ॐ श्री गणेशाय नम:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 %matplotlib inline
```

```
1 housing = pd.read_excel("housing.xlsx")
2 housing.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY

Dataset Description:

Field	Type	Description
longitude	(Signed numeric - float)	Longitude value for the block in California, USA
latitude	(numeric - float)	Latitude value for the block in California, USA
housing_median_age	(numeric-int)	Median age of the house in the block
total_rooms	(numeric-int)	Count of the total number of rooms (excluding bedrooms) in all houses in the block
total_rooms	(numeric-int)	Count of the total number of rooms (excluding bedrooms) in all houses in the block
total_bedrooms	(numeric - float)	Count of the total number of bedrooms in all houses in the block
population	(numeric-float)	Count of the total number of bedrooms in all houses in the block

Field	Type	Description
households	(numeric - int)	Count of the total number of population in the block
median_income	(numeric - float)	Median of the total household income of all the houses in the block
ocean_proximity	(numeric - int)	Median of the household prices of all the houses in the use_value block

```
1 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  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   ocean_proximity     20640 non-null  object 
 9   median_house_value  20640 non-null  int64  
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
```

- There are 20640 nun-null instances & 'total_bedrooms' has only 20433 non-null values (207 values missing)

```
1 for item in housing.columns:
2     print(item,":", housing[item].nunique())
```

```
longitude : 844
latitude : 862
housing_median_age : 52
total_rooms : 5926
total_bedrooms : 1923
population : 3888
```

```
households : 1815
median_income : 12928
ocean_proximity : 5
median_house_value : 3842
```

```
1 housing['ocean_proximity'].value_counts()
```

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

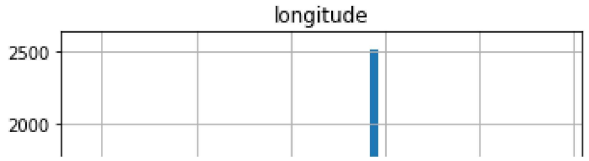
- ocean_proximity is a categorical variable

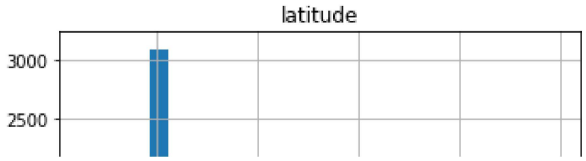
```
1 housing.describe()
```

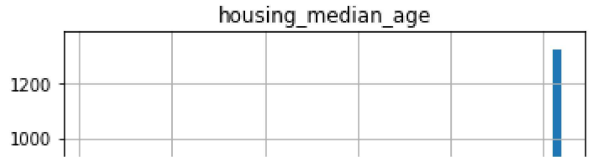
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	m
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	

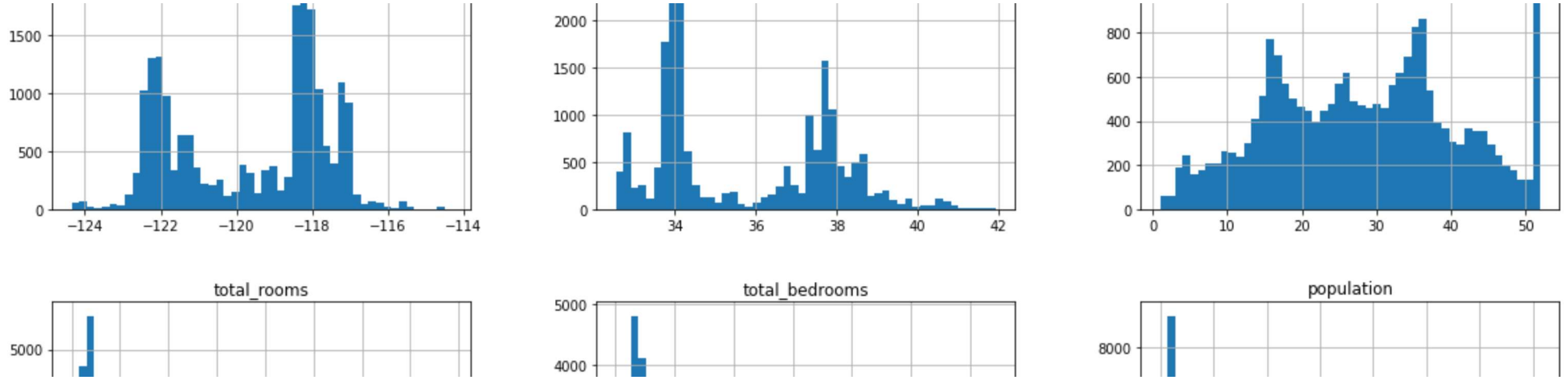
- **Plot the histogram to check the variation of each feature**

```
1 housing.hist(bins=50, figsize=(20, 15))
2 plt.show()
```





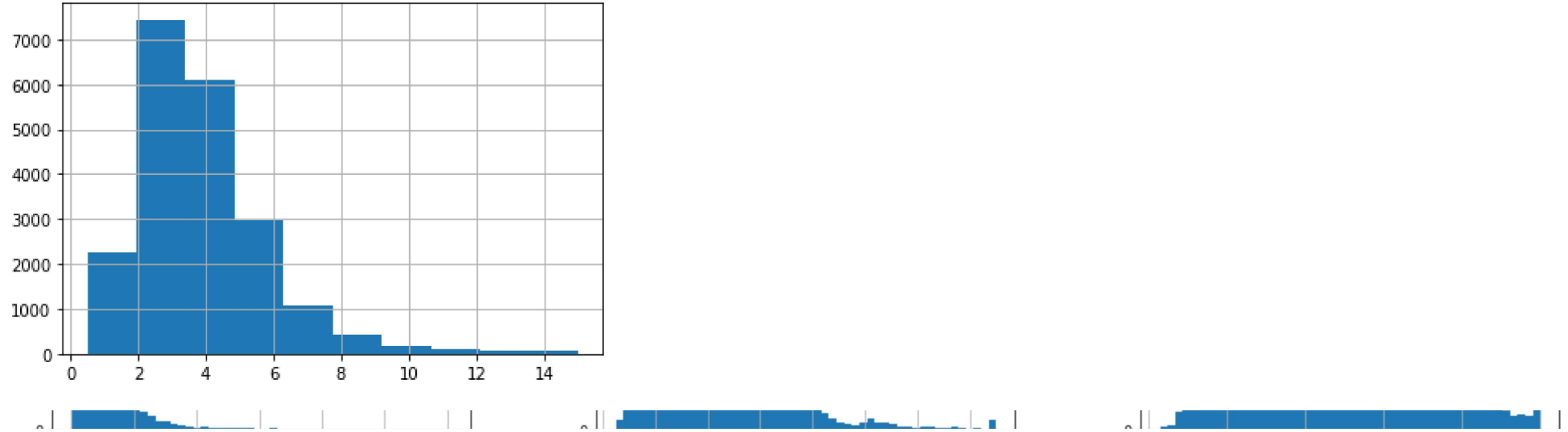




- **median income looks like an imp feature**

```
1 housing['median_income'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f26cecbed50>
```



- **Dividing the income category into 5 groups for balace the distribution of data**

```
1 housing['income_cat'] = np.ceil(housing['median_income'] / 1.5)
```

```
print(housing['income_cat'].unique())
# putting everything above 5th category as 5th category
housing['income_cat'].where(housing['income_cat'] < 5, other=5.0, inplace=True)
housing['income_cat'].unique()
```

```
[ 6.  5.  4.  3.  2.  1.  8.  7.  9. 11. 10.]
array([5., 4., 3., 2., 1.])
```

- **Split the data into train & test & check the distribution proportion**

```
from sklearn.model_selection import train_test_split
```

```
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=53)
```

- StratifiedShuffleSplit is a combination of both ShuffleSplit and StratifiedKFold. Using Stratified Shuffle Split the proportion of distribution of class labels is almost even between train and test dataset.

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=53)

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_train_set["income_cat"].value_counts() / len(strat_train_set)
```

```
3.0    0.350594
2.0    0.318859
4.0    0.176296
5.0    0.114462
1.0    0.039789
Name: income_cat, dtype: float64
```

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

```
3.0    0.350533
2.0    0.318798
4.0    0.176357
5.0    0.114341
1.0    0.039971
Name: income_cat, dtype: float64
```

```
def icatogary_proportions(data):
    return data["income_cat"].value_counts()/len(data)
```

```
comparing_props = pd.DataFrame({"Overall": icatogary_proportions(housing),
                                "Random sampling(Train Set)" :icatogary_proportions(train_set),
                                "Random sampling(Test Set)" :icatogary_proportions(test_set),
                                "strat. sampling(Train Set)":icatogary_proportions(strat_train_set),
                                "strat. sampling(Test Set)":icatogary_proportions(strat_test_set)
                               }).sort_index()

comparing_props["% random train error"] = 100 * comparing_props["Random sampling(Train Set)"] / comparing_props["Overall"] - 100
comparing_props["% random test error"] = 100 * comparing_props["Random sampling(Test Set)"] / comparing_props["Overall"] - 100
comparing_props["% strat. train error"] = 100 * comparing_props["strat. sampling(Train Set)"] / comparing_props["Overall"] - 100
comparing_props["% strat. test error"] = 100 * comparing_props["strat. sampling(Test Set)"] / comparing_props["Overall"] - 100

comparing_props
```

	Overall	Random sampling(Train Set)	Random sampling(Test Set)	strat. sampling(Train Set)	strat. sampling(Test Set)	% random train error	% random test error	% strat. train error	% strat. test error
1.0	0.039826	0.041546	0.032946	0.039780	0.039971	4.318735	17.274939	0.091241	0.364964

1.0	0.039826	0.041546	0.032946	0.039789	0.039971	4.318735	-17.274939	-0.091241	0.364964
2.0	0.318847	0.320494	0.312258	0.318859	0.318798	0.516639	-2.066555	0.003799	-0.015195
3.0	0.350581	0.347808	0.361676	0.350594	0.350533	-0.791183	3.164732	0.003455	-0.013820

as seen above the proportions stratified sampling has given better proportions.

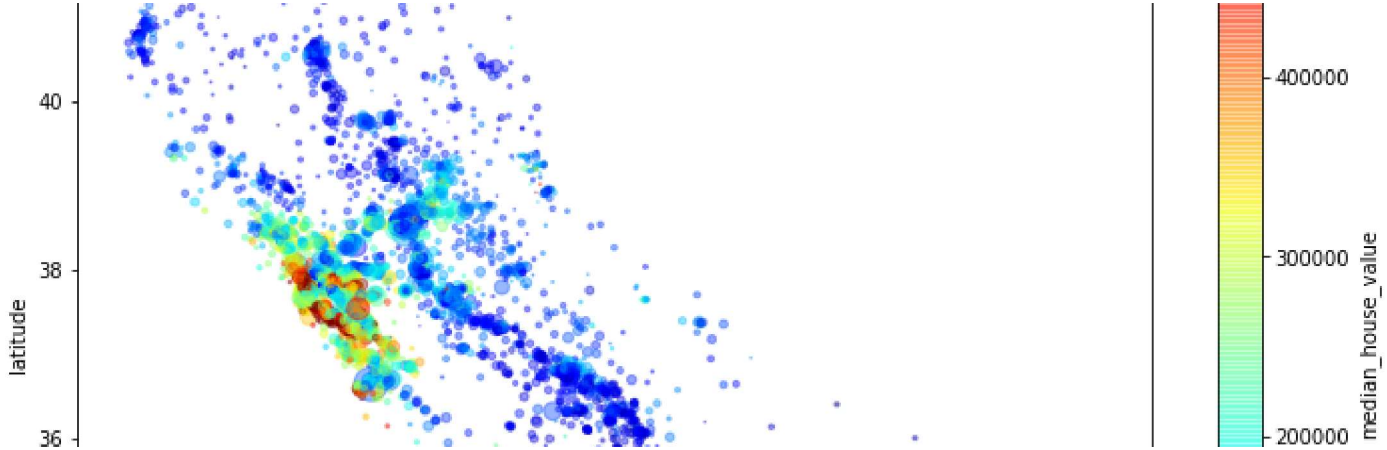
[why stratified?] : because the feature-space are less and also because its a mid-sized dataset & we don't want to miss out any class

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

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

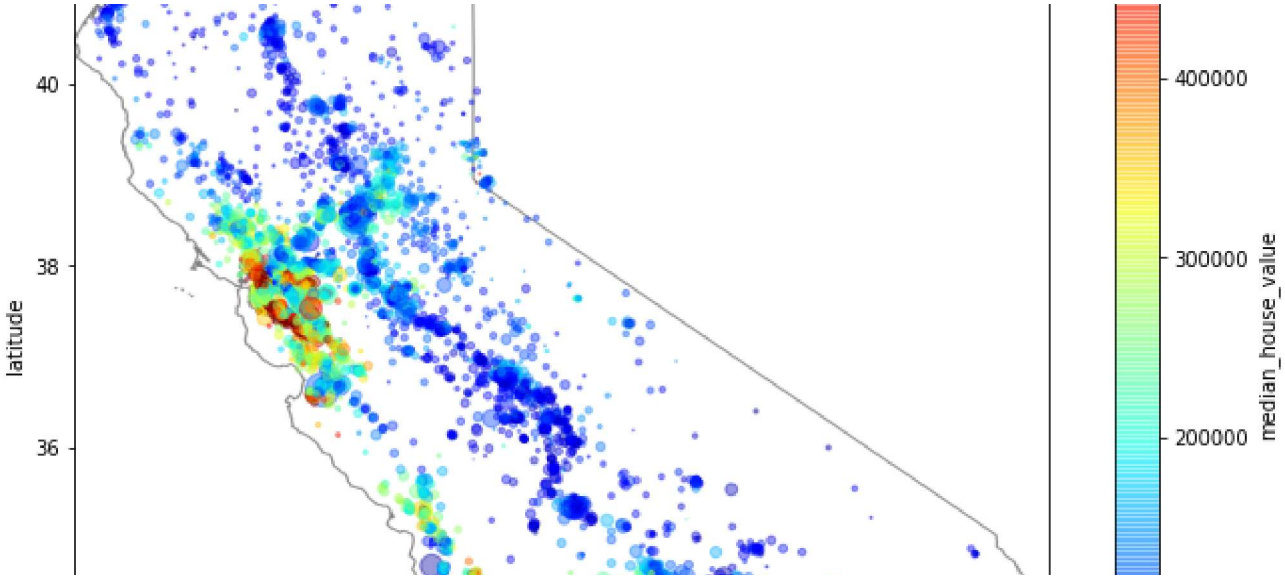
```
1 housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
2              s=housing['population']/100, label="population", figsize=(12,8),
3              c="median_house_value", cmap=plt.get_cmap("jet"), sharex=False)
4
5 plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f26d04b2a50>
```




```
1 import matplotlib.image as mping
2
3 ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
4             s=housing['population']/100, label="population", figsize=(12,8),
5             c="median_house_value", cmap=plt.get_cmap("jet"), sharex=False)
6
7 # Load the png image
8 california_img= mping.imread("california.png")
9 plt.imshow(california_img, extent=[-124.53, -113.8, 32.45, 42.05], alpha=0.5, cmap=plt.get_cmap("jet"))
10
11 plt.legend()
12
```

```
<matplotlib.legend.Legend at 0x7f26d020f9d0>
```



- **Correlation**

Correlation defines -1(less correlated) and 1(highly correlated)

```
1 Correlation = housing.corr()
2 Correlation
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_incom
longitude	1.000000	-0.924177	-0.108571	0.040877	0.066663	0.096814	0.053117	-0.01392
latitude	-0.924177	1.000000	0.011189	-0.032883	-0.064236	-0.106824	-0.069185	-0.08045

latitude	-0.024174	1.000000	-0.011173	0.032883	0.064250	0.105824	0.099773	-0.9994
housing_median_age	-0.108571	0.011189	1.000000	-0.361303	-0.320676	-0.295214	-0.303272	-0.11954
total_rooms	0.040877	0.032883	0.361303	1.000000	0.931166	0.854233	0.919604	0.20017

```
1 Correlation["median_house_value"].sort_values(ascending=False)
```

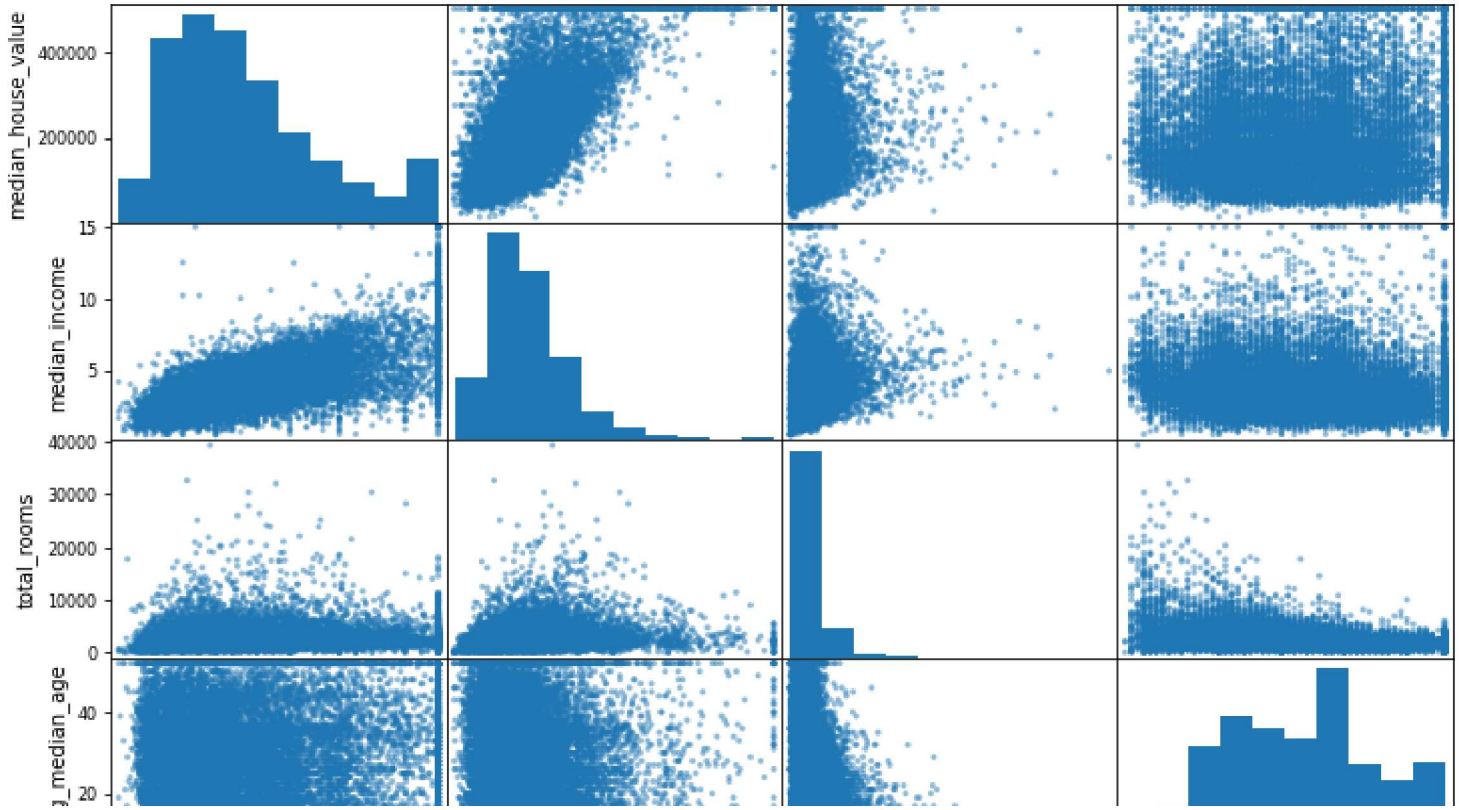
```
median_house_value    1.000000
median_income         0.684453
total_rooms           0.136386
housing_median_age    0.109590
households            0.068493
total_bedrooms        0.052798
population           -0.023337
longitude            -0.044110
latitude             -0.145566
Name: median_house_value, dtype: float64
```

- We can check one by one with other feature also. but we can see from above median_house_value is highly correlated with median_income, remaining are very less.

```
1 imp_attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]
2
3 from pandas.plotting import scatter_matrix
4
5 scatter_matrix(housing[imp_attributes], figsize=(12, 8))
```

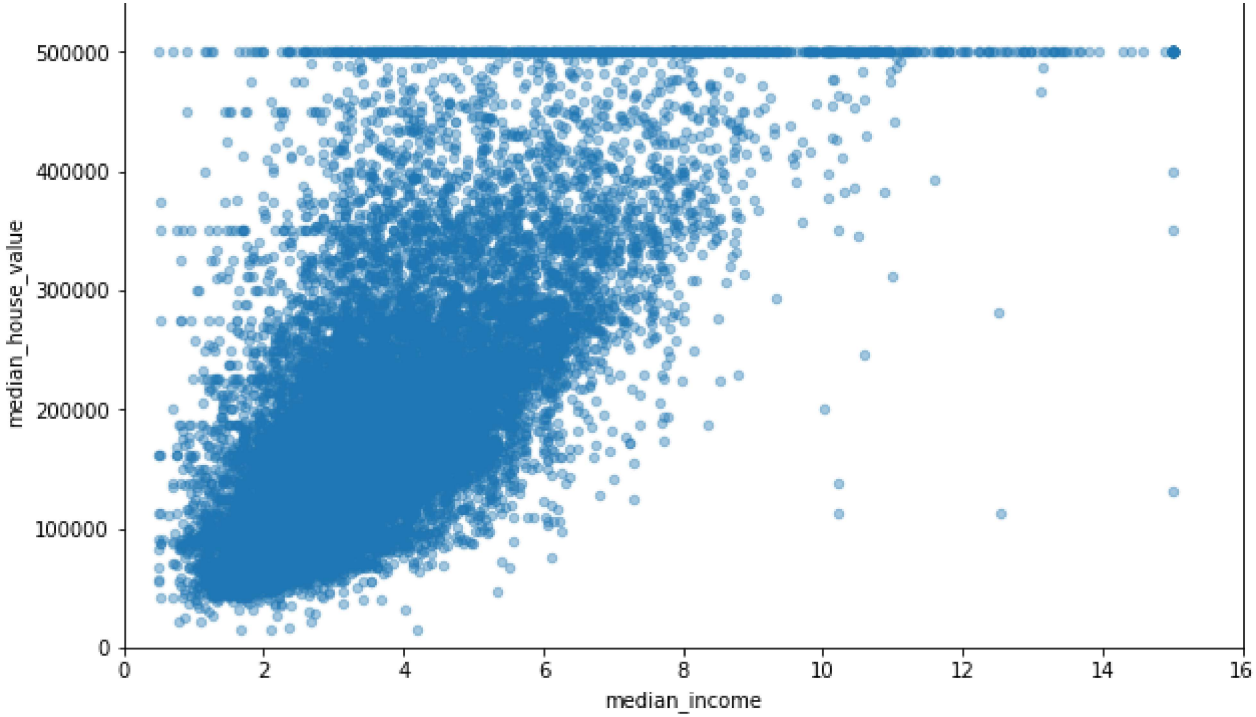
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f26d01649d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cee53b90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cf04fa50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cf052a10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f26cedd8c50>,
```

```
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cf0639d0>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cff4f450>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cf2d2d50>],
 [<matplotlib.axes._subplots.AxesSubplot object at 0x7f26cf2d2e10>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cf47bf50>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cecc3890>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26d04364d0>],
 [<matplotlib.axes._subplots.AxesSubplot object at 0x7f26d08ab310>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cef25b90>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26d018f490>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7f26cffad750>]],
dtype=object)
```




```
1 housing.plot(kind="scatter", x="median_income", y="median_house_value",figsize=(10,6), alpha=0.4)
2 plt.axis([0,16,0,550000])
```

```
(0.0, 16.0, 0.0, 550000.0)
```



Feature Engineering

```
housing["bedrooms_per_total_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"] = housing["population"]/housing["households"]
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
```

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	m
count	16512.000000	16512.000000	16512.000000	16512.000000	16341.000000	16512.000000	16512.000000	16512.000000	
mean	-119.565343	35.625289	28.604833	2639.802204	539.075271	1425.748728	500.307837	3.869101	

std	2.002905	2.136777	12.578033	2187.662988	423.385176	1139.767186	383.803724	1.898535
min	-124.300000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900
25%	-121.792500	33.930000	18.000000	1442.000000	295.000000	784.000000	279.000000	2.566300
50%	-118.490000	34.250000	29.000000	2125.000000	435.000000	1166.000000	409.000000	3.525250
75%	-118.000000	37.710000	37.000000	3156.250000	649.000000	1729.000000	607.000000	4.744475
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100

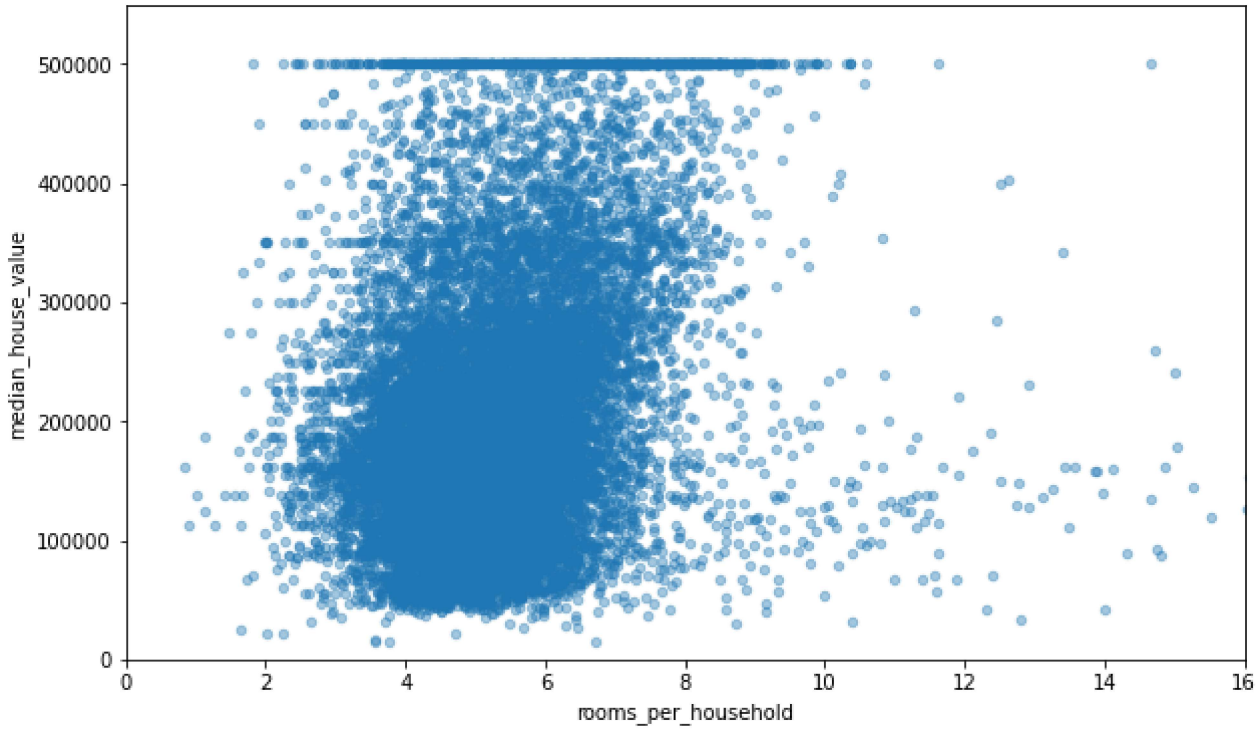
```
1 new_correlation = housing.corr()
2 new_correlation["median_house_value"].sort_values(ascending =False)
```

```
median_house_value          1.000000
median_income               0.684453
rooms_per_household         0.147300
total_rooms                 0.136386
housing_median_age          0.109590
households                  0.068493
total_bedrooms              0.052798
population_per_household   -0.022249
population                 -0.023337
longitude                  -0.044110
latitude                   -0.145566
bedrooms_per_total_room    -0.258101
Name: median_house_value, dtype: float64
```

[Observation]: The new 'bedrooms_per_housing' is highly correlated in a reciprocatice way to the 'median_house_value'. So, the houses with lesser bedroom/total_rooms will tend to be more expensive.

```
1 housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",figsize=(10,6), alpha=0.4)
2 plt.axis([0,16,0,550000])
```

```
(0.0, 16.0, 0.0, 550000.0)
```



Preparing the data

```
1 housing = strat_train_set.drop("median_house_value", axis=1)
2 housing_labels = strat_train_set["median_house_value"].copy()
```

```
1 housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 11091 to 827
Data columns (total 9 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  int64
 3   total_rooms         16512 non-null  int64
 4   total_bedrooms      16341 non-null  float64
 5   population          16512 non-null  int64
 6   households          16512 non-null  int64
 7   median_income       16512 non-null  float64
 8   ocean_proximity     16512 non-null  object
dtypes: float64(4), int64(4), object(1)
memory usage: 1.3+ MB
```

```
1 housing_labels.shape
```

```
(16512,)
```

- **Data Cleansing**

```
1 sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
2 sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proxi
9149	-118.50	34.46	17	10267	NaN	4956	1483	5.5061	<1H OC
20372	-118.88	34.17	15	4260	NaN	1701	669	5.1033	<1H OC
3354	-120.67	40.50	15	5343	NaN	2503	902	3.5962	INL
14015	-117.17	32.75	52	1052	NaN	381	201	3.0726	NEAR OC
7668	-118.08	33.92	38	1335	NaN	1011	269	3.6908	<1H OC

```
1 median = housing["total_bedrooms"].median()
2 sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True)
3 sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proxi
9149	-118.50	34.46	17	10267	435.0	4956	1483	5.5061	<1H OC
20372	-118.88	34.17	15	4260	435.0	1701	669	5.1033	<1H OC
3354	-120.67	40.50	15	5343	435.0	2503	902	3.5962	INL
14015	-117.17	32.75	52	1052	435.0	381	201	3.0726	NEAR OC
7668	-118.08	33.92	38	1335	435.0	1011	269	3.6908	<1H OC

- Filling NaN value by using simple Imputer

```
1 from sklearn.impute import SimpleImputer
2
3 imputer= SimpleImputer(strategy="median")
```

```
1 housing_num = housing.drop("ocean_proximity", axis=1)
2
3 imputer.fit(housing_num)
```

```
SimpleImputer(strategy='median')
```

```
1 # Imputer Basicly computes across all the attributes. to see this...
2 print(imputer.statistics_)
3 print(housing_num.median().values)
4 #Both should be same
```

```
[-118.49      34.25      29.      2125.       435.      1166.
  409.         3.52525]

[-118.49      34.25      29.      2125.       435.      1166.
  409.         3.52525]
```

```
1 x = imputer.transform(housing_num)
```

```
2 x
```

```
array([[-1.1789e+02,  3.3820e+01,  1.8000e+01, ...,  1.8940e+03,
         7.2600e+02,  3.6761e+00],
       [-1.2218e+02,  3.7750e+01,  4.5000e+01, ...,  2.8200e+02,
         8.0000e+01,  4.0469e+00],
       [-1.1555e+02,  3.2790e+01,  2.2000e+01, ...,  6.9200e+02,
         1.4100e+02,  1.2083e+00],
       ...,
       [-1.2069e+02,  3.5520e+01,  2.6000e+01, ...,  1.2910e+03,
         5.2200e+02,  2.9250e+00],
       [-1.2220e+02,  3.7820e+01,  3.9000e+01, ...,  1.2650e+03,
         5.0000e+02,  6.3302e+00],
       [-1.2209e+02,  3.7630e+01,  3.5000e+01, ...,  7.9000e+02,
         2.4300e+02,  4.7019e+00]])
```

```
1 housing_tr = pd.DataFrame(x, columns=housing_num.columns)
2 housing_tr.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-117.89	33.82	18.0	3197.0	809.0	1894.0	726.0	3.6761
1	-122.18	37.75	45.0	330.0	76.0	282.0	80.0	4.0469
2	-115.55	32.79	22.0	565.0	162.0	692.0	141.0	1.2083
3	-117.79	33.80	11.0	10535.0	1620.0	4409.0	1622.0	6.6700
4	-122.68	38.43	29.0	488.0	63.0	161.0	62.0	6.0774

- **Handling categorical Values**

```
1 housing_cat = housing["ocean_proximity"]
2 housing_cat.value_counts()
```

```
<1H OCEAN     7328
INLAND        5213
```

```
NEAR OCEAN    2128
NEAR BAY      1838
ISLAND           5
Name: ocean_proximity, dtype: int64
```

```
1 housing_cat.head()
```

```
11091    <1H OCEAN
339       NEAR BAY
2725         INLAND
10326    <1H OCEAN
19179    <1H OCEAN
Name: ocean_proximity, dtype: object
```

Convert categorical features into numerical features

- Using Pandas

```
1 housing_cat_encoded, housing_categories = housing_cat.factorize()
2 housing_cat_encoded[:10]
```

```
array([0, 1, 2, 0, 0, 2, 2, 0, 0, 2])
```

```
1 print(housing_categories[:10])
```

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

- Using Scikit-learn'S OneHotEncoder

```
1 from sklearn.preprocessing import OneHotEncoder
2
3 encoder = OneHotEncoder()
```

```
4
5 housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(1,-1))
6 housing_cat_1hot
```

```
<1x16512 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

- Since 1 hot encoder returns a sparse matrix, need to change it to a dense array

```
1 housing_cat_1hot.toarray()
```

```
array([[1., 1., 1., ..., 1., 1., 1.]])
```

Custom Transformations

```
1 from sklearn.base import BaseEstimator, TransformerMixin
2
3 #column indexes
4 rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6
5
6 class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
7
8     def __init__(self, add_bedrooms_per_room = True):
9         self.add_bedrooms_per_room = add_bedrooms_per_room
10
11    def fit(self, X, y=None):
12        return self # nothing to do here
13
14    def transform(self, X, y=None):
15        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
16        population_per_household = X[:, population_ix] / X[:, household_ix]
17
18        if self.add_bedrooms_per_room:
19            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
20            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
```

```
        else:
            return np.c_[X, rooms_per_household, population_per_household]


attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)

housing_extra_attribs = pd.DataFrame(housing_extra_attribs, columns=list(housing.columns)+["rooms_per_household",
                                                                                           "population_per_household"])
housing_extra_attribs.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
0	-117.89	33.82	18	3197	809.0	1894	726	3.6761	<1H OCEAN
1	-122.18	37.75	45	330	76.0	282	80	4.0469	NEAR BAY
2	-115.55	32.79	22	565	162.0	692	141	1.2083	INLAND
3	-117.79	33.8	11	10535	1620.0	4409	1622	6.67	<1H OCEAN
4	-122.68	38.43	29	488	63.0	161	62	6.0774	<1H OCEAN

Setting up Pipeline for all the preprocessings

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
                        ("Imputer", SimpleImputer(strategy = "median")),
                        ("attribs_adder", CombinedAttributesAdder()),
                        ("std_scaler",StandardScaler())
```

```
                        ('std_scaler',StandardScaler())
                        ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
housing_num_tr
```

```
class DataFrameSelector(BaseEstimator, TransformerMixin):

    def __init__(self, attribute_names):
        self.attibute_names = attribute_names

    def fit(self, X, y=None):
        return self # do nothing

    def transform(self, X, y=None):
        return X[self.attibute_names].values
```

```
# complete Pipeline

num_attribs = list(housing_num.columns)
cat_attribs = ["ocean_proximity"]

num_pipeline = Pipeline([
    ("selector", DataFrameSelector(num_attribs)),
    ("imputer", SimpleImputer(strategy="median")),
    ("attribs_adder", CombinedAttributesAdder()),
    ("std_scaler", StandardScaler())
])

cat_pipeline =Pipeline([
    ("selector", DataFrameSelector(cat_attribs)),
    ("cat_encoder", OneHotEncoder(sparse=False))
])
```

```
from sklearn.pipeline import FeatureUnion

full_pipeline = FeatureUnion(transformer_list=[
    ('num_pipeline', num_pipeline),
    ('cat_pipeline', cat_pipeline)
])
```

```
housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
```

```
from sklearn.pipeline import FeatureUnion

full_pipeline = FeatureUnion(transformer_list=[('num_pipeline', num_pipeline),
                                               ('cat_pipeline', cat_pipeline)
                                              ])
```

```
housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
```

Selecting & Training Models

```
from sklearn.linear_model import LinearRegression
linReg= LinearRegression()

linReg.fit(housing_prepared, housing_labels)
```

```
# Trying the full pipeline on a few training instances

some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data
some_data_prepared = full_pipeline.transform(some_data)
```

```
some_data_prepared = full_pipeline.transform(some_data)
```

```
print("Prediction:", linReg.predict(some_data_prepared))
print("Actual labels:", list(some_labels))
```

```
from sklearn.metrics import mean_squared_error

housing_predictions = linReg.predict(housing_prepared)

lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse =np.sqrt(lin_mse)
lin_rmse
```

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

tree_reg =DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

```
housing_predictions = tree_reg.predict(housing_prepared)

tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Cross Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels, cv= 10, scoring = "neg_mean_squared_error")

tree_rmse_scores = np.sqrt(-scores)
```

```
tree_rmse_scores = np.sqrt(-scores)
```

```
def display_scores(scores):
    print("Scores:", scores)
    print("mean:", scores.mean())
    print("std deviation:", scores.std())
```

```
display_scores(lin_rmse)
```

```
display_scores(tree_rmse_scores)
```

```
lin_scores = cross_val_score(linReg, housing_prepared, housing_labels, cv=10, scoring="neg_mean_squared_error")

lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
from sklearn.ensemble import RandomForestRegressor

rfReg= RandomForestRegressor(random_state=29)
rfReg.fit(housing_prepared, housing_labels)
```

```
housing_pred = rfReg.predict(housing_prepared)

rf_scores=cross_val_score(rfReg, housing_prepared, housing_labels, cv=10, scoring="neg_mean_squared_error")

rf_rmse_scores = np.sqrt(-rf_scores)

display_scores(rf_rmse_scores)
```

```
##Fine Tuning Model:[TODO ]
```

```
from sklearn.model_selection import GridSearchCV

param_grid = [{'n_estimators': [3, 10, 30], 'max_features':[2, 4, 6, 8]}
```

```
param_grid = [{ n_estimators : [ 3, 10, 30], max_features :[2, 4, 6, 8]},
              {'bootstrap': [False], 'n_estimators': [3,10], 'max_features':[2,3,4]}
              ]
rf_reg = RandomForestRegressor()
grid_search =GridSearchCV(rf_reg, param_grid, cv=5, scoring="neg_mean_squared_error")

grid_search.fit(housing_prepared, housing_labels)
```

```
#To get the best combinations of hyperparameters
grid_search.best_params_
```

```
# To get the best estimators directly
grid_search.best_estimator_
```

```
cv_res = grid_search.cv_results_

for mean_score, params in zip(cv_res["mean_test_score"], cv_res["params"]):
    print(np.sqrt(-mean_score), params)
```

```
df = pd.DataFrame(grid_search.cv_results_)
df
```

```
  from sklearn.model_selection import RandomizedSearchCV
  from scipy.stats import randint

  params_disttibs={
      "n_estimators" : randint(low = 1, high = 200),
      "max_features" : randint(low =1, high =8)
  }

  rf_reg = RandomForestRegressor(random_state=29)

  rnd_search = RandomizedSearchCV(rf_reg, param_distributions=params_disttibs, n_iter = 10, cv =5,
                                  scoring = "neg_mean_squared_error", random_state=29)
```

```
1  pd.DataFrame(grid_search.cv_results_).to_excel("California Housing Price Predicted.xlsx")
```

```
1
```

✓ 0s completed at 20:53