Predictive Analytics

ALY 6020, CRN 80405

Professor Vladimir Shapiro

Predictive Analytics - Module 5 Assignment

Submitted By - Richa Umesh Rambhia

Module 5 Assignment - Investing in Nashville

Table of Contents

Introduction

Logistic Regression Model

Logistic Regression is a *statistical type of machine learning model* that helps in the **classification and predictive analytics** in order to estimate the probability and likelihood of an event occuring. [1] The difference between a Linear Regression model and a Logistic Regression model is that Linear Regression algorithms are used to identify relationships between a continuous dependent variable and one or more independent variable, whereas Logistic Regression models are used to **predict a category based on categorical type** variable versus the continuous data points.

The different types of Logistic Regression algorithms which are defined based on the categorical data points are as follows.

1. **Binary Logistic Regression**
2. **Multinomial Logistic Regression**
3. **Ordinal Logistic Regression**

Decision Tree Model

Decision Tree algorithm is a powerful tool for **classification and regression** based problems, which is a **flowchart-like tree structure** where the internal node denotes a test on the attribute, each branch represents the outcome of the test, and each leaf node (terminal node) represents a class label. Decision Tree algorithms are a tree-like model that helps in decision making that is followed by a sequence of if-else questions about the various input data until a prediction is made. [2]

Random Forest Model

Random Forest model is a machine learning algorithm which uses the **ensemble of decision trees** in order to make predictions and is a powerful algorithm which can be used as **both a classification algorithm or a regression algorithm.** [3] This algorithm is trained on multiple decision trees and on different subsets of the training data which uses random subsets of the features.

The main advantage of using a Random Forest Classifier is the ability of the algorithm to **handle a large number of features** and parameters along with large datasets. It is also **less prone to overfitting** as compared to other machine learning algorithms and models.

Gradient Boost Model : XGBoost Algorithm

XGBoost model is a machine learning algorithm and a **type of the Gradient Boost** model which uses a **regularized model** and has better model performance as compared to any traditional gradient boost algorithms. [4] This algorithm **builds an ensemble of weak models**, which typically is the Decision Tree models and **reduces the risk of overfitting** to improve the generalization performance.

The model also uses the gradient boosting technique to **optimize the model** such that it involves minimizing a loss function which is done by adding weak models that are good at prediction of the previous models.

XGBoost model also provides other features which prove it to be a better model when compared to others. The advanced features provided by the model are *handling missing values, regularization, parallel processing, tree pruning*, etc. [4]

Neural Network Model

Neural Network model is a type of machine learning algorithm which is designed in a way to **mimic the behavior of the human brain** and is composed of layers of the **interconnected nodes or neurons** in order to process and transmit the information. These algorithms are widely used for a variety of machine learning tasks in different applications. [5]

Problem Statement

You just started working for a real estate company and they are looking to make a huge investment into the growing Nashville area. They've acquired a dataset about recent sales and want you to build a model to help them accurately find the best value deals when they go to visit next week. There is a concern that houses are going over their asking price and this dataset will help us observe that.

Hint: You will have to create the dependent variable to understand whether it is over/under price (you can have multiple categories but remember the limitations of logistic vs decision tree type models).

Investing in Nashville - Housing Data Analysis

Investing in Nashville analysis is based on the **housing dataset** which has over 31 different features and parameters in order to understand and analyze the price ranges of the house that are classified as under price or over price. This will help in recommending the company what parameters are **influencing the price of the house** and which range of parameters are under priced or over priced based on the sale price and the total value of the house.

The housing dataset has about **556636 rows of data points and 31 field values** where the goal is to build different classification models based on the target variable to **classify the house price range as under price or over price**. The various factors that affect the response parameter can be further analyzed to understand the parameters of the dataset that are affecting and contributing to the target variable.

Analysis

Task 1:

Use proper data cleansing techniques to ensure that you have the highest quality data to model this problem. Detail your process and discuss the decisions you made to clean the data.

Task 2:

Build a Logistic Regression model to accurately identify overpricing/underpricing and determine what is driving those prices.

Task 3:

Build a Decision Tree model and compare the results with the results of the previous model.

Task 4:

Build a Random Forest model and compare the results with the results of the previous models.

Task 5:

Build a Gradient Boost model and compare the results with the results of the previous models.

Task 6:

Build a Neural Network model and compare the results with those of the previous model.

Task 7:

Use multiple benchmarking metrics to compare and contrast the five models. Based on your findings, provide evidence of which model you believe the real estate company should use and what are the key variables to focus on to drive value and how can they get the most value out of the houses they should be targeting.

Installing required packages

In [89]:
```
!pip install pandas_profiling
!pip install featurewiz
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-w
heels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting pandas_profiling
  Downloading pandas_profiling-3.6.6-py2.py3-none-any.whl (324 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 324.4/324.4 kB 11.6 MB/s eta 0:00:00
Collecting ydata-profiling (from pandas_profiling)
  Downloading ydata_profiling-4.1.2-py2.py3-none-any.whl (345 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 345.9/345.9 kB 41.1 MB/s eta 0:00:00
Collecting scipy<1.10,>=1.4.1 (from ydata-profiling->pandas_profiling)
  Downloading scipy-1.9.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (33.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 33.7/33.7 MB 41.3 MB/s eta 0:00:00
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in /usr/local/lib/python3.10/dist-packages (from y
data-profiling->pandas_profiling) (1.5.3)
Collecting matplotlib<3.7,>=3.2 (from ydata-profiling->pandas_profiling)
  Downloading matplotlib-3.6.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 95.7 MB/s eta 0:00:00
Requirement already satisfied: pydantic<1.11,>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from yda
ta-profiling->pandas_profiling) (1.10.7)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ydata-
```

In [2]:
```
!pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.12.0-cp38-cp38-win_amd64.whl (1.9 kB)
Collecting tensorflow-intel==2.12.0
  Downloading tensorflow_intel-2.12.0-cp38-cp38-win_amd64.whl (272.8 MB)
     ---------------------------------------- 272.8/272.8 MB 2.0 MB/s eta 0:00:00
Requirement already satisfied: packaging in c:\users\rramb\appdata\local\programs\python\python38\lib\sit
e-packages (from tensorflow-intel==2.12.0->tensorflow) (23.0)
Collecting protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
  Downloading protobuf-4.23.0-cp38-cp38-win_amd64.whl (422 kB)
     ---------------------------------------- 422.5/422.5 kB 9.0 MB/s eta 0:00:00
Collecting wrapt<1.15,>=1.11.0
  Using cached wrapt-1.14.1-cp38-cp38-win_amd64.whl (35 kB)
Collecting tensorflow-estimator<2.13,>=2.12.0
  Downloading tensorflow_estimator-2.12.0-py2.py3-none-any.whl (440 kB)
     ---------------------------------------- 440.7/440.7 kB 9.2 MB/s eta 0:00:00
Requirement already satisfied: numpy<1.24,>=1.22 in c:\users\rramb\appdata\roaming\python\python38\site-p
ackages (from tensorflow-intel==2.12.0->tensorflow) (1.22.4)
Collecting libclang>=13.0.0
  Downloading libclang-16.0.0-py2.py3-none-win_amd64.whl (24.4 MB)
```

Importing libraries

In [5]:
```
import pandas as pd
import numpy as np
#import pandas_profiling
#import ydata_profiling
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import KNNImputer
#from featurewiz import featurewiz
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.utils.class_weight import compute_class_weight
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc, precision_score, recall_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import tensorflow as tf
from tensorflow import keras
```

Loading the dataset

In [6]:
```
housing_data = pd.read_csv("Nashville_housing_data_2013_2016.csv")
housing_data
```

Out[6]:

	Unnamed: 0.1	Unnamed: 0	Parcel ID	Land Use	Property Address	Suite/ Condo #	Property City	Sale Date	Sale Price	Legal Reference	...	Building Value	Tota Valu
0	0	0	105 03 0D 008.00	RESIDENTIAL CONDO	1208 3RD AVE S	8	NASHVILLE	2013-01-24	132000	20130128-0008725	...	NaN	Nal
1	1	1	105 11 0 080.00	SINGLE FAMILY	1802 STEWART PL	NaN	NASHVILLE	2013-01-11	191500	20130118-0006337	...	134400.0	168300.(
2	2	2	118 03 0 130.00	SINGLE FAMILY	2761 ROSEDALE PL	NaN	NASHVILLE	2013-01-18	202000	20130124-0008033	...	157800.0	191800.(
3	3	3	119 01 0 479.00	SINGLE FAMILY	224 PEACHTREE ST	NaN	NASHVILLE	2013-01-18	32000	20130128-0008863	...	243700.0	268700.(
4	4	4	119 05 0 186.00	SINGLE FAMILY	316 LUTIE ST	NaN	NASHVILLE	2013-01-23	102000	20130131-0009929	...	138100.0	164800.(
...	...	...	...	...	...	...	...	...	...	...	...	...	.
56631	56631	56631	093 13 0B 274.00	RESIDENTIAL CONDO	320 11TH AVE S	274.0	NASHVILLE	2016-10-06	210000	20161007-0106599	...	NaN	Nal
56632	56632	56632	093 13 0D 044.00	RESIDENTIAL CONDO	700 12TH AVE S	608.0	NASHVILLE	2016-10-25	338000	20161101-0115186	...	NaN	Nal
56633	56633	56633	093 13 0D 048.00	RESIDENTIAL CONDO	700 12TH AVE S	613.0	NASHVILLE	2016-10-04	742000	20161010-0106889	...	NaN	Nal
56634	56634	56634	093 13 0D 056.00	RESIDENTIAL CONDO	700 12TH AVE S	708.0	NASHVILLE	2016-10-26	320000	20161031-0114730	...	NaN	Nal
56635	56635	56635	093 13 0D 094.00	RESIDENTIAL CONDO	700 12TH AVE S	1008.0	NASHVILLE	2016-10-27	330000	20161104-0117077	...	NaN	Nal

56636 rows × 31 columns

Table 1. Housing Data for Investing in Nashville

Land Use: What was land used for

Sale Price: Sale price house

Sold As Vacant: Was anyone living in the house

Multiple Parcels Involved in Sale: Were multiple properties in sale

Acreage: How big is the lot

Tax District: Which district is the house in

Land Value: How much is land worth

Building Value: How much is building worth

Total Value: How much is total property worth

Finished Area: How much of the house is finished

Foundation Type: Self explanatory

Year Built: Self explanatory

Exterior Wall: Type

Grade: Grade that was given to condition of house

Bedrooms: Self explanatory

Full Bath: Self explanatory

Half Bath: Self explanatory

Figure 1. Data Dictionary for Housing Data Analysis

Step 1: Exploratory Data Analysis

EDA is performed on the data in order to analyze various parameters and features of the dataset and to understand the *structure* of the dataset such that various *trends and patterns* between the variables is known. Exploratory Data Analysis helps in understanding the *relationship between the various independent and dependent variables* of the dataset that would further be useful in building the model such as description analysis and statistical analysis.

Descriptive Analysis

In [112]:
```
# displaying number of rows and columns
print("Total number of Rows and Columns:", housing_data.shape)

print("\n-------------------------------------------------------")

# displaying field values/column names
print("\nColumn Names:\n")
housing_data.columns
```

```
Total number of Rows and Columns: (56636, 31)

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

Column Names:
```

Out[112]:
```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Parcel ID', 'Land Use',
       'Property Address', 'Suite/ Condo   #', 'Property City', 'Sale Date',
       'Sale Price', 'Legal Reference', 'Sold As Vacant',
       'Multiple Parcels Involved in Sale', 'Owner Name', 'Address', 'City',
       'State', 'Acreage', 'Tax District', 'Neighborhood', 'image',
       'Land Value', 'Building Value', 'Total Value', 'Finished Area',
       'Foundation Type', 'Year Built', 'Exterior Wall', 'Grade', 'Bedrooms',
       'Full Bath', 'Half Bath'],
      dtype='object')
```

In [113]:
```python
# displaying data types
print("Data types:\n")
housing_data.dtypes
```

```
Data types:
```

Out[113]:
```
Unnamed: 0.1                         int64
Unnamed: 0                           int64
Parcel ID                           object
Land Use                            object
Property Address                    object
Suite/ Condo   #                    object
Property City                       object
Sale Date                           object
Sale Price                           int64
Legal Reference                     object
Sold As Vacant                      object
Multiple Parcels Involved in Sale   object
Owner Name                          object
Address                             object
City                                object
State                               object
Acreage                            float64
Tax District                        object
Neighborhood                       float64
image                               object
Land Value                         float64
Building Value                     float64
Total Value                        float64
Finished Area                      float64
Foundation Type                     object
Year Built                         float64
Exterior Wall                       object
Grade                               object
Bedrooms                           float64
Full Bath                          float64
Half Bath                          float64
dtype: object
```

Table 2. Data types for Housing Data

From the *descriptive analysis*, it is observed that there are total **56636 rows of data** and **31 field values** and the data type for each of the field value is displayed in order to understand what data type values are present in the dataset.

Here, there are different types of data points that are present in the dataset which are **numerical data type** having either *'int' or 'float'* values, along with **object data type** which further needs to be corrected to *string or category* type of data and *int or float* datatype as per the data type requirement.

Statistical Analysis

In [114]:
```python
# dataset info
print("Dataset Info:\n")
housing_data.info()
```

```
Dataset Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56636 entries, 0 to 56635
Data columns (total 31 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Unnamed: 0.1                       56636 non-null  int64
 1   Unnamed: 0                         56636 non-null  int64
 2   Parcel ID                          56636 non-null  object
 3   Land Use                           56636 non-null  object
 4   Property Address                   56477 non-null  object
 5   Suite/ Condo   #                   6109 non-null   object
 6   Property City                      56477 non-null  object
 7   Sale Date                          56636 non-null  object
 8   Sale Price                         56636 non-null  int64
 9   Legal Reference                    56636 non-null  object
 10  Sold As Vacant                     56636 non-null  object
 11  Multiple Parcels Involved in Sale  56636 non-null  object
 12  Owner Name                         25261 non-null  object
 13  Address                            26017 non-null  object
 14  City                               26017 non-null  object
 15  State                              26017 non-null  object
 16  Acreage                            26017 non-null  float64
 17  Tax District                       26017 non-null  object
 18  Neighborhood                       26017 non-null  float64
 19  image                              25335 non-null  object
 20  Land Value                         26017 non-null  float64
 21  Building Value                     26017 non-null  float64
 22  Total Value                        26017 non-null  float64
 23  Finished Area                      24166 non-null  float64
 24  Foundation Type                    24164 non-null  object
 25  Year Built                         24165 non-null  float64
 26  Exterior Wall                      24165 non-null  object
 27  Grade                              24165 non-null  object
 28  Bedrooms                           24159 non-null  float64
 29  Full Bath                          24277 non-null  float64
 30  Half Bath                          24146 non-null  float64
dtypes: float64(10), int64(3), object(18)
memory usage: 13.4+ MB
```

Table 3. Information about the dataset

In [115]:
```python
# describing the dataset
print("Describing the dataset:\n")
round(housing_data.describe(),1)
```

```
Describing the dataset:
```

Out[115]:

	Unnamed: 0.1	Unnamed: 0	Sale Price	Acreage	Neighborhood	Land Value	Building Value	Total Value	Finished Area	Year Built	Bedrooms	
count	56636.0	56636.0	56636.0	26017.0	26017.0	26017.0	26017.0	26017.0	24166.0	24165.0	24159.0	24
mean	28317.5	28317.5	327211.1	0.5	4356.2	69072.7	160802.5	232397.1	1927.0	1963.7	3.1	
std	16349.5	16349.5	928742.5	1.6	2170.3	106040.5	206804.1	281070.3	1687.0	26.5	0.9	
min	0.0	0.0	50.0	0.0	107.0	100.0	0.0	100.0	0.0	1799.0	0.0	
25%	14158.8	14158.8	135000.0	0.2	3126.0	21000.0	75900.0	102800.0	1239.0	1948.0	3.0	
50%	28317.5	28317.5	205450.0	0.3	3929.0	28800.0	111400.0	148500.0	1632.0	1960.0	3.0	
75%	42476.2	42476.2	329000.0	0.4	6228.0	60000.0	180700.0	268500.0	2212.0	1983.0	3.0	
max	56635.0	56635.0	54278060.0	160.1	9530.0	2772000.0	12971800.0	13940400.0	197988.0	2017.0	11.0	

Table 4. Dataset Description

Statistical Analysis helps in understanding about each of the numerical field type based on the **total count values, minimum value, maximum value, standard deviation**, etc. giving an overall analysis of the field data points about the various rows present in the dataset.

For example, as observed in the dataset, we see that there are multiple field values having the *minimum, maximum values* along with the *total count of values* which is **56636** and *standard deviation* of the column values. It can be observed that the maximum value of *Sale Price* is **54278060.0** and the maximum value of *Neighborhood* is **9530.0**.

Thus, similarly, other parameters of the dataset can be analyzed based on their statistical values.

Data Profiling

In [96]:
```
housing_data_report = housing_data.profile_report(title='Housing Data Analysis Report', explorative = True)
housing_data_report
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[96]:

In [97]:
```
# Saving the profile report
housing_data_report.to_file(output_file="Housing Data Analysis Report.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

The data profiling report generated for the dataset helps in understanding various parameters such as the data type of the field values, the missing and duplicate values present in the dataset, the correlation between each of the field value, and the analysis of each of the field value on a individual basis based on correlation plot, histogram, and interaction graphs.

From the profiling report, it is observed that there are **12 numerical variable type, 17 categorical data type, and 2 boolean data type** field values present in the dataset of which the numerical data type have **integer and float values**, along with string and datetime objects. Apart from this, there are **missing values** present in the dataset, and the missing values visualization or plot also helps in understanding the same, for which the percent of missing values is **37% i.e., 648773 missing cells** are present in the dataset, which is quite a high percent. For each field value a separate visualization is also displayed in order to specifially analyze a particular field value, and there are **no duplicate values present in the dataset.**

It can also be observed that there is a constant data field present in the dataset; **'State'**, and each of the field value displays the *correlation with each other indicating overall correlation* along with any *imbalance data present in the column data.*

Further cleaning of the data is implemented in the below steps.

Step 2: Data Cleaning

Task 1: Use proper data cleansing techniques to ensure you have the highest quality data to model this problem. Detail your process and discuss the decisions you made to clean the data.

---

1. Dropping irrelevant columns or field values
2. Renaming the field values
3. Checking for null values in each column of the dataset, i.e., missing values
4. Dropping data values or fields having maximum null values
5. Replacing missing values using various imputation methods
6. Checking for incorrect data types in field values and correcting the data type of the column
7. Checking for outliers in the dataset

   a. *Boxplot*

   b. *Distribution Plot*

---

1. Dropping irrelevant columns or field values

In [7]:
```
housing_data = housing_data.drop(['Unnamed: 0.1', 'Unnamed: 0', 'image'], axis=1)
print("Dropped irrelevant columns.")
```

```
Dropped irrelevant columns.
```

In [117]:
```
housing_data.shape
```

Out[117]:
```
(56636, 28)
```

After dropping the irrelevant columns or field values, the total number of columns present for further analysis is **28 field values.**

2. Renaming the field values

In [8]:
```
housing_data = housing_data.rename(columns={'Suite/ Condo   #': 'Suite/Condo'})
print("Rename successful.")
```

```
Rename successful.
```

In [119]:
```
housing_data.columns
```

Out[119]:
```
Index(['Parcel ID', 'Land Use', 'Property Address', 'Suite/Condo',
       'Property City', 'Sale Date', 'Sale Price', 'Legal Reference',
       'Sold As Vacant', 'Multiple Parcels Involved in Sale', 'Owner Name',
       'Address', 'City', 'State', 'Acreage', 'Tax District', 'Neighborhood',
       'Land Value', 'Building Value', 'Total Value', 'Finished Area',
       'Foundation Type', 'Year Built', 'Exterior Wall', 'Grade', 'Bedrooms',
       'Full Bath', 'Half Bath'],
      dtype='object')
```

3. Checking for null values in each column of the dataset, i.e., missing values

In [120]:
```
for x in range(28):
    print("%-45s %10d" % (housing_data.columns.values[x], housing_data.iloc[:,x].isna().sum()))
```

```
Parcel ID                                              0
Land Use                                               0
Property Address                                     159
Suite/Condo                                        50527
Property City                                        159
Sale Date                                              0
Sale Price                                             0
Legal Reference                                        0
Sold As Vacant                                         0
Multiple Parcels Involved in Sale                      0
Owner Name                                         31375
Address                                            30619
City                                               30619
State                                              30619
Acreage                                            30619
Tax District                                       30619
Neighborhood                                       30619
Land Value                                         30619
Building Value                                     30619
Total Value                                        30619
Finished Area                                      32470
Foundation Type                                    32472
Year Built                                         32471
Exterior Wall                                      32471
Grade                                              32471
Bedrooms                                           32477
Full Bath                                          32359
Half Bath                                          32490
```

Table 5. Missing or null values

As observed from the table above, there are missing values present in the field values or columns of the dataset that needs to be addressed by either imputation methods or dropping the rows of data if there are less than 20% of missing values. Hence, the data is analyzed in order to understand which imputation method needs to be implemented.

4. Dropping data values or fields having maximum null values

In [9]:
```
# dropping columns having maximum null values

housing_data = housing_data.drop(['Suite/Condo', 'Owner Name', 'Address', 'City', 'State', 'Year Built', 'Tax
print("Dropped columns having maximum null values.")
```

```
Dropped columns having maximum null values.
```

- Dropping **'Suite/Condo'** field value from the dataset as it has maximum null values present of the total data points.
- Dropping **'Address', 'Owner Name', 'City', 'State', and 'Year Built'** as it has maximum null values and can be considered irrelevant for the training of the model.
- Dropping **'Tax District', 'Neighborhood', 'Finished Area', 'Foundation Type', 'Exterior Wall', and 'Grade'** as it has over **30000 null values** present in the dataset for which imputation methods will not be effective.

In [10]:
```
# dropping rows of data having null values

housing_data.dropna(subset=['Property Address'], inplace=True)
housing_data.dropna(subset=['Property City'], inplace=True)

print("Rows of data having null values dropped.")
```

```
Rows of data having null values dropped.
```

In [123]:
```
housing_data.shape
```

Out[123]:
```
(56477, 16)
```

Property Address and Property City have 159 rows of data with null values from the total **56477 data points** of the dataset, and hence dropping 159 rows of data will not have a bigger impact on the training of the model, hence the rows having null values are dropped.

In [124]:
```
# checking for null values

for x in range(16):
    print("%-45s %10d" % (housing_data.columns.values[x], housing_data.iloc[:,x].isna().sum()))
```

```
Parcel ID                                              0
Land Use                                               0
Property Address                                       0
Property City                                          0
Sale Date                                              0
Sale Price                                             0
Legal Reference                                        0
Sold As Vacant                                         0
Multiple Parcels Involved in Sale                      0
Acreage                                            30462
Land Value                                         30462
Building Value                                     30462
Total Value                                        30462
Bedrooms                                           32320
Full Bath                                          32202
Half Bath                                          32333
```

Table 6. Checking for null values

5. Replacing missing values using various imputation methods

In [125]:
```
# displaying unique data

print("Displaying the unique data present in columns\n")
housing_data.nunique()
```

```
Displaying the unique data present in columns
```

Out[125]:
```
Parcel ID                            48559
Land Use                                39
Property Address                     45068
Property City                           14
Sale Date                             1116
Sale Price                            8081
Legal Reference                      52761
Sold As Vacant                           2
Multiple Parcels Involved in Sale        2
Acreage                                519
Land Value                            1122
Building Value                        4405
Total Value                           5848
Bedrooms                                12
Full Bath                               11
Half Bath                                4
dtype: int64
```

Table 7. Unique Data Count

KNN Imputation method for replacing missing values

In [11]:
```
# KNN imputation method for replacing missing values

imputer = KNNImputer(n_neighbors=3)
housing_data['Acreage'] = imputer.fit_transform(housing_data[['Acreage']])
housing_data['Land Value'] = imputer.fit_transform(housing_data[['Land Value']])
housing_data['Building Value'] = imputer.fit_transform(housing_data[['Building Value']])
housing_data['Total Value'] = imputer.fit_transform(housing_data[['Total Value']])
housing_data['Bedrooms'] = imputer.fit_transform(housing_data[['Bedrooms']])
housing_data['Full Bath'] = imputer.fit_transform(housing_data[['Full Bath']])
housing_data['Half Bath'] = imputer.fit_transform(housing_data[['Half Bath']])
print("Imputation successful.")
```

```
Imputation successful.
```

Since there is a high amount of missing values that are present in the dataset indicating random values missing, it **cannot be imputed using mean or median** method as the approach would be inefficient. Thus, using a **KNN imputation** or a **regression imputation** method to replace the missing values or null values can be implemented.

The above code successful imputed the missing values for all the missing rows of data in the field values using the **KNN imputation method** which is more efficient as compared to other imputation methods in this scenario.

```
In [127]: # checking for dataframe after imputation

housing_data.head(5)
```

Out[127]:

	Parcel ID	Land Use	Property Address	Property City	Sale Date	Sale Price	Legal Reference	Sold As Vacant	Multiple Parcels Involved in Sale	Acreage	Land Value	Building Value
0	105 03 0D 008.00	RESIDENTIAL CONDO	1208 3RD AVE S	NASHVILLE	2013-01-24	132000	20130128-0008725	No	No	0.498923	69068.557601	160784.677109
1	105 11 0 080.00	SINGLE FAMILY	1802 STEWART PL	NASHVILLE	2013-01-11	191500	20130118-0006337	No	No	0.170000	32000.000000	134400.000000
2	118 03 0 130.00	SINGLE FAMILY	2761 ROSEDALE PL	NASHVILLE	2013-01-18	202000	20130124-0008033	No	No	0.110000	34000.000000	157800.000000
3	119 01 0 479.00	SINGLE FAMILY	224 PEACHTREE ST	NASHVILLE	2013-01-18	32000	20130128-0008863	No	No	0.170000	25000.000000	243700.000000
4	119 05 0 186.00	SINGLE FAMILY	316 LUTIE ST	NASHVILLE	2013-01-23	102000	20130131-0009929	No	No	0.340000	25000.000000	138100.000000

Table 8. Housing Data after data imputation

```
In [128]: # checking for any missing values after data cleaning & imputations

for x in range(16):
    print("%-45s %10d" % (housing_data.columns.values[x], housing_data.iloc[:,x].isna().sum()))
```

```
Parcel ID                                              0
Land Use                                               0
Property Address                                       0
Property City                                          0
Sale Date                                              0
Sale Price                                             0
Legal Reference                                        0
Sold As Vacant                                         0
Multiple Parcels Involved in Sale                      0
Acreage                                                0
Land Value                                             0
Building Value                                         0
Total Value                                            0
Bedrooms                                               0
Full Bath                                              0
Half Bath                                              0
```

6. Checking for incorrect data types in field values and correcting the data type of the column

```
In [12]: # correcting the data types for the variables of the dataset which are of object type to string/category and

housing_data['Parcel ID'] = housing_data['Parcel ID'].astype('category')
housing_data['Land Use'] = housing_data['Land Use'].astype('category')
housing_data['Property Address'] = housing_data['Property Address'].astype('category')
housing_data['Property City'] = housing_data['Property City'].astype('category')
housing_data['Sale Date'] = housing_data['Sale Date'].astype('datetime64[ns]')
housing_data['Multiple Parcels Involved in Sale'] = housing_data['Multiple Parcels Involved in Sale'].astype(
housing_data['Legal Reference'] = housing_data['Legal Reference'].astype('category')
housing_data['Sold As Vacant'] = housing_data['Sold As Vacant'].astype('category')

print("Data Type conversion successful.")
```

```
Data Type conversion successful.
```

In [130]:
```python
# checking for the correct data type of the variable

housing_data.dtypes
```

Out[130]:
```
Parcel ID                                category
Land Use                                 category
Property Address                         category
Property City                            category
Sale Date                          datetime64[ns]
Sale Price                                  int64
Legal Reference                          category
Sold As Vacant                           category
Multiple Parcels Involved in Sale        category
Acreage                                   float64
Land Value                                float64
Building Value                            float64
Total Value                               float64
Bedrooms                                  float64
Full Bath                                 float64
Half Bath                                 float64
dtype: object
```

Table 9. Housing Data data type conversion

7. Checking for outliers in the dataset

a. Boxplot

The below code creates **boxplots** for the various field values of the marketing dataset in order to check for outliers present in the dataset. Here, the boxplots are implemented for the variables **Land Value**, **Acreage**, **Total Value**, and **Sale Price** as shown in the figures below.

There are outliers present in the variables as observed in the boxplot below, which will not be removed as each of the data point is important for analysis and model building.

In [131]:
```python
# creating boxplot for 'Land Value' and 'Acreage' variable

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].boxplot(housing_data['Land Value'])
axs[1].boxplot(housing_data['Acreage'])
axs[0].set_title('Boxplot for column, Land Value')
axs[1].set_title('Boxplot for column, Acreage')
axs[0].set_ylabel('Data Values')
axs[1].set_ylabel('Data Values')

plt.show()
```

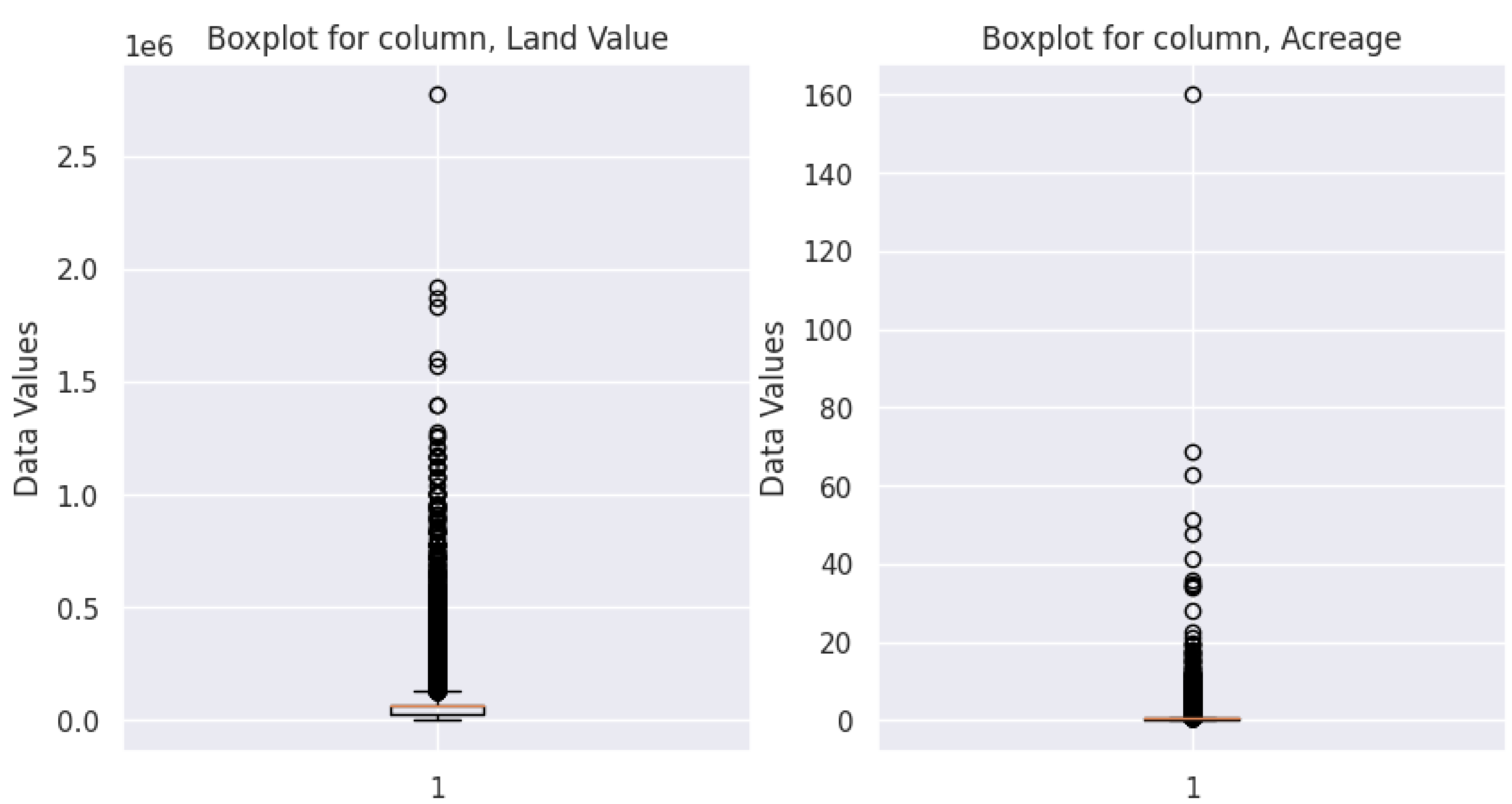

Figure 2. Boxplot for Land Value & Acreage column

In [132]:
```python
# creating boxplot for 'Total Value' and 'Sale Price' variable

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].boxplot(housing_data['Total Value'])
axs[1].boxplot(housing_data['Sale Price'])
axs[0].set_title('Boxplot for column, Total Value')
axs[1].set_title('Boxplot for column, Sale Price')
axs[0].set_ylabel('Data Values')
axs[1].set_ylabel('Data Values')

plt.show()
```

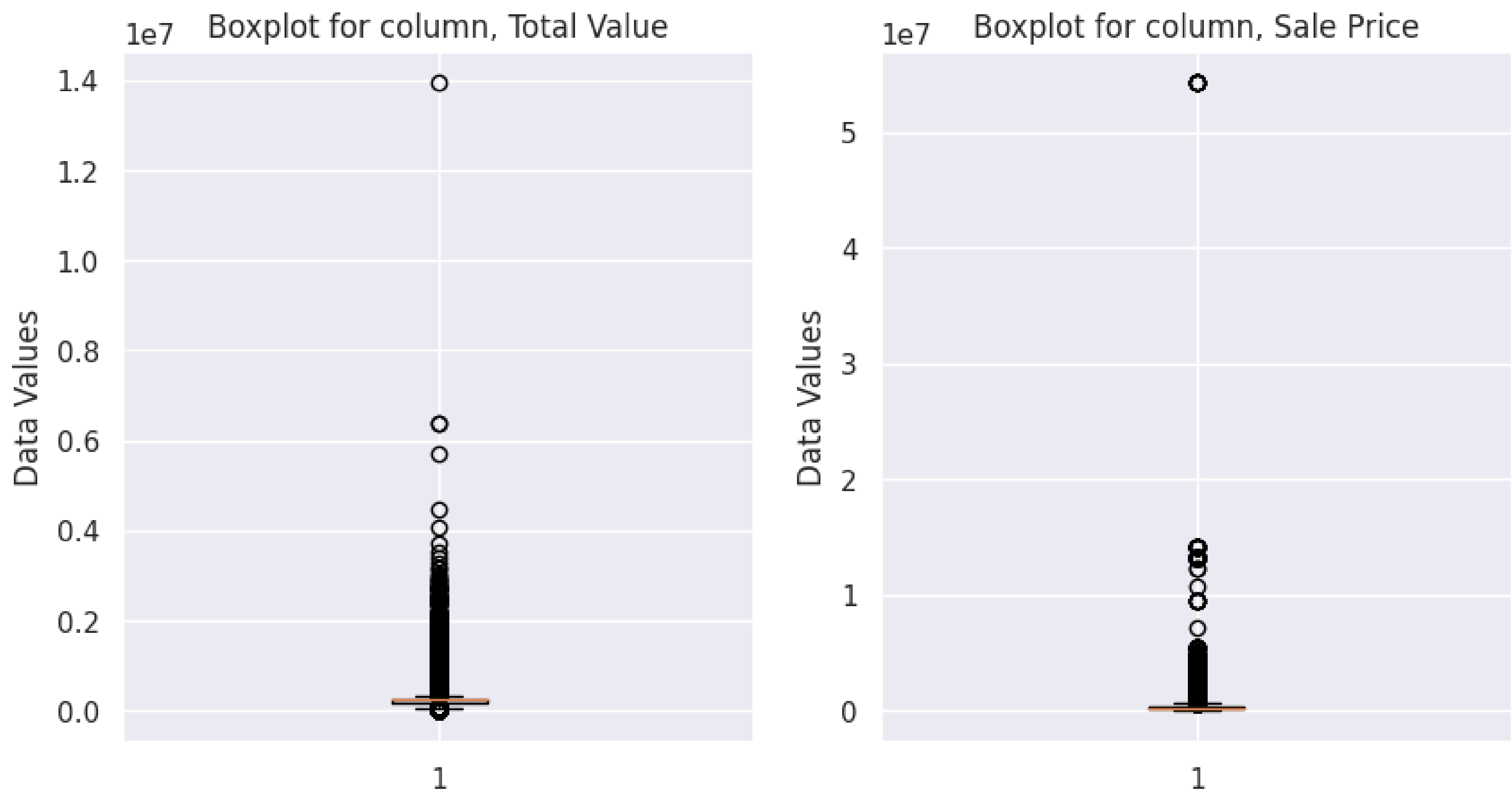

Figure 3. Boxplot for Total Value & Sale Price column

b. Distribution Plot

The distribution plot for the various parameters of the dataset values gives an overview of the outliers that are present and the distribution of the data points across present in the dataset.

The plot below for *Full Bath* and *Half Bath* shows that the data is **right skewed**, i.e., the data is concentrated towards a certain range of values and is not equally distributed.

In [133]:
```python
# distribution plot for Full Bath & Half Bath

plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(housing_data['Full Bath'])
plt.subplot(1,2,2)
sns.distplot(housing_data['Half Bath'])
plt.show()
```

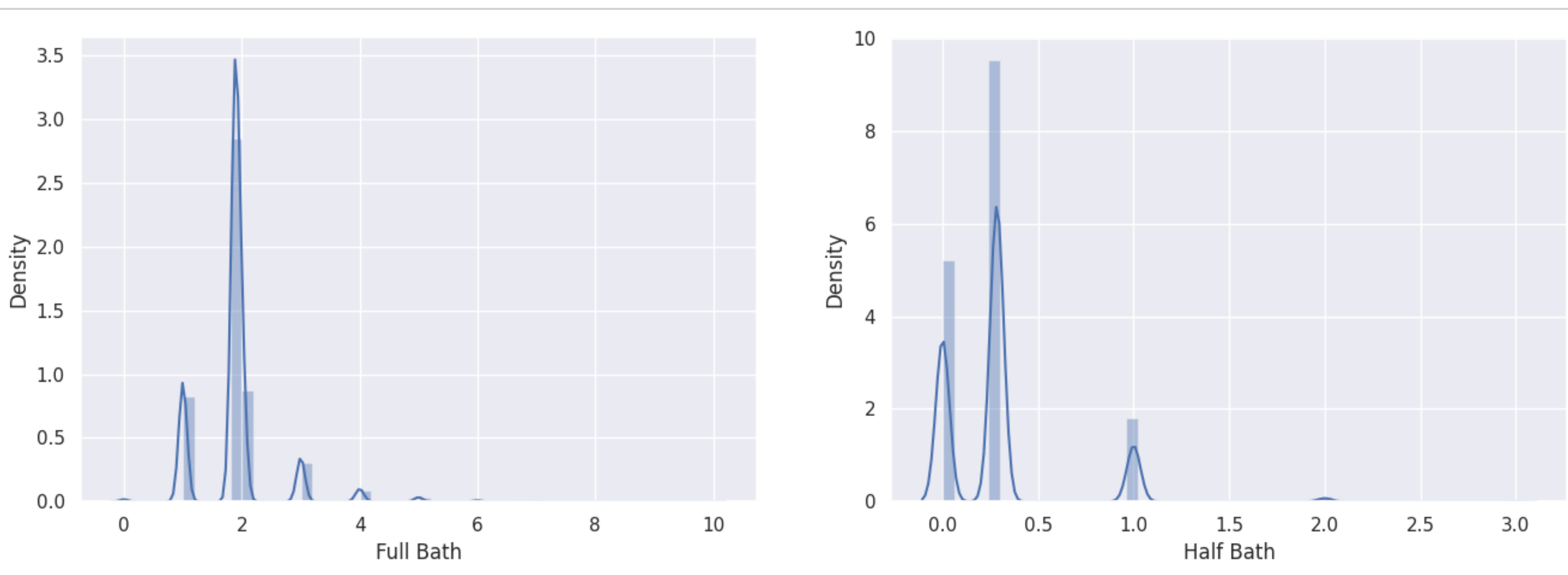

Figure 4. Distribution Plot for Full Bath & Half Bath

Step 3: Data Visualization

In [134]:
```
# Multiple Parcels Involved in Sale Analysis

plt.figure(figsize=(6,5))
sns.countplot(x='Multiple Parcels Involved in Sale', data=housing_data, palette="pastel")
plt.title('\nCount of Multiple Parcels Involved in Sale Analysis')
plt.show()
```

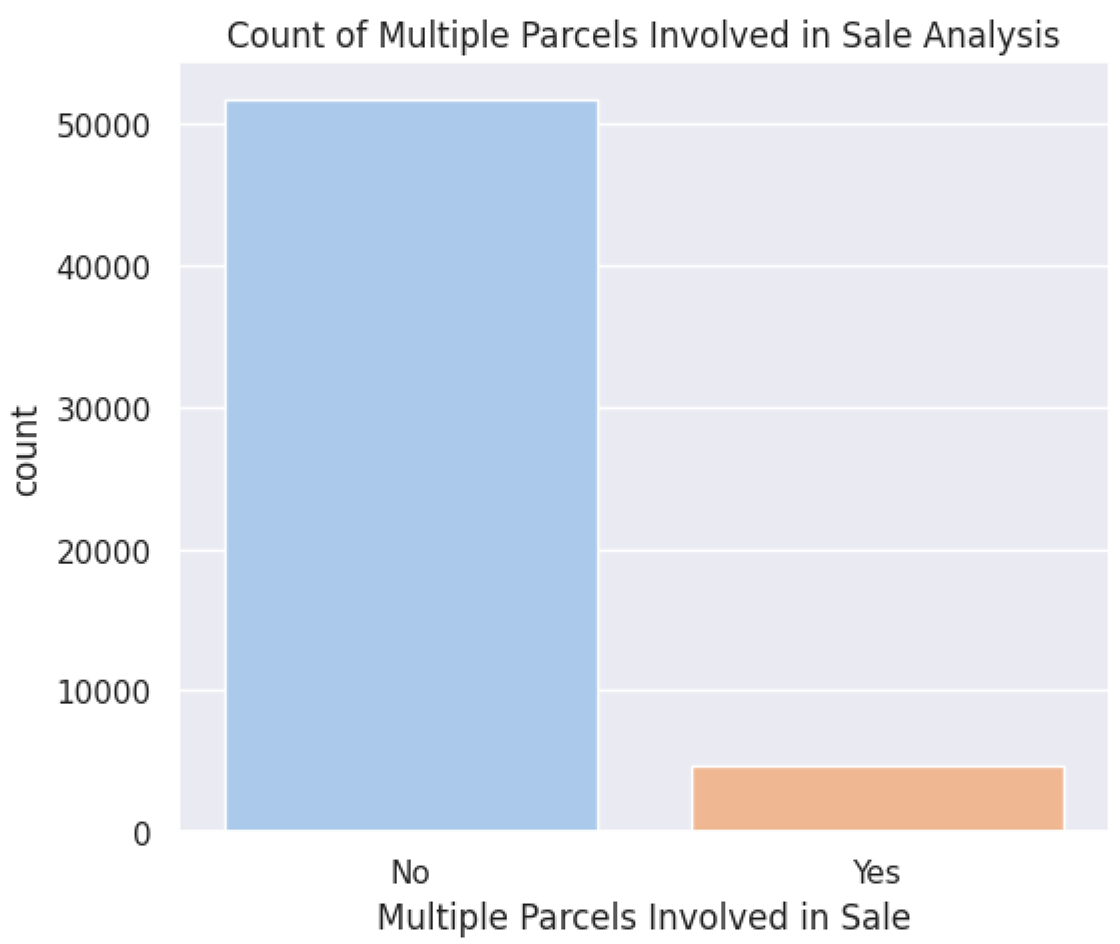

Figure 5. Count of Multiple Parcels Involved in Sale Analysis

From the above graph of *Count of Multiple Parcels Involved in Sale Analysis*, it is observed that the data is **biased towards one class** of no reponse, and hence there is a possibility that the model will be biased to this class which will classify majority of the data points into a certain class itself.

Step 4: Pre-Modeling Steps

1. Label Encoding
2. Correlation Plot
3. Creating the target variable
4. Defining the features for model training
5. Spliting the dataset into train & test set
6. Class Bias
7. Standardization

1. Label Encoding

Since some of the selected features for prediction are categorical data and most of the ML models require the data to be numerical or binary, it is important that these features are converted into **binary** or **numerical** type data in order for the model to be able to classify the classes.

Label Encoding is a method which helps to **convert the categorical variables** into **numerical values**, thus helping to transform the data point into a format where the algorithm is able to process the data for classification. *LabelEncoder()* function is used to encode the categorical type data to numerical type, where new columns of data are created for the categorical field value in the dataset which will be used in the training of the model.

In [13]:
```
labelencoder = LabelEncoder()

housing_data['Parcel ID_Label'] = labelencoder.fit_transform(housing_data["Parcel ID"])
housing_data['Land Use_Label'] = labelencoder.fit_transform(housing_data["Land Use"])
housing_data['Property Address_Label'] = labelencoder.fit_transform(housing_data["Property Address"])
housing_data['Property City_Label'] = labelencoder.fit_transform(housing_data["Property City"])
housing_data['Legal Reference_Label'] = labelencoder.fit_transform(housing_data["Legal Reference"])
housing_data['Sold As Vacant_Label'] = labelencoder.fit_transform(housing_data["Sold As Vacant"])
housing_data['Multiple Parcels Involved in Sale_Label'] = labelencoder.fit_transform(housing_data["Multiple P
print("Label Encoding successful.")
```

```
Label Encoding successful.
```

In [136]:
```
housing_data.columns
```

Out[136]:
```
Index(['Parcel ID', 'Land Use', 'Property Address', 'Property City',
       'Sale Date', 'Sale Price', 'Legal Reference', 'Sold As Vacant',
       'Multiple Parcels Involved in Sale', 'Acreage', 'Land Value',
       'Building Value', 'Total Value', 'Bedrooms', 'Full Bath', 'Half Bath',
       'Parcel ID_Label', 'Land Use_Label', 'Property Address_Label',
       'Property City_Label', 'Legal Reference_Label', 'Sold As Vacant_Label',
       'Multiple Parcels Involved in Sale_Label'],
      dtype='object')
```

2. Corelation Plot

A **correlation plot** or matrix is a *visual representation of the variables* present in the dataset which helps in understanding the *relationship* between the different variables and how highly the variables are corelated to each other.

The values of the correlation plot range from **-1 to 1**, where -1 indicates a **negative correlation** between the variables, 0 indicates **no correlation**, and 1 indicates a **positive correlation**.

The variables that have positive correlation are said to be highly correlated to each and hence either of the two variables must be removed for the model building as it may lead to **multicollinearity** where the efficiency of the model may reduce.

In [137]:
```
# plotting correlation matrix

plt.figure(figsize = (10,7))
ax = plt.subplot()
sns.heatmap(housing_data.corr(),annot=True, fmt='.1f', ax=ax, cmap="Blues")
ax.set_title('Correlation Plot');
```

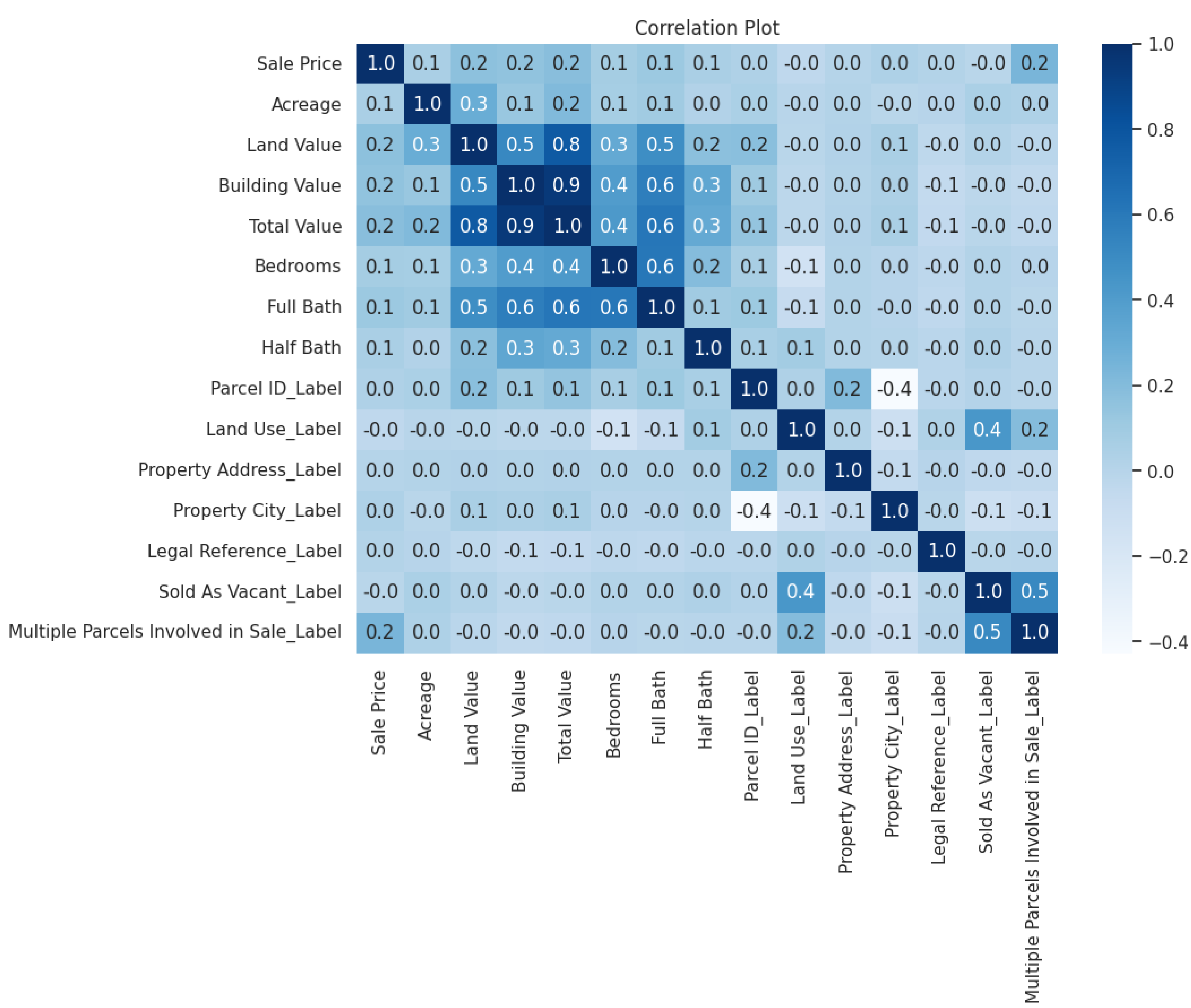

Figure 6. Correlation Matrix

Looking for strong corelations between independent & dependent variables

---

As observed in the correlation matrix above, we see that there are many variables or features that are *highly correlated* to each other and hence we need to analyze the features that are strongly correlated such that these features are excluded from the training of the model in order to avoid **multicollinearity** and *improve the efficiency of the model*. The following features are strongly correlated with the other features in the dataset and can be excluded from model building to avoid multicollinearity.

Correlation among the variables:

1. **Land Value** is highly correlated with **Total Value** with a correlation value of 0.8
2. **Building Value** is highly correlated with **Total Value** and **Full Bath** with correlation value of **0.9 and 0.6** respectively
3. **Total Value** is correlated with **Full Bath**
4. **Bedrooms** is correlated with **Full Bath** with correlation value of **0.6**

Checking VIF score for Multicollinearity

VIF score i.e., **Variance Inflation Factor** is a *measure of multicollinearity* between the independent variables in the regression analysis. This calculates the variance of the variables which helps in understanding the coefficient value and how much the variable is inflated due to collinearity in the model. The VIF score from **range 0 to 5** can be accepted to be considered for the training of the model, while values above 5 are considered to have high multicollinearity which would affect the accuracy and performance of the model, hence should be excluded.

In [138]:
```
# checking VIF score for field values

numerical_housing_data = housing_data.select_dtypes(include=[np.number])
vif_score1 = pd.DataFrame()
vif_score1["Feature"] = numerical_housing_data.columns
vif_score1["VIF Score"] = [variance_inflation_factor(numerical_housing_data.values, i) for i in range(numeric
print(vif_score1)
```

```
                                  Feature    VIF Score
0                              Sale Price     1.282007
1                                 Acreage     1.357335
2                              Land Value   397.049402
3                          Building Value  1876.714469
4                             Total Value  3554.579685
5                                Bedrooms    36.504612
6                               Full Bath    22.015875
7                               Half Bath     2.153554
8                         Parcel ID_Label     5.007941
9                          Land Use_Label    21.507329
10                 Property Address_Label     4.174267
11                    Property City_Label     7.396453
12                 Legal Reference_Label     3.969017
13                   Sold As Vacant_Label     1.713568
14  Multiple Parcels Involved in Sale_Label     1.630923
```

Table 10. VIF Score Table

From the above VIF score table it is observed that the VIF score for** Building Value, Land Value, and Total Value** is exceptionally high indicating that the variables are highly correlated and there are high chances for the issue of multicollinearity to occur, hence these features are excluded from the training of the model.

Apart from that, other features such as **Full Bath, Land Use, Bedrooms, and Property City** also have high VIF score which is not in the range from 0 to 5, hence these features are also excluded from the training of the model.

3. Creating the dependent or the target variable

The aim here is to build a model which will help the organization accurately find best value deals, but there is a concern that houses are going over their asking price.

Thus, a **dependent variable is created** in order to understand whether it is **over price or under price** based on the **Sale Price and Total Value**, where *Sale Price is their selling price of the house and Total Value is the actual price of the house*. The code below creates a **binary target variable** where **1 indicates that the sale price was greater than the total value, i.e., it is over price** and **0 indicates that the sale price was less than or equal to the total value, i.e., it is under price.**

This will help in understanding and analyzing the housing data based on the prediction of the class categories for which the features contributing in the classification can be analyzed and recommended to the organization.

In [14]:
```
housing_data['SalePrice_Target'] = (housing_data["Sale Price"] > housing_data["Total Value"]).astype(int)
print("Target Variable created.")
```

```
Target Variable created.
```

a. Understanding the top features selected by Correlation Matrix

In [140]:
```
corr_result = housing_data.corr()
correlation_response = corr_result['SalePrice_Target'].sort_values(ascending=False)
topfeatures = correlation_response[1:6]
print("The top features selected by correlation matrix are:\n")
print(topfeatures)
```

```
The top features selected by correlation matrix are:

Legal Reference_Label                     0.190161
Property City_Label                       0.168698
Sale Price                                0.145435
Multiple Parcels Involved in Sale_Label   0.055552
Acreage                                  -0.007859
Name: SalePrice_Target, dtype: float64
```

b. Lasso Regression to select the most important features for model training

In [141]:
```
# creating a new dataframe excluding categorical variable

new_housingdata_lasso = pd.DataFrame()
new_housingdata_lasso = housing_data.drop(columns=['Parcel ID', 'Land Use', 'Property Address', 'Property Cit
print("Dataframe created.")
```

```
Dataframe created.
```

In [142]:
```
A = new_housingdata_lasso.drop(['SalePrice_Target'], axis=1)
B = new_housingdata_lasso['SalePrice_Target']
lasso_result = Lasso(alpha=0.1)
lasso_result.fit(A, B)
coef = pd.Series(lasso_result.coef_, index=A.columns)
features_lasso = coef.abs().sort_values(ascending=False).head(5).index
print("The top features selected by Lasso regression:\n")
print(features_lasso)
```

```
The top features selected by Lasso regression:

Index(['Property City_Label', 'Legal Reference_Label', 'Parcel ID_Label',
       'Building Value', 'Total Value'],
      dtype='object')
```

c. Features selected for model building

The features that are selected for the model building based on the Correlation Plot values, Lasso Regression, and VIF Score are as follows:

Correlation Plot Values	Lasso Regression	Features with VIF Scores in range 0 to 5
Legal Reference_Label	Property City_Label	Sale Price
Property City_Label	Legal Reference_Label	Acreage
Sale Price	Parcel ID_Label	Half Bath
Multiple Parcels Involved in Sale_Label	Building Value	Parcel ID_Label
Acreage	Total Value	Property Address_Label
		Legal Reference_Label
		Sold As Vacant_Label
		Multiple Parcels Involved in Sale_Label

Table 11. Feature Selection & Extraction Table

4. Defining the features for model training (Dimensionality Reduction)

The model is trained & built on the above mentioned features that are selected from the analysis of the Feature selection and extraction, Correlation matrix, Lasso Regression, and VIF score for multicollinearity.

In [15]:
```
X = housing_data.drop(columns=['Parcel ID', 'Land Use', 'Property Address', 'Property City', 'Sale Date', 'Le
y = housing_data['SalePrice_Target']
```

5. Splitting the dataset into train & test set

The dataset is split into training and testing data with a random split of **80%** train set and **20%** for test data.

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42)
```

6. Handling imbalance data (Class Bias)

Class Bias in a dataset occurs when **distribution of data points in a dataset is uneven**, with one or more classes being overrepresented or underrepresented, which means that either of the category has majority of the data points and thus while training of the model, the prediction will be biased to that category. Hence, it is important to **handle class imbalance** in the dataset which can be performed using various methods, and one such method that is implemented below is the **class weights**.

In [17]:
```
class_weights = compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y_train),
                                        y = y_train
                                    )
class_weights = dict(zip(np.unique(y_train), class_weights))
print(class_weights)
```

```
{0: 1.2408953584180171, 1: 0.8374295670225386}
```

7. Standardization

Standardization is performed on the split dataset in order to make the features selected comparable to a standardized scale.

In [18]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("Standardization successful")
```

```
Standardization successful
```

Label Count in train & test dataset

In [147]:
```
print('Labels count in y:', np.bincount(y))
print('Labels count in y_train:', np.bincount(y_train))
print('Labels count in y_test:', np.bincount(y_test))
```

```
Labels count in y: [22766 33711]
Labels count in y_train: [18205 26976]
Labels count in y_test: [4561 6735]
```

Step 5: Model Building - Logistic Regression

Task 2: Build a logistic regression model to accurately identify overpricing/underpricing and determine what is driving those prices.

Building a Logistic Regression model to accurately identify overpricing/underpricing and determine what is driving those prices.

Fitting the Logistic Regression model

The LR model is fit with a balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [19]:
```
logisticreg_model = LogisticRegression(solver='newton-cg', class_weight='balanced')
logisticreg_model.fit(X_train_scaled, y_train)
```

Out[19]:
```
LogisticRegression
LogisticRegression(class_weight='balanced', solver='newton-cg')
```

Displaying the coefficients & intercepts after fitting the model

As observed from the below code, the coefficient values of the variables are displayed which are either positive or negative, which indicates that the variables with **positive** value have a **positive relationship with the target variable** whereas values having a **negative sign** indicate that there is a **negative relationship between the independent variable and the target variable.**

In [149]:
```
coefficient_values = pd.DataFrame({'Feature': X.columns, 'Coefficient': logisticreg_model.coef_[0]})
print('Coefficients:')
print(coefficient_values)
```

```
Coefficients:
                                 Feature  Coefficient
0                                Acreage    -0.002972
1                              Half Bath    -0.047619
2                        Parcel ID_Label    -0.193682
3                 Property Address_Label     0.010830
4                  Legal Reference_Label     0.403349
5                    Sold As Vacant_Label    -0.375955
6  Multiple Parcels Involved in Sale_Label     0.341439
```

Table 12. Coefficients Table

Summary Report of the Logistic Regression model

Summary report of the Logistic Regression model provides an overview of the model build and how accurately the model fits the data for each independent variable to predict or classify the target variable or dependent variable. The report is used to evaluate the overall fit of the model, identify which independent variables are most important in predicting the dependent variable, and analyze the statistical significance of each coefficients.

- From the summary report for the Logistic Regression model below, it is observed that the p-value for one of the features of the dataset is **greater that the significance value of 0.05**, hence the variable or feature is considered **statistically insignificant**. The variable that is statistically insignificant is **Acreage**. Thus, this indicates that the variable is not contributing in the classification of the response variable.
- The remaining features have **p-value less than the significance value of 0.05**, hence are consider as **statistically significant variables**, indicating that these features are contributing in the prediction of the target variable.
- Of the statistically significant variables, features having highest coefficient values are **Legal Reference_Label and Property Address_Label**, indicating that they have the **highest positive influence on the target variable**, whereas **Parcel ID_Label** has the **highest negative impact on the target variable**, which means that if the Parcel ID is higher, the price of the house is under priced.
- **Hence, the variables that are significant which have an impact on the business are Legal Reference_Label and Property Address_Label, and the variable that has the highest negative impact is the Parcel ID feature.**

In [150]:
```
model_summary = sm.Logit(endog=y, exog=X)
summary_report = model_summary.fit()
print(summary_report.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.639254
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:       SalePrice_Target   No. Observations:                56477
Model:                          Logit   Df Residuals:                    56470
Method:                           MLE   Df Model:                            6
Date:                Fri, 12 May 2023   Pseudo R-squ.:                 0.05190
Time:                        21:17:41   Log-Likelihood:                -36103.
converged:                       True   LL-Null:                       -38080.
Covariance Type:            nonrobust   LLR p-value:                     0.000
===========================================================================================================
                                              coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------------------------
Acreage                                     0.0019      0.009      0.209      0.834      -0.016       0.020
Half Bath                                  -0.1337      0.027     -4.981      0.000      -0.186      -0.081
Parcel ID_Label                         -1.276e-05   5.91e-07    -21.602      0.000   -1.39e-05   -1.16e-05
Property Address_Label                   1.753e-06   6.26e-07      2.801      0.005    5.26e-07    2.98e-06
Legal Reference_Label                    2.766e-05   5.06e-07     54.678      0.000    2.67e-05    2.86e-05
Sold As Vacant_Label                       -1.3173      0.041    -32.493      0.000      -1.397      -1.238
Multiple Parcels Involved in Sale_Label     1.2311      0.044     28.238      0.000       1.146       1.317
===========================================================================================================
```

Ranking the top three variables by the highest coefficient (by absolute value).

In [151]:
```
coef_sort = abs(summary_report.params).sort_values(ascending=False).head(3)
table2 = pd.DataFrame({'Coefficient (abs)': coef_sort})
table2 = table2.loc[coef_sort.index]
print(table2)
```

```
                                         Coefficient (abs)
Sold As Vacant_Label                              1.317300
Multiple Parcels Involved in Sale_Label           1.231116
Half Bath                                         0.133734
```

Thus, the features that are driving the prices are analyzed based on the absolute coefficient values mentioned above.

Model Testing

In [152]:
```
y_pred = logisticreg_model.predict(X_test_scaled)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. AUC-ROC curve

In [153]:
```
# Accuracy of the model on training and testing set

print('Accuracy of Logistic Regressor model on training set: {:.3f}'.format(logisticreg_model.score(X_train_s
print('Accuracy of Logistic Regressor model on test set:     {:.3f}'.format(logisticreg_model.score(X_test_sc

model_result1 = logisticreg_model.score(X_test_scaled, y_test)
model_result1 = round(model_result1,4)
print("Overall Accuracy of the model is ", model_result1)
```

```
Accuracy of Logistic Regressor model on training set: 0.609
Accuracy of Logistic Regressor model on test set:     0.606
Overall Accuracy of the model is  0.6065
```

In [154]:
```
# Confusion Matrix

confusionmatrix_LR = confusion_matrix(y_test, y_pred)

fig = sns.heatmap(confusionmatrix_LR, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['Under Price','Over Price'])
fig.yaxis.set_ticklabels(['Under Price','Over Price'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the Logistic Regression Model')
sns.set(font_scale=1.0)
```

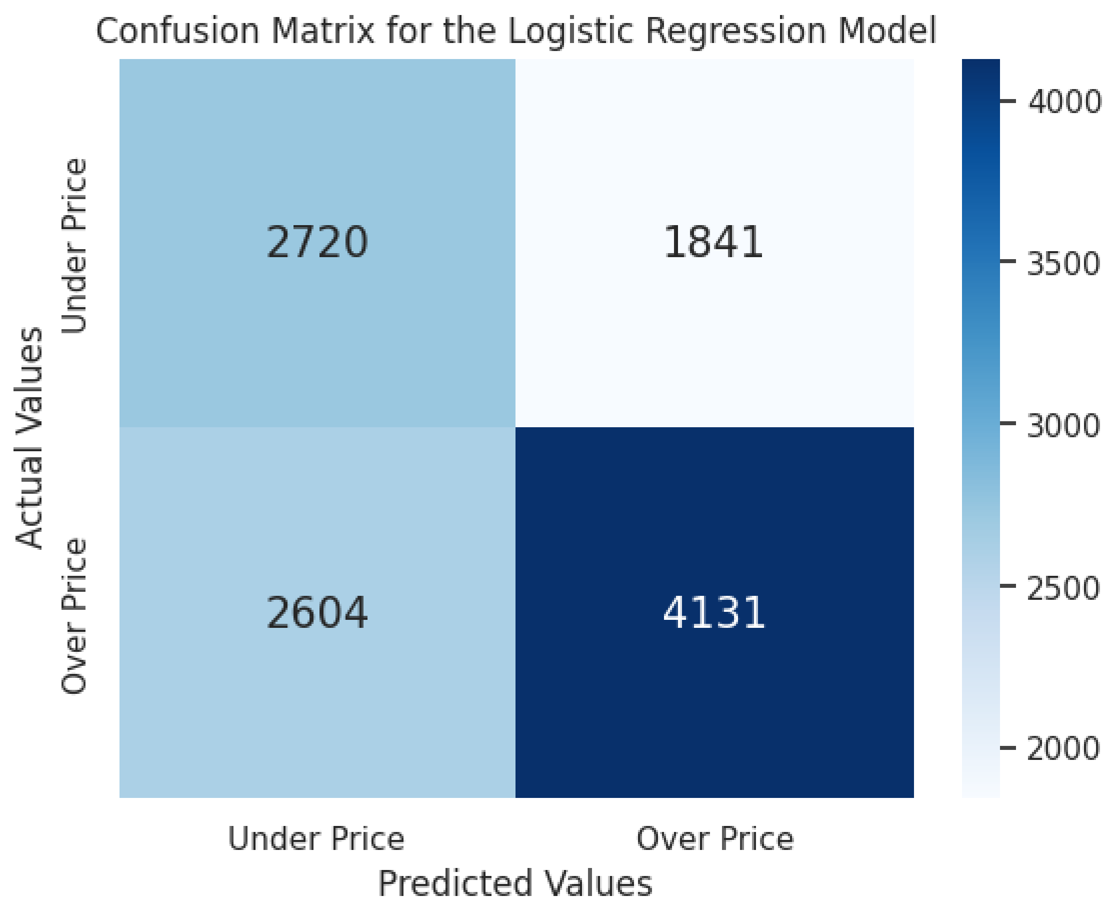

Figure 7. Confusion Matrix for Logistic Regression Model

In [155]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (logisticreg_model, metrics.classification_report(y_test, y_pred
```

```
 Classification report LogisticRegression(class_weight='balanced', solver='newton-cg'):
              precision    recall  f1-score   support

           0       0.51      0.60      0.55      4561
           1       0.69      0.61      0.65      6735

    accuracy                           0.61     11296
   macro avg       0.60      0.60      0.60     11296
weighted avg       0.62      0.61      0.61     11296
```

In [156]:
```
# AUC-ROC Curve

ypred_prob = logisticreg_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, ypred_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8,5))
plt.title('ROC Curve')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.xlabel('False Positive Rate')
plt.show()
```

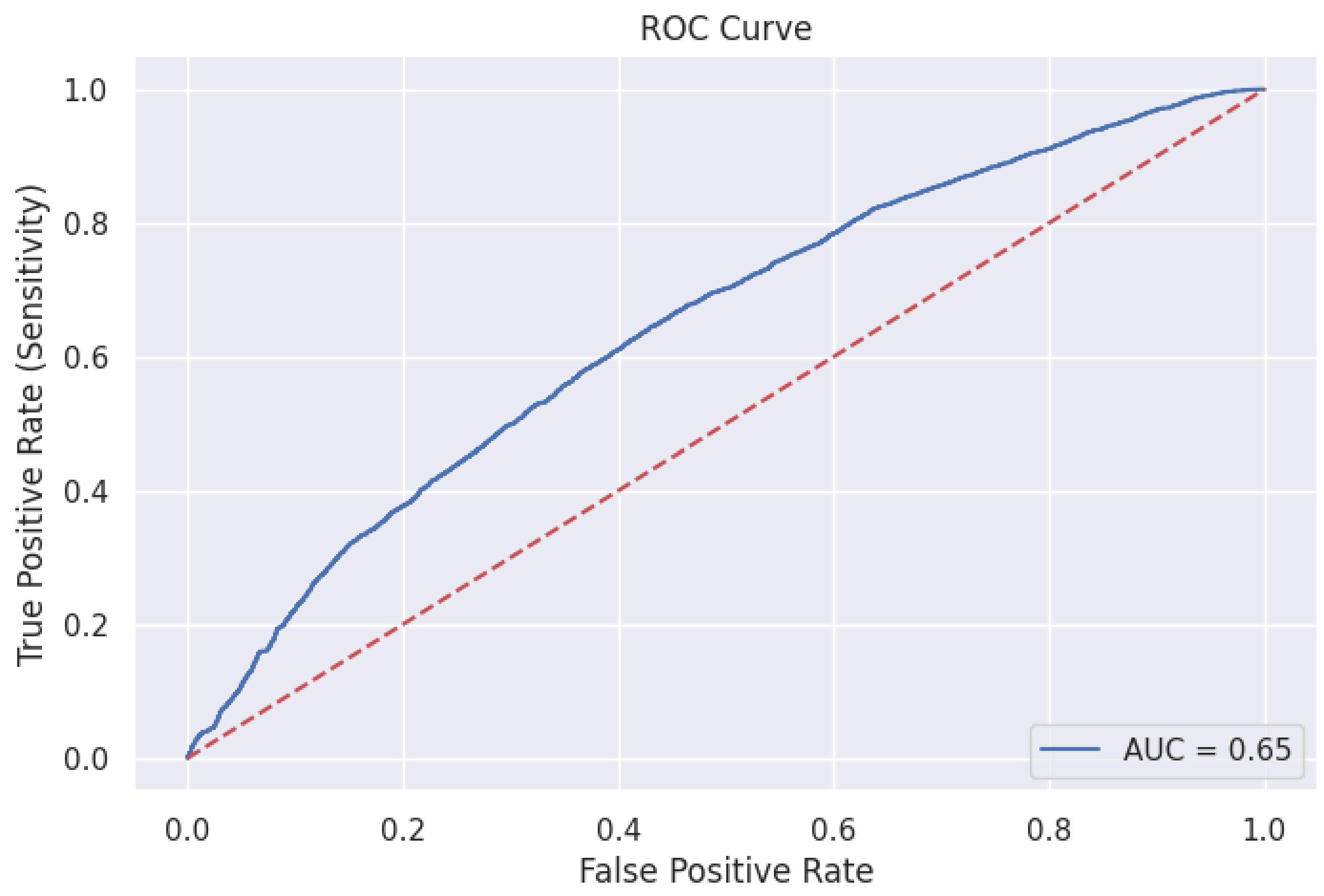

Figure 8. ROC Curve

Accuracy metric:

From the above evaluation metrics it is observed that the Logistic Regression model performed well in classifying the price range of the house with an accuracy of **60.9% for training data and 60.6% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy difference between the training and testing data is almost the same which indicates that the model is neither overfitted nor underfitted and it is able to make predictions on the training data as well as accurately predict on the new or unseen data.

Confusion matrix:

The confusion matrix of the Logistic Regression model indicates that the **under price category is correctly classified 2720 times** whereas the **over price class is correctly classified 4131 times**, which is a good percent of values where the data is been correctly classified. However, the **under price category is wrongly classified times as over price class 1841 times** and **over price class is classified as under price category 2604 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **under price and over price is 51% and 69% respectively**, and as it can be observed the precision score is less for both the categories. Similarly, the recall score for the **under price class is 60%** and for **over price class is 61%** which indicates that the model can perform better as the score is not efficient to classify the classes.

ROC Curve:

The AUC score for the Logistic Regression model built to classify the house price range is **65%** which is not a good AUC score indicating that the model requires improvement.

Step 5: Model Building - Decision Tree

Task 3: Build a decision tree model and compare the results with the results of the previous model.

Building a Decision Tree model to identify overpricing/underpricing of the housing dataset.

Fitting the Decision Tree model

The Decision Tree model is fit with a max depth of 4 and balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [20]:
```
decisiontree_model = DecisionTreeClassifier(max_depth=4, random_state=42, class_weight="balanced")
decisiontree_model.fit(X_train_scaled, y_train)
```

Out[20]:
```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', max_depth=4, random_state=42)
```

Plotting the Decision Tree

In [158]:
```
# plotting the decision tree

plt.figure(figsize=(18,8))
plot_tree(decisiontree_model, filled=True, rounded=True, feature_names=X_train.columns, class_names=["Under P
plt.show()
```

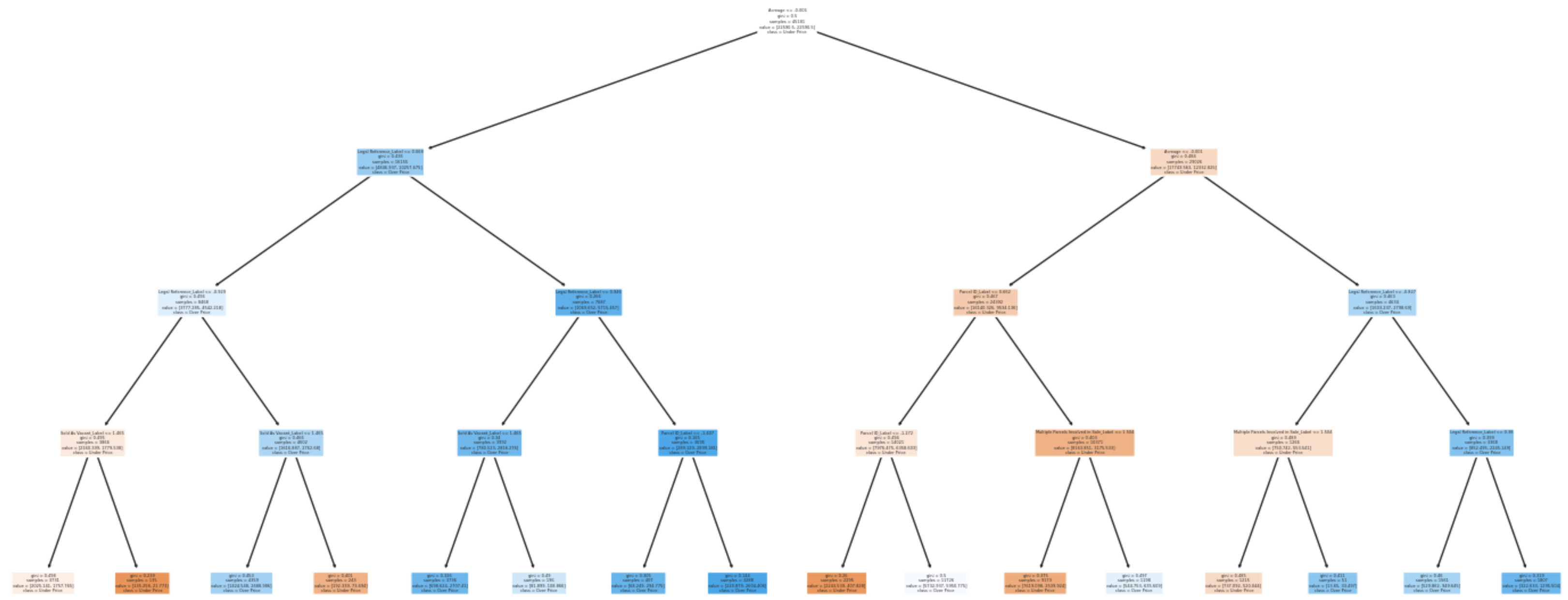

Figure 9. Decision Tree

Model Testing

In [159]:
```
y_pred = decisiontree_model.predict(X_test_scaled)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [160]:
```
# Accuracy of the model on training and testing set

print('Accuracy of Decision Tree model on training set: {:.3f}'.format(decisiontree_model.score(X_train_scale
print('Accuracy of Decision Tree model on test set:     {:.3f}'.format(decisiontree_model.score(X_test_scaled

model_result2 = decisiontree_model.score(X_test_scaled, y_test)
model_result2 = round(model_result2,4)
print("Overall Accuracy of the model is ", model_result2)
```

```
Accuracy of Decision Tree model on training set: 0.687
Accuracy of Decision Tree model on test set:     0.695
Overall Accuracy of the model is  0.6951
```

In [161]:
```
# Confusion Matrix

confusionmatrix_LR = confusion_matrix(y_test, y_pred)

fig = sns.heatmap(confusionmatrix_LR, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['Under Price','Over Price'])
fig.yaxis.set_ticklabels(['Under Price','Over Price'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the Decision Tree Model')
sns.set(font_scale=1.0)
```

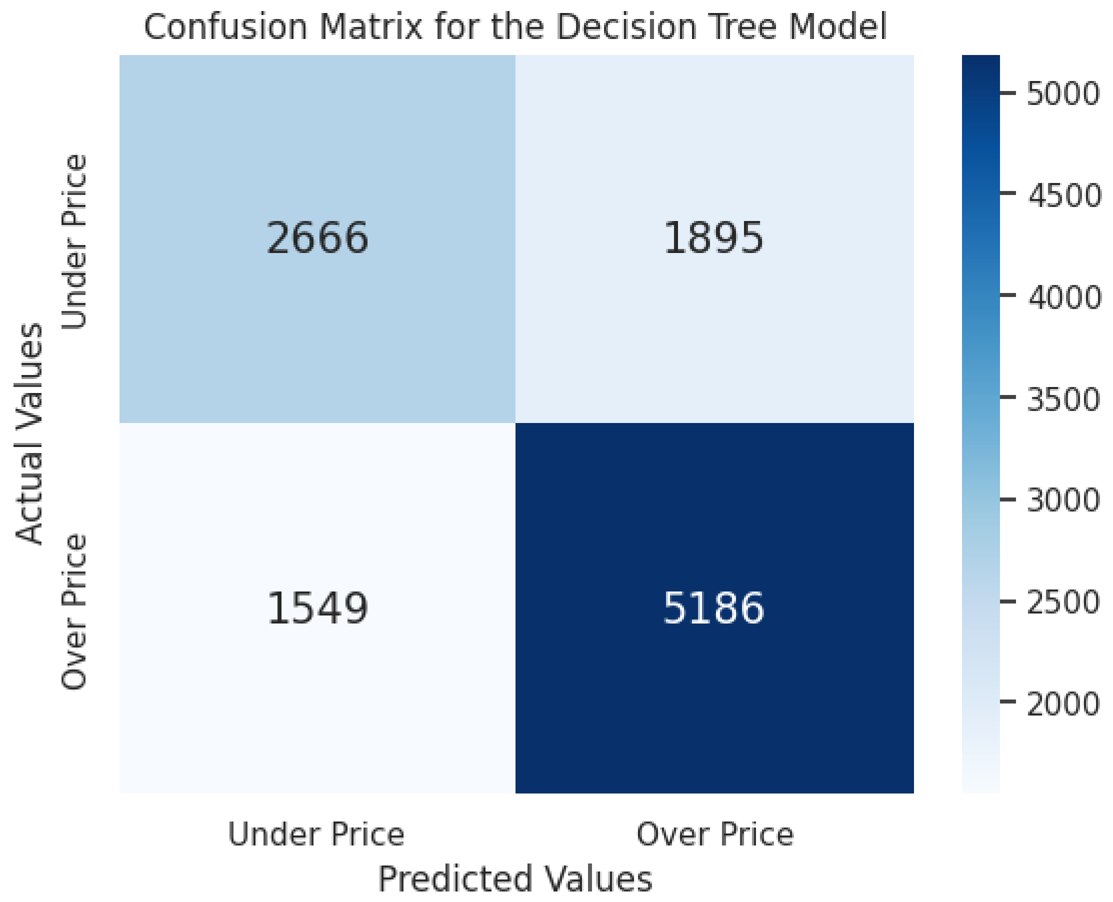

Figure 10. Confusion Matrix for Decision Tree model

In [162]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (decisiontree_model, metrics.classification_report(y_test, y_pre
```

```
 Classification report DecisionTreeClassifier(class_weight='balanced', max_depth=4, random_state=42):
              precision    recall  f1-score   support

           0       0.63      0.58      0.61      4561
           1       0.73      0.77      0.75      6735

    accuracy                           0.70     11296
   macro avg       0.68      0.68      0.68     11296
weighted avg       0.69      0.70      0.69     11296
```

Feature Importance for Decision Tree model

In [163]:
```
feature_importances = pd.Series(decisiontree_model.feature_importances_, index=X.columns)
feature_importances.nlargest(10).plot(kind='barh', color="darkblue", title = "Feature Importance for Decision
```

Out[163]:
```
<Axes: title={'center': 'Feature Importance for Decision Tree model'}>
```

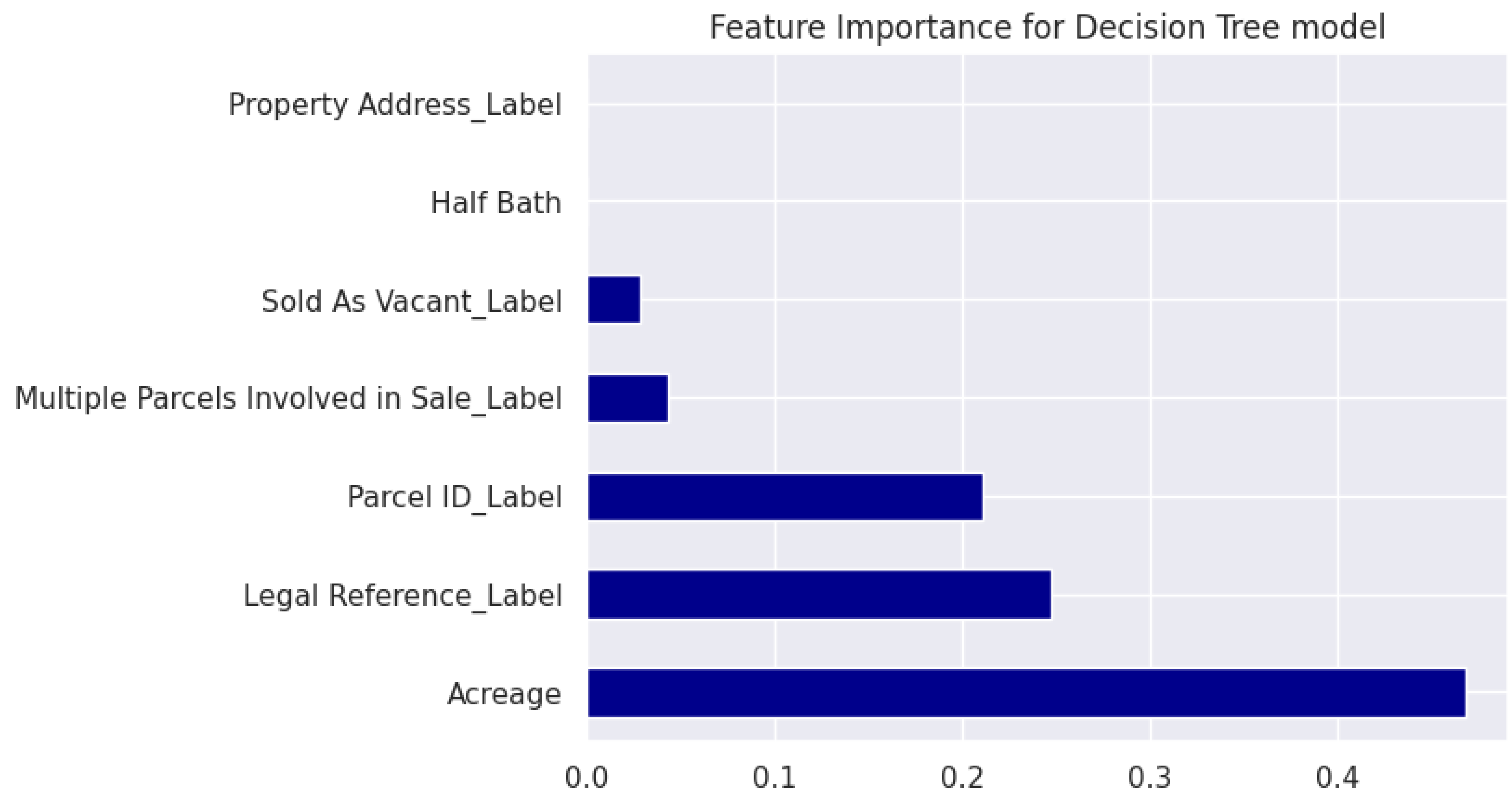

Figure 11. Feature Importance for Decision Tree model

Feature Importance Score

In [164]:
```
# extracting feature importance

feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance Score' : np.round(decisiontree_mod
feature_importance.sort_values('Importance Score', ascending=False, inplace = True)
print(feature_importance)
```

```
                                  Feature  Importance Score
0                                 Acreage             0.469
4                   Legal Reference_Label             0.248
2                         Parcel ID_Label             0.211
6  Multiple Parcels Involved in Sale_Label             0.044
5                    Sold As Vacant_Label             0.029
1                               Half Bath             0.000
3                  Property Address_Label             0.000
```

Table 13. Feature Importance for Decision Tree model

Accuracy metric:

From the above evaluation metrics it is observed that the Decision Tree model performed well in classifying the price range of the house with an accuracy of **68.7% for training data and 69.5% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy difference between the training and testing data is almost the same which indicates that the model is neither overfitted nor underfitted and it is able to make predictions on the training data as well as accurately predict on the new or unseen data.

Confusion matrix:

The confusion matrix of the Decision Tree model indicates that the **under price category is correctly classified 2666 times** whereas the **over price class is correctly classified 5186 times**, which is a good percent of values where the data is been correctly classified. However, the **under price category is wrongly classified times as over price class 1895 times** and **over price class is classified as under price category 1549 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **under price and over price is 63% and 73% respectively**, and as it can be observed the precision score is less for both the categories. Similarly, the recall score for the **under price class is 58%** and for **over price class is 77%** which indicates that the model can perform better as the score is not efficient to classify the classes.

Feature Importance Score:

The feature importance graph shows that variables **Acerage, Legal Reference, and Parcel ID** have the highest feature importance with **score of 0.469, 0.248, and 0.211 respectively** indicating that the model classification for house price is based on these features and that they are highly contributing in the prediction of the target variable. Thus, **it is recommended that the company should focus on these parameters and features while analyzing the house prices.**

Decision Tree model in comparison with Logistic Regression model

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression, it is observed that **Decision Tree model performed better in classifying the house price range** as compared to the Logistic Regression model.
- Decision Tree model **performed well** as compared to the Logistic Regression model.
- This is because the **accuracy of the training and testing data slightly increased with respect to the Decision Tree model** as compared to that of the Logistic Regression model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the Decision Tree model as compared to the LR model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be effective in analyzing the data.
- Also, the **precision and recall score is relatively high for the Decision Tree model** in comparison with the Logistic Regression model, indicating that the prediction of classes is better performed in Decision Tree model.

Step 5: Model Building - Random Forest

Task 4: Build a Random Forest model and compare the results with the results of the previous models.

Building a Random Forest model to identify overpricing/underpricing of the housing dataset.

Fitting the Random Forest model

The Random Forest model is fit with a max depth of 4 and balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [21]:
```
randomforest_model = RandomForestClassifier(n_estimators=5000, max_depth = 4, random_state = 42, class_weight
randomforest_model.fit(X_train_scaled, y_train)
```

Out[21]:
```
                    RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=4, n_estimators=5000,
                       random_state=42)
```

Model Testing

In [166]:
```
y_pred = randomforest_model.predict(X_test_scaled)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [167]:
```
# Accuracy of the model on training and testing set

print('Accuracy of Random Forest model on training set: {:.3f}'.format(randomforest_model.score(X_train_scale
print('Accuracy of Random Forest model on test set:     {:.3f}'.format(randomforest_model.score(X_test_scaled

model_result3 = randomforest_model.score(X_test_scaled, y_test)
model_result3 = round(model_result3,4)
print("Overall Accuracy of the model is ", model_result3)
```

```
Accuracy of Random Forest model on training set: 0.667
Accuracy of Random Forest model on test set:     0.672
Overall Accuracy of the model is  0.6717
```

In [168]:
```
# Confusion Matrix

confusionmatrix_LR = confusion_matrix(y_test, y_pred)

fig = sns.heatmap(confusionmatrix_LR, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['Under Price','Over Price'])
fig.yaxis.set_ticklabels(['Under Price','Over Price'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the Random Forest Model')
sns.set(font_scale=1.0)
```

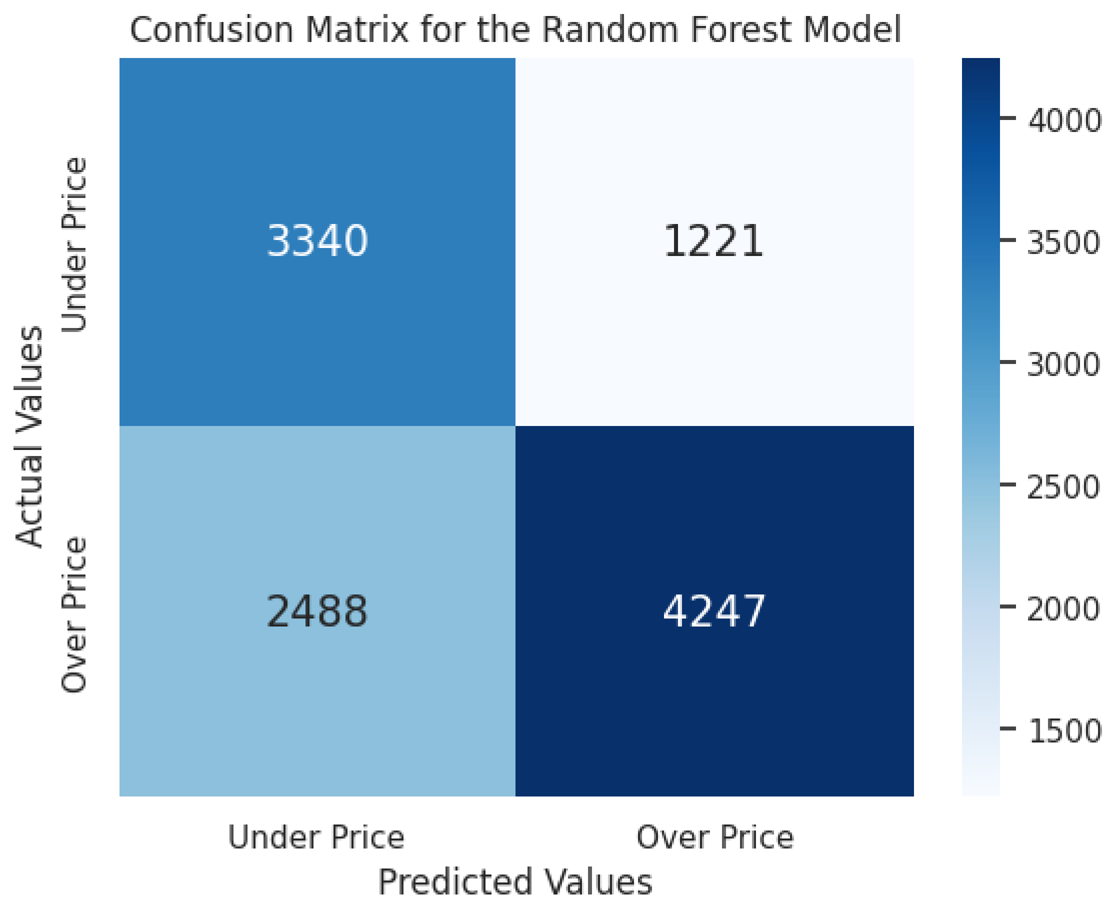

Figure 12. Confusion Matrix for Random Forest Model

In [169]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (randomforest_model, metrics.classification_report(y_test, y_pre
```

```
 Classification report RandomForestClassifier(class_weight='balanced', max_depth=4, n_estimators=5000,
                       random_state=42):
              precision    recall  f1-score   support

           0       0.57      0.73      0.64      4561
           1       0.78      0.63      0.70      6735

    accuracy                           0.67     11296
   macro avg       0.67      0.68      0.67     11296
weighted avg       0.69      0.67      0.67     11296
```

Feature Importance for Random Forest model

In [170]:
```
feature_importances = pd.Series(randomforest_model.feature_importances_, index=X.columns)
feature_importances.nlargest(10).plot(kind='barh', color="darkblue", title = "Feature Importance for Random F
```

Out[170]:
```
<Axes: title={'center': 'Feature Importance for Random Forest model'}>
```

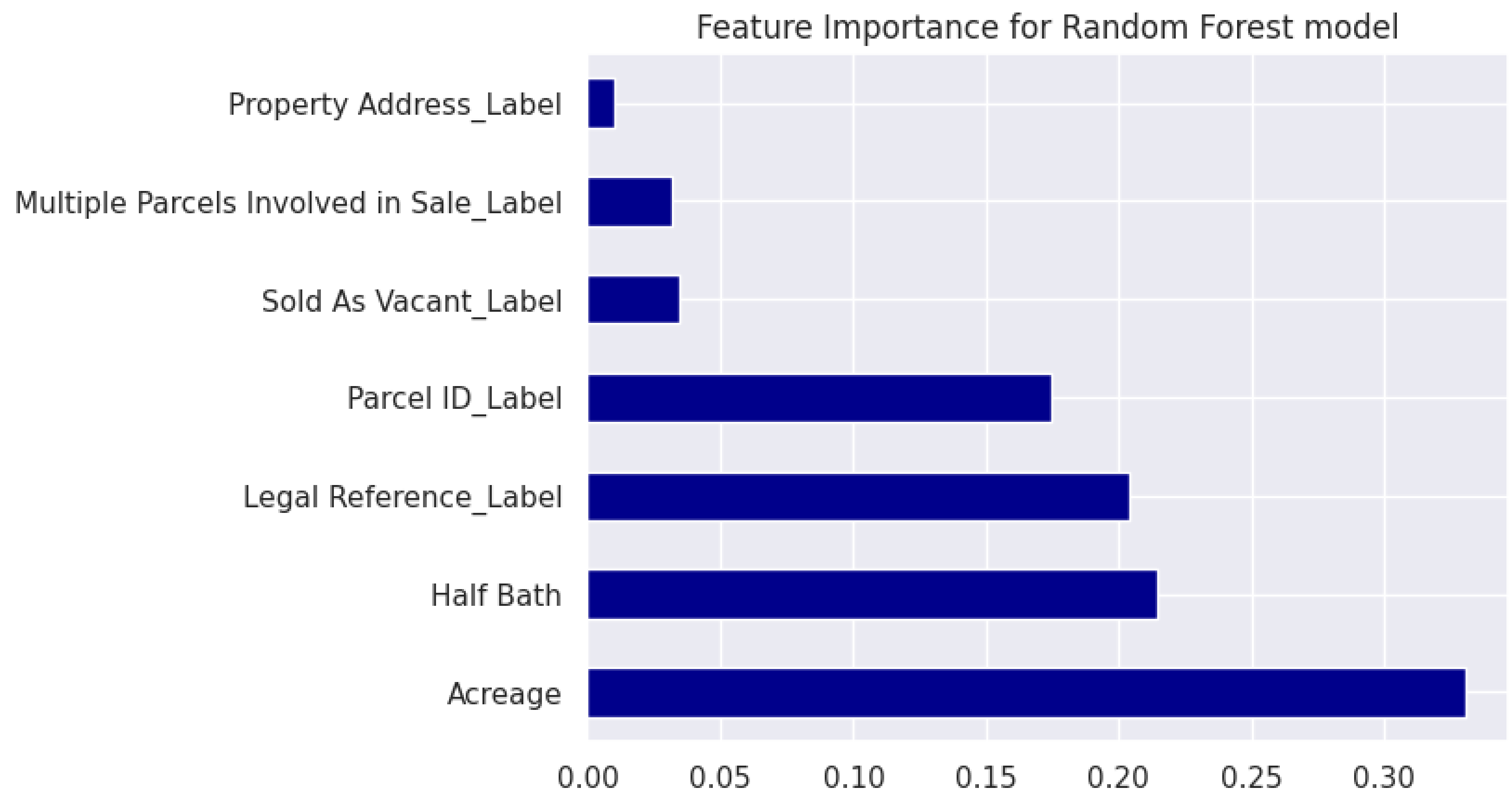

Figure 13. Feature Importance for Random Forest model

Feature Importance Score

In [171]:
```
# extracting feature importance

feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance Score' : np.round(randomforest_mod
feature_importance.sort_values('Importance Score', ascending=False, inplace = True)
print(feature_importance)
```

```
                                   Feature  Importance Score
0                                  Acreage             0.330
1                                Half Bath             0.215
4                    Legal Reference_Label             0.204
2                          Parcel ID_Label             0.175
5                     Sold As Vacant_Label             0.034
6  Multiple Parcels Involved in Sale_Label             0.032
3                   Property Address_Label             0.010
```

Table 14. Feature Importance for Random Forest model

Accuracy metric:

From the above evaluation metrics it is observed that the Random Forest model performed well in classifying the price range of the house with an accuracy of **66.7% for training data and 67.2% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy difference between the training and testing data is almost the same which indicates that the model is neither overfitted nor underfitted and it is able to make predictions on the training data as well as accurately predict on the new or unseen data.

Confusion matrix:

The confusion matrix of the Decision Tree model indicates that the **under price category is correctly classified 3340 times** whereas the **over price class is correctly classified 4247 times**, which is a good percent of values where the data is been correctly classified. However, the **under price category is wrongly classified times as over price class 1221 times** and **over price class is classified as under price category 2488 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **under price and over price is 57% and 78% respectively**, and as it can be observed the precision score is less for the under price category. Similarly, the recall score for the **under price class is 73%** and for **over price class is 63%** which indicates that the model can perform better as the score is not efficient to classify the classes.

Feature Importance Score:

The feature importance graph shows that variables **Acerage, Half Bath, and Legal Reference** have the highest feature importance with **score of 0.330, 0.215, and 0.204 respectively** indicating that the model classification for house price is based on these features and that they are highly contributing in the prediction of the target variable. Thus, **it is recommended that the company should focus on these parameters and features while analyzing the house prices.**

Random Forest model in comparison with Logistic Regression and Decision Tree model

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression and Decision Tree model, it is observed that **Decision Tree model performed better in classifying the house price range** as compared to the Logistic Regression and Random Forest model.
- Decision Tree model **performed well** as compared to the Logistic Regression model and Random Forest model.
- This is because the **accuracy of the training and testing data is better with respect to the Decision Tree model** as compared to that of the Logistic Regression and Random Forest model. Although, Random Forest Classifier performed better as compared to the Logistic Regression model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the Decision Tree model as compared to the LR and Random Forest model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be effective in analyzing the data.
- Also, the **precision and recall score is relatively high for the Decision Tree model** in comparison with the Logistic Regression and Random Forest model, indicating that the prediction of classes is better performed in Decision Tree model.

Step 5: Model Building - XGBoost

Task 5: Build a Gradient Boost model and compare the results with the results of the previous models.

Building a Gradient Boost model to identify overpricing/underpricing

Fitting the XGBoost model

The XGBoost model is fit with a max depth of 3, learning rate of 0.1, and balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [22]:
```
xgb_model = xgb.XGBClassifier(learning_rate=0.1, n_estimators=100, max_depth=3)
xgb_model.fit(X_train_scaled, y_train)
```

Out[22]:
```
XGBClassifier
XGBClassifier(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, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=3, 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=0,
              reg_alpha=0, reg_lambda=1, ...)
```

Model Testing

In [174]:
```
y_pred = xgb_model.predict(X_test_scaled)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [175]:
```
# Accuracy of the model on training and testing set

print('Accuracy of XGBoost model on training set: {:.3f}'.format(xgb_model.score(X_train_scaled, y_train)))
print('Accuracy of XGBoost model on test set:     {:.3f}'.format(xgb_model.score(X_test_scaled, y_test)))

model_result4 = xgb_model.score(X_test_scaled, y_test)
model_result4 = round(model_result4, 4)
print("Overall Accuracy of the model is ", model_result4)
```

```
Accuracy of XGBoost model on training set: 0.759
Accuracy of XGBoost model on test set:     0.758
Overall Accuracy of the model is  0.7584
```

In [176]:
```
# Confusion Matrix

confusionmatrix_LR = confusion_matrix(y_test, y_pred)

fig = sns.heatmap(confusionmatrix_LR, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['Under Price','Over Price'])
fig.yaxis.set_ticklabels(['Under Price','Over Price'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the XGBoost Model')
sns.set(font_scale=1.0)
```

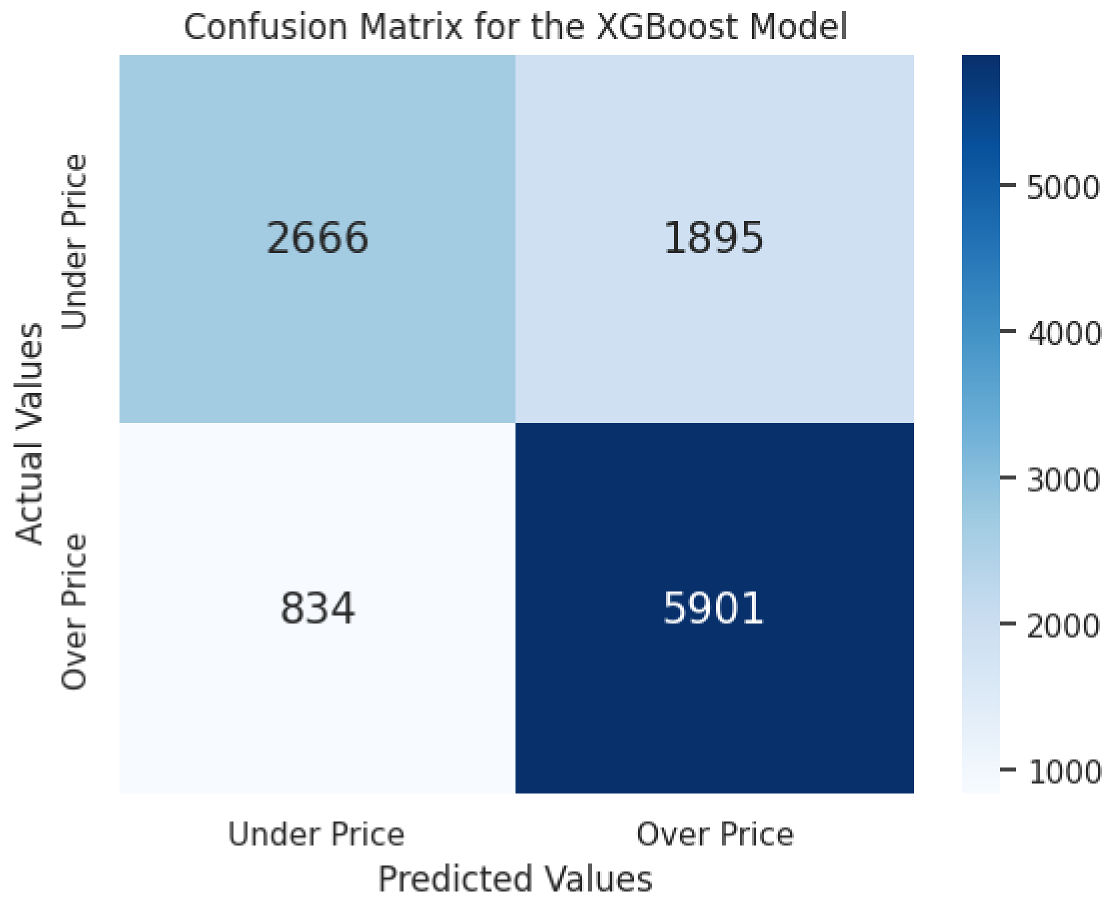

Figure 14. Confusion Matrix for XGBoost Classifier

In [177]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (xgb_model, metrics.classification_report(y_test, y_pred)))
```

```
 Classification report XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...):
              precision    recall  f1-score   support

           0       0.76      0.58      0.66      4561
           1       0.76      0.88      0.81      6735

    accuracy                           0.76     11296
   macro avg       0.76      0.73      0.74     11296
weighted avg       0.76      0.76      0.75     11296
```

Feature Importance for XGBoost model

In [178]:
```
feature_importances = pd.Series(xgb_model.feature_importances_, index=X.columns)
feature_importances.nlargest(10).plot(kind='barh', color="darkblue", title = "Feature Importance for XGBoost
```

Out[178]:
```
<Axes: title={'center': 'Feature Importance for XGBoost model'}>
```

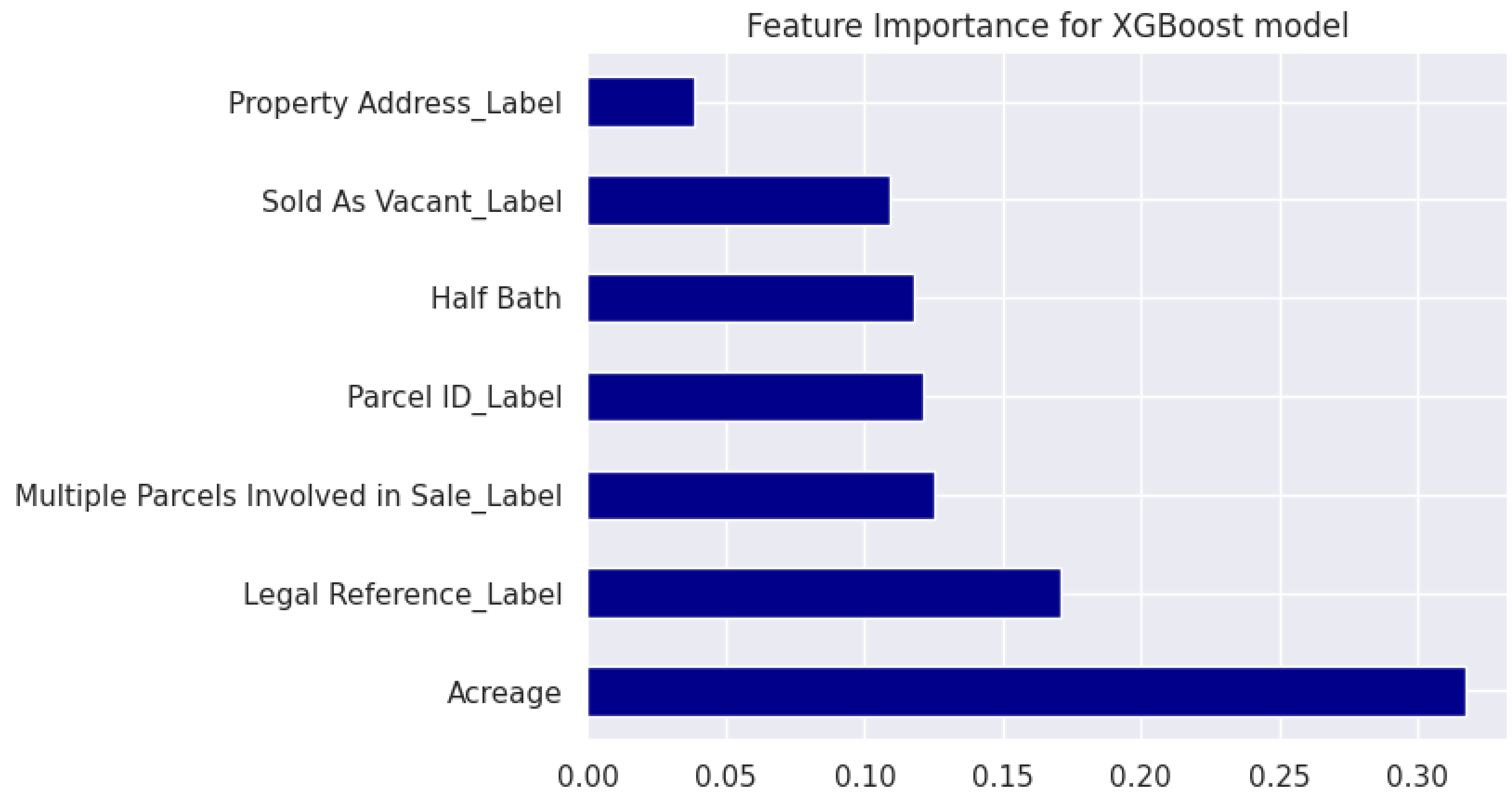

Figure 15. Feature Importance for XGBoost Model

Feature Importance Score

In [179]:
```
# extracting feature importance

feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance Score' : np.round(xgb_model.featur
feature_importance.sort_values('Importance Score', ascending=False, inplace = True)
print(feature_importance)
```

```
                                   Feature  Importance Score
0                                  Acreage             0.317
4                    Legal Reference_Label             0.171
6  Multiple Parcels Involved in Sale_Label             0.125
2                          Parcel ID_Label             0.121
1                                Half Bath             0.118
5                     Sold As Vacant_Label             0.109
3                   Property Address_Label             0.039
```

Table 15. Feature Importance for XGBoost model

Accuracy metric:

From the above evaluation metrics it is observed that the XGBoost model performed well in classifying the price range of the house with an accuracy of **75.9% for training data and 75.8% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy between the training and testing data is almost the same which indicates that the model is neither overfitted nor underfitted and it is able to make predictions on the training data as well as accurately predict on the new or unseen data.

Confusion matrix:

The confusion matrix of the XGBoost model indicates that the **under price category is correctly classified 2666 times** whereas the **over price class is correctly classified 5901 times**, which is a good percent of values where the data is been correctly classified. However, the **under price category is wrongly classified times as over price class 1895 times** and **over price class is classified as under price category 834 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **under price and over price is 76% and 76% respectively**, and as it can be observed the precision score is the same for the under price and over price category. Similarly, the recall score for the **under price class is 58%** and for **over price class is 88%** which indicates that the recall score is good for the over price category but has a low score for the under price class.

Feature Importance Score:

The feature importance graph shows that variables **Acerage, Legal Reference, and Multiple Parcels Involved in Sale** have the highest feature importance with **score of 0.317, 0.171, and 0.125 respectively** indicating that the model classification for house price is based on these features and that they are highly contributing in the prediction of the target variable. Thus, **it is recommended that the company should focus on these parameters and features while analyzing the house prices.**

XGBoost model in comparison with Logistic Regression, Decision Tree and Random Forest model

---

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression, Decision Tree, and Random Forest model, it is observed that **XGBoost model performed better in classifying the house price range** as compared to the Logistic Regression, Decision Tree, and Random Forest model.
- XGBoost model **performed well** as compared to the remaining three models.
- This is because the **accuracy of the training and testing data is much better with respect to the XGBoost model** as compared to that of the Logistic Regression, Decision Tree, and Random Forest model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the XGBoost model as compared to the LR, Decision Tree, and Random Forest model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be effective in analyzing the data.
- Also, the **precision and recall score is relatively high for the XGBoost model** in comparison with the other models built, indicating that the prediction of classes is better performed in XGBoost model.

Step 5: Model Building - Neural Network

Task 6: Build a Neural Network model and compare the results with those of the previous model.

Building a Neural Network model to identify overpricing/underpricing.

In [23]:
```
nnmodel = keras.Sequential([
    keras.layers.Dense(64, activation='tanh', input_shape=(X_train_scaled.shape[1],)),
    keras.layers.Dense(32, activation='tanh'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Fitting the Neural Network model

The Neural Network model is fit with a batch size of 32, validation split of 0.2, and 10 epochs.

In [24]:
```
nnmodel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [182]:
```
nnmodel.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
1130/1130 [==============================] - 5s 3ms/step - loss: 0.6106 - accuracy: 0.6552 - val_loss: 0.591
0 - val_accuracy: 0.6736
Epoch 2/10
1130/1130 [==============================] - 3s 2ms/step - loss: 0.5780 - accuracy: 0.6898 - val_loss: 0.576
9 - val_accuracy: 0.6975
Epoch 3/10
1130/1130 [==============================] - 2s 2ms/step - loss: 0.5713 - accuracy: 0.6987 - val_loss: 0.575
8 - val_accuracy: 0.6949
Epoch 4/10
1130/1130 [==============================] - 3s 2ms/step - loss: 0.5685 - accuracy: 0.7005 - val_loss: 0.574
0 - val_accuracy: 0.6943
Epoch 5/10
1130/1130 [==============================] - 3s 2ms/step - loss: 0.5663 - accuracy: 0.7011 - val_loss: 0.571
7 - val_accuracy: 0.7036
Epoch 6/10
1130/1130 [==============================] - 4s 3ms/step - loss: 0.5643 - accuracy: 0.7035 - val_loss: 0.568
2 - val_accuracy: 0.7053
Epoch 7/10
1130/1130 [==============================] - 2s 2ms/step - loss: 0.5620 - accuracy: 0.7045 - val_loss: 0.570
9 - val_accuracy: 0.6995
Epoch 8/10
1130/1130 [==============================] - 2s 2ms/step - loss: 0.5605 - accuracy: 0.7054 - val_loss: 0.568
0 - val_accuracy: 0.7081
Epoch 9/10
1130/1130 [==============================] - 2s 2ms/step - loss: 0.5581 - accuracy: 0.7097 - val_loss: 0.567
9 - val_accuracy: 0.7020
Epoch 10/10
1130/1130 [==============================] - 2s 2ms/step - loss: 0.5568 - accuracy: 0.7076 - val_loss: 0.562
8 - val_accuracy: 0.7076
```

Out[182]:
```
<keras.callbacks.History at 0x7f36a91c1a80>
```

In [183]:
```
test_loss, test_acc = nnmodel.evaluate(X_test_scaled, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
353/353 [==============================] - 1s 2ms/step - loss: 0.5523 - accuracy: 0.7172
Test loss: 0.5523316860198975
Test accuracy: 0.7172450423240662
```

Model Testing

In [184]:
```
y_pred = nnmodel.predict(X_test_scaled)
```

```
353/353 [==============================] - 1s 1ms/step
```

Evaluating the performance of the model

---

1. Accuracy of the model
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [185]:
```
# converting predicted values to binary values

y_pred = (y_pred > 0.5).astype(int)
```

In [186]:
```
# Confusion Matrix

confusionmatrix_LR = confusion_matrix(y_test, y_pred)
fig = sns.heatmap(confusionmatrix_LR, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['Under Price','Over Price'])
fig.yaxis.set_ticklabels(['Under Price','Over Price'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the Neural Network Model')
sns.set(font_scale=1.0)
```

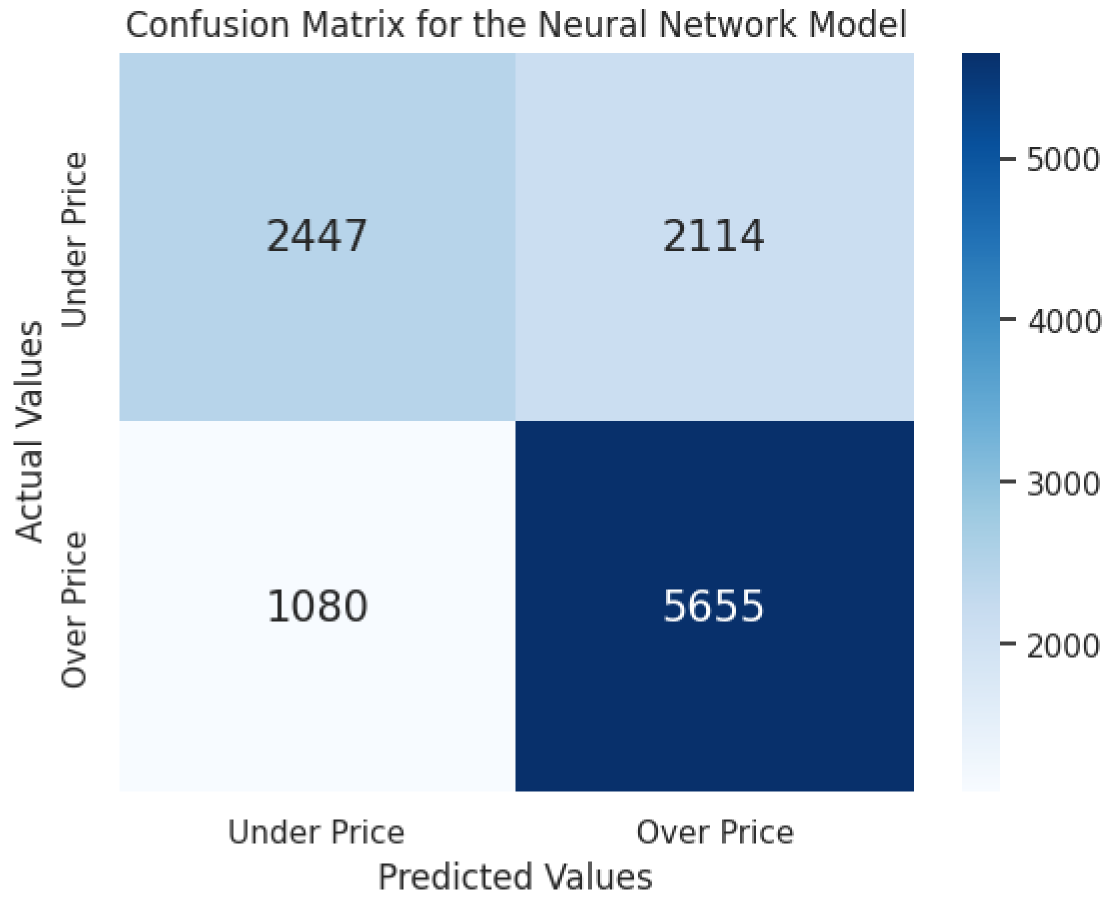

Figure 16. Confusion Matrix for Neural Network model

In [187]:
```python
# Classification Report

print("\n Classification report %s:\n%s\n" % (nnmodel, metrics.classification_report(y_test, y_pred)))
```

```
 Classification report &lt;keras.engine.sequential.Sequential object at 0x7f36a9236050&gt;:
              precision    recall  f1-score   support

           0       0.69      0.54      0.61      4561
           1       0.73      0.84      0.78      6735

    accuracy                           0.72     11296
   macro avg       0.71      0.69      0.69     11296
weighted avg       0.71      0.72      0.71     11296
```

Feature Importance for Neural Network model

In [188]:
```python
nnmodel_weights = nnmodel.get_weights()
w = nnmodel_weights[0]
feature_importance_nn = np.mean(np.abs(w), axis=1)
feature_importances_nn = pd.Series(feature_importance_nn, index=X.columns)
feature_importances_nn.nlargest(10).plot(kind='barh', color="darkblue", title = "Feature Importance for Neura
```

Out[188]: <Axes: title={'center': 'Feature Importance for Neural Network model'}>

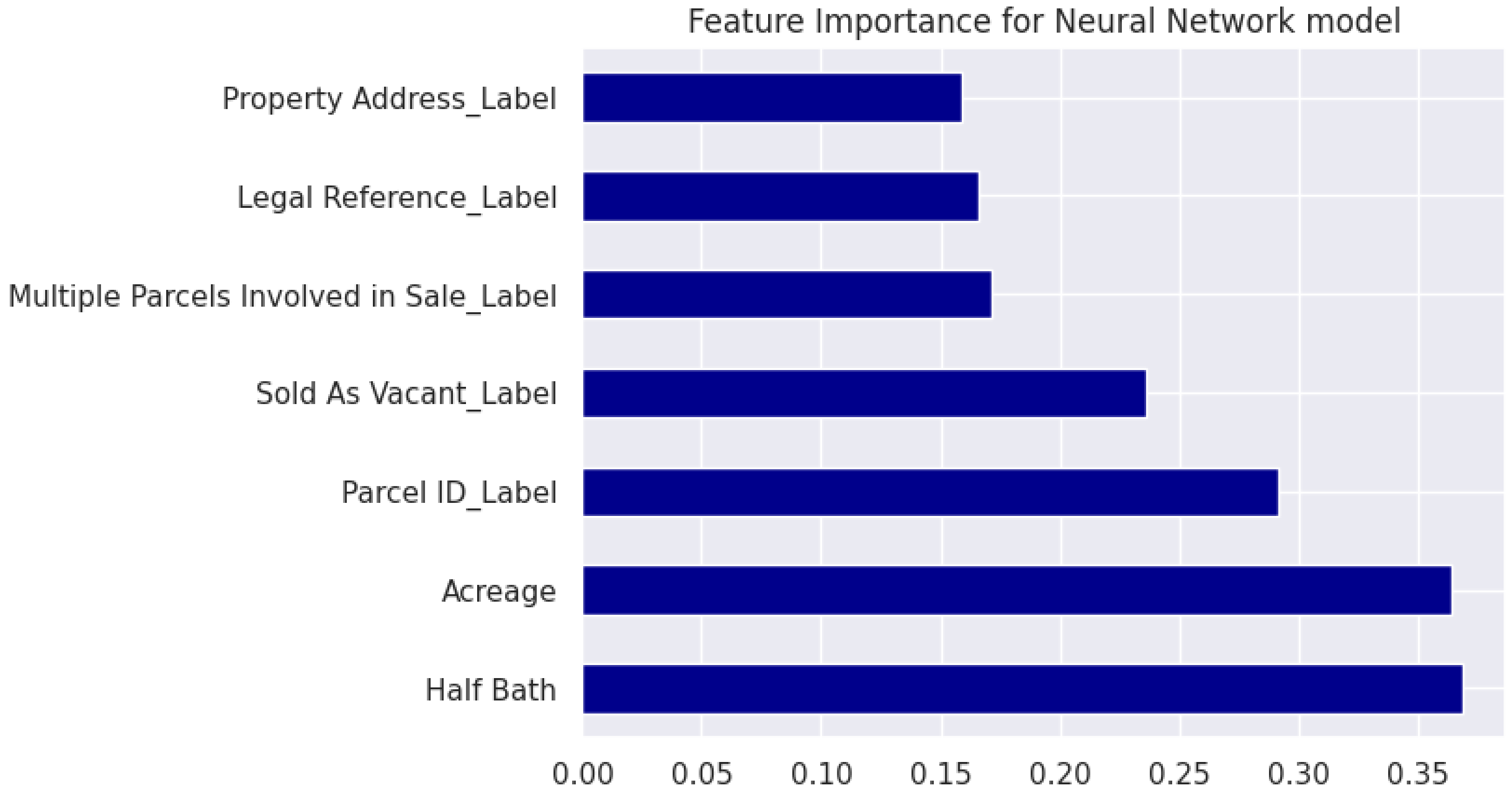

Figure 17. Feature Importance for Neural Network model

Accuracy metric:

From the above evaluation metrics it is observed that the Neural Network model performed well in classifying the price range of the house with a test accuracy of **71.2% and test loss of 55.3%,** which indicates that the model has good accuracy in classification.

Confusion matrix:

The confusion matrix of the Neural Network model indicates that the **under price category is correctly classified 2474 times** whereas the **over price class is correctly classified 5565 times**, which is a good percent of values where the data is been correctly classified. However, the **under price category is wrongly classified times as over price class 2087 times** and **over price class is classified as under price category 1170 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **under price and over price is 68% and 73% respectively**, and as it can be observed the precision score is slightly different for the under price and over price category. Similarly, the recall score for the **under price class is 54%** and for **over price class is 83%** which indicates that the recall score is good for the over price category but has a low score for the under price class.

Feature Importance Score:

The feature importance graph shows that variables **Acerage, Half Bath, and Parcel ID** have the highest feature importance with **score of 0.38, 0.36, and 0.28 respectively** indicating that the model classification for house price is based on these features and that they are highly contributing in the prediction of the target variable. Thus, **it is recommended that the company should focus on these parameters and features while analyzing the house prices.**

Neural Network model in comparison with Logistic Regression, Decision Tree, Random Forest, and XGBoost model

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression, Decision Tree, Random Forest, and XGBoost model, it is observed that **XGBoost model performed better in classifying the house price range** as compared to the Logistic Regression, Decision Tree, Random Forest, and Neural Network model.
- XGBoost model **performed well** as compared to the remaining three models. Although, Neural Network performed slightly well as compared to LR, Decision Tree, and Random Forest model.
- This is because the **accuracy of the training and testing data is much better with respect to the XGBoost model** as compared to that of the Logistic Regression, Decision Tree, Random Forest, and Neural Network model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the XGBoost model as compared to the LR, Decision Tree, Random Forest, and Neural Network model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be effective in analyzing the data.
- Also, the **precision and recall score is relatively high for the XGBoost model** in comparison with the other models built, indicating that the prediction of classes is better performed in XGBoost model.

Results

Task 7: Use multiple benchmarking metrics to compare and contrast the five models. Based on your findings, provide evidence of which model you believe the real estate company should use and what are the key variables to focus on to drive value and how can they get the most value out of the houses they should be targeting.

1. Comparing the accuracy of the models built (accuracy, precision, recall)

In [110]:
```
metrics_data = []
metrics_data.append(['Logistic Regression Model', model_result1, '69%', '61%', 'Sold As Vacant, Half Bath, Mu
metrics_data.append(['Decision Tree Model', model_result2, '73%', '77%', 'Acreage, Legal Reference, Parcel ID
metrics_data.append(['Random Forest Model', model_result3, '78%', '63%', 'Acreage, Half Bath, Legal Reference
metrics_data.append(['XGBoost Model', model_result4, '76%', '88%', 'Acreage, Legal Reference, Multiple Parcel
metrics_data.append(['Neural Network Model', test_acc, '73%', '83%', 'Acerage, Half Bath, Parcel ID'])
metrics_df = pd.DataFrame(metrics_data, columns=['Model', 'Accuracy', 'Precision', 'Recall', 'Significant Var
metrics_df = metrics_df.sort_values("Accuracy", ascending=False)
metrics_df
```

Out[110]:

	Model	Accuracy	Precision	Recall	Significant Variables
3	XGBoost Model	0.758400	76%	88%	Acreage, Legal Reference, Multiple Parcels Inv...
4	Neural Network Model	0.711668	73%	83%	Acerage, Half Bath, Parcel ID
1	Decision Tree Model	0.695100	73%	77%	Acreage, Legal Reference, Parcel ID
2	Random Forest Model	0.671700	78%	63%	Acreage, Half Bath, Legal Reference
0	Logistic Regression Model	0.606500	69%	61%	Sold As Vacant, Half Bath, Multiple Parcels In...

Table 13. Model Comparison Table (Ranked based on Accuracy Metric)

2. Discussing the models to be recommended based on the evaluation metrics

Based on the evaluation metrics as shown in the table above, it is observed that **XGBoost model has performed the best** as compared to the Logistic Regression, Decision Tree, Random Forest, and Neural Network models for the classification of the house price range. This is because the accuracy and precision - recall score for the XGBoost model is higher as compared to the other models and hence the **model that should be recommended to the real estate company to be used based on the evaluation metrics is the XGBoost model.**

3. Discussing the key variables they should focus on their business context

The key variables based on the model selection, which is the XGBoost model are **'Acreage', 'Legal Reference', 'Multiple Parcels Involved in Sale'.** The feature importance score for the XGBoost model for the three variables is higher as compared to the other models and hence they are the key variables that the real estate company should focus on for their business context.

Conclusion

Recommendations

- Based on the analysis and results, we conclude that XGBoost model is recommended to be implemented for the classification of the house price range. The XGBoost model is **neither overfitted nor underfitted** based on the accuracy values of the train and test dataset, but the performance of the model can be improved such that it can be used in future for further prediction and classification, which can be done by updating new features and data points that will increase the efficiency and performance of the model, implying a *best-fit model for training*.
- The features that should be focused upon are **'Acreage', 'Legal Reference', 'Multiple Parcels Involved in Sale'** based on the feature importance score of the XGBoost model. Hence, the company should focus on these features for analyzing the house price range and understanding whether the price of the house falls in the under price category or over price category, as these features influence the classification of the price range.

Future Scope

The XGBoost model selected is able to classify the target variable and also extract the features that contribute to the prediction, but the performance and efficiency of the model can be improved and thus requires reevaluating the performance of the model by adding new features to the model and increasing the training data. Based on the results, it is observed that the dataset has high amount of null values and thus it is important that both the quantity and quality of data is improved to increase the efficiency of the model.

Thus, the model can be improved and updated based on new features being added to the dataset that can help to better analyze the target variable.

References

[1] What is Logistic regression? | IBM. (n.d.). https://www.ibm.com/topics/logistic-regression (https://www.ibm.com/topics/logistic-regression)

[2] GeeksforGeeks. (2023). Decision Tree. GeeksforGeeks. https://www.geeksforgeeks.org/decision-tree/ (https://www.geeksforgeeks.org/decision-tree/)

[3] Yiu, T. (2021, December 10). Understanding Random Forest - Towards Data Science. Medium. https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (https://towardsdatascience.com/understanding-random-forest-58381e0602d2)

[4] Aliyev, V. (2021, December 15). Gradient Boosting Classification explained through Python. Medium. https://towardsdatascience.com/gradient-boosting-classification-explained-through-python-60cc980eeb3d (https://towardsdatascience.com/gradient-boosting-classification-explained-through-python-60cc980eeb3d)

[5] Knocklein, O. (2021, December 10). Classification Using Neural Networks - Towards Data Science. Medium. https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f (https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f)

[6] Singh, K. (2022). How to Improve Class Imbalance using Class Weights in Machine Learning. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/ (https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/)