In [1]:
```
import pandas as pd
import numpy as np
Cali_df = pd.read_csv(r'C:\Users\keert\Desktop\AI ML TILL  CERT\AI and ML\datasets\
Cali_df.head(8)
```

Out[1]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
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
5	-122.25	37.85	52	919	213.0	413	193
6	-122.25	37.84	52	2535	489.0	1094	514
7	-122.25	37.84	52	3104	687.0	1157	647

Null Values Counts

In [2]:
```
Cali_df.isna().sum()
```

Out[2]:
```
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 [3]:
```
mean_value=Cali_df['total_bedrooms'].mean()
Cali_df['total_bedrooms'].fillna(value=mean_value, inplace=True) #filling missing
```

In [4]:
```
Cali_df.isna().sum()
```

Out[4]:
```
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 [5]:
```
#identify category data
Cali_df.select_dtypes(exclude=["number","bool_"])
```

Out[5]:

	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 [6]:
```
Cali_df["ocean_proximity"].value_counts()
```

Out[6]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [7]:
```
class_mapping = {label:idx for idx,label in enumerate(np.unique(Cali_df['ocean_pro
class_mapping
```

Out[7]:
```
{'<1H OCEAN': 0, 'INLAND': 1, 'ISLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
```

In [11]:
```
# np.unique(Cali_df['ocean_proximity'])
```

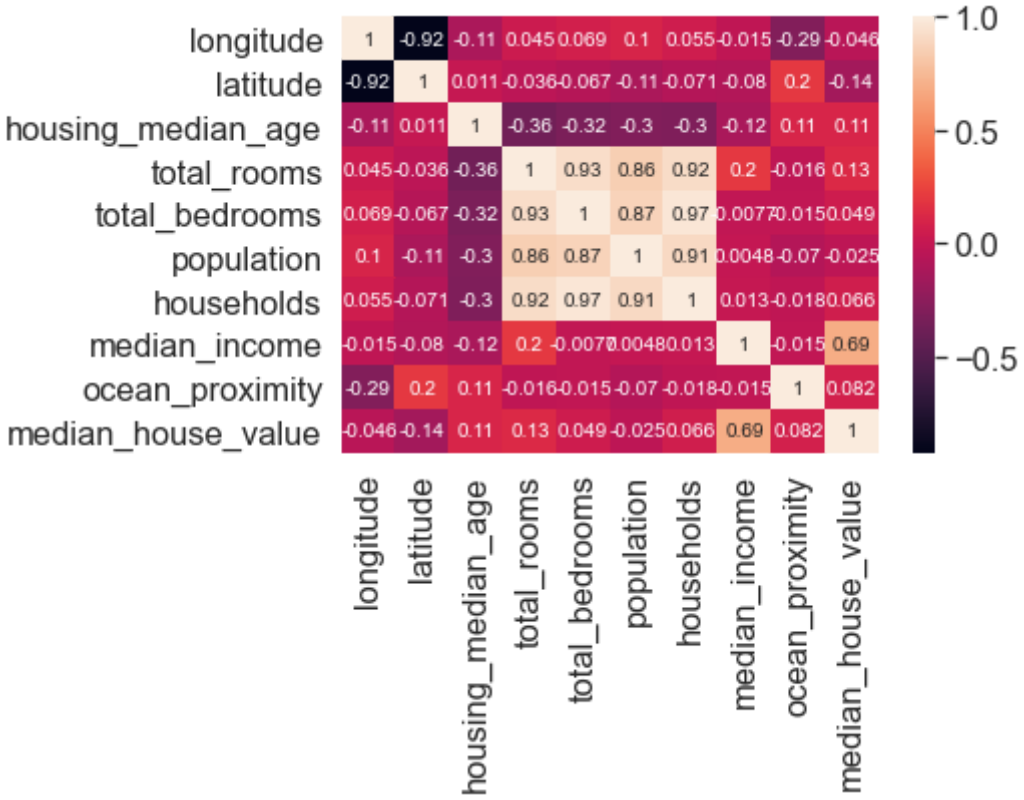
In [13]:
```
Cali_df['ocean_proximity'] =Cali_df['ocean_proximity'].map(class_mapping)
```

In [14]:
```
Cali_df['ocean_proximity']
```

Out[14]:
```
0        3
1        3
2        3
3        3
4        3
        ..
20635    1
20636    1
20637    1
20638    1
20639    1
Name: ocean_proximity, Length: 20640, dtype: int64
```

In [367…
```
import seaborn as sb
sb.heatmap(Cali_df.corr(),annot=True)
```

Out[367]:
```
<AxesSubplot:>
```



In [368…
```
# creating test and train set

y=Cali_df.iloc[:,-1:]
X=Cali_df.iloc[:,:-1]
```

In [369…
```
# Splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, test_size
```

In [370…
```
#from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import scale
from sklearn.preprocessing import minmax_scale
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import Normalizer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelBinarizer
```

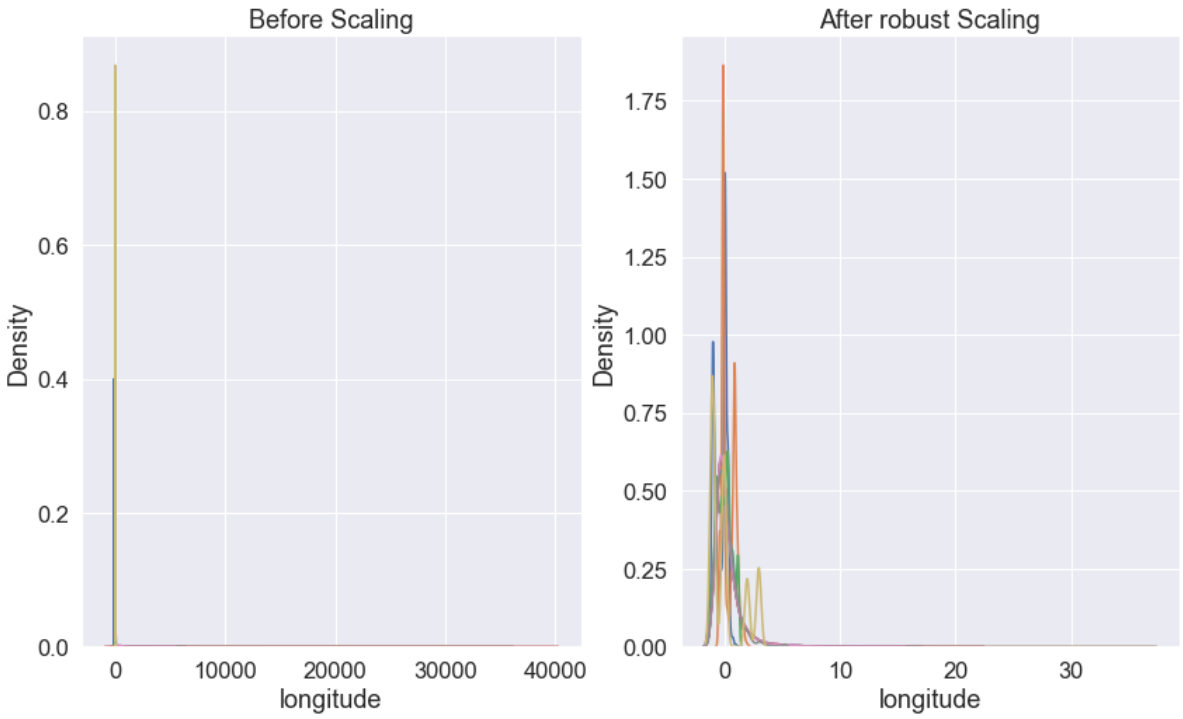
Scaling data using Robust Scalar

In [371…
```
scaler = RobustScaler()
X_train_scaled =scaler.fit_transform(pd.DataFrame(X_train))
#X_train_scaled
X_test_scaled =scaler.fit_transform(pd.DataFrame(X_test))
#X_test_scaled
```

In [372…
```
colnames=list(Cali_df.columns)
colnames.remove("median_house_value")


X_train_scaled = pd.DataFrame(X_train_scaled, columns=colnames)
X_test_scaled= pd.DataFrame(X_test_scaled, columns=colnames)
```

In [375…
```
# import plotting libraries
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
# sns.set(style="white", color_codes=True)
# sns.set(font_scale=1.5)
```

In [376…
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(14, 8))
ax1.set_title('Before Scaling')

sns.kdeplot(X_train['longitude'], ax=ax1)
sns.kdeplot(X_train['latitude'], ax=ax1)
sns.kdeplot(X_train['housing_median_age'], ax=ax1)
sns.kdeplot(X_train['total_rooms'], ax=ax1)
sns.kdeplot(X_train['total_bedrooms'], ax=ax1)
sns.kdeplot(X_train['population'], ax=ax1)
sns.kdeplot(X_train['households'], ax=ax1)
sns.kdeplot(X_train['median_income'],  ax=ax1)
sns.kdeplot(X_train['ocean_proximity'],  ax=ax1)


ax2.set_title('After robust Scaling')
sns.kdeplot(X_train_scaled['longitude'], ax=ax2)
sns.kdeplot(X_train_scaled['latitude'], ax=ax2)
sns.kdeplot(X_train_scaled['housing_median_age'], ax=ax2)
sns.kdeplot(X_train_scaled['total_rooms'], ax=ax2)
sns.kdeplot(X_train_scaled['total_bedrooms'], ax=ax2)
sns.kdeplot(X_train_scaled['population'], ax=ax2)
sns.kdeplot(X_train_scaled['households'], ax=ax2)
sns.kdeplot(X_train_scaled['median_income'],  ax=ax2)
sns.kdeplot(X_train_scaled['ocean_proximity'],  ax=ax2);
```



In [379…
```
# import the ML algorithm
from sklearn.linear_model import LinearRegression
# instantiate
linreg = LinearRegression()
```

```
# fit the model to the training data (learn the coefficients)
linreg.fit(X_train_scaled, y_train)
```

Out[379]:
```
LinearRegression()
```

In [380…
```
# return beta coeff
linreg.coef_
```

Out[380]:
```
array([[-162257.45501854, -160525.74398954,   22077.54724331,
         -12451.55612941,   30836.42970186,  -36552.20876899,
          23354.8313849 ,   87805.78334548,     428.30524124]])
```

In [381…
```
y_pred = linreg.predict(X_test_scaled)
y_pred
```

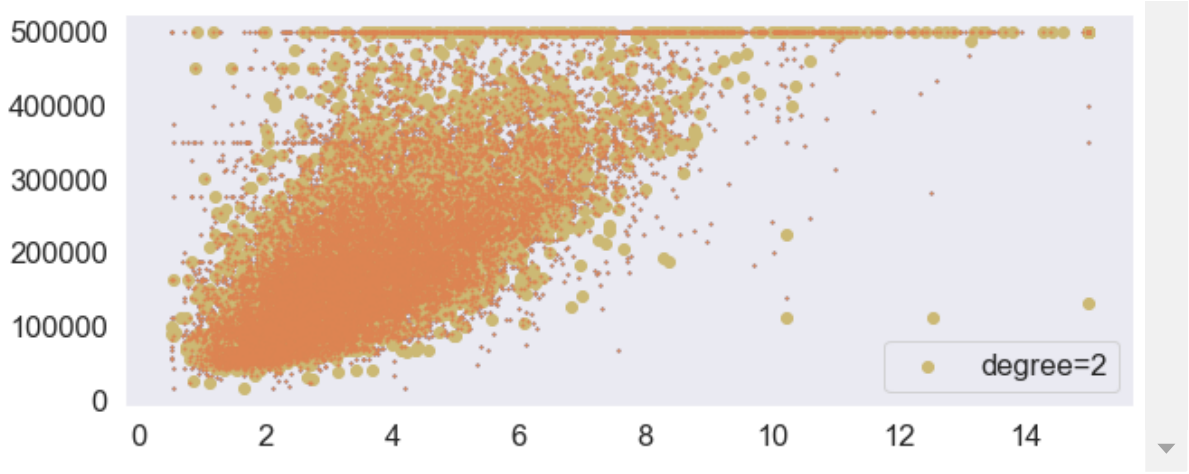
Out[381]:
```
array([[243042.88275132],
       [107472.52210259],
       [256537.28804575],
       ...,
       [284603.42337753],
       [259474.62685262],
       [159026.55024748]])
```

In [382…
```
from sklearn import metrics
# Model evaluation metrics for regression
print('y-intercept             : ', linreg.intercept_)
print('beta coefficients       : ', linreg.coef_)
print('Mean Abs Error   MAE    : ', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Sq  Error MSE      : ', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Sq Error RMSE : ', np.sqrt(metrics.mean_squared_error(y_test, y_pr
print('MAPE                    : ', np.mean(np.abs((y_test - y_pred) / y_test)) * 1
print('MPE                     : ', np.mean((y_test - y_pred) / y_test) * 100)
print('r2 value                : ', metrics.r2_score(y_test, y_pred))
```

```
y-intercept             :  [205222.38829937]
beta coefficients       :  [[-162257.45501854 -160525.74398954   22077.54724331  -
12451.55612941
    30836.42970186  -36552.20876899   23354.8313849    87805.78334548
      428.30524124]]
Mean Abs Error   MAE    :  51652.60532422244
Mean Sq  Error MSE      :  4911017383.769714
Root Mean Sq Error RMSE :  70078.65141232182
MAPE                    :  median_house_value    32.210016
dtype: float64
MPE                     :  median_house_value   -14.534111
dtype: float64
r2 value                :  0.6255964270944927
```

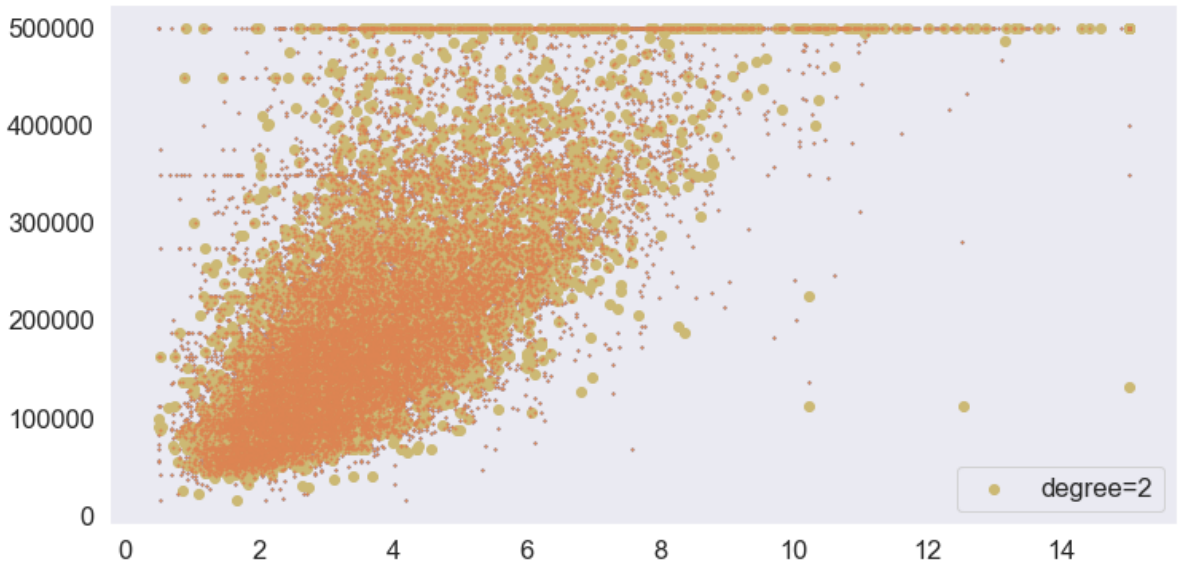
In [392…
```
#bonus exercise
linreg.fit(X_train['median_income'].values.reshape(-1,1), y_train)

y_pred_MedInc = linreg.predict(X_test['median_income'].values.reshape(-1,1))
plt.figure(figsize=(10, 4))
X1=X_train['median_income'].values.reshape(-1,1)
X2=X_test['median_income'].values.reshape(-1,1)
plt.scatter(X1, y_train, s=1)
plt.scatter(X2, y_test, color='y', label='degree=2')
plt.scatter(X1, y_train, s=1)
plt.grid()
plt.legend()
```

Out[392]:
```
<matplotlib.legend.Legend at 0x1b052189640>
```



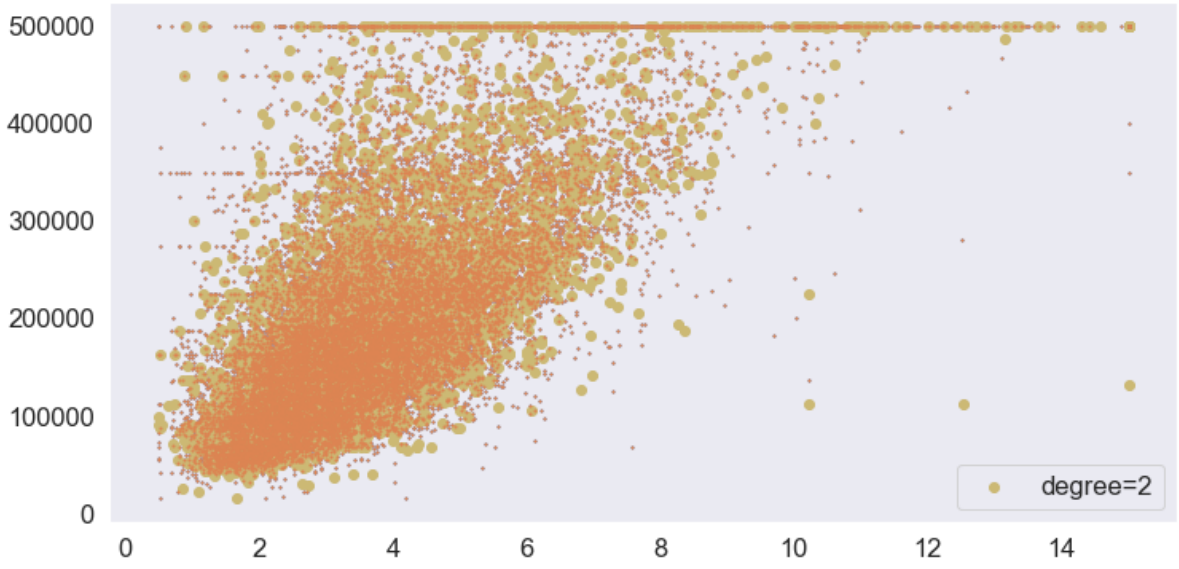
In [389…

Out[389]: <matplotlib.legend.Legend at 0x1b052025e80>



In [388…

Out[388]: <matplotlib.legend.Legend at 0x1b04b6e4ee0>



In [ ]: