Lab 7: Support vector machines

In [49]:
```
ID = 1631938
Name = 'MIN SOE HTUT'
```

Skeleton Code

In [50]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('https://raw.githubusercontent.com/martianunlimited/compx310_
datasets/main/housing.csv')
X = df.drop(columns=['median_house_value'])
X.dropna(inplace=True)
y = df['median_house_value']
y = y[X.index]
```

Task 1. Define a function preprocess_features that takes in X as the

In [51]:
```
def preprocess_features(X):
    # Handling missing values for numeric features only
    # Select only the numeric columns from the dataset
    numeric_features = X.select_dtypes(include=['number'])

    # Fill missing values (NaNs) in numeric columns with the mean of each colu
mn
    numeric_features.fillna(numeric_features.mean(), inplace=True)

    # Encoding categorical variables using one-hot encoding
    # This converts categorical columns into dummy/indicator variables, removi
ng the first category to avoid multicollinearity
    X = pd.get_dummies(X, drop_first=True)

    # Return the preprocessed DataFrame with filled numeric features and encod
ed categorical features
    return X
```

Task 2. Define a function run_reg that takes a regressor and X_train

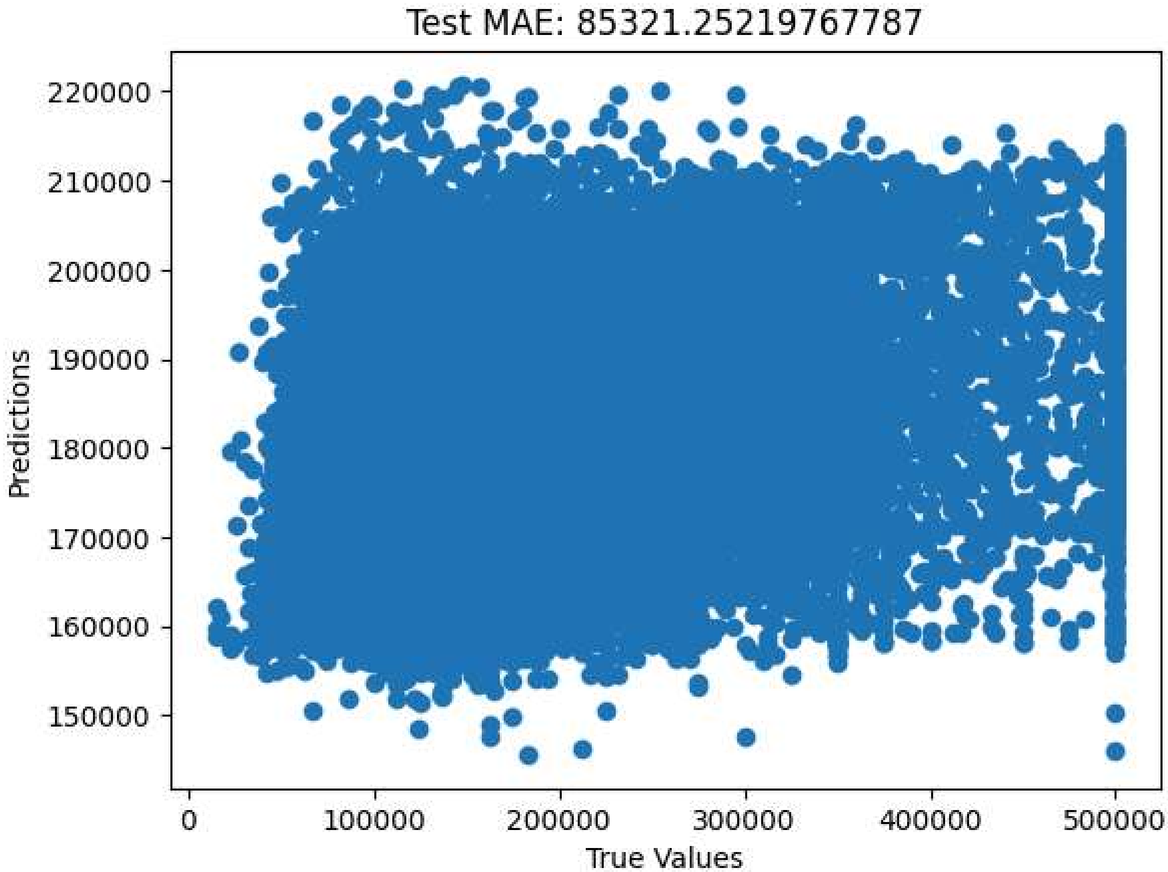
In [52]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

def run_reg(regressor, X_train, X_test, y_train, y_test):
    # Fit the provided regressor to the training data (X_train, y_train)
    regressor.fit(X_train, y_train)

    # Predict the target values for the test set (X_test)
    predictions = regressor.predict(X_test)

    # Apply thresholds to the predictions, clipping values outside the range
[15000, 500000]
    predictions = predictions.clip(15000, 500000)

    # Calculate the Mean Absolute Error (MAE) between the true and predicted v
alues
    mae = mean_absolute_error(y_test, predictions)

    # Create a scatter plot of the true values vs the predicted values
    plt.scatter(y_test, predictions)

    # Set the plot title to display the MAE
    plt.title(f"Test MAE: {mae}")

    # Label the x-axis as "True Values"
    plt.xlabel("True Values")

    # Label the y-axis as "Predictions"
    plt.ylabel("Predictions")

    # Display the plot
    plt.show()

    # Return the MAE value
    return mae
```

Skeleton Code

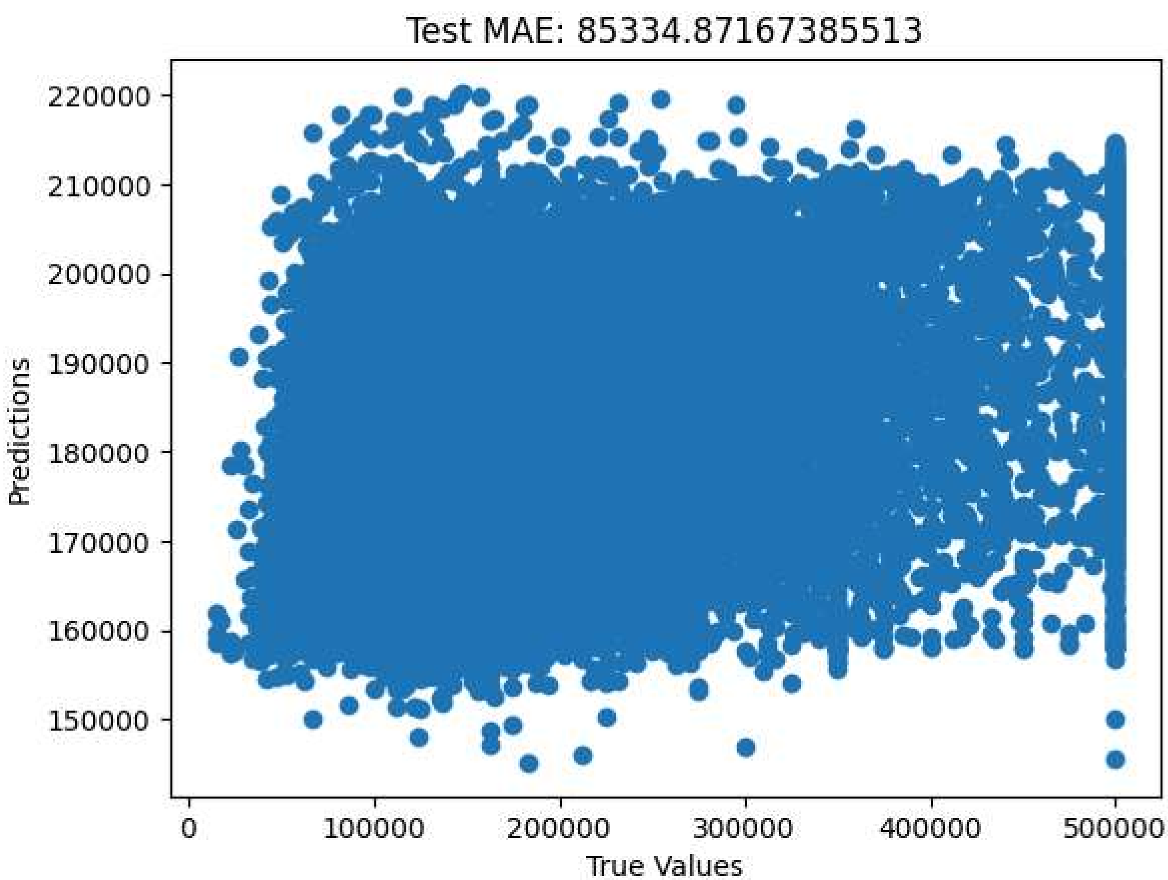
In [53]:
```
X = preprocess_features(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, rando
m_state=ID)
```

Task 3: Call run_reg for all possible combinations of the following

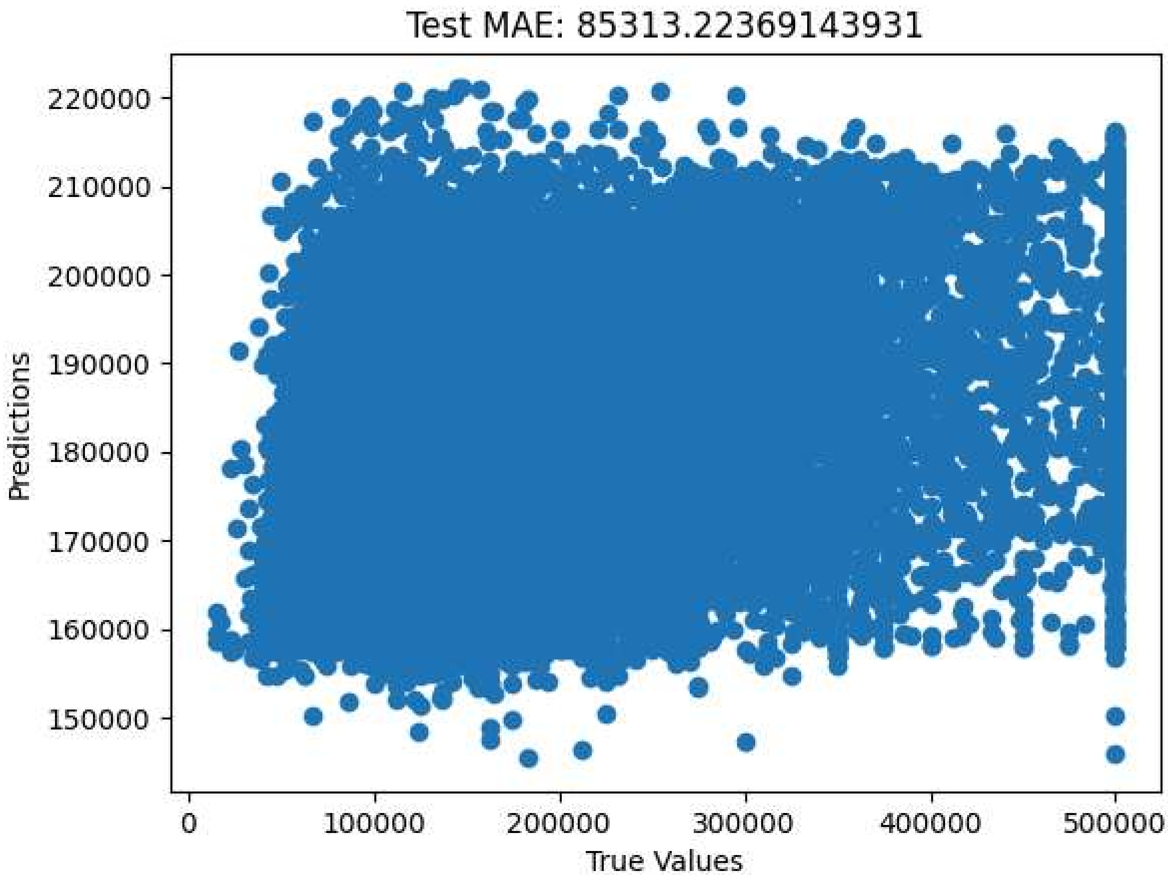
In [54]:
```
from sklearn.model_selection import ParameterGrid
import numpy as np

# Define the hyperparameter grid for SVR (C: cost, epsilon: error margin)
param_grid = {
    'C': [1000, 10000, 100000, 1000000, 10000000],       # Different values fo
r regularization parameter (C)
    'epsilon': [2000, 5000, 10000, 20000, 50000, 100000]  # Different values f
or epsilon (error tolerance)
}

# Initialize an empty array to store the MAE values for each combination of C
and epsilon
maes = np.zeros((len(param_grid['C']), len(param_grid['epsilon'])))

# Loop through each combination of C and epsilon in the param_grid
for i, C in enumerate(param_grid['C']):
    for j, epsilon in enumerate(param_grid['epsilon']):
        # Instantiate an SVR model with the current C and epsilon
        reg = SVR(kernel='rbf', C=C, epsilon=epsilon)

        # Run the regression model using the run_reg function and calculate MA
E
        mae = run_reg(reg, X_train, X_test, y_train, y_test)

        # Store the calculated MAE in the 'maes' array
        maes[i, j] = mae

        # Print the current combination of C, epsilon, and the corresponding M
AE
        print(f"C: {C}, Epsilon: {epsilon}, MAE: {mae}")
```



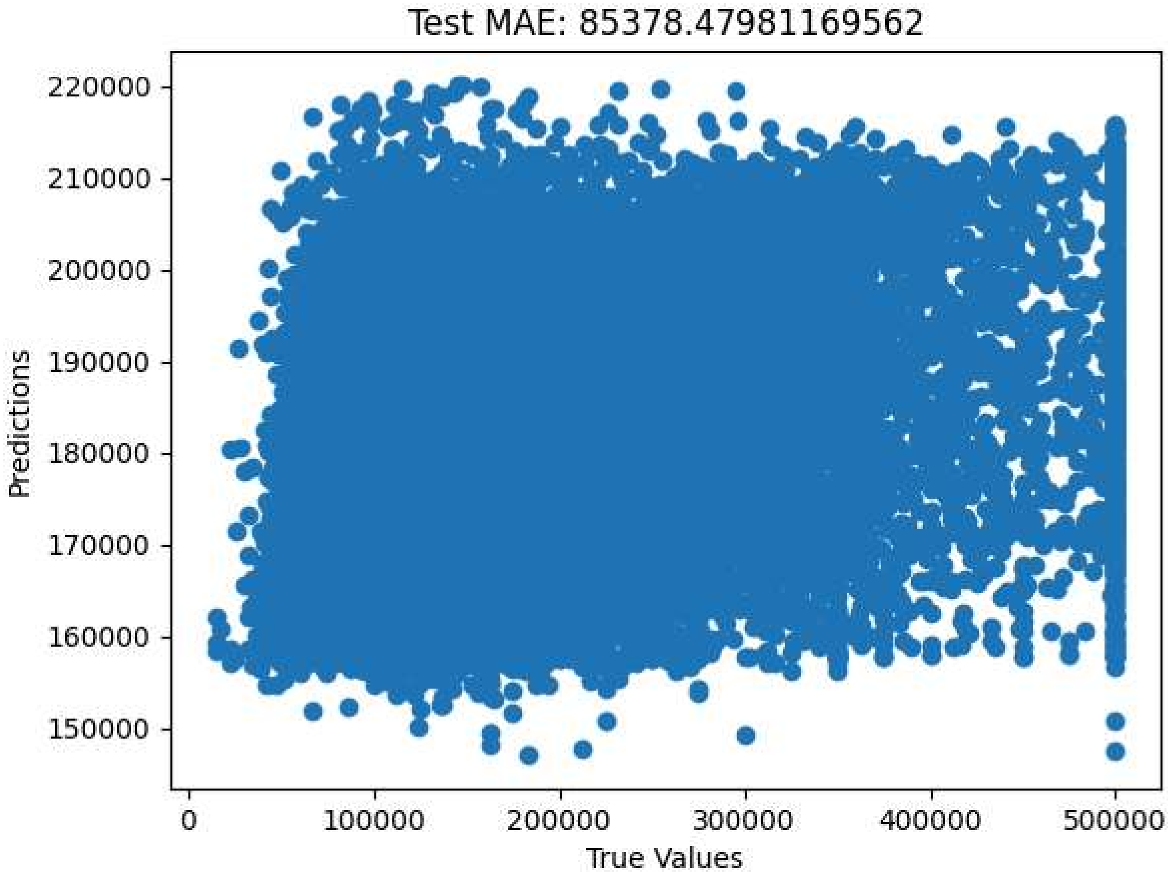
C: 1000, Epsilon: 2000, MAE: 85321.25219767787



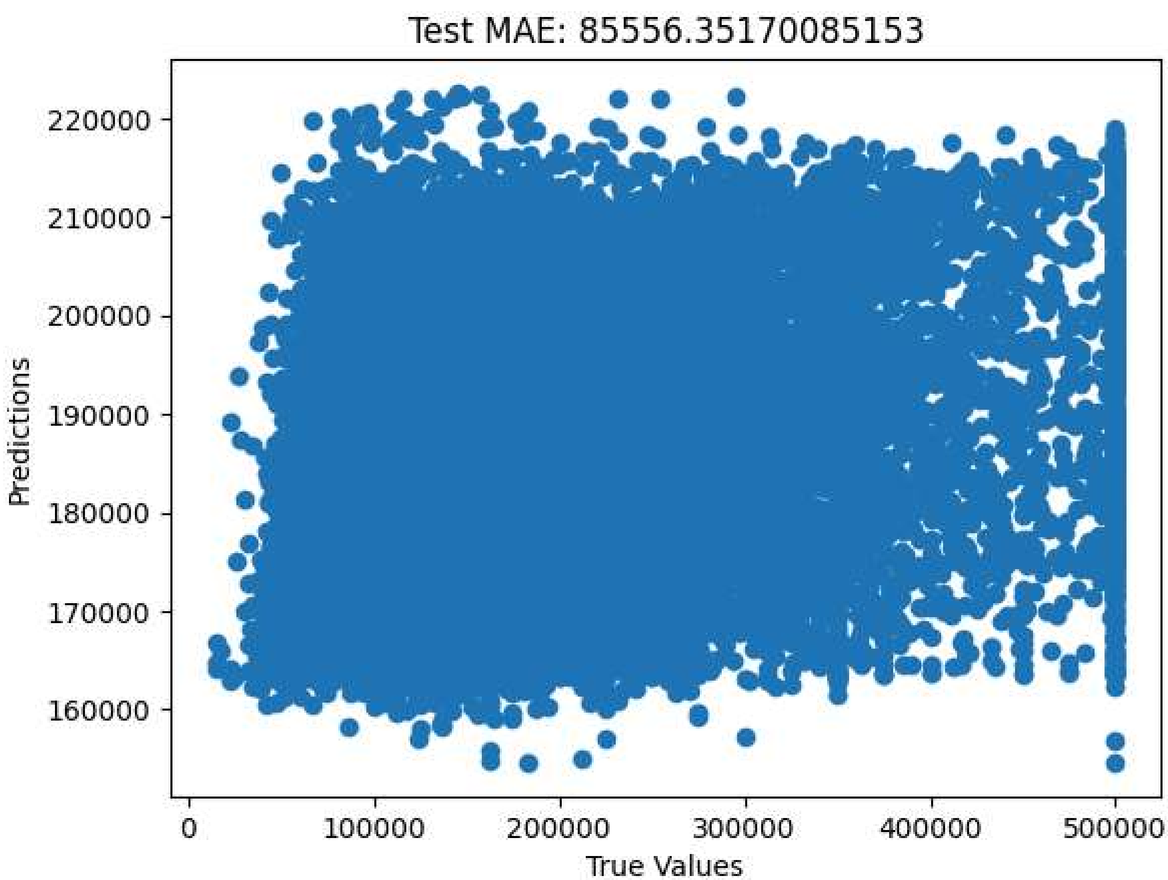
```
C: 1000, Epsilon: 5000, MAE: 85334.87167385513
```



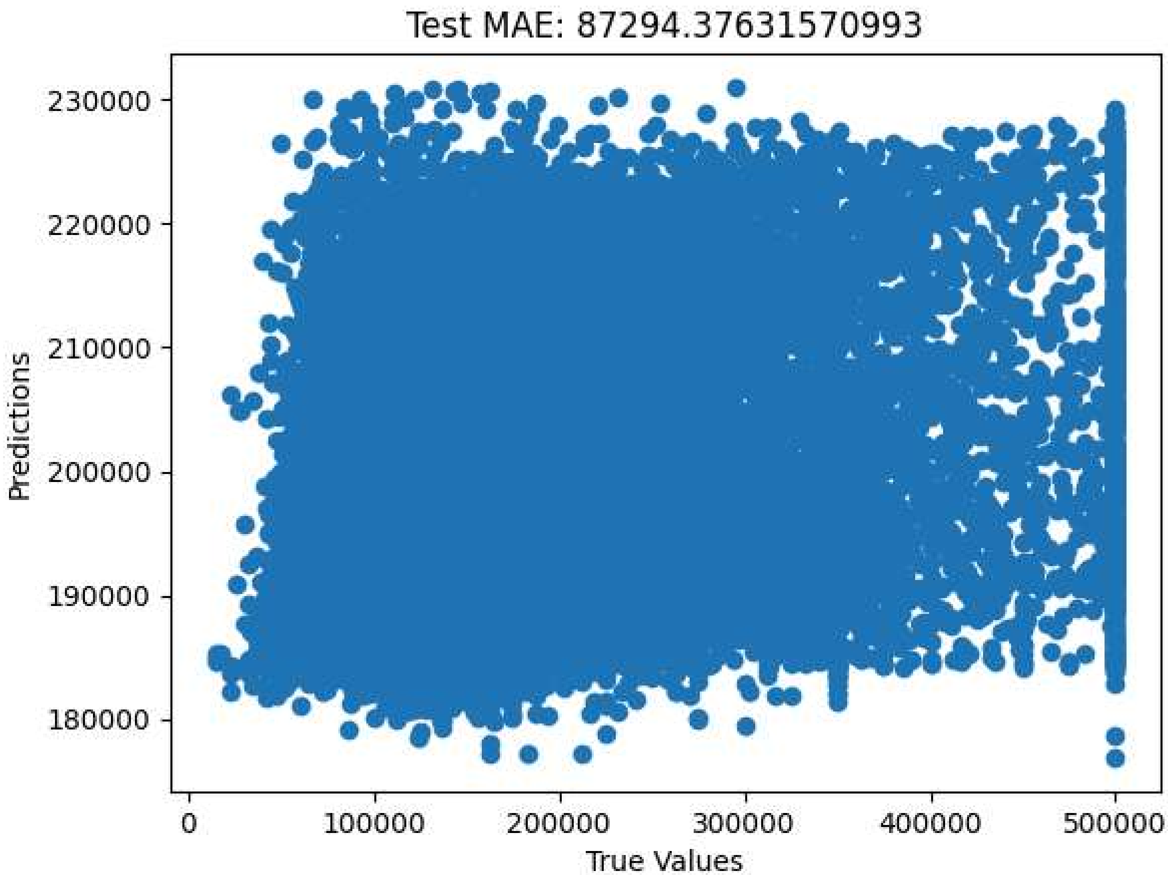

```
C: 1000, Epsilon: 10000, MAE: 85313.22369143931
```



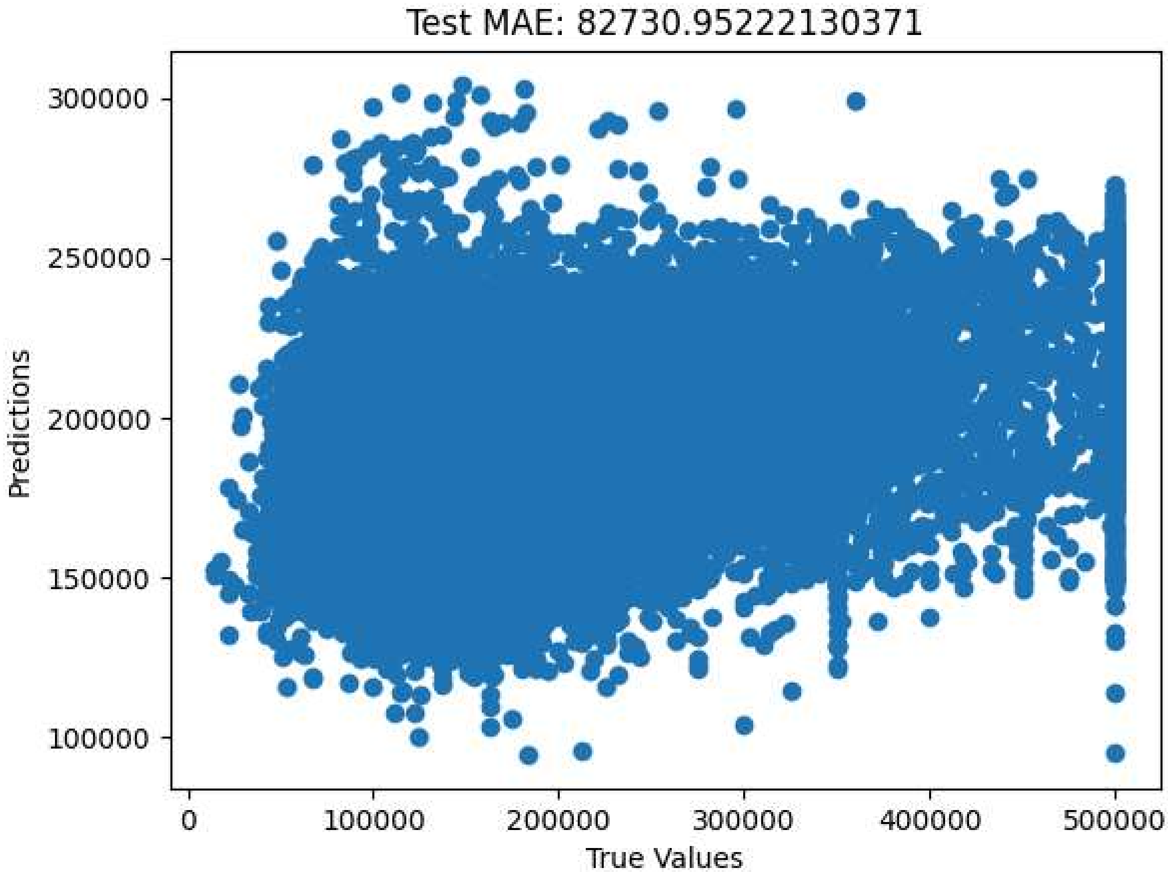

```
C: 1000, Epsilon: 20000, MAE: 85378.47981169562
```



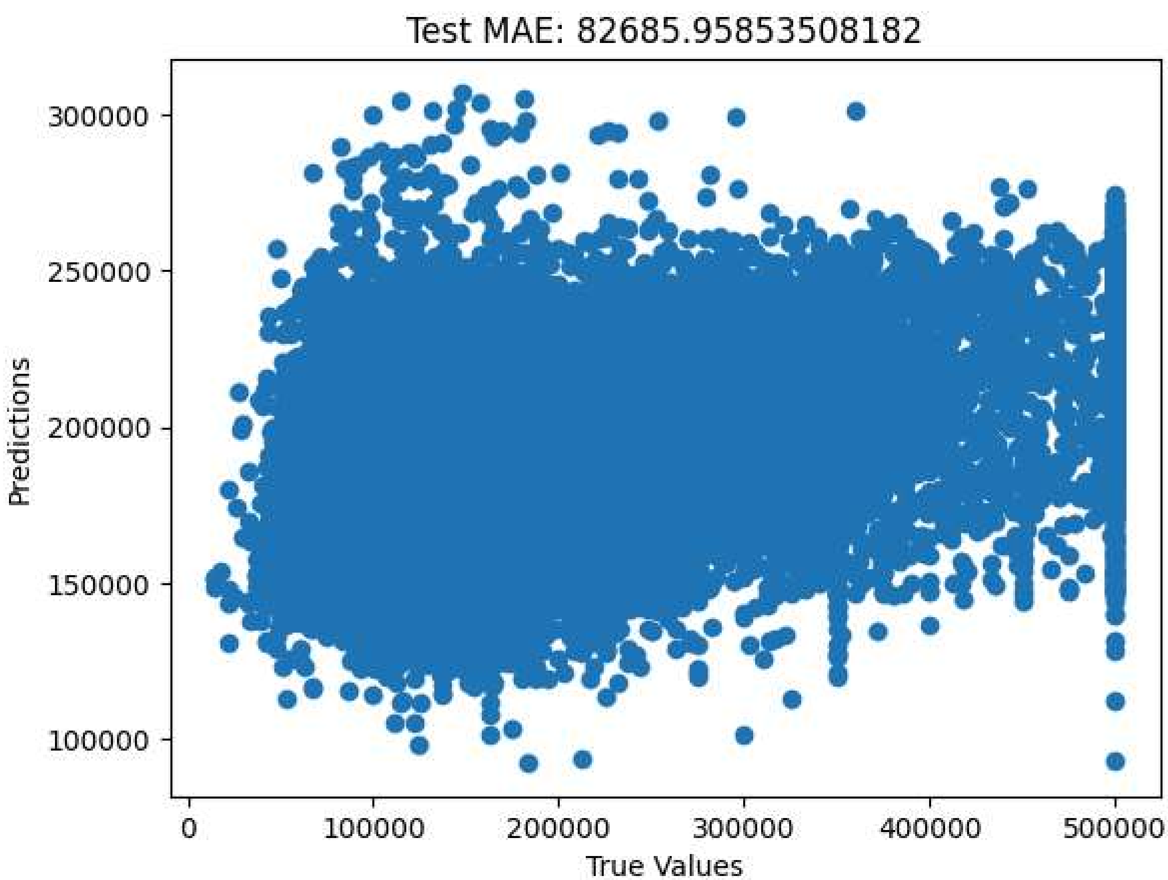
```
C: 1000, Epsilon: 50000, MAE: 85556.35170085153
```



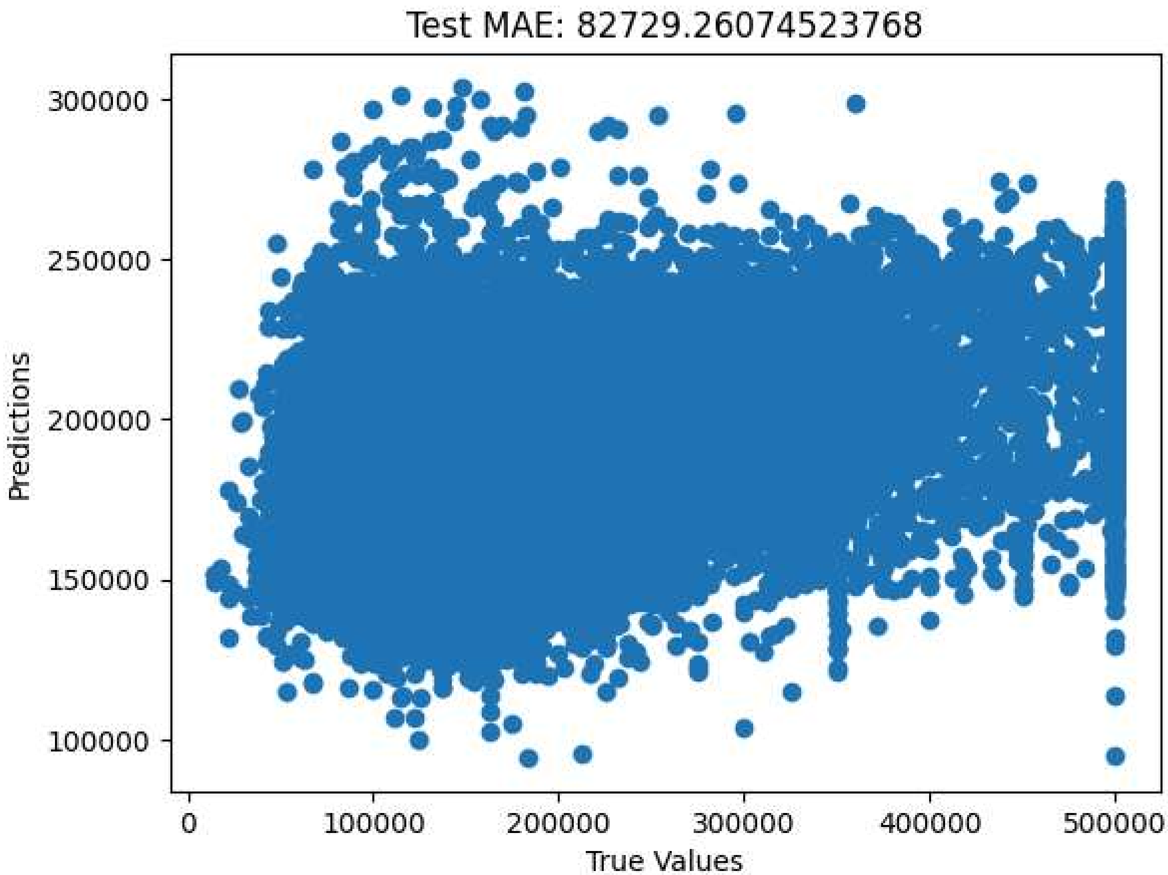

```
C: 1000, Epsilon: 100000, MAE: 87294.37631570993
```



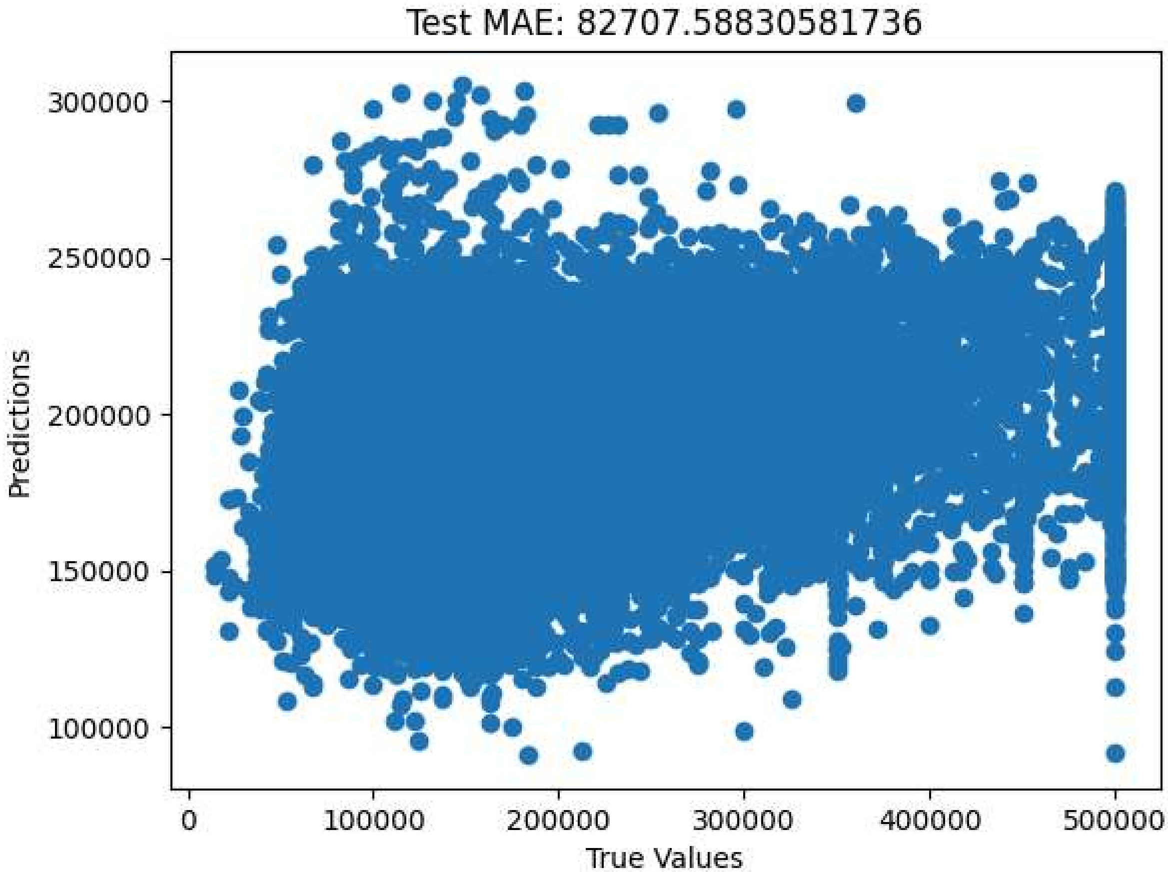

```
C: 10000, Epsilon: 2000, MAE: 82730.95222130371
```



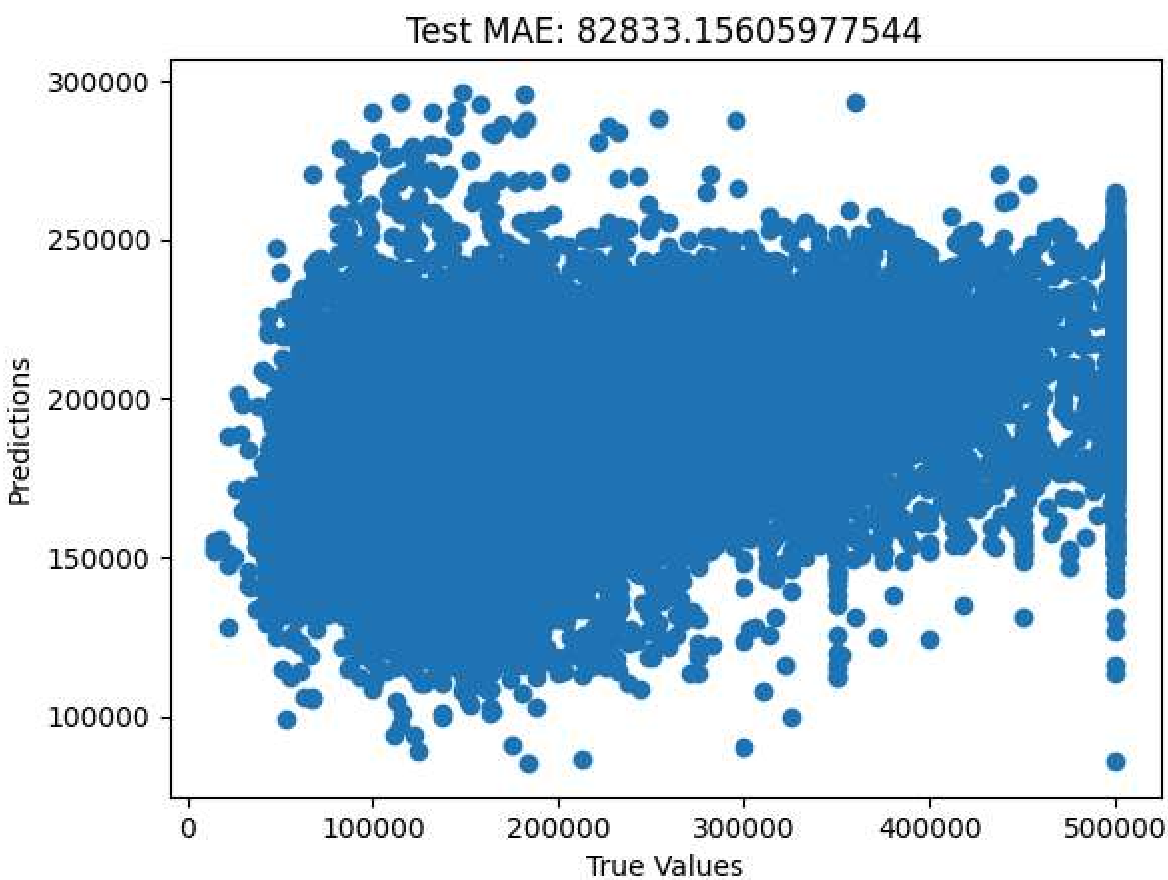
C: 10000, Epsilon: 5000, MAE: 82685.95853508182



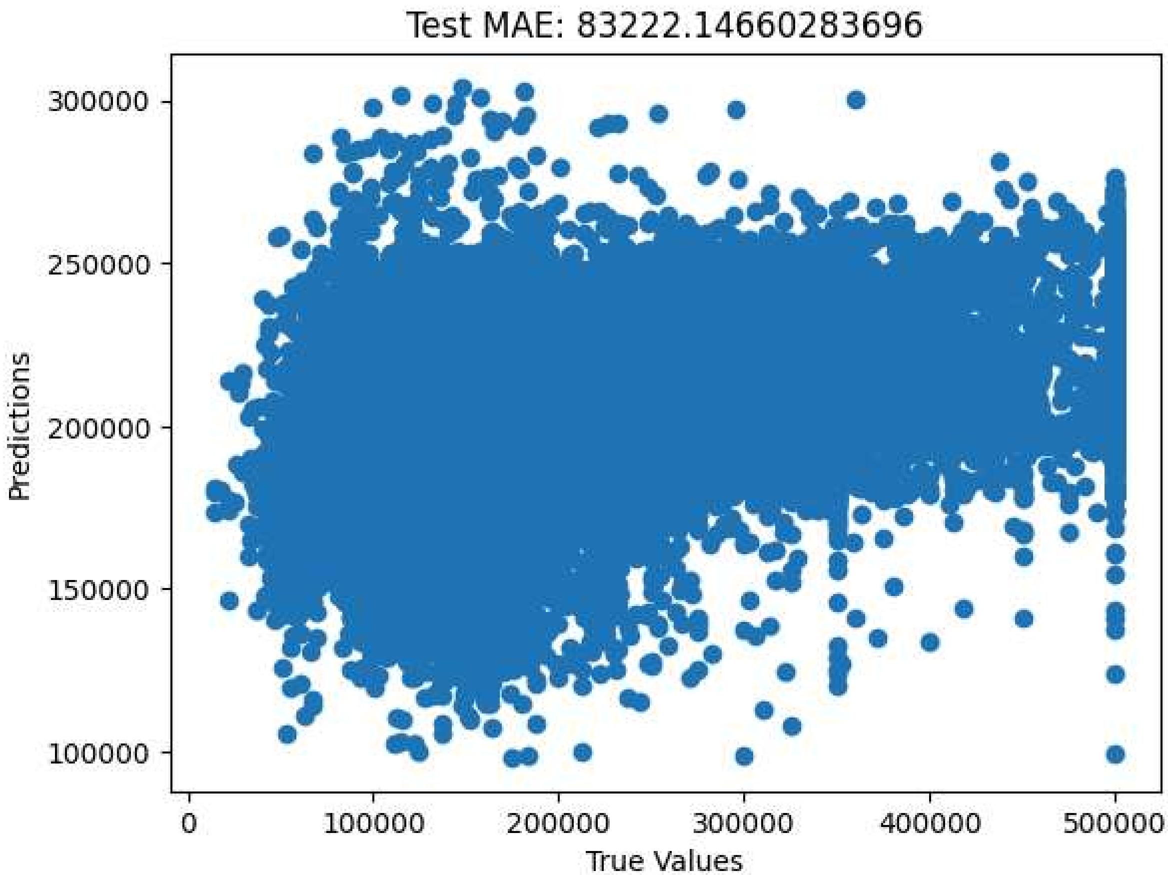
C: 10000, Epsilon: 10000, MAE: 82729.26074523768



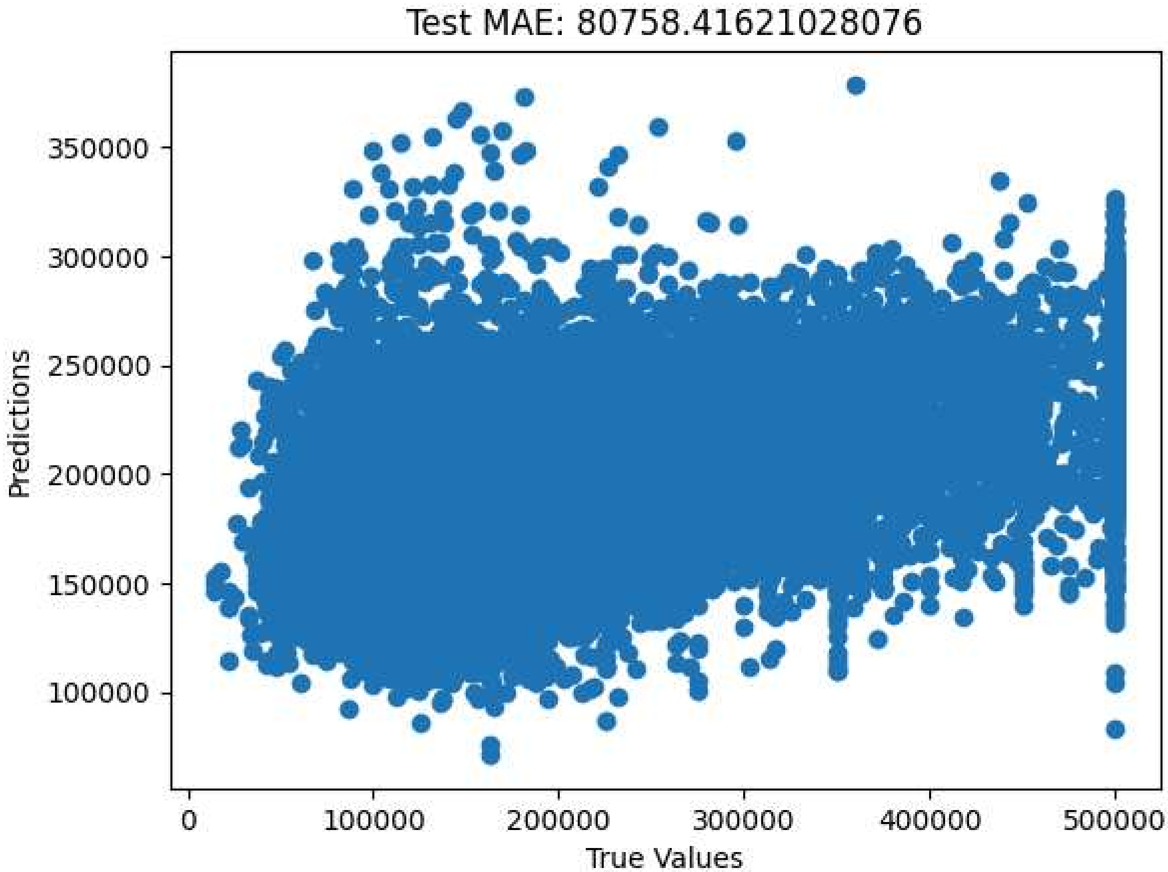

```
C: 10000, Epsilon: 20000, MAE: 82707.58830581736
```



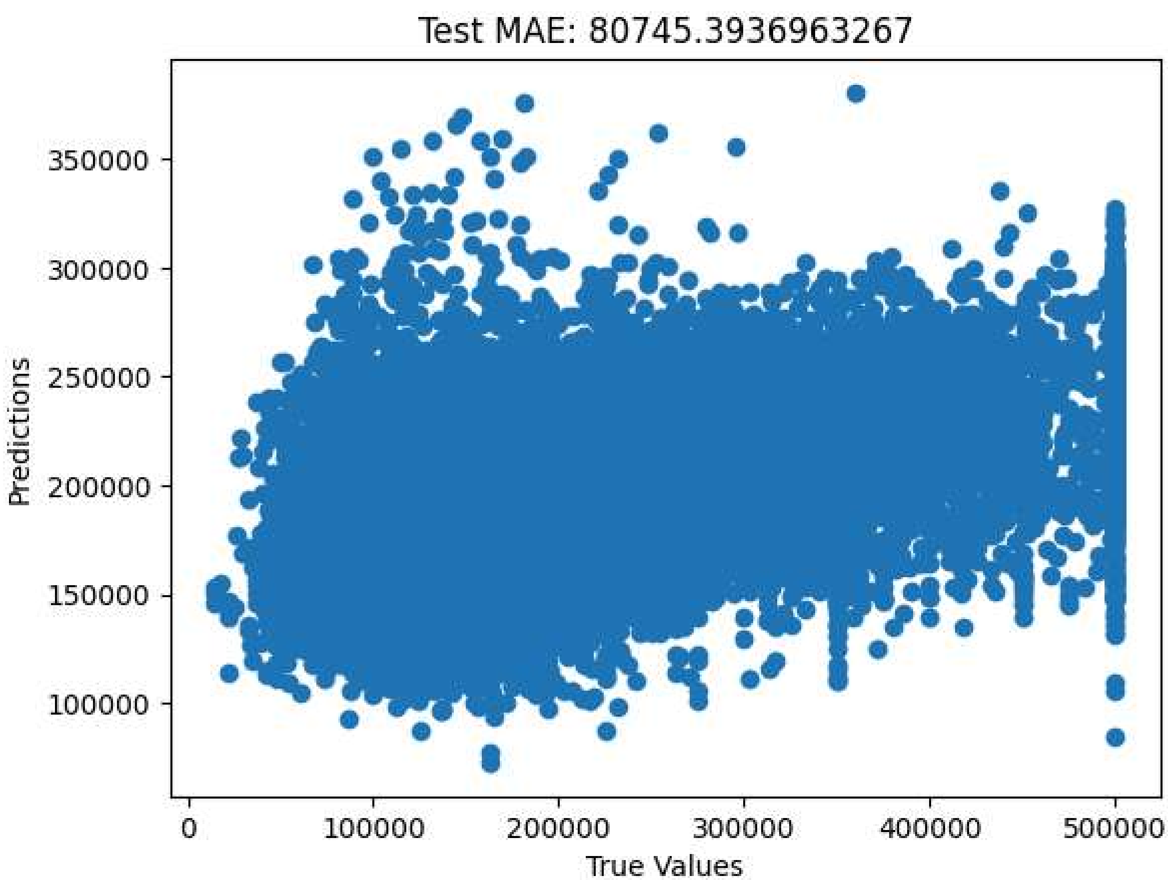
```
C: 10000, Epsilon: 50000, MAE: 82833.15605977544
```



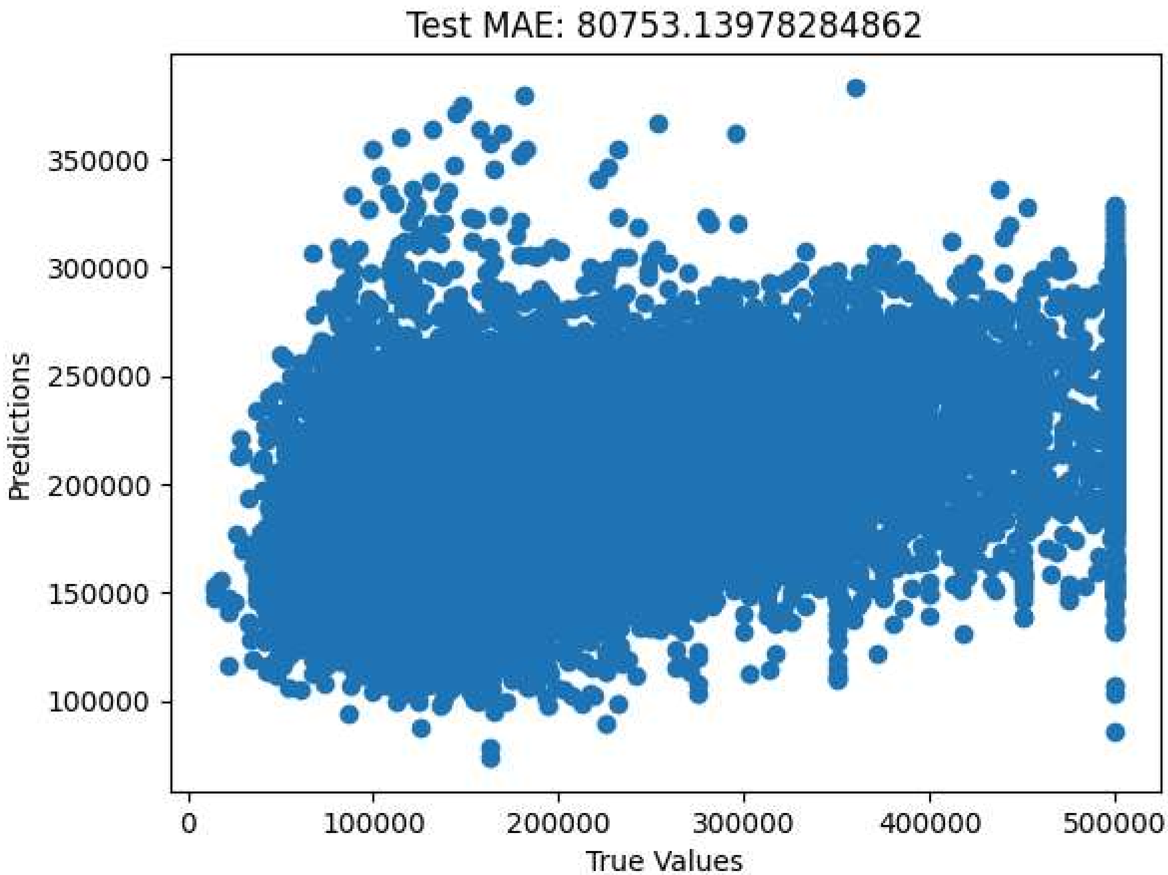

```
C: 10000, Epsilon: 100000, MAE: 83222.14660283696
```



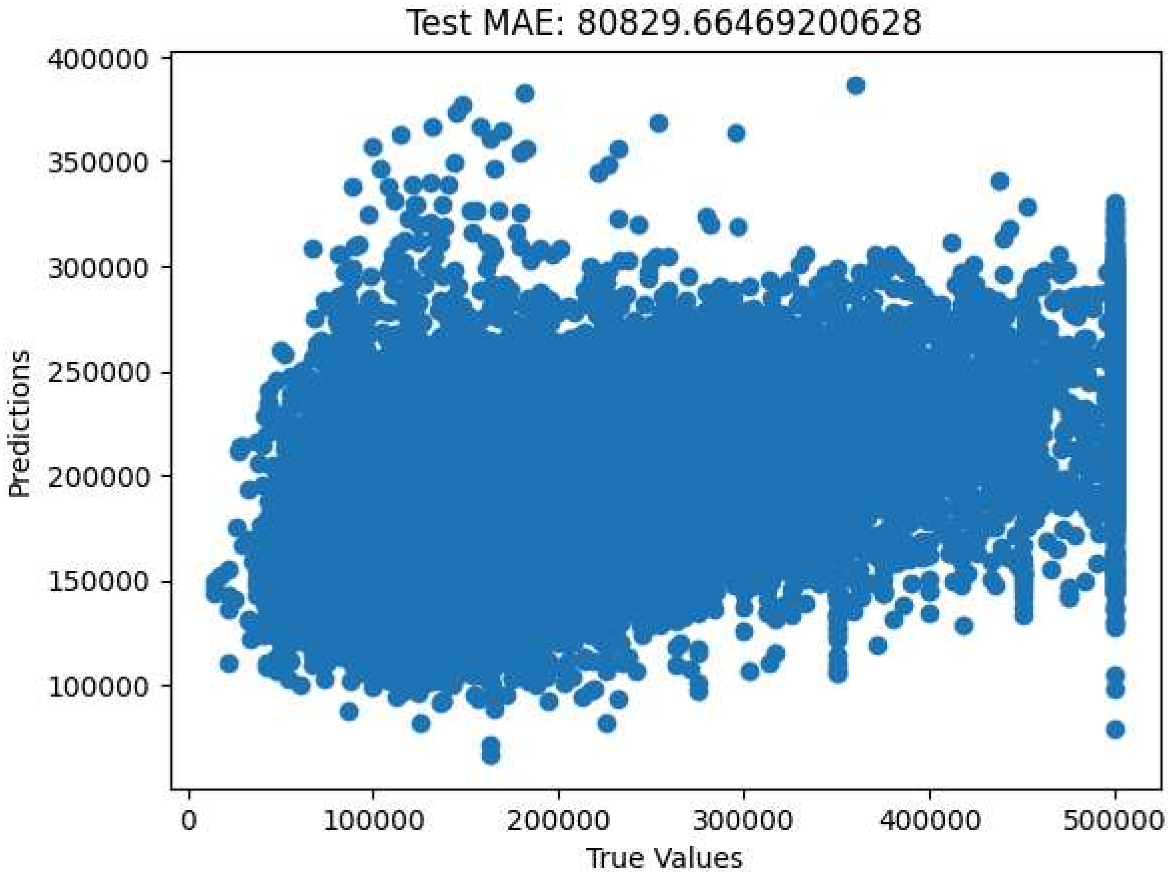

```
C: 100000, Epsilon: 2000, MAE: 80758.41621028076
```



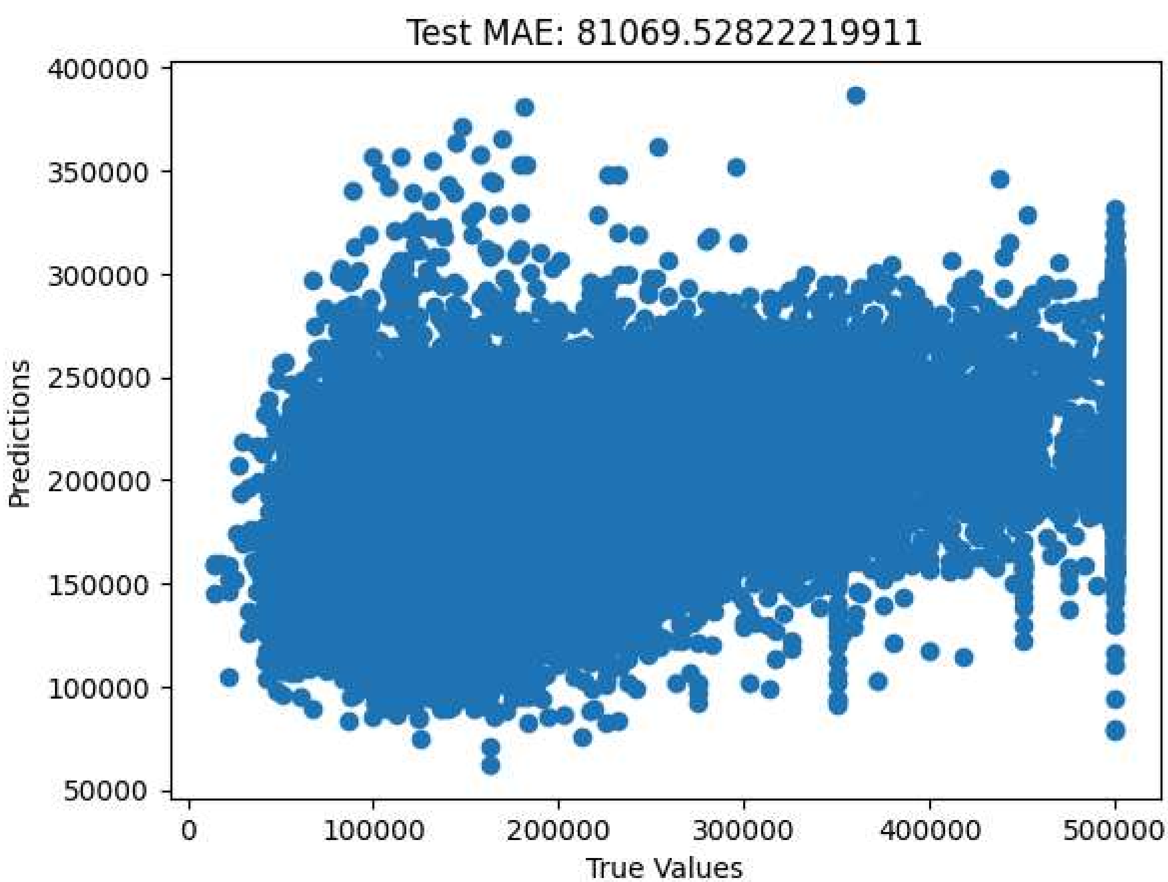
```
C: 100000, Epsilon: 5000, MAE: 80745.3936963267
```



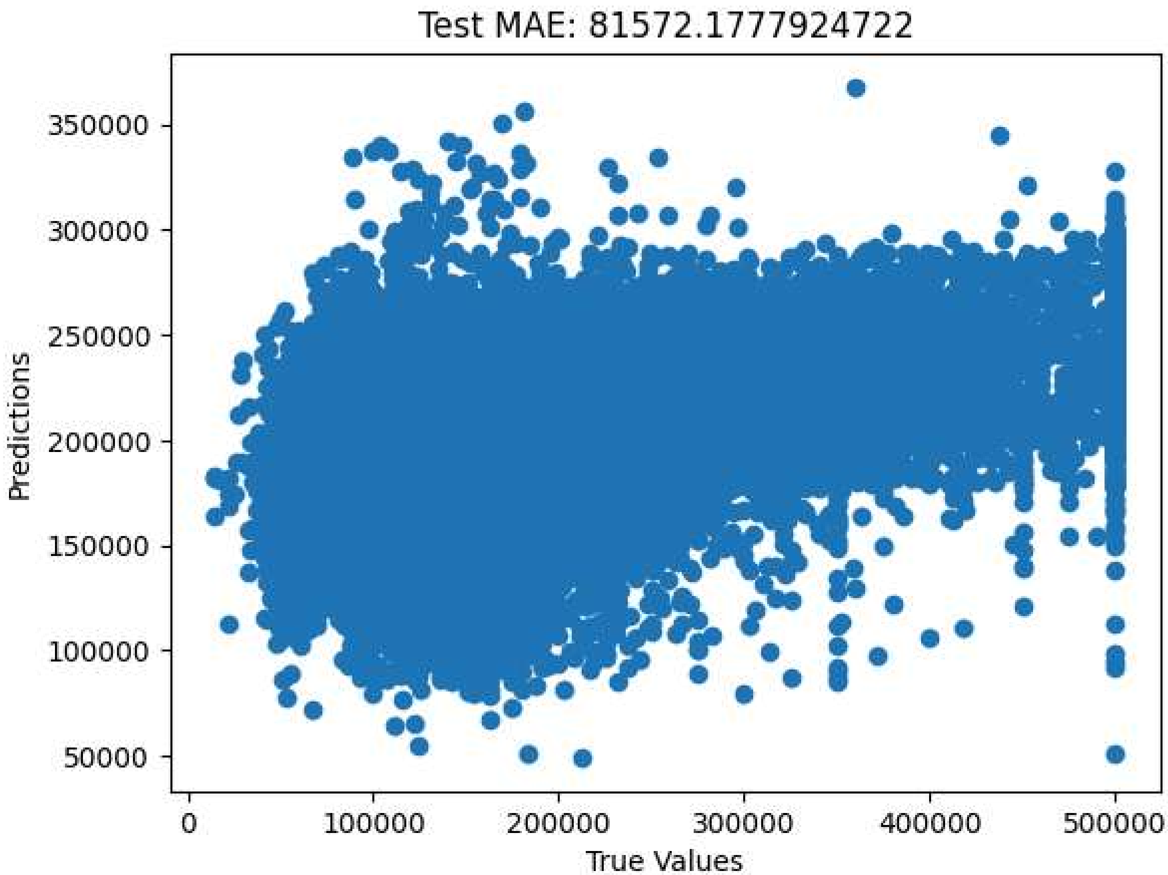

```
C: 100000, Epsilon: 10000, MAE: 80753.13978284862
```



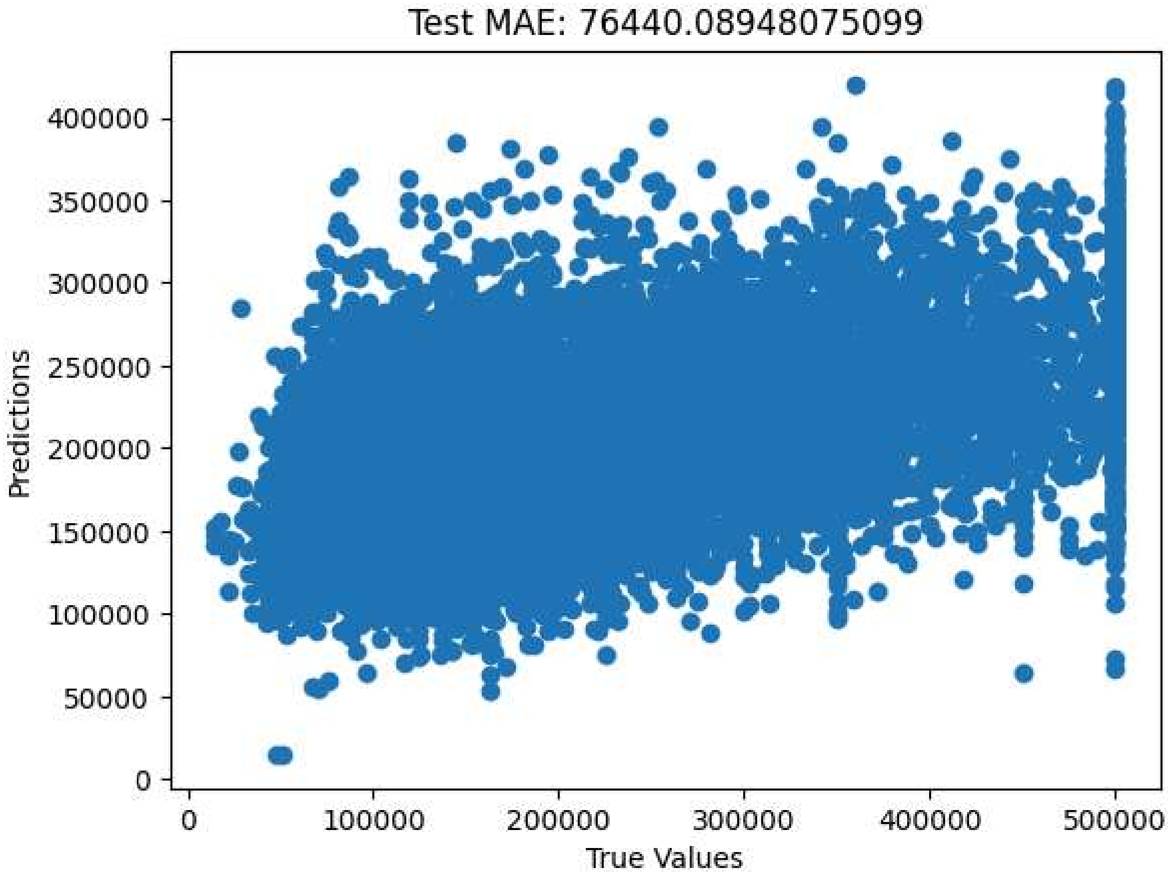

```
C: 100000, Epsilon: 20000, MAE: 80829.66469200628
```



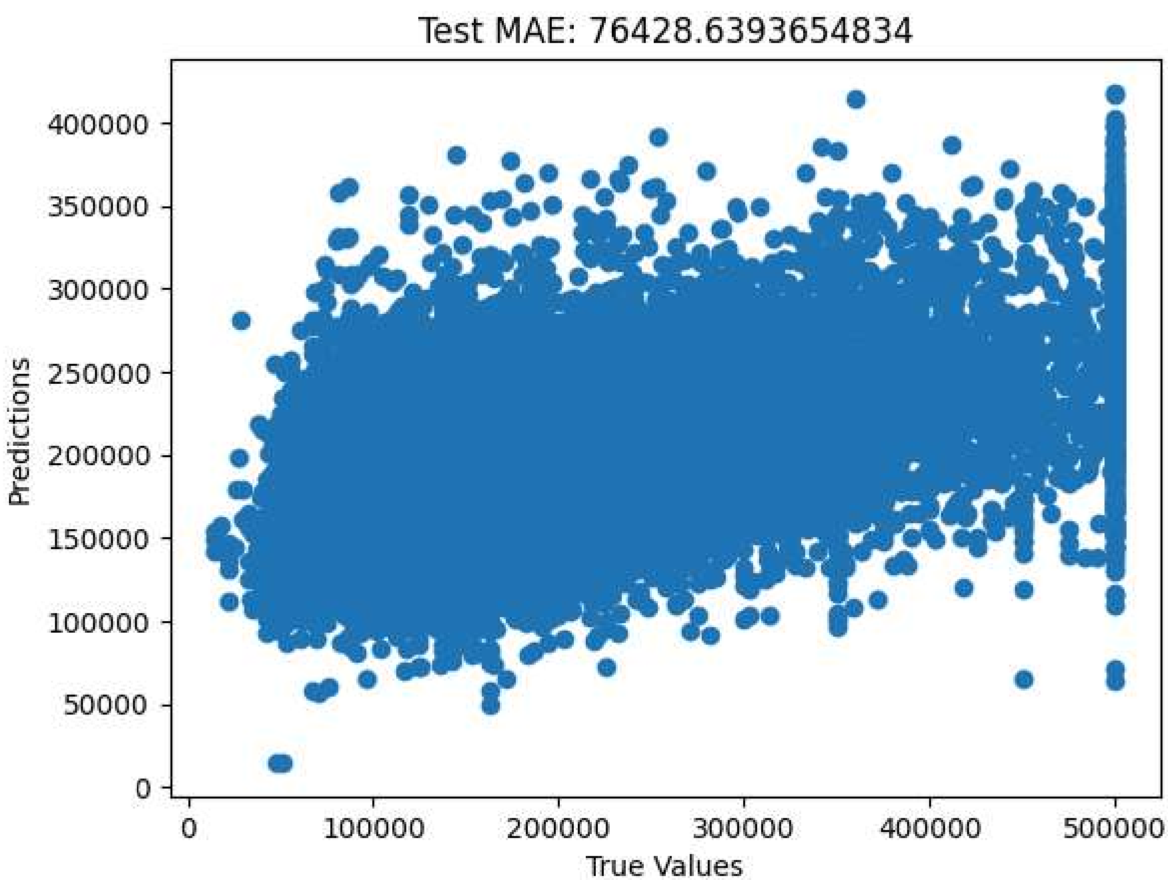
```
C: 100000, Epsilon: 50000, MAE: 81069.52822219911
```



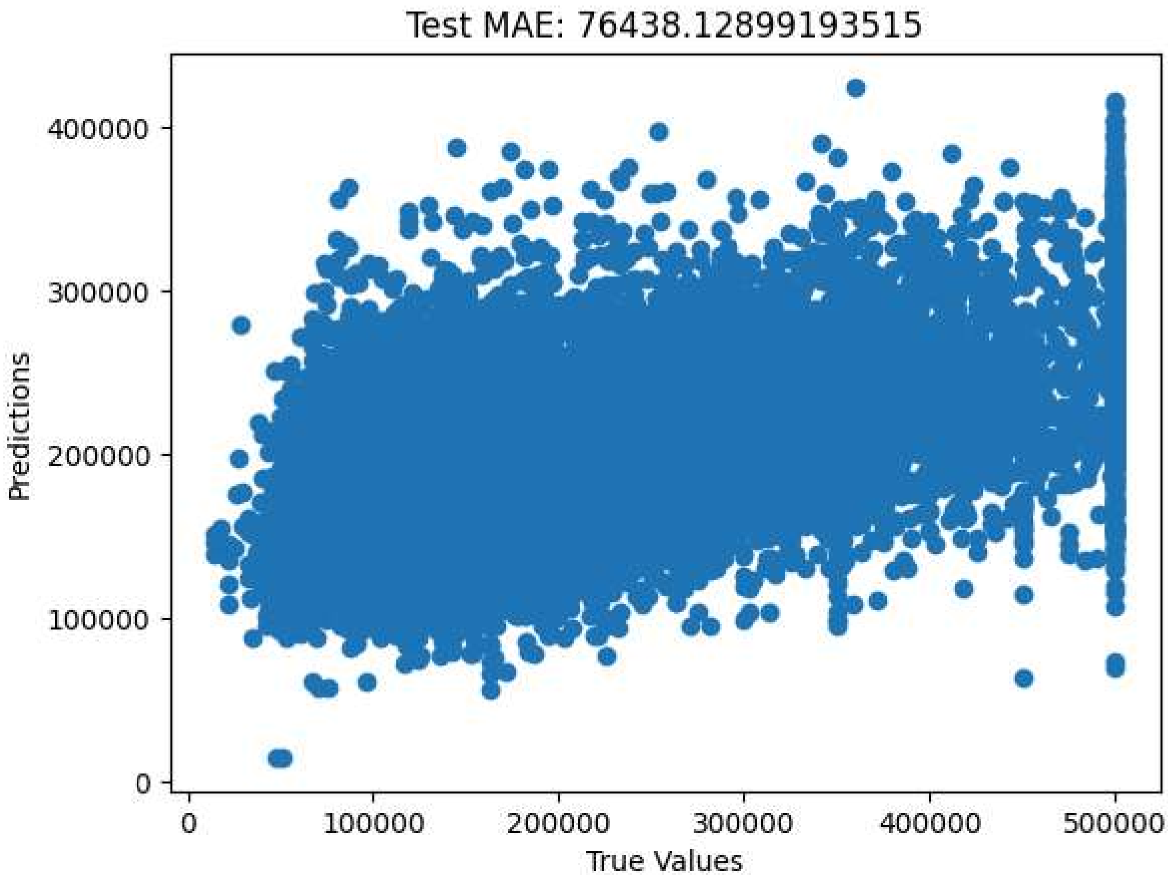

```
C: 100000, Epsilon: 100000, MAE: 81572.1777924722
```



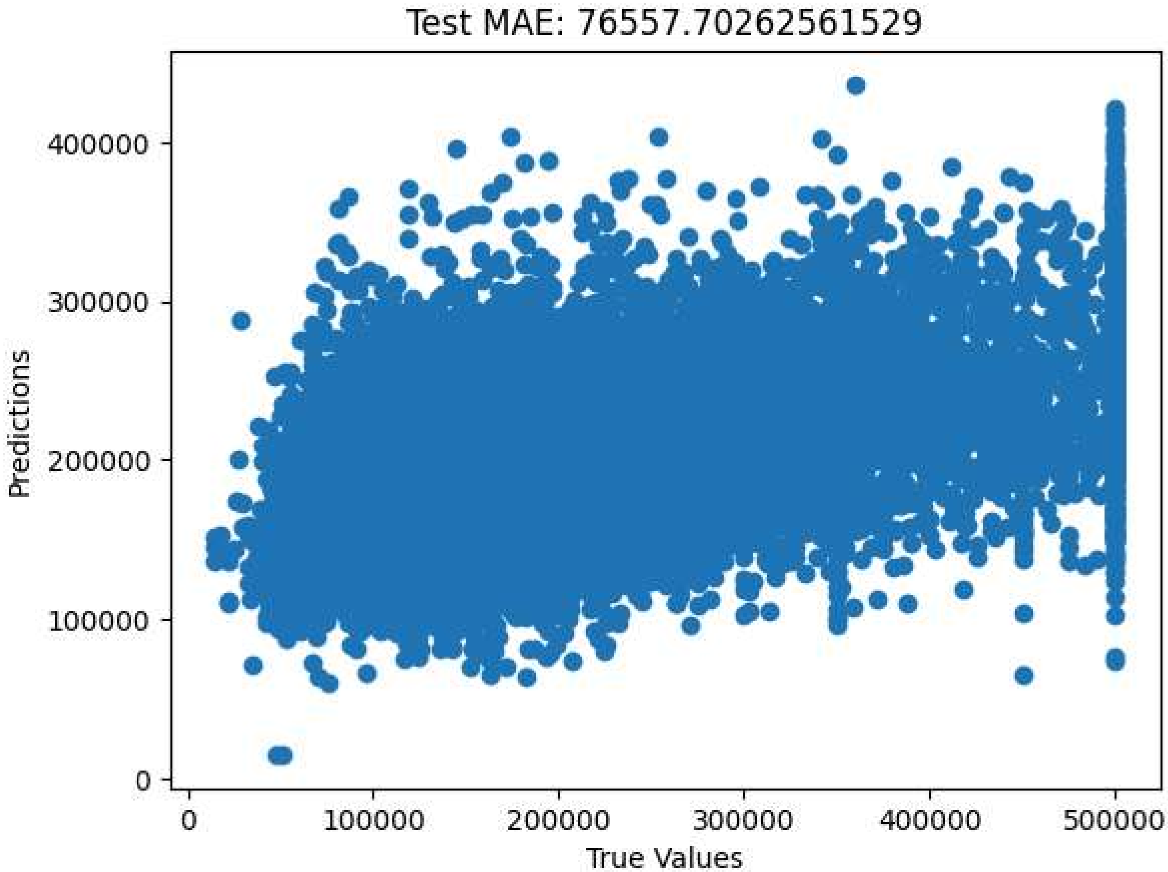

```
C: 1000000, Epsilon: 2000, MAE: 76440.08948075099
```



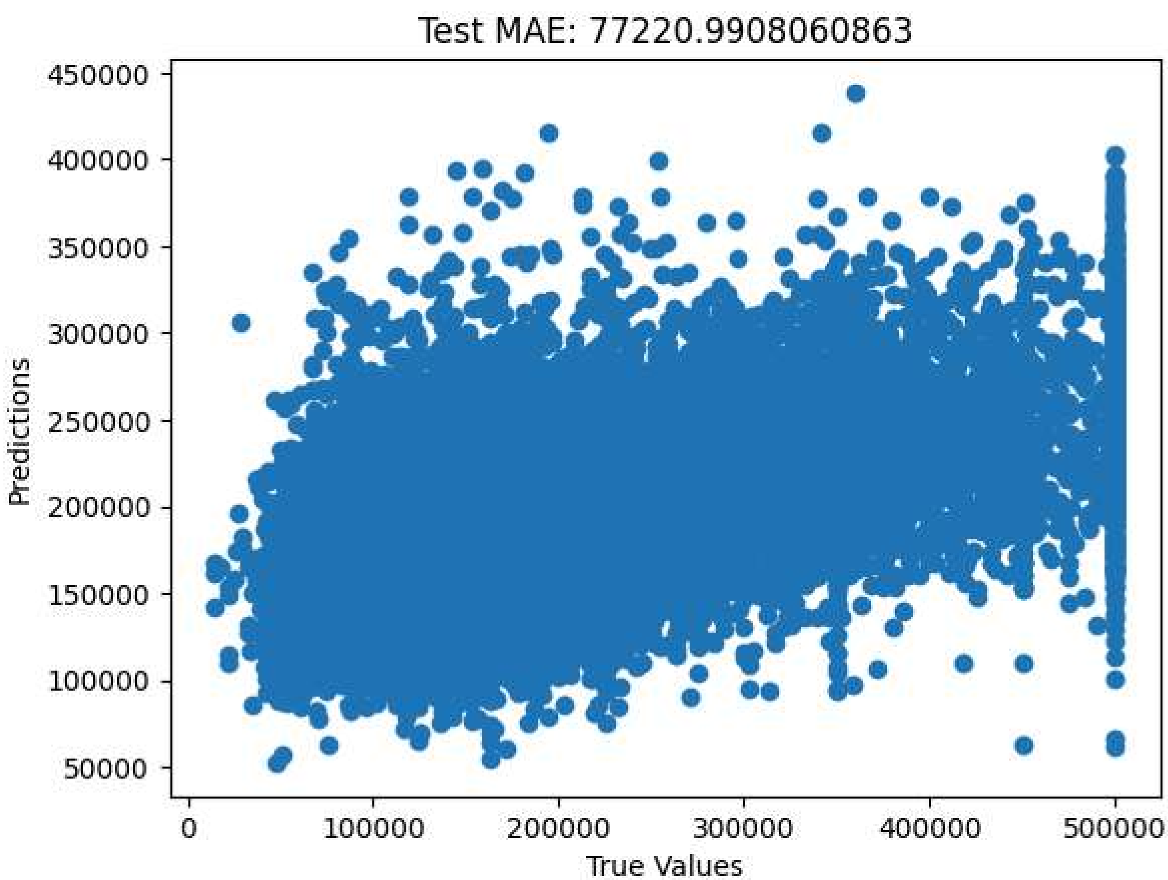
```
C: 1000000, Epsilon: 5000, MAE: 76428.6393654834
```



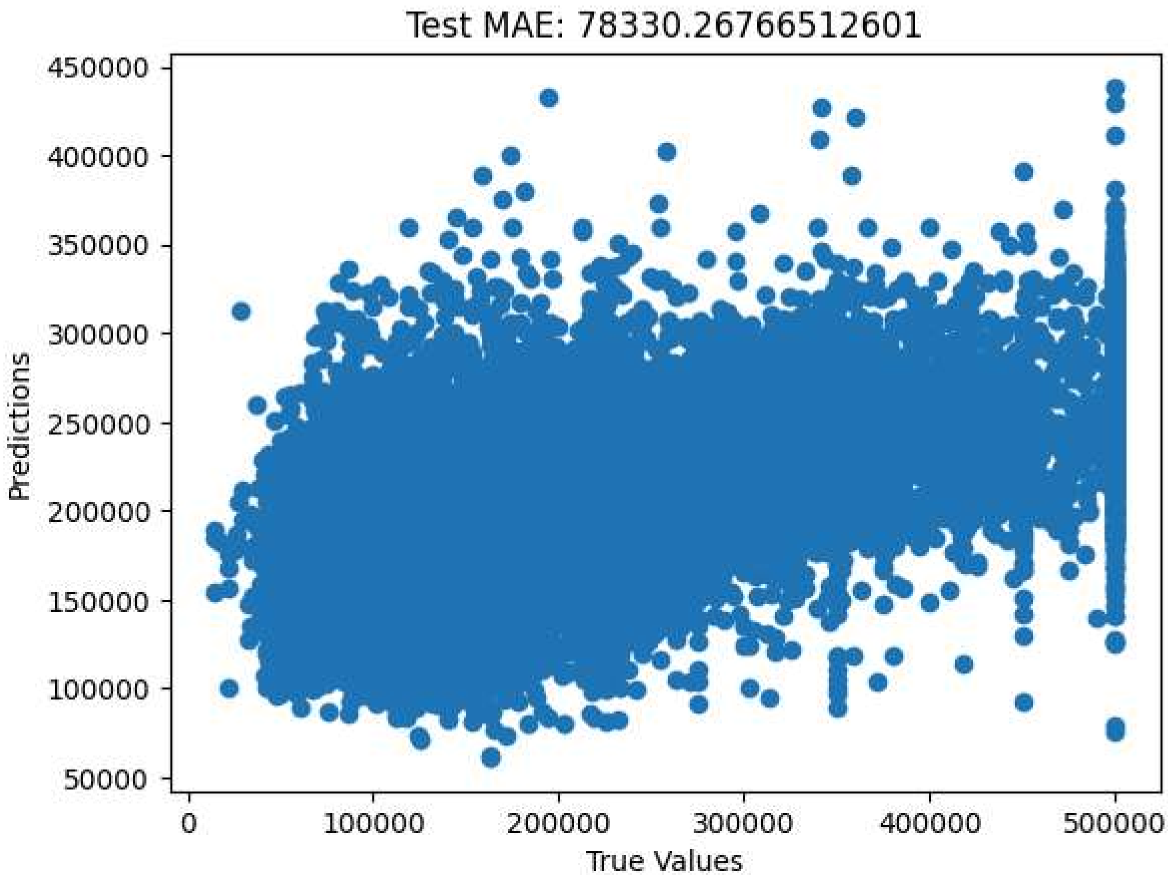

```
C: 1000000, Epsilon: 10000, MAE: 76438.12899193515
```



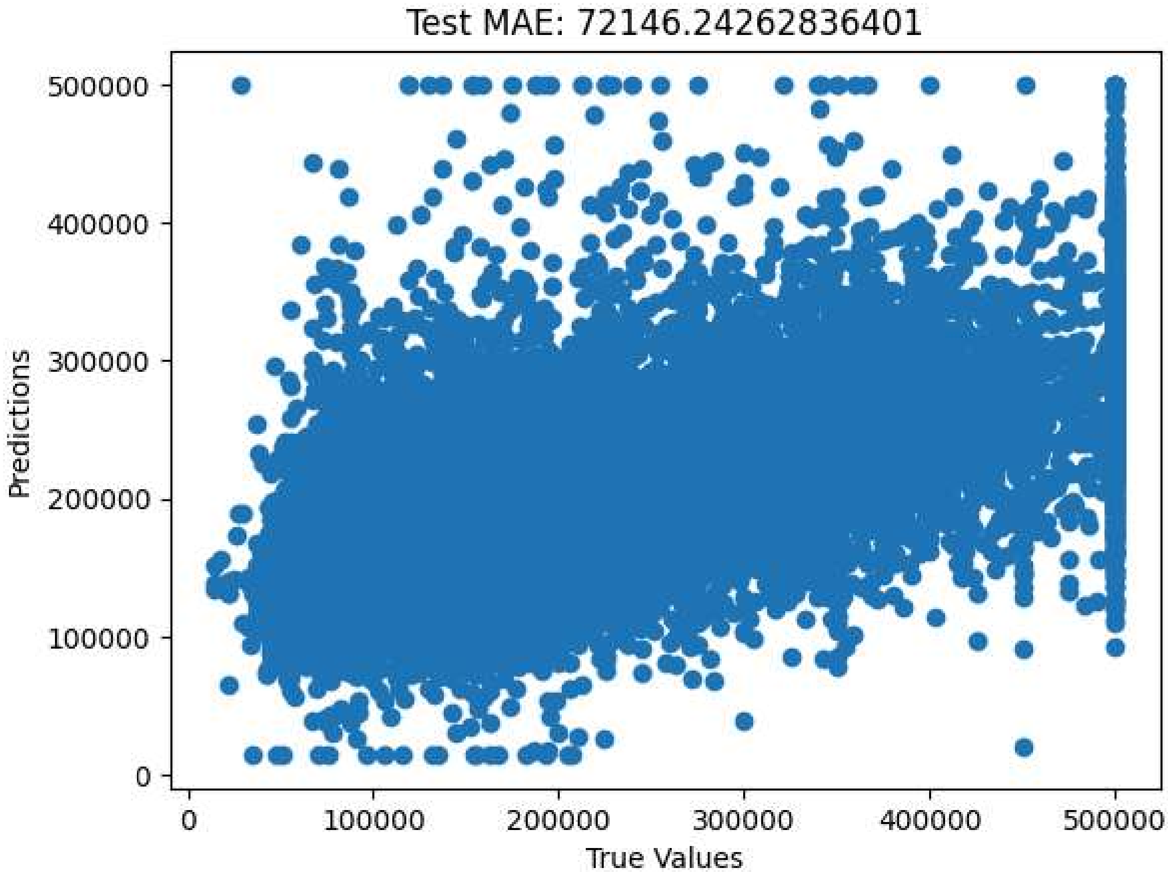

```
C: 1000000, Epsilon: 20000, MAE: 76557.70262561529
```



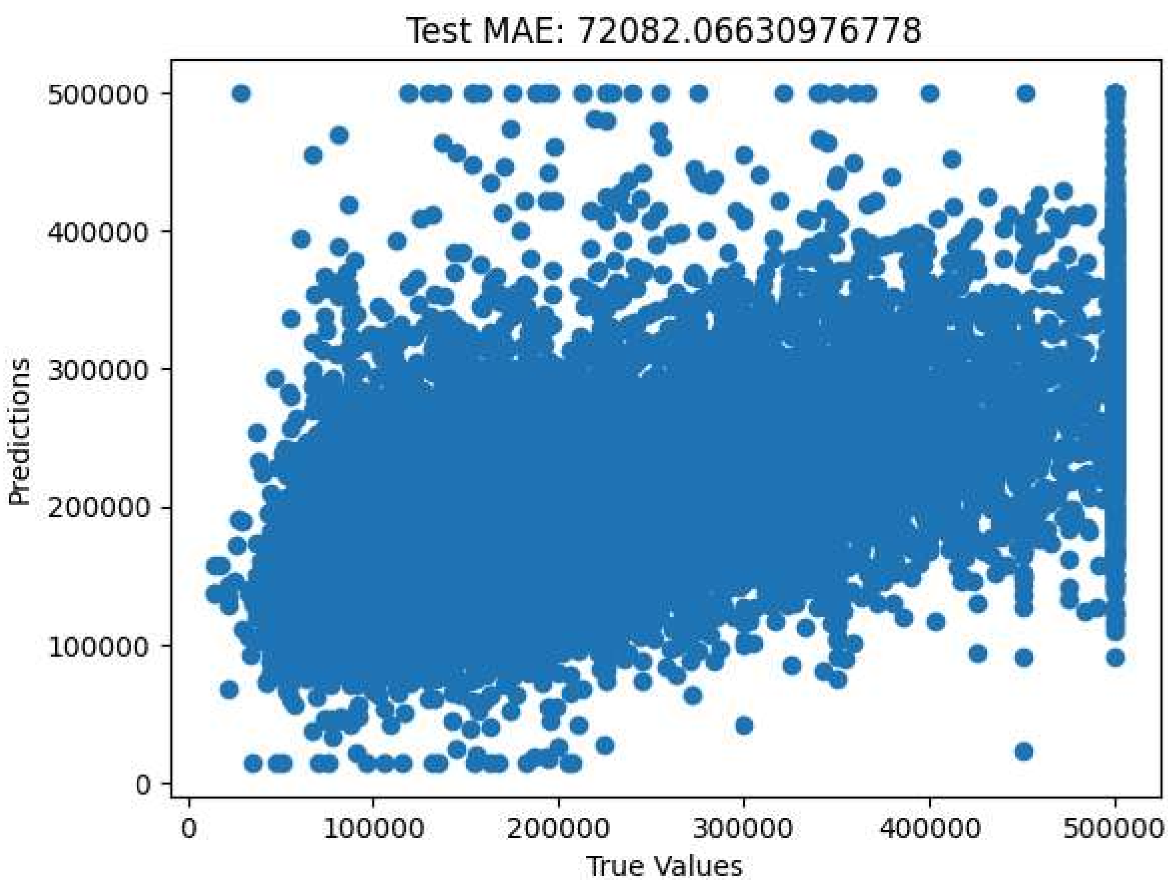
```
C: 1000000, Epsilon: 50000, MAE: 77220.9908060863
```



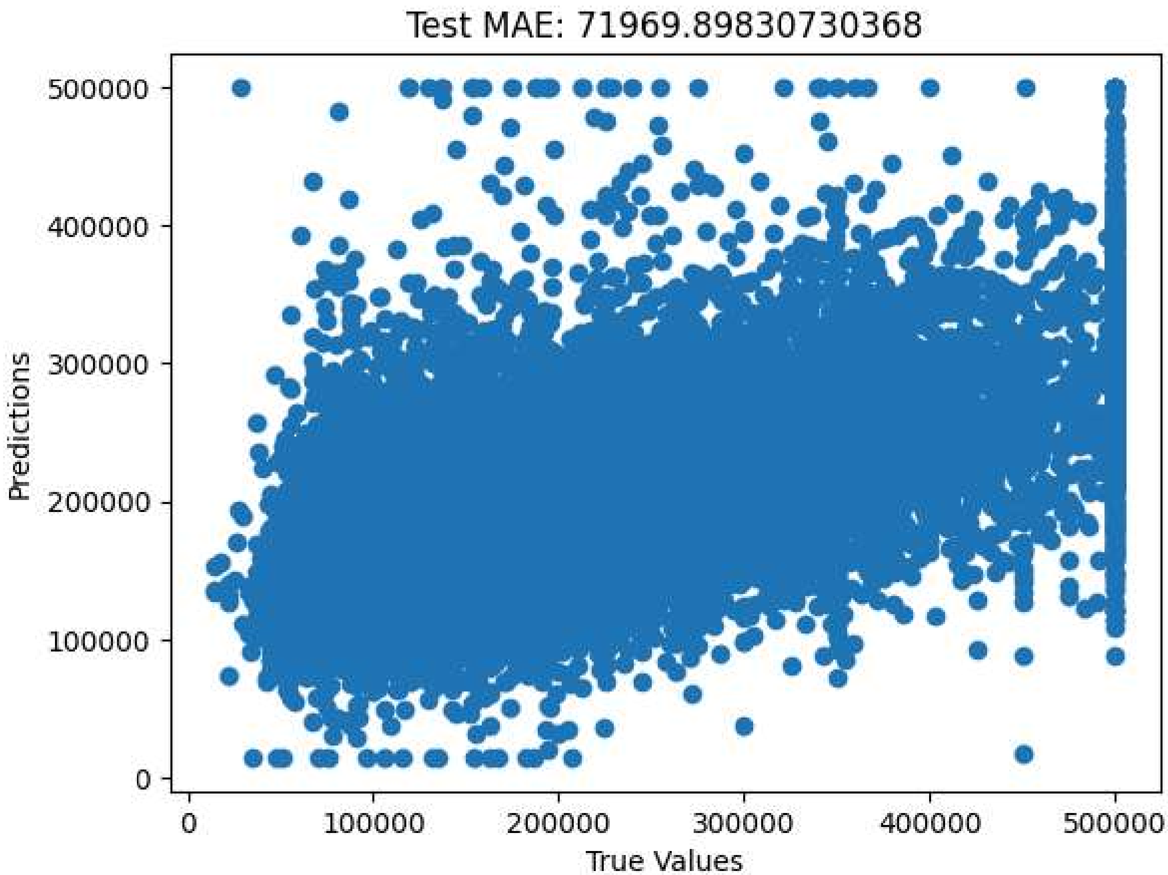

```
C: 1000000, Epsilon: 100000, MAE: 78330.26766512601
```



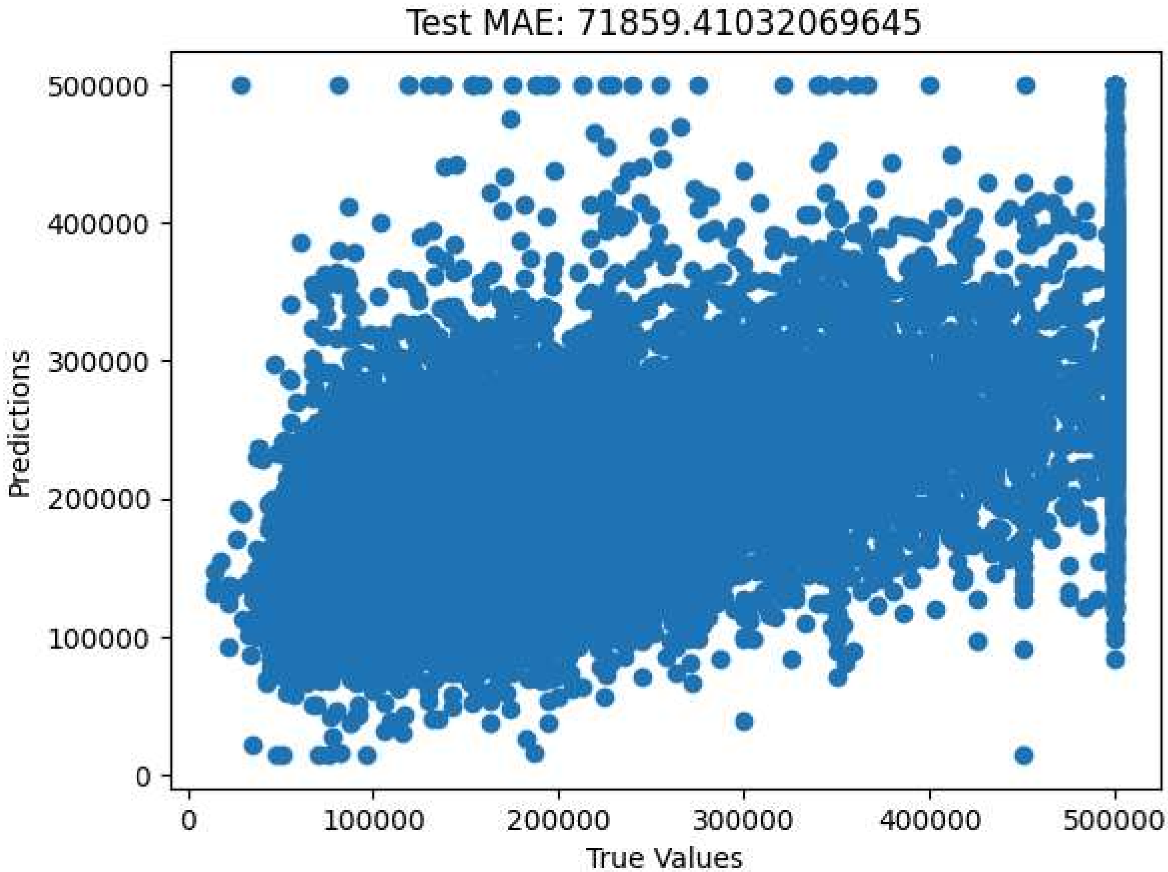

```
C: 10000000, Epsilon: 2000, MAE: 72146.24262836401
```



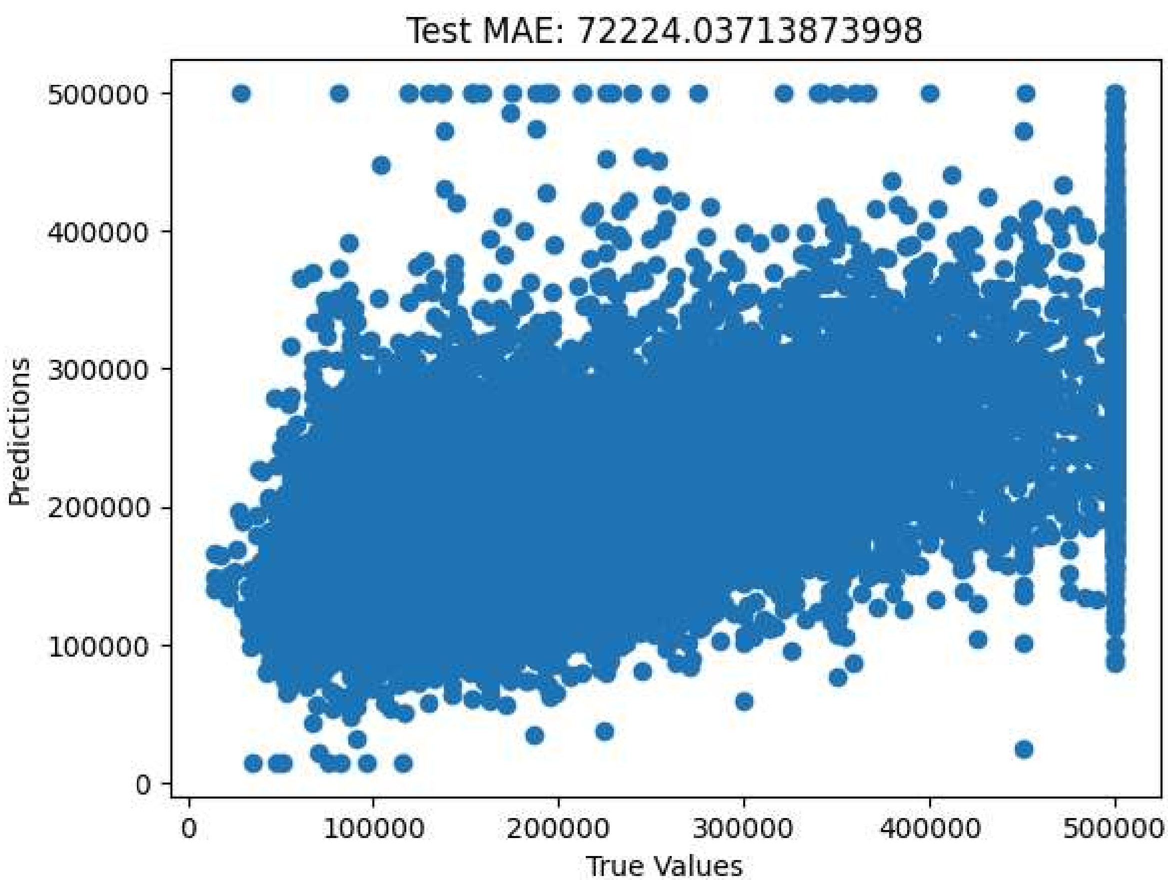
```
C: 10000000, Epsilon: 5000, MAE: 72082.06630976778
```



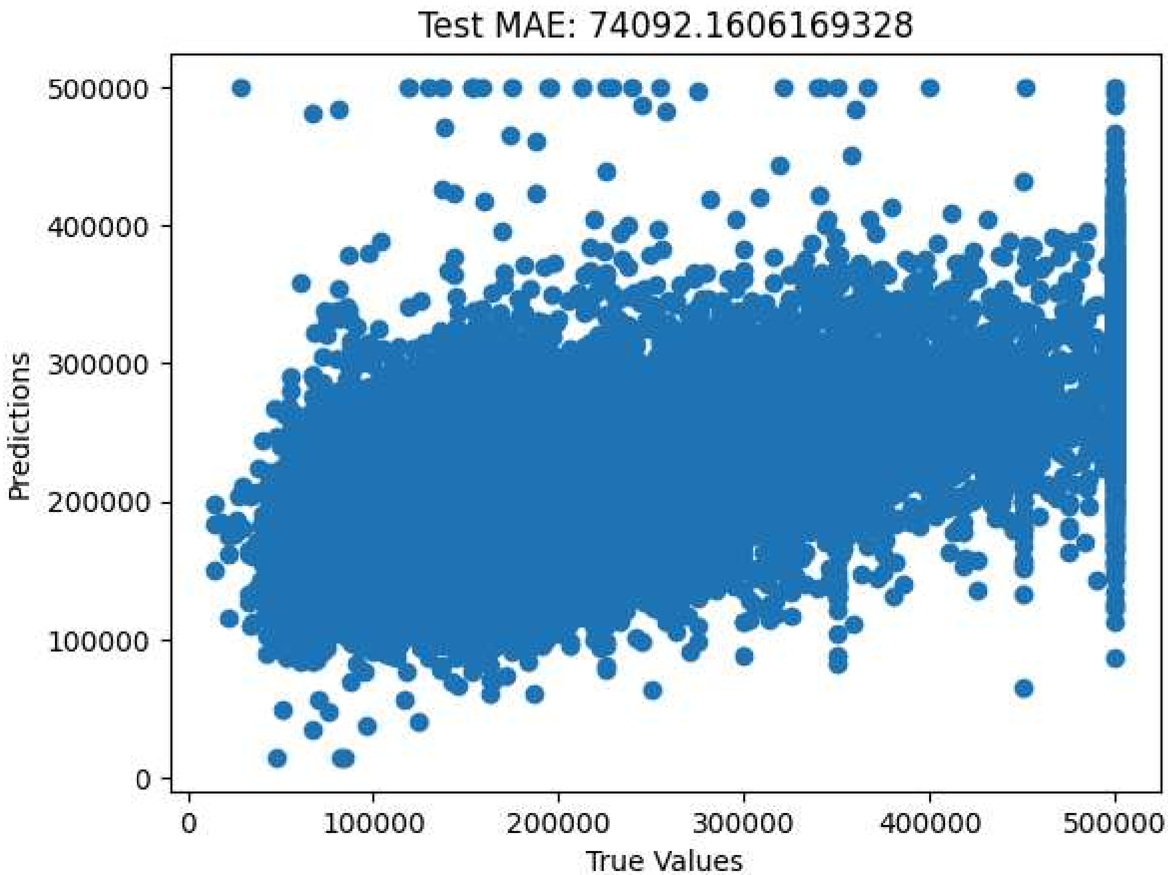

```
C: 10000000, Epsilon: 10000, MAE: 71969.89830730368
```




```
C: 10000000, Epsilon: 20000, MAE: 71859.41032069645
```



```
C: 10000000, Epsilon: 50000, MAE: 72224.03713873998
```

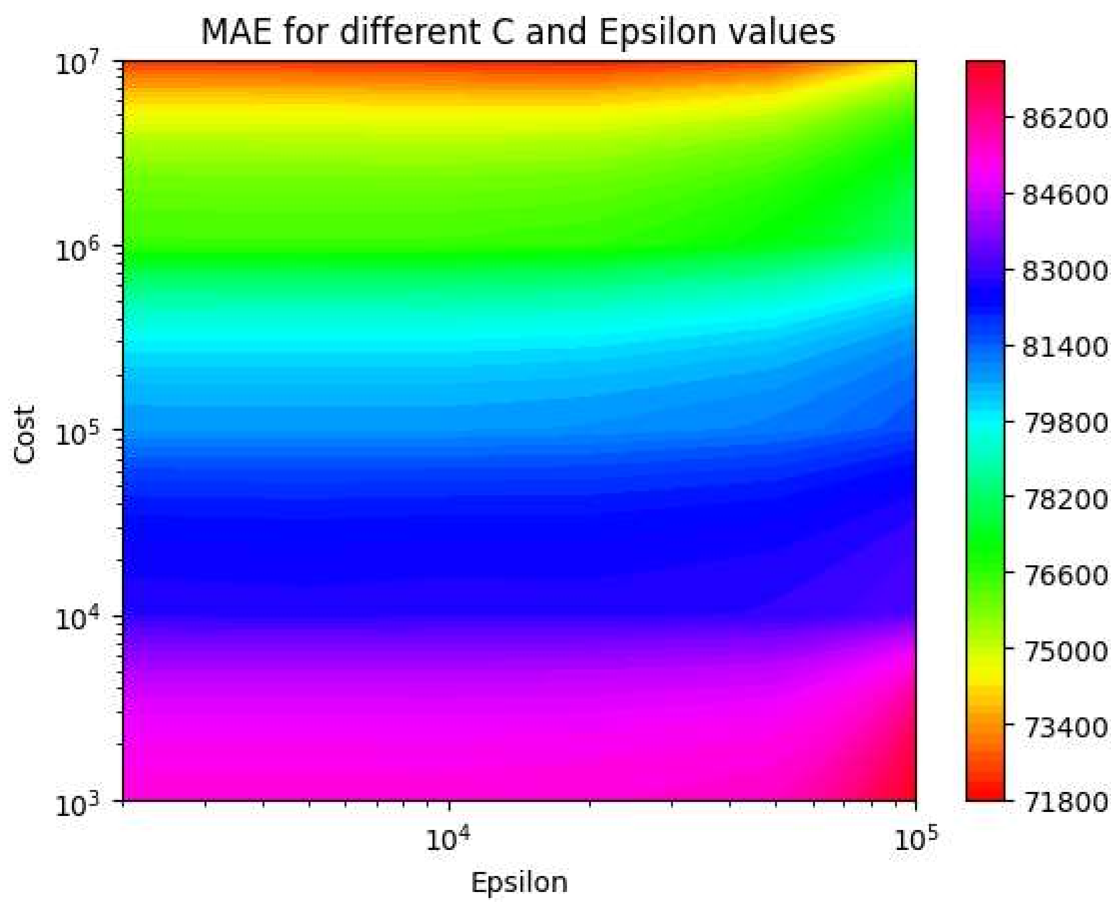



```
C: 10000000, Epsilon: 100000, MAE: 74092.1606169328
```

Part B Discussions

Task 4: visualize the MAE values for the range of Cost and Epsilon using

In [55]:
```
import matplotlib.pyplot as plt

plt.contourf([2000,5000,10000,20000,50000,100000],[1000,10000,100000,1000000,1
0000000],maes,100,cmap='hsv')
# Set the Y-axis (Cost) to logarithmic scale to better visualize the wide rang
e of values
plt.yscale('log')
# Set the X-axis (Epsilon) to logarithmic scale to better visualize the wide r
ange of values
plt.xscale('log')
# Add a color bar to the plot to indicate the range of MAE values
plt.colorbar()
# Label the X-axis as 'Epsilon' for clarity
plt.xlabel('Epsilon')
# Label the Y-axis as 'Cost' for clarity
plt.ylabel('Cost')
# Set the title of the plot to describe what is being visualized
plt.title('MAE for different C and Epsilon values')
# Display the contour plot
plt.show()
```



Task 5: Discuss the following question

In [56]:
```
min_mae_index = np.unravel_index(np.argmin(maes, axis=None), maes.shape)
best_C = param_grid['C'][min_mae_index[0]]
best_epsilon = param_grid['epsilon'][min_mae_index[1]]
min_mae = maes[min_mae_index]

print(f"Best parameters: C={best_C}, Epsilon={best_epsilon}, with MAE={min_mae}")
```

```
Best parameters: C=10000000, Epsilon=20000, with MAE=71859.41032069645
```

For what set of parameters is the MAE loss the lowest?

Best parameters: C=10000000, Epsilon=20000, with MAE=71859.41032069645

Is this MAE loss better than the MAE loss you obtained in Lab 5?

No, the MAE from Lab 5 (49,861.89) is better than the MAE from Lab 7 (71,859.41). This suggests that the SGDRegressor performed better than the SVR for this particular task and dataset.

Note any interesting observations you have on the effects the hyperparameters (cost, and epsilon) has on the test MAE

In Lab 7, the hyperparameters cost and epsilon had significant effects on the test MAE. A large C value made the model more flexible by penalizing errors heavily which can reduce bias but also increase the risk of overfitting. This likely contributed to the relatively high test MAE as the model may have fit the training data too aggressively without generalizing well. The large epsilon value created a wider margin of tolerance for errors, which could lead to underfitting by allowing larger deviations between predictions and true values. The combination of a very high cost and large epsilon resulted in a suboptimal balance causing the model to perform worse on the test set compared to Lab 5 where the simpler SGDRegressor yielded a lower MAE. This demonstrates the importance of carefully tuning both cost and epsilon to find an ideal balance between bias, variance, and model flexibility.