In [2]:**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** sklearn.model\_selection

**from** sklearn.model\_selection **import** train\_test\_split

In [38]:**!**pip install xgboost

Collecting xgboost

Downloading xgboost-1.7.1-py3-none-win\_amd64.whl (89.1 MB)

Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.7.3) Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.21.5) Installing collected packages: xgboost

Successfully installed xgboost-1.7.1

In [39]:**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.ensemble **import** StackingRegressor

**from** sklearn.ensemble **import** RandomForestRegressor

**from** sklearn.ensemble **import** AdaBoostRegressor

**from** sklearn.ensemble **import** GradientBoostingRegressor

**from** sklearn.tree **import** DecisionTreeRegressor

*#neural network model*

**from** sklearn.neural\_network **import** MLPRegressor

**from** xgboost **import** XGBRegressor

*#evaluating parameters*

**from** sklearn.metrics **import** mean\_absolute\_error,r2\_score,mean\_squared\_error

In [4]:*#reading data*

data**=**pd**.**read\_csv("DS - Assignment Part 1 data set csv.csv")

In [33]:*# Understanding the nature of given data*

In [11]:data**.**columns

Out[11]:

Index(['Transaction date', 'House Age',

'Distance from nearest Metro station (km)',

'Number of convenience stores', 'latitude', 'longitude', 'Number of bedrooms', 'House size (sqft)', 'House price of unit area'], dtype='object')

In [4]:data**.**head()

Out[4]:

**Transaction date**

**House Age**

**Distance from nearest Metro station (km)**

**Number of convenience**

**stores latitude longitude Number of**

**bedrooms**

**House size (sqft)**

**House price of unit area**

**0** 2012.917 32.0 84.87882 10 24.98298 121.54024 1 575 37.9 **1** 2012.917 19.5 306.59470 9 24.98034 121.53951 2 1240 42.2 **2** 2013.583 13.3 561.98450 5 24.98746 121.54391 3 1060 47.3 **3** 2013.500 13.3 561.98450 5 24.98746 121.54391 2 875 54.8 **4** 2012.833 5.0 390.56840 5 24.97937 121.54245 1 491 43.1

In [5]:data**.**describe()

Out[5]:

**Transaction**

**date House Age Distance from nearest Metro station (km)**

**Number of convenience**

**stores latitude longitude Number of bedrooms**

**House size (sqft)**

**House price of unit area**

**count** 414.000000 414.000000 414.000000 414.000000 414.000000 414.000000 414.000000 414.000000 414.000000 **mean** 2013.148971 17.712560 1083.885689 4.094203 24.969030 121.533361 1.987923 931.475845 37.980193 **std** 0.281967 11.392485 1262.109595 2.945562 0.012410 0.015347 0.818875 348.910269 13.606488 **min** 2012.667000 0.000000 23.382840 0.000000 24.932070 121.473530 1.000000 402.000000 7.600000 **25%** 2012.917000 9.025000 289.324800 1.000000 24.963000 121.528085 1.000000 548.000000 27.700000 **50%** 2013.167000 16.100000 492.231300 4.000000 24.971100 121.538630 2.000000 975.000000 38.450000 **75%** 2013.417000 28.150000 1454.279000 6.000000 24.977455 121.543305 3.000000 1234.750000 46.600000 **max** 2013.583000 43.800000 6488.021000 10.000000 25.014590 121.566270 3.000000 1500.000000 117.500000

In [8]:data**.**info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 414 entries, 0 to 413

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Transaction date 414 non-null float64

1 House Age 414 non-null float64

2 Distance from nearest Metro station (km) 414 non-null float64

3 Number of convenience stores 414 non-null int64

4 latitude 414 non-null float64

5 longitude 414 non-null float64

6 Number of bedrooms 414 non-null int64

7 House size (sqft) 414 non-null int64

8 House price of unit area 414 non-null float64

dtypes: float64(6), int64(3)

memory usage: 29.2 KB

In [9]:data**.**isnull()

Out[9]:

**Transaction date**

**House Age**

**Distance from nearest Metro station (km)**

**Number of convenience**

**stores latitude longitude Number of**

**bedrooms**

**House size (sqft)**

**House price of unit area**

**0** False False False False False False False False False **1** False False False False False False False False False **2** False False False False False False False False False **3** False False False False False False False False False **4** False False False False False False False False False **...** ... ... ... ... ... ... ... ... ...

**409** False False False False False False False False False **410** False False False False False False False False False **411** False False False False False False False False False **412** False False False False False False False False False **413** False False False False False False False False False

414 rows × 9 columns

In [32]:data**.**nunique()

Out[32]:

Transaction date 12 House Age 236 Distance from nearest Metro station (km) 259 Number of convenience stores 11 latitude 234 longitude 232 Number of bedrooms 3 House size (sqft) 328 House price of unit area 270 dtype: int64

In [35]:*# Visualizing relationship between dependent and Independent features*

In [18]:*#Relation between Price and Transaction date*

plt**.**figure(figsize**=**(10,7))

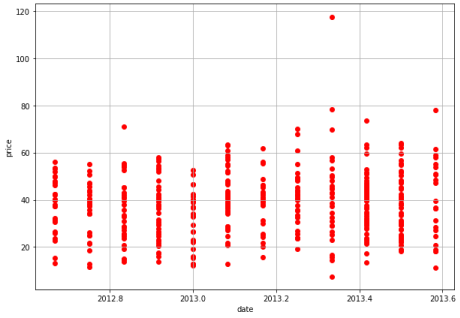
plt**.**scatter(data['Transaction date'],data['House price of unit area'],color**=**"red")

plt**.**xlabel("date",fontsize**=**10)

plt**.**ylabel("price",fontsize**=**10)

plt**.**grid(**True**)

plt**.**show()



In [14]:*#Relation between Price and Distance from nearest Metro station (km)*

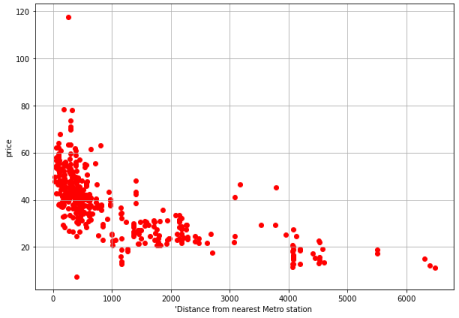
plt**.**figure(figsize**=**(10,7))

plt**.**scatter(data['Distance from nearest Metro station (km)'],data['House price of unit area'],color**=**"red") plt**.**xlabel("'Distance from nearest Metro station ",fontsize**=**10)

plt**.**ylabel("price",fontsize**=**10)

plt**.**grid(**True**)

plt**.**show()



In [36]:*#Relation between Price and house area*

plt**.**figure(figsize**=**(10,7))

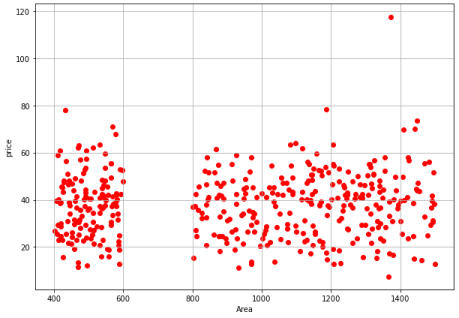
plt**.**scatter(data['House size (sqft)'],data['House price of unit area'],color**=**"red")

plt**.**xlabel("Area",fontsize**=**10)

plt**.**ylabel("price",fontsize**=**10)

plt**.**grid(**True**)

plt**.**show()



In [37]:*#Relation between Price and latitude*

plt**.**figure(figsize**=**(10,7))

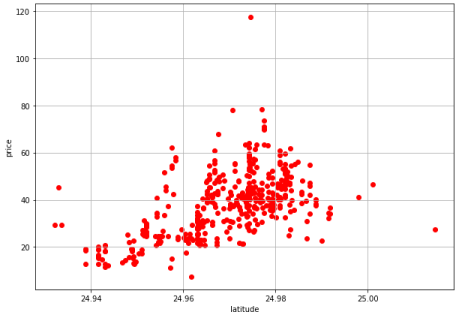
plt**.**scatter(data['latitude'],data['House price of unit area'],color**=**"red")

plt**.**xlabel("latitude",fontsize**=**10)

plt**.**ylabel("price",fontsize**=**10)

plt**.**grid(**True**)

plt**.**show()



In [38]:*#Relation between Price and longitude*

plt**.**figure(figsize**=**(10,7))

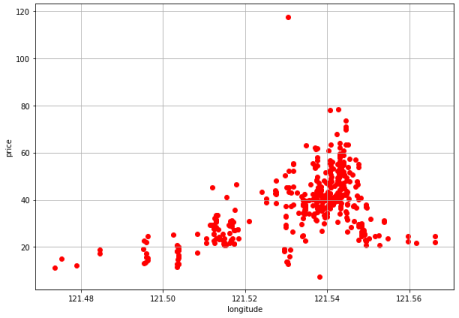
plt**.**scatter(data['longitude'],data['House price of unit area'],color**=**"red")

plt**.**xlabel("longitude",fontsize**=**10)

plt**.**ylabel("price",fontsize**=**10)

plt**.**grid(**True**)

plt**.**show()



In [39]:*#Relation between Price and bedrooms*

plt**.**figure(figsize**=**(10,7))

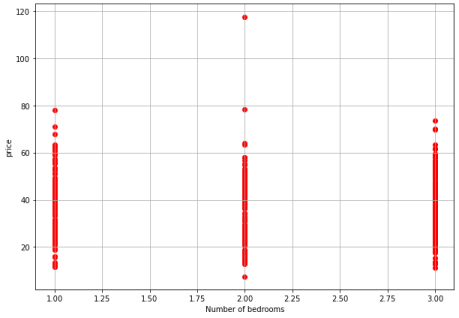
plt**.**scatter(data['Number of bedrooms'],data['House price of unit area'],color**=**"red")

plt**.**xlabel("Number of bedrooms",fontsize**=**10)

plt**.**ylabel("price",fontsize**=**10)

plt**.**grid(**True**)

plt**.**show()



In [5]:x **=** data**.**drop('House price of unit area',axis **=**1)

y**=**data['House price of unit area']

In [6]:xtrain,xtest,ytrain,ytest **=** train\_test\_split(x,y,test\_size**=**.2,random\_state**=**42)

In [52]:*# LinearRegression*

L**=** LinearRegression()

L**.**fit(xtrain,ytrain)

Out[52]:

LinearRegression()

In [53]:predtest**=**L**.**predict(xtest)

Lscore**=**L**.**score(xtrain,ytrain)

print(Lscore)

rmse\_L**=**mean\_squared\_error(ytest,predtest,squared**=False**)

print(rmse\_L)

0.5599600043686416

7.4543198074643335

In [11]:*#RandomForestRegressor*

rf**=**RandomForestRegressor(max\_depth **=**10,n\_estimators**=**50,criterion**=**'mse')

rf**.**fit(xtrain,ytrain)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\\_forest.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be remov ed in version 1.2. Use `criterion='squared\_error'` which is equivalent.

warn(

Out[11]:

RandomForestRegressor(criterion='mse', max\_depth=10, n\_estimators=50)

In [20]:predtest**=**rf**.**predict(xtest)

In [13]:rfscore**=**rf**.**score(xtrain,ytrain)

print(rfscore)

0.9422063736562449

In [19]:rmse**=**mean\_squared\_error(ytest,predtest,squared**=False**)

print(rmse)

5.991984434322764

In [21]:pd**.**DataFrame(rf**.**feature\_importances\_,index**=**xtest**.**columns)**.**sort\_values(0,ascending**=False**)

Out[21]:

**0**

**Distance from nearest Metro station (km)** 0.532852 **House Age** 0.157235

**latitude** 0.134909

**longitude** 0.074192

**Transaction date** 0.035678

**House size (sqft)** 0.033227

**Number of convenience stores** 0.025328 **Number of bedrooms** 0.006579

In [22]:*# most important variable is "Distance from nearest Metro station (km)". Number of bedrooms is of least importance*

In [25]:*#AdaBoostRegressor*

adaboost **=** AdaBoostRegressor(base\_estimator**=**DecisionTreeRegressor(max\_depth **=**10,min\_samples\_split**=**100,random\_state**=**42),random\_state**=**42) adaboost**.**fit(xtrain,ytrain)

Out[25]:

AdaBoostRegressor(base\_estimator=DecisionTreeRegressor(max\_depth=10, min\_samples\_split=100, random\_state=42), random\_state=42)

In [26]:predtest**=**adaboost**.**predict(xtest)

adaboostscore**=**adaboost**.**score(xtrain,ytrain)

print(adaboostscore)

rmse\_adaboost**=**mean\_squared\_error(ytest,predtest,squared**=False**)

print(rmse\_adaboost)

0.7803649908108372

6.880893467955216

In [27]:*#GradientBoosting*

gradientboost**=**GradientBoostingRegressor(max\_depth**=**10,min\_samples\_split**=**100,learning\_rate**=**0.01,random\_state**=**42) gradientboost**.**fit(xtrain,ytrain)

Out[27]:

GradientBoostingRegressor(learning\_rate=0.01, max\_depth=10, min\_samples\_split=100, random\_state=42)

In [28]:predtest**=**gradientboost**.**predict(xtest)

gradientboostscore**=**gradientboost**.**score(xtrain,ytrain)

print(gradientboostscore)

rmse\_gradientboost**=**mean\_squared\_error(ytest,predtest,squared**=False**)

print(rmse\_gradientboost)

0.5972387243192092

7.67842493446194

In [29]:*#r2 drops then rmse increases*

In [ ]:mlpregressor**=**MLPRegressor(hidden\_layer\_sizes**=**8,activation**=**"relu",solver**=**"adam",verbose**=**"True",n\_iter\_no\_change**=**1000,max\_iter**=**20000,tol**=**0.001,random\_ mlpregressor**.**fit(xtrain,ytrain)

In [31]:predtest**=**mlpregressor**.**predict(xtest)

mlpregressorscore**=**mlpregressor**.**score(xtrain,ytrain)

print(mlpregressorscore)

rmse\_mlpregressor**=**mean\_squared\_error(ytest,predtest,squared**=False**)

print(rmse\_mlpregressor)

0.5993779066597805

7.6337458075450275

In [32]:*# we can use other activation functions as well*

In [41]:xgboost**=**XGBRegressor(max\_depth **=**10,learning\_rate**=**0.1,reg\_alpha**=**1,random\_state**=**42)

xgboost**.**fit(xtrain,ytrain)

Out[41]:

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None, colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=0, gpu\_id=-1, grow\_policy='depthwise', importance\_type=None, interaction\_constraints='', learning\_rate=0.1, max\_bin=256, max\_cat\_threshold=64, max\_cat\_to\_onehot=4, max\_delta\_step=0, max\_depth=10, max\_leaves=0, min\_child\_weight=1, missing=nan, monotone\_constraints='()', n\_estimators=100, n\_jobs=0, num\_parallel\_tree=1, predictor='auto', random\_state=42, ...)

In [42]:predtest**=**xgboost**.**predict(xtest)

xgboostscore**=**xgboost**.**score(xtrain,ytrain)

print(xgboostscore)

rmse\_xgboost**=**mean\_squared\_error(ytest,predtest,squared**=False**)

print(rmse\_xgboost)

0.9996214017389085

5.8657395746949526

In [43]:*# xgboost gave 99 percent accuracy . It is one of the best algorithm to use* In [44]:*# stacking model - combining different algorithms*

In [ ]:stacking **=** StackingRegressor(estimators**=**estimators,final\_estimator **=**final\_estimator ,cv**=**5) stacking**.**fit(xtrain,ytrain)

In [48]:estimators **=** [

("GradientBoost",gradientboost),

("MLP",mlpregressor),

("AdaBoost",adaboost),

("RandomForest",rf),

]

final\_estimator **=**XGBRegressor(max\_depth **=**10,learning\_rate**=**0.1,reg\_alpha**=**1,random\_state**=**42)

In [51]:predtest**=**stacking**.**predict(xtest)

stackingscore**=**stacking**.**score(xtrain,ytrain)

print(stackingscore)

rmse\_stacking**=**mean\_squared\_error(ytest,predtest,squared**=False**)

print(rmse\_stacking)

0.7241149778236378

7.046476088026656

In [55]:score**=**[]

score**.**extend([

Lscore,

xgboostscore,

stackingscore,

mlpregressorscore,

gradientboostscore,

adaboostscore,

rfscore]

)

In [57]:comparison\_frame**=**pd**.**DataFrame(

{

"Model" : [

"LinearRegression",

"XGBRegression",

"StackingRegression",

"MLPRegression",

"GradientBoost",

"AdaBoost",

"RandomForestRegression"

],

"Score" : score

}

)

In [60]:comparison\_frame**.**sort\_values(by **=**"Score",ascending **=True**)

Out[60]:

**Model Score**

**0** LinearRegression 0.559960 **4** GradientBoost 0.597239 **3** MLPRegression 0.599378 **2** StackingRegression 0.724115 **5** AdaBoost 0.780365 **6** RandomForestRegression 0.942206 **1** XGBRegression 0.999621

In [1]: *# Above analysis shows that we can choose either XGB Regressor or RandomForestRegression for this model*