Homework 1: Chapter 2 of Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow 2nd Edition

Author: Kyle Kolodziej

- Source : https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ch02.html#idm45022192813432 (https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ch02.html#idm45022192813432)

Working with the California Housing Data set to predict median housing prices...

Part I: Big Picture

Step 1: Frame the Problem

The book covers the importance of defining the problem and how your answer will be used. This will be essential in being able to determine what algorithms and performance metrics used. The data will be used in a sequence of data processing components (data pipeline).

Another important thing is to look at the current solution (if there is one in place). For this scenario, there is a group of experts than manually estimate the median housing prices.

In terms of desiging the system, this would be a supervised univariate regression problem. Supervised because the data is labeled. Univariate regression because we are trying to predict one value for a median housing price. The data isn't super large so batch learning should work fine.

Step 2: Select a Performance Measure

One common performance metric for regression problems is Root Mean Squared Error which puts an emphasis on larger errors. Another common metric is Mean Absolute Error. This metric is better suited for data with more outliers.

RMSE - euclidean norm (l2) which is distance MAE - manhattan norm (l1) which is ditance between two points but can only travel orthogonally

Moving up in norm levels increases the sensitivity to outliers...more outliers = use lower norm measure

Step 3: Check Assumptions

Example used of knowing how the data will be used with categories vs. specific values needed for housing prices.

Part II: Get the Data

In [15]:
```
import os
import tarfile
import urllib

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"

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")
    print(tgz_path)
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

In [16]:
```
import pandas as pd

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

In [17]:
```
housing = load_housing_data()
housing.head()
```

Out[17]:

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

Data read in nicely following code from the book...now let's take a look at the number of instances

In [18]:

```
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
```

There are some null values for total_bedrooms with only 20,433 non-null vs. the rest having 20,640

In [20]:

```
housing['ocean_proximity'].unique()
```

Out[20]:

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

In [19]:

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

Out[19]:

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

Categorical data for the ocean_proximity feature with 5 different categories

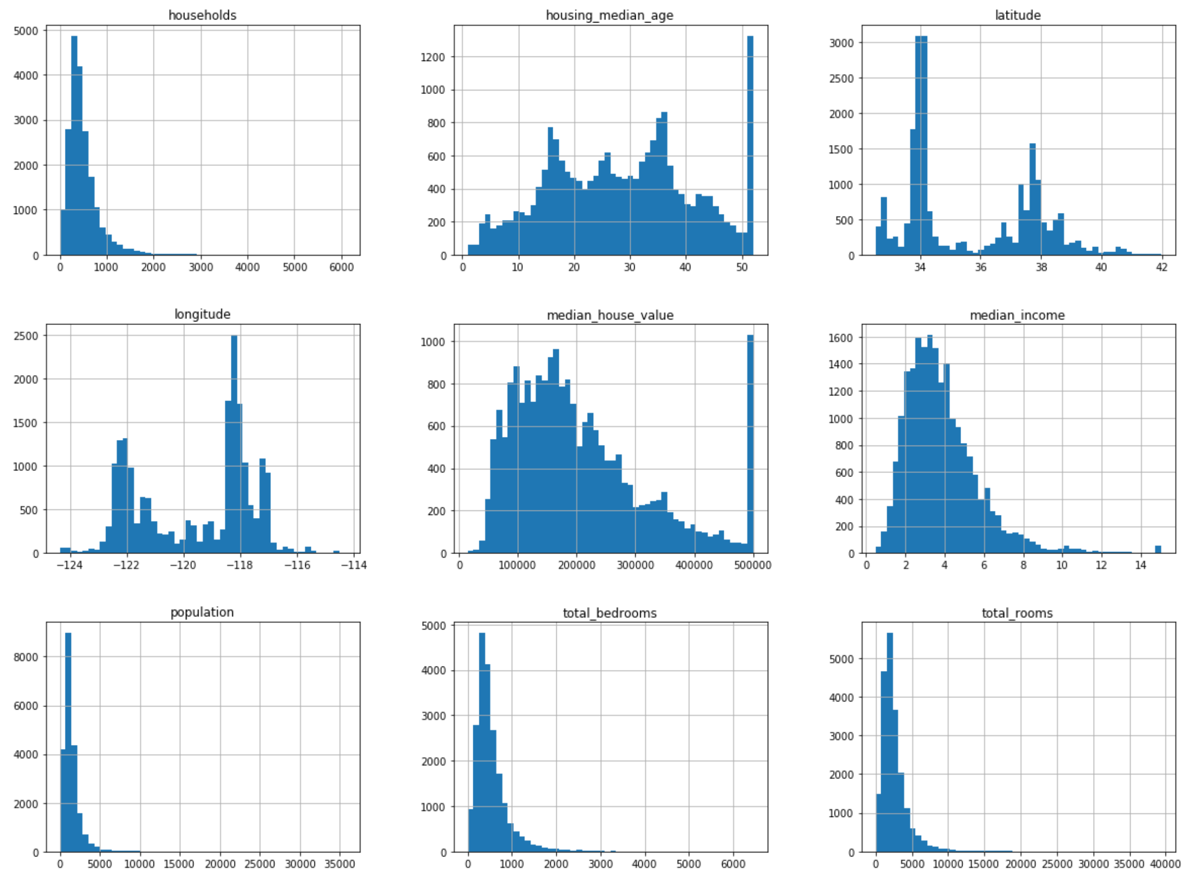
In [21]:

```
housing.describe()
```

Out[21]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.00000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.47674
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.46212
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.00000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.00000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.00000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.00000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.00000

In [25]:
```
import matplotlib.pyplot as plt
plt = housing.hist(bins=50, figsize=(20,15))
```



Some takeaways from the plots...

- median_income is not in terms of USD...actually scaled from 0.5 to 15
- median house value and age were capped in the dataset. Either don't need to worry about predicting these values precisely beyond their capped values (500k) or will need to remove them from the training and testing set
- most of the features are tail heavy with a lot of the data skewed to the right of each features peak

Test Set

It is time to set aside data for a testing set to avoid data snooping...

In [26]:
```
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]
```

In [29]:
```
train_set, test_set = split_train_test(housing, 0.2)
```

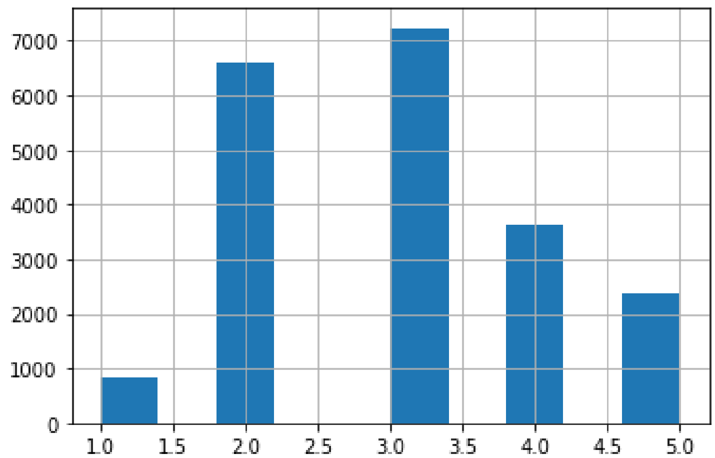
In [30]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

In [31]:
```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

In [32]:
```
housing["income_cat"].hist()
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28effba9648>
```



Keep instances the same for median income categories with the stratified shuffle spilt...

In [33]:
```
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 [34]:
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

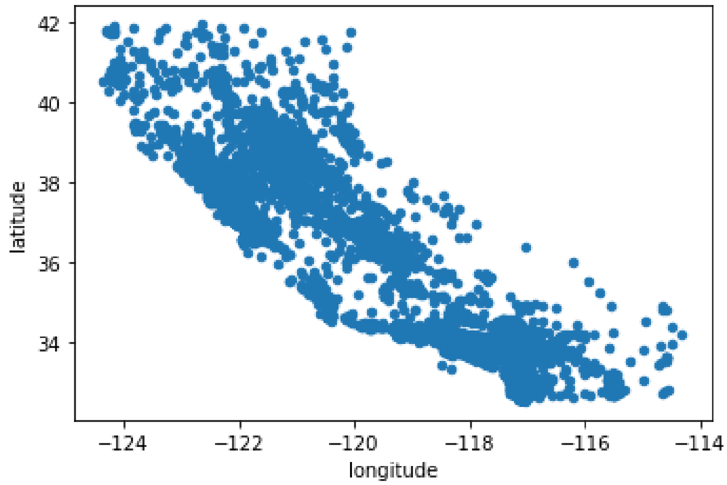
Out[34]:
```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

In [35]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

In [36]:
```
housing = strat_train_set.copy() # Create a copy to play around with it and make v
```

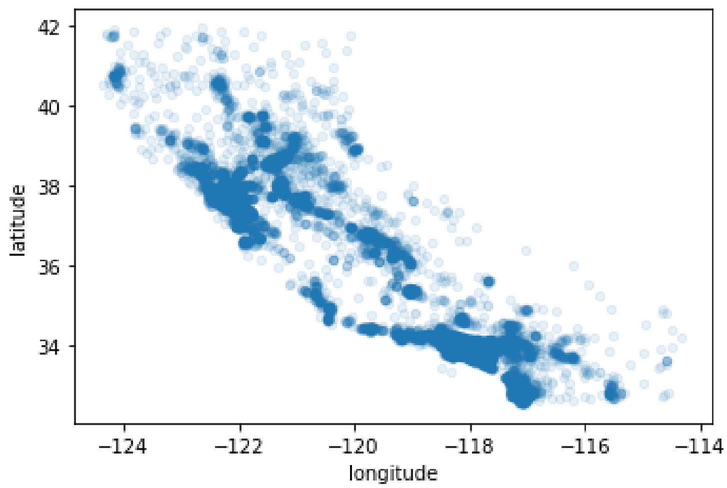
In [37]:
```
housing.plot(kind="scatter", x="longitude", y="latitude")
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x28eff466248>



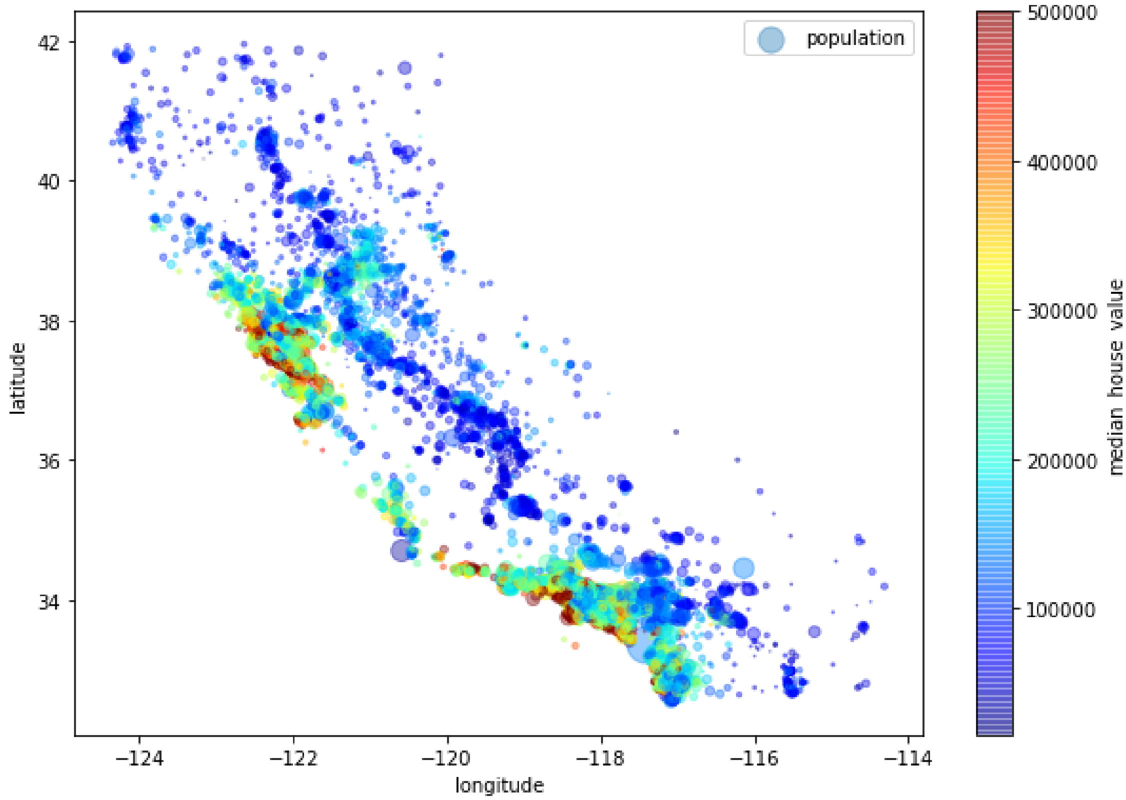
In [38]:
```
# adding on alpha option to help show where there is a high density of points
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x28efd2db948>



In [43]:
```
# circle radius corresponds to population size
# colors for median housing price
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="jet", colorbar=True,
             sharex=False)
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x28e884835c8>



In [44]:
```
corr_matrix = housing.corr()
```

In [45]:
```
# let's see the correlation to the median housing value
corr_matrix["median_house_value"].sort_values(ascending=False)
```

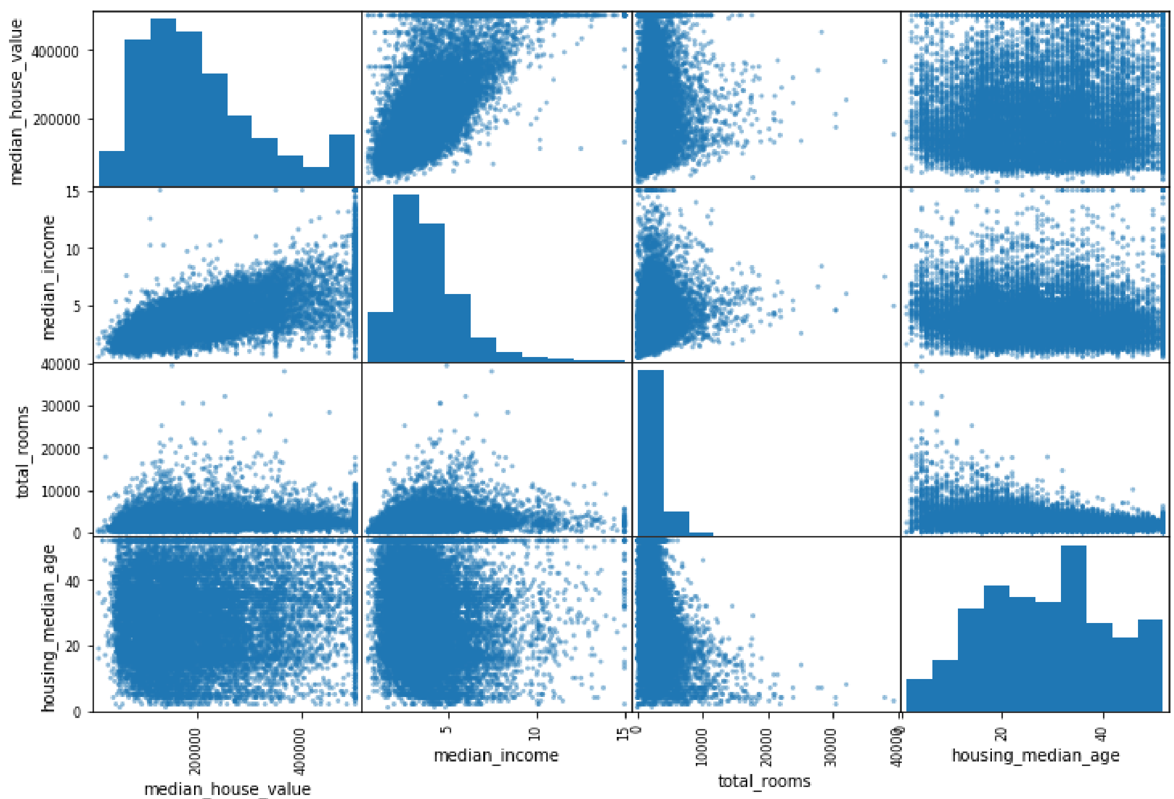
Out[45]:
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

In [46]:

```
from pandas.plotting import scatter_matrix
# just look at some of the most correlated factors
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```

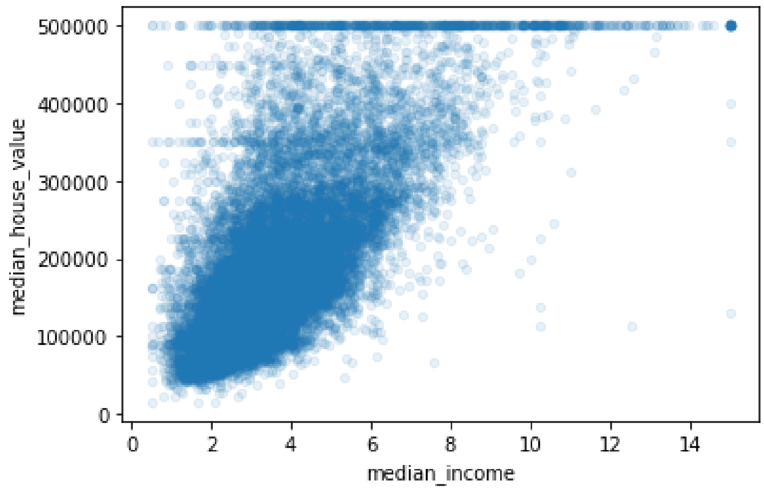
Out[46]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000028E8852D088>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E888148C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89D05AC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89E7E148>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89EB1748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89EE9448>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89F1F1C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89F58208>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89F5EDC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E89F96FC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E8A002548>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E8A040E88>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000028E8A070708>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E8A0A9808>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E8A0E2948>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000028E8A11CA48>]],
      dtype=object)
```



In [47]:
```
# just look at income vs housing price
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28e8a508c88>
```



Can see the capped median housing value line clearly with this plot

Cleaning up the data

- Tail heavy distribution --> transform by computing their logaritm
- Number of rooms not useful --> get number of rooms per house .... apply to other similar scenarios

In [48]:
```
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 [49]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[49]:
```
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
```

The transformed data of rooms per household (slightly) and bedrooms per room (more so) show stronger correlation to the median housing value after those transformations

Time for the Machine Learning Algo

In [50]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

Dealing with the missing values seen earlier with total bedrooms...

- Get rid of the corresponding districts
- Get rid of the whole attribute
- Set the values to some value (zero, the mean, the median, etc.).

In [51]:
```
"""
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)
"""
```

Out[51]:
```
'\nhousing.dropna(subset=["total_bedrooms"])    # option 1\nhousing.drop("total_
bedrooms", axis=1)       # option 2\nmedian = housing["total_bedrooms"].median()
# option 3\nhousing["total_bedrooms"].fillna(median, inplace=True)\n'
```

In [53]:
```
# Using sklearn to replace the missing values with the median value
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

In [54]:
```
housing_num = housing.drop("ocean_proximity", axis=1) # can only perform on numer
```

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

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

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

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

In [57]:
```
housing_num.median().values
```

Out[57]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

In [58]:
```
X = imputer.transform(housing_num)
```

In [59]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

Dealing with Categorical Data

In [60]:
```
housing_cat = housing[["ocean_proximity"]]
```

In [61]:
```
housing_cat.head(10)
```

Out[61]:

	ocean_proximity
17606	<1H OCEAN
18632	<1H OCEAN
14650	NEAR OCEAN
3230	INLAND
3555	<1H OCEAN
19480	INLAND
8879	<1H OCEAN
13685	INLAND
4937	<1H OCEAN
4861	<1H OCEAN

In [62]:
```
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

Out[62]:
```
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

In [63]:
```
ordinal_encoder.categories_
```

Out[63]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

In [64]:
```
# problem with values for the ordinal encoder being interpreted like numerical dat
# with closer numbers being closer in correlation....so OHE
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[64]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
	with 16512 stored elements in Compressed Sparse Row format>
```

Note: sparse matrix because with all the rows/cols needed to store the 1's and 0's would take up a lot of unnecesary space. Instead, sparse matrix to just store the location of the 1's

In [65]:

```
# transformer with the transformations used earlier for per household added in
from sklearn.base import BaseEstimator, TransformerMixin

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  # nothing else to do
    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_per_room]

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

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

Feature Scaling

Min Max Scaling

- Values are shifter and rescaled from 0 to 1 (can change the feature range if needed) by: (actual - min) / (max - min)
- Outliers can have a big impact

Standardization

- Zero mean value and unit variance
- Values are shifted and rescaled by: (actual - mean) / (standard dev)
- No specific range for this one (could have problems for neural networks expecting 0 - 1)
- Outliers don't have as big of an impact

Transformation Pipelines

Transformations need to be run in a specific order...scikit-learn Pipeline class helps with this

In [66]:
```
# Pipeline for numerical attributes
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

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

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

In [67]:
```
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),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Training a Model

In [68]:
```
from sklearn.linear_model import LinearRegression

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

Out[68]:
```
LinearRegression()
```

In [70]:
```
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  59218.98886849
 189747.55849879]
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

In [71]:
```
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[71]:
```
68628.19819848923
```

Prediction error of $68,628 isn't great

Let's try a more complex model to see how it works

In [72]:
```
from sklearn.tree import DecisionTreeRegressor

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

Out[72]:
```
DecisionTreeRegressor()
```

In [73]:
```
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[73]:
```
0.0
```

No error at all...seems like it has to be overfit for this model

Cross Validation

In [74]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

In [76]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [69770.83809053 67850.08930405 72550.36893596 69336.80900428
 71212.0581407  74900.56403306 71045.04499323 70865.4247175
 77214.75294503 70866.3466747 ]
Mean: 71561.22968390491
Standard deviation: 2595.637923619615
```

In [77]:
```
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)
display_scores(lin_rmse_scores)
```

```
Scores: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.6740017983484
```

Cross validation shows how the decision tree doesn't even do as good as linear regression

Now going to try a random forest

In [79]:
```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
```

Out[79]:
```
RandomForestRegressor()
```

In [82]:
```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[82]:
```
18684.689942482844
```

In [83]:
```
from sklearn.model_selection import cross_val_score

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)
```

```
Scores: [49455.0856104  47577.35429796 50329.01264255 52159.23496585
 49478.72824828 53202.69354813 48842.77685549 48053.38267675
 52520.11533211 50149.23404446]
Mean: 50176.76182219745
Standard deviation: 1807.0680155246348
```

In [84]:
```
# Saving and Loading a model code for future reference
"""
import joblib

joblib.dump(my_model, "my_model.pkl")
# and later...
my_model_loaded = joblib.load("my_model.pkl")
"""
```

Out[84]:
```
'\nimport joblib\n\njoblib.dump(my_model, "my_model.pkl")\n# and later...\nmy_mo
del_loaded = joblib.load("my_model.pkl")\n'
```

Improving the Model

- Grid Searcg
- Randomized Search
- Ensemble Methods

Grid Search

Tune the hyperparameters with scitkit-learn's GridSearchCV which will try out hyperparameters you give it

In [85]:
```
from sklearn.model_selection import GridSearchCV

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

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(housing_prepared, housing_labels)
```

Out[85]:
```
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 [86]:
```
grid_search.best_params_
```

Out[86]:
```
{'max_features': 6, 'n_estimators': 30}
```

In [87]:
```
grid_search.best_estimator_
```

Out[87]:
```
RandomForestRegressor(max_features=6, n_estimators=30)
```

In [88]:
```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
63995.252619978695 {'max_features': 2, 'n_estimators': 3}
55921.23857861423 {'max_features': 2, 'n_estimators': 10}
52706.75642022026 {'max_features': 2, 'n_estimators': 30}
61006.119057123506 {'max_features': 4, 'n_estimators': 3}
52755.72920814952 {'max_features': 4, 'n_estimators': 10}
50775.57431360015 {'max_features': 4, 'n_estimators': 30}
59645.996739704235 {'max_features': 6, 'n_estimators': 3}
52045.06255301316 {'max_features': 6, 'n_estimators': 10}
50019.04166115086 {'max_features': 6, 'n_estimators': 30}
59310.7572881733 {'max_features': 8, 'n_estimators': 3}
51887.01625926025 {'max_features': 8, 'n_estimators': 10}
50089.50441911475 {'max_features': 8, 'n_estimators': 30}
62576.12414506158 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54042.37812436334 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
58582.33216902641 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52567.951542242794 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58718.007690010505 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51847.35904551882 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Analyzing the Model

In [89]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[89]:
```
array([8.15276897e-02, 7.48700125e-02, 4.40150034e-02, 1.71422426e-02,
       1.63114573e-02, 1.79170824e-02, 1.62568606e-02, 3.07391644e-01,
       6.56521157e-02, 1.10918850e-01, 7.63181034e-02, 8.16982568e-03,
       1.56173850e-01, 8.89740616e-05, 3.22882697e-03, 4.01746156e-03])
```

In [90]:
```
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
sorted(zip(feature_importances, attributes), reverse=True)
```

Out[90]:
```
[(0.30739164404202046, 'median_income'),
 (0.1561738504088931, 'INLAND'),
 (0.11091884973925017, 'pop_per_hhold'),
 (0.08152768966798245, 'longitude'),
 (0.0763181033790475, 'bedrooms_per_room'),
 (0.07487001245179165, 'latitude'),
 (0.06565211568098155, 'rooms_per_hhold'),
 (0.04401500343857811, 'housing_median_age'),
 (0.017917082447677837, 'population'),
 (0.017142242576800543, 'total_rooms'),
 (0.016311457332505563, 'total_bedrooms'),
 (0.016256860568338572, 'households'),
 (0.008169825676777123, '<1H OCEAN'),
 (0.004017461560324831, 'NEAR OCEAN'),
 (0.0032288269673980924, 'NEAR BAY'),
 (8.89740616324666e-05, 'ISLAND')]
```

Evaluate on the Test Set

In [91]:
```
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_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [92]:
```
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[92]:
```
array([45673.88103211, 49546.24423198])
```

Next Steps

- Presenting the model: how it was made, steps taken (both good and bad), assumptions, performance
- Deploying model to be used via the cloud or application to be used
- Monitor the performance of the model

In [ ]: