Housing Regression from Kaggle

1: Importing the pandas library for use.

2: Create a DataFrame object from the main_file_path string using the pandas.read_csv() method.

3: Prints the columns of the dataset to show what variables are available for analysis.

In [3]:
```
import pandas as pd
main_file_path = 'train.csv' #set a string for the file path of your csv file
data = pd.read_csv(main_file_path) #create a data object/variable from the file path string using the pandas read_csv m
ethod
print(data.columns) #print the columns of the data using the column method in order to ascertain what variables there a
re
```

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

4:Uses dot notation to print out the head (first five rows) of the "SalePrice" column of our DataFrame.

In [6]:
```
saleprice_data = data.SalePrice #use dot notation (to get a field of the data object) that aligns with the colum you ar
e looking for
print(saleprice_data.head()) # print the top few entries of the data colum selected
```

```
0    208500
1    181500
2    223500
3    140000
4    250000
Name: SalePrice, dtype: int64
```

5: Creates a list of two "features" (x variables) that we will use for analysis from the list printed above in step 3.

6: Uses a list splicing method to select the values under the columns we will use for analysis.

7: Prints and gives a basic statistical analysis of the columns selected.

In [7]:
```
listofcolumns = ['Neighborhood','YearBuilt']  #create a list using column names that are of interest
twodatacolumns = data[listofcolumns] # "splice" the data using the list of relevant combos
print(twodatacolumns)
twodatacolumns.describe() #describes the smaller data sheet created
```

```
     Neighborhood  YearBuilt
0         CollgCr       2003
1         Veenker       1976
2         CollgCr       2001
3         Crawfor       1915
4         NoRidge       2000
5         Mitchel       1993
6         Somerst       2004
7          NWAmes       1973
8         OldTown       1931
9         BrkSide       1939
10         Sawyer       1965
11        NridgHt       2005
12         Sawyer       1962
13        CollgCr       2006
14          NAmes       1960
15        BrkSide       1929
16          NAmes       1970
17         Sawyer       1967
18        SawyerW       2004
19          NAmes       1958
20        NridgHt       2005
21         IDOTRR       1930
22        CollgCr       2002
23        MeadowV       1976
24         Sawyer       1968
25        NridgHt       2007
26          NAmes       1951
27        NridgHt       2007
28          NAmes       1957
29        BrkSide       1927
...           ...        ...
1430      Gilbert       2005
1431      NPkVill       1976
1432      OldTown       1927
1433      Gilbert       2000
1434      Mitchel       1977
1435        NAmes       1962
1436        NAmes       1971
1437      NridgHt       2008
1438      OldTown       1957
1439       NWAmes       1979
1440      Crawfor       1922
1441      CollgCr       2004
1442      Somerst       2008
1443      BrkSide       1916
1444      CollgCr       2004
1445       Sawyer       1966
1446      Mitchel       1962
1447      CollgCr       1995
1448      Edwards       1910
1449      MeadowV       1970
1450        NAmes       1974
1451      Somerst       2008
1452      Edwards       2005
1453      Mitchel       2006
1454      Somerst       2004
1455      Gilbert       1999
1456       NWAmes       1978
1457      Crawfor       1941
1458        NAmes       1950
1459      Edwards       1965

[1460 rows x 2 columns]
```

Out[7]:

	YearBuilt
count	1460.000000
mean	1971.267808
std	30.202904
min	1872.000000
25%	1954.000000
50%	1973.000000
75%	2000.000000
max	2010.000000

9: Defines the y variable of what value will be predicted (and trained with) as the SalePrice column, which we defined above.

10: Defines the x variable dataset as the data[listofxvalues] columns, used to predict the y.

11: Imports and creates a Decision Tree Regressor object called iowamodel that can modified and trained to predict values. This prints the values that change how the tree works and optimizes it, although all of these values are set to default as of right now.

12: Fits the data defined as x and y to the model, training it to make predictions.

In [8]:
```
y = saleprice_data #define your y variable (what will be predicted), we already did this above
listofxvalues = ['LotArea','YearBuilt','1stFlrSF','2ndFlrSF','FullBath','BedroomAbvGr','TotRmsAbvGrd']# the x values be
ing used
x = data[listofxvalues] # once again splice the data into the values we are looking for
from sklearn.tree import DecisionTreeRegressor #imports the regression system
iowamodel = DecisionTreeRegressor() #declare a model object
iowamodel.fit(x,y) # train the model using the x and y declared above
```

Out[8]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

13: Prints the x values that will be used to make a prediction, the predictions, and the actual y values. Notice how the values are the same, as the model has been trained using this same testing dataset. This is called bias, as the model isn't actually predicting but "remembering".

In [36]:
```
print('Making predictions for the following five houses:')
print(x.head()) # top 5 lines of the column
print('\nThe predictions are:')
print(iowamodel.predict(x.head())) #using the trained model predict y values from those x value
print('\nThe actual y values are:')
print(y.head()) #actual first 5 price values
```

```
Making predictions for the following five houses:
   LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
0     8450       2003       856       854         2             3
1     9600       1976      1262         0         2             3
2    11250       2001       920       866         2             3
3     9550       1915       961       756         1             3
4    14260       2000      1145      1053         2             4

   TotRmsAbvGrd
0             8
1             6
2             6
3             7
4             9

The predictions are:
[208500. 181500. 223500. 140000. 250000.]

The actual y values are:
0    208500
1    181500
2    223500
3    140000
4    250000
Name: SalePrice, dtype: int64
```

14: Imports a method that will calculate the average residual for every data point in the model.

15: Predicts all values possible from our set of x data.

16: Calculates and prints the mean residual for every point of our model, which is exceptionally low, once again due to bias.

In [10]:
```
from sklearn.metrics import mean_absolute_error #import to find mean absolute error (simple residual)
predictedvalues = iowamodel.predict(x) #predict A L L of the possibilities
mean_absolute_error(y, predictedvalues) #calculate and print error
```

Out[10]:
```
62.35433789954339
```

17: Imports and uses a method that randomly splits the x and y data into a training set and a testing set that will help to prevent model bias, but increase the error because of it.

18: Creates a new model and trains it with the new testing and training data.

19: Prints the mean redisuals for every point of our model which is now much higher, but also less biased. This model is now extrapolating rather than interpolating.

In [37]:
```
from sklearn.model_selection import train_test_split #okay so now we are going to split the data set between training a
nd predicting to actually test the effectiveness of the model as well as to prevent biases from interpolation

train_x, val_x, train_y, val_y = train_test_split(x, y, random_state =0) #WHEW okay so we have a tuple of values setup
 that is equal to this method that uses a RNG to determine which values are in each category
iowamodel2 = DecisionTreeRegressor() #creating a new model
iowamodel2.fit(train_x, train_y) #training the model using the specified training data
predictions =  iowamodel2.predict(val_x) #predicts the values
mean_absolute_error(val_y, predictions) #finds the total mean absolute error and prints
```

Out[37]:
```
33919.509589041096
```

20: Defines a function MAE (mean absolute error, or mean residuals) that takes a numleaves argument as well as the dataset.

21: Creates a model with the numleaves value passed to it, which changes the max_leaf_nodes value. This value changes how many final "nodes" there are for the tree to choose from. If there are too many nodes there are only a few examples for each node, meaning this model would be overfit and unreliably predict values, while too few nodes would not allow enough diversity for predictions, leading to an underfit model.

22: Returns the MAE of the model created with the given max_leaf_nodes value.

23: Uses a for loop to go through a list of values for the max_leaf_nodes values, and find the minimum value among them. At each stage it prints the MAE for the given value.

24: Prints the best value found for the max_leaf_nodes value (lowest MAE).

In [39]:
```
def mae(max_leaf_nodes, trainx, trainy, valx, valy):#defining a neat function that calculates the average residual for
 us. used to prevent over or underfitting from the model
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=0) #some more fancy schmancy stuff, but b
asically it creates a model with some specific attributes, notably the number of leaves and the random state agaiuun? d
ont know what random state is for
    model.fit(trainx, trainy) #just training the model, though we could pass a model as an argument if necessary
    predicted = model.predict(valx) #once again predicts values
    return (mean_absolute_error(valy, predicted))#returns the average residual!
train_x, val_x, train_y, val_y = train_test_split(x, y, random_state =0) #once again splitting the model into test sets

min = 999999999 # arbitrarily setting up a very large value to detect a min
num = 0 #declaring a variable that will print out the best leaf config
for i in [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85]: #for loop through a list of a bunch of possible values. rn i man
ually
    result = mae(i, train_x, train_y, val_x, val_y)#calculates a result through the values
    if result < min: #testing to find min value
        min = result
        num = i
    print('The MAE for %s is %f:'%(i, result))
print('The leaf configuration that best fits is: %s and the MAE calculated is %s' %(num, min))
```

```
The MAE for 75 is 27455.537405:
The MAE for 76 is 27407.400419:
The MAE for 77 is 27344.129436:
The MAE for 78 is 27258.932587:
The MAE for 79 is 27280.232090:
The MAE for 80 is 27280.232090:
The MAE for 81 is 27241.884768:
The MAE for 82 is 27203.783574:
The MAE for 83 is 27458.229984:
The MAE for 84 is 27458.229984:
The MAE for 85 is 27648.298477:
The leaf configuration that best fits is: 82 and the MAE calculated is 27203.783573767258
```

25: Imports pandas, time, and various sklearn libraries that will be used for building and analyzing the model.

26: Creates a directory string to refer the the train.csv file and creates a DataFrame object from that csv file.

27: Creates a y value of the DataFrame that is the SalePrice column, and an x value from the given xlist columns.

28: Splits the x and y data into test and train data sets

29: Declares some variables(and a dictionary) that will be used to return the optimal model values.

30: Goes through a for loop of different n_estimator values to optimize a RandomForest regressor rather than a decision tree. RandomForest essentially averages the results of multiple decision trees, allowing for a more accurate prediction.

31: In this for loop, the time spent to calculate the predictions for the model and the minimum value are recorded and printed. This is because as you increase n_estimators the model becomes more accurate, but it also takes much longer to process and has diminishing returns. n_estimators is essentially the number of decision trees in the random forest.

32: Creates a final model using the optimal n_estimators value and trains it with the entire dataset.

33: Creates a new string for the test data directory, and then predicts the prices from the given x values.

34: The predicted values are put into a DataFrame with the Id and turned into a csv, which is output and turned into Kaggle for scoring

35: This model recieved a Root Mean Squared Logarithmic Error of 0.19069, which is the top 72%.

In [47]:
```
#This is me using the techniques I've learned throughout this course in one block to show how to create these models
import pandas as pd
import time #going to be used for time spent calcs
from sklearn.ensemble import RandomForestRegressor #importing a different regression method
from sklearn.metrics import mean_absolute_error # import residual function
from sklearn.model_selection import train_test_split #splits data

directory = 'train.csv' #once again declaring a directory variable as string
data = pd.read_csv(directory) #create a data frame using pandas from the csv file

y = data.SalePrice # uses dot notation to identify what we are predicting
xlist = ['LotArea','YearBuilt','1stFlrSF','2ndFlrSF','FullBath','BedroomAbvGr','TotRmsAbvGrd'] #defines an x list of co
lumns/features to be used
x = data[xlist] #creating our x data using slicing with our list


train_x, val_x, train_y, val_y = train_test_split(x, y, random_state = 0) #splits the data into test and train sets
min = 99999999999999
g = 0
t2 = {}
for i in (5, 25, 50, 75, 100, 200, 500, 1000, 2000, 5000): #tests different values for n_estimators, explained in the c
omments below
    start = time.time()
    model = RandomForestRegressor(n_estimators = i, random_state =0) #instantiate model using a different value, still
 unsure as to what random_state does
    model.fit(train_x, train_y) # fit the model with training data
    predicted = model.predict(val_x) #define predicted value
    mae = mean_absolute_error(val_y, predicted)
    if(mae < min):
        min = mae
        g = i
    end = time.time()
    t = end-start
    t2[i] = t
    print('n_estimators: %s \t MAE: %0.2f \t Time spent in seconds to calculate: %0.2f'%(i, mae, t)) # print the MAE (a
vergae residual)
print('The best fit is an n_estimators value of %s with an MAE of %0.2f, and a time spent calculating of %0.2f'%(g, min
, t2[g]))

 #http://scikit-learn.org/stable/modules/ensemble.html#forest
 #The main parameters to adjust when using these methods is n_estimators and max_features.
 #The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute.
 #In addition, note that results will stop getting significantly better beyond a critical number of trees.
 #The latter is the size of the random subsets of features to consider when splitting a node.
 #The lower the greater the reduction of variance, but also the greater the increase in bias.

model = RandomForestRegressor(n_estimators = 2000, random_state =0) #building a final model
model.fit(x, y) #fitting the model with data, no splitting
testdata = pd.read_csv('test.csv') #declaring the testdata as a separate csv
testx = testdata[xlist]
predictedprices = model.predict(testx)
submission = pd.DataFrame({'Id':testdata.Id, 'SalePrice':predictedprices})
submission.to_csv('submisson.csv', index=False)
print(submission)
```

```
n_estimators: 5         MAE: 25330.81   Time spent in seconds to calculate: 0.03
n_estimators: 25        MAE: 23591.65   Time spent in seconds to calculate: 0.08
n_estimators: 50        MAE: 23242.59   Time spent in seconds to calculate: 0.17
n_estimators: 75        MAE: 22964.55   Time spent in seconds to calculate: 0.25
n_estimators: 100       MAE: 23093.06   Time spent in seconds to calculate: 0.34
n_estimators: 200       MAE: 23007.88   Time spent in seconds to calculate: 0.65
n_estimators: 500       MAE: 22907.49   Time spent in seconds to calculate: 1.66
n_estimators: 1000      MAE: 22909.57   Time spent in seconds to calculate: 3.26
n_estimators: 2000      MAE: 22897.42   Time spent in seconds to calculate: 6.45
n_estimators: 5000      MAE: 22946.20   Time spent in seconds to calculate: 16.47
The best fit is an n_estimators value of 2000 with an MAE of 22897.42, and a time spent calculating of 6.45
        Id      SalePrice
0     1461  121550.027500
1     1462  155215.732000
2     1463  183307.725500
3     1464  178695.345000
4     1465  187554.218500
5     1466  182530.208500
6     1467  172034.398500
7     1468  174431.388500
8     1469  190225.981500
9     1470  115569.656000
10    1471  189907.909000
11    1472   93429.728571
12    1473   90885.108333
13    1474  144861.007000
14    1475  125732.668405
15    1476  323603.639000
16    1477  245513.105500
17    1478  276044.833000
18    1479  333591.023000
19    1480  453528.977500
20    1481  314215.724000
21    1482  203650.725500
22    1483  202758.019500
23    1484  164398.395000
24    1485  173339.290000
25    1486  207448.420000
26    1487  282236.427000
27    1488  250997.725500
28    1489  199188.553500
29    1490  238174.335500
...    ...            ...
1429  2890   82913.205500
1430  2891  141249.894000
1431  2892  103344.150000
1432  2893  120833.885357
1433  2894   78923.921500
1434  2895  295954.008000
1435  2896  269585.545500
1436  2897  197413.860000
1437  2898  136480.738350
1438  2899  220304.128500
1439  2900  160795.696167
1440  2901  234200.400000
1441  2902  191044.514500
1442  2903  367104.007000
1443  2904  307746.492500
1444  2905  186885.162000
1445  2906  204407.345000
1446  2907  120982.247000
1447  2908  124082.240000
1448  2909  139163.404333
1449  2910   81064.750000
1450  2911   93619.251786
1451  2912  149742.655000
1452  2913   89285.183333
1453  2914   89145.483333
1454  2915   83897.491667
1455  2916   87476.200000
1456  2917  155757.312500
1457  2918  124403.350000
1458  2919  227485.470000

[1459 rows x 2 columns]
```