Day 15

Machine Learning

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
import os

print(os.path.sep)
```

```
\
```

In [3]:

```


FILE_PATH = ".." +os.path.sep + "Python-Practice-code" +os.path.sep + "housing.csv"
housing = pd.read_csv(FILE_PATH)
```

In [4]:

```
type(housing)
```

Out[4]: pandas.core.frame.DataFrame

In [5]:

```
len(housing)
```

Out[5]: 20640

In [6]:

```
housing.head()
```

Out[6]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

In [7]:

```
housing[:5]
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

In [8]:
```python
housing.head(10)
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	

In [9]:
```python
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 [10]:
```python
len(housing[housing.total_bedrooms.isnull()])
```

Out[10]: 207

In [11]:
```python
len(housing[housing['total_bedrooms'].isna()])
```

Out[11]: 207

In [12]:
```python
housing.ocean_proximity.unique()
```

Out[12]:
```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

In [13]:
```
1 housing.describe()
```

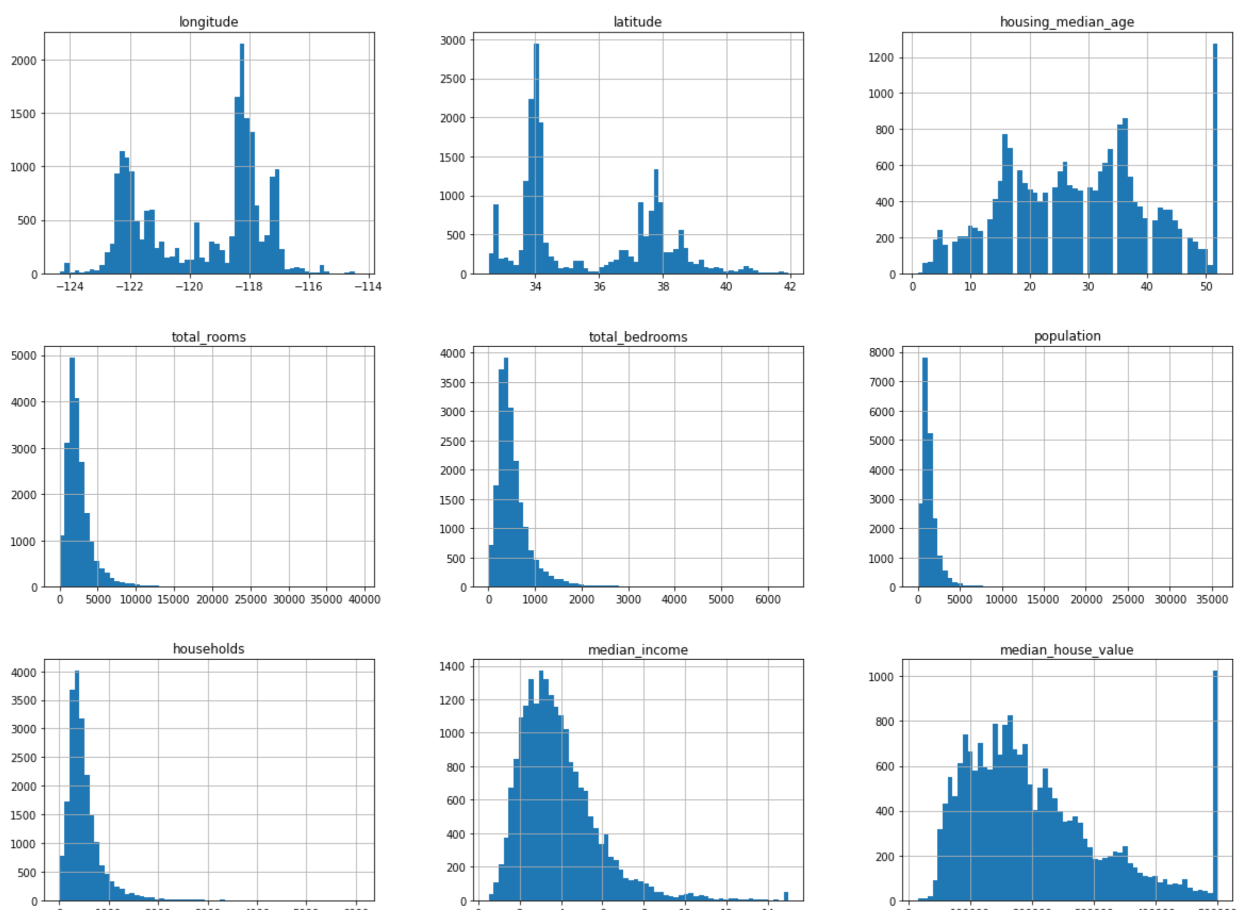
Out[13]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.

In [14]:
```
1 housing.hist(bins=60,figsize=(20,15))
```

```
d:\python-practice-code\venv\lib\site-packages\pandas\plotting\_matplotlib\tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use ax.get_subplotspec().is_first
_col() instead.
  if ax.is_first_col():
```

Out[14]:
```
array([[<AxesSubplot:title={'center':'longitude'}>,
        <AxesSubplot:title={'center':'latitude'}>,
        <AxesSubplot:title={'center':'housing_median_age'}>],
       [<AxesSubplot:title={'center':'total_rooms'}>,
        <AxesSubplot:title={'center':'total_bedrooms'}>,
        <AxesSubplot:title={'center':'population'}>],
       [<AxesSubplot:title={'center':'households'}>,
        <AxesSubplot:title={'center':'median_income'}>,
        <AxesSubplot:title={'center':'median_house_value'}>]],
      dtype=object)
```



In [15]:
```
#(housing.median_income) # housing['median_income']   ## Two notations for selecting a particular attribute from a dataframe
```

In [16]:
```
np.min(housing.median_income)
```

Out[16]: 0.4999

In [17]:
```
np.max(housing['median_income'])
```

Out[17]: 15.0001

In [18]:
```
from sklearn.model_selection import StratifiedShuffleSplit
```

In [19]:
```
corr = housing.corr()
```

In [20]:
```
corr
```

Out[20]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	mediar
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069608	0.099773	0.055310	-
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066983	-0.108785	-0.071035	-
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.320451	-0.296244	-0.302916	-
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930380	0.857126	0.918484	
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000000	0.877747	0.979728	-
population	0.099773	-0.108785	-0.296244	0.857126	0.877747	1.000000	0.907222	
households	0.055310	-0.071035	-0.302916	0.918484	0.979728	0.907222	1.000000	
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007723	0.004834	0.013033	
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049686	-0.024650	0.065843	

In [21]:
```
#import seaborn as sns
```

In [22]:
```
#plt.figure(figsize=(15,8))
#sns.heatmap(corr,annot=True,cmap='ocean')
```

In [23]:
```
corr.median_house_value.sort_values(ascending=False)
```

Out[23]:
```
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
```

In [24]:
```
len(housing.median_income.unique())
```

Out[24]: 12928

In [25]:
```
len(np.ceil(housing.median_income).unique())
```

Out[25]: 16

In [26]:
```
#housing['income_cat1']=np.ceil(housing.median_income)
```

In [27]:
```
#housing.income_cat1.value_counts().sort_index(ascending=True)
```

In [28]:
```
housing['income_cat']=np.ceil(housing.median_income / 1.5)
```

In [29]:
```
housing.income_cat.value_counts().sort_index(ascending=True)
```

Out[29]:
```
1.0      822
2.0     6581
3.0     7236
4.0     3639
5.0     1423
6.0      532
7.0      189
8.0      105
9.0       50
10.0      14
11.0      49
Name: income_cat, dtype: int64
```

In [30]:
```
# Capping
housing.income_cat.where(housing.income_cat < 5,5.0, inplace=True)
```

In [31]:
```
housing.income_cat.value_counts().sort_index(ascending=True)
```

Out[31]:
```
1.0     822
2.0    6581
3.0    7236
4.0    3639
5.0    2362
Name: income_cat, dtype: int64
```

In [32]:
```
housing.income_cat.value_counts().sort_index(ascending=True) / len(housing)
```

Out[32]:
```
1.0    0.039826
2.0    0.318847
3.0    0.350581
4.0    0.176308
5.0    0.114438
Name: income_cat, dtype: float64
```

In [33]:
```
from sklearn.model_selection import StratifiedShuffleSplit
```

In [34]:
```
type(housing.income_cat)
```

Out[34]:
```
pandas.core.series.Series
```

In [35]:
```
strat_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
```

In [36]:
```
#strat_split.split(housing,housing.income_cat) will create a generator which will generate 2 index train_ix and test_ix
# we are receiving their value by for loop because a generator's value is accepted via a for loop.
# iloc takes the index and train_ix and test_ix are actually list whereas strat_train and strat_test are actually dataframe
for train_ix, test_ix in strat_split.split(housing, housing.income_cat):
    strat_train = housing.iloc[train_ix]
    strat_test = housing.iloc[test_ix]
```

In [37]:
```
strat_train.income_cat.value_counts().sort_index(ascending=True) / len(strat_train)
```

Out[37]:
```
1.0    0.039850
2.0    0.318859
3.0    0.350594
4.0    0.176296
5.0    0.114402
Name: income_cat, dtype: float64
```

In [38]:
```
strat_test.income_cat.value_counts().sort_index(ascending=True) / len(strat_test)
```

Out[38]:
```
1.0    0.039729
2.0    0.318798
3.0    0.350533
4.0    0.176357
5.0    0.114583
Name: income_cat, dtype: float64
```

In [39]:
```
#as inplace =True is not written here that means the drop command is not being performed on the actual dataset otherway
#we can say that the categorical attribute income_cat is being dropped after taking the whole data frame temporarily in
#another data frame. axis=1 is written to make the dataframe understand that the drop function will work at axis = 1
#otherwise dataframe understands that the function will be performed at axis=0
#strat_train and strat_test these two datasets are being formed in this way which will be used for
#learning purpose of the machine.
strat_train = strat_train.drop('income_cat', axis=1)
strat_test = strat_test.drop('income_cat',axis=1)
```

In [40]:
```
strat_train
```

Out[40]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	mediai
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	
...	...	...	...	...	...	...	...	...	
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312	
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682	
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723	
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625	
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750	

16512 rows × 10 columns

In [41]:
```
strat_test
```

Out[41]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	mediai
5241	-118.39	34.12	29.0	6447.0	1012.0	2184.0	960.0	8.2816	
10970	-117.86	33.77	39.0	4159.0	655.0	1669.0	651.0	4.6111	
20351	-119.05	34.21	27.0	4357.0	926.0	2110.0	876.0	3.0119	
6568	-118.15	34.20	52.0	1786.0	306.0	1018.0	322.0	4.1518	
13285	-117.68	34.07	32.0	1775.0	314.0	1067.0	302.0	4.0375	
...	...	...	...	...	...	...	...	...	
20519	-121.53	38.58	33.0	4988.0	1169.0	2414.0	1075.0	1.9728	
17430	-120.44	34.65	30.0	2265.0	512.0	1402.0	471.0	1.9750	
4019	-118.49	34.18	31.0	3073.0	674.0	1486.0	684.0	4.8984	
12107	-117.32	33.99	27.0	5464.0	850.0	2400.0	836.0	4.7110	
2398	-118.91	36.79	19.0	1616.0	324.0	187.0	80.0	3.7857	

4128 rows × 10 columns

In [42]:
```
x_train = strat_train.drop('median_house_value', axis=1)
y_train = strat_train.median_house_value
x_test = strat_test.drop('median_house_value', axis=1)
y_test = strat_test.median_house_value
```

In [43]:
```
x_train
```

Out[43]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEA
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<
...	...	...	...	...	...	...	...	...	
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312	
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682	
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723	
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625	<
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750	I

16512 rows × 9 columns

In [44]:
```
y_train
```

Out[44]:
```
17606    286600.0
18632    340600.0
14650    196900.0
3230      46300.0
3555     254500.0
           ...
6563     240200.0
12053    113000.0
13908     97800.0
11159    225900.0
15775    500001.0
Name: median_house_value, Length: 16512, dtype: float64
```

In [45]:
```
x_test
```

Out[45]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_
5241	-118.39	34.12	29.0	6447.0	1012.0	2184.0	960.0	8.2816	<1
10970	-117.86	33.77	39.0	4159.0	655.0	1669.0	651.0	4.6111	<1
20351	-119.05	34.21	27.0	4357.0	926.0	2110.0	876.0	3.0119	<1
6568	-118.15	34.20	52.0	1786.0	306.0	1018.0	322.0	4.1518	
13285	-117.68	34.07	32.0	1775.0	314.0	1067.0	302.0	4.0375	
...	...	...	...	...	...	...	...	...	
20519	-121.53	38.58	33.0	4988.0	1169.0	2414.0	1075.0	1.9728	
17430	-120.44	34.65	30.0	2265.0	512.0	1402.0	471.0	1.9750	NEA
4019	-118.49	34.18	31.0	3073.0	674.0	1486.0	684.0	4.8984	<1
12107	-117.32	33.99	27.0	5464.0	850.0	2400.0	836.0	4.7110	
2398	-118.91	36.79	19.0	1616.0	324.0	187.0	80.0	3.7857	

4128 rows × 9 columns

In [46]:
```
y_test
```

Out[46]:
```
5241     500001.0
10970    240300.0
20351    218200.0
6568     182100.0
13285    121300.0
           ...
20519     76400.0
17430    134000.0
4019     311700.0
12107    133500.0
2398      78600.0
Name: median_house_value, Length: 4128, dtype: float64
```

Data Preprocessing

In [47]:
```
x_train_num=x_train.drop('ocean_proximity',axis=1)
x_train_cat=x_train['ocean_proximity']
```

In [48]:
```
x_train_num
```

Out[48]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347
...	...	...	...	...	...	...	...	...
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750

16512 rows × 8 columns

In [49]:
```
x_train_cat
```

Out[49]:
```
17606     <1H OCEAN
18632     <1H OCEAN
14650    NEAR OCEAN
3230         INLAND
3555      <1H OCEAN
            ...
6563         INLAND
12053        INLAND
13908        INLAND
11159     <1H OCEAN
15775      NEAR BAY
Name: ocean_proximity, Length: 16512, dtype: object
```

Imputation

In [50]:
```
x_train_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 17606 to 15775
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           16512 non-null  float64
 1   latitude            16512 non-null  float64
 2   housing_median_age  16512 non-null  float64
 3   total_rooms         16512 non-null  float64
 4   total_bedrooms      16354 non-null  float64
 5   population          16512 non-null  float64
 6   households          16512 non-null  float64
 7   median_income       16512 non-null  float64
dtypes: float64(8)
memory usage: 1.1 MB
```

In [51]:
```
# total_bedrooms has some missing values which have to be tackled.
```

In [52]:
```
16512-16354
```

Out[52]:
```
158
```

In [53]:
```
from sklearn.impute import SimpleImputer
```

In [54]:
```
imputer = SimpleImputer(strategy='median')
```

In [55]:
```
imputer.fit(x_train_num)
```

Out[55]:
```
SimpleImputer(strategy='median')
```

In [56]:
```
imputer.statistics_
```

Out[56]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

In [57]:
```
x_train_num_ndarray = imputer.transform(x_train_num)
```

In [58]:
```
x_train_num_ndarray[:5]
```

Out[58]:
```
array([[-1.2189e+02,  3.7290e+01,  3.8000e+01,  1.5680e+03,  3.5100e+02,
         7.1000e+02,  3.3900e+02,  2.7042e+00],
       [-1.2193e+02,  3.7050e+01,  1.4000e+01,  6.7900e+02,  1.0800e+02,
         3.0600e+02,  1.1300e+02,  6.4214e+00],
       [-1.1720e+02,  3.2770e+01,  3.1000e+01,  1.9520e+03,  4.7100e+02,
         9.3600e+02,  4.6200e+02,  2.8621e+00],
       [-1.1961e+02,  3.6310e+01,  2.5000e+01,  1.8470e+03,  3.7100e+02,
         1.4600e+03,  3.5300e+02,  1.8839e+00],
       [-1.1859e+02,  3.4230e+01,  1.7000e+01,  6.5920e+03,  1.5250e+03,
         4.4590e+03,  1.4630e+03,  3.0347e+00]])
```

In [59]:
```
x_train_num_df = pd.DataFrame(x_train_num_ndarray, columns = x_train_num.columns)
```

In [60]:
```
x_train_num_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16512 entries, 0 to 16511
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           16512 non-null  float64
 1   latitude            16512 non-null  float64
 2   housing_median_age  16512 non-null  float64
 3   total_rooms         16512 non-null  float64
 4   total_bedrooms      16512 non-null  float64
 5   population          16512 non-null  float64
 6   households          16512 non-null  float64
 7   median_income       16512 non-null  float64
dtypes: float64(8)
memory usage: 1.0 MB
```

Scaling

In [61]:
```
from sklearn.preprocessing import StandardScaler
```

In [62]:
```
scaler = StandardScaler()
```

In [63]:
```
x_train_num_ndarray = scaler.fit_transform(x_train_num_df)
```

In [64]:
```
x_train_num_ndarray[:5]
```

Out[64]:
```
array([[-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
        -0.63621141, -0.42069842, -0.61493744],
       [-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
        -0.99833135, -1.02222705,  1.33645936],
       [ 1.18684903, -1.34218285,  0.18664186, -0.31365989, -0.15334458,
        -0.43363936, -0.0933178 , -0.5320456 ],
       [-0.01706767,  0.31357576, -0.29052016, -0.36276217, -0.39675594,
         0.03604096, -0.38343559, -1.04556555],
       [ 0.49247384, -0.65929936, -0.92673619,  1.85619316,  2.41221109,
         2.72415407,  2.57097492, -0.44143679]])
```

For handling Categorical Attribute

Ordinal Encoder

In [65]:
```
x_train_cat.unique()
```

Out[65]:
```
array(['<1H OCEAN', 'NEAR OCEAN', 'INLAND', 'NEAR BAY', 'ISLAND'],
      dtype=object)
```

In [66]:
```
from sklearn.preprocessing import OrdinalEncoder
```

In [67]:
```
ord_encoder = OrdinalEncoder()
```

In [68]:
```
x_train_cat_ndarray = ord_encoder.fit_transform(x_train_cat.values.reshape(-1, 1))
```

In [69]:
```
x_train_cat_ndarray[:5]
```

Out[69]:
```
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.]])
```

In [70]:
```
x_train_cat[:5]
```

Out[70]:
```
17606     <1H OCEAN
18632     <1H OCEAN
14650    NEAR OCEAN
3230         INLAND
3555      <1H OCEAN
Name: ocean_proximity, dtype: object
```

In [71]:
```
ord_encoder.categories_
```

Out[71]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

In [72]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [73]:
```
hot_encoder = OneHotEncoder(sparse=False)
```

In [74]:
```
x_train_cat_hot_encoded_ndarray = hot_encoder.fit_transform(x_train_cat.values.reshape(-1, 1))
```

In [75]:
```
x_train_cat_hot_encoded_ndarray
```

Out[75]:
```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

Combine the Preprocessed Results

In [76]:
```
x_train_final = np.c_[x_train_num_ndarray, x_train_cat_hot_encoded_ndarray]
```

In [77]:
```
x_train_final.shape
```

Out[77]: (16512, 13)

In [78]:
```
from sklearn.linear_model import LinearRegression
```

In [79]:
```
lin_reg = LinearRegression()
```

In [80]:
```
lin_reg.fit(x_train_final, y_train)
```

Out[80]: LinearRegression()

In [81]:
```
y_hat = lin_reg.predict(x_train_final)
```

In [82]:
```
y_hat[0]
```

Out[82]: 212480.0

In [83]:
```
y_train[0]
```

Out[83]: 452600.0

Custom Transformer

In [84]:
```
from sklearn.base import BaseEstimator, TransformerMixin
```

In [85]:
```
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self,attrs=[]):
        self.attrs = attrs
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attrs].values
    '''
    def fit_transform(self, X, y=None):
        return self.fit(X, y).transform(X)
    '''
```

In [86]:
```
ds = DataFrameSelector(attrs = x_train.columns[:-1])
ds.fit_transform(x_train).shape[1]
```

Out[86]: 8

In [87]:
```
x_train.columns[:-1]
```

Out[87]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income'],
      dtype='object')
```

In [88]:
```
x_train[['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income']].values
```

Out[88]:
```
array([[-121.89  ,   37.29  ,   38.    , ...,  710.    ,  339.    ,
           2.7042],
       [-121.93  ,   37.05  ,   14.    , ...,  306.    ,  113.    ,
           6.4214],
       [-117.2   ,   32.77  ,   31.    , ...,  936.    ,  462.    ,
           2.8621],
       ...,
       [-116.4   ,   34.09  ,    9.    , ..., 2098.    ,  765.    ,
           3.2723],
       [-118.01  ,   33.82  ,   31.    , ..., 1356.    ,  356.    ,
           4.0625],
       [-122.45  ,   37.77  ,   52.    , ..., 1269.    ,  639.    ,
           3.575 ]])
```

Pipelining

In [89]:
```
from sklearn.pipeline import Pipeline, FeatureUnion
```

In [90]:
```
num_pipe = Pipeline([
        ('selector', DataFrameSelector(attrs=x_train.columns[:-1])),
        ('imputer' , SimpleImputer(strategy='median')),
        ('scaler' , StandardScaler())])

cat_pipe = Pipeline([
        ('selector', DataFrameSelector(attrs=[x_train.columns[-1]])),
        ('hot_encoder', OneHotEncoder(sparse=False))
])

full_pipe = FeatureUnion([
        ('num_pipe', num_pipe),
        ('cat_pipe', cat_pipe)])
```

In [91]:
```
x_train_final =full_pipe.fit_transform(x_train)
```

In [92]:
```
x_test_final = full_pipe.fit_transform(x_test)
```

In [93]:
```
Lin_reg = LinearRegression()
```

In [94]:
```
Lin_reg.fit(x_train_final, y_train)
```

Out[94]:
```
LinearRegression()
```

In [95]:
```
y_pred_train = Lin_reg.predict(x_train_final)
```

In [96]:
```
from sklearn.metrics import mean_squared_error
```

In [97]:
```
mse_train = mean_squared_error(y_train, y_pred_train)
```

In [98]:
```
rmse_train = np.sqrt(mse_train)
```

In [99]:
```
rmse_train
```

Out[99]: 69054.94433245908

In [100]:
```
y_pred_test = Lin_reg.predict(x_test_final)
```

In [101]:
```
mse_test = mean_squared_error(y_test, y_pred_test)
```

In [102]:
```
rmse_test = np.sqrt(mse_test)
```

In [103]:
```
rmse_test
```

Out[103]: 67368.05288226946

In [104]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [105]:
```
tree_reg = DecisionTreeRegressor()
```

In [106]:
```
tree_reg.fit(x_train_final, y_train)
```

Out[106]: DecisionTreeRegressor()

In [107]:
```
y_pred_train_tree = tree_reg.predict(x_train_final)
```

In [108]:
```
mse = mean_squared_error(y_train, y_pred_train_tree)
```

In [109]:
```
mse
```

Out[109]: 0.0

In [110]:
```
y_pred_test_tree = tree_reg.predict(x_test_final)
```

In [111]:
```
mse = mean_squared_error(y_test, y_pred_test_tree)
```

In [112]:
```
rmse = np.sqrt(mse)
```

In [113]:
```
rmse
```

Out[113]: 74340.08257413437

In [114]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [115]:
```
forest_reg = RandomForestRegressor()
```

In [116]:
```
forest_reg.fit(x_train_final, y_train)
```

Out[116]: RandomForestRegressor()

In [117]:
```
y_pred_train = forest_reg.predict(x_train_final)
```

In [118]:
```
rmse = np.sqrt(mean_squared_error(y_train, y_pred_train))
```

In [119]:
```
rmse
```

Out[119]:
```
18309.90304151225
```

In [120]:
```
y_pred_test = forest_reg.predict(x_test_final)
```

In [121]:
```
rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
```

In [122]:
```
rmse
```

Out[122]:
```
54457.38614560365
```

k-Fold Cross Validation

In [123]:
```
from sklearn.model_selection import cross_val_score
```

In [126]:
```
mses = -cross_val_score(lin_reg, x_train_final, y_train, cv = 10, scoring='neg_mean_squared_error')
```

In [127]:
```
mses
```

Out[127]:
```
array([4.54755748e+09, 4.54152781e+09, 4.69813534e+09, 5.57389458e+09,
       4.66688012e+09, 5.13865443e+09, 4.27207885e+09, 4.70206808e+09,
       5.25226443e+09, 4.63734690e+09])
```

Performance Tuning

In [129]:
```
from sklearn.model_selection import GridSearchCV
```

In [133]:
```
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)
```

In [134]:
```
grid_search.fit(x_train_final, y_train)
```

Out[134]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

In [135]:
```
grid_search.best_params_
```

Out[135]:
```
{'max_features': 8, 'n_estimators': 30}
```

In [136]:
```
final_model = grid_search.best_params_
```

In [ ]:
```

```