```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
data = pd.read_csv('ENB2012_data.csv')
data.columns=["Relative Compactness", "Surface Area", "Wall Area",
"Roof Area", "Overall Height", "Orientation",
              "Glazing Area", "Glazing Area Distribution", "Heating
Load", "Cooling Load"]
data.head()
```

```
   Relative Compactness  Surface Area  Wall Area  Roof Area  Overall
Height  \
0                  0.98         514.5      294.0     110.25
7.0
1                  0.98         514.5      294.0     110.25
7.0
2                  0.98         514.5      294.0     110.25
7.0
3                  0.98         514.5      294.0     110.25
7.0
4                  0.90         563.5      318.5     122.50
7.0

   Orientation  Glazing Area  Glazing Area Distribution  Heating Load
\
0          2.0           0.0                        0.0         15.55

1          3.0           0.0                        0.0         15.55

2          4.0           0.0                        0.0         15.55

3          5.0           0.0                        0.0         15.55

4          2.0           0.0                        0.0         20.84


   Cooling Load
0         21.33
1         21.33
2         21.33
3         21.33
4         28.28
```

Cleaning the data

```python
data.shape
```

```
(1296, 10)
```

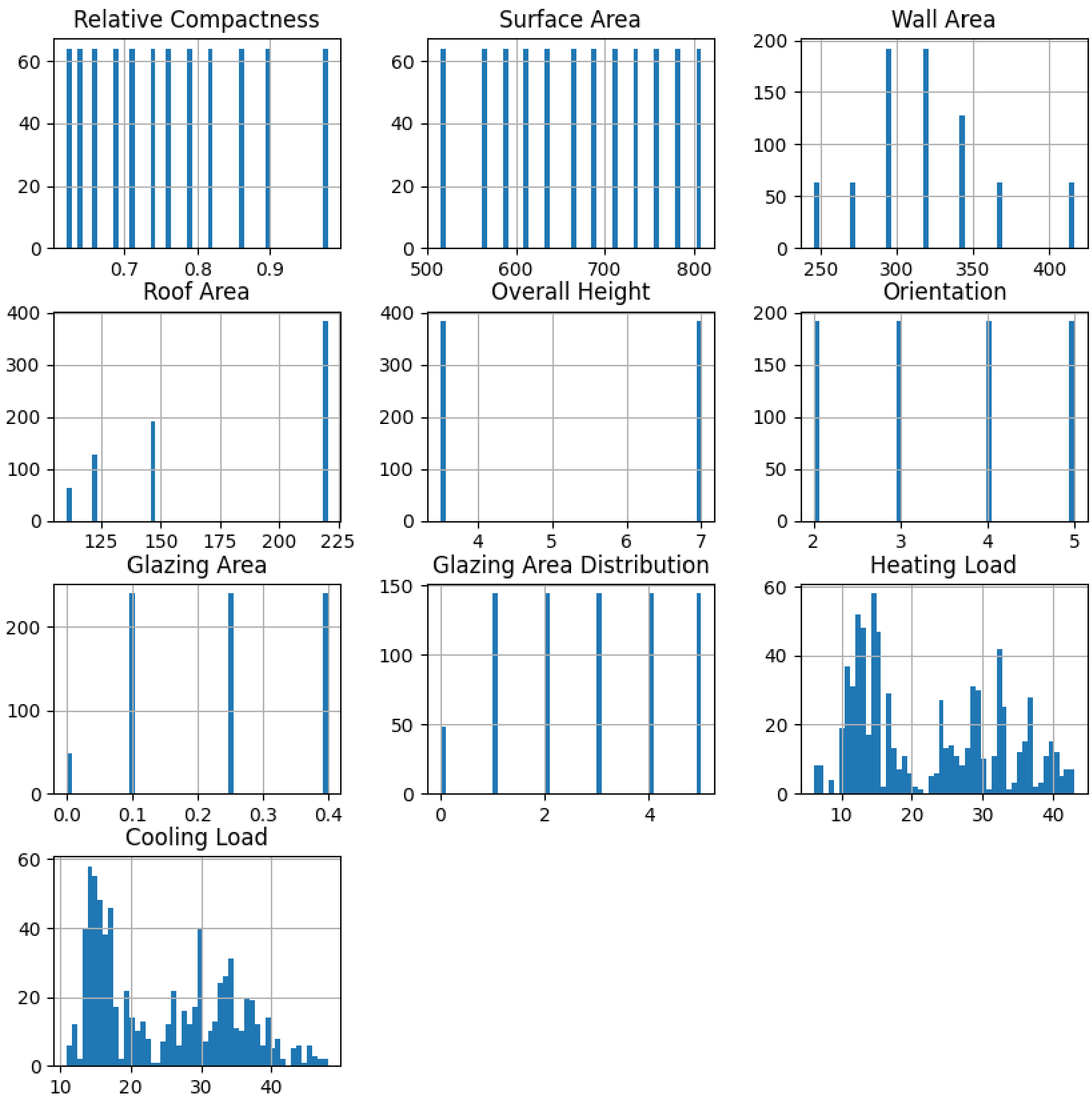
```python
data.isnull().sum()
```

```
Relative Compactness         528
Surface Area                 528
Wall Area                    528
Roof Area                    528
Overall Height               528
Orientation                  528
Glazing Area                 528
Glazing Area Distribution    528
Heating Load                 528
Cooling Load                 528
dtype: int64
```

```python
#We are going to drop these datas
```

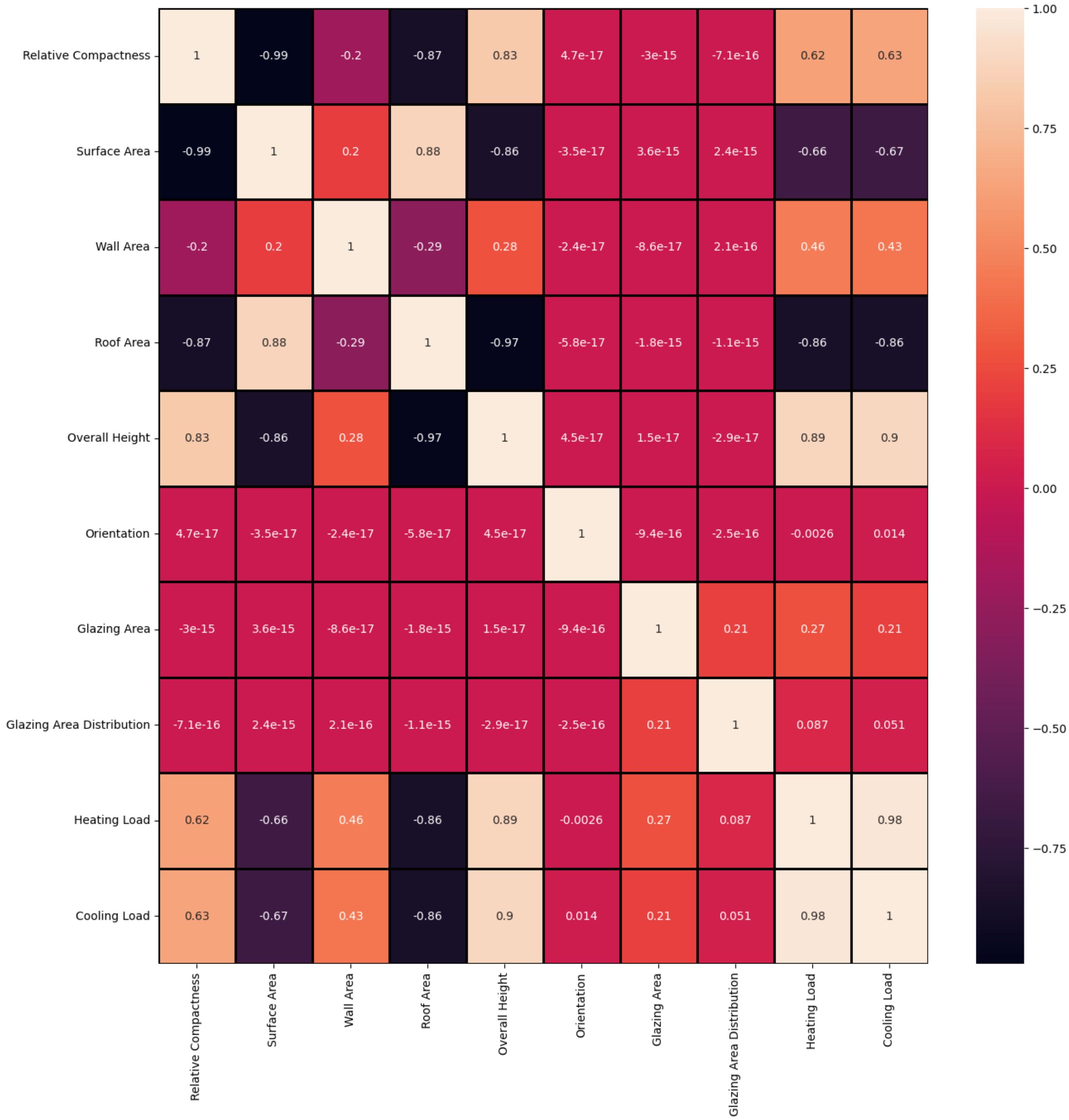
```python
data = data.dropna()
```

```python
data.hist(bins=50, figsize=(10,10))
plt.show()
```




```
plt.figure(figsize=(15,15))
corr_mat = data.corr()
sns.heatmap(corr_mat, annot = True, linewidth = 1, linecolor =
'black')

<Axes: >
```




```
data.corr()

                                  Relative Compactness  Surface Area     Wall
Area  \
Relative Compactness                      1.000000e+00 -9.919015e-01 -
2.037817e-01
Surface Area                             -9.919015e-01  1.000000e+00
1.955016e-01
Wall Area                                -2.037817e-01  1.955016e-01
1.000000e+00
```

```
Roof Area                                  -8.688234e-01  8.807195e-01 -
2.923165e-01
Overall Height                              8.277473e-01 -8.581477e-01
2.809757e-01
Orientation                                 4.678592e-17 -3.459372e-17 -
2.429499e-17
Glazing Area                               -2.960552e-15  3.636925e-15 -
8.567455e-17
Glazing Area Distribution                  -7.107006e-16  2.438409e-15
2.067384e-16
Heating Load                                6.222722e-01 -6.581202e-01
4.556712e-01
Cooling Load                                6.343391e-01 -6.729989e-01
4.271170e-01

                              Roof Area  Overall Height   Orientation
\
Relative Compactness      -8.688234e-01    8.277473e-01  4.678592e-17

Surface Area               8.807195e-01   -8.581477e-01 -3.459372e-17

Wall Area                 -2.923165e-01    2.809757e-01 -2.429499e-17

Roof Area                  1.000000e+00   -9.725122e-01 -5.830058e-17

Overall Height            -9.725122e-01    1.000000e+00  4.492205e-17

Orientation               -5.830058e-17    4.492205e-17  1.000000e+00

Glazing Area              -1.759011e-15    1.489134e-17 -9.406007e-16

Glazing Area Distribution -1.078071e-15   -2.920613e-17 -2.549352e-16

Heating Load              -8.618283e-01    8.894307e-01 -2.586534e-03

Cooling Load              -8.625466e-01    8.957852e-01  1.428960e-02


                           Glazing Area  Glazing Area Distribution  \
Relative Compactness      -2.960552e-15              -7.107006e-16
Surface Area               3.636925e-15               2.438409e-15
Wall Area                 -8.567455e-17               2.067384e-16
Roof Area                 -1.759011e-15              -1.078071e-15
Overall Height             1.489134e-17              -2.920613e-17
Orientation               -9.406007e-16              -2.549352e-16
Glazing Area               1.000000e+00               2.129642e-01
Glazing Area Distribution  2.129642e-01               1.000000e+00
Heating Load               2.698410e-01               8.736759e-02
Cooling Load               2.075050e-01               5.052512e-02
```

```
                             Heating Load  Cooling Load
Relative Compactness             0.622272      0.634339
Surface Area                    -0.658120     -0.672999
Wall Area                        0.455671      0.427117
Roof Area                       -0.861828     -0.862547
Overall Height                   0.889431      0.895785
Orientation                     -0.002587      0.014290
Glazing Area                     0.269841      0.207505
Glazing Area Distribution        0.087368      0.050525
Heating Load                     1.000000      0.975862
Cooling Load                     0.975862      1.000000
```

```python
#Standard Scaling

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data = scaler.fit_transform(data)
data = pd.DataFrame(data)

data.head()
```

```
   Relative Compactness  Surface Area  Wall Area  Roof Area  Overall 
Height  \
0              1.000000      0.000000   0.285714   0.000000           
1.0   
1              1.000000      0.000000   0.285714   0.000000           
1.0   
2              1.000000      0.000000   0.285714   0.000000           
1.0   
3              1.000000      0.000000   0.285714   0.000000           
1.0   
4              0.777778      0.166667   0.428571   0.111111           
1.0   

   Orientation  Glazing Area  Glazing Area Distribution  Heating Load 
\
0     0.000000           0.0                        0.0      0.257212   

1     0.333333           0.0                        0.0      0.257212   

2     0.666667           0.0                        0.0      0.257212   

3     1.000000           0.0                        0.0      0.257212   

4     0.000000           0.0                        0.0      0.399838   


   Cooling Load  
0      0.280905  
1      0.280905  
2      0.280905  
```

```
3      0.280905
4      0.468085

data.columns=["Relative Compactness", "Surface Area", "Wall Area", 
"Roof Area", "Overall Height", "Orientation", 
              "Glazing Area", "Glazing Area Distribution", "Heating 
Load", "Cooling Load"]

from sklearn.model_selection import train_test_split

data_heat = data.drop(columns=['Cooling Load'])
x_heat = data.iloc[:,:-1]
y_heat = data['Heating Load']
data_cool = data.drop(columns=['Heating Load'])
x_cool = data.iloc[:,:-1]
y_cool = data['Cooling Load']

x_trainh,x_testh,y_train,y_test = 
train_test_split(x_heat,y_heat,test_size = 0.2,random_state = 42)

import numpy as np

class LinearRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def fit(self, x, y):
        """
        Train the linear regression model using gradient descent.
        X: Input features (m samples, n features).
        y: Target values (m samples).
        """
        m, n = x.shape  # Number of samples (m) and features (n)
        self.weights = np.zeros(n)  # Initialize weights to zeros
        self.bias = 0  # Initialize bias to zero

        # Gradient Descent
        for i in range(self.num_iterations):
            # Calculate predictions
            y_pred = np.dot(x, self.weights) + self.bias

            # Compute gradients
            dw = (1 / m) * np.dot(x.T, (y_pred - y))  # Gradient of 
weights
            db = (1 / m) * np.sum(y_pred - y)  # Gradient of bias

            # Update parameters
            self.weights -= self.learning_rate * dw
```

```python
            self.bias -= self.learning_rate * db

            # Optionally log the loss every 100 iterations
            if i % 100 == 0:
                loss = self._mean_squared_error(y, y_pred)
                print(f"Iteration {i}, Loss: {loss:.4f}")

    def predict(self, x):
        """
        Predict target values for the input data.
        X: Input features.
        """
        return np.dot(x, self.weights) + self.bias

    def _mean_squared_error(self, y_true, y_pred):
        """
        Calculate the Mean Squared Error (MSE).
        """
        m = len(y_true)
        return (1 / (2 * m)) * np.sum((y_true - y_pred) ** 2)


lr = LinearRegression()
lr.fit(x_trainh,y_trainh)
```

```
Iteration 0, Loss: 0.1314
Iteration 100, Loss: 0.0123
Iteration 200, Loss: 0.0065
Iteration 300, Loss: 0.0045
Iteration 400, Loss: 0.0037
Iteration 500, Loss: 0.0033
Iteration 600, Loss: 0.0031
Iteration 700, Loss: 0.0029
Iteration 800, Loss: 0.0027
Iteration 900, Loss: 0.0025
```

```python
lr.predict(x_testh)
```

```
array([0.31264796, 0.19661021, 0.75313694, 0.79118424, 0.25894376,
       0.56995218, 0.60003386, 0.67243593, 0.29970241, 0.6234076 ,
       0.31453626, 0.71602794, 0.64763214, 0.11769758, 0.31844015,
       0.77266095, 0.79526793, 0.17003058, 0.23027899, 0.71390028,
       0.68819616, 0.7650688 , 0.14383708, 0.6637388 , 0.21030756,
       0.65997486, 0.76576517, 0.69599437, 0.23936694, 0.25586847,
       0.0935149 , 0.15934566, 0.2529412 , 0.61409808, 0.64213592,
       0.67369552, 0.75141761, 0.64583926, 0.76478897, 0.10662259,
       0.69588482, 0.16803721, 0.04086191, 0.7009893 , 0.05780816,
       0.0431583 , 0.07855469, 0.10511404, 0.61870977, 0.71936387,
       0.76340446, 0.22498387, 0.20938072, 0.73793282, 0.19617363,
       0.20466209, 0.15602555, 0.63458024, 0.85827366, 0.77430007,
       0.27139484, 0.32798626, 0.23187304, 0.57612347, 0.70078715,
```

```
       0.67153702, 0.65942289, 0.13948672, 0.20923218, 0.16093725,
       0.61920724, 0.76817222, 0.20755745, 0.55975863, 0.6923202 ,
       0.15638264, 0.70970292, 0.61604304, 0.73482908, 0.47249986,
       0.61325758, 0.73086184, 0.12141545, 0.24903991, 0.15603054,
       0.23889314, 0.57219752, 0.76234902, 0.64505796, 0.24492283,
       0.18678499, 0.61693619, 0.12362944, 0.21764804, 0.78005084,
       0.22133177, 0.57833065, 0.64430223, 0.12452213, 0.10245403,
       0.83650858, 0.66738995, 0.08569425, 0.64898786, 0.63617573,
       0.30694842, 0.5465312 , 0.31992319, 0.25683785, 0.15044068,
       0.70558543, 0.712408  , 0.59451613, 0.27324055, 0.16310589,
       0.23215115, 0.15065393, 0.10079013, 0.20755091, 0.52447163,
       0.63686437, 0.15786855, 0.60502908, 0.25210824, 0.62782228,
       0.32593535, 0.26490745, 0.66917631, 0.61446311, 0.83371429,
       0.50208345, 0.71770521, 0.24999366, 0.21867331, 0.23269005,
       0.3231513 , 0.71723665, 0.80323776, 0.22077199, 0.30539541,
       0.10904018, 0.86201799, 0.68873734, 0.71161318, 0.77558856,
       0.14748264, 0.11053702, 0.61495847, 0.77868216, 0.71610997,
       0.76096814, 0.6239441 , 0.61769913, 0.18599598])
```

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score

lr_model = LinearRegression()
lr_model.fit(x_trainh, y_trainh)
y_pred_lr = lr_model.predict(x_testh)

ridge_model = Ridge(alpha=1.0)
ridge_model.fit(x_trainh, y_trainh)
y_pred_ridge = ridge_model.predict(x_testh)

lasso_model = Lasso(alpha=0.1)
lasso_model.fit(x_trainh, y_trainh)
y_pred_lasso = lasso_model.predict(x_testh)

lr_model.score(x_testh,y_test),
ridge_model.score(x_testh,y_test),lasso_model.score(x_testh,y_test)
```

```
(1.0, 0.996548203430539, 0.23762460452011158)
```

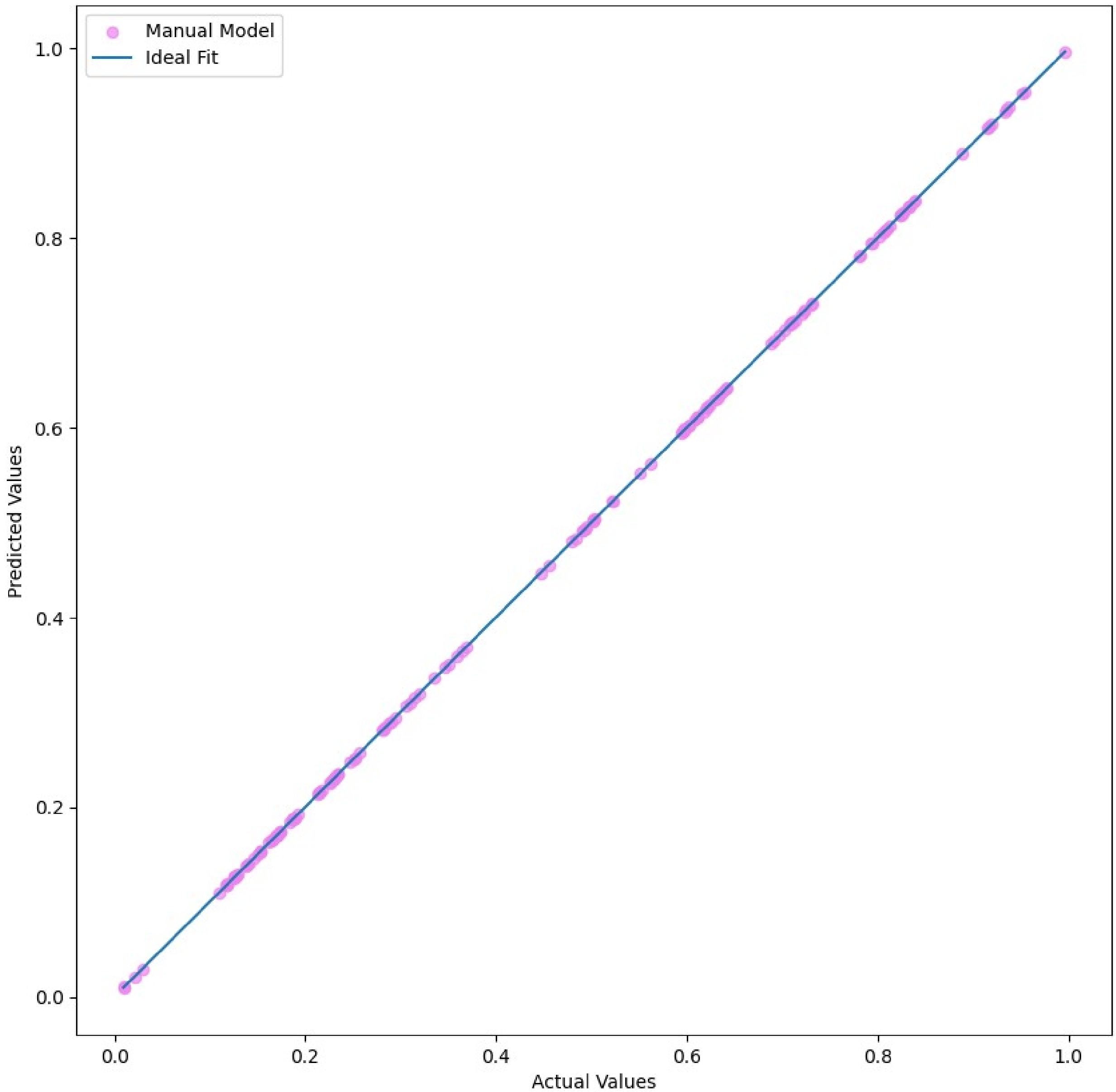
```python
#The score provided by the lr and the ridge model have great
accuracies for the testing set

lr_model.score(x_trainh,y_train),
ridge_model.score(x_trainh,y_train),lasso_model.score(x_trainh,y_train
)
```

```
(1.0, 0.9967934826293361, 0.24522041903055625)
```

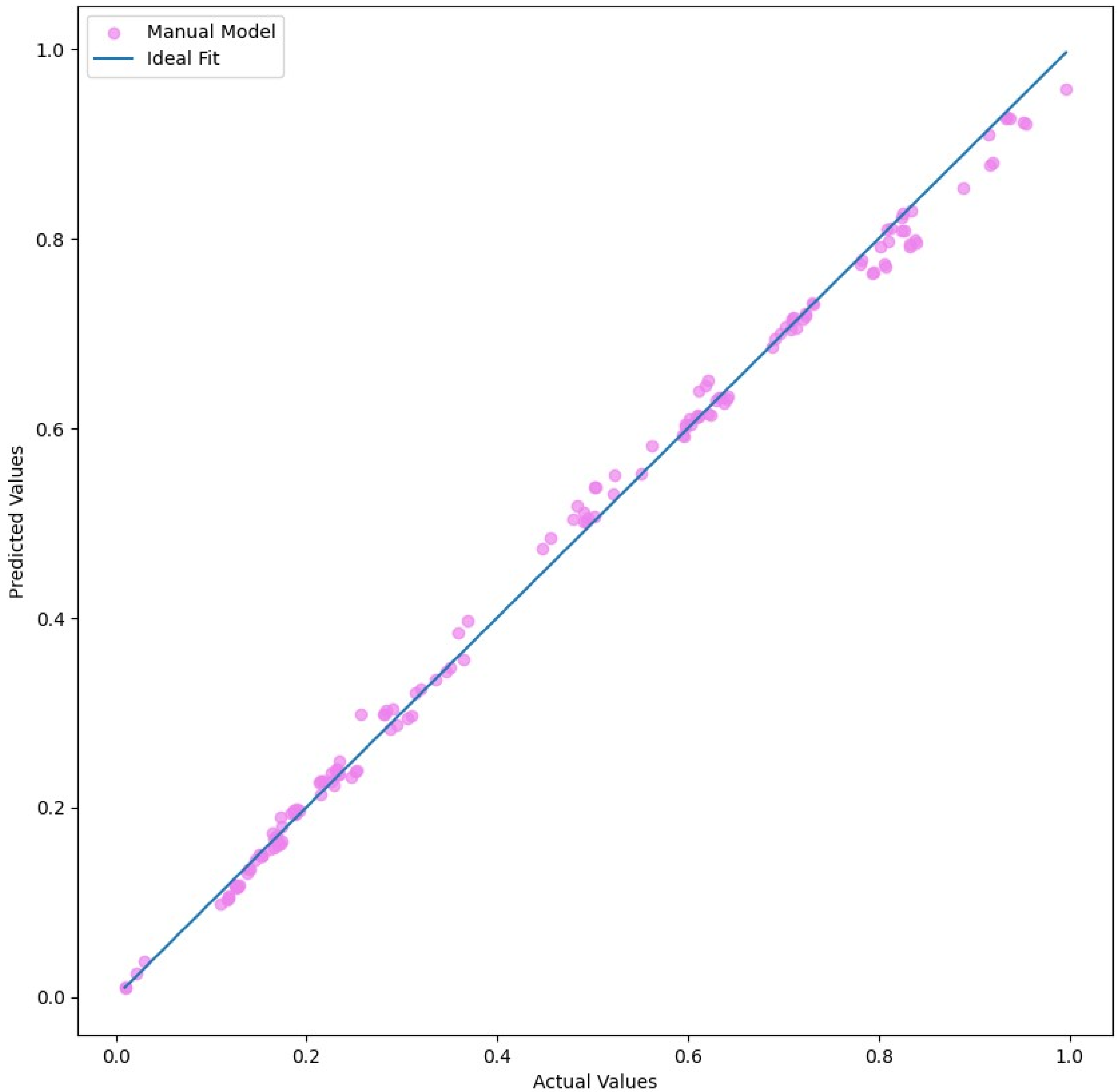
```python
plt.figure(figsize=(10, 10))
plt.scatter(y_test, y_pred_lr, label='Manual Model', alpha=0.7,
color='violet')
```

```python
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
label="Ideal Fit", )
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.legend()
plt.show()
```




```python
plt.figure(figsize=(10, 10))
plt.scatter(y_test, y_pred_ridge, label='Manual Model', alpha=0.7,
color='violet')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
label="Ideal Fit", )
```

```
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.legend()
plt.show()
```



Insights and Recommendations:

Insight: A large negative coefficient (-2.549774e+01) indicates that increasing a building's compactness significantly reduces its heating load. Recommendation: Focus on designing compact structures to minimize exposed surface area, thereby improving energy efficiency.

Insight: A large positive coefficient (1.139458e+15) shows that as surface area increases, heating load rises dramatically. Recommendation: Optimize the building's surface area to limit heat loss.

Avoid designs with excessive exposed areas and incorporate materials with high insulation properties.

Insight: A large negative coefficient (-6.646837e+14) suggests that increasing the wall area, when properly insulated, reduces heating load. Recommendation: Use insulation materials with high R-values for walls and design to minimize thermal bridging for better energy efficiency.

Insight: A large negative coefficient (-8.545933e+14) indicates that reducing heat loss through roofs with proper materials and insulation lowers the heating load. Recommendation: Install energy-efficient roofing systems, such as green roofs or cool roofs, to improve thermal resistance and energy savings.