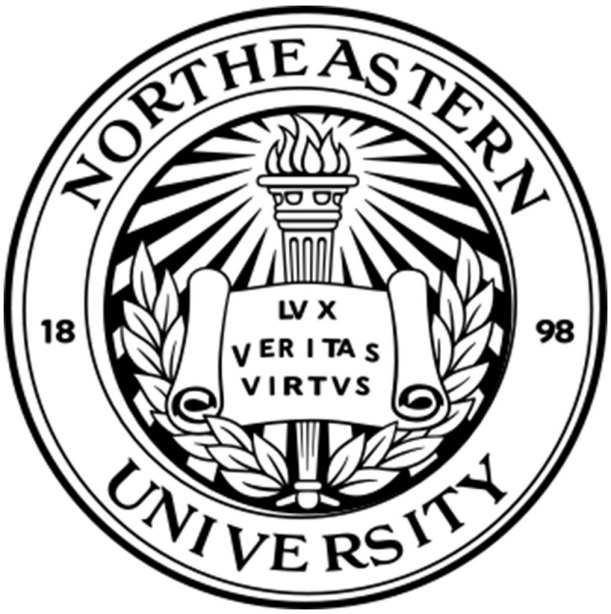


Module 5 Assignment

Submitted to Professor Dr. Mary Donhoffner

By Ananya Sharma

Introduction

In this dataset, a company is trying to understand the real estate market of Nashville before making some investments. Real estate markets are always a subject of study since they give a huge return on investments. We as analysts are trying to understand various aspects that can affect the price of real estate, for instance, the year it was built, commercial plot, land size, type of flooring, etc. (Bukovac, 2022)

Specifically for this project, we are trying if a property was undervalued or sold over its value.

Data Analysis

The dataset has 22651 rows and 26 columns. The data majorly has data types in float, integer, and object types.

```
1 Housing_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22651 entries, 0 to 22650
Data columns (total 26 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Unnamed: 0                       22651 non-null  int64
 1   Parcel ID                        22651 non-null  object
 2   Land Use                         22651 non-null  object
 3   Property Address                 22649 non-null  object
 4   Suite/ Condo   #                 0 non-null      float64
 5   Property City                    22649 non-null  object
 6   Sale Date                        22651 non-null  object
 7   Legal Reference                  22651 non-null  object
 8   Sold As Vacant                   22651 non-null  object
 9   Multiple Parcels Involved in Sale  22651 non-null  object
 10  City                             22651 non-null  object
 11  State                            22651 non-null  object
 12  Acreage                          22651 non-null  float64
 13  Tax District                     22651 non-null  object
 14  Neighborhood                     22651 non-null  int64
 15  Land Value                       22651 non-null  int64
 16  Building Value                   22651 non-null  int64
 17  Finished Area                    22650 non-null  float64
 18  Foundation Type                  22650 non-null  object
 19  Year Built                       22651 non-null  int64
 20  Exterior Wall                    22651 non-null  object
 21  Grade                            22651 non-null  object
 22  Bedrooms                         22648 non-null  float64
 23  Full Bath                        22650 non-null  float64
 24  Half Bath                        22543 non-null  float64
 25  Sale Price Compared To Value     22651 non-null  object
dtypes: float64(6), int64(5), object(15)
memory usage: 4.5+ MB
```

Figure 1: Housing Dataset Information

The summary tells us that the data is skewed and not evenly distributed. Furthermore, there are a lot of columns that don't tell us very clearly anything regarding the Sale price and must be obliterated. For instance, Unnamed 0, Suite/Condo that has a lot of NAN values and Parcel ID. Upon further exploration, we try to check if the dataset given has any duplicated or null values.

```
# Checking for the duplicated values

Housing_dataset.duplicated()

#As we can see the values are not duplicated
```

```
0        False
1        False
2        False
3        False
4        False
         ...
22646    False
22647    False
22648    False
22649    False
22650    False
Length: 22651, dtype: bool
```

Figure 2: Duplicated Value

The figure above shows that the dataset does not have any duplicated values and the column Suite/Condo has null values which must be removed up ahead. Acreage shows a huge range difference between max and 75%. Similarly, Full bath and Land Value also show some variation in the range.

```
Housing_dataset.describe()
```

	Unnamed: 0	Suite/ Condo #	Acreage	Neighborhood	Land Value	Building Value	Finished Area	Year Built	Bedrooms	Full Bath	Half Bath
count	22651.000000	0.0	22651.000000	22651.000000	2.265100e+04	2.265100e+04	22650.000000	22651.000000	22648.000000	22650.000000	22543.000000
mean	27889.491192	NaN	0.454705	4432.715024	7.013797e+04	1.722402e+05	1915.377151	1961.947684	3.104910	1.887285	0.270239
std	16598.865706	NaN	0.611818	2142.603595	1.029035e+05	1.696424e+05	1079.094521	25.643908	0.629287	0.951220	0.480186
min	1.000000	NaN	0.040000	107.000000	9.000000e+02	1.400000e+03	450.000000	1832.000000	0.000000	0.000000	0.000000
25%	13324.500000	NaN	0.200000	3130.000000	2.200000e+04	8.550000e+04	1250.000000	1947.000000	3.000000	1.000000	0.000000
50%	27712.000000	NaN	0.280000	4026.000000	3.000000e+04	1.188000e+05	1645.824995	1959.000000	3.000000	2.000000	0.000000
75%	42330.500000	NaN	0.460000	6229.000000	6.030000e+04	1.882500e+05	2213.375000	1977.000000	4.000000	2.000000	1.000000
max	56615.000000	NaN	17.500000	9530.000000	1.869000e+06	5.824300e+06	19728.249880	2017.000000	11.000000	10.000000	3.000000

Figure 3: Housing Dataset Summary

```
#Dropping the Column that has Null Values and the unwanted columns
Housing_dataset=Housing_dataset.drop(['Unnamed: 0', 'Parcel ID','Suite/ Condo   #'],axis=1)

#Renaming the Building Value into Sale Price
Housing_dataset.rename(columns={'Building Value':'Sale_Price'},inplace=True)
```

Figure 4: Dropping a few columns

Furthermore, I have renamed the column of Building Value to Sale Price, to make it easy for me to perform some EDA with respect to the property prices. Further down the project, I have dropped the column altogether to avoid any confusion.
Having done analyses on the Ames Housing dataset, for a Linear Regression Model in the intermediate analytics course, I was curious to explore any linear relations between the variables before I can apply any regression analysis or classification approaches.

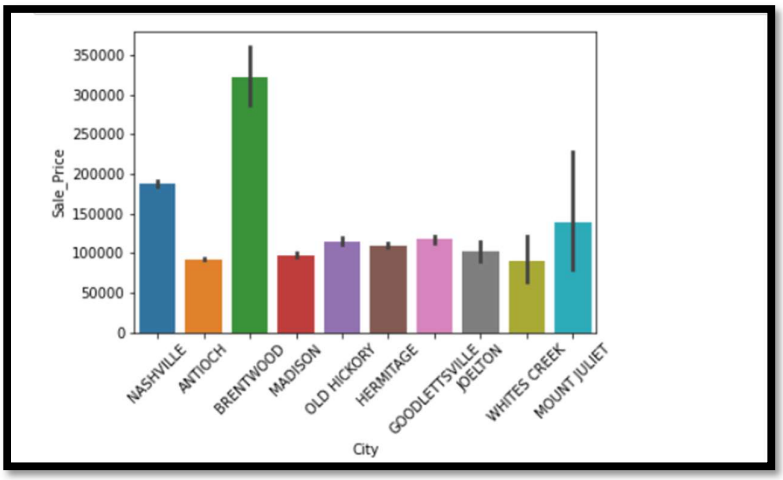


Figure 5: Housing Dataset

From the above bar plot, we understand that Brentwood has higher-priced properties ranging between $300K to $350K. The second-highest properties are found in the city of Nashville. This city has property prices ranging between $150K and $200K. All the other cities have property prices ranging from $50K to $100K.

Furthermore, upon examining the column of the target variable I find that more houses have been sold undervalued and roughly around 160K houses have been overpriced and sold in the real estate market.

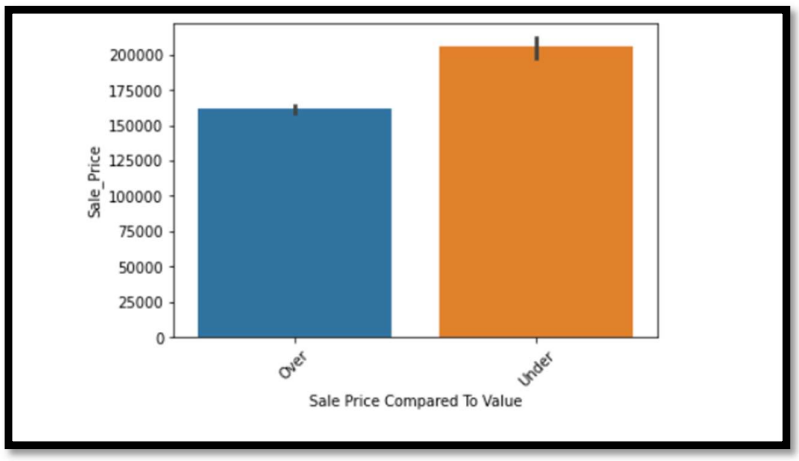


Figure 6: Price Comparison

On Further analysis of the Sales price by plotting a Histogram, I figure out that the data has been concentrated between 0 and 1 on the X-axis.

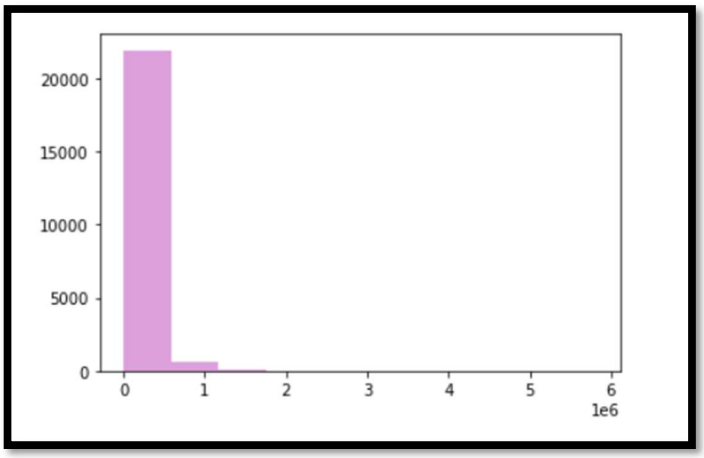


Figure 7: Exploring the Building Values (renamed to Sale_Price)

After fitting the logarithmic function to the data, we see that the range has improved, and we can see a histogram. The data seems to be slightly right-skewed here.

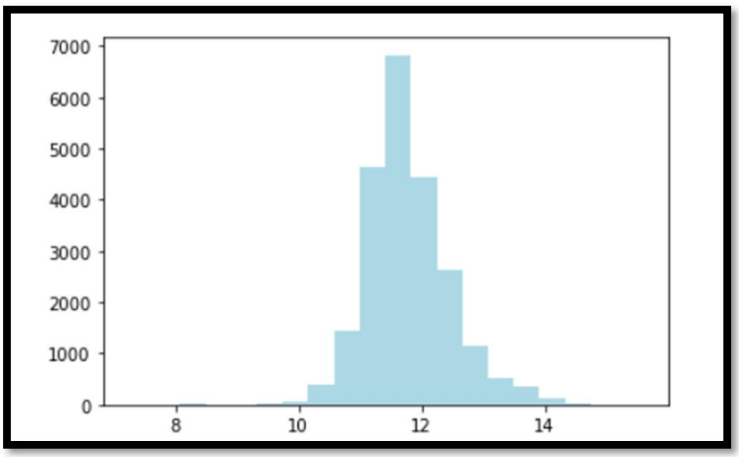


Figure 8: Sale_Price after Logarithmic fit

We try to plot a bar using the value count for Land use and we find that people occupy Single Family units more than other units.

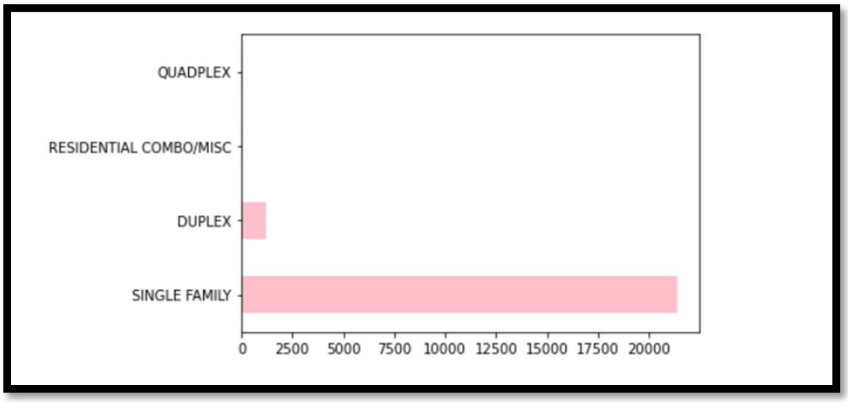


Figure 10: Land Use Value Counts

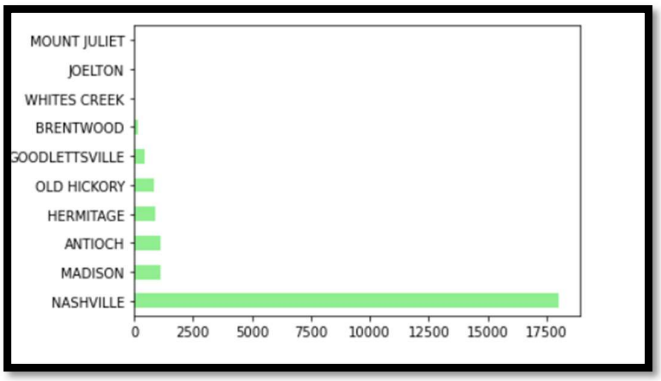


Figure 11: Value Counts of Property City

Furthermore, we figure out that Nashville has more properties than the other cities.

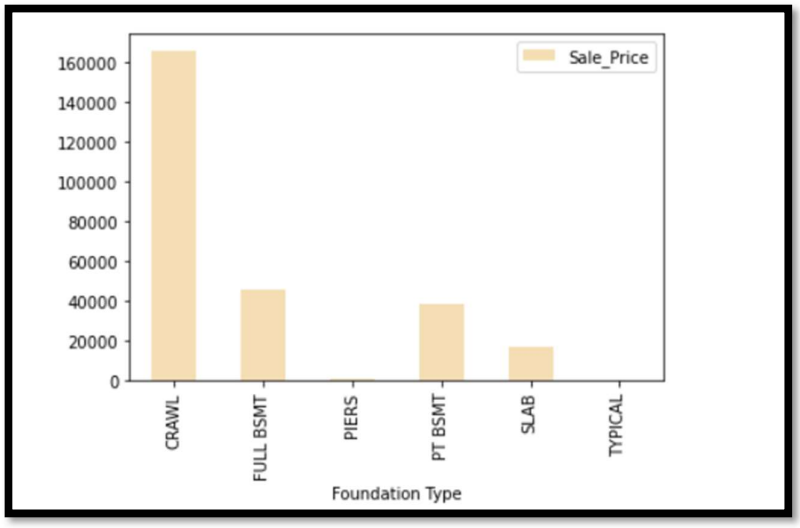


Figure 12: Sale Price by Foundation Price

Value Counts show that the foundation of Type Crawl is more common than the other types of foundations. Around 160K real estate properties prefer Crawl type foundation structure.

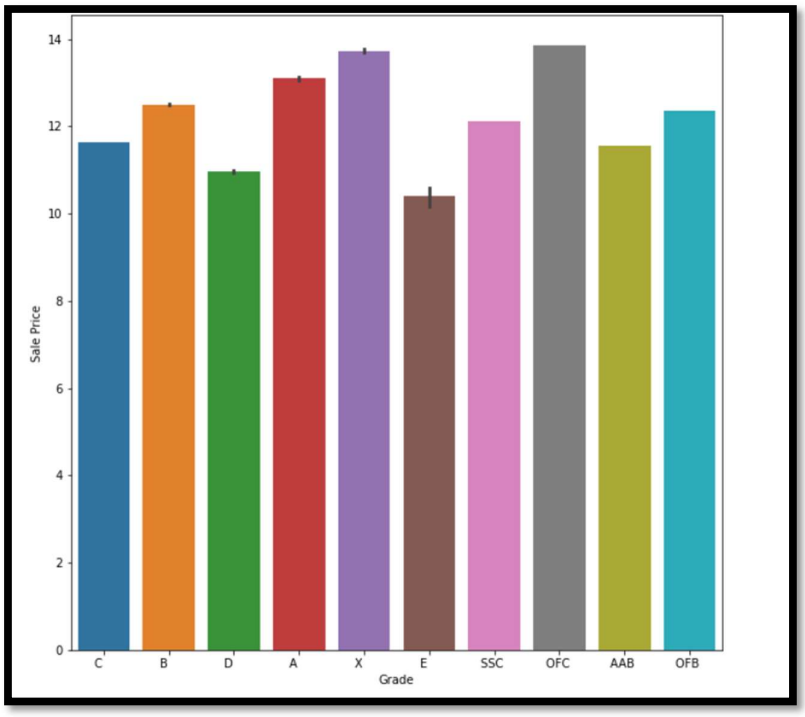


Figure 13: Real Estate Grades

While buying a property it is essential to check the grade before buying it, to understand its competitiveness and marketability. Property with Grade A and Office Space seems to have more Building Value.

```
4]:  1  Housing_dataset.isnull().sum()

4]: Land Use                          0
    Property Address                  2
    Property City                     2
    Sale Date                         0
    Legal Reference                   0
    Sold As Vacant                    0
    Multiple Parcels Involved in Sale 0
    City                              0
    State                             0
    Acreage                           0
    Tax District                      0
    Neighborhood                      0
    Land Value                        0
    Sale_Price                        0
    Finished Area                     1
    Foundation Type                   1
    Year Built                        0
    Exterior Wall                     0
    Grade                             0
    Bedrooms                          3
    Full Bath                         1
    Half Bath                       108
    Sale Price Compared To Value      0
    Sale_Date                         0
    dtype: int64
```

Figure 14: Null Values

```
94]:  1  Housing_dataset.isnull().sum()

94]:  Land Use                              0
      Property Address                      2
      Property City                         2
      Sale Date                             0
      Legal Reference                       0
      Sold As Vacant                        0
      Multiple Parcels Involved in Sale     0
      City                                  0
      State                                 0
      Acreage                               0
      Tax District                          0
      Neighborhood                          0
      Land Value                            0
      Sale_Price                            0
      Finished Area                         0
      Foundation Type                       1
      Year Built                            0
      Exterior Wall                         0
      Grade                                 0
      Bedrooms                              0
      Full Bath                             0
      Half Bath                             0
      Sale Price Compared To Value          0
      Sale_Date                             0
      Sale Price                        22557
      dtype: int64
```

Figure 15: After filling the null with mean values

The above figure shows that the columns have some null values, and we try to remove the same using the mean method.

Next, we try to plot a few box plots to understand the various outliers in the columns. We try to remove the outliers using the Interquartile Range Formula.

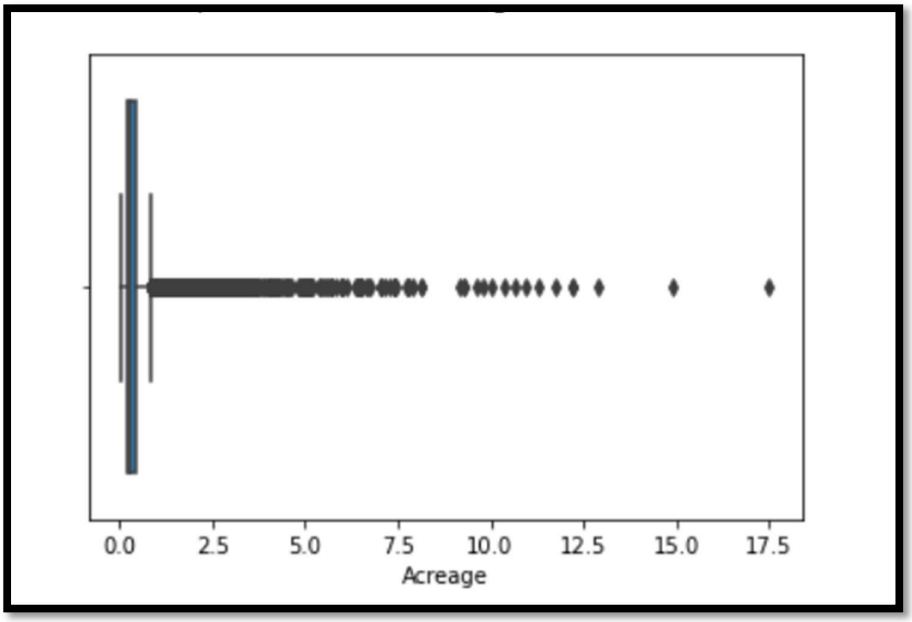


Figure 16: Outliers in Acreage

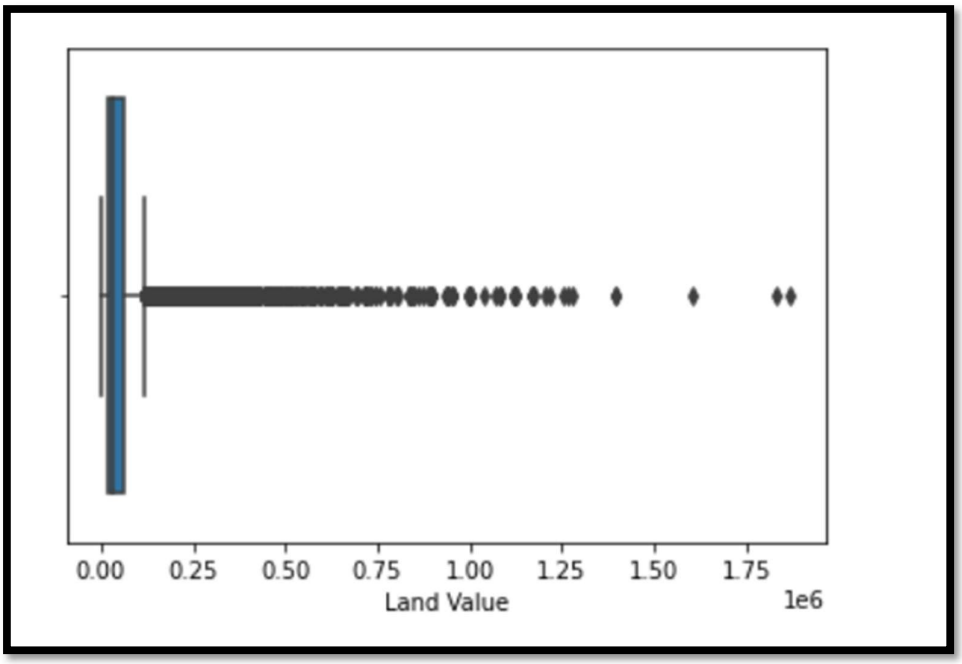


Figure 17: Outliers in Land Value

Furthermore, we try to check the Value Counts of the number of bedrooms and bathrooms as per the average number that a house can have to get a realistic analysis.

```
0]:  1 Housing_dataset['Full Bath'].value_counts()

0]: 2.0     9263
    1.0     8906
    3.0     3137
    4.0      921
    5.0      304
    6.0       80
    7.0       15
    0.0       11
    8.0        6
    10.0       4
    9.0        3
    Name: Full Bath, dtype: int64
```

Figure 18: Value Counts for Bathroom

```
1  Housing_dataset['Bedrooms'].value_counts()

3.0     12273
4.0      4646
2.0      4577
5.0       820
6.0       194
1.0        79
7.0        29
8.0        21
0.0         4
10.0        2
9.0         2
11.0        1
Name: Bedrooms, dtype: int64
```

Figure 19: Checking the count of the bedrooms

Here we are checking the counts of the bedroom so that we can limit the size and bring it down to an average value to make the range easily readable. I added this step after fitting my models. My models refused to work due to a large range of issues.

```
1  Housing_dataset['Bedrooms'][Housing_dataset['Bedrooms']>=5]=5
2  Housing_dataset['Bedrooms'][Housing_dataset['Bedrooms']<1]=1
3  Housing_dataset['Full Bath'][Housing_dataset['Full Bath']>=3]=3
4  Housing_dataset['Full Bath'][Housing_dataset['Full Bath']<1]=1
```

Figure 20: Cap the number of Bathrooms and Bedrooms

In the above figure, I am capping the number of bathrooms to a maximum of three and a minimum of one.
Likewise, the bedrooms have been capped at a maximum of five and a minimum of one in a house. Before fitting the minimum capping, I saw that there were some extra data where zero bathrooms and zero bedrooms were also fitted with the data source, and I removed those by adding the minimum capping.

We explore the heat map later to understand the linear relation of the variables.

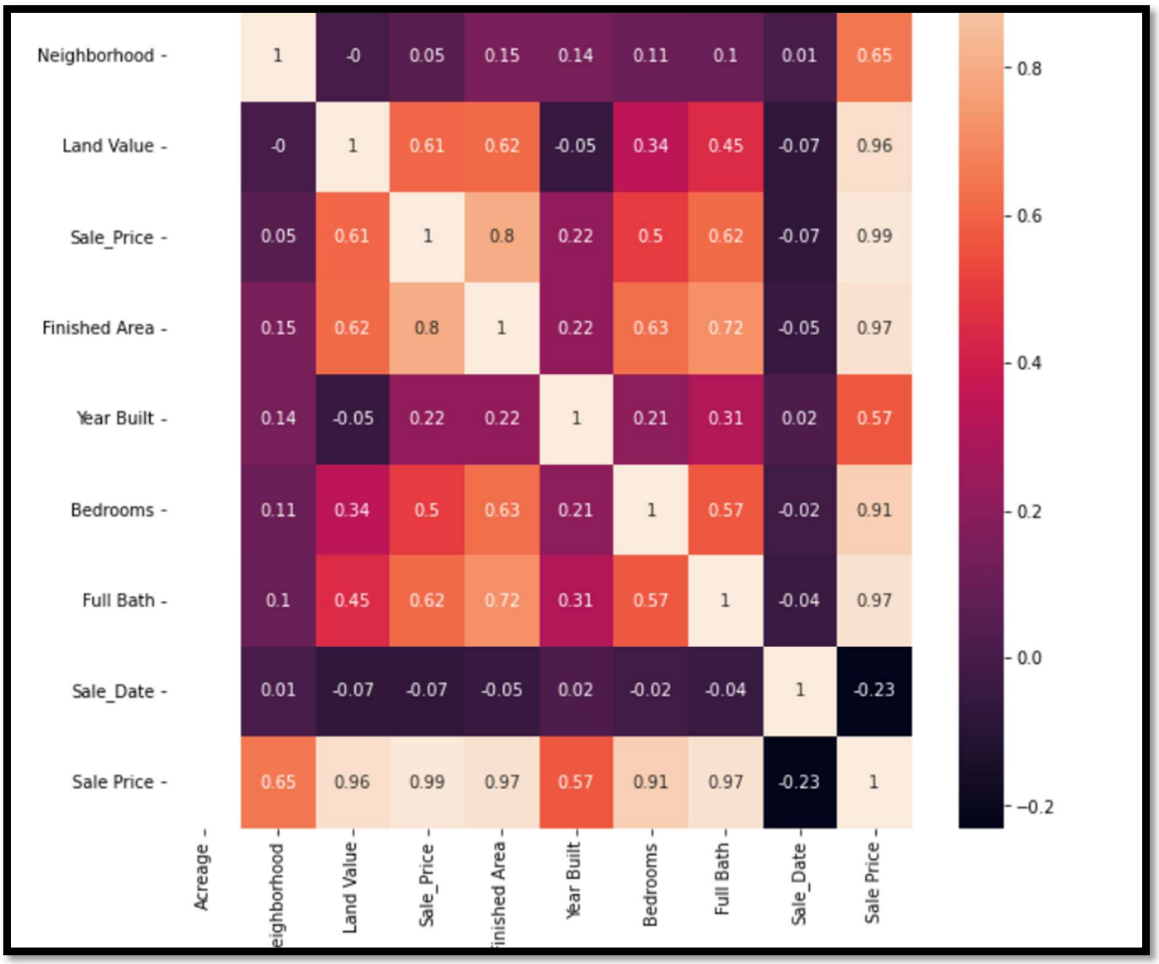


Figure 21: Correlation Plot

Finished Area and Sale_Price (Built Value) show a correlation of 0.8.

Neighborhood shows a moderate relationship of 0.65 with the property being sold as under the value or over prices. Similarly, Land Value shows a very high relation with the target variable to the tune of 0.96

A higher correlation can help to decide which variable I want to keep and which I want to drop for the classification. (How Can One Interpret a Heat Map Plot, 2019)

However, I will be dropping the Built value column (Renamed to Sale_Price). The purpose of having a numerical column was to understand the real estate market and the relation the price may hold with the several parameters that have been given to us.

Test and Train the Model

```
#Fit the columns for test and train
y=new_encoded_data[['Sale Price Compared To Value']]
x=new_encoded_data.drop('Sale Price Compared To Value',axis=1)
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3, random_state = 0)
x_train.shape,y_train.shape, x_test.shape, y_test.shape
```

```
((15855, 48), (15855, 1), (6796, 48), (6796, 1))
```

Figure 22: Train and test

We consider Sale Price Compared to Value as our Target variable for studying the classification and henceforth we tune our data accordingly. We train the model with 70% data and test it with 30% data.

We create the y parameter with the Sale Price Compared to Value Column and drop the same while passing the Dataframe new_encoded_data to the x parameter.

Part 1: Linear Regression

We have been asked to perform a linear Regression Model, but the target variable is categorical – Over and Under. I will have to convert it to zeros and ones before I can execute any model over the dataset. The conversion into Binary Values makes me go ahead with logistic regression rather than exploring the Linear Regression approach.

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(lm, x_test, y_test)
plt.show()
```

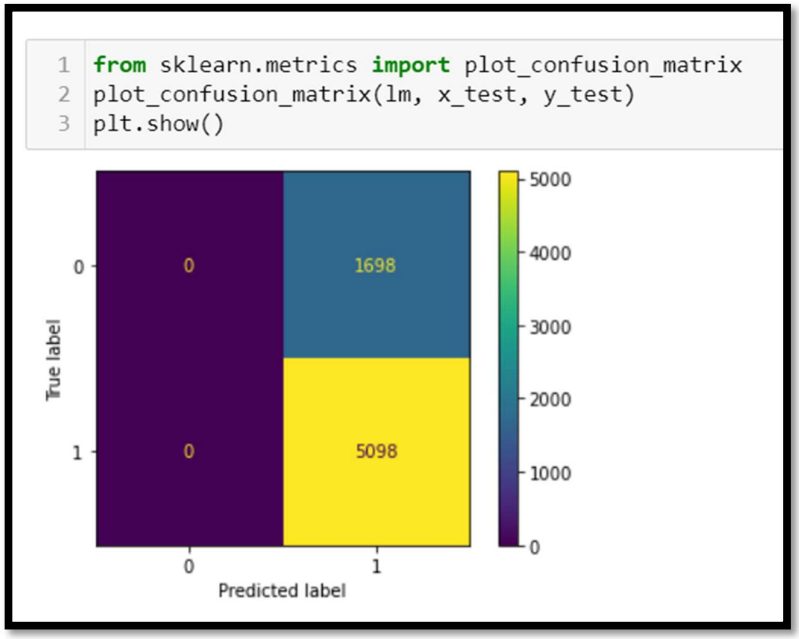


Figure 23: Confusion Matrix for Linear Regression

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1698
1	0.75	1.00	0.86	5098
accuracy			0.75	6796
macro avg	0.38	0.50	0.43	6796
weighted avg	0.56	0.75	0.64	6796

Figure 24: Precision and Recall for Linear Regression Model

The figure above does not give us proper precision and recall.

After trying to fit a linear regression model and failing a couple of times, I realized that this model is meant for classification and not linear regression specifically because the target variable is binary. Furthermore, in the code, I have converted the values into 0's and 1's so that it is easy for the system to run the various algorithms.

The above results are not very conclusive since the confusion matrix shows that the True Positives are 0 and the True positives are 5098. I am trying the other models since I am not very happy with the confusion matrix results.

Part 2: Support Vector Machine

We import the SVC from sklearn.SVM to test the Support Vector Machine Model Next.

```
[125]:  1  svc.score(x_test, y_test)
[125]: 0.7501471453796351
```

Figure 25: Support Vector Machine Model

```
[[   0 1698]
 [   0 5098]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1698
           1       0.75      1.00      0.86      5098

    accuracy                           0.75      6796
   macro avg       0.38      0.50      0.43      6796
weighted avg       0.56      0.75      0.64      6796
```

Figure 26: Support Vector Machine Model

The above model makes use of the Support Vector Machine Model. The accuracy of the model comes to 75%. The precision and the Recall are still not very conclusive; hence we explore further ahead.

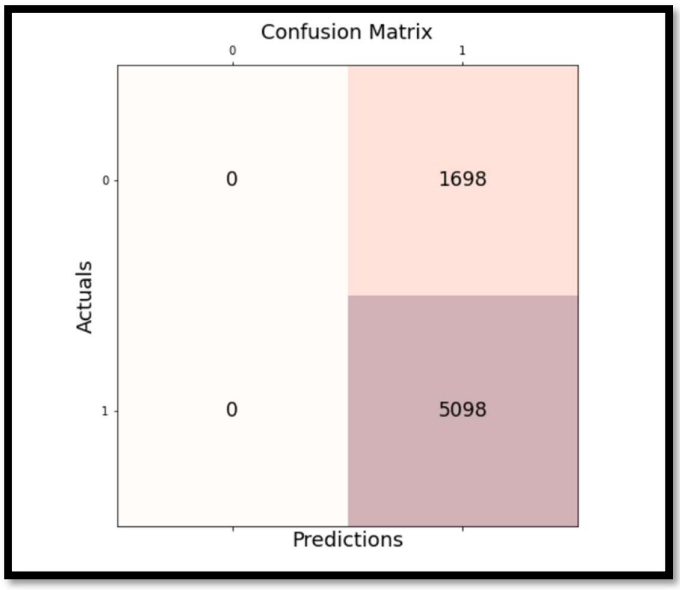
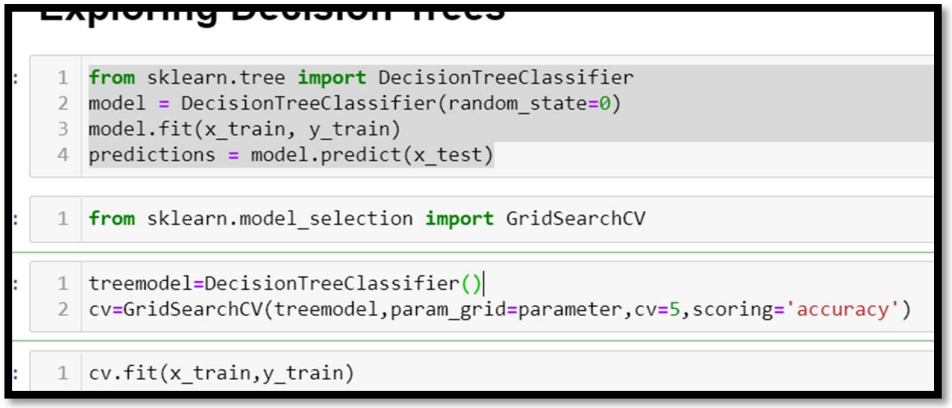


Figure 27: Confusion Matrix for Support Vector Machine Model

The above Support Vector Machine Model confusion matrix is not very promising, and I explore other methods for classifying the dataset.

Part 3 Decisions Trees



```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=0)
model.fit(x_train, y_train)
predictions = model.predict(x_test)
```

```
from sklearn.model_selection import GridSearchCV
```

```
treemodel=DecisionTreeClassifier()
cv=GridSearchCV(treemodel,param_grid=parameter,cv=5,scoring='accuracy')
```

```
cv.fit(x_train,y_train)
```

Figure 28: Importing Decision Tree Classifier

We import the decision tree classifier from sklearn.tree and also the GridsearchCV() function here. The cross-validation score is put as 5. This means that our sample will be divided into groups of 5 cross-validation points. Furthermore, we fit the same as our train and test sample.

Then we performed pre-pruning of the decision tree because when a decision tree is developed to its depth, we tend to encounter an overfitting problem. As a result, we will be doing pre prunning of the decision tree to get rid of it and to choose the model with the greatest accuracy.

```
{'criterion': 'entropy',
 'max_depth': 2,
 'max_features': 'sqrt',
 'splitter': 'best'}
```

Figure 29: Parameter- Entropy, Max depth

In the figure above the parameters are given for training the model.

```
46]: 1 from sklearn.metrics import accuracy_score, classification_report
     2 score=accuracy_score(y_pred,y_test)
     3 score

46]: 0.753825779870512
```

Figure 30: Accuracy for Decision trees

The above figure shows us that the accuracy is around 75% for the Decision Trees.

	precision	recall	f1-score	support
0	0.02	0.88	0.03	33
1	1.00	0.75	0.86	6763
accuracy			0.75	6796
macro avg	0.51	0.82	0.45	6796
weighted avg	0.99	0.75	0.85	6796

Figure 31: Precision and Recall for Decision Tree

The above figure looks a bit more promising with a better recall score of 88%. It tells us that the true positives will be predicted more accurately.
The precision is still not very promising since it is a prominent indicator of accuracy.

```
decision_tree_imp = pd.DataFrame({'Feature Names':x.columns, 'Importances':model.feature_importances_})
decision_tree_imp.sort_values("Importances", ascending=False)
```

	Feature Names	Importances
48	Sale Price Compared To Value_Under	1.0
25	Foundation Type_PIERS	0.0
27	Foundation Type_SLAB	0.0
28	Foundation Type_TYPICAL	0.0
29	Exterior Wall_BRICK	0.0
30	Exterior Wall_BRICK/FRAME	0.0
31	Exterior Wall_CONC BLK	0.0
32	Exterior Wall_FRAME	0.0
33	Exterior Wall_FRAME/STONE	0.0
34	Exterior Wall_LOG	0.0
35	Exterior Wall_METAL	0.0
36	Exterior Wall_STONE	0.0
37	Exterior Wall_STUCCO	0.0
38	Grade_A	0.0
39	Grade_AAB	0.0
40	Grade_B	0.0

Figure 32: Feature Selection

The feature significance of each variable is shown in the above fig. As seen in the above chart, the main features that determine whether a home is overpriced or under-priced are the Building Value, Finished Area, and Year Built.

```
55]:
paramsgrid = {
    'n_estimators': parameters,
    'max_depth': max_depth
}
paramsgrid

55]: {'n_estimators': [5000, 6000, 7000], 'max_depth': [3, 4]}

58]:
num_combinations = 1
for i in paramsgrid.keys(): num_combinations *= len(paramsgrid[i])

print('Number of combinations = ', num_combinations)
paramsgrid

Number of combinations =  6

58]: {'n_estimators': [5000, 6000, 7000], 'max_depth': [3, 4]}
```

Figure 33: Selecting parameters

Then we created a Random Forest model by selecting a few parameters,
parameters = [5000,6000,7000]
max_depth = [3,4]

We trained the model based on the above parameters and the best parameters for the model were' max_depth': 4, and 'n_estimators': 5000 with an accuracy of 71%.
Then we determined the feature importance of each variable,

Part 4: Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rf = GridSearchCV(estimator=RandomForestClassifier(),
                          param_grid=paramsgrid)

rf.fit(x_train, y_train)
```

Figure 34: Importing Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(class_weight='balanced',
                                    criterion='gini',
                                    max_depth=25,
                                    max_features='sqrt',
                                    min_samples_leaf=0.005,
                                    min_samples_split=0.005,
                                    n_estimators=10)
model.fit(x_train, y_train)
pred = model.predict(x_test)
from sklearn import metrics
print()
print("ACCURACY OF THE RF(tuned) MODEL: ", metrics.accuracy_score(y_test, pred))
```

```
ACCURACY OF THE RF(tuned) MODEL:  0.9995585638610948
```

Figure 35: Random Forest Classification

The Random Forest Classifier gives us an accuracy of 99%. We fit the criteria of Gini and give balanced weight classifications.

The depth of each tree in the forest is represented by max depth. The more splits a tree has, the more information it can collect about the data. We plot the training and test errors after fitting each decision tree with a depth ranging up to 25. (Fraj, 2018)

So far Random Forest Classifier has given a good accuracy percentage for my model as compared to the rest of the models that I have fit above.

Next, we try to set up a gradient Boosting Model to this dataset and understand which model is the best.

Part 5: Gradient Boosting Model

Gradient Boosting Method

```
from sklearn.ensemble import GradientBoostingClassifier
GradientBoosting_Classifier=GradientBoostingClassifier()
GradientBoosting_Classifier.fit(x_train, y_train)
y_train_pred = GradientBoosting_Classifier.predict(x_trair
print(metrics.accuracy_score(y_train, y_train_pred))
y_test_pred=GradientBoosting_Classifier.predict(x_test)
print(metrics.accuracy_score(y_test, y_test_pred))
```

Figure 36: Gradient Boosting Model

```
y_train_prediction_results = gbm.predict_proba(x_train)[:,1]
y_test_prediction_results = gbm.predict_proba(x_test)[:,1]
```

```
d_gbm = pd.DataFrame({'Feature Names':x.columns, 'Importances':gbm.feature_importances_})
d_gbm.sort_values("Importances", ascending=False)
```

	Feature Names	Importances
48	Sale Price Compared To Value_Under	9.390793e-01
7	Sale_Date	3.132392e-02
12	Sold As Vacant_No	4.908793e-03
2	Sale_Price	4.137092e-03
13	Sold As Vacant_Yes	3.939665e-03
1	Land Value	3.937396e-03
3	Finished Area	3.117210e-03
0	Acreage	2.627078e-03
5	Full Bath	1.045383e-03
41	Grade_C	8.817192e-04
6	Half Bath	7.343802e-04
4	Bedrooms	4.728508e-04
38	Grade_A	4.215029e-04

Figure 37: Gradient Boosting Feature Selection

In the above figure, we find that features are giving us good values and indicating which feature we can consider for model fitting.

Conclusion

- The dataset has a categorical target variable which must be converted to the binary value of 0 and 1 to fit the various models.
- The dataset has a lot of outliers, and it must be cleaned using the IQR method.
- The most important factors to consider for a good-value property, according to our analysis and models, are building value, land value, finished area, and year built. Before making an investment in the property, we would advise the corporation to focus on the building value, land value, finished area, and year of construction.
- On a comparative study, I find that Random Forest Classifier has given me the best accuracy of 99% over the rest of the models and I would like to use this option for finding the best real estate properties.

Model	Accuracy
Logistic Regression	75%
Support Vector Machine	75%
Decision Tree	75%
Random Forest Classifier	99%
Gradient Boosting	75%

Figure 38: Comparative Study between the Various Models

References

Bukovac, J. (2022, September 13). *Home prices are starting to decrease, but what does this mean?* https://www.wsmv.com. Retrieved October 20, 2022, from https://www.wsmv.com/2022/09/13/home-prices-are-starting-decrease-what-does-this-mean/

How can one interpret a heat map plot? (2019, February 14). Cross Validated. Retrieved October 21, 2022, from https://stats.stackexchange.com/questions/392517/how-can-one-interpret-a-heat-map-plot'\

V, L. G. (2022b, August 2). *4 Ways to Evaluate your Machine Learning Model: Cross-Validation Techniques (with Python code)*. Analytics Vidhya. Retrieved October 21, 2022, from https://www.analyticsvidhya.com/blog/2021/05/4-ways-to-evaluate-your-machine-learning-model-cross-validation-techniques-with-python-code/

Fraj, M. B. (2018, August 14). *In-Depth: Parameter tuning for Random Forest - All things AI*. Medium. Retrieved October 21, 2022, from https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d

Appendix

```
  #Reading the libraries of the date
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Reading the Files

```
#Reading the CSV Files
Housing_dataset=pd.read_csv("week 4 - Nashville_housing_data.csv")


Housing_dataset.shape
#Reading the top five
Housing_dataset.head(n=5)
```

Info about the Housing Data Set

```
Housing_dataset.info()
```

Reading the Summary of the Dataset

```
Housing_dataset.describe()
Housing_dataset
```

Checking for Duplicates

```
# Checking for the duplicated values

Housing_dataset.duplicated()

#As we can see the values are not duplicated
```

Checking for Null Values

```
#Checking for Null Values
Housing_dataset.isnull()
#From the dataset we can see that the column Suite/Condo shows that it has null values
```

Checking the Columns

```
#checking the Columns of the Dataset
Housing_dataset.columns
#Dropping the Column that has Null Values and the unwanted columns
Housing_dataset=Housing_dataset.drop(['Unnamed: 0', 'Parcel ID','Suite/ Condo   #'],axis=1)
#Renaming the Building Value into Sale Price
Housing_dataset.rename(columns={'Building Value':'Sale_Price'},inplace=True)
#Checking the Column names
Housing_dataset.columns
#Converting the Date into Year Like Format and Creating a New column all together
import datetime
```

```
type(Housing_dataset['Sale Date'])
Housing_dataset['Sale Date'] = pd.to_datetime(Housing_dataset['Sale Date'])
# add a column for Year
Housing_dataset['Sale_Date'] = Housing_dataset['Sale Date'].dt.year
Housing_dataset['Sale Date'] = Housing_dataset['Sale Date'].dt.strftime('%m-%d-%Y')
Housing_dataset['Sale_Date']
```

```
Housing_dataset.columns
```

```
#Checking a Boxplot for Checking which city has the highest value of Real Estate by Built Value
sns.barplot(x = 'City',y = 'Sale_Price',data = Housing_dataset)
plt.xticks(rotation = 45)
plt.show()
# from the graph we see that Brentwood has got amazing
```

```
#Checking to see if the real estate price is over values or under valued as compared to the house price
sns.barplot(x = 'Sale Price Compared To Value',y = 'Sale_Price',data = Housing_dataset)
plt.xticks(rotation = 45)
plt.show()
```

```
#Making histogram for the Sale Price
plt.hist(Housing_dataset['Sale_Price'], bins=10,color="plum")
```

```
# Sale Price afer applying the logarithmic function
Housing_dataset['Sale_Price'] = np.log(Housing_dataset['Sale_Price'])
plt.hist(Housing_dataset['Sale_Price'], bins=20,color="lightblue")
```

```
Housing_dataset['Land Use'].value_counts().plot(kind='barh',color="pink")
```

```
Housing_dataset['Property City'].value_counts().plot(kind='barh',color='lightgreen')
```

```
table = pd.pivot_table(Housing_dataset,index=['Foundation Type'],values=['Sale_Price'], aggfunc=np.sum)
table
table.plot(kind='bar',color="wheat")
```

```
plt.figure(figsize=[10,10])
sns.barplot(x='Grade',y='Sale_Price', data=Housing_dataset)
plt.xlabel('Grade')
plt.ylabel('Sale Price')
# Adding the legends
plt.show()
```

```
Housing_dataset.isnull().sum()
```

```
Feature = [x for x in Housing_dataset.columns if Housing_dataset[x].dtypes != 'O']
```

```
Feature
```

```
#sns.boxplot(x=Housing_dataset["Land Value"])
```

```
for x in Feature:
    sns.boxplot(Housing_dataset[x])
    plt.title(Feature)
    plt.figure(figsize=(15,15))
```

```
IQR1=Housing_dataset['Sale_Price'].quantile(0.75)-
Housing_dataset['Sale_Price'].quantile(0.25)
```

```
lower_bridge1=Housing_dataset['Sale_Price'].quantile(0.25)-(IQR1*1.5)
upper_bridge1=Housing_dataset['Sale_Price'].quantile(0.75)+(IQR1*1.5)
print(lower_bridge1, upper_bridge1)

Housing_dataset.loc[Housing_dataset['Sale_Price']>=14.15,'Sale Price']=14.15
Housing_dataset.loc[Housing_dataset['Sale_Price']<=8.9885,'Sale Price']=8.9885

IQR2=Housing_dataset['Acreage'].quantile(0.75)-Housing_dataset['Acreage'].quantile(0.25)
lower_bridge2=Housing_dataset['Acreage'].quantile(0.25)-(IQR2*1.5)
upper_bridge2=Housing_dataset['Acreage'].quantile(0.75)+(IQR2*1.5)
print(lower_bridge2,upper_bridge2)

Housing_dataset.loc[Housing_dataset['Acreage']>=-0.32,'Acreage']=0.32
Housing_dataset.loc[Housing_dataset['Acreage']<-0.58,'Acreage']=-0.58

Housing_dataset['Land Value'].describe()

#Housing_dataset.describe()

IQR3=Housing_dataset['Land Value'].quantile(0.75)-Housing_dataset['Land
Value'].quantile(0.25)
lower_bridge3=Housing_dataset['Land Value'].quantile(0.25)-(IQR3*1.5)
upper_bridge3=Housing_dataset['Land Value'].quantile(0.75)+(IQR3*1.5)
print(upper_bridge3,lower_bridge3)

Housing_dataset.loc[Housing_dataset['Land Value']>=117750.0,'Land Value']=117750
Housing_dataset.loc[Housing_dataset['Land Value']<-35450.0,'Land Value']=-35450.0

IQR4=Housing_dataset['Finished Area'].quantile(0.75)-Housing_dataset['Finished
Area'].quantile(0.25)
lower_boundary4=Housing_dataset['Finished Area'].quantile(0.25)-(IQR4*1.5)
upper_boundary4=Housing_dataset['Finished Area'].quantile(0.75)+(IQR4*1.5)
print(lower_boundary4,upper_boundary4)

Housing_dataset.loc[Housing_dataset['Finished Area']>=3658.4375,'Finished
Area']=3658.4375
Housing_dataset.loc[Housing_dataset['Finished Area']<-195.0625,'Finished Area']=-
195.0625

#Checking for Null Values
Housing_dataset.isnull().sum()*100/len(Housing_dataset)

Housing_dataset['Bedrooms'].value_counts()

Housing_dataset['Full Bath'].value_counts()

Housing_dataset['Bedrooms'][Housing_dataset['Bedrooms']>=5]=5
Housing_dataset['Bedrooms'][Housing_dataset['Bedrooms']<1]=1
Housing_dataset['Full Bath'][Housing_dataset['Full Bath']>=3]=3F
Housing_dataset['Full Bath'][Housing_dataset['Full Bath']<1]=1

# Checks the maximum value of the columns
np.max(Housing_dataset)

Housing_dataset['Finished Area'].fillna((Housing_dataset['Finished Area'].mean()),
inplace=True)

Housing_dataset['Bedrooms'].fillna((Housing_dataset['Bedrooms'].mean()), inplace=True)
Housing_dataset['Half Bath'].fillna((Housing_dataset['Half Bath'].mean()), inplace=True)
Housing_dataset['Full Bath'].fillna((Housing_dataset['Full Bath'].mean()), inplace=True)
```

```
Housing_dataset.isnull().sum()

Housing_dataset=Housing_dataset.drop(['Half Bath'],axis=1)

Housing_dataset.describe()

plt.figure(figsize=[10,10])
matrix = Housing_dataset.corr().round(2)
sns.heatmap(matrix, annot=True)
plt.show()

# The above correlation plot shows that Sale Price has a positive correlation with the Finished Area(0.79)
Housing_dataset.head()
```

Dropping Columns

```
Housing_dataset=Housing_dataset.drop(['Property City','Property Address','Sale Date','City','Legal Reference','State','Neighborhood','Sale Price'], axis=1)


new_encoded_data= pd.get_dummies(Housing_dataset, columns=['Land Use','Sold As Vacant','Multiple Parcels Involved in Sale', 'Tax District', 'Foundation Type',
                                 'Exterior Wall', 'Grade'])

new_encoded_data['Sale Price Compared To Value']=new_encoded_data['Sale Price Compared To Value'].replace("Under",0).replace("Over",1)

new_encoded_data

#Fit the columns for test and train
y=new_encoded_data[['Sale Price Compared To Value']]
x=new_encoded_data.drop('Sale Price Compared To Value',axis=1)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3, random_state = 0)
x_train.shape,y_train.shape, x_test.shape, y_test.shape

#logistic Regressiomn
from sklearn import linear_model
lm = linear_model.LogisticRegression(multi_class='ovr', solver='liblinear')
lm.fit(x_train, y_train)

lm.score(x_test, y_test)

from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(lm, x_test, y_test)
plt.show()

#Recall and Precision

from sklearn import metrics
print(metrics.classification_report(y_test, lm.predict(x_test)))

#Importing SVC function
from sklearn.svm import SVC

x.info() # Checking the Data Type of X and understanding the data types present in X

y.info() # Checking what information is stored in Y and understanding the data types present in y
```

Support Vector Function Model

```
svc = SVC()
svc.fit(x_train, y_train)

print(metrics.accuracy_score(y_test, predictions))

svc.score(x_test, y_test)

y_pred = svc.predict(x_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
#
# Print the confusion matrix using Matplotlib
#
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Reds, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()

parameter={
 'criterion':['gini','entropy','log_loss'],
  'splitter':['best','random'],
  'max_depth':[1,2,3,4,5],
  'max_features':['auto', 'sqrt', 'log2']

}
```

Exploring Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=0)
model.fit(x_train, y_train)
predictions = model.predict(x_test)

from sklearn.model_selection import GridSearchCV

treemodel=DecisionTreeClassifier()
cv=GridSearchCV(treemodel,param_grid=parameter,cv=5,scoring='accuracy')
```

```
cv.fit(x_train,y_train)

cv.best_params_

y_pred=cv.predict(x_test)

from sklearn.metrics import accuracy_score, classification_report
score=accuracy_score(y_pred,y_test)
score

print(classification_report(y_pred,y_test))

# Performing the feature importance to check if the Features are important or not
decision_tree_imp = pd.DataFrame({'Feature Names':x.columns,
'Importances':model.feature_importances_})
decision_tree_imp.sort_values("Importances", ascending=False)

parameters = [5000,6000,7000]
max_depth = [3,4]

paramsgrid = {
   'n_estimators': parameters,
   'max_depth': max_depth
}
paramsgrid

num_combinations = 1
for i in paramsgrid.keys(): num_combinations *= len(paramsgrid[i])

print('Number of combinations = ', num_combinations)
paramsgrid
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn. model_selection import GridSearchCV
rf = GridSearchCV(estimator=RandomForestClassifier(),
                  param_grid=paramsgrid)

rf.fit(x_train, y_train)

rf.best_params_

rf1 = pd.DataFrame({'Feature Names':x.columns,
'Importances':model.feature_importances_})
rf1.sort_values ("Importances", ascending=False)
```

Fitting the Random Forest Classifier Model

```
 from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(class_weight='balanced',
                               criterion='gini',
                               max_depth=25,
                               max_features='sqrt',
                               min_samples_leaf=0.005,
                               min_samples_split=0.005,
                               n_estimators=10)
model.fit(x_train, y_train)
pred = model.predict(x_test)
from sklearn import metrics
print()
print("ACCURACY OF THE RF(tuned) MODEL: ", metrics.accuracy_score(y_test, pred))
```

Gradient Boosting Method

```
from sklearn.ensemble import GradientBoostingClassifier
GradientBoosting_Classifier=GradientBoostingClassifier()
GradientBoosting_Classifier.fit(x_train, y_train)
y_train_pred = GradientBoosting_Classifier.predict(x_train)
print(metrics.accuracy_score(y_train, y_train_pred))
y_test_pred=GradientBoosting_Classifier.predict(x_test)
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
gbm = GradientBoostingClassifier(n_estimators=5000,
                               learning_rate=0.05,
                               max_depth=3,
                               subsample=0.5,
                               validation_fraction=0.1,
                               n_iter_no_change=20,
                               max_features='log2',
                               verbose=1)
gbm.fit(x_train, y_train)


y_train_prediction_results = gbm.predict_proba(x_train)[:,1]
y_test_prediction_results = gbm.predict_proba(x_test)[:,1]


d_gbm = pd.DataFrame({'Feature Names':x.columns,
'Importances':gbm.feature_importances_})
d_gbm.sort_values("Importances", ascending=False)


# Fitting a neural network model lastly to understand if this model will be good or not
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(3, solver='sgd',learning_rate_init = 0.01, max_iter = 15000)
mlp.fit(x_train, y_train)
mlp.score(x_test, y_test)
```