In [53]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [54]:

```
df = pd.read_csv('1553768847-housing.csv')
```

In [55]:

```
#How many columns
len(df.columns)
```

Out[55]:

```
10
```

In [56]:

```
#view first 5 lines of df
df.head(5)
```

Out[56]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
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 [57]:

```
#are there duplacate row
sum(df.duplicated())
```

Out[57]:

```
0
```

In [58]:

```
# how many null values? Where do the null values exist? Do the empty values affect
df.isnull().sum()
```

Out[58]:

```
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 [59]:

```
#how many rows? How many columns?
df.shape
```

Out[59]:

```
(20640, 10)
```

In [60]:
```
#using mean, median and mode. fill null values givin these data condidions:
#mean - if data is normal distribution and there are few outliers
#median - use if your data is skewed and has outliers
#mode - use when working with categorical data
```

In [61]:
```
#lets take a look at the mean value
df['total_bedrooms'].mean()
```

Out[61]:
```
537.8705525375618
```

In [62]:
```
#our null values are all in total bedrooms. our data is numerical, so mode is off t
#seeing that houses can have any number of bedrooms(i.e potenticaly many outliers),
df['total_bedrooms'].median()
```

Out[62]:
```
435.0
```

In [63]:
```
#fill in missing values using median
df['total_bedrooms'] = df['total_bedrooms'].fillna(df['total_bedrooms'].median())
```

In [64]:
```
#no more missing values, good!
df.isnull().sum()
```

Out[64]:
```
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 [65]:
```
# identify unique non-digit values. Determne best way to encode (i.e. turn string v
df['ocean_proximity'].value_counts()
```

Out[65]:
```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

In [79]:
```
# replace values on ocean prox with numerical digits
df["ocean_proximity"].replace(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'IS
```

In [80]:
```
df.head(5)
```

Out[80]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
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 [81]:

```
# x data with house value dropped
x = df.drop('median_house_value',axis=1)

# y data with housing prices
y = df['median_house_value']
```

In [82]:

```
x.columns
```

Out[82]:

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

In [100…

```
#import train test split, use 20 - 30 % of data in test size
from sklearn.model_selection import train_test_split

x_train , x_test , y_train , y_test = train_test_split(x , y , test_size=0.25  )
```

In [101…

```
from sklearn import linear_model
```

In [102…

```
#fit training data. keep training and test data seperate, we dont want to give any
reg = linear_model.LinearRegression()
reg.fit(x_train , y_train)
```

Out[102…

```
▾ LinearRegression
LinearRegression()
```

In [103…

```
y_pred = reg.predict(x_test)
```

In [104…

```
y_pred
```

Out[104…

```
array([ 77175.32791787, 175177.19766364, 318518.21504612, ...,
       257381.44185985,  31229.57102564, 127814.59296108])
```

In [105…

```
#r2 score. want to be close to 1
reg.score(x_test , y_test)
```

Out[105…

```
0.6315061947601698
```

In [106…

```
reg.score(x_train , y_train)
```

Out[106…

```
0.637186977923716
```

In [107…

```
#use of built in r2 score. get MSE from train and test data
from sklearn.metrics import mean_absolute_error, r2_score

r2_score(y_test, y_pred)
```

Out[107…

```
0.6315061947601698
```

In [108…

```
mean_absolute_error(y_test, y_pred)
```

Out[108…

```
51088.66228130789
```

In [109…

```
y_pred_train = reg.predict(x_train)
```

In [110…

```
mean_absolute_error(y_train, y_pred_train)
```

Out[110…

```
50982.53564532673
```

In [ ]: