Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Read the Dataset

In [2]:
```
df=pd.read_csv('tesla.csv')
df
```

Out[2]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.000000	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.420000	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.920000	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.100000	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.000000	15.830000	16.110001	6866900	16.110001
...	...	...	...	...	...	...	...
1687	3/13/2017	244.820007	246.850006	242.779999	246.169998	3010700	246.169998
1688	3/14/2017	246.110001	258.119995	246.020004	258.000000	7575500	258.000000
1689	3/15/2017	257.000000	261.000000	254.270004	255.729996	4816600	255.729996
1690	3/16/2017	262.399994	265.750000	259.059998	262.049988	7100400	262.049988
1691	3/17/2017	264.000000	265.329987	261.200012	261.500000	6475900	261.500000

1692 rows × 7 columns

In [3]:
```
df.isnull()
```

Out[3]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1687	False	False	False	False	False	False	False
1688	False	False	False	False	False	False	False
1689	False	False	False	False	False	False	False
1690	False	False	False	False	False	False	False
1691	False	False	False	False	False	False	False

1692 rows × 7 columns

checking null values

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Volume       0
Adj Close    0
dtype: int64
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1692 entries, 0 to 1691
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1692 non-null   object 
 1   Open       1692 non-null   float64
 2   High       1692 non-null   float64
 3   Low        1692 non-null   float64
 4   Close      1692 non-null   float64
 5   Volume     1692 non-null   int64  
 6   Adj Close  1692 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 92.7+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

	Open	High	Low	Close	Volume	Adj Close
count	1692.000000	1692.000000	1692.000000	1692.000000	1.692000e+03	1692.000000
mean	132.441572	134.769698	129.996223	132.428658	4.270741e+06	132.428658
std	94.309923	95.694914	92.855227	94.313187	4.295971e+06	94.313187
min	16.139999	16.629999	14.980000	15.800000	1.185000e+05	15.800000
25%	30.000000	30.650000	29.215000	29.884999	1.194350e+06	29.884999
50%	156.334999	162.370002	153.150002	158.160004	3.180700e+06	158.160004
75%	220.557495	224.099999	217.119999	220.022503	5.662100e+06	220.022503
max	287.670013	291.420013	280.399994	286.040009	3.716390e+07	286.040009

In [7]:
```
df.shape
```

Out[7]:
```
(1692, 7)
```

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```
# assume 'df' is your DataFrame with a 'Date' column
df['Date'] = pd.to_datetime(df['Date'])
df['Unix Timestamp'] = df['Date'].apply(lambda x: x.timestamp())
```

dropping unwanted columns

In [10]:
```
df=df.drop(columns=['Adj Close','Date'],axis=1)
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Unix Timestamp'], dtype='obje
ct')
```

MinMax Normalization

In [12]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [13]:
```
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df_scaled
```

Out[13]:

	Open	High	Low	Close	Volume	Unix Timestamp
0	0.010533	0.030460	0.009645	0.029936	0.503377	0.000000
1	0.035539	0.050184	0.031347	0.029714	0.460748	0.000408
2	0.032630	0.033808	0.019931	0.022795	0.218659	0.000815
3	0.025264	0.023545	0.014053	0.012581	0.135544	0.001223
4	0.014216	0.012264	0.003202	0.001147	0.182166	0.002854
...	...	...	...	...	...	...
1687	0.842191	0.837803	0.858262	0.852464	0.078072	0.998369
1688	0.846941	0.878816	0.870469	0.896240	0.201294	0.998777
1689	0.887047	0.889297	0.901552	0.887840	0.126820	0.999185
1690	0.906935	0.906583	0.919599	0.911227	0.188469	0.999592
1691	0.912827	0.905055	0.927662	0.909192	0.171611	1.000000

1692 rows × 6 columns

Statistical feature extraction

In [14]:

```
features = df_scaled.drop('Close', axis=1)
target_variables = df_scaled['Close']

# Statistical feature extraction (row-wise)
stat_features = pd.DataFrame()
stat_features['mean'] = features.mean(axis=1)
stat_features['std'] = features.std(axis=1)
stat_features['min'] = features.min(axis=1)
stat_features['max'] = features.max(axis=1)
stat_features['range'] = features.max(axis=1) - features.min(axis=1)
stat_features['median'] = features.median(axis=1)

# Quantiles (25th and 75th)
stat_features['25%'] = features.quantile(0.25, axis=1)
stat_features['75%'] = features.quantile(0.75, axis=1)

# Variance
stat_features['variance'] = features.var(axis=1)

stat_features
```

Out[14]:

	mean	std	min	max	range	median	25%	75%	v
0	0.110803	0.219735	0.000000	0.503377	0.503377	0.010533	0.009645	0.030460	0
1	0.115645	0.193768	0.000408	0.460748	0.460340	0.035539	0.031347	0.050184	0
2	0.061168	0.089036	0.000815	0.218659	0.217843	0.032630	0.019931	0.033808	0
3	0.039926	0.054299	0.001223	0.135544	0.134322	0.023545	0.014053	0.025264	0
4	0.042940	0.078000	0.002854	0.182166	0.179312	0.012264	0.003202	0.014216	0
...	...	...	...	...	...	...	...	...	
1687	0.722939	0.366552	0.078072	0.998369	0.920298	0.842191	0.837803	0.858262	0
1688	0.759260	0.317429	0.201294	0.998777	0.797483	0.870469	0.846941	0.878816	0
1689	0.760780	0.357428	0.126820	0.999185	0.872365	0.889297	0.887047	0.901552	0
1690	0.784236	0.335285	0.188469	0.999592	0.811124	0.906935	0.906583	0.919599	0
1691	0.783431	0.344080	0.171611	1.000000	0.828389	0.912827	0.905055	0.927662	0

1692 rows × 9 columns

Principle Component Analysis

In [15]:

```
from sklearn.decomposition import PCA
# Apply PCA
pca = PCA(n_components=3)
X_pca = pca.fit_transform(stat_features)

# Create a DataFrame with PCA results
df_pca = pd.DataFrame(data=X_pca, columns=['Principal Component 1', 'Principal C
df_pca
```

Out[15]:

	Principal Component 1	Principal Component 2	Principal Component 3
0	-0.617023	-0.395733	0.189968
1	-0.622444	-0.333381	0.147415
2	-0.851568	-0.095923	-0.004825
3	-0.936504	-0.018251	-0.054248
4	-0.912142	-0.072199	-0.014792
...	...	...	...
1687	1.053848	-0.176483	-0.017460
1688	1.050010	0.002144	0.048293
1689	1.113395	-0.081652	-0.019253
1690	1.125041	0.012709	0.007680
1691	1.136931	-0.007244	-0.006151

1692 rows × 3 columns

Concatenate normalized data and statistical features

In [16]:

```
# Concatenate normalized data and statistical features
combined_output = pd.concat([ df_pca,df_scaled], axis=1)

print(combined_output.head())
```

```
   Principal Component 1  Principal Component 2  Principal Component 3  \
0              -0.617023              -0.395733               0.189968
1              -0.622444              -0.333381               0.147415
2              -0.851568              -0.095923              -0.004825
3              -0.936504              -0.018251              -0.054248
4              -0.912142              -0.072199              -0.014792

       Open      High       Low     Close    Volume  Unix Timestamp
0  0.010533  0.030460  0.009645  0.029936  0.503377        0.000000
1  0.035539  0.050184  0.031347  0.029714  0.460748        0.000408
2  0.032630  0.033808  0.019931  0.022795  0.218659        0.000815
3  0.025264  0.023545  0.014053  0.012581  0.135544        0.001223
4  0.014216  0.012264  0.003202  0.001147  0.182166        0.002854
```

spliting Training and testing data

In [17]:

```
from sklearn.model_selection import train_test_split
x=combined_output.drop('Close',axis=1)
y=combined_output['Close']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_
```

POA Optimizer

In [18]:
```
from mealpy.optimizer import Optimizer
from mealpy import *
import pandas as pd
from sklearn.linear_model import LinearRegression

# Define the Original POA class
class OriginalPOA(Optimizer):
    def __init__(self, epoch: int = 10000, pop_size: int = 100, **kwargs: object
        super().__init__(**kwargs)
        self.epoch = self.validator.check_int("epoch", epoch, [1, 100000])
        self.pop_size = self.validator.check_int("pop_size", pop_size, [5, 10000
        self.set_parameters(["epoch", "pop_size"])
        self.is_parallelizable = False
        self.sort_flag = False

    def evolve(self, epoch):
        ## UPDATE location of food
        kk = self.generator.permutation(self.pop_size)[0]
        for idx in range(0, self.pop_size):
            # PHASE 1: Moving towards prey (exploration phase)
            if self.compare_target(self.pop[kk].target, self.pop[idx].target, se
                pos_new = self.pop[idx].solution + self.generator.random() * (se
            else:
                pos_new = self.pop[idx].solution + self.generator.random() * (se
            pos_new = self.correct_solution(pos_new)
            agent = self.generate_agent(pos_new)
            if self.compare_target(agent.target, self.pop[idx].target, self.prob
                self.pop[idx] = agent

            # PHASE 2: Winging on the water surface (exploitation phase)
            pos_new = self.pop[idx].solution + 0.2 * (1 - epoch/self.epoch) *(2*
            pos_new = self.correct_solution(pos_new)
            agent = self.generate_agent(pos_new)
            if self.compare_target(agent.target, self.pop[idx].target, self.prob
                self.pop[idx] = agent

# Define the objective function
def objective_function(solution):
    # Use the transformed data to train a model and predict the target variable
    model = LinearRegression()
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    # Calculate the fitness value (e.g., mean squared error)
    fitness = np.mean((y_pred - y_test) ** 2)
    return fitness


# Define the bounds for the problem
from mealpy.utils.space import FloatVar
bounds = [FloatVar(lb=x_train.min()[i], ub=x_train.max()[i]) for i in range(x_tr

# Define the problem dictionary
problem_dict = {
    "bounds": bounds,
    "minmax": "min",
```

```
    "obj_func": objective_function
}


# Create a POA model
model = OriginalPOA(epoch=100, pop_size=50)

# Solve the problem
g_best = model.solve(problem_dict)

# Print the solution and fitness value
print(f"Solution: {g_best.solution}, Fitness: {g_best.target.fitness}")
```

```
2024/07/28 10:59:14 AM, INFO, __main__.OriginalPOA: Solving single objective opti
mization problem.
2024/07/28 10:59:17 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 1, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 2.7
3120 seconds
2024/07/28 10:59:18 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 2, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
6235 seconds
2024/07/28 10:59:19 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 3, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
8347 seconds
2024/07/28 10:59:19 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 4, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
4050 seconds
2024/07/28 10:59:20 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 5, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
5229 seconds
2024/07/28 10:59:20 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 6, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
4070 seconds
2024/07/28 10:59:21 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 7, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
5158 seconds
2024/07/28 10:59:21 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 8, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
4088 seconds
2024/07/28 10:59:22 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 9, Curr
ent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.5
4071 seconds
2024/07/28 10:59:22 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 10, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53464 seconds
2024/07/28 10:59:23 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 11, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55398 seconds
2024/07/28 10:59:24 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 12, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55488 seconds
2024/07/28 10:59:24 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 13, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55210 seconds
2024/07/28 10:59:25 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 14, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54379 seconds
2024/07/28 10:59:25 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 15, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54511 seconds
2024/07/28 10:59:26 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 16, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53589 seconds
2024/07/28 10:59:26 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 17, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53637 seconds
2024/07/28 10:59:27 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 18, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53336 seconds
2024/07/28 10:59:27 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 19, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55754 seconds
2024/07/28 10:59:28 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 20, Cur
```

```
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
58647 seconds
2024/07/28 10:59:28 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 21, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55529 seconds
2024/07/28 10:59:29 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 22, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55431 seconds
2024/07/28 10:59:30 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 23, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54655 seconds
2024/07/28 10:59:30 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 24, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
58132 seconds
2024/07/28 10:59:31 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 25, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
56209 seconds
2024/07/28 10:59:31 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 26, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55523 seconds
2024/07/28 10:59:32 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 27, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55665 seconds
2024/07/28 10:59:32 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 28, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54476 seconds
2024/07/28 10:59:33 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 29, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53505 seconds
2024/07/28 10:59:34 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 30, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55802 seconds
2024/07/28 10:59:34 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 31, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55297 seconds
2024/07/28 10:59:35 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 32, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54373 seconds
2024/07/28 10:59:35 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 33, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
56944 seconds
2024/07/28 10:59:36 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 34, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
58507 seconds
2024/07/28 10:59:36 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 35, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
56463 seconds
2024/07/28 10:59:37 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 36, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53979 seconds
2024/07/28 10:59:37 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 37, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54980 seconds
2024/07/28 10:59:38 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 38, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
57115 seconds
2024/07/28 10:59:39 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 39, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54270 seconds
2024/07/28 10:59:39 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 40, Cur
```

```
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54465 seconds
2024/07/28 10:59:40 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 41, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53130 seconds
2024/07/28 10:59:40 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 42, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54086 seconds
2024/07/28 10:59:41 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 43, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53722 seconds
2024/07/28 10:59:41 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 44, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55603 seconds
2024/07/28 10:59:42 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 45, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
57614 seconds
2024/07/28 10:59:42 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 46, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53205 seconds
2024/07/28 10:59:43 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 47, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54745 seconds
2024/07/28 10:59:43 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 48, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53985 seconds
2024/07/28 10:59:44 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 49, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53404 seconds
2024/07/28 10:59:45 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 50, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53940 seconds
2024/07/28 10:59:45 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 51, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
52977 seconds
2024/07/28 10:59:46 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 52, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55625 seconds
2024/07/28 10:59:46 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 53, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
57695 seconds
2024/07/28 10:59:47 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 54, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53144 seconds
2024/07/28 10:59:47 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 55, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53145 seconds
2024/07/28 10:59:48 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 56, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53567 seconds
2024/07/28 10:59:48 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 57, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55577 seconds
2024/07/28 10:59:49 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 58, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54930 seconds
2024/07/28 10:59:49 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 59, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
56603 seconds
2024/07/28 10:59:50 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 60, Cur
```

```
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54493 seconds
2024/07/28 10:59:51 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 61, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
52204 seconds
2024/07/28 10:59:51 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 62, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54574 seconds
2024/07/28 10:59:52 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 63, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
56377 seconds
2024/07/28 10:59:52 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 64, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53255 seconds
2024/07/28 10:59:53 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 65, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
51370 seconds
2024/07/28 10:59:53 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 66, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
56212 seconds
2024/07/28 10:59:54 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 67, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
59163 seconds
2024/07/28 10:59:54 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 68, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55477 seconds
2024/07/28 10:59:55 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 69, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
56337 seconds
2024/07/28 10:59:56 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 70, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53375 seconds
2024/07/28 10:59:56 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 71, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
59446 seconds
2024/07/28 10:59:57 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 72, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54873 seconds
2024/07/28 10:59:57 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 73, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53428 seconds
2024/07/28 10:59:58 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 74, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53209 seconds
2024/07/28 10:59:58 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 75, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54789 seconds
2024/07/28 10:59:59 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 76, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
52751 seconds
2024/07/28 10:59:59 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 77, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54787 seconds
2024/07/28 11:00:00 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 78, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55093 seconds
2024/07/28 11:00:00 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 79, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55647 seconds
2024/07/28 11:00:01 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 80, Cur
```

```
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55778 seconds
2024/07/28 11:00:02 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 81, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53087 seconds
2024/07/28 11:00:02 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 82, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
58875 seconds
2024/07/28 11:00:03 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 83, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53346 seconds
2024/07/28 11:00:03 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 84, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54780 seconds
2024/07/28 11:00:04 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 85, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
52386 seconds
2024/07/28 11:00:04 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 86, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55384 seconds
2024/07/28 11:00:05 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 87, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53436 seconds
2024/07/28 11:00:05 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 88, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55922 seconds
2024/07/28 11:00:06 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 89, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
52886 seconds
2024/07/28 11:00:07 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 90, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54572 seconds
2024/07/28 11:00:07 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 91, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53434 seconds
2024/07/28 11:00:08 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 92, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53633 seconds
2024/07/28 11:00:08 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 93, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
55710 seconds
2024/07/28 11:00:09 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 94, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
58266 seconds
2024/07/28 11:00:09 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 95, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54892 seconds
2024/07/28 11:00:10 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 96, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54370 seconds
2024/07/28 11:00:10 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 97, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54533 seconds
2024/07/28 11:00:11 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 98, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
54580 seconds
2024/07/28 11:00:11 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 99, Cur
rent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.
53197 seconds
2024/07/28 11:00:12 AM, INFO, __main__.OriginalPOA: >>>Problem: P, Epoch: 100, Cu
```

```
rrent best: 3.668744032993749e-05, Global best: 3.668744032993749e-05, Runtime: 0.57347 seconds
Solution: [-0.2740154  -0.25495294 -0.11414563  0.6899788   0.00727423  0.7261986
5
  0.29153982  0.14319515], Fitness: 3.668744032993749e-05
```

In [19]:

```
# Define the number of features to select (e.g., 3)
k = 3

# Select the top k features with the highest weights
selected_features = np.argsort(g_best.solution)[::-1][:k]

# Print the selected features
print("Selected features:", selected_features)

# Select the corresponding columns from the training and testing data
x_train_selected = x_train.iloc[:, selected_features]
x_test_selected = x_test.iloc[:, selected_features]

# Print the shape of the selected data
print("x_train_selected shape:", x_train_selected.shape)
print("x_test_selected shape:", x_test_selected.shape)
```

```
Selected features: [5 3 6]
x_train_selected shape: (1353, 3)
x_test_selected shape: (339, 3)
```

In [20]:

```
# Print the selected feature columns
print("Selected feature columns:")
print(x_train_selected.head().to_string(header=True, index=True))
```

```
Selected feature columns:
           Low      Open    Volume
980   0.677455  0.663057  0.139475
22    0.017218  0.014952  0.008325
1260  0.952716  0.939012  0.053521
45    0.017406  0.012816  0.010161
771   0.402532  0.397230  0.205637
```

In [21]:

```
# Add the 'Close' column to the selected feature columns
x_train_selected['Close'] = y_train
x_test_selected['Close'] = y_test

# Print the updated selected feature columns
print("Updated selected feature columns:")
print(x_train_selected.head().to_string(header=True, index=True))
```

```
Updated selected feature columns:
           Low      Open    Volume     Close
980   0.677455  0.663057  0.139475  0.679581
22    0.017218  0.014952  0.008325  0.015320
1260  0.952716  0.939012  0.053521  0.937500
45    0.017406  0.012816  0.010161  0.017207
771   0.402532  0.397230  0.205637  0.395722
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_13036\700162215.py:2: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  x_train_selected['Close'] = y_train
C:\Users\DELL\AppData\Local\Temp\ipykernel_13036\700162215.py:3: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  x_test_selected['Close'] = y_test
```

In [22]:
```
x_train_selected.shape
```

Out[22]:
```
(1353, 4)
```

In [23]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

In [ ]:

In [24]:
```
w=x_train_selected.drop('Close',axis=1)
z=x_train_selected['Close']
```

In [25]:
```
w_train,w_test,z_train,z_test=train_test_split(w,z,test_size=0.2,random_state=0)
```

In [26]:
```
print(f"'w_train:{w_train.shape}")
print(f"'w_test:{w_test.shape}")
```

```
'w_train:(1082, 3)
'w_test:(271, 3)
```

In [27]:
```
models=[('LinearRegression',LinearRegression()),
        ('DecisionTree',DecisionTreeRegressor())]
```

In [28]:
```
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
```

In [29]:
```
for name,model in models:
    print(name)
    model.fit(w_train,z_train)
    z_pred=model.predict(w_test)
    print("mean squared error:",mean_squared_error(z_test,z_pred))
    print('\n')
    print("MeanAbsoluteError:",mean_absolute_error(z_test,z_pred))
    print('\n')
    print("RSquared(R2):",r2_score(z_test,z_pred))
    print('\n')
```

```
LinearRegression
mean squared error: 6.030012073534787e-05


MeanAbsoluteError: 0.004995614662627104


RSquared(R2): 0.9995040929980761


DecisionTree
mean squared error: 0.00014938054337527927


MeanAbsoluteError: 0.007163780418875161


RSquared(R2): 0.9987714973617361
```

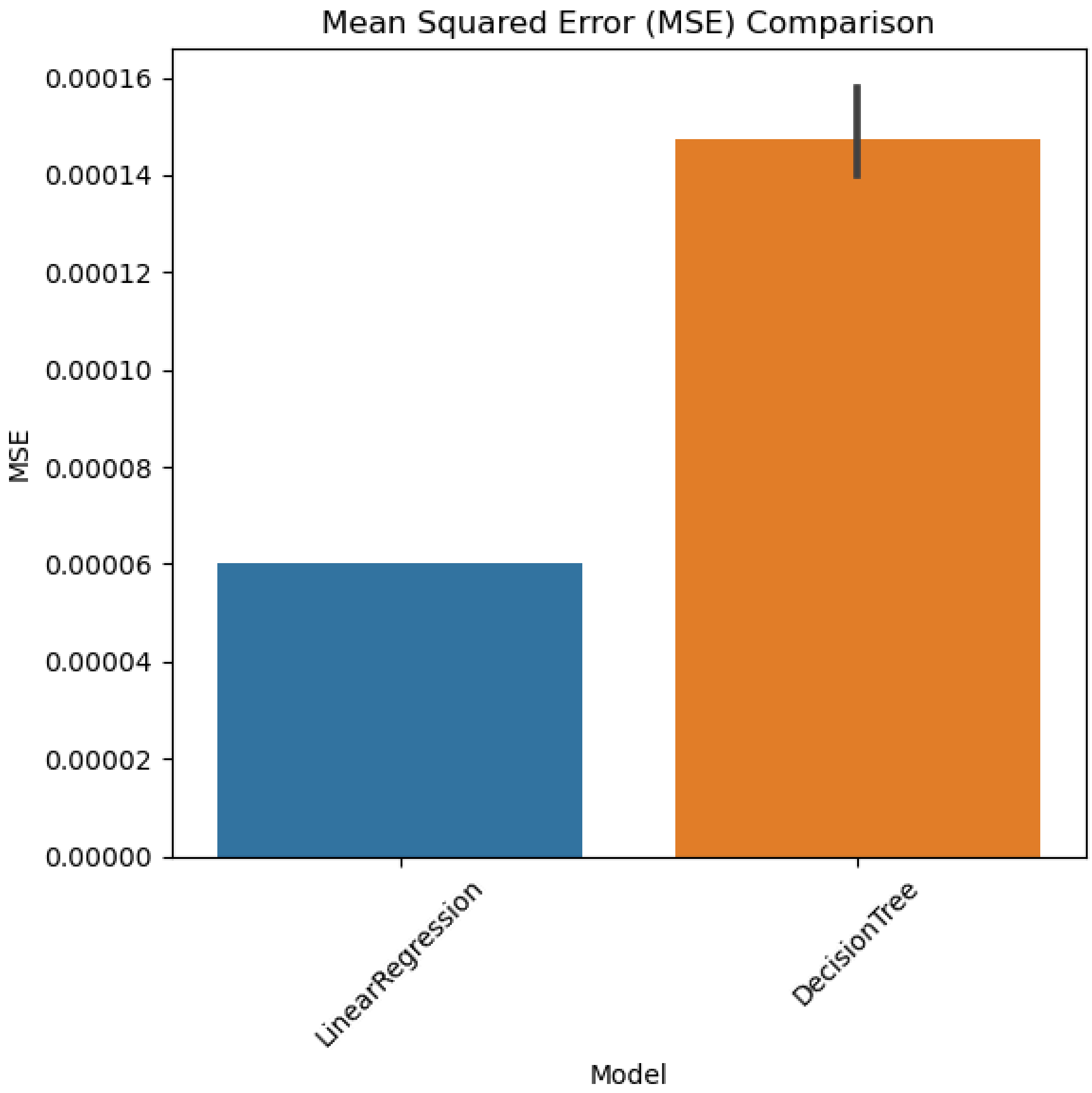
In [32]:

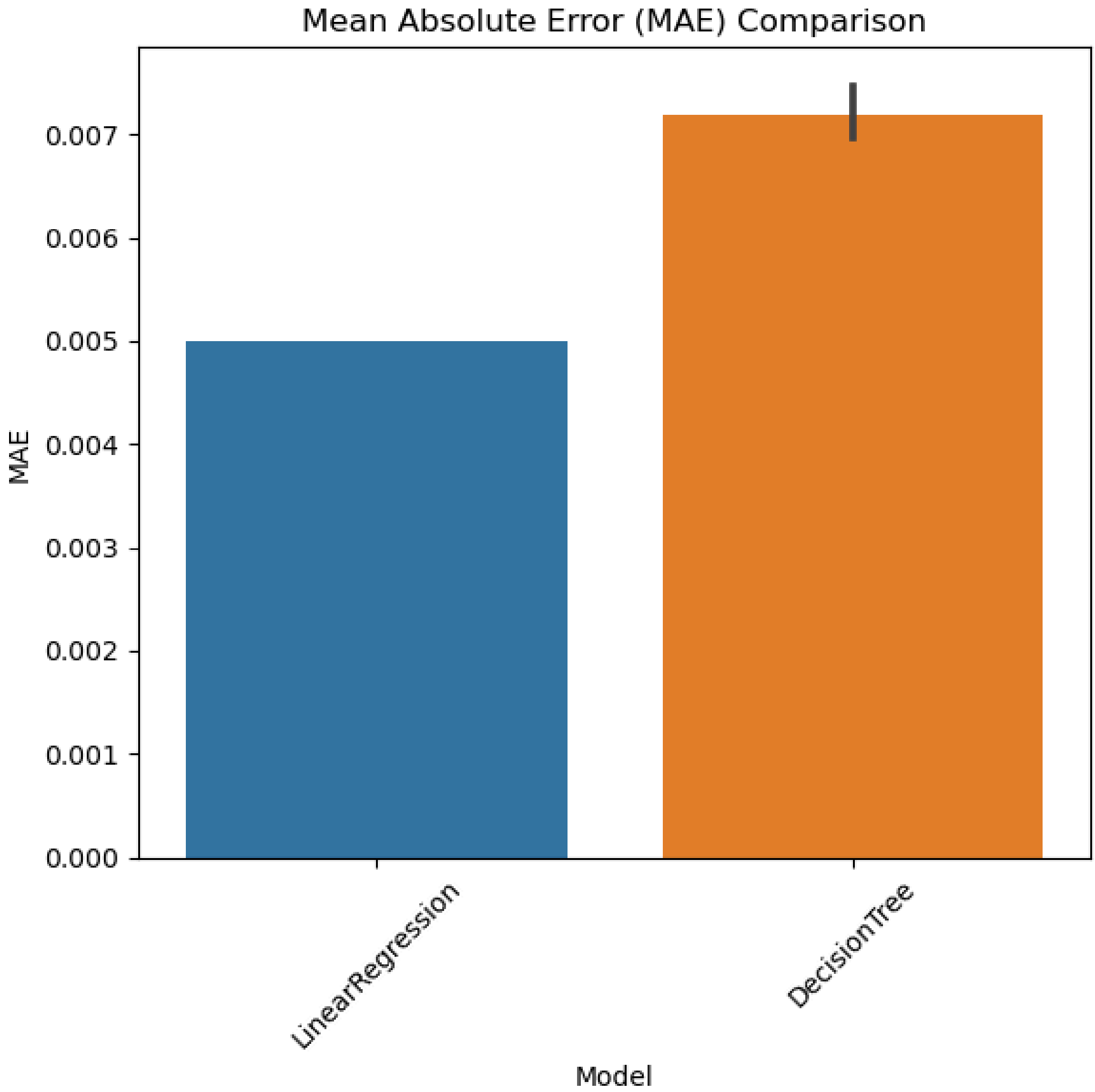
```
model_names = []
mse_scores = []
mae_scores = []
r2_scores = []
```

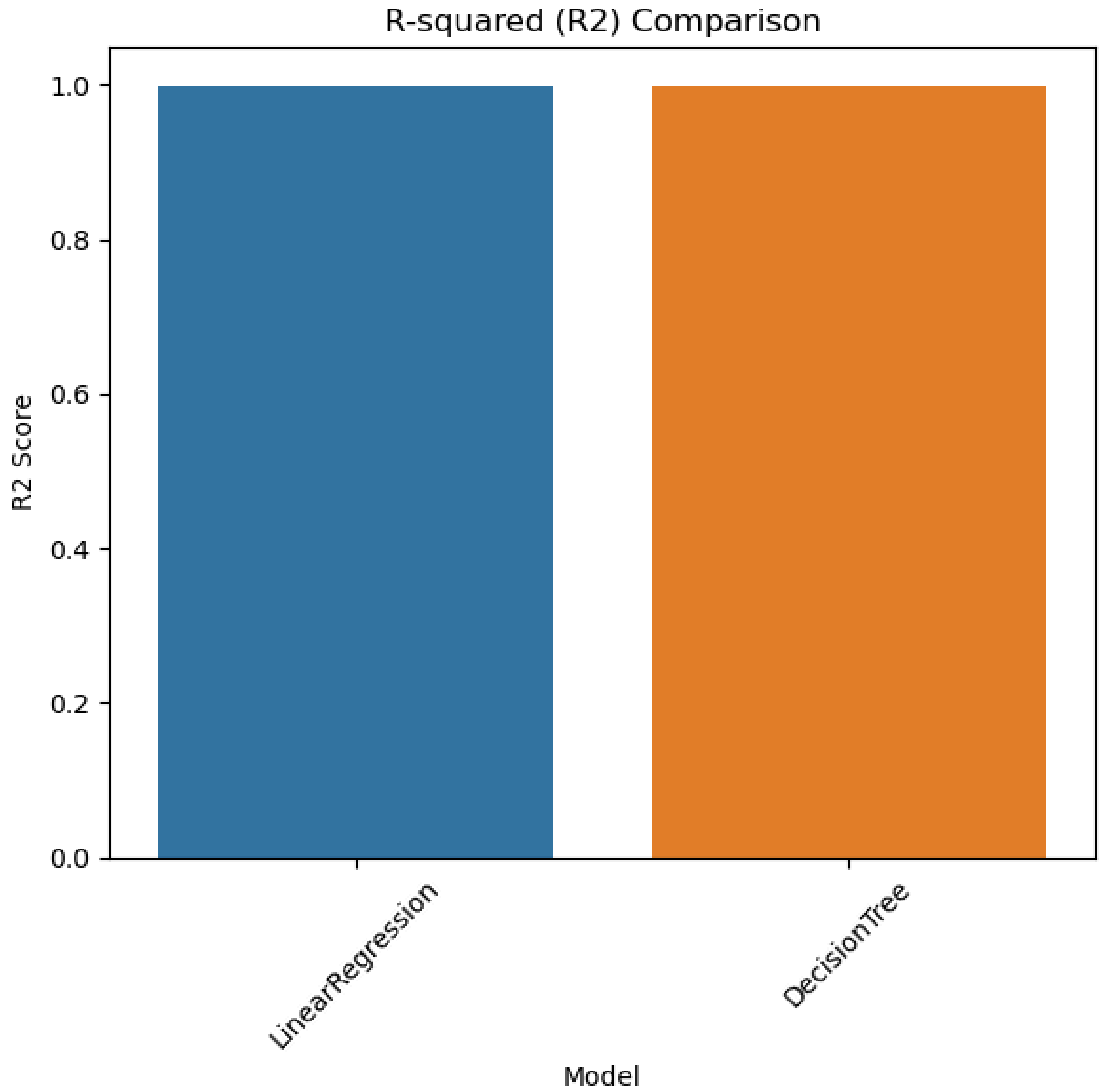
In [35]:

```
for name, model in models:
    model_names.append(name)
    model.fit(w_train, z_train)
    z_pred = model.predict(w_test)
    mse = mean_squared_error(z_test, z_pred)
    mae = mean_absolute_error(z_test, z_pred)
    r2 = r2_score(z_test, z_pred)
    mse_scores.append(mse)
    mae_scores.append(mae)
    r2_scores.append(r2)
# Bar plot for Mean Squared Error (MSE)
plt.figure(figsize=(6,6))
sns.barplot(x=model_names, y=mse_scores)
plt.title('Mean Squared Error (MSE) Comparison')
plt.xlabel('Model')
plt.ylabel('MSE')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Bar plot for Mean Absolute Error (MAE)
plt.figure(figsize=(6, 6))
sns.barplot(x=model_names, y=mae_scores)
plt.title('Mean Absolute Error (MAE) Comparison')
plt.xlabel('Model')
plt.ylabel('MAE')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Bar plot for R-squared (R2)
plt.figure(figsize=(6, 6))
sns.barplot(x=model_names, y=r2_scores)
plt.title('R-squared (R2) Comparison')
```

```
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




Mean Absolute Error (MAE) Comparison
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
MAE
LinearRegression
DecisionTree
Model


R-squared (R2) Comparison
R2 Score
1.0
0.8
0.6
0.4
0.2
0.0
LinearRegression
DecisionTree
Model

```
In [ ]:
```