In [105…
```python
#Importing all Libraries
import pandas as pd    ##..data preprocessing, data frame (read,drop,..& so on)
import numpy as np     ##..Linear algebra
import matplotlib.pyplot as plt   ##..Data Visualization
%matplotlib inline
#statement
import seaborn as sns   ##..Stastical data visualization

import warnings
warnings.filterwarnings('ignore')   ##..ignore unwanted warnings
```

1.Load the data:-

Read the “housing.csv” file from the folder into the program.

Print first few rows of this data

In [106…
```python
##Import dataset
house_data=pd.read_excel('1553768847_housing.xlsx')
```

EDA

In [107…
```python
## check shape of data
house_data.shape
```

Out[107]:
```
(20640, 10)
```

In [108…
```python
##check dataset
house_data.head()
```

Out[108]:

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

In [109…
```python
##check data tpyes of each col
house_data.dtypes
```

Out[109]:
```
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
```

one col(ocean_proximity)has a catgorical dtype and all remaining col have numerical dtype

In [110…
```python
## check columns
house_data.columns
```

Out[110]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity', 'median_house_value'],
      dtype='object')
```

2. Handle missing values :-

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

In [111…
```python
## check null values
house_data.isnull().sum().any()
```

Out[111]:
```
True
```

In [112…
```python
house_data.isnull().sum()
```

Out[112]:
```
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 there is null values in total bedrooms col.

so we have to raplace the null values with mean

In [113…
```python
house_data.total_bedrooms=house_data.total_bedrooms.fillna(house_data.total_bedrooms.mean())
house_data.isnull().sum()
#check again null values records
```

Out[113]:
```
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
```

now there is no null record found in dataset

3. Encode categorical data :

Convert categorical column in the dataset to numerical data.

In [114…
```python
## import label encoder
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

In [115…
```python
le=LabelEncoder()
house_data['ocean_proximity']=le.fit_transform(house_data['ocean_proximity'])
```

In [116…
```python
house_data
```

Out[116]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	medi
0	-122.23	37.88	41	880	129.0	322	126	8.3252	3	
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	3	
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	3	
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	3	
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	3	
...	...	...	...	...	...	...	...	...	...	
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	1	
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	1	
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	1	
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	1	
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	1	

20640 rows × 10 columns

In [117…
```python
#Extracting Independent and dependent Variable
x= house_data.drop('median_house_value',axis=1)
y= house_data['median_house_value']
```

In [118…
```python
x
```

Out[118]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
0	-122.23	37.88	41	880	129.0	322	126	8.3252	3
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	3
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	3
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	3
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	3
...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	1
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	1
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	1
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	1
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	1

20640 rows × 9 columns

In [119…
```python
y
```

Out[119]:
```
0        452600
1        358500
2        352100
3        341300
4        342200
          ...
20635     78100
20636     77100
20637     92300
20638     84700
20639     89400
Name: median_house_value, Length: 20640, dtype: int64
```

In [120…
```python
print(house_data.shape)
print(x.shape)
print(y.shape)
```

```
(20640, 10)
(20640, 9)
(20640,)
```

4. Split the dataset :

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

In [121…
```python
# Splitting the dataset into training and test set.
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=0)
```

In [122…
```python
print (x_train.shape, y_train.shape)
print (x_test.shape, y_test.shape)
```

```
(16512, 9) (16512,)
(4128, 9) (4128,)
```

5. Standardize data

Standardize training and test datasets.

In [123…
```python
#feature Scaling
from sklearn.preprocessing import StandardScaler
#st_x= StandardScaler()
#x_train= st_x.fit_transform(x_train)
#x_test= st_x.transform(x_test)
```

In [124…
```python
# Get column names first
names = house_data.columns
# Create the Scaler object
scaler = StandardScaler()
# Fit your data on the scaler object
scaled_df = scaler.fit_transform(house_data)
scaled_df = pd.DataFrame(scaled_df, columns=names)
scaled_df.head()
```

Out[124]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_h
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 [125…
```python
x_train
```

Out[125]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
12069	-117.55	33.83	6	502	76.000000	228	65	4.2386	1
15925	-122.44	37.73	52	2381	492.000000	1485	447	4.3898	3
11162	-118.00	33.83	26	1718	385.000000	1022	368	3.9333	0
4904	-118.26	34.01	38	697	208.000000	749	206	1.4653	0
4683	-118.36	34.08	52	2373	601.000000	1135	576	3.1765	0
...	...	...	...	...	...	...	...	...	...
13123	-121.26	38.27	20	1314	229.000000	712	219	4.4125	1
19648	-120.89	37.48	27	1118	195.000000	647	209	2.9135	1
9845	-121.90	36.58	31	1431	537.870553	704	393	3.1977	4
10799	-117.93	33.62	34	2125	498.000000	1052	468	5.6315	0
2732	-115.56	32.80	15	1171	328.000000	1024	298	1.3882	1

16512 rows × 9 columns

In [126…
```python
x_test
```

Out[126]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
14740	-117.05	32.58	22	2101	399.0	1551	371	4.1518	4
10101	-117.97	33.92	32	2620	398.0	1296	429	5.7796	0
20566	-121.84	38.65	29	3167	548.0	1554	534	4.3487	1
2670	-115.60	33.20	37	709	187.0	390	142	2.4511	1
15709	-122.43	37.79	25	1637	394.0	649	379	5.0049	3
...	...	...	...	...	...	...	...	...	...
6655	-118.13	34.16	33	2682	716.0	2050	692	2.4817	0
3505	-118.45	34.25	36	1453	270.0	808	275	4.3839	0
1919	-120.92	38.86	11	1720	345.0	850	326	3.2027	1
1450	-121.95	37.96	18	2739	393.0	1072	374	6.1436	1
4148	-118.20	34.12	52	1580	426.0	1462	406	3.3326	0

4128 rows × 9 columns

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.

In [127…
```python
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(x_train,y_train)
```

Out[127]:
```
LinearRegression()
```

In [128…
```python
#Prediction of Test and Training set result
y_pred= regressor.predict(x_test)
x_pred= regressor.predict(x_train)
```

In [129…
```python
y_pred
```

Out[129]:
```
array([210776.44901247, 279878.45675391, 190478.34412314, ...,
        80625.22422957, 279916.99817768, 207126.99095494])
```

In [130…
```python
x_pred
```

Out[130]:
```
array([176068.00203749, 279731.67082882, 215080.09345602, ...,
       269791.25746823, 310713.48968759,  34706.78050531])
```

In [131…
```python
#import mean sq. error
from sklearn.metrics import mean_squared_error,r2_score
from math import sqrt
```

In [132…
```python
print(sqrt(mean_squared_error(y_test,y_pred)))
print((r2_score(y_test,y_pred)))
```

```
69826.89013012734
0.626076440482
```

7. Perform Decision Tree Regression :

Perform Decision Tree Regression on training data.

Predict output for test dataset using the fitted model.

Print root mean squared error from Decision Tree Regression.

In [133…
```python
#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeRegressor
DeTreeRegressor=DecisionTreeRegressor()
```

In [134…
```python
DeTreeRegressor.fit(x_train,y_train)
```

Out[134]:
```
DecisionTreeRegressor()
```

In [135…
```python
y_pred=DeTreeRegressor.predict(x_test)
```

In [136…
```python
print(sqrt(mean_squared_error(y_test,y_pred)))
print((r2_score(y_test,y_pred)))
```

```
66971.42634582732
0.6560332003175486
```

8. Perform Random Forest Regression :

Perform Random Forest Regression on training data.

Predict output for test dataset using the fitted model.

Print RMSE (root mean squared error) from Random Forest Regression

In [137…
```python
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestRegressor
RFRegressor=RandomForestRegressor()
RFRegressor.fit(x_train,y_train)
```

Out[137]:
```
RandomForestRegressor()
```

In [138…
```python
y_pred=RFRegressor.predict(x_test)
```

In [139…
```python
print(sqrt(mean_squared_error(y_test,y_pred)))
print((r2_score(y_test,y_pred)))
```

```
48585.0442094834
0.8189733006373846
```

9. 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.

In [140…
```python
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')
```

In [144…
```python
x_train_Income=x_train[['median_income']]
x_test_Income=x_test[['median_income']]
```

In [142…
```python
print(x_train_Income.shape)
print(y_train.shape)
```

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

In [145…
```python
linreg=LinearRegression()
linreg.fit(x_train_Income,y_train)
```

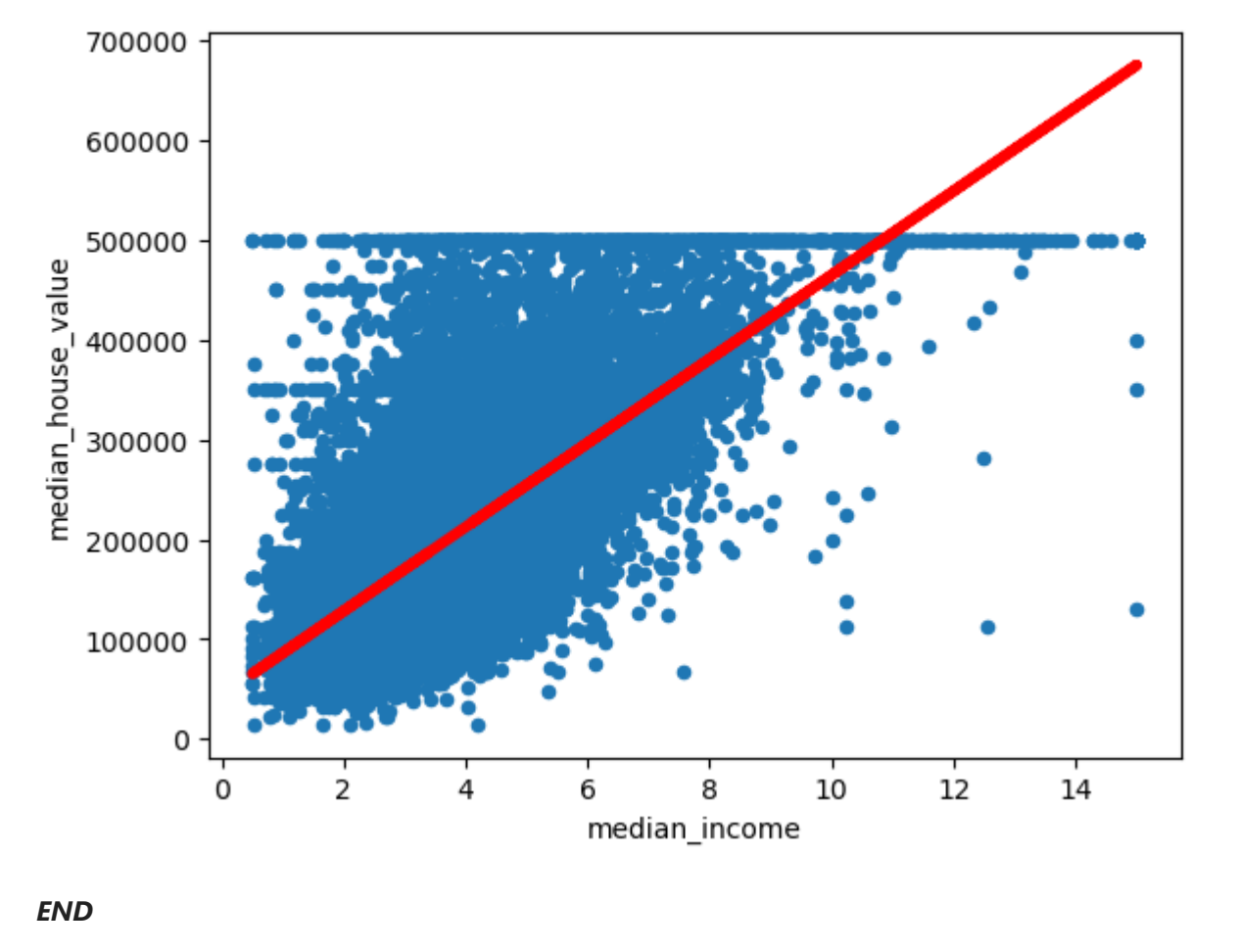
In [147…
```python
y_predict = linreg.predict(x_test_Income)
```

In [148…
```python
#print intercept and coefficient of the linear equation
print(linreg.intercept_, linreg.coef_)
print(sqrt(mean_squared_error(y_test,y_predict)))
print((r2_score(y_test,y_predict)))
```

```
44320.63522765713 [42032.17769894]
84941.05152406936
0.4466846804895944
```

In [158…
```python
#plot least square line
house_data.plot(kind='scatter',x='median_income',y='median_house_value')
plt.plot(x_test_Income,y_predict,c='red',linewidth=4)
```

Out[158]:
```
[<matplotlib.lines.Line2D at 0x21139b51b80>]
```



END

In [ ]:
```python
#Thank you!!!
```