Chapter 2: End to End Machine Learning Project

1. Import Libraries

In [ ]:
```
import os
import tarfile
import urllib
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import random
```

2. Download Data

In [ ]:
```
DOWNLOAD_ROOT= "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"
```

In [ ]:
```
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()
```

In [ ]:
```
fetch_housing_data()
```

In [ ]:
```
def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

In [ ]:
```
housing = load_housing_data()
housing.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household:
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.(
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.(
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.(
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.(
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.(

3. Data Pre-processing

- There are 20,640 instancs in the dataset, which means that it is fairly small by Machine

Learning standards.

- total_bedrooms has only 20,433 nonnull values

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

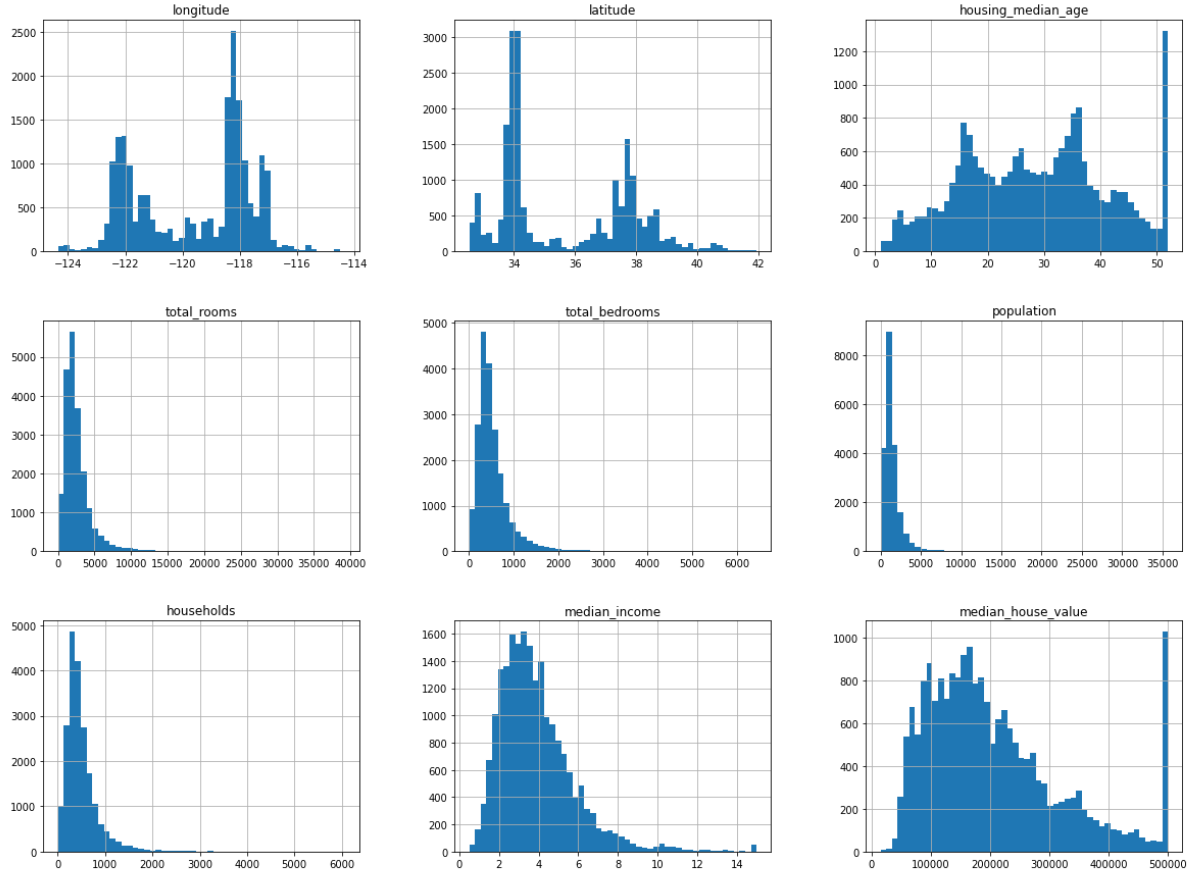
Out[ ]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

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

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populat
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000

In [ ]:
```
%matplotlib inline
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



3.1 Train Test Split

In [ ]:
```
# from scratch
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]
```

In [ ]:
```
train_set, test_set = split_train_test(housing, 0.2)
```

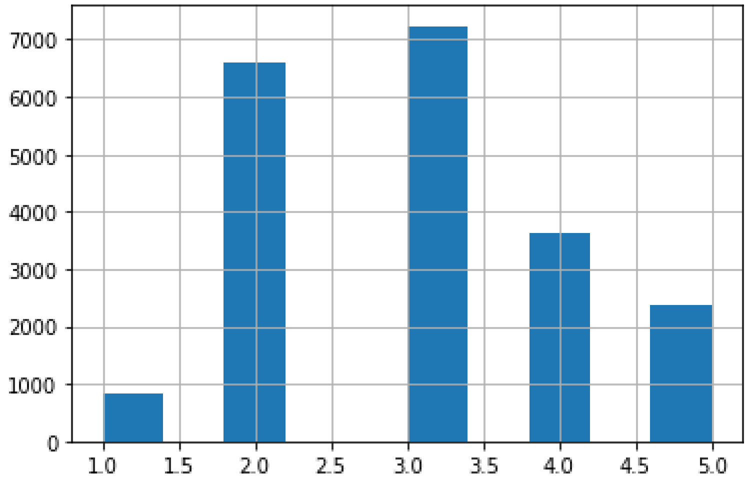
In [ ]:
```
# using scikit-learn
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

We want to maintain the ratio in the sample. To do this, we use stratified sampling. Since the median income is a continuous numerical attributem we need to first create an income category attribute. It is important to have a sufficient number of instances in your dataset

In [ ]:
```
# create income category attribute
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6.0, np.inf],
                               labels=[1,2,3,4,5])
```

In [ ]:
```
housing["income_cat"].hist()
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
housing.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household:
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.(
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.(
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.(
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.(
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.(

In [ ]:
```
# stratified sampling
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 [ ]:
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

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

Test set generated using stratified sampling has income category proportions almost identical to those in the full dataset, whereas the test set generated using purely random sampling is skewed

Now we remove income_cat attribute so that the data is back to its original state

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

4. Exploratory Data Analysis

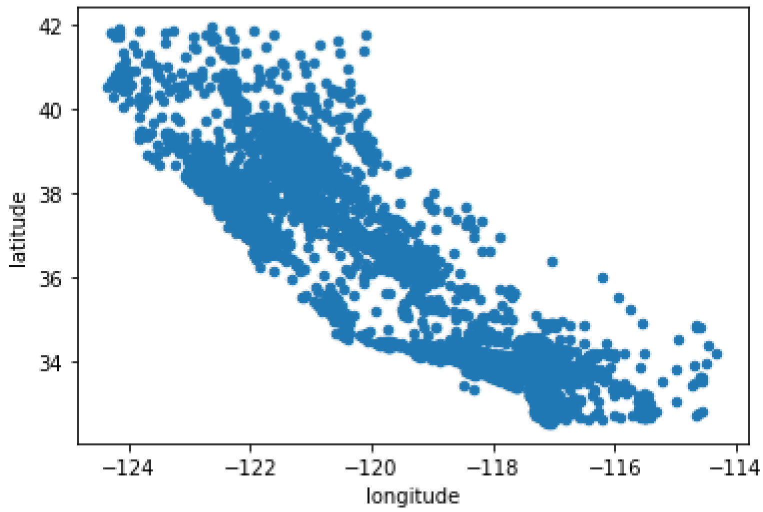
Create a copy of the training data so that we can play with it without harming the training set

In [ ]:
```
housing = strat_train_set.copy()
```

4.1 Latitude and Longitude

In [ ]:
```
# visualizing geographical data
housing.plot(kind="scatter", x="longitude", y="latitude")
```

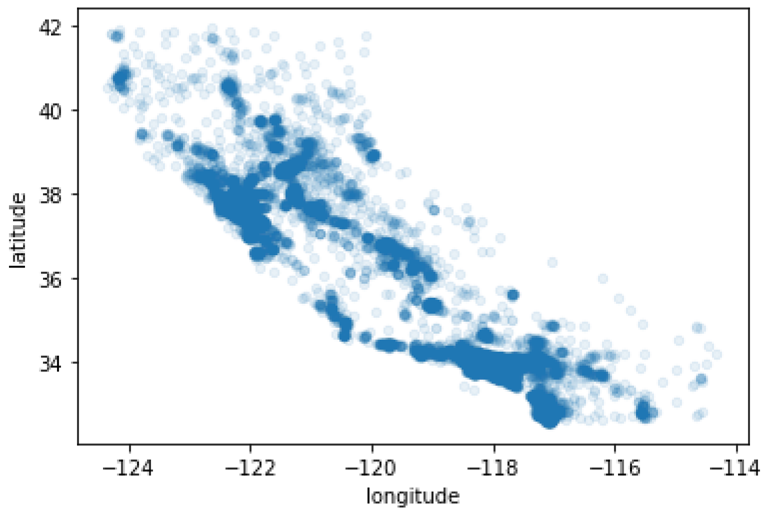
Out[ ]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



alpha option makes it much easier to visualize the places where there is a high density of data points

In [ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

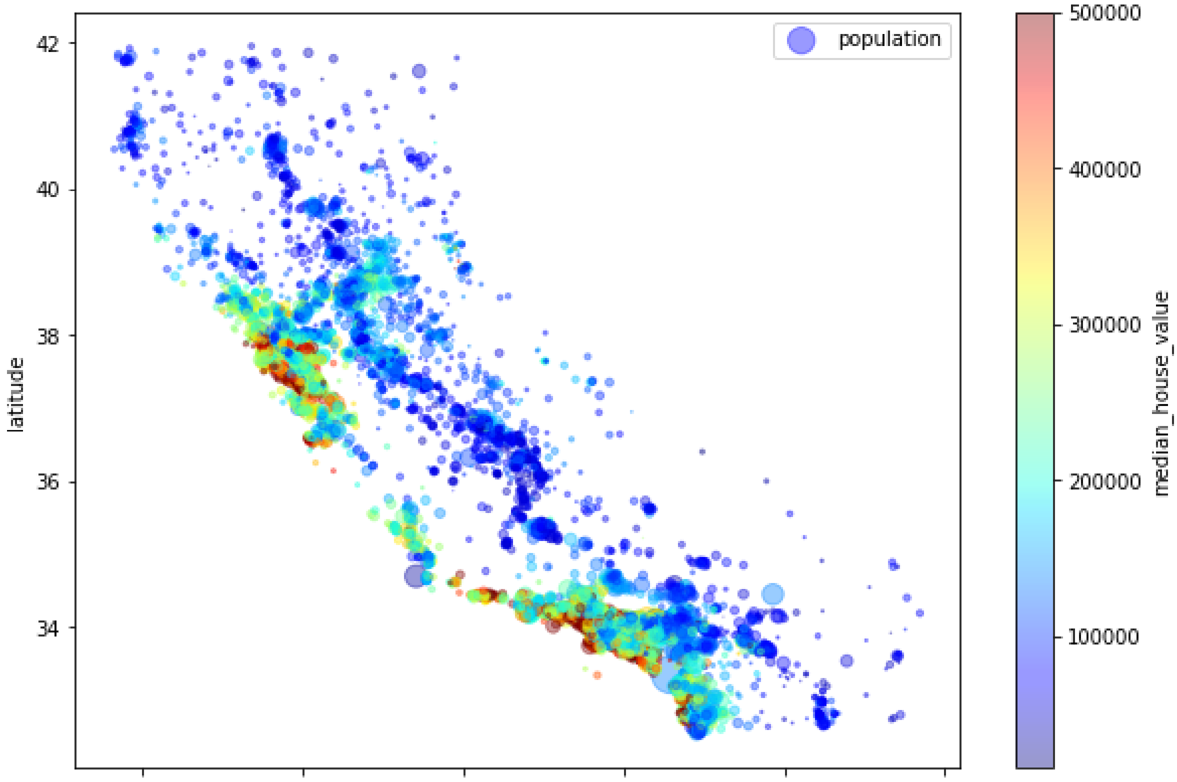
Out[ ]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



4.2 Housing Prices

In [ ]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True)
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x22b3487df40>



4.3 Spearman Correlations

Since dataset is not too large, can easily compute standard pearson's correlation coefficient between every pair of attributes

Note that the correlation coefficient only measures linear correlations. It may completely miss out on nonlinear relationships

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

In [ ]:
```
corr_matrix["median_house_value"].sort_values(ascending=True)
```

Out[ ]:
```
latitude             -0.142673
longitude            -0.047466
population           -0.026882
total_bedrooms        0.047781
households            0.064590
housing_median_age    0.114146
total_rooms           0.135140
median_income         0.687151
median_house_value    1.000000
Name: median_house_value, dtype: float64
```

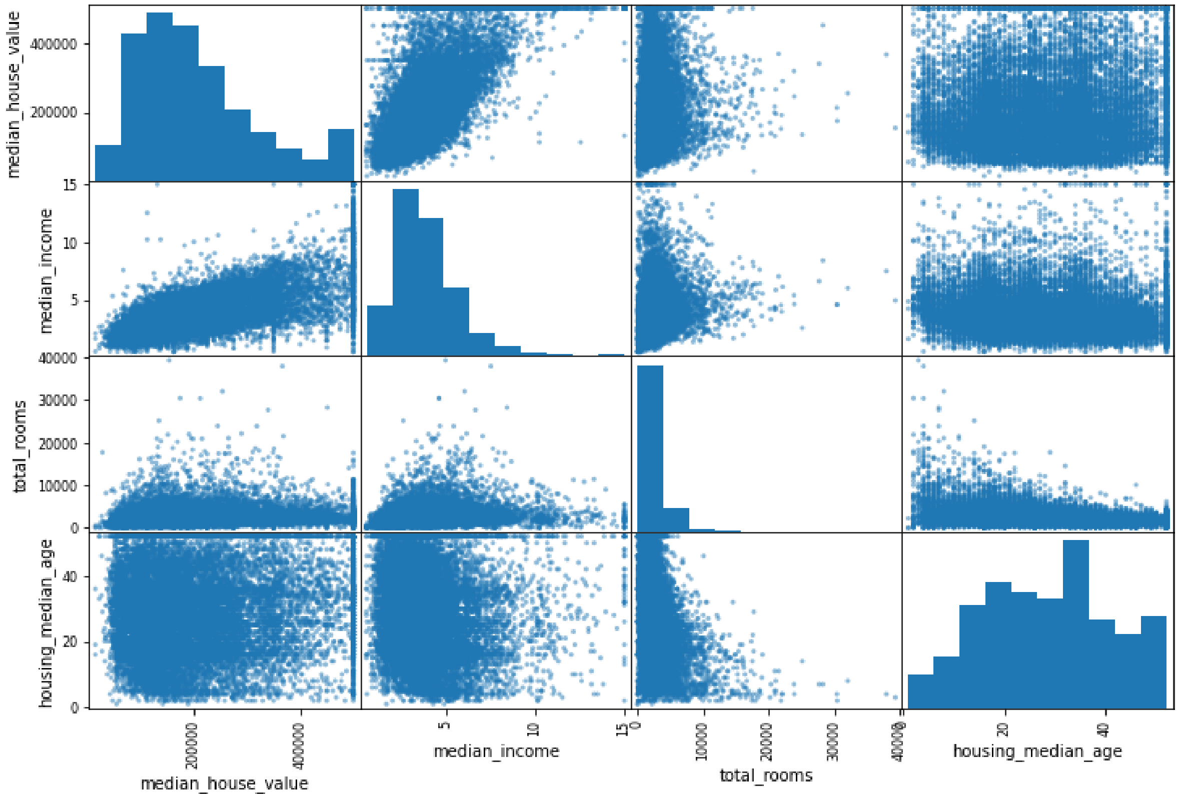
The correlation coefficient ranges from -1 to 1. When it is close to 1, it means that there is a strong positive correlation. i.e, the median house value tends to go up when the median income goes up. When the coefficient is close to -1, it means that there is a strong negative correlation. i.e prices have a slight tendency to go when when you go north. When coefficients are close to 0, there is no linear correlation

Another way to check correlation between attributes is to use the pandas scatter_matrix() function, which plots every numerical attribute against every other numerical attribute. Here we just focus on a few promising attributes that seem most correlated with the median housing value

In [ ]:
```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms", "housing_mediar
scatter_matrix(housing[attributes], figsize=(12,8))
```

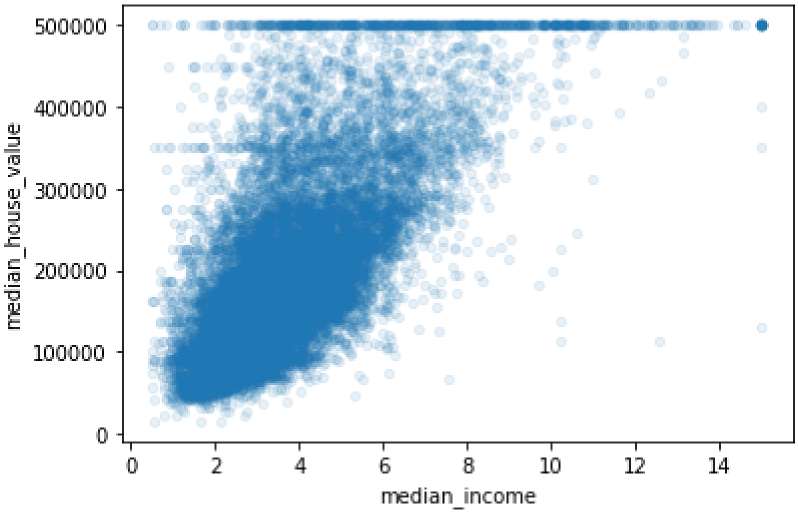
Out[ ]:
```
array([[<AxesSubplot:xlabel='median_house_value', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='median_income', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='median_house_value'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='median_income'>,
        <AxesSubplot:xlabel='median_income', ylabel='median_income'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='median_income'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='median_income'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='median_income', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='total_rooms'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='median_income', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='housing_median_age'>]],
      dtype=object)
```



Zoom into most promising attribute, median_income

In [ ]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
```

Out[ ]:
```
<AxesSubplot:xlabel='median_income', ylabel='median_house_value'>
```



5. Feature Engineering

In [ ]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"] = housing["population"]/housing["households"]
```

In [ ]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[ ]:
```
median_house_value          1.000000
median_income               0.687151
rooms_per_household         0.146255
total_rooms                 0.135140
housing_median_age          0.114146
households                  0.064590
total_bedrooms              0.047781
population_per_household   -0.021991
population                 -0.026882
longitude                  -0.047466
latitude                   -0.142673
bedrooms_per_room          -0.259952
Name: median_house_value, dtype: float64
```

6. Prepare data for Machine Learning Algorithms

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

6.1 Missing Values

In [ ]:
```
# missing values
#housing.dropna(subset=["total_bedrooms"]) # option 1
#housing.drop("total_bedrooms", axis=1) # option 2
median = housing["total_bedrooms"].median() # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

In [ ]:
```
# using SimpleImputer from scikit-learn
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

Since median can only be computed on numerical attributes, need to create a copy of the data without the text attribute

In [ ]:
```
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
```

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

See the median values

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

Out[ ]:
```
array([-118.51    ,   34.26    ,   29.      , 2119.      ,  433.      ,
        1164.      ,  408.      ,    3.54155])
```

In [ ]:
```
housing_num.median().values
```

Out[ ]:
```
array([-118.51    ,   34.26    ,   29.      , 2119.      ,  433.      ,
        1164.      ,  408.      ,    3.54155])
```

Now we use this "trained" imputer to perform the training set by replacing missing values with the learned medians

In [ ]:
```
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

6.2 Categorical Attributes

In [ ]:
```
housing_cat = housing[["ocean_proximity"]]
```

In [ ]:
```
housing_cat.head(10)
```

Out[ ]:

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN

Problem with ordinal encoding is that ML algorithms will assume that two nearby values are more similar than two distant values

In [ ]:
```
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_cat_encod = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encod[:10]
```

Out[ ]:
```
array([[1.],
       [4.],
       [1.],
       [4.],
       [0.],
       [3.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

In [ ]:
```
ordinal_encoder.categories_
```

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

To fixe the issue, a common solution is to create one binary attribute per category. This is called one-hot encoding

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

In [ ]:
```
housing_cat.head()
```

Out[ ]:

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN

In [ ]:
```
housing_cat_1hot.toarray()
```

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

In [ ]:
```
housing.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	
20496	-118.70	34.28	27.0	3536.0	646.0	1837.0	

6.3 Custom Transformers

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

# columm index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room =True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

In [ ]:
```
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
```

In [ ]:
```
housing_extra_attribs = attr_adder.transform(housing.values)
```

In [ ]:
```
housing.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	
20496	-118.70	34.28	27.0	3536.0	646.0	1837.0	

In [ ]:
```
pd.DataFrame(housing_extra_attribs)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10
0	-121.46	38.52	29.0	3873.0	797.0	2237.0	706.0	2.1736	INLAND	5.485836	3.168555
1	-117.23	33.09	7.0	5320.0	855.0	2015.0	768.0	6.3373	NEAR OCEAN	6.927083	2.623698
2	-119.04	35.37	44.0	1618.0	310.0	667.0	300.0	2.875	INLAND	5.393333	2.223333
3	-117.13	32.75	24.0	1877.0	519.0	898.0	483.0	2.2264	NEAR OCEAN	3.886128	1.859213
4	-118.7	34.28	27.0	3536.0	646.0	1837.0	580.0	4.4964	<1H OCEAN	6.096552	3.167241
...	...	...	...	...	...	...	...	...	...	...	...
16507	-117.07	33.03	14.0	6665.0	1231.0	2026.0	1001.0	5.09	<1H OCEAN	6.658342	2.023976
16508	-121.42	38.51	15.0	7901.0	1422.0	4769.0	1418.0	2.8139	INLAND	5.571932	3.363188
16509	-122.72	38.44	48.0	707.0	166.0	458.0	172.0	3.1797	<1H OCEAN	4.110465	2.662791
16510	-122.7	38.31	14.0	3155.0	580.0	1208.0	501.0	4.1964	<1H OCEAN	6.297405	2.411178
16511	-122.14	39.97	27.0	1079.0	222.0	625.0	197.0	3.1319	INLAND	5.477157	3.172589

16512 rows × 11 columns

6.4 Feature Scaling

Machine Learning algorithms don't perform well when the input numerical attributes have very different scales. The two common ways to get all attributes to have the same scale is Min-max scaling and Standardization

Min-max scaling (normalization): values are shifted and rescaled so that they end up ranging from 0 to 1. We do this by subtracting the min value and divding by the max - min

Standardization: subtracts the mean value then divide by the standard deviation so that the resulting distribution has unit variance. Unlike min-max scaling, standardization does not bound values to a specific range, which may be problem for some algorithms such as neural network which often expects input value ranging from 0 to 1. Standardization is less affected by outliers.

6.5 Transformation Pipeline

In [ ]:

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

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

In [ ]:
```
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

In [ ]:
```
housing_num.head()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	
20496	-118.70	34.28	27.0	3536.0	646.0	1837.0	

In [ ]:
```
housing_num_tr
```

Out[ ]:
```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.01739526,
         0.00622264, -0.12112176],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.56925554,
        -0.04081077, -0.81086696],
       [ 0.26758118, -0.1259716 ,  1.22045984, ..., -0.01802432,
        -0.07537122, -0.33827252],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ..., -0.5092404 ,
        -0.03743619,  0.32286937],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.32814891,
        -0.05915604, -0.45702273],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.01407228,
         0.00657083, -0.12169672]])
```

It would be more convenient to have a single transformation that is able to handle all columns

In [ ]:
```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ['ocean_proximity']

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ('cat', OneHotEncoder(), cat_attribs)
])
```

In [ ]:
```
housing_prepared = full_pipeline.fit_transform(housing)
```

In [ ]:
```
housing_prepared
```

Out[ ]:
```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.        ,
         0.        ,  0.        ],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ...,  0.        ,
         0.        ,  0.        ],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.        ,
         0.        ,  0.        ],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.        ,
         0.        ,  0.        ]])
```

In [ ]:
```
housing_labels
```

Out[ ]:
```
12655     72100.0
15502    279600.0
2908      82700.0
14053    112500.0
20496    238300.0
           ...
15174    268500.0
12661     90400.0
19263    140400.0
19140    258100.0
19773     62700.0
Name: median_house_value, Length: 16512, dtype: float64
```

7. Select and Train Model

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

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels) #X and y
```

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

In [ ]:
```
# try out a few instances
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)
print("Predictions:", lin_reg.predict(some_data_prepared))
print("Labels:", list(some_labels))
```

```
Predictions: [ 85657.90192014 305492.60737488 152056.46122456 186095.70946094
 244550.67966089]
Labels: [72100.0, 279600.0, 82700.0, 112500.0, 238300.0]
```

7.1 Evaluate

In [ ]:
```
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[ ]:
```
68627.87390018745
```

Most districts 'median_housing_values' range between $120,000 and$265,000 so a typical prediction error of $68,628 is not very satisfying. This is an example of a model underfitting the training data. When this happens, it can mean that the features do not provide enough information to make good predictions or that the model is not powerful enough.

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

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

Out[ ]:
```
DecisionTreeRegressor()
```

In [ ]:
```
housing_predictions = tree_reg.predict(housing_prepared)
```

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

Out[ ]:
```
0.0
```

7.2 Better evaluation using Cross-Validation

Scikit-learn's cross-validation features expect a utility function (greater is better) rather than cost function (lower is better)

In [ ]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels, scoring="neg_
tree_rmse_scores = np.sqrt(-scores)
```

In [ ]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard Deviation:", scores.std())
```

In [ ]:
```
display_scores(tree_rmse_scores)
```

```
Scores: [73595.34989436 71078.52459185 69549.47296616 71345.99126578
 70166.33220054 76977.87971415 72352.43394193 74235.84481341
 69807.32873363 70415.71803662]
Mean: 71952.48761584316
Standard Deviation: 2244.7068070704195
```

In [ ]:
```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
```

In [ ]:
```
display_scores(lin_rmse_scores)
```

```
Scores: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard Deviation: 2880.328209818065
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels, scor
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [51288.86399569 48548.33786579 46843.02711713 52135.1563919
 47534.95835781 51965.20371787 52720.12700305 50096.26481324
 48484.56448099 53605.01439966]
Mean: 50322.15181431368
Standard Deviation: 2233.730812992924
```

The goal here is to shortlist a few (2-5) promising models and not to tweak hyperparameters yet

8. Save model

Save every model you experiment with so that you can come back easily to any model you want. Make sure to save both the hyperparameters and trained parameters

In [ ]:
```
import joblib
joblib.dump(forest_reg, "forest_reg.pkl")
```

Out[ ]:
```
['forest_reg.pkl']
```

In [ ]:
```
forest_reg = joblib.load("forest_reg.pkl")
```

In [ ]:
```
forest_reg
```

Out[ ]:
```
RandomForestRegressor()
```

9. Fine-Tune Model

9.1 Grid Search

In [ ]:
```
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]}
]
```

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

In [ ]:
```
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring="neg_mean_squared_error",
                           return_train_score=True)
```

In [ ]:
```
grid_search.fit(housing_prepared, housing_labels)
```

Out[ ]:
```
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 [ ]:
```
grid_search.best_params_
```

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

In [ ]:
```
grid_search.best_estimator_
```

Out[ ]:
```
RandomForestRegressor(max_features=8, n_estimators=30)
```

In [ ]:
```
cvres = grid_search.cv_results_
for mean_scores, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_scores), params)
```

```
64625.37866083861 {'max_features': 2, 'n_estimators': 3}
55709.01784542573 {'max_features': 2, 'n_estimators': 10}
52652.38595997133 {'max_features': 2, 'n_estimators': 30}
59318.92493975303 {'max_features': 4, 'n_estimators': 3}
52617.73090393591 {'max_features': 4, 'n_estimators': 10}
50513.242108365404 {'max_features': 4, 'n_estimators': 30}
59067.96906477404 {'max_features': 6, 'n_estimators': 3}
51830.393221499646 {'max_features': 6, 'n_estimators': 10}
50136.706199635075 {'max_features': 6, 'n_estimators': 30}
58844.29171384842 {'max_features': 8, 'n_estimators': 3}
51879.74936349815 {'max_features': 8, 'n_estimators': 10}
49887.8688009439 {'max_features': 8, 'n_estimators': 30}
62199.62541720343 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
53935.505658194954 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
60678.56333144854 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52487.355134418096 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58518.28049544005 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51964.54961393742 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

9.2 Randomized Search

Grid search approach is fine when you are exploring relatively few combinations. But when the hyperameter search space is large, it is often preferable to use RandomizedSearchCV instead. Instead of trying out all possible combinations, it evaluates a given number of random combinations by selecting a random value for each hyperparameter at every iteration.

9.3 Ensemble Methods

Another way is to combine models that perform best. The group ("ensemble") will often perform better than the best individual model, especially if the individual models make very different type of errors.

10. Analyze Best Models and Their Errors

In [ ]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[ ]:
```
array([6.96480103e-02, 6.24908584e-02, 4.10599432e-02, 1.62746820e-02,
       1.47822858e-02, 1.46339289e-02, 1.42083937e-02, 3.74445552e-01,
       5.01867330e-02, 1.11215723e-01, 5.86809923e-02, 1.24428146e-02,
       1.52677196e-01, 9.71049230e-05, 3.76922812e-03, 3.38655391e-03])
```

In [ ]:
```
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
```

In [ ]:
```
sorted(zip(feature_importances, attributes), reverse=True)
```

Out[ ]:
```
[(0.3744455518604707, 'median_income'),
 (0.1526771964538734, 'INLAND'),
 (0.1121572264952793, 'pop_per_hhold'),
 (0.06964801026259029, 'longitude'),
 (0.06249085837505133, 'latitude'),
 (0.05868099227016819, 'bedrooms_per_room'),
 (0.050186733002872866, 'rooms_per_hhold'),
 (0.041059943175765584, 'housing_median_age'),
 (0.016274682031136026, 'total_rooms'),
 (0.01478228582767489, 'total_bedrooms'),
 (0.014633928868762023, 'population'),
 (0.014208393694326022, 'households'),
 (0.012442814570907781, '<1H OCEAN'),
 (0.003769228121263406, 'NEAR BAY'),
 (0.003386553912617258, 'NEAR OCEAN'),
 (9.710492299223573e-05, 'ISLAND')]
```

11. Evaluate System on the Test Set

In [ ]:
```
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()

X_test_prepared = full_pipeline.transform(X_test)
```

In [ ]:
```
final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [ ]:
```
final_rmse
```

Out[ ]:
```
48072.45224597982
```

We might want to have an idea of how precise this estimate is. For this, we can compute 95% confidence interval for the generalization error

In [ ]:
```
from scipy import stats
confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=  squared_errors.mean(),
                         scale = stats.sem(squared_errors)))
```

Out[ ]:
```
array([46019.98682257, 50040.80477705])
```

In [ ]: