Import Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler, normalize,Imputer,La
belEncoder
import warnings
import seaborn as sb
warnings.filterwarnings
%matplotlib inline
```

Step 1: Load the data

In [2]:
```
ml_dataset=pd.read_csv('housing.csv')
```

Exploring dataset

In [3]:
```
#top rows
ml_dataset.head(15)
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41	880	129.0	322
1	-122.22	37.86	21	7099	1106.0	2401
2	-122.24	37.85	52	1467	190.0	496
3	-122.25	37.85	52	1274	235.0	558
4	-122.25	37.85	52	1627	280.0	565
5	-122.25	37.85	52	919	213.0	413
6	-122.25	37.84	52	2535	489.0	1094
7	-122.25	37.84	52	3104	687.0	1157
8	-122.26	37.84	42	2555	665.0	1206
9	-122.25	37.84	52	3549	707.0	1551
10	-122.26	37.85	52	2202	434.0	910
11	-122.26	37.85	52	3503	752.0	1504
12	-122.26	37.85	52	2491	474.0	1098
13	-122.26	37.84	52	696	191.0	345
14	-122.26	37.85	52	2643	626.0	1212

In [4]:
```
#bottom rows
ml_dataset.tail(10)
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populat
20630	-121.32	39.29	11	2640	505.0	1257
20631	-121.40	39.33	15	2655	493.0	1200
20632	-121.45	39.26	15	2319	416.0	1047
20633	-121.53	39.19	27	2080	412.0	1082
20634	-121.56	39.27	28	2332	395.0	1041
20635	-121.09	39.48	25	1665	374.0	845
20636	-121.21	39.49	18	697	150.0	356
20637	-121.22	39.43	17	2254	485.0	1007
20638	-121.32	39.43	18	1860	409.0	741
20639	-121.24	39.37	16	2785	616.0	1387

In [5]:
```
ml_dataset.shape
```

Out[5]:
```
(20640, 10)
```

In [6]:
```
#we create matrix for dependent variables and independent variables.

x = ml_dataset.iloc [:,:-1].values
y = ml_dataset.iloc [:,9].values
```

In [7]:
```
ml_dataset.dtypes
```

Out[7]:
```
longitude             float64
latitude              float64
housing_median_age      int64
total_rooms             int64
total_bedrooms        float64
population              int64
households              int64
median_income         float64
ocean_proximity        object
median_house_value      int64
dtype: object
```

In [8]:
```
#seeing the first row
ml_dataset.ix[0]
```

```
C:\Users\jmohan200\AppData\Local\Continuum\anaconda3\lib\site-packag
es\ipykernel_launcher.py:2: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer
-is-deprecated
```

Out[8]:
```
longitude               -122.23
latitude                  37.88
housing_median_age           41
total_rooms                 880
total_bedrooms              129
population                  322
households                  126
median_income            8.3252
ocean_proximity        NEAR BAY
median_house_value       452600
Name: 0, dtype: object
```

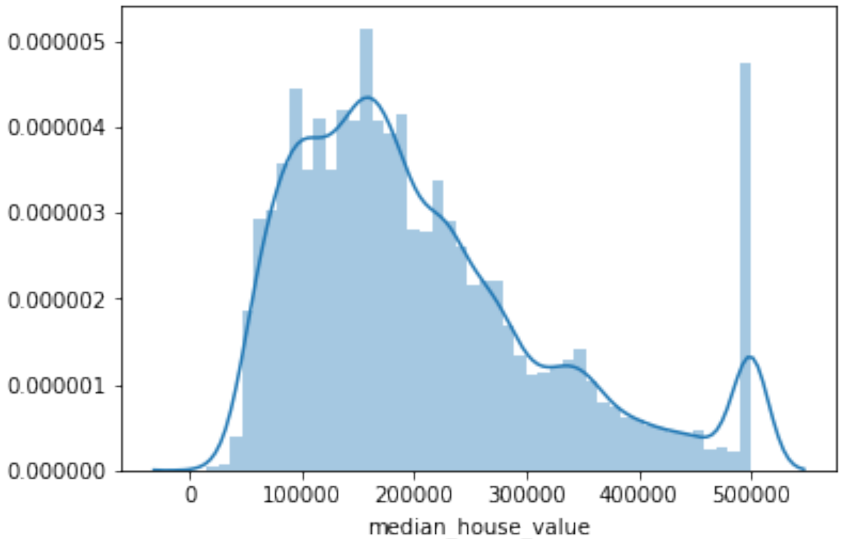
In [9]:
```
#descriptive statistics
ml_dataset.describe()
```

Out[9]:

	longitude	latitude	housing_median_age	total_rooms	total_bedroo
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.00000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000

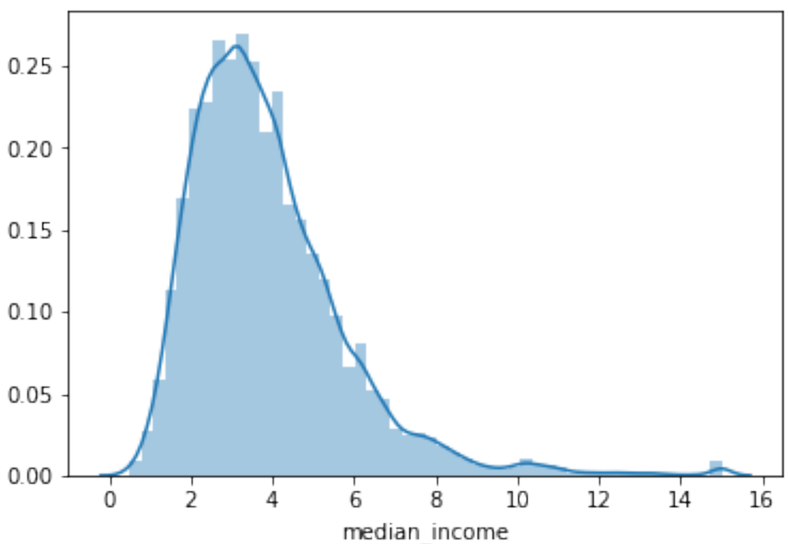
In [10]:
```
#histogram
sb.distplot(ml_dataset['median_house_value'])
```

Out[10]: `<matplotlib.axes._subplots.AxesSubplot at 0x191a675e4a8>`



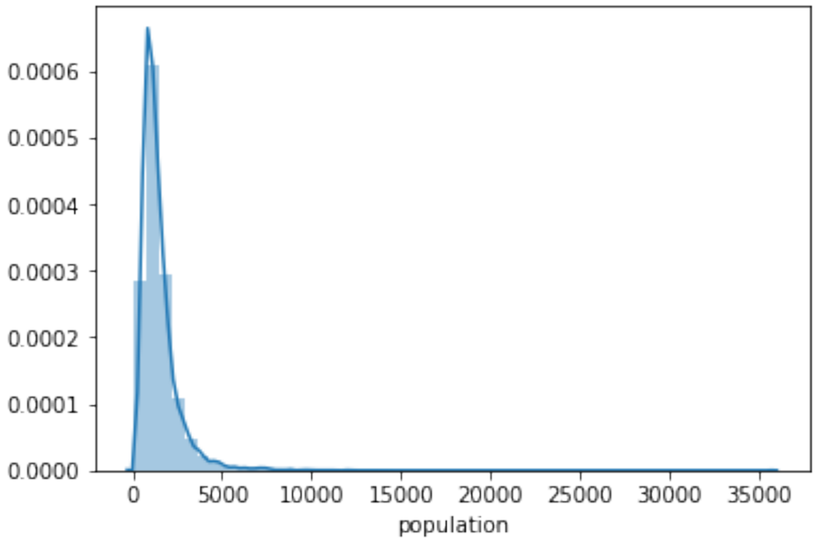
In [11]:
```
sb.distplot(ml_dataset['median_income'])
```

Out[11]: `<matplotlib.axes._subplots.AxesSubplot at 0x191a6fb2668>`



In [12]:
```
sb.distplot(ml_dataset['population'])
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x191a700e9e8>



In [13]:
```
#Skewness and kurtosis
print("Skewness- %f" % ml_dataset['median_house_value'].skew())
print("Kurtosis- %f" % ml_dataset['median_house_value'].kurt())
```

```
Skewness- 0.977763
Kurtosis- 0.327870
```

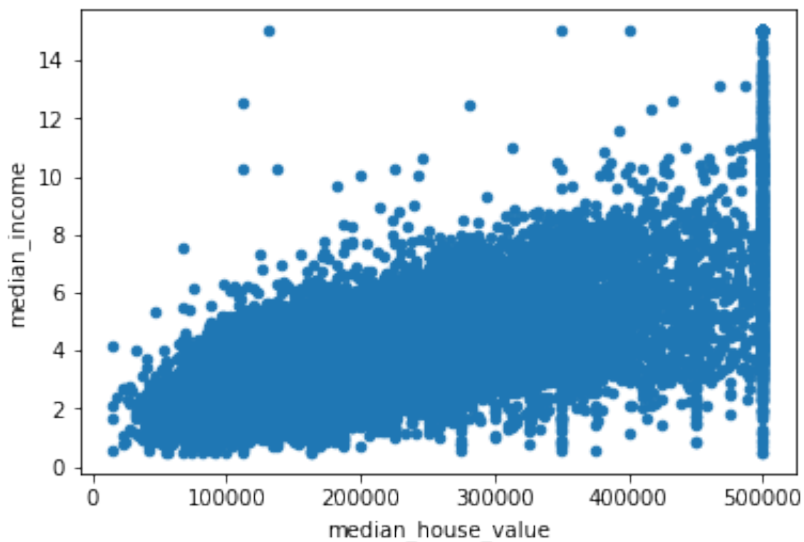
In [14]:
```
print("Skewness- %f" % ml_dataset['median_income'].skew())
print("Kurtosis- %f" % ml_dataset['median_income'].kurt())
```

```
Skewness- 1.646657
Kurtosis- 4.952524
```

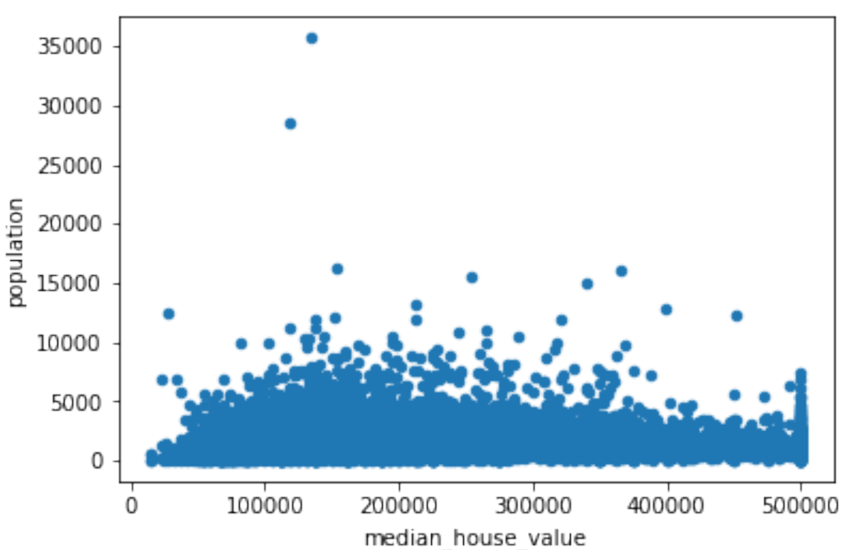
In [15]:
```
print("Skewness- %f" % ml_dataset['population'].skew())
print("Kurtosis- %f" % ml_dataset['population'].kurt())
```

```
Skewness- 4.935858
Kurtosis- 73.553116
```

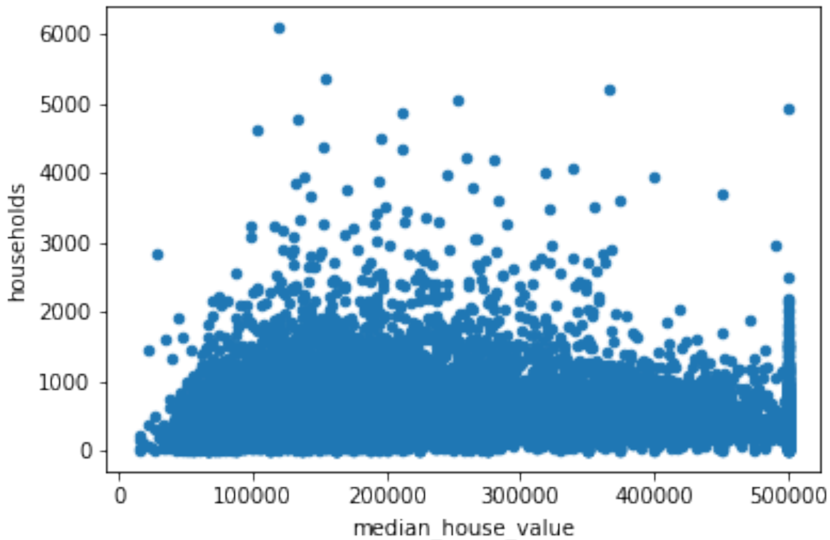
In [16]:
```
#scatterplot
scatterplot = pd.concat([ml_dataset['median_house_value'], ml_dataset[
'median_income']], axis=1)
scatterplot.plot.scatter(x='median_house_value', y='median_income');
```



In [17]:
```
scatterplot = pd.concat([ml_dataset['median_house_value'], ml_dataset[
'population']], axis=1)
scatterplot.plot.scatter(x='median_house_value', y='population');
```



In [18]:
```
scatterplot = pd.concat([ml_dataset['median_house_value'], ml_dataset[
'households']], axis=1)
scatterplot.plot.scatter(x='median_house_value', y='households');
```

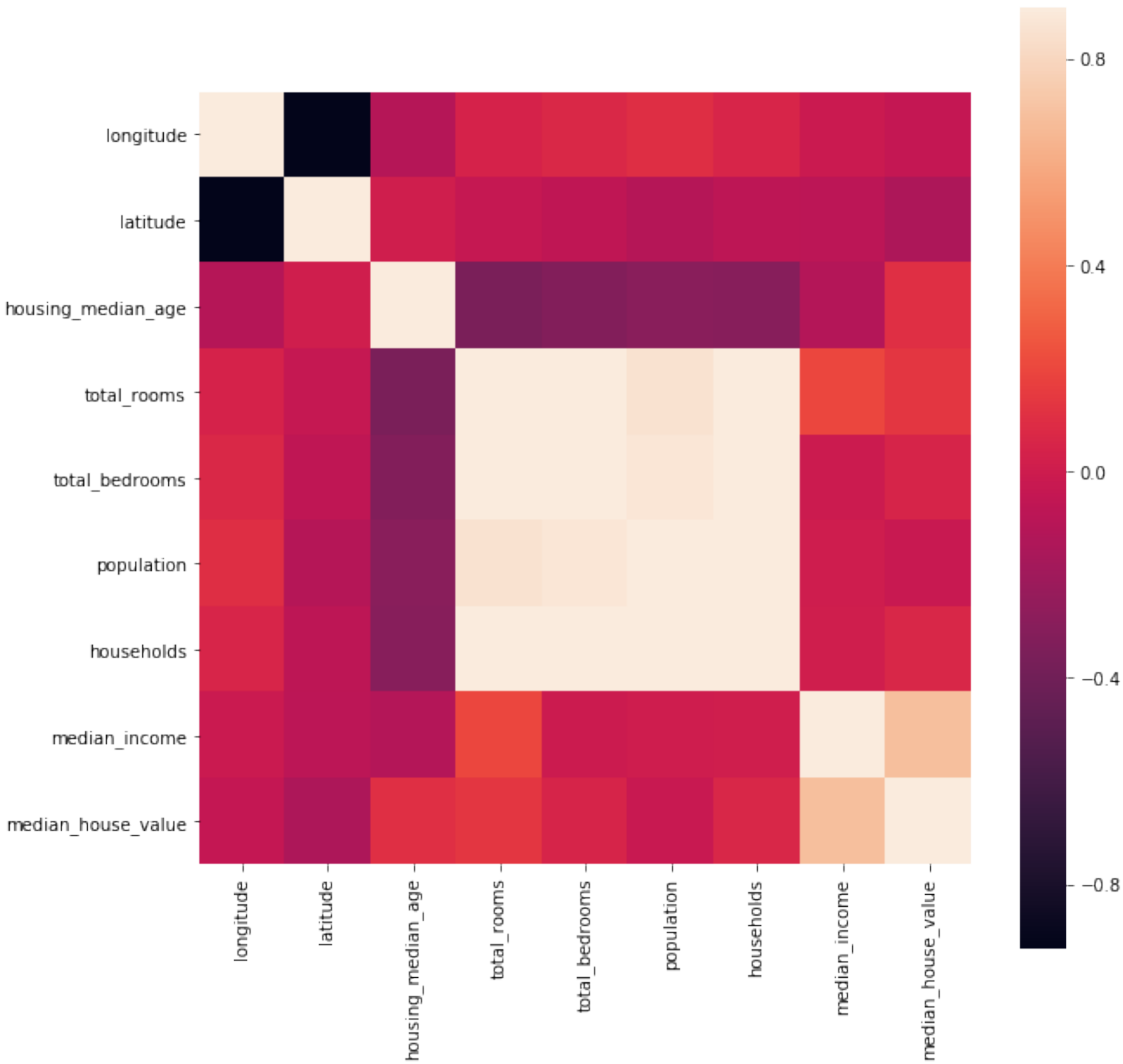


In [19]:
```
#correlations
ml_dataset.corr()
```

Out[19]:

	longitude	latitude	housing_median_age	total_rooms	total_b
longitude	1.000000	-0.924664	-0.108197	0.044568	0.0696(
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.0669
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.3204
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.9303
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.0000(
population	0.099773	-0.108785	-0.296244	0.857126	0.8777
households	0.055310	-0.071035	-0.302916	0.918484	0.9797
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.0077
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.0496

In [20]:
```
corrmat = ml_dataset.corr()
plt.subplots(figsize=(10, 10))
sb.heatmap(corrmat, vmax=.9, square=True);
```



In [21]:
```
#covariance
ml_dataset.cov()
```

Out[21]:

	longitude	latitude	housing_median_age	total_roo
longitude	4.014139	-3.957054	-2.728244	1.948037e+
latitude	-3.957054	4.562293	0.300346	-1.682178e
housing_median_age	-2.728244	0.300346	158.396260	-9.919120e
total_rooms	194.803750	-168.217847	-9919.120060	4.759445e+
total_bedrooms	58.768508	-60.299623	-1700.312817	8.567306e+
population	226.377839	-263.137814	-4222.270582	2.117613e+
households	42.368072	-58.010245	-1457.581290	7.661046e+
median_income	-0.057765	-0.323860	-2.846140	8.208524e+
median_house_value	-10627.425205	-35532.559074	153398.801329	3.377289e+

Step 2: Handle missing value

In [22]:
```
#checking missing value
ml_dataset.isnull().values.any()
```

Out[22]: True

In [23]:
```
ml_dataset.isnull().values.sum()
```

Out[23]: 207

In [24]:
```
 #First, let's count the number of null values
total = ml_dataset.isnull().sum().sort_values(ascending=False)
# Then, let's calculate the percentage of missing data per feature
percent = (ml_dataset.isnull().sum()/ml_dataset.isnull().count()).sort
_values(ascending=False)
# Finally, let's concatenate Total and Percent into another dataframe
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Per
cent'])
missing_data.head(20)
```

Out[24]:

	Total	Percent
total_bedrooms	207	0.010029
median_house_value	0	0.000000
ocean_proximity	0	0.000000
median_income	0	0.000000
households	0	0.000000
population	0	0.000000
total_rooms	0	0.000000
housing_median_age	0	0.000000
latitude	0	0.000000
longitude	0	0.000000

In [25]:
```
#Missing value
# First create an Imputer
missingValueImputer = Imputer (missing_values = 'NaN', strategy = 'mea
n', axis=0)
# Set which columns imputer should perform
missingValueImputer = missingValueImputer.fit (x[:,4:5])
# update values of X with new values
x[:,4:5] = missingValueImputer.transform(x[:,4:5])
```

Step 3: Encode categorical data

In [26]:
```
#encode categorical data
X_labelencoder = LabelEncoder()
x[:, 8] = X_labelencoder.fit_transform(x[:, 8])
print (x)
```

```
[[-122.23 37.88 41 ..., 126 8.3252 3]
 [-122.22 37.86 21 ..., 1138 8.3014 3]
 [-122.24 37.85 52 ..., 177 7.2574 3]
 ...,
 [-121.22 39.43 17 ..., 433 1.7 1]
 [-121.32 39.43 18 ..., 349 1.8672 1]
 [-121.24 39.37 16 ..., 530 2.3886 1]]
```

Step 4: Split the dataset

In [27]:
```
#split the dataset
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split (x, y, test_size =
0.2,
                                                     random_state = 0)
```

```
C:\Users\jmohan200\AppData\Local\Continuum\anaconda3\lib\site-packag
es\sklearn\cross_validation.py:41: DeprecationWarning: This module w
as deprecated in version 0.18 in favor of the model_selection module
into which all the refactored classes and functions are moved. Also
note that the interface of the new CV iterators are different from t
hat of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Step 5: Standardize data

In [28]:
```
#Standardize data
independent_scalar = StandardScaler()
X_train = independent_scalar.fit_transform (X_train)
X_test = independent_scalar.transform (X_test) # only transform
print(X_train)
```

```
[[ 1.00389865 -0.8400624  -1.79507596 ..., -1.1356496   0.19001247
  -0.11814798]
 [-1.43477229  0.98536392  1.85553889 ..., -0.13688171  0.26931072
   1.28686421]
 [ 0.77948108 -0.8400624  -0.20785212 ..., -0.34343319  0.02989505
  -0.82065408]
 ...,
 [-1.1654712   0.44709718  0.18895385 ..., -0.27806879 -0.35589721
   1.98937031]
 [ 0.81439048 -0.93835459  0.42703742 ..., -0.08197562  0.92053182
  -0.82065408]
 [ 1.99632302 -1.32216217 -1.08082523 ..., -0.52645348 -1.30490629
  -0.11814798]]
```

```
C:\Users\jmohan200\AppData\Local\Continuum\anaconda3\lib\site-packag
es\sklearn\utils\validation.py:475: DataConversionWarning: Data with
input dtype object was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Step 6: Perform Linear Regression

In [29]:
```
# Simple Linear Regression to the Training data set set
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
linear_regressor.fit(X_train, y_train)
```

Out[29]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normaliz
e=False)
```

In [30]:
```
# Predicting the Test set results using simple regression
y_linearpredict = linear_regressor.predict(X_test)
y_linearpredict
```

Out[30]:
```
array([ 210776.44901247,  279878.4567539 ,  190478.34412314, ...,
         80625.22422957,  279916.99817768,  207126.99095493])
```

In [31]:
```
#root mean squared error (RMSE) from Linear Regression.

from sklearn.metrics import mean_squared_error
from math import sqrt

Linear_rms = sqrt(mean_squared_error(y_test, y_linearpredict))
Linear_rms
```

Out[31]:
```
69826.89013012727
```

Step 7: Perform Decision Tree Regression

In [32]:
```
#Decision tree regression to the Training data set
from sklearn.tree import DecisionTreeRegressor
decisiontree_regressor= DecisionTreeRegressor(random_state = 0)
decisiontree_regressor.fit(X_train, y_train)
```

Out[32]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=
None,
           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,
           presort=False, random_state=0, splitter='best')
```

In [33]:
```
# Predicting the Test set results using Decision tree
y_decisiontreepredict = decisiontree_regressor.predict(X_test)
y_decisiontreepredict
```

Out[33]:
```
array([ 134800.,  267600.,  160300., ...,  120100.,  250300.,  17390
0.])
```

In [34]:
```
#root mean squared error (RMSE) from Decision tree.

Decisiontree_rms = sqrt(mean_squared_error(y_test, y_decisiontreepredi
ct))
Decisiontree_rms
```

Out[34]:
```
67116.265343338
```

Step 8: Perform Random Forest Regression

In [35]:
```
#Random Forest regression to the Training data set
from sklearn.ensemble import RandomForestRegressor
RFregressor = RandomForestRegressor(n_estimators = 6, random_state = 0
)
RFregressor.fit(X_train, y_train)
```

Out[35]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=Non
e,
           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=6, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=Fa
lse)
```

In [36]:
```
# Predicting the Test set results using Random Forest
y_RFpredict = RFregressor.predict(X_test)
y_RFpredict
```

Out[36]:
```
array([ 146666.66666667,  224966.66666667,  151416.66666667, ...,
        149016.66666667,  201066.66666667,  197666.66666667])
```

In [37]:
```
#root mean squared error (RMSE) from Random forest.

RF_rms = sqrt(mean_squared_error(y_test, y_RFpredict))
RF_rms
```

Out[37]:
```
52898.0450940454
```

Step 9: Perform Linear Regression with one independent variable

In [38]:
```
#One independent variable
Z = ml_dataset.iloc [:,7:8].values
#split the dataset
Z_train, Z_test, y_train, y_test = train_test_split (Z, y, test_size =
0.2,
                                                     random_state = 0)
```

In [39]:
```
#Standardize data
oneindependent_scalar = StandardScaler()
Z_train = oneindependent_scalar.fit_transform (Z_train)
Z_test = oneindependent_scalar.transform (Z_test) # only transform
print(Z_train)
```

```
[[ 0.19001247]
 [ 0.26931072]
 [ 0.02989505]
 ...,
 [-0.35589721]
 [ 0.92053182]
 [-1.30490629]]
```

In [40]:
```
# regression(One independent) variable using testing data

onelinear_regressor = LinearRegression()
onelinear_regressor.fit(Z_train, y_train)
```

Out[40]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normaliz
e=False)
```

In [41]:
```
# Predicting the Test set results using  regression(One independent)
y_onelinearpredict = onelinear_regressor.predict(Z_test)
y_onelinearpredict
```

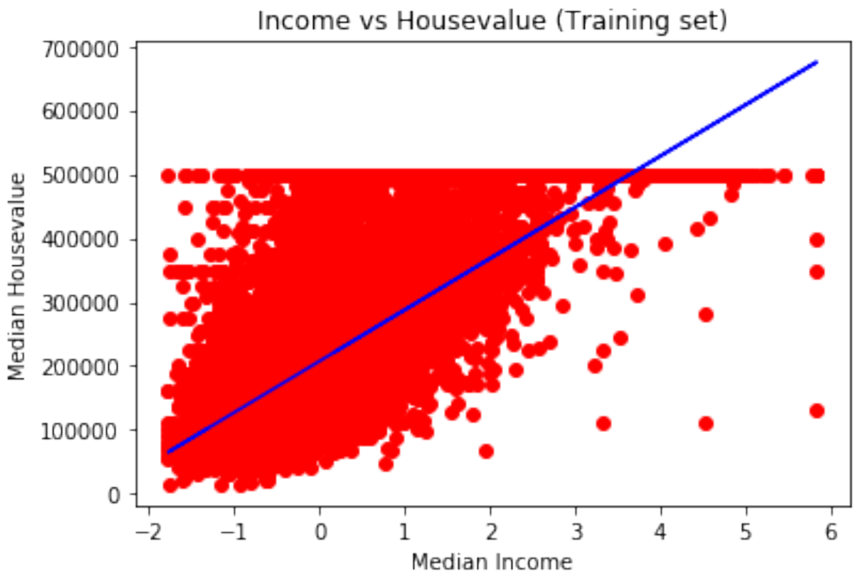
Out[41]:
```
array([ 218829.83059812,  287249.80945645,  227105.96638704, ...,
        178937.09074405,  302549.52213887,  184397.07062714])
```

In [42]:
```
#root mean squared error (RMSE) from Linear Regression.


OneLinear_rms = sqrt(mean_squared_error(y_test, y_onelinearpredict))
OneLinear_rms
```

Out[42]:
```
84941.05152406936
```

In [43]:
```
#Plot the fitted model for training data(one independent)
plt.scatter(Z_train, y_train, color = 'red')
plt.plot(Z_train, onelinear_regressor.predict(Z_train), color = 'blue'
)
plt.title('Income vs Housevalue (Training set)')
plt.xlabel('Median Income')
plt.ylabel('Median Housevalue')
plt.show()
```



In [46]:
```
#Plot the fitted model for t data(one independent)
plt.scatter(Z_test, y_test, color = 'red')
plt.plot(Z_test, onelinear_regressor.predict(Z_test), color = 'blue')
plt.title('Income vs Housevalue (Test set)')
plt.xlabel('Median Income')
plt.ylabel('Median Housevalue')
plt.show()
```