Predicting house prices with multivariate linear regression with a regularization parameter and random forest regressor

CS-C3240 - Machine Learning

January 30, 2025

1 Introduction

Today's housing prices are highly fluctuating – buying a house on the right time and in the right place is crucial for future prosperity. This report performs a rigorous data analysis and feature selection on a housing price dataset, and will train a model to predict housing prices in Boston's suburbs. The application domain is real estate, since the final model will try to predict housing prices depending on the house properties related to its location and physical condition.

In the Section 2, the problem is formulated by defining the dataset and by giving a general overview of the problem and methods to solve it. Section 3 analyzes and selects the features of the dataset. Furthermore, the section will present the hypothesis spaces, alongside with appropriate loss functions and model validation techniques. In the Section 4, the chosen models are trained and evaluated, and the final model to solve the problem is decided. Finally, Section 5 concludes the report and results.

2 Problem formulation

2.1 Problem

The Boston Housing dataset is a widely used dataset, which provides information about the housing landscape in Boston, offering information about many features in a Boston suburb, and additionally the median value of those suburbs' owner-occupied properties [8].

This project will evaluate and train both a regularized linear regression model and a random forest regressor, which can predict the median value of a house in the suburbs based on the given features of the suburb. As the used dataset contains the desired labels for each data point, we can utilize them in the training process and therefore the type of this machine learning task is supervised.

The task includes first picking the necessary features, performing feature engineering in terms of missing values and normalization, dividing the dataset into appropriate training-, test- and validation-sets, selecting a justified hypothesis space and the loss function. Eventually training and testing with the model and dataset will be performed and the results will be analyzed.

2.2 Dataset

The dataset consists of a multidimensional input, where individual data points represent suburbs in Boston. The features are mainly continuous: for example per capita crime rates by town (CRIM), nitrix oxide concentrations of air (NOX), average numbers of rooms per area (RM), proportions of houses built prior to 1940 (AGE) and distances to Boston employment centres (DIS). Moreover, some of the features are categorical: the CHAS-feature where a boolean value is telling whether area is located next to the Charles river has only two possible values, and also the index of accessibility to radial highways only has nine unique values (see Appendix). The label for each data point is the median value of owner-occupied homes in the area in 1000$'s (MEDV). The type of the label is continuous.

The dataset was taken from StatLib archive that is maintained at Carnegie Mellon University and is publicly available at https://lib.stat.cmu.edu/datasets/boston. The data itself was collected by U.S Census Service in Boston, Massachusetts. Furthermore, the dataset is published by Harrison, D. and Rubinfeld, D.L. in their "Hedonic prices and the demand for clean air" in 1978 [8]. There are 13 features and 511 data points in total, refer to Appendix for comprehensive descriptions of the features.

	CRIM	INDUS	NOX	RM	AGE	RAD	TAX	PTRATIO	LSTAT
MEDV	-0.38	-0.46	-0.41	0.67	-0.37	-0.38	-0.46	-0.45	-0.56

Table 1: The correlations between median house values and the possible features. Only includes the features that have an absolute value of over 0.35 for the correlation.

3 Methods

3.1 Feature selection

As features should be properties of data points that can be measured and computed easily in an automated fashion [5], it is necessary to leave and select some features from the dataset.

After visualizing the potential feature columns as x-axes and the label values in the y-axis (see Appendix) as a scatter plot, it is clear that the index of accessibility to radial highways (RAD), the vicinity of the Charles river (CHAS), the pupil-to-teacher ratio (PTRATIO), and the proportion of residential land zoned for lots over 25,000 square (ZN) feet are not very strongly correlated to the median house value. This is because the median house value can be high at any point in the scatter plot, regardless of the values of these features.

Moreover, as a correlation matrix is plotted between the feature columns the label column (see Appendix), only columns CRIM, INDUS, NOX, RM, AGE, RAD, TAX, PTRATIO and LSTAT seem to have a correlation that either over 0.35 or less than -0.35. When these values combined with the earlier plots are considered, columns CRIM, INDUS, NOX, RM, AGE, RAD, TAX, and LSTAT are left as possible features. These will be the selected features for later predictions.

3.2 Feature engineering

3.2.1 Detection of outliers

From the previous scatter plots, some outliers are easy to spot. For example the data point at row 509 has the highest MEDV-value (67.0), but scores relatively high on some features that normally affect negatively to the MEDV value, for example on the LSTAT-plot. However, as this data point seems to otherwise have rather normal feature values, it will not be removed from the dataset (see red dots in the Appendix).

Moreover, there are five missing values in the RM-feature (see Appendix). Since there are only five missing values, we will replace them with the median of the feature.

3.2.2 Data normalization

The dataset includes two categorical, but because they are already encoded as numerical values (in binary and from 1 to 24, see Appendix), no further encoding is needed.

3.3 ML model choice

The housing price problem is a typical regression problem, as different prices form a continuous distribution. To address this task, the first model that will be used is linear regression, as it aims to keep the model both simple and accurate. The model suits our purpose, because based on the previous scatter plots, there seems to be a linear correlation between some features and the label (see Appendix). Linear regression tries to find the best-fit relationship between the features and the target variable ($\hat{y}$), which can be represented by a hyperplane with multiple variables. [10]

As we have multidimensional data, it is best to use a multivariate linear regression model, which can take all of them into account. The formula for multivariate linear regression model is:

$$y_i = \alpha + w_1 x_i^{(1)} + w_2 x_i^{(2)} + ... + w_n x_i^{(n)}$$

where y_i is the predicted value for the i^{th} element of the label y, n denotes the number of independent variables, w_i is the i^{th} parameter and x_i^j represents the i^{th} value of the j^{th} feature [4]. After the training process, the model will have found the optimal parameter vector $\boldsymbol{w}$. Thus, the formal definition for our hypothesis space is when taking into account the categorical values in the dataset:

$$\mathcal{H}^{(n)} := \{h^{(\mathbf{w})} : R^{(n-2)} \times \{0,1\} \times \{1,2,\ldots,24\} \to R : h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}\} \quad (1)$$

where $\mathbf{w} \in R^n$ and n is the number of features.

Another supervised learning model considered is the Random Forest Regressor, which applies the bagging technique and leverages ensemble learning to perform regression tasks. The model is selected due to its strong performance in both classification and regression tasks, its ability to work with varying data types, and because it can also take the potential non-linear relationships

of our dataset into account [7]. A decision tree is also a supervised learning model, where data is repeatedly split into multiple decision trees in parallel based on specific parameters or features [2]. When the Random Forest model makes a prediction, it aggregates the outputs from all decision trees by averaging their predictions to generate the final result. The hypothesis space of this model is generated through bagging and is produced by merging the predictions $h^{(b)}(\mathbf{x})$ from each individual hypothesis (decision tree) $h^{(b)}$, for each $b = 1, ..., B$ in the ensemble by taking the average [5]. In other words, the hypothesis space is all possible combinations of decision trees that the model can come up with.

3.4 Loss function

The dataset contains rather many features compared to the amount of data points, so it is useful to harness a regularization parameter in the loss function. There are two options for the parameter: Lasso and Ridge. The main difference between these two lies in how they handle large coefficients. The Ridge loss function tends to penalize high coefficients more aggressively, while the Lasso penalty minimizes model parameters more uniformly, without giving special priority to large coefficients. [3]

This difference arises from the formulas used in each case: Ridge uses the L2-norm, while Lasso employs the L1-norm as a regularization term. Both of these loss functions are effective for linear regression. However, Lasso regularization is preferred because it can force a weight to be zero, which might be useful since some features might be completely irrelated to the labels even after the feature selection. Additionally, the Lasso regularization usually delivers a hypothesis with smaller expected loss with a slight increase in the computational complexity, which is not a problem with the small dataset in this project [5]. The Lasso loss is calculated using the formula:

$$L(\mathbf{w}) = \frac{1}{2n}\sum_{i=1}^{n}((w_0 + w_1 x_1^{(i)} + ... + w_p x_p^{(i)}) - y_i)^2 + \frac{\lambda}{2n}\sum_{j=1}^{p}|w_j|$$

where the term $\frac{1}{2n}$ normalizes the mean squared error (MSE) term, λ is the regularization term (which can be manually fine-tuned), and $\sum_{j=1}^{p}|w_j|$ is the sum of the absolute values of the coefficients.

To determine the optimal value for λ, we use mean squared error on the cross-validation set for a range of possible lambda values.[9]

In the Random Forest Regressor, the MSE loss function is used since it is a regression task, whereas other common RFR loss functions such as Gini Index are meant for classification tasks. [6] Additionally, it is available in the scikit-learn library as a built-in parameter in the respective class. Another possible loss function for the Random Forest Regressor is the mean absolute error (MAE), which is the average of the absolute differences between the predicted value and the true value of the target [11]. The mean squared error (MSE) is calculated using the formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

, where $\hat{y}_i$ is the predicted value of a target variable and y_i is the actual true value [12]. MSE is also used to calculate the validation error and compare the models to each other, as it is the most well-known regression error metric and both models support it.

3.5 Model validation

There are several ways to split data into different sets, but let's consider two common approaches. The first approach is to split the data into a training set and a test set. The original dataset is shuffled, and a predefined ρ percent of the data points are randomly assigned to the training set, while the remaining $(1-\rho)$ percent are assigned to the validation set. It is also common to further split the test set into two parts: a cross-validation set and a validation set. The cross-validation set is used for fine-tuning the model, while the validation set measures how well the model's predictions generalize to unseen data.

Another widely-used technique is k-Fold Cross Validation (k-Fold CV), which is more advanced compared to a single split. In k-Fold CV the original dataset is divided evenly into k folds (subsets). Then, one fold is selected as the test set, while the remaining folds form the training set. This process is repeated for all k folds. After all iterations are complete, the average error across all possible folds is calculated for reliable estimates. This technique is particularly useful for small datasets, as it helps mitigate the issue of the validation set containing only outliers. [5]

However, in this project, the dataset is large enough and there are not too many outliers based on the previous scatter plots, making the simpler method result in subsets with enough variance. According to [1], the training set should contain about 67% ($\rho = 0.67$) of the data, with the cross-validation and validation sets each containing about 16.5% of the data points. The cross-validation

is especially needed in this task, since we are using a regularization parameter which needs to be tuned for an optimal value. If the model performs poorly on the validation set, the k-Fold CV technique may be considered as an alternative. With these split ratios, the size of training set is 342, validation set is 84 and cross-validation set is 85. The sizes are large enough to get trustworthy results from validation and testing after the training.

4 Results

Both the Random Forest Regressor and Linear Regression with Lasso Regression are first trained on the training dataset specified earlier. For the latter one, the best possible value for the regularization parameter is first found by trial-and-error for values from 0.0001 to 1 with 50 different values in between. In each iteration, the model is trained with the cross-validation set and then evaluated with mean squared error. The regularization parameter value with smallest cross-validation error value is chosen (see Appendix). Afterwards, also the Lasso Regression is trained normally with the training dataset.

After the training, both sets are evaluated with the validation set, which has not been part of the training process. In addition, the training error is measured. Both results are in the table 2.

Model	Validation error	Train error	Regularization parameter
Linear Regression with Lasso	41.83	39.87	0.034
Random Forest Regressor	17.54	2.49	None

Table 2: The training and test errors of each model, rounded to two decimals.

For the Lasso Linear Regression, both the validation set and training set errors are quite high. For the Random Forest Regressor, the validation error is more than two times better than with the other method, and the training error is more than ten times better. Based on the errors, it is clear that Random Forest Regressor works better in the housing price dataset. Thus, the final chosen method is Random Forest Regressor.

The final method is evaluated with mean squared error against the previously created cross-validation set, because the Random Forest Regressor, unlike the Lasso Linear Regressor, has not seen this dataset previously. This is a preferred option because further splitting of the dataset would lead to smaller training set which might cause changes in the performance of the model. If the final chosen model was Lasso Linear Regressor, constructing a new set for final test error would be necessary. Please refer to Section 3.5 to see how the cross-validation set (hence final test set) was constructed.

With the cross-validation set as the final test set, the test error is around 20.27 (see Appendix). This is worse than the test error, but still lower than with the Lasso Linear Regressor. The reason for this might be the worse luck of selecting the dataset for measurement, there might be more outliers or exceptional values in the test dataset.

5 Conclusion

This paper analyzed both Linear Regression with Lasso Regularization Parameter and Random Forest Regressor on their performance for predicting Boston housing prices. The model with the best performance was Random Forest Regressor, so it was chosen as the final model for this task. It achieved a training error of 2.49 and a validation error of 17.54. The difference between these two might be that the model is overfitting to the training data. The final test error was 20.27, which is quite close to the validation error, which probably is because worse luck when selecting the test set compared to the validation set. So, the test error on the price prediction was around 4000$, which is around 21% of the mean house price (see Appendix).

There is some room for improvement. The dataset is fairly small with around 500 values, since with different runs of the model the accuracy and errors fluctuated quite a lot, so the most straightforward way to improve is to gather more data. Also normalization of the features, so that they are all have values in between 0 and 1, could have improved the training process because then no feature is overpowering others. Moreover, some of the features such as CRIM seem to be very skewed (see Appendix), so a logarithmic or a square root transformation to reduce skewness could have improved the results furthermore.

References

[1] J. Brownlee. Train-test split for evaluating machine learning algorithms. *Machine Learning Mastery*, 2020.

[2] Builtin.com. What is decision tree classification? Contributed by: Afroz Chakure, Accessed: 2024-10-07.

[3] M. L. Compass. Lasso regression explained, step by step. Accessed: 2024-09-17.

[4] HackerEarth. Multivariate linear regression. Contributed by: Shubhakar Reddy Tipireddy, Accessed: 2024-09-16.

[5] A. Jung. *Machine Learning: The Basics*. Springer, 2022.

[6] D. D. of Data Science. An algorithm-wise summary of loss functions in machine learning. Contributed by: Avi Chawla, Accessed: 2024-10-07.

[7] N. Sahai. Random Forest Regression – How it Helps in Predictive Analytics? Contributed by: Afroz Chakure, Accessed: 2024-10-07.

[8] U. C. Service. The boston housing dataset. *StatLib archive*, 1978.

[9] R. Sneiderman. From linear regression to ridge regression, the lasso, and the elastic net. *Towards Data Science*, 2020.

[10] Y. University. Linear regression. Accessed: 2024-09-16.

[11] A. Vijayan, A. Vasuki, A. C. JOHNVICTOR, S. Ranganathan, P. Rishi, and S. Edward. Enhancing software testing with machine learning techniques. page 332, 2023.

[12] vitalflux.com. Mse vs rmse vs mae vs mape vs r-squared: When to use? Contributed by: Ajitesh Kumar, Accessed: 2024-10-07.

6 Appendix

Project

October 7, 2024

0.1 Imports

[35]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
import math
scaler = MinMaxScaler()
data = pd.read_csv("./housing_prices.csv")
```

[36]:
```
# Check how many unique values each column has
descs = {'CRIM': 'distance to employment centre',
         'ZN': 'proportion of residential land zoned for lots over 25,000 sq.ft.
 ↪',
         'INDUS': 'proportion of non-retail business acres per town',
         'CHAS': 'Charles River dummy variable (1 if tract bounds river; 0␣
 ↪otherwise)',
         'NOX': 'nitric oxides concentration (parts per 10 million)',
         'RM': 'average number of rooms per dwelling',
         'AGE': 'proportion of owner-occupied units built prior to 1940',
         'DIS': 'weighted distances to five Boston employment centres',
         'RAD': 'index of accessibility to radial highways',
         'TAX': 'full-value property-tax rate per $10,000',
         'PTRATIO': 'pupil-teacher ratio by town',
         'B': '1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town',
         'LSTAT': '% lower status of the population',
         'MEDV': 'Median value of owner-occupied homes in $1000 s'
        }
for col in data.columns:
    unique_count = data[col].nunique()
```

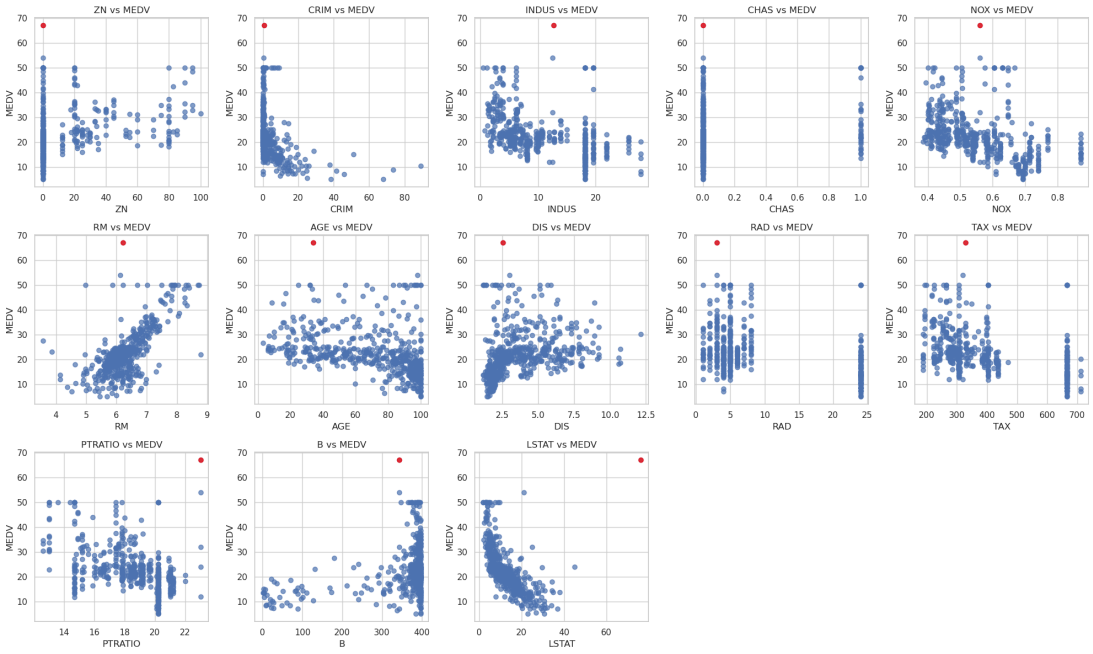
```
    print(f"Column '{col}' has {unique_count} unique values, description:␣
 ↪{descs[col]}.")
# Get minimum and maximum value for RAD.
print(" \n Min value in RAD is " + str(min(data['RAD'])) + " and max is " +␣
 ↪str(max(data['RAD'])) + ".")
```

```
Column 'CRIM' has 509 unique values, description: distance to employment centre.
Column 'ZN' has 26 unique values, description: proportion of residential land
zoned for lots over 25,000 sq.ft..
Column 'INDUS' has 79 unique values, description: proportion of non-retail
business acres per town.
Column 'CHAS' has 2 unique values, description: Charles River dummy variable (1
if tract bounds river; 0 otherwise).
Column 'NOX' has 82 unique values, description: nitric oxides concentration
(parts per 10 million).
Column 'RM' has 444 unique values, description: average number of rooms per
dwelling.
Column 'AGE' has 357 unique values, description: proportion of owner-occupied
units built prior to 1940.
Column 'DIS' has 416 unique values, description: weighted distances to five
Boston employment centres.
Column 'RAD' has 9 unique values, description: index of accessibility to radial
highways.
Column 'TAX' has 67 unique values, description: full-value property-tax rate per
$10,000.
Column 'PTRATIO' has 47 unique values, description: pupil-teacher ratio by town.
Column 'B' has 360 unique values, description: 1000(Bk - 0.63)^2 where Bk is the
proportion of blacks by town.
Column 'LSTAT' has 460 unique values, description: % lower status of the
population.
Column 'MEDV' has 231 unique values, description: Median value of owner-occupied
homes in $1000 s.

 Min value in RAD is 1 and max is 24.
```

0.2 Data analysis

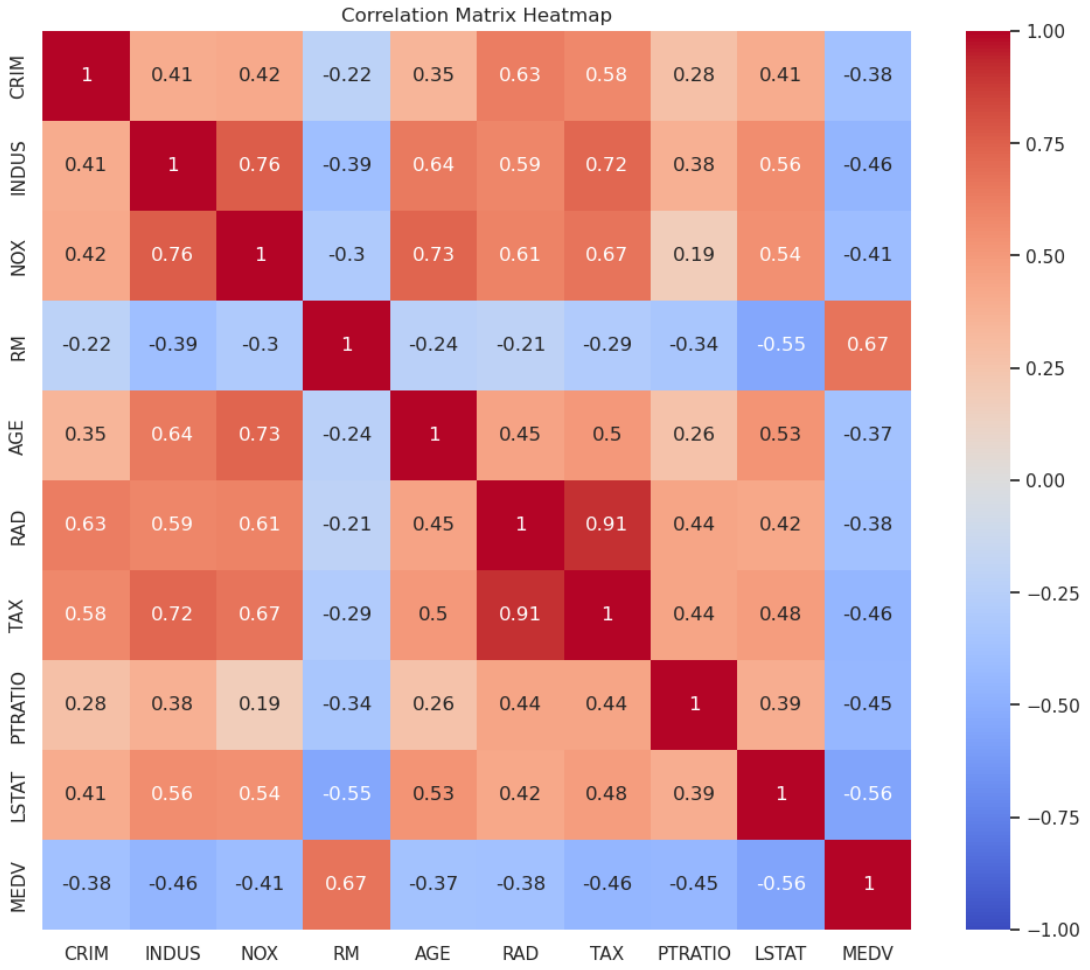
[37]:
```
# Simple XY scatter plots between label and different features
feature_candidates = ['ZN', 'CRIM', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',␣
 ↪'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
num_features = len(feature_candidates)
num_rows = 3
num_cols = (num_features + num_rows - 1) // num_rows
plt.figure(figsize=(num_cols * 4, num_rows * 4))
# Pick an outlier manually
a = data[data['LSTAT'] > 50]
```

```
for i, feature in enumerate(feature_candidates):
    plt.subplot(num_rows, num_cols, i + 1)
    plt.scatter(data[feature], data['MEDV'], alpha=0.7)
    plt.scatter(a[feature], a['MEDV'], color='red', label='LSTAT > 50', alpha=0.
 ↪7)

    plt.xlabel(feature)
    plt.ylabel('MEDV')
    plt.title(f'{feature} vs MEDV')
    plt.gca()
plt.tight_layout()
plt.show()
```



[38]:
```
# Draw a correlation matrix (columns compared to other columns)
# Note especially the last column with comparisons to the label column
corr_matrix = data.corr()
medv_corr = corr_matrix['MEDV']

# Filter out columns with correlation between -0.35 and 0.35 with 'MEDV'
columns_to_keep = medv_corr[~medv_corr.abs().between(-0.35, 0.35)].index
filtered_data = data[columns_to_keep]

filtered_corr_matrix = filtered_data.corr()
plt.figure(figsize=(12, 10))
```

```
sns.heatmap(filtered_corr_matrix, annot=True, cmap='coolwarm', center=0,␣
 ↪vmin=-1, vmax=1)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



[39]:
```
# See rows which have missing values.
rows_with_nan = filtered_data[filtered_data.isnull().any(axis=1)]
print(rows_with_nan)
filtered_data = filtered_data.fillna(filtered_data.median())
filtered_data['RM'].median()
```

```
      CRIM  INDUS    NOX  RM   AGE  RAD  TAX  PTRATIO  LSTAT  MEDV
10 0.22489   7.87  0.524 NaN  94.3    5  311     15.2  20.45  15.0
35 0.06417   5.96  0.499 NaN  68.2    5  279     19.2   9.68  18.9
63 0.12650   5.13  0.453 NaN  43.4    8  284     19.7   9.50  25.0
96 0.11504   2.89  0.445 NaN  69.6    2  276     18.0  11.34  21.4
```

```
135  0.55778  21.89  0.624 NaN  98.2    4  437     21.2  16.96  18.1
```

[39]:
```
6.209
```

0.3 Split into training, testing and validation sets

[40]:
```
# Split into different training, testing and validation sets
train_size = 0.67
test_size = 1-0.67
cross_size = 0.5
X = filtered_data.drop(columns=['MEDV'], axis=1)
y = filtered_data['MEDV']

X_train, X_splitted, y_train, y_splitted = train_test_split(X, y,␣
 ↪test_size=test_size, train_size=train_size, shuffle=True, random_state=43)
X_test, X_cross, y_test, y_cross = train_test_split(X_splitted, y_splitted,␣
 ↪test_size = cross_size, shuffle=True, random_state = 43)
print("Size of training set: " + str(len(X_train)))
print("Size of testing set: " + str(len(y_test)))
print("Size of testing set: " + str(len(y_cross)))
```

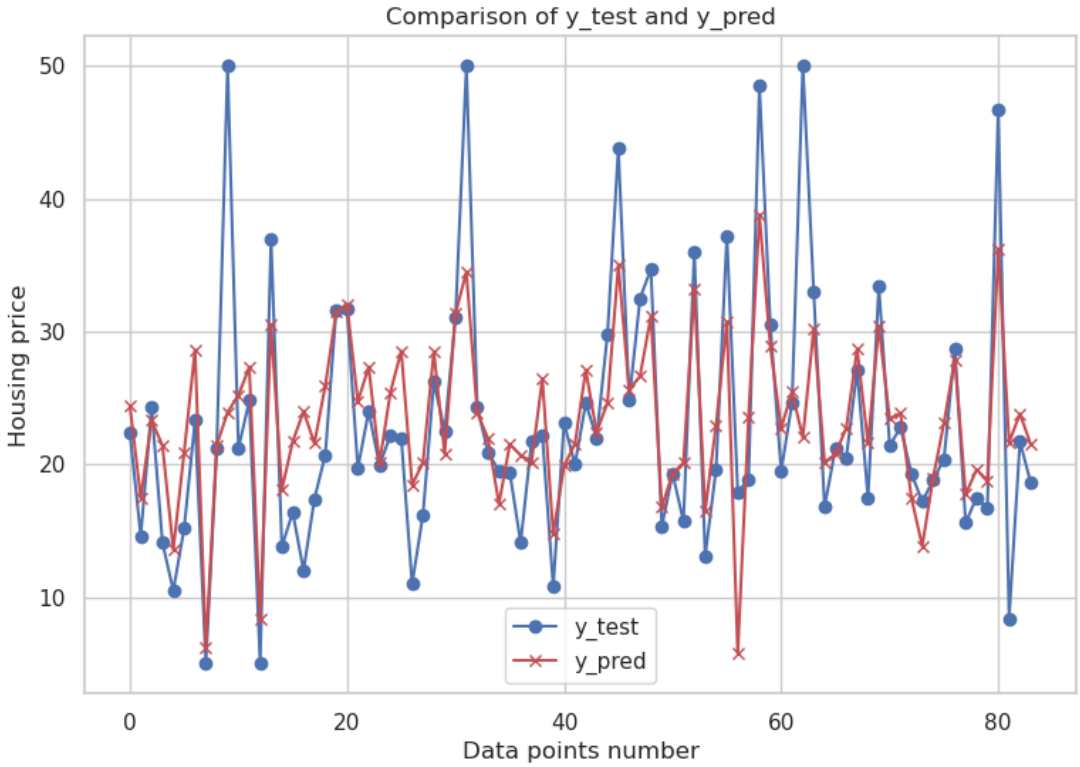
```
Size of training set: 342
Size of testing set: 84
Size of testing set: 85
```

0.4 Perform Lasso Linear Regression

[41]:
```
clf = Lasso()
parameter_options = {'alpha': np.logspace(-4, 0, 50)}  # parameter options from␣
 ↪10^(-4) to 1
# Search for the best parameter value with the help of mean squared error for␣
 ↪each iteration
grid_search = GridSearchCV(clf, parameter_options, cv=5,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1)
# Note that we use the cross-validation set to search for the best alpha value
grid_search.fit(X_cross, y_cross)
best_alpha = grid_search.best_params_['alpha']

# Create the model with the previously gotten best alpha value
final_clf = Lasso(alpha=best_alpha)
# Fit the training set
final_clf.fit(X_train, y_train)
y_pred = final_clf.predict(X_test)
y_train_pred = final_clf.predict(X_train)
y_test = np.array(y_test)
mse = mean_squared_error(y_test, y_pred)
R2_loss = final_clf.score(X_test, y_test)
```

```
train_loss = mean_squared_error(y_train, y_train_pred)
print("Mean squared test error with the best regularization parameter: " +␣
 ↪str(mse))
print("Best regularization parameter value: " + str(best_alpha))
print("R2 error with the best regularization parameter: " + str(R2_loss))
print("Mean squared error with training set: " + str(train_loss))
# Visualize the results
plt.figure(figsize=(9, 6))
plt.plot(y_test, label='y_test', marker='o', color='b')
plt.plot(y_pred, label='y_pred', marker='x', color='r')
plt.title('Comparison of y_test and y_pred')
plt.xlabel('Data points number')
plt.ylabel('Housing price')
plt.legend()
plt.show()
```

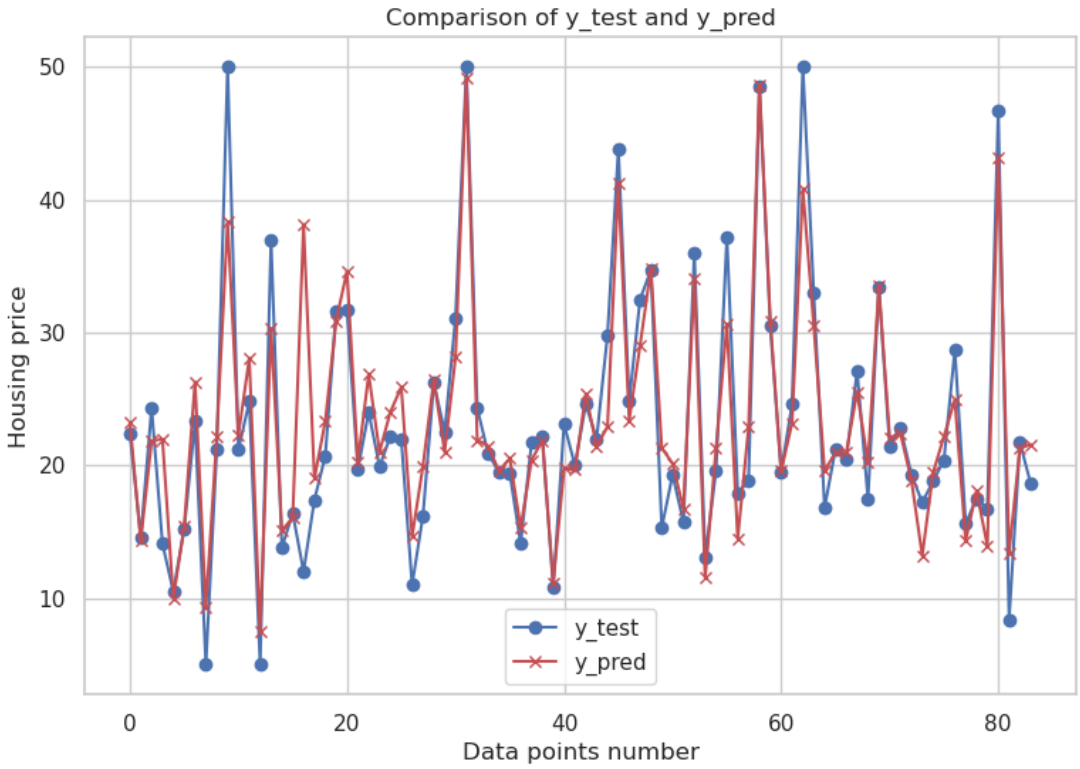
```
Mean squared test error with the best regularization parameter:
39.87078951582707
Best regularization parameter value: 0.033932217718953266
R2 error with the best regularization parameter: 0.571683148497583
Mean squared error with training set: 41.83446620198448
```



0.5 Perform random forest regressor

[42]:
```
rfr = RandomForestRegressor(random_state=0, criterion='squared_error')
rfr.fit(X_train, y_train)
y_pred_rfr = rfr.predict(X_test)
y_train_pred_rfr = rfr.predict(X_train)
y_cross_pred_rfr = rfr.predict(X_cross)

mse = mean_squared_error(y_test, y_pred_rfr)
mse_train = mean_squared_error(y_train_pred_rfr, y_train)
cross_validation_mse_rfr = mean_squared_error(y_cross, y_cross_pred_rfr)
print("Mean squared error of RandomForestRegressor with test set: " + str(mse))
print("Mean squared error of RandomForestRegressor with train set: " +␣
 ↪str(mse_train))
print("Final test error with cross-validation set: " +␣
 ↪str(cross_validation_mse_rfr))

plt.figure(figsize=(9, 6))
plt.plot(y_test, label='y_test', marker='o', color='b')
plt.plot(y_pred_rfr, label='y_pred', marker='x', color='r')
plt.title('Comparison of y_test and y_pred')
plt.xlabel('Data points number')
plt.ylabel('Housing price')
plt.legend()
plt.show()
```

```
Mean squared error of RandomForestRegressor with test set: 17.54466514285715
Mean squared error of RandomForestRegressor with train set: 2.4881796900584776
Final test error with cross-validation set: 20.27085898823528
```



0.6 Skewness and error percentage

[43]:
```
import seaborn as sns
sns.set(style="whitegrid")

fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(16, 12))
fig.suptitle("Distributions of Features", fontsize=16)

axes = axes.flatten()
med = y.median()
perc = round(math.sqrt(cross_validation_mse_rfr) / med * 100, 2)
print("Percentage of the final test error compared to the median house value: "␣
 ↪+ str(perc))
for i, feature in enumerate(columns_to_keep):
    sns.histplot(filtered_data[feature], ax=axes[i], kde=True, bins=30)
    axes[i].set_title(feature)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout(rect=[0, 0, 1, 0.95])
```

```
plt.show()
```

```
Percentage of the final test error compared to the median house value: 21.24
```