Investing in Nashville using Machine Learning

Syed Faizan

Problem Statement:

Imagine 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. By looking at the variable, Sale Price Compared To Value, that will help us see which properties are being over/under valued. If we can build an accurate model, this could help the company identify what the key factors in finding the best deal may be.

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Set pandas options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

import pandas as pd

file_path = r'C:\\Users\\sfaiz\\OneDrive\\Desktop\\Module 4 ALY 6020\\
week 4 - Nashville_housing_data.csv'
df = pd.read_csv(file_path)

# Display the first few rows to verify successful loading
print(df.head())
```

```
   Unnamed: 0        Parcel ID       Land Use   Property Address  \
0           1  105 11 0 080.00  SINGLE FAMILY   1802  STEWART PL
1           2  118 03 0 130.00  SINGLE FAMILY  2761  ROSEDALE PL
2           3  119 01 0 479.00  SINGLE FAMILY  224  PEACHTREE ST
3           4  119 05 0 186.00  SINGLE FAMILY      316  LUTIE ST
4           5  119 05 0 387.00  SINGLE FAMILY   2626  FOSTER AVE

   Suite/ Condo   # Property City  Sale Date   Legal Reference Sold As
Vacant  \
0              NaN     NASHVILLE  1/11/2013  20130118-0006337
No
1              NaN     NASHVILLE  1/18/2013  20130124-0008033
No
2              NaN     NASHVILLE  1/18/2013  20130128-0008863
No
3              NaN     NASHVILLE  1/23/2013  20130131-0009929
```

```
No
4                NaN     NASHVILLE   1/4/2013  20130118-0006110
No

  Multiple Parcels Involved in Sale       City State  Acreage  \
0                                No  NASHVILLE    TN     0.17
1                                No  NASHVILLE    TN     0.11
2                                No  NASHVILLE    TN     0.17
3                                No  NASHVILLE    TN     0.34
4                                No  NASHVILLE    TN     0.17

              Tax District  Neighborhood  Land Value  Building
Value  \
0  URBAN SERVICES DISTRICT          3127       32000          134400

1       CITY OF BERRY HILL          9126       34000          157800

2  URBAN SERVICES DISTRICT          3130       25000          243700

3  URBAN SERVICES DISTRICT          3130       25000          138100

4  URBAN SERVICES DISTRICT          3130       25000           86100


   Finished Area Foundation Type  Year Built Exterior Wall Grade
Bedrooms  \
0     1149.00000         PT BSMT        1941         BRICK  C
2.0
1     2090.82495            SLAB        2000   BRICK/FRAME  C
3.0
2     2145.60001       FULL BSMT        1948   BRICK/FRAME  B
4.0
3     1969.00000           CRAWL        1910         FRAME  C
2.0
4     1037.00000           CRAWL        1945         FRAME  C
2.0

   Full Bath  Half Bath Sale Price Compared To Value
0        1.0        0.0                         Over
1        2.0        1.0                         Over
2        2.0        0.0                        Under
3        1.0        0.0                        Under
4        1.0        0.0                        Under
```

```
df.shape
```

```
(22651, 26)
```

```
df.columns.tolist()
```

```
['Unnamed: 0',
 'Parcel ID',
 'Land Use',
 'Property Address',
 'Suite/ Condo   #',
 'Property City',
 'Sale Date',
 'Legal Reference',
 'Sold As Vacant',
 'Multiple Parcels Involved in Sale',
 'City',
 'State',
 'Acreage',
 'Tax District',
 'Neighborhood',
 'Land Value',
 'Building Value',
 'Finished Area',
 'Foundation Type',
 'Year Built',
 'Exterior Wall',
 'Grade',
 'Bedrooms',
 'Full Bath',
 'Half Bath',
 'Sale Price Compared To Value']
```

Removing the columns with large number of missing values and columns not relevant to our analysis right at the outset

```
# Display the missing values summary
# Check for missing values in the DataFrame
missing_values = df.isnull().sum()

# Display missing values for each column
missing_values_summary = pd.DataFrame({
    'Column Name': df.columns,
    'Missing Values': missing_values,
    'Percentage (%)': (missing_values / len(df)) * 100
}).sort_values(by='Percentage (%)', ascending=False)
print(missing_values_summary)
```

```
                                                          Column
Name  \
```

```
Suite/ Condo   #                                  Suite/ Condo   #
Half Bath                                                 Half Bath
Bedrooms                                                   Bedrooms
Property Address                                   Property Address
Property City                                         Property City
Full Bath                                                 Full Bath
Foundation Type                                     Foundation Type
Finished Area                                         Finished Area
Unnamed: 0                                               Unnamed: 0
Land Value                                               Land Value
Grade                                                         Grade
Exterior Wall                                         Exterior Wall
Year Built                                               Year Built
Building Value                                       Building Value
Tax District                                           Tax District
Neighborhood                                           Neighborhood
Parcel ID                                                 Parcel ID
Acreage                                                     Acreage
State                                                         State
City                                                           City
Multiple Parcels Involved in Sale  Multiple Parcels Involved in Sale
Sold As Vacant                                       Sold As Vacant
Legal Reference                                     Legal Reference
Sale Date                                                 Sale Date
Land Use                                                   Land Use
Sale Price Compared To Value            Sale Price Compared To Value
```

```
                                  Missing Values  Percentage (%)
Suite/ Condo   #                           22651      100.000000
Half Bath                                    108        0.476800
Bedrooms                                       3        0.013244
Property Address                               2        0.008830
Property City                                  2        0.008830
Full Bath                                      1        0.004415
Foundation Type                                1        0.004415
Finished Area                                  1        0.004415
Unnamed: 0                                     0        0.000000
Land Value                                     0        0.000000
Grade                                          0        0.000000
Exterior Wall                                  0        0.000000
Year Built                                     0        0.000000
Building Value                                 0        0.000000
Tax District                                   0        0.000000
Neighborhood                                   0        0.000000
Parcel ID                                      0        0.000000
Acreage                                        0        0.000000
State                                          0        0.000000
City                                           0        0.000000
Multiple Parcels Involved in Sale              0        0.000000
Sold As Vacant                                 0        0.000000
Legal Reference                                0        0.000000
Sale Date                                      0        0.000000
Land Use                                       0        0.000000
Sale Price Compared To Value                   0        0.000000
```

Remove the Suite/ condo column entirely and the other rows with missing values as they are small in number

```python
# Remove the 'Suite/ Condo #' column entirely
df_cleaned = df.drop(columns=['Suite/ Condo   #'])

# Drop rows with missing values
df_cleaned = df_cleaned.dropna()

# Verify that there are no missing values remaining
missing_values_after_cleaning = df_cleaned.isnull().sum()

missing_values_after_cleaning
```

```
Unnamed: 0                            0
Parcel ID                             0
```

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

df1 = df_cleaned
```

df1 is the renominated cleaned Data Frame in Pandas

```
df1.shape

(22536, 25)
```

Removing redundant columns

```
# Remove the specified columns from df1
columns_to_remove = ['Unnamed: 0', 'Parcel ID', 'Property Address',
'Legal Reference',
                     'Property City', 'City', 'State', 'Tax District']

df1 = df1.drop(columns=columns_to_remove)

# Display the remaining columns to confirm
df1.columns.tolist()
```

```
['Land Use',
 'Sale Date',
 'Sold As Vacant',
 'Multiple Parcels Involved in Sale',
 'Acreage',
 'Neighborhood',
 'Land Value',
 'Building Value',
 'Finished Area',
 'Foundation Type',
 'Year Built',
 'Exterior Wall',
 'Grade',
 'Bedrooms',
 'Full Bath',
 'Half Bath',
 'Sale Price Compared To Value']
```

```
df1.shape
```

```
(22536, 17)
```

```
# Check the data types of all variables in df1
data_types = df1.dtypes

# Display the data types
data_types
```

```
Land Use                             object
Sale Date                            object
Sold As Vacant                       object
Multiple Parcels Involved in Sale    object
Acreage                             float64
Neighborhood                          int64
Land Value                            int64
Building Value                        int64
Finished Area                       float64
Foundation Type                      object
Year Built                            int64
Exterior Wall                        object
Grade                                object
Bedrooms                            float64
Full Bath                           float64
Half Bath                           float64
Sale Price Compared To Value         object
dtype: object
```

Automated EDA

We use automated EDA Tools such as Summary Tools, D Tools, SweetViz to create an Exploratory Data Analysis Report.

```
from summarytools import dfSummary
dfSummary(df1)

<pandas.io.formats.style.Styler at 0x204daa92c90>
```

Convert 'Sale Date' into a date using datetime module, then convert it into an integer column called 'Time since sale' which gives time since sale in days. Also, convert 'Year Built' into age.

```
from datetime import datetime

# Convert 'Sale Date' to datetime format
df1['Sale Date'] = pd.to_datetime(df1['Sale Date'], errors='coerce')

# Create a new column 'timesincesale' as the difference between the
present date and 'Sale Date'
df1['timesincesale'] = (datetime.now() - df1['Sale Date']).dt.days

# Remove the old 'Sale Date' column
df1 = df1.drop(columns=['Sale Date'])

# Transform 'Year Built' column into an 'Age' column
df1['Age'] = 2024 - df1['Year Built']

# Drop the original 'Year Built' column
df1 = df1.drop(columns=['Year Built'])

# Display the first few rows to verify the transformation
df1.head()


# Display the first few rows of the updated DataFrame
df1.head()

        Land Use Sold As Vacant Multiple Parcels Involved in Sale
Acreage  \
0  SINGLE FAMILY             No                                  No
0.17
1  SINGLE FAMILY             No                                  No
```

```
0.11
2  SINGLE FAMILY             No                                No
0.17
3  SINGLE FAMILY             No                                No
0.34
4  SINGLE FAMILY             No                                No
0.17

   Neighborhood  Land Value  Building Value  Finished Area Foundation
Type  \
0          3127       32000          134400     1149.00000          PT
BSMT
1          9126       34000          157800     2090.82495
SLAB
2          3130       25000          243700     2145.60001        FULL
BSMT
3          3130       25000          138100     1969.00000
CRAWL
4          3130       25000           86100     1037.00000
CRAWL

  Exterior Wall Grade  Bedrooms  Full Bath  Half Bath  \
0         BRICK  C          2.0        1.0        0.0
1   BRICK/FRAME  C          3.0        2.0        1.0
2   BRICK/FRAME  B          4.0        2.0        0.0
3         FRAME  C          2.0        1.0        0.0
4         FRAME  C          2.0        1.0        0.0

  Sale Price Compared To Value  timesincesale  Age
0                         Over           4347   83
1                         Over           4340   24
2                        Under           4340   76
3                        Under           4335  114
4                        Under           4354   79
```

```
df1.dtypes
```

```
Land Use                                     object
Sold As Vacant                               object
Multiple Parcels Involved in Sale            object
Acreage                                     float64
Neighborhood                                  int64
Land Value                                    int64
Building Value                                int64
Finished Area                               float64
Foundation Type                              object
Exterior Wall                                object
Grade                                        object
Bedrooms                                    float64
Full Bath                                   float64
```

```
Half Bath                                      float64
Sale Price Compared To Value                    object
timesincesale                                    int64
Age                                              int64
dtype: object
```

```
df1.columns.tolist()
```

```
['Land Use',
 'Sold As Vacant',
 'Multiple Parcels Involved in Sale',
 'Acreage',
 'Neighborhood',
 'Land Value',
 'Building Value',
 'Finished Area',
 'Foundation Type',
 'Exterior Wall',
 'Grade',
 'Bedrooms',
 'Full Bath',
 'Half Bath',
 'Sale Price Compared To Value',
 'timesincesale',
 'Age']
```

Use ordinal encoding to encode the 'object' class variables as there is an ordinal relationship between the levels of the categorical variables and one hot encoding produces hyperdimnetionality

```
from sklearn.preprocessing import OrdinalEncoder

# Select columns with object data type
object_columns = df1.select_dtypes(include=['object']).columns

# Apply ordinal encoding to these columns
encoder = OrdinalEncoder()
df1[object_columns] = encoder.fit_transform(df1[object_columns])

# Display the first few rows of the updated DataFrame
df1.head()
```

```
   Land Use  Sold As Vacant  Multiple Parcels Involved in Sale
Acreage  \
0       3.0             0.0                                0.0
0.17
1       3.0             0.0                                0.0
0.11
2       3.0             0.0                                0.0
0.17
```

```
3       3.0             0.0                                0.0
0.34
4       3.0             0.0                                0.0
0.17

   Neighborhood  Land Value  Building Value  Finished Area  Foundation
Type  \
0          3127       32000          134400     1149.00000
3.0
1          9126       34000          157800     2090.82495
4.0
2          3130       25000          243700     2145.60001
1.0
3          3130       25000          138100     1969.00000
0.0
4          3130       25000           86100     1037.00000
0.0

   Exterior Wall  Grade  Bedrooms  Full Bath  Half Bath  \
0            0.0    2.0       2.0        1.0        0.0
1            1.0    2.0       3.0        2.0        1.0
2            1.0    1.0       4.0        2.0        0.0
3            3.0    2.0       2.0        1.0        0.0
4            3.0    2.0       2.0        1.0        0.0

   Sale Price Compared To Value  timesincesale  Age
0                           0.0           4347   83
1                           0.0           4340   24
2                           1.0           4340   76
3                           1.0           4335  114
4                           1.0           4354   79
```

Scaling all features via normalization

```
from sklearn.preprocessing import MinMaxScaler

# Select numeric columns for scaling
numeric_columns = df1.select_dtypes(include=['float64',
'int64']).columns

# Apply MinMaxScaler for normalization
scaler = MinMaxScaler()
df1[numeric_columns] = scaler.fit_transform(df1[numeric_columns])

# Display the first few rows of the normalized DataFrame
df1.head()
```

```
   Land Use  Sold As Vacant  Multiple Parcels Involved in Sale  
Acreage  \
0       1.0             0.0                                0.0  
0.007446   
1       1.0             0.0                                0.0  
0.004009   
2       1.0             0.0                                0.0  
0.007446   
3       1.0             0.0                                0.0  
0.017182   
4       1.0             0.0                                0.0  
0.007446   

   Neighborhood  Land Value  Building Value  Finished Area  Foundation 
Type  \
0      0.320492    0.016648        0.022841       0.036258             
0.6   
1      0.957126    0.017719        0.026859       0.085113             
0.8   
2      0.320811    0.012901        0.041612       0.087954             
0.2   
3      0.320811    0.012901        0.023476       0.078793             
0.0   
4      0.320811    0.012901        0.014546       0.030449             
0.0   

   Exterior Wall     Grade  Bedrooms  Full Bath  Half Bath  \
0          0.000  0.285714  0.181818        0.1   0.000000   
1          0.125  0.285714  0.272727        0.2   0.333333   
2          0.125  0.142857  0.363636        0.2   0.000000   
3          0.375  0.285714  0.181818        0.1   0.000000   
4          0.375  0.285714  0.181818        0.1   0.000000   

   Sale Price Compared To Value  timesincesale       Age  
0                           0.0       0.993562  0.410811  
1                           0.0       0.988555  0.091892  
2                           1.0       0.988555  0.372973  
3                           1.0       0.984979  0.578378  
4                           1.0       0.998569  0.389189  
```

```python
from IPython.display import IFrame

# Display the report in an iframe
IFrame(src='df1.html', width=1050, height=700)
```

```
<IPython.lib.display.IFrame at 0x204db879850>
```

```python
import sweetviz as sv


my_report = sv.analyze(df1)
```

```
# Generate the report and save it as an HTML file
my_report.show_html("df1.html")
IFrame(src='df1.html', width=1050, height=700)
```

{"model_id":"818faf84d6894eea9ea95c686f61c442","version_major":2,"version_minor":0}

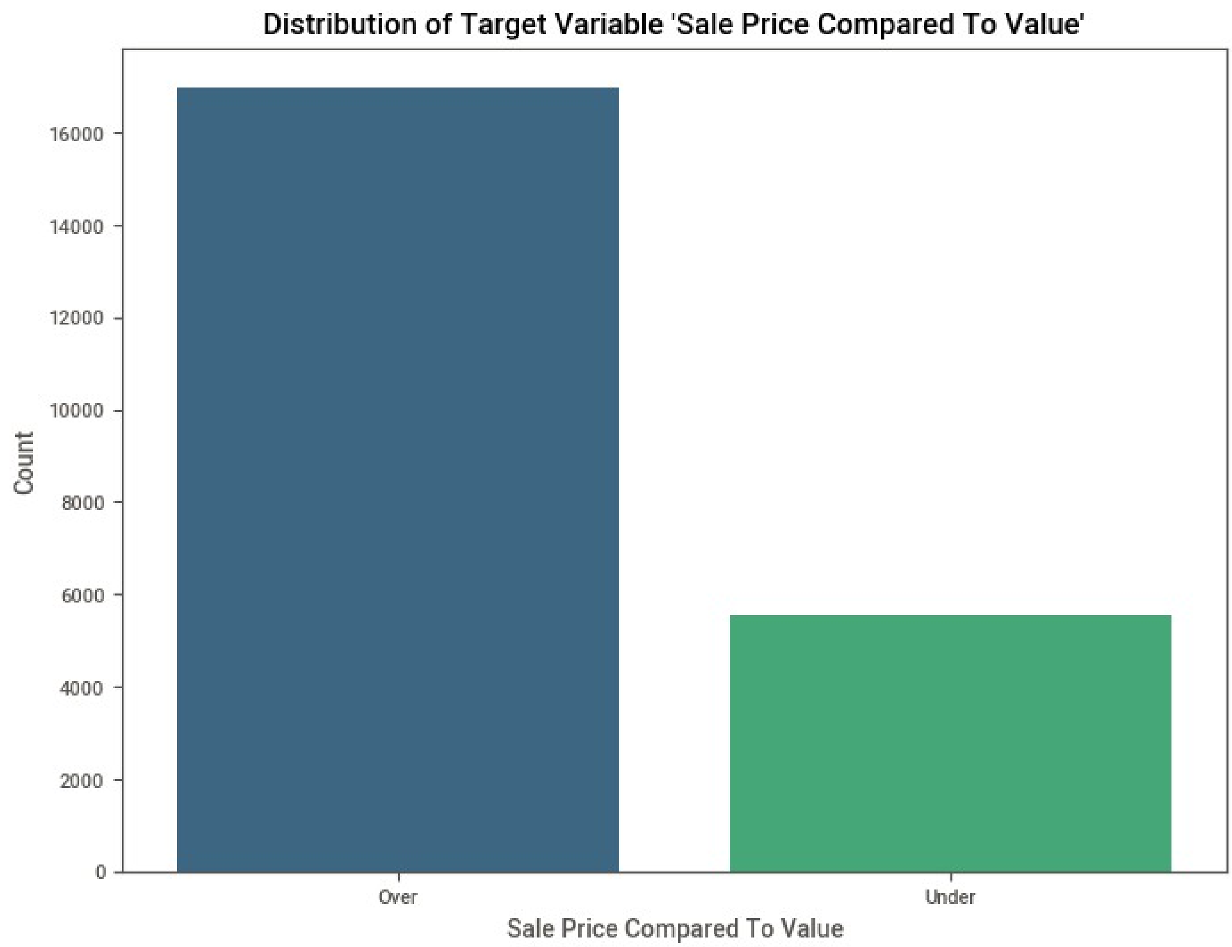
```
Report df1.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS saved in your notebook/colab files.
```

```
<IPython.lib.display.IFrame at 0x204e0ef3d40>
```

Examining the target variable for data imbalance

```
import matplotlib.pyplot as plt
import seaborn as sns

# Count the occurrences of each class in the target variable
response_counts = df1['Sale Price Compared To Value'].value_counts()

# Plot the imbalance
plt.figure(figsize=(8, 6))
sns.barplot(x=response_counts.index, y=response_counts.values, palette="viridis", legend=False)

# Add labels and title
plt.title("Distribution of Target Variable 'Sale Price Compared To Value'")
plt.xlabel("Sale Price Compared To Value")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=["Over", "Under"], rotation=0)
plt.show()
```

```
C:\Users\sfaiz\AppData\Local\Temp\ipykernel_159280\2535649754.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=response_counts.index, y=response_counts.values, palette="viridis", legend=False)
```

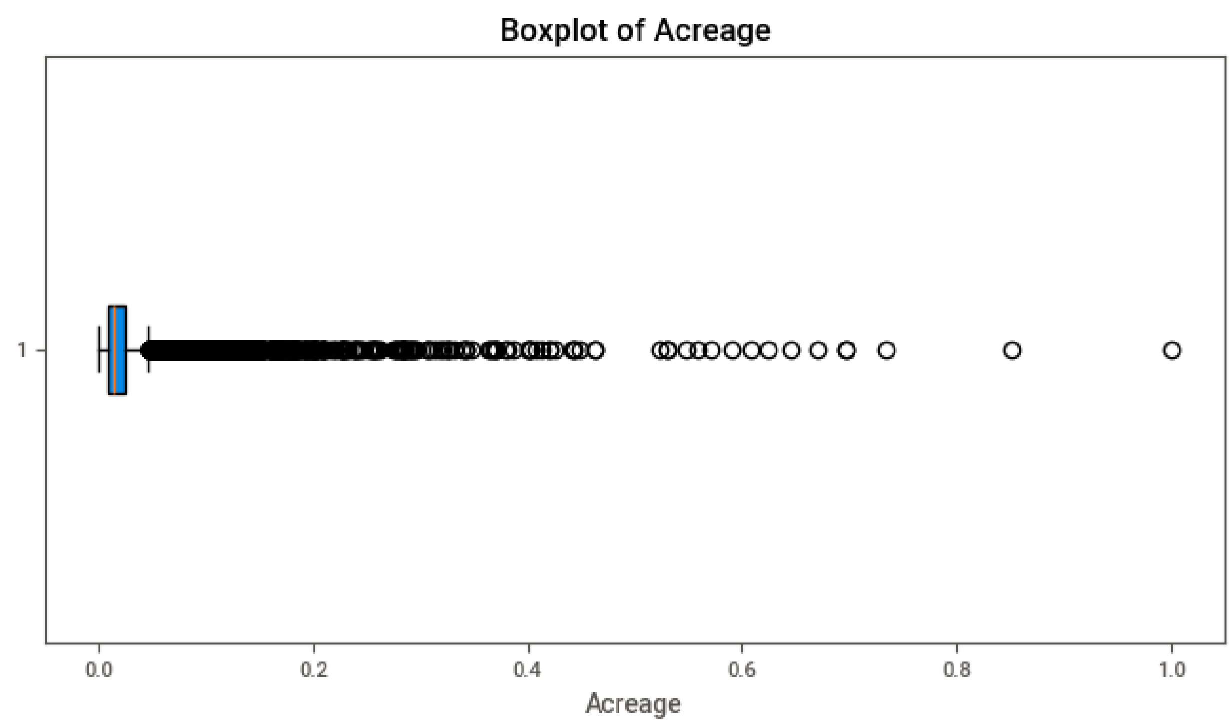



```
import numpy as np
df1['Sale Price Compared To Value'].value_counts()

Sale Price Compared To Value
0.0    16979
1.0     5557
Name: count, dtype: int64
```

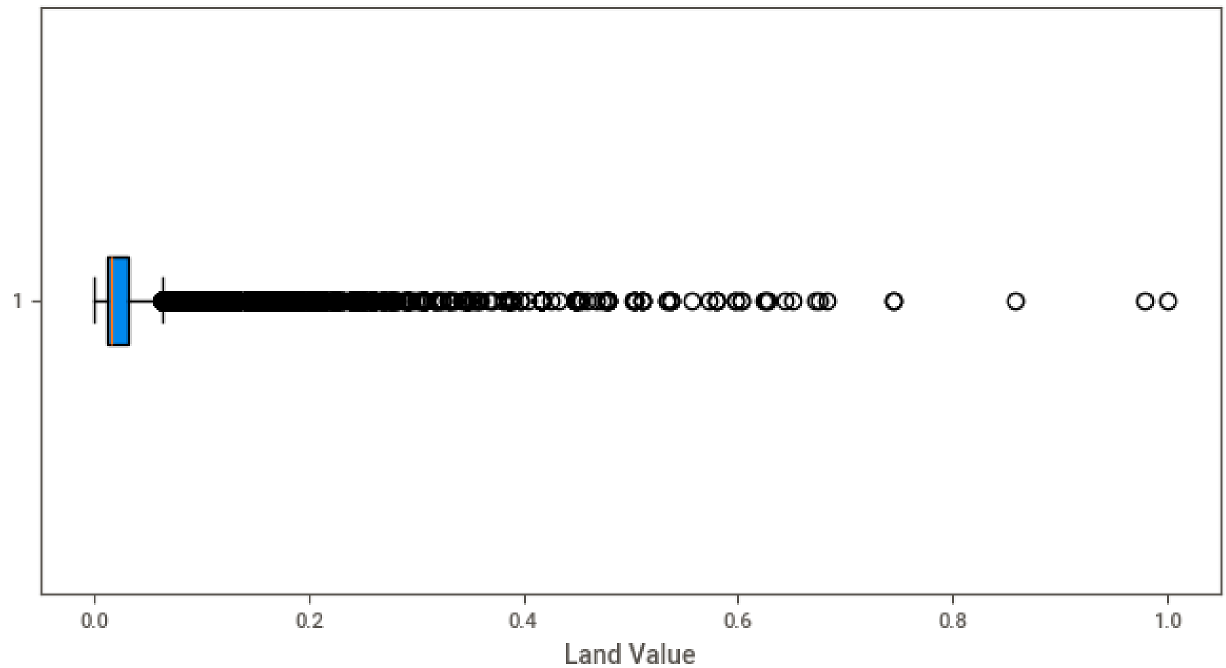
Outlier detection

```
# Specify the numerical features explicitly
numerical_features = [
    'Acreage', 'Land Value', 'Building Value',
    'Finished Area', 'Bedrooms', 'Full Bath', 'Half Bath',
'timesincesale', 'Age'
]

# Create boxplots for each numerical feature
for column in numerical_features:
```

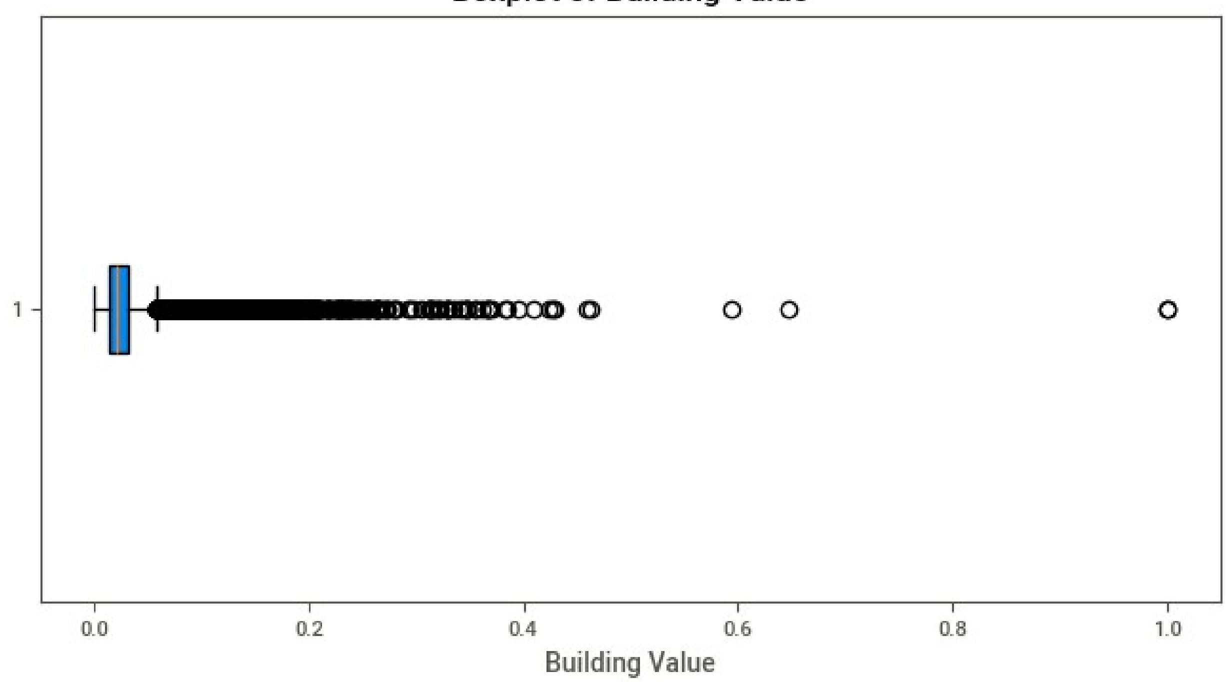
```
    plt.figure(figsize=(8, 4))
    plt.boxplot(df1[column], vert=False, patch_artist=True)
    plt.title(f'Boxplot of {column}')
    plt.xlabel(column)
    plt.show()

# Identify outliers using the 1.5 IQR rule for each specified
numerical feature
outliers = {}
for column in numerical_features:
    Q1 = df1[column].quantile(0.25)
    Q3 = df1[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers[column] = df1[(df1[column] < lower_bound) | (df1[column]
> upper_bound)][column]

# Display the number of outliers for each numerical feature
outlier_summary = {col: len(outliers[col]) for col in
numerical_features}
outlier_summary
```

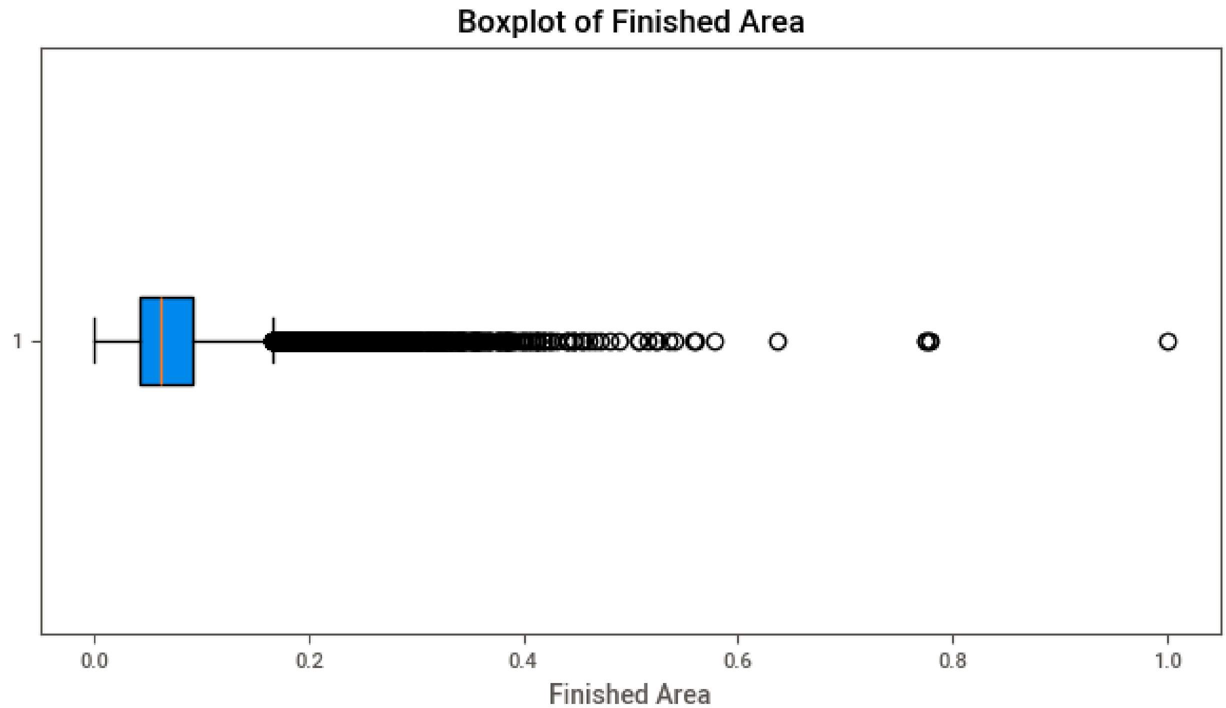


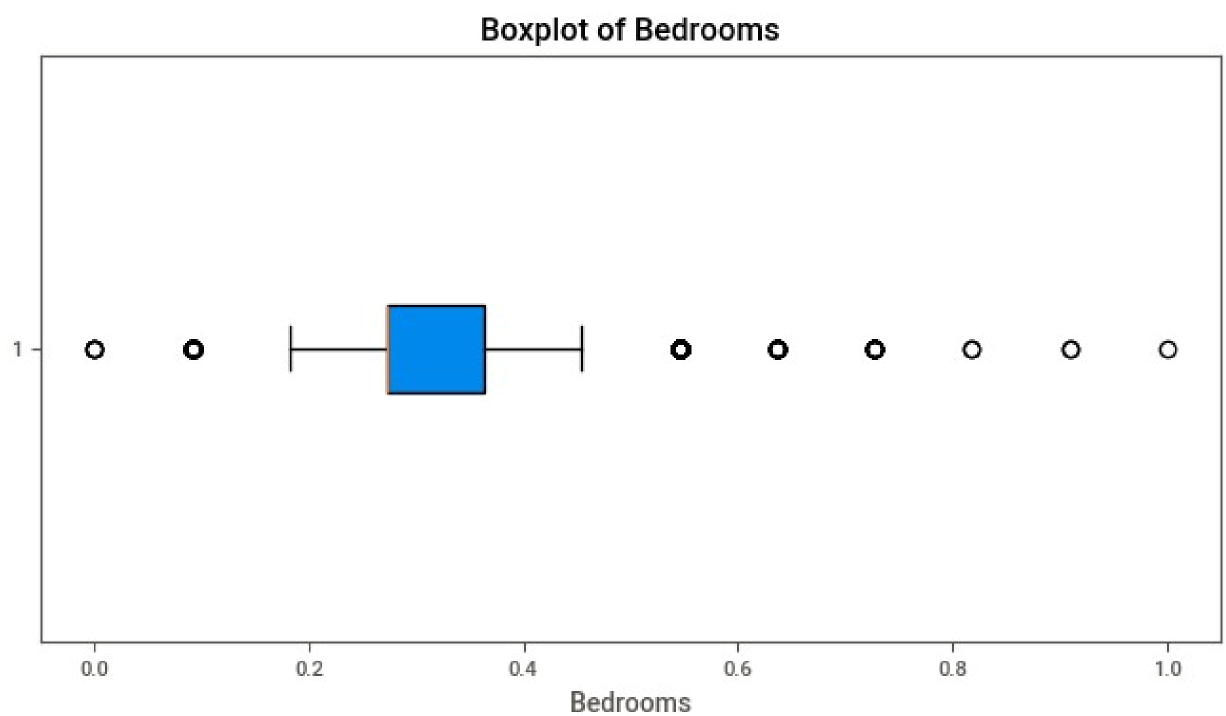
Boxplot of Land Value

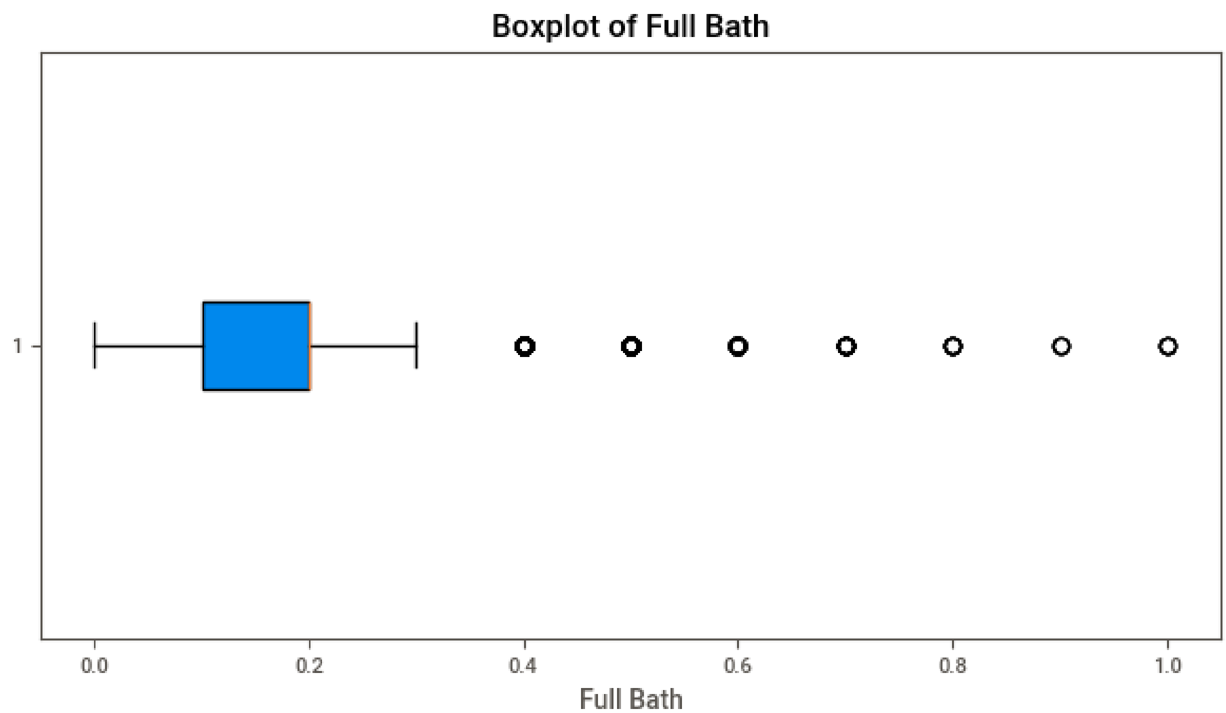


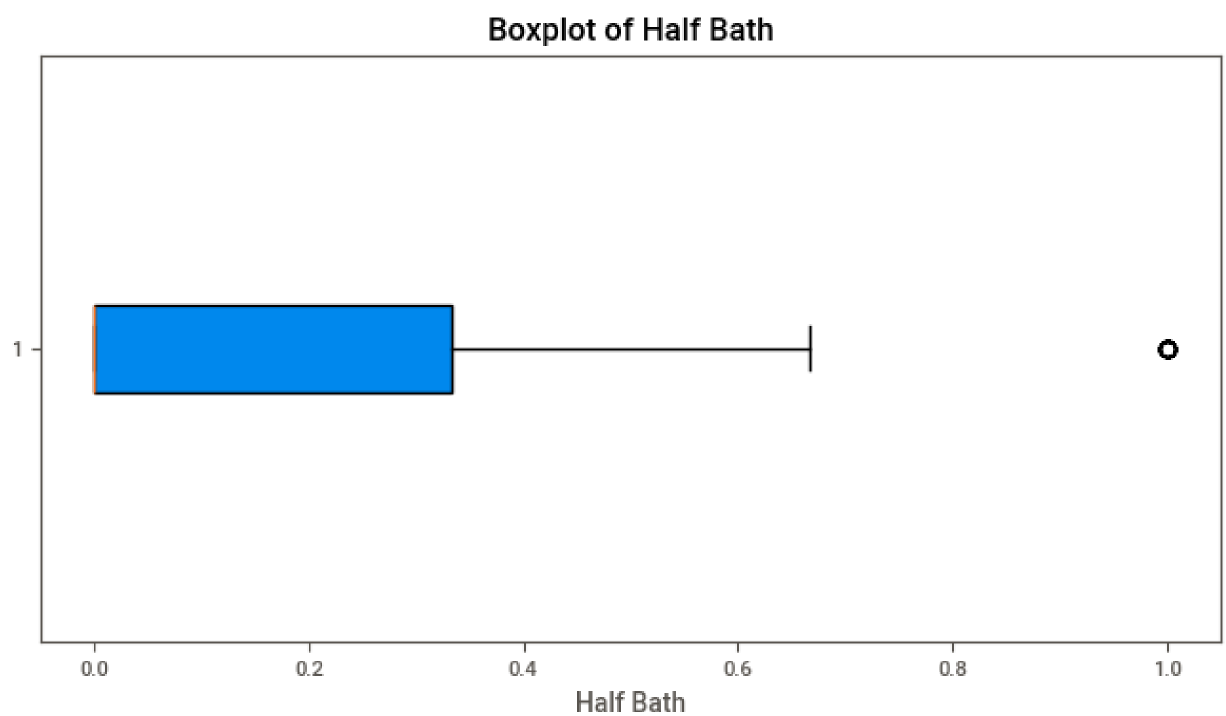
Boxplot of Building Value

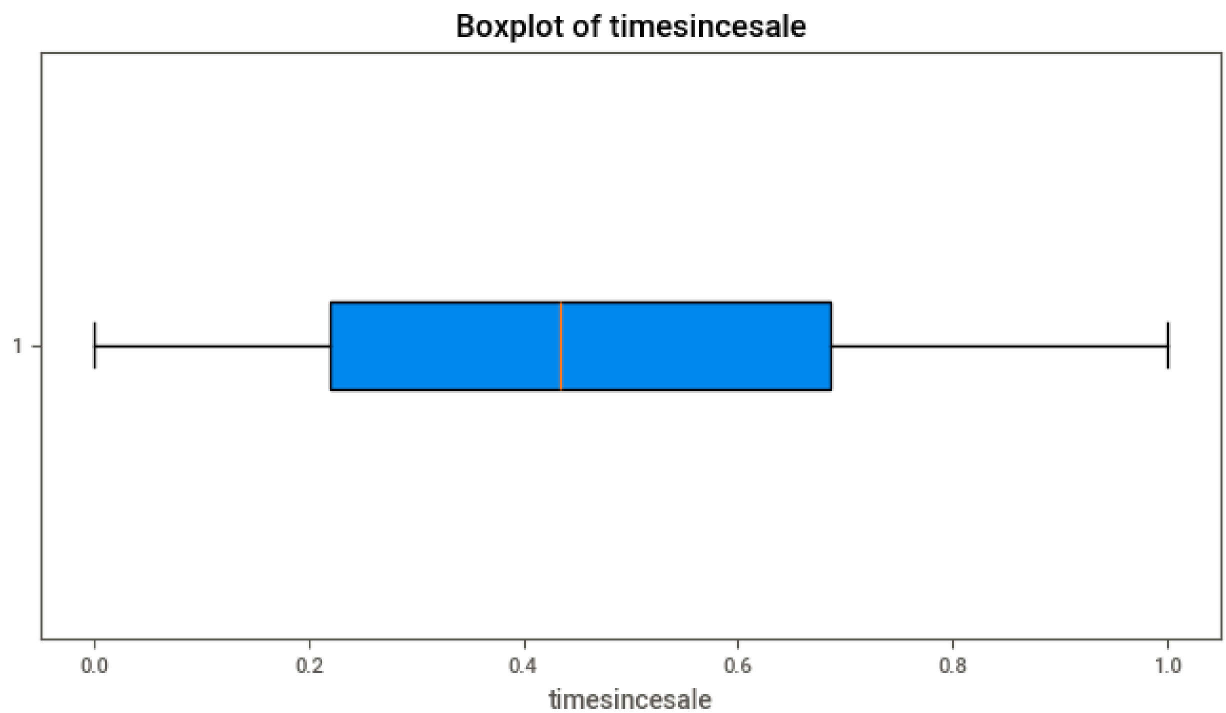


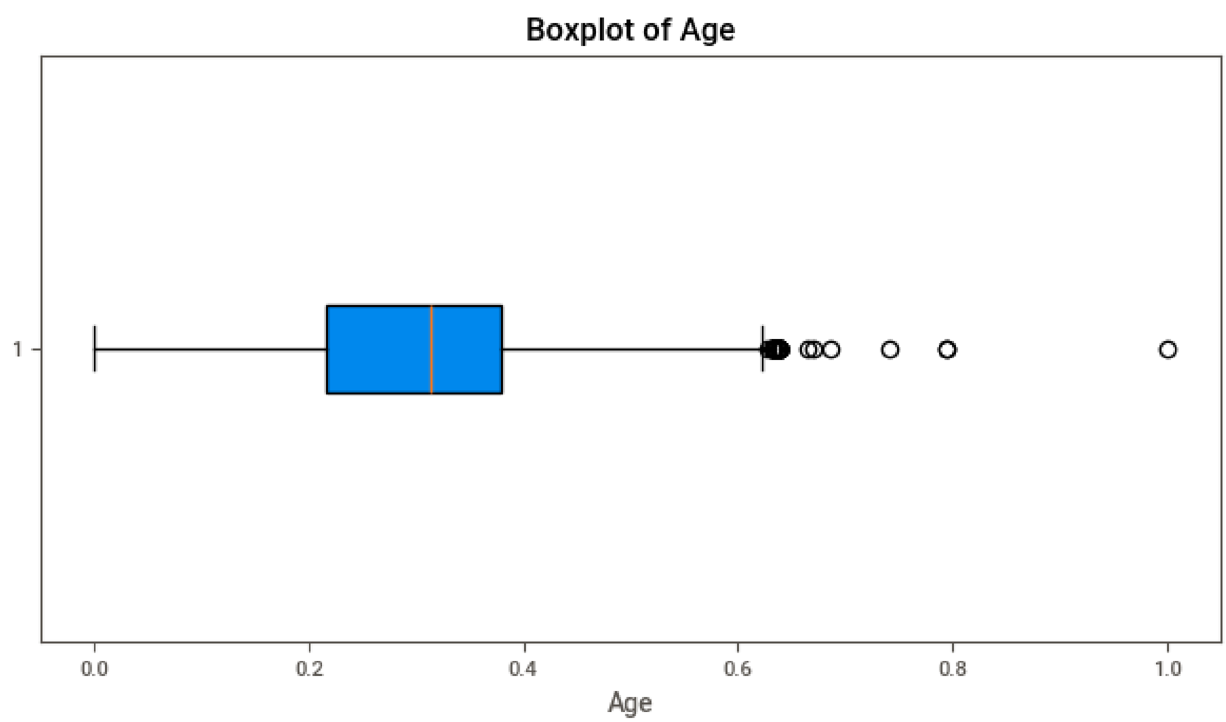

Boxplot of Finished Area
1
0.0
0.2
0.4
0.6
0.8
1.0
Finished Area


Boxplot of Bedrooms
1
0.0
0.2
0.4
0.6
0.8
1.0
Bedrooms


Boxplot of Full Bath
1
0.0
0.2
0.4
0.6
0.8
1.0
Full Bath


Boxplot of Half Bath
1
0.0
0.2
0.4
0.6
0.8
1.0
Half Bath



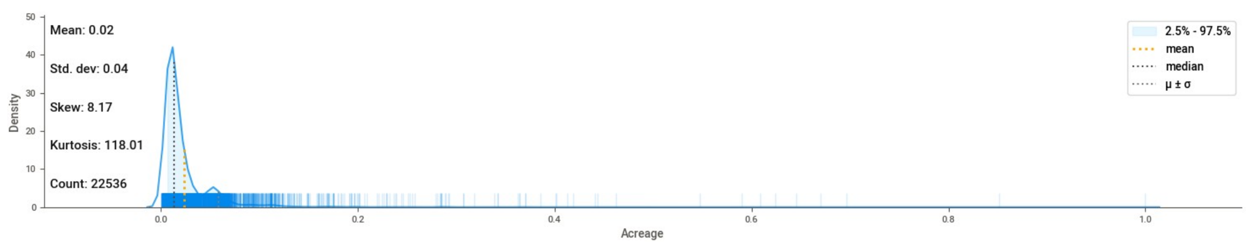


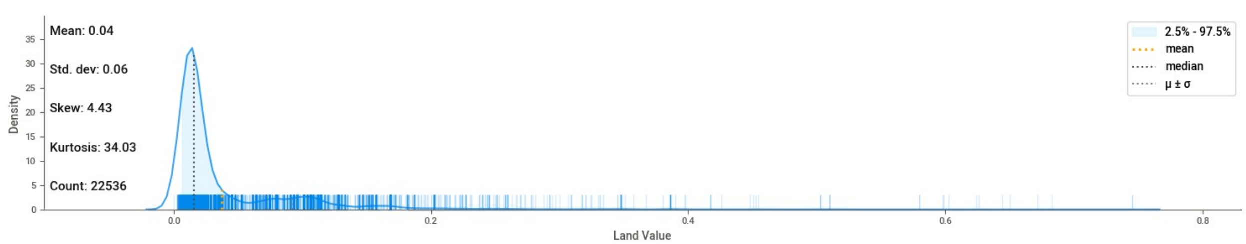

```
{'Acreage': 3036,
 'Land Value': 4175,
 'Building Value': 1927,
 'Finished Area': 1380,
 'Bedrooms': 332,
```

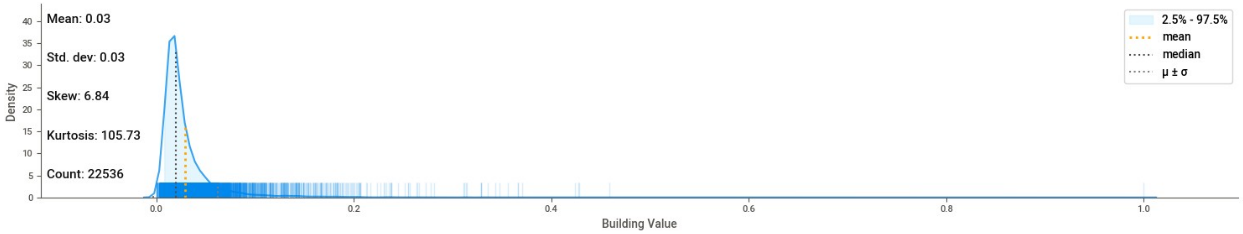
```
 'Full Bath': 1327,
 'Half Bath': 23,
 'timesincesale': 0,
 'Age': 168}
```

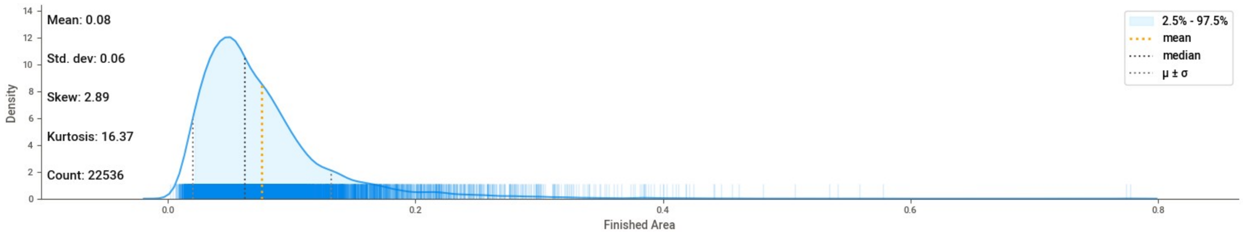
```python
import klib


# Use klib.dist_plot for each numerical feature
for column in numerical_features:
    klib.dist_plot(df1[column])
```

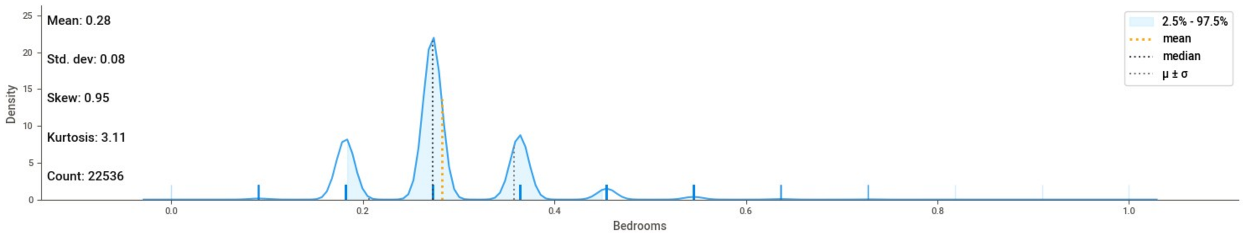
```
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
Large dataset detected, using 10000 random samples for the plots.
Summary statistics are still based on the entire dataset.
```

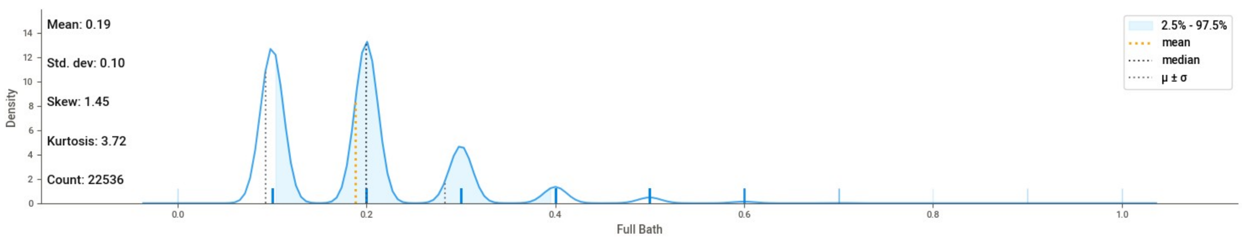


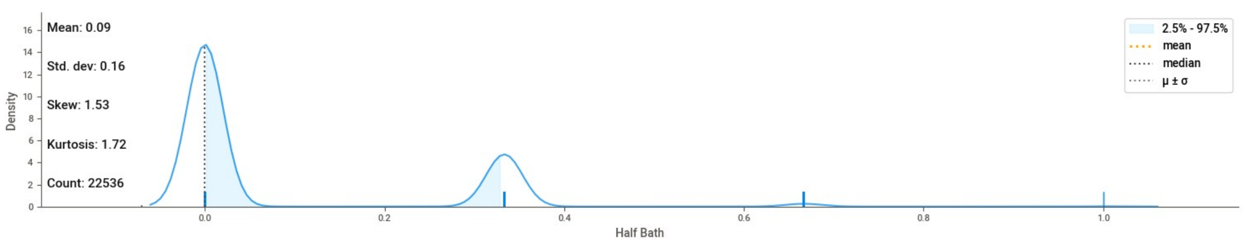


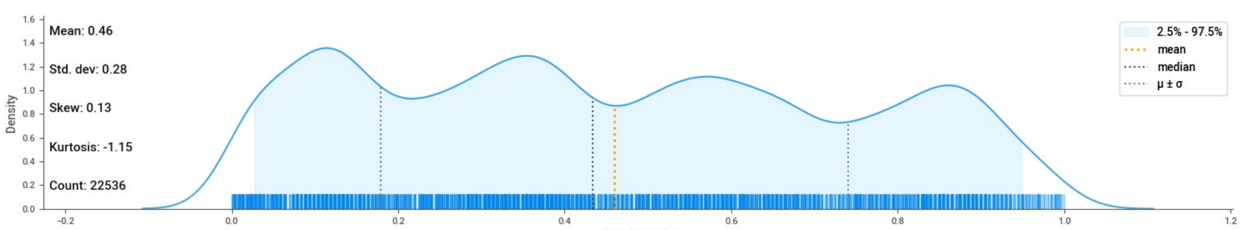

Mean: 0.03
Std. dev: 0.03
Skew: 6.84
Kurtosis: 105.73
Count: 22536
Density
Building Value
2.5% - 97.5%
mean
median
μ ± σ

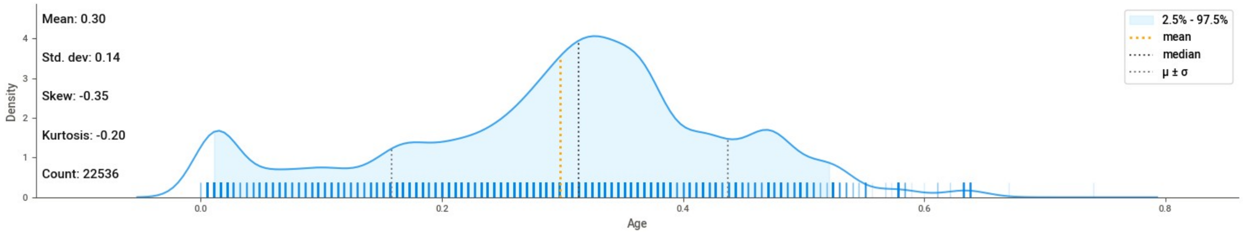

Mean: 0.08
Std. dev: 0.06
Skew: 2.89
Kurtosis: 16.37
Count: 22536
Density
Finished Area
2.5% - 97.5%
mean
median
μ ± σ


Mean: 0.28
Std. dev: 0.08
Skew: 0.95
Kurtosis: 3.11
Count: 22536
Density
Bedrooms
2.5% - 97.5%
mean
median
μ ± σ


Mean: 0.19
Std. dev: 0.10
Skew: 1.45
Kurtosis: 3.72
Count: 22536
Density
Full Bath
2.5% - 97.5%
mean
median
μ ± σ

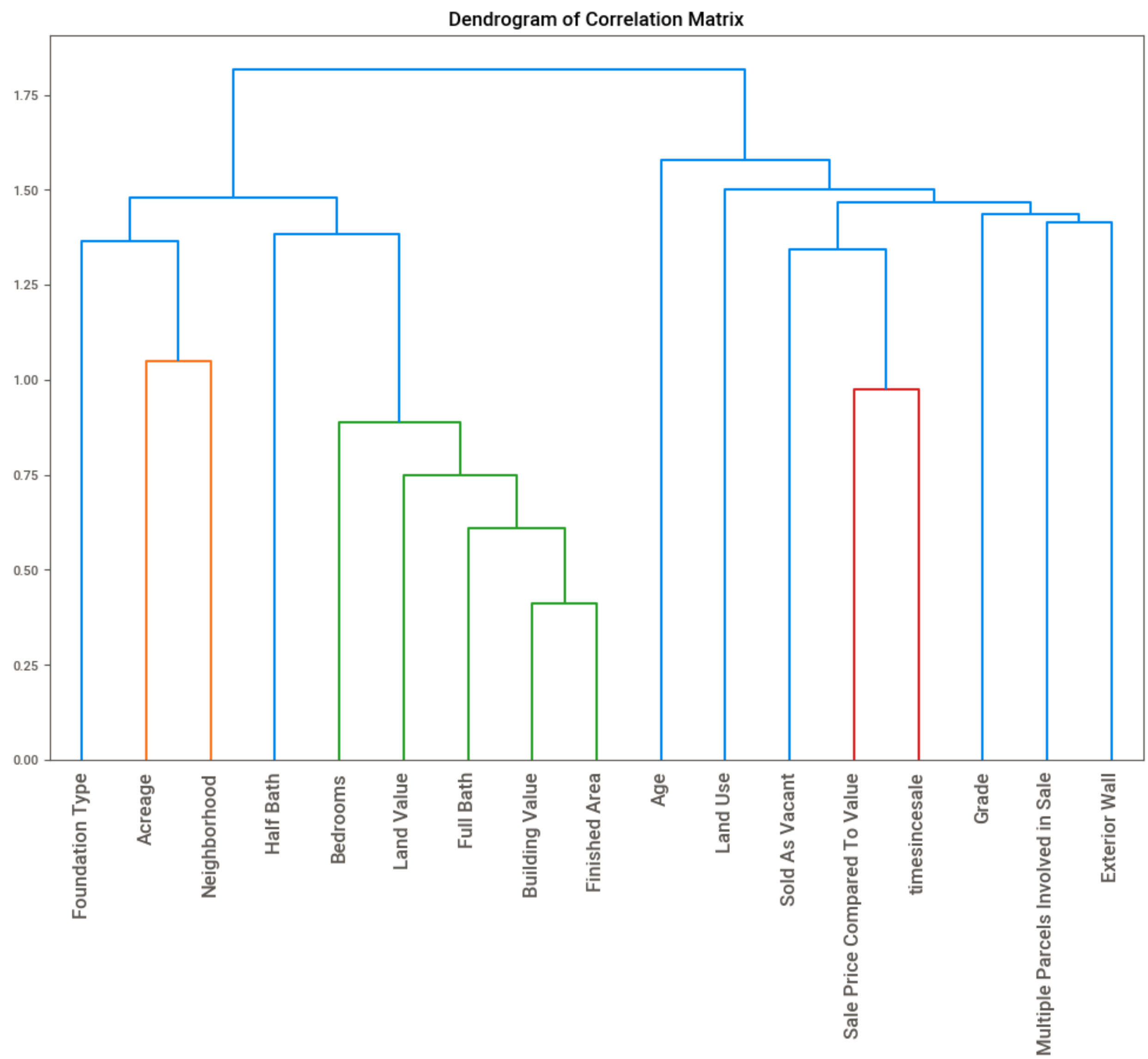

Mean: 0.09
Std. dev: 0.16
Skew: 1.53
Kurtosis: 1.72
Count: 22536
Density
Half Bath
2.5% - 97.5%
mean
median
μ ± σ

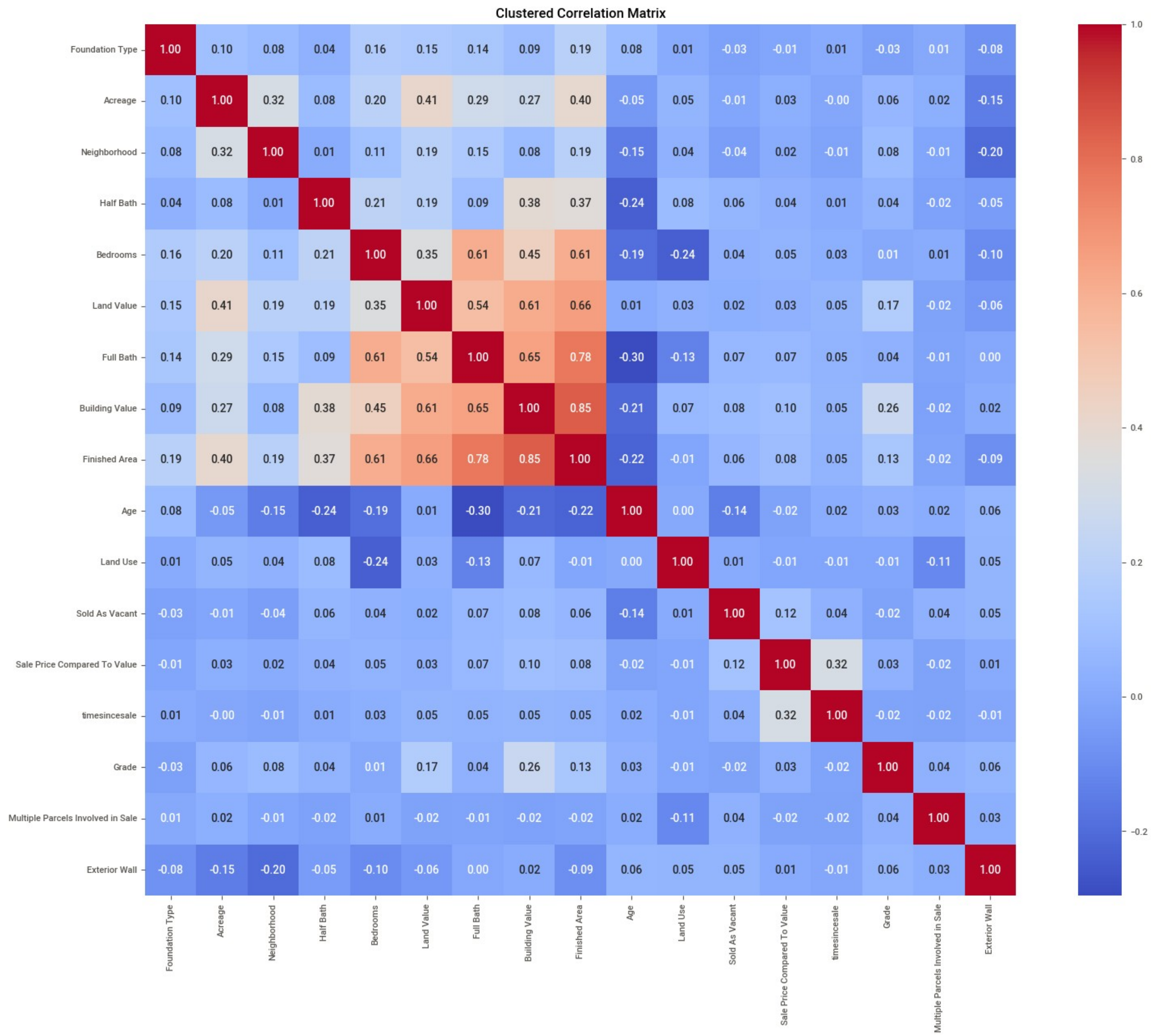

Mean: 0.46
Std. dev: 0.28
Skew: 0.13
Kurtosis: -1.15
Count: 22536
Density
timesincesale
2.5% - 97.5%
mean
median
μ ± σ



Using Hierarchical Clustering to create clusters and then check the Pearson's correlation coefficient

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy.cluster.hierarchy import linkage, dendrogram

# Compute the correlation matrix for all features in df1
correlation_matrix = df1.corr()

# Compute linkage for clustering
linkage_matrix = linkage(correlation_matrix, method='average')

# Plot dendrogram
plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, labels=correlation_matrix.columns,
leaf_rotation=90)
plt.title("Dendrogram of Correlation Matrix")
plt.show()

# Reorder correlation matrix based on clustering
ordered_indices = dendrogram(linkage_matrix, no_plot=True)['leaves']
reordered_corr = correlation_matrix.iloc[ordered_indices,
ordered_indices]

# Plot the reordered heatmap
plt.figure(figsize=(20, 14))
sns.heatmap(reordered_corr, annot=True, fmt=".2f", cmap="coolwarm",
cbar=True, square=True)
plt.title("Clustered Correlation Matrix")
plt.show()
```


Dendrogram of Correlation Matrix
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Foundation Type
Acreage
Neighborhood
Half Bath
Bedrooms
Land Value
Full Bath
Building Value
Finished Area
Age
Land Use
Sold As Vacant
Sale Price Compared To Value
timesincesale
Grade
Multiple Parcels Involved in Sale
Exterior Wall



Assigning the target variable based on the assignment rubric and giving it the simple and intuitive nomenclature 'target'

```
# Assign 'Sale Price Compared To Value' as the target variable
df1['target'] = df1['Sale Price Compared To Value']



# Display the first few rows to confirm the target column creation
df1.head()
```

```
   Land Use  Sold As Vacant  Multiple Parcels Involved in Sale  
Acreage  \
0       1.0             0.0                                0.0  
0.007446   
1       1.0             0.0                                0.0  
0.004009   
2       1.0             0.0                                0.0  
0.007446   
3       1.0             0.0                                0.0  
0.017182   
4       1.0             0.0                                0.0  
0.007446   

   Neighborhood  Land Value  Building Value  Finished Area  Foundation 
Type  \
0      0.320492    0.016648        0.022841       0.036258             
0.6   
1      0.957126    0.017719        0.026859       0.085113             
0.8   
2      0.320811    0.012901        0.041612       0.087954             
0.2   
3      0.320811    0.012901        0.023476       0.078793             
0.0   
4      0.320811    0.012901        0.014546       0.030449             
0.0   

   Exterior Wall     Grade  Bedrooms  Full Bath  Half Bath  \
0          0.000  0.285714  0.181818        0.1   0.000000   
1          0.125  0.285714  0.272727        0.2   0.333333   
2          0.125  0.142857  0.363636        0.2   0.000000   
3          0.375  0.285714  0.181818        0.1   0.000000   
4          0.375  0.285714  0.181818        0.1   0.000000   

   Sale Price Compared To Value  timesincesale       Age  target  
0                           0.0       0.993562  0.410811     0.0  
1                           0.0       0.988555  0.091892     0.0  
2                           1.0       0.988555  0.372973     1.0  
3                           1.0       0.984979  0.578378     1.0  
4                           1.0       0.998569  0.389189     1.0  

# Drop the 'Sale Price Compared To Value' column
df1 = df1.drop(columns=['Sale Price Compared To Value'])

df1.head()

   Land Use  Sold As Vacant  Multiple Parcels Involved in Sale  
Acreage  \
0       1.0             0.0                                0.0  
0.007446   
1       1.0             0.0                                0.0  
```

```
0.004009
2       1.0             0.0                                0.0
0.007446
3       1.0             0.0                                0.0
0.017182
4       1.0             0.0                                0.0
0.007446

   Neighborhood  Land Value  Building Value  Finished Area  Foundation
Type  \
0      0.320492    0.016648        0.022841       0.036258
0.6
1      0.957126    0.017719        0.026859       0.085113
0.8
2      0.320811    0.012901        0.041612       0.087954
0.2
3      0.320811    0.012901        0.023476       0.078793
0.0
4      0.320811    0.012901        0.014546       0.030449
0.0

   Exterior Wall     Grade  Bedrooms  Full Bath  Half Bath
timesincesale  \
0          0.000  0.285714  0.181818        0.1   0.000000
0.993562
1          0.125  0.285714  0.272727        0.2   0.333333
0.988555
2          0.125  0.142857  0.363636        0.2   0.000000
0.988555
3          0.375  0.285714  0.181818        0.1   0.000000
0.984979
4          0.375  0.285714  0.181818        0.1   0.000000
0.998569

        Age  target
0  0.410811     0.0
1  0.091892     0.0
2  0.372973     1.0
3  0.578378     1.0
4  0.389189     1.0
```

```
df.shape
```

```
(22651, 26)
```

Data Splicing and Remedial measures for the imbalanced data set using SMOTE

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Define the features (X) and target (y)
X = df1.drop(columns=['target'])
y = df1['target']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Apply SMOTE to the training set to handle class imbalance
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Display the shape of the training sets after applying SMOTE
X_train_smote.shape, y_train_smote.shape
```

```
((27166, 16), (27166,))
```

```
X_train = X_train_smote
y_train = y_train_smote

X_train.shape, y_train.shape
```

```
((27166, 16), (27166,))
```

Logistic Regression Model

```
import time


import statsmodels.api as sm
import pandas as pd

# Add a constant to the predictors (for intercept)
X_train_sm = sm.add_constant(X_train)

# Measure training time
start_time = time.time()

# Fit the logistic regression model
logit_model = sm.Logit(y_train, X_train_sm)
```

```
logit_result = logit_model.fit()

# Calculate training time
training_time = time.time() - start_time

# Print the summary and training time
print(logit_result.summary())
print(f"Training Time: {training_time:.4f} seconds")
```

```
Optimization terminated successfully.
         Current function value: 0.608621
         Iterations 7
                           Logit Regression Results

======================================================================
========
Dep. Variable:                 target   No. Observations:
27166
Model:                          Logit   Df Residuals:
27149
Method:                           MLE   Df Model:
16
Date:                Fri, 06 Dec 2024   Pseudo R-squ.:
0.1219
Time:                        22:41:47   Log-Likelihood:
-16534.
converged:                       True   LL-Null:
-18830.
Covariance Type:            nonrobust   LLR p-value:
0.000
======================================================================
===============================
                                        coef    std err          z
P>|z|      [0.025      0.975]
----------------------------------------------------------------------
-------------------------------
const                                -1.8373      0.119    -15.445
0.000      -2.071      -1.604
Land Use                             -0.1626      0.064     -2.534
0.011      -0.288      -0.037
Sold As Vacant                        2.9829      0.289     10.338
0.000       2.417       3.548
Multiple Parcels Involved in Sale    -0.4927      0.103     -4.775
0.000      -0.695      -0.290
Acreage                               2.8655      0.531      5.401
0.000       1.826       3.905
Neighborhood                          0.4492      0.065      6.942
0.000       0.322       0.576
Land Value                           -4.4886      0.363    -12.377
0.000      -5.199      -3.778
```

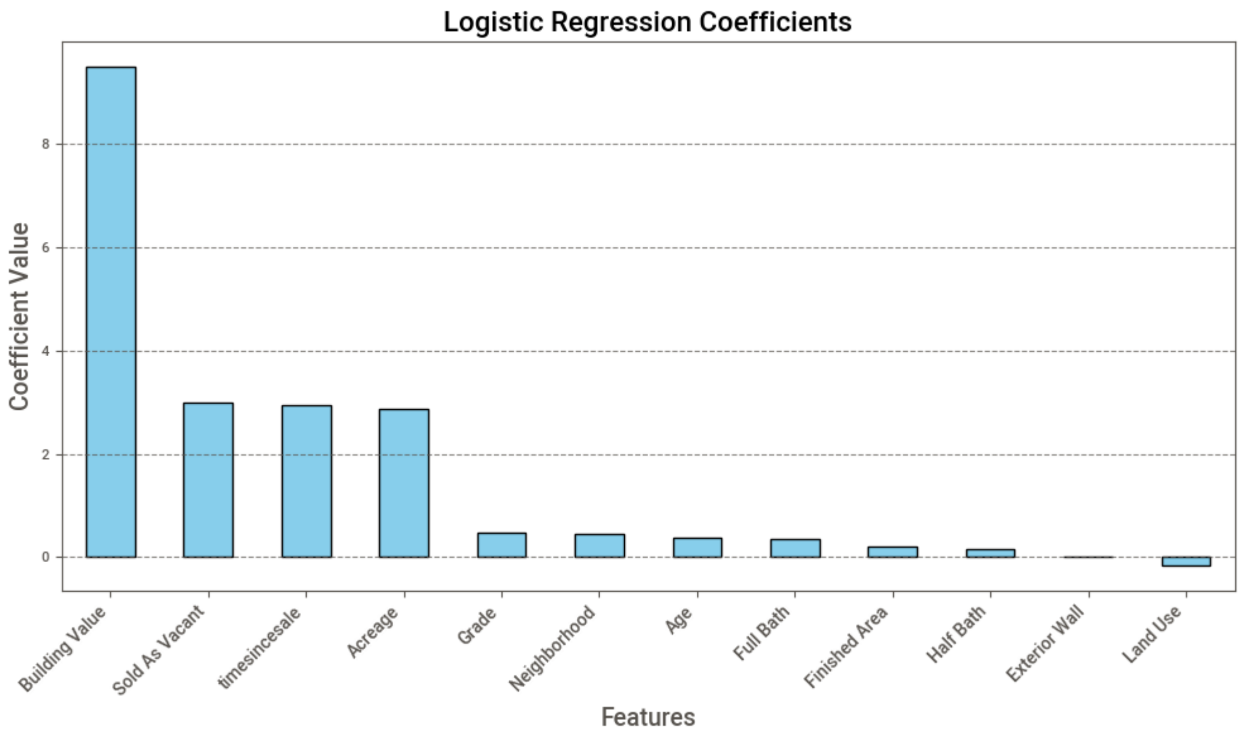
```
Building Value                                 9.4861      1.030      9.206
0.000       7.467      11.506
Finished Area                                  0.2168      0.685      0.316
0.752      -1.126       1.560
Foundation Type                               -0.1900      0.053     -3.568
0.000      -0.294      -0.086
Exterior Wall                                  0.0025      0.068      0.037
0.970      -0.130       0.135
Grade                                          0.4632      0.104      4.439
0.000       0.259       0.668
Bedrooms                                      -0.8585      0.256     -3.358
0.001      -1.360      -0.357
Full Bath                                      0.3476      0.280      1.242
0.214      -0.201       0.896
Half Bath                                      0.1634      0.102      1.599
0.110      -0.037       0.364
timesincesale                                  2.9297      0.051     57.660
0.000       2.830       3.029
Age                                            0.3876      0.110      3.516
0.000       0.172       0.604
======================================================================
===============================
Training Time: 0.0444 seconds
```

```
logit_result.params.sort_values(ascending = False)
```

```
Building Value                          9.486093
Sold As Vacant                          2.982857
timesincesale                           2.929700
Acreage                                 2.865522
Grade                                   0.463191
Neighborhood                            0.449218
Age                                     0.387581
Full Bath                               0.347563
Finished Area                           0.216812
Half Bath                               0.163409
Exterior Wall                           0.002513
Land Use                               -0.162634
Foundation Type                        -0.190045
Multiple Parcels Involved in Sale   -0.492655
Bedrooms                               -0.858504
const                                  -1.837344
Land Value                             -4.488609
dtype: float64
```

```
import matplotlib.pyplot as plt

# Assuming logit_result.params exists, create sorted coefficients
sorted_coefficients = logit_result.params.sort_values(ascending=False)
```

```python
# Plot the coefficients as a colorful bar plot
plt.figure(figsize=(10, 6))
sorted_coefficients[0:12].plot(kind='bar', color='skyblue',
edgecolor='black')
plt.title('Logistic Regression Coefficients', fontsize=16)
plt.xlabel('Features', fontsize=14)
plt.ylabel('Coefficient Value', fontsize=14)
plt.xticks(rotation=45, fontsize=10, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Predictions

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import time
import pandas as pd
from sklearn.model_selection import train_test_split

# Assuming df1 is available
# Define predictors (X) and target (y)
X = df1.drop(columns=['target'])
y = df1['target']
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Initialize and train the logistic regression model
logistic = LogisticRegression()

# Measure training time
start_train = time.time()
logistic.fit(X_train, y_train)
elapsed_train = time.time() - start_train

# Measure prediction time
start_pred = time.time()
y_pred = logistic.predict(X_test)
elapsed_pred = time.time() - start_pred

# Print elapsed times
print(f"Training Time: {elapsed_train:.4f} seconds")
print(f"Prediction Time: {elapsed_pred:.4f} seconds")
```

```
Training Time: 0.0379 seconds
Prediction Time: 0.0020 seconds
```

Metrics for the Preliminary Logistic regression Model

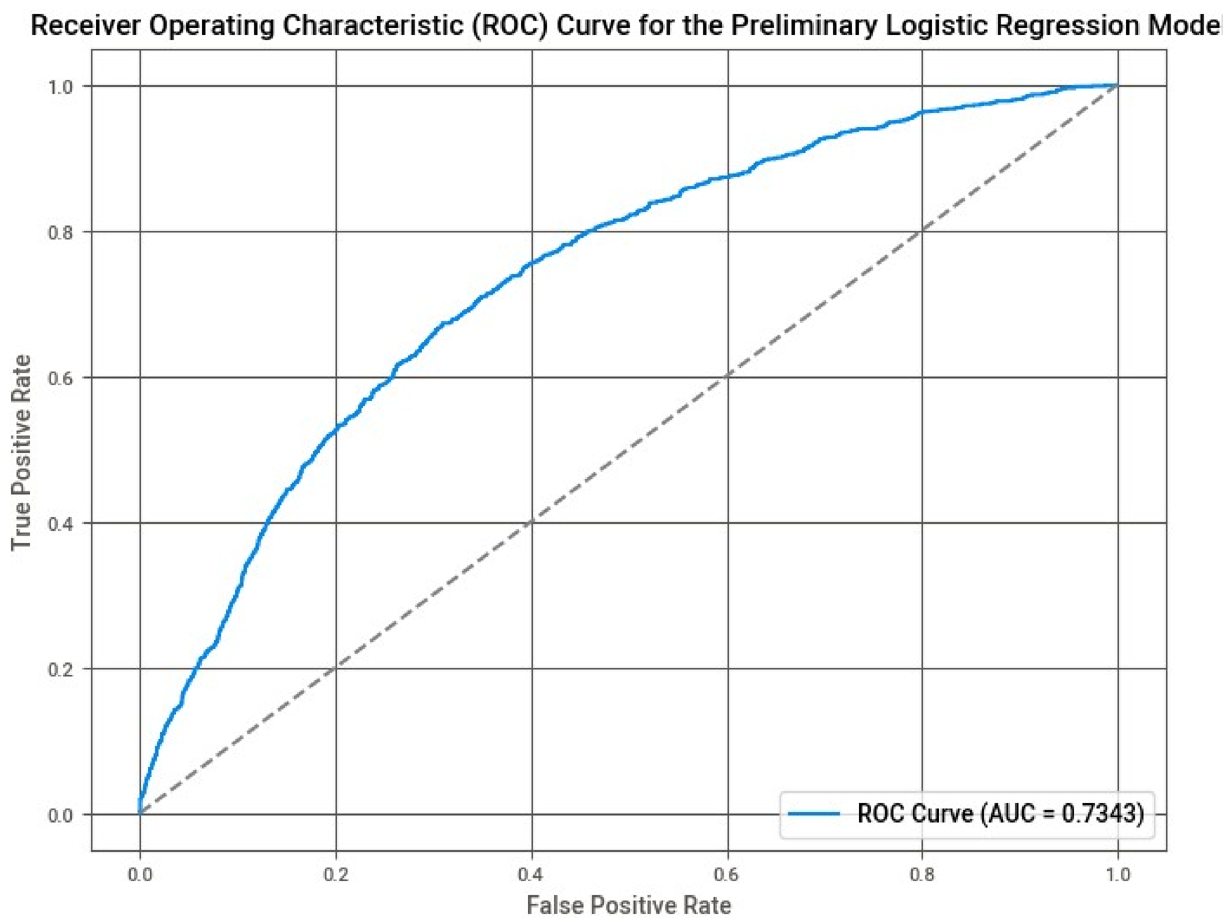
```
score=accuracy_score(y_pred,y_test)
print(f" The Logisitc Regression Model Metrics: \n The accuracy is :
{score}")
print(f"The Classification Report is : \n
{classification_report(y_pred,y_test)}")
print(f"The Confusion Matrix is : \n
{confusion_matrix(y_pred,y_test)}")
```

```
 The Logisitc Regression Model Metrics:
 The accuracy is : 0.7622005323868678
The Classification Report is :
               precision    recall  f1-score   support

         0.0       0.97      0.77      0.86      4268
         1.0       0.13      0.58      0.21       240

    accuracy                           0.76      4508
   macro avg       0.55      0.68      0.53      4508
```

```
weighted avg       0.93      0.76      0.83      4508

The Confusion Matrix is :
 [[3296  972]
 [ 100  140]]
```

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predict probabilities for the positive class
y_pred_proba = logistic.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC
auc_score = roc_auc_score(y_test, y_pred_proba)
print(f"AUC Score: {auc_score:.4f}")

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal
line
plt.title("Receiver Operating Characteristic (ROC) Curve for the
Preliminary Logistic Regression Model")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

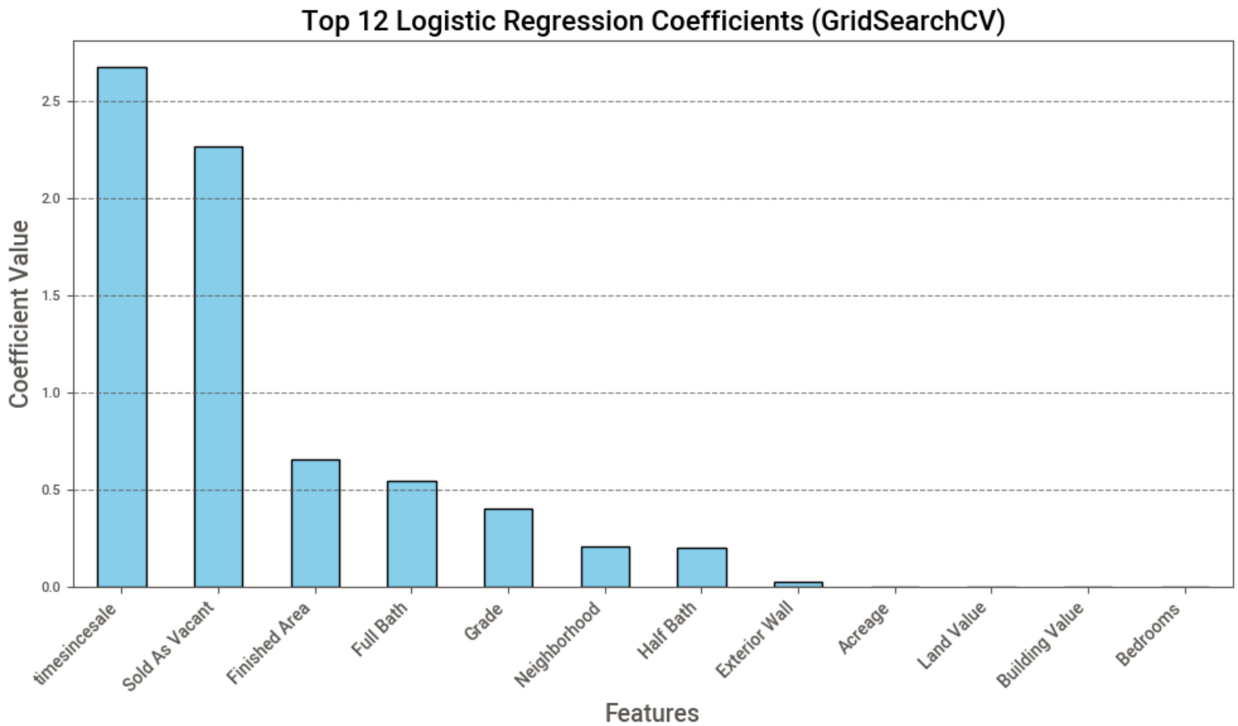
```
AUC Score: 0.7343
```



Hyperparameter Tuning and rectifying the Target variable imbalance by assigning weights

```
## Hyperparamter tuning
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
penalty=['l1', 'l2', 'elasticnet']
c_values=[100,10,1.0,0.1,0.01]
solver=['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
class_weight=[{0:w,1:y} for w in [1,10,50,100] for y in [1,10,50,100]]

params=dict(penalty=penalty,C=c_values,solver=solver,class_weight=clas
s_weight)

import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")

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

```
cv=StratifiedKFold()
grid=GridSearchCV(estimator=model,param_grid=params,scoring='accuracy'
,cv=cv)

# Measure training time
start_train = time.time()

grid.fit(X_train,y_train)

elapsed_train = time.time() - start_train


# Print elapsed times
print(f"Training Time: {elapsed_train:.4f} seconds")

Training Time: 290.4097 seconds

grid.best_params_

{'C': 0.1,
 'class_weight': {0: 1, 1: 1},
 'penalty': 'l1',
 'solver': 'liblinear'}

best_model = grid.best_estimator_

# Assuming coefficients are from a grid search fitted logistic
regression model
grid_coefficients = pd.Series(best_model.coef_.flatten(),
index=X_train.columns).sort_values(ascending=False)

# Plot the top 12 coefficients as a colorful bar plot
plt.figure(figsize=(10, 6))
grid_coefficients[:12].plot(kind='bar', color='skyblue',
edgecolor='black')
plt.title('Top 12 Logistic Regression Coefficients (GridSearchCV)',
fontsize=16)
plt.xlabel('Features', fontsize=14)
plt.ylabel('Coefficient Value', fontsize=14)
plt.xticks(rotation=45, fontsize=10, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```




```
# Measure prediction time
start_pred = time.time()
y_pred1=grid.predict(X_test)
elapsed_pred = time.time() - start_pred


print(f"Prediction Time: {elapsed_pred:.4f} seconds")
```

```
Prediction Time: 0.0030 seconds
```

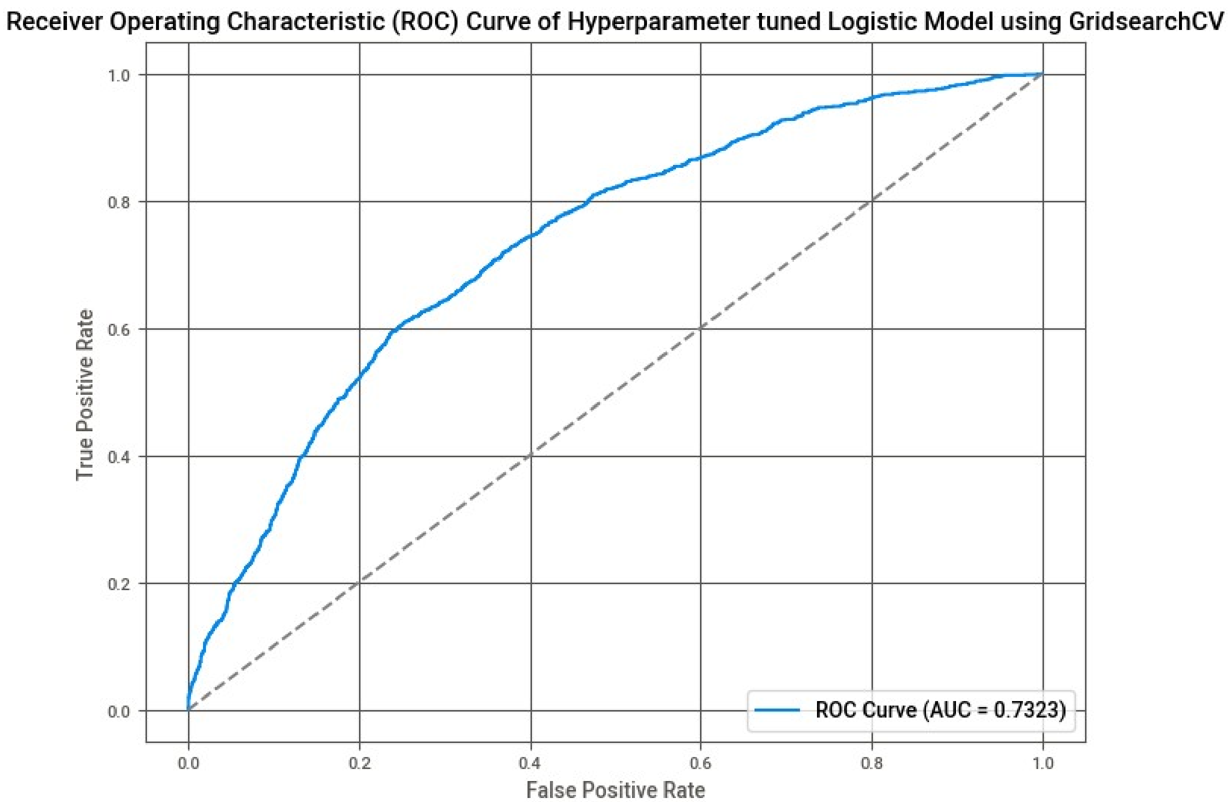
```
score=accuracy_score(y_pred1,y_test)
print(f" The Hyperparameter Tuned Logisitc Regression Model Metrics: \
n The accuracy is : {score}")
print(f"The Classification Report is : \n
{classification_report(y_pred1,y_test)}")
print(f"The Confusion Matrix is : \n
{confusion_matrix(y_pred1,y_test)}")
```

```
 The Hyperparameter Tuned Logisitc Regression Model Metrics:
 The accuracy is : 0.7639751552795031
The Classification Report is :
              precision    recall  f1-score   support

         0.0       0.98      0.77      0.86      4326
         1.0       0.10      0.63      0.18       182

    accuracy                           0.76      4508
   macro avg       0.54      0.70      0.52      4508
```

```
weighted avg       0.94      0.76      0.83      4508

The Confusion Matrix is :
 [[3329  997]
 [  67  115]]
```

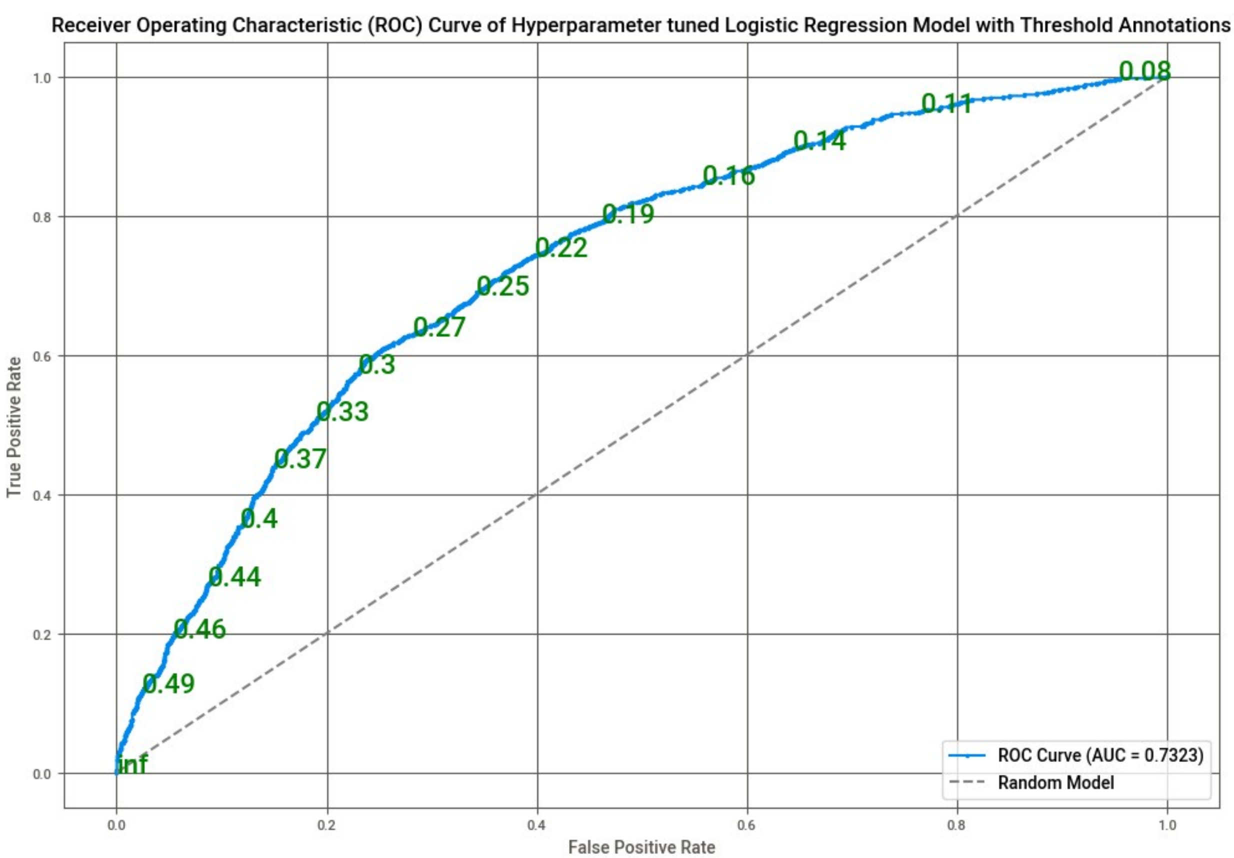
```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predict probabilities for the positive class
y_pred_proba1 = grid.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba1)

# Calculate the AUC
auc_score = roc_auc_score(y_test, y_pred_proba1)
print(f"AUC Score: {auc_score:.4f}")

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal
line
plt.title("Receiver Operating Characteristic (ROC) Curve of
Hyperparameter tuned Logistic Model using GridsearchCV")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

```
AUC Score: 0.7323
```




```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import numpy as np

# Predict probabilities for the positive class
y_pred_proba1 = grid.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba1)

# Calculate the AUC
auc_score = roc_auc_score(y_test, y_pred_proba1)
print(f"AUC Score: {auc_score:.4f}")

# Plot the ROC curve with limited annotations
fig = plt.figure(figsize=(12, 8))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})",
marker='.')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Model')

# Add limited annotations for thresholds (e.g., every 10th point)

for i, xy in enumerate(zip(fpr, tpr, thresholds)):
    if i % 100 == 0:  # Annotate every 10th point
```

```
        plt.annotate(f'{np.round(xy[2], 2)}', xy=(xy[0], xy[1]),
fontsize=16, color='green')

# Axis labels and title
plt.title("Receiver Operating Characteristic (ROC) Curve of
Hyperparameter tuned Logistic Regression Model with Threshold
Annotations")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")

# Show legend
plt.legend(loc="lower right")

# Show grid
plt.grid()

# Show the plot
plt.show()

AUC Score: 0.7323
```



Decision Tree Model Implementation

```
from sklearn.tree import DecisionTreeClassifier, export_text,
export_graphviz
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
import time
import graphviz

# Initialize the decision tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Set up the hyperparameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy'
)


# Measure training time
start_train = time.time()
grid_search.fit(X_train, y_train)
elapsed_train = time.time() - start_train

# Retrieve the best parameters and best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print("Best Parameters from GridSearchCV:")
print(best_params)
print(f"Training Time: {elapsed_train:.2f} seconds")
```

```
Best Parameters from GridSearchCV:
{'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 1,
'min_samples_split': 2}
Training Time: 19.91 seconds
```

```
# Evaluate the best model
start_pred = time.time()
```

```
y_pred = best_model.predict(X_test)
elapsed_pred = time.time() - start_pred

accuracy = accuracy_score(y_test, y_pred)
print(f"\nTuned Model Accuracy: {accuracy:.2f}")
print(f"Prediction Time: {elapsed_pred:.2f} seconds")
```

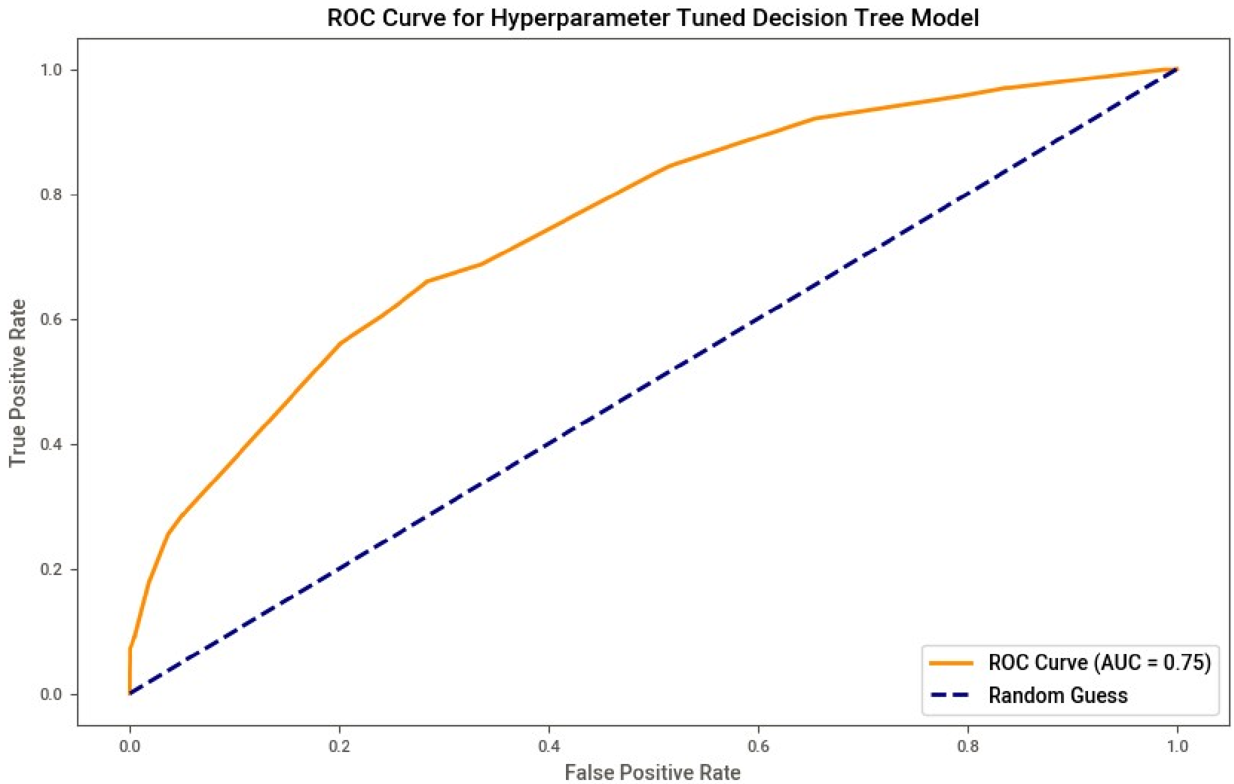
```
Tuned Model Accuracy: 0.79
Prediction Time: 0.00 seconds
```

```
# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.96      0.87      3396
         1.0       0.70      0.26      0.37      1112

    accuracy                           0.79      4508
   macro avg       0.75      0.61      0.62      4508
weighted avg       0.77      0.79      0.75      4508
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming the best_model is the hyperparameter tuned Decision Tree
model
# and y_test and X_test are available
y_prob = best_model.predict_proba(X_test)[:, 1]  # Get the probability
scores for the positive class

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Hyperparameter Tuned Decision Tree Model')
```

```
plt.legend(loc="lower right")
plt.show()
```



As Tree is too large owing to many features I exported image using Graphviz module

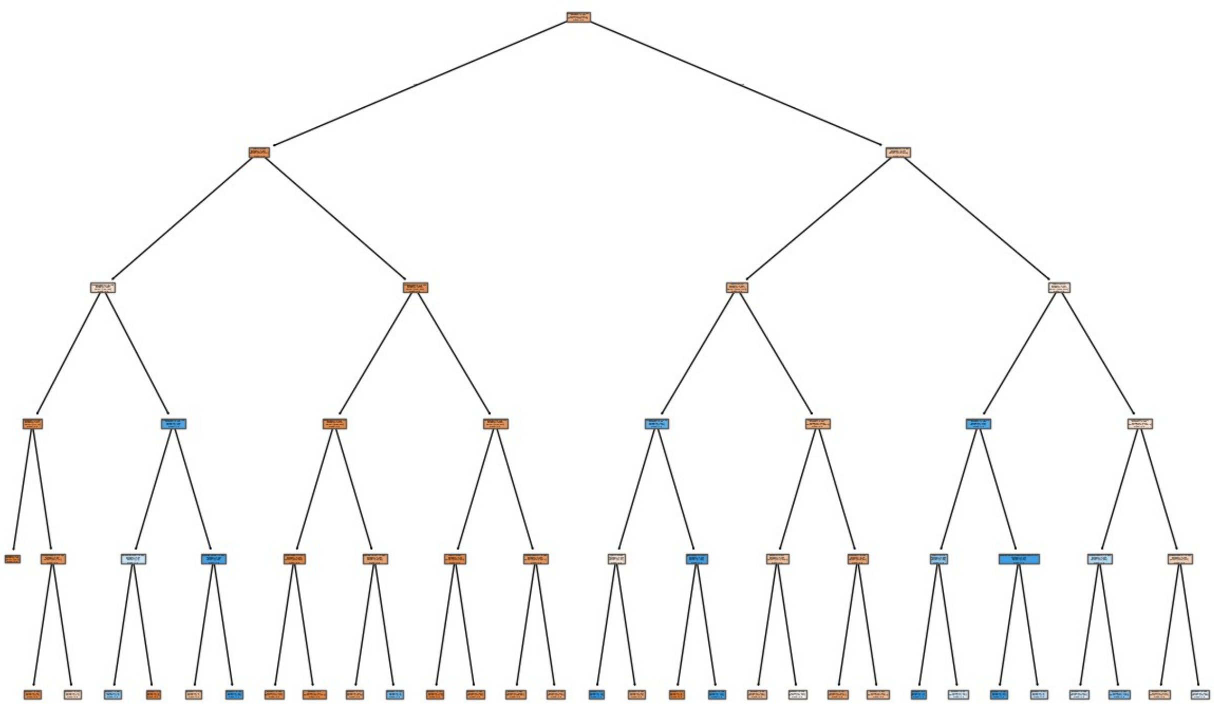
```
# Visualize the tree using Graphviz
dot_data = export_graphviz(
    best_model,
    out_file=None,
    feature_names=X.columns,
    class_names=[str(cls) for cls in best_model.classes_],
    filled=True,
    rounded=True,
    special_characters=True
)
graph = graphviz.Source(dot_data)
graph.view()
```

```
'Source.gv.pdf'
```

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Visualize the decision tree for df1
```

```
plt.figure(figsize=(15, 10))
plot_tree(
    best_model,
    feature_names=list(X_train.columns),  # Ensure correct column names from df1
    class_names=[str(cls) for cls in best_model.classes_],  # Class labels from the model
    filled=True
)
plt.title("Tuned Decision Tree Visualization")
plt.show()
```

Tuned Decision Tree Visualization



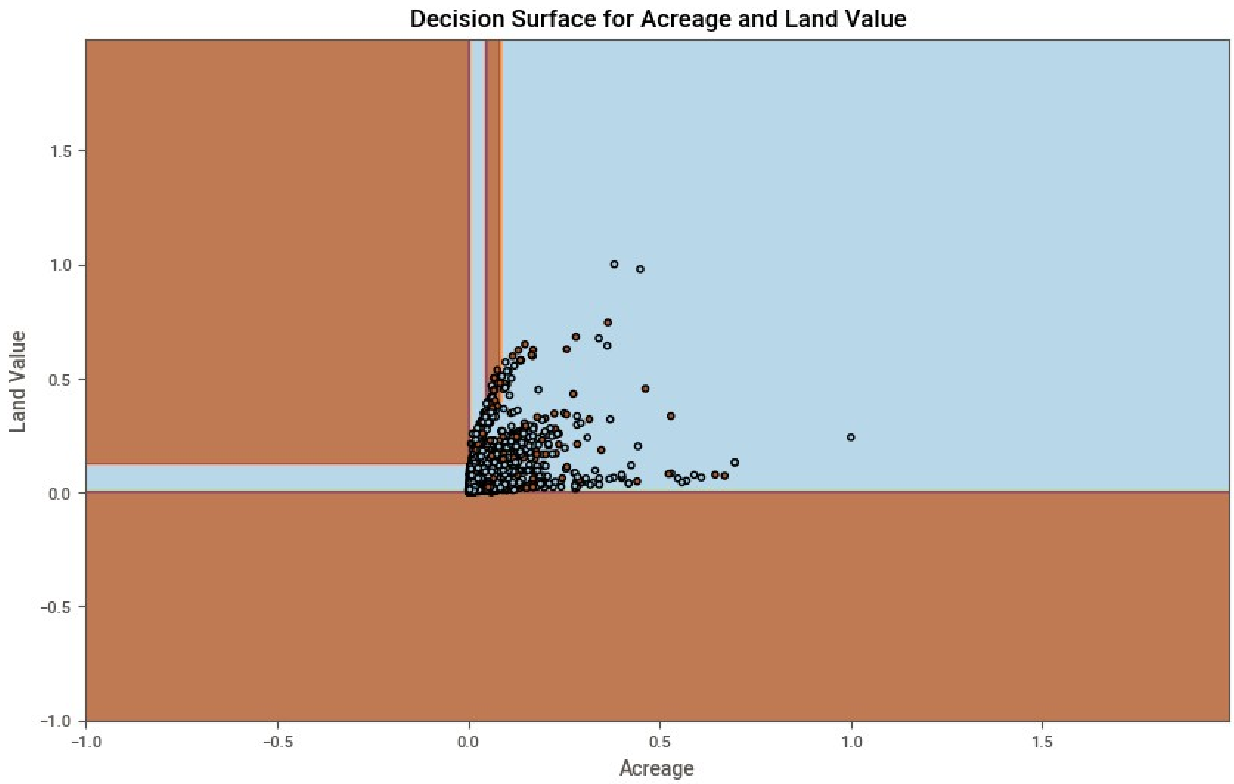
```
# Text representation of the tree
tree_text = export_text(best_model, feature_names=list(X.columns))
print("\nText Representation of the Decision Tree:")
print(tree_text)


Text Representation of the Decision Tree:
|--- timesincesale <= 0.40
|   |--- Age <= 0.01
|   |   |--- timesincesale <= 0.17
|   |   |   |--- Acreage <= 0.01
```

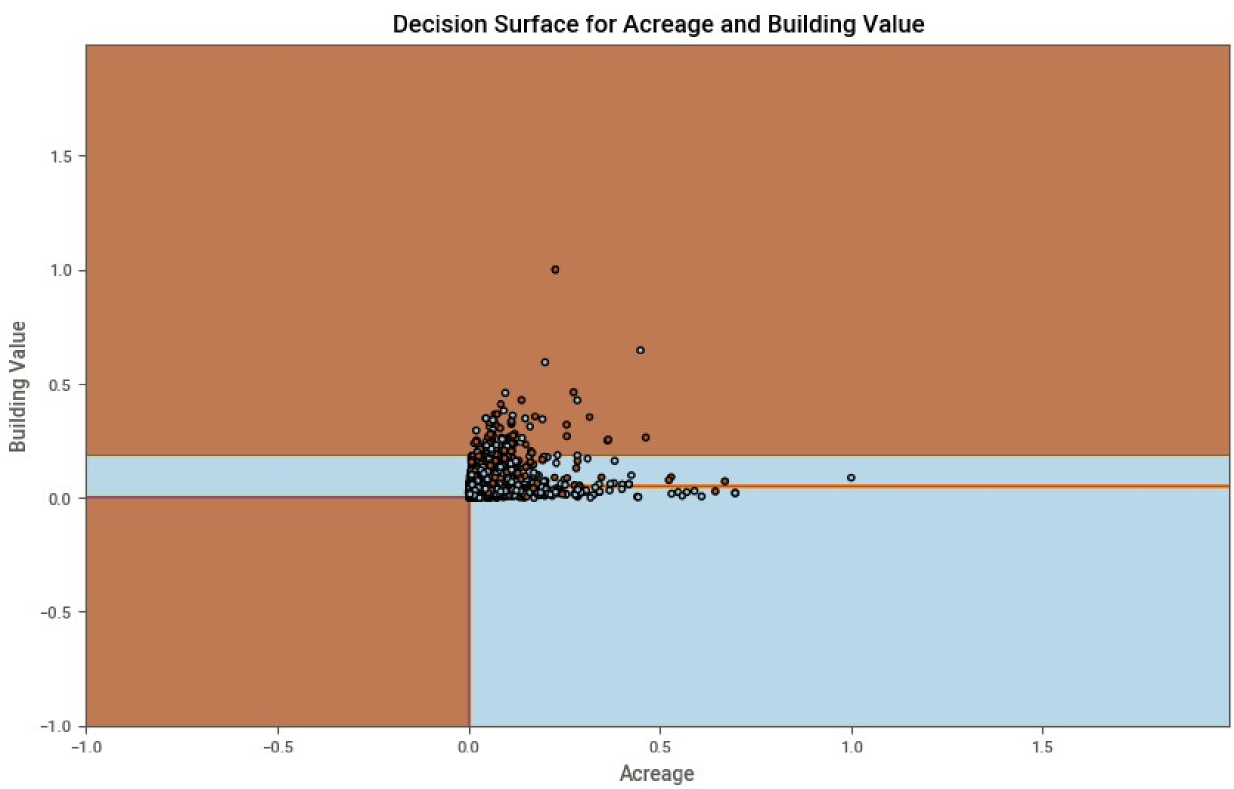
```
|   |   |   |   |--- class: 0.0
|   |   |   |--- Acreage >  0.01
|   |   |   |   |--- timesincesale <= 0.13
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- timesincesale >  0.13
|   |   |   |   |   |--- class: 0.0
|   |   |--- timesincesale >  0.17
|   |   |   |--- timesincesale <= 0.19
|   |   |   |   |--- Neighborhood <= 0.61
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- Neighborhood >  0.61
|   |   |   |   |   |--- class: 0.0
|   |   |   |--- timesincesale >  0.19
|   |   |   |   |--- Building Value <= 0.03
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Building Value >  0.03
|   |   |   |   |   |--- class: 1.0
|   |--- Age >  0.01
|   |   |--- timesincesale <= 0.23
|   |   |   |--- Finished Area <= 0.16
|   |   |   |   |--- Land Value <= 0.01
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Land Value >  0.01
|   |   |   |   |   |--- class: 0.0
|   |   |   |--- Finished Area >  0.16
|   |   |   |   |--- Building Value <= 0.26
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Building Value >  0.26
|   |   |   |   |   |--- class: 1.0
|   |   |--- timesincesale >  0.23
|   |   |   |--- Neighborhood <= 0.27
|   |   |   |   |--- Age <= 0.06
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Age >  0.06
|   |   |   |   |   |--- class: 0.0
|   |   |   |--- Neighborhood >  0.27
|   |   |   |   |--- timesincesale <= 0.35
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- timesincesale >  0.35
|   |   |   |   |   |--- class: 0.0
|--- timesincesale >  0.40
|   |--- timesincesale <= 0.68
|   |   |--- Age <= 0.01
|   |   |   |--- timesincesale <= 0.45
|   |   |   |   |--- Age <= 0.01
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- Age >  0.01
|   |   |   |   |   |--- class: 0.0
|   |   |   |--- timesincesale >  0.45
```

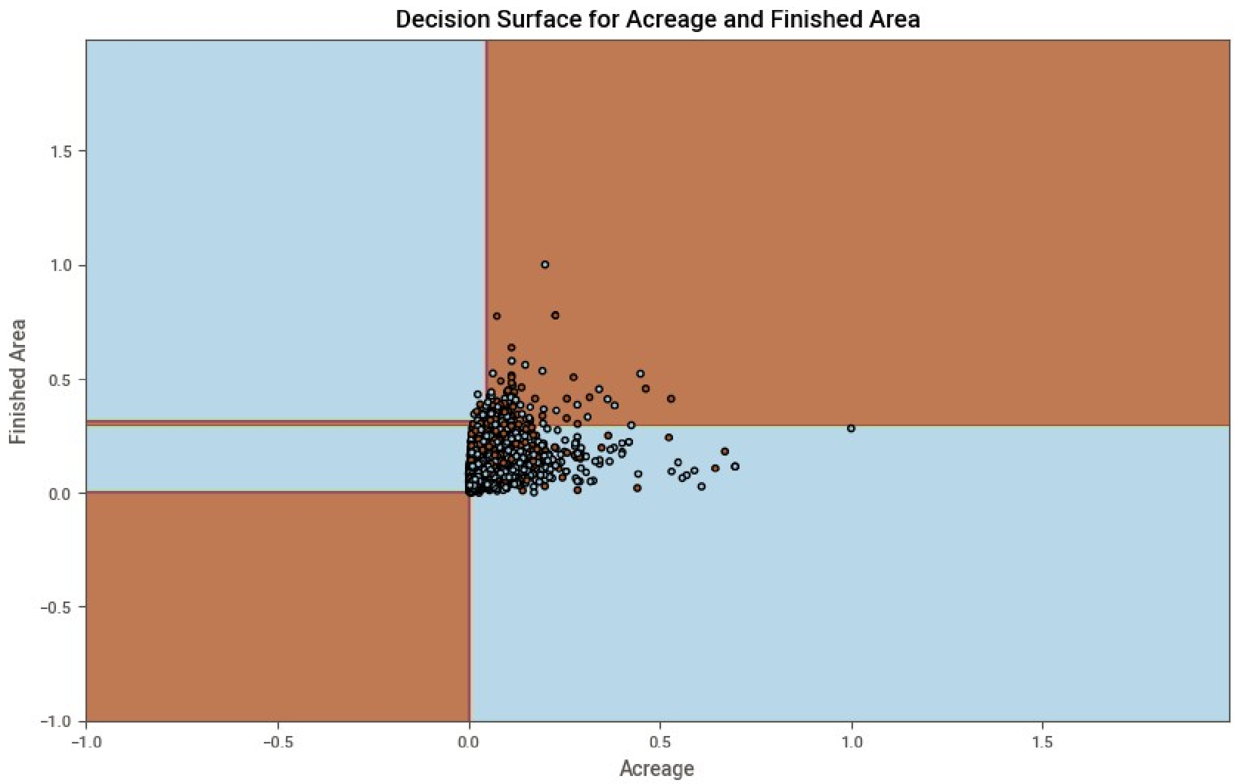
```
|   |   |   |   |--- Bedrooms <= 0.23
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Bedrooms >  0.23
|   |   |   |   |   |--- class: 1.0
|   |   |--- Age >  0.01
|   |   |   |--- Land Value <= 0.01
|   |   |   |   |--- Grade <= 0.36
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Grade >  0.36
|   |   |   |   |   |--- class: 0.0
|   |   |   |--- Land Value >  0.01
|   |   |   |   |--- Acreage <= 0.03
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Acreage >  0.03
|   |   |   |   |   |--- class: 0.0
|   |--- timesincesale >  0.68
|   |   |--- Age <= 0.02
|   |   |   |--- timesincesale <= 0.72
|   |   |   |   |--- Age <= 0.01
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- Age >  0.01
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- timesincesale >  0.72
|   |   |   |   |--- Multiple Parcels Involved in Sale <= 0.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- Multiple Parcels Involved in Sale >  0.50
|   |   |   |   |   |--- class: 1.0
|   |   |--- Age >  0.02
|   |   |   |--- Land Value <= 0.01
|   |   |   |   |--- timesincesale <= 0.90
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- timesincesale >  0.90
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- Land Value >  0.01
|   |   |   |   |--- timesincesale <= 0.93
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- timesincesale >  0.93
|   |   |   |   |   |--- class: 1.0
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from itertools import combinations

# Assuming `numerical_features`, `X_train`, and `y_train` are defined
numerical_features = ['Acreage', 'Land Value', 'Building Value',
                      'Finished Area', 'Age', 'Bedrooms', 'Full Bath',
'Half Bath']
```

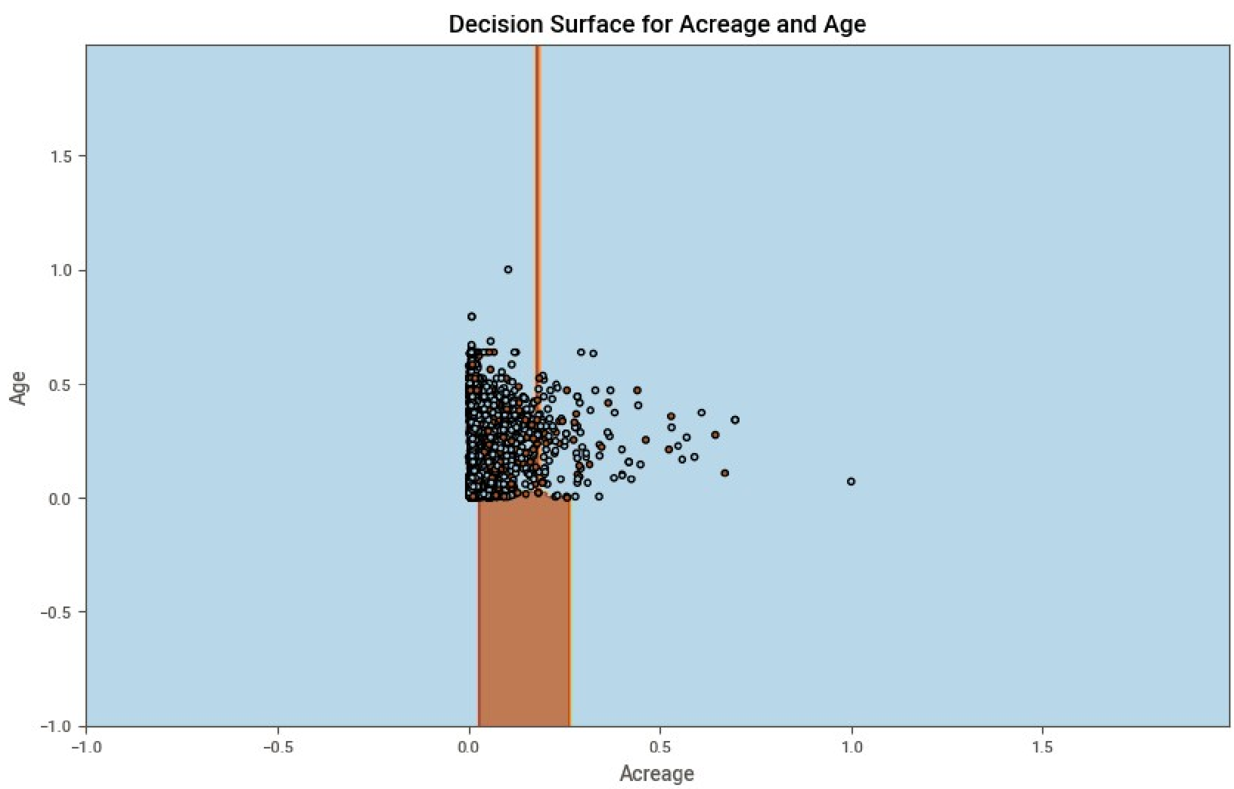
```python
# Generate all unique combinations of 2 numerical features
feature_combinations = combinations(numerical_features, 2)

# Iterate through all pairs for decision surface visualization
for feature_1, feature_2 in feature_combinations:
    # Subset the data for the current pair of features
    X_subset = X_train[[feature_1, feature_2]].values

    # Train a decision tree on the selected features
    dt_model_2d = DecisionTreeClassifier(max_depth=5, random_state=42)
    dt_model_2d.fit(X_subset, y_train)

    # Create a meshgrid for plotting
    x_min, x_max = X_subset[:, 0].min() - 1, X_subset[:, 0].max() + 1
    y_min, y_max = X_subset[:, 1].min() - 1, X_subset[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))

    # Predict for every point in the meshgrid
    Z = dt_model_2d.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the decision surface
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
    plt.scatter(X_subset[:, 0], X_subset[:, 1], c=y_train,
edgecolors='k', cmap=plt.cm.Paired)
    plt.title(f"Decision Surface for {feature_1} and {feature_2}")
    plt.xlabel(feature_1)
    plt.ylabel(feature_2)
    plt.show()
```


Decision Surface for Acreage and Land Value
Land Value
Acreage
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Surface for Acreage and Building Value
Building Value
Acreage
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

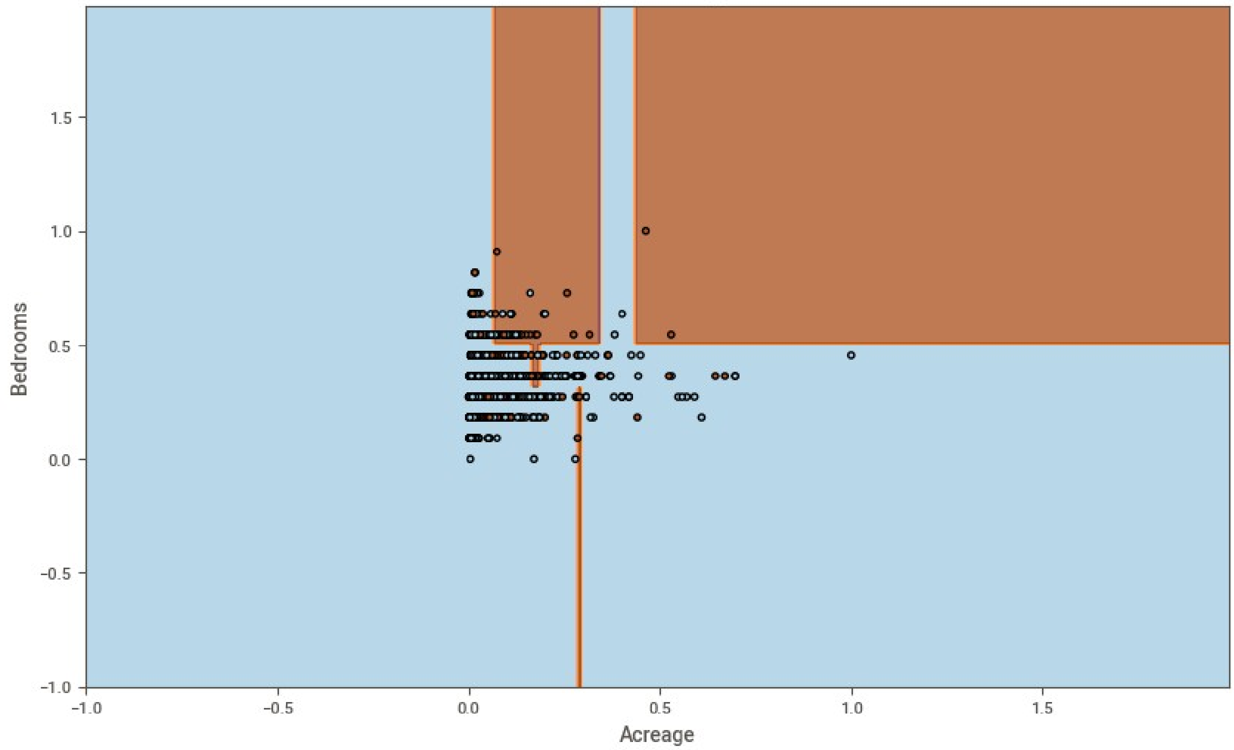


Decision Surface for Acreage and Finished Area
Finished Area
Acreage
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

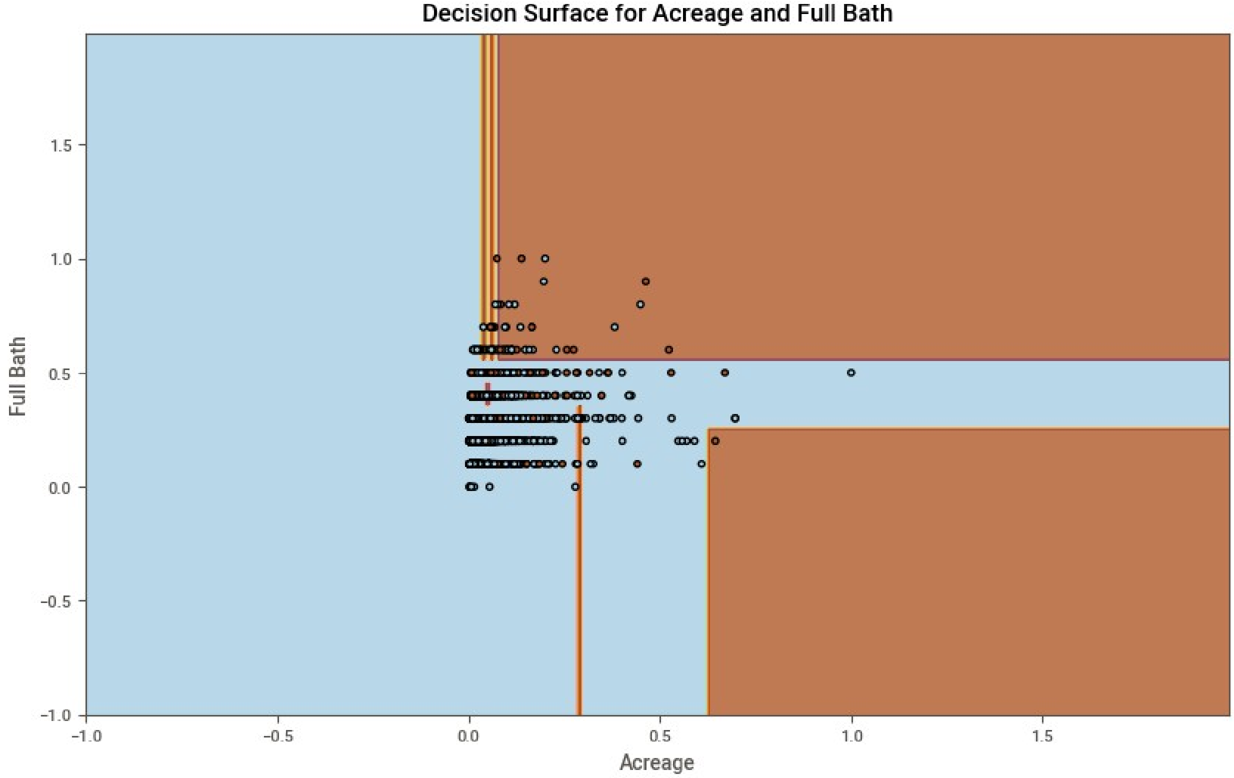


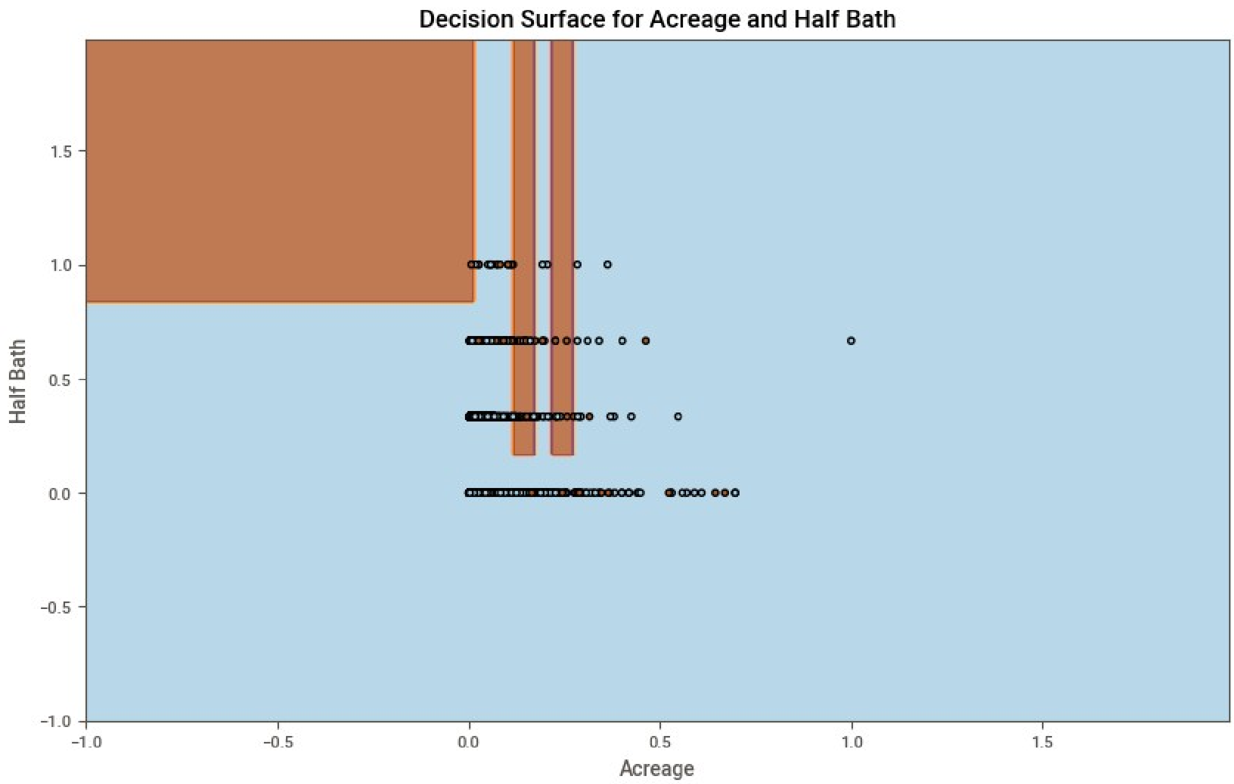
Decision Surface for Acreage and Age
Age
Acreage
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

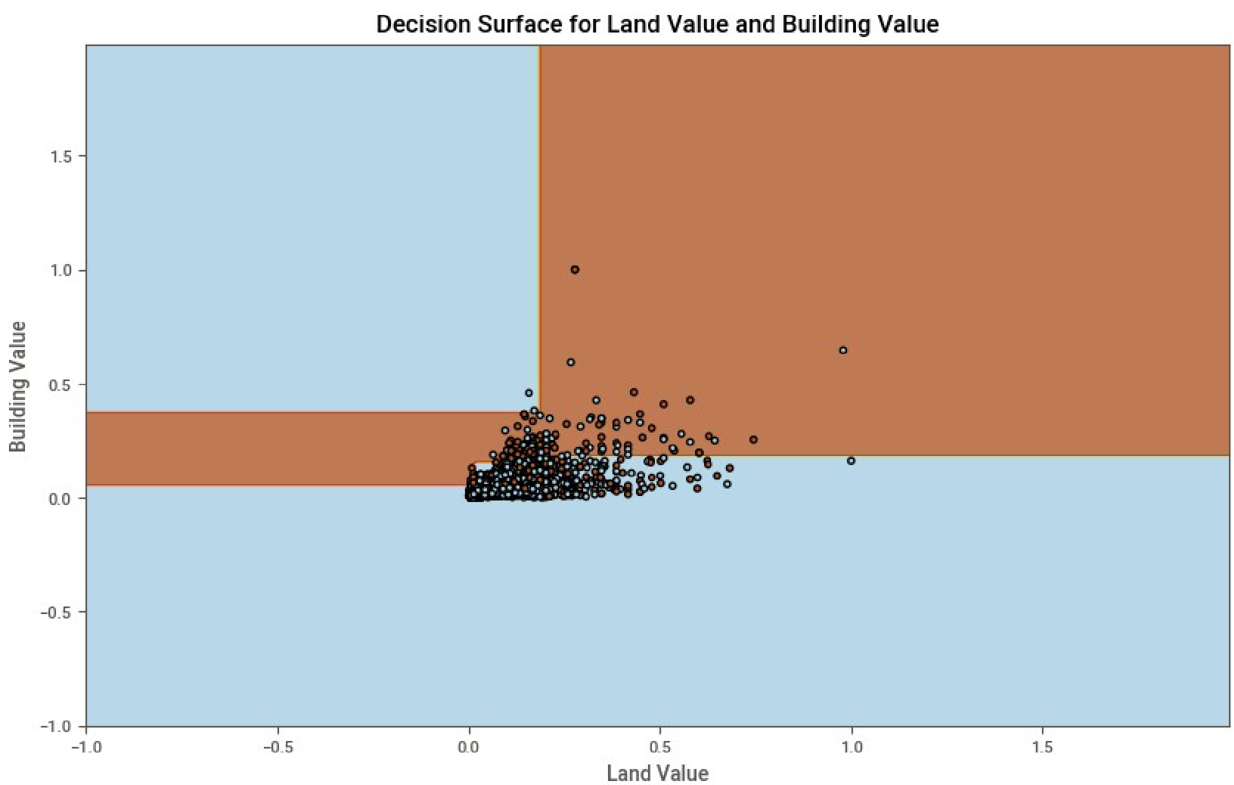
Decision Surface for Acreage and Bedrooms

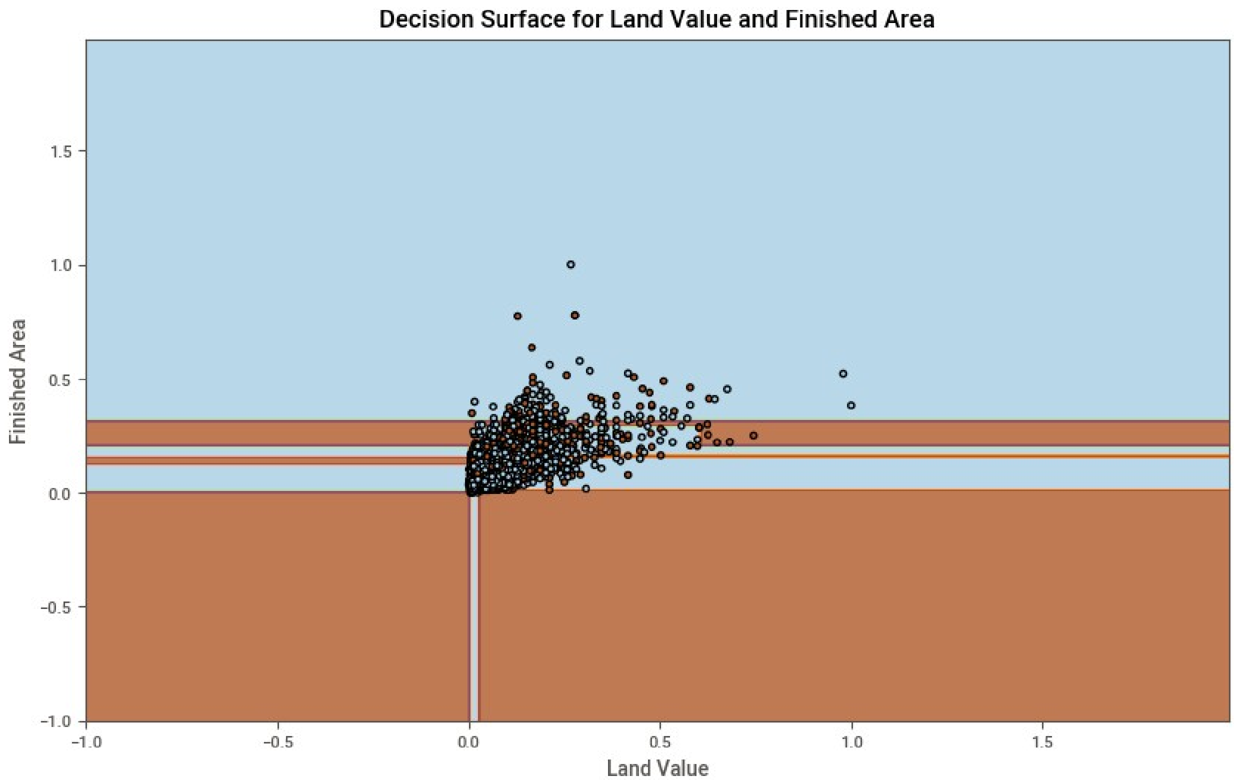


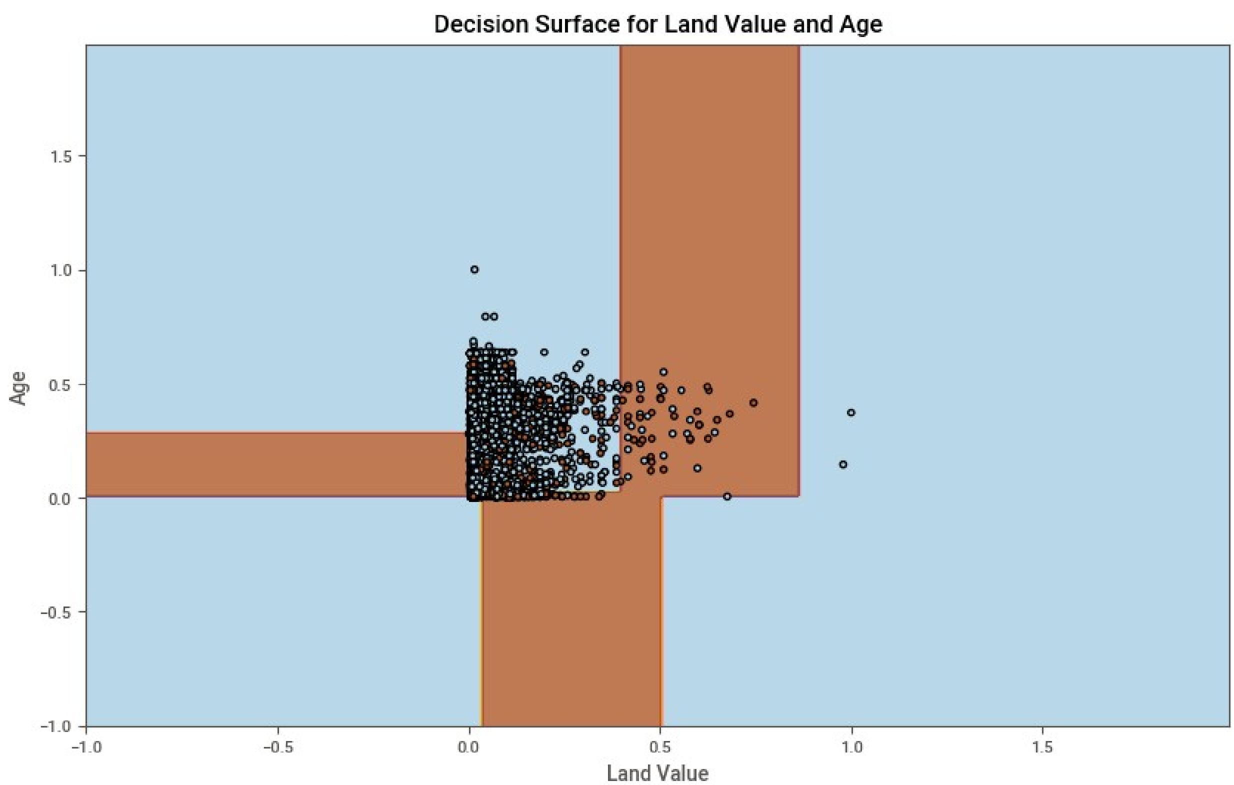
Decision Surface for Acreage and Full Bath

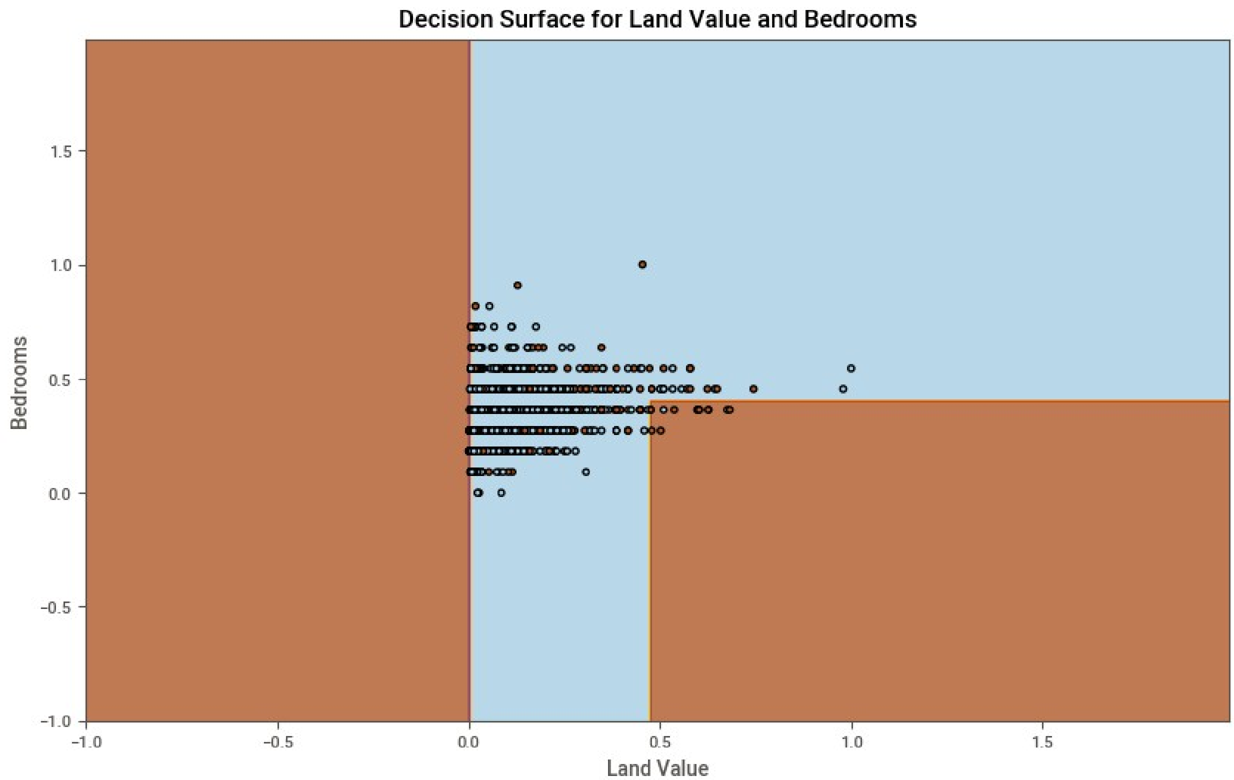


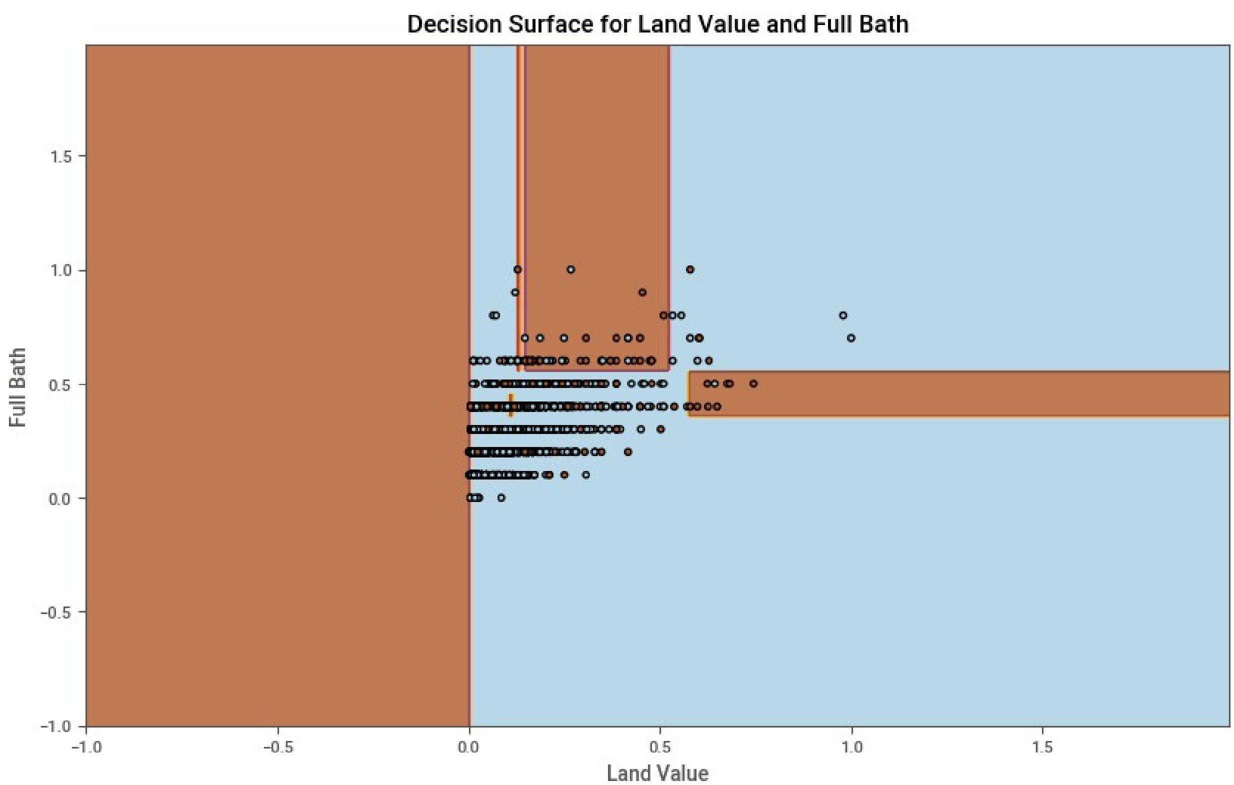

Decision Surface for Acreage and Half Bath
Half Bath
Acreage
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

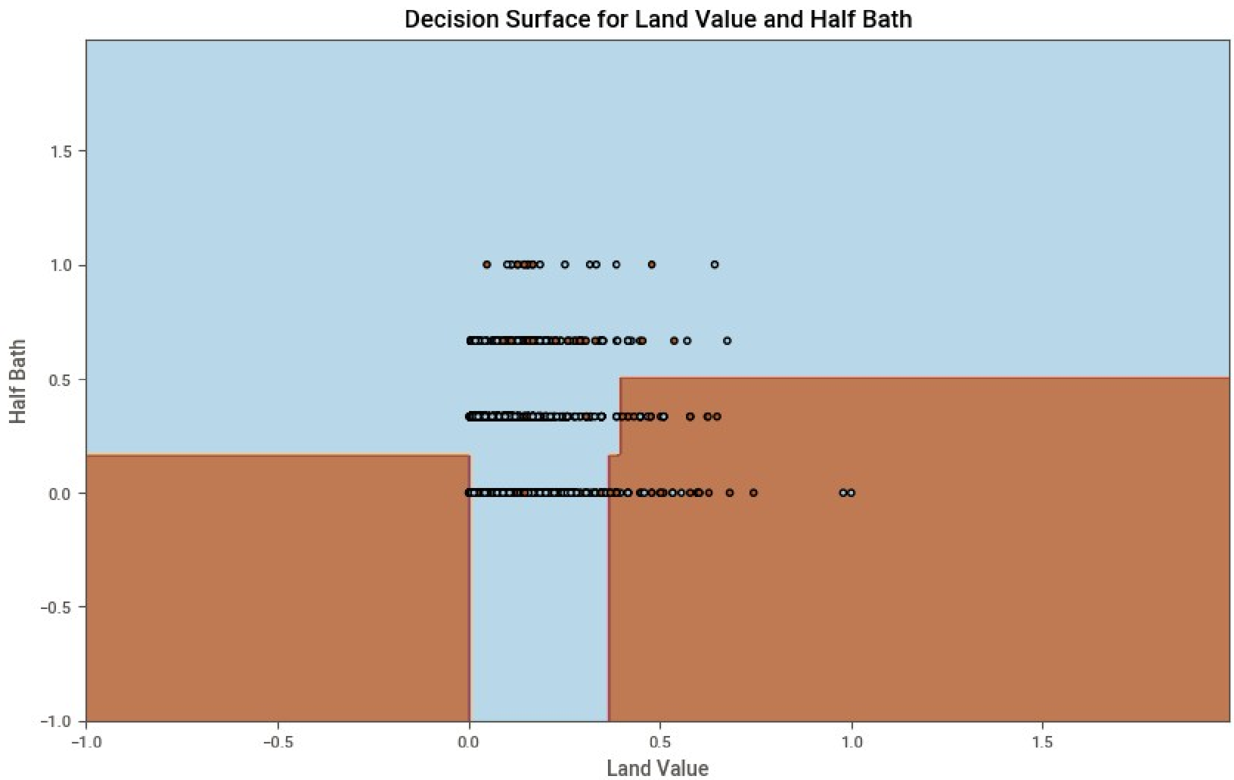

Decision Surface for Land Value and Building Value
Building Value
Land Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

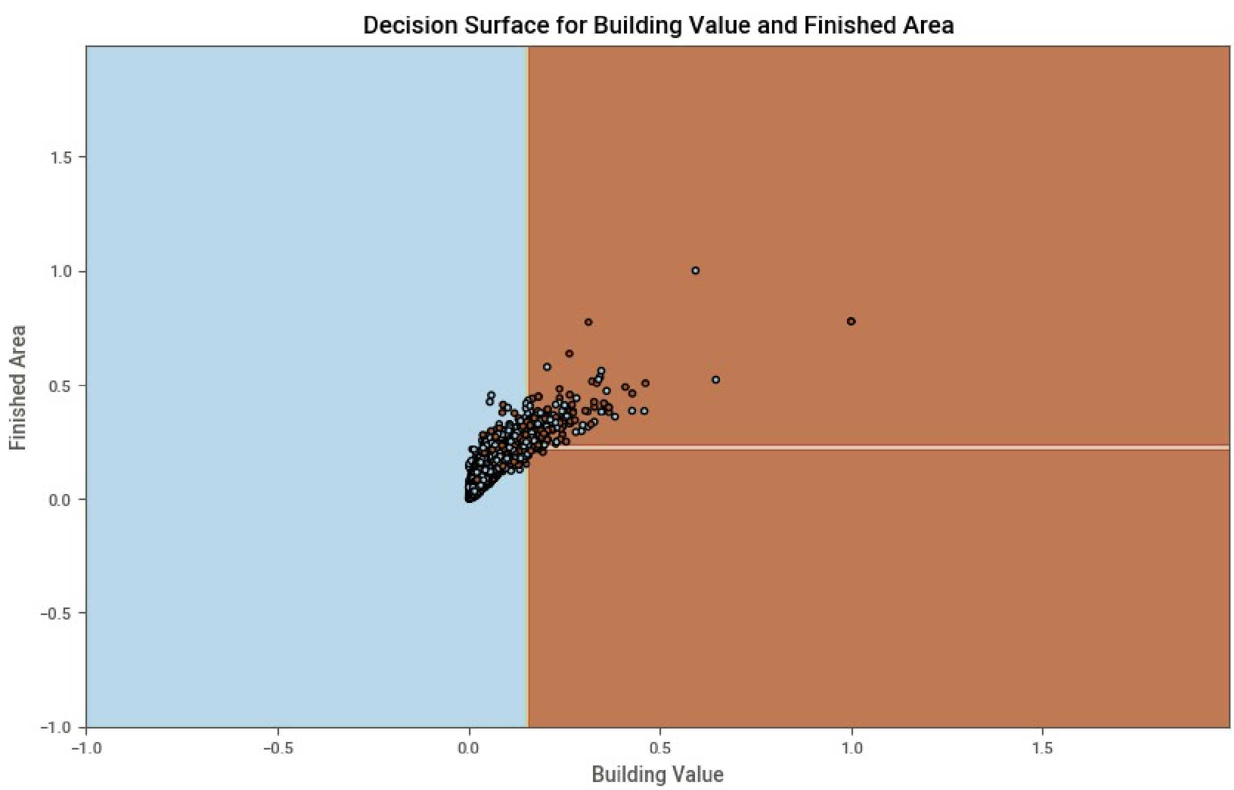

Decision Surface for Land Value and Finished Area
Finished Area
Land Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

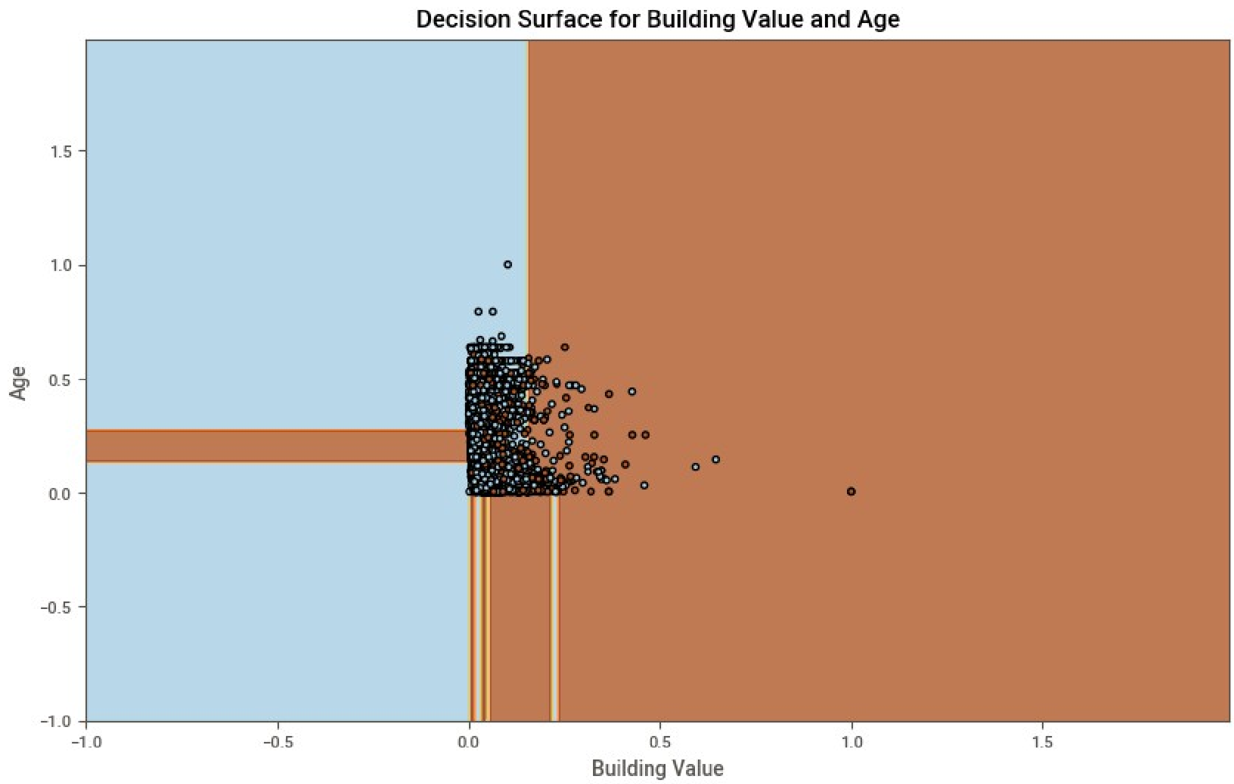

Decision Surface for Land Value and Age
Age
Land Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

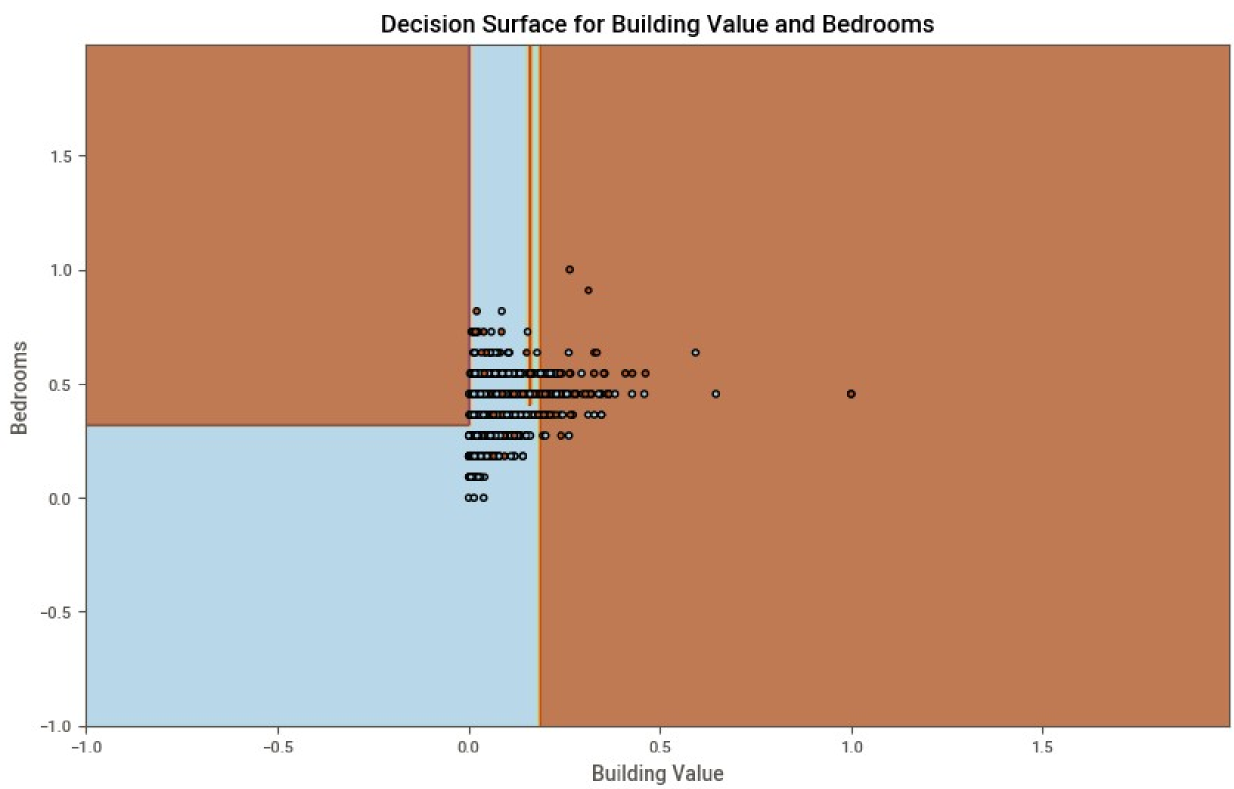

Decision Surface for Land Value and Bedrooms
Bedrooms
Land Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

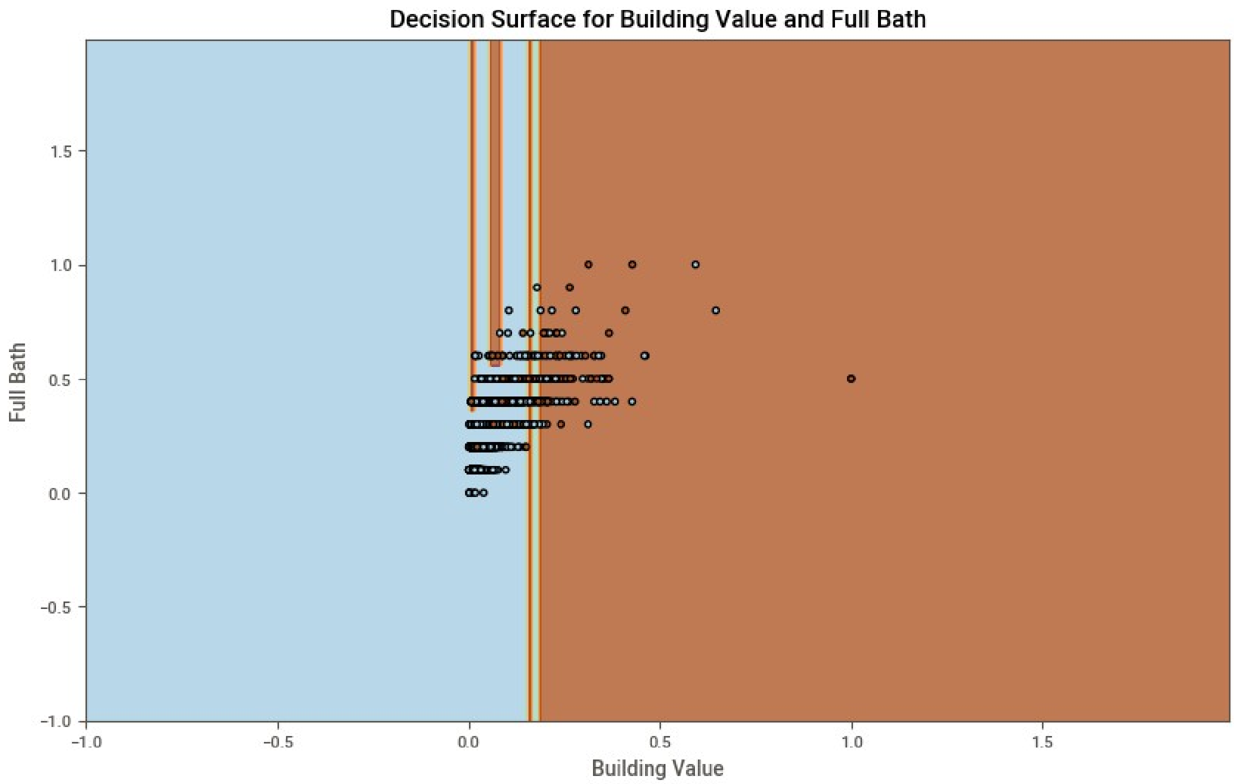

Decision Surface for Land Value and Full Bath
Full Bath
Land Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

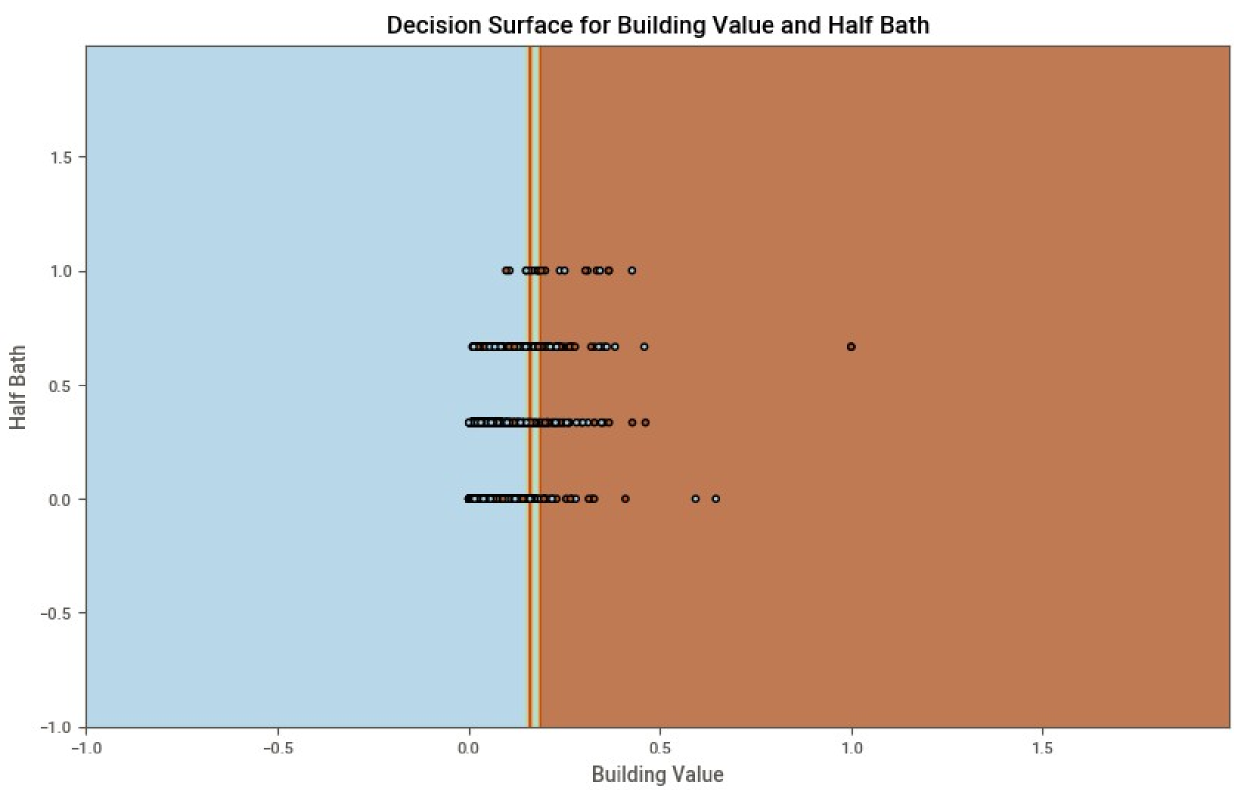

Decision Surface for Land Value and Half Bath
Half Bath
Land Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Surface for Building Value and Finished Area
Finished Area
Building Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

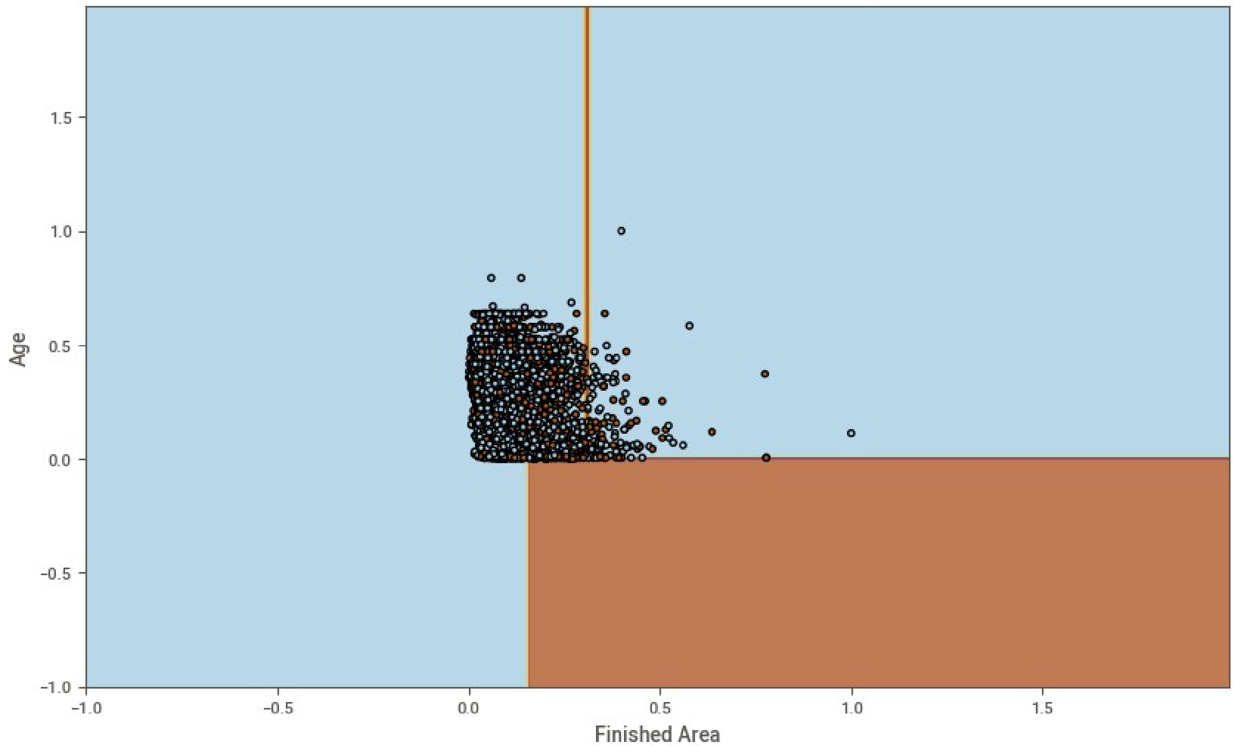

Decision Surface for Building Value and Age
Age
Building Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Surface for Building Value and Bedrooms
Bedrooms
Building Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

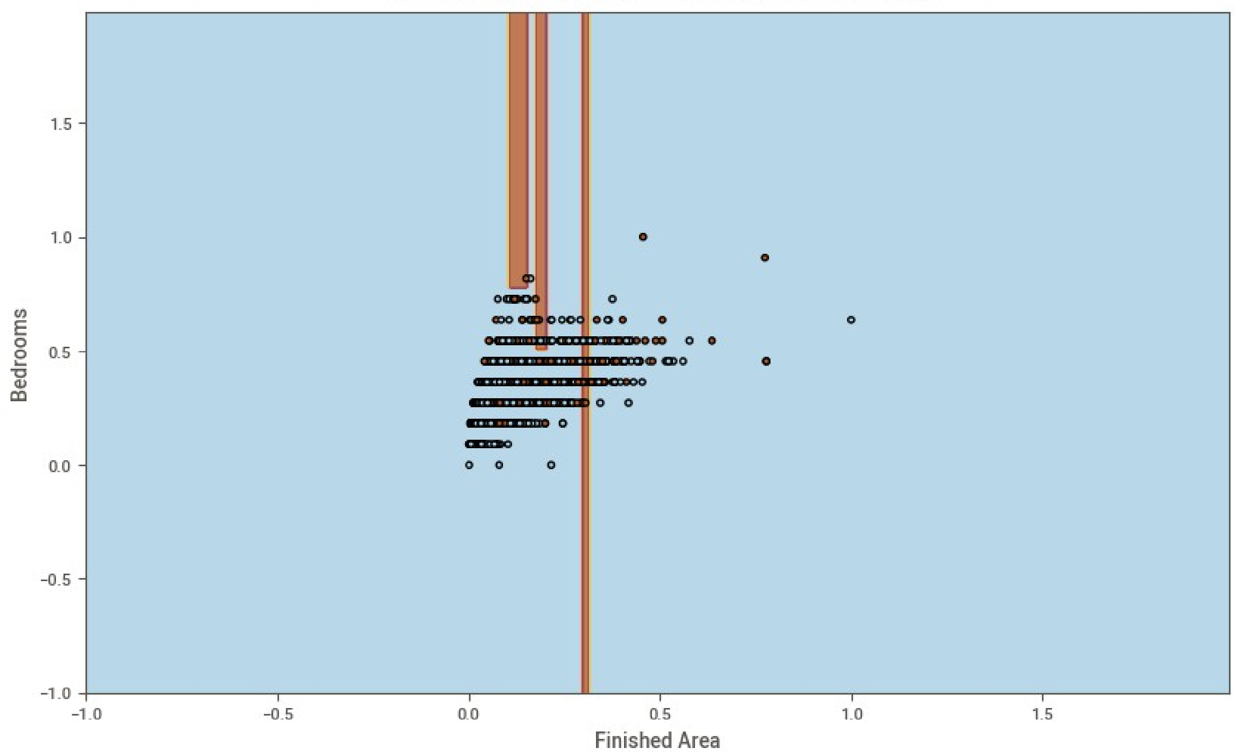

Decision Surface for Building Value and Full Bath
Full Bath
Building Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

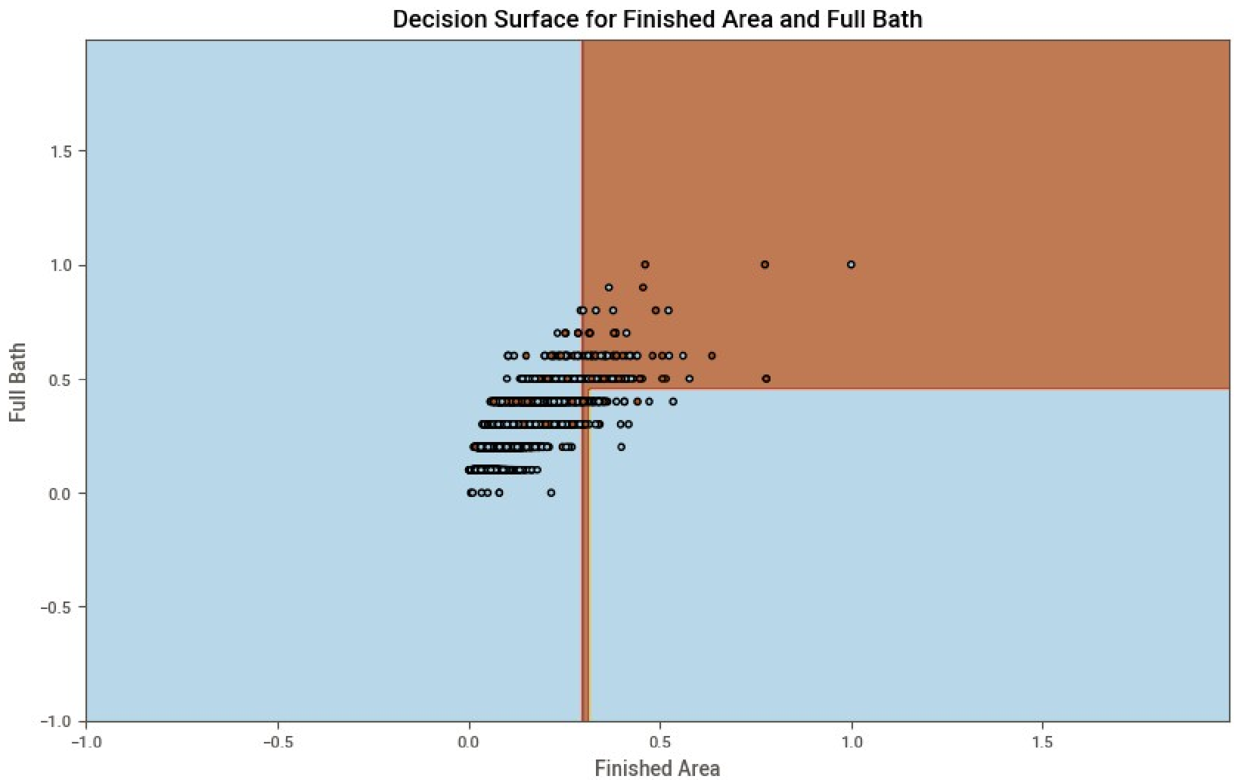

Decision Surface for Building Value and Half Bath
Half Bath
Building Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

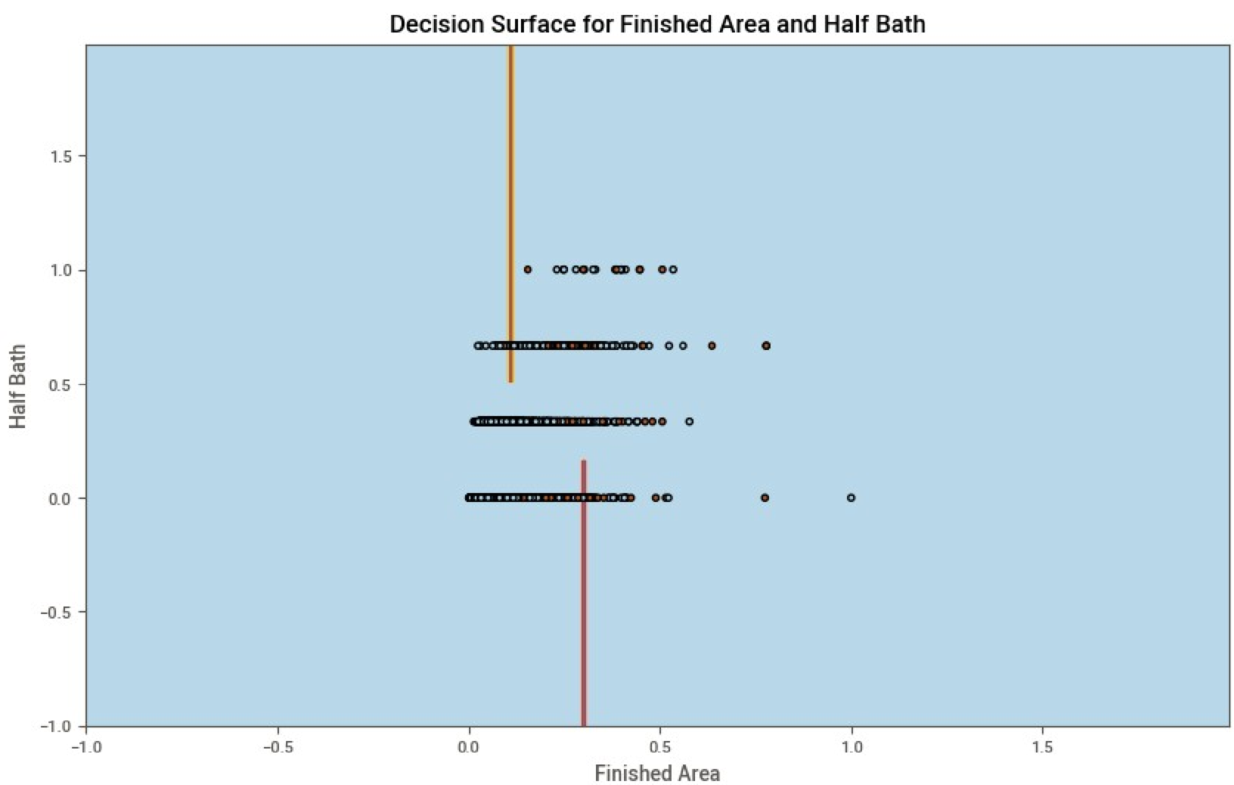
Decision Surface for Finished Area and Age

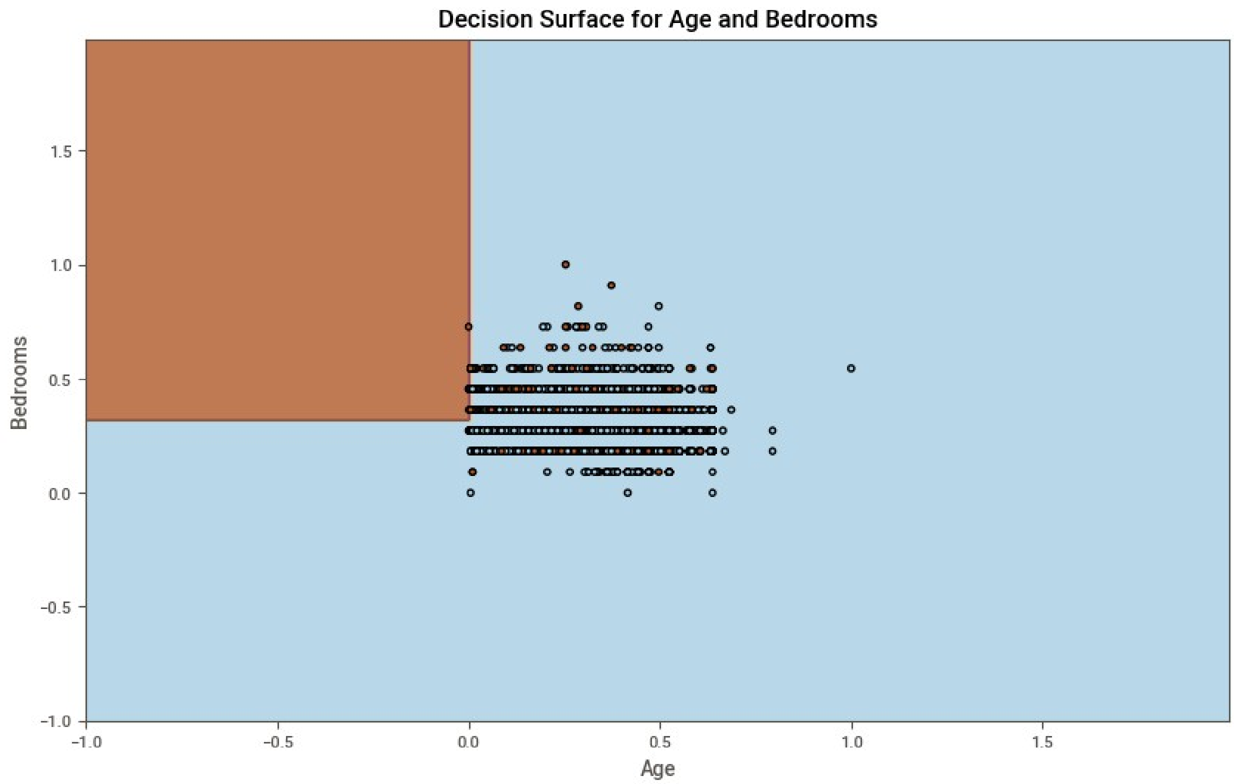


Decision Surface for Finished Area and Bedrooms

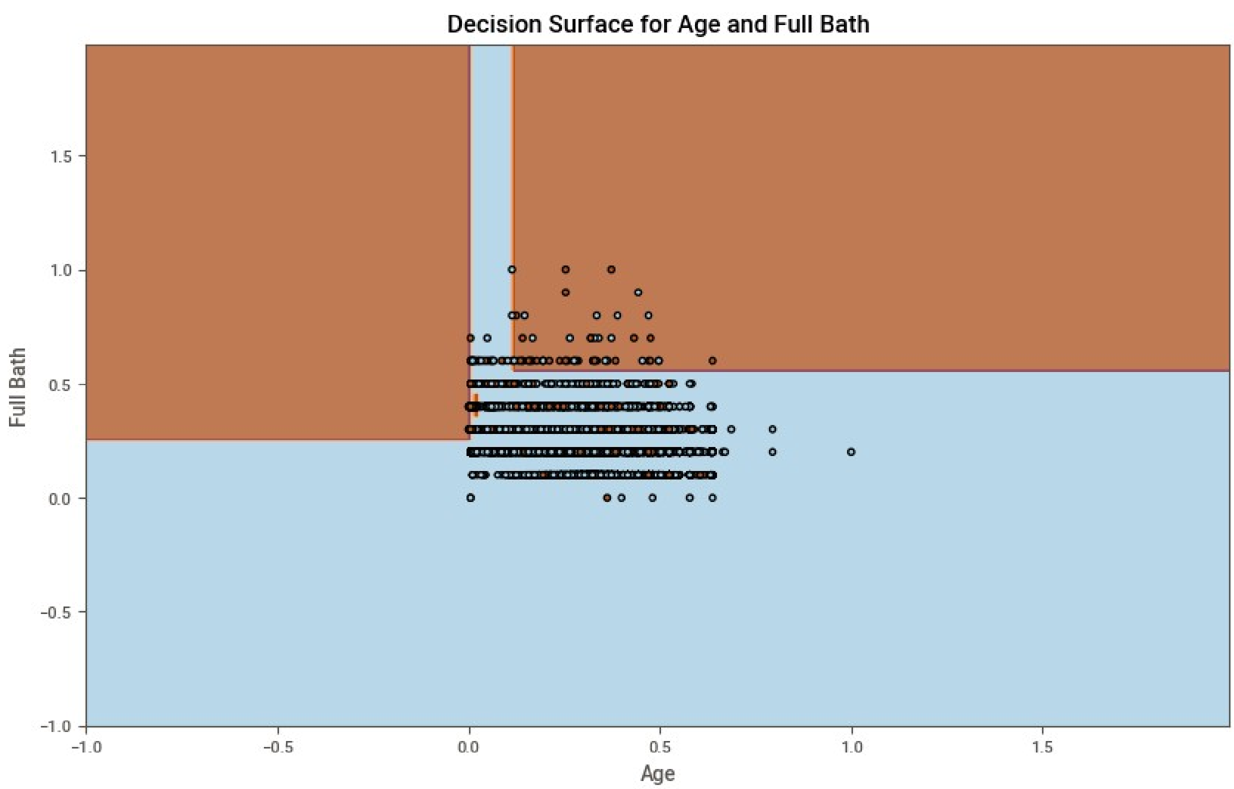



Decision Surface for Finished Area and Full Bath
Full Bath
Finished Area
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Surface for Finished Area and Half Bath
Half Bath
Finished Area
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

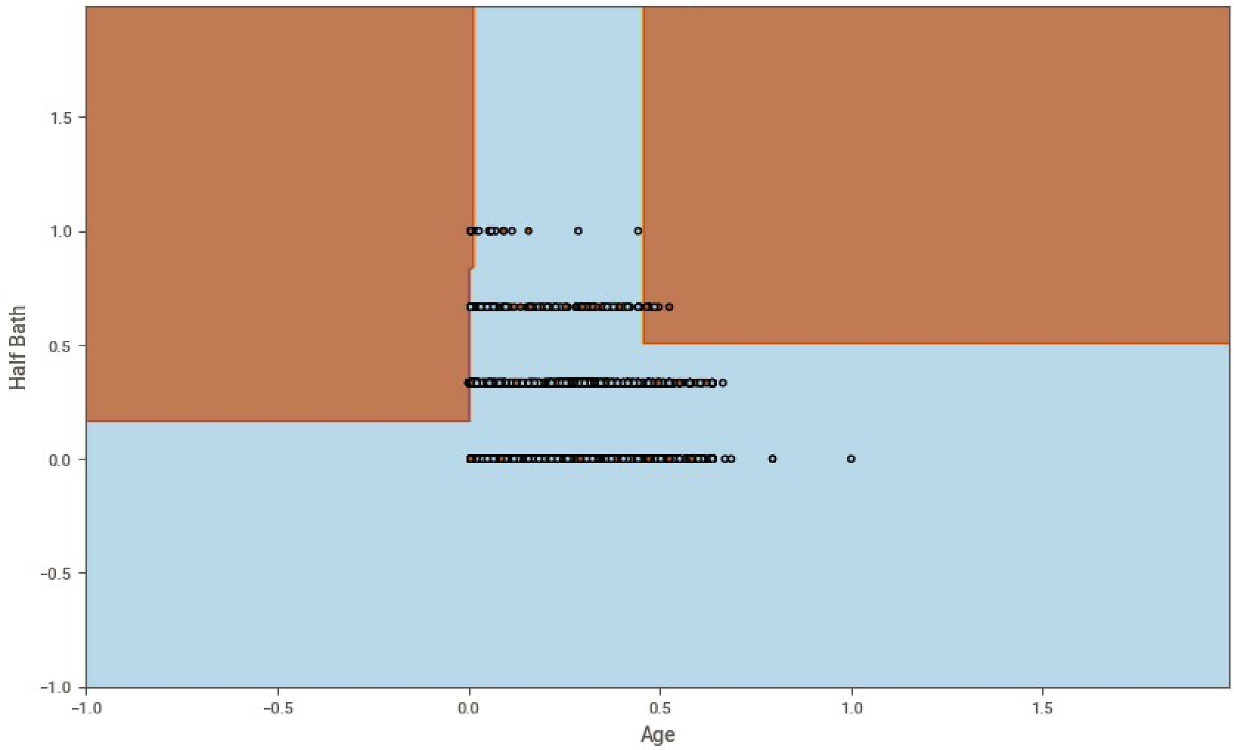


Decision Surface for Age and Bedrooms
Bedrooms
Age
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

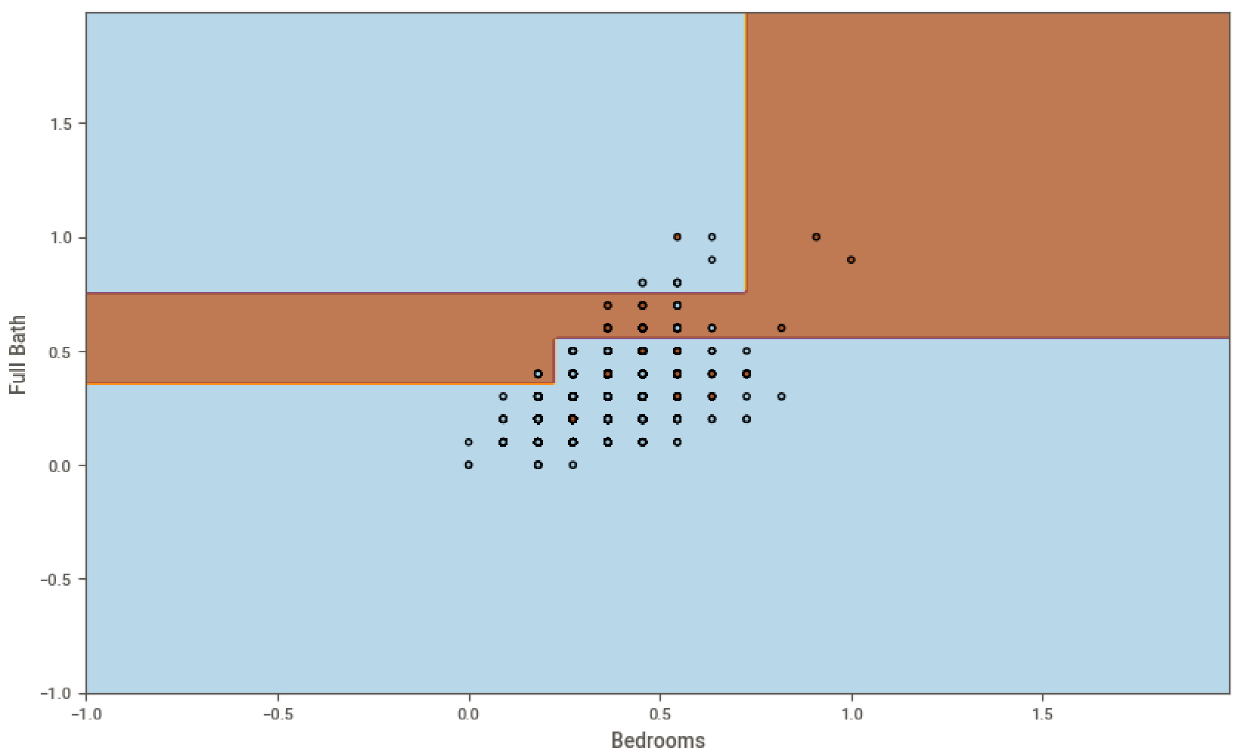


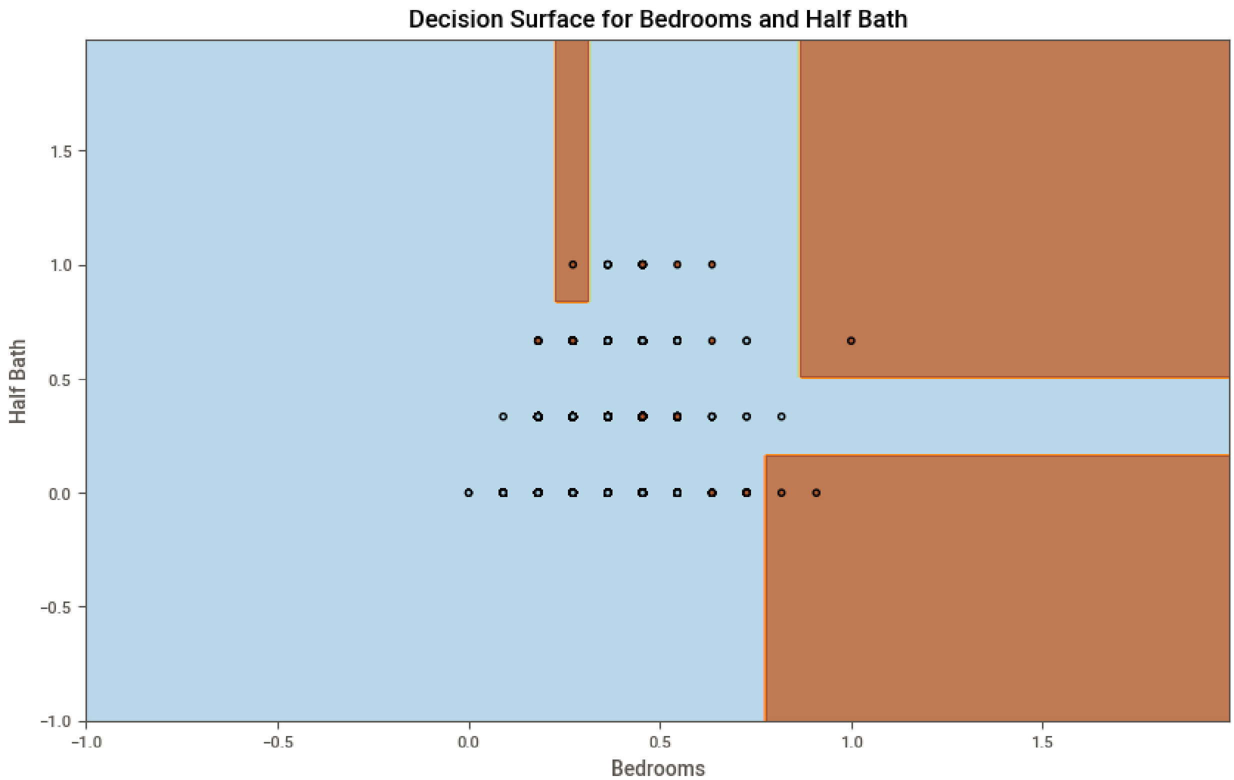
Decision Surface for Age and Full Bath
Full Bath
Age
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

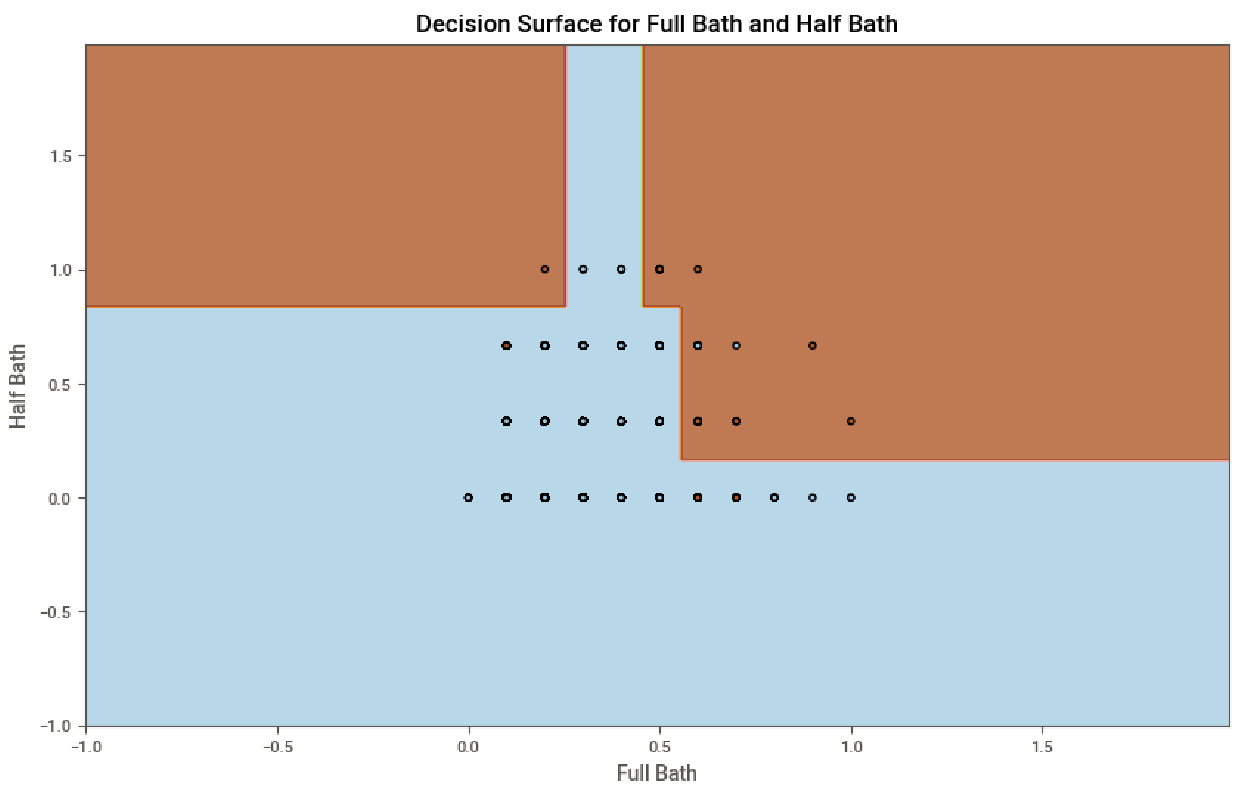
Decision Surface for Age and Half Bath



Decision Surface for Bedrooms and Full Bath



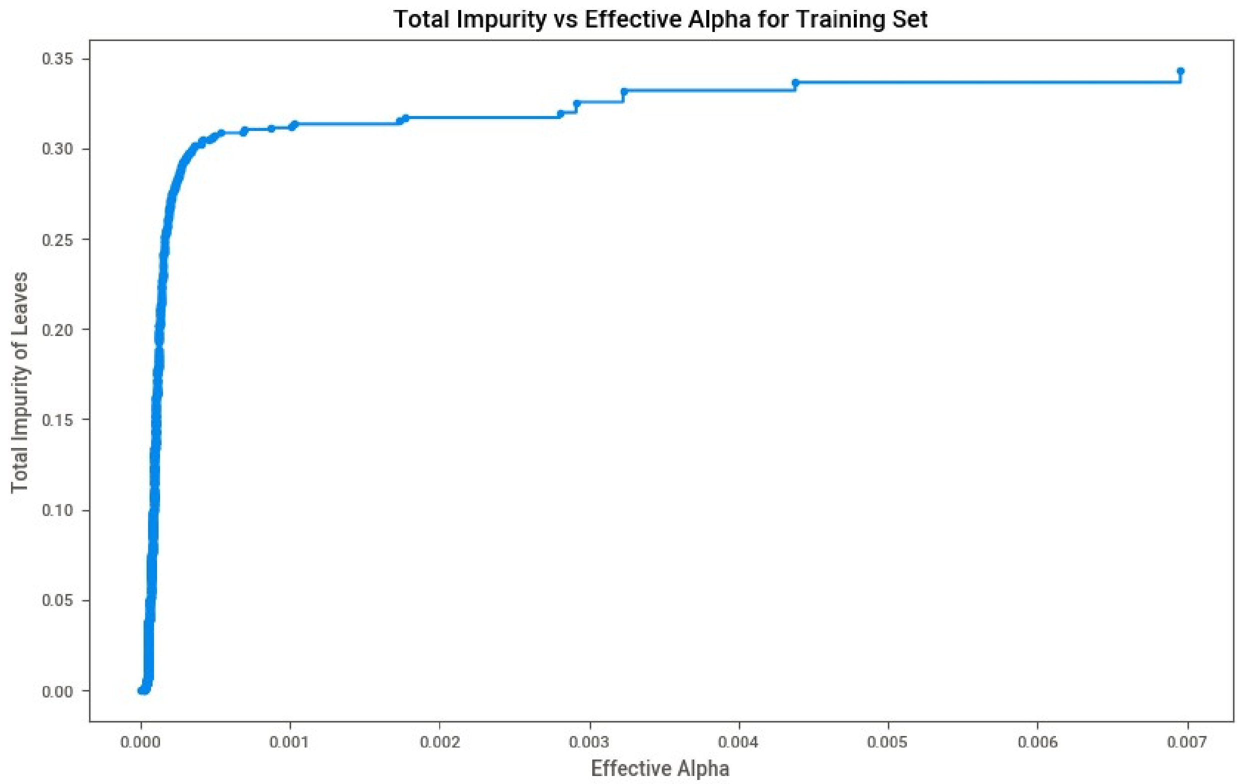

Decision Surface for Bedrooms and Half Bath
Half Bath
Bedrooms
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Surface for Full Bath and Half Bath
Half Bath
Full Bath
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

Minimal Cost complexity Pruning

```
import time
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

# Initialize the DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)

# Measure the time for cost complexity pruning path calculation
start_time = time.time()
path = clf.cost_complexity_pruning_path(X_train, y_train)
elapsed_time = time.time() - start_time

# Extract alphas and impurities
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# Print the time elapsed for pruning path calculation
print(f"Time elapsed for calculating pruning path: {elapsed_time:.4f}
seconds")

# Plot Total Impurity vs Effective Alpha
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o",
drawstyle="steps-post")
ax.set_xlabel("Effective Alpha")
ax.set_ylabel("Total Impurity of Leaves")
ax.set_title("Total Impurity vs Effective Alpha for Training Set")
plt.show()
```

```
Time elapsed for calculating pruning path: 0.1440 seconds
```

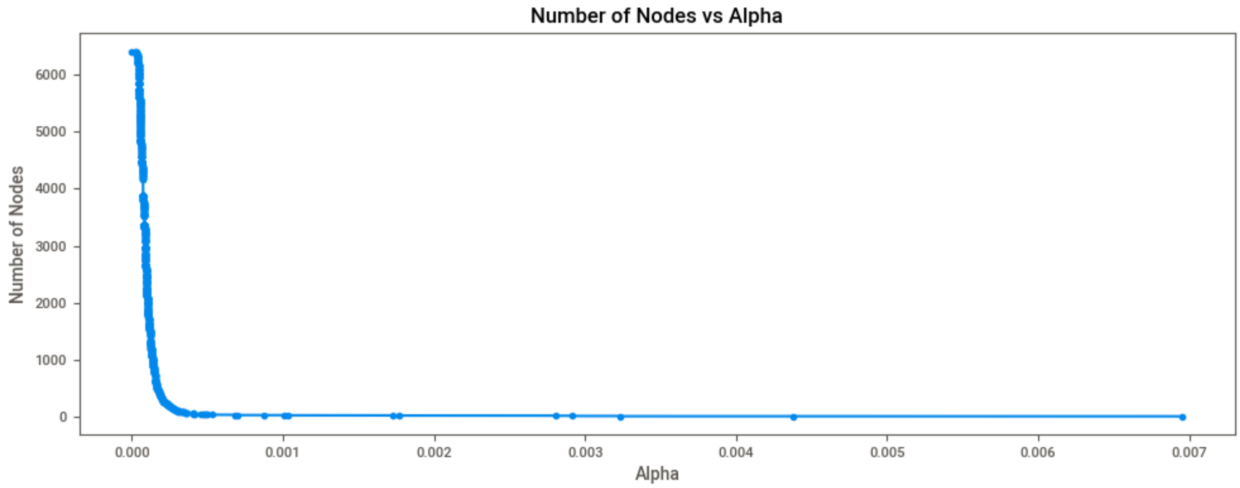


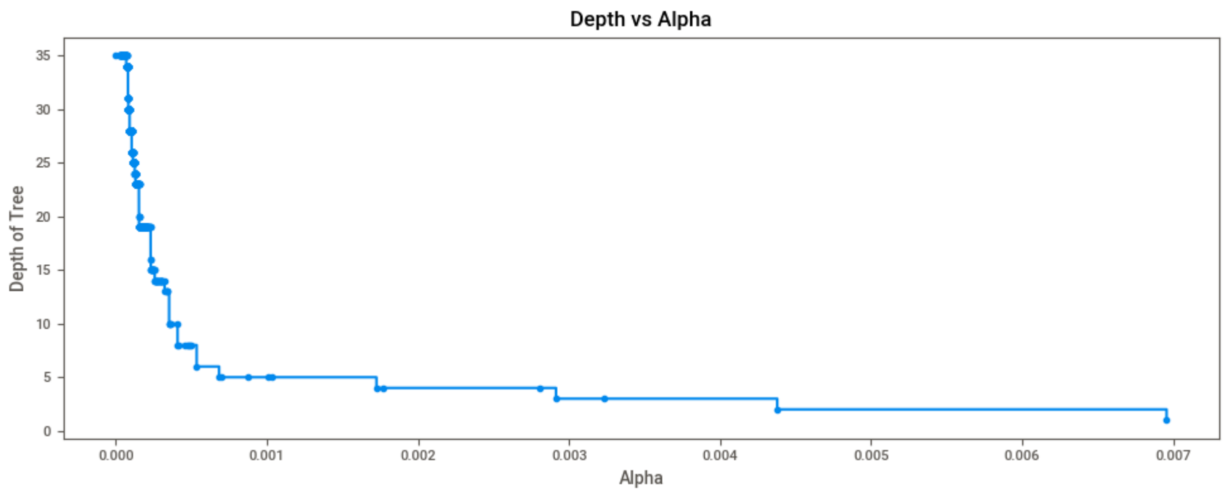

```
import time

# Create a list to store classifiers
clfs = []

# Measure the time taken for training all models
start_time = time.time()

# Iterate over all ccp_alpha values and train DecisionTreeClassifier
for each
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

# Calculate elapsed time
elapsed_time = time.time() - start_time

# Print the elapsed time
print(f"Time elapsed for training all models: {elapsed_time:.4f}
seconds")

# Print the number of nodes and the ccp_alpha for the last tree
print(
    "Number of nodes in the last tree is: {} with ccp_alpha:
{}".format(
```

```
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

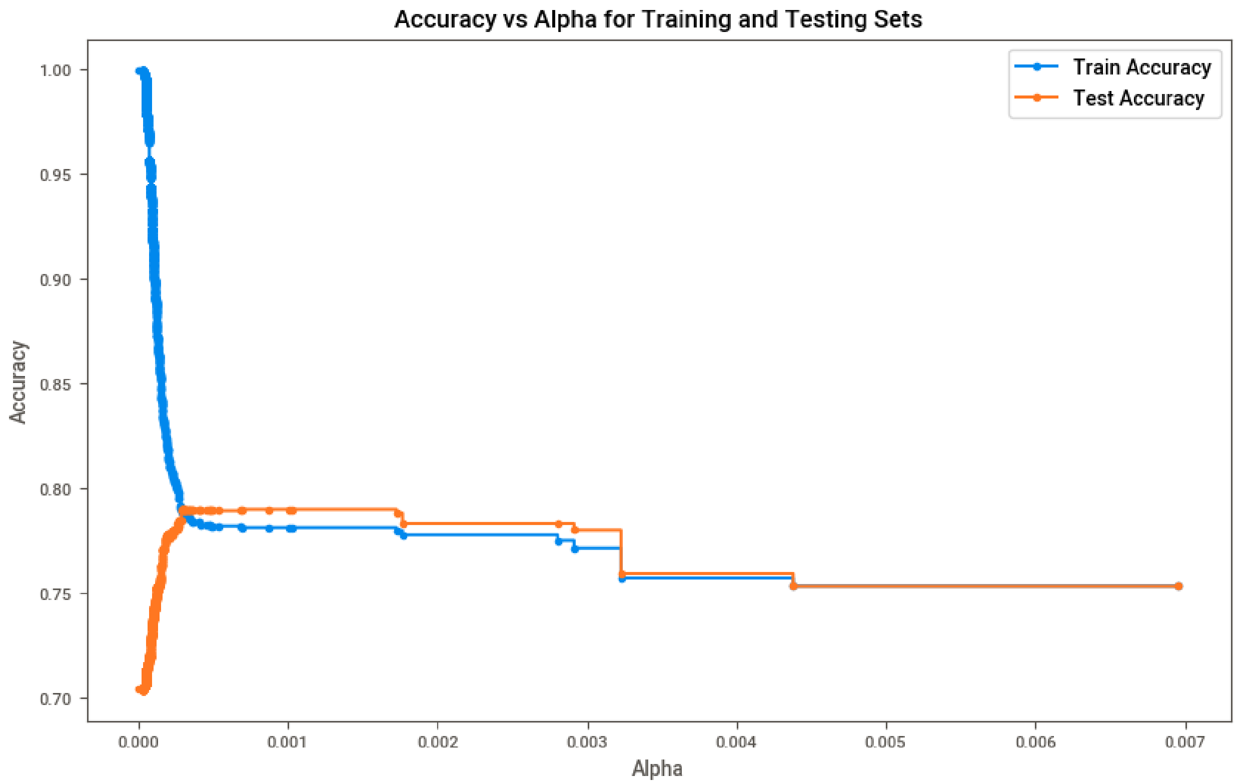
```
Time elapsed for training all models: 178.9514 seconds
Number of nodes in the last tree is: 1 with ccp_alpha:
0.02811421740588199
```

```
import matplotlib.pyplot as plt

# Remove the last element to avoid the fully pruned tree
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

# Extract the number of nodes and depth for each classifier
node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]

# Create subplots for visualization
fig, ax = plt.subplots(2, 1, figsize=(10, 8))

# Plot Number of Nodes vs Alpha
ax[0].plot(ccp_alphas, node_counts, marker="o", drawstyle="steps-
post")
ax[0].set_xlabel("Alpha")
ax[0].set_ylabel("Number of Nodes")
ax[0].set_title("Number of Nodes vs Alpha")

# Plot Depth vs Alpha
ax[1].plot(ccp_alphas, depth, marker="o", drawstyle="steps-post")
ax[1].set_xlabel("Alpha")
ax[1].set_ylabel("Depth of Tree")
ax[1].set_title("Depth vs Alpha")

# Adjust layout for better spacing
fig.tight_layout()

# Show the plots
plt.show()
```






```
# Compute training and testing accuracy scores for each classifier
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

# Plot Accuracy vs Alpha for training and testing sets
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_xlabel("Alpha")
ax.set_ylabel("Accuracy")
ax.set_title("Accuracy vs Alpha for Training and Testing Sets")

# Plot training and testing accuracy
ax.plot(ccp_alphas, train_scores, marker="o", label="Train Accuracy",
drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker="o", label="Test Accuracy",
drawstyle="steps-post")

# Add legend and show plot
ax.legend()
plt.show()
```

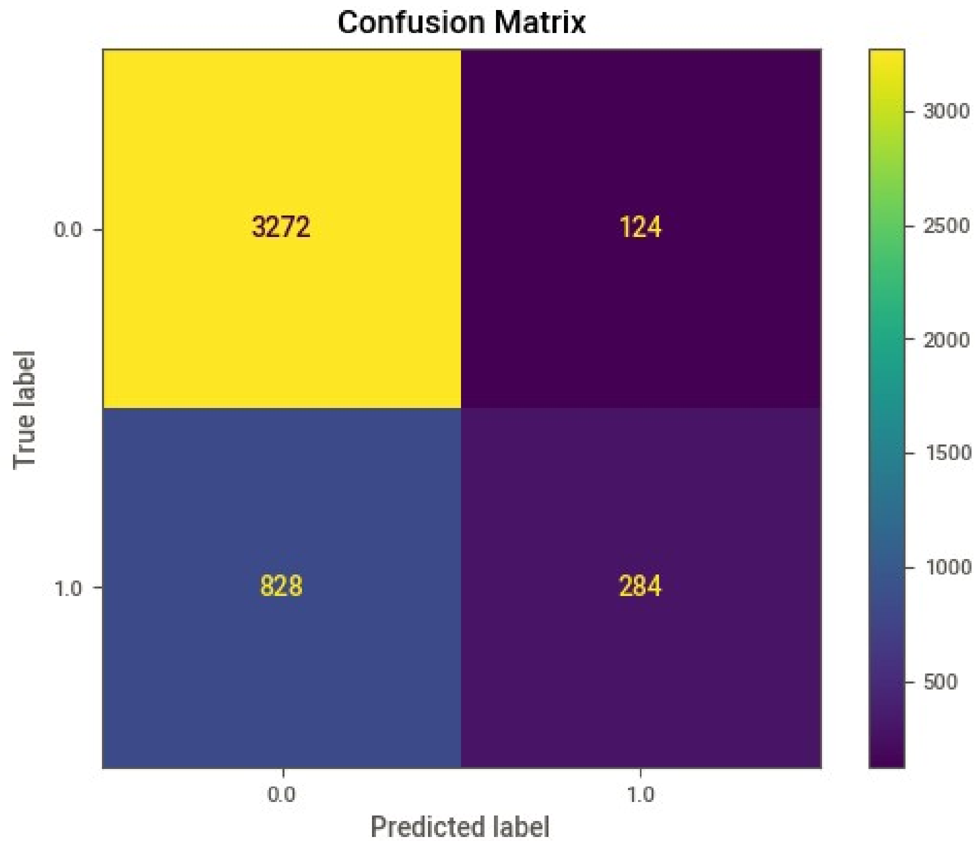



```
from sklearn.metrics import classification_report
import time

# Set the optimal ccp_alpha value
optimal_ccp_alpha = 0.0033

# Train the pruned DecisionTreeClassifier using the optimal ccp_alpha
pruned_model = DecisionTreeClassifier(random_state=0,
ccp_alpha=optimal_ccp_alpha)

# Train the model
pruned_model.fit(X_train, y_train)

# Measure prediction time
start_time = time.time()
y_pred3 = pruned_model.predict(X_test)
elapsed_time = time.time() - start_time

# Generate the classification report
report = classification_report(y_test, y_pred3)

# Print results
print(f"Time elapsed for pruned tree to predict on the test set:
{elapsed_time:.4f} seconds")
print("\nClassification Report:")
print(report)
```

```
Time elapsed for pruned tree to predict on the test set: 0.0010
seconds

Classification Report:
              precision    recall  f1-score   support

         0.0       0.76      1.00      0.86      3396
         1.0       0.78      0.03      0.07      1112

    accuracy                           0.76      4508
   macro avg       0.77      0.52      0.46      4508
weighted avg       0.76      0.76      0.67      4508
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay



# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Display results
print(f"Time elapsed to predict on the test set: {elapsed_time:.4f}
seconds")
print("\nClassification Report:")
print(report)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=pruned_model.classes_)
disp.plot(cmap="viridis", values_format="d")
plt.title("Confusion Matrix")
plt.show()
```

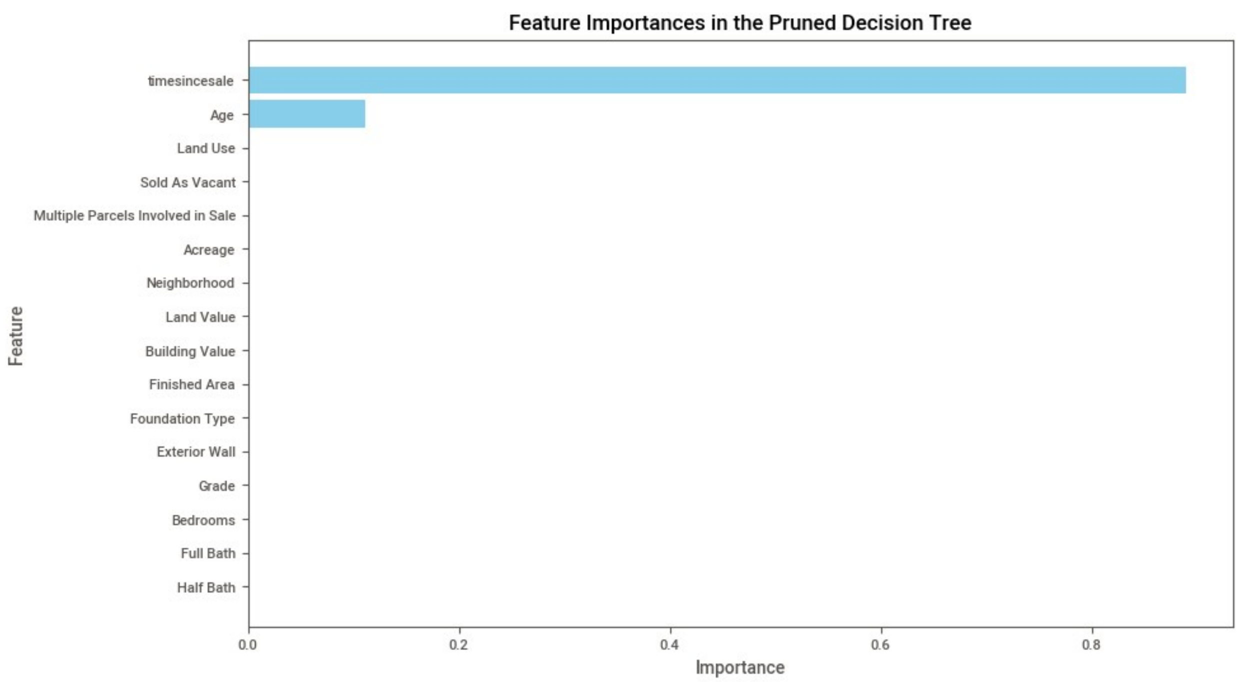
```
Time elapsed to predict on the test set: 0.0010 seconds

Classification Report:
              precision    recall  f1-score   support

         0.0       0.76      1.00      0.86      3396
         1.0       0.78      0.03      0.07      1112

    accuracy                           0.76      4508
   macro avg       0.77      0.52      0.46      4508
weighted avg       0.76      0.76      0.67      4508
```




```python
# Retrieve feature importances from the pruned decision tree
feature_importances = pruned_model.feature_importances_

# Create a DataFrame to display feature importances alongside feature
names
importance_df = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": feature_importances
}).sort_values(by="Importance", ascending=False)

# Print the feature importance table
print("Feature Importances:")
print(importance_df)

# Plot the feature importances
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.barh(importance_df["Feature"], importance_df["Importance"],
color="skyblue")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.title("Feature Importances in the Pruned Decision Tree")
```

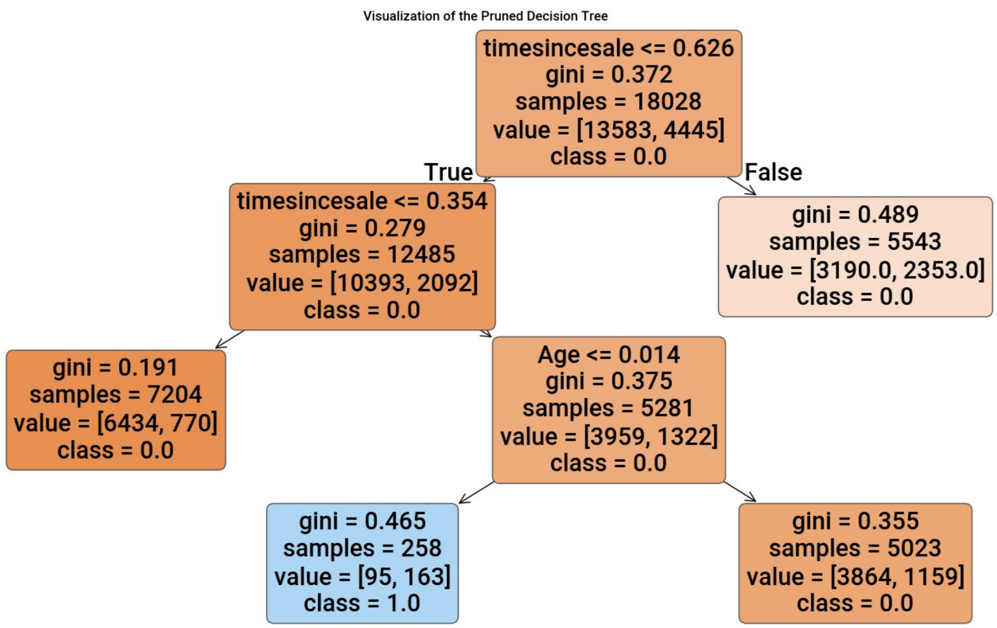
```
plt.gca().invert_yaxis()
plt.show()
```

```
Feature Importances:
                                Feature  Importance
14                        timesincesale    0.889006
15                                  Age    0.110994
0                              Land Use    0.000000
1                        Sold As Vacant    0.000000
2    Multiple Parcels Involved in Sale    0.000000
3                               Acreage    0.000000
4                          Neighborhood    0.000000
5                            Land Value    0.000000
6                        Building Value    0.000000
7                         Finished Area    0.000000
8                       Foundation Type    0.000000
9                         Exterior Wall    0.000000
10                                Grade    0.000000
11                             Bedrooms    0.000000
12                            Full Bath    0.000000
13                            Half Bath    0.000000
```



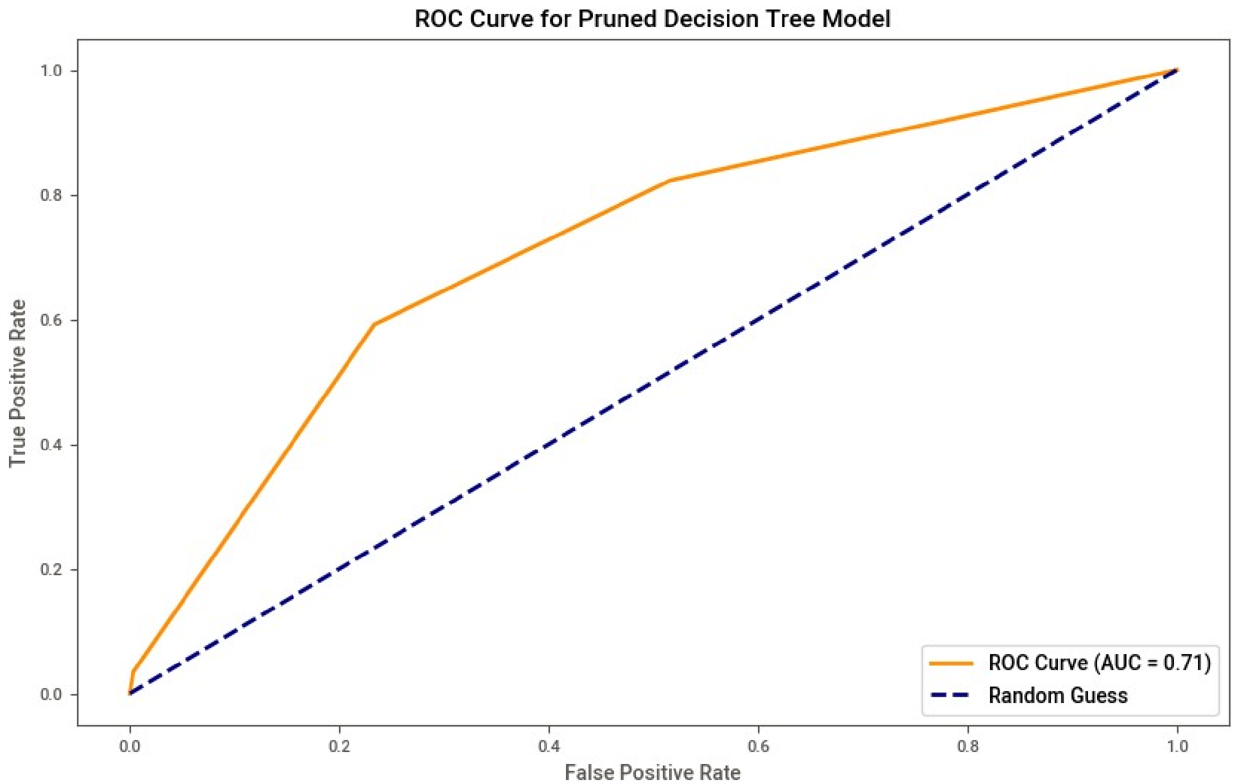

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Visualize the pruned decision tree
plt.figure(figsize=(20, 10))
plot_tree(
```

```
    pruned_model,
    feature_names=X_train.columns,  # Replace with your feature column names
    class_names=[str(cls) for cls in pruned_model.classes_],  # Replace with your class names if applicable
    filled=True,
    rounded=True
)
plt.title("Visualization of the Pruned Decision Tree")
plt.show()
```




```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming pruned_model is the trained Decision Tree model with optimal ccp_alpha
# Ensure X_test and y_test are defined and available

# Get probability scores for the positive class
y_prob3 = pruned_model.predict_proba(X_test)[:, 1]  # Probability scores for the positive class

# Compute ROC curve and AUC
fpr3, tpr3, thresholds3 = roc_curve(y_test, y_prob3)
roc_auc3 = auc(fpr3, tpr3)

# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr3, tpr3, color='darkorange', lw=2, label=f'ROC Curve (AUC
```

```
= {roc_auc3:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Pruned Decision Tree Model')
plt.legend(loc="lower right")
plt.show()
```



Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
import time

# Initialize Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Hyperparameter grid for Random Forest
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
```

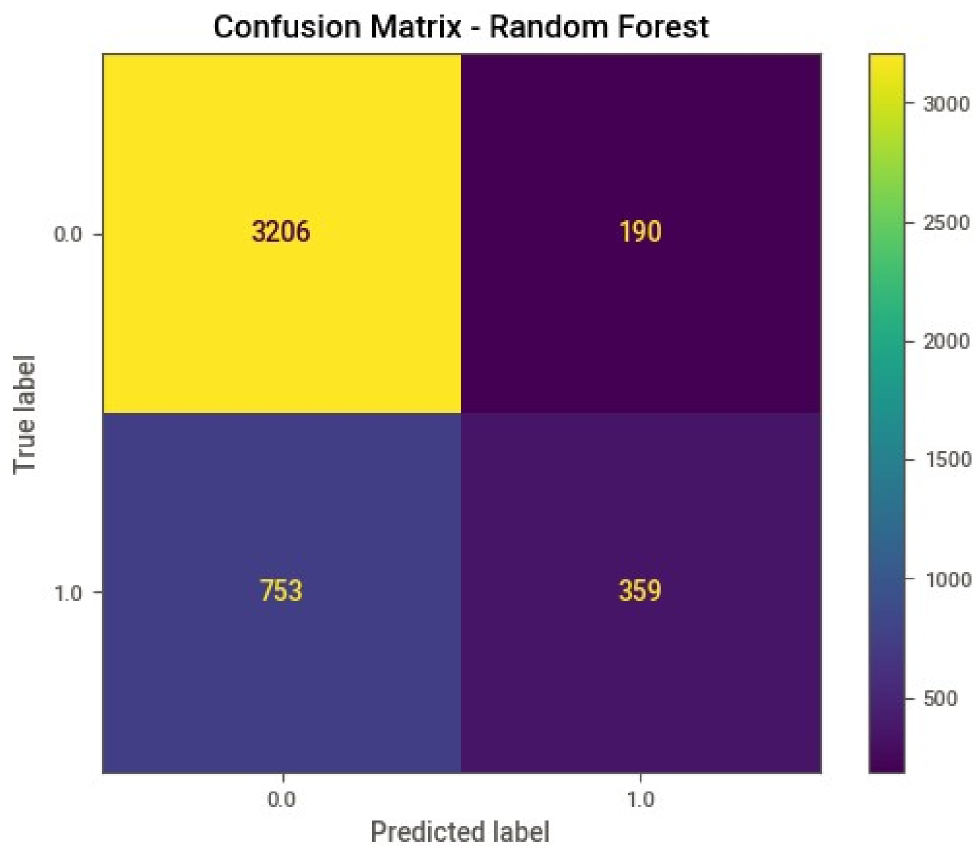
```
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None]
}

# Grid Search for Random Forest
rf_grid_search = GridSearchCV(estimator=rf_model,
param_grid=rf_param_grid, cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)

# Measure training time
start_train_time = time.time()
rf_grid_search.fit(X_train, y_train)
elapsed_train_time = time.time() - start_train_time

# Retrieve the best model and its parameters
rf_best_model = rf_grid_search.best_estimator_

# Measure prediction time
start_pred_time = time.time()
rf_y_pred = rf_best_model.predict(X_test)
elapsed_pred_time = time.time() - start_pred_time
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits

Best Parameters (Random Forest): {'max_depth': None, 'max_features':
'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
200}
Accuracy (Random Forest): 0.79
Training Time: 528.9220 seconds
Prediction Time: 0.1356 seconds

Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.94      0.87      3396
         1.0       0.65      0.32      0.43      1112

    accuracy                           0.79      4508
   macro avg       0.73      0.63      0.65      4508
weighted avg       0.77      0.79      0.76      4508
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Evaluate Random Forest
rf_accuracy = accuracy_score(y_test, rf_y_pred)
rf_classification_report = classification_report(y_test, rf_y_pred)

# Print results
print(f"\nBest Parameters (Random Forest):
```

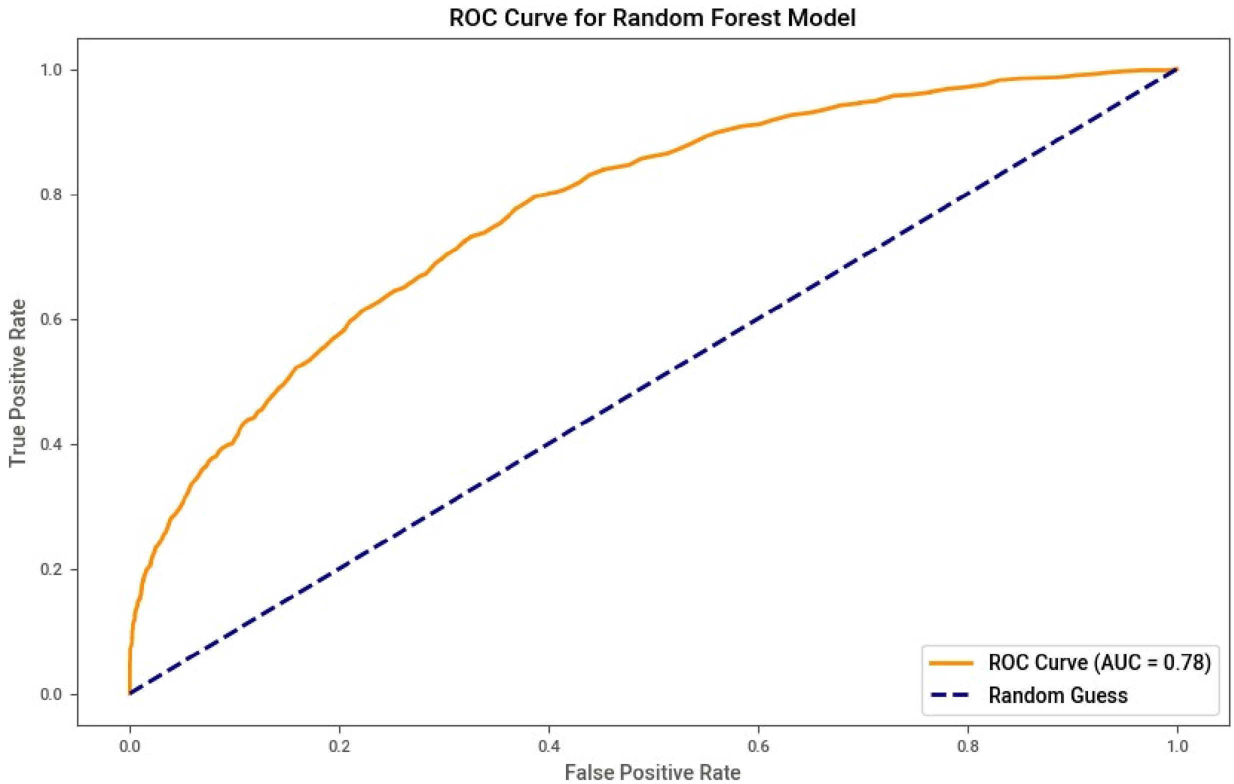
```
{rf_grid_search.best_params_}")
print(f"Accuracy (Random Forest): {rf_accuracy:.2f}")
print(f"Training Time: {elapsed_train_time:.4f} seconds")
print(f"Prediction Time: {elapsed_pred_time:.4f} seconds")
print("\nClassification Report:")
print(rf_classification_report)

# Generate the confusion matrix for the Random Forest model
rf_conf_matrix = confusion_matrix(y_test, rf_y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=rf_conf_matrix,
display_labels=rf_best_model.classes_)
disp.plot(cmap="viridis", values_format="d")
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

```
Best Parameters (Random Forest): {'max_depth': None, 'max_features':
'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
200}
Accuracy (Random Forest): 0.79
Training Time: 789.2710 seconds
Prediction Time: 0.0090 seconds

Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.94      0.87      3396
         1.0       0.65      0.32      0.43      1112

    accuracy                           0.79      4508
   macro avg       0.73      0.63      0.65      4508
weighted avg       0.77      0.79      0.76      4508
```




```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Generate probability scores for the positive class
rf_y_prob = rf_best_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, rf_y_prob)
rf_roc_auc = auc(rf_fpr, rf_tpr)

# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(rf_fpr, rf_tpr, color='darkorange', lw=2, label=f'ROC Curve
(AUC = {rf_roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Random Forest Model')
plt.legend(loc="lower right")
plt.show()
```

```
# Print AUC value for reference
print(f"Area Under the Curve (AUC): {rf_roc_auc:.4f}")
```




```
Area Under the Curve (AUC): 0.7772
```

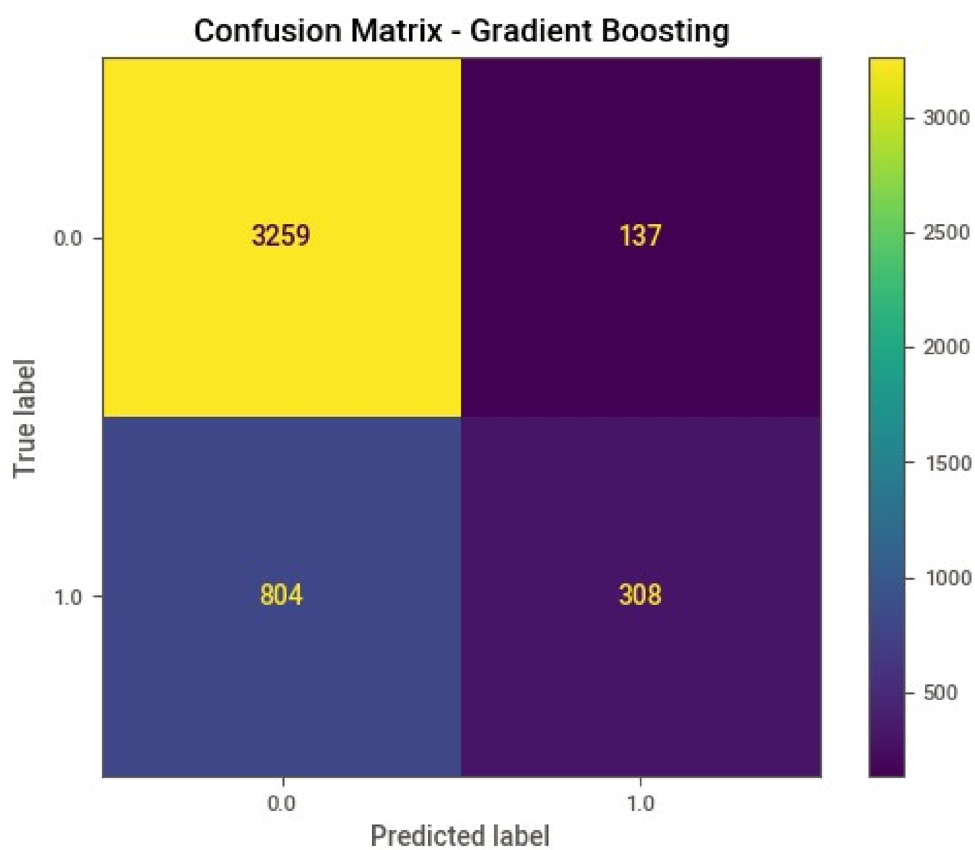
Gradient boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, ConfusionMatrixDisplay
import pandas as pd
import time
import matplotlib.pyplot as plt



# Initialize Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)

# Hyperparameter grid for Gradient Boosting
gb_param_grid = {
    'n_estimators': [50, 100, 200],
```

```
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Grid Search for Gradient Boosting
gb_grid_search = GridSearchCV(estimator=gb_model,
param_grid=gb_param_grid, cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)

# Measure training time
start_train_time = time.time()
gb_grid_search.fit(X_train, y_train)
elapsed_train_time = time.time() - start_train_time

# Retrieve the best model and its parameters
gb_best_model = gb_grid_search.best_estimator_

# Measure prediction time
start_pred_time = time.time()
gb_y_pred = gb_best_model.predict(X_test)
elapsed_pred_time = time.time() - start_pred_time

# Evaluate Gradient Boosting
gb_accuracy = accuracy_score(y_test, gb_y_pred)
gb_classification_report = classification_report(y_test, gb_y_pred)

# Generate the confusion matrix
gb_conf_matrix = confusion_matrix(y_test, gb_y_pred)

# Print results
print(f"\nBest Parameters (Gradient Boosting):
{gb_grid_search.best_params_}")
print(f"Accuracy (Gradient Boosting): {gb_accuracy:.2f}")
print(f"Training Time: {elapsed_train_time:.4f} seconds")
print(f"Prediction Time: {elapsed_pred_time:.4f} seconds")
print("\nClassification Report:")
print(gb_classification_report)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=gb_conf_matrix,
display_labels=gb_best_model.classes_)
disp.plot(cmap="viridis", values_format="d")
plt.title("Confusion Matrix - Gradient Boosting")
plt.show()
```

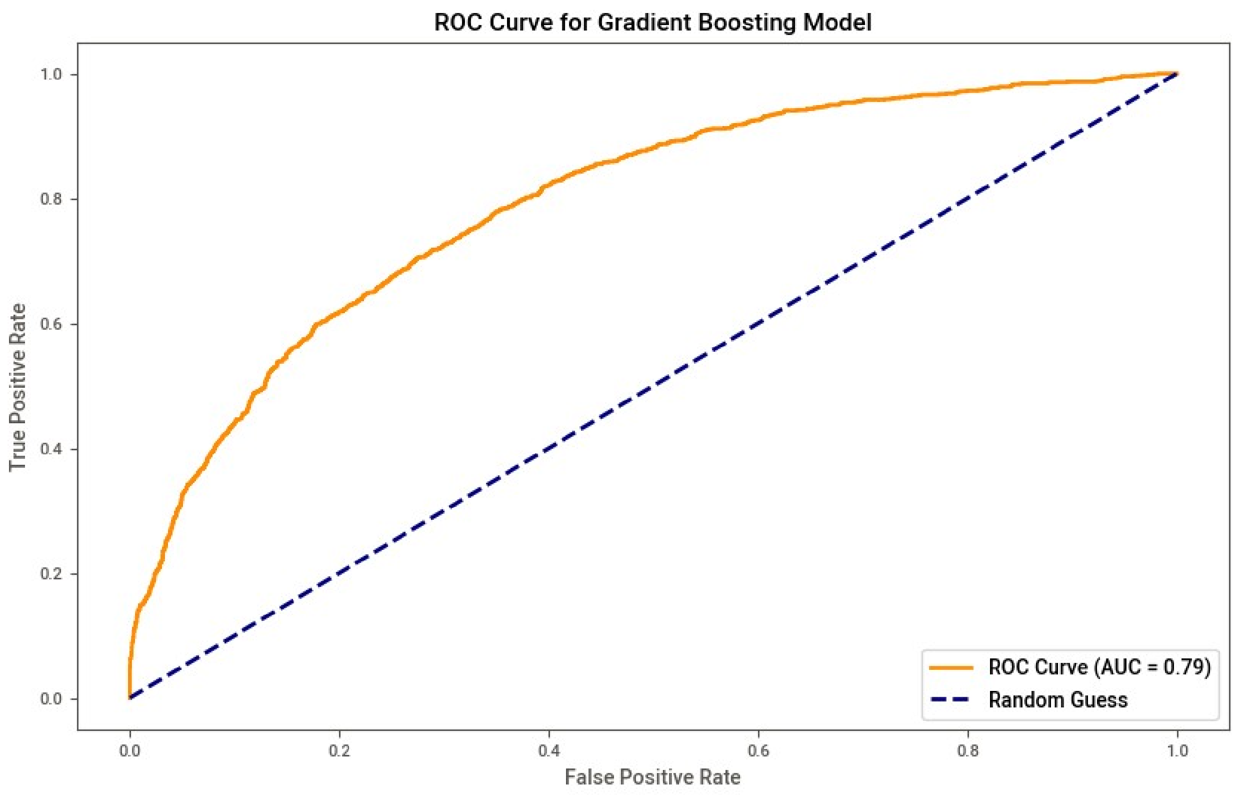
```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits

Best Parameters (Gradient Boosting): {'learning_rate': 0.1,
```

```
'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 2,
'n_estimators': 200}
Accuracy (Gradient Boosting): 0.79
Training Time: 789.2710 seconds
Prediction Time: 0.0090 seconds

Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.96      0.87      3396
         1.0       0.69      0.28      0.40      1112

    accuracy                           0.79      4508
   macro avg       0.75      0.62      0.63      4508
weighted avg       0.77      0.79      0.76      4508
```




```
from sklearn.metrics import roc_curve, auc

# Generate probability scores for the positive class
gb_y_prob = gb_best_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
```

```python
gb_fpr, gb_tpr, gb_thresholds = roc_curve(y_test, gb_y_prob)
gb_roc_auc = auc(gb_fpr, gb_tpr)

# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(gb_fpr, gb_tpr, color='darkorange', lw=2, label=f'ROC Curve
(AUC = {gb_roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Gradient Boosting Model')
plt.legend(loc="lower right")
plt.show()

# Print AUC value for reference
print(f"Area Under the Curve (AUC): {gb_roc_auc:.4f}")
```




```
Area Under the Curve (AUC): 0.7920
```