In [105…

```
#Importing Libraries needed in the project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

In [106…

```
#Reading the dataset from the excel and creating a dataframe
# Printing First 5 rows
df_housing = pd.read_excel('1553768847_housing.xlsx')
df_housing.head()
```

Out[106…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41	880	129.0	322	126	
1	-122.22	37.86	21	7099	1106.0	2401	1138	
2	-122.24	37.85	52	1467	190.0	496	177	
3	-122.25	37.85	52	1274	235.0	558	219	
4	-122.25	37.85	52	1627	280.0	565	259	

In [107…

```
# Checking the mean/ counts and std in the dataframe
df_housing.describe()
```

Out[107…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	2(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	(

In [108…

```
# Checking the data for null columns and datatypes to see if any non numrical column ex
df_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
```

In [109…

```
#count of Null values
df_housing.isnull().sum()
```

Out[109…

```
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 [110…

```
#Fill the missing values with the mean of the respective column.
df_housing.fillna(df_housing['total_bedrooms'].mean(), inplace=True)
```

In [111…

```
df_housing.isnull().sum()
```

Out[111…

```
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
```

In [112…

```
#Convert categorical column in the dataset to numerical data using one hot encoder
df_housing_final =pd.get_dummies(data= df_housing, columns=['ocean_proximity'])
df_housing_final.shape
```

Out[112…

```
(20640, 14)
```

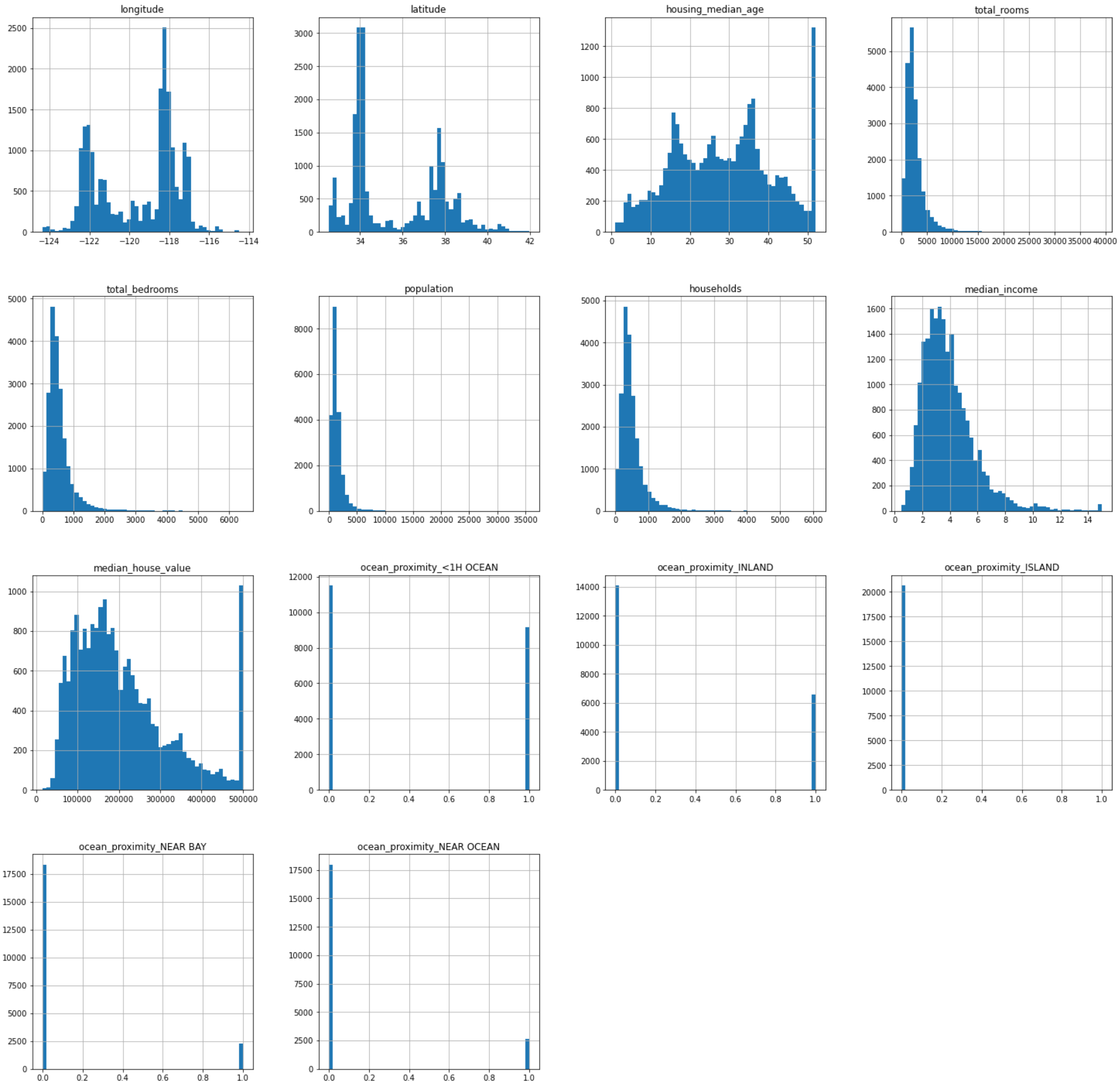
In [113…

```
#Extract input (X) and output (Y) data from the dataset.
x_input = df_housing_final.drop(columns =['median_house_value'])
y_output = df_housing_final['median_house_value']
print(x_input.shape,y_output.shape)
```

```
(20640, 13) (20640,)
```

In [114…

```
df_housing_final.hist(figsize=(25,25),bins=50);
```



In [115…

```
#Split the data into 80% training dataset and 20% test dataset.
x_train,x_test,y_train,y_test=train_test_split(x_input,y_output,test_size=0.2,random_st
print( x_train.shape,x_test.shape, y_train.shape, y_test.shape)
```

```
(16512, 13) (4128, 13) (16512,) (4128,)
```

In [116…

```
#Standardize training and test datasets.
scale= StandardScaler()
scaled_train_data = scale.fit_transform(x_train)
scaled_test_data = scale.fit_transform(x_test)
print(scaled_train_data , scaled_test_data)
```

```
[[-1.42250942  0.97229046  1.85890297 ... -0.01740407  2.82640555
  -0.38546202]
 [-1.38265919  1.08459626  1.06434823 ... -0.01740407  2.82640555
  -0.38546202]
 [-0.8297373   1.06119922 -1.0014941  ... -0.01740407 -0.35380627
  -0.38546202]
 ...
 [ 0.65468363 -0.79652586  1.06434823 ... -0.01740407 -0.35380627
  -0.38546202]
 [ 1.20262424 -0.89011402 -1.47822694 ... -0.01740407 -0.35380627
  -0.38546202]
 [-1.30794002  1.00972573  0.50815991 ... -0.01740407  2.82640555
  -0.38546202]] [[ 0.59953305 -0.73685251  0.81226638 ...  0.          -0.35109159
  -0.38047173]
 [-0.11505424  0.53929953  0.65331708 ...  0.          -0.35109159
  -0.38047173]
 [-1.44358273  0.9850144   1.36858896 ...  0.           2.84825961
  -0.38047173]
 ...
 [-1.4184212   0.92402184 -0.22090411 ...  0.          -0.35109159
   2.62831619]
 [ 0.73037298 -0.72277731  1.05069034 ...  0.          -0.35109159
  -0.38047173]
 [ 1.09269893 -0.76969466  1.84543688 ...  0.          -0.35109159
  -0.38047173]]
```

In [117…

```
#Perform Linear Regression on training data.
regressor = LinearRegression()
regressor.fit(x_train, y_train)
print(regressor.intercept_, regressor.coef_)
```

```
-2224231.212392007 [-2.65375452e+04 -2.51693181e+04  1.06947068e+03 -5.49854147e+00
  7.81334978e+01 -3.84586986e+01  7.13344968e+01  3.93198087e+04
 -2.41189059e+04 -6.34495807e+04  1.31959049e+05 -2.62977843e+04
 -1.80927777e+04]
```

In [118…

```
#Predict output for test dataset using the fitted model.
y_predict = regressor.predict(x_test)
```

In [119…

```
#Print root mean squared error (RMSE) from Linear Regression.
np.sqrt(metrics.mean_squared_error(y_test, y_predict))
```

Out[119…

```
68949.62451074323
```

Bonus exercise: Perform Linear Regression with one independent variable

In [120…

```
#Extract just the median_income column from the independent variables (from X_train and

x_train_mi = x_train['median_income']
x_train_mi = x_train_mi.values.reshape(-1,1)
x_test_mi = x_test['median_income']
x_test_mi = x_test_mi.values.reshape(-1,1)
y_train = y_train.values.reshape(-1,1)
print(x_train_mi.shape, x_test_mi.shape)
```

```
(16512, 1) (4128, 1)
```

In [121…

```
#Perform Linear Regression to predict housing values based on median_income.
regressor.fit(x_train_mi, y_train)
print(regressor.intercept_, regressor.coef_)
```

```
[44721.83362107] [[42055.4573838]]
```

In [122…

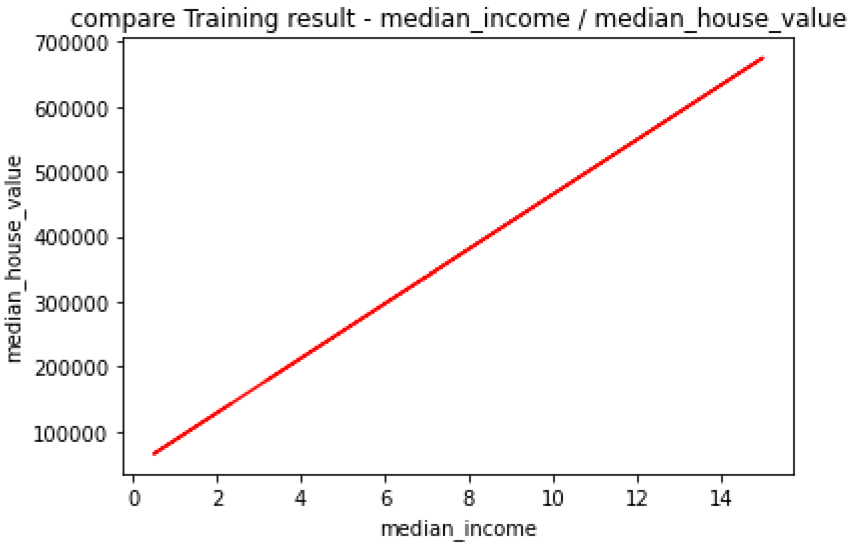
```
#Predict output for test dataset using the fitted model.

y_predict_mi = regressor.predict(x_test_mi)
y_predict_mi = y_predict_mi.reshape(-1,1)
```

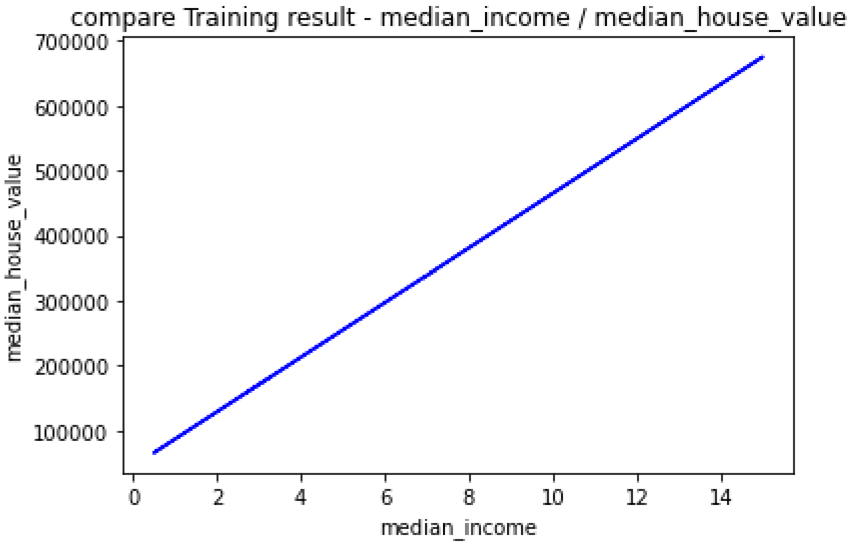
In [123…

```
np.sqrt(metrics.mean_squared_error(y_test, y_predict_mi))
```

Out[123… 83228.17849797675

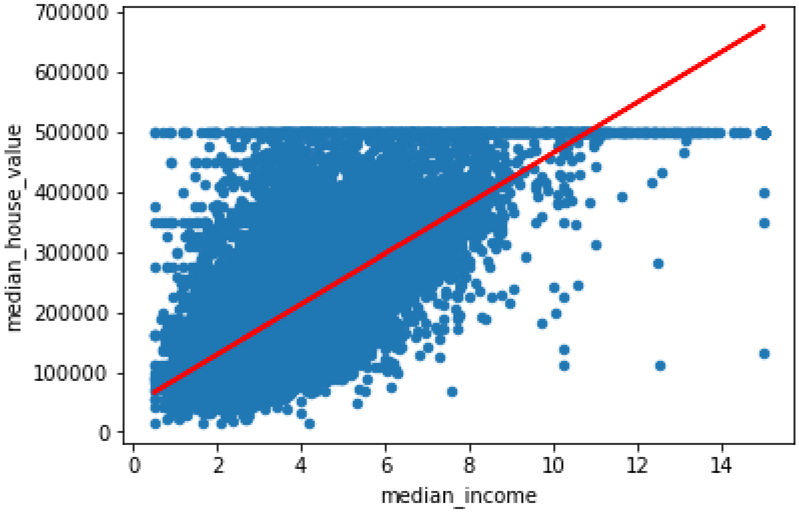
In [124…
```
#Plot the fitted model for training data as well as for test data to check if the fitte
plt.plot (x_train_mi, regressor.predict(x_train_mi), color = 'red')
plt.title ('compare Training result - median_income / median_house_value')
plt.xlabel('median_income')
plt.ylabel('median_house_value')
plt.show()

plt.plot (x_test_mi, regressor.predict(x_test_mi), color = 'blue')
plt.title ('compare Training result - median_income / median_house_value')
plt.xlabel('median_income')
plt.ylabel('median_house_value')
plt.show()
```





In [125…
```
df_housing_final.plot(kind='scatter',x='median_income',y='median_house_value')
plt.plot(x_test_mi,y_predict_mi,c='red',linewidth=2)
```

Out[125… [<matplotlib.lines.Line2D at 0x24769cfba30>]


700000
600000
500000
400000
300000
200000
100000
0
median_house_value
0
2
4
6
8
10
12
14
median_income

```
In [ ]:
```