Rachid LAMJOUN
PG AI Program

California Housing Price Prediction

(AI Data Science with Python Project: Project1 California Housing Price Prediction)

Writeup

We have the subject to build a model of housing prices to predict median house values in California. The model should be able to predict this values for any district. A single dataset in our possession which will be used for training and testing.

First, we will answer several questions that we will serve as a guide to ultimately build our prediction model. We build several model types based on several algorithms. These are regression models, and to appreciate the efficiency of the models we plot the predictions for the training and test data for two models, one efficient and another less so.

For a better readability of our presentation, we preferred to give the answers to the questions at the end. The answers will be in the DETAILS paragraph and each question will have the link to the answer.

For efficiency and ease of testing we have imagined a set of functions which are behind three main functions called once the data is loaded:
get_data_for_fit() which will **prepare** training and test data,
traitement() which will **train** the models and give the scores and metrics, and finally
show_Data() which will **show** us three plots

- The first is interested in Test data and their predictions for LinearRegression model
- The second aims to understand the behavior of the predictions of the Test data by LinearRegressor, by comparing them to the predictions of the powerful XGBRRegressor model, of the Train data
- The third is interested in the behavior of Test data for the model XGBRRegressor.

The Build of the algorithms is easily done after importing the necessary modules. each trained model is put in the models list, the first of which is LinearRegression and the last is

XGBRRegressor. It will be noted that the most efficient is the latter, on the other hand the first one is the least efficient. The Plots that we will deal with later will concern these two algorithms.

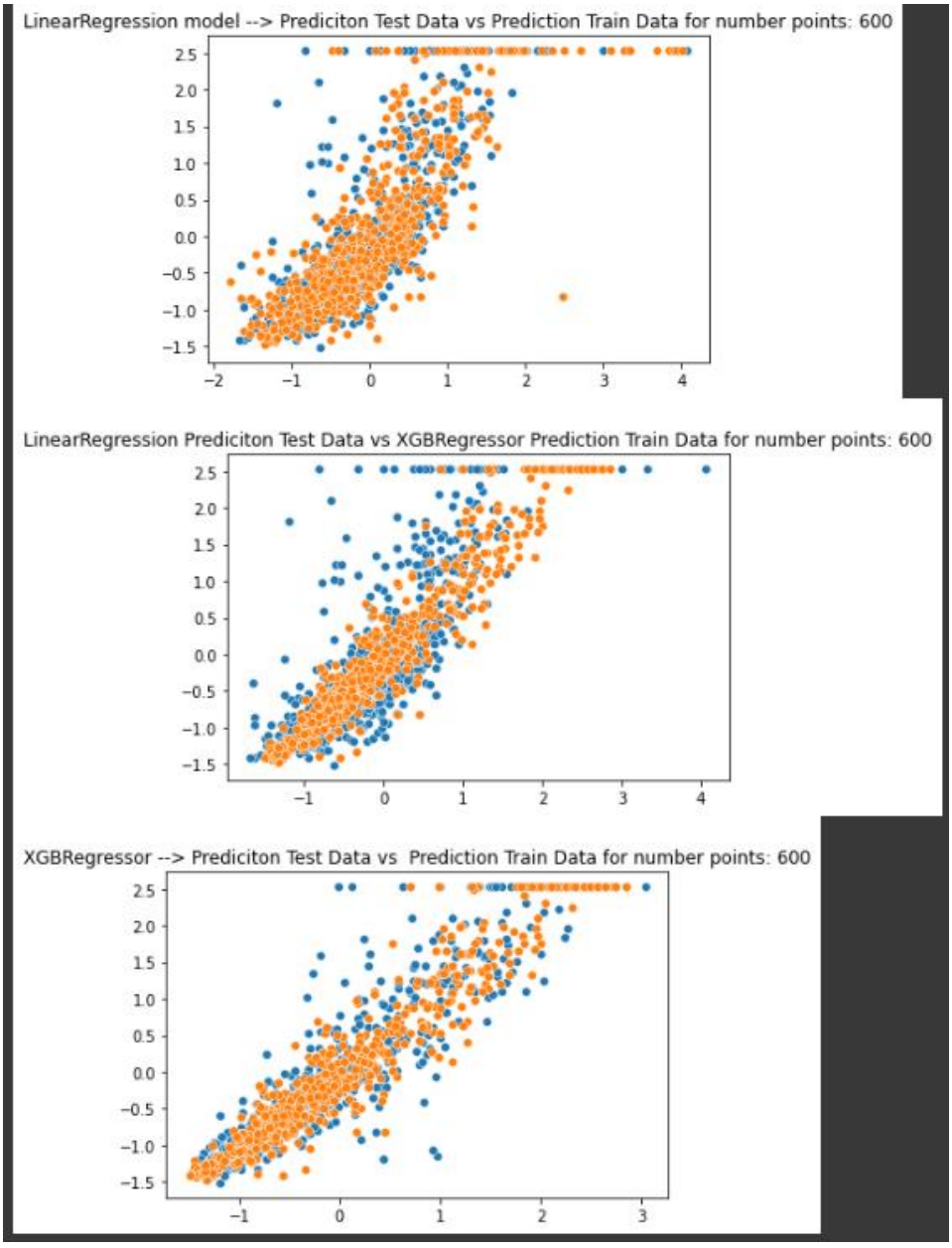
```
# dict of algos
dico_algos={1:LinearRegression,2:DecisionTreeRegressor,
            3:Ridge,4:GradientBoostingRegressor,
            5:RandomForestRegressor,6:XGBRegressor}
#
# dict of params algos
dico_params={1:{},2:{},3:{'alpha':9.0},4:{'n_estimators':2000, 'max_depth':5, 'learning_rate':0.01, '
min_samples_split':5},
                        5:{'n_estimators':400,'max_depth':15,'n_jobs':5},6:{'objective':'reg:squarederror'
, 'eval_metric':'mae', 'n_estimators':1500, 'seed':123,'verbose':2}}
```

```
df,X_train,X_test,y_train,y_test,index_column_median_income
            =get_data_for_fit()
#
models_list =traitement(X_train,X_test,y_train,y_test,dico_algos)
```

```
------>  <class 'sklearn.linear_model._base.LinearRegression'>

Accuracy: 63.79 %
Mean Absolute Error: 0.4292
Root Mean Squared Error: 0.3528
========================================================
------>  <class 'sklearn.tree._classes.DecisionTreeRegressor'>

Accuracy: 64.51 %
Mean Absolute Error: 0.3715
Root Mean Squared Error: 0.3458
========================================================
------>  <class 'sklearn.linear_model._ridge.Ridge'>

Accuracy: 63.82 %
Mean Absolute Error: 0.4295
Root Mean Squared Error: 0.3525
========================================================
------>  <class 'sklearn.ensemble._gb.GradientBoostingRegressor'>

Accuracy: 82.99 %
Mean Absolute Error: 0.2727
Root Mean Squared Error: 0.1657
========================================================
------>  <class 'sklearn.ensemble._forest.RandomForestRegressor'>

Accuracy: 82.04 %
Mean Absolute Error: 0.2768
Root Mean Squared Error: 0.175
========================================================
------>  <class 'xgboost.sklearn.XGBRegressor'>

Accuracy: 83.52 %
Mean Absolute Error: 0.2694
Root Mean Squared Error: 0.1606
========================================================
```

```
len(models_list)
```
```
6
```

```
model_LReg=models_list[0]
model_XGBR=models_list[5]
y_pred_LReg=model_LReg.predict(X_test)
y_pred_XGBR=model_XGBR.predict(X_test)
```

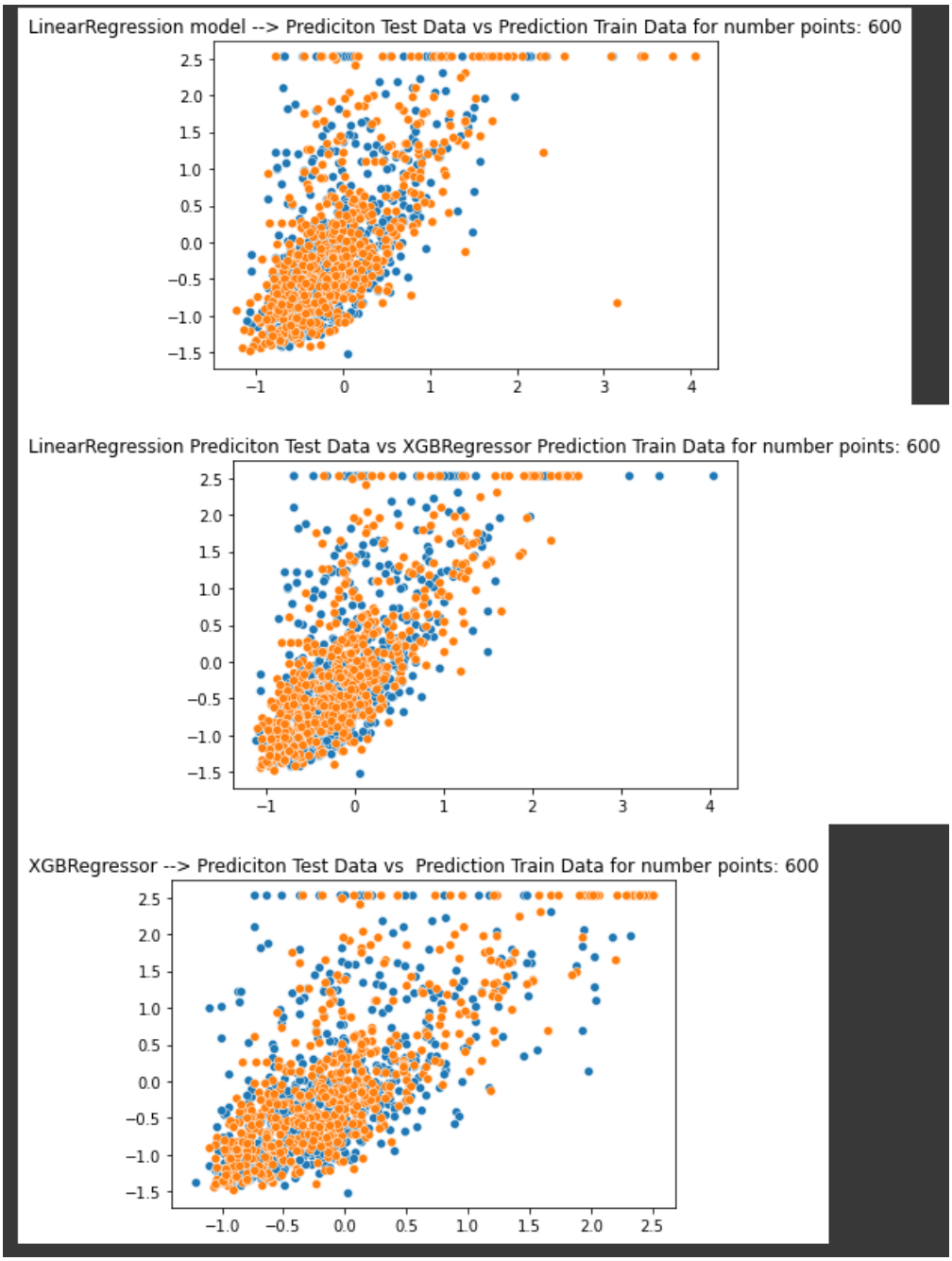


In the case where we are reduced to the single feature **median_income** we note below that the performance of the models drops drastically and in particular for XGBRRegressor.

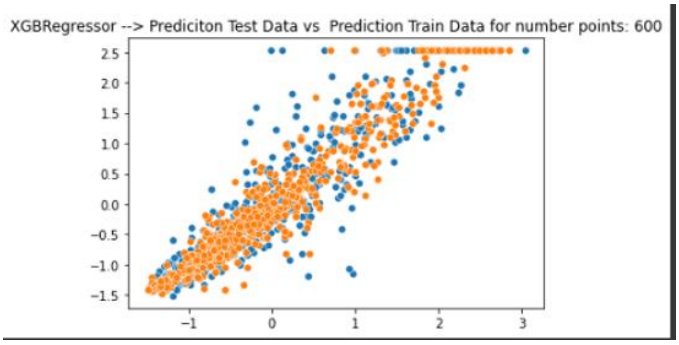
```
df,X_train,X_test,y_train,y_test,index_column_median_income=
        get_data_for_fit()
#
X_train=X_train[:,index_column_median_income:index_column_median_income+1]
X_test=X_test[:,index_column_median_income:index_column_median_income+1]
#
models_list =traitement(X_train,X_test,y_train,y_test,dico_algos)
```

```
------>  <class 'sklearn.linear_model._base.LinearRegression'>

Accuracy: 44.67 %
Mean Absolute Error: 0.5491
Root Mean Squared Error: 0.5391
==========================================================
------>  <class 'sklearn.tree._classes.DecisionTreeRegressor'>

Accuracy: 8.99 %
Mean Absolute Error: 0.692
Root Mean Squared Error: 0.8867
==========================================================
------>  <class 'sklearn.linear_model._ridge.Ridge'>

Accuracy: 44.67 %
Mean Absolute Error: 0.5491
Root Mean Squared Error: 0.5391
==========================================================
------>  <class 'sklearn.ensemble._gb.GradientBoostingRegressor'>

Accuracy: 45.19 %
Mean Absolute Error: 0.5434
Root Mean Squared Error: 0.534
==========================================================
------>  <class 'sklearn.ensemble._forest.RandomForestRegressor'>

Accuracy: 41.13 %
Mean Absolute Error: 0.5627
Root Mean Squared Error: 0.5736
==========================================================
------>  <class 'xgboost.sklearn.XGBRegressor'>

Accuracy: 43.48 %
Mean Absolute Error: 0.5515
Root Mean Squared Error: 0.5507
==========================================================
```

```
nbre_points_to_plot=600
show_Data(models_list,nbre_points_to_plot)
```



It is clearly seen that the points do not follow a linear trend even for XGBRegressor which has a worse trend than that of LinearRegression. The comparison of Plots is obvious when we take all the features and when we only base it on median_income, in the first case (all features) for XGBRegressor the points have a clearly linear trend.



QUESTIONS:

1. **Load the data** :
   1.1. Read the "**housing.csv**" file from the folder into the program.
   1.2. Print first few rows of this data.
   1.3. Extract input (X) and output (Y) data from the dataset.

2. **Handle missing values** :

   - Fill the missing values with the mean of the respective column.

3. **Encode categorical data** :

   - Convert categorical column in the dataset to numerical data.

4. **Split the dataset** :

   - Split the data into 80% training dataset and 20% test dataset.

5. **Standardize data** :

   - Standardize training and test datasets.

6. **Perform Linear Regression** :

   6.1. Perform Linear Regression on training data.

   6.2. Predict output for test dataset using the fitted model.

   6.3. Print root mean squared error (RMSE) from Linear Regression.

   [HINT: Import **mean_squared_error** from **sklearn.metrics**]

7. **Bonus exercise: Perform Linear Regression with one independent variable** :

   7.1. Extract just the median_income column from the independent variables (from **X_train** and **X_test**).
   7.2. Perform Linear Regression to predict housing values based on **median_income**.
   7.3. Predict output for test dataset using the fitted model.
   7.4. Plot the fitted model for training data as well as for test data to check if the fitted model satisfies the test data.

DETAILS

Load the data :

1.1. Read the “housing.csv” file from the folder into the program.

```
import pandas as pd

# Loading train_data, test_data and test_data_hidden
df_init = pd.read_excel('/content/California_Housing_Data.xlsx')
#
df=df_init.copy(deep=True)
#
```

1.2. Print first few rows of this data

```
# Top 5 records
df.head()
```

	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

1.3. Extract input (X) and output (Y) data from the dataset.

```
# List of columns
df.columns
```

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

```
# define X, Y
list_col_Y=['median_house_value']
```

```
list_cols_X= delete_sub_list(list(df.columns),list_col_Y)
#
X,Y=get_inputX_outputY(df,list_cols_X,list_col_Y)
df.shape,X.shape,Y.shape
```

```
((20640, 10), (20640, 9), (20640, 1))
```

```
X.head()
```

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

```
Y.head()
```

	median_house_value
0	452600
1	358500
2	352100
3	341300
4	342200

2. Handle missing values :

- **Fill the missing values with the mean of the respective column.**

```
# dataset Information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   ocean_proximity     20640 non-null  object 
 9   median_house_value  20640 non-null  int64  
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
```

```
# checking
df.isnull().sum()
```

```
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
```

```
# missing data
df=replace_missing_values(df,'total_bedrooms','mean')

# checking
df.count()
```

```
longitude             20640
latitude              20640
housing_median_age    20640
total_rooms           20640
total_bedrooms        20640
population            20640
households            20640
median_income         20640
ocean_proximity       20640
median_house_value    20640
dtype: int64
```

3. Encode categorical data :

- **Convert categorical column in the dataset to numerical data.**

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
#
```

```
if len(dict(X.dtypes.value_counts()).keys()) > 1:
transformer = make_column_transformer(
(OneHotEncoder(), ['ocean_proximity']),
remainder='passthrough')

transformed_X = transformer.fit_transform(X)

X = pd.DataFrame(transformed_X,
columns=transformer.get_feature_names_out()
)
X.shape
```

```
(20640, 13)
```

```
X.head()
```

	onehotencoder__ocean_proximity_<1H OCEAN	onehotencoder__ocean_proximity_INLAND	onehotencoder__ocean_proximity_ISLAND	onehotencoder__ocean_proximity_NEAR BAY	onehotencoder__ocean_proximity_NEAR OCEAN	remainder__longitude	remainder__latitude
0	0.0	0.0	0.0	1.0	0.0	-122.23	37.88
1	0.0	0.0	0.0	1.0	0.0	-122.22	37.86
2	0.0	0.0	0.0	1.0	0.0	-122.24	37.85
3	0.0	0.0	0.0	1.0	0.0	-122.25	37.85
4	0.0	0.0	0.0	1.0	0.0	-122.25	37.85

4. Split the dataset :

- **Split the data into 80% training dataset and 20% test dataset.**

```
from sklearn.model_selection import train_test_split
#
in_test_size=20/100
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=in_test_size,random_state=0)
#
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
((16512, 13), (4128, 13), (16512, 1), (4128, 1))
```

5. Standardize data :

- **Standardize training and test datasets.**

```
from sklearn.preprocessing import StandardScaler
X_train,X_test,y_train,y_test=standardize_data(X_train,X_test,y_train,y_test)
#
X_train[:3]
```

```
array([[-0.89150581,  1.46934754, -0.01348032, -0.35239609, -0.38649771,
         1.00389865, -0.8400624 , -1.79507596, -0.97773624, -1.09836953,
        -1.04760128, -1.1356496 ,  0.19001247],
       [-0.89150581, -0.68057418, -0.01348032,  2.83771591, -0.38649771,
        -1.43477229,  0.98536392,  1.85553889, -0.11850059, -0.10859305,
         0.05210918, -0.13688171,  0.26931072],
       [ 1.12169768, -0.68057418, -0.01348032, -0.35239609, -0.38649771,
         0.77948108, -0.8400624 , -0.20785212, -0.42167953, -0.36317498,
        -0.35295521, -0.34343319,  0.02989505]])
```

6. Perform Linear Regression :

6.1. Perform Linear Regression on training data.

```
from sklearn.linear_model import LinearRegression
model_LReg=LinearRegression()
model_LReg.fit(X_train,y_train)
print('model.score(X_test, y_test)=',np.round(model_LReg.score(X_test,
y_test),2))
```

```
model.score(X_test, y_test)= 0.638
```

6.2. Predict output for test dataset using the fitted model.

```
y_pred=model_LReg.predict(X_test)
y_pred
```

```
array([[ 0.09145029],
       [ 0.69256643],
       [-0.24882139],
       ...,
       [-1.03029024],
       [ 0.40731097],
       [ 0.0513233 ]])
```

6.3. Print root mean squared error (RMSE) from Linear Regression.

```
print("=========================================================")
y_pred=model_LReg.predict(X_test)
print('Accuracy:',np.round(model_LReg.score(X_test, y_test)*100,2),'%')
print('Mean Absolute Error:', np.round(metrics.mean_absolute_error(y_test, y_pred),4))
print('Root Mean Squared Error:', np.round(metrics.mean_squared_error(y_test, y_pred),4))
print("=========================================================")
```

```
======================================================
Accuracy: 63.8 %
Mean Absolute Error: 0.4286
Root Mean Squared Error: 0.3527
======================================================
```

7. Bonus exercise: Perform Linear Regression with one independent variable :

7.1. Extract just the median_income column from the independent variables (from X_train and X_test).

```
type(X),X.shape
```

```
(pandas.core.frame.DataFrame, (20640, 13))
```

```
X.columns
```

```
Index(['onehotencoder__ocean_proximity_<1H OCEAN',
       'onehotencoder__ocean_proximity_INLAND',
       'onehotencoder__ocean_proximity_ISLAND',
       'onehotencoder__ocean_proximity_NEAR BAY',
       'onehotencoder__ocean_proximity_NEAR OCEAN', 'remainder__longitude',
       'remainder__latitude', 'remainder__housing_median_age',
       'remainder__total_rooms', 'remainder__total_bedrooms',
       'remainder__population', 'remainder__households',
       'remainder__median_income'],
      dtype='object')
```

```
idx=list(X.columns).index('remainder__median_income')
idx
```

```
12
```

```
X_train=X_train[:,idx:idx+1]
X_test=X_test[:,idx:idx+1]
X_train
```

```
array([], shape=(16512, 0), dtype=float64)
```

7.2. Perform Linear Regression to predict housing values based on median_income.

```
from sklearn.linear_model import LinearRegression
model_LReg=LinearRegression()
model_LReg.fit(X_train,y_train)
print('model.score(X_test, y_test)=',np.round(model_LReg.score(X_test, y_test),4))
```

```
model.score(X_test, y_test)= 0.4467
```

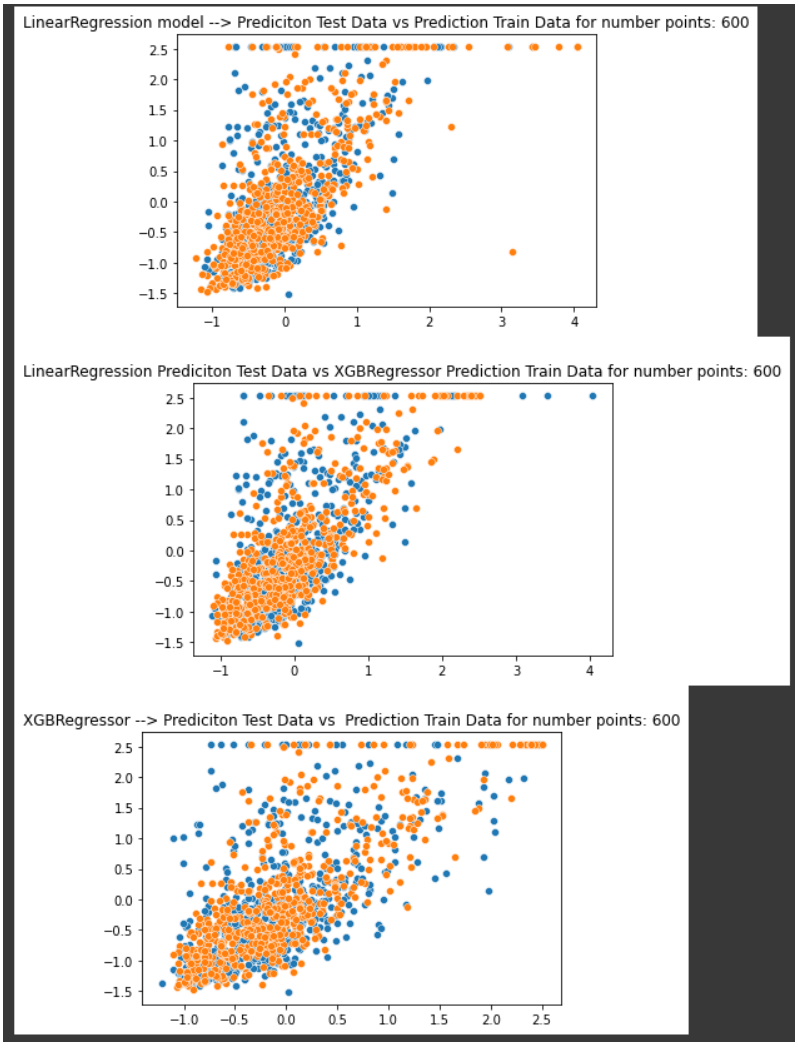
7.3. Predict output for test dataset using the fitted model.

```
y_pred=model_LReg.predict(X_test)
y_pred
```

```
array([[ 0.10009624],
       [ 0.69151432],
       [ 0.17163465],
       ...,
       [-0.24473414],
       [ 0.82376409],
       [-0.19753841]])
```

7.4. Plot the fitted model for training data as well as for test data to check if the fitted model satisfies the test data.

```
nbre_points_to_plot=600
show_Data(models_list,nbre_points_to_plot)
```



LISTE OF FUNCTIONS

```
# Re-init Data
def reinit_input_df():
 l_df=df_init.copy(deep=True)
 return l_df
```

```
# replace missing Data
def replace_missing_values(i_df,i_column,i_method):
 l_df=i_df
 #
 if i_method=='median':
  l_df[i_column].replace([np.nan], l_df[i_column].median(), inplace=True)
 #
 if i_method=='mean':
  l_df[i_column].replace([np.nan], l_df[i_column].mean(), inplace=True)

 return l_df
```

```
# Define X and Y by choosing columns
def get_inputX_outputY(i_df, i_list_column_X, i_list_column_Y):
 return i_df[i_list_column_X], i_df[i_list_column_Y]
```

```
# delete sub_list from list
def delete_sub_list(i_list,i_sub_list):
 l_list=i_list
 for ele in l_list:
  if ele in i_sub_list:
   l_list.remove(ele)
 return l_list
```

```
# list of columns with count of its values by type of columns
def get_value_counts(i_df,i_type):
 lst_nbre_val_col=[]
 cmpt=0
 for col in i_df.select_dtypes(i_type).columns:
  cmpt=i_df[col].value_counts().count()
  lst_nbre_val_col.append((col,cmpt))
 #
 return lst_nbre_val_col
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
```

```
# Coding X
def get_coded_X(i_X):
 l_X=i_X
 if len(dict(i_X.dtypes.value_counts()).keys()) > 1:
  l_transformer = make_column_transformer(
                (OneHotEncoder(), ['ocean_proximity']),
                 remainder='passthrough')

  transformed_X = l_transformer.fit_transform(l_X)

  l_X = pd.DataFrame(transformed_X,
                     columns=l_transformer.get_feature_names_out()
                          )
 return l_X
```

```
from sklearn.preprocessing import StandardScaler

# Standardize data
def standardize_data(i_X_train,i_X_test,i_y_train,i_y_test):
 sc = StandardScaler()
 l_X_train=sc.fit_transform(i_X_train)
 l_X_test=sc.transform(i_X_test)
 l_y_train=sc.fit_transform(i_y_train)
 l_y_test=sc.transform(i_y_test)
 return l_X_train,l_X_test,l_y_train,l_y_test
```

get_data_for_fit()

```
from sklearn.model_selection import train_test_split

# return standardize data for fit()
def get_data_for_fit(i_Prcent_split=20/100, i_column='median_income'):
 #
 # reinit data
 l_df=reinit_input_df()

 # missing data
 l_df=replace_missing_values(l_df,'total_bedrooms','mean')

 # define X, Y
 l_list_col_Y=['median_house_value']
 l_list_cols_X= delete_sub_list(list(l_df.columns),l_list_col_Y)
 #
 l_X,l_Y=get_inputX_outputY(l_df,l_list_cols_X,l_list_col_Y)

 # coding non num value
```

```
 l_X=get_coded_X(l_X)

 # Split data
 l_X_train,l_X_test,l_y_train,l_y_test = train_test_split(l_X,l_Y,test_size=i_Prcent_split,random
_state=0)

 # standardize data
 l_X_train,l_X_test,l_y_train,l_y_test = standardize_data(l_X_train,l_X_test,l_y_train,l_y_test)
 l_new_name_col= 'remainder__'+ i_column
 return l_df,l_X_train,l_X_test,l_y_train,l_y_test,list(l_X.columns).index(l_new_name_col)
```

```
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# for fit and score different algos
def def_pipeline(i_list_numerical_columns,i_list_categorical_columns,i_idx_of_algo,i_scaling=Fal
se,i_verbose=False):
 #
 l_numerical_features = i_list_numerical_columns
 l_categorical_features = i_list_categorical_columns
 #
 l_numerical_pipeline = make_pipeline(SimpleImputer(strategy='mean'), StandardScaler())
 l_categorical_pipeline = make_pipeline(SimpleImputer(strategy='most_frequent'), OneHotEnco
der())
 preprocessor = make_column_transformer((l_numerical_pipeline, l_numerical_features),
                                        (l_categorical_pipeline, l_categorical_features))

 i_str_algo=str(dico_algos[i_idx_of_algo])
 l_funct=dico_algos[i_idx_of_algo]
 #
 l_param={}
 l_param=dico_params[i_idx_of_algo]
 #
 l_pipe=Pipeline(steps=[('------ ' + i_str_algo, l_funct().set_params(**l_param))])
 print('------> ', i_str_algo)
 #
 j=i_idx_of_algo
 if i_scaling:
   l_pipe.steps.insert(0,['-------PreProcessing',preprocessor])
 #
 l_pipe.verbose=False
 if i_verbose:
   l_pipe.verbose=True
 return l_pipe
```

traitement()

```
# fit() and score the list of algos
def traitement(i_X_train,i_X_test,i_y_train,i_y_test,
           i_dico_algos,i_scaling=False,i_verbose=False):
 #
 l_models_list=[]
 #
 for i in range(0,len(i_dico_algos)):
  # if i!=3:
  l_pipe=def_pipeline([],[],i+1,i_scaling,i_verbose)
  #
  print()
  #
  l_pipe.fit(i_X_train, np.ravel(i_y_train))
  l_models_list.append(l_pipe)
  #
  l_y_pred=l_pipe.predict(i_X_test)
  # print('pipe.score(X_test, y_test)=',np.round(l_pipe.score(i_X_test,np.ravel(i_y_test)),2))
  print('Accuracy:',np.round(l_pipe.score(i_X_test, i_y_test)*100,2),'%')
  print('Mean Absolute Error:', np.round(metrics.mean_absolute_error(i_y_test, l_y_pred),4))
  print('Root Mean Squared Error:', np.round(metrics.mean_squared_error(i_y_test, l_y_pred),4))
  print("============================================================")
 return l_models_list
```

show_Data()

```
# Plot Data, appreciate pred Train and test data

import seaborn as sns
import matplotlib.pyplot as plt

def show_Data(i_models_list,i_nbre_points_to_plot=500):
 model_LReg=i_models_list[0]
 model_XGBR=i_models_list[5]
 y_pred_LReg=model_LReg.predict(X_test)
 y_pred_XGBR=model_XGBR.predict(X_test)

 plt.title('LinearRegression model --
> Prediciton Test Data vs Prediction Train Data for number points: ' + str(i_nbre_points_to_plot)
)
```

```
 sns.scatterplot(x=y_pred_LReg[:i_nbre_points_to_plot], y=np.ravel(y_test)[:i_nbre_points_to_p
lot]);
 sns.scatterplot(x=model_LReg.predict(X_train)[:i_nbre_points_to_plot], y=np.ravel(y_train)[:i_
nbre_points_to_plot]);

 plt.figure()
 plt.title('\nLinearRegression Prediciton Test Data vs XGBRegressor Prediction Train Data for
number points: ' + str(nbre_points_to_plot))
 sns.scatterplot(x=y_pred_LReg[:i_nbre_points_to_plot], y=np.ravel(y_test)[:i_nbre_points_to_p
lot]);
 sns.scatterplot(x=model_XGBR.predict(X_train)[:i_nbre_points_to_plot], y=np.ravel(y_train)[:i
_nbre_points_to_plot]);

 plt.figure()
 plt.title('\nXGBRegressor --
> Prediciton Test Data vs  Prediction Train Data for number points: ' + str(nbre_points_to_plot))
 sns.scatterplot(x=y_pred_XGBR[:i_nbre_points_to_plot], y=np.ravel(y_test)[:i_nbre_points_to_
plot]);
 sns.scatterplot(x=model_XGBR.predict(X_train)[:i_nbre_points_to_plot], y=np.ravel(y_train)[:i
_nbre_points_to_plot]);
```