California Housing Price Prediction

In [56]:
```
import pandas as pd
```

In [57]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [58]:
```
# 1. Load the data :
```

In [59]:
```
# Read the “housing.csv” file from the folder into the program.
```

In [60]:
```
df = pd.read_excel('1553768847_housing.xlsx')
```

In [61]:
```
df.shape
```

Out[61]:
```
(20640, 10)
```

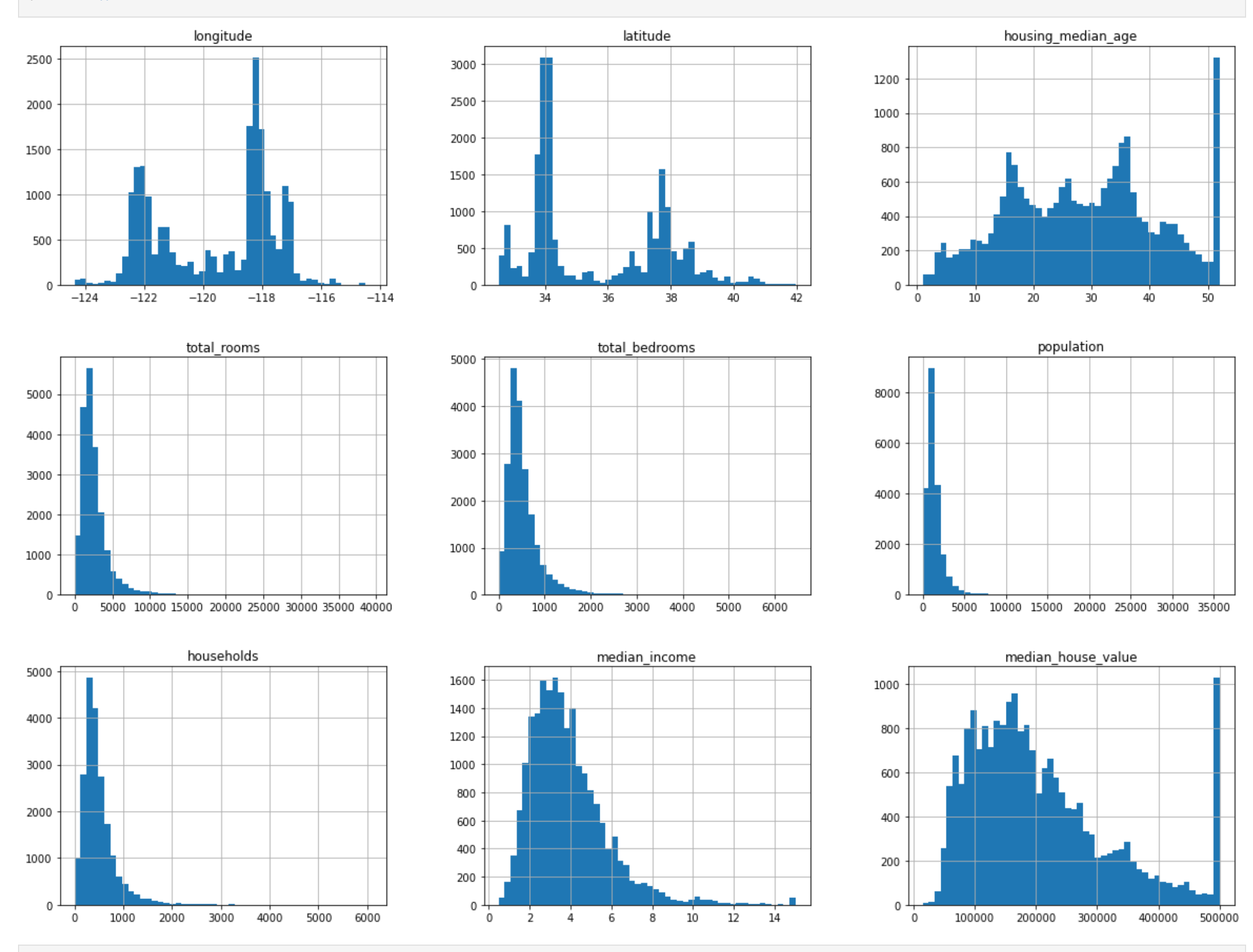
In [62]:
```
# Print first few rows of this data.
```

In [63]:
```
df.head()
```

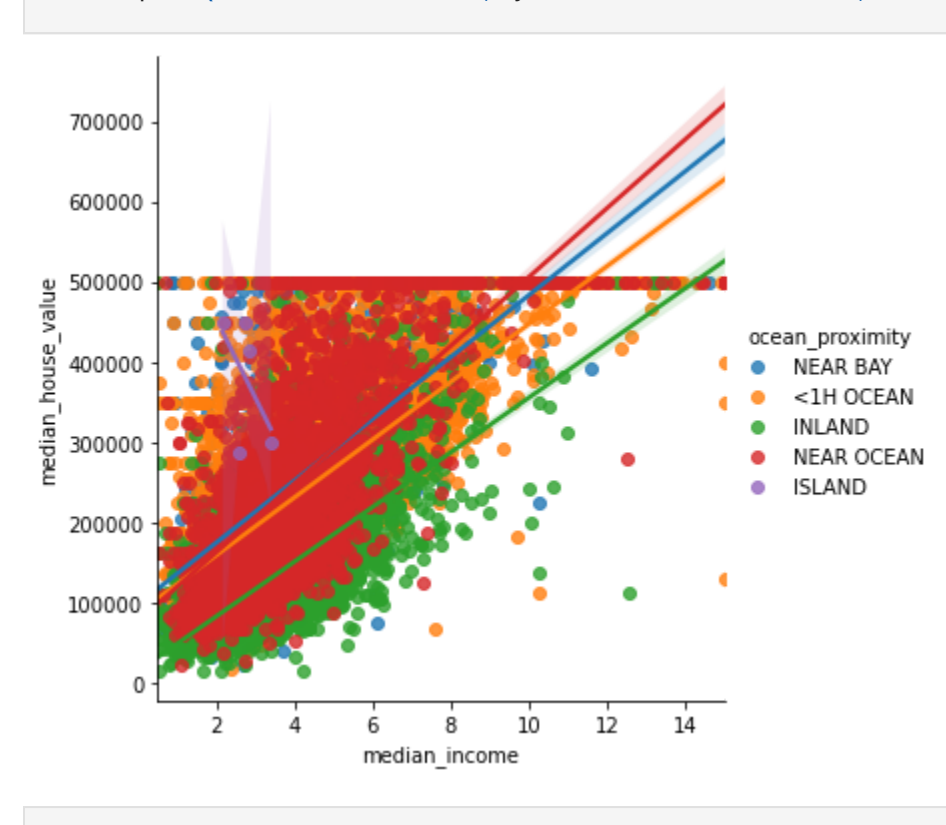
Out[63]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	342200

In [64]:
```
# To plot a histogram to understand the data
df.hist(bins=50, figsize=(20,15))
plt.show()
```



In [65]:
```
import seaborn as sns
sns.lmplot(x="median_income", y="median_house_value", hue="ocean_proximity", data=df);
```



In [66]:
```
df.tail()
```

Out[66]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	INLAND	78100
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	INLAND	77100
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	INLAND	92300
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	INLAND	84700
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	INLAND	89400

Predict the median housing price

In [67]:
```
df.corr()
```

Out[67]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069608	0.099773	0.055310	-0.015176	-0.045967
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066983	-0.108785	-0.071035	-0.079809	-0.144160
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.320451	-0.296244	-0.302916	-0.119034	0.105623
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930380	0.857126	0.918484	0.198050	0.134153
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000000	0.877747	0.979728	-0.007723	0.049686
population	0.099773	-0.108785	-0.296244	0.857126	0.877747	1.000000	0.907222	0.004834	-0.024650
households	0.055310	-0.071035	-0.302916	0.918484	0.979728	0.907222	1.000000	0.013033	0.065843
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007723	0.004834	0.013033	1.000000	0.688075
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049686	-0.024650	0.065843	0.688075	1.000000

In [68]:
```
df.columns
```

Out[68]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity', 'median_house_value'],
      dtype='object')
```

In [69]:
```
df.isnull().sum()
```

Out[69]:
```
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 [70]:
```
# Fill missing values with mean
```

In [71]:
```
df = df.fillna(df.mean())
```

In [72]:
```
df.isnull().sum()
```

Out[72]:
```
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 [73]:
```
df.dtypes
```

Out[73]:
```
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
```

In [74]:
```
# select categorical data
```

In [75]:
```
df_obj = df.select_dtypes(include=['object']).copy()
```

In [76]:
```
df_obj
```

Out[76]:

	ocean_proximity
0	NEAR BAY
1	NEAR BAY
2	NEAR BAY
3	NEAR BAY
4	NEAR BAY
...	...
20635	INLAND
20636	INLAND
20637	INLAND
20638	INLAND
20639	INLAND

20640 rows × 1 columns

In [77]:
```
# convert categorical data to dummy variables
```

In [78]:
```
df_dummy = pd.get_dummies(df[['ocean_proximity']])
```

In [79]:
```
df_dummy
```

Out[79]:

	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_NEAR BAY	ocean_proximity_NEAR OCEAN
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	1	0
...	...	...	...	...	...
20635	0	1	0	0	0
20636	0	1	0	0	0
20637	0	1	0	0	0
20638	0	1	0	0	0
20639	0	1	0	0	0

20640 rows × 5 columns

In [80]:
```
df_num = df.select_dtypes(include=['int64','float64']).copy()
df_num
```

Out[80]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200
...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	78100
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	77100
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	92300
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	84700
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	89400

20640 rows × 9 columns

In [81]:
```
df = pd.concat([df_num, df_dummy], axis=1)
```

In [82]:
```
df
```

Out[82]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity_<1H OCEAN	ocean_proximity_INLAN
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600	0	
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500	0	
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100	0	
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300	0	
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200	0	
...	...	...	...	...	...	...	...	...	...	...	
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	78100	0	
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	77100	0	
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	92300	0	
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	84700	0	
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	89400	0	

20640 rows × 14 columns

In [83]:
```
df.corr()
```

Out[83]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity_<1H OCEAN
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069260	0.099773	0.055310	-0.015176	-0.045967	0.321121
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066658	-0.108785	-0.071035	-0.079809	-0.144160	-0.446969
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.318998	-0.296244	-0.302916	-0.119034	0.105623	0.045300
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.927253	0.857126	0.918484	0.198050	0.134153	-0.003031
total_bedrooms	0.069260	-0.066658	-0.318998	0.927253	1.000000	0.873910	0.974725	-0.007682	0.049454	0.018220
population	0.099773	-0.108785	-0.296244	0.857126	0.873910	1.000000	0.907222	0.004834	-0.024650	0.074613
households	0.055310	-0.071035	-0.302916	0.918484	0.974725	0.907222	1.000000	0.013033	0.065843	0.042435
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007682	0.004834	0.013033	1.000000	0.688075	0.168876
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049454	-0.024650	0.065843	0.688075	1.000000	0.256617
ocean_proximity_<1H OCEAN	0.321121	-0.446969	0.045300	-0.003031	0.018220	0.074613	0.042435	0.168876	0.256617	1.000000
ocean_proximity_INLAND	-0.055575	0.351166	-0.236645	0.025624	-0.006433	-0.020732	-0.039402	-0.237496	-0.484859	-0.607669
ocean_proximity_ISLAND	0.009446	-0.016572	0.017020	-0.007572	-0.004361	-0.010412	-0.009077	-0.009228	0.023416	-0.013872
ocean_proximity_NEAR BAY	-0.474489	0.358771	0.255172	-0.023022	-0.019785	-0.060880	-0.010093	0.056197	0.160284	-0.314813
ocean_proximity_NEAR OCEAN	0.045509	-0.160818	0.021622	-0.009175	0.000676	-0.024264	0.001714	0.027344	0.141862	-0.342620

In [84]:
```
# Finding the correlation
corr_matrix = df.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[84]:
```
median_house_value            1.000000
median_income                 0.688075
ocean_proximity_<1H OCEAN     0.256617
ocean_proximity_NEAR BAY      0.160284
ocean_proximity_NEAR OCEAN    0.141862
total_rooms                   0.134153
housing_median_age            0.105623
households                    0.065843
total_bedrooms                0.049454
ocean_proximity_ISLAND        0.023416
population                   -0.024650
longitude                    -0.045967
latitude                     -0.144160
ocean_proximity_INLAND       -0.484859
Name: median_house_value, dtype: float64
```

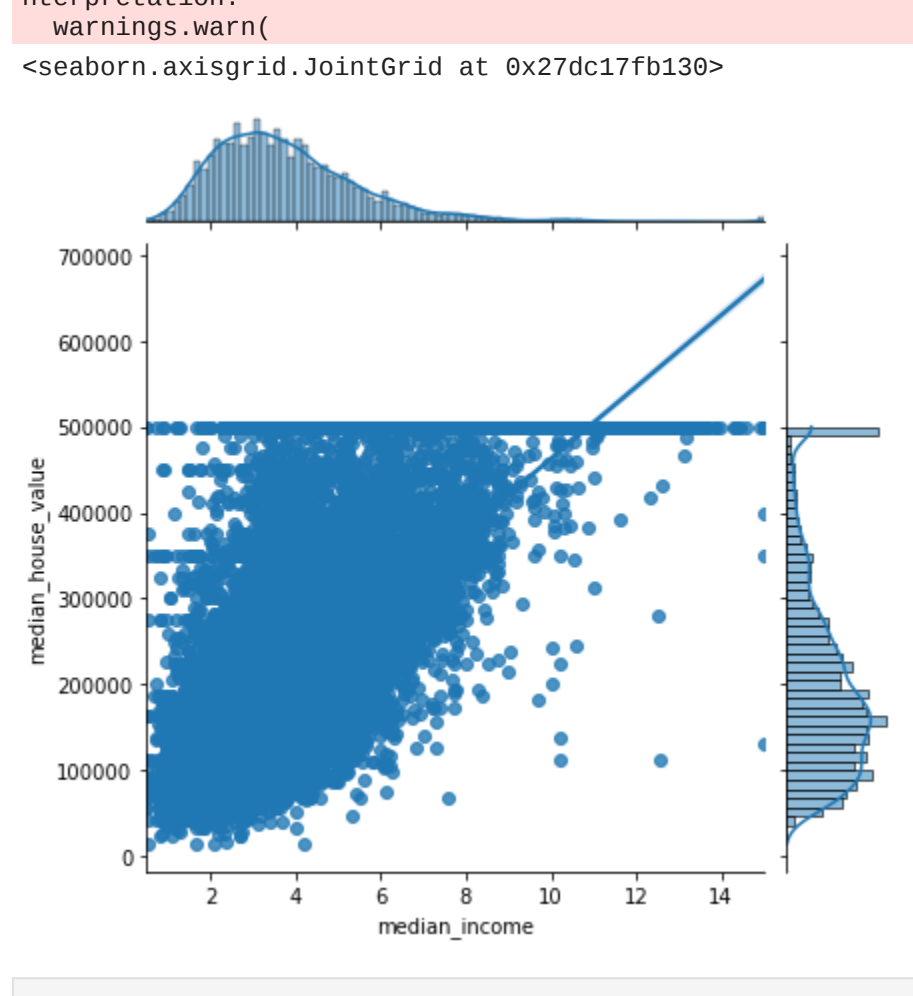
In [85]:
```
df.columns
```

Out[85]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity_<1H OCEAN',
       'ocean_proximity_INLAND', 'ocean_proximity_ISLAND',
       'ocean_proximity_NEAR BAY', 'ocean_proximity_NEAR OCEAN'],
      dtype='object')
```

In [86]:
```
sns.jointplot(df['median_income'], df['median_house_value'], kind = 'reg')
```

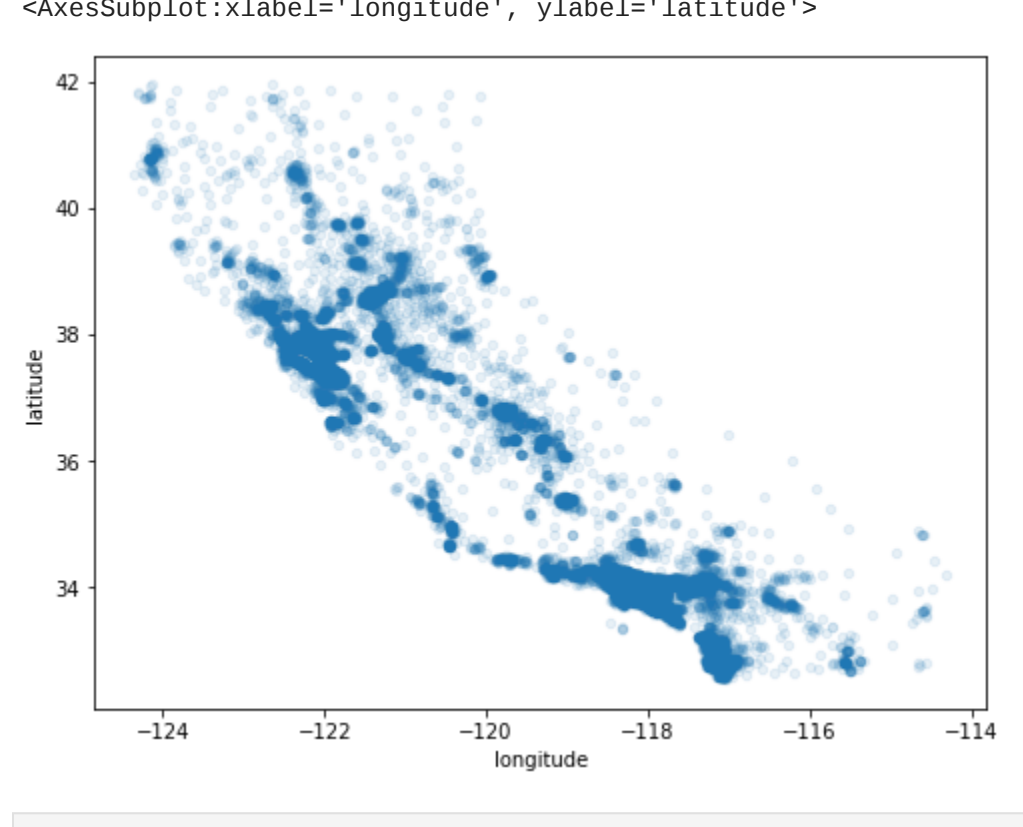
```
C:\Users\srivi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From versi
on 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misi
nterpretation.
  warnings.warn(
```

Out[86]:
```
<seaborn.axisgrid.JointGrid at 0x27dc17fb130>
```



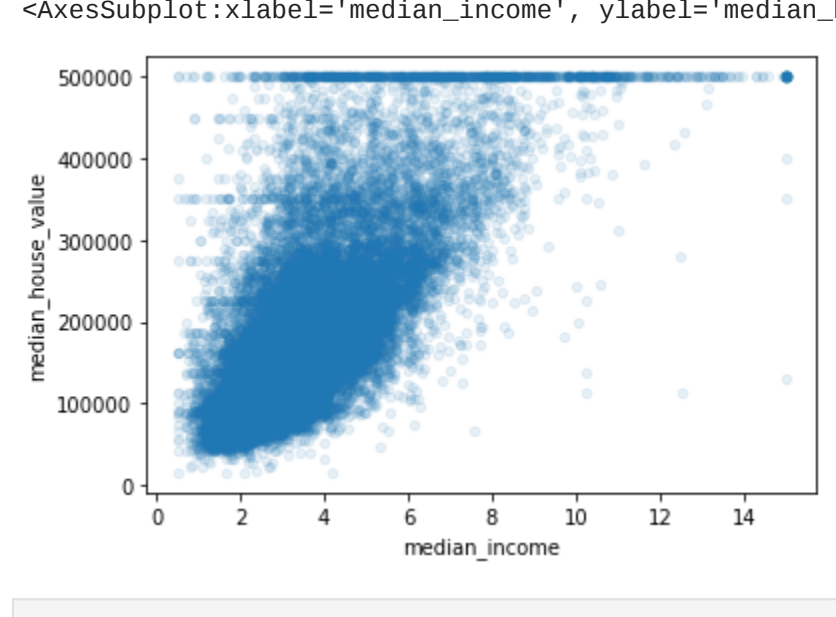
In [87]:
```
df.plot(kind="scatter", x="longitude", y="latitude", figsize = (8,6), alpha=0.1)
```

Out[87]:
```
<AxesSubplot:xlabel='longitude', ylabel='latitude'>
```



In [88]:
```
df.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
```

Out[88]:
```
<AxesSubplot:xlabel='median_income', ylabel='median_house_value'>
```



In [89]:
```
# Extract input (X) and output (Y) data from the dataset.
```

In [90]:
```
X = df[['median_income','total_rooms','housing_median_age','total_bedrooms', 'households', 'population', 'ocean_proximity_<1H OCEAN','ocean_proximi

y = df['median_house_value']
```

In [91]:
```
X.head()
```

Out[91]:

	median_income	total_rooms	housing_median_age	total_bedrooms	households	population	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_
0	8.3252	880	41	129.0	126	322	0	0	0	
1	8.3014	7099	21	1106.0	1138	2401	0	0	0	
2	7.2574	1467	52	190.0	177	496	0	0	0	
3	5.6431	1274	52	235.0	219	558	0	0	0	
4	3.8462	1627	52	280.0	259	565	0	0	0	

In [92]:
```
y.head()
```

Out[92]:
```
0    452600
1    358500
2    352100
3    341300
4    342200
Name: median_house_value, dtype: int64
```

In [93]:
```
# pip install sklearn
```

In [94]:
```
from sklearn.model_selection import train_test_split
```

In [95]:
```
# need to 'random' split data in train and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, random_state= 100)

# 80% in train
# 20% data in test
```

In [96]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

In [97]:
```
#Standardize training and test datasets.
#==============================================================================
# Feature scaling is to bring all the independent variables in a dataset into
# same scale, to avoid any variable dominating  the model. Here we will not
# transform the dependent variables.
#==============================================================================
from sklearn.preprocessing import StandardScaler
independent_scaler = StandardScaler()
X_train = independent_scaler.fit_transform(X_train)
X_test = independent_scaler.transform(X_test)
print(X_train[0:5,:])
print("test data")
print(X_test[0:5,:])
```

```
[[-0.24995448 -0.97047275 -1.24008206 -1.02650658 -1.1510077  -1.06404624
  -0.88604801  1.47409087 -0.01556621 -0.35650943 -0.3900092  -1.55444193
   1.63677864]
 [-0.3397354   0.278105   -1.39861471  0.38320985  0.41345981  0.33490581
  -0.88604801 -0.67838423 -0.01556621 -0.35650943  2.56404211 -0.62304675
  -0.14864428]
 [-0.27257105  0.63339882 -0.84375045  0.4238288   0.5311873   0.26233518
  -0.88604801  1.47409087 -0.01556621 -0.35650943 -0.3900092  -1.36019374
   2.2677527 ]
 [ 0.74180041 -0.45313925 -1.71568    -0.7254485  -0.68009775 -0.49747066
   1.12860701 -0.67838423 -0.01556621 -0.35650943 -0.3900092   1.32441591
  -1.34515806]
 [-0.80598461 -0.18781648 -0.76448413  0.09409852  0.12829768  0.00615208
  -0.88604801 -0.67838423 -0.01556621  2.80497487 -0.3900092  -1.37513591
   1.08993444]]
test data
[[-0.08641931 -0.70469094  0.58304336 -0.85208404 -0.85014856 -0.86294688
   1.12860701 -0.67838423 -0.01556621 -0.35650943 -0.3900092   0.7366906
  -0.85440046]
 [-1.38099368 -0.46645129  1.85130452 -0.20218087 -0.26674345 -0.23516715
  -0.88604801 -0.67838423 -0.01556621  2.80497487 -0.3900092  -1.34027085
   1.01982621]
 [ 0.50588168  0.86337582 -1.24008206  0.41188205  0.44223765  0.64617264
  -0.88604801  1.47409087 -0.01556621 -0.35650943 -0.3900092   0.22865687
  -0.12527487]
 [-0.51861188 -0.743709   -0.76448413 -0.80190769 -0.91555272 -0.88043378
  -0.88604801  1.47409087 -0.01556621 -0.35650943 -0.3900092  -1.77857446
   1.71623463]
 [-0.51492153  1.7943007   0.18671175  2.46194426  2.51685761  1.33865391
  -0.88604801 -0.67838423 -0.01556621  2.80497487 -0.3900092  -1.46478892
   1.09460833]]
```

In [98]:
```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()

lm.fit(X_train, y_train)   # training
```

Out[98]:
```
LinearRegression()
```

In [99]:
```
print("Intercept is "+ str(lm.intercept_))
print("coefficients  is "+str(lm.coef_))
```

```
Intercept is 207155.7384326552
coefficients  is [ 73137.11024033  -9970.89077627  13187.82226366  30756.38131542
  26782.7251827  -43017.63670541   6052.65005594 -12857.73939296
   3077.35953444   2970.13993194   5854.30540256 -53220.96609143
 -53875.69762748]
```

In [100…
```
# predict the price of house on test data
prediction = lm.predict(X_test)
```

In [101…
```
print(prediction)
```

```
[222683.93415873 157719.56920823 182780.86329646 ... 309946.79321349
 106703.12035302 173065.98895344]
```

In [102…
```
# Actual value
y_test
```

Out[102…
```
8151     245300
53       104200
3039     146200
9484     154200
9307     324200
          ...
16733    117200
5264     500001
12374    239300
19662     85700
11942    113900
Name: median_house_value, Length: 4128, dtype: int64
```

In [103…
```
'''
error = Actual - prediction
MSE
RMSE
MAE
'''

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, mean_absolute_percentage_error
from math import sqrt

# error metrics for test data
MSE_test = mean_squared_error(y_test, prediction)
RMSE_test = sqrt(MSE_test)

MAE_test = mean_absolute_error(y_test, prediction)
MAPE_test = mean_absolute_percentage_error(y_test, prediction)

print('Mean square error for test data is :', MSE_test)
print('Root Mean square error for test data is :', RMSE_test)
print('Mean Absolute error for test data is :', MAE_test)
print('Mean Absolute percentage error for test data is :', MAPE_test)
```

```
Mean square error for test data is : 4508965795.672326
Root Mean square error for test data is : 67148.83316687138
Mean Absolute error for test data is : 49193.19405515743
Mean Absolute percentage error for test data is : 0.28348907603528345
```

In [104…
```
# predict the price of house on train data
prediction_train = lm.predict(X_train)
```

In [105…
```
prediction_train
```

Out[105…
```
array([132411.16647085, 227974.6216864 , 108262.57505254, ...,
       140461.85311686, 171218.58837006, 278179.79826821])
```

In [106…
```
# error metric for train data

'''
error = Actual - prediction
Actual = y_train
Prediction = prediction_train
MSE
RMSE
MAE
'''

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, mean_absolute_percentage_error
from math import sqrt

# error metrics for train data
MSE_train = mean_squared_error(y_train, prediction_train)
RMSE_train = sqrt(MSE_train)

MAE_train = mean_absolute_error(y_train, prediction_train)
MAPE_train = mean_absolute_percentage_error(y_train, prediction_train)

print('Mean square error for train data is :', MSE_train)
print('Root Mean square error for train data is :', RMSE_train)
print('Mean Absolute error for train data is :', MAE_train)
print('Mean Absolute percentage error for train data is :', MAPE_train)

###########################################################################
r_square = r2_score(y_train, prediction_train)
print('R square for train data is :', r_square)
```

```
Mean square error for train data is : 4774929421.829318
Root Mean square error for train data is : 69100.86411781923
Mean Absolute error for train data is : 50029.206642993304
Mean Absolute percentage error for train data is : 0.28841643462182376
R square for train data is : 0.6403184439877669
```

In [107…
```
print(prediction[0:5])
print(y_test[0:5])
```

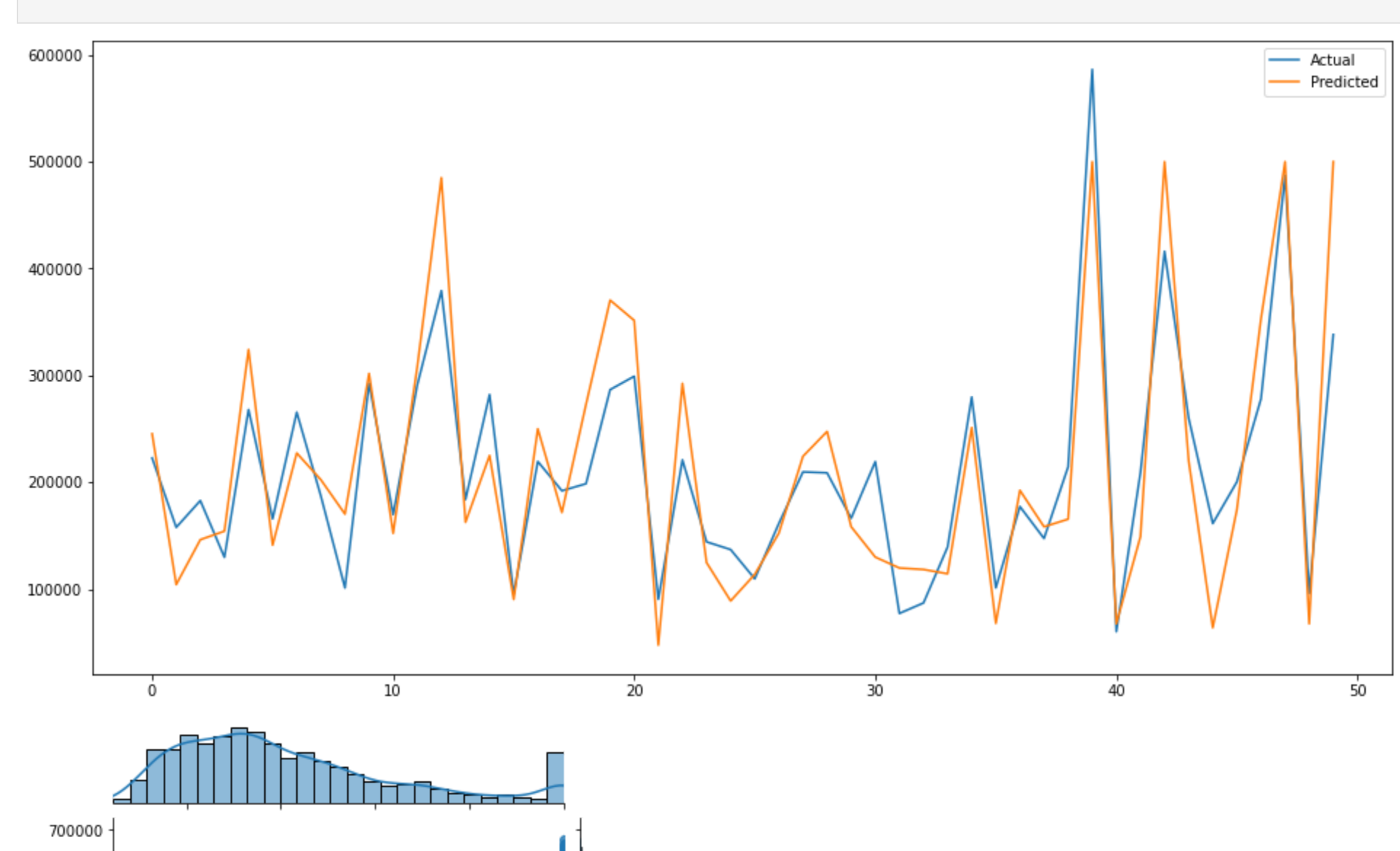
```
[222683.93415873 157719.56920823 182780.86329646 129736.60918772
 267954.76052913]
8151    245300
53      104200
3039    146200
9484    154200
9307    324200
Name: median_house_value, dtype: int64
```

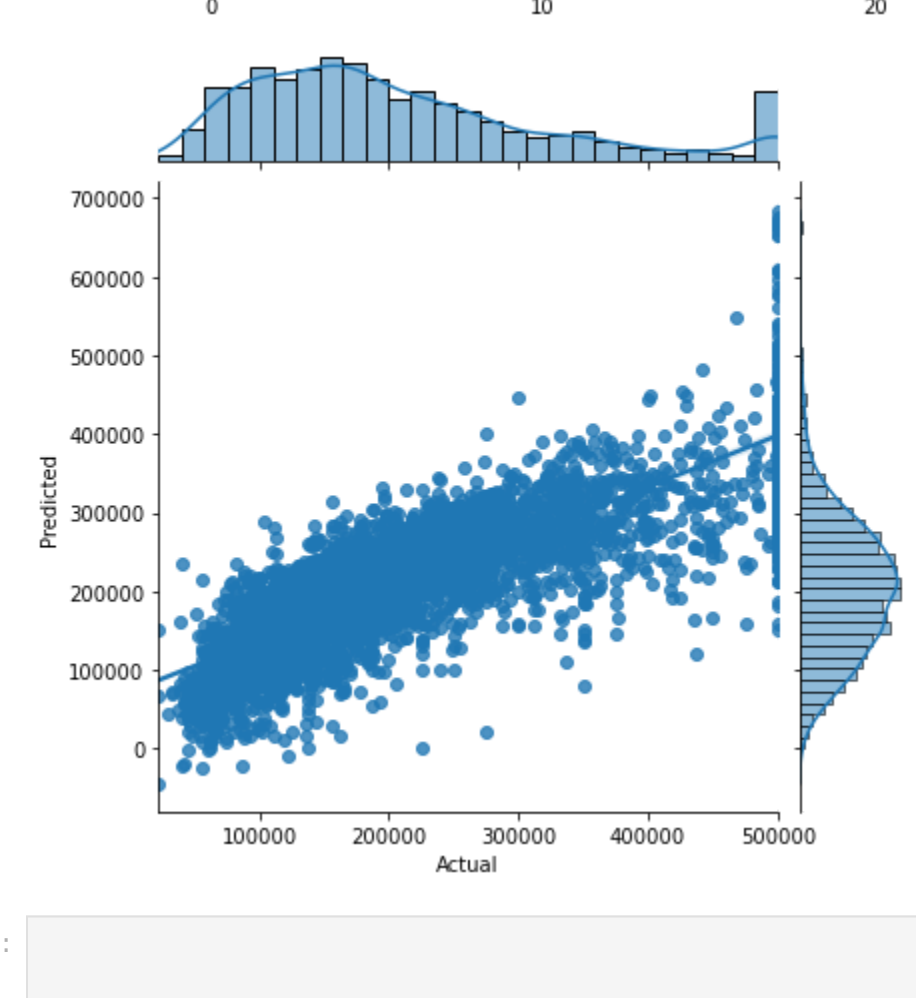
In [108…
```
test = pd.DataFrame({'Predicted':prediction,'Actual':y_test})
print(test)
```

```
           Predicted  Actual
8151   222683.934159  245300
53     157719.569208  104200
3039   182780.863296  146200
9484   129736.609188  154200
9307   267954.760529  324200
...              ...     ...
16733  132167.033875  117200
5264   507540.536441  500001
12374  309946.793213  239300
19662  106703.120353   85700
11942  173065.988953  113900

[4128 rows x 2 columns]
```

In [109…
```
fig= plt.figure(figsize=(16,8))
test = test.reset_index()
test = test.drop(['index'],axis=1)
plt.plot(test[:50])
plt.legend(['Actual','Predicted'])
sns.jointplot(x='Actual',y='Predicted',data=test,kind='reg',);
```





In [ ]: