End to End Machine Learning Project

Training Dataset

Training Dataset: The sample of data used to fit the model. The actual dataset that we use to train the model (weights and biases in the case of Neural Network). The model sees and learns from this data.

Validation Dataset

Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. We as machine learning engineers use this data to fine-tune the model hyperparameters. Hence the model occasionally sees this data, but never does it “Learn” from this. We(mostly humans, at-least as of 2017) use the validation set results and update higher level hyperparameters. So the validation set in a way affects a model, but indirectly.

Test Dataset

Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained(using the train and validation sets). The test set is generally what is used to evaluate competing models (For example on many Kaggle competitions, the validation set is released initially along with the training set and the actual test set is only released when the competition is about to close, and it is the result of the the model on the Test set that decides the winner). Many a times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled data that spans the various classes that the model would face, when used in the real world.

In [1]:

```
import os
import tarfile
from six.moves import urllib
```

In [2]:

```
download_root = "https://github.com/ageron/handson-ml/tree/master/"
housing_path = "dataset/housing"
housing_url = download_root + housing_path + "/housing.tgz"
```

In [3]:

```
def fetch_housing_data(housing_url = housing_url, housing_path = housing_path):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    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 [4]:

```
import pandas as pd
def load_housing_data(housing_path = housing_path):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

In [5]:

```
housing = load_housing_data()
housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
0	-122.23	37.88	41.0	880.0	129.0	322.0	1
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	11
2	-122.24	37.85	52.0	1467.0	190.0	496.0	1
3	-122.25	37.85	52.0	1274.0	235.0	558.0	2
4	-122.25	37.85	52.0	1627.0	280.0	565.0	2

In [6]:

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [7]:

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

Out[7]:

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

In [8]:

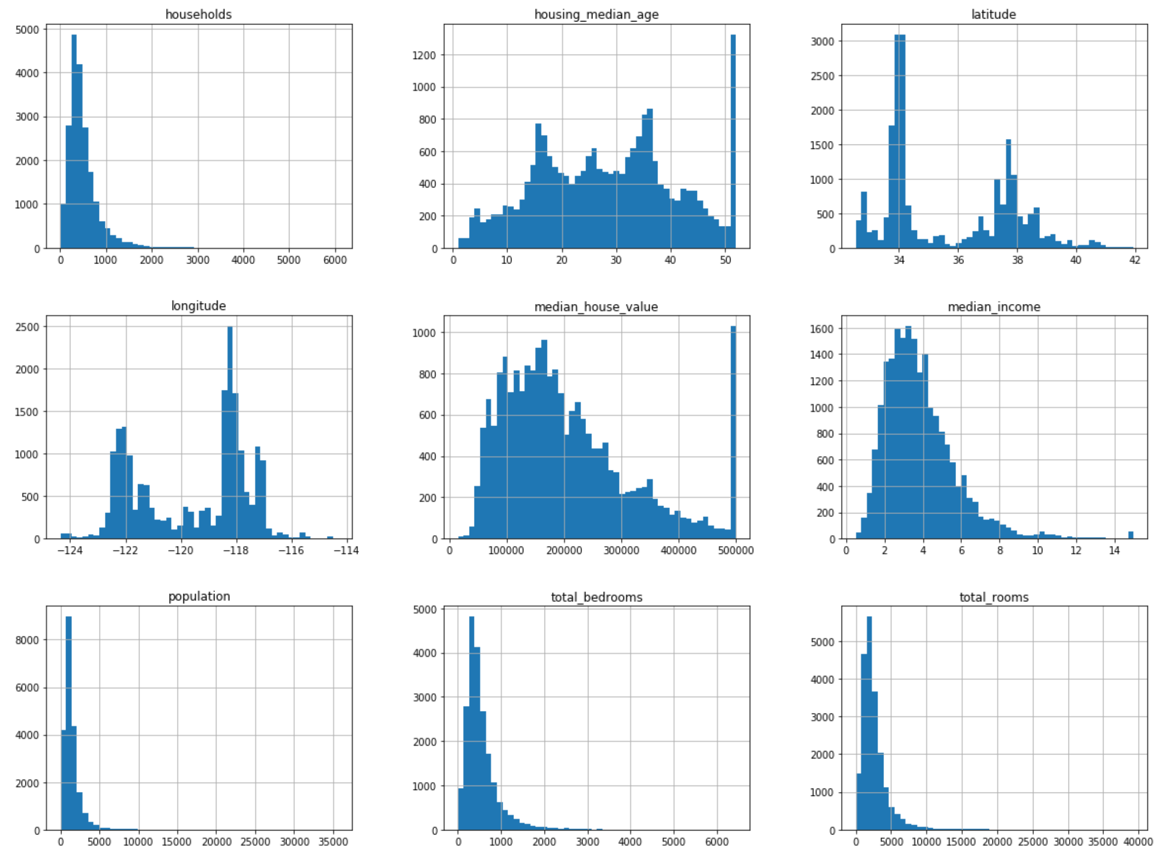
```
housing.describe()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	pc
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	2064
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	142
std	2.003532	2.135952	12.585558	2181.615252	421.385070	113
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	78
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	116
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	172
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	3568

In [9]:

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins = 50, figsize = (20,15))
plt.show()
```



Create Test set

In [10]:

```
import numpy as np
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]

train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

```
16512 train + 4128 test
```

In [11]:

```
#incase we keep updating our dataset.
#new test set will have 20% of the new instances.
#But it will not contain any instance that was previously in the training set.
import hashlib

def test_set_check(identifier, test_ratio, hash):
    return hash(np.int64(identifier)).digest()[-1] < 256 * test_ratio

def split_train_test_by_id(data, test_ratio, id_column, hash = hashlib.md5):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio, hash))
    return data.loc[~in_test_set], data.loc[in_test_set]

#use row index as identifier column
housing_with_id = housing.reset_index()
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
print(len(train_set), "train +", len(test_set), "test")
```

```
16362 train + 4278 test
```

In [12]:

```
#function is also provided by scikit-learn
#random_state -> set random generator seed, i.e, same data is selected as test data eve
rytime
#can pass multiple dataset with identical number of rows... it will split them on same
 indices

from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size = 0.2, random_state = 42)
print(len(train_set), "train +", len(test_set), "test")
```

```
16512 train + 4128 test
```

In [13]:

```
# stratified sampling -> population is divided into homogeneous subgroups  called strat
a
#limit the number of income categories
#merging all categories greater than 5 into category 5

housing["income_cat"] = np.ceil(housing["median_income"]/1.5)
housing["income_cat"].where(housing["income_cat"]<5, 5.0, inplace=True)

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]
print(len(strat_train_set), "train +", len(strat_test_set), "test")

#dropping the income_cat attribute
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1,inplace=True)
```

```
16512 train + 4128 test
```

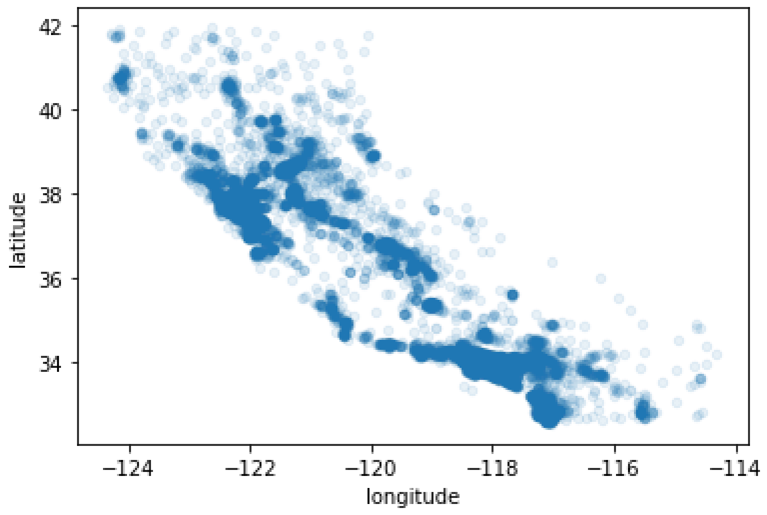
Visualize the data

In [14]:

```
#keep test set aside so that we can play with training data
housing = strat_train_set.copy()
housing.plot(kind = "scatter", x="longitude", y="latitude", alpha = 0.1)
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8012a1a88>
```

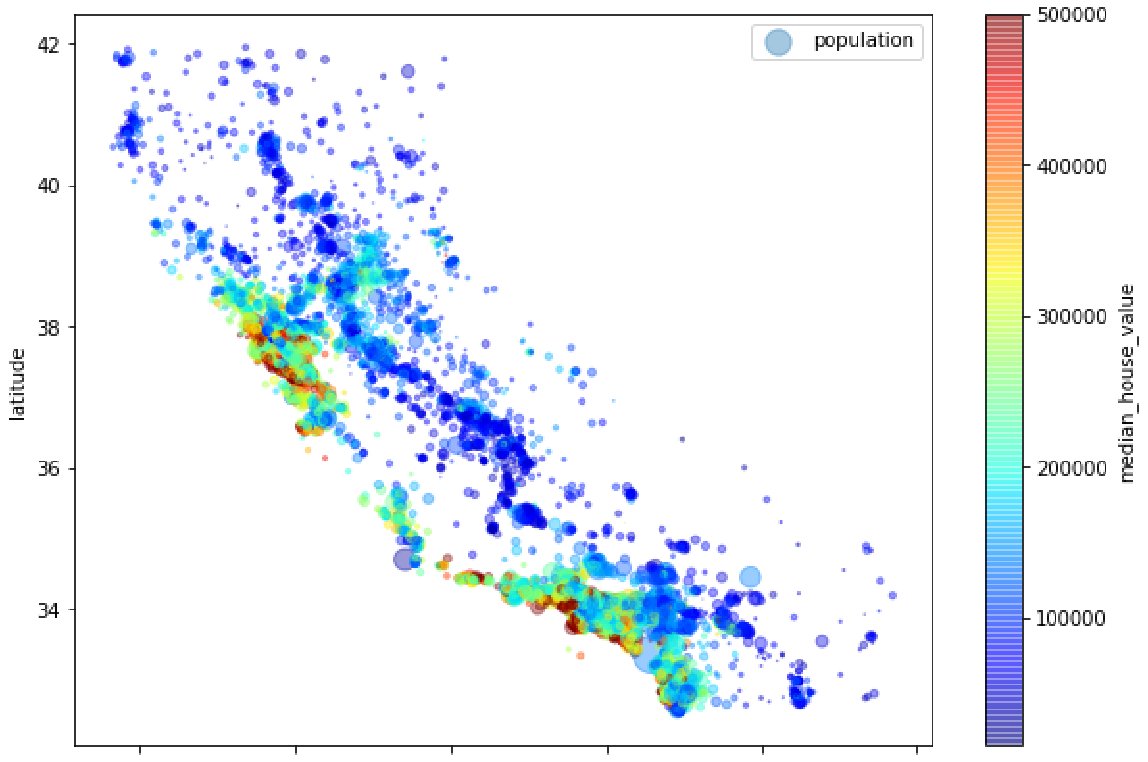


In [15]:

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

Out[15]:

```
<matplotlib.legend.Legend at 0x1e80138bc88>
```

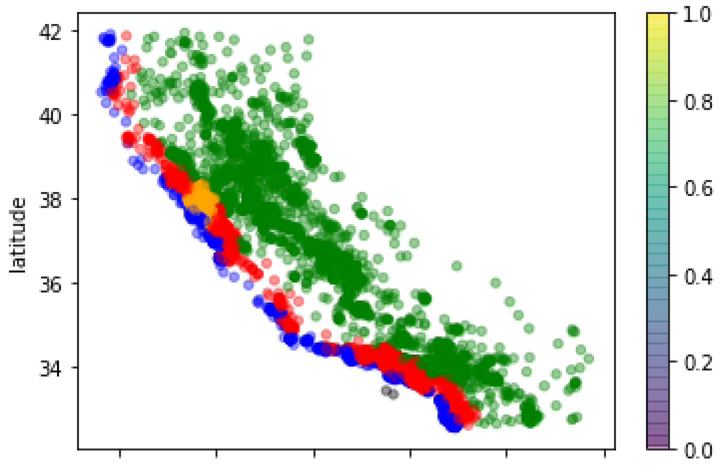


In [16]:

```
#get color code for ocean proximity???
color_dict = {'<1H OCEAN':'red', 'NEAR OCEAN' : 'blue', 'INLAND': 'green', 'NEAR BAY' :
'orange', 'ISLAND' : 'black'}
housing.plot(kind="scatter", x = "longitude",y="latitude", alpha=0.4, color=[color_dict
.get(x, '#333333') for x in housing["ocean_proximity"]], colorbar = True)
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e809835848>
```



In [17]:

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

Out[17]:

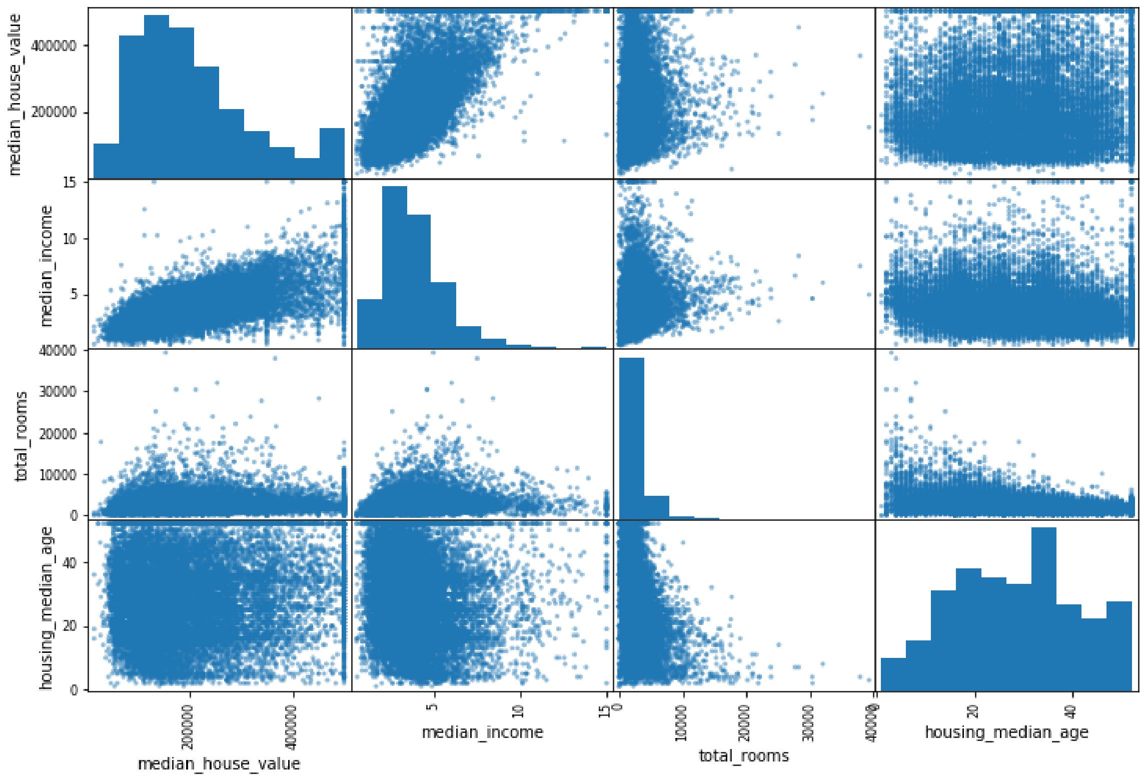
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

In [18]:

```
from pandas.plotting import scatter_matrix
attribute = ["median_house_value", "median_income", "total_rooms", "housing_median_age"
]
scatter_matrix(housing[attribute], figsize = (12,8))
```

Out[18]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001E8013337C
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E80971804
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E80974518
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E80977528
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E80979F38
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E8097D84C
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809906F8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809A9960
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E809AA520
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809ADD40
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809B4594
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809B7CA0
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E809BB6B0
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809BEEC0
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809C27D4
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001E809C5FE8
8>]],
      dtype=object)
```



In [19]:

```
#try out different combinations
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"]
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[19]:

```
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

In [20]:

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

#filling missing values
from sklearn.preprocessing import Imputer
imputer = Imputer(strategy="median")
#copy of data with only numeric attribute
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
#median of each attribute is stored in statistics_ instance
print(imputer.statistics_)
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
```

```
[-118.51     34.26     29.      2119.5      433.      1164.       408.
    3.5409]

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:6
6: DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated
in version 0.20 and will be removed in 0.22. Import impute.SimpleImputer f
rom sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

1. Estimators -> Any object that can estimate some parameters based on datasets.(eg imputer) Estimation is performed by fit() method. Parameter : only a single dataset(and a second dataset containing labels in case of supervised) any other parameter needed to guide the estimation process is considered a hyperparameter.
2. Transformers -> Some estimators can also transform a dataset. It is performed by transform() method with a dataset to transform as a parameter. fit_tranform() : fit and then transform.
3. Predictors -> Some estimators are capable of making predictions given the dataset. Predictor has a predict() method that takes dataset of new instances and return dataset of corresponding predictions. score() method gives quality of prediction.

Text and categorical attribute

In [21]:

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

```
[0 0 4 ... 1 0 3]
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

In [22]:

```
#one-hot encoding is a method to take one label (eg Island) as hot(1) and other all the
other labels as cold(0).
#this is used so that ml algo doesn't assume that 2 nearby valyes are more similar.
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
#housing_cat_1hot is sparse matrix not a numpy array... useful when we have categorical
attribute with thousands of categopries.
print(housing_cat_1hot.toarray())
```

```
[[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.]]

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoder
s.py:415: FutureWarning: The handling of integer data will change in versi
on 0.22. Currently, the categories are determined based on the range [0, m
ax(values)], while in the future they will be determined based on the uniq
ue values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the c
ategories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```

In [23]:

```
#we can directly change from text categorical to one-hot
from sklearn.preprocessing import LabelBinarizer
encoder = LabelBinarizer()
housing_cat_1hot = encoder.fit_transform(housing_cat)
#this directly gives us numpy array ... we can store sparse matrix instead by passing s
parse_output = True to labelbinarizer constructor.
print(housing_cat_1hot)
```

```
[[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]]
```

Custom Transformers

Our own transformers for tasks such as cleanup operations or combining specific attributes.

- Create a class and implement 1. fit() : returning self 2. transform() and 3. fit_transform() we can get last one by adding TransformerMixin as a base class.
- If we add BaseEstimator as base class, we will get 2 extra methods (get_params() and set_params()), that will be useful for automatic hypermeter tuning.

In [24]:

```
np.c_[np.array([1,2,3]),np.array([4,5,6])]
```

Out[24]:

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [25]:

```
#transformer class that adds the combined attributes
from sklearn.base import BaseEstimator, TransformerMixin
rooms_ix, bedroom_ix, population_ix, household_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, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        #3rd column of each row / 6th column of each row.
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedroom_ix] / X[:, rooms_ix]
            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 is numpy array of values

# eg-
print(np.c_[np.array([1,2,3]),np.array([4,5,6])])
print(np.c_[np.array([[1,2,3]]),np.array([[4,5,6]])])

#add_bedrooms_per_room is hyperparameter.
#we can keep this hyperparameter true or false to find out if adding this parameter is
 helping ml algo or not.
```

```
[[1 4]
 [2 5]
 [3 6]]
[[1 2 3 4 5 6]]
```

Feature Scaling

Algo dont perform well when input numerical attributes have very different scales (eg. Total number of rooms : 6-39320, median inccome : 0-15). Two types of scaling -

1. min-max scaling

Value end up ranging from 0-1. Transformer in sklearn - MinMaxScaler. feature_range - hyperparameter in MinMaxScaler to change the range from 0-1.

1. standardization

subtract the mean and divide by variance(0 mean and unit variance). not affected by outliers unlike min-max scaling. Transformer in sklearn - StandardScaler

Fit the Scalers to the training data only, not to the whole dataset. Training data will use fit_transform while Testing dataset will only go through transform.

Transformation Pipelines

In [26]:

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

num_pipeline = Pipeline([
    ('imputer', Imputer(strategy = 'median')),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_sacler', StandardScaler())
])
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:6
6: DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated
in version 0.20 and will be removed in 0.22. Import impute.SimpleImputer f
rom sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Pipeline constructor takes a list of name/estimator pairs. Name can not contain "__"

The last estimator must be a transformer(must have a fit_transform() method)

1. when we call pipeline's fit() method - fit_transform is sequentially called on all transformers, while fit() is called on the final estimator.
2. when we call pipeline's fit_transform method - fit_transform is also called on the final transformer.

In [27]:

```
# cutom transformer to extract the numerical column into a numpy array and directly fee
d the pandas dataframe into the pipeline.
from sklearn.base import BaseEstimator, TransformerMixin

class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [28]:

```
#Build a custom labelBinarizer as error was encountered (error : fit_transform() takes
 2 positional arguments but 3 were given)
class CustomLabelBinarizer(BaseEstimator, TransformerMixin):
    def __init__(self, sparse_output=False):
      self.sparse_output = sparse_output
    def fit(self, X, y=None):
      self.enc = LabelBinarizer(sparse_output=self.sparse_output)
      self.enc.fit(X)
      return self
    def transform(self, X, y=None):
      return self.enc.transform(X)
```

In [29]:

```
#we can also select the categorical data and apply LabelBinarizer.

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

num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attribs)),
    ('imputer', Imputer(strategy = 'median')),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_sacler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attribs)),
    ('label_binarizer', CustomLabelBinarizer())
])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:6
6: DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated
in version 0.20 and will be removed in 0.22. Import impute.SimpleImputer f
rom sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

In [30]:

```
#join 2 pipelines into a single pipeline.
#when fit or transform method is called on the pipeline then the method runs on each tr
ansformers fit or transform method parallely.

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)
print(housing_prepared.shape)
housing_prepared
```

```
(16512, 16)
```

Out[30]:

```
array([[-1.15604281,  0.77194962,  0.74333089, ...,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, ...,  0.        ,
         0.        ,  1.        ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, ...,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, ...,  0.        ,
         1.        ,  0.        ]])
```

Training and evaluating

In [31]:

```
#Linear Regression Model

from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

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 : [210644.60459286 317768.80697211 210956.43331178  59218.9888
6849
 189747.55849879]
Labels : [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

In [32]:

```
#Evaluate LinearRegression model

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)
print(lin_rmse)
```

```
68628.19819848922
```

In [33]:

```
#rmse value is not avoidable, lets try something else
# Decision Tree Model (Non- linear)
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
#Evaluate
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
print(tree_rmse)
```

```
0.0
```

Either our model is too perfect or it overfitting the data.

We cant use the test set until our model is finalized, so we will go for using validation set(or more better "cross validation")

Cross Validation

k- folds cross validation - 10 subsets called folds. Trains and evaluates the model 10 times, 9 folds for training and 1 fold for validation.

pros:

we get estimate of the performance and how precise the performance is (mean and sd of the results) later information is not available in case of simple validation set.

cons:

[illegible]

In [34]:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels, scoring = "neg_mea
n_squared_error", cv = 10) #10 folds
tree_rmse_scores = np.sqrt(-scores)

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)

#cross validation feature has utility function (greater the better) rather than cost fu
nction(smaller the better)
#scoring function is actually opposite of MSE i.e a negative value.
```

In [35]:

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

display_scores(tree_rmse_scores)
display_scores(lin_rmse_scores)
```

```
scores: [68474.31123028 66732.27846603 71960.52187204 69246.46017084
 71462.28268759 74155.52862137 70531.3469805  69701.34693236
 75777.38470993 72065.21043796]
mean: 71010.66721088756
std deviation: 2542.1289642409265
scores: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
mean: 69052.46136345083
std deviation: 2731.6740017983434
```

This means that the linear regression model performed better than decision tree model.

Random Forest

Random forest works by training many decision trees on subsets of features.

Ensemble Learning - Building a model on top of many other models.

In [36]:

```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels, scoring = 
"neg_mean_squared_error", cv =10)
forest_rmse_scores = np.sqrt(-forest_scores)

display_scores(forest_rmse_scores)

#Evaluation on training data
forest_reg.fit(housing_prepared, housing_labels)
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
print(forest_rmse)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
scores: [52739.70181785 50402.38370928 52078.2839059  55092.17721783
 52115.52156719 55700.83238335 51778.48999169 51671.51840845
 56389.81851209 53368.8617289 ]
mean: 53133.75892425266
std deviation: 1866.1161091599724
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
22041.74299667929
```

The performance is still better on the training set, i.e. the model is still overfitting the data.

1. Reguralize the data, with more training data try tweaking the hyperparameters.
2. Try different algorithms such as SVMs and Neural Networks.

Saving The Model

In [37]:

```
from sklearn.externals import joblib
joblib.dump(lin_reg, "linear_regression.pkl")

#loading model from saved pkl file
lin_reg = joblib.load("linear_regression.pkl")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\externals\joblib\__init
__.py:15: DeprecationWarning: sklearn.externals.joblib is deprecated in 0.
21 and will be removed in 0.23. Please import this functionality directly
from joblib, which can be installed with: pip install joblib. If this warn
ing is raised when loading pickled models, you may need to re-serialize th
ose models with scikit-learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

Fine Tune Your Model

Grid Search

In [38]:

```
#To get the best combination of hyperparameters values.
#generic value for a hyperparameter - power of 10 or smaller value for more fine graine
d search

from sklearn.model_selection import GridSearchCV
param_grid = [
    #different combinations to try cross validation
    {'n_estimators':[3,10,30], 'max_features':[2,4,6,8]},
    {'bootstrap':[False], 'n_estimators':[3,10], 'max_features':[2,4,3]}
]
forest_reg = RandomForestRegressor()
grid_search = GridSearchCV(forest_reg, param_grid, cv =5, scoring = "neg_mean_squared_e
rror")
grid_search.fit(housing_prepared, housing_labels)
```

Out[38]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='ms
e',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=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='warn', n_jobs=N
one,
                                             oob_score=False, random_state
=None,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 4, 3],
                          'n_estimators': [3, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring='neg_mean_squared_error', verbose=0)
```

In [39]:

```
#best combination of parameters
print(grid_search.best_params_)
#best estimator
print(grid_search.best_estimator_)
#evaluation scores
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
{'max_features': 8, 'n_estimators': 30}
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features=8, max_leaf_nodes=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)
64507.15578935048 {'max_features': 2, 'n_estimators': 3}
55627.8102486769 {'max_features': 2, 'n_estimators': 10}
53006.72417206441 {'max_features': 2, 'n_estimators': 30}
59830.25230879779 {'max_features': 4, 'n_estimators': 3}
52613.14781308339 {'max_features': 4, 'n_estimators': 10}
50444.86970996411 {'max_features': 4, 'n_estimators': 30}
58774.25557514167 {'max_features': 6, 'n_estimators': 3}
52211.60906347798 {'max_features': 6, 'n_estimators': 10}
49995.423729310445 {'max_features': 6, 'n_estimators': 30}
59127.20119586027 {'max_features': 8, 'n_estimators': 3}
52212.736503064545 {'max_features': 8, 'n_estimators': 10}
49839.258444375075 {'max_features': 8, 'n_estimators': 30}
62430.59203112265 {'bootstrap': False, 'max_features': 2, 'n_estimators':
3}
54275.44886856527 {'bootstrap': False, 'max_features': 2, 'n_estimators':
10}
58821.09911069408 {'bootstrap': False, 'max_features': 4, 'n_estimators':
3}
51963.23943012689 {'bootstrap': False, 'max_features': 4, 'n_estimators':
10}
60684.19820060283 {'bootstrap': False, 'max_features': 3, 'n_estimators':
3}
52604.18454555106 {'bootstrap': False, 'max_features': 3, 'n_estimators':
10}
```

Randomized Search

- use RandomisedSearchCV instead of GridSearchCV.
- Random combinations of hyperparameters by setting fix number of itterations.

Ensemble Methods

- combine the models that performs best.

Analyze the best Model

In [40]:

```
# Use grid search for finding outliers, missing features and feature selection
# use it to add a new feature you are not sure about eg - add_bedrooms_per_room hyperpa
rameter

feature_importances = grid_search.best_estimator_.feature_importances_
print(feature_importances)
# importance scores next to their corresponding attributes names
extra_attribs = ["rooms_per_hhold","pop_per_hhold", "bedrooms_per_room"]
cat_one_hot_attribs = list(encoder.classes_)
attributes = num_attribs+extra_attribs+cat_one_hot_attribs
print(sorted(zip(feature_importances, attributes), reverse = True))

#only one ocean proximity category is really important, we can drop others.
```

```
[6.58682175e-02 6.15241989e-02 4.27366995e-02 1.65659102e-02
 1.45825389e-02 1.54746059e-02 1.37237430e-02 3.59617057e-01
 4.05110206e-02 1.14017455e-01 7.78095970e-02 9.97100129e-03
 1.61988271e-01 6.90106656e-06 1.94473284e-03 3.65805029e-03]
[(0.35961705686361456, 'median_income'), (0.1619882710421918, 'INLAND'),
(0.11401745505599106, 'pop_per_hhold'), (0.07780959702315834, 'bedrooms_pe
r_room'), (0.06586821752458845, 'longitude'), (0.06152419888855334, 'latit
ude'), (0.04273669951870106, 'housing_median_age'), (0.04051102056781711,
'rooms_per_hhold'), (0.016565910174223288, 'total_rooms'), (0.015474605938
59778, 'population'), (0.01458253888494001, 'total_bedrooms'), (0.01372374
3026926175, 'households'), (0.009971001294784832, '<1H OCEAN'), (0.0036580
50290143026, 'NEAR OCEAN'), (0.0019447328392128377, 'NEAR BAY'), (6.901066
556302453e-06, 'ISLAND')]
```

Evaluate on Test Set

In [41]:

```
# get predictors and labels from test dataset.
# use transsform() (not fit_transform) of full_pipeline on test dataset.

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)
final_predictions = final_model.predict(X_test_prepared)

#Final Evaluation
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
print(final_rmse)

#Dont tweak the parameters and hyperparameters to get better result to avoid generaliza
tion.
```

```
48000.188354320075
```