# AI Assignment 4

**Vikranth Udandarao**
**2022570**

## Theory

Vikranth Udandarao

2022570 × AI Assignment 4

Theory

1- Total records = 10
Approve = Reject = 5

$P(A) = P(R) = \frac{5}{10} = 0.5$

$\text{Entropy} \overset{(S)}{=} - \sum p(u) \log_2 p(u)$

$S(\text{class}) = -P(A)\log_2 P(A) - P(R)\log_2 P(R)$
$= -0.5 \log_2 0.5 \times 2$
$= 1$

Entropy for each feature -

1- Long term debt. $\rightarrow$ 5 Yes
$\searrow$ 5 No

Yes $\rightarrow$ 1 A
$\searrow$ 4 R

$S = -\frac{1}{5}\log\frac{1}{5} - \frac{4}{5}\log\frac{4}{5}$

$= 0.7219$

No $\rightarrow$ 4 A
$\searrow$ 1 R

weighted Entropy (Long Term debt) $= \frac{5}{10} \times 0.7219$

$+ \frac{5}{10} \times 0.7219$

$= 0.7219$

∴ Information Gain (long term debt)

$= 1 - 0.7219$

$= 0.2781$

Unemployed $\nearrow$ 8 no
$\searrow$ 2 yes

no $\nearrow$ 5A
$\searrow$ 3 R

$S = -\frac{5}{8} \log_2 \left(\frac{5}{8}\right) - \frac{3}{8} \log_2 \left(\frac{3}{8}\right)$

$= 0.954$

Yes $\rightarrow$ 0A
$\searrow$ 2R

$S = 0$

weighted Entropy (Unemployed) $= \frac{8}{10} \times 0.954 + 0.$

$= 0.7632$

IG (Unemployment) = 1 - 0.7632
= 0.2368

3- Credit Rating $\rightarrow$ 7 bad
$\rightarrow$ 3 good

bad $\rightarrow$ 3A
$\rightarrow$ 4R

$$S = -\frac{3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right)$$
$$= 0.9852$$

good $\rightarrow$ 2A
$\rightarrow$ 1R

$$S = -\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.9183$$

$$\text{Weighted Entropy}-(CR) = \frac{7}{10} \times 0.9852 + \frac{3}{10} \times 0.9183$$
$$= 0.965$$

$$IG(CR) = 0.034$$

4- Downpayment $\rightarrow$ No = 5
$\rightarrow$ Yes = 5

No $\rightarrow$ 3A
$\rightarrow$ 2R

$$S = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$$

Yes $\begin{matrix} \nearrow 2A \\ \searrow 3R \end{matrix}$ $\rightarrow S = 0.971$

Weighted Entropy ( Downpayment < 20% )

$$= \frac{5}{10} \times 0.971 + \frac{5}{10} \times 0.971$$

$$= 0.971$$

IG (Down payment) $= 1 - 0.971$

$= 0.029$

$\therefore$ We can say, Root $\rightarrow$ long term debt

We now split data based on long term debt

$$\begin{tikzcd} & LTD \arrow[dl, "\text{yes } 5"'] \arrow[dr, "\text{No}"] & \\ \text{reject } (4R, 1A) & & \text{approve } (4A, 1R) \end{tikzcd}$$

For 2nd split -

1. long term debt = Ye

$$S = -\frac{1}{5}\log\frac{1}{5} - \frac{4}{5}\log\frac{4}{5} \approx 0.722$$

Remaining features -

1- Unemployed
2- Credit Rating
3- Down payment < 20%

1- unemployed -

$$H = -\left[\frac{4}{5}\left(-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}\right) + \frac{1}{5}\left(-1\log_2 1 - 0\log_2 0\right)\right.$$

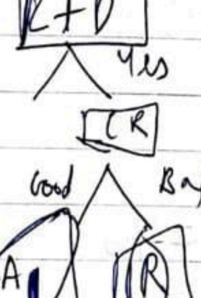
$$= 0.649$$

2- Credit Rating -

$$H = \frac{4}{5}(0) + \frac{1}{5}(0)$$

$= 0$ → Nothing can be lower than 0,

we take this as best split



2- Long term debt - No

4 A } T = 5
1 R }

~~H~~ ≈ 0.722 (previous).

Remaining factors -

1- Unemployment

2- Credit Rating

3- Down Payment < 20% -

1- Unemployment -

$$H = -\left[0 + 1\left[-\frac{1}{5}\log\frac{1}{5} - \frac{4}{5}\log\frac{4}{5}\right]\right] -$$

$$\approx 0.722$$

2- Credit rating -

$$H = -\left[\frac{2}{5}\left(-\log 0.0 - 1\log 1.1\right)\right.$$

$$\left. + \frac{3}{5}\left(-\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3}\right)\right] -$$

3 - Down payment $< 20\%$

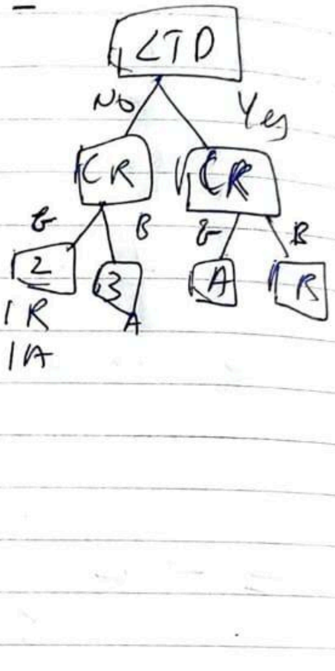
$$H = -\left[\frac{2}{5}\left(-\log 0 \cdot 0 - \log 1 + 1\right) + \frac{3}{5}\left(-\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3}\right)\right]$$

$= 0.55$

Since features ② & ③ have same I G (0.17

we can take any one of them.

I am taking Credit Rating.



For 3rd split,

1- Credit Rating = Bad.

3A.

$S = 0 \rightarrow$ No need to split

2- Credit Rating = Good.

1 A

1 R

$$S = -\left[\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right] = 1$$

For re[illegible] features -

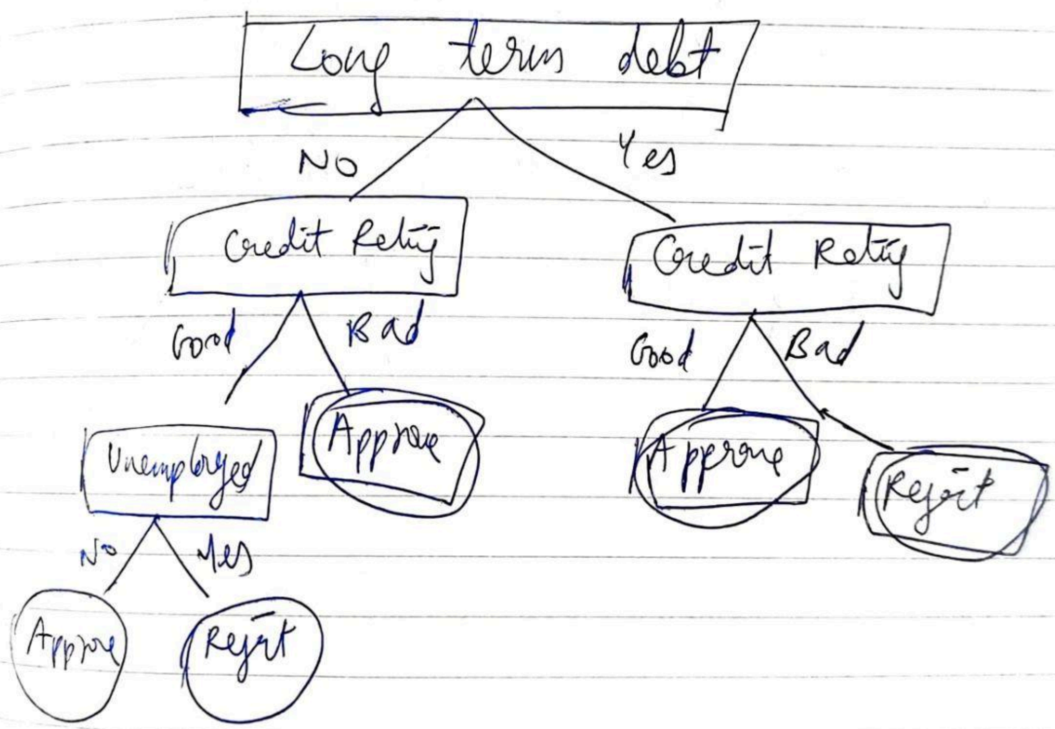
1- Unemployed -

$$H = -\left[\frac{1}{2}\left(-\log(1)\times 1) - \log 0 \times 0\right) + \frac{1}{2}\left(-\log 1 - 0\right)\right]$$

$= 0$

Since $S = 0$, we take this split.

Tree Construction



Traing error ~~f~~ for all points

|  |  |  |  |
|---|---|---|---|
| Point 1 | - | Correctly Classified | Approve |
| Point 2 | - | // | // |
| Point 3 | - | // | // |
| Point 4 | - | // | // |
| Point 5 | - | // | // |
| Point 6 | - | // | Reject. |
| Point 7 | - | // | // |
| Point 8 | - | // | // |
| Point 9 | - | // | // |
| Point 10 | - | // | // |

Since there is no max depth mentioned in the question, ~~[illegible]~~ I could just overfit the decision tree on the traing data

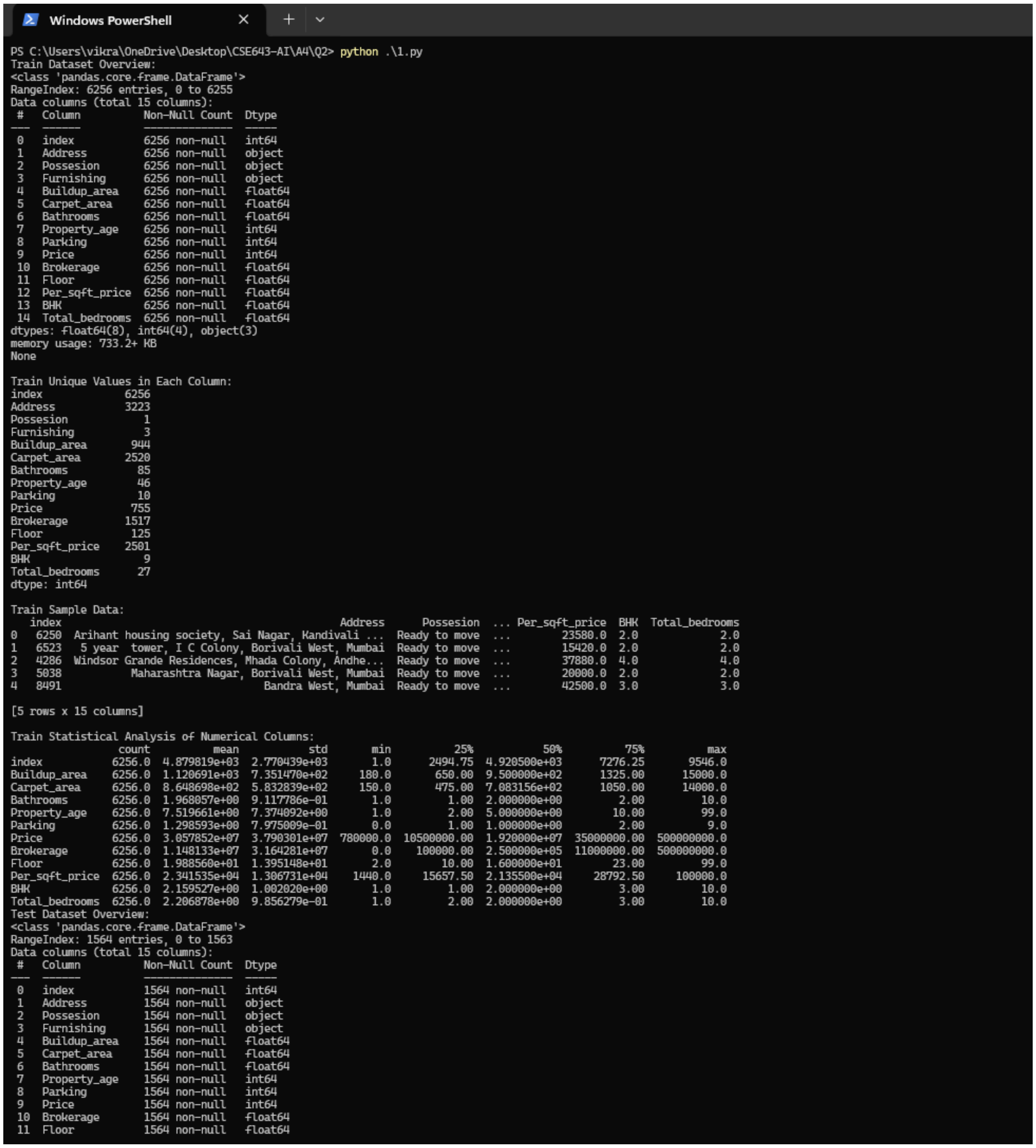
Here 10/10 are correctly classified.

$\boxed{\text{Traing error} = 0}$ .

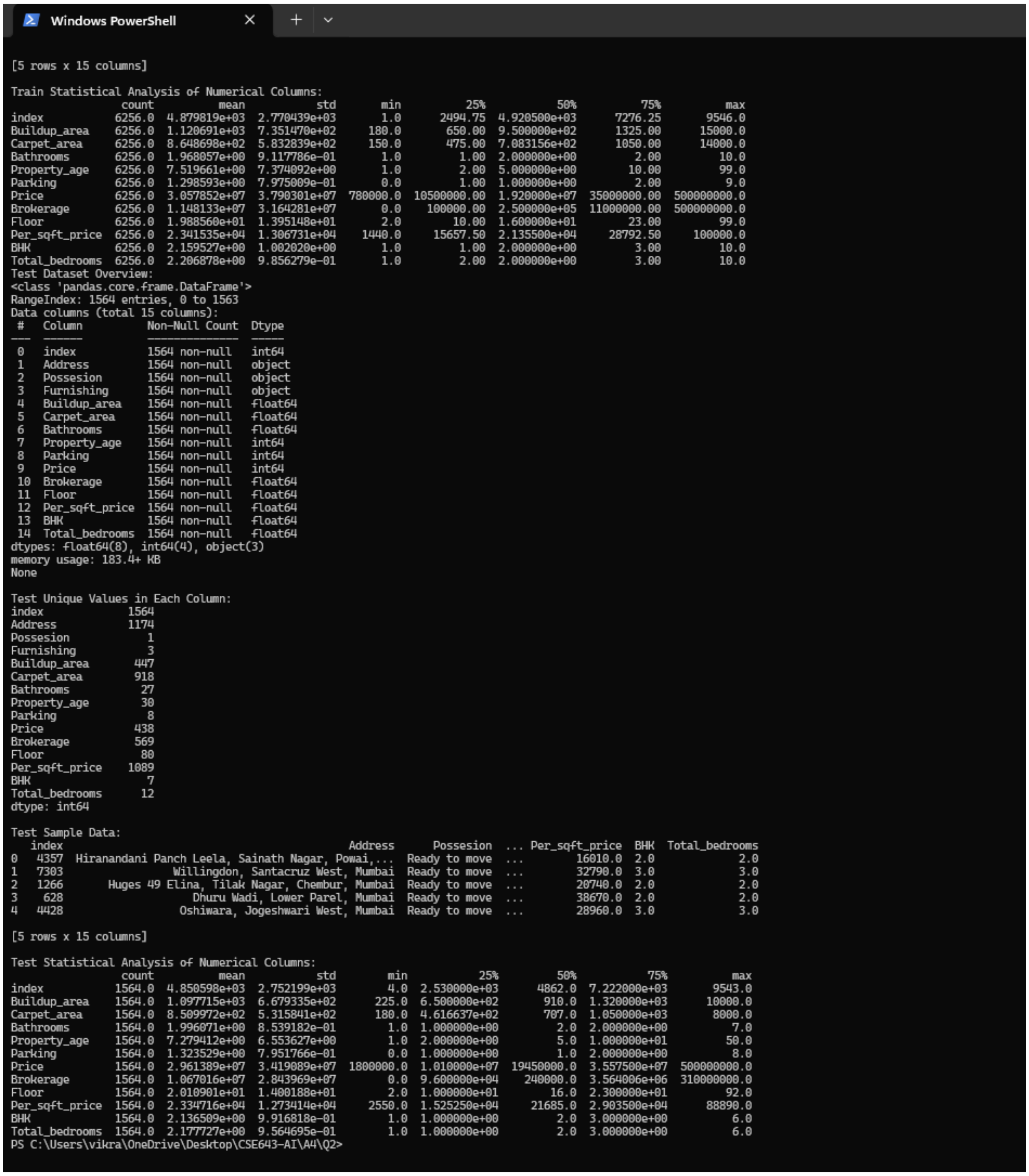
# Coding

2.

## Task 1:



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> python .\1.py
Train Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6256 entries, 0 to 6255
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   index           6256 non-null   int64
 1   Address         6256 non-null   object
 2   Possesion       6256 non-null   object
 3   Furnishing      6256 non-null   object
 4   Buildup_area    6256 non-null   float64
 5   Carpet_area     6256 non-null   float64
 6   Bathrooms       6256 non-null   float64
 7   Property_age    6256 non-null   int64
 8   Parking         6256 non-null   int64
 9   Price           6256 non-null   int64
 10  Brokerage       6256 non-null   float64
 11  Floor           6256 non-null   float64
 12  Per_sqft_price  6256 non-null   float64
 13  BHK             6256 non-null   float64
 14  Total_bedrooms  6256 non-null   float64
dtypes: float64(8), int64(4), object(3)
memory usage: 733.2+ KB
None

Train Unique Values in Each Column:
index             6256
Address           3223
Possesion            1
Furnishing           3
Buildup_area       944
Carpet_area       2520
Bathrooms           85
Property_age        46
Parking             10
Price              755
Brokerage         1517
Floor              125
Per_sqft_price    2501
BHK                  9
Total_bedrooms      27
dtype: int64

Train Sample Data:
   index                                        Address     Possesion  ... Per_sqft_price  BHK  Total_bedrooms
0   6250  Arihant housing society, Sai Nagar, Kandivali ...  Ready to move  ...        23580.0  2.0             2.0
1   6523   5 year  tower, I C Colony, Borivali West, Mumbai  Ready to move  ...        15420.0  2.0             2.0
2   4286  Windsor Grande Residences, Mhada Colony, Andhe...  Ready to move  ...        37880.0  4.0             4.0
3   5038           Maharashtra Nagar, Borivali West, Mumbai  Ready to move  ...        20000.0  2.0             2.0
4   8491                             Bandra West, Mumbai  Ready to move  ...        42500.0  3.0             3.0

[5 rows x 15 columns]

Train Statistical Analysis of Numerical Columns:
                 count          mean           std       min          25%           50%          75%          max
index           6256.0  4.879819e+03  2.770439e+03       1.0      2494.75  4.920500e+03      7276.25       9546.0
Buildup_area    6256.0  1.120691e+03  7.351470e+02     180.0       650.00  9.500000e+02      1325.00      15000.0
Carpet_area     6256.0  8.648698e+02  5.832839e+02     150.0       475.00  7.083156e+02      1050.00      14000.0
Bathrooms       6256.0  1.968057e+00  9.117786e-01       1.0         1.00  2.000000e+00         2.00         10.0
Property_age    6256.0  7.519661e+00  7.374092e+00       1.0         2.00  5.000000e+00        10.00         99.0
Parking         6256.0  1.298593e+00  7.975009e-01       0.0         1.00  1.000000e+00         2.00          9.0
Price           6256.0  3.057852e+07  3.790301e+07  780000.0  10500000.00  1.920000e+07  35000000.00  500000000.0
Brokerage       6256.0  1.148133e+07  3.164281e+07       0.0    100000.00  2.500000e+05  11000000.00  500000000.0
Floor           6256.0  1.988560e+01  1.395148e+01       2.0        10.00  1.600000e+01        23.00         99.0
Per_sqft_price  6256.0  2.341535e+04  1.306731e+04    1440.0     15657.50  2.135500e+04     28792.50     100000.0
BHK             6256.0  2.159527e+00  1.002020e+00       1.0         1.00  2.000000e+00         3.00         10.0
Total_bedrooms  6256.0  2.206878e+00  9.856279e-01       1.0         2.00  2.000000e+00         3.00         10.0
Test Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1564 entries, 0 to 1563
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   index           1564 non-null   int64
 1   Address         1564 non-null   object
 2   Possesion       1564 non-null   object
 3   Furnishing      1564 non-null   object
 4   Buildup_area    1564 non-null   float64
 5   Carpet_area     1564 non-null   float64
 6   Bathrooms       1564 non-null   float64
 7   Property_age    1564 non-null   int64
 8   Parking         1564 non-null   int64
 9   Price           1564 non-null   int64
 10  Brokerage       1564 non-null   float64
 11  Floor           1564 non-null   float64
```



```
[5 rows x 15 columns]

Train Statistical Analysis of Numerical Columns:
                count          mean           std       min          25%           50%          75%          max
index          6256.0  4.879819e+03  2.770439e+03       1.0      2494.75  4.920500e+03      7276.25       9546.0
Buildup_area   6256.0  1.120691e+03  7.351470e+02     180.0       650.00  9.500000e+02      1325.00      15000.0
Carpet_area    6256.0  8.648698e+02  5.832839e+02     150.0       475.00  7.083156e+02      1050.00      14000.0
Bathrooms      6256.0  1.968057e+00  9.117786e-01       1.0         1.00  2.000000e+00         2.00         10.0
Property_age   6256.0  7.519661e+00  7.374092e+00       1.0         2.00  5.000000e+00        10.00         99.0
Parking        6256.0  1.298593e+00  7.975009e-01       0.0         1.00  1.000000e+00         2.00          9.0
Price          6256.0  3.057852e+07  3.790301e+07  780000.0  10500000.00  1.920000e+07  35000000.00  500000000.0
Brokerage      6256.0  1.148133e+07  3.164281e+07       0.0    100000.00  2.500000e+05  11000000.00  500000000.0
Floor          6256.0  1.988560e+01  1.395148e+01       2.0        10.00  1.600000e+01        23.00         99.0
Per_sqft_price 6256.0  2.341535e+04  1.306731e+04    1440.0     15657.50  2.135500e+04     28792.50     100000.0
BHK            6256.0  2.159527e+00  1.002020e+00       1.0         1.00  2.000000e+00         3.00         10.0
Total_bedrooms 6256.0  2.206878e+00  9.856279e-01       1.0         2.00  2.000000e+00         3.00         10.0
Test Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1564 entries, 0 to 1563
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   index           1564 non-null   int64
 1   Address         1564 non-null   object
 2   Possesion       1564 non-null   object
 3   Furnishing      1564 non-null   object
 4   Buildup_area    1564 non-null   float64
 5   Carpet_area     1564 non-null   float64
 6   Bathrooms       1564 non-null   float64
 7   Property_age    1564 non-null   int64
 8   Parking         1564 non-null   int64
 9   Price           1564 non-null   int64
 10  Brokerage       1564 non-null   float64
 11  Floor           1564 non-null   float64
 12  Per_sqft_price  1564 non-null   float64
 13  BHK             1564 non-null   float64
 14  Total_bedrooms  1564 non-null   float64
dtypes: float64(8), int64(4), object(3)
memory usage: 183.4+ KB
None

Test Unique Values in Each Column:
index             1564
Address           1174
Possesion            1
Furnishing           3
Buildup_area       447
Carpet_area        918
Bathrooms           27
Property_age        30
Parking              8
Price              438
Brokerage          569
Floor               80
Per_sqft_price    1089
BHK                  7
Total_bedrooms      12
dtype: int64

Test Sample Data:
   index                                      Address      Possesion  ... Per_sqft_price  BHK  Total_bedrooms
0   4357  Hiranandani Panch Leela, Sainath Nagar, Powai,...  Ready to move  ...        16010.0  2.0             2.0
1   7303              Willingdon, Santacruz West, Mumbai  Ready to move  ...        32790.0  3.0             3.0
2   1266      Huges 49 Elina, Tilak Nagar, Chembur, Mumbai  Ready to move  ...        20740.0  2.0             2.0
3    628                Dhuru Wadi, Lower Parel, Mumbai  Ready to move  ...        38670.0  2.0             2.0
4   4428             Oshiwara, Jogeshwari West, Mumbai  Ready to move  ...        28960.0  3.0             3.0

[5 rows x 15 columns]

Test Statistical Analysis of Numerical Columns:
                count          mean           std        min           25%         50%           75%          max
index          1564.0  4.850598e+03  2.752199e+03        4.0  2.530000e+03      4862.0  7.222000e+03       9543.0
Buildup_area   1564.0  1.097715e+03  6.679335e+02      225.0  6.500000e+02       910.0  1.320000e+03      10000.0
Carpet_area    1564.0  8.509972e+02  5.315841e+02      180.0  4.616637e+02       707.0  1.050000e+03       8000.0
Bathrooms      1564.0  1.996071e+00  8.539182e-01        1.0  1.000000e+00         2.0  2.000000e+00          7.0
Property_age   1564.0  7.279412e+00  6.553627e+00        1.0  2.000000e+00         5.0  1.000000e+01         50.0
Parking        1564.0  1.323529e+00  7.951766e-01        0.0  1.000000e+00         1.0  2.000000e+00          8.0
Price          1564.0  2.961389e+07  3.419089e+07  1800000.0  1.010000e+07  19450000.0  3.557500e+07  500000000.0
Brokerage      1564.0  1.067016e+07  2.843969e+07        0.0  9.600000e+04    240000.0  3.564006e+06  310000000.0
Floor          1564.0  2.010901e+01  1.400188e+01        2.0  1.000000e+01        16.0  2.300000e+01         92.0
Per_sqft_price 1564.0  2.334716e+04  1.273414e+04     2550.0  1.525250e+04     21685.0  2.903500e+04      88890.0
BHK            1564.0  2.136509e+00  9.916818e-01        1.0  1.000000e+00         2.0  3.000000e+00          6.0
Total_bedrooms 1564.0  2.177727e+00  9.564695e-01        1.0  1.000000e+00         2.0  3.000000e+00          6.0
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2>
```

Train Numerical Columns Statistics:

```
,count,mean,std,min,25%,50%,75%,max
index,6256.0,4879.818893861892,2770.43933334215,1.0,2494.75,4920.5,7
276.25,9546.0
```

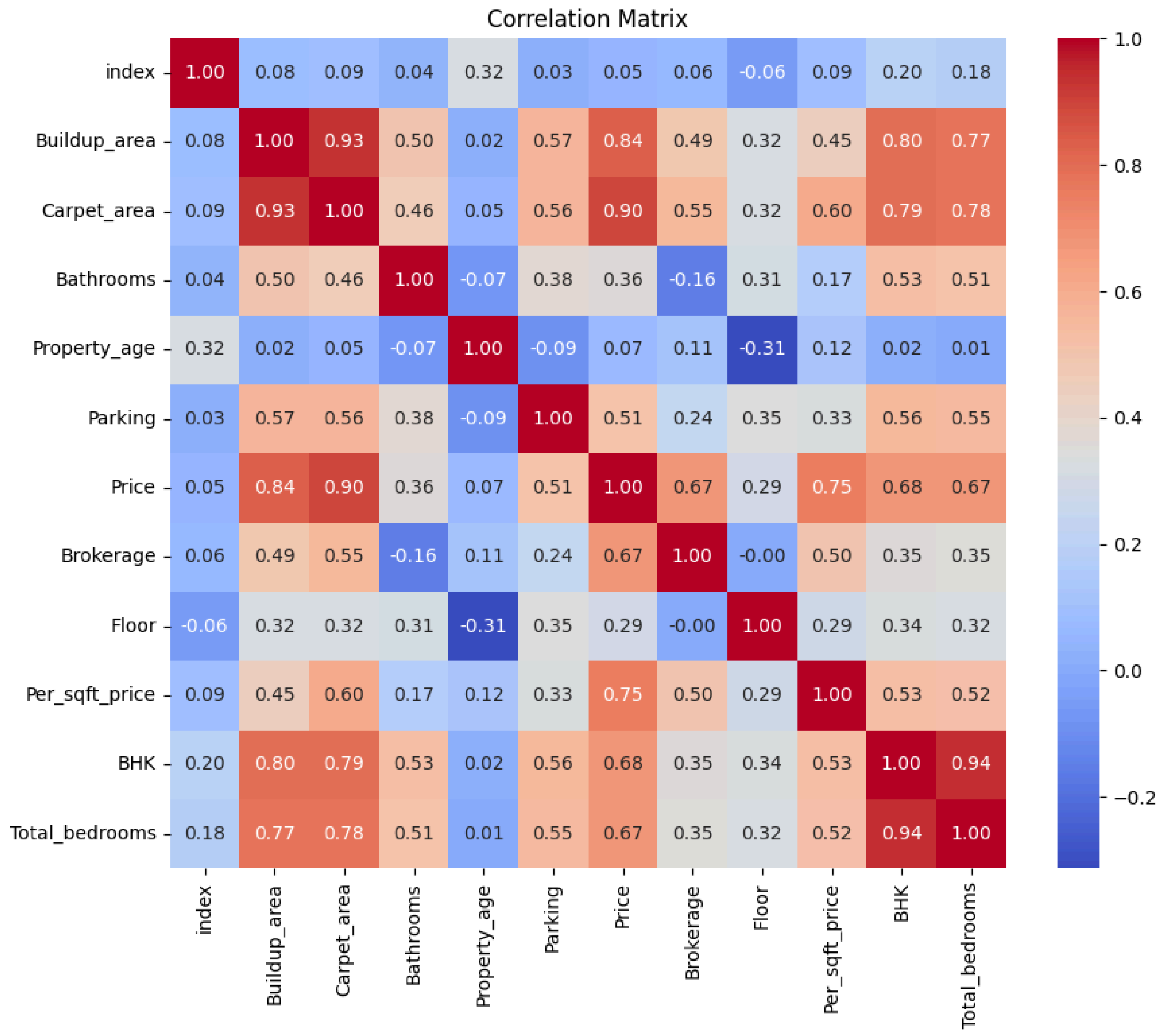
```
Buildup_area,6256.0,1120.690537084399,735.1470375967365,180.0,650.0,950.0,1325.0,15000.0
Carpet_area,6256.0,864.8698014775575,583.2839178380216,150.0,475.0,708.31558255,1050.0,14000.0
Bathrooms,6256.0,1.9680573820030371,0.9117785996995053,0.999999964,1.0,2.0,2.0,10.0
Property_age,6256.0,7.519661125319693,7.3740918955506976,1.0,2.0,5.0,10.0,99.0
Parking,6256.0,1.2985933503836318,0.79750085476234,0.0,1.0,1.0,2.0,9.0
Price,6256.0,30578515.505115088,37903010.081599094,780000.0,10500000.0,19200000.0,35000000.0,500000000.0
Brokerage,6256.0,11481327.208449103,31642814.65150307,0.0,100000.0,250000.0,11000000.0,500000000.0
Floor,6256.0,19.885595252484336,13.951480267711663,2.0,10.0,16.0,23.0,99.0
Per_sqft_price,6256.0,23415.351551259588,13067.30858046999,1440.0,15657.5,21355.0,28792.5,100000.0
BHK,6256.0,2.159526854219949,1.0020197189199798,1.0,1.0,2.0,3.0,10.0
Total_bedrooms,6256.0,2.206877851878996,0.9856278695926552,1.0,2.0,2.0,3.0,10.0
```

Test Numerical Columns Statistics:

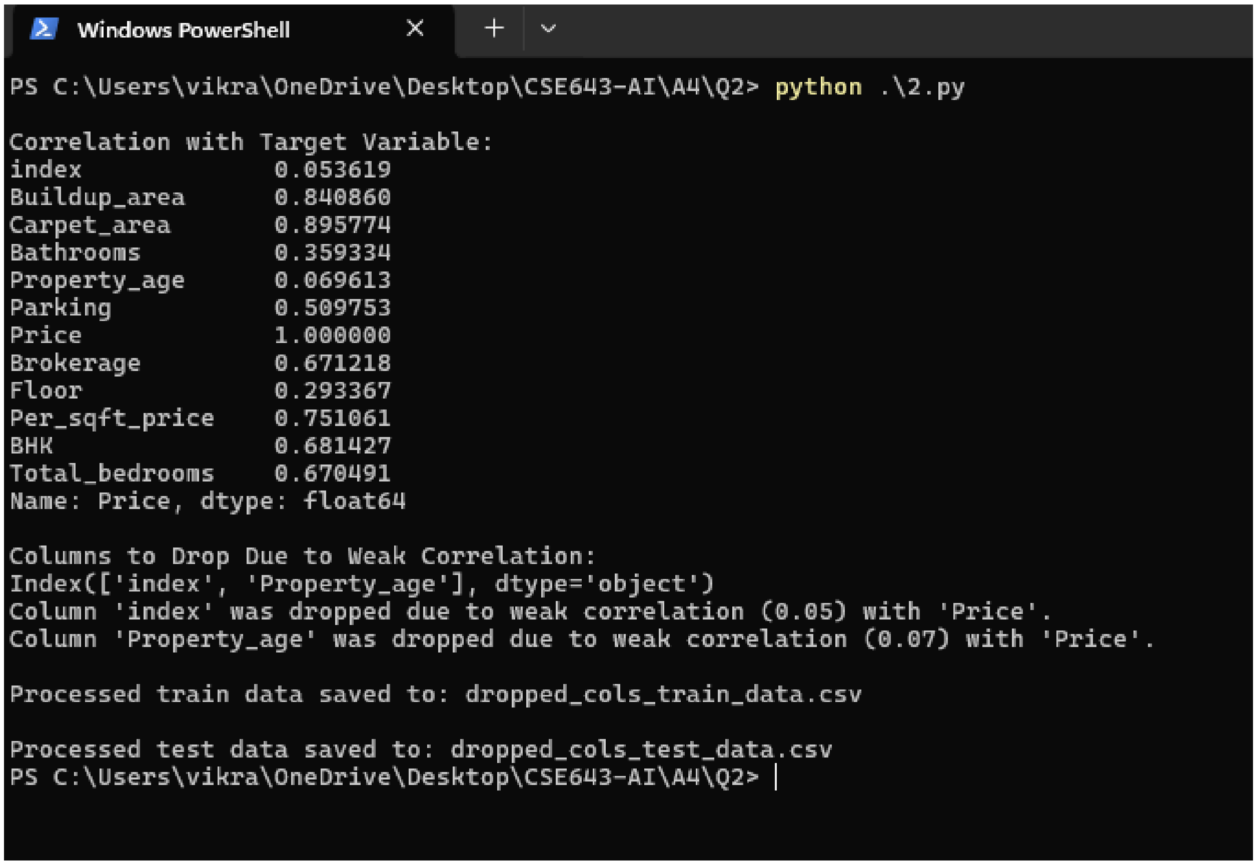
```
,count,mean,std,min,25%,50%,75%,max
index,1564.0,4850.597826086957,2752.1988963764957,4.0,2530.0,4862.0,7222.0,9543.0
Buildup_area,1564.0,1097.7153781317136,667.933500183208,225.0,650.0,910.0,1320.0,10000.0
Carpet_area,1564.0,850.9972019076727,531.5840509327888,180.0,461.6637147,707.0,1050.0,8000.0
Bathrooms,1564.0,1.9960705783893864,0.8539181827361639,0.999999948,1.0,2.0,2.0,7.0
Property_age,1564.0,7.279411764705882,6.553627248174288,1.0,2.0,5.0,10.0,50.0
Parking,1564.0,1.3235294117647058,0.7951766454109096,0.0,1.0,1.0,2.0,8.0
Price,1564.0,29613888.746803068,34190887.99612117,1800000.0,10100000.0,19450000.0,35575000.0,500000000.0
```

```
Brokerage,1564.0,10670160.705711264,28439693.63912475,0.0,96000.0,240000.0,3564005.8025,310000000.0
Floor,1564.0,20.109013351560105,14.001882797074007,2.0,10.0,16.0,23.0,92.0
Per_sqft_price,1564.0,23347.163656592074,12734.140363671408,2550.0,15252.5,21685.0,29035.0,88890.0
BHK,1564.0,2.1365089514066495,0.991681762668803,1.0,1.0,2.0,3.0,6.0
Total_bedrooms,1564.0,2.177726665986573,0.9564695349191122,1.0,1.0,2.0,3.0,6.0
```

## Task 2.



| | index | Buildup_area | Carpet_area | Bathrooms | Property_age | Parking | Price | Brokerage | Floor | Per_sqft_price | BHK | Total_bedrooms |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| index | 1.00 | 0.08 | 0.09 | 0.04 | 0.32 | 0.03 | 0.05 | 0.06 | -0.06 | 0.09 | 0.20 | 0.18 |
| Buildup_area | 0.08 | 1.00 | 0.93 | 0.50 | 0.02 | 0.57 | 0.84 | 0.49 | 0.32 | 0.45 | 0.80 | 0.77 |
| Carpet_area | 0.09 | 0.93 | 1.00 | 0.46 | 0.05 | 0.56 | 0.90 | 0.55 | 0.32 | 0.60 | 0.79 | 0.78 |
| Bathrooms | 0.04 | 0.50 | 0.46 | 1.00 | -0.07 | 0.38 | 0.36 | -0.16 | 0.31 | 0.17 | 0.53 | 0.51 |
| Property_age | 0.32 | 0.02 | 0.05 | -0.07 | 1.00 | -0.09 | 0.07 | 0.11 | -0.31 | 0.12 | 0.02 | 0.01 |
| Parking | 0.03 | 0.57 | 0.56 | 0.38 | -0.09 | 1.00 | 0.51 | 0.24 | 0.35 | 0.33 | 0.56 | 0.55 |
| Price | 0.05 | 0.84 | 0.90 | 0.36 | 0.07 | 0.51 | 1.00 | 0.67 | 0.29 | 0.75 | 0.68 | 0.67 |
| Brokerage | 0.06 | 0.49 | 0.55 | -0.16 | 0.11 | 0.24 | 0.67 | 1.00 | -0.00 | 0.50 | 0.35 | 0.35 |
| Floor | -0.06 | 0.32 | 0.32 | 0.31 | -0.31 | 0.35 | 0.29 | -0.00 | 1.00 | 0.29 | 0.34 | 0.32 |
| Per_sqft_price | 0.09 | 0.45 | 0.60 | 0.17 | 0.12 | 0.33 | 0.75 | 0.50 | 0.29 | 1.00 | 0.53 | 0.52 |
| BHK | 0.20 | 0.80 | 0.79 | 0.53 | 0.02 | 0.56 | 0.68 | 0.35 | 0.34 | 0.53 | 1.00 | 0.94 |
| Total_bedrooms | 0.18 | 0.77 | 0.78 | 0.51 | 0.01 | 0.55 | 0.67 | 0.35 | 0.32 | 0.52 | 0.94 | 1.00 |



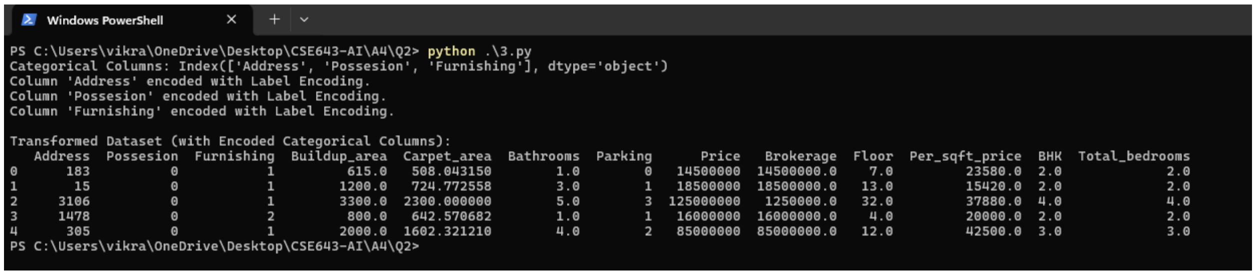
```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> python .\2.py

Correlation with Target Variable:
index              0.053619
Buildup_area       0.840860
Carpet_area        0.895774
Bathrooms          0.359334
Property_age       0.069613
Parking            0.509753
Price              1.000000
Brokerage          0.671218
Floor              0.293367
Per_sqft_price     0.751061
BHK                0.681427
Total_bedrooms     0.670491
Name: Price, dtype: float64

Columns to Drop Due to Weak Correlation:
Index(['index', 'Property_age'], dtype='object')
Column 'index' was dropped due to weak correlation (0.05) with 'Price'.
Column 'Property_age' was dropped due to weak correlation (0.07) with 'Price'.

Processed train data saved to: dropped_cols_train_data.csv

Processed test data saved to: dropped_cols_test_data.csv
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> |
```

‘Index’ and ‘Property Age’ were dropped as they do not lie within the correlation coefficient range of -1 to 1

After analysing the training dataset, I noticed that ‘Possesion’ only had ‘Ready to move’ in all the rows, so it will not be of any use to predict the target variable - Price.
This was supported by the fact that in Task 3, after label encoding ‘Possesion’ only had 0 in the entire dataset.
So I dropped ‘Possesion’ as well.

Rest of the features look meaningful to me in buying a house context, so it makes sense to keep the rest of the features.

## Task 3:



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> python .\3.py
Categorical Columns: Index(['Address', 'Possesion', 'Furnishing'], dtype='object')
Column 'Address' encoded with Label Encoding.
Column 'Possesion' encoded with Label Encoding.
Column 'Furnishing' encoded with Label Encoding.

Transformed Dataset (with Encoded Categorical Columns):
   Address  Possesion  Furnishing  Buildup_area  Carpet_area  Bathrooms  Parking      Price   Brokerage  Floor  Per_sqft_price  BHK  Total_bedrooms
0      183          0           1         615.0   508.043150        1.0        0   14500000  14500000.0    7.0         23580.0  2.0             2.0
1       15          0           1        1200.0   724.772558        3.0        1   18500000  18500000.0   13.0         15420.0  2.0             2.0
2     3106          0           1        3300.0  2300.000000        5.0        3  125000000   1250000.0   32.0         37880.0  4.0             4.0
3     1478          0           2         800.0   642.570682        1.0        1   16000000  16000000.0    4.0         20000.0  2.0             2.0
4      305          0           1        2000.0  1602.321210        4.0        2   85000000  85000000.0   12.0         42500.0  3.0             3.0
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2>
```

Categorical Columns: Index(['Address', 'Possesion', 'Furnishing'], dtype='object')
Column 'Address' encoded with Label Encoding.
Column 'Possesion' encoded with Label Encoding.
Column 'Furnishing' encoded with Label Encoding.

Here, I realized that 'Possesion' is of no meaningful use to predict the target variable - Price and thus dropped it.

Address is high cardinality as each unique address has a unique numeric value, whereas Furnishing is low cardinality as there are only 3 numerics values - Unfurnished, Semi Furnished and Fully Furnished.

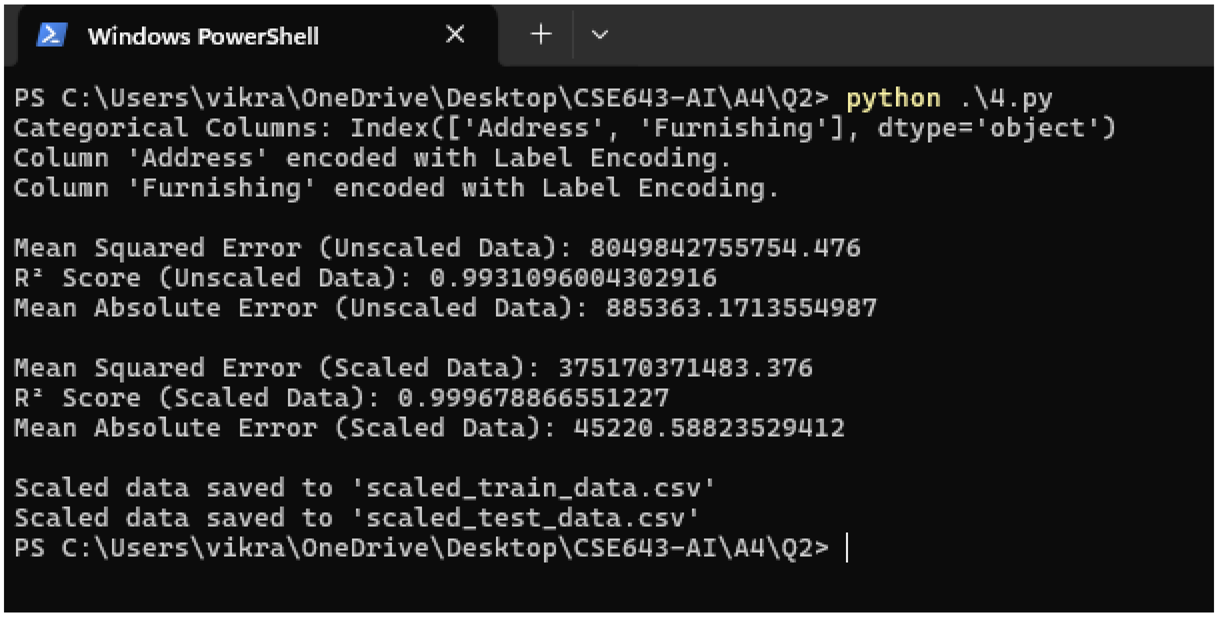
To Address this, I checked some statistics:

Mean of 'Address': 1579.5012787723786
Median of 'Address': 1532.0
Mode of 'Address': 2914

Then I scaled feature encoding but then the threshold of 0.01 (1%) but most of the values were lesser than 1% of total, and when I tried to do feature encoding, I did not get a good accuracy (around 93%), so I decided to not scale Address.

## Task 4:

I have only standard scaled the numerical columns, not including the label encoded categorical features as it was giving more accuracy. (When I standard scaled the label encoded columns, then the accuracy was around 96%)
I have also not Scaled the Target Column - 'Price' as accuracy was lesser when I scaled it,
I decided to do the analysis here itself on training Decision Tree on scaled and unscaled data



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> python .\4.py
Categorical Columns: Index(['Address', 'Furnishing'], dtype='object')
Column 'Address' encoded with Label Encoding.
Column 'Furnishing' encoded with Label Encoding.

Mean Squared Error (Unscaled Data): 8049842755754.476
R² Score (Unscaled Data): 0.9931096004302916
Mean Absolute Error (Unscaled Data): 885363.1713554987

Mean Squared Error (Scaled Data): 375170371483.376
R² Score (Scaled Data): 0.999678866551227
Mean Absolute Error (Scaled Data): 45220.58823529412

Scaled data saved to 'scaled_train_data.csv'
Scaled data saved to 'scaled_test_data.csv'
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> |
```

The model performance on Scaled data was slightly better than the performance on Unscaled data.

Unscaled data: 99,31%
Scaled data: 99.96%

## Task 5:

I have taken:

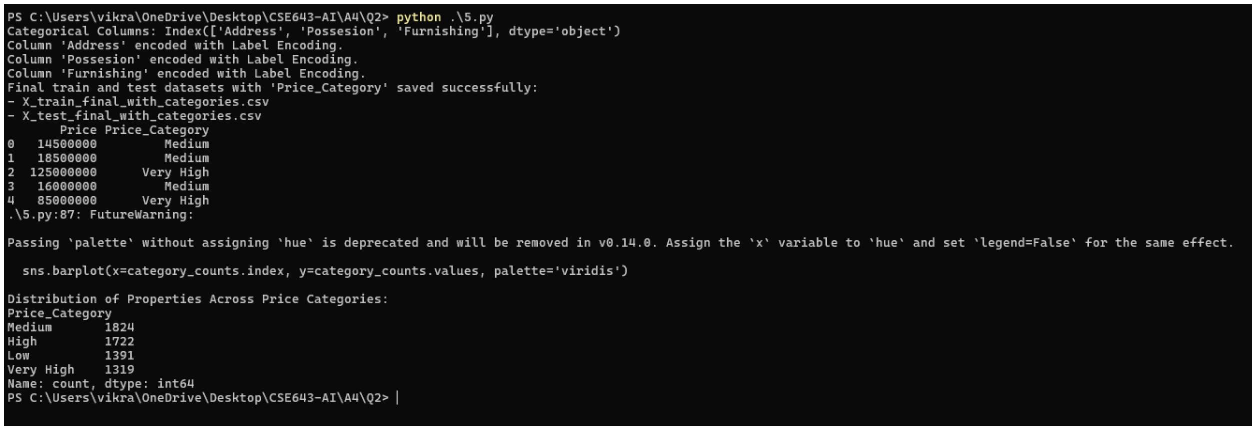
<10,000,000 as Low
10,000,000 - 20,000,000 as Medium
20,000,000 - 40,000,000 as High
>40,000,000 as Very High

There are a lot of properties with prices within 50,000,000 - Around 5,200 properties.
There are around 800 properties between 50,000,000 and 100,000,000.

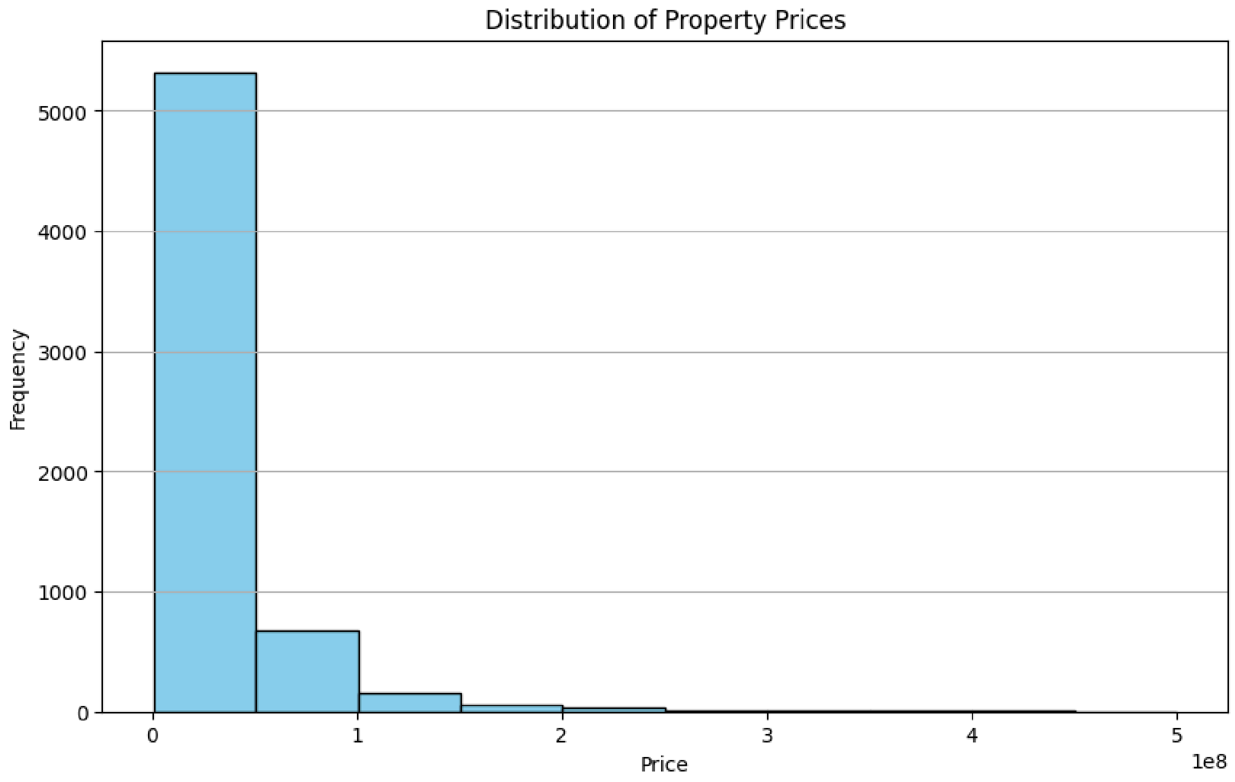
From the classes plot below, Medium has the highest properties, followed by High and Low, and Very High is the least.

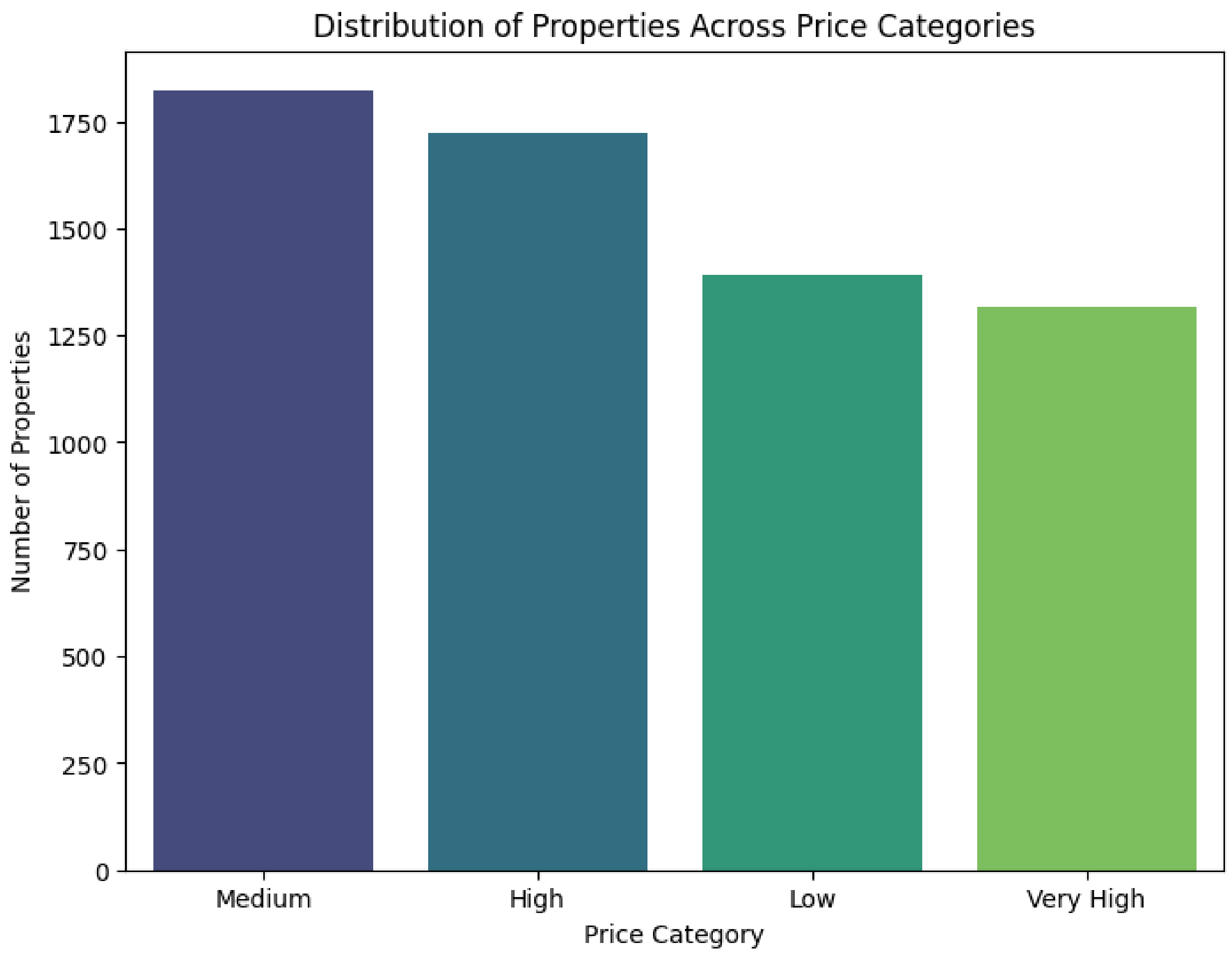
The Very High and Low categories have noticeably fewer properties compared to Medium and High.
This could result in poor model performance for these categories due to insufficient data for training.

The imbalance could lead to bias in predictions, where the model may overpredict the Medium category while underpredicting the Very High and Low categories due to lack of representation.



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> python .\5.py
Categorical Columns: Index(['Address', 'Possesion', 'Furnishing'], dtype='object')
Column 'Address' encoded with Label Encoding.
Column 'Possesion' encoded with Label Encoding.
Column 'Furnishing' encoded with Label Encoding.
Final train and test datasets with 'Price_Category' saved successfully:
- X_train_final_with_categories.csv
- X_test_final_with_categories.csv
       Price Price_Category
0   14500000         Medium
1   18500000         Medium
2  125000000      Very High
3   16000000         Medium
4   85000000      Very High
.\5.py:87: 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=category_counts.index, y=category_counts.values, palette='viridis')

Distribution of Properties Across Price Categories:
Price_Category
Medium       1824
High         1722
Low          1391
Very High    1319
Name: count, dtype: int64
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> |
```





## Task 6:

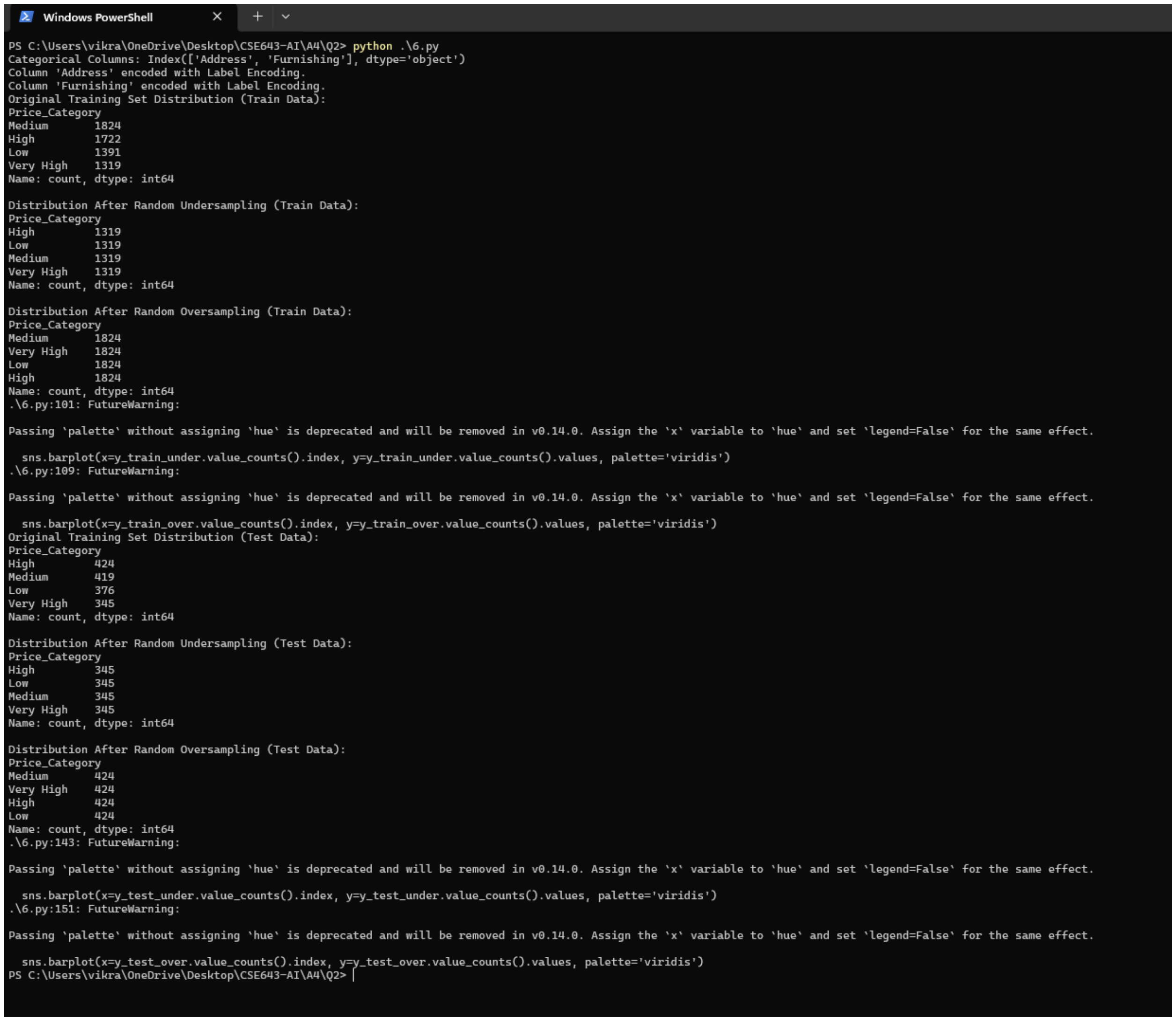
## Benefits and Limitations of the Methods

### 1. Random Undersampling

- **What it does**: Reduces the number of samples in overrepresented classes to match the size of the underrepresented class.
- **Benefits**:
  - Simplifies the dataset, leading to faster model training.
  - Prevents overemphasis on the majority class during training.
  - Useful for cases where resources and time are limited.
- **Limitations**:
  - Results in loss of valuable information, as many samples from the majority class are discarded.
  - May lead to underfitting, as the model is trained on reduced data.

### 2. Random Oversampling

- **What it does**: Replicates samples from underrepresented classes to balance the dataset.
- **Benefits**:
    - Preserves all available data and enriches the underrepresented classes.
    - Helps the model generalize better across all classes.
- **Limitations**:
    - Risks overfitting the model to the replicated samples.
    - Increases the computational cost of training due to the enlarged dataset.



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> python .\6.py
Categorical Columns: Index(['Address', 'Furnishing'], dtype='object')
Column 'Address' encoded with Label Encoding.
Column 'Furnishing' encoded with Label Encoding.
Original Training Set Distribution (Train Data):
Price_Category
Medium       1824
High         1722
Low          1391
Very High    1319
Name: count, dtype: int64

Distribution After Random Undersampling (Train Data):
Price_Category
High         1319
Low          1319
Medium       1319
Very High    1319
Name: count, dtype: int64

Distribution After Random Oversampling (Train Data):
Price_Category
Medium       1824
Very High    1824
Low          1824
High         1824
Name: count, dtype: int64
.\6.py:101: 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=y_train_under.value_counts().index, y=y_train_under.value_counts().values, palette='viridis')
.\6.py:109: 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=y_train_over.value_counts().index, y=y_train_over.value_counts().values, palette='viridis')
Original Training Set Distribution (Test Data):
Price_Category
High         424
Medium       419
Low          376
Very High    345
Name: count, dtype: int64

Distribution After Random Undersampling (Test Data):
Price_Category
High         345
Low          345
Medium       345
Very High    345
Name: count, dtype: int64

Distribution After Random Oversampling (Test Data):
Price_Category
Medium       424
Very High    424
High         424
Low          424
Name: count, dtype: int64
.\6.py:143: 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=y_test_under.value_counts().index, y=y_test_under.value_counts().values, palette='viridis')
.\6.py:151: 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=y_test_over.value_counts().index, y=y_test_over.value_counts().values, palette='viridis')
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q2> |
```

## Observations from the Plots

### 1. Original Data

- The distribution of properties across price categories (`Low`, `Medium`, `High`, and `Very High`) is imbalanced, with the `Medium` category dominating the dataset.

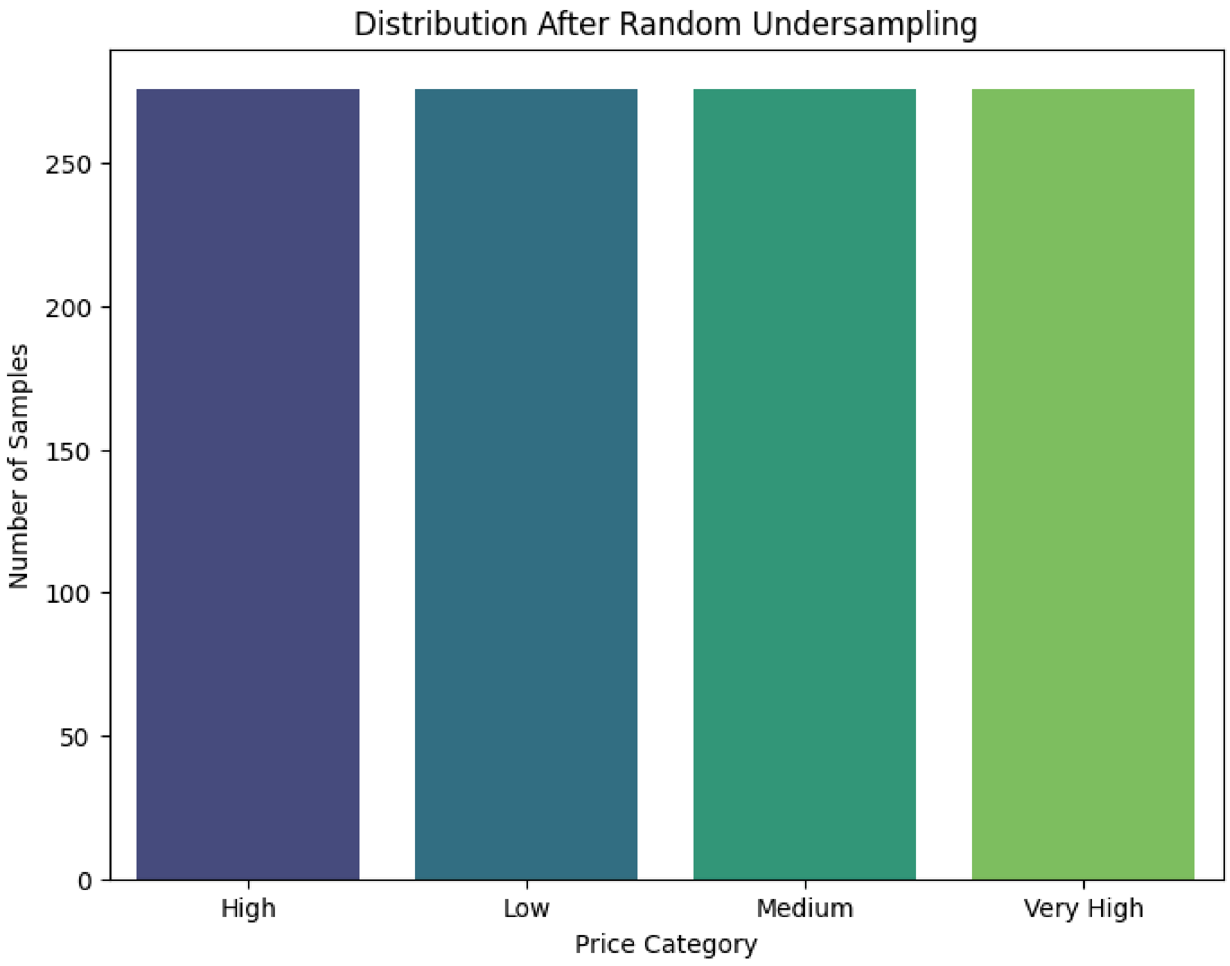
- This imbalance could lead the model to favor predictions for the majority class (`Medium`) while neglecting the minority classes (`Low` and `Very High`).

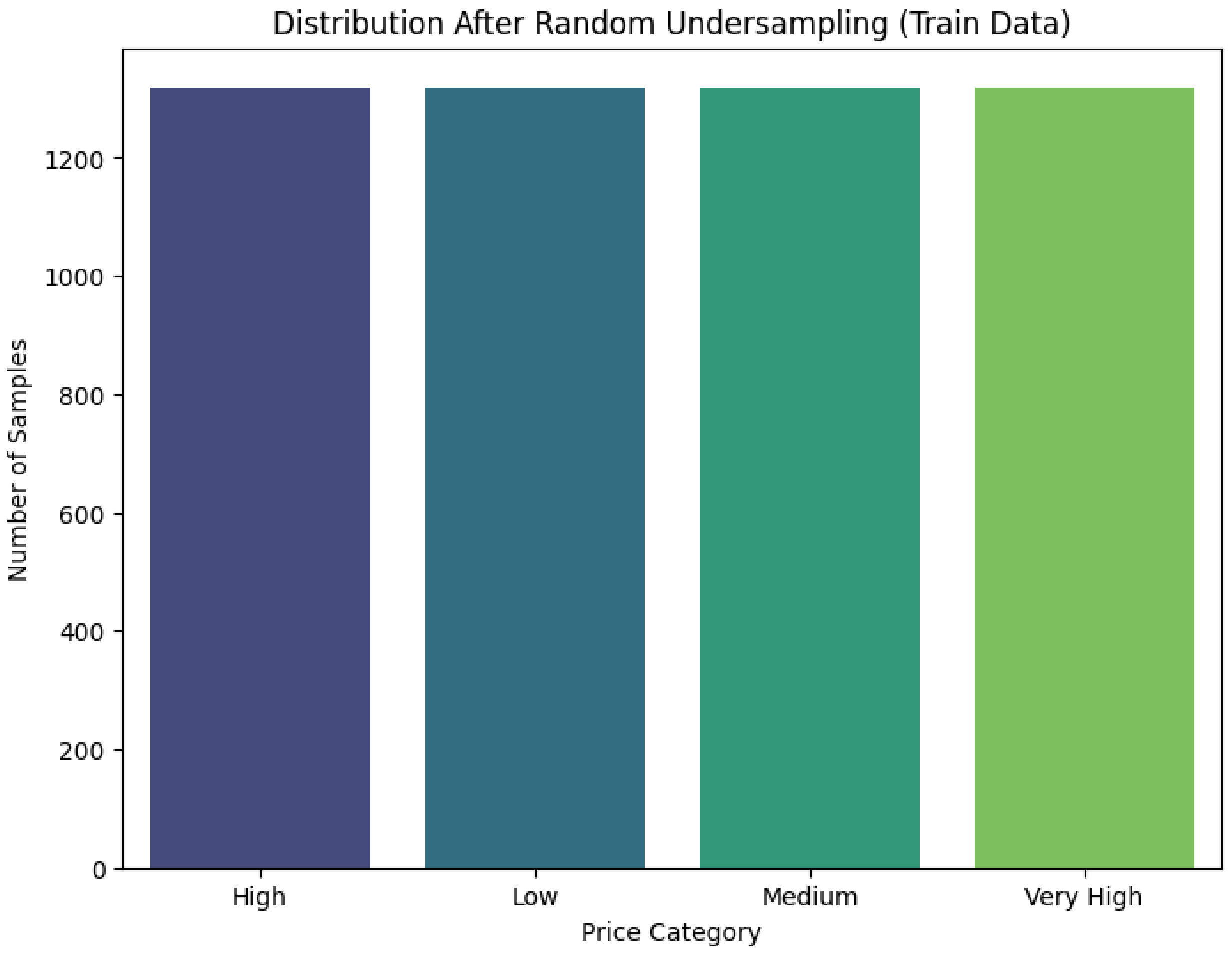
**2. After Random Undersampling**

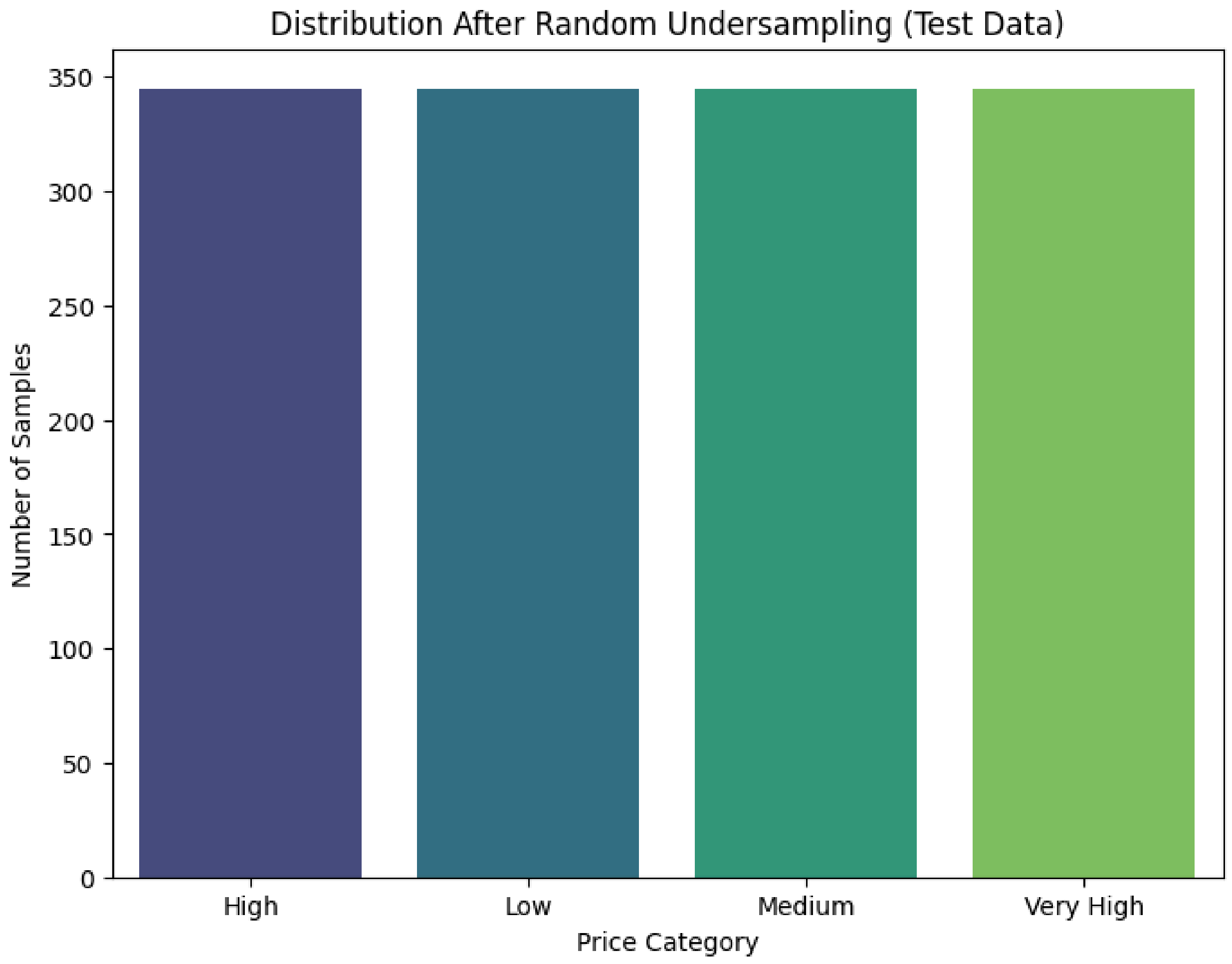
- **Train Data**: The number of samples across all categories is reduced to match the size of the smallest category.
- **Test Data**: Similarly, all categories in the test data are balanced by reducing the samples of the majority class.
- **Implications**:
    - Ensures a balanced representation of classes during training and testing.
    - However, the loss of majority class samples means that some data diversity is sacrificed.

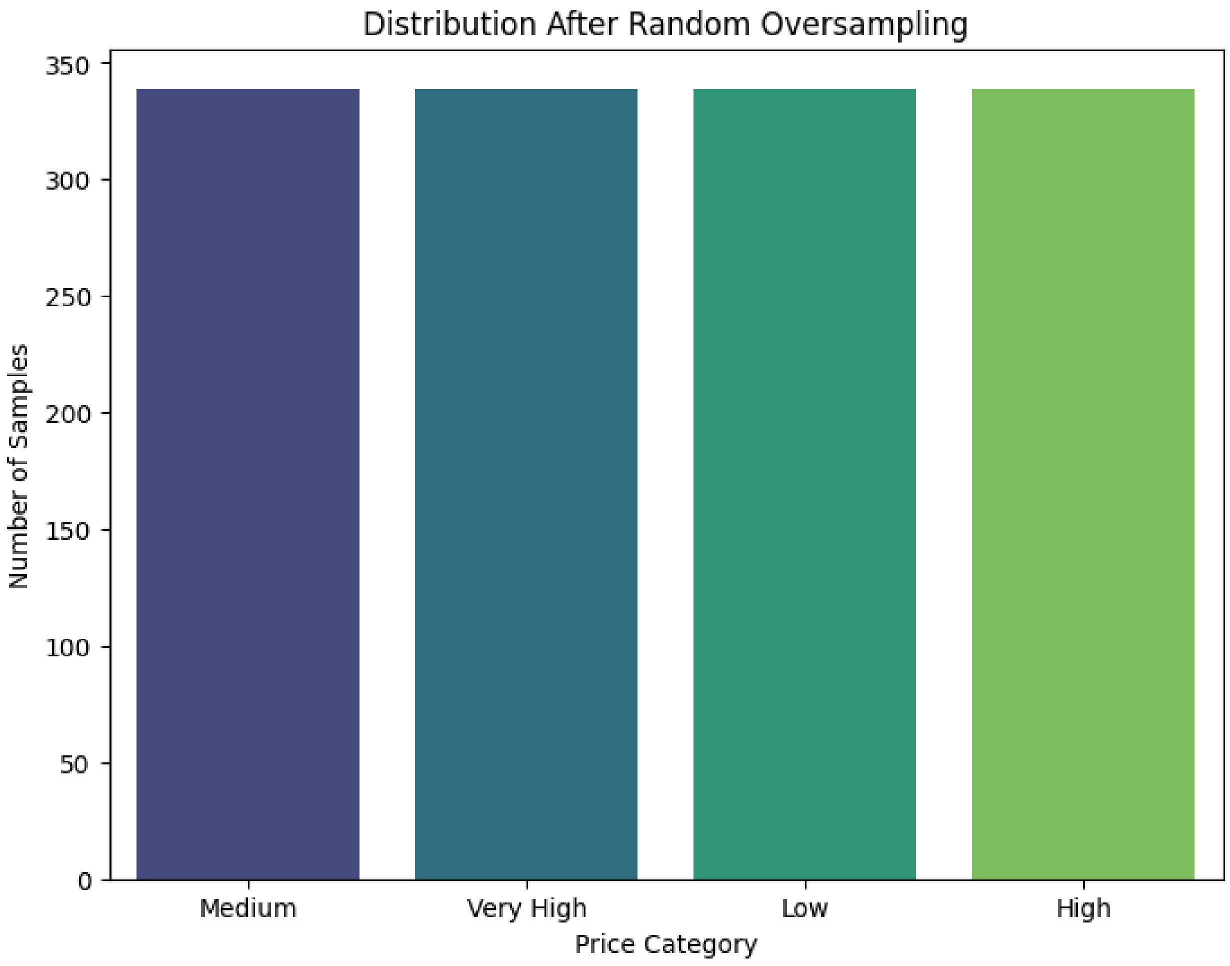
**3. After Random Oversampling**

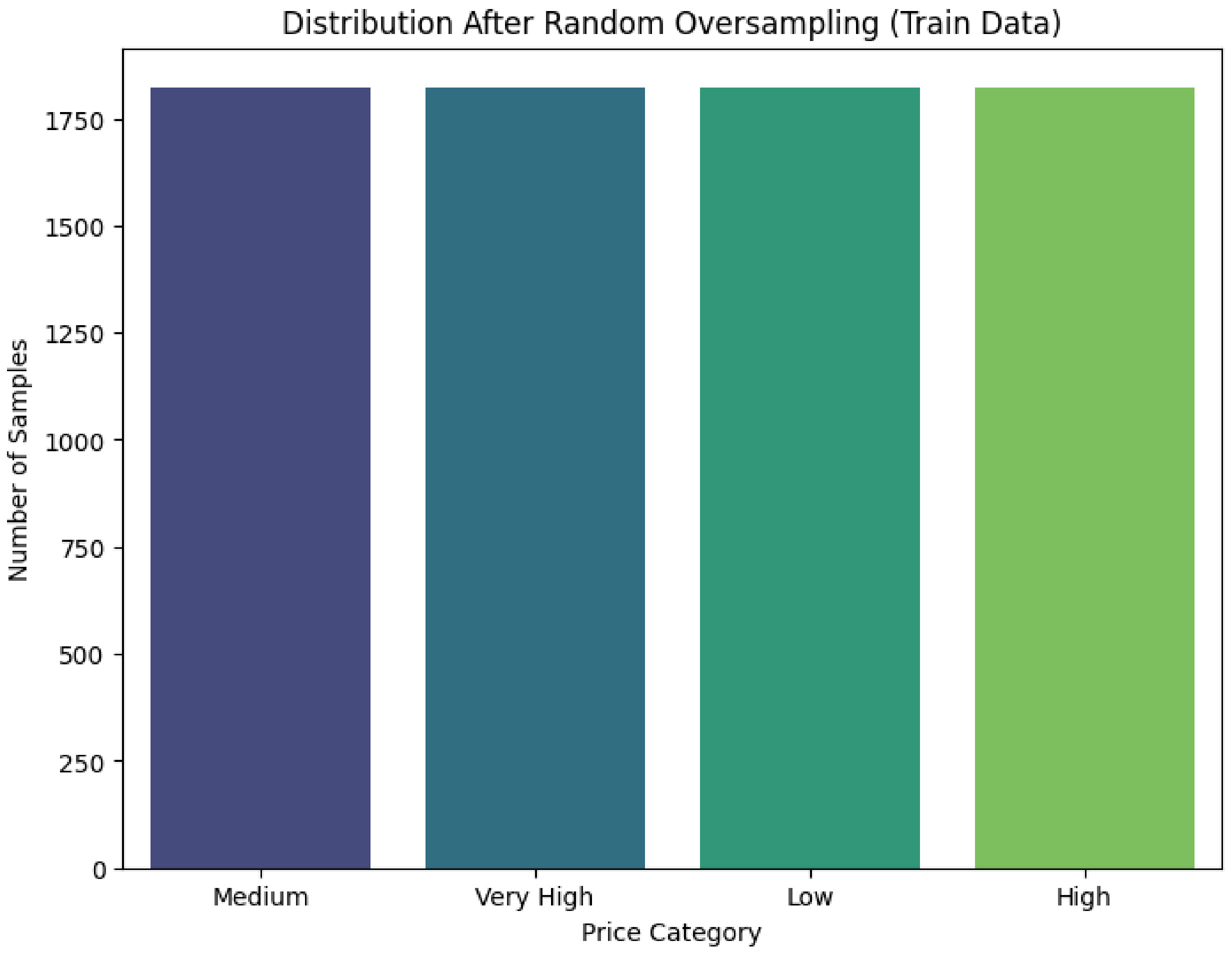
- **Train Data**: All categories are balanced by oversampling the minority classes to match the majority class size.
- **Test Data**: Minority categories are oversampled in the test dataset as well.
- **Implications**:
    - Balancing ensures the model is exposed to equal representation of all classes, reducing class bias.

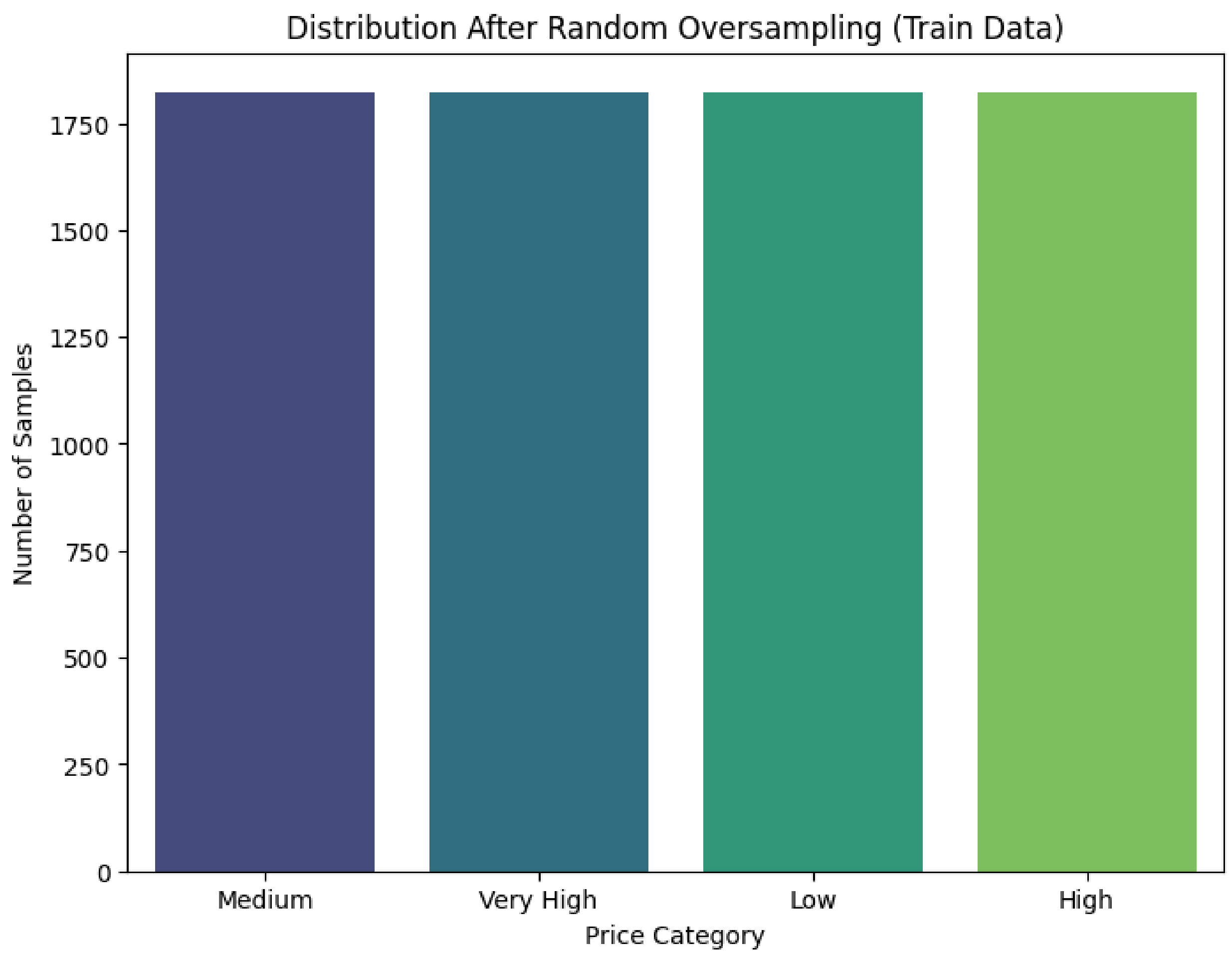

Distribution After Random Undersampling
Number of Samples
0
50
100
150
200
250
High
Low
Medium
Very High
Price Category


Distribution After Random Undersampling (Train Data)
Number of Samples
0
200
400
600
800
1000
1200
High
Low
Medium
Very High
Price Category


Distribution After Random Undersampling (Test Data)
Number of Samples
0
50
100
150
200
250
300
350
High
Low
Medium
Very High
Price Category

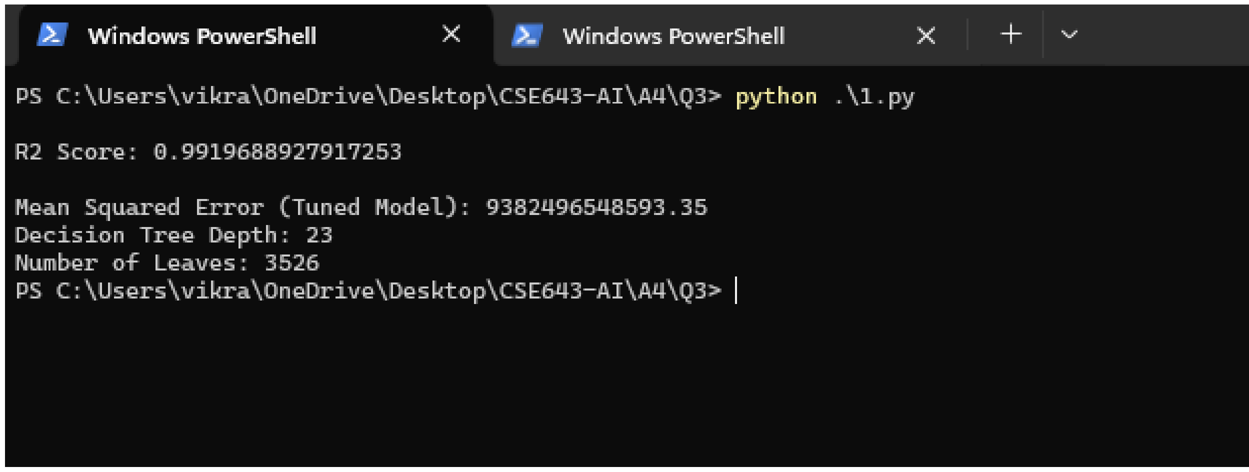

Distribution After Random Oversampling
Number of Samples
0
50
100
150
200
250
300
350
Medium
Very High
Low
High
Price Category


Distribution After Random Oversampling (Train Data)
Number of Samples
0
250
500
750
1000
1250
1500
1750
Medium
Very High
Low
High
Price Category



3.

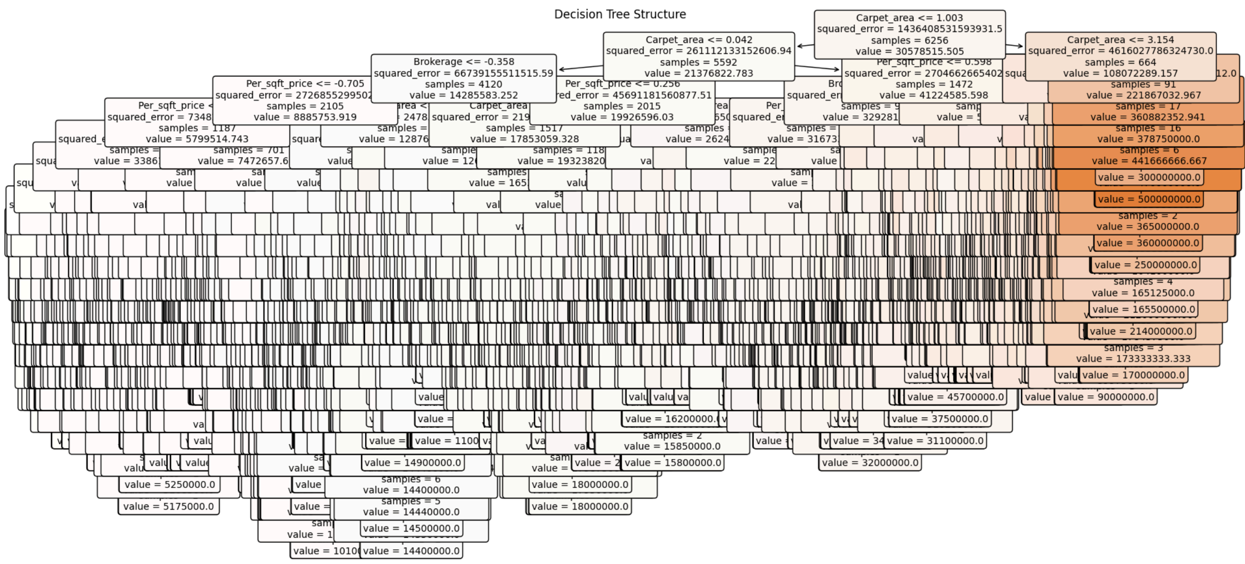
**Task 1:**

a. 

```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\1.py

R2 Score: 0.9919688927917253

Mean Squared Error (Tuned Model): 9382496548593.35
Decision Tree Depth: 23
Number of Leaves: 3526
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> |
```

This is the Trained Decision Tree on Scaled Unsampled Data as this has the best R2 score and least MSE among all the other combinations.

I have directly trained GridSearchCV on undersampled and oversampled data below.



b.

**Tree Depth**

- **Maximum Depth**: The tree appears to have a considerable depth, indicating the model's complexity. Depth is determined by the longest path from the root to a leaf.
- The `max_depth=10` parameter restricts the depth, balancing complexity and generalization.

**Splitting Decisions**

i. **Root Node**:
   1. The first decision (root node) splits based on the `Carpet_area` feature.
   2. It decides whether the `Carpet_area` value is less than or equal to **0.042** (standardized value).

ii. **Subsequent Nodes**:
   1. Nodes split the data further based on features such as:
      a. `Per_sqft_price`
      b. `Brokerage`
      c. `Carpet_area` (repeated at deeper levels due to high importance)
   2. Each split minimizes the **squared error** in predicting the property price.

iii. **Leaf Nodes**:

1. At the leaf nodes, the tree outputs the predicted property price (`value`).
2. The **samples** count at each leaf shows how many data points ended up in that segment.

### Key Features in Splitting

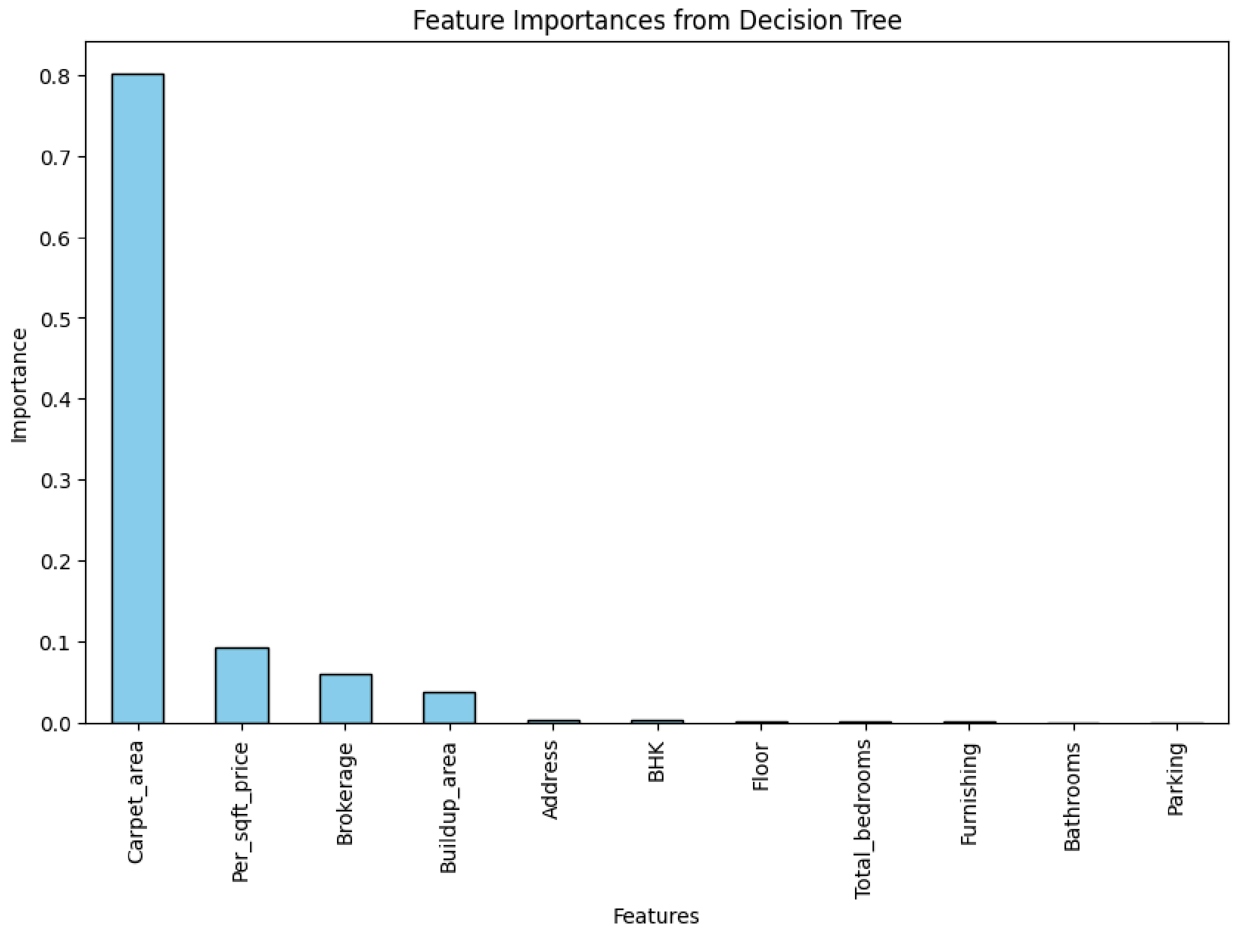
The tree prioritizes features based on their importance in reducing error:

**Carpet_area**: Used most frequently, indicating its high predictive power for property prices.

**Per_sqft_price**: Appears at multiple levels, often in early splits.

**Brokerage**: Helps refine predictions for higher-priced properties.

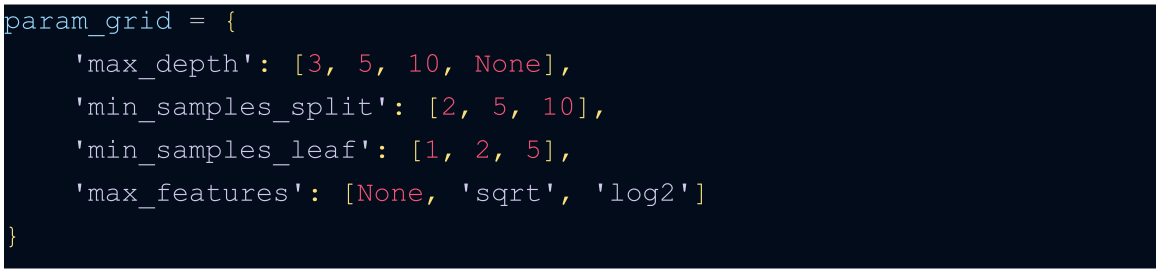
## Task 2:



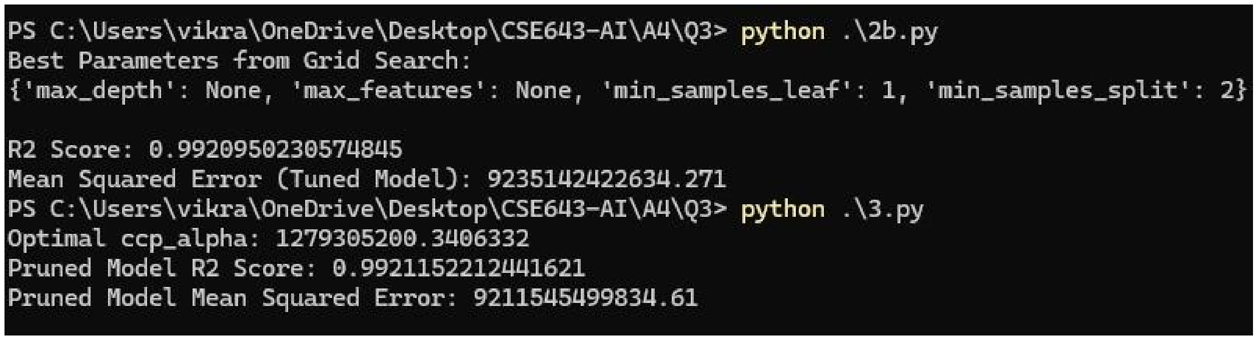
a.
b. Carpet Area has the highest importance at around 80%.
This matches my expectation as Carpet Area is basically the size of the property.
Per Sqft price is around 10%, though i expected it to be at around 15-20% in general.

Brokerage fees is at around 5%, which makes sense as it reflects transaction characteristics and thus it has less direct influence on the property price. Buildup Area is at 2%, which may be redundant due to Carpet Area. Address, Furnishing, BHK and other features have negligible importance.

c. I have used 'GridSearchCV' on these hyperparameters:



```
param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': [None, 'sqrt', 'log2']
}
```



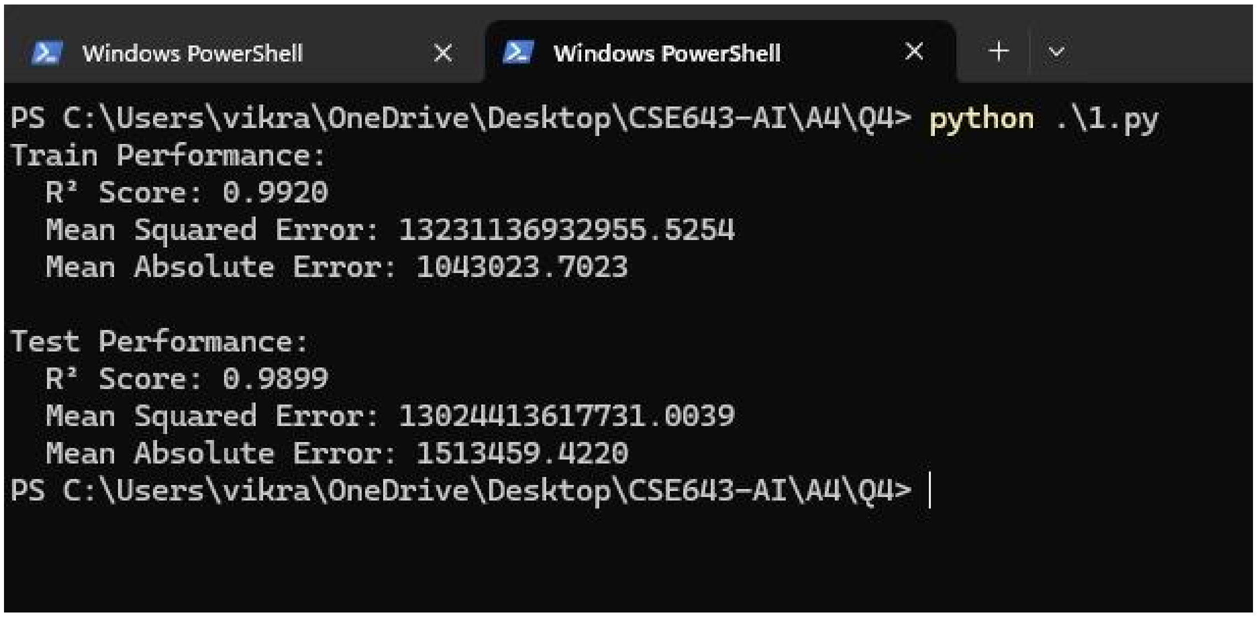
```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\2b.py
Best Parameters from Grid Search:
{'max_depth': None, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

R2 Score: 0.9920950230574845
Mean Squared Error (Tuned Model): 9235142422634.271
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\3.py
Optimal ccp_alpha: 1279305200.3406332
Pruned Model R2 Score: 0.9921152212441621
Pruned Model Mean Squared Error: 9211545499834.61
```

Grid Search values for unsampled data

{'max_depth': 10, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 5} were the Best hyperparameters for Scaled Unsampled Pruned Data.

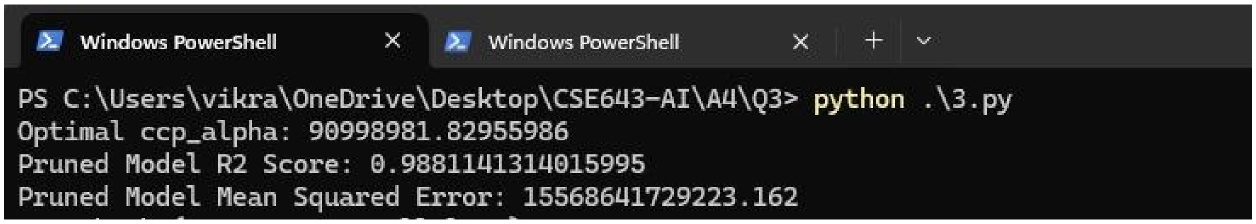
The r2 score was 99.21 compared to the 99.19%.
It is a very slight minor increase, which can sometimes also decrease due to a possibility of overfitting to the training set.

On Oversampled and Undersampled data:



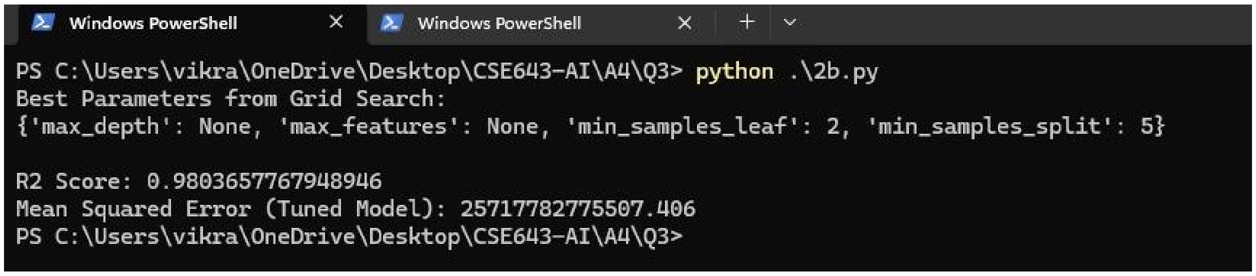
```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4> python .\1.py
Train Performance:
  R² Score: 0.9920
  Mean Squared Error: 13231136932955.5254
  Mean Absolute Error: 1043023.7023

Test Performance:
  R² Score: 0.9899
  Mean Squared Error: 13024413617731.0039
  Mean Absolute Error: 1513459.4220
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4>
```

Performance for best hyperparameter for undersampling



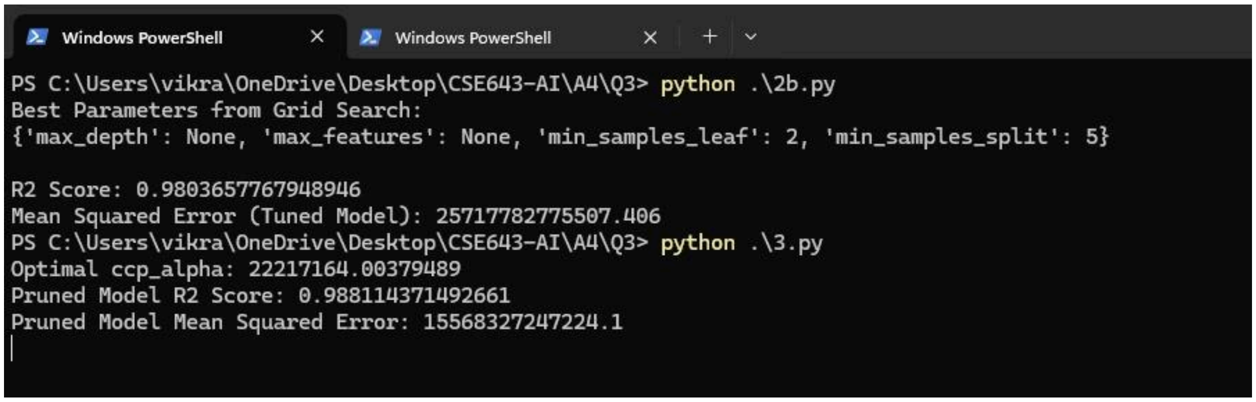
```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\3.py
Optimal ccp_alpha: 90998981.82955986
Pruned Model R2 Score: 0.9881141314015995
Pruned Model Mean Squared Error: 15568641729223.162
```

Optimal cc_alpha value for undersampling



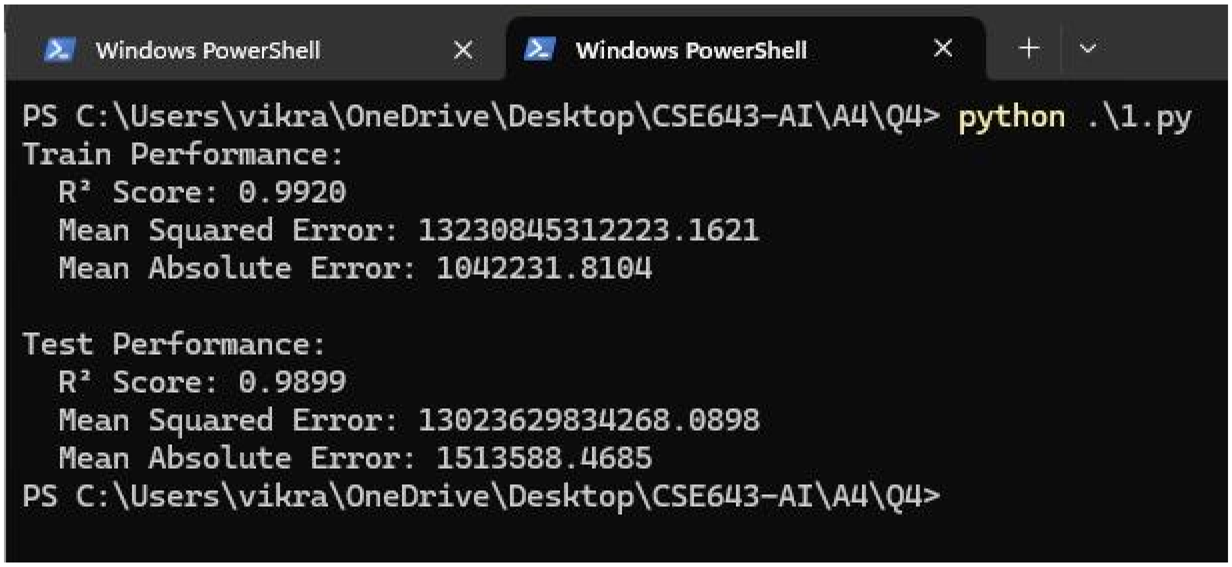
```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\2b.py
Best Parameters from Grid Search:
{'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 5}

R2 Score: 0.9803657767948946
Mean Squared Error (Tuned Model): 25717782775507.406
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3>
```

Grid Search values for oversampling



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\2b.py
Best Parameters from Grid Search:
{'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 5}

R2 Score: 0.9803657767948946
Mean Squared Error (Tuned Model): 25717782775507.406
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\3.py
Optimal ccp_alpha: 22217164.00379489
Pruned Model R2 Score: 0.988114371492661
Pruned Model Mean Squared Error: 15568327247224.1
```

Optimal cc_alpha value for oversampling

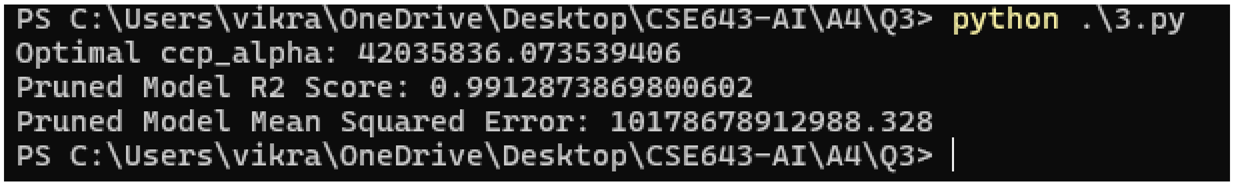


```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4> python .\1.py
Train Performance:
  R² Score: 0.9920
  Mean Squared Error: 13230845312223.1621
  Mean Absolute Error: 1042231.8104

Test Performance:
  R² Score: 0.9899
  Mean Squared Error: 13023629834268.0898
  Mean Absolute Error: 1513588.4685
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4>
```

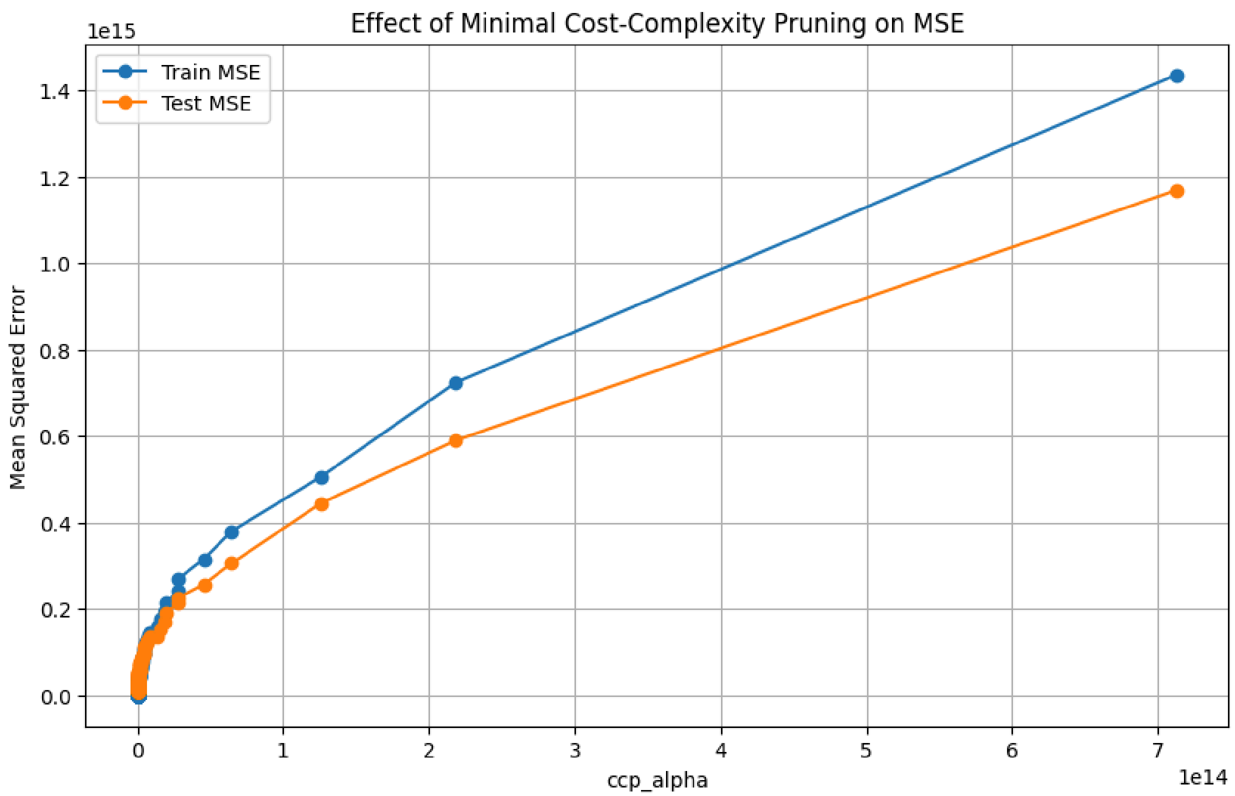
Performance for best hyperparameter for oversampling

## Task 3:

a. The Best Optimal ccp alpha value came to be 42035836.073539406



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\3.py
Optimal ccp_alpha: 42035836.073539406
Pruned Model R2 Score: 0.9912873869800602
Pruned Model Mean Squared Error: 10178678912988.328
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3>
```



b.

Pruning and unpruned decision trees are two approaches to managing the complexity of a decision tree model, and they have a significant impact on the model's generalization ability.

**Unpruned Decision Tree**:

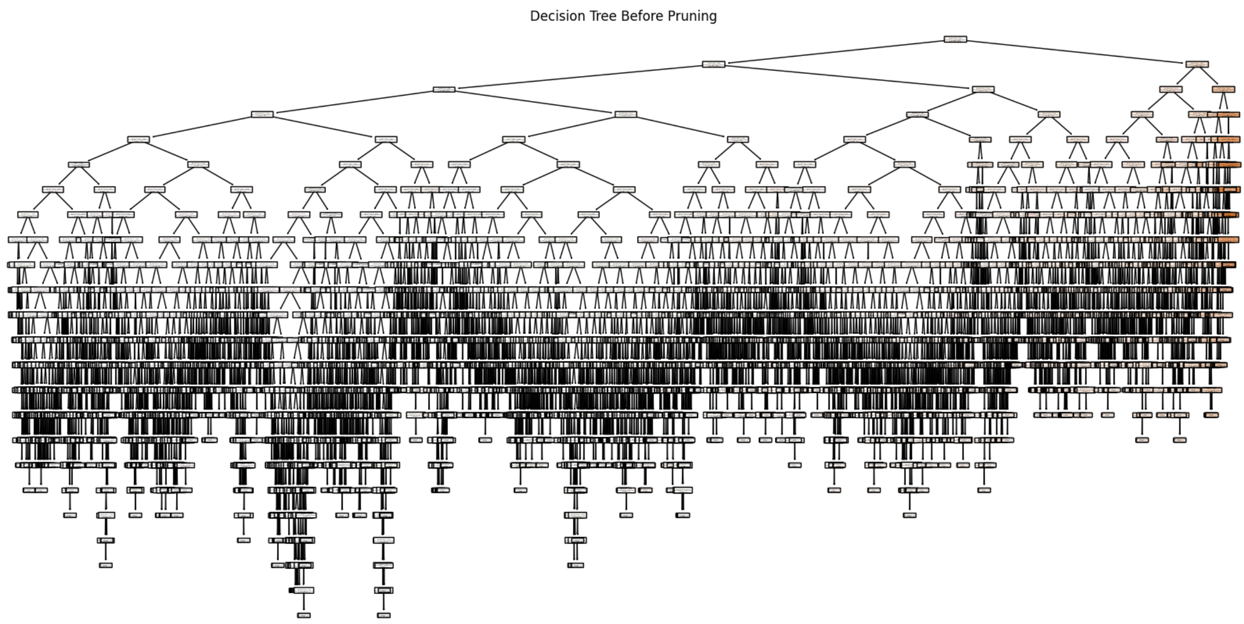
- An unpruned decision tree is a tree that is grown fully without any constraints during training. This means the tree will continue to split the data until all the training data is perfectly classified, or the tree reaches the maximum depth allowed by the hyperparameters.
- **Overfitting Risk**: Unpruned trees have a high risk of overfitting. Because they are allowed to grow deep and create many branches, they can memorize the training data, capturing noise and outliers that are not representative of the overall population.
- **Training Data**: An unpruned tree will perform very well on the training data but may perform poorly on unseen test data due to its complexity and overfitting.
- **Large and Complex**: An unpruned tree can become very large and complex, with many levels and nodes. It might split the data into smaller and smaller subsets until it can perfectly classify all the training examples.
- **High Variance**: Since the tree is overly complex, it will likely have high variance. This means that small changes in the training data could lead to significant changes in the tree structure and, consequently, in the predictions made by the model.

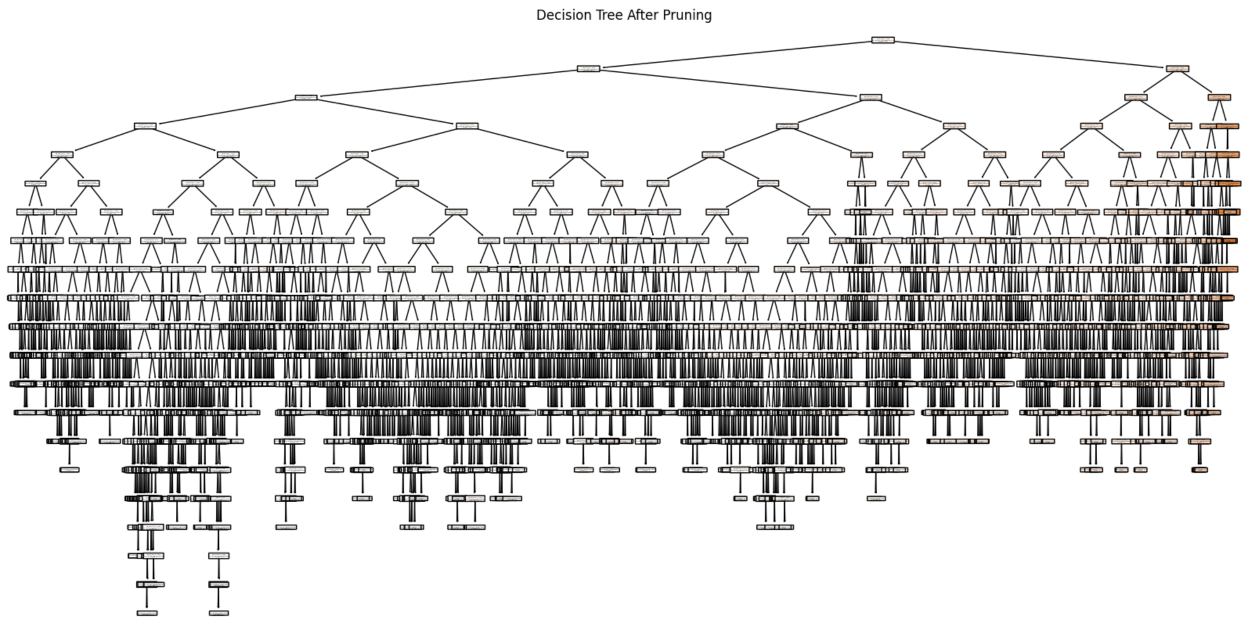
- **Good on Training Data, Poor on Test Data**: An unpruned tree will often have low bias but high variance, leading to excellent performance on training data but poor generalization to test data.

**Pruned Decision Tree**:

- A pruned decision tree is one where some of the nodes or branches are removed after the tree has been grown. The goal of pruning is to remove sections of the tree that provide little predictive power to reduce overfitting.
- **Reduced Overfitting**: Pruning helps reduce the model's complexity, which reduces overfitting. By removing parts of the tree that don't provide significant predictive value, pruned trees can generalize better on unseen data.
- **Training Data**: A pruned tree may have a slightly higher error on the training data compared to an unpruned tree, but it will often perform better on the test data because it avoids overfitting.
- **Simplified Tree**: A pruned tree is smaller and more interpretable than an unpruned tree. By removing less important branches, the tree structure becomes simpler and easier to understand, while still retaining its ability to predict.
- **Balanced Bias-Variance Tradeoff**: Pruned trees aim for a good balance between bias and variance. They have a lower variance compared to unpruned trees and, as a result, are better at generalizing to unseen data. They may introduce some bias by simplifying the model, but this is often a necessary tradeoff to avoid overfitting.
- **Good Generalization**: Pruned trees typically have better performance on test data, since they generalize better, especially when compared to unpruned trees that overfit the training data.

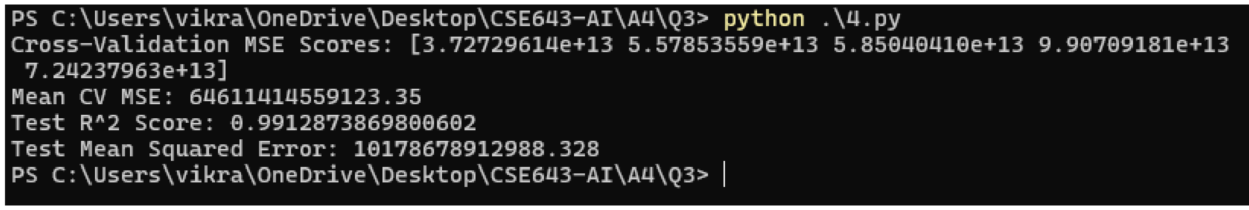
Here, I have used **Cost Complexity Pruning**. One of the pruning methods is cost-complexity pruning (`ccp_alpha`), which eliminates nodes that do not significantly improve the tree's prediction. This method tries to minimize both the size and complexity of the tree.





## Task 4:

a.



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> python .\4.py
Cross-Validation MSE Scores: [3.72729614e+13 5.57853559e+13 5.85040410e+13 9.90709181e+13
 7.24237963e+13]
Mean CV MSE: 64611414559123.35
Test R^2 Score: 0.9912873869800602
Test Mean Squared Error: 10178678912988.328
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q3> 
```

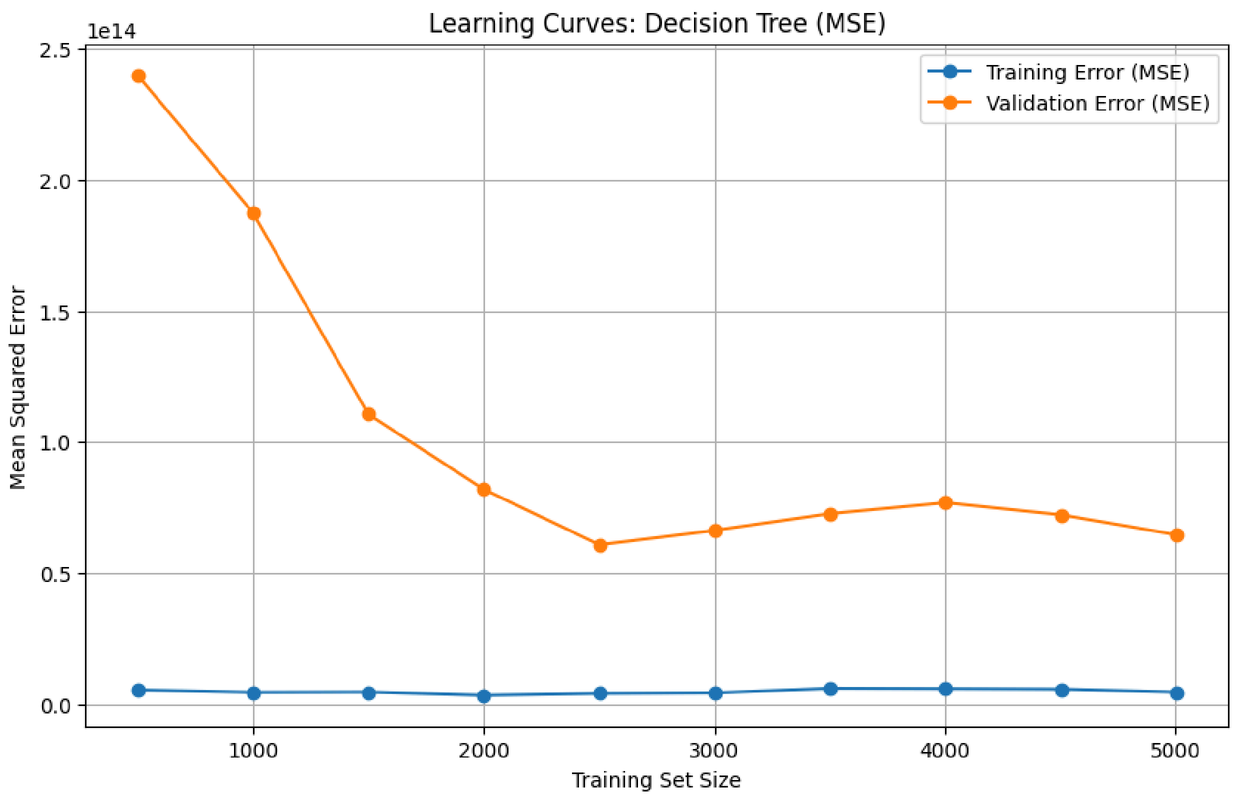
b. Here, we can see that the training error is very very near to zero and remains almost constant across different training set sizes. This indicates that the model is able to perform very well on the training data.

As the training set size increases, the training error remains small, meaning that the model is able to fit the training data very well, even with larger training sets.

The validation error initially starts high and decreases as more training data is used. However, as the training set size increases beyond a certain point, the validation error plateaus at a higher value compared to the training error.
Here, we can see that the validation error does not decrease as much as the training error. This indicates that while the model is learning the patterns in the training data, it is not generalizing well to the unseen data, which suggests overfitting.

There is a clear risk of overfitting, but the validation error decreases with every set, and is also less error. The model's ability to achieve a very low training error, but a higher and more stable validation error, is a classic sign of overfitting. This occurs because the model is becoming overly complex, capturing noise or irrelevant patterns in the training data that do not generalize well to new data.

The fact that the validation error levels off as the training set size increases suggests that the model has reached its capacity to generalize. Even with more data, the model cannot improve its performance on unseen data, which is another clear sign of overfitting.



c.

**Cross-validation** is a technique where the dataset is split into multiple parts (folds) and the model is trained and validated on different subsets. It helps **detect overfitting** by providing a more reliable estimate of a model's ability to generalize to unseen data.

**Key Benefits:**

1. **Reduces Bias**: Evaluates the model on multiple data subsets, giving a more robust performance estimate.
2. **Better Generalization**: Helps prevent overfitting by showing how well the model performs on unseen data.
3. **Hyperparameter Tuning**: Assists in selecting optimal hyperparameters (e.g., tree depth, pruning) to avoid overly complex models.
4. **Prevents Overfitting to a Single Validation Set**: Since the validation set changes in each fold, the model's performance is less likely to be skewed by a single dataset.

**How It Works:**

- **Model Complexity**: Cross-validation helps identify the optimal tree depth and structure by testing the model on various data splits.
- **Cost-Complexity Pruning**: Cross-validation helps in pruning trees to the right complexity, balancing bias and variance.

**Limitations:**

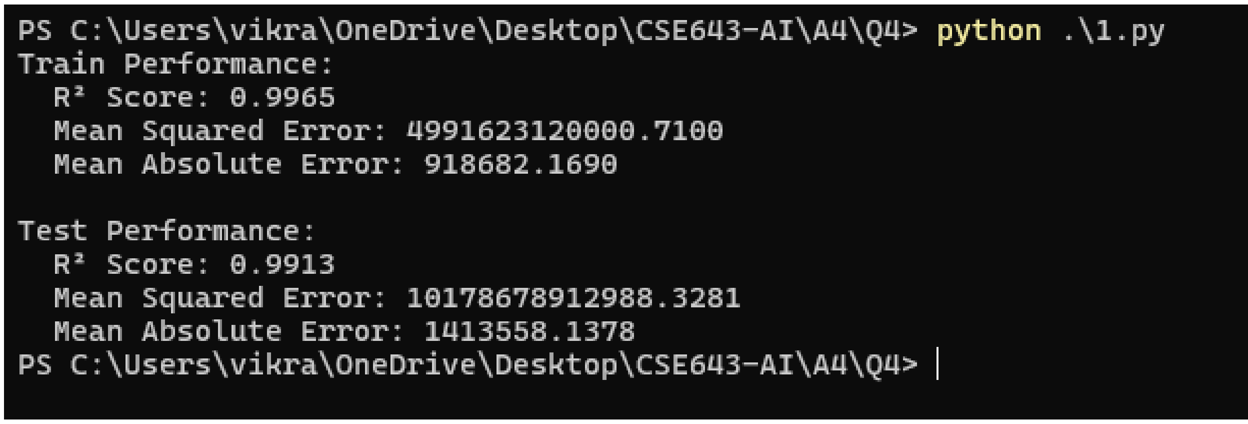
- **Computational Cost**: It requires multiple training iterations, increasing computational time.
- **Not Effective for Noisy Data**: Cross-validation might still struggle with highly noisy datasets.

4.

## Task 1:

a. I took the evaluation metrics from scikit-learn

```
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
```

b. 

```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4> python .\1.py
Train Performance:
  R² Score: 0.9965
  Mean Squared Error: 4991623120000.7100
  Mean Absolute Error: 918682.1690

Test Performance:
  R² Score: 0.9913
  Mean Squared Error: 10178678912988.3281
  Mean Absolute Error: 1413558.1378
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4>
```

**Train Performance:**

- **R² Score: 0.9965**: The model explains 99.65% of the variance in the training data, indicating very strong predictive performance on the training set.
- **Mean Squared Error (MSE): 4991623100000.7100**: This is the average squared difference between the predicted and actual values. A lower MSE indicates better fit, but the high scale of the value suggests a high range of target values.
- **Mean Absolute Error (MAE): 918682.1690**: This is the average absolute difference between the predicted and actual values, showing how much on average the model's predictions deviate from the true values.

**Test Performance:**

- **R² Score: 0.9913**: The model explains 99.13% of the variance in the test data. Though it is slightly lower than the training R² score, it is still a very strong value, suggesting good generalization to unseen data.

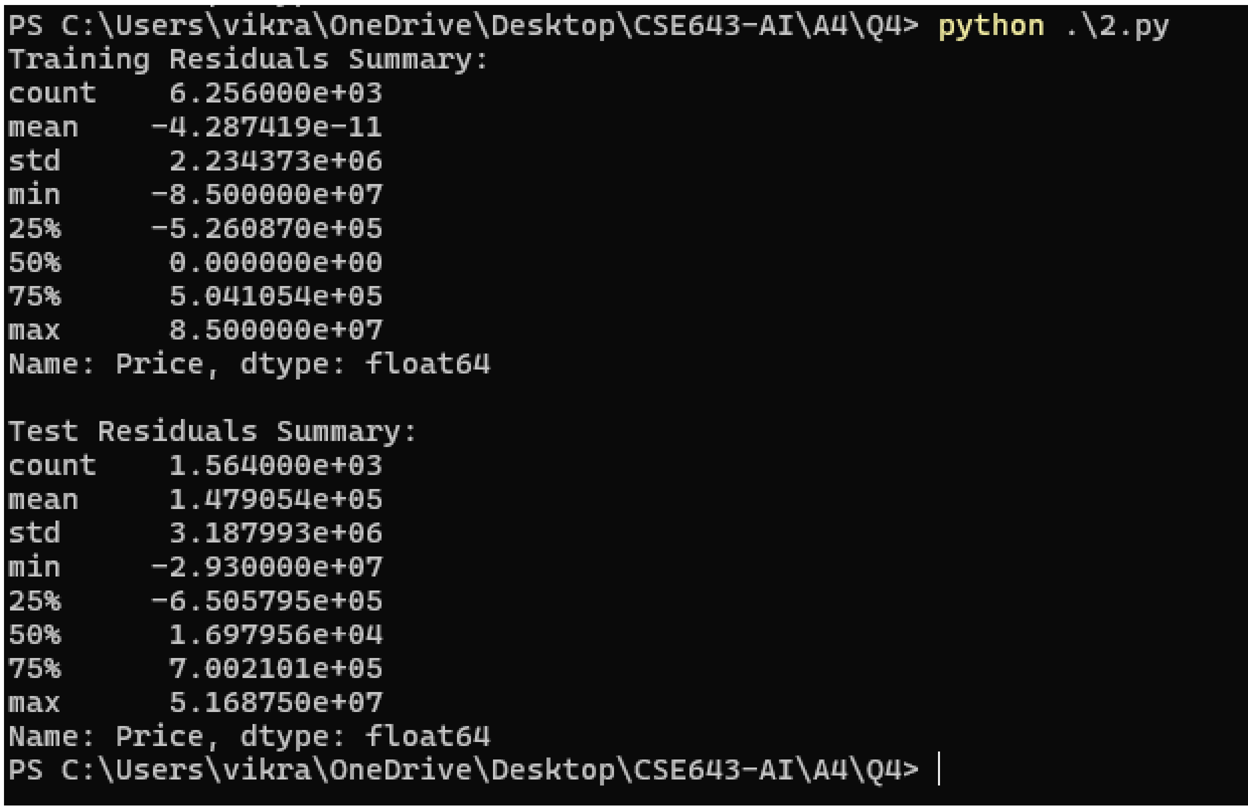
- **Mean Squared Error (MSE): 10178678912988.3281**: The MSE on the test set is higher than on the training set, which is expected but still relatively low. This may indicate slight overfitting, as the model performs slightly better on the training data.
- **Mean Absolute Error (MAE): 1413558.1378**: The test MAE is higher than the train MAE, which is also expected. However, the magnitude of this difference is not very large, meaning the model's performance remains good on the test set.

The model performs well on both the training and test datasets, with very high R² scores (close to 1) and relatively low MSE and MAE values, suggesting a good fit to the data and strong predictive capabilities.

The slight drop in test R² and increase in MSE/MAE compared to training performance suggests minor overfitting, but the performance is still excellent, indicating that the model generalizes well to unseen data.

## Task 2:

a.



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4> python .\2.py
Training Residuals Summary:
count    6.256000e+03
mean    -4.287419e-11
std      2.234373e+06
min     -8.500000e+07
25%     -5.260870e+05
50%      0.000000e+00
75%      5.041054e+05
max      8.500000e+07
Name: Price, dtype: float64

Test Residuals Summary:
count    1.564000e+03
mean     1.479054e+05
std      3.187993e+06
min     -2.930000e+07
25%     -6.505795e+05
50%      1.697956e+04
75%      7.002101e+05
max      5.168750e+07
Name: Price, dtype: float64
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4>
```

**Training Residuals Summary:**

- **Count**: 6256 (The number of training data points used).

- **Mean**: -4.287419e-11 (The residuals are centered close to zero, which suggests that on average, the model's predictions are quite close to the actual values).
- **Standard Deviation (std)**: 2.234373e+06 (The spread of residuals is large, indicating that while the predictions are accurate on average, there is a significant variation in how close the predictions are to the true values).
- **Min**: -8.500000e+07 (The lowest residual, suggesting that some predictions are off by a large margin).
- **25th Percentile (Q1)**: -5.260870e+05 (A quarter of the training residuals are smaller than this value).
- **50th Percentile (Median)**: 0.000000e+00 (The median residual is close to zero, indicating that half the predictions are overestimated and half are underestimated, but very close to the true value).
- **75th Percentile (Q3)**: 5.040154e+05 (Three-quarters of the residuals are smaller than this value).
- **Max**: 8.500000e+07 (The highest residual, indicating that some predictions are significantly off).

**Test Residuals Summary:**

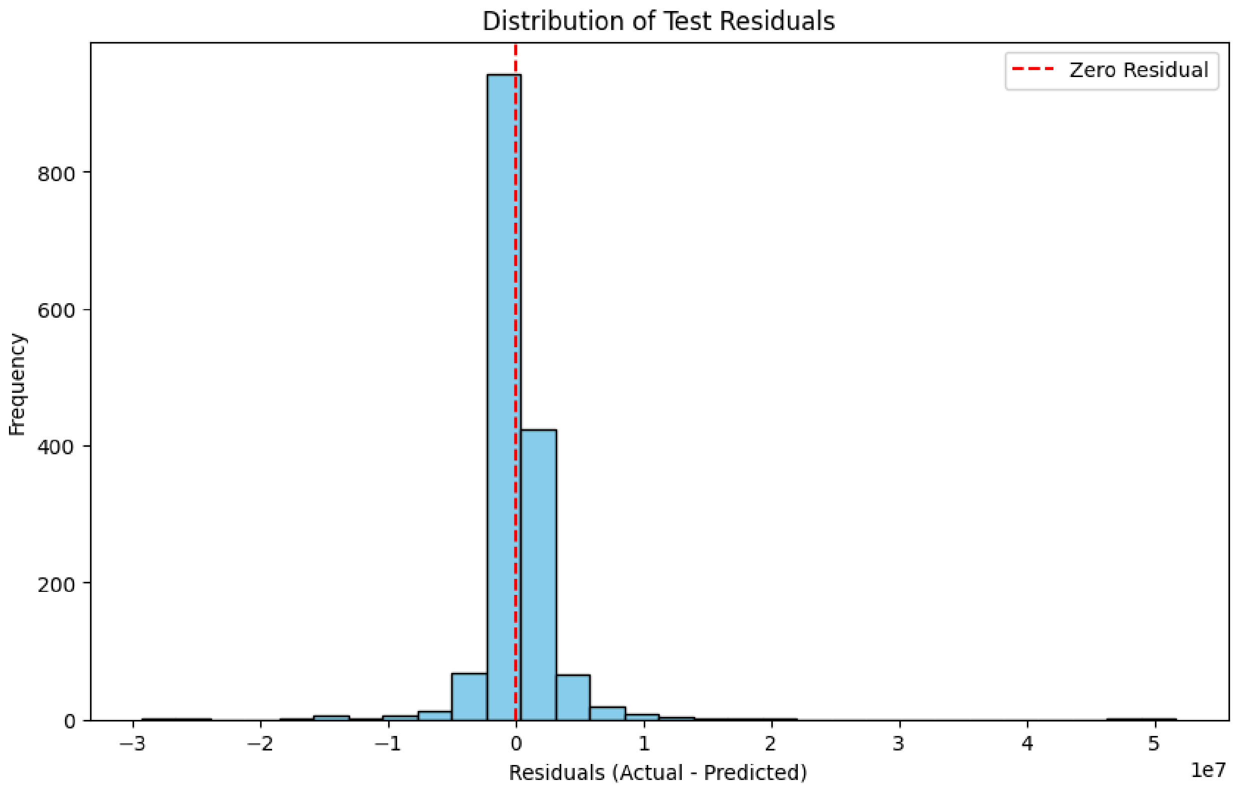
- **Count**: 1564 (The number of test data points used).
- **Mean**: 1.479054e+05 (The residuals are somewhat positive on average, suggesting that on average, the model tends to slightly underestimate the prices for the test set).
- **Standard Deviation (std)**: 3.187993e+06 (The spread of residuals is again quite large, indicating that the test predictions vary widely).
- **Min**: -2.930000e+07 (The lowest residual is quite large in magnitude, implying significant underestimation for some test samples).
- **25th Percentile (Q1)**: -6.505795e+05 (A quarter of the test residuals are below this value).
- **50th Percentile (Median)**: 1.697956e+04 (The median residual is a small positive number, suggesting that many predictions are slightly underestimating the actual prices).
- **75th Percentile (Q3)**: 7.002101e+05 (Three-quarters of the test residuals are smaller than this value).
- **Max**: 5.168750e+07 (The highest residual in the test set, suggesting some major overestimations).

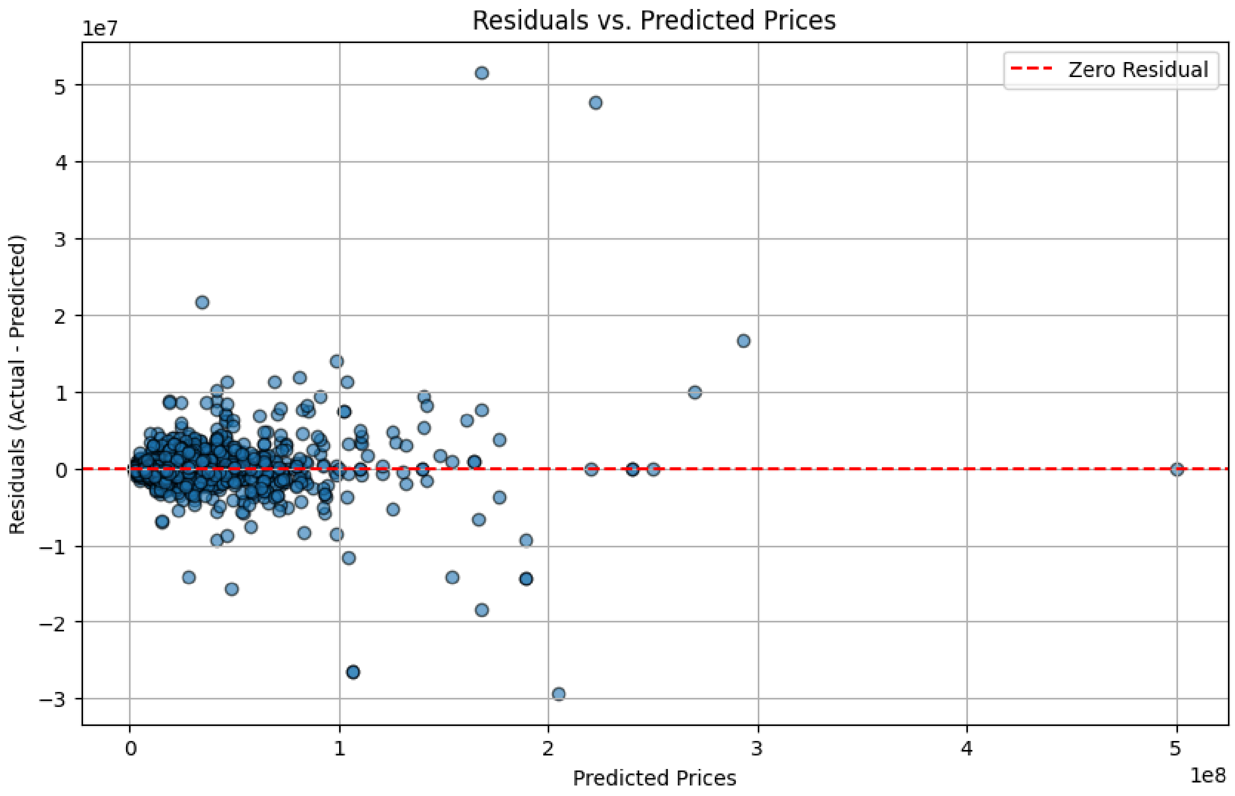
b. Here, it shows a highly concentrated peak around zero, indicating that the majority of the predictions are quite close to the true values. This suggests that the model is generally performing well for most of the data points. However, the histogram also reveals the presence of some extreme residuals, where the predictions are off by a large margin in either direction (positive or negative). This indicates that while the model is accurate on average, there are specific cases where it performs poorly.

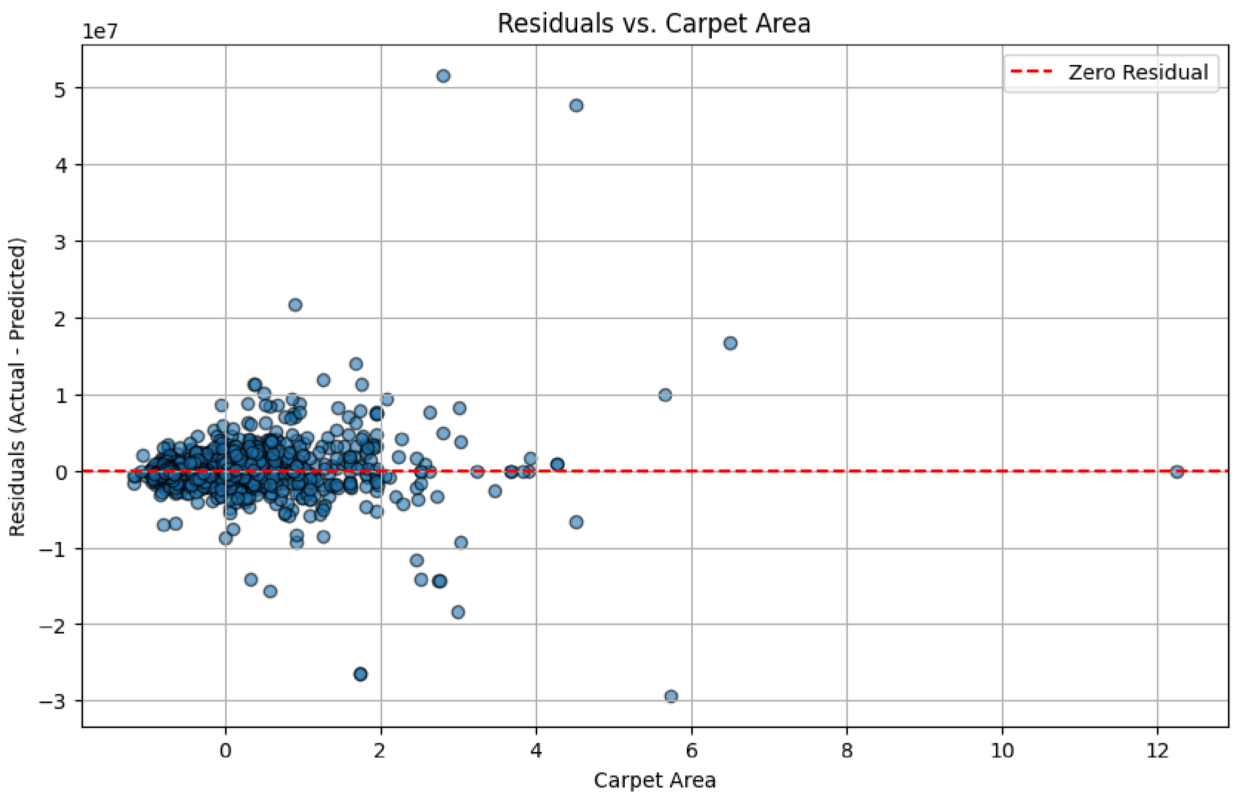
The residuals are mostly clustered around zero, which is good because it indicates that most of the predictions are accurate.

There are a few residuals that fall far away from zero, both positive and negative. This suggests that the model has difficulty in accurately predicting certain instances. These could be extreme property prices, for example, which are more challenging to model.

We can try Outlier Handling if extreme property prices are influencing the model's predictions too heavily, they can be treated separately.

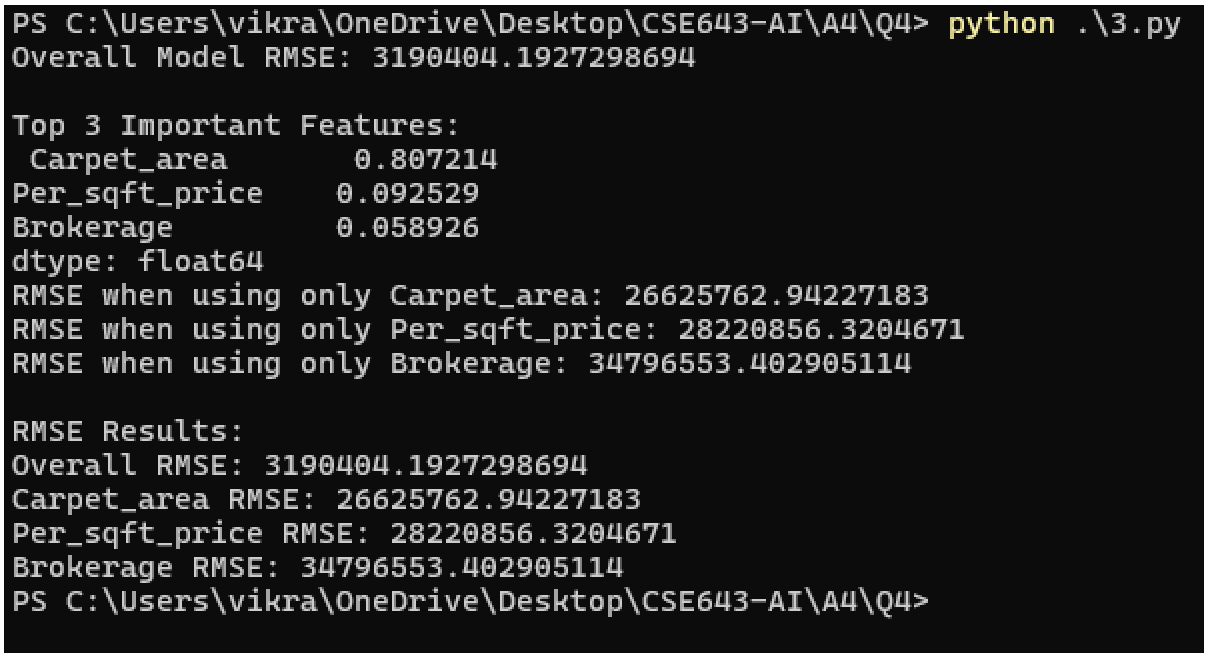

Distribution of Test Residuals
Zero Residual
Frequency
Residuals (Actual - Predicted)
1e7


Residuals vs. Predicted Prices
Zero Residual
Residuals (Actual - Predicted)
Predicted Prices
1e7
1e8



## Task 3:

a.



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4> python .\3.py
Overall Model RMSE: 3190404.1927298694

Top 3 Important Features:
 Carpet_area       0.807214
Per_sqft_price     0.092529
Brokerage          0.058926
dtype: float64
RMSE when using only Carpet_area: 26625762.94227183
RMSE when using only Per_sqft_price: 28220856.3204671
RMSE when using only Brokerage: 34796553.402905114

RMSE Results:
Overall RMSE: 3190404.1927298694
Carpet_area RMSE: 26625762.94227183
Per_sqft_price RMSE: 28220856.3204671
Brokerage RMSE: 34796553.402905114
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q4>
```

**Carpet Area**: RMSE when using only this feature is **26,625,762.94**.

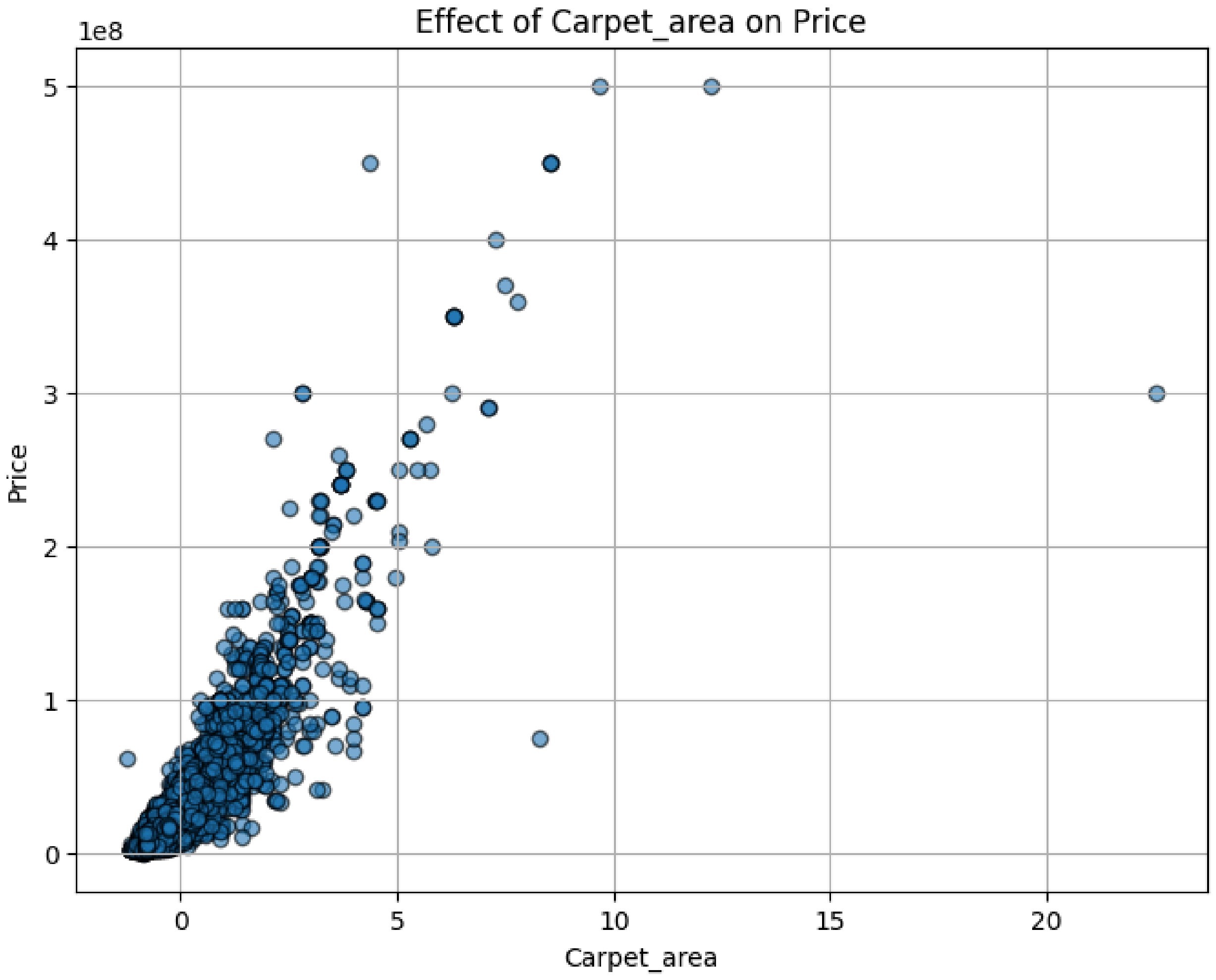
- **Effect on Price**: This feature appears to have the most influence on the price, as shown by its high importance score of **0.8072**. Carpet area is likely a key determinant in property pricing. A larger carpet area generally corresponds to a higher property price. However, the relatively high RMSE when using only this feature suggests that while it is an important factor, it does not account for all the variability in property prices.

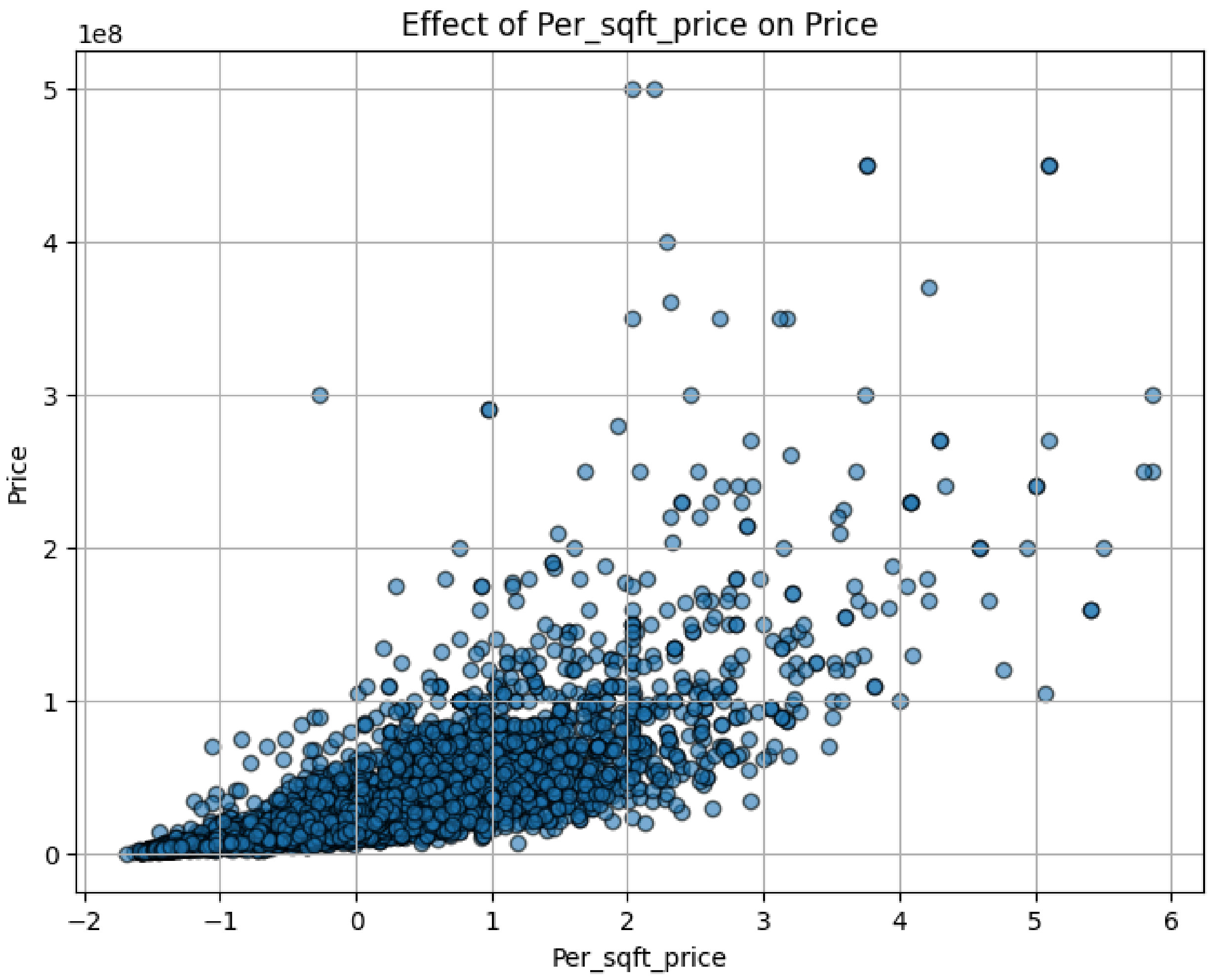
  **Per Square Foot Price**: RMSE when using only this feature is **28,220,856.32**.

- **Effect on Price**: This feature, with an importance score of **0.0925**, has a moderate impact on pricing. The per square foot price helps normalize property prices across different areas, but it alone doesn't explain all the variance in property values. The higher RMSE compared to **Carpet Area** implies that other factors are necessary for accurate pricing predictions.

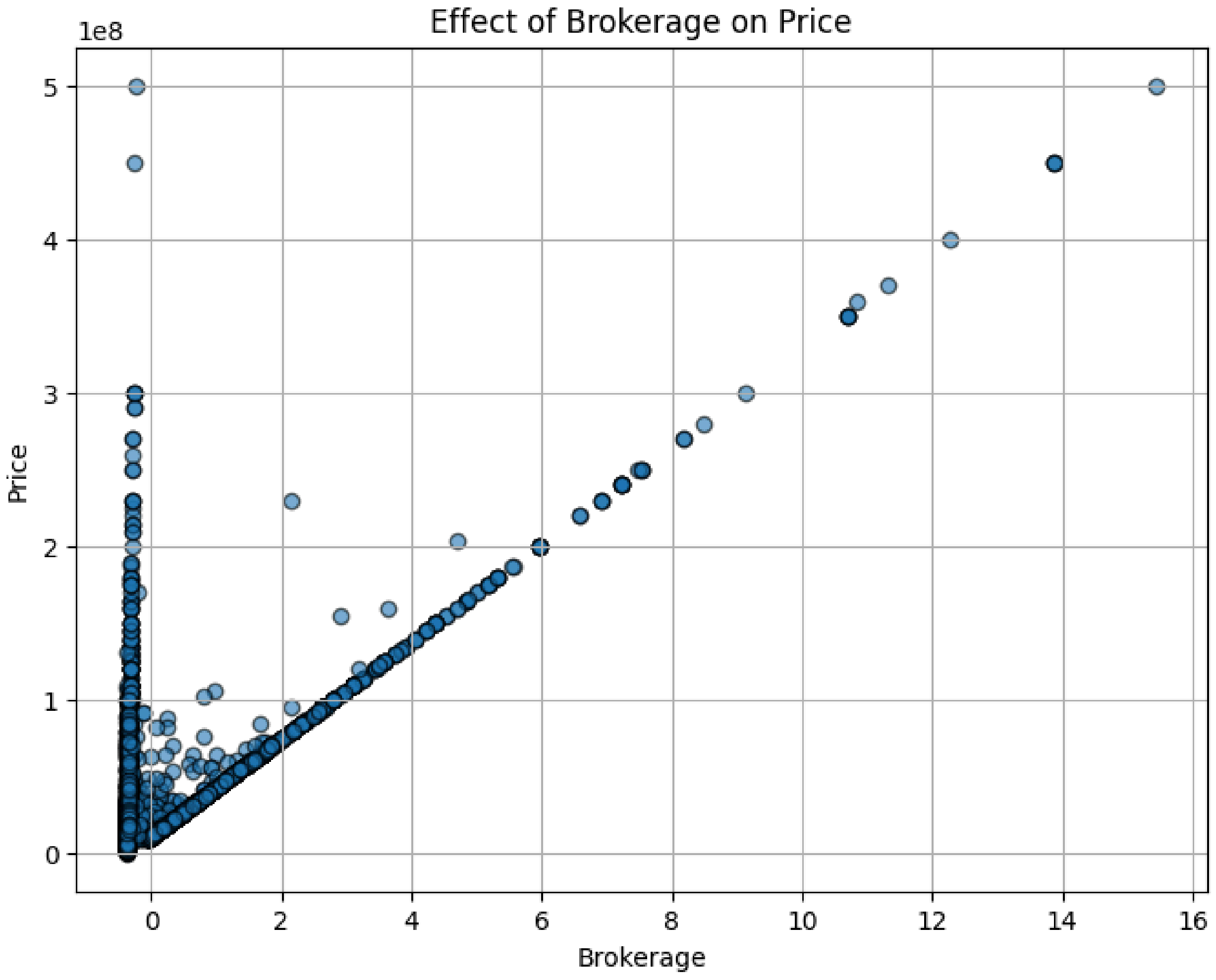
  **Brokerage**: RMSE when using only this feature is **34,796,553.40**.

- **Effect on Price**: Although brokerage plays a role in pricing, its importance score of **0.0589** suggests it has less influence than carpet area or per square foot price. The RMSE is the highest for this feature, indicating that the model's predictions are less accurate when only using brokerage as the feature. This could be due to its smaller impact or variability in how brokerage fees are applied across different regions or property types.


Effect of Carpet_area on Price
1e8
Price
Carpet_area
0
1
2
3
4
5
0
5
10
15
20

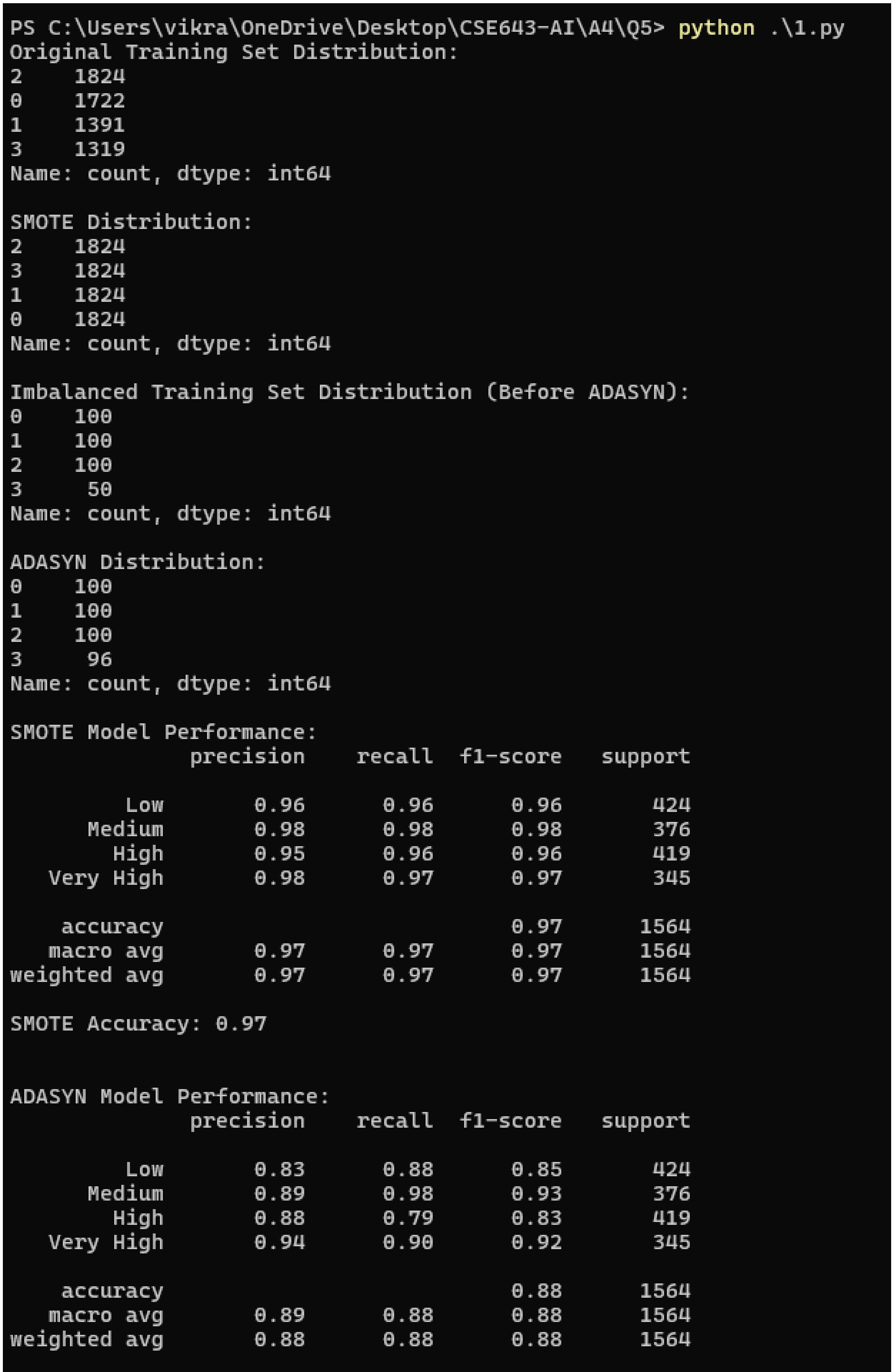


Effect of Per_sqft_price on Price
1e8
Price
Per_sqft_price
0
1
2
3
4
5
−2
−1
0
1
2
3
4
5
6


Effect of Brokerage on Price
1e8
Price
Brokerage
0
1
2
3
4
5
0
2
4
6
8
10
12
14
16

5.

**Task 1:**



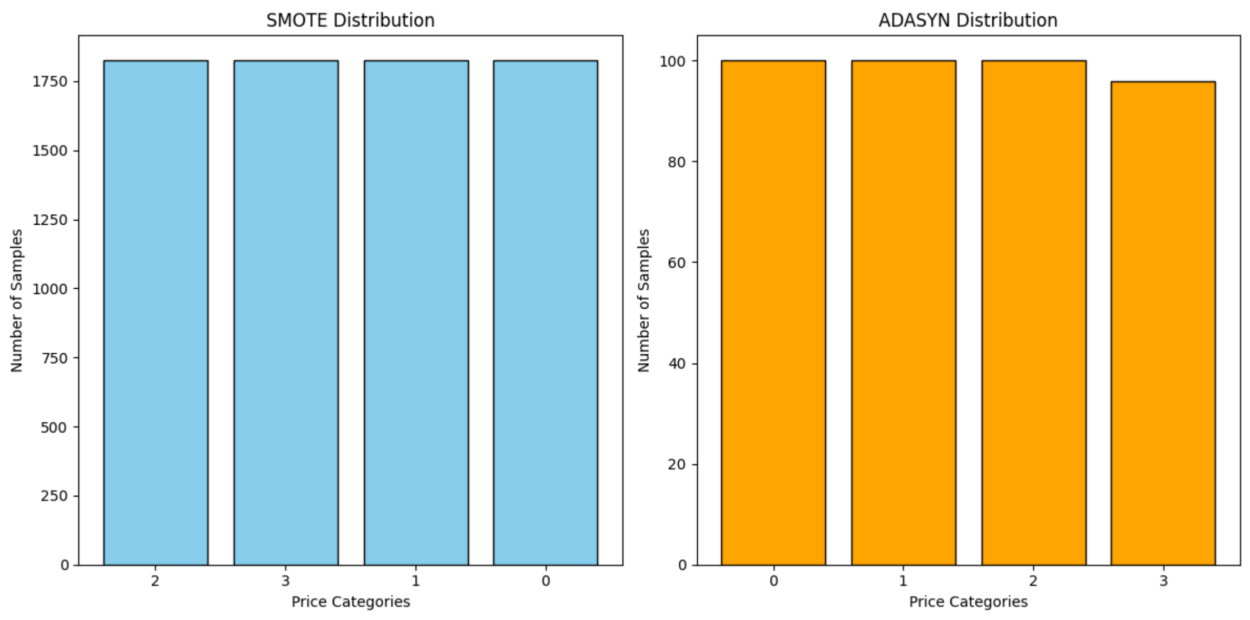
```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q5> python .\1.py
Original Training Set Distribution:
2    1824
0    1722
1    1391
3    1319
Name: count, dtype: int64

SMOTE Distribution:
2    1824
3    1824
1    1824
0    1824
Name: count, dtype: int64

Imbalanced Training Set Distribution (Before ADASYN):
0    100
1    100
2    100
3     50
Name: count, dtype: int64

ADASYN Distribution:
0    100
1    100
2    100
3     96
Name: count, dtype: int64

SMOTE Model Performance:
              precision    recall  f1-score   support

         Low       0.96      0.96      0.96       424
      Medium       0.98      0.98      0.98       376
        High       0.95      0.96      0.96       419
   Very High       0.98      0.97      0.97       345

    accuracy                           0.97      1564
   macro avg       0.97      0.97      0.97      1564
weighted avg       0.97      0.97      0.97      1564

SMOTE Accuracy: 0.97


ADASYN Model Performance:
              precision    recall  f1-score   support

         Low       0.83      0.88      0.85       424
      Medium       0.89      0.98      0.93       376
        High       0.88      0.79      0.83       419
   Very High       0.94      0.90      0.92       345

    accuracy                           0.88      1564
   macro avg       0.89      0.88      0.88      1564
weighted avg       0.88      0.88      0.88      1564
```

**SMOTE:**

- SMOTE generates synthetic samples by selecting examples that are close in the feature space, drawing a line between them, and creating new samples along that line.
- **Effectiveness**: SMOTE provides a more balanced dataset by generating new samples, which helps in improving the model's ability to learn from the minority class. In your case, the distribution after SMOTE is much more balanced across the price categories, with each category having nearly equal representation.
- **Impact on Model Performance**: The accuracy and performance metrics such as precision, recall, and F1-score indicate a good balance, with a macro accuracy of **0.97**, showing SMOTE's ability to effectively balance the classes while retaining the model’s performance.

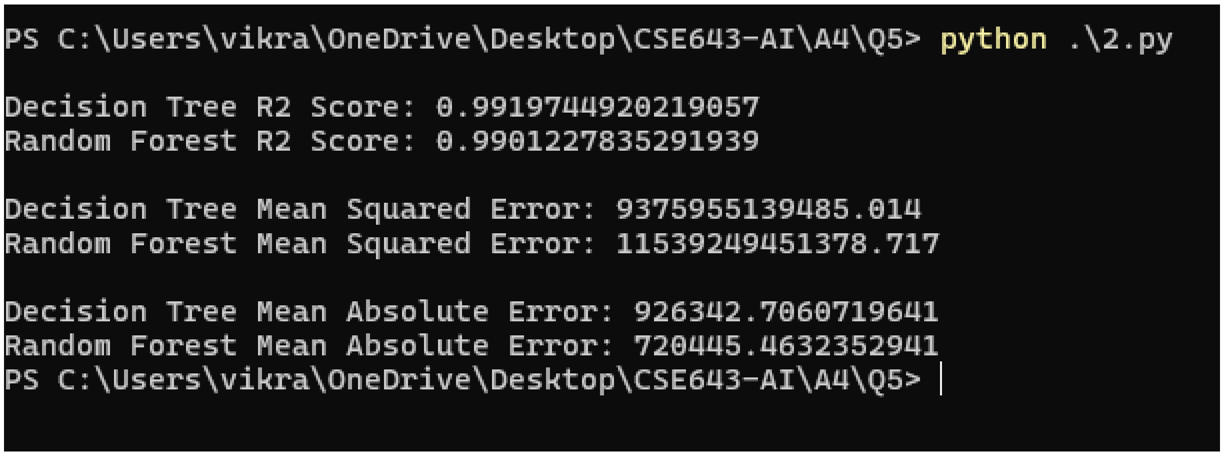
**ADASYN:**

- ADASYN focuses on generating synthetic data for those minority class examples that are harder to learn. This makes it adaptive and focused on the difficult-to-learn regions.
- **Effectiveness**: ADASYN performs similarly to SMOTE, but with a focus on the minority class examples that are harder to classify. In this case, you can see that the distribution across the price categories after ADASYN also becomes balanced, but with a slight difference in how the synthetic data is distributed.
- **Impact on Model Performance**: ADASYN tends to focus more on the hard-to-learn samples, which can lead to higher recall for those specific categories. However, its overall accuracy is slightly lower (**0.88**) compared to SMOTE (**0.97**). This difference indicates that while ADASYN can improve the learning for the minority class, it might introduce more complexity that could reduce overall accuracy.

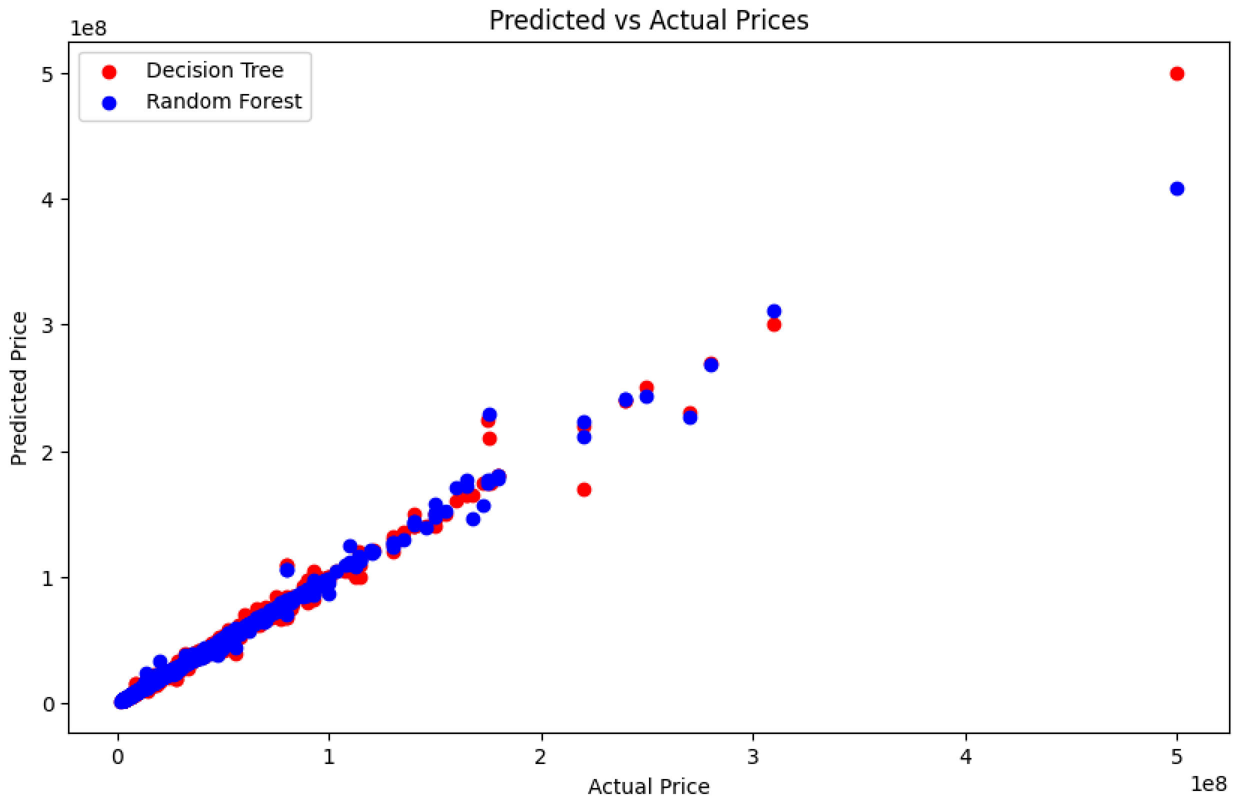


2.

## Task 2:



```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q5> python .\2.py

Decision Tree R2 Score: 0.9919744920219057
Random Forest R2 Score: 0.9901227835291939

Decision Tree Mean Squared Error: 9375955139485.014
Random Forest Mean Squared Error: 11539249451378.717

Decision Tree Mean Absolute Error: 926342.7060719641
Random Forest Mean Absolute Error: 720445.4632352941
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A4\Q5> 
```



**R² Scores**:

- **Decision Tree**: 0.99197
- **Random Forest**: 0.99012

Both models perform well, but the **Decision Tree** has a slightly higher R² score, indicating that it captures more variance in the target variable. However, the difference is marginal, suggesting both models are quite effective.
**Mean Squared Error (MSE)**:

- **Decision Tree**: 9.37595e+13
- **Random Forest**: 1.15392e+13

  The **Random Forest** model has a lower MSE, which indicates that it performs better in terms of minimizing the average squared errors between the actual and predicted values. This shows that Random Forest is less likely to have large errors compared to the single Decision Tree.
  **Mean Absolute Error (MAE)**:

- **Decision Tree**: 926342.7060791641
- **Random Forest**: 720445.4632352941

  The **Random Forest** also outperforms the Decision Tree in MAE, suggesting that on average, the Random Forest's predictions are closer to the actual values.
  **Visual Comparison**:

- In the scatter plot of predicted vs actual prices, **Random Forest** predictions (blue) tend to cluster more tightly around the diagonal line, indicating better accuracy. The **Decision Tree** predictions (red) are slightly more dispersed, showing that the Decision Tree's predictions are less consistent.

  **Overfitting**:

- **Decision Tree**: A single decision tree can easily overfit to the training data, especially if it's not pruned properly. It can learn the noise and specific patterns in the training data that don't generalize well to new data.
- **Random Forest**: An ensemble of decision trees reduces overfitting by averaging the predictions of multiple trees, each trained on a different subset of the data. This makes the model more robust and generalizable.

  **Interpretability**:

- **Decision Tree**: One of the main advantages of a decision tree is its interpretability. It provides a clear, visual representation of how decisions are made, which is useful for understanding the logic behind predictions.
- **Random Forest**: While Random Forest improves predictive power, it sacrifices interpretability. It's more challenging to explain the decision-making process because it involves many trees working together.

  **Performance**:

- **Decision Tree**: A single decision tree may be faster to train and make predictions, but its performance can be worse than an ensemble model like Random Forest due to overfitting.
- **Random Forest**: Random Forest generally provides better performance due to the averaging of multiple trees, but it requires more computational power, both in terms of

memory and training time. However, in this case, the performance of Random Forest is slightly better, as seen in the lower MSE and MAE.