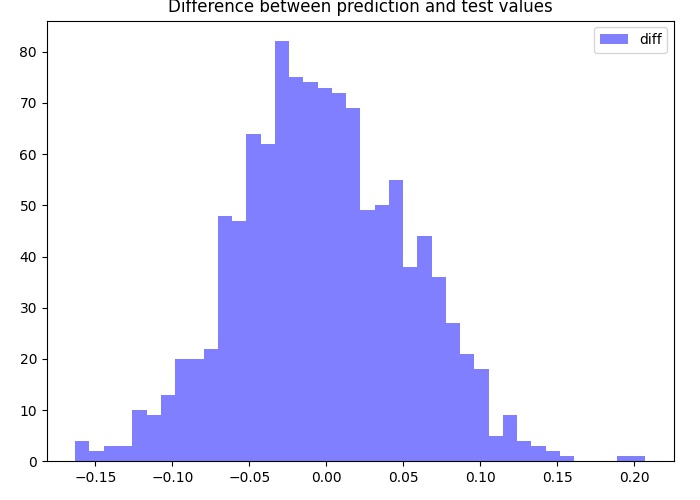
**Linear Model**

We use the residual histogram to visually demonstrate the fitting effect.

1. f, ax = plt.subplots(figsize=(7, 5))
2. f.tight\_layout()
3. ax.hist((energy\_test - energy\_pred)/energy\_test,
4. bins=40, label='diff', color='b', alpha=.5)
5. ax.set\_title("Difference between prediction and test values")
6. ax.legend(loc='best')
7. plt.savefig('./image\_diff/diff\_'+str(index)+'.jpg')
8. **print**("MSE:", MSE(energy\_test, energy\_pred))



It is not difficult to figure out that most of the data have a difference within 10%. Sounds like a nice prediction.

# MSE (mean squared error)

The skewed data in this graph is a little bit clearer than it was before.

Other measures can also be observed and fitted. The most commonly used measures are mean Squared error (MSE) and mean Absolute deviation (MAD). Let's implement these two metrics in Python. Next we use the built-in metrics of Scikit-Learn to assess the effectiveness of the regression model:

1. **def** MSE(target, predictions):
2. squared\_deviation = np.power(target - predictions, 2)
3. **return** np.mean(squared\_deviation)

The calculation formula of MSE is:

Calculate the difference between the predicted value and the actual value, square it and then average it. That's what we're looking for when we're looking for the optimal correlation coefficient. Gauss-markov theorem has in fact shown that the optimal linear BLUE estimate of the regression coefficient of linear regression is an unbiased estimate of the least mean square error (the error variable is unrelated, 0 means, covariance). In the topic of making up the deficiency of linear regression with ridge regression, we will see what happens when our correlation coefficient is biased.

# MAD (mean absolute deviation)

You can look at each measure and determine which measure is more important. For example, with MSE, large errors are penalized more because the square amplifies it.

1. **def** MAD(target, predictions):
2. absolute\_deviation = np.abs(target - predictions)
3. **return** np.mean(absolute\_deviation)

# Boostrapping

# The other thing to note is that the correlation coefficients are random variables, so they're distributed. Let's bootstrapping and look at the distribution of the correlation coefficients. Bootstrapping is a common means to learn the uncertainty of parameter estimation:

# # TBD