In [1]:

```
import pandas as pd

housing = pd.read_csv("/content/drive/MyDrive/housing.csv")
housing.head()
```

Out[1]:

	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 [2]:

```
housing.describe()
```

Out[2]:

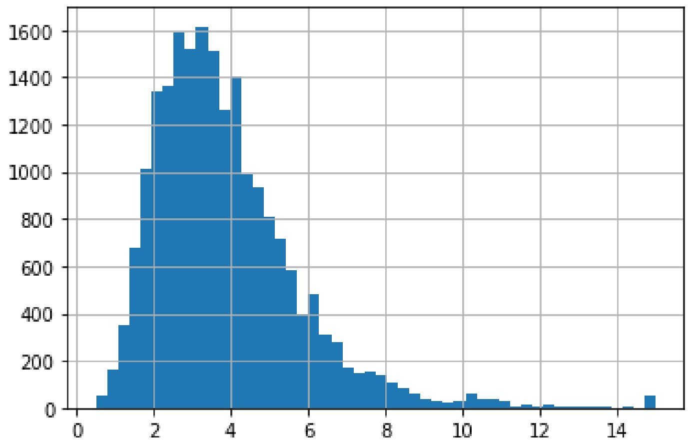
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.
std	2.003532	2.135952	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 [3]:

```
housing['median_income'].hist(bins=50)
```

Out[3]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f59259a2390>

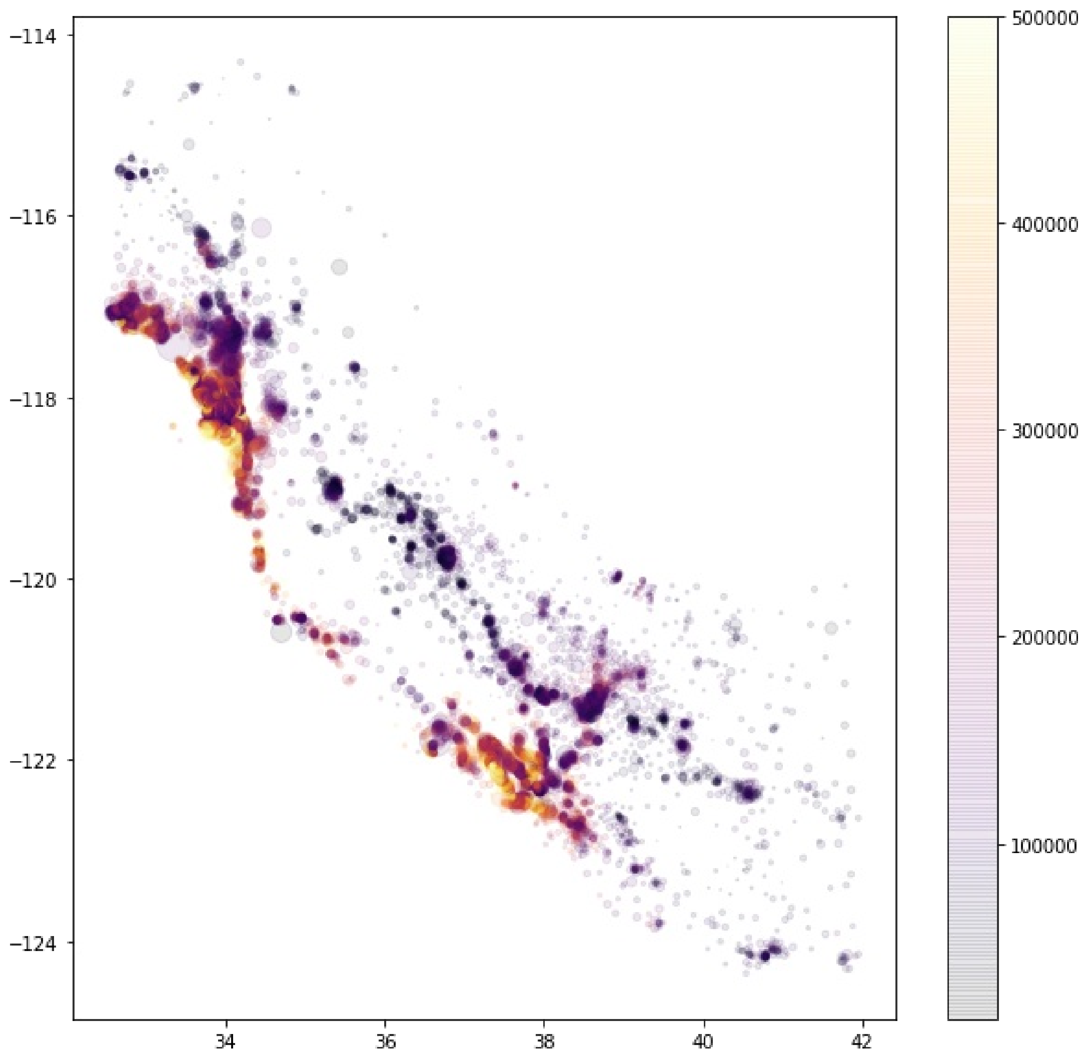


In [4]:

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.scatter(x=housing['latitude'],y=housing['longitude'],alpha=0.1,s=housing['population
']/100,c=housing['median_house_value'],cmap='inferno')
plt.colorbar()
```

Out[4]:

```
<matplotlib.colorbar.Colorbar at 0x7f5924120fd0>
```



In [5]:

```python
corr_matrix = housing.corr()

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

Out[5]:

```
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
```

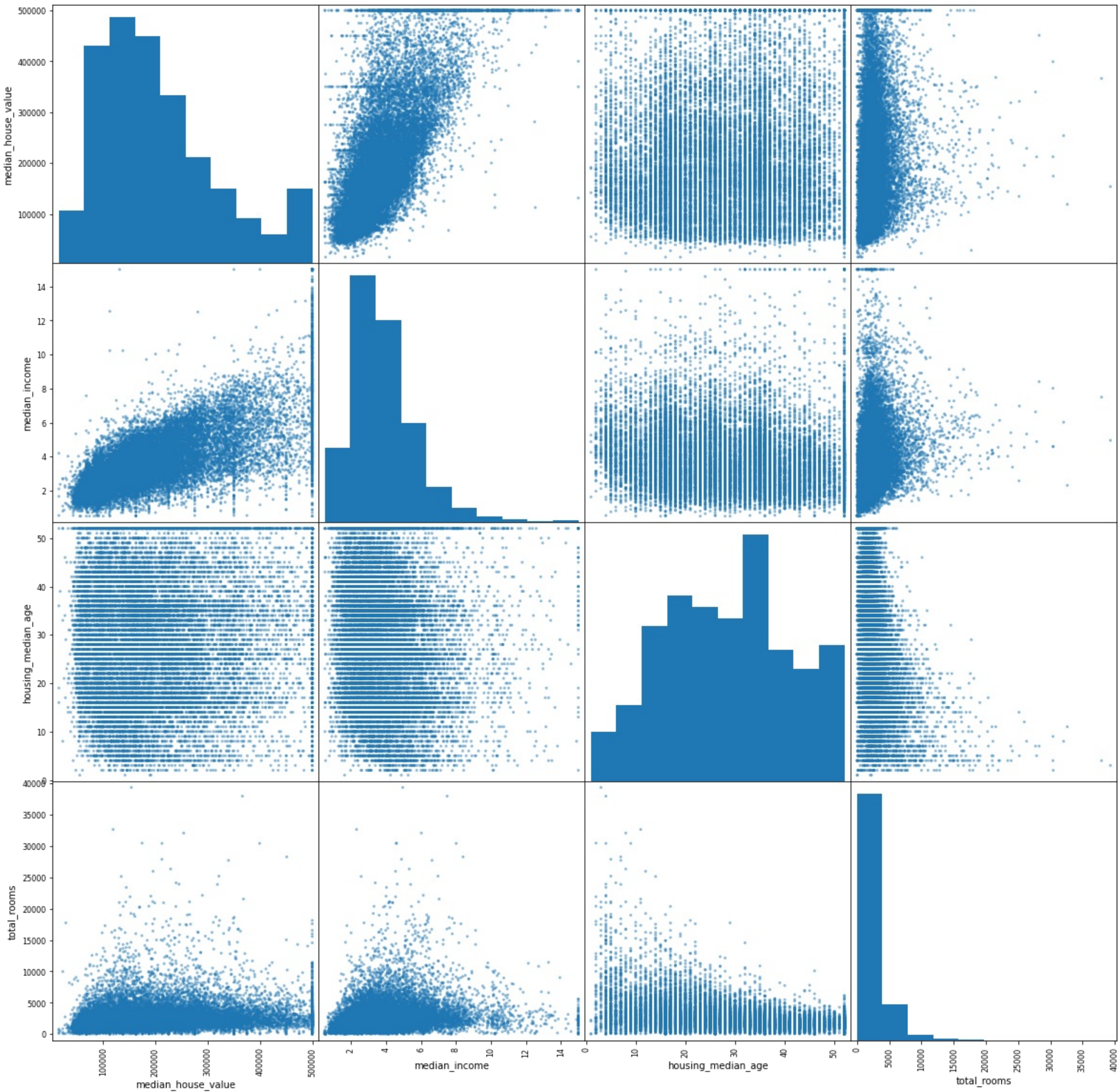
In [6]:

```python
from pandas.plotting import scatter_matrix

attributes = ['median_house_value','median_income','housing_median_age','total_rooms']
scatter_matrix(housing[attributes],figsize=(20,20))
```

Out[6]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f59240c77f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f59228374a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5922869710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f592281c978>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f59227cdbe0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5922783e48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f59227430f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f59226f8320>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f59226f8390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f59226dd828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5922693a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5922646cf8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f59225fdf60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f59225ba208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5922570470>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f59225a26d8>]],
      dtype=object)
```

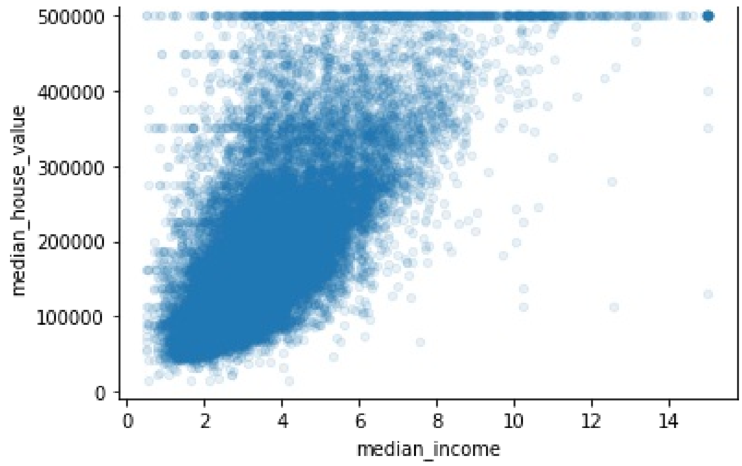


In [7]:

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

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f59222ddda0>
```



In [8]:

```
housing.columns
```

Out[8]:

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

In [9]:

```
housing['population_per_household'] = housing['population'] / housing['households']
housing['bedrooms_per_room'] = housing['total_bedrooms'] / housing['total_rooms']
housing['rooms_per_household'] = housing['total_rooms'] / housing['households']
```

In [10]:

```
corr_matrix = housing.corr()

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

Out[10]:

```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

In [11]:

```
housing.head()
```

Out[11]:

	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 [12]:

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

In [13]:

```
print(len(train_set))
print(len(test_set))
```

```
16512
4128
```

In [14]:

```
import numpy as np
housing['income_cat'] = np.ceil(housing['median_income']/1.5)
housing['income_cat'].where(housing['income_cat']<5,5.0,inplace=True)
```

In [15]:

```
housing.head()
```

Out[15]:

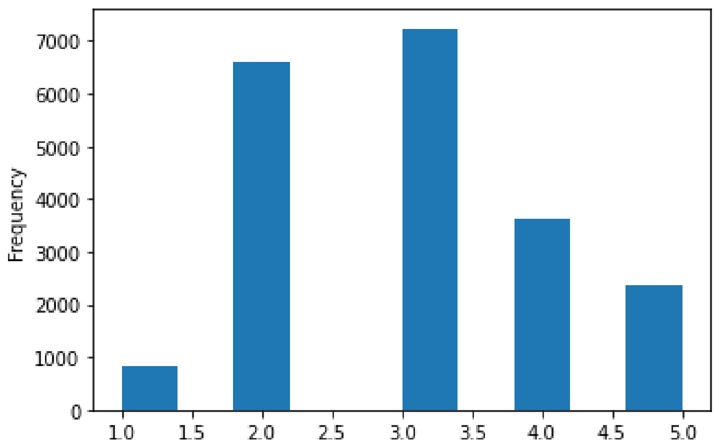
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	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 [16]:

```
housing['income_cat'].plot(kind='hist',)
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f59127a6358>
```



In [17]:

```
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]
```

In [18]:

```
housing['income_cat'].value_counts() / len(housing['income_cat'])
```

```
housing["income_cat"].value_counts() / len(housing["income_cat"])
```

Out[18]:

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

In [19]:

```
for set_ in (strat_train_set, strat_test_set):

  set_.drop("income_cat", axis=1, inplace=True)
```

In [20]:

```
housing = strat_train_set.copy()
```

In [21]:

```
housing.describe()
```

Out[21]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000	16512.000000	16512.
mean	-119.575834	35.639577	28.653101	2622.728319	534.973890	1419.790819	497.060380	3.
std	2.001860	2.138058	12.574726	2138.458419	412.699041	1115.686241	375.720845	1.
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000	2.000000	0.
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000	279.000000	2.
50%	-118.510000	34.260000	29.000000	2119.500000	433.000000	1164.000000	408.000000	3.
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.250000	602.000000	4.
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000	5358.000000	15.

In [22]:

```
housing = strat_train_set.drop(["median_house_value",'population_per_household','bedrooms
_per_room','rooms_per_household'], axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

In [23]:

```
from sklearn.impute import SimpleImputer

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

Out[23]:

```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

In [24]:

```
imputer.statistics_
```

Out[24]:

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

In [25]:

```
X = imputer.transform(housing_num)
```

In [26]:

```
housing_tr = pd.DataFrame(data=X,columns=housing_num.columns)
housing_tr.head()
```

Out[26]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042
1	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214
2	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621
3	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839
4	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347

Label Encoding

In [27]:

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
housing_cat = housing['ocean_proximity']
housing_cat_encoded = encoder.fit_transform(housing_cat)
print(housing_cat_encoded.shape)
print(encoder.classes_)
```

```
(16512,)
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

Now applying One Hot Encoding

In [28]:

```
from sklearn.preprocessing import OneHotEncoder
onehotenco = OneHotEncoder()
housing_cat_1hot = onehotenco.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot
```

Out[28]:

```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
 with 16512 stored elements in Compressed Sparse Row format>
```

In [29]:

```
housing_tr['ocean_proximity'] = housing_cat_encoded
```

In [30]:

```
housing_tr.head()
```

Out[30]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_prox
0	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	
1	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	
2	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	
3	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	
4	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	

SELECT AND TRAIN A MODEL

In [31]:

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_tr,housing_labels)
```

Out[31]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [32]:

```
some_data = housing_tr.iloc[:5]
some_label = housing_labels[:5]


print("prediction",lin_reg.predict(some_data))
print("label's",list(some_label))
```

```
prediction [207901.47824371 323216.63913327 205102.81901373  75423.92526847
 188676.68780642]
label's [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

In [33]:

```
from sklearn.metrics import mean_squared_error

# housing_labels = housing_labels.reshape(-1,1)
housing_predictions = lin_reg.predict(housing_tr)
lin_mse = mean_squared_error(housing_predictions,housing_labels)
lin_mse = np.sqrt(lin_mse)
print(lin_mse)
```

```
69957.9936286799
```

In [34]:

```
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_tr, housing_labels)
housing_predictions = tree_reg.predict(housing_tr)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[34]:

```
0.0
```

In [35]:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg,housing_tr,housing_labels,scoring='neg_mean_squared_err
or',cv=10)
tree_rmse_score = np.sqrt(-scores)

def display_score(scores):
  print('Scores:', scores)
  print('Mean:', scores.mean())
  print('Standard Deviation', scores.std())

display_score(tree_rmse_score)
```

```
Scores: [65589.79204859 71283.27542878 70717.66887942 72235.16386952
 66418.63883654 74438.29528469 68702.62576639 70137.14813444
 72324.04794659 69890.16179496]
```

```
Mean: 70173.68179899339
Standard Deviation 2567.994316412716
```

In [36]:

```python
lin_score = cross_val_score(lin_reg,housing_tr,housing_labels, scoring='neg_mean_squared_error',cv=10)
lin_rmse_score = np.sqrt(-lin_score)
display_score(lin_rmse_score)
```

```
Scores: [68230.55806124 68520.93622918 69600.91124405 74990.90394949
 68974.73419338 72198.27981692 66607.90832448 69745.60718443
 73514.29993282 68943.8776868 ]
Mean: 70132.8016622785
Standard Deviation 2472.4661735125324
```

In [37]:

```python
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()
forest_reg.fit(housing_tr, housing_labels)
```

Out[37]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [38]:

```python
forest_score = cross_val_score(forest_reg,housing_tr,housing_labels, scoring='neg_mean_squared_error',cv=10)
forest_rmse_score = np.sqrt(-forest_score)
display_score(forest_rmse_score)
```

```
Scores: [47822.23905053 46766.25819041 50142.31518706 51395.03252716
 49990.2402097  53559.27403843 48386.59358769 50832.50422872
 52584.05782371 50592.12823831]
Mean: 50207.06430817236
Standard Deviation 1993.23814669634
```

In [39]:

```python
housing_predictions = forest_reg.predict(housing_tr)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[39]:

```
18628.591256239313
```

In [40]:

```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')

grid_search.fit(housing_tr,housing_labels)
```

Out[40]:

```
GridSearchCV(cv=5, error_score=nan,
```

```
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [41]:

```
grid_search.best_params_
```

Out[41]:

```
{'max_features': 4, 'n_estimators': 30}
```

In [42]:

```
grid_search.best_estimator_
```

Out[42]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=4, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=30, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [43]:

```
cvres = grid_search.cv_results_
cvres_df = pd.DataFrame(cvres)
cvres_df = cvres_df[["mean_test_score","params"]]
cvres_df['mean_test_score']= -cvres_df["mean_test_score"]
```

In [44]:

```
cvres_df
```

Out[44]:

	mean_test_score	params
0	3.855828e+09	{'max_features': 2, 'n_estimators': 3}
1	2.957060e+09	{'max_features': 2, 'n_estimators': 10}
2	2.708003e+09	{'max_features': 2, 'n_estimators': 30}
3	3.446072e+09	{'max_features': 4, 'n_estimators': 3}
4	2.752787e+09	{'max_features': 4, 'n_estimators': 10}
5	2.537503e+09	{'max_features': 4, 'n_estimators': 30}
6	3.436985e+09	{'max_features': 6, 'n_estimators': 3}
7	2.738078e+09	{'max_features': 6, 'n_estimators': 10}
8	2.539674e+09	{'max_features': 6, 'n_estimators': 30}

	mean_test_score	params
9	[illegible]	{'max_features': 8, 'n_estimators': 3}
10	2.783666e+09	{'max_features': 8, 'n_estimators': 10}
11	2.601937e+09	{'max_features': 8, 'n_estimators': 30}
12	3.715371e+09	{'bootstrap': False, 'max_features': 2, 'n_est...
13	2.811944e+09	{'bootstrap': False, 'max_features': 2, 'n_est...
14	3.406680e+09	{'bootstrap': False, 'max_features': 3, 'n_est...
15	2.700748e+09	{'bootstrap': False, 'max_features': 3, 'n_est...
16	3.229060e+09	{'bootstrap': False, 'max_features': 4, 'n_est...
17	2.658038e+09	{'bootstrap': False, 'max_features': 4, 'n_est...

In [45]:

```
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()

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
X_test_cat = X_test['ocean_proximity']
X_test_cat_encoded = encoder.fit_transform(X_test_cat)
X_test['ocean_proximity'] = X_test_cat_encoded
for i in X_test.columns:
  X_test[i].fillna(X_test[i].median(),inplace=True)

X_test.drop(columns=['population_per_household','bedrooms_per_room','rooms_per_household'
],inplace=True)
final_predictions = final_model.predict(X_test)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
print(final_rmse)
```

```
47862.778264449844
```

In [46]:

```
## 47269 is the final rmse value
```

In [46]:

```

```