In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.cluster import KMeans
import warnings
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures


# Ignore FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Load dataset
houseprice_df = pd.read_excel('HousePrice_Dataset.xlsx')
houseprice_df.head()
```

Out[1]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inc
0	-122.23	37.88	41	880	129.0	322	126	8.3
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3
2	-122.25	37.85	52	1627	280.0	565	259	3.8
3	-122.25	37.85	52	919	213.0	413	193	4.0
4	-122.25	37.84	52	2535	489.0	1094	514	3.6

In [2]:
```python
# Checking input data shape & info
df = houseprice_df.copy()
print(df.shape)
print(df.info())
```

```
(18565, 10)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18565 entries, 0 to 18564
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           18565 non-null  float64
 1   latitude            18565 non-null  float64
 2   housing_median_age  18565 non-null  int64
 3   total_rooms         18565 non-null  int64
 4   total_bedrooms      18376 non-null  float64
 5   population          18565 non-null  int64
 6   households          18565 non-null  int64
 7   median_income       18565 non-null  float64
 8   median_house_value  18565 non-null  int64
 9   ocean_proximity     18565 non-null  object
dtypes: float64(4), int64(5), object(1)
memory usage: 1.4+ MB
None
```

In [3]:
```python
#Checking for NULL values in Data
df.isnull().sum()
```

Out[3]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        189
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

In [4]:
```
#Dropping NULL Values
df = df.dropna()
```

In [5]:
```
#Confirming dropna operation
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18376 entries, 0 to 18564
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           18376 non-null  float64
 1   latitude            18376 non-null  float64
 2   housing_median_age  18376 non-null  int64  
 3   total_rooms         18376 non-null  int64  
 4   total_bedrooms      18376 non-null  float64
 5   population          18376 non-null  int64  
 6   households          18376 non-null  int64  
 7   median_income       18376 non-null  float64
 8   median_house_value  18376 non-null  int64  
 9   ocean_proximity     18376 non-null  object 
dtypes: float64(4), int64(5), object(1)
memory usage: 1.5+ MB
```

In [6]:
```
#Checking for any duplicates
df.duplicated().sum()
```

Out[6]:
```
0
```

In [7]:
```
#Checking data composition for numerical features
df.describe()
```

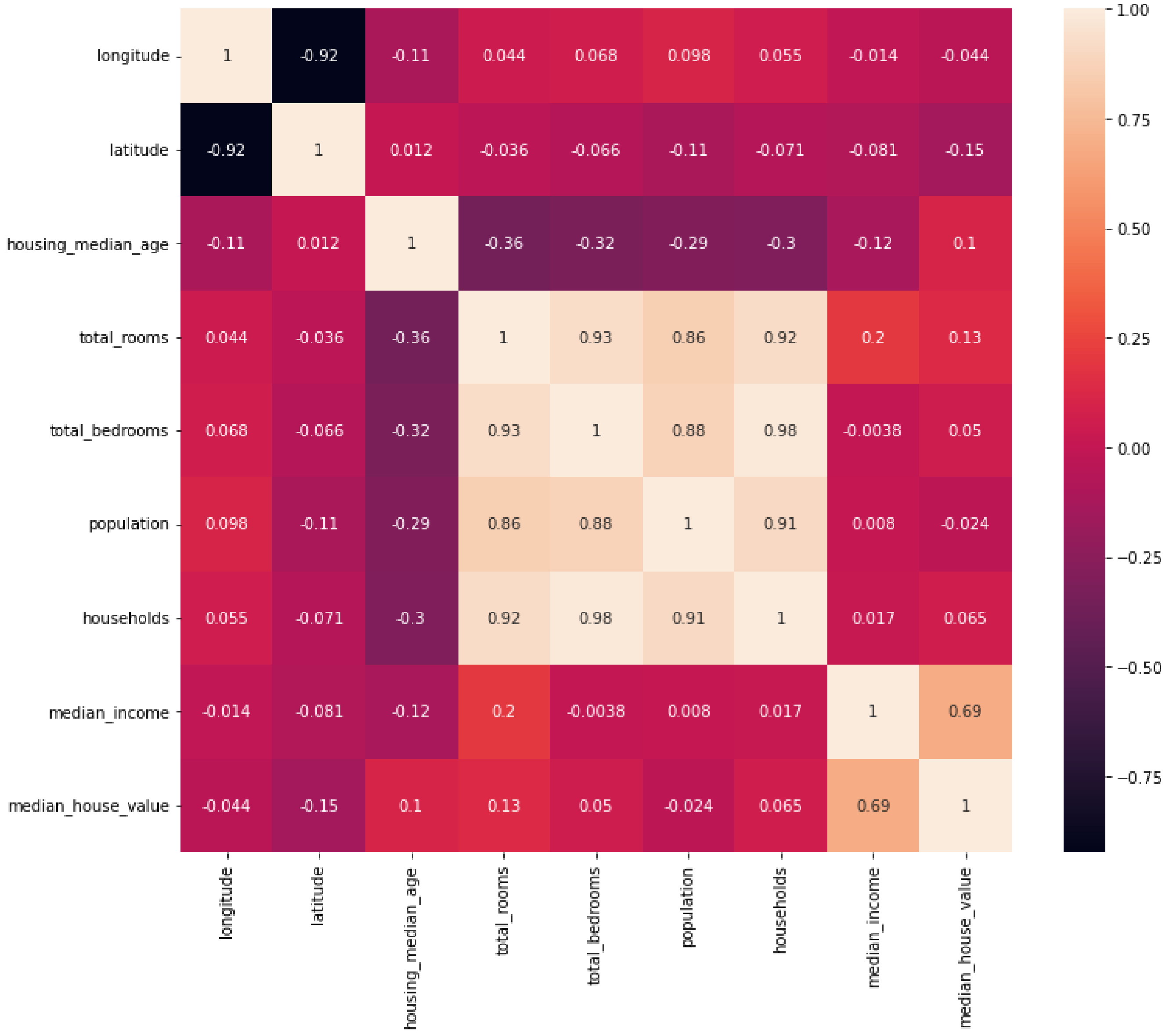
Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
count	18376.000000	18376.000000	18376.000000	18376.000000	18376.000000	18376.000000	18376.00
mean	-119.571095	35.635164	28.605736	2635.302188	537.711199	1425.810786	499.37
std	2.003042	2.137485	12.570789	2200.534974	424.125849	1143.481721	384.51
min	-124.350000	32.540000	1.000000	2.000000	2.000000	3.000000	2.00
25%	-121.800000	33.930000	18.000000	1444.000000	295.000000	786.000000	280.00
50%	-118.500000	34.260000	29.000000	2123.000000	434.000000	1165.500000	408.00
75%	-118.010000	37.720000	37.000000	3137.000000	646.000000	1722.000000	603.00
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.00

In [8]:
```
#Standardizing the numerical feilds.
# df[['housing_median_age']] = preprocessing.scale(df[['housing_median_age']]).astype('float6
# df[['total_rooms']] = preprocessing.scale(df[['total_rooms']]).astype('float64')
```

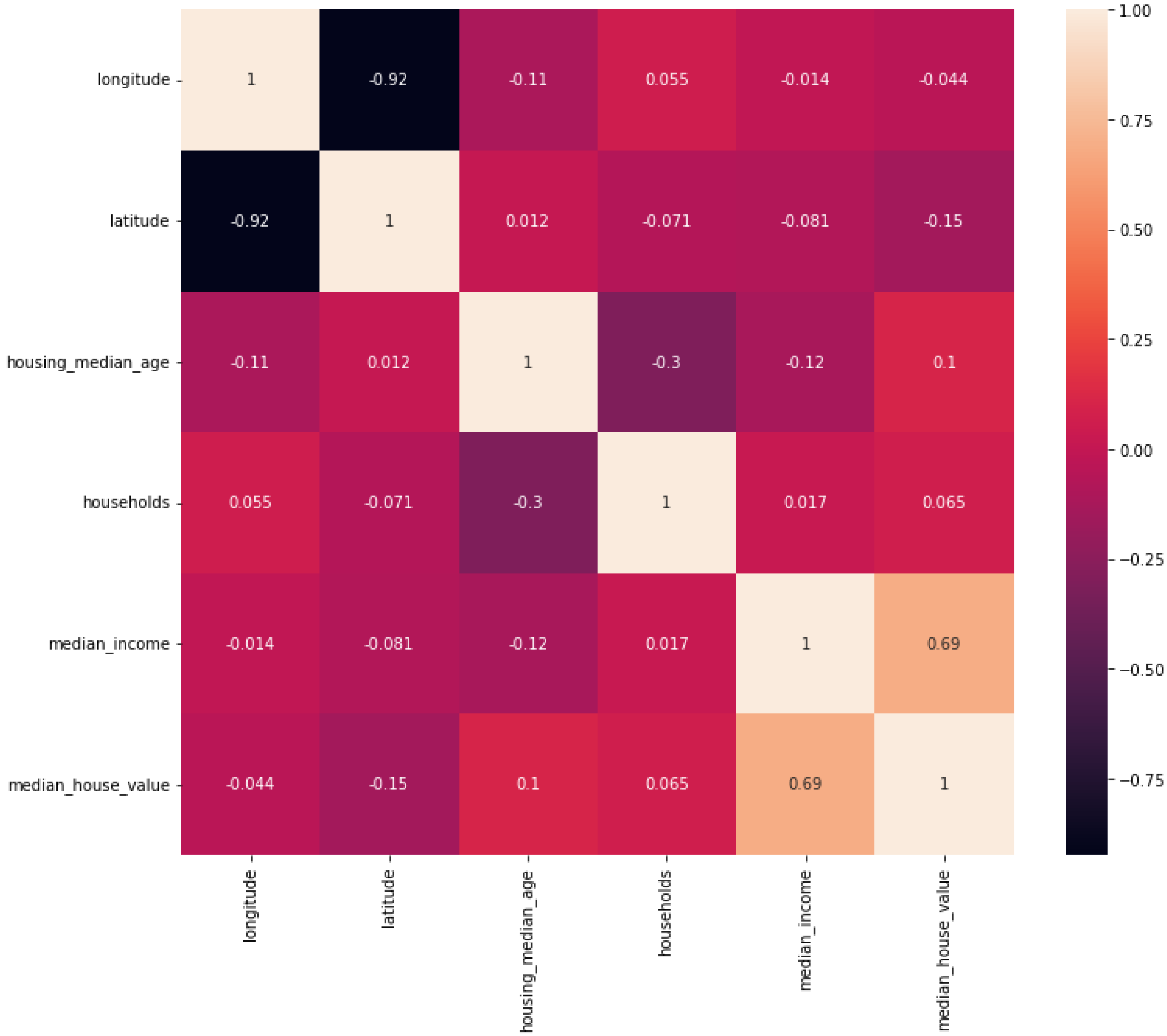
```
# df[['total_bedrooms']] = preprocessing.scale(df[['total_bedrooms']]).astype('float64')
# df[['population']] = preprocessing.scale(df[['population']]).astype('float64')
# df[['households']] = preprocessing.scale(df[['households']]).astype('float64')
# df[['median_income']] = preprocessing.scale(df[['median_income']]).astype('float64')
# df[['median_house_value']] = preprocessing.scale(df[['median_house_value']]).astype('float6

# df.head()
```

In [9]:
```
# Correlation matrix
figure1, a = plt.subplots(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



In [10]:
```
df=df.drop(['total_rooms','total_bedrooms','population'], axis=1)
```

In [11]:
```
# Correlation matrix
figure1, a = plt.subplots(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



In [12]:
```
#Checking unique values in categorical column
df['ocean_proximity'].unique()
```

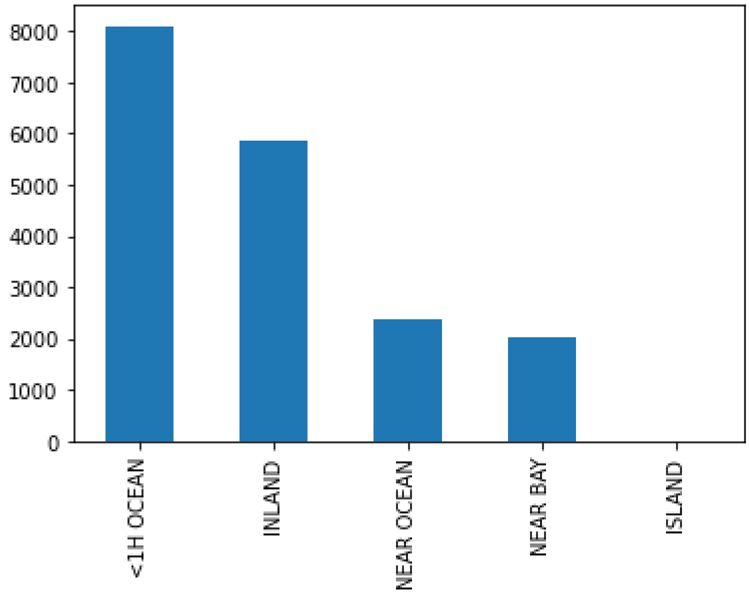
Out[12]:
```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

In [13]:
```
#House price based on categorical value.
df.groupby(['ocean_proximity'])['median_house_value'].describe()
```

Out[13]:

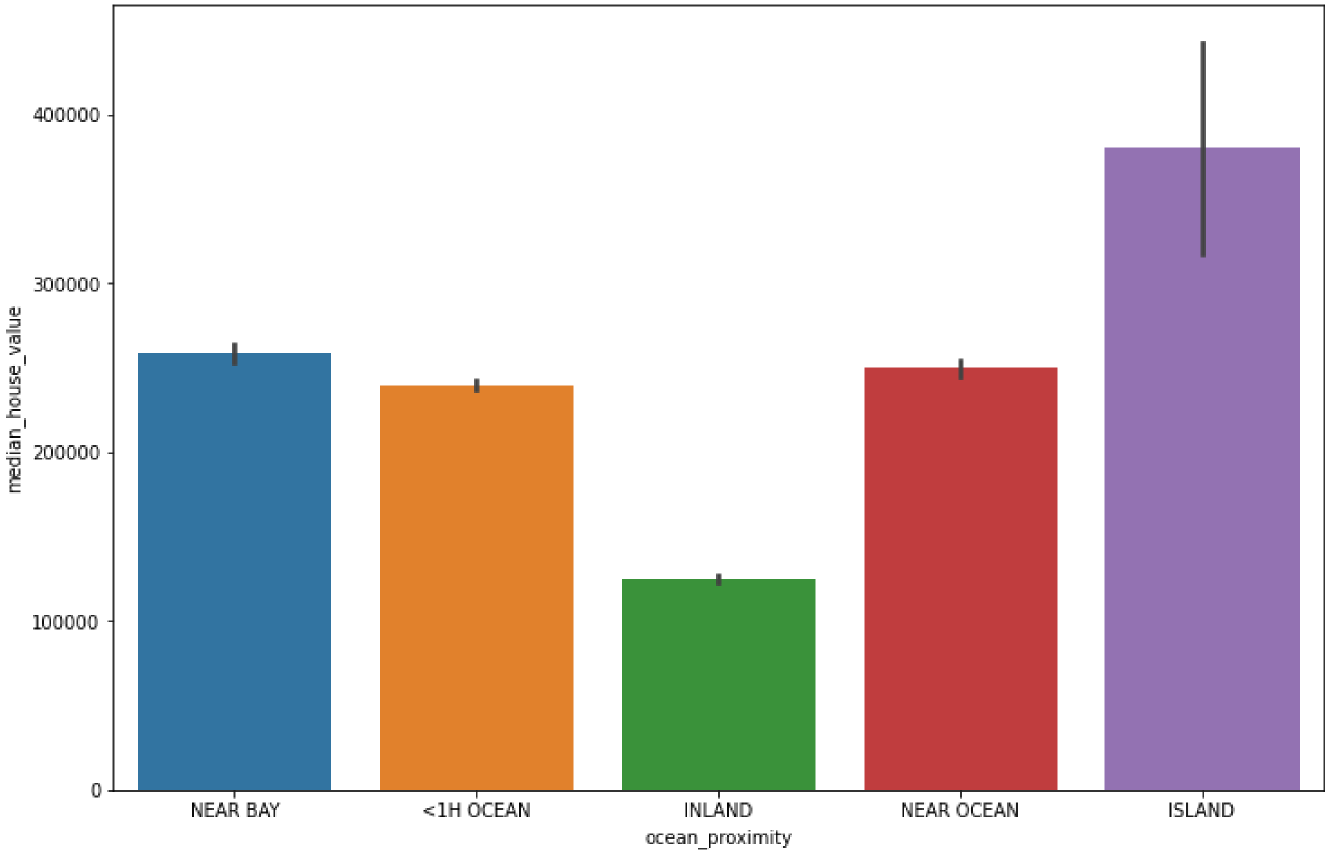
ocean_proximity	count	mean	std	min	25%	50%	75%	max
<1H OCEAN	8096.0	239973.465415	106101.457273	17500.0	164175.0	214750.0	289425.0	500001.0
INLAND	5869.0	124937.335492	70783.931014	14999.0	77500.0	108300.0	148600.0	500001.0
ISLAND	5.0	380440.000000	80559.561816	287500.0	300000.0	414700.0	450000.0	450000.0
NEAR BAY	2034.0	258756.622911	122646.084078	22500.0	162500.0	231800.0	345525.0	500001.0
NEAR OCEAN	2372.0	249858.342327	122701.540906	22500.0	150000.0	229800.0	323825.0	500001.0

In [14]:
```
#Same information using a bar chart.
df['ocean_proximity'].value_counts().plot(kind='bar')
plt.show()
```



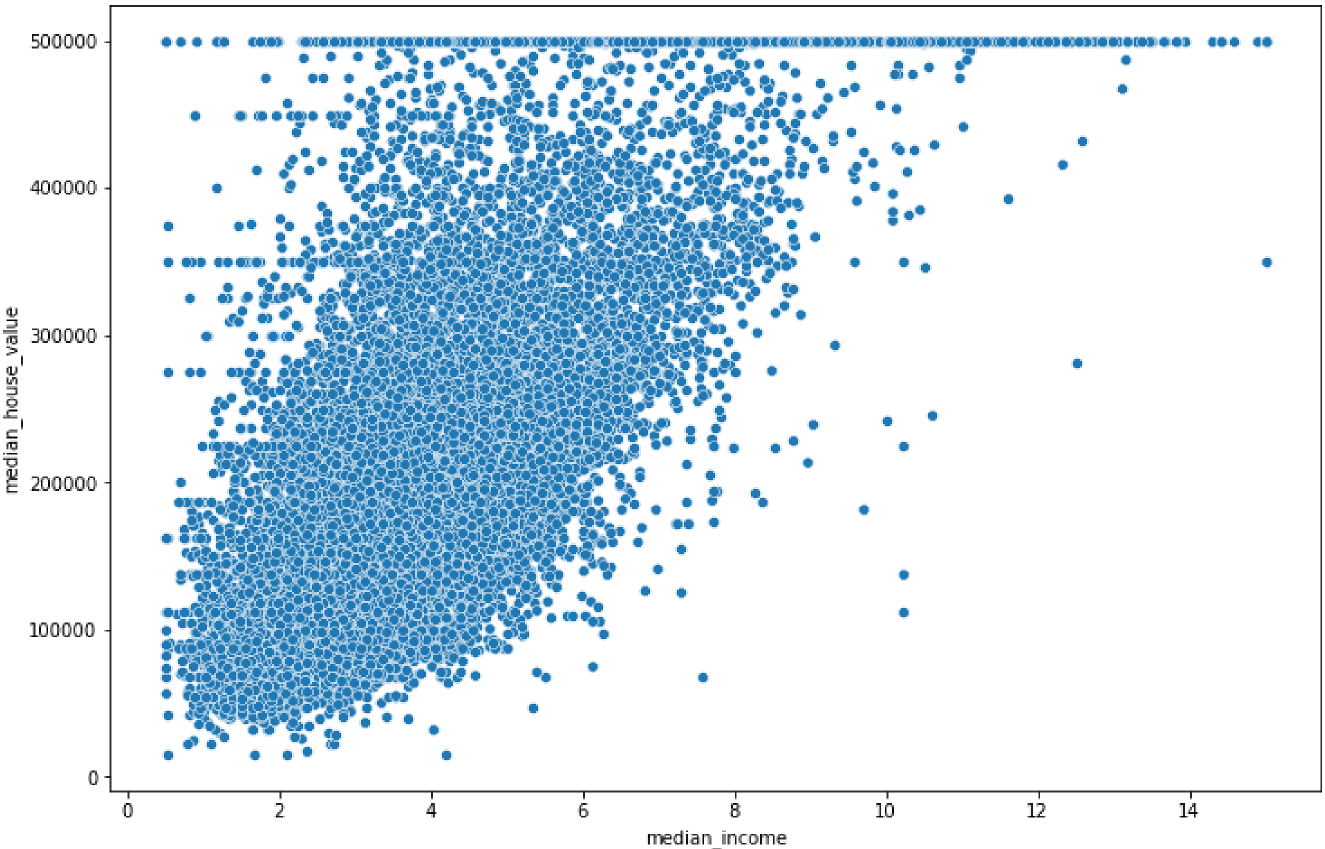
In [15]:
```
figure2, ax = plt.subplots(figsize = (12,8))
sns.barplot(x = 'ocean_proximity', y = 'median_house_value' , data = df)
```

Out[15]: <AxesSubplot:xlabel='ocean_proximity', ylabel='median_house_value'>



In [16]:
```
figure3, ax = plt.subplots(figsize = (12,8))
sns.scatterplot(x = 'median_income', y = 'median_house_value' , data = df)
```

Out[16]: <AxesSubplot:xlabel='median_income', ylabel='median_house_value'>



In [17]:
```
# Convert categorical variable 'ocean_proximity' into dummy/indicator variables
df = pd.get_dummies(df, columns=['ocean_proximity'])
```

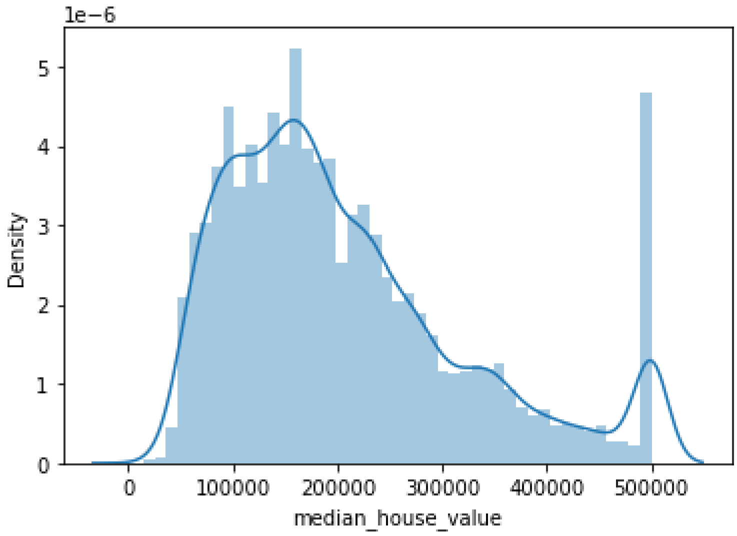
In [18]:
```
#Renaing feature name as it should not contain charaters like <>= etc...This will give proble
df.rename(columns ={'ocean_proximity_<1H OCEAN' : 'ocean_proximity_LT1H OCEAN'}, inplace = Tr
```

In [19]:
```
#Checking for dummies action & feature name change
df.head()
```

Out[19]:

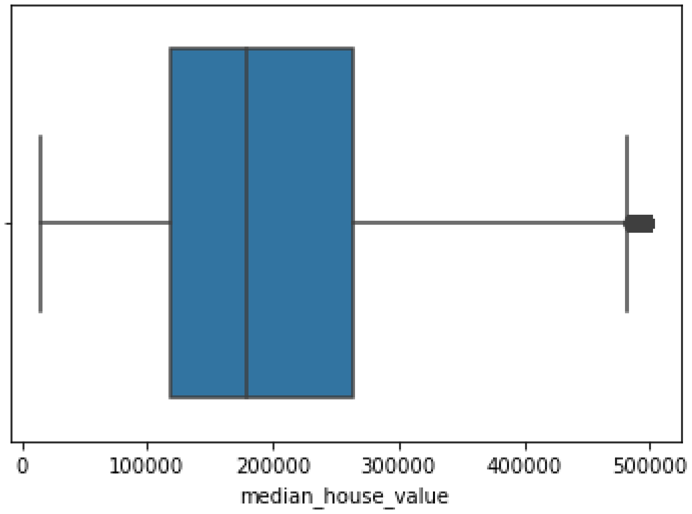
	longitude	latitude	housing_median_age	households	median_income	median_house_value	ocean_proximity C
0	-122.23	37.88	41	126	8.3252	452600	
1	-122.22	37.86	21	1138	8.3014	358500	
2	-122.25	37.85	52	259	3.8462	342200	
3	-122.25	37.85	52	193	4.0368	269700	
4	-122.25	37.84	52	514	3.6591	299200	

In [20]:
```
#Distribution of median house value which we are predicting
#Data shows right skewed and some outliers
sns.distplot(df['median_house_value'])
plt.show()
```



In [21]:
```
#Confirming via box plot
sns.boxplot(df['median_house_value'])
```

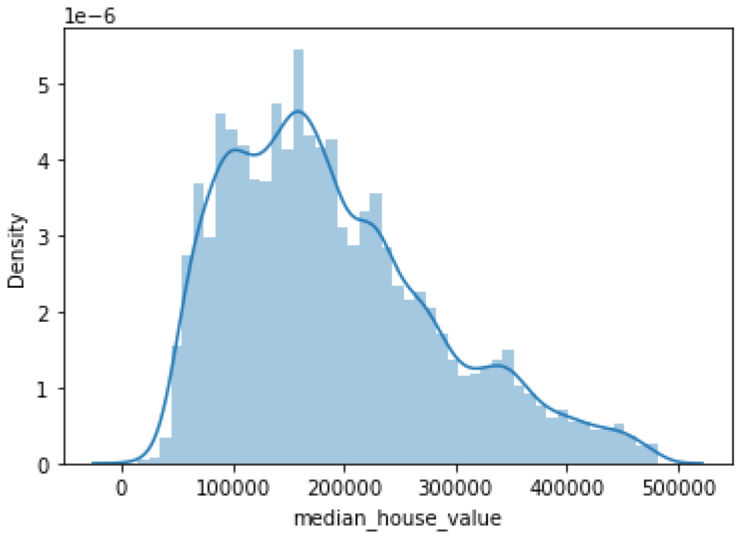
Out[21]: <AxesSubplot:xlabel='median_house_value'>



In [22]:
```
#Trying to identify outliers and remove them
# IQR
# Calculate the upper and lower limits
Q1 = df['median_house_value'].quantile(0.25)
Q3 = df['median_house_value'].quantile(0.75)
IQR = Q3 - Q1

# Set the threshold for outliers
lower_threshold = Q1 - 1.5 * IQR
upper_threshold = Q3 + 1.5 * IQR

# Identify outlier rows using boolean masks
outliers_lower = df['median_house_value'] < lower_threshold
outliers_upper = df['median_house_value'] > upper_threshold

# Display the number of outliers
print(f"Number of lower outliers: {outliers_lower.sum()}")
print(f"Number of upper outliers: {outliers_upper.sum()}")

# Remove the outliers
df = df[~(outliers_lower | outliers_upper)].copy()

# Display the shape of the cleaned DataFrame
print(f"Shape of the cleaned DataFrame: {df.shape}")
```

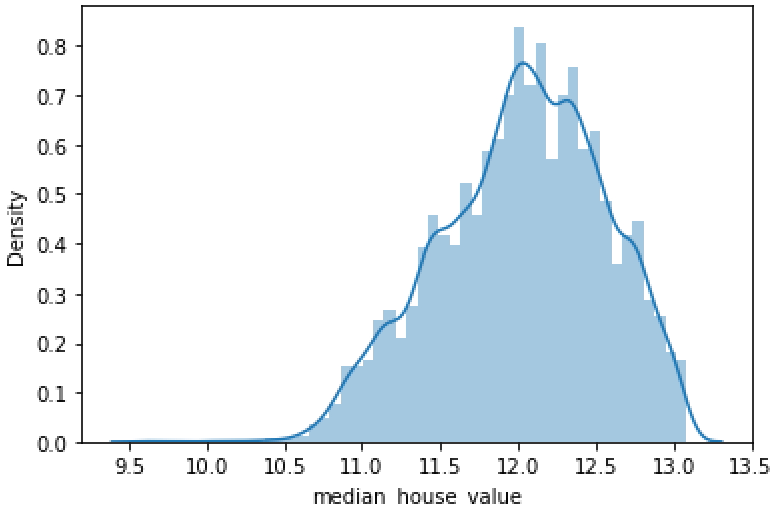
```
Number of lower outliers: 0
Number of upper outliers: 957
Shape of the cleaned DataFrame: (17419, 11)
```

In [23]:
```python
#Confirming outliers removal
sns.distplot(df['median_house_value'])
plt.show()
```



In [24]:
```python
sns.distplot(np.log(df['median_house_value']))
```

Out[24]:
```
<AxesSubplot:xlabel='median_house_value', ylabel='Density'>
```



In [25]:
```python
#Checking if any column belongs to object
category_columns = df.dtypes[df.dtypes=='object'].index
numerical_columns = df.dtypes[df.dtypes!='object'].index
print(category_columns)
print(numerical_columns)
```

```
Index([], dtype='object')
Index(['longitude', 'latitude', 'housing_median_age', 'households',
       'median_income', 'median_house_value', 'ocean_proximity_LT1H OCEAN',
       'ocean_proximity_INLAND', 'ocean_proximity_ISLAND',
       'ocean_proximity_NEAR BAY', 'ocean_proximity_NEAR OCEAN'],
      dtype='object')
```

In [26]:
```python
#Seperating input features and labels
X = df.drop('median_house_value', axis = 1)
Y = np.log(df['median_house_value'])
print(type(X))
print(type(Y))
```

```
print(X.shape)
print(Y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(17419, 10)
(17419,)
```

In [27]:
```
# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
df.head()
```

```
(13935, 10)
(3484, 10)
(13935,)
(3484,)
```

Out[27]:

	longitude	latitude	housing_median_age	households	median_income	median_house_value	ocean_proximity C
0	-122.23	37.88	41	126	8.3252	452600	
1	-122.22	37.86	21	1138	8.3014	358500	
2	-122.25	37.85	52	259	3.8462	342200	
3	-122.25	37.85	52	193	4.0368	269700	
4	-122.25	37.84	52	514	3.6591	299200	

In [28]:
```
# # Standardize the features using StandardScaler
# scaler = StandardScaler()
# X_train = scaler.fit_transform(x_train)
# X_test = scaler.transform(x_test)
```

In [29]:
```
#Defining functions to calculate R2 and error rate
def model_evaluation(y_test, y_pred):
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    # r2_scr = r2_score(y_test, y_pred)
    return {'mae': mae, 'mse': mse, 'rmse': rmse}

def model_instantiation(model, x_train, x_test, y_train, y_test, y_pred, model_name):
    training_r2 = model.score(x_train, y_train)
    testing_r2 = model.score(x_test, y_test)
    eval_return = model_evaluation(y_test, y_pred)
    result_metric = {
        'Train_R2': training_r2,
        'Test_R2': testing_r2,
        'Test_MSE': eval_return['mse'],
        'Test_RMSE': eval_return['rmse'],
        'Test_MAE': eval_return['mae']
    }
    result = pd.DataFrame(result_metric, index=[model_name])
    return result
```

In [30]:
```
from sklearn.linear_model import LinearRegression
```

In [31]:
```
#Instantiating Linear reg model
linear_reg = LinearRegression()
```

```
linear_reg.fit(x_train, y_train)
```

Out[31]:

```
▾ LinearRegression
LinearRegression()
```

In [32]:

```
#Predicting via LR
y_pred_LR = linear_reg.predict(x_test)
```

In [33]:

```
LR_df = model_instantiation(linear_reg ,x_train,x_test,y_train,y_test,y_pred_LR , 'Linear Reg
LR_df
```

Out[33]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
Linear Regression	0.630962	0.624499	0.10712	0.327293	0.251028

In [34]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
```

In [35]:

```
#Decision Tree
Decs_Tree = DecisionTreeRegressor(max_depth = 8 , min_samples_leaf =10 , min_samples_split =
Decs_Tree.fit(x_train, y_train)
```

Out[35]:

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=8, min_samples_leaf=10, min_samples_split=10)
```

In [36]:

```
y_pred_DT = Decs_Tree.predict(x_test)
```

In [37]:

```
DT_df = model_instantiation(Decs_Tree ,x_train,x_test,y_train,y_test,y_pred_DT , 'DTree_Regre
DT_df
```

Out[37]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
DTree_Regression	0.763267	0.717443	0.080606	0.283912	0.208418

In [38]:

```
#Randon forest
Rand_Forest = RandomForestRegressor(n_estimators=300, max_depth = 10, min_samples_split=12)
Rand_Forest.fit(x_train, y_train)
```

Out[38]:

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=10, min_samples_split=12, n_estimators=300)
```

In [39]:

```
y_pred_RF = Rand_Forest.predict(x_test)
```

In [40]:

```
RF_df = model_instantiation(Rand_Forest ,x_train,x_test,y_train,y_test,y_pred_RF , 'RF_Regres
RF_df
```

Out[40]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
RF_Regression	0.850595	0.774959	0.064198	0.253374	0.180186

In [41]:

```
Rand_Forest_2 = RandomForestRegressor(n_estimators=300, max_depth = 9, min_samples_split=4)
Rand_Forest_2.fit(x_train, y_train)

y_pred_RF_2 = Rand_Forest_2.predict(x_test)
```

```
RF_df_2 = model_instantiation(Rand_Forest_2 ,x_train,x_test,y_train,y_test,y_pred_RF_2 , 'RF2
RF_df_2
```

Out[41]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
RF2_Regression	0.830124	0.763076	0.067588	0.259977	0.186851

In [42]:

```
ada =AdaBoostRegressor(n_estimators = 300, random_state = 10)
ada.fit(x_train,y_train)

y_pred_ada = ada.predict(x_test)

ada_df = model_instantiation(ada ,x_train,x_test,y_train,y_test,y_pred_ada , 'ADA_Regression'
ada_df
```

Out[42]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
ADA_Regression	0.599874	0.586475	0.117968	0.343464	0.271343

In [43]:

```
from xgboost.sklearn import XGBRegressor

xgb_model = XGBRegressor()

# Define the parameter grid to search
# param_grid = {
#     'n_estimators': [100, 200, 300],
#     'learning_rate': [0.01, 0.1, 0.2],
#     'max_depth': [3, 4, 5],
#     'subsample': [0.8, 0.9, 1.0],
#     'colsample_bytree': [0.8, 0.9, 1.0],
#     'reg_alpha': [0, 0.1, 0.5],  # Regularization term on weights (L1)
#     'reg_lambda': [0, 0.1, 0.5]  # Regularization term on weights (L2)
# }

# # # Create GridSearchCV
# grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, scoring='neg_mean_sq

# grid_search.fit(x_train,y_train,early_stopping_rounds=10,eval_set=[(x_test, y_test)], verbo
# XGB_Best_Model = grid_search.best_estimator_
# y_pred_xgboost = XGB_Best_Model.predict(x_test)


xgb_model.fit(x_train,y_train)
y_pred_xgboost = xgb_model.predict(x_test)

xgboost_df = model_instantiation(xgb_model, x_train, x_test, y_train, y_test, y_pred_xgboost,
xgboost_df
```

Out[43]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
XGB_Regression	0.923149	0.801669	0.056579	0.237862	0.163638

In [ ]:

In [44]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,  Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.pipeline import make_pipeline
from tensorflow.keras.regularizers import l2

# # Separate features (X) and target variable (y)
X = df.drop('median_house_value', axis=1)
```

```
y = df['median_house_value']

# # Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# # Create a pipeline with StandardScaler and PolynomialFeatures
pipeline = make_pipeline(StandardScaler(), PolynomialFeatures(degree=2, include_bias=False))

# # Fit and transform the training data
X_train_poly = pipeline.fit_transform(X_train)

# # Transform the testing data
X_test_poly = pipeline.transform(X_test)

# Build a multi-layer neural network model
model = Sequential()
model.add(Dense(128, input_dim=X_train_poly.shape[1], activation='relu', kernel_regularizer=l
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(1, activation='relu'))

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Compile the model with an adjusted learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='mean_squared_error')

# Train the model
model.fit(X_train_poly, y_train, epochs=50, batch_size=32, validation_split=0.2, callbacks=[e

# Make predictions on the test set
predictions = model.predict(X_test_poly)

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
# xgboost_df = model_instantiation(model, x_train, x_test, y_train, y_test, predictions, 'Ten
# xgboost_df
```

```
Epoch 1/50
349/349 [==============================] - 3s 4ms/step - loss: 45521571840.0000 - val_loss: 4
5659099136.0000
Epoch 2/50
349/349 [==============================] - 1s 4ms/step - loss: 45513981952.0000 - val_loss: 4
5651877888.0000
Epoch 3/50
349/349 [==============================] - 1s 3ms/step - loss: 45483630592.0000 - val_loss: 4
5618933760.0000
Epoch 4/50
349/349 [==============================] - 1s 3ms/step - loss: 45420085248.0000 - val_loss: 4
5588664320.0000
Epoch 5/50
349/349 [==============================] - 1s 3ms/step - loss: 45316804608.0000 - val_loss: 4
5505753088.0000
Epoch 6/50
349/349 [==============================] - 1s 3ms/step - loss: 45167734784.0000 - val_loss: 4
5382037504.0000
Epoch 7/50
349/349 [==============================] - 1s 3ms/step - loss: 44966608896.0000 - val_loss: 4
5313585152.0000
Epoch 8/50
349/349 [==============================] - 1s 3ms/step - loss: 44703805440.0000 - val_loss: 4
5178540032.0000
Epoch 9/50
349/349 [==============================] - 1s 3ms/step - loss: 44380065792.0000 - val_loss: 4
5011771392.0000
Epoch 10/50
349/349 [==============================] - 1s 3ms/step - loss: 43981303808.0000 - val_loss: 4
4739723264.0000
Epoch 11/50
349/349 [==============================] - 1s 3ms/step - loss: 43510579200.0000 - val_loss: 4
4404719616.0000
Epoch 12/50
349/349 [==============================] - 1s 2ms/step - loss: 42954420224.0000 - val_loss: 4
4059979776.0000
Epoch 13/50
349/349 [==============================] - 1s 3ms/step - loss: 42316414976.0000 - val_loss: 4
3515187200.0000
Epoch 14/50
349/349 [==============================] - 1s 3ms/step - loss: 41571135488.0000 - val_loss: 4
3261763584.0000
Epoch 15/50
349/349 [==============================] - 1s 3ms/step - loss: 40685912064.0000 - val_loss: 4
2615050240.0000
Epoch 16/50
349/349 [==============================] - 1s 3ms/step - loss: 39722323968.0000 - val_loss: 4
1998520320.0000
Epoch 17/50
349/349 [==============================] - 1s 3ms/step - loss: 38568656896.0000 - val_loss: 4
1106513920.0000
Epoch 18/50
349/349 [==============================] - 1s 3ms/step - loss: 37363703808.0000 - val_loss: 4
0114827264.0000
Epoch 19/50
349/349 [==============================] - 1s 4ms/step - loss: 36029636608.0000 - val_loss: 3
9349116928.0000
Epoch 20/50
349/349 [==============================] - 1s 3ms/step - loss: 34568691712.0000 - val_loss: 3
8629601280.0000
Epoch 21/50
349/349 [==============================] - 1s 3ms/step - loss: 33029892096.0000 - val_loss: 3
8276018176.0000
Epoch 22/50
349/349 [==============================] - 1s 3ms/step - loss: 31390441472.0000 - val_loss: 3
6098195456.0000
```

```
Epoch 23/50
349/349 [==============================] - 1s 3ms/step - loss: 29657585664.0000 - val_loss: 3
5949268992.0000
Epoch 24/50
349/349 [==============================] - 1s 3ms/step - loss: 27854807040.0000 - val_loss: 3
4343340032.0000
Epoch 25/50
349/349 [==============================] - 1s 3ms/step - loss: 26037495808.0000 - val_loss: 3
2776638464.0000
Epoch 26/50
349/349 [==============================] - 1s 3ms/step - loss: 24125790208.0000 - val_loss: 3
1230275584.0000
Epoch 27/50
349/349 [==============================] - 1s 3ms/step - loss: 22231838720.0000 - val_loss: 3
0732843008.0000
Epoch 28/50
349/349 [==============================] - 1s 2ms/step - loss: 20321949696.0000 - val_loss: 2
8998916096.0000
Epoch 29/50
349/349 [==============================] - 1s 3ms/step - loss: 18474164224.0000 - val_loss: 2
7636432896.0000
Epoch 30/50
349/349 [==============================] - 1s 4ms/step - loss: 16617832448.0000 - val_loss: 2
5972879360.0000
Epoch 31/50
349/349 [==============================] - 1s 3ms/step - loss: 15004934144.0000 - val_loss: 2
5463404544.0000
Epoch 32/50
349/349 [==============================] - 1s 4ms/step - loss: 13281187840.0000 - val_loss: 2
4184819712.0000
Epoch 33/50
349/349 [==============================] - 1s 3ms/step - loss: 12014547968.0000 - val_loss: 2
1572216832.0000
Epoch 34/50
349/349 [==============================] - 1s 3ms/step - loss: 10627654656.0000 - val_loss: 2
2135050240.0000
Epoch 35/50
349/349 [==============================] - 1s 3ms/step - loss: 9489682432.0000 - val_loss: 20
834973696.0000
Epoch 36/50
349/349 [==============================] - 1s 4ms/step - loss: 8421891584.0000 - val_loss: 19
097423872.0000
Epoch 37/50
349/349 [==============================] - 1s 4ms/step - loss: 7558081024.0000 - val_loss: 19
530541056.0000
Epoch 38/50
349/349 [==============================] - 1s 3ms/step - loss: 6951711744.0000 - val_loss: 18
991861760.0000
Epoch 39/50
349/349 [==============================] - 1s 3ms/step - loss: 6637931008.0000 - val_loss: 16
955921408.0000
Epoch 40/50
349/349 [==============================] - 1s 3ms/step - loss: 6067017216.0000 - val_loss: 15
321756672.0000
Epoch 41/50
349/349 [==============================] - 1s 3ms/step - loss: 5854266880.0000 - val_loss: 14
848336896.0000
Epoch 42/50
349/349 [==============================] - 1s 3ms/step - loss: 5741652992.0000 - val_loss: 13
548585984.0000
Epoch 43/50
349/349 [==============================] - 1s 3ms/step - loss: 5499480064.0000 - val_loss: 13
736945664.0000
Epoch 44/50
349/349 [==============================] - 1s 3ms/step - loss: 5346961408.0000 - val_loss: 13
284270080.0000
```

```
Epoch 45/50
349/349 [==============================] - 1s 4ms/step - loss: 5179998720.0000 - val_loss: 12
758976512.0000
Epoch 46/50
349/349 [==============================] - 1s 3ms/step - loss: 5294595072.0000 - val_loss: 12
697150464.0000
Epoch 47/50
349/349 [==============================] - 1s 3ms/step - loss: 5125353472.0000 - val_loss: 11
694658560.0000
Epoch 48/50
349/349 [==============================] - 1s 3ms/step - loss: 5200043520.0000 - val_loss: 97
71138048.0000
Epoch 49/50
349/349 [==============================] - 1s 3ms/step - loss: 5213671936.0000 - val_loss: 94
03393024.0000
Epoch 50/50
349/349 [==============================] - 1s 3ms/step - loss: 5141705728.0000 - val_loss: 10
682987520.0000
109/109 [==============================] - 1s 1ms/step
Mean Squared Error: 10499794813.146599
R-squared: -0.1518082573703612
```

In [45]:
```
from sklearn.linear_model import Ridge, Lasso



lasso = Lasso(alpha=1.0)
lasso.fit(x_train, y_train)
y_pred_lasso = lasso.predict(x_test)


lasso_df = model_instantiation(lasso, x_train, x_test, y_train, y_test, y_pred_lasso, 'Lasso'
lasso_df
```

```
C:\Users\sumit\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:628: C
onvergenceWarning: Objective did not converge. You might want to increase the number of itera
tions, check the scale of the features or consider increasing regularisation. Duality gap: 4.
345e+12, tolerance: 1.269e+10
  model = cd_fast.enet_coordinate_descent(
```

Out[45]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
Lasso	0.585009	0.588948	3.747114e+09	61213.678646	46199.161283

In [46]:
```
ridge = Ridge(alpha=1.0)
ridge.fit(x_train, y_train)
y_pred_ridge = ridge.predict(x_test)


ridge_df = model_instantiation(ridge, x_train, x_test, y_train, y_test, y_pred_ridge, 'Ridge'
ridge_df
```

Out[46]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
Ridge	0.584982	0.588941	3.747182e+09	61214.231846	46202.274281

In [47]:
```
all_results = pd.concat([LR_df,DT_df,RF_df,RF_df_2,ada_df,xgboost_df,lasso_df,ridge_df])
all_results
```

Out[47]:

	Train_R2	Test_R2	Test_MSE	Test_RMSE	Test_MAE
Linear Regression	0.630962	0.624499	1.071205e-01	0.327293	0.251028
DTree_Regression	0.763267	0.717443	8.060598e-02	0.283912	0.208418
RF_Regression	0.850595	0.774959	6.419829e-02	0.253374	0.180186
RF2_Regression	0.830124	0.763076	6.758802e-02	0.259977	0.186851
ADA_Regression	0.599874	0.586475	1.179678e-01	0.343464	0.271343
XGB_Regression	0.923149	0.801669	5.657852e-02	0.237862	0.163638
Lasso	0.585009	0.588948	3.747114e+09	61213.678646	46199.161283
Ridge	0.584982	0.588941	3.747182e+09	61214.231846	46202.274281