In [1]:

```
#Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Acquire Housing dataset

In [2]:

```
#Read housing dataset
df_housing_dataset = pd.read_csv('F:\SML\housing.csv')
```

In [3]:

```
df_housing_dataset.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
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

In [4]:

```
df_housing_dataset.columns
```

Out[4]:

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity', 'median_house_value'],
      dtype='object')
```

In [5]:

```
#Check shape of entire dataset
df_housing_dataset.shape
```

Out[5]:

```
(20640, 10)
```

In [6]:

```
df_housing_dataset.describe()
```

Out[6]:

	longitude	latitude	housing_median_age	total_rooms	total_bedro
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	6445.00000

Visualize data to understand the relationship among variables

In [7]:

```
corr = df_housing_dataset.corr()
```

In [8]:

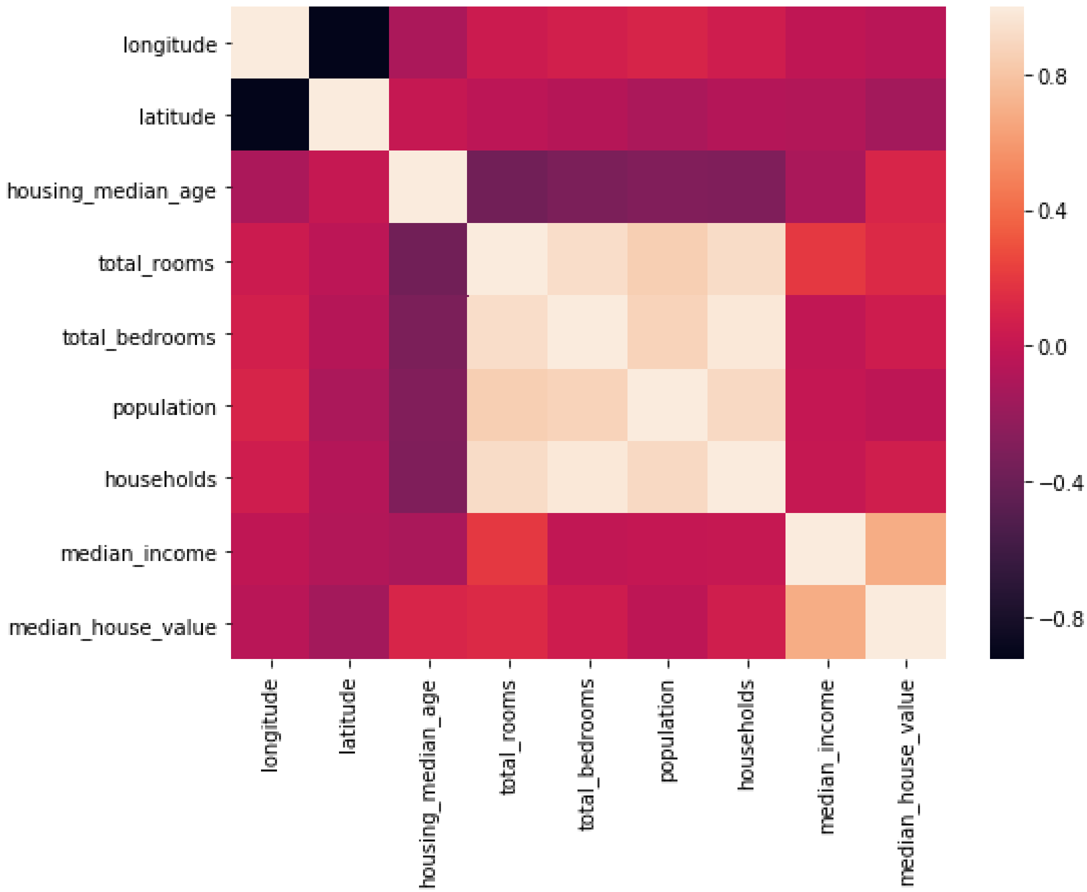
```
df_housing_dataset.corr()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	tot
longitude	1.000000	-0.924664	-0.108197	0.044568	0.0
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.9
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.0
population	0.099773	-0.108785	-0.296244	0.857126	0.8
households	0.055310	-0.071035	-0.302916	0.918484	0.9
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.0

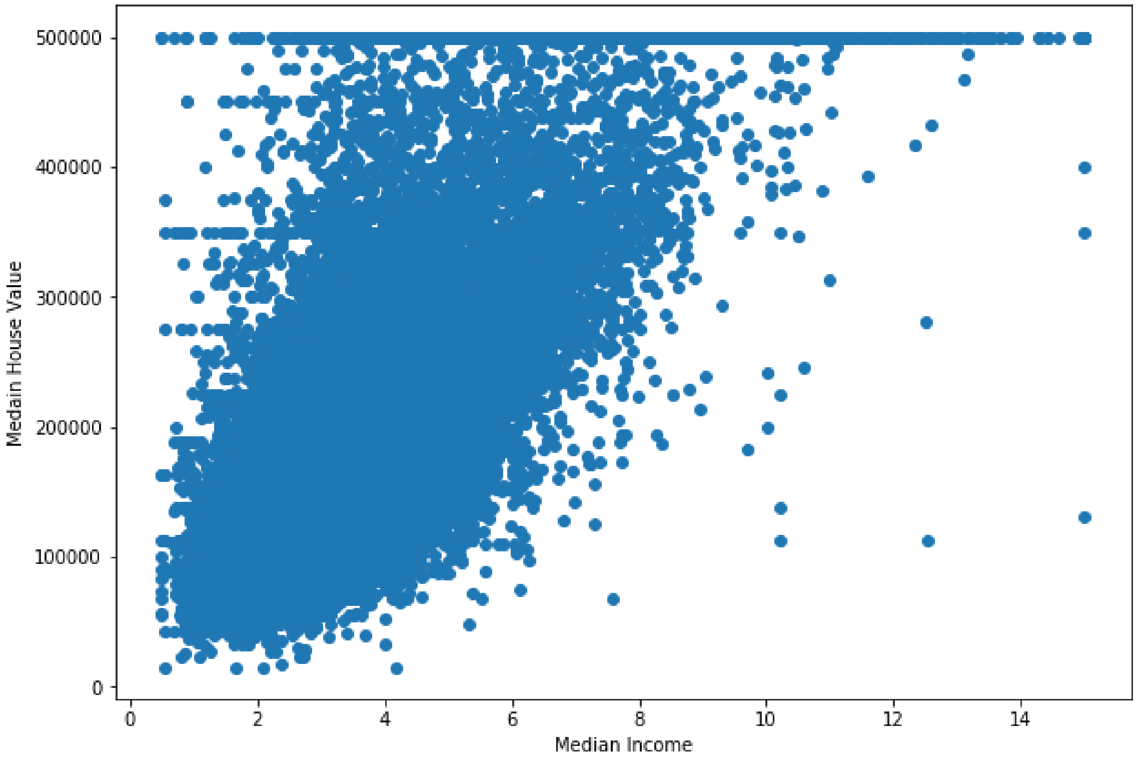
In [9]:

```
#Seaborn heatmap to view correlations between features in dataset
#Median income has a positive correlation against median house value
plt.figure(figsize=(8,6))
pltheatmap =sns.heatmap(corr)
```



In [10]:

```
#Scatter plot of median income with median house value
plt.figure(figsize=(10,7))
plt.scatter(df_housing_dataset['median_income'],df_housing_dataset['median_house_value'
])
plt.title='Scatter plot to correlate median income vs median house value'
plt.xlabel('Median Income')
plt.ylabel('Medain House Value')
plt.show()
```



In [11]:

```
df_housing_dataset.ocean_proximity.unique()
```

Out[11]:

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

In [12]:

```
df_housing_dataset.ocean_proximity.isnull().sum()
```

Out[12]:

```
0
```

In [13]:

```
#Slice dataset and store independent and dependent variables
X = df_housing_dataset.iloc[:,:-1].values
y = df_housing_dataset.iloc[:,9].values
print (X,y)
```

```
[[-122.23 37.88 41 ... 126 8.3252 'NEAR BAY']
 [-122.22 37.86 21 ... 1138 8.3014 'NEAR BAY']
 [-122.24 37.85 52 ... 177 7.2574 'NEAR BAY']
 ...
 [-121.22 39.43 17 ... 433 1.7 'INLAND']
 [-121.32 39.43 18 ... 349 1.8672 'INLAND']
 [-121.24 39.37 16 ... 530 2.3886 'INLAND']] [452600 358500 352100 ...  92
300  84700  89400]
```

In [14]:

```
#Label Encode ocean proximity column
from sklearn.preprocessing import LabelEncoder
ocean_proximity_labelencoder = LabelEncoder()
X[:,8] =  ocean_proximity_labelencoder.fit_transform(X[:,8])
```

In [15]:

```
X[:,8]
```

Out[15]:

```
array([3, 3, 3, ..., 1, 1, 1], dtype=object)
```

In [16]:

```
#Correlation between Ocean proximity and Median house value
corr1 = np.corrcoef(X[:,8].astype('float64'),y.astype('float64'))
print(corr1)
```

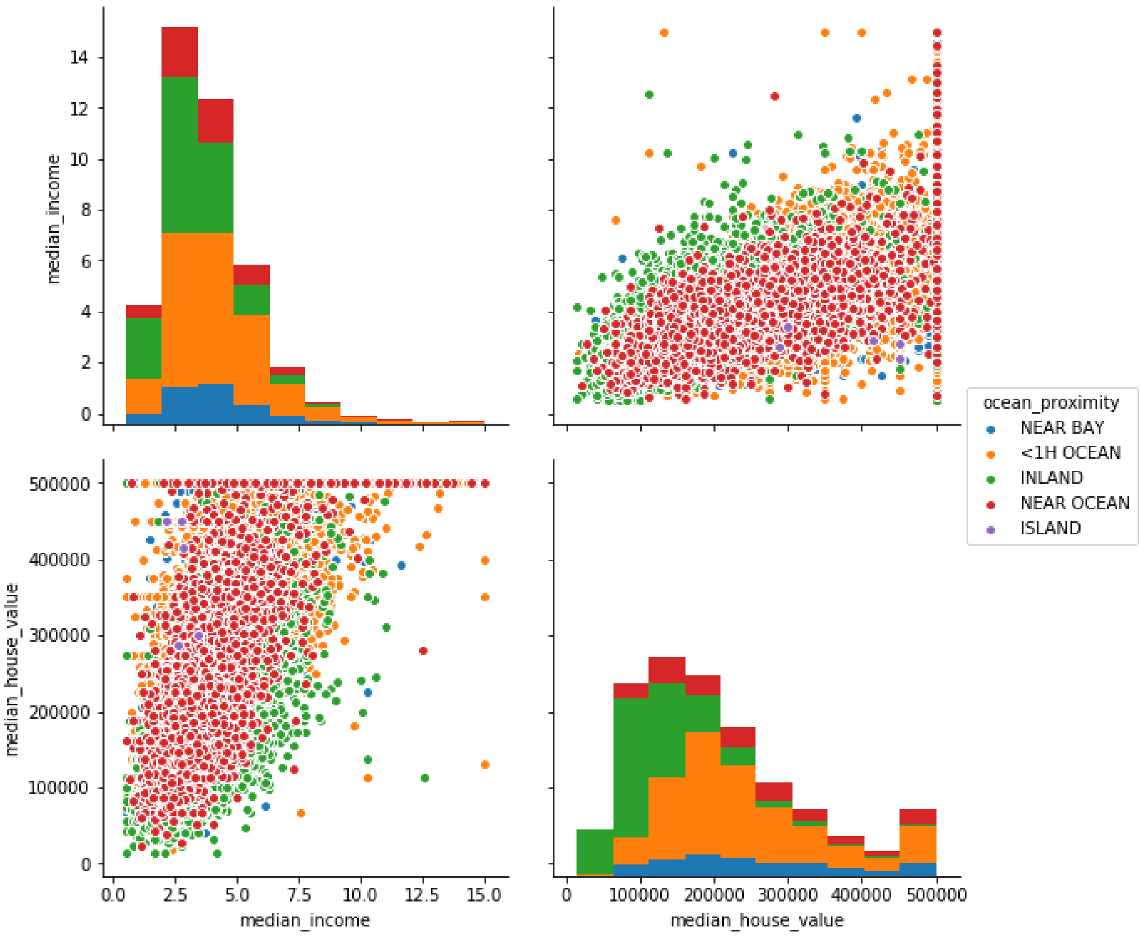
```
[[1.         0.08175023]
 [0.08175023 1.        ]]
```

In [17]:

```
#Seaborn pairplot of median income vs median house value with hue as OCean Proximity
#hue="ocean_proximity"
sns.pairplot(df_housing_dataset, size=4 ,
             vars=["median_income","median_house_value"], hue="ocean_proximity")
```

Out[17]:

```
<seaborn.axisgrid.PairGrid at 0x22c5ac79dd8>
```



Handle missing values

In [18]:

```
df_housing_dataset.isnull().sum()
```

Out[18]:

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
ocean_proximity         0
median_house_value      0
dtype: int64
```

In [19]:

```
df_X = pd.DataFrame(X)
```

In [20]:

```
df_X.isnull().sum()
```

Out[20]:

```
0      0
1      0
2      0
3      0
4    207
5      0
6      0
7      0
8      0
dtype: int64
```

In [24]:

```
#==============================================================================
# Handle the missing values, we can see that in dataset there are some missing
# values, we will use strategy to impute mean of column values in these places
#==============================================================================>>>
from sklearn.impute import SimpleImputer

#from sklearn.preprocessing import Imputer
# First create an Imputer
missingValueImputer = Imputer (missing_values = 'NaN', strategy = 'mean',
                               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])
```

```
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:5
8: 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 [25]:

```
#Notice missing values in total bedrooms column have been imputed with mean of total be
drooms
df_X = pd.DataFrame(X)
df_X.isnull().sum()
```

Out[25]:

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
dtype: int64
```

Principal Component Analysis

In [26]:

```
X.shape
```

Out[26]:

```
(20640, 9)
```

In [27]:

```
#Feature Scaling
#from sklearn.preprocessing import StandardScaler
#stdsclr = StandardScaler()
#X_std = stdsclr.fit_transform(X)
from sklearn.preprocessing import StandardScaler
stdsclr = StandardScaler()
(stdsclr.fit(X))
X_std = stdsclr.fit_transform(X)
```

```
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\validation.py:59
5: DataConversionWarning: Data with input dtype object was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\validation.py:59
5: DataConversionWarning: Data with input dtype object was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\validation.py:59
5: DataConversionWarning: Data with input dtype object was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [28]:

```
#PCA
from sklearn.decomposition.pca import PCA
PCA = PCA(n_components=6)
principal_components = PCA.fit_transform(X_std)
```

In [29]:

```
principal_components
```

Out[29]:

```
array([[-2.15719994,  1.70225453,  1.8547863 ,  1.70382331,  0.70467378,
         0.14044674],
       [ 2.87263151,  2.3047156 ,  1.9782144 ,  1.45640433,  0.22588098,
        -0.38227627],
       [-2.03519184,  1.79186986,  0.9709968 ,  1.89443788,  1.26251199,
         0.04699636],
       ...,
       [-0.45515873,  1.6384034 , -0.31193923, -1.51947927, -0.81951544,
        -0.24827694],
       [-0.86707319,  1.61941222, -0.20800024, -1.4886554 , -0.79020992,
        -0.28420985],
       [ 0.13316801,  1.6848017 , -0.05224996, -1.36842899, -0.68749535,
        -0.17045861]])
```

In [30]:

```
#Cal the cumulative proportion of var explained by each component
PCA.explained_variance_ratio_
```

Out[30]:

```
array([0.43398133, 0.22466942, 0.11957182, 0.10398544, 0.08531692,
       0.01648695])
```

In [31]:

```
df_X = pd.DataFrame(X)
print(df_X.columns)
```

```
RangeIndex(start=0, stop=9, step=1)
```

In [32]:

```
df_housing_dataset.head()
```

Out[32]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
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

In [33]:

```
# Dump components relations with features: This gives us the picture of how features ar
e related to components
print(pd.DataFrame(PCA.components_,columns=df_X.columns,index = ['PC-1','PC-2','PC-3',
'PC-4','PC-5','PC-6']))
```

```
             0         1         2         3         4         5         6  \
PC-1  0.081446 -0.077765 -0.219732  0.482987  0.488518  0.471762  0.490642   

PC-2 -0.670071  0.655264  0.033190  0.084062  0.072089  0.031852  0.074866   

PC-3 -0.089342  0.065996 -0.428611  0.085889 -0.120442 -0.114825 -0.113064   

PC-4  0.110276 -0.277884  0.419471  0.082480  0.029807  0.002983  0.041821   

PC-5 -0.140912  0.061118  0.762079  0.085413  0.046079  0.096782  0.078822   

PC-6 -0.113470 -0.073868 -0.042409 -0.313566 -0.391694  0.841691 -0.123976   


             7         8  
PC-1  0.045539 -0.041798  
PC-2 -0.032873  0.317125  
PC-3  0.856744 -0.148639  
PC-4  0.377072  0.763565  
PC-5  0.290296 -0.535139  
PC-6  0.052332  0.039623  
```

In [34]:

```
principal_components.shape
```

Out[34]:

```
(20640, 6)
```

Machine Learning Model Selection and Training

In [35]:

```
#Let's check our target label
y
```

Out[35]:

```
array([452600, 358500, 352100, ...,  92300,  84700,  89400], dtype=int64)
```

In [36]:

```
#Split Dataset for model training and testing [80/20 split]
#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(principal_components,y, test_size=0.1,
random_state=1)
```

In [37]:

```
X_train.shape
```

Out[37]:

```
(18576, 6)
```

Linear Regression ML Model

In [38]:

```
#Linear Regression Model
from sklearn.linear_model import LinearRegression
linReg = LinearRegression()
linReg.fit(X_train,y_train)
```

Out[38]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [39]:

```
linReg.predict(X_test)
```

Out[39]:

```
array([216366.32806176, 141489.28480139, 236970.65976977, ...,
       407406.64357423, 208935.87886274, 199176.10996327])
```

In [40]:

```
#Quick check accuracy of the model
score = linReg.score(X_train,y_train)
print(score)
```

```
0.5415486643334289
```

In [41]:

```
#Quick check accuracy of the model
score = linReg.score(X_test,y_test)
print(score)
```

```
0.5350362554208468
```

In [42]:

```
linreg_predictions = linReg.predict(X_test)
```

In [43]:

```
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,linreg_predictions))
#RMSE below
```

Out[43]:

```
79072.94760163683
```

Decision Tree ML model

In [44]:

```
#Train with DT model
from sklearn.tree import DecisionTreeRegressor
DTRegressor = DecisionTreeRegressor(max_depth=9, min_samples_split=5)
DTRegressor.fit(X_train,y_train)
```

Out[44]:

```
DecisionTreeRegressor(criterion='mse', max_depth=9, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=5, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

In [45]:

```
#Quick check accuracy of the model
score = DTRegressor.score(X_train,y_train)
print(score)
```

```
0.7538343085921241
```

In [46]:

```
#Quick check accuracy of the model
score = DTRegressor.score(X_test,y_test)
print(score)
```

0.6347818324506117

In [47]:

```
DTR_predictions = DTRegressor.predict(X_test)
```

In [48]:

```
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,DTR_predictions))
#RMSE below
```

Out[48]:

70080.07690286402

Random Forest ML Model - Model prediction accuracy is good compared to LR and DT models

In [49]:

```
#Declare hyper parameters to tune RF model
hyperparameters = { 'randomforestregressor__max_features' : ['auto', 'sqrt', 'log2'],
                  'randomforestregressor__max_depth': [None, 5, 3, 1],
                  'randomforestregressor__min_samples_split': [2, 5],
                  'randomforestregressor__min_samples_leaf': [10, 5]}
```

In [50]:

```
#Make a Random forest pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(RandomForestRegressor(n_estimators=50))
```

In [52]:

```
#Cross Validation to find best parameters
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(pipeline, hyperparameters, cv=10)
# Fit and tune model
clf.fit(X_train, y_train)
```

Out[52]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('randomforestregressor', RandomForestRegressor(bootstrap=Tru
e, criterion='mse', 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=50, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=Fals
e))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'randomforestregressor__max_features': ['auto', 'sqrt',
'log2'], 'randomforestregressor__max_depth': [None, 5, 3, 1], 'randomfores
tregressor__min_samples_split': [2, 5], 'randomforestregressor__min_sample
s_leaf': [10, 5]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [53]:

```
clf.best_params_
```

Out[53]:

```
{'randomforestregressor__max_depth': None,
 'randomforestregressor__max_features': 'auto',
 'randomforestregressor__min_samples_leaf': 5,
 'randomforestregressor__min_samples_split': 5}
```

In [54]:

```
clf.best_score_
```

Out[54]:

```
0.7427108313947447
```

In [55]:

```
clf.best_estimator_
```

Out[55]:

```
Pipeline(memory=None,
     steps=[('randomforestregressor', RandomForestRegressor(bootstrap=Tru
e, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=5, min_samples_split=5,
           min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=Fals
e))])
```

In [56]:

```
#Quick check accuracy of the model after CV
score = clf.score(X_train,y_train)
print(score)
```

```
0.8789239810104083
```

In [57]:

```
#Quick check accuracy of the model
score = clf.score(X_test,y_test)
print(score)
```

```
0.7389279737958411
```

In [58]:

```
X_test
```

Out[58]:

```
array([[ 0.165356  , -1.06409846, -0.64909599, -0.10464544,  0.86158045,
        -0.52274848],
       [-0.62981174,  0.38872249, -1.02298362, -0.37742015,  0.26363882,
        -0.16050481],
       [-0.94940543,  1.98117187, -0.50598625,  1.11304819,  0.5866817 ,
         0.43237097],
       ...,
       [ 9.94353045, -0.33380172, -0.03629273,  0.54202254, -0.30348569,
        -4.77201298],
       [-1.87167268, -1.53793658, -0.0138332 , -0.23525572,  0.41688408,
        -0.09706565],
       [-1.53844573,  1.2624476 ,  0.47595115, -0.61976029,  0.10913802,
        -0.04103344]])
```

In [59]:

```
RF_predictions = clf.predict(X_test)
```

In [60]:

```
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,RF_predictions))
#RMSE below for the 10% test set [unseen data]
```

Out[60]:

```
59251.391364186624
```

ML Model Training with only Median income feature to predict housing value

In [61]:

```
#Let's train the model only with median income and check how model behaves
X
```

Out[61]:

```
array([[-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]], dtype=object)
```

In [62]:

```
df_X_final = pd.DataFrame(X)
df_X_final.head()
```

Out[62]:

	0	1	2	3	4	5	6	7	8
0	-122.23	37.88	41	880	129	322	126	8.3252	3
1	-122.22	37.86	21	7099	1106	2401	1138	8.3014	3
2	-122.24	37.85	52	1467	190	496	177	7.2574	3
3	-122.25	37.85	52	1274	235	558	219	5.6431	3
4	-122.25	37.85	52	1627	280	565	259	3.8462	3

In [63]:

```
X = np.delete(X,[0,1,2,3,4,5,6,8],axis=1)
```

In [64]:

```
X
```

Out[64]:

```
array([[8.3252],
       [8.3014],
       [7.2574],
       ...,
       [1.7],
       [1.8672],
       [2.3886]], dtype=object)
```

In [77]:

```
#Split Dataset for model training and testing [80/20 split]
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=1/4,random_state=0)
```

In [78]:

```
X_train.shape
```

Out[78]:

```
(15480, 1)
```

In [79]:

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
stdsclr = StandardScaler()
X_train_std = stdsclr.fit_transform(X_train)
```

```
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\validation.py:59
5: DataConversionWarning: Data with input dtype object was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\validation.py:59
5: DataConversionWarning: Data with input dtype object was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [80]:

```
#Feature Scaling
X_test_std = stdsclr.fit_transform(X_test)
```

```
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\validation.py:59
5: DataConversionWarning: Data with input dtype object was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\SaiRam\Anaconda3\lib\site-packages\sklearn\utils\validation.py:59
5: DataConversionWarning: Data with input dtype object was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [81]:

```
#Linear Regression Model
from sklearn.linear_model import LinearRegression
linReg1 = LinearRegression()
linReg1.fit(X_train_std,y_train)
```

Out[81]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [82]:

```
#Quick check accuracy of the model
score = linReg1.score(X_train_std,y_train)
print(score)
```

```
0.48061930819884535
```

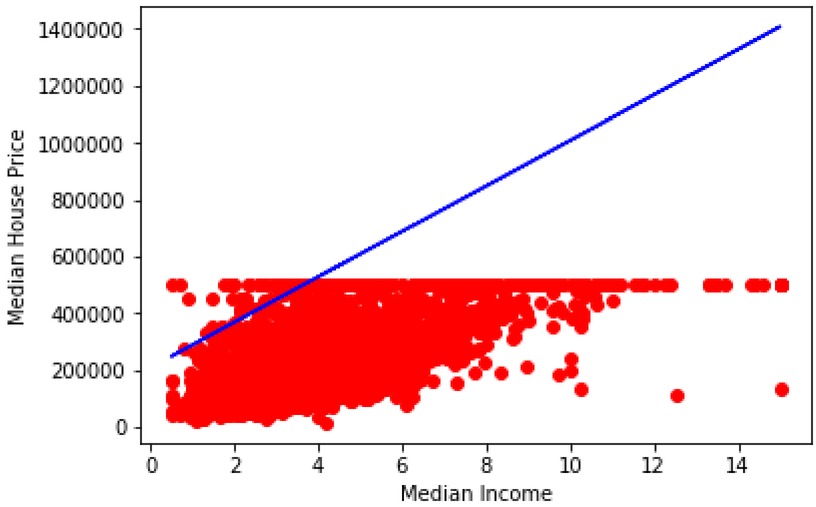
In [83]:

```
#Quick check accuracy of the model
score = linReg1.score(X_test_std,y_test)
print(score)
```

```
0.45147717106069024
```

In [84]:

```
#===============================================================================
# Visualize the linear regressor algo outcome
#===============================================================================
# Visualising the Regression results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, linReg1.predict(X_test), color = 'blue')
#plt.title('Median House Price Prediction')
plt.xlabel('Median Income')
plt.ylabel('Median House Price')
plt.show()
```



In [85]:

```
#==============================================================================
# Fitting the Polynomial Regression algorithm to the Training set
#==============================================================================

from sklearn.preprocessing import PolynomialFeatures
polyagent = PolynomialFeatures(degree=5)
X_Poly = polyagent.fit_transform(X_train)
```

In [86]:

```
linReg1.fit (X_Poly, y_train )
```

Out[86]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [87]:

```
X_Poly_test = polyagent.fit_transform(X_test)
```

In [88]:

```
score = linReg1.score(X_Poly,y_train)
print(score)
```

```
0.4927588433614648
```

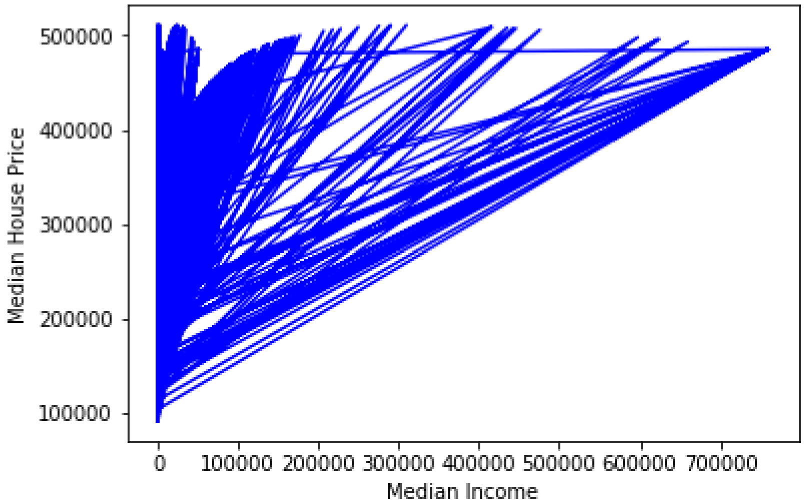
In [89]:

```
score = linReg1.score(X_Poly_test,y_test)
print(score)
```

```
0.46653755542283765
```

In [90]:

```
#==============================================================================
# Visualize the poly regressor algo outcome
#==============================================================================
# Visualising the Regression results
#plt.scatter(X_Poly_test, y_test, color = 'red')
plt.plot(X_Poly_test, linReg1.predict(X_Poly_test), color = 'blue')
#plt.title('Median House Price Prediction')
plt.xlabel('Median Income')
plt.ylabel('Median House Price')
plt.show()
```



In [91]:

```
#Let's fit DT Reg

from sklearn.tree import DecisionTreeRegressor
DTRegressor = DecisionTreeRegressor(max_depth=3)
DTRegressor.fit(X_train,y_train)
```

Out[91]:

```
DecisionTreeRegressor(criterion='mse', max_depth=3, 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=None, splitter='best')
```

In [92]:

```
#Quick check accuracy of the model on train
score = DTRegressor.score(X_train,y_train)
print(score)
```

0.48925102276025023

In [93]:

```
#==============================================================================
# Visualize the DT regressor algo outcome
#==============================================================================
# Visualising the Regression results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, DTRegressor.predict(X_test), color = 'blue')
#plt.title('Median House Price Prediction')
plt.xlabel('Median Income')
plt.ylabel('Median House Price')
plt.show()
```