price-prediction

September 9, 2024

[295]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1 Import raw House Data

1.1 Data understanding

[296]:
```python
data = pd.read_csv('/content/raw_house_data - raw_house_data.csv')
data
```

[296]:

	MLS	sold_price	zipcode	longitude	latitude	lot_acres
0	21530491	5300000.0	85637	-110.378200	31.356362	2154.00
1	21529082	4200000.0	85646	-111.045371	31.594213	1707.00
2	3054672	4200000.0	85646	-111.040707	31.594844	1707.00
3	21919321	4500000.0	85646	-111.035925	31.645878	636.67
4	21306357	3411450.0	85750	-110.813768	32.285162	3.21
…	…	…	…	…	…	…
4995	21810382	495000.0	85641	-110.661829	31.907917	4.98
4996	21908591	550000.0	85750	-110.858556	32.316373	1.42
4997	21832452	475000.0	85192	-110.755428	32.964708	12.06
4998	21900515	550000.0	85745	-111.055528	32.296871	1.01
4999	4111490	450000.0	85621	-110.913054	31.385259	4.16

	taxes	year_built	bedrooms	bathrooms	sqrt_ft	garage
0	5272.00	1941	13	10.0	10500.0	0.0
1	10422.36	1997	2	2.0	7300.0	0.0
2	10482.00	1997	2	3.0	NaN	NaN
3	8418.58	1930	7	5.0	9019.0	4.0
4	15393.00	1995	4	6.0	6396.0	3.0
…	…	…	…	…	…	…
4995	2017.00	2005	5	3.0	3601.0	3.0
4996	4822.01	1990	4	3.0	2318.0	3.0
4997	1000.00	1969	3	2.0	1772.0	0.0
4998	5822.93	2009	4	4.0	3724.0	3.0
4999	2814.48	1988	4	4.0	4317.0	NaN

```
                                    kitchen_features  fireplaces  \
0                Dishwasher, Freezer, Refrigerator, Oven         6.0
1                          Dishwasher, Garbage Disposal          5.0
2            Dishwasher, Garbage Disposal, Refrigerator          5.0
3      Dishwasher, Double Sink, Pantry: Butler, Refri…         4.0
4      Dishwasher, Garbage Disposal, Refrigerator, Mi…         5.0
…                                                   …           …
4995   Dishwasher, Double Sink, Garbage Disposal, Gas…         1.0
4996   Dishwasher, Double Sink, Electric Range, Garba…         1.0
4997   Dishwasher, Electric Range, Island, Refrigerat…         0.0
4998   Dishwasher, Double Sink, Garbage Disposal, Gas…         1.0
4999   Compactor, Dishwasher, Double Sink, Island, Ap…         3.0

                     floor_covering  HOA
0              Mexican Tile, Wood    0
1             Natural Stone, Other    0
2       Natural Stone, Other: Rock  NaN
3     Ceramic Tile, Laminate, Wood  NaN
4                  Carpet, Concrete   55
…                               …    …
4995          Carpet, Ceramic Tile  NaN
4996          Carpet, Ceramic Tile   43
4997                  Ceramic Tile  NaN
4998          Carpet, Ceramic Tile  NaN
4999          Carpet, Mexican Tile  NaN

[5000 rows x 16 columns]
```

[297]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   MLS               5000 non-null   int64
 1   sold_price        5000 non-null   float64
 2   zipcode           5000 non-null   int64
 3   longitude         5000 non-null   float64
 4   latitude          5000 non-null   float64
 5   lot_acres         4990 non-null   float64
 6   taxes             5000 non-null   float64
 7   year_built        5000 non-null   int64
 8   bedrooms          5000 non-null   int64
 9   bathrooms         4994 non-null   float64
 10  sqrt_ft           4944 non-null   float64
 11  garage            4993 non-null   float64
```

```
 12  kitchen_features  4967 non-null   object
 13  fireplaces        4975 non-null   float64
 14  floor_covering    4999 non-null   object
 15  HOA               4438 non-null   object
dtypes: float64(9), int64(4), object(3)
memory usage: 625.1+ KB
```

[298]:
```
data.shape
```

[298]:
```
(5000, 16)
```

[299]:
```
data.describe()
```

[299]:
```
                MLS    sold_price       zipcode    longitude     latitude  \
count  5.000000e+03  5.000000e+03   5000.000000  5000.000000  5000.000000
mean   2.127070e+07  7.746262e+05  85723.025600  -110.912107    32.308512
std    2.398508e+06  3.185556e+05     38.061712     0.120629     0.178028
min    3.042851e+06  1.690000e+05  85118.000000  -112.520168    31.356362
25%    2.140718e+07  5.850000e+05  85718.000000  -110.979260    32.277484
50%    2.161469e+07  6.750000e+05  85737.000000  -110.923420    32.318517
75%    2.180480e+07  8.350000e+05  85749.000000  -110.859078    32.394334
max    2.192856e+07  5.300000e+06  86323.000000  -109.454637    34.927884

         lot_acres         taxes  year_built     bedrooms    bathrooms  \
count  4990.000000  5.000000e+03  5000.00000  5000.000000  4994.000000
mean      4.661317  9.402828e+03  1992.32800     3.933800     3.829896
std      51.685230  1.729385e+05    65.48614     1.245362     1.387063
min       0.000000  0.000000e+00     0.00000     1.000000     1.000000
25%       0.580000  4.803605e+03  1987.00000     3.000000     3.000000
50%       0.990000  6.223760e+03  1999.00000     4.000000     4.000000
75%       1.757500  8.082830e+03  2006.00000     4.000000     4.000000
max    2154.000000  1.221508e+07  2019.00000    36.000000    36.000000

            sqrt_ft       garage   fireplaces
count   4944.000000  4993.000000  4975.000000
mean    3716.366828     2.816143     1.885226
std     1120.683515     1.192946     1.136578
min     1100.000000     0.000000     0.000000
25%     3047.000000     2.000000     1.000000
50%     3512.000000     3.000000     2.000000
75%     4130.250000     3.000000     3.000000
max    22408.000000    30.000000     9.000000
```

[300]:
```
data= data.drop_duplicates()
data.shape
## no duplicates
```

[300]: (5000, 16)

1.2 Handle Missing Values

[301]:
```
data.isnull().sum()
```

[301]:
```
MLS                   0
sold_price            0
zipcode               0
longitude             0
latitude              0
lot_acres            10
taxes                 0
year_built            0
bedrooms              0
bathrooms             6
sqrt_ft              56
garage                7
kitchen_features     33
fireplaces           25
floor_covering        1
HOA                 562
dtype: int64
```

[302]:
```
data['HOA'] = data['HOA'].fillna(0)
data['sqrt_ft'] = data['sqrt_ft'].fillna(data['sqrt_ft'].median())
data['lot_acres'] = data['lot_acres'].fillna(data['lot_acres'].median())
data['fireplaces'] = data['fireplaces'].fillna(data['fireplaces'].median())
data['garage'] = data['garage'].fillna(data['garage'].median())
data['bathrooms'] = data['bathrooms'].fillna(data['bathrooms'].median())
data['kitchen_features'] = data['kitchen_features'].
  ↪fillna(data['kitchen_features'].mode()[0])
data['floor_covering'] = data['floor_covering'].fillna(data['floor_covering'].
  ↪mode()[0])
data.isnull().sum()
```

[302]:
```
MLS                 0
sold_price          0
zipcode             0
longitude           0
latitude            0
lot_acres           0
taxes               0
year_built          0
bedrooms            0
bathrooms           0
sqrt_ft             0
```

```
garage              0
kitchen_features    0
fireplaces          0
floor_covering      0
HOA                 0
dtype: int64
```

[303]:

```
data
```

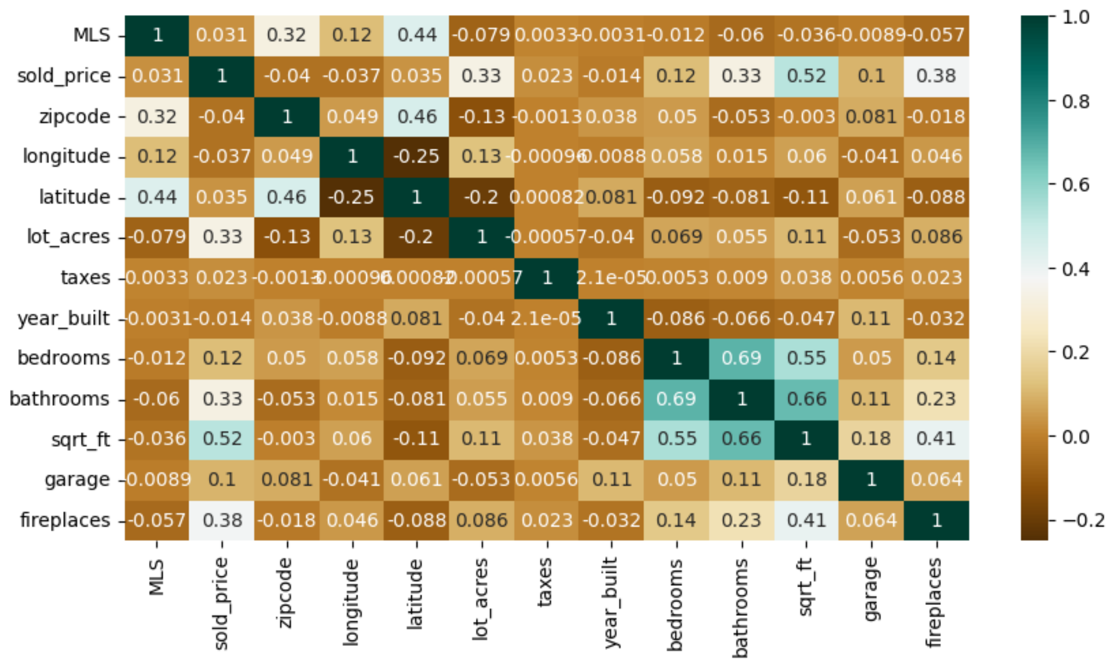
[303]:

```
           MLS  sold_price  zipcode   longitude   latitude  lot_acres  \
0     21530491   5300000.0    85637 -110.378200  31.356362    2154.00
1     21529082   4200000.0    85646 -111.045371  31.594213    1707.00
2      3054672   4200000.0    85646 -111.040707  31.594844    1707.00
3     21919321   4500000.0    85646 -111.035925  31.645878     636.67
4     21306357   3411450.0    85750 -110.813768  32.285162       3.21
…          …          …      …          …          …        …
4995  21810382    495000.0    85641 -110.661829  31.907917       4.98
4996  21908591    550000.0    85750 -110.858556  32.316373       1.42
4997  21832452    475000.0    85192 -110.755428  32.964708      12.06
4998  21900515    550000.0    85745 -111.055528  32.296871       1.01
4999   4111490    450000.0    85621 -110.913054  31.385259       4.16

         taxes  year_built  bedrooms  bathrooms  sqrt_ft  garage  \
0      5272.00        1941        13       10.0  10500.0     0.0
1     10422.36        1997         2        2.0   7300.0     0.0
2     10482.00        1997         2        3.0   3512.0     3.0
3      8418.58        1930         7        5.0   9019.0     4.0
4     15393.00        1995         4        6.0   6396.0     3.0
…          …          …        …        …        …
4995   2017.00        2005         5        3.0   3601.0     3.0
4996   4822.01        1990         4        3.0   2318.0     3.0
4997   1000.00        1969         3        2.0   1772.0     0.0
4998   5822.93        2009         4        4.0   3724.0     3.0
4999   2814.48        1988         4        4.0   4317.0     3.0

                                       kitchen_features  fireplaces  \
0                 Dishwasher, Freezer, Refrigerator, Oven         6.0
1                           Dishwasher, Garbage Disposal         5.0
2              Dishwasher, Garbage Disposal, Refrigerator         5.0
3     Dishwasher, Double Sink, Pantry: Butler, Refri…        4.0
4     Dishwasher, Garbage Disposal, Refrigerator, Mi…        5.0
…                                                   …          …
4995  Dishwasher, Double Sink, Garbage Disposal, Gas…        1.0
4996  Dishwasher, Double Sink, Electric Range, Garba…        1.0
4997  Dishwasher, Electric Range, Island, Refrigerat…        0.0
4998  Dishwasher, Double Sink, Garbage Disposal, Gas…        1.0
4999  Compactor, Dishwasher, Double Sink, Island, Ap…        3.0
```

```
                    floor_covering HOA
0               Mexican Tile, Wood   0
1              Natural Stone, Other   0
2         Natural Stone, Other: Rock   0
3       Ceramic Tile, Laminate, Wood   0
4                  Carpet, Concrete  55
…                               …  ..
4995           Carpet, Ceramic Tile   0
4996           Carpet, Ceramic Tile  43
4997                   Ceramic Tile   0
4998           Carpet, Ceramic Tile   0
4999           Carpet, Mexican Tile   0

[5000 rows x 16 columns]
```

[304]:
```python
## Corelation matrix
import seaborn as sns
plt.figure(figsize=(10, 5))
c = data.corr(numeric_only=True)
sns.heatmap(c, cmap='BrBG', annot=True)
```

[304]: <Axes: >



1.3 Other feature

[305]:
```
data['price_per_sqft'] = data['sold_price']/data['sqrt_ft']
data["category"] = data["price_per_sqft"]//500
print(data['category'].unique())
```

```
[1. 2. 0.]
```

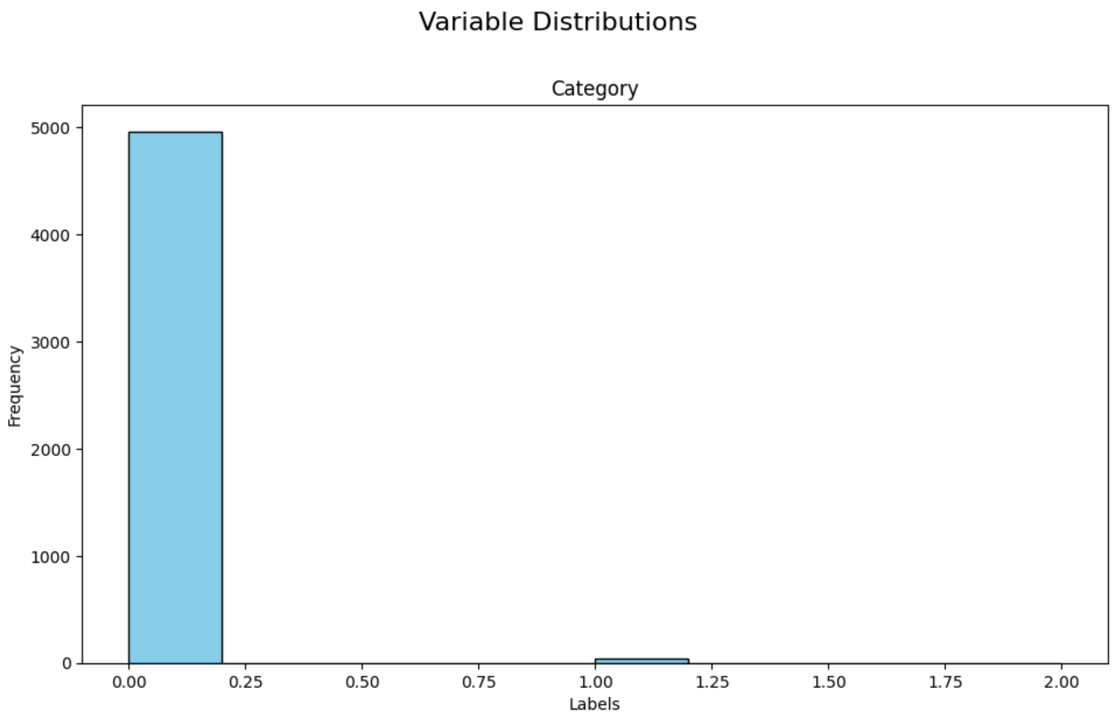
[383]:
```
price_range_per_category = data.groupby('category')['price_per_sqft'].
  ↪agg(['min', 'max'])
print(price_range_per_category)
```

```
                  min          max
category
0.0         24.544805   498.946668
1.0        501.543210   969.305331
2.0       1195.899772  1208.333333
```

[306]:
```
data['category'].value_counts()
```

[306]:
```
category
0.0    4960
1.0      38
2.0       2
Name: count, dtype: int64
```

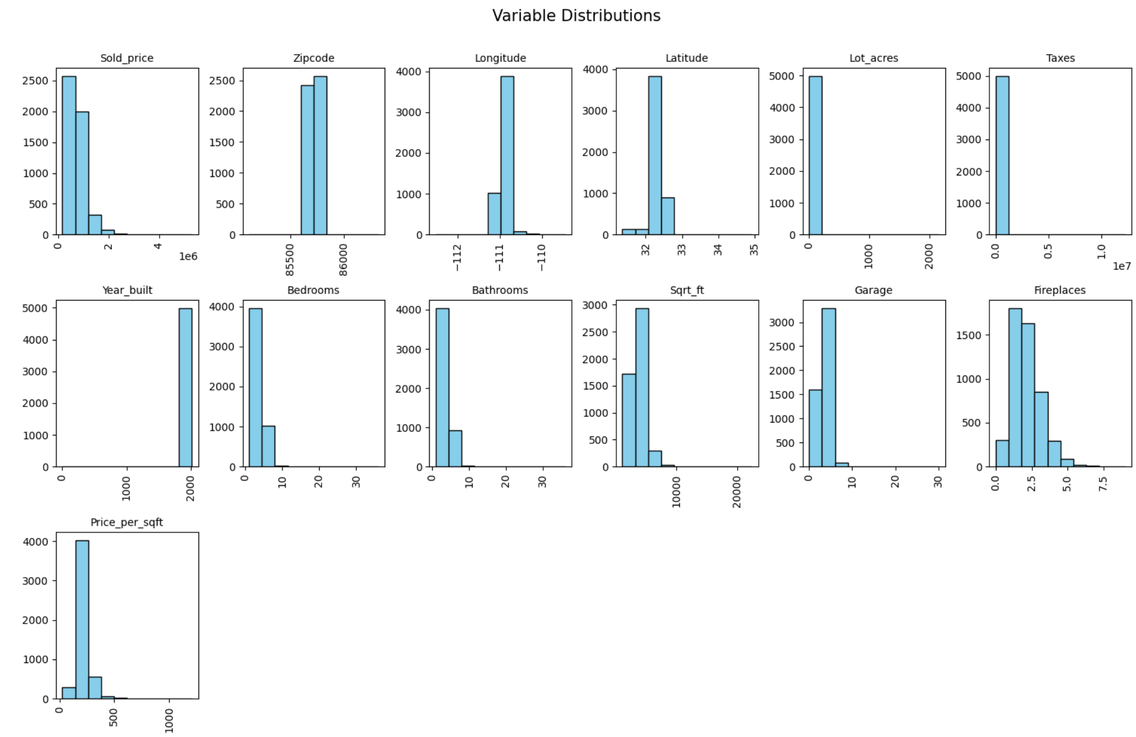
[307]:
```
# Function to plot variable distributions
def plot_variable_distributions(df):
    features = ['category']
    plt.figure(figsize=(10, 6))

    for i, feature in enumerate(features, 1):
        plt.subplot(1, 1, i)
        df[feature].hist(grid=False, color='skyblue', edgecolor='black')
        plt.title(feature.capitalize(), fontsize=12)
        plt.xlabel("Labels", fontsize=10)
        plt.ylabel("Frequency", fontsize=10)
        plt.xticks(rotation=0)

    plt.tight_layout(pad=2.0)
    plt.suptitle("Variable Distributions", fontsize=16, y=1.05)
    plt.show()

plot_variable_distributions(data)
```



1.4 Distribution

[308]:
```
columns_to_plot=['sold_price', 'zipcode', 'longitude', 'latitude', 'lot_acres',
       'taxes', 'year_built', 'bedrooms', 'bathrooms', 'sqrt_ft', 'garage',␣
 ↪'fireplaces', 'price_per_sqft']
# Function to plot variable distributions
def plot_variable_distributions(df):
    features = columns_to_plot
    plt.figure(figsize=(15, 15))

    for i, feature in enumerate(features, 1):
        plt.subplot(5, 6, i)  # Adjust the grid size as per the number of␣
 ↪features
        df[feature].hist(grid=False, color='skyblue', edgecolor='black')
        plt.title(feature.capitalize(), fontsize=10)
        plt.xlabel("")
        plt.ylabel("")
        plt.xticks(rotation=90)

    plt.tight_layout(pad=1)
    plt.suptitle("Variable Distributions", fontsize=15, y=1.03)
    plt.show()
```

```
# Call the function to plot histograms of the DataFrame excluding the first␣
↪column
plot_variable_distributions(data)
```



1.5 Label Encoding

[309]:
```
data = pd.get_dummies(data, columns=['kitchen_features', 'floor_covering'],␣
↪drop_first=True)
encoded_columns = data.filter(like='kitchen_features_').columns.union(data.
↪filter(like='floor_covering_').columns)
data[encoded_columns] = data[encoded_columns].astype(int)
data
```

[309]:
```
           MLS  sold_price  zipcode   longitude   latitude  lot_acres  \
0     21530491   5300000.0    85637 -110.378200  31.356362    2154.00
1     21529082   4200000.0    85646 -111.045371  31.594213    1707.00
2      3054672   4200000.0    85646 -111.040707  31.594844    1707.00
3     21919321   4500000.0    85646 -111.035925  31.645878     636.67
4     21306357   3411450.0    85750 -110.813768  32.285162       3.21
…          …          …        …          …          …        …
4995  21810382    495000.0    85641 -110.661829  31.907917       4.98
4996  21908591    550000.0    85750 -110.858556  32.316373       1.42
4997  21832452    475000.0    85192 -110.755428  32.964708      12.06
```

```
4998  21900515    550000.0    85745 -111.055528  32.296871        1.01
4999   4111490    450000.0    85621 -110.913054  31.385259        4.16

         taxes  year_built  bedrooms  bathrooms  …  \
0      5272.00        1941        13       10.0  …
1     10422.36        1997         2        2.0  …
2     10482.00        1997         2        3.0  …
3      8418.58        1930         7        5.0  …
4     15393.00        1995         4        6.0  …
…          …          …        …          …   …
4995   2017.00        2005         5        3.0  …
4996   4822.01        1990         4        3.0  …
4997   1000.00        1969         3        2.0  …
4998   5822.93        2009         4        4.0  …
4999   2814.48        1988         4        4.0  …

      floor_covering_Other: Travertine  floor_covering_Other: travertine  \
0                                    0                                 0
1                                    0                                 0
2                                    0                                 0
3                                    0                                 0
4                                    0                                 0
…                                  …                               …
4995                                 0                                 0
4996                                 0                                 0
4997                                 0                                 0
4998                                 0                                 0
4999                                 0                                 0

      floor_covering_Vinyl, Wood floor_covering_Wood  \
0                              0                   0
1                              0                   0
2                              0                   0
3                              0                   0
4                              0                   0
…                            …                   …
4995                           0                   0
4996                           0                   0
4997                           0                   0
4998                           0                   0
4999                           0                   0

      floor_covering_Wood, Other  floor_covering_Wood, Other: Lime Stone  \
0                              0                                       0
1                              0                                       0
2                              0                                       0
3                              0                                       0
```

```
4                           0                                      0
…                       …                                      …
4995                        0                                      0
4996                        0                                      0
4997                        0                                      0
4998                        0                                      0
4999                        0                                      0

      floor_covering_Wood, Other: Porcelain tile  \
0                                             0
1                                             0
2                                             0
3                                             0
4                                             0
…                                         …
4995                                          0
4996                                          0
4997                                          0
4998                                          0
4999                                          0

      floor_covering_Wood, Other: Travertine  \
0                                         0
1                                         0
2                                         0
3                                         0
4                                         0
…                                     …
4995                                      0
4996                                      0
4997                                      0
4998                                      0
4999                                      0

      floor_covering_Wood, Other: Travertine/Marble  \
0                                                0
1                                                0
2                                                0
3                                                0
4                                                0
…                                            …
4995                                             0
4996                                             0
4997                                             0
4998                                             0
4999                                             0
```

```
      floor_covering_Wood, Other: porcelain tile
0                                              0
1                                              0
2                                              0
3                                              0
4                                              0
…                                          …
4995                                           0
4996                                           0
4997                                           0
4998                                           0
4999                                           0

[5000 rows x 2195 columns]
```

1.6 Variable transformation

[310]:
```python
data['Lot_acres_log'] = np.log1p(data['lot_acres'])
data['Taxes_log'] = np.log1p(data['taxes'])
data['Bathrooms_log'] = np.log1p(data['bathrooms'])
data['Bedrooms_log'] = np.log1p(data['bedrooms'])
data['Garage_sqrt'] = np.sqrt(data['garage'])
data['Fireplaces_sqrt'] = np.sqrt(data['fireplaces'])
data['price_per_sqft_log'] = np.log1p(data['price_per_sqft'])

# columns_to_drop = ['lot_acres', 'taxes', 'bathrooms', 'garage', 'fireplaces']
# data = data.drop(columns=columns_to_drop)
data
```

```
<ipython-input-310-9a36597b6cb2>:1: PerformanceWarning: DataFrame is highly
fragmented.  This is usually the result of calling `frame.insert` many times,
which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe =
frame.copy()`
  data['Lot_acres_log'] = np.log1p(data['lot_acres'])
<ipython-input-310-9a36597b6cb2>:2: PerformanceWarning: DataFrame is highly
fragmented.  This is usually the result of calling `frame.insert` many times,
which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe =
frame.copy()`
  data['Taxes_log'] = np.log1p(data['taxes'])
<ipython-input-310-9a36597b6cb2>:3: PerformanceWarning: DataFrame is highly
fragmented.  This is usually the result of calling `frame.insert` many times,
which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe =
frame.copy()`
  data['Bathrooms_log'] = np.log1p(data['bathrooms'])
```

```
<ipython-input-310-9a36597b6cb2>:4: PerformanceWarning: DataFrame is highly
fragmented.  This is usually the result of calling `frame.insert` many times,
which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe =
frame.copy()`
  data['Bedrooms_log'] = np.log1p(data['bedrooms'])
<ipython-input-310-9a36597b6cb2>:5: PerformanceWarning: DataFrame is highly
fragmented.  This is usually the result of calling `frame.insert` many times,
which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe =
frame.copy()`
  data['Garage_sqrt'] = np.sqrt(data['garage'])
<ipython-input-310-9a36597b6cb2>:6: PerformanceWarning: DataFrame is highly
fragmented.  This is usually the result of calling `frame.insert` many times,
which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe =
frame.copy()`
  data['Fireplaces_sqrt'] = np.sqrt(data['fireplaces'])
<ipython-input-310-9a36597b6cb2>:7: PerformanceWarning: DataFrame is highly
fragmented.  This is usually the result of calling `frame.insert` many times,
which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe =
frame.copy()`
  data['price_per_sqft_log'] = np.log1p(data['price_per_sqft'])
```

[310]:

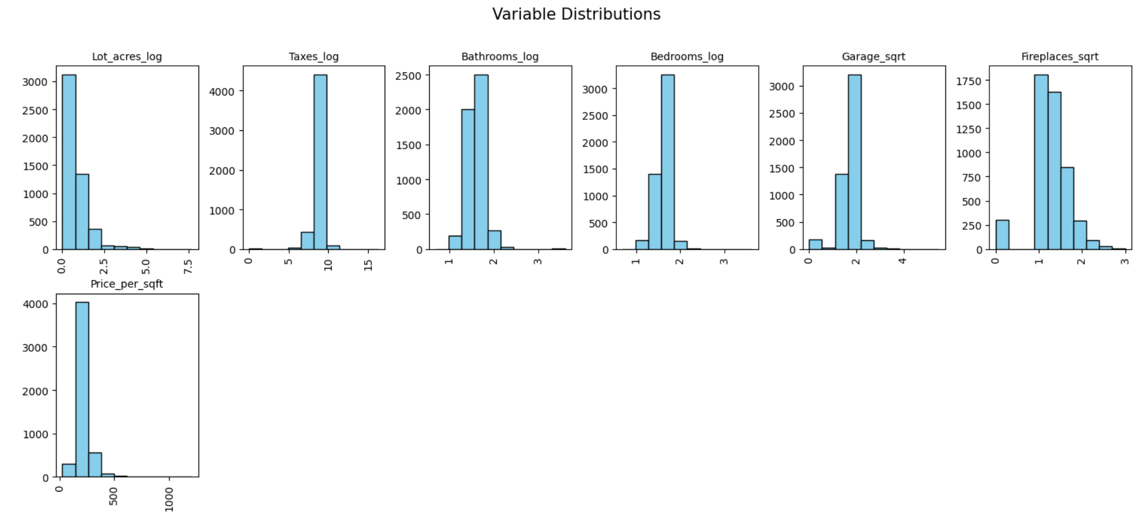
```
           MLS  sold_price  zipcode   longitude   latitude  lot_acres  \
0     21530491   5300000.0    85637 -110.378200  31.356362    2154.00
1     21529082   4200000.0    85646 -111.045371  31.594213    1707.00
2      3054672   4200000.0    85646 -111.040707  31.594844    1707.00
3     21919321   4500000.0    85646 -111.035925  31.645878     636.67
4     21306357   3411450.0    85750 -110.813768  32.285162       3.21
…          …          …      …           …          …         …
4995  21810382    495000.0    85641 -110.661829  31.907917       4.98
4996  21908591    550000.0    85750 -110.858556  32.316373       1.42
4997  21832452    475000.0    85192 -110.755428  32.964708      12.06
4998  21900515    550000.0    85745 -111.055528  32.296871       1.01
4999   4111490    450000.0    85621 -110.913054  31.385259       4.16

         taxes  year_built  bedrooms  bathroooms  …  \
0      5272.00        1941        13       10.0  …
1     10422.36        1997         2        2.0  …
2     10482.00        1997         2        3.0  …
3      8418.58        1930         7        5.0  …
4     15393.00        1995         4        6.0  …
…          …          …       …          …  …
4995   2017.00        2005         5        3.0  …
4996   4822.01        1990         4        3.0  …
```

```
4997  1000.00        1969          3        2.0  …
4998  5822.93        2009          4        4.0  …
4999  2814.48        1988          4        4.0  …

      floor_covering_Wood, Other: Travertine  \
0                                          0
1                                          0
2                                          0
3                                          0
4                                          0
…                                        …
4995                                       0
4996                                       0
4997                                       0
4998                                       0
4999                                       0

      floor_covering_Wood, Other: Travertine/Marble  \
0                                                 0
1                                                 0
2                                                 0
3                                                 0
4                                                 0
…                                               …
4995                                              0
4996                                              0
4997                                              0
4998                                              0
4999                                              0

      floor_covering_Wood, Other: porcelain tile Lot_acres_log  Taxes_log  \
0                                             0      7.675546   8.570355
1                                             0      7.443078   9.251805
2                                             0      7.443078   9.257510
3                                             0      6.457821   9.038315
4                                             0      1.437463   9.641733
…                                           …            …         …
4995                                          0      1.788421   7.609862
4996                                          0      0.883768   8.481153
4997                                          0      2.569554   6.908755
4998                                          0      0.698135   8.669731
4999                                          0      1.640937   7.942888

      Bathrooms_log  Bedrooms_log  Garage_sqrt  Fireplaces_sqrt  \
0          2.397895      2.639057     0.000000         2.449490
1          1.098612      1.098612     0.000000         2.236068
2          1.386294      1.098612     1.732051         2.236068
```

```
3          1.791759      2.079442     2.000000         2.000000
4          1.945910      1.609438     1.732051         2.236068
…             …            …          …             …
4995       1.386294      1.791759     1.732051         1.000000
4996       1.386294      1.609438     1.732051         1.000000
4997       1.098612      1.386294     0.000000         0.000000
4998       1.609438      1.609438     1.732051         1.000000
4999       1.609438      1.609438     1.732051         1.732051

      price_per_sqft_log
0               6.226066
1               6.356702
2               7.087490
3               6.214501
4               6.281093
…                    …
4995            4.930595
4996            5.473419
4997            5.594930
4998            5.001868
4999            4.656234

[5000 rows x 2202 columns]
```

[311]:

```python
columns_to_plot=['Lot_acres_log','Taxes_log', 'Bathrooms_log', 'Bedrooms_log',␣
↪'Garage_sqrt', 'Fireplaces_sqrt', 'price_per_sqft']
# Function to plot variable distributions
def plot_variable_distributions(df):
    features = columns_to_plot
    plt.figure(figsize=(15, 15))

    for i, feature in enumerate(features, 1):
        plt.subplot(5, 6, i)  # Adjust the grid size as per the number of␣
↪features
        df[feature].hist(grid=False, color='skyblue', edgecolor='black')
        plt.title(feature.capitalize(), fontsize=10)
        plt.xlabel("")
        plt.ylabel("")
        plt.xticks(rotation=90)

    plt.tight_layout(pad=1)
    plt.suptitle("Variable Distributions", fontsize=15, y=1.03)
    plt.show()

# Call the function to plot histograms of the DataFrame excluding the first␣
↪column
plot_variable_distributions(data)
```



[312]:
```
data.shape
```

[312]: (5000, 2202)

Outlier Detection and Treatment

[313]:
```
## convert to numeric
def convert_to_numeric(df):
    # Convert all columns to numeric, forcing errors to NaN
    df = df.apply(pd.to_numeric, errors='coerce').fillna(0)
    return df
data= convert_to_numeric(data)
data
```

[313]:
```
           MLS  sold_price  zipcode   longitude   latitude  lot_acres  \
0     21530491   5300000.0    85637 -110.378200  31.356362    2154.00
1     21529082   4200000.0    85646 -111.045371  31.594213    1707.00
2      3054672   4200000.0    85646 -111.040707  31.594844    1707.00
3     21919321   4500000.0    85646 -111.035925  31.645878     636.67
4     21306357   3411450.0    85750 -110.813768  32.285162       3.21
…          …          …      …         …         …        …
4995  21810382    495000.0    85641 -110.661829  31.907917       4.98
4996  21908591    550000.0    85750 -110.858556  32.316373       1.42
4997  21832452    475000.0    85192 -110.755428  32.964708      12.06
4998  21900515    550000.0    85745 -111.055528  32.296871       1.01
4999   4111490    450000.0    85621 -110.913054  31.385259       4.16

         taxes  year_built  bedrooms  bathrooms  …  \
0      5272.00        1941        13       10.0  …
1     10422.36        1997         2        2.0  …
```

```
2     10482.00        1997         2        3.0  …
3      8418.58        1930         7        5.0  …
4     15393.00        1995         4        6.0  …
…        …          …        …         …   …
4995   2017.00        2005         5        3.0  …
4996   4822.01        1990         4        3.0  …
4997   1000.00        1969         3        2.0  …
4998   5822.93        2009         4        4.0  …
4999   2814.48        1988         4        4.0  …

      floor_covering_Wood, Other: Travertine  \
0                                        0
1                                        0
2                                        0
3                                        0
4                                        0
…                                    …
4995                                     0
4996                                     0
4997                                     0
4998                                     0
4999                                     0

      floor_covering_Wood, Other: Travertine/Marble  \
0                                               0
1                                               0
2                                               0
3                                               0
4                                               0
…                                           …
4995                                            0
4996                                            0
4997                                            0
4998                                            0
4999                                            0

      floor_covering_Wood, Other: porcelain tile  Lot_acres_log  Taxes_log  \
0                                            0       7.675546   8.570355
1                                            0       7.443078   9.251805
2                                            0       7.443078   9.257510
3                                            0       6.457821   9.038315
4                                            0       1.437463   9.641733
…                                        …             …        …
4995                                         0       1.788421   7.609862
4996                                         0       0.883768   8.481153
4997                                         0       2.569554   6.908755
4998                                         0       0.698135   8.669731
```

```
4999                                             0      1.640937   7.942888

      Bathrooms_log  Bedrooms_log  Garage_sqrt  Fireplaces_sqrt  \
0          2.397895      2.639057     0.000000         2.449490
1          1.098612      1.098612     0.000000         2.236068
2          1.386294      1.098612     1.732051         2.236068
3          1.791759      2.079442     2.000000         2.000000
4          1.945910      1.609438     1.732051         2.236068
…               …             …            …                …
4995       1.386294      1.791759     1.732051         1.000000
4996       1.386294      1.609438     1.732051         1.000000
4997       1.098612      1.386294     0.000000         0.000000
4998       1.609438      1.609438     1.732051         1.000000
4999       1.609438      1.609438     1.732051         1.732051

      price_per_sqft_log
0               6.226066
1               6.356702
2               7.087490
3               6.214501
4               6.281093
…                    …
4995            4.930595
4996            5.473419
4997            5.594930
4998            5.001868
4999            4.656234

[5000 rows x 2202 columns]
```

[314]:
```python
# Count the number of NaN values in the "HOA" column
null_count = data['HOA'].isna().sum()
print(f"Number of null values in 'HOA': {null_count}")
```

```
Number of null values in 'HOA': 0
```

[315]:
```python
# plt.figure(figsize=(8, 6))
# plt.boxplot(data['Taxes_log'])
# plt.title('Box Plot for bathrooms')
# plt.ylabel('Values')
# plt.show()
```

[316]:
```python
def treat_outliers(df, features):
    df_filtered = df.copy()
    for column in features:
        Q1 = df_filtered[column].quantile(0.25)
        Q3 = df_filtered[column].quantile(0.75)
```

```
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df_filtered = df_filtered[(df_filtered[column] >= lower_bound) &␣
 ↪(df_filtered[column] <= upper_bound)]
    return df_filtered



# data['Lot_acres_log'] = np.log1p(data['lot_acres'])
# data['Taxes_log'] = np.log1p(data['taxes'])
# data['Bathrooms_log'] = np.log1p(data['bathrooms'])
# data['Bedrooms_log'] = np.log1p(data['bedrooms'])
# data['Garage_sqrt'] = np.sqrt(data['garage'])
# data['Fireplaces_sqrt'] = np.sqrt(data['fireplaces'])


n_features= ['sold_price', 'longitude', 'latitude', 'year_built', 'sqrt_ft',
'HOA', 'Lot_acres_log', 'Taxes_log', 'Bathrooms_log', 'Bedrooms_log',␣
 ↪'Garage_sqrt', 'Fireplaces_sqrt','price_per_sqft']
data_cleaned = treat_outliers(data, n_features)
data_cleaned
```

[316]:

```
           MLS  sold_price  zipcode   longitude   latitude  lot_acres  \
313   21510119   1000000.0    85755 -110.992170  32.458323       3.49
359   21125701   1200000.0    85750 -110.846659  32.326433       1.05
361   21317020   1150000.0    85658 -111.085082  32.464902       0.75
371   21305294   1200000.0    85718 -110.942288  32.347119       0.99
391   21408527   1165000.0    85718 -110.942544  32.348593       1.06
…          …           …      …          …          …        …
4989  21902512    545000.0    85745 -111.061493  32.306472       1.19
4993  21908358    565000.0    85750 -110.820216  32.307646       0.83
4994  21909379    535000.0    85718 -110.922291  32.317496       0.18
4996  21908591    550000.0    85750 -110.858556  32.316373       1.42
4998  21900515    550000.0    85745 -111.055528  32.296871       1.01

         taxes  year_built  bedrooms  bathrooms  …  \
313   14400.00        2005         3        4.0  …
359    9450.00        1999         4        3.0  …
361   13534.17        2007         4        5.0  …
371   12434.42        2002         3        4.0  …
391   13129.23        2005         3        3.0  …
…          …           …        …         …  …
4989   6326.96        2007         4        3.0  …
4993   4568.71        1986         4        3.0  …
4994   4414.00        2002         3        2.0  …
4996   4822.01        1990         4        3.0  …
```

```
4998   5822.93        2009         4        4.0  …

      floor_covering_Wood, Other: Travertine  \
313                                        0
359                                        0
361                                        0
371                                        0
391                                        0
…                                        …
4989                                       0
4993                                       0
4994                                       0
4996                                       0
4998                                       0

      floor_covering_Wood, Other: Travertine/Marble  \
313                                               0
359                                               0
361                                               0
371                                               0
391                                               0
…                                               …
4989                                              0
4993                                              0
4994                                              0
4996                                              0
4998                                              0

      floor_covering_Wood, Other: porcelain tile  Lot_acres_log  Taxes_log  \
313                                            0       1.501853   9.575053
359                                            0       0.717840   9.153876
361                                            0       0.559616   9.513047
371                                            0       0.688135   9.428304
391                                            0       0.722706   9.482672
…                                            …             …         …
4989                                           0       0.783902   8.752733
4993                                           0       0.604316   8.427205
4994                                           0       0.165514   8.392763
4996                                           0       0.883768   8.481153
4998                                           0       0.698135   8.669731

      Bathrooms_log  Bedrooms_log  Garage_sqrt  Fireplaces_sqrt  \
313        1.609438      1.386294     1.732051         1.732051
359        1.386294      1.609438     1.732051         1.414214
361        1.791759      1.609438     1.732051         1.732051
371        1.609438      1.386294     1.732051         1.414214
391        1.386294      1.386294     1.732051         1.732051
```

```
…          …          …          …          …
4989       1.386294   1.609438   2.000000      1.000000
4993       1.386294   1.609438   1.414214      1.414214
4994       1.098612   1.386294   1.414214      1.000000
4996       1.386294   1.609438   1.732051      1.000000
4998       1.609438   1.609438   1.732051      1.000000

      price_per_sqft_log
313             5.485913
359             5.665869
361             5.684828
371             5.632955
391             5.586044
…                    …
4989            4.993865
4993            5.307541
4994            5.541405
4996            5.473419
4998            5.001868

[2934 rows x 2202 columns]
```

[317]:

```
# plt.figure(figsize=(8, 6))
# plt.boxplot(data['Taxes_log'])
# plt.title('Box Plot for bathrooms')
# plt.ylabel('Values')
# plt.show()
```

1.7 Scaling

[318]:

```
# def min_max_scale(df, exclude_column):
#     df_scaled = df.copy()
#     for column in df_scaled.columns:
#         if column != exclude_column:
#             min_value = df_scaled[column].min()
#             max_value = df_scaled[column].max()
#             df_scaled[column] = (df_scaled[column] - min_value) / (max_value␣
↪- min_value)
#     return df_scaled
# data_scaled = min_max_scale(data, 'sold_price')
# data_scaled
```

[319]:

```
# def min_max_scale(df, exclude_columns):
#     df_scaled = df.copy()
#     for column in df_scaled.columns:
#         if column not in exclude_columns:
```

```
#               min_value = df_scaled[column].min()
#               max_value = df_scaled[column].max()
#               df_scaled[column] = (df_scaled[column] - min_value) / (max_value␣
↪- min_value)
#       return df_scaled
# exclude_columns = ['MLS','sold_price', 'category']
# #exclude_columns=['MLS','sold_price', 'category','longitude', 'latitude',␣
↪'lot_acres', 'sqrt_ft', 'bedrooms', 'bathrooms', 'garage', 'fireplaces']
# data_scaled = min_max_scale(data, exclude_columns)
# data_scaled
```

[320]:
```
def min_max_scale(df, exclude_columns):
    df_scaled = df.copy()
    scaling_params = {}
    for column in df_scaled.columns:
        if column not in exclude_columns:
            min_value = df_scaled[column].min()
            max_value = df_scaled[column].max()
            df_scaled[column] = (df_scaled[column] - min_value) / (max_value -␣
↪min_value)
            scaling_params[column] = {'min': min_value, 'max': max_value}
    return df_scaled, scaling_params
exclude_columns = ['MLS', 'sold_price', 'category']
data_scaled, scaling_params = min_max_scale(data, exclude_columns)
```

[321]:
```
data_scaled.shape
```

[321]:
```
(5000, 2202)
```

[322]:
```
data_scaled.isnull().sum()
```

[322]:
```
MLS                   0
sold_price            0
zipcode               0
longitude             0
latitude              0
                     ..
Bathrooms_log         0
Bedrooms_log          0
Garage_sqrt           0
Fireplaces_sqrt       0
price_per_sqft_log    0
Length: 2202, dtype: int64
```

##Model 1

[323]:
```
model_data = data_scaled.copy()
model_data
```

[323]:
```
           MLS  sold_price   zipcode  longitude  latitude  lot_acres  \
0     21530491   5300000.0  0.430705   0.698727  0.000000   1.000000
1     21529082   4200000.0  0.438174   0.481090  0.066597   0.792479
2      3054672   4200000.0  0.438174   0.482612  0.066773   0.792479
3     21919321   4500000.0  0.438174   0.484172  0.081062   0.295576
4     21306357   3411450.0  0.524481   0.556641  0.260057   0.001490
…          …         …        …         …        …          …
4995  21810382    495000.0  0.434025   0.606205  0.154431   0.002312
4996  21908591    550000.0  0.524481   0.542031  0.268796   0.000659
4997  21832452    475000.0  0.061411   0.575672  0.450325   0.005599
4998  21900515    550000.0  0.520332   0.477777  0.263336   0.000469
4999   4111490    450000.0  0.417427   0.524253  0.008091   0.001931

         taxes  year_built  bedrooms  bathrooms  …  \
0     0.000432    0.961367  0.342857   0.257143  …
1     0.000853    0.989104  0.028571   0.028571  …
2     0.000858    0.989104  0.028571   0.057143  …
3     0.000689    0.955919  0.171429   0.114286  …
4     0.001260    0.988113  0.085714   0.142857  …
…          …         …        …         …   …
4995  0.000165    0.993066  0.114286   0.057143  …
4996  0.000395    0.985636  0.085714   0.057143  …
4997  0.000082    0.975235  0.057143   0.028571  …
4998  0.000477    0.995047  0.085714   0.085714  …
4999  0.000230    0.984646  0.085714   0.085714  …

      floor_covering_Wood, Other: Travertine  \
0                                        0.0
1                                        0.0
2                                        0.0
3                                        0.0
4                                        0.0
…                                         …
4995                                     0.0
4996                                     0.0
4997                                     0.0
4998                                     0.0
4999                                     0.0

      floor_covering_Wood, Other: Travertine/Marble  \
0                                               0.0
1                                               0.0
2                                               0.0
3                                               0.0
```

4	0.0
…	…
4995	0.0
4996	0.0
4997	0.0
4998	0.0
4999	0.0

	floor_covering_Wood, Other: porcelain tile	Lot_acres_log	Taxes_log	\
0	0.0	1.000000	0.525203	
1	0.0	0.969713	0.566963	
2	0.0	0.969713	0.567313	
3	0.0	0.841350	0.553880	
4	0.0	0.187278	0.590858	
…	…	…	…	
4995	0.0	0.233002	0.466343	
4996	0.0	0.115141	0.519736	
4997	0.0	0.334772	0.423378	
4998	0.0	0.090956	0.531293	
4999	0.0	0.213788	0.486751	

	Bathrooms_log	Bedrooms_log	Garage_sqrt	Fireplaces_sqrt	\
0	0.584264	0.666917	0.000000	0.816497	
1	0.138964	0.138964	0.000000	0.745356	
2	0.237561	0.138964	0.316228	0.745356	
3	0.376525	0.475121	0.365148	0.666667	
4	0.429356	0.314038	0.316228	0.745356	
…	…	…	…	…	
4995	0.237561	0.376525	0.316228	0.333333	
4996	0.237561	0.314038	0.316228	0.333333	
4997	0.138964	0.237561	0.000000	0.000000	
4998	0.314038	0.314038	0.316228	0.333333	
4999	0.314038	0.314038	0.316228	0.577350	

	price_per_sqft_log
0	0.774003
1	0.807869
2	0.997321
3	0.771005
4	0.788268
…	…
4995	0.438162
4996	0.578885
4997	0.610386
4998	0.456639
4999	0.367036

```
[5000 rows x 2202 columns]
```

1.8 Select featutes

[324]:
```
# ## Testing on category
# import statsmodels.api as sm
# X_features_cat= model_data.drop(['category', 'sold_price', 'MLS'], axis=1)
# y_features_cat= model_data['category']

# def forward_regression(X, y,
#                        threshold_in,
#                        verbose=True):
#     initial_list = []
#     included = list(initial_list)
#     model=sm.OLS(X,y)
#     while True:
#         changed=False
#         excluded = list(set(X.columns)-set(included))
#         new_pval = pd.Series(index=excluded)
#         for new_column in excluded:
#             model = sm.OLS(y, sm.add_constant(pd.
↪DataFrame(X[included+[new_column]]))).fit()
#             new_pval[new_column] = model.pvalues[new_column]
#         best_pval = new_pval.min()
#         if best_pval < threshold_in:
#             best_feature = new_pval.idxmin()
#             included.append(best_feature)
#             changed=True
#             if verbose:
#                 print('Add  {:30} with p-value {:.6}'.format(best_feature,␣
↪best_pval))

#         if not changed:
#             break

#     return included



# model=forward_regression(X_features_cat,y_features_cat,0.05)
# print(f'Useful predictors are :{model}')
```

[325]:
```
#X_features_cat
```

1.9 Sampling

[326]:
```python
majority_class = model_data[model_data['category'] == 0.0]
minority_class_1 = model_data[model_data['category'] == 1.0]
minority_class_2 = model_data[model_data['category'] == 2.0]
model_data['category'].value_counts()
```

[326]:
```
category
0.0    4960
1.0      38
2.0       2
Name: count, dtype: int64
```

[327]:
```python
oversampled_minority_class_1 = minority_class_1.sample(n=len(majority_class),␣
↪replace=True, random_state=42)
oversampled_minority_class_2 = minority_class_2.sample(n=len(majority_class),␣
↪replace=True, random_state=42)
balanced_model_data = pd.concat([majority_class, oversampled_minority_class_1,␣
↪oversampled_minority_class_2])
balanced_model_data = balanced_model_data.sample(frac=1, random_state=42)

balanced_model_data['category'].value_counts()
```

[327]:
```
category
2.0    4960
0.0    4960
1.0    4960
Name: count, dtype: int64
```

[328]:
```python
balanced_model_data
```

[328]:
```
           MLS  sold_price   zipcode  longitude  latitude  lot_acres  \
2      3054672   4200000.0  0.438174   0.482612  0.066773   0.792479
3647   3061363    606000.0  0.411618   0.505405  0.136145   0.000715
2      3054672   4200000.0  0.438174   0.482612  0.066773   0.792479
2      3054672   4200000.0  0.438174   0.482612  0.066773   0.792479
4906  21211486    498000.0  0.523651   0.566979  0.258002   0.000455
…          …           …         …          …         …          …
76    21731870   1700000.0  0.523651   0.574165  0.247111   0.008435
279   21231400   1450000.0  0.419917   0.608654  0.020774   0.102136
693   21224223   1073942.0  0.448133   0.463122  0.311589   0.000135
896   21428632    950000.0  0.497925   0.517280  0.277182   0.000395
53    21424173   2150000.0  0.515353   0.533842  0.322932   0.000000

         taxes  year_built  bedrooms  bathrooms  …  \
2     0.000858    0.989104  0.028571   0.057143  …
3647  0.000165    0.992075  0.057143   0.057143  …
```

```
2     0.000858    0.989104  0.028571   0.057143  …
2     0.000858    0.989104  0.028571   0.057143  …
4906  0.000426    0.987122  0.057143   0.057143  …
…        …           …        …          …   …
76    0.000704    0.967311  0.085714   0.085714  …
279   0.000098    0.980684  0.000000   0.000000  …
693   0.000069    0.997028  0.028571   0.057143  …
896   0.000299    0.997028  0.085714   0.085714  …
53    0.001560    0.994552  0.028571   0.057143  …

      floor_covering_Wood, Other: Travertine  \
2                                        0.0
3647                                     0.0
2                                        0.0
2                                        0.0
4906                                     0.0
…                                      …
76                                       0.0
279                                      0.0
693                                      0.0
896                                      0.0
53                                       0.0

      floor_covering_Wood, Other: Travertine/Marble  \
2                                               0.0
3647                                            0.0
2                                               0.0
2                                               0.0
4906                                            0.0
…                                             …
76                                              0.0
279                                             0.0
693                                             0.0
896                                             0.0
53                                              0.0

      floor_covering_Wood, Other: porcelain tile  Lot_acres_log  Taxes_log  \
2                                            0.0       0.969713   0.567313
3647                                         0.0       0.121446   0.466312
2                                            0.0       0.969713   0.567313
2                                            0.0       0.969713   0.567313
4906                                         0.0       0.088997   0.524396
…                                          …             …          …
76                                           0.0       0.384773   0.555205
279                                          0.0       0.703294   0.434540
693                                          0.0       0.033176   0.413081
896                                          0.0       0.080149   0.502739
```

```
53                                                0.0       0.000000    0.603955

      Bathrooms_log  Bedrooms_log  Garage_sqrt  Fireplaces_sqrt  \
2          0.237561      0.138964     0.316228         0.745356
3647       0.237561      0.237561     0.316228         0.471405
2          0.237561      0.138964     0.316228         0.745356
2          0.237561      0.138964     0.316228         0.745356
4906       0.237561      0.237561     0.316228         0.471405
…               …             …           …               …
76         0.314038      0.314038     0.000000         0.471405
279        0.000000      0.000000     0.258199         0.333333
693        0.237561      0.138964     0.258199         0.471405
896        0.314038      0.314038     0.316228         0.333333
53         0.237561      0.138964     0.000000         0.471405

      price_per_sqft_log
2               0.997321
3647            0.496719
2               0.997321
2               0.997321
4906            0.438291
…                    …
76              0.863317
279             1.000000
693             0.791956
896             0.624234
53              0.889234

[14880 rows x 2202 columns]
```

[329]:
```
model_data=balanced_model_data
model_data
```

[329]:
```
           MLS  sold_price   zipcode  longitude  latitude  lot_acres  \
2      3054672   4200000.0  0.438174   0.482612  0.066773   0.792479
3647   3061363    606000.0  0.411618   0.505405  0.136145   0.000715
2      3054672   4200000.0  0.438174   0.482612  0.066773   0.792479
2      3054672   4200000.0  0.438174   0.482612  0.066773   0.792479
4906  21211486    498000.0  0.523651   0.566979  0.258002   0.000455
…          …          …         …          …         …          …
76    21731870   1700000.0  0.523651   0.574165  0.247111   0.008435
279   21231400   1450000.0  0.419917   0.608654  0.020774   0.102136
693   21224223   1073942.0  0.448133   0.463122  0.311589   0.000135
896   21428632    950000.0  0.497925   0.517280  0.277182   0.000395
53    21424173   2150000.0  0.515353   0.533842  0.322932   0.000000

         taxes  year_built  bedrooms  bathrooms  …  \
```

```
2     0.000858    0.989104  0.028571   0.057143  …
3647  0.000165    0.992075  0.057143   0.057143  …
2     0.000858    0.989104  0.028571   0.057143  …
2     0.000858    0.989104  0.028571   0.057143  …
4906  0.000426    0.987122  0.057143   0.057143  …
…        …          …         …         …    …
76    0.000704    0.967311  0.085714   0.085714  …
279   0.000098    0.980684  0.000000   0.000000  …
693   0.000069    0.997028  0.028571   0.057143  …
896   0.000299    0.997028  0.085714   0.085714  …
53    0.001560    0.994552  0.028571   0.057143  …

      floor_covering_Wood, Other: Travertine  \
2                                        0.0
3647                                     0.0
2                                        0.0
2                                        0.0
4906                                     0.0
…                                      …
76                                       0.0
279                                      0.0
693                                      0.0
896                                      0.0
53                                       0.0

      floor_covering_Wood, Other: Travertine/Marble  \
2                                               0.0
3647                                            0.0
2                                               0.0
2                                               0.0
4906                                            0.0
…                                             …
76                                              0.0
279                                             0.0
693                                             0.0
896                                             0.0
53                                              0.0

      floor_covering_Wood, Other: porcelain tile  Lot_acres_log  Taxes_log  \
2                                            0.0       0.969713   0.567313
3647                                         0.0       0.121446   0.466312
2                                            0.0       0.969713   0.567313
2                                            0.0       0.969713   0.567313
4906                                         0.0       0.088997   0.524396
…                                          …             …          …
76                                           0.0       0.384773   0.555205
279                                          0.0       0.703294   0.434540
```

```
693                                          0.0      0.033176   0.413081
896                                          0.0      0.080149   0.502739
53                                           0.0      0.000000   0.603955

      Bathrooms_log  Bedrooms_log  Garage_sqrt  Fireplaces_sqrt  \
2          0.237561      0.138964     0.316228         0.745356
3647       0.237561      0.237561     0.316228         0.471405
2          0.237561      0.138964     0.316228         0.745356
2          0.237561      0.138964     0.316228         0.745356
4906       0.237561      0.237561     0.316228         0.471405
…               …             …           …                …
76         0.314038      0.314038     0.000000         0.471405
279        0.000000      0.000000     0.258199         0.333333
693        0.237561      0.138964     0.258199         0.471405
896        0.314038      0.314038     0.316228         0.333333
53         0.237561      0.138964     0.000000         0.471405

      price_per_sqft_log
2               0.997321
3647            0.496719
2               0.997321
2               0.997321
4906            0.438291
…                    …
76              0.863317
279             1.000000
693             0.791956
896             0.624234
53              0.889234

[14880 rows x 2202 columns]
```

1.10 Train test split

[330]:
```
train_size = int(0.8 * len(model_data))
train_df = model_data[:train_size]
test_df = model_data[train_size:]

# X_train_model1 = train_df.drop(['category', 'sold_price', 'MLS'], axis=1)
X_train_model1 = train_df[['longitude', 'latitude', 'lot_acres', 'sqrt_ft',␣
 ↪'bedrooms', 'bathrooms','garage', 'fireplaces']]
y_train_model1 = train_df['category']

# X_test_model1 = test_df.drop(['category', 'sold_price', 'MLS'], axis=1)
X_test_model1 = test_df[['longitude', 'latitude', 'lot_acres', 'sqrt_ft',␣
 ↪'bedrooms', 'bathrooms', 'garage', 'fireplaces']]
```

```python
y_test_model1 = test_df['category']

len(X_train_model1), len(y_train_model1), len(X_test_model1), len(y_test_model1)
```

[330]: (11904, 11904, 2976, 2976)

[331]:
```python
#y_train_model1.values.counts()
```

[332]:
```python
X_train_model1= X_train_model1.to_numpy()
X_test_model1=X_test_model1.to_numpy()
X_train_model1
```

[332]:
```
array([[4.82611658e-01, 6.67732132e-02, 7.92479109e-01, …,
        5.71428571e-02, 1.00000000e-01, 5.55555556e-01],
       [5.05405426e-01, 1.36144758e-01, 7.14948932e-04, …,
        5.71428571e-02, 1.00000000e-01, 2.22222222e-01],
       [4.82611658e-01, 6.67732132e-02, 7.92479109e-01, …,
        5.71428571e-02, 1.00000000e-01, 5.55555556e-01],
       …,
       [5.56640921e-01, 2.60057197e-01, 1.49025070e-03, …,
        1.42857143e-01, 1.00000000e-01, 5.55555556e-01],
       [5.43842160e-01, 2.60086036e-01, 7.89229341e-05, …,
        2.85714286e-02, 6.66666667e-02, 1.11111111e-01],
       [4.82611658e-01, 6.67732132e-02, 7.92479109e-01, …,
        5.71428571e-02, 1.00000000e-01, 5.55555556e-01]])
```

[333]:
```python
y_train_model1=y_train_model1.to_numpy().astype('int')
y_test_model1=y_test_model1.to_numpy().astype('int')
y_train_model1
```

[333]: array([2, 0, 2, …, 1, 0, 2])

[334]:
```python
train_df['category'].value_counts()
```

[334]:
```
category
0.0    4000
1.0    3967
2.0    3937
Name: count, dtype: int64
```

[335]:
```python
def accuracy(y, y_hat):
  return np.mean(y==y_hat)
```

1.11 Naive Bayes

[336]:
```
from scipy.stats import multivariate_normal as mvn
class GausSNB():

  def fit(self, X, y, epsilon = 1e-3):
    self.likelihoods= dict()
    self.priors= dict()
    self.K= set(y.astype(int))
    for k in self.K:
      X_k= X[y==k]
      # Naive Assumption: Observations as linearly independent of each other
      self.likelihoods[k]= {"mean": X_k.mean(axis=0), "cov": X_k.var(axis=0) +␣
↪epsilon}
      self.priors[k]= len(X_k) / len(X)

  def predict(self, X):
    N, D = X.shape
    p_hat = np.zeros((N, len(self.K)))

    for k, l in self.likelihoods.items():
      p_hat[:, k] = mvn.logpdf(X, l["mean"], l["cov"]) + np.log(self.priors[k])

    return p_hat.argmax(axis=1)
```

[337]:
```
NaiveBayes = GausSNB()
NaiveBayes.fit(X_train_model1, y_train_model1)
```

[338]:
```
y_hat_NaiveBayes = NaiveBayes.predict(X_test_model1)
y_hat_NaiveBayes
```

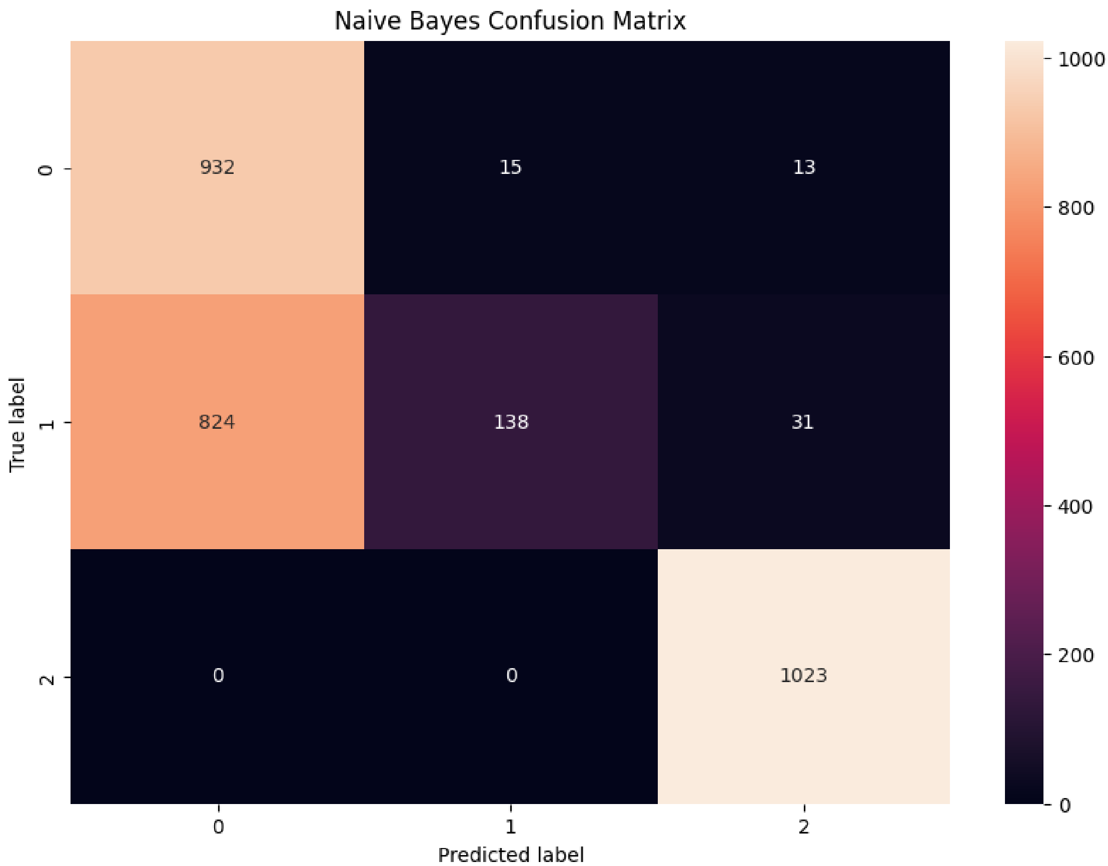
[338]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[339]:
```
accuracy(y_test_model1, y_hat_NaiveBayes)
```

[339]:
```
0.7032930107526881
```

[340]:
```
plt.figure(figsize=(10,7))
y_actu_NaiveBayes = pd.Series(y_test_model1, name='Actual')
y_pred_NaiveBayes = pd.Series(y_hat_NaiveBayes, name='Predicted')
cm = pd.crosstab(y_actu_NaiveBayes, y_pred_NaiveBayes)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.title("Naive Bayes Confusion Matrix")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

[340]:
```
Text(0.5, 47.7222222222222, 'Predicted label')
```



1.12 Gaussian Naive Bayes

[341]:
```
class GaussBayes:

  def fit(self, X, y, epsilon=1e-3):
    self.likelihoods= dict()
    self.priors= dict()
    self.K= set(y.astype(int))

    for k in self.K:
      X_k= X[y==k]
      N_k, D =X_k.shape
      mu_k = X_k.mean(axis=0)

      self.likelihoods[k]= {"mean": X_k.mean(axis=0), "cov": (1/(N_k-1)) * np.
↪matmul((X_k - mu_k).T, X_k - mu_k) + epsilon*np.identity(D)}
      self.priors[k]= len(X_k) / len(X)

  def predict(self, X):
```

```
    N, D= X.shape
    P_hat= np.zeros((N, len(self.K)))
    for k, l in self.likelihoods.items():
      P_hat[:,k]= mvn.logpdf(X, l["mean"], l["cov"]) + np.log(self.priors[k])
    return P_hat.argmax(axis=1)
```

[342]:
```
GaussianNaiveBayes = GaussBayes()
GaussianNaiveBayes.fit(X_train_model1, y_train_model1)
```

[343]:
```
y_hat_GaussianNaiveBayes= GaussianNaiveBayes.predict(X_test_model1)
y_hat_GaussianNaiveBayes
```

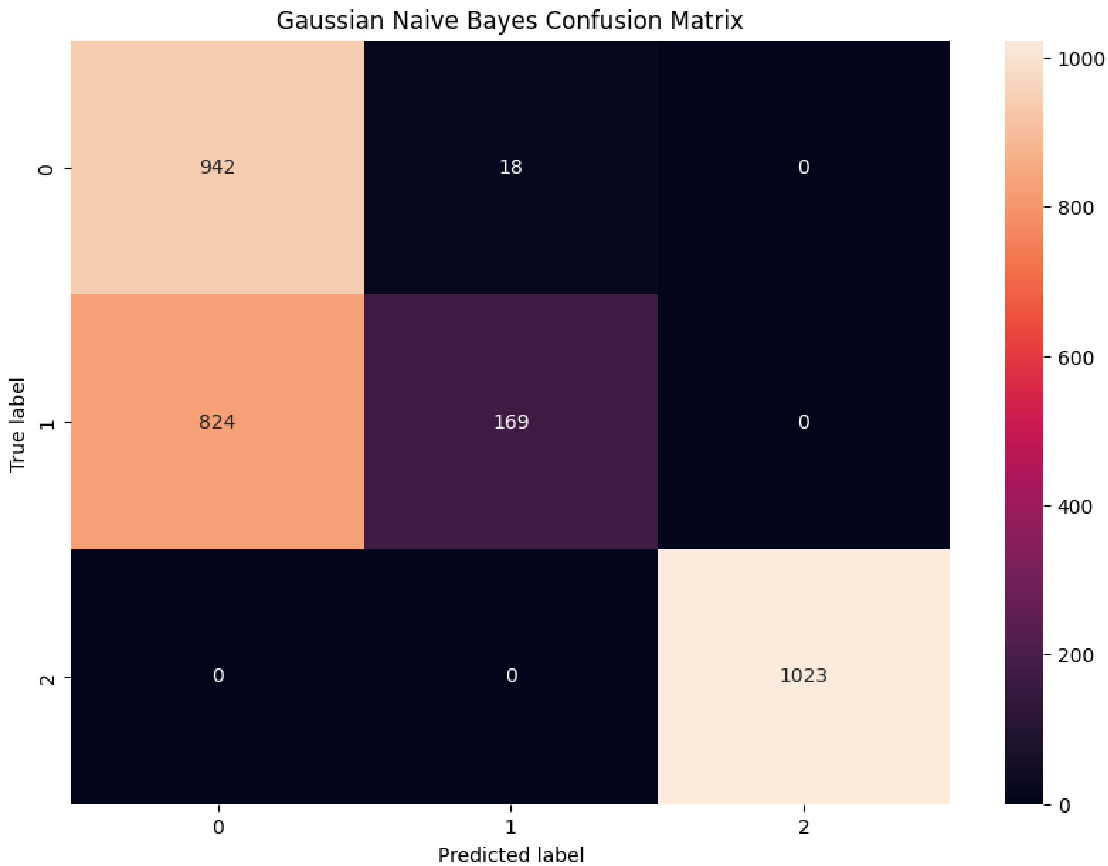
[343]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[344]:
```
accuracy(y_test_model1, y_hat_GaussianNaiveBayes)
```

[344]:
```
0.7170698924731183
```

[345]:
```
plt.figure(figsize=(10,7))
y_actu_GaussianNaiveBayes = pd.Series(y_test_model1, name='Actual')
y_pred_GaussianNaiveBayes = pd.Series(y_hat_GaussianNaiveBayes,␣
 ↪name='Predicted')
cm = pd.crosstab(y_actu_GaussianNaiveBayes, y_pred_GaussianNaiveBayes)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.title("Gaussian Naive Bayes Confusion Matrix")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

[345]:
```
Text(0.5, 47.7222222222222, 'Predicted label')
```



1.13 KNN Classifier

[346]:
```
class KNNClassifier():

  def fit(self, X, y):
    self.X= X
    self.y= y

  def predict(self, X, K, epsilon=1e-3):
    N= len(X)
    y_hat= np.zeros(N)

    for i in range(N):
      dist2= np.sum((self.X-X[i])**2, axis=1)
      idxt= np.argsort(dist2)[:K]
      gamma_k = 1/(np.sqrt(dist2[idxt]+epsilon))

      y_hat[i]= np.bincount(self.y[idxt], weights= gamma_k).argmax()
    return y_hat
```

[347]:
```
model1 = KNNClassifier()
model1.fit(X_train_model1,y_train_model1)
```

[348]:
```
best_k = 0
best_acc = 0
for i in range(3, 15):
  y_hat_k_model1 = model1.predict(X_test_model1, i)
  acc = accuracy(y_test_model1, y_hat_k_model1)
  if acc > best_acc:
    best_acc = acc
    best_k = i
print(best_k, best_acc)
```

```
3 0.9986559139784946
```

[349]:
```
y_hat_model1= model1.predict(X_test_model1, 3)
y_hat_model1
```

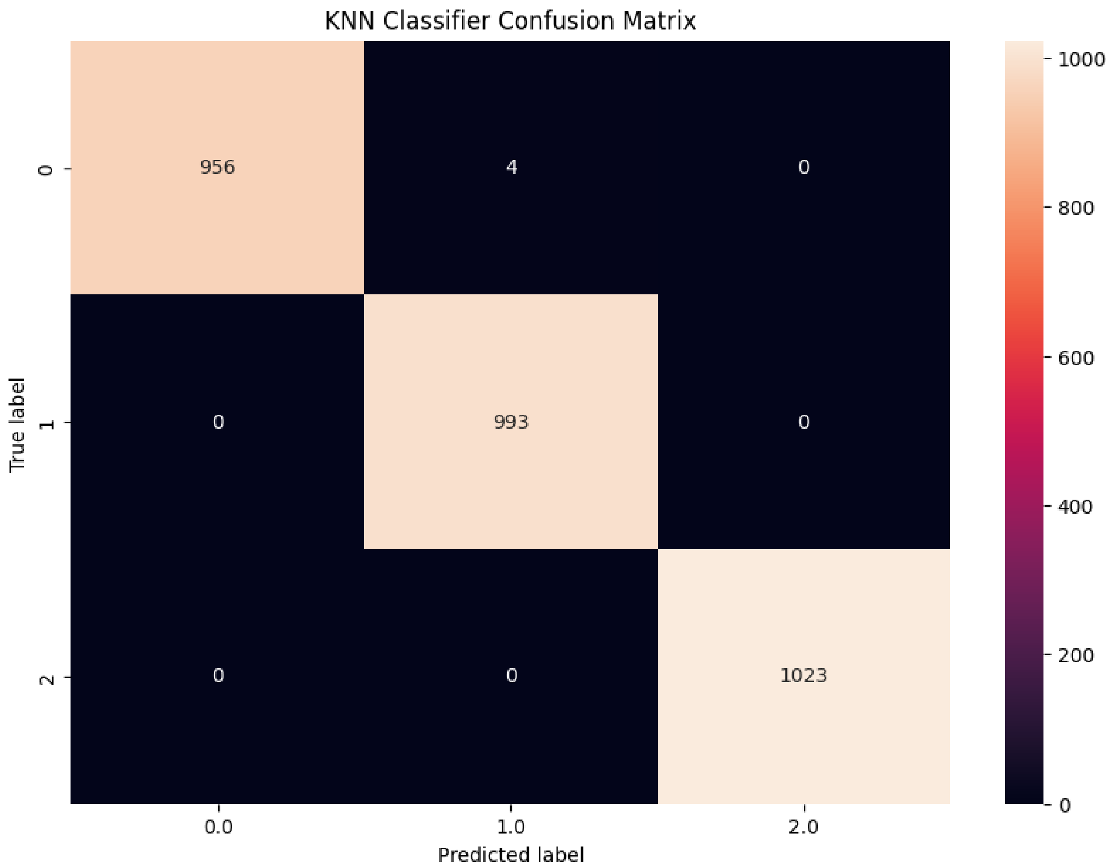
[349]:
```
array([0., 0., 1., …, 1., 0., 1.])
```

[350]:
```
accuracy(y_test_model1, y_hat_model1)
```

[350]:
```
0.9986559139784946
```

[351]:
```
plt.figure(figsize=(10,7))
y_actu_KNN_Classifier = pd.Series(y_test_model1, name='Actual')
y_pred_KNN_Classifier  = pd.Series(y_hat_model1, name='Predicted')
cm = pd.crosstab(y_actu_KNN_Classifier, y_pred_KNN_Classifier)
ax = sns.heatmap(cm, annot=True, fmt="d")
plt.title("KNN Classifier Confusion Matrix")
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

[351]:
```
Text(0.5, 47.7222222222222, 'Predicted label')
```



1.14 Dataset for regressors

[352]:
```
X_train_model2 = train_df[['longitude', 'latitude', 'lot_acres', 'sqrt_ft',␣
↪'bedrooms', 'bathrooms', 'garage', 'fireplaces']]
y_train_model2 = train_df['sold_price']

X_test_model2 = test_df[['longitude', 'latitude', 'lot_acres', 'sqrt_ft',␣
↪'bedrooms', 'bathrooms', 'garage', 'fireplaces']]
y_test_model2 = test_df['sold_price']

len(X_train_model2), len(y_train_model2), len(X_test_model2), len(y_test_model2)
```

[352]: (11904, 11904, 2976, 2976)

[353]:
```
X_train_model2= X_train_model2.to_numpy()
X_test_model2=X_test_model2.to_numpy()
X_train_model2
```

```
[353]: array([[4.82611658e-01, 6.67732132e-02, 7.92479109e-01, …,
         5.71428571e-02, 1.00000000e-01, 5.55555556e-01],
        [5.05405426e-01, 1.36144758e-01, 7.14948932e-04, …,
         5.71428571e-02, 1.00000000e-01, 2.22222222e-01],
        [4.82611658e-01, 6.67732132e-02, 7.92479109e-01, …,
         5.71428571e-02, 1.00000000e-01, 5.55555556e-01],
        …,
        [5.56640921e-01, 2.60057197e-01, 1.49025070e-03, …,
         1.42857143e-01, 1.00000000e-01, 5.55555556e-01],
        [5.43842160e-01, 2.60086036e-01, 7.89229341e-05, …,
         2.85714286e-02, 6.66666667e-02, 1.11111111e-01],
        [4.82611658e-01, 6.67732132e-02, 7.92479109e-01, …,
         5.71428571e-02, 1.00000000e-01, 5.55555556e-01]])
```

```
[354]: X_test_model2[0]
```

```
[354]: array([0.58436923, 0.26360862, 0.00158774, 0.0907171 , 0.08571429,
       0.05714286, 0.06666667, 0.33333333])
```

```
[355]: y_train_model2=y_train_model2.to_numpy().astype('int')
y_test_model2=y_test_model2.to_numpy().astype('int')
y_train_model2
```

```
[355]: array([4200000,  606000, 4200000, …, 3411450,  625000, 4200000])
```

```
[356]: y_test_model2
```

```
[356]: array([ 600000,  550000, 1937000, …, 1073942,  950000, 2150000])
```

1.15 MVLinearRegression

```
[357]: def MAPE(y_true, y_pred):
  y_true, y_pred = np.array(y_true), np.array(y_pred)
  return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

```
[358]: def OLS(Y, Y_hat, N):
  return ((1/(2*N))*np.sum((Y-Y_hat)**2))
```

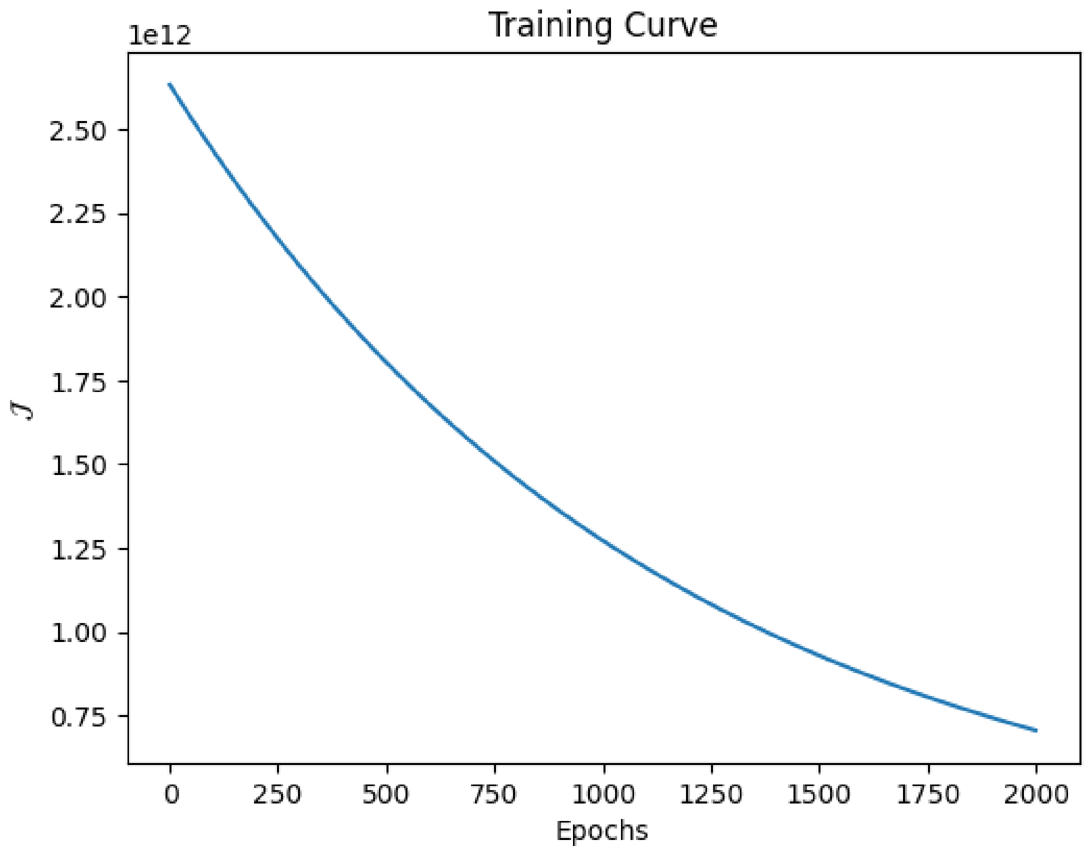
```
[359]: class MVLinearRegression():
  def fit(self, X, y, eta=1e-3, epochs= 1e3, show_curve=False):
    epochs = int(epochs)
    N, D = X.shape
    Y = y

    #Begin Optimization
    self.W = np.random.randn(D)
    self.J = np.zeros(epochs)
```

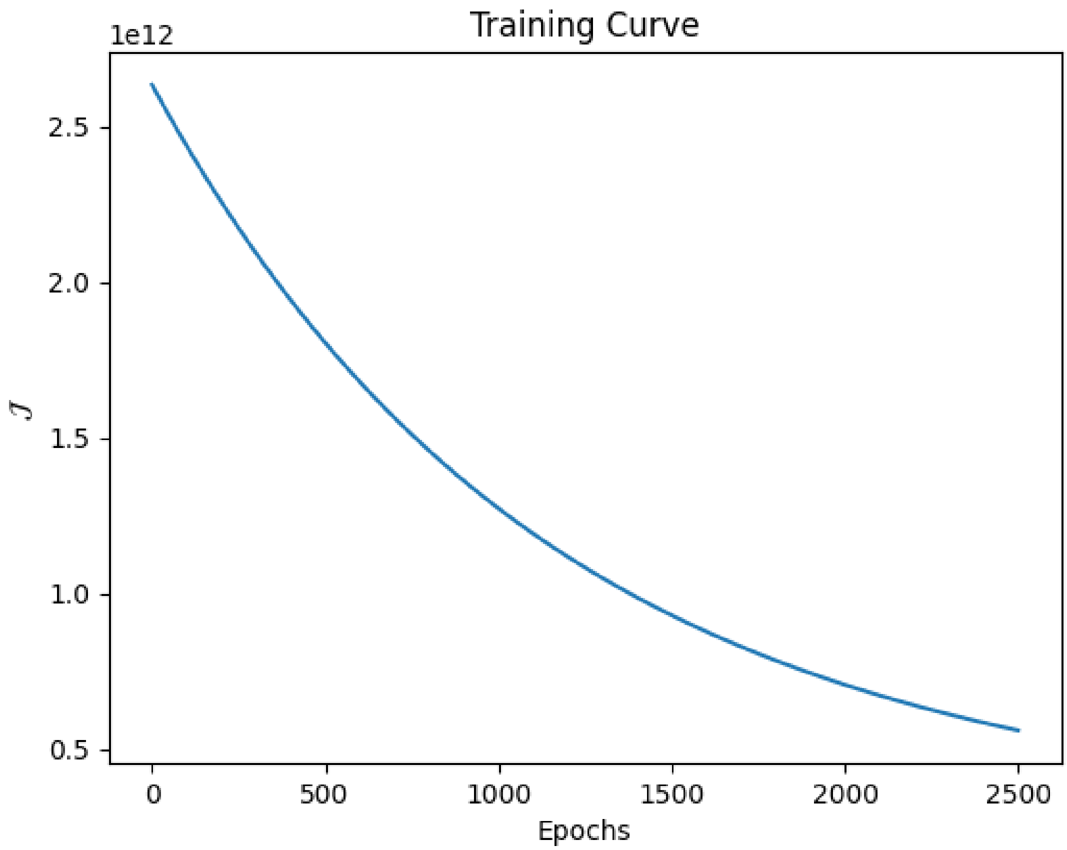
```
    #Stochastic Gradient Descent
    for epoch in range(epochs):
      Y_hat = self.predict(X)
      self.J[epoch]= OLS(Y,Y_hat, N)
      #weight Update Rule
      self.W -= eta*(1/N)*(X.T@(Y_hat-Y))

    if show_curve:
      plt.figure()
      plt.plot(self.J)
      plt.xlabel("Epochs")
      plt.ylabel("$\mathcal{J}$")
      plt.title("Training Curve")

  def predict(self, X):
    return X@self.W
```

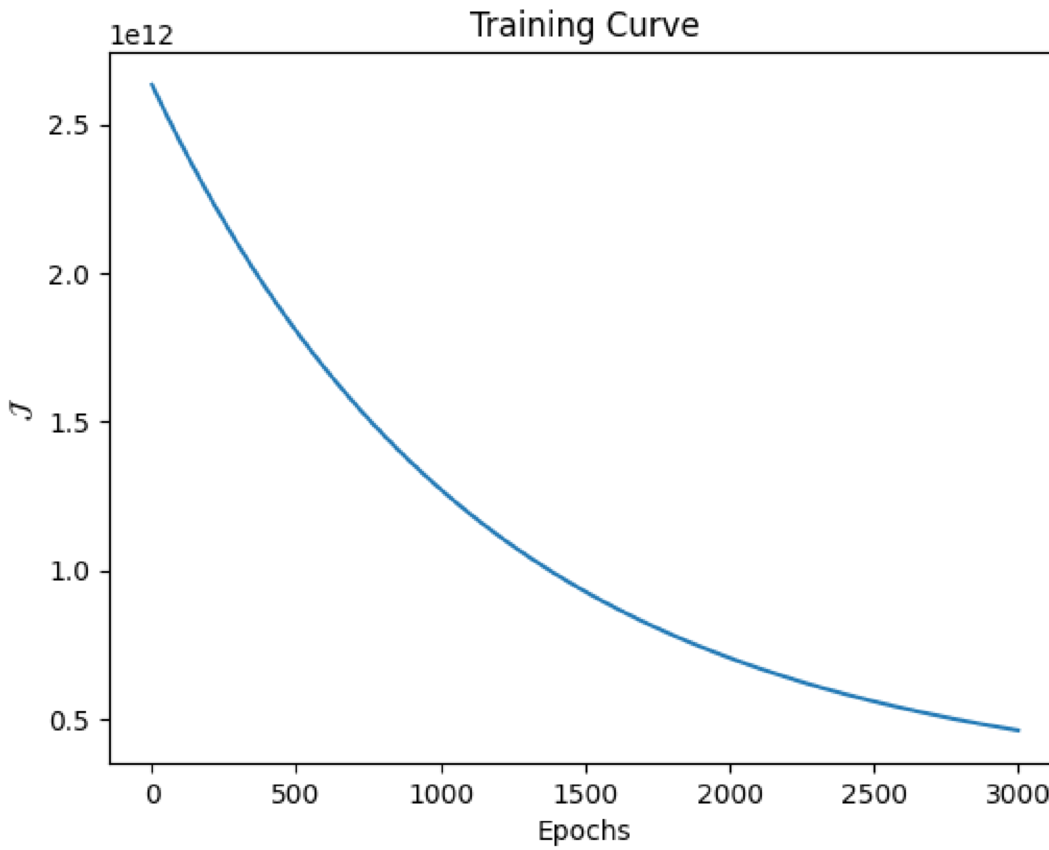
[360]:
```
model2 = MVLinearRegression()
```

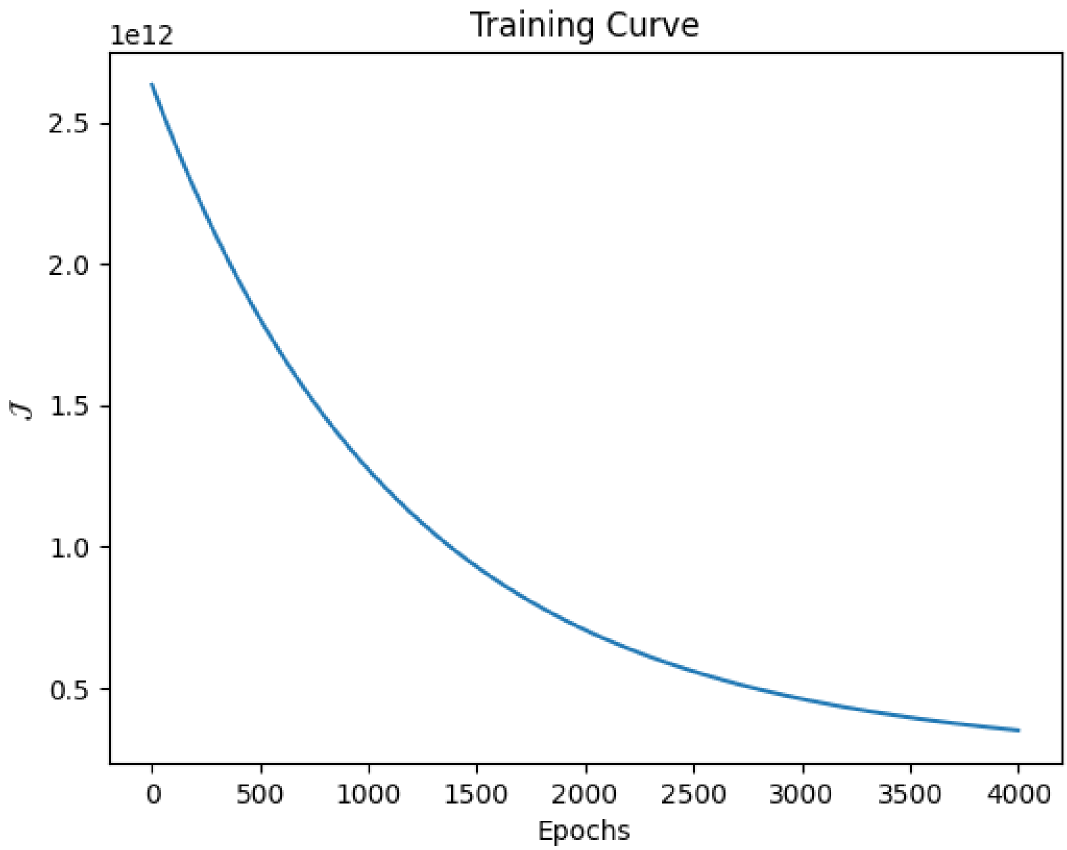
[361]:
```
best_eta = 1e-3
best_epochs = 1e3
best_err_model2 = float('inf')
eta_values = [0.001, 0.0008, 0.0006, 0.0005, 0.0001]
epoch_values = [2000, 2500, 3000, 4000]
for eta in eta_values:
    for epochs in epoch_values:
        model2.fit(X_train_model2, y_train_model2, eta=eta, epochs=epochs,␣
 ↪show_curve=True)
        y_hat_model2 = model2.predict(X_test_model2)
        err = MAPE(y_test_model2, y_hat_model2)
        if err < best_err_model2:
            best_err_model2 = err
            best_eta = eta
            best_epochs = epochs
print(f"Best eta: {best_eta}, Best epochs: {best_epochs}, Best error:␣
 ↪{best_err_model2}")
```

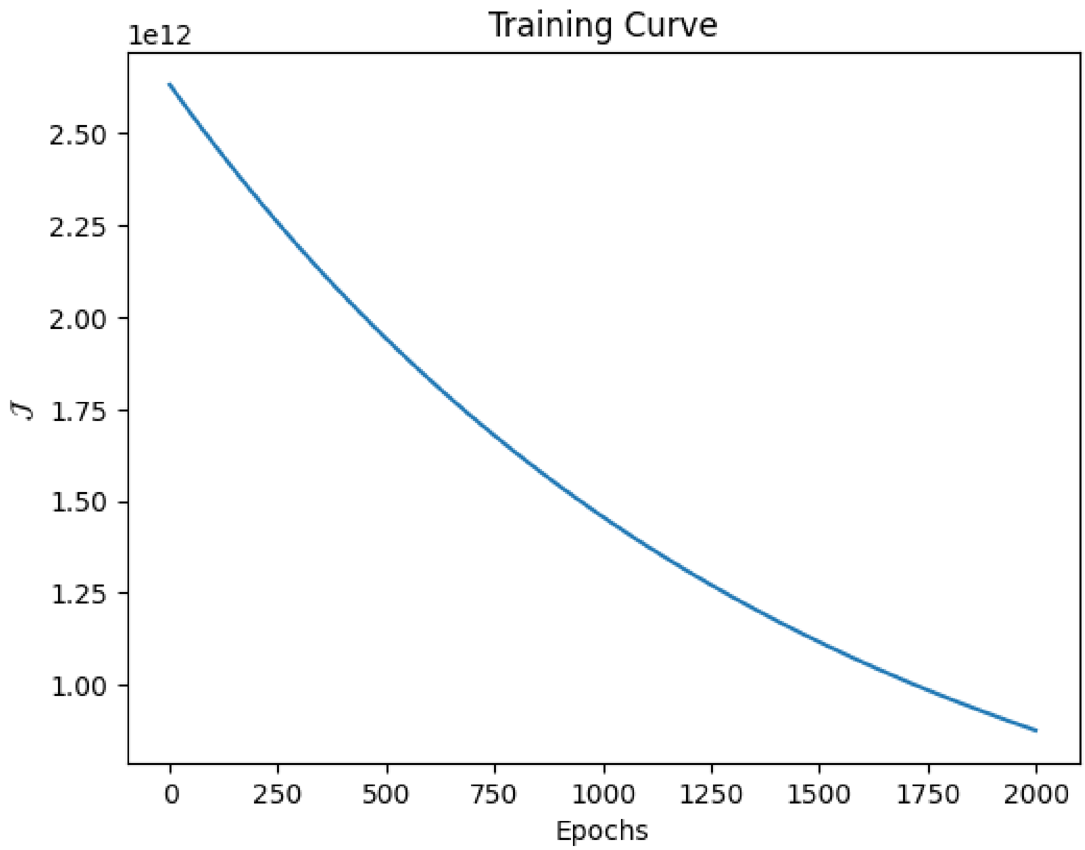
```
Best eta: 0.0008, Best epochs: 2000, Best error: 45.08475867648159
```

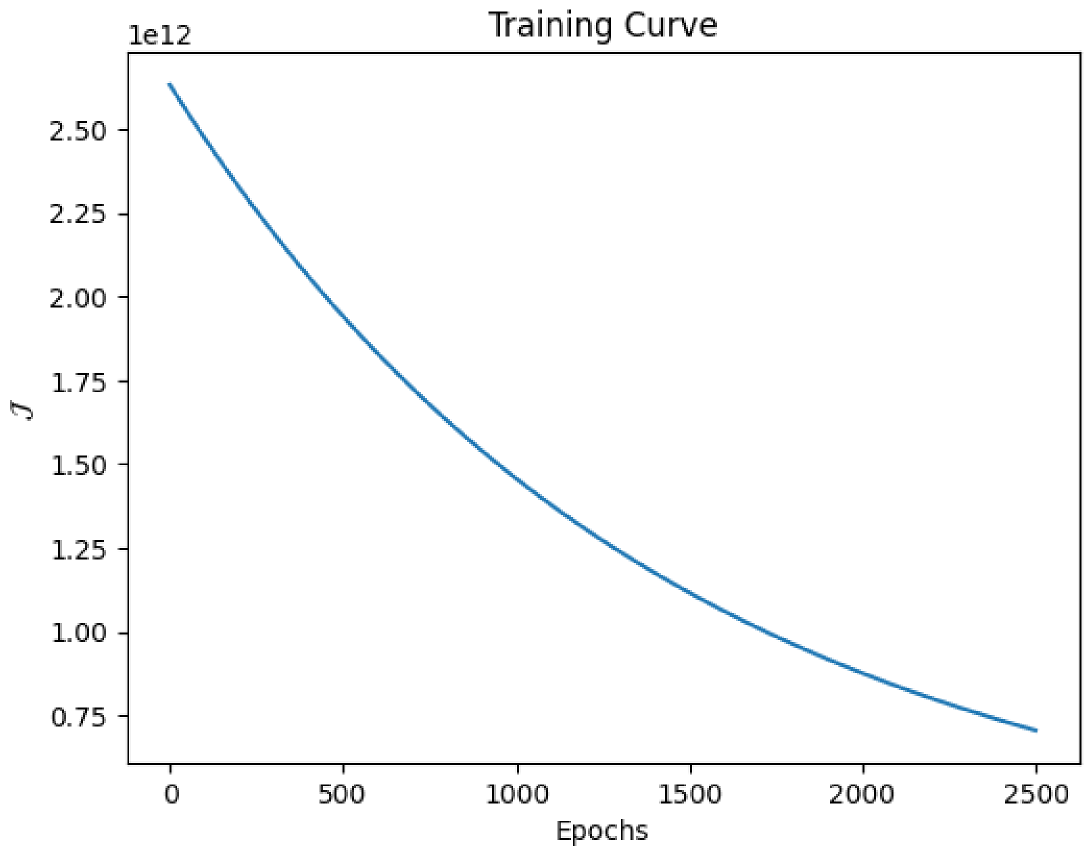

Training Curve
1e12
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
$\mathcal{J}$
0
250
500
750
1000
1250
1500
1750
2000
Epochs

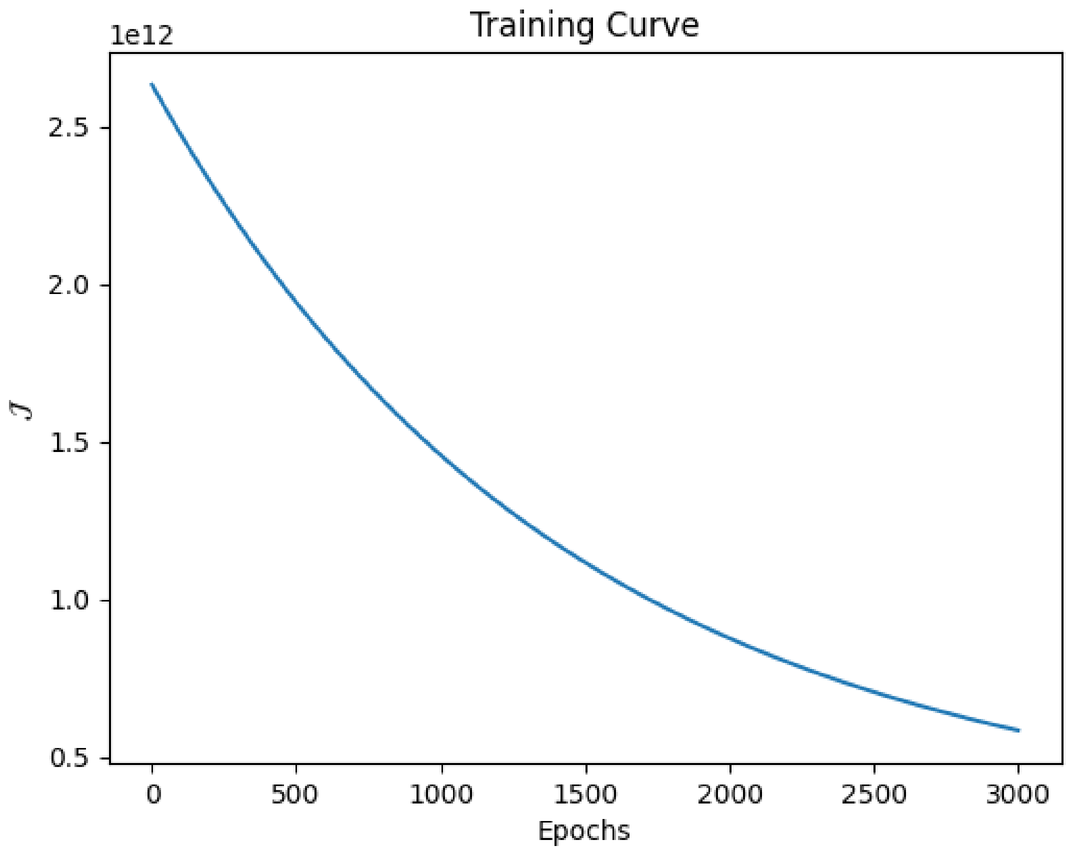

Training Curve
1e12
2.5
2.0
1.5
1.0
0.5
$\mathcal{J}$
0
500
1000
1500
2000
2500
Epochs

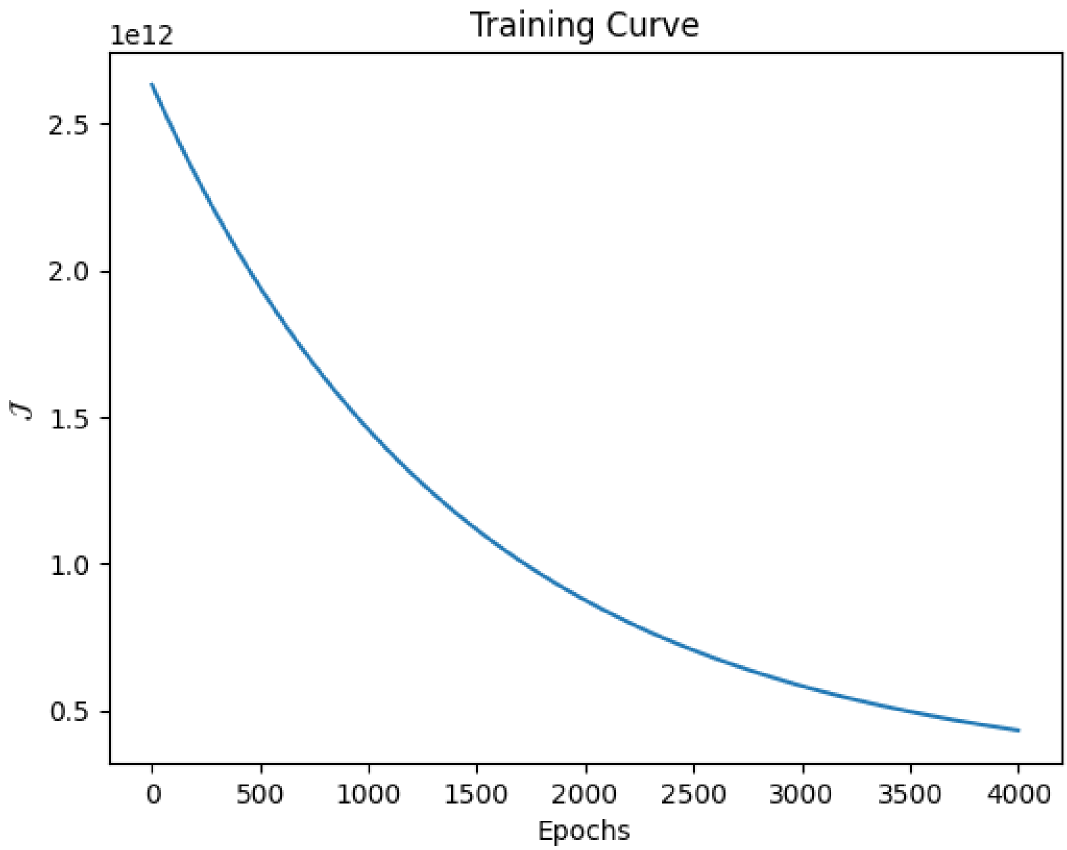

Training Curve
1e12
2.5
2.0
1.5
1.0
0.5
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
Epochs


Training Curve
1e12
2.5
2.0
1.5
1.0
0.5
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
3500
4000
Epochs

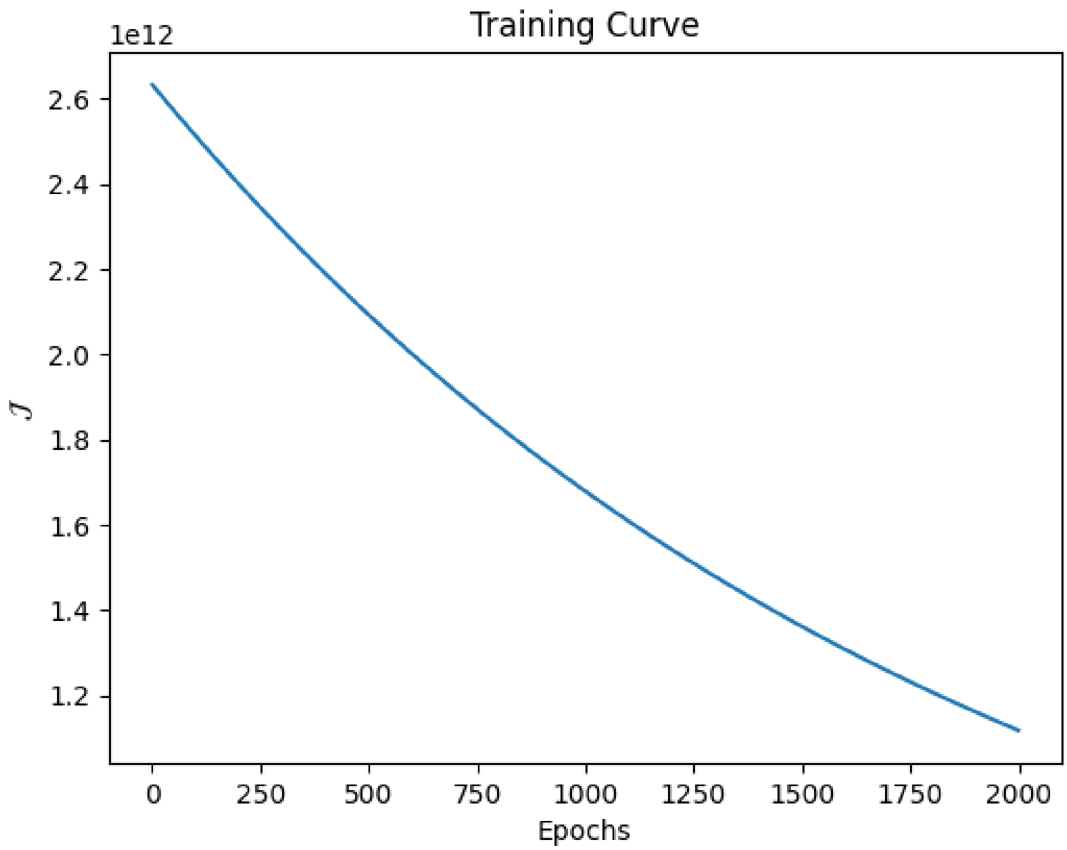

Training Curve
1e12
2.50
2.25
2.00
1.75
1.50
1.25
1.00
$\mathcal{J}$
0
250
500
750
1000
1250
1500
1750
2000
Epochs

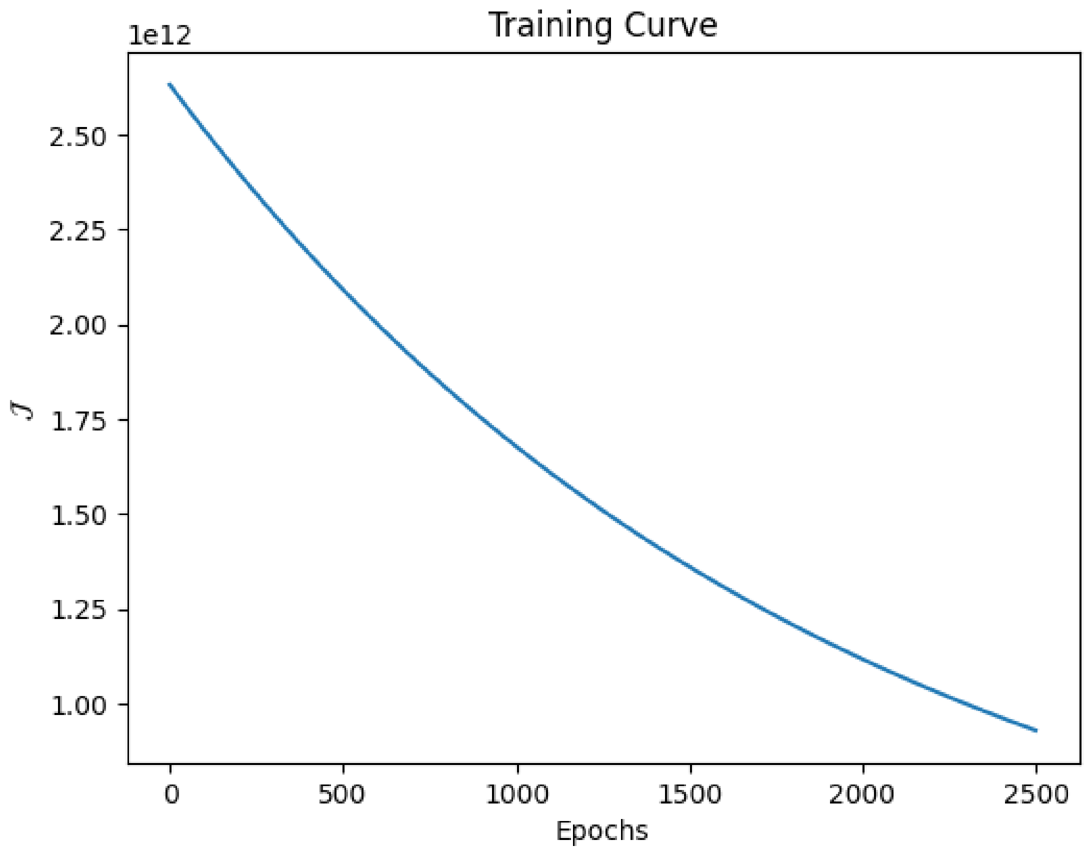

Training Curve
1e12
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
$\mathcal{J}$
0
500
1000
1500
2000
2500
Epochs

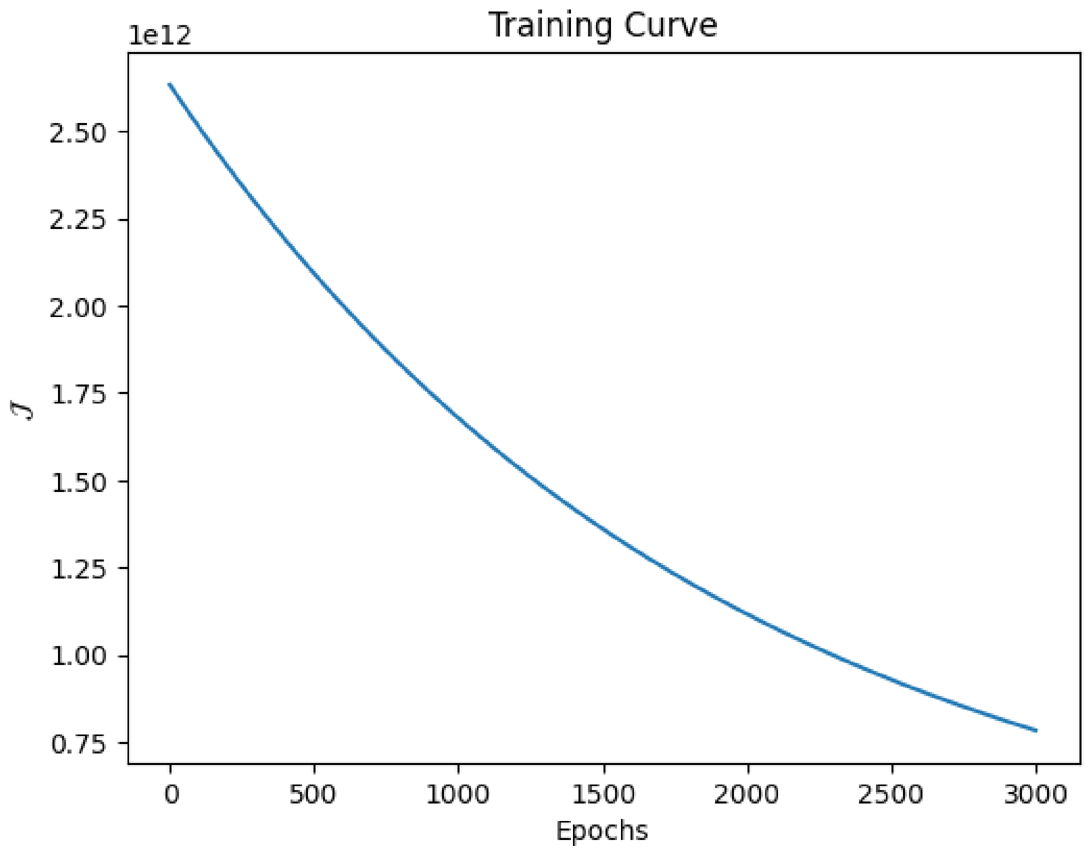

Training Curve
1e12
2.5
2.0
1.5
1.0
0.5
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
Epochs

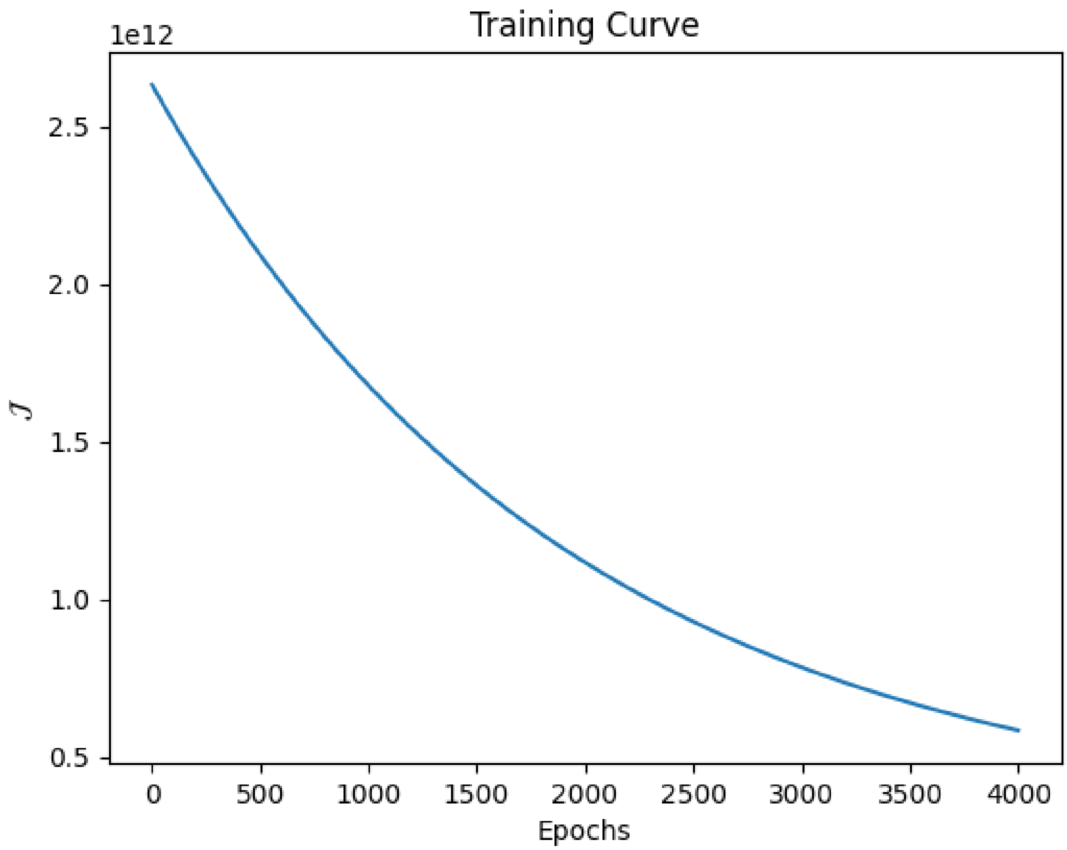


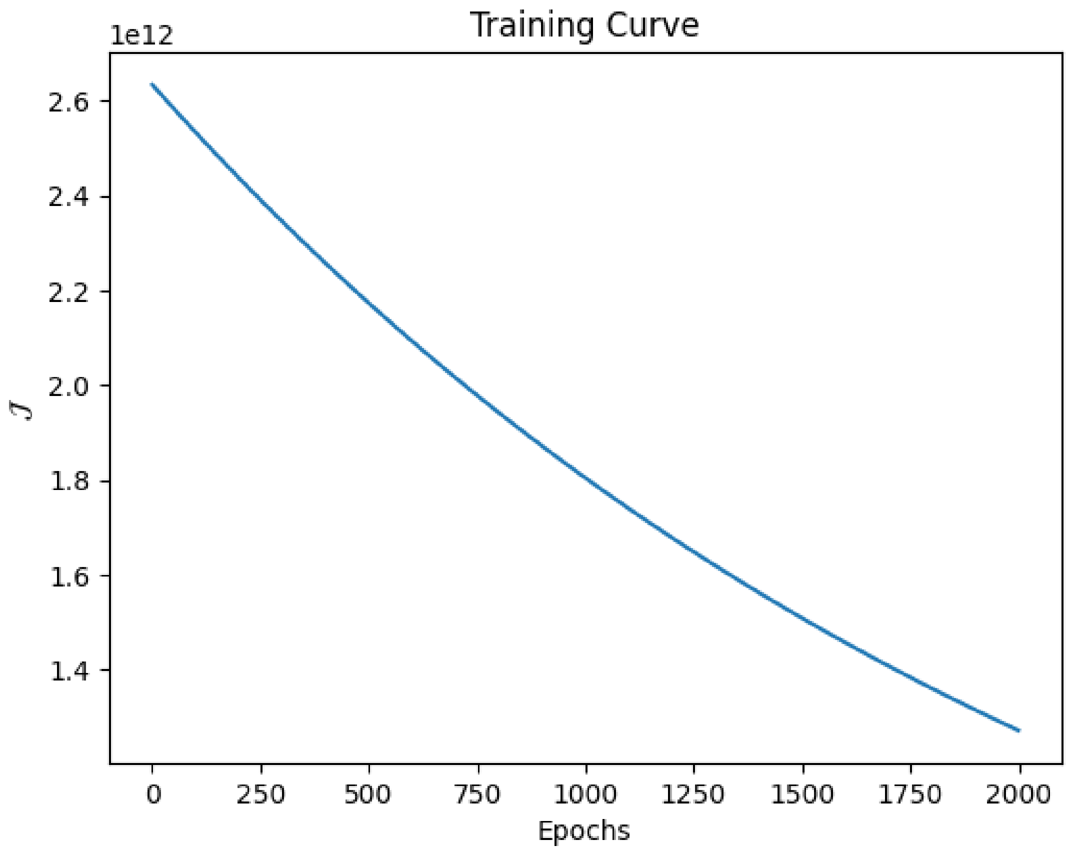
Training Curve
1e12
2.5
2.0
1.5
1.0
0.5
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
3500
4000
Epochs


Training Curve
1e12
2.6
2.4
2.2
2.0
1.8
1.6
1.4
1.2
$\mathcal{J}$
0
250
500
750
1000
1250
1500
1750
2000
Epochs

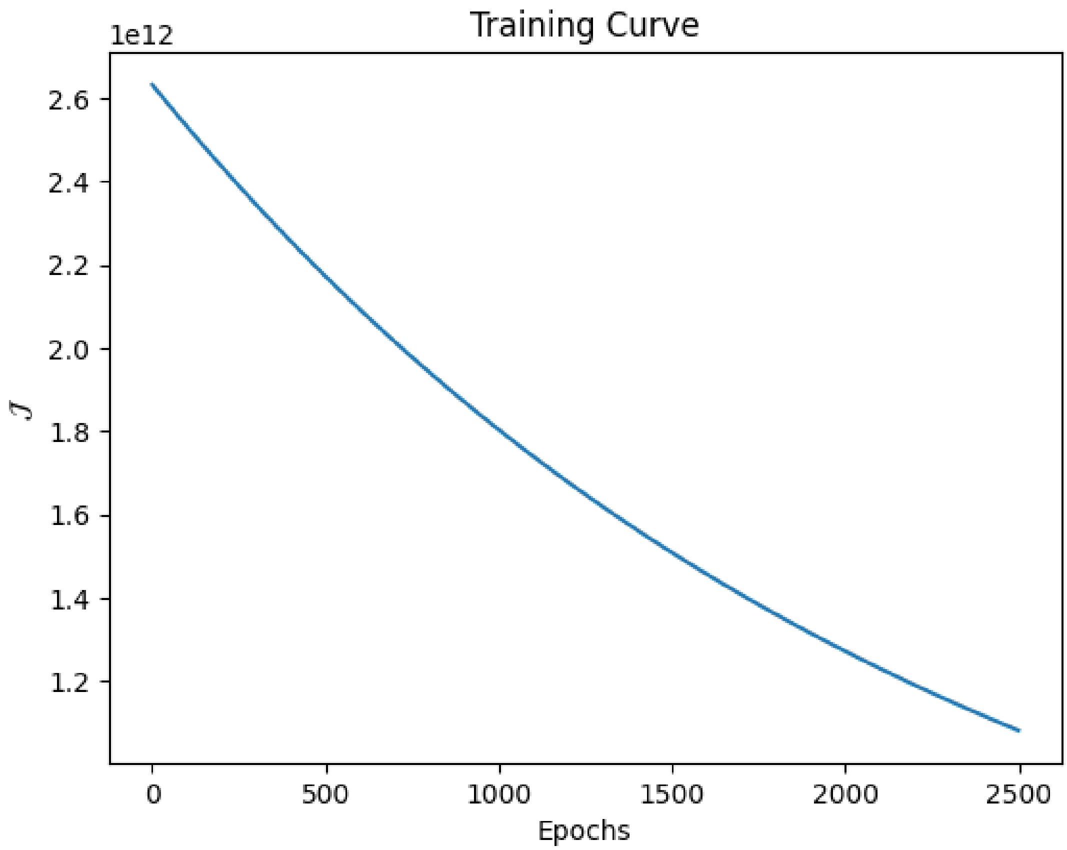

Training Curve
1e12
2.50
2.25
2.00
1.75
1.50
1.25
1.00
$\mathcal{J}$
0
500
1000
1500
2000
2500
Epochs


Training Curve
1e12
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
Epochs

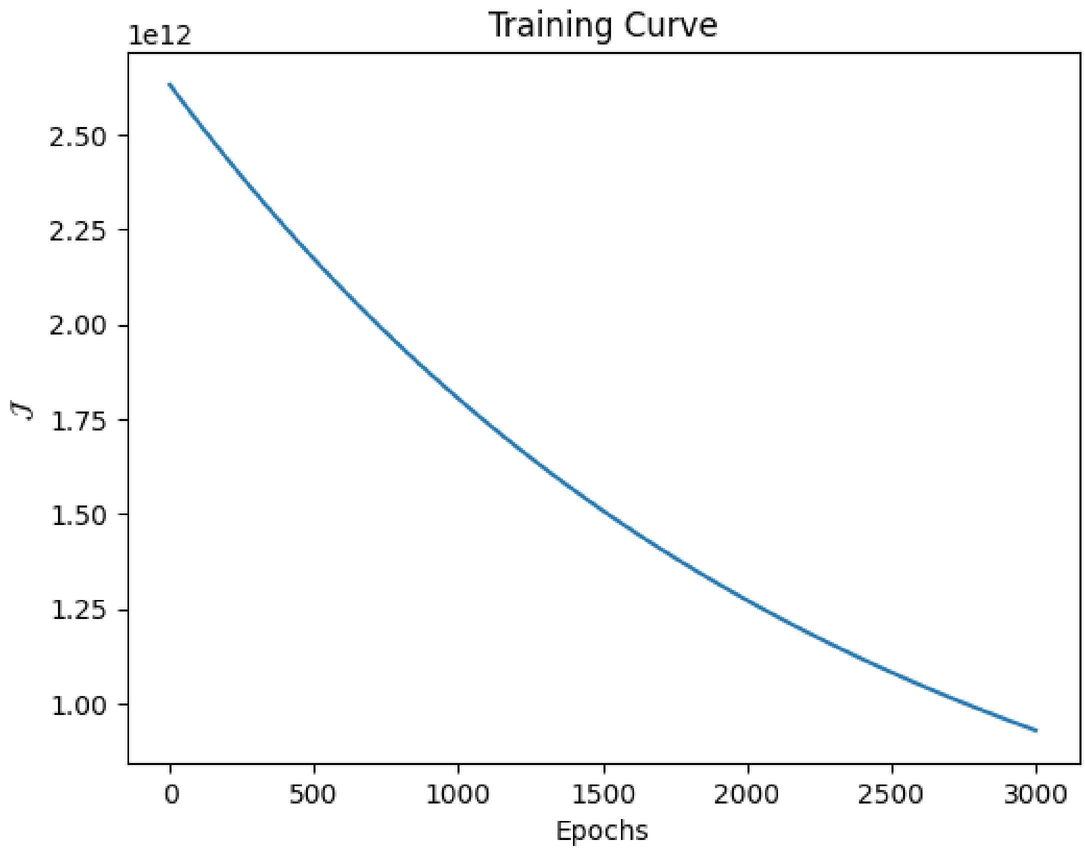

Training Curve
1e12
2.5
2.0
1.5
1.0
0.5
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
3500
4000
Epochs

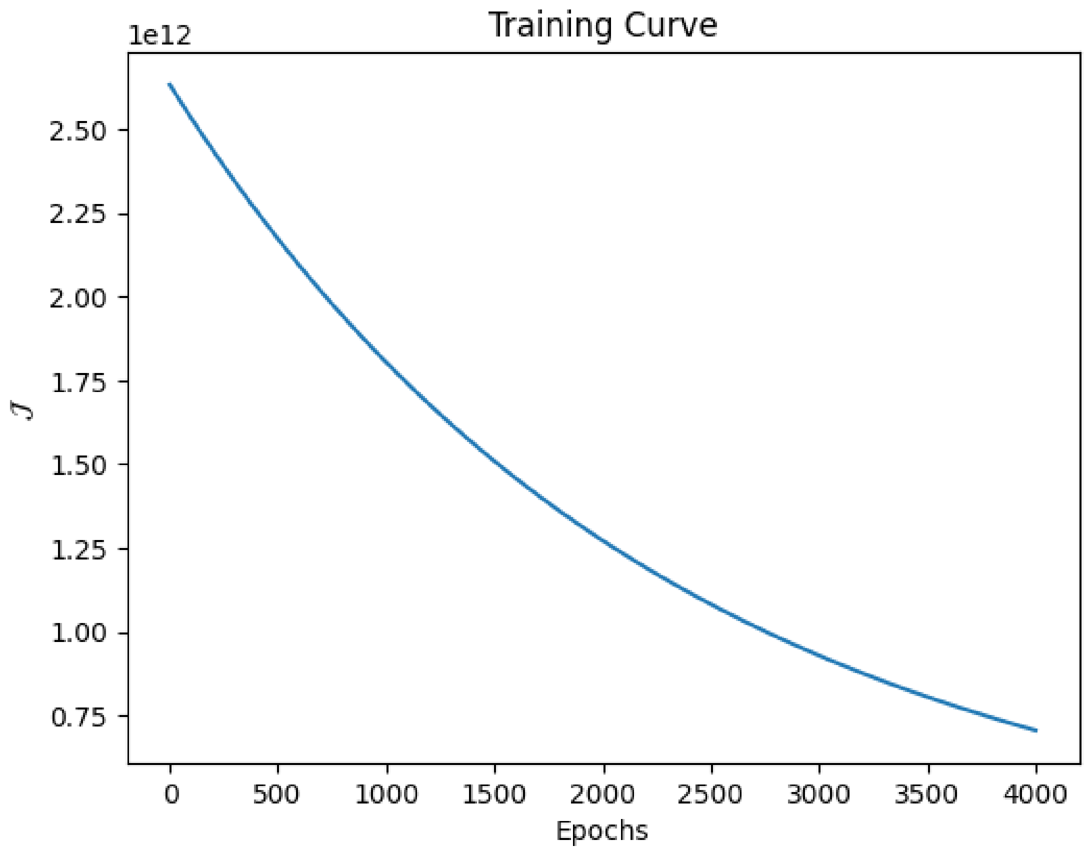


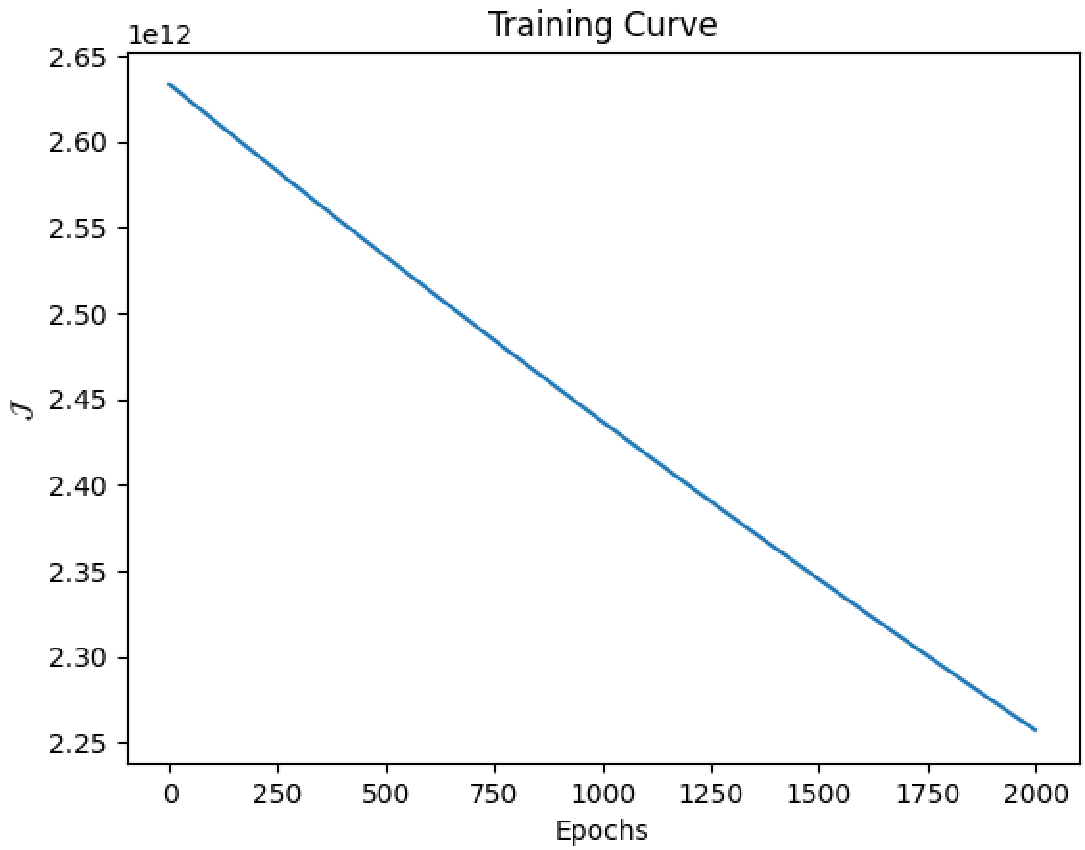
Training Curve
1e12
2.6
2.4
2.2
2.0
1.8
1.6
1.4
$\mathcal{J}$
0
250
500
750
1000
1250
1500
1750
2000
Epochs

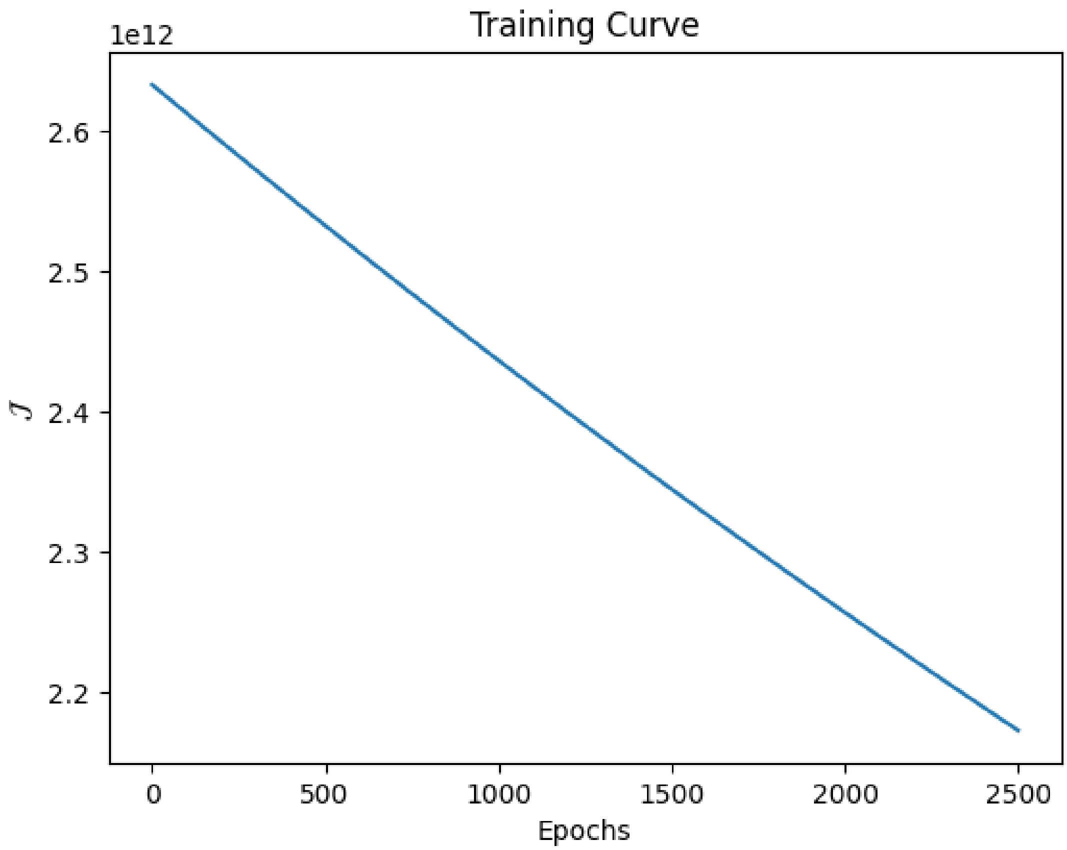


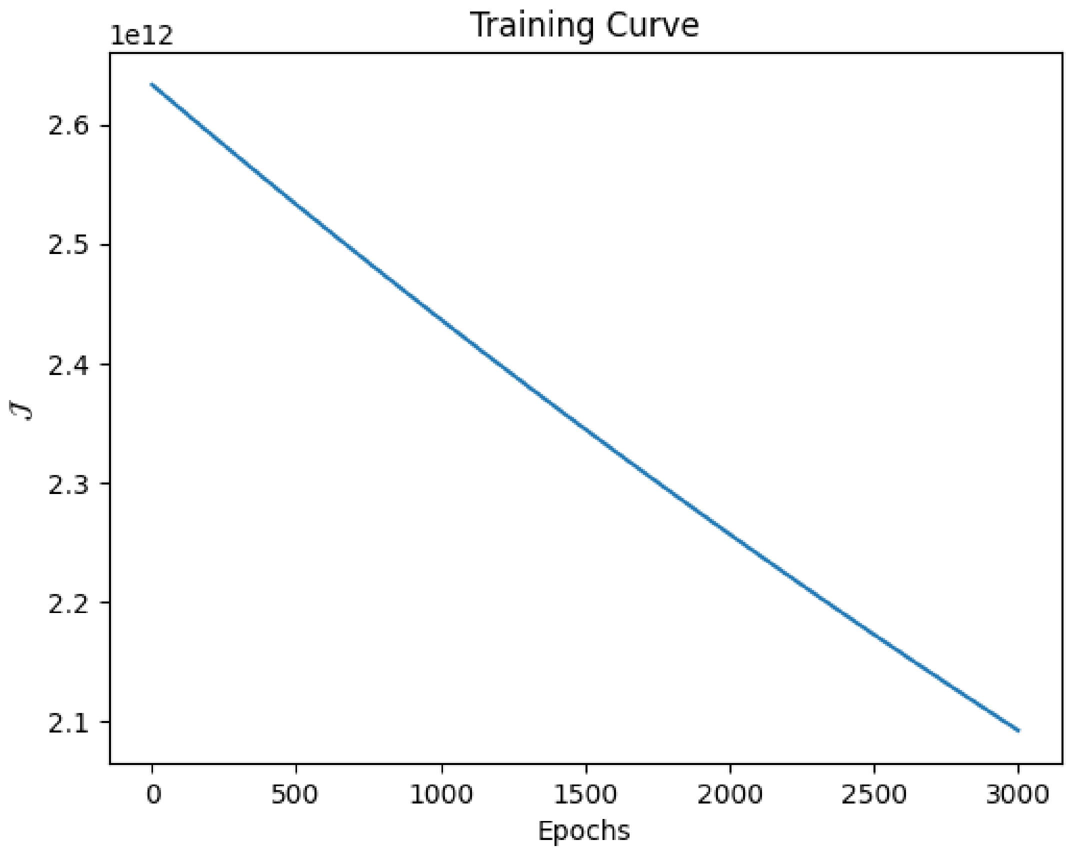
Training Curve
1e12
2.6
2.4
2.2
2.0
1.8
1.6
1.4
1.2
$\mathcal{J}$
0
500
1000
1500
2000
2500
Epochs


Training Curve
1e12
2.50
2.25
2.00
1.75
1.50
1.25
1.00
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
Epochs

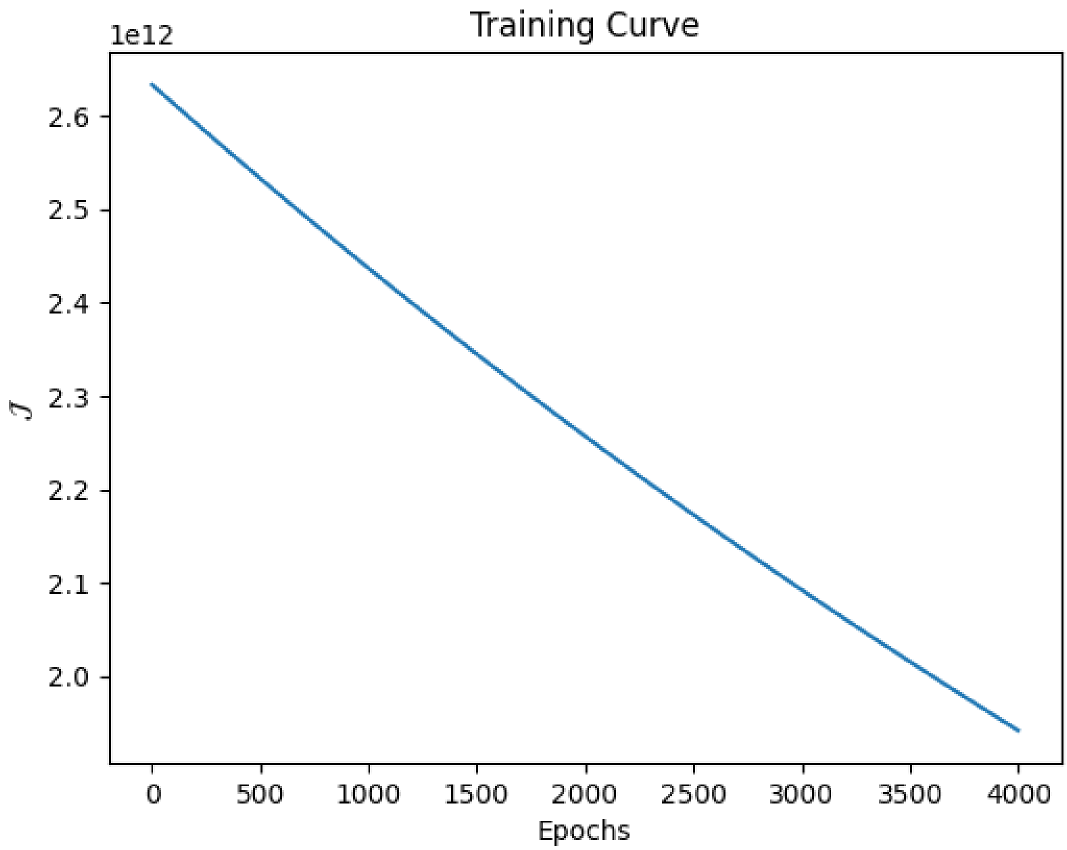

Training Curve
1e12
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
3500
4000
Epochs


Training Curve
1e12
2.65
2.60
2.55
2.50
2.45
2.40
2.35
2.30
2.25
$\mathcal{J}$
0
250
500
750
1000
1250
1500
1750
2000
Epochs

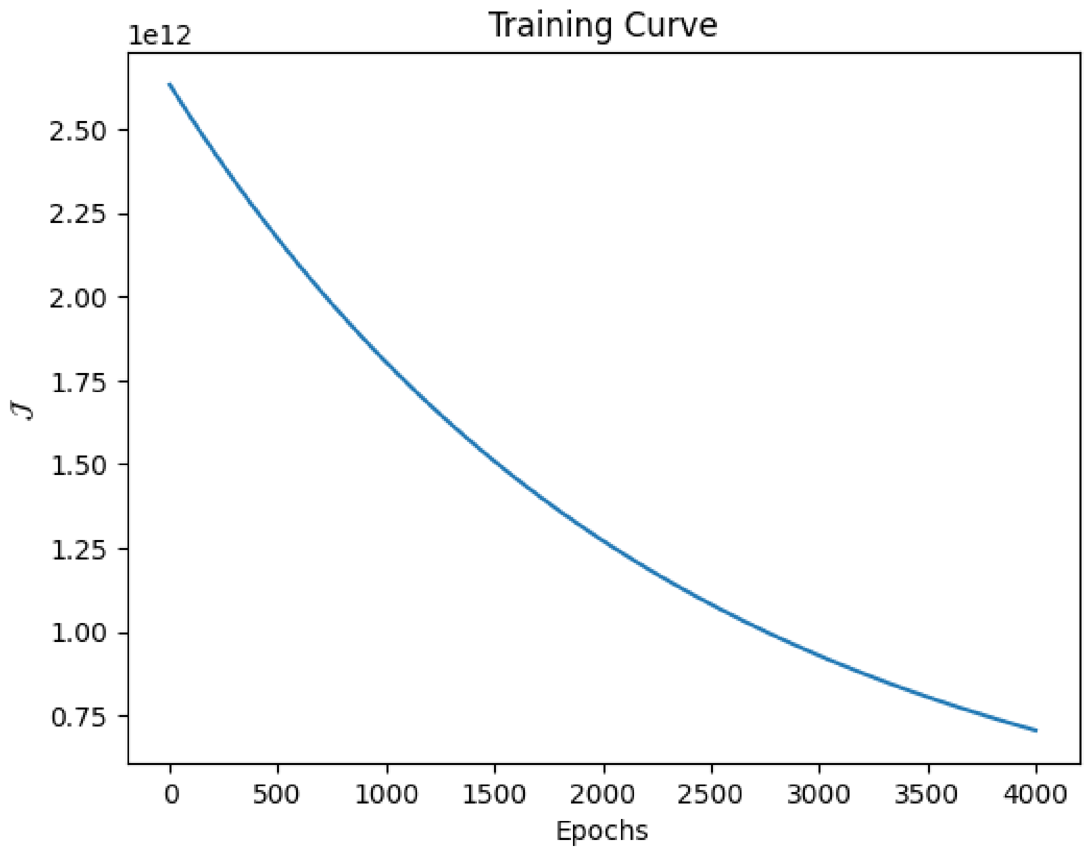

Training Curve
1e12
2.6
2.5
2.4
2.3
2.2
$\mathcal{J}$
0
500
1000
1500
2000
2500
Epochs



Training Curve
1e12
2.6
2.5
2.4
2.3
2.2
2.1
$\mathcal{J}$
0
500
1000
1500
2000
2500
3000
Epochs



[362]:
```
model2.fit(X_train_model2, y_train_model2, eta=0.0005, epochs=4000,␣
↪show_curve=True)
```



[363]:
```
y_hat_model2 = model2.predict(X_test_model2)
y_hat_model2
```

[363]:
```
array([1191008.40033922,  793439.93488291,  980361.88771729, …,
        933171.8396964 ,  922287.71323394, 1022416.05227339])
```

[364]:
```
y_train_model2
```

[364]:
```
array([4200000,  606000, 4200000, …, 3411450,  625000, 4200000])
```

[365]:
```
MAPE(y_test_model2, y_hat_model2)
```

[365]:
```
45.37299933178553
```

1.16 KNN Regressor

[366]:
```
class KNNRegressor():
  def fit(self, X, y):
    self.X = X
    self.y = y
```

```
  def predict(self, X, K, epsilon = 1e-3):
    N = len(X)
    y_hat = np.zeros(N)

    for i in range(N):
      dist2 = np.sum((self.X-X[i])**2, axis=1)
      idxt = np.argsort(dist2)[:K]
      gamma_k = np.exp(-dist2[idxt])/(np.exp(-dist2[idxt]).sum()+epsilon)
      y_hat[i] = gamma_k.dot(self.y[idxt])
    return y_hat
```

[367]:
```
# Instantiate our class
model3 = KNNRegressor()
```

[368]:
```
model3.fit(X_train_model2, y_train_model2)
```

[369]:
```
best_k_model3 = 3
best_err_model3 = 100
for i in range(3, 11):
  y_hat_model3 = model3.predict(X_test_model2, i)
  err = MAPE(y_test_model2, y_hat_model3)
  if err < best_err_model3:
    best_err_model3 = err
    best_k_model3 = i
print(best_k_model3, best_err_model3)
```

```
4 5.34680260574573
```

[370]:
```
y_hat_model3 = model3.predict(X_test_model2, 4)
y_hat_model3
```

[370]:
```
array([ 664809.11334947,  626402.4145537 , 1936515.87103224, …,
       1073673.5816046 ,  757311.62212901, 2149462.63434141])
```

[371]:
```
MAPE(y_test_model2, y_hat_model3)
```

[371]:
```
5.34680260574573
```

1.17 Teacher testing

1.17.1 Price calculation

[372]:
```
def scale_new_data_point(new_data_point, scaling_params, columns):
    scaled_data_point = []
    for i, column in enumerate(columns):
        if column in scaling_params:
```

```
            min_value = scaling_params[column]['min']
            max_value = scaling_params[column]['max']
            scaled_value = (new_data_point[i] - min_value) / (max_value -
↪min_value)
            scaled_data_point.append(scaled_value)
        else:
            scaled_data_point.append(new_data_point[i])

    return scaled_data_point

columns = ['longitude', 'latitude', 'lot_acres', 'sqrt_ft', 'bedrooms',
↪'bathrooms', 'garage', 'fireplaces']
teacher_test = [[-110.3782, 31.356362, 2154, 10500, 13, 10, 0, 6]]

scaled_teacher_test = [scale_new_data_point(point, scaling_params, columns) for
↪point in teacher_test]
scaled_teacher_test
```

[372]:
```
[[0.6987265827682044,
  0.0,
  1.0,
  0.44114886427632816,
  0.34285714285714286,
  0.2571428571428571,
  0.0,
  0.6666666666666666]]
```

[373]:
```
y_hat_KNN_classifier_teacher_test = model1.predict(scaled_teacher_test, 3)
y_hat_KNN_classifier_teacher_test
```

[373]:
```
array([1.])
```

[374]:
```
y_hat_KNN_regressor_teacher_test = model3.predict(scaled_teacher_test, 4)
y_hat_KNN_regressor_teacher_test
```

[374]:
```
array([5298675.33116721])
```

[375]:
```
#['longitude', 'latitude', 'lot_acres', 'sqrt_ft', 'bedrooms', 'bathrooms',
↪'garage', 'fireplaces']
```

[376]:
```
# teacher_test=[[0.58436923, 0.26360862, 0.00158774, 0.0907171 , 0.08571429,
#        0.05714286, 0.06666667, 0.33333333]]
```

[377]:
```
# teacher_test=[[-110.3782, 31.356362, 2154, 10500, 13, 10, 0, 6]]
```