California Housing Price Prediction ¶

DESCRIPTION

Background of Problem Statement :

The US Census Bureau has published California Census Data which has 10 types of metrics such as the population, median income, median housing price, and so on for each block group in California. The dataset also serves as an input for project scoping and tries to specify the functional and nonfunctional requirements for it.

Problem Objective :

The project aims at building a model of housing prices to predict median house values in California using the provided dataset. This model should learn from the data and be able to predict the median housing price in any district, given all the other metrics.

Districts or block groups are the smallest geographical units for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). There are 20,640 districts in the project dataset.

Domain:

Finance and Housing

Analysis Tasks to be performed:

1. Build a model of housing prices to predict median house values in California using the provided dataset.
2. Train the model to learn from the data to predict the median housing price in any district, given all the other metrics.
3. Predict housing prices based on median_income and plot the regression chart for it.

1.Load the data :

- Read the “housing.csv” file from the folder into the program.
- Print first few rows of this data.
- Extract input (X) and output (Y) data from the dataset.

2.Handle missing values :

- Fill the missing values with the mean of the respective column.

3.Encode categorical data :

- Convert categorical column in the dataset to numerical data.

4. Split the dataset :

- Split the data into 80% training dataset and 20% test dataset.

5. Standardize data :

- Standardize training and test datasets.

6. Perform Linear Regression :

- Perform Linear Regression on training data.
- Predict output for test dataset using the fitted model.
- Print root mean squared error (RMSE) from Linear Regression.

```
[HINT: Import mean_squared_error from sklearn.metrics]
```

7. Bonus exercise: Perform Linear Regression with one independent variable :

- Extract just the median_income column from the independent variables (from **X_train** and **X_test**).
- Perform Linear Regression to predict housing values based on **median_income**.
- Predict output for test dataset using the fitted model.
- Plot the fitted model for training data as well as for test data to check if the fitted model satisfies the test data.

Dataset Description :

Field : Description

- longitude (signed numeric - float) : Longitude value for the block in California, USA
- latitude (numeric - float) : Latitude value for the block in California, USA
- housing_median_age (numeric - int) : Median age of the house in the block
- total_rooms (numeric - int) : Count of the total number of rooms (excluding bedrooms) in all houses in the block
- total_bedrooms (numeric - float) : Count of the total number of bedrooms in all houses in the block

- population (numeric - int) : Count of the total number of population in the block
- households (numeric - int) : Count of the total number of households in the block
- median_income (numeric - float) : Median of the total household income of all the houses in the block
- ocean_proximity (numeric - categorical) : Type of the landscape of the block [Unique Values : 'NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND']
- median_house_value (numeric - int) : Median of the household prices of all the houses in the block

Dataset Size : 20640 rows x 10 columns

In [74]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Load the Data

In [55]:
```
#Importing California Housing Price Prediction Data
path=r'D:\Python Simplilearn Material\California Housing Price Prediction'
California_Housing=pd.read_excel(path+r'\California_Housing price.xlsx')
```

In [6]:
```
print("The shape of the California housing price data",California_Housing.shape)
print("The size of the California housing price data",California_Housing.size)
```

```
The shape of the California housing price data (20640, 10)
The size of the California housing price data 206400
```

In [7]:
```
#Displaying first 5 rows of the data
California_Housing.head()
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_v
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	45
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	35
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	35
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	34
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	34

In [8]:
```
California_Housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   ocean_proximity     20640 non-null  object 
 9   median_house_value  20640 non-null  int64  
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
```

From the above information Firstly we can see that all the columns have a count of 20640 values except **total_bedrooms** column it has only 20433 values which indicates that this column has missing values. Secondly we see the Dtype of the data present in the dataset, all the columns as numerical data (float64 and int64) expect the column **ocean_proximity** which is an object indicating it as a categorical data

In [9]:
```
#Distribution of values in the 'ocean_proximity' column
California_Housing['ocean_proximity'].value_counts()
```

Out[9]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

From the above results we can see 5 categories of values present in the ocean_proximity column and the frequency of their distribution

In [10]:
```
#Descriptive Statistics
California_Housing.describe()
```

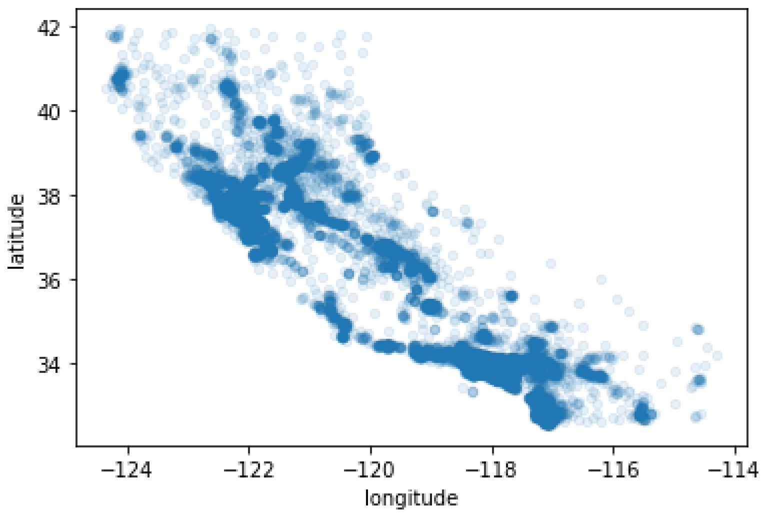
Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.0
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.8
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.6
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.0
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.0
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.0
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.0
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.0

In [80]:
```
# visualize the geographical data with latitude and longitude.
# To create a scatterplot of all the districts

California_Housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

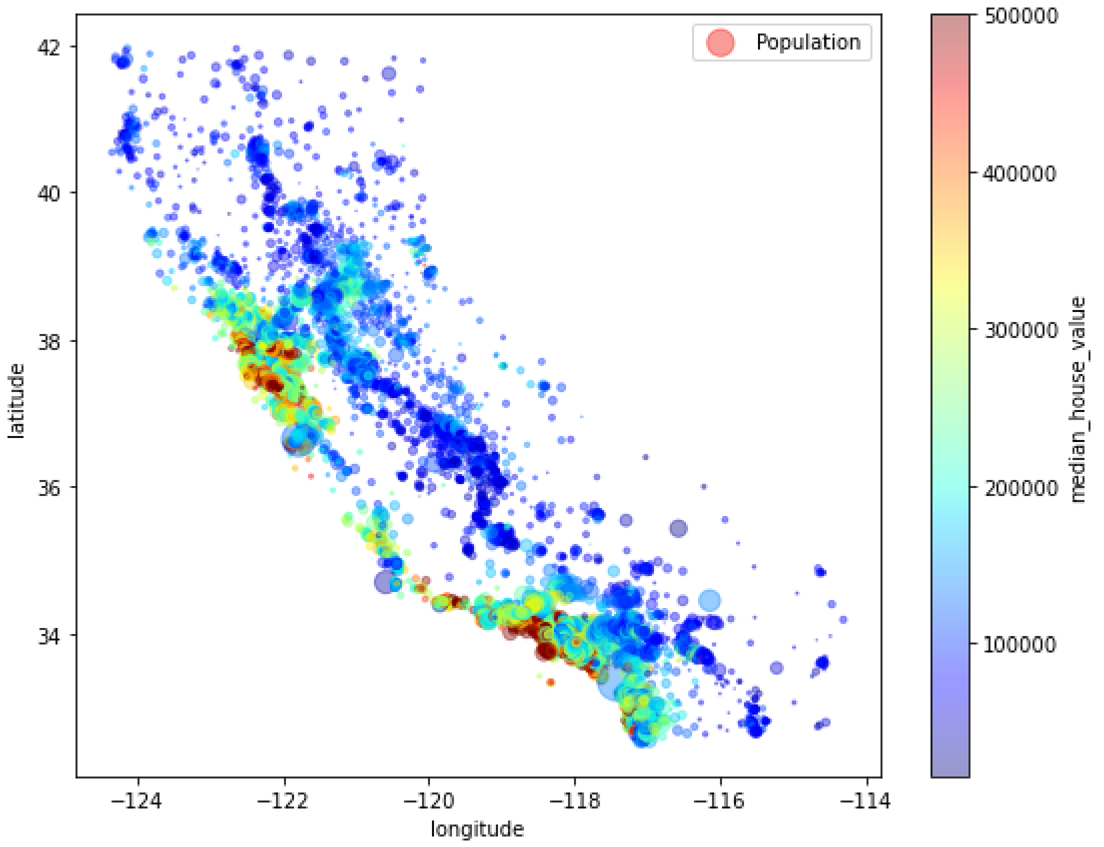
Out[80]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



In [81]:
```
# Now will look at the housing prices with a scatterplot.
# The color represents the price ,radius of each circle represents the districts population

California_Housing.plot(kind="scatter",x="longitude",y="latitude",alpha=0.4,
                        s=California_Housing['population']/100,label="Population",figsize=(9,7),
                        c="median_house_value",cmap=plt.get_cmap("jet"),colorbar=True,sharex=False)
plt.legend()
```

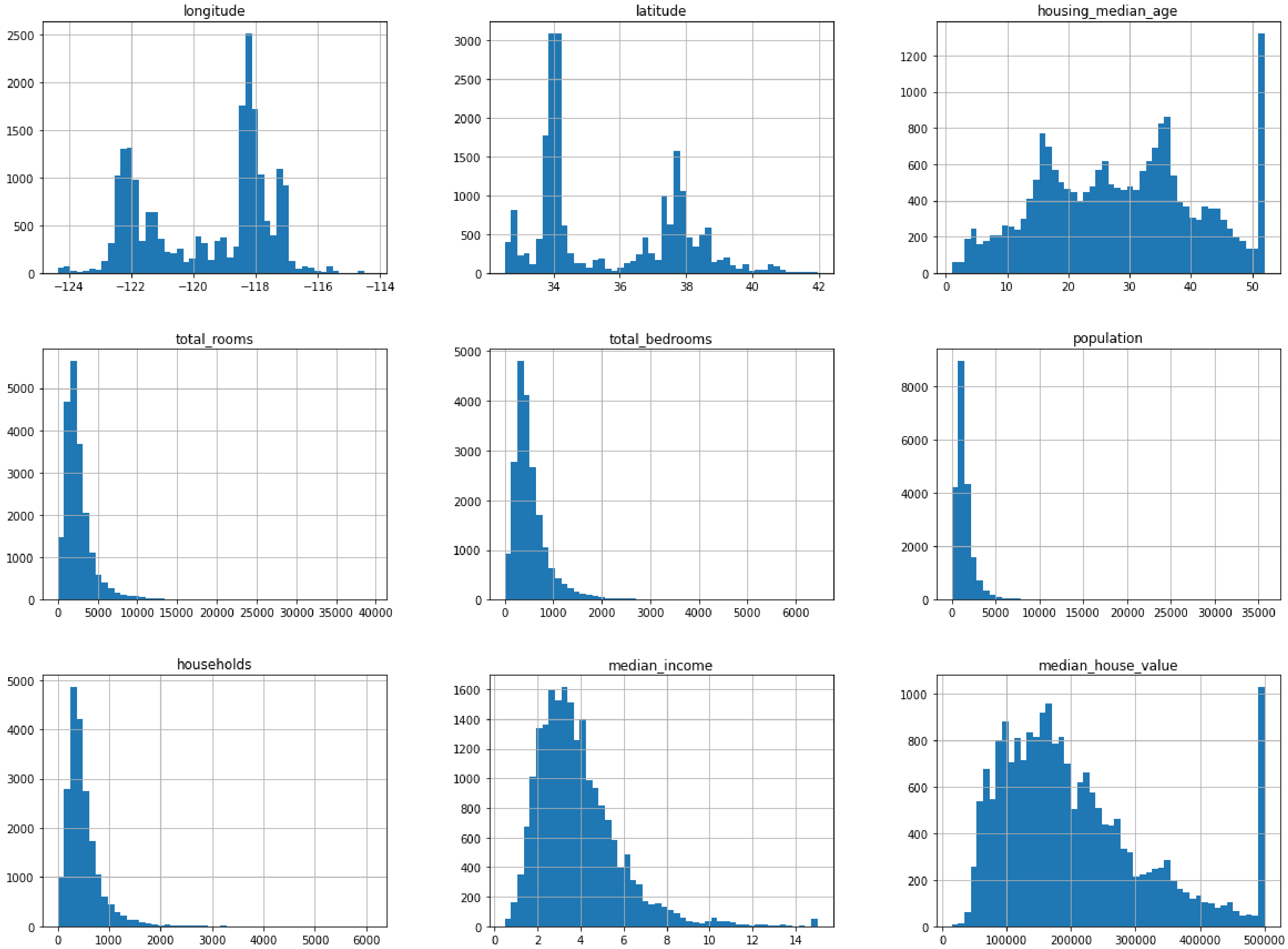
Out[81]: <matplotlib.legend.Legend at 0x78cd3db490>



From the above scatterplot we can see, that the housing prices are related to the location close to the ocean and to the population density.

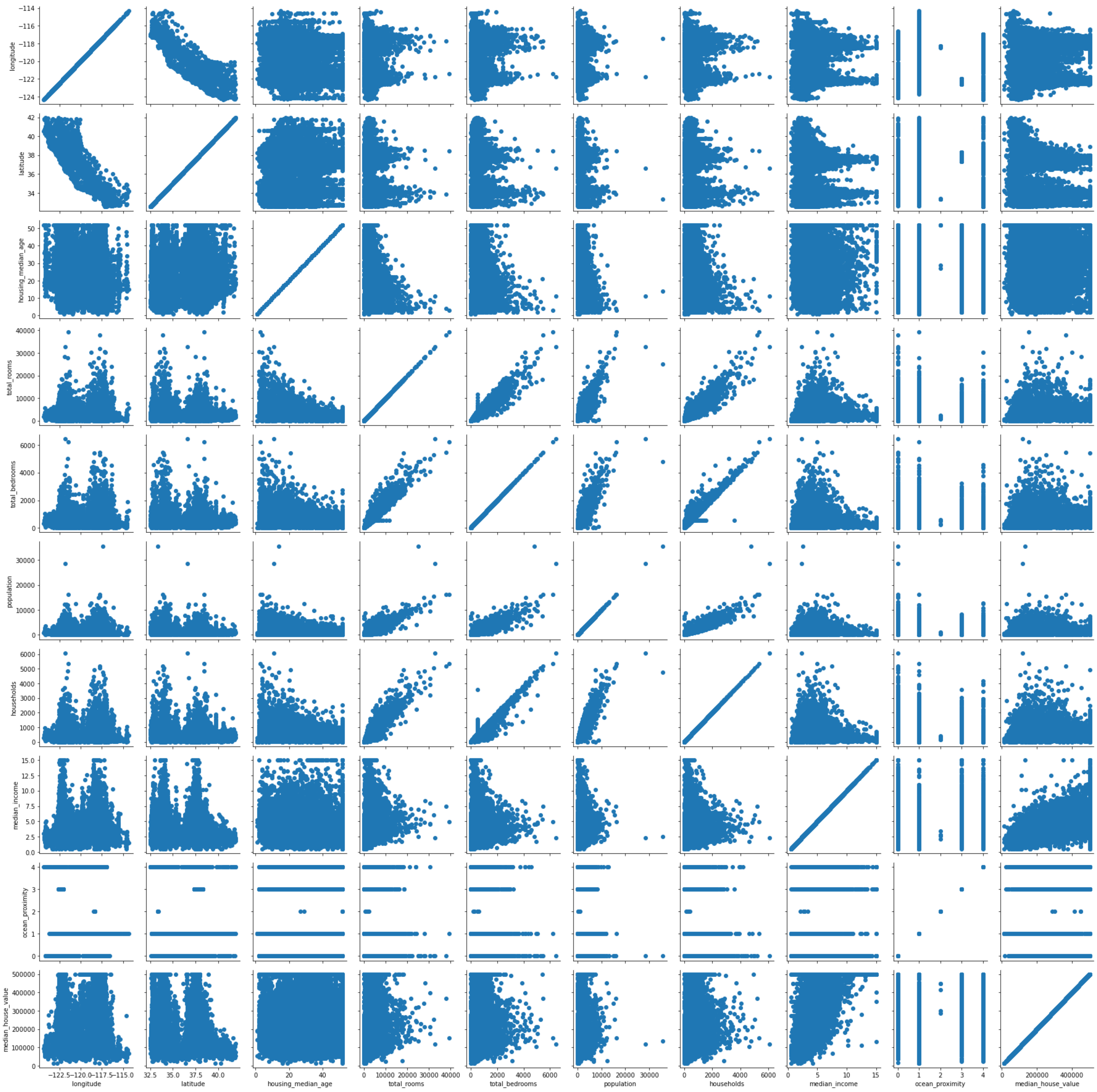
In [56]:
```
#Distribution of different features
California_Housing.hist(bins=50,figsize=(20,15))
```

Out[56]:
```
array([[<AxesSubplot:title={'center':'longitude'}>,
        <AxesSubplot:title={'center':'latitude'}>,
        <AxesSubplot:title={'center':'housing_median_age'}>],
       [<AxesSubplot:title={'center':'total_rooms'}>,
        <AxesSubplot:title={'center':'total_bedrooms'}>,
        <AxesSubplot:title={'center':'population'}>],
       [<AxesSubplot:title={'center':'households'}>,
        <AxesSubplot:title={'center':'median_income'}>,
        <AxesSubplot:title={'center':'median_house_value'}>]],
      dtype=object)
```



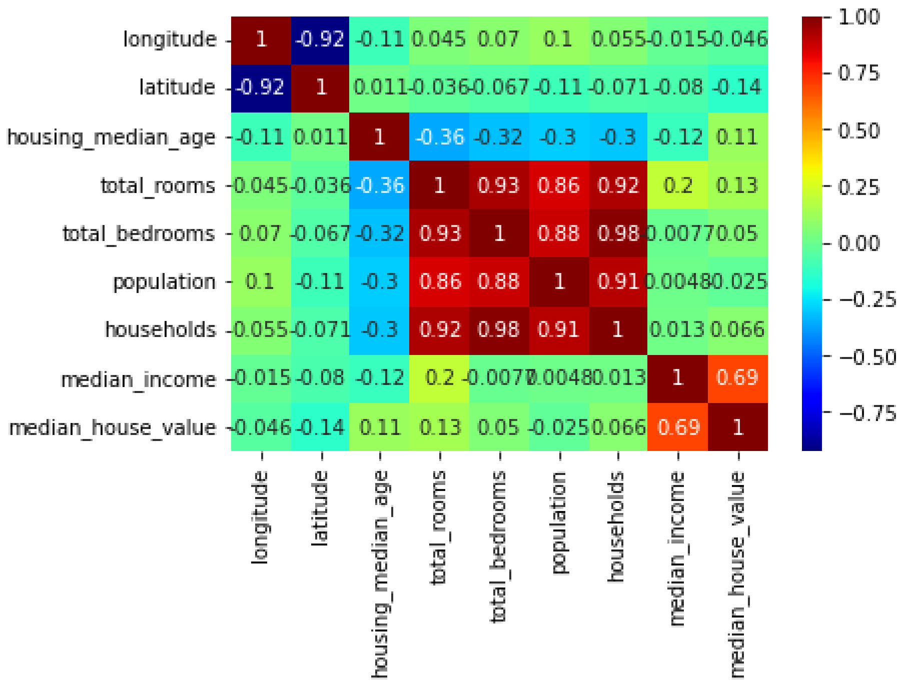
In [135]:
```
#Creating a Pair grid with scatter plot for visualizing linear relationship
grid1=sns.PairGrid(California_Housing)
grid1.map(plt.scatter)
```

Out[135]: <seaborn.axisgrid.PairGrid at 0x78e3f1c6a0>



In [45]:
```
#Plotting heatmap for understanding the correlation(to see how closely two variables are related) visually
sns.heatmap(California_Housing.corr(),annot=True,cmap='jet')
```

Out[45]:
```
<AxesSubplot:>
```



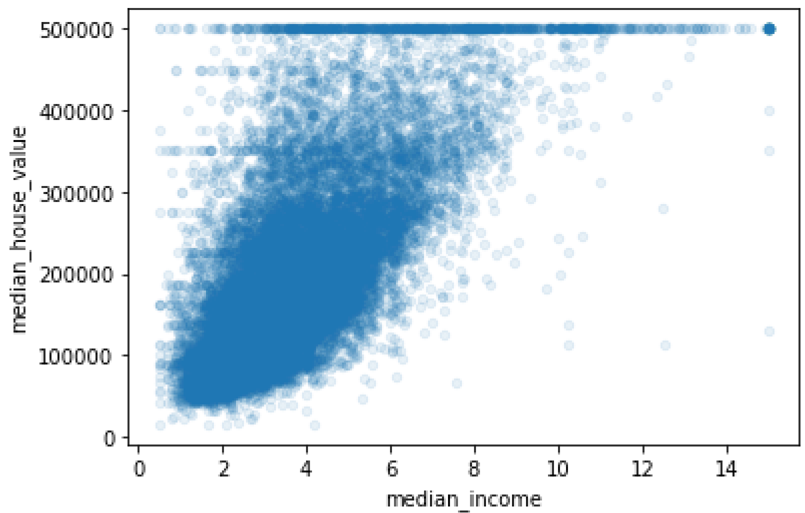
In [83]:
```
#correlation of 'median_house_value'
corr_matrix=California_Housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

Out[83]:
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
ocean_proximity       0.081750
households            0.065843
total_bedrooms        0.049454
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

we can see that the median_income is correlated the most with the median house value. Because of that, we will generate a more detailed scatterplot below:

In [82]:
```
California_Housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
```

Out[82]:
```
<AxesSubplot:xlabel='median_income', ylabel='median_house_value'>
```



The above scatterplot reveals, that the correlation is indeed very strong because we can clearly see an upward trend and the points are not to dispersed.

Handle Missing Values

In [59]:
```
# Checking for missing values
print("Number of missing values present in the California Housing Data:\n",California_Housing.isnull().sum())
```

```
Number of missing values present in the California Housing Data:
 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
```

We can see **total_bedrooms** column as **207** missing values

In [67]:
```
#Lets Fill the missing values with the mean of the total_bedrooms column.
California_Housing['total_bedrooms']=California_Housing['total_bedrooms'].fillna(California_Housing
                                                                                ['total_bedrooms'].mean())
#Checking for missing values after treating it by using mean
print("Number of missing values:\n",California_Housing.isnull().sum())
```

```
Number of missing values:
 longitude            0
latitude            0
housing_median_age    0
total_rooms         0
total_bedrooms      0
population          0
households          0
median_income       0
ocean_proximity     0
median_house_value    0
dtype: int64
```

From the above result we can see that there are no missing values as all the missing values in total_bedrooms columns are treated with mean of total_bedrooms column .

In [64]:
```
California_Housing.shape
```

Out[64]:
```
(20640, 10)
```

Encode Categorical Data

In [68]:
```
#Converting categorical variable "ocean_proximity" in the dataset to numerical data using "LabelEncoder"

#import label encoder
from sklearn import preprocessing
#label_encoder object knowns how to understand word labels
label_encoder=preprocessing.LabelEncoder()
#Encode labels in column 'ocean_proximity'.
California_Housing['ocean_proximity']=label_encoder.fit_transform(California_Housing['ocean_proximity'])

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

Out[68]:
```
array([3, 0, 1, 4, 2])
```

In [69]:
```
California_Housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   ocean_proximity     20640 non-null  int32  
 9   median_house_value  20640 non-null  int64  
dtypes: float64(4), int32(1), int64(5)
memory usage: 1.5 MB
```

From the above information we can see all the missing values from 'total_bedrooms' are handled and 'ocean_proximity' is converted from object to int32.Hence the two challenges in the dataset are solved!

Standardize Data

In [120]:
```
#Standardizing training and test datasets using "StandardScalar"

# Getting the column nams
Names = California_Housing.keys()

#importing StandardScaler
from sklearn.preprocessing import StandardScaler

# Creating the Scaler object scale
scaler=StandardScaler()

# Fitting the data on the scaler object
scaled_California_Housing = scaler.fit_transform(California_Housing)
scaled_California_Housing= pd.DataFrame(scaled_California_Housing, columns=Names)
scaled_California_Housing.head()
```

Out[120]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_
0	-1.327835	1.052548	0.982143	-0.804819	-0.975228	-0.974429	-0.977033	2.344766	1.291089	2.1
1	-1.322844	1.043185	-0.607019	2.045890	1.355088	0.861439	1.669961	2.332238	1.291089	1.3
2	-1.332827	1.038503	1.856182	-0.535746	-0.829732	-0.820777	-0.843637	1.782699	1.291089	1.2
3	-1.337818	1.038503	1.856182	-0.624215	-0.722399	-0.766028	-0.733781	0.932968	1.291089	1.1
4	-1.337818	1.038503	1.856182	-0.462404	-0.615066	-0.759847	-0.629157	-0.012881	1.291089	1.1

In [121]:
```
#Descriptive Statistics of Scaled data
Scaled_California_Housing.describe().round(2)
```

Out[121]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_hc
count	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	
mean	-0.00	0.00	0.00	0.00	0.00	-0.00	0.00	0.00	-0.00	
std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	
min	-2.39	-1.45	-2.20	-1.21	-1.28	-1.26	-1.30	-1.77	-0.82	
25%	-1.11	-0.80	-0.85	-0.54	-0.57	-0.56	-0.57	-0.69	-0.82	
50%	0.54	-0.64	0.03	-0.23	-0.24	-0.23	-0.24	-0.18	-0.12	
75%	0.78	0.97	0.66	0.23	0.25	0.26	0.28	0.46	-0.12	
max	2.63	2.96	1.86	16.82	14.09	30.25	14.60	5.86	2.00	

Extract input (X) and output (Y) data from the dataset.

In [122]:
```
#Creating Features Dataset-It as all columns except 'median_house_value'
X=scaled_California_Housing.drop(columns=['median_house_value'])
print("Shape of training data(X)=",X.shape)

#Displaying first 5 rows
X.head()
```

```
Shape of training data(X)= (20640, 9)
```

Out[122]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
0	-1.327835	1.052548	0.982143	-0.804819	-0.975228	-0.974429	-0.977033	2.344766	1.291089
1	-1.322844	1.043185	-0.607019	2.045890	1.355088	0.861439	1.669961	2.332238	1.291089
2	-1.332827	1.038503	1.856182	-0.535746	-0.829732	-0.820777	-0.843637	1.782699	1.291089
3	-1.337818	1.038503	1.856182	-0.624215	-0.722399	-0.766028	-0.733781	0.932968	1.291089
4	-1.337818	1.038503	1.856182	-0.462404	-0.615066	-0.759847	-0.629157	-0.012881	1.291089

In [123]:
```
#Creating Label Dataset
Y=scaled_California_Housing['median_house_value']
print("Shape of testing data(Y)=",Y.shape)

#Displaying first 5 rows
Y.head()
```

```
Shape of testing data(Y)= (20640,)
```

Out[123]:
```
0    2.129631
1    1.314156
2    1.258693
3    1.165100
4    1.172900
Name: median_house_value, dtype: float64
```

Split the dataset.

In [124]:
```
#Split the data into 80% training dataset and 20% test dataset.

# importing train_test_split
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1)

print ("Shape of x_train=",x_train.shape,"\nShape of y_train=",y_train.shape)
print ("Shape of x_test=",x_test.shape,"\nShape of y_test=",y_test.shape)
```

```
Shape of x_train= (16512, 9)
Shape of y_train= (16512,)
Shape of x_test= (4128, 9)
Shape of y_test= (4128,)
```

Apply various algorithms to develop a model

- Linear Regression
- Decision Tree Regression
- Random Forest Regression
- Lasso
- Ridge
- Elastic Net

In [142]:
```
from sklearn.linear_model import LinearRegression,Ridge,Lasso,ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import statsmodels.formula.api as smf
from sklearn.metrics import mean_squared_error,r2_score
from math import sqrt
```

1.Perform Linear Regression :

In [159]:
```
#Perform Linear Regression on training data.
LR=LinearRegression()
LR.fit(x_train,y_train)
#Predict output for test dataset using the fitted model
y_predict = LR.predict(x_test)
#Print root mean squared error (RMSE) from Linear Regression
print("RMSE from Linear Regression=",sqrt(mean_squared_error(y_test,y_predict)))
print(("R2 score=",r2_score(y_test,y_predict)))
```

```
RMSE from Linear Regression= 0.6056598120301221
('R2 score=', 0.6276223517950296)
```

2.Perform Decision Tree Regression :

In [160]:
```
#Perform Decision Tree Regression on training data.
DTR=DecisionTreeRegressor()
DTR.fit(x_train,y_train)
#Predict output for test dataset using the fitted model
y_predict = DTR.predict(x_test)
#Print root mean squared error (RMSE) from Decision Tree Regression
print("RMSE from Decision Tree Regression=",sqrt(mean_squared_error(y_test,y_predict)))
print(("R2 score=",r2_score(y_test,y_predict)))
```

```
RMSE from Decision Tree Regression= 0.5925564556422914
('R2 score=', 0.6435607199654507)
```

3.Perform Random Forest Regression :

In [161]:
```
#Perform Random Forest Regression on training data.
RFR=RandomForestRegressor(n_estimators=100,random_state=5)
RFR.fit(x_train,y_train)
#Predict output for test dataset using the fitted model
y_predict = RFR.predict(x_test)
#Print root mean squared error (RMSE) from Random Forest Regression
print("RMSE from Random Forest Regression=",sqrt(mean_squared_error(y_test,y_predict)))
print(("R2 score=",r2_score(y_test,y_predict)))
```

```
RMSE from Random Forest Regression= 0.4269250784402476
('R2 score=', 0.8149754214338779)
```

4.Perform Lasso Regression

In [154]:
```
#Lasso Regression(determine which variables should be retained in the model)
# Performing Lasso Regression on training data.
LaR=Lasso(alpha=0.001,normalize=True)
LaR.fit(x_train,y_train)
#Print root mean squared error (RMSE) from Lasso Regression
print("RMSE from Lasso Regression =",sqrt(mean_squared_error(y_test,LaR.predict(x_test))))
print('R2 Value/Coefficient of determination:{}'.format(LaR.score(x_test,y_test)))
```

```
RMSE from Lasso Regression = 0.7193140967070711
R2 Value/Coefficient of determination:0.4747534206169959
```

5.Perform Ridge Regression

In [155]:
```
#Ridge Regression(addresses multicollinearity issues)
# Performing Ridge Regression on training data.
RR=Ridge(alpha=0.001,normalize=True)
RR.fit(x_train,y_train)
#Print root mean squared error (RMSE) from Ridge Regression
print("RMSE from Ridge Regression =",sqrt(mean_squared_error(y_test,RR.predict(x_test))))
print('R2 Value/Coefficient of determination:{}'.format(RR.score(x_test,y_test)))
```

```
RMSE from Ridge Regression = 0.6056048844852343
R2 Value/Coefficient of determination:0.6276898909055972
```

6.Performing ElasticNet Regression

In [156]:
```
# Performing ElasticNet Regression  on training data.
ENR=ElasticNet(alpha=0.001,normalize=True)
ENR.fit(x_train,y_train)
#Print root mean squared error (RMSE) from ElasticNet Regression
print('RMSE from ElasticNet Regression',sqrt(mean_squared_error(y_test,ENR.predict(x_test))))
print('R2 Value/Coefficient of determination:{}'.format(ENR.score(x_test,y_test)))
```

```
RMSE from ElasticNet Regression 0.944358169398106
R2 Value/Coefficient of determination:0.09468529806704551
```

7. Bonus exercise: Perform Linear Regression with one independent variable

In [166]:
```
# 1.Extract just the median_income column from the independent variables (from X_train and X_test).
x_train_Income=x_train[['median_income']]
x_test_Income=x_test[['median_income']]
print("Shape of x_train:",x_train_Income.shape)
print("Shape of y_train:",y_train.shape)
```

```
Shape of x_train: (16512, 1)
Shape of y_train: (16512,)
```

In [189]:
```
# 2.Perform Linear Regression to predict housing values based on median_income
LR=LinearRegression()
LR.fit(x_train_Income,y_train)

# 3. Predict output for test dataset using the fitted model.
y_predict = LR.predict(x_test_Income)

#print intercept and co-efficient of the linear equation
print(LR.intercept_, LR.coef_)
print("RMSE from Linear Regression =",sqrt(mean_squared_error(y_test,y_predict)))
print("R2 score=",r2_score(y_test,y_predict))
```

```
0.005623019866893162 [0.69238221]
RMSE from Linear Regression = 0.7212595914243148
R2 score= 0.47190835934467734
```

In [ ]:
```
#4. Plot the fitted model for training data as well as for test data to check if the fitted model satisfies the test
```

In [195]:

```
# visualize the Training Data
plt.scatter(x_train_Income, y_train, color = 'limegreen')
plt.plot (x_train_Income, LR.predict(x_train_Income), color = 'red')
plt.title ('compare Training result - median_income / median_house_value')
plt.xlabel('median_income')
plt.ylabel('median_house_value')
plt.show()
```

In [196]:

```
# visualize the Testing Data
plt.scatter(x_test_Income, y_test, color = 'b')
plt.plot (x_test_Income, LR.predict(x_test_Income), color = 'b')
plt.title ('compare Testing result - median_income / median_house_value')
plt.xlabel('median_income')
plt.ylabel('median_house_value')
plt.show()
```

In [ ]:

```

```

In [ ]:

```

```