```
print("Hello World")
```

```
Hello World
```

```
import os
import tarfile
import urllib

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

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  \
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
```

Each row in this data represents one district.

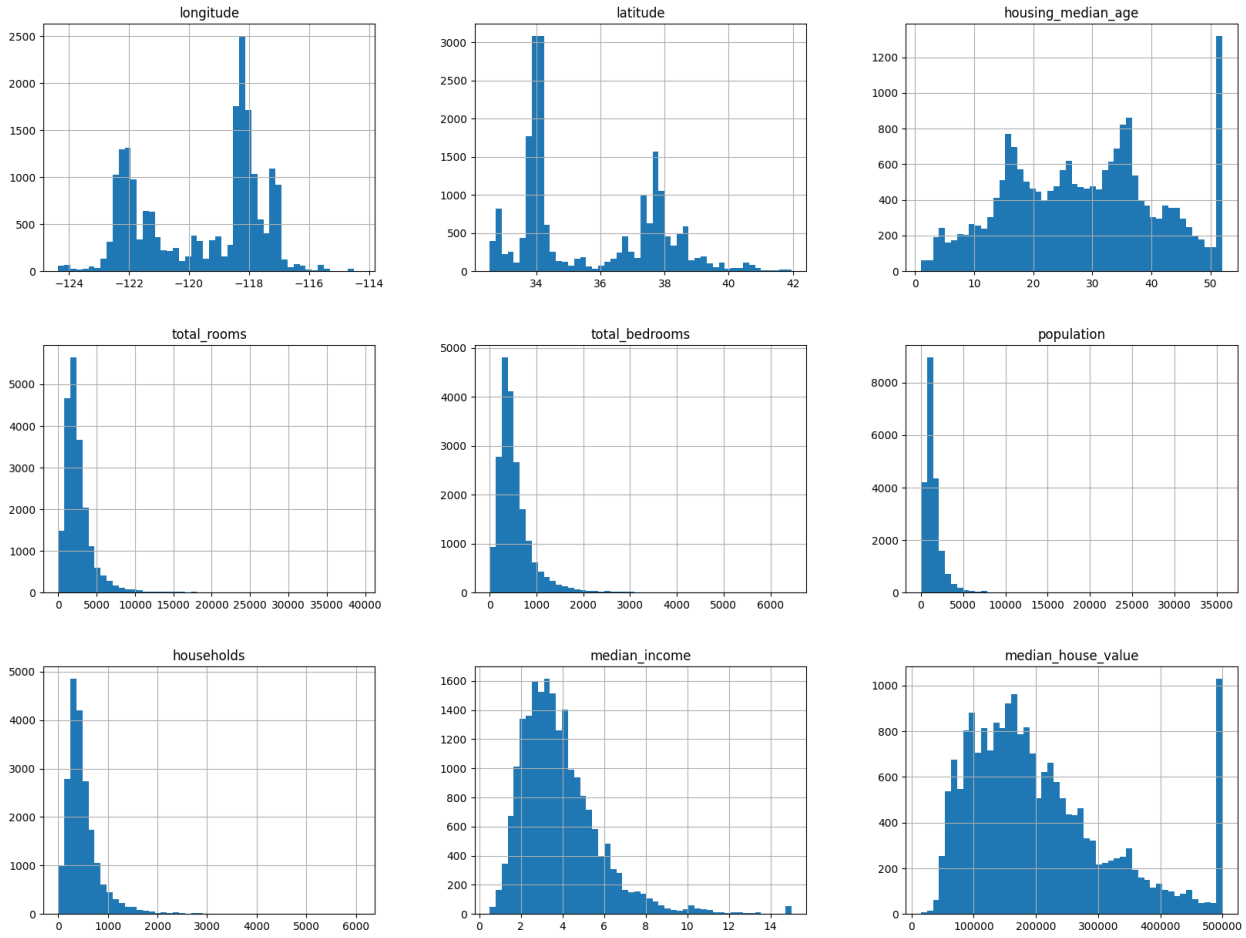
```
housing.info()

<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
```

207 districts aree missing the totat_bedrooms features as it is only 20433 ocean_proximity is the only attribute that is not numerical. It has repetitive values so can be a categorical attribute.

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

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

housing.describe()
#std stands for standard deviation

          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
```

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



1. For median income 3 actually means about $30,000. In the data.
2. housing median age and the median house value are also capped. Median house value can be a problem as it is the target value.
3. The data in many histograms is tail-heavy

Even when we don't know our data well yet we are creating a test dataset. This is to avoid data snooping bias. It is usually a simple process of separating 20% of the data or even less if the data set is huge.

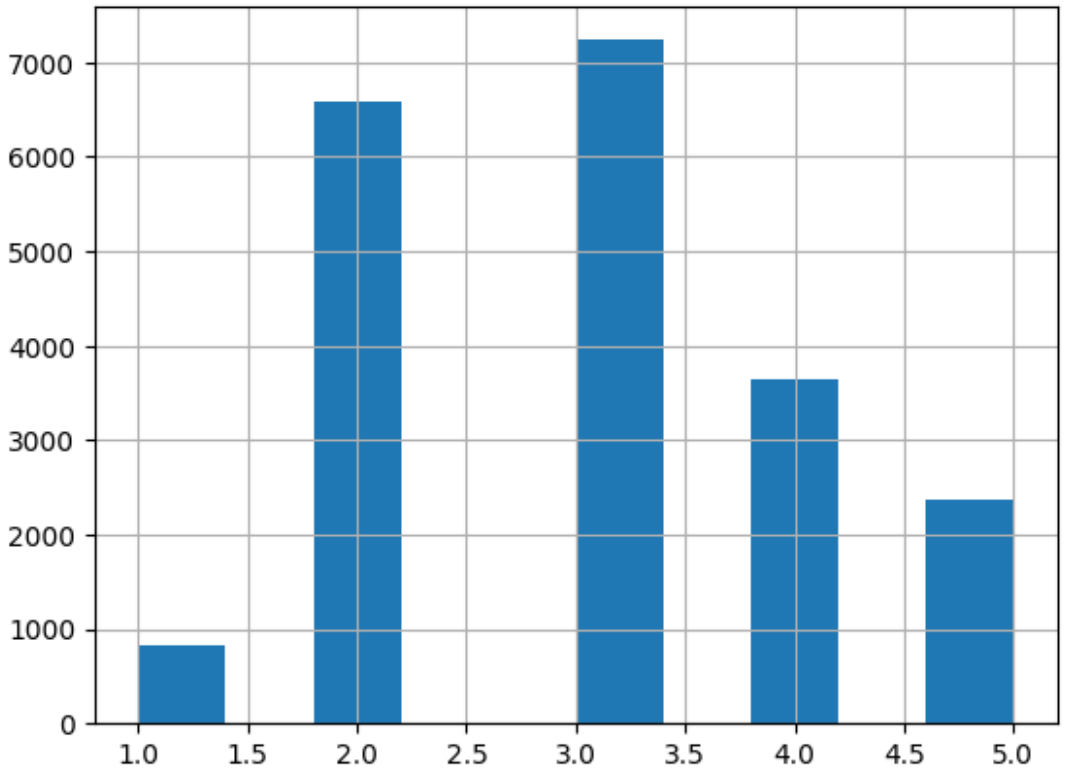
```
import numpy as np
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set))
print(len(test_set))
```

```
16512
4128
```

This works but is not a good idea as every time we run this we get a different trianing and testing dataset. And over time that becomes a problem as we will have the whole data in the training set atleast ones. Which is not what we will want to have.

```
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]

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")

from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2,
random_state=42)

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()
```

```
<Axes: >
```




```
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)
```

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

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