importing libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

importing data set from california_houses

In [2]:
```
dataread.read_csv("C:\Users\User\Desktop\cere 7\machine learning\ML- Assignment 1\California_Houses.csv",sep=',')
data.head()
```

Out[2]:

	Median_House_Value	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	Households	Latitude	Longitude	Distance_to_coast	Distance_to_LA	Distance_to_SanDiego	Distance_to_SanJose	Distance_to_SanFra
0	452600.0	8.325	41.0	880	129	322	126	37.88	-122.23	92564.00000	539.60000	526.00000	674537.51791	21250.
1	358500.0	8.3014	21.0	7099	1106	2401	1138	37.86	-122.22	101257.73073	554279.85009	733236.88430	650409.90974	20880.
2	352100.0	7.2574	52	1467	190	496	177	37.85	-122.24	82913.06106	554610.717909	733525.692927	649817.299833	18811.
3	341300.0	5.6431	52	1274	235	558	219	37.85	-122.25	77568.08872	555184.266086	734095.290744	652871.138412	18031.
4	342200.0	3.8462	52	1627	280	565	259	37.85	-122.25	77568.08872	555184.266086	734095.290744	652871.138412	18031.

exploring presence of null

In [3]:
```
data.info()
data.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Median_House_Value  20640 non-null  float64
 1   Median_Income       20640 non-null  float64
 2   Median_Age          20640 non-null  int64
 3   Tot_Rooms           20640 non-null  int64
 4   Tot_Bedrooms        20640 non-null  int64
 5   Population          20640 non-null  int64
 6   Households          20640 non-null  int64
 7   Latitude            20640 non-null  float64
 8   Longitude           20640 non-null  float64
 9   Distance_to_coast   20640 non-null  float64
 10  Distance_to_LA      20640 non-null  float64
 11  Distance_to_SanDiego 20640 non-null  float64
 12  Distance_to_SanJose 20640 non-null  float64
 13  Distance_to_SanFrancisco 20640 non-null  float64
dtypes: float64(9), int64(5)
memory usage: 2.2 MB
```

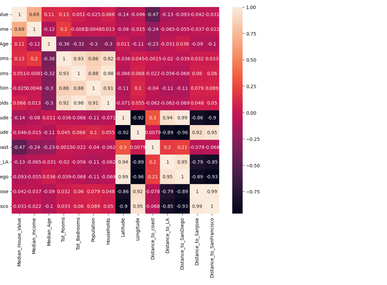
Out[3]:
```
Median_House_Value    0
Median_Income         0
Median_Age            0
Tot_Rooms             0
Tot_Bedrooms          0
Population            0
Households            0
Latitude              0
Longitude             0
Distance_to_coast     0
Distance_to_LA        0
Distance_to_SanDiego  0
Distance_to_SanJose   0
Distance_to_SanFrancisco  0
dtype: int64
```

explore data

by heat map know the relation between different features and median house value

In [4]:
```
plt.figure(figsize=(16, 8))
sns.heatmap(data.corr(),annot=True)
```

Out[4]:
```
<Axes: >
```



feature engineering

we will edit our features a little to be more related to the target

In [5]:
```
data["rooms per household"]=data["Tot_Rooms"]/data["Households"]
data["bedrooms ratio"]=data["Tot_Bedrooms"]/data["Tot_Rooms"]
```

features relations with the target is in first row

In [6]:
```
plt.figure(figsize=(16, 8))
sns.heatmap(data.corr(),annot=True)
```

Out[6]:
```
<Axes: >
```

dividing data into target and features

the target is Y features X

In [7]:
```
data=data.sample(frac=1).reset_index(drop=True)
print( data.iloc[0])
X=data.drop(['Median_House_Value'],axis=1).values
Y=data['Median_House_Value']
print(X.shape)
print(Y.shape)
print(Y.shape)
```

```
Median_House_Value    111300.000000
Median_Income              2.584200
Median_Age                 7.000000
Tot_Rooms                  2021.000000
Tot_Bedrooms               409.000000
Population                 1615.000000
Households                 406.000000
Latitude                   34.090000
Longitude                  -121.290000
Distance_to_coast          20099.857394
Distance_to_LA             525595.828311
Distance_to_SanDiego       899328.257189
Distance_to_SanJose        1057.966667
Distance_to_SanFrancisco   184284.566777
rooms per household        4.977833
bedrooms ratio             0.218299
Name: 0, dtype: float64
(20640, 13)
  1.0200000e+03  4.0000000e+02  2.0210000e+03  4.4100000e+02
  1.0100000e+03  4.0600000e+02  1.8920000e+03  1.2120000e+02
  2.0800570e+04  5.2816582e+05  8.9932925e+05  9.3781665e+04
  1.8410654e+05  4.9778325e+00  2.1620000e-01]
(20640, 13)
(20640,)
```

splitting data into training,test and validation set

In [8]:
```
X_train,X_Imp,Y_train,Y_Imp=train_test_split(X,Y,test_size=0.3,shuffle=False)
X_test,X_validation,Y_test,Y_validation=train_test_split(X_Imp,Y_Imp,test_size=0.5,shuffle=False)
```

features normalization

we will normalize train features using z-score normalization using sklearn standard scaler

normalization will make the code run faster

In [9]:
```
normalizer=StandardScaler()
X_train=normalizer.fit_transform(X_train)
X_validation=normalizer.transform(X_validation)
X_test = normalizer.transform(X_test)
```

linear regression model

we are using LinearRegression from scikit learn to predict our model parameters which will use closed form

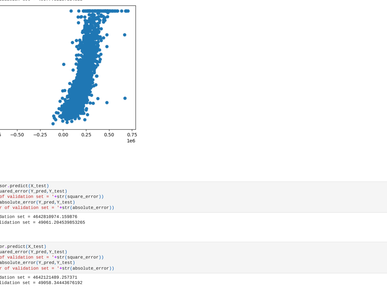
we are showing the values of linear regression coef. and intercept

after fitting the X_train data we are predicting the values of validation set and comparing it with the real values by square mean error=1/m*(Y-T)^2 and absolute error = 1/m*|Y-T|

In [10]:
```
linear_regressor=LinearRegression(copy_X=True)
linear_regressor.fit(X_train,Y_train)
Y_pred=linear_regressor.predict(X_validation)
print(Y_pred.shape)

square_error=mean_squared_error(Y_pred,Y_validation)
print("square error of validation set ",
w = linear_regressor.coef_
b = linear_regressor.intercept_
print(w,b)
print("square error of validation set ",+str(square_error))
absolute_error=mean_absolute_error(Y_pred,Y_validation)
print("absolute error of validation set ",+str(absolute_error))
```

```
[ 17908.62301545  16806.36570025   7624.32218091   2521.13609651
 -53339.40607994  45946.62141372  -8896.42227804  -58792.77364163
 -9696.35563785  -32942.42867227  -6601.06636106  39790.33566695
 -40190.72152263   6870.50943604  16423.16372403 207320.20586362
square error of validation set + 4956527897.37973
absolute error of validation set + 49877.0406894687
```



regro regression

it is like linear regression but we will add alpha(the values of w)^2 to the cost function to control their values

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm

'sag' uses stochastic gradient descent

In [11]:
```
ridge_regressor=Ridge(alpha=3.5,max_iter=15000,solver='sag')
ridge_regressor.fit(X_train,Y_train)
Y_pred=ridge_regressor.predict(X_validation)
square_error=mean_squared_error(Y_pred,Y_validation)
plt.scatter(Y_pred,Y_validation)
w = ridge_regressor.coef_
b = ridge_regressor.intercept_
print(w,b,ridge_regressor.n_iter_)
print("square error of validation set + "+str(square_error))
absolute_error=mean_absolute_error(Y_pred,Y_validation)
print("absolute error of validation set + "+str(absolute_error))
```

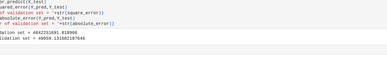
```
[ 17812.05390016  16717.06782931   7607.80036571   3664.69569715
 -53397.75874614  45792.57933695  -9726.56239863  -57114.69763874
 -9793.39661894  -32687.16788372  -6433.75712605  39194.48622406
 -40197.16192192   6872.50943604  16423.16372403 207320.20586362 [424]
square error of validation set + 4956527728.867328
absolute error of validation set + 49880.3361738113
```


lasso regression

Lasso is a modification of linear regression, where the model is penalized for the sum of absolute values of the weights (reg. is given by l1-norm)

In [12]:
```
lasso_regressor=Lasso(alpha=2,max_iter=15000)
lasso_regressor.fit(X_train,Y_train)
Y_pred=lasso_regressor.predict(X_validation)
square_error=mean_squared_error(Y_pred,Y_validation)
plt.scatter(Y_pred,Y_validation)
w = lasso_regressor.coef_
b = lasso_regressor.intercept_
print(w,b,lasso_regressor.n_iter_)
print("square error of validation set + "+str(square_error))
absolute_error=mean_absolute_error(Y_pred,Y_validation)
print("absolute error of validation set + "+str(absolute_error))
```

```
[ 17811.00715274  18712.05982371   7604.07928124   3586.91146154
 -53312.45310131  45877.09957174  -8811.49861608  -58899.29642983
 -9798.91658529  -32700.32908185  -6548.92986080  39226.81888891
 -40197.16192192   6872.50943604  16423.16372403 207320.20586362 1923
square error of validation set + 4956125384.388128
absolute error of validation set + 49877.6315784888
```



test set results

linear regressor

In [13]:
```
Y_pred=linear_regressor.predict(X_test)
square_error=mean_squared_error(Y_pred,Y_test)
print("square error of validation set + "+str(square_error))
absolute_error=mean_absolute_error(Y_pred,Y_test)
print("absolute error of validation set + "+str(absolute_error))

square error of validation set + 4943218974.233876
absolute error of validation set + 49661.29435905265
```

ridge regressor

In [14]:
```
Y_pred=ridge_regressor.predict(X_test)
square_error=mean_squared_error(Y_pred,Y_test)
print("square error of validation set + "+str(square_error))
absolute_error=mean_absolute_error(Y_pred,Y_test)
print("absolute error of validation set + "+str(absolute_error))

square error of validation set + 4943124349.257371
absolute error of validation set + 49658.54487678282
```

lasso regressor

In [15]:
```
Y_pred=lasso_regressor.predict(X_test)
square_error=mean_squared_error(Y_pred,Y_test)
print("square error of validation set + "+str(square_error))
absolute_error=mean_absolute_error(Y_pred,Y_test)
print("absolute error of validation set + "+str(absolute_error))

square error of validation set + 4942751681.818966
absolute error of validation set + 49659.13188218766
```

In [ ]: