THE UNIVERSITY OF TEXAS AT DALLAS

COURSE: BUAN 6340 Programming for Data Science

INSTRUCTOR: Rabih Neouchi

PROJECT REPORT

ON

"Melbourne Housing Prices"

Submitted by:

ALEX SHARIFIAN	**AAS093020**
AMRUTHA B	**AXB210026**
BENJAMIN JOSEPH GARTMAN	**BXG123130**
SUNGHO YIM	**SXY170630**

TABLE OF CONTENTS

1. MOTIVATION SECTION

1.1 Project Idea

The increasing unaffordability of housing is a significant challenge for many cities in the world. To obtain an improved understanding of the current housing market, we would like to examine the top influential factors of housing prices to provide a model for more accurate estimations. Our objective is to observe and examine major factors affecting housing prices and to obtain accurate predictions. Methods of statistical and machine learning regression models are applied to obtain predictions and estimates.

1.2 Data Source

The data set contains publicly available information posted on a weekly basis on Domain.com.au. https://www.kaggle.com/datasets/dansbecker/melbourne-housing-snapshot

1.3 Data Description

Number of observations: 13,580

Missing data: 4.64%

Categorical variables: CouncilArea, Regionname, Year Built

Variables/Columns:

1. **Suburb**: The name of the suburb the property is located in'
2. **Address**: The street address of the property
3. **Rooms**: The number of rooms in the property
4. **Type**: The type of property (h- house/cottage/etc, u - unit/duplex, t - townhouse)
5. **Price**: The price in dollars
6. **Method**: The type of listing
7. **SellerG**: The name of the real estate agent
8. **Date**: The date the property was sold
9. **Distance**: The distance to the central business district
10. **Postcode**: The postcode for the property
11. **Bedroom2**: The number of bedrooms (scraped from a different source)
12. **Bathroom**: The number of bathrooms
13. **Car**: The number of car parking spots for the property
14. **Landsize**: The size of the lot of land
15. **BuildingArea**: The size of the property (building specifically)
16. **YearBuilt**: The year built
17. **CouncilArea**: The governing council for the area
18. **Latitude**: A measure of how far north a house is; a higher value is farther north
19. **Longitude**: A measure of how far west a house is; a higher value is farther west
20. **Regionname:** Combination of general region and cardinal direction
21. **Propertycount:** The number of properties that exist in the suburb

2. DATA CLEANSING ACTIVITIES

2.1 Handling Duplicate Records

Duplicate entries are problematic for multiple reasons. First off, when an entry appears more than once, it receives a disproportionate weight during training. Thus, models that succeed on frequent entries will look like they perform well, while this is not the case. Additionally, duplicate entries can ruin the split between train, validation, and test sets in cases where identical entries are not all in the same set. This can lead to biased performance estimates that will lead to disappointing models in production.

One option to eliminate the duplicate records is to run a script to automatically detect and delete duplicate entries. Using pandas, this can be done easily with drop_duplicates() functionality

```
# check for duplicates
print('Duplicate Rows:', house.duplicated().sum())
```

```
Duplicate Rows: 0
```

2.2 Handling Missing Values

Many machine learning algorithms fail if the dataset contains missing values. It may lead to building a biased machine learning model which will lead to incorrect results if the missing values are not handled properly. Missing data can also lead to a lack of precision in the statistical analysis.

```
# num of nulls on each column
house.isna().sum().sort_values(ascending = False)
```

```
BuildingArea     6450
YearBuilt        5375
CouncilArea      1369
Car                62
Suburb              0
latBin              0
Propertycount       0
Regionname          0
Longtitude          0
Lattitude           0
Landsize            0
Bathroom            0
Address             0
Bedroom2            0
Postcode            0
Distance            0
Date                0
SellerG             0
Method              0
Price               0
Type                0
Rooms               0
lonBin              0
dtype: int64
```

3. DATA PREPARATION ACTIVITIES

As part of data preparation activities, we are converting the values of YearBuilt to int for better calculations and converting the column names to upper case to maintain uniformity. Along with this, we are also using One hot encoding which is a process of converting categorical data variables so they can be provided to machine learning algorithms to improve predictions.

```
#convert house build to int
house['YearBuilt'] = house['YearBuilt'].astype('int')
house['Date'] = pd.to_datetime(house['Date'], format = '%d/%m/%Y')
house['YEAR'] = pd.DatetimeIndex(house['Date']).year
house['MONTH'] = pd.DatetimeIndex(house['Date']).month
```

```
house['YearBuilt']
```

```
0        1962
1        1900
2        1900
3        1962
4        2014
         ...
13575    1981
13576    1995
13577    1997
13578    1920
13579    1920
Name: YearBuilt, Length: 13580, dtype: int32
```

```
#Convert column names to upper case
house.columns= [x.upper() for x in house.columns]
#print(house.columns)
print(house.head(20))
```

```
       SUBURB  ROOMS TYPE      PRICE METHOD  SELLERG  DISTANCE  POSTCODE  \
0  Abbotsford      2    h  1480000.0      S   Biggin       2.5    3067.0
1  Abbotsford      2    h  1035000.0      S   Biggin       2.5    3067.0
2  Abbotsford      3    h  1465000.0     SP   Biggin       2.5    3067.0
3  Abbotsford      3    h   850000.0     PI   Biggin       2.5    3067.0
4  Abbotsford      4    h  1600000.0     VB   Nelson       2.5    3067.0
```

```
house = house.apply(lambda x: x.mask(x.map(x.value_counts())<threshold, 'RARE') if x.name in obj_columns else x)
```

```
# one hot encoding
house_type = pd.get_dummies(house['TYPE'],prefix='TYPE')
house = house.drop('TYPE',axis=1)
house = house.join(house_type)


house_method = pd.get_dummies(house['METHOD'],prefix='METHOD')
house = house.drop('METHOD',axis=1)
house = house.join(house_method)


house_reg = pd.get_dummies(house['REGIONNAME'],prefix='REGIONNAME')
house = house.drop('REGIONNAME',axis=1)
house = house.join(house_reg)
```

4. VISUALIZATION TECHNIQUES

4.1 Descriptive Summary Statistics

Summary statistics is a part of descriptive statistics that summarizes and provides the gist of information about the sample data.

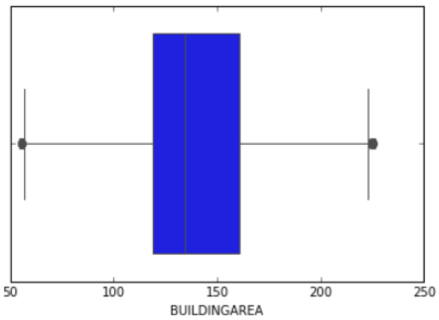
```
# Descriptive Summary Statistics (of numerical variables)
num_list = house.select_dtypes([np.int64, np.float64, np.int32]).columns
house[num_list].describe()
```

	ROOMS	PRICE	DISTANCE	BEDROOM2	BATHROOM	CAR	LANDSIZE	BUILDINGAREA	YEARBUILT	LATTITUDE
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	2.914728	1.534242	1.609551	558.416127	153.948820	1964.563549	-37.809203
std	0.955748	6.393107e+05	5.868725	0.965921	0.691712	0.960572	3990.669241	466.878614	29.295335	0.079260
min	1.000000	8.500000e+04	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000	-38.182550
25%	2.000000	6.500000e+05	6.100000	2.000000	1.000000	1.000000	177.000000	118.975286	1958.000000	-37.856822
50%	3.000000	9.030000e+05	9.200000	3.000000	1.000000	2.000000	440.000000	134.200516	1962.000000	-37.802355
75%	3.000000	1.330000e+06	13.000000	3.000000	2.000000	2.000000	651.000000	162.000000	1979.000000	-37.756400
max	10.000000	9.000000e+06	48.100000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000	-37.408530

4.2 Univariate Analysis

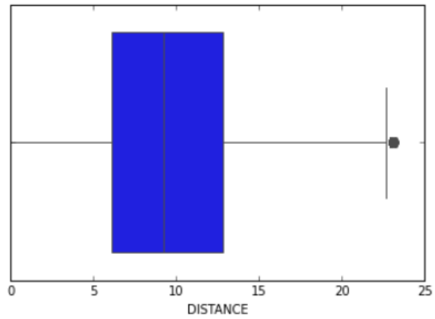
```
print(sns.boxplot(x=house['BUILDINGAREA']))
```

AxesSubplot(0.125,0.125;0.775x0.775)



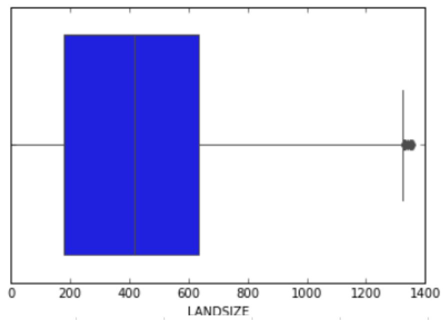

```
print(sns.boxplot(x=house['DISTANCE']))
```

AxesSubplot(0.125,0.125;0.775x0.775)



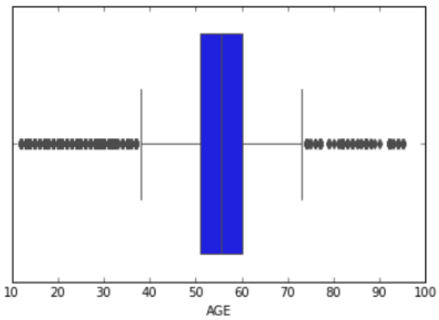

```
print(sns.boxplot(x=house['LANDSIZE']))
```

AxesSubplot(0.125,0.125;0.775x0.775)



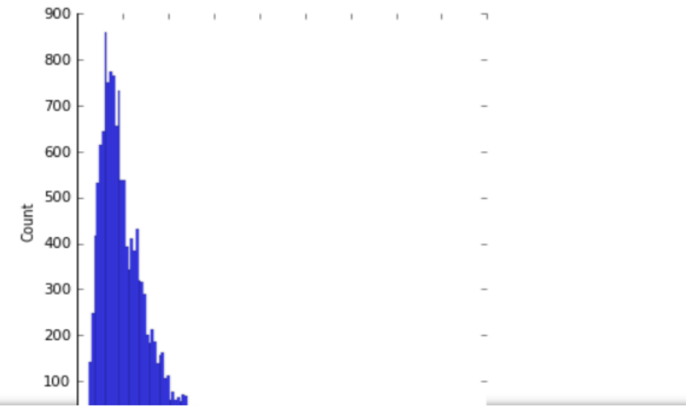

```
print(sns.boxplot(x=house['AGE']))
```

AxesSubplot(0.125,0.125;0.775x0.775)



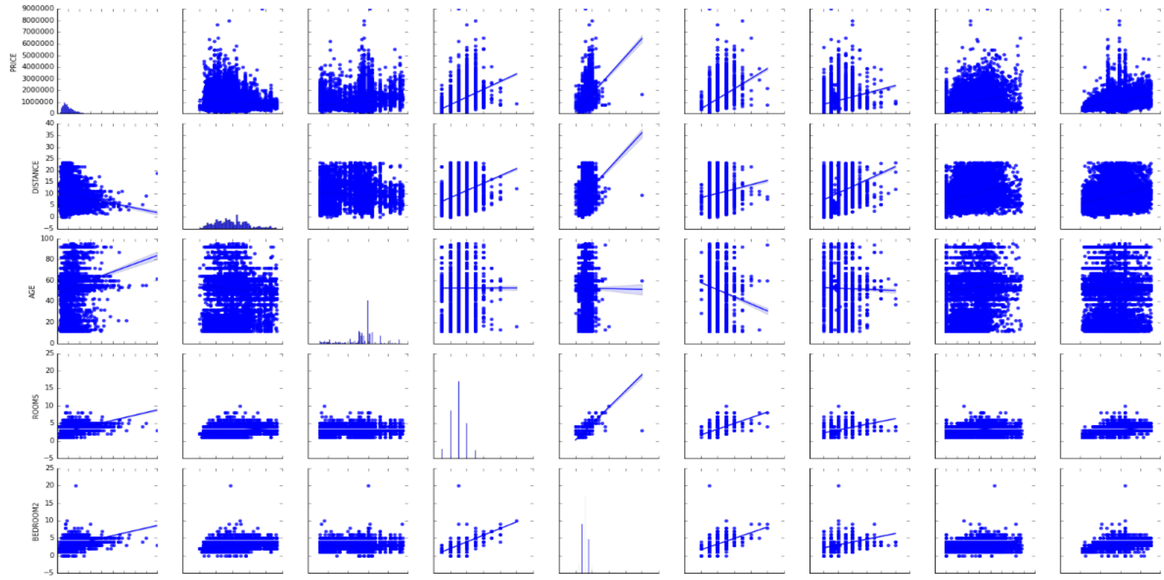
```
sns.displot(house['PRICE'])
plt.ioff()
```

```
<matplotlib.pyplot._IoffContext at 0x1f912857250>
```

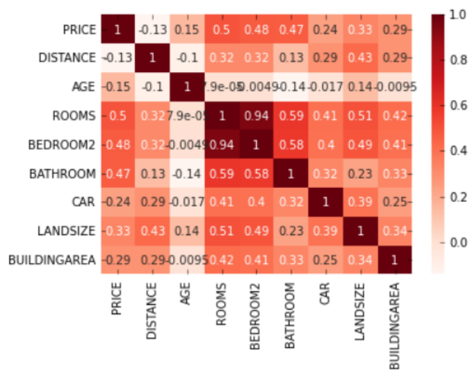


4.3 Multivariate Analysis

```
sns.pairplot(data = house, vars=['PRICE','DISTANCE','AGE','ROOMS','BEDROOM2','BATHROOM','CAR', 'LANDSIZE', 'BUILDINGAREA'],ki
plt.show()
```




```
#plot the correlation matrix of ROOMS, BEDROOM2, BATHROOM,CAR, LANDSIZE and BUILDINGAREA in data dataframe.
sns.heatmap(house[['PRICE','DISTANCE','AGE','ROOMS','BEDROOM2','BATHROOM','CAR', 'LANDSIZE', 'BUILDINGAREA']].corr(), annot=T
plt.show()
```



5. REGRESSION ANALYSIS

5.1 Linear Regression

Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x)

```
from sklearn.preprocessing import StandardScaler
scaling = StandardScaler()
X = house.loc[:, ~house.columns.isin(['PRICE','YEARBUILT'])]
y=scaling.fit_transform(house[['PRICE']])
```

```
# Data splitting for training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, random_state = 42 )
```

```
# create an object of linear regression
model = LinearRegression()

# Train the model using the training set
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Test set prediction
y_pred = model.predict(X_test)

# Test set residual
e = y_test - y_pred
# Test set MAE
mae_test = np.sum(np.abs(e))/y_test.shape[0]
mae_test
```

```
0.36408831593307267
```

```
# Test set MSE
mse_test = np.sum(np.square(e))/y_test.shape[0]
print('mse_test: ',mse_test)

# Test set RMSE
print('RMSE: ',mse_test**0.50)

model.score(X_test, y_test)
```

```
mse_test:  0.3171223082393231
RMSE:  0.5631361365063718

0.6914126495794632
```

5.2 Polynomial Regression

Polynomial Regression is a form of Linear regression known as a special case of Multiple linear regression which estimates the relationship as an nth degree polynomial. Polynomial Regression is sensitive to outliers so the presence of one or two outliers can also badly affect the performance

```
# polynomial regression
param_poly = {'polynomialfeatures__degree' : range(1, 5)  }

poly_cv = make_pipeline(PolynomialFeatures() , LinearRegression())

grid_poly = GridSearchCV( poly_cv,  param_poly, cv = 5)

grid_poly.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('polynomialfeatures',
                                        PolynomialFeatures()),
                                       ('linearregression',
                                        LinearRegression())]),
             param_grid={'polynomialfeatures__degree': range(1, 5)})
```

```
# Report the coefficients (including the intercept)
print('Intercept:',grid_poly.best_estimator_.named_steps['linearregression'].intercept_)
print('Coefs:',grid_poly.best_estimator_.named_steps['linearregression'].coef_)

# chosen hyperparameter (i.e., polynomial degree)
print('Chosen Hyperparamter:', grid_poly.best_estimator_.named_steps['polynomialfeatures'].degree)

# and the model performance (based on default measure, R-squared)
print('Model Performance:', grid_poly.best_score_)
```

```
Intercept: [-64.32123331]
Coefs: [[ 8.90895592e-12  2.28411747e-01 -2.91339184e-02  1.42519424e-02
   2.52014361e-01  5.30762748e-02  6.11204244e-04  1.80216696e-04
  -5.22905474e+00 -3.06845139e+00  1.47859015e-05 -3.67423160e-01
   1.71991982e-02  1.37361291e-01  1.33457213e-02  1.30306923e-04
```

5.3 Ridge Regression

Ridge regression is a model tuning method that is used to analyze any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

```
# Define Ridge model
ridge = Ridge(alpha = 1)

# train the model
ridge.fit(X_train, y_train)

# R-square for test set
R2_test_ridge = ridge.score(X_test, y_test)
print('R squared ridge : ', R2_test_ridge)
```

```
R squared ridge :  0.6911678887655799
```

5.4 Lasso Regression

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination.

```
# Lasso regression model. Hyper-tune its parameters. Evaluate the performance.
from sklearn.linear_model import Lasso

# Define model-alpha = 100
my_lasso = Lasso(alpha=100)

my_lasso.fit(X_train, y_train)

# prediction on test
y_pred_test_lasso = my_lasso.predict(X_test)

# Estimated coefficients and intercept
print('Coeffficents : ', my_lasso.coef_)
print('Intercept : ', my_lasso.intercept_)

# mse for test set
mse_lasso = np.sum((y_pred_test_lasso - y_test)**2)/y_pred_test_lasso.shape[0]
print('MSE lasso:', mse_lasso)

# R-square for test set
R2_test_lasso = my_lasso.score(X_test, y_test)
print('R squared lasso:', R2_test_lasso)
```

```
Coeffficents :  [ 0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -4.65401728e-06 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
```

Preferred Model for Prediction:

For our dataset, we would like to use Linear Regression Model for predications.

This is because model score for linear regression model is higher when compared to other regression models.

6. CLASSIFICATION TECHNIQUES

6.1 Logistic Regression Model

This type of statistical model is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1.

```
# Classification section
def my_func(row):
    if row['PRICE'] < 1045000:
        val = 0
    else:
        val = 1
    return val
house['PRICE_c'] = house.apply(my_func, axis=1)
house[['PRICE','PRICE_c']].head()
# classification
X = house.loc[:, ~house.columns.isin(['PRICE','PRICE_c'])]#,'YEARBUILT'
y=house['PRICE_c']
```

```
# Data splitting for training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, random_state = 42 )
```

```
#logistic regression model
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print('Coefficients : ', logreg.coef_)
print('Intercept : ', logreg.intercept_)
```

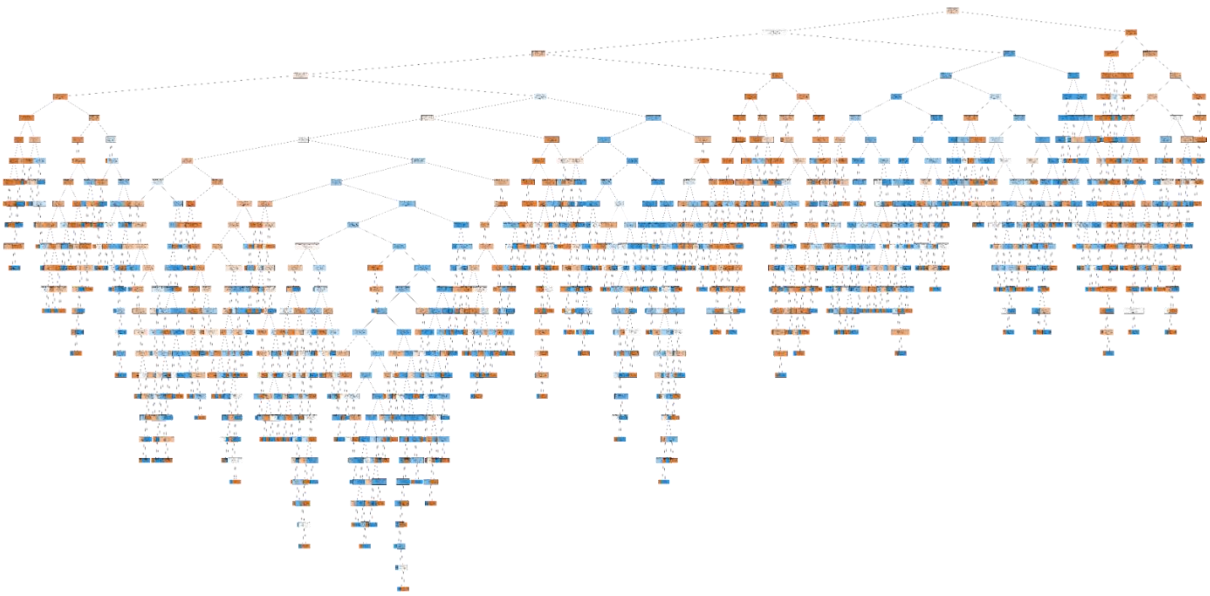
```
Coefficients :  [[ 5.66433601e-02 -2.23637663e-01  5.42651053e-02  4.83158603e-02
   1.84818887e-02  2.86473134e-03  2.94202445e-02 -2.18608350e-02
  -4.62013336e-03  5.00724411e-03 -3.38555833e-05 -5.59903257e-03
   4.49841242e-03  1.80553613e-02  1.01874479e-02 -7.18236678e-04
   1.83836077e-02  2.97811042e-03 -2.13526230e-02  1.95475824e-03
   1.83825290e-03 -1.87162930e-04 -6.12787088e-03  2.53111776e-03
   1.72750204e-04 -1.18325970e-02 -2.23055984e-03 -3.21234531e-03
```

6.2 Decision Tree Model

Decision tree models are used to develop classification systems that predict or classify future observations based on a set of decision rules. Tree based algorithms are a popular family of related non-parametric and supervised methods for both classification and regression.

```
from sklearn.tree import DecisionTreeClassifier

# A Basic Tree

clf_tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)

# Train the Model
clf_tree.fit(X_train, y_train)

# Obtain and print accuracy
clf_tree.score(X_test, y_test)

clf_tree.feature_importances_
#target_names
house.feature_names = X.columns
house.target_names = 'PRICE_c'
```

```
# Plot the decision tree
from sklearn import tree

# set a proper figure size (in case that the figure is too small to read or ratio is not proper)
fig = plt.figure(figsize=(200,100))

tree.plot_tree(clf_tree,
               feature_names = house.feature_names, # specify variable names
               class_names = house.target_names, # specify class (Y) names
               filled = True, # whether to color the boxes
               impurity = False, # whether to report gini index
               fontsize = 10) # set fontsize to read
plt.show()

# save the figure to read through the boxes, it is saved under the same directory as the coding doc.
fig.savefig("decision_tree_basic.jpg")
```



6.3 K-NN Classification Model

k-NN is a type of classification where the function is only approximated locally, and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

```
# KNN Model
from sklearn.neighbors import KNeighborsClassifier
# Define objective, train the model.
n_neighbors = 3
knn = KNeighborsClassifier(n_neighbors)
knn.fit(X_train, y_train)
# Do prediction, get accuracy
## prediction: knn.predict(X_test)
knn.score(X_test, y_test)
```

```
0.8122238586156112
```

```
from sklearn.model_selection import GridSearchCV

# define function
base_knn = KNeighborsClassifier()

# define a list of parameters
param_knn = {'n_neighbors': range(3, 27, 2)}  # exactly the same as the input variable name.

#apply grid search
grid_knn = GridSearchCV(base_knn, param_knn, cv = 5)
grid_knn.fit(X_train, y_train) # this line will take time. To get results quickly, start a new cell
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': range(3, 27, 2)})
```

```
# the best hyperparameter chosen:
print(grid_knn.best_params_)

# When best case, the validation score of through CV is:
print(grid_knn.best_score_)

# For each different grid, show the mean test(validation) score and mean training score (across 5 folds CV)
grid_knn.cv_results_['mean_test_score']

# Show how best_score is obtained
grid_knn.cv_results_['mean_test_score'].max()

# most important - Final model performance on Test set
grid_knn.score(X_test, y_test)
```

```
{'n_neighbors': 5}
0.7939194103938877

0.8095238095238095
```

Preferred Model for Prediction:

For our dataset, we would like to use Decision Tree for predictions.

This is because we have a balanced data set and Decision Tree has the highest accuracy score.

7. CONCLUSION

Takeaway:

As our main motivation of inspecting the data set was to assign values to the differences between each house listed and the price for the listing based on size, location, neighborhood, number of rooms/bathrooms.

We have concluded that the Linear Regression was our most accurate model.

Suggestions:

Having a greater number of records can increase the prediction accuracy.

Having a more recent data related to key areas, date, etc can give better results.