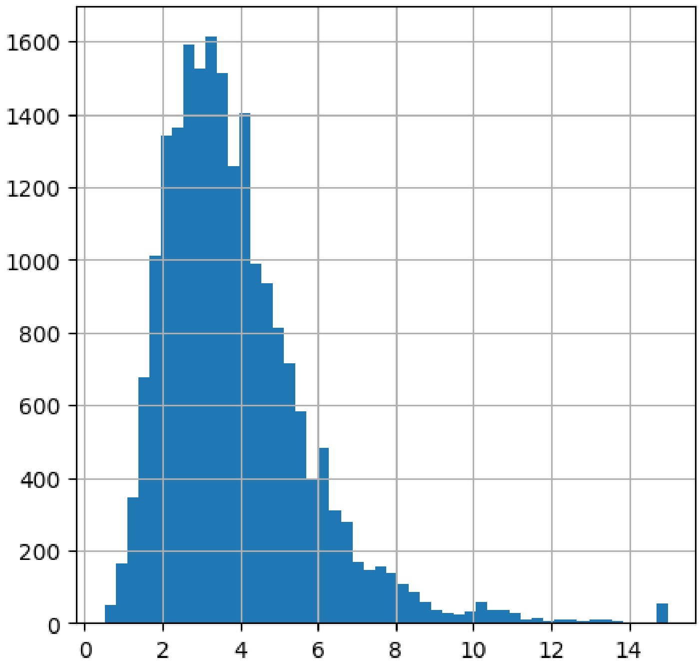
Avoiding Sampling Bias

It is important for a model to consider each and every category of an attribute to avoid sampling bias. for example, a survey on population of a country consisting of 55% males and 45% females need to take into consideration of 55 males and 45 females in a 100 people. If the percentages does not represent the whole population of the country, then the survey may be considered as biased.

Suppose the experts told you that the median income is a very important attribute to predict median housing prices. You should ensure that the test set is representative of the various categories of incomes in the whole dataset.

In [ ]:
```
plt = __import__("matplotlib").pyplot # using matplotlib's pyplot
```

In [ ]:
```
housing["median_income"].hist(figsize=(5, 5), bins=50)

plt.show()
```



In [ ]:
```
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[["median_income", "income_cat"]]
```

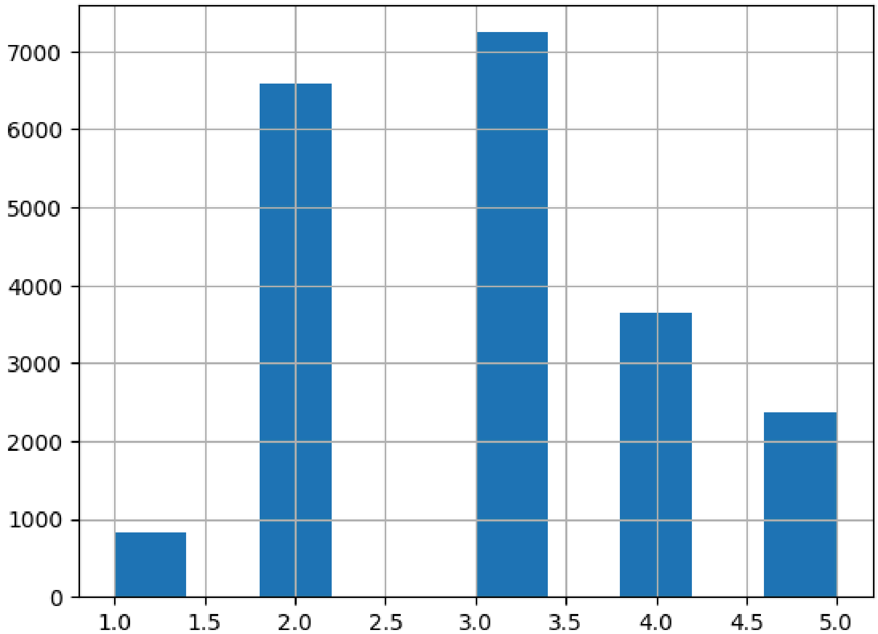
Out[ ]:

	median_income	income_cat
0	8.3252	5
1	8.3014	5
2	7.2574	5
3	5.6431	4
4	3.8462	3
...	...	...
20635	1.5603	2
20636	2.5568	2
20637	1.7000	2
20638	1.8672	2
20639	2.3886	2

20640 rows × 2 columns

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

Out[ ]: <Axes: >



Stratified Sampling

In [ ]:
```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_indexes, test_indexes in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_indexes]
    strat_test_set = housing.loc[test_indexes]

strat_train_set
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	706.0	
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	768.0	
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	300.0	
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	483.0	
20496	-118.70	34.28	27.0	3536.0	646.0	1837.0	580.0	
...	...	...	...	...	...	...	...	
15174	-117.07	33.03	14.0	6665.0	1231.0	2026.0	1001.0	
12661	-121.42	38.51	15.0	7901.0	1422.0	4769.0	1418.0	
19263	-122.72	38.44	48.0	707.0	166.0	458.0	172.0	
19140	-122.70	38.31	14.0	3155.0	580.0	1208.0	501.0	
19773	-122.14	39.97	27.0	1079.0	222.0	625.0	197.0	

16512 rows × 11 columns

In [ ]:
```
# percentage of categories in train set
highest = max([j for _,j in strat_train_set["income_cat"].value_counts().items()])
```

```
for i,j in strat_train_set["income_cat"].value_counts().items():
    print(str(i) + ":", str((j/highest)*100) + "%")
```

```
3: 100.0%
2: 90.9483503195716%
4: 50.28502332008983%
5: 32.64812575574365%
1: 11.349110381758507%
```

In [ ]:
```
# percentage of categories in test set
highest = max([j for _,j in strat_test_set["income_cat"].value_counts().items()])
for i,j in strat_test_set["income_cat"].value_counts().items():
    print(str(i) + ":", str((j/highest)*100) + "%")
```

```
3: 100.0%
2: 90.94678645473392%
4: 50.310988251554946%
5: 32.61921216309606%
1: 11.402902557014514%
```

Since both the train set and test set have the same number of categories shuffled randomly, the likelyhood of a Sampling Bias is significantly reduced.

You should remove the income_cat attribute so the data is back to its original state.

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

Visualize the Data

The goal is to go a little more in-depth into understanding of the data. First, make sure the test-set is left aside and only the training set is explored.
Note: If the training set is very large, you may want to sample an exploration set, to make manipulations easy and fast. In this case, the training set is fairly small so you can directly work on the full set.

In [ ]:
```
# Plotting latitudes and longitudes to see the location of all districts

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

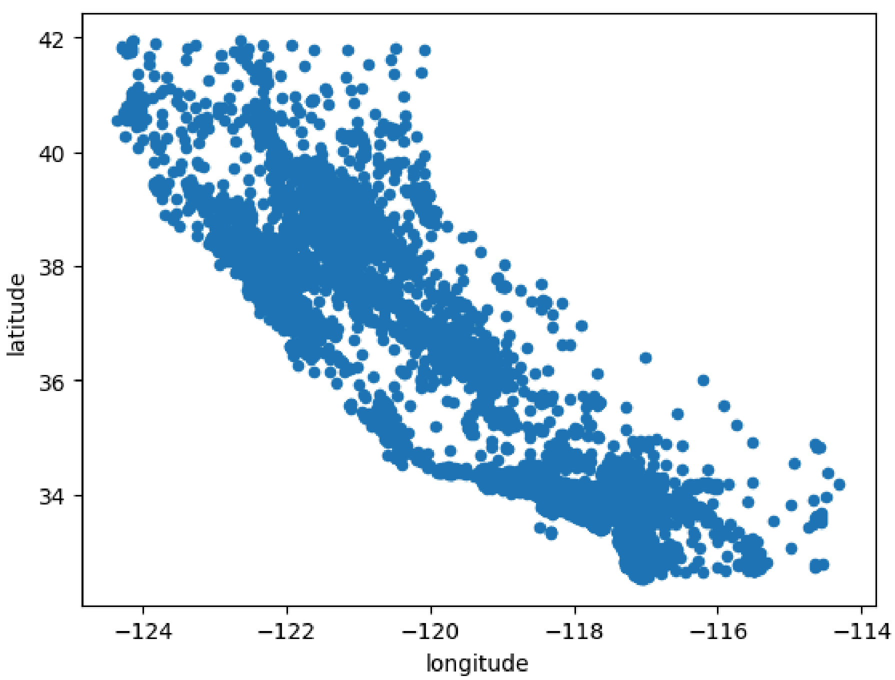
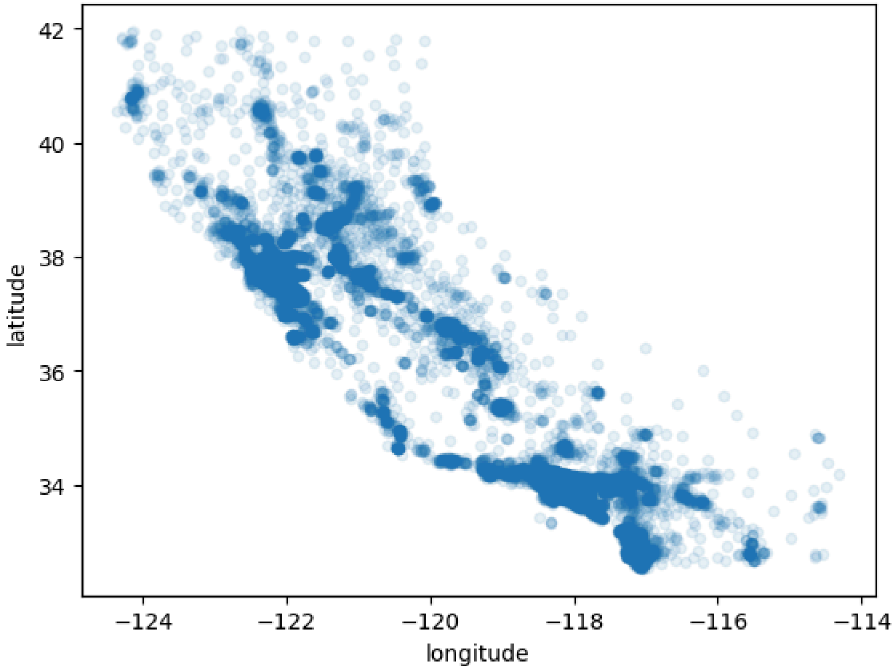


fig: scattered districts of California

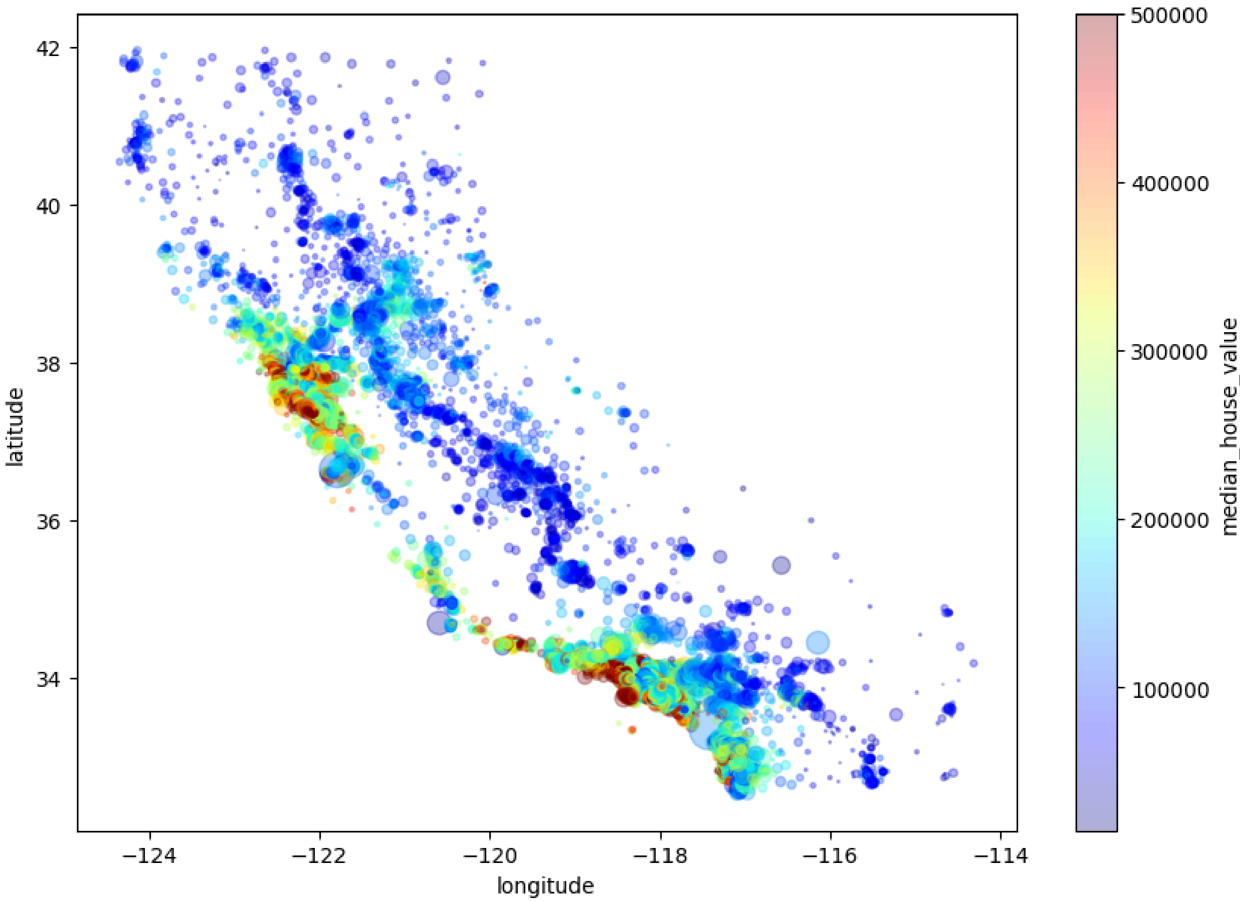
In [ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
plt.show()
```



highly densed areas can now be seen clearly.

Now let's visualize the population of each district in the state. The radius of the circles represents the district's population and the color represents the price (blue meaning more affordable and red being more expensive).

In [ ]:
```
housing.plot(
    kind="scatter",
    alpha=0.3,
    x="longitude",
    y="latitude",
    s=housing["population"]/100,
    c="median_house_value",
    cmap=plt.get_cmap("jet"),
    colorbar=True,
    figsize=(10,7)
)
plt.show()
```



Takeaways from the graph:

1. The housing prices are related to the location(eg, close to the ocean)
2. The housing prices are related to population density

Looking for Correlations

We can compute the standard correlation coefficient between every pair of attributes using the corr() method.

In [ ]:
```python
# housing_numeric_only = housing.select_dtypes(include=[np.number])
# corr_matrix = housing_numeric_only.corr()
# corr_matrix["median_house_value"].sort_values(ascending=False)

# (or)

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

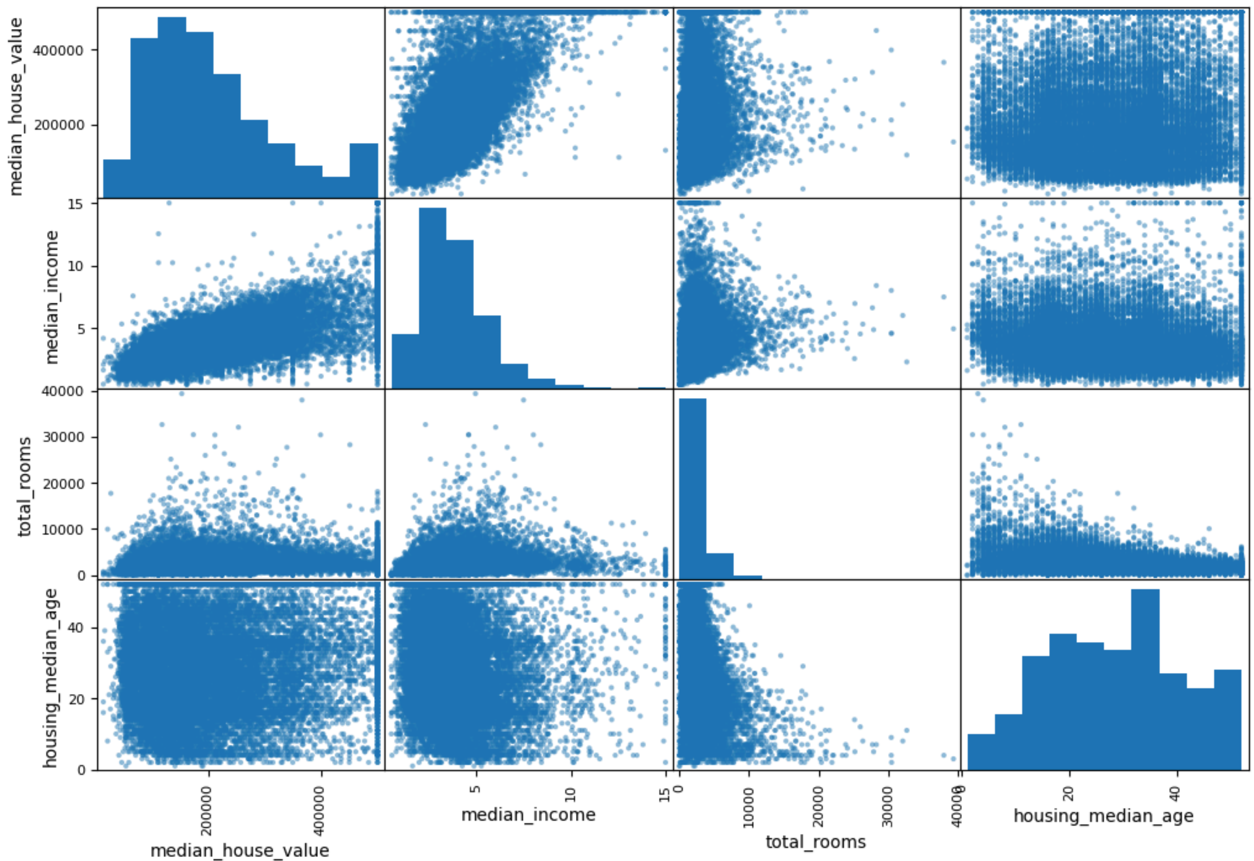
Out[ ]:
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value,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

These correlations co-efficients are with respect the the `median_house_value`. For example, the `median_house_value` tends to go up when the `median_income` goes up, because the correlation coefficient between the two is 0.688, which is pretty close to 1 and has a strong positive correlation.

Also, the `median_house_value` tends to go a little up the more you travel to the south because of the negative correlation between `median_house_value` and `latitude`.

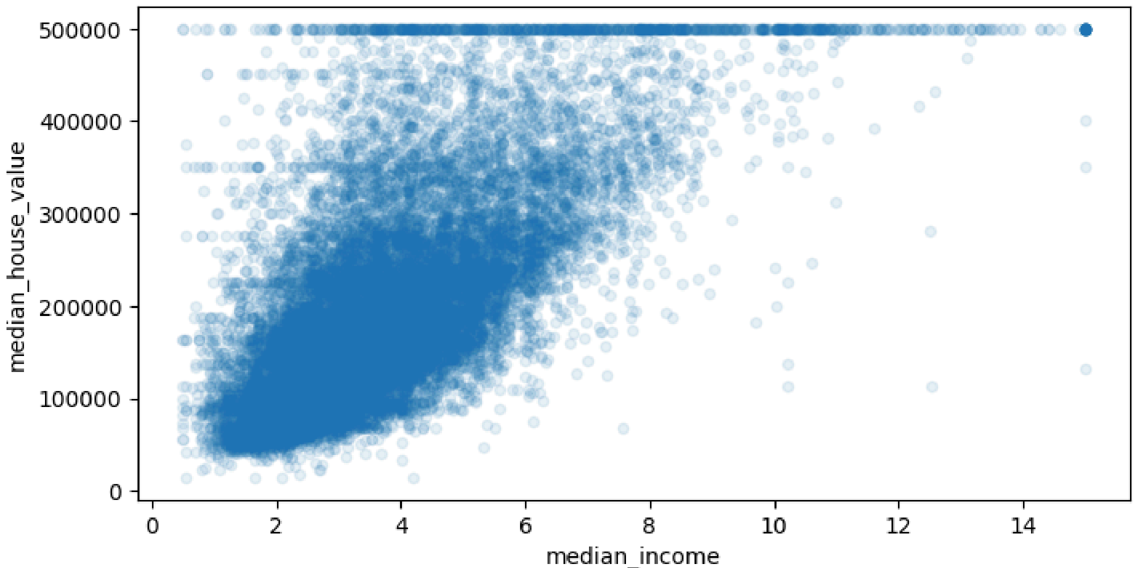
Another way is to use `pandas.plotting.scatter_matrix` function, which plots every numerical attribute against every other numerical attribute. This results in 11^2 = 121 correlations and 121 plots which won't fit in the graph. So we will only focus on a few promising attributes that seem most correlated with the `median_house_value`.

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

attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]

_ = scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```



Since the most useful attribute is the `median_income` attribute, we will focus on that.

In [ ]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1, figsize=(8,4))
plt.show()
```


500000
400000
300000
200000
100000
0
median_house_value
0
2
4
6
8
10
12
14
median_income