In [1]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

import matplotlib.pyplot as plt
```

In [2]:
```
housing = pd.read_csv(r"C:\Users\sempa\Documents\Machine_learning_Lab\housing.csv"
```

In [3]:
```
display(housing)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
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 [4]:
```
1 print(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
None
```

In [5]:
```
1 print(housing.ocean_proximity.value_counts())
```

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

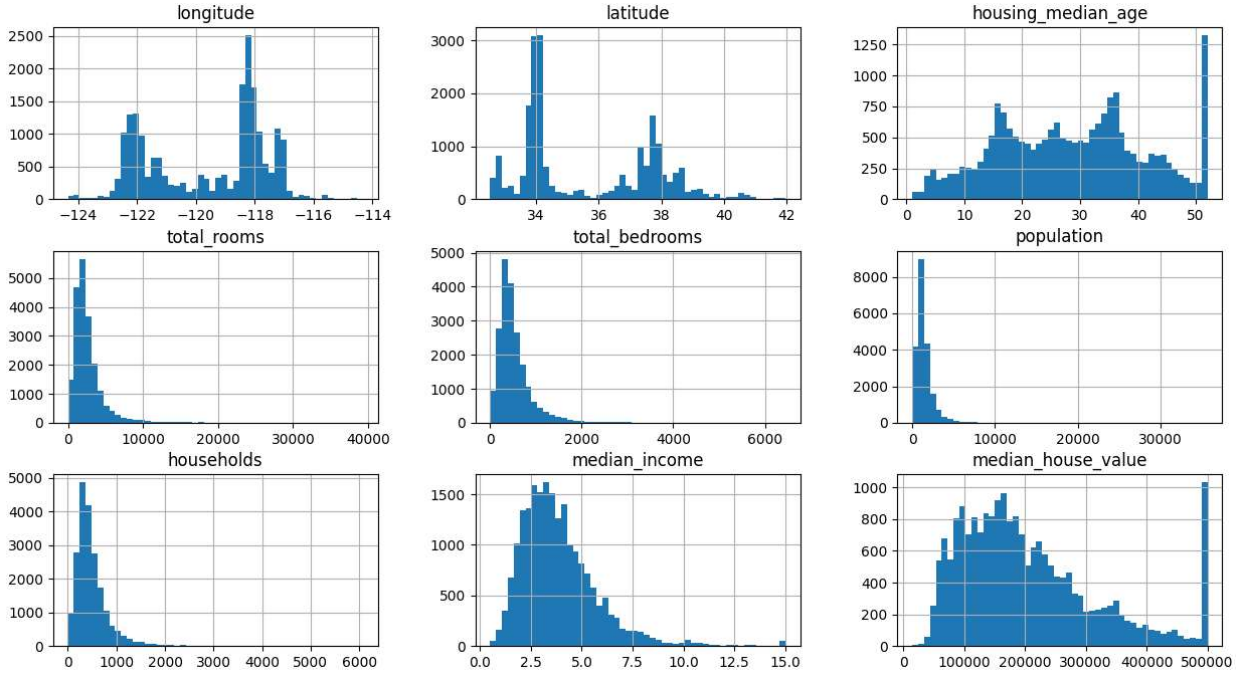
In [6]:
```
1 display(housing.describe())
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	ho
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	2064
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	49
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	38
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	28
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	40
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	60
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	608

In [7]:
```
1 print("There are", sum(housing.duplicated()), "duplicates in the datsset")
```

```
There are 0 duplicates in the datsset
```

In [8]:
```
housing.hist(bins=50, figsize=(15,8))
plt.show()
# figsize=(15, 8) sets the width of the figure to 15 inches and the height to 8 inc
```



creating test set

In [9]:
```
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

In [10]:
```
housing = train_set.drop("median_house_value", axis=1)
# housing_labels = train_set["median_house_value"].copy()
```

handling missing values

```
Though the missing values are there only in attribute 'total_bedrooms', but all
the numeric columns are made available to Imputer so that it can impute missing
values in any attributes in testing dataset.
```

In [11]:
```
# Instantiates imputer
imputer = SimpleImputer(strategy="median")

# Considers only numeric columns as the imputer works on numeric data
housing_num = housing.drop("ocean_proximity", axis=1)

# Fits and then Transforms the missing values in each column with learned median
X_train_num = imputer.fit_transform(housing_num)
```

Scaling Features

```
As machine learning algorithms don't work well with (numerical) attributes
having different scales, let's apply scaling transformations to numeric
attributes using StandardScaler.
```

In [12]:
```
# Instantiate standard scaler
std_scaler = StandardScaler()

# Fits and then Transforms to scale numeric attributes using standardization
X_train_num = std_scaler.fit_transform(X_train_num)
```

In [13]:
```
# print("Shape of the transformed dataset with only numerical attributes is",
#       X_train_num.shape)
```

Encoding Categorical Attributes

```
Encodes the only categorical attribute 'ocean_proximity' using One-Hot encoding
```

In [14]:
```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat = housing[["ocean_proximity"]]

# Fits and then Transforms the categorical values into one-hot-encoded columns
X_train_cat = cat_encoder.fit_transform(housing_cat)
```

In [15]:
```
print(X_train_cat)
```

```
[[0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0.]
 ...
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]]
```

In [16]:
```
print(cat_encoder.categories_)
```

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

Combining Transformed Data

In [17]:
```
X_train = np.append(X_train_num, X_train_cat, axis=1)
```

In [18]:
```
display(
    pd.DataFrame(
        X_train,
        index=housing_num.index,
#         columns=list(housing_num.columns)+list(cat_encoder.categories_[0])
    )
)
```

	0	1	2	3	4	5	6	7	8	9	10	
14196	1.172993	-1.350415	0.428537	1.570557	1.376799	1.081011	1.507507	0.379698	0.0	0.0	0.0	0
8267	1.268028	-1.378536	-1.473509	-0.809439	-0.900718	-0.643842	-0.878707	0.420068	0.0	0.0	0.0	0
17445	-1.352939	0.988349	-0.046974	1.994289	2.441082	1.363196	2.593828	-0.092320	0.0	0.0	0.0	1
14265	-1.127856	0.758691	-0.284730	0.646558	0.230833	0.661262	0.394820	0.682999	1.0	0.0	0.0	0
2271	1.793222	-1.083261	-1.632013	-1.117906	-1.181804	-1.203802	-1.255755	-1.255560	0.0	1.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
11284	-1.402957	1.082087	1.617317	-0.777706	-0.742156	-0.731143	-0.804879	-1.335305	0.0	0.0	0.0	1
11964	0.592779	-0.816108	0.507789	-0.400173	-0.499510	-0.613860	-0.496385	1.421304	1.0	0.0	0.0	0
5390	0.117604	0.304062	-0.997998	-0.005374	-0.026228	-0.309630	0.052048	-0.911522	0.0	1.0	0.0	0
860	1.187999	-0.727057	-0.522486	-0.078641	0.041040	0.122465	-0.016506	-0.634382	0.0	1.0	0.0	0
15795	0.352690	-0.661440	-0.522486	-0.655906	-0.811827	-0.665888	-0.804879	1.262077	1.0	0.0	0.0	0

16512 rows × 13 columns

In [ ]:
```

```

In [ ]:
```

```