OPTION #1: Simple Linear Regression in Scikit Learn

Scott Miner

Colorado State University – Global Campus

Abstract

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.dummy import DummyRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.utils import shuffle
from itertools import combinations
#set output options

params = {'legend.fontsize': 20,
          'figure.figsize' : (15, 5),
         'axes.labelsize'  : 30,
         'axes.titlesize'  : 30,
         'xtick.labelsize' : 25,
         'ytick.labelsize' : 25,
         'lines.markersize': 25,
         'lines.linewidth' : 5}
plt.rcParams.update(params)
pd.options.display.width = 0
pd.options.display.float_format = '{:,.2f}'.format

#load data as a bunch object (bo)
boston_dataset = load_boston()
print(f'Loading boston dataset...')
print(f'Type dataset object: {type(boston_dataset)}')
print(f'Dataset keys: {boston_dataset.keys()}')

# create dataframe
boston_df = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
# add target column
boston_df['MEDV'] = boston_dataset.target
# apply log transformation
boston_df['LOGLSTAT'] = boston_df['LSTAT'].apply(np.log)

# basic dataset informatoin
rows, cols = boston_df.shape
print(f'Dataset rows: {rows}')
print(f'Dataset cols: {cols} ({cols-1} features and 1 target variable)')
print(f'Total elements in dataset: {boston_df.size}')
print('----------------------------------------------------------------------')
# print column descriptions
lines = boston_dataset.DESCR.splitlines()[11:28]
print('Description of cols:')
print(*lines, sep='\n')
print('----------------------------------------------------------------------')
print('Info:')
print(boston_df.info())
print('----------------------------------------------------------------------')
print(boston_df.describe())
```

Figure 1. Program code (part 1)

```
print('-----------------------------------------------------------------------')
print('First 5 and last 5 rows of the dataset:')
print(pd.concat([boston_df.head(), boston_df.tail()]))

# graphs

############### heatmap ###############
mask = np.zeros_like(boston_df.corr())
mask[np.triu_indices_from(mask)] = True

ax = sns.heatmap(boston_df.corr().round(2), square=True,
                 annot=True, mask=mask, cmap='coolwarm', annot_kws={'size': 20})
ax.set_title('Heatmap of the Boston Housing Dataset',
             fontsize=25)
ax.tick_params(axis='both', labelsize=20)
# color bar object
cbar = ax.collections[0].colorbar
cbar.ax.tick_params(labelsize=20)

############### LSTAT distribution ###############
g = sns.displot(boston_df['LSTAT'], bins=30)
g.fig.suptitle('Distribution of LSTAT\n(% lower status of the population)',
               fontsize=25)
g.axes[0,0].set_xlabel('LSTAT', fontsize=20)
g.axes[0,0].set_ylabel('Count', fontsize=20)
g.axes[0,0].tick_params(axis='both', labelsize=20)

############### Histogram: all variables ###############
fig, axes = plt.subplots(round(len(boston_df.columns)/3),3,figsize=(20, 40))
i = 0
for triaxis in axes:
    for axis in triaxis:
        if i < len(boston_df.columns):
            boston_df.hist(column = boston_df.columns[i], bins = 100, ax=axis)
            axis.set_title(boston_df.columns[i], fontsize=25)
            axis.tick_params(axis='both', labelsize=20)
            i += 1
fig.suptitle('Histograms for All Variables in the Boston Housing Dataset',
             fontsize=26)

############### Scatterplots ###############
features = ['LSTAT', 'RM']
target = boston_df['MEDV']
fig, axes = plt.subplots(1, len(features), figsize=(10,5))

for col,ax in zip(features, axes.flat):
    x = boston_df[col]
    y = target
    ax.scatter(x,y,marker='o',s=150)
    ax.set_title(col, fontsize=26)
    ax.set_xlabel(col, fontsize=20)
    ax.set_ylabel('MEDV', fontsize=20)
    ax.tick_params(axis='both', labelsize=20)
fig.suptitle('Scatterplot showing the LSTAT and RM variables against MEDV',
```

No issues found

Figure 2. Program code (part 2)

```
fig.suptitle('Scatterplot showing the LSTAT and RM variables against MEDV',
              fontsize=30)

############### Transformations ###############
fig, axes = plt.subplots(1, 2)
xs = ['LSTAT', 'LOGLSTAT']
colors = ['green', 'red']
x_line_points = [[0,40],[0,4]]
y_line_points = [[30,0],[50,0]]

for ax,x,color,x_lp,y_lp in zip(axes.flat,xs,colors,
                                x_line_points,y_line_points):
    ax.scatter(boston_df[x], boston_df['MEDV'], color=color, s=150)
    ax.set_xlabel(x,fontsize=20)
    ax.set_ylabel('MEDV',fontsize=20)
    ax.tick_params(axis='both', labelsize=20)
    ax.plot(x_lp,y_lp)

def rms_error(actual, predicted):
     ' root-mean-squared-error function'
     # lesser values are better(a<b means a is better)
     mse = mean_squared_error(actual, predicted)
     return np.sqrt(mse)
rms_scorer = make_scorer(rms_error)

boston_ftrs = boston_df[['LOGLSTAT', 'RM']]
boston_tgt = boston_df['MEDV']

############### Train-Test-Split ###############
boston_tts = train_test_split(boston_ftrs, boston_tgt,random_state=2021)
(boston_train_ftrs, boston_test_ftrs,
 boston_train_tgt,  boston_test_tgt) = boston_tts
print('------------------------------------------------------------------')
print('Split the data into training and test features:' )
print(f'Training Features: {boston_train_ftrs.shape}')
print(f'Training Target: {boston_train_tgt.shape}')
print(f'Testing Features: {boston_test_ftrs.shape}')
print(f'Testing Target: {boston_test_tgt.shape}')
print('------------------------------------------------------------------')
# preprocessing
scaler = StandardScaler()
# create models
lr = LinearRegression()
lasso = Lasso()
ridge = Ridge()

std_lr_pipe = make_pipeline(scaler, lr)
std_lasso_pipe = make_pipeline(scaler, lasso)
std_ridge_pipe = make_pipeline(scaler, ridge)

# Create models
regressors = {'baseline'        : DummyRegressor(strategy='mean'),
              'std_lr_pipe'     : std_lr_pipe,
              'std_lasso_pipe'  : std_lasso_pipe,
              'std_ridge_pipe'  : std_ridge_pipe}
```

Figure 3. Program code (part 3)

```
fig, ax = plt.subplots(1, 1, figsize=(8,4))
scores = {}
for mod_name, model in regressors.items():
    cv_results = cross_val_score(model,
                                  boston_train_ftrs, boston_train_tgt,
                                  scoring = rms_scorer,
                                  cv=10)
    key = mod_name
    scores[key] = [cv_results.mean(), cv_results.std()]
    lbl = f'{mod_name:s} ({cv_results.mean():5.3f})$\pm${cv_results.std():.2f}'
    ax.plot(cv_results, 'o--', label=lbl, markersize=11)
    ax.set_xlabel('CV-Fold #')
    ax.set_ylabel('RMSE')
    ax.legend(bbox_to_anchor=(1.00, 1.00), fancybox=True, shadow=True)

df = pd.DataFrame.from_dict(scores, orient='index').sort_values(0)
df.columns = ['RMSE', 'STD_DEV']
print('Results on training data')
print(df)
y_predicted = std_lr_pipe.fit(boston_train_ftrs, boston_train_tgt).predict(boston_test_ftrs)
#################### Regression Errors ##################################
from itertools import permutations
def regression_errors(figsize, predicted, actual, errors='all'):
    ''' figsize -> subplots;
        predicted/actual data -> columns in a DataFrame
        errors -> 'all' or sequence of indices
    '''
    fig, axes = plt.subplots(1, 2, figsize=figsize,
                             sharex=True, sharey=True)
    df = pd.DataFrame({'actual':actual,
                       'predicted': predicted})

    for ax, (x,y) in zip(axes, permutations(['actual',
                                             'predicted'])):
        # plot the data as '.'; perfect as y=x line
        ax.plot(df[x], df[y], '.', label='data')
        ax.plot(df['actual'], df['actual'], '-',
                label='perfection')
        ax.legend()

        ax.set_xlabel('{} Value'.format(x.capitalize()))
        ax.set_ylabel('{} Value'.format(y.capitalize()))
        ax.set_aspect('equal')

    axes[1].yaxis.tick_right()
    axes[1].yaxis.set_label_position('right')

    # show connecting bars from data to perfect
    # for all or only those specified?
    if errors == 'all':
        errors = range(len(df))
    if errors:
        acts = df.actual.iloc[errors]
```

Figure 4. Program code (part 4)

```
        acts = df.actual.iloc[errors]
        preds = df.predicted.iloc[errors]
        axes[0].vlines(acts, preds, acts, 'r')
        axes[1].hlines(acts, preds, acts, 'r')

regression_errors((10, 5), y_predicted, boston_test_tgt, errors=[0,1,2,3,4,
                                                                 122,123,124,125,126])

def regression_residuals(ax, predicted, actual,
                         show_errors=None, right=False):
    ''' figsize -> subplots;
        predicted/actual data -> columns of a DataFrame
        errors -> 'all' or sequence of indices
    '''
    df = pd.DataFrame({'actual':actual,
                       'predicted':predicted})
    df['error'] = df.actual - df.predicted
    ax.plot(df.predicted, df.error, '.')
    ax.plot(df.predicted, np.zeros_like(predicted), '-')

    if right:
        ax.yaxis.tick_right()
        ax.yaxis.set_label_position('right')
        ax.set_xlabel('Predicted Value')
        ax.set_ylabel('Residual')

    if show_errors == 'all':
        show_errors = range(len(df))
    if show_errors:
        preds = df.predicted.iloc[show_errors]
        errors = df.error.iloc[show_errors]
        ax.vlines(preds, 0, errors, 'r')

fig, ax= plt.subplots(1,1,figsize=(8,8))

regression_residuals(ax, y_predicted, boston_test_tgt, show_errors=[0,1,2,3,4,
                                                                    122,123,124,125,126])
ax.set_xlabel('Predicted')
ax.set_ylabel('Residual')


coeff = list(zip(boston_ftrs, std_lr_pipe.named_steps['linearregression'].coef_))

df_coeff = pd.DataFrame(coeff)
df_coeff.columns = ['variable', 'coeff']
df_coeff.set_index('variable')
df_intercept = pd.DataFrame([['INTERCEPT', std_lr_pipe.named_steps['linearregression'].intercept_]])
df_intercept.columns = ['variable', 'coeff']
df_results = df_intercept.append(df_coeff, ignore_index=True)
df_results = df_results.set_index(['variable'])
df_results.index.name=None
df_results
print('---------------------------------------------------------------------')
print('RMSE on hold-out data:')
test_score = np.sqrt(mean_squared_error(boston_test_tgt, y_predicted))
```

Figure 5. Program code (part 5)

```python
print('----------------------------------------------------------------------')
print('RMSE on hold-out data:')
test_score = np.sqrt(mean_squared_error(boston_test_tgt, y_predicted))
test_results = pd.DataFrame([('std_lr_pipe', test_score)])
test_results.columns = ['pipeline', 'RMSE']
test_results = test_results.set_index('pipeline')
test_results.index.name=None
print(test_results)
print('----------------------------------------------------------------------')
print('Coefficients:')
print(df_results)
print('----------------------------------------------------------------------')
print('Display True vs. Actual Values:')
y_test = pd.DataFrame(boston_test_tgt)
y_test['PREDS'] = y_predicted
y_test = y_test.rename(columns={'MEDV': 'TRUE'})
y_test = y_test[['PREDS', 'TRUE']]
print(y_test)

plt.show()
```

Figure 6. Program code (part 6)

```
C:\WINDOWS\system32\cmd.exe
Loading boston dataset...
Type dataset object: <class 'sklearn.utils.Bunch'>
Dataset keys: dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
Dataset rows: 506
Dataset cols: 15 (14 features and 1 target variable)
Total elements in dataset: 7590
----------------------------------------------------------------------
Description of cols:
    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - 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 black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None
----------------------------------------------------------------------
Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 15 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   CRIM      506 non-null    float64
 1   ZN        506 non-null    float64
 2   INDUS     506 non-null    float64
 3   CHAS      506 non-null    float64
 4   NOX       506 non-null    float64
 5   RM        506 non-null    float64
 6   AGE       506 non-null    float64
 7   DIS       506 non-null    float64
 8   RAD       506 non-null    float64
 9   TAX       506 non-null    float64
 10  PTRATIO   506 non-null    float64
 11  B         506 non-null    float64
 12  LSTAT     506 non-null    float64
 13  MEDV      506 non-null    float64
 14  LOGLSTAT  506 non-null    float64
dtypes: float64(15)
memory usage: 59.4 KB
None
----------------------------------------------------------------------
        CRIM     ZN  INDUS   CHAS    NOX     RM    AGE    DIS    RAD    TAX  PTRATIO      B  LSTAT   MEDV  LOGLSTAT
count 506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00   506.00 506.00 506.00 506.00    506.00
mean    3.61  11.36  11.14   0.07   0.55   6.28  68.57   3.80   9.55 408.24    18.46 356.67  12.65  22.53      2.37
std     8.60  23.32   6.86   0.25   0.12   0.70  28.15   2.11   8.71 168.54     2.16  91.29   7.14   9.20      0.60
min     0.01   0.00   0.46   0.00   0.39   3.56   2.90   1.13   1.00 187.00    12.60   0.32   1.73   5.00      0.55
25%     0.08   0.00   5.19   0.00   0.45   5.89  45.02   2.10   4.00 279.00    17.40 375.38   6.95  17.02      1.94
50%     0.26   0.00   9.69   0.00   0.54   6.21  77.50   3.21   5.00 330.00    19.05 391.44  11.36  21.20      2.43
75%     3.68  12.50  18.10   0.00   0.62   6.62  94.07   5.19  24.00 666.00    20.20 396.23  16.96  25.00      2.83
max    88.98 100.00  27.74   1.00   0.87   8.78 100.00  12.13  24.00 711.00    22.00 396.90  37.97  50.00      3.64
----------------------------------------------------------------------
First 5 and last 5 rows of the dataset:
     CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  LSTAT  MEDV  LOGLSTAT
0    0.01 18.00   2.31  0.00 0.54 6.58 65.20 4.09 1.00 296.00    15.30 396.90   4.98 24.00      1.61
1    0.03  0.00   7.07  0.00 0.47 6.42 78.90 4.97 2.00 242.00    17.80 396.90   9.14 21.60      2.21
2    0.03  0.00   7.07  0.00 0.47 7.18 61.10 4.97 2.00 242.00    17.80 392.83   4.03 34.70      1.39
3    0.03  0.00   2.18  0.00 0.46 7.00 45.80 6.06 3.00 222.00    18.70 394.63   2.94 33.40      1.08
4    0.07  0.00   2.18  0.00 0.46 7.15 54.20 6.06 3.00 222.00    18.70 396.90   5.33 36.20      1.67
501  0.06  0.00  11.93  0.00 0.57 6.59 69.10 2.48 1.00 273.00    21.00 391.99   9.67 22.40      2.27
502  0.05  0.00  11.93  0.00 0.57 6.12 76.70 2.29 1.00 273.00    21.00 396.90   9.08 20.60      2.21
503  0.06  0.00  11.93  0.00 0.57 6.98 91.00 2.17 1.00 273.00    21.00 396.90   5.64 23.90      1.73
504  0.11  0.00  11.93  0.00 0.57 6.79 89.30 2.39 1.00 273.00    21.00 393.45   6.48 22.00      1.87
505  0.05  0.00  11.93  0.00 0.57 6.03 80.80 2.50 1.00 273.00    21.00 396.90   7.88 11.90      2.06
----------------------------------------------------------------------
Split the data into training and test features:
```

Figure 7. Program output (part 1)

```
------------------------------------------------
Split the data into training and test features:
Training Features: (379, 2)
Training Target: (379,)
Testing Features: (127, 2)
Testing Target: (127,)
------------------------------------------------
Results on training data
          RMSE  STD_DEV
ridge     4.93     0.86
lr        4.93     0.85
lasso     5.20     1.02
baseline  9.47     1.52
------------------------------------------------
RMSE on hold-out data:
    RMSE
lr  5.05
------------------------------------------------
Coefficients:
           coeff
INTERCEPT  24.52
LOGLSTAT  -10.35
RM          3.58
------------------------------------------------
Display True vs. Actual Values:
     LSTAT   RM  PREDS  TRUE
210  17.27 5.96  16.37 21.70
24   16.30 5.92  16.84 15.60
36   11.41 5.84  20.24 20.00
439  22.88 5.63  12.27 12.80
161   1.73 7.49  45.66 50.00
..     ...  ...    ...   ...
336   9.80 5.87  21.91 19.50
385  30.81 5.28   7.94  7.20
352   7.79 5.88  24.34 18.60
106  18.66 5.84  15.13 19.50
391  18.76 6.05  15.84 23.20
```

Figure 8. Program output (part 2)

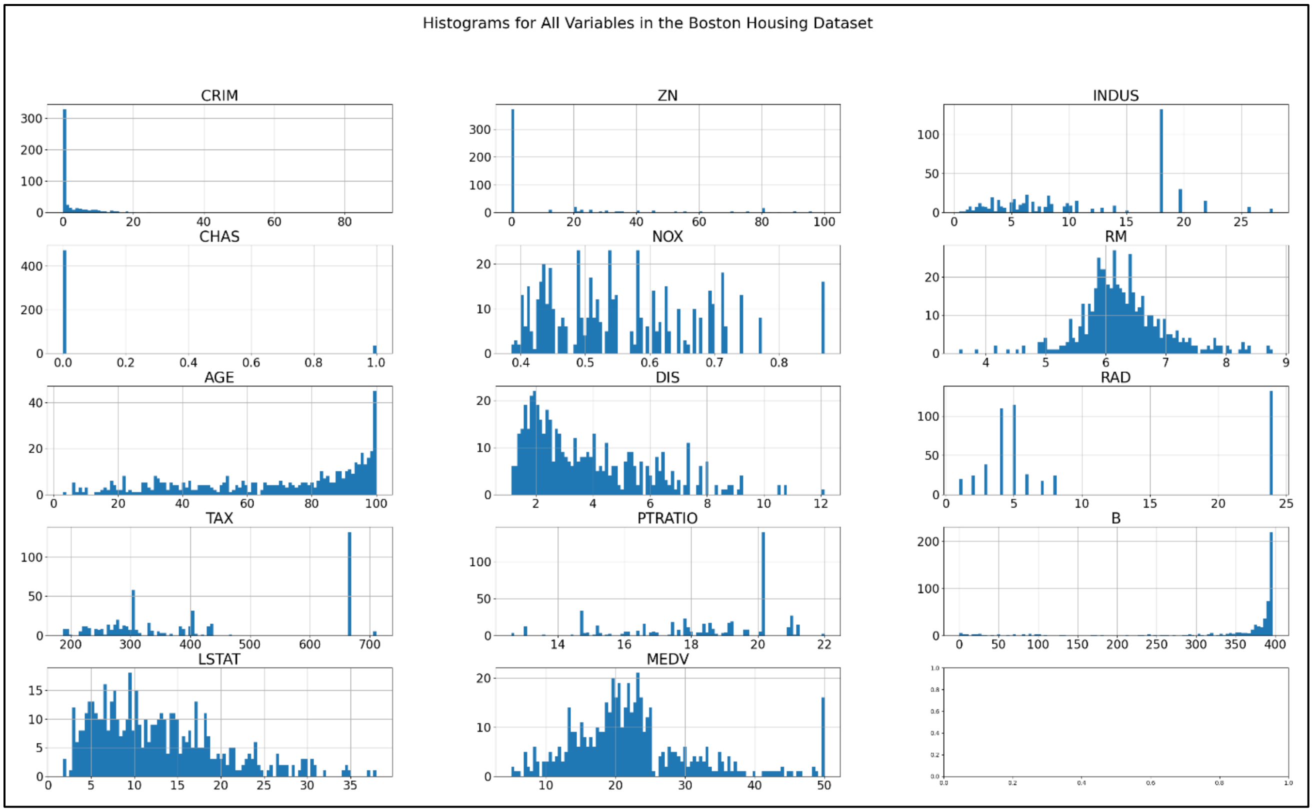

Figure 9. Histogram of variables in the Boston housing dataset

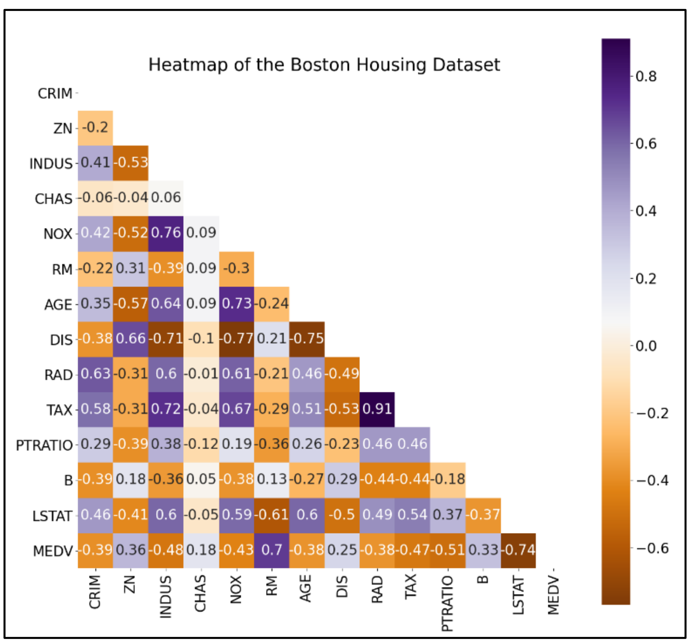

Figure 10. Heatmap of the Boston housing dataset

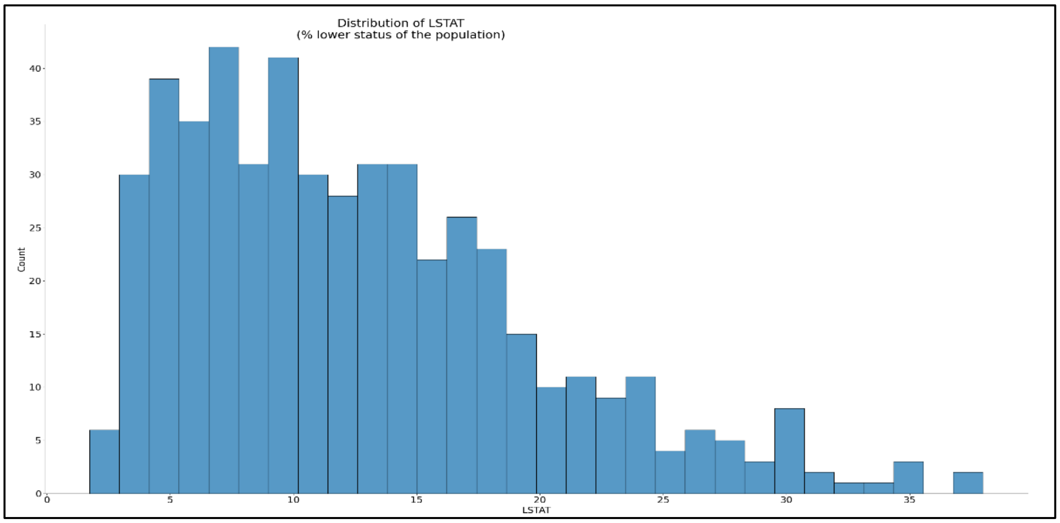

Figure 11. Distribution of the *LSTAT* variable

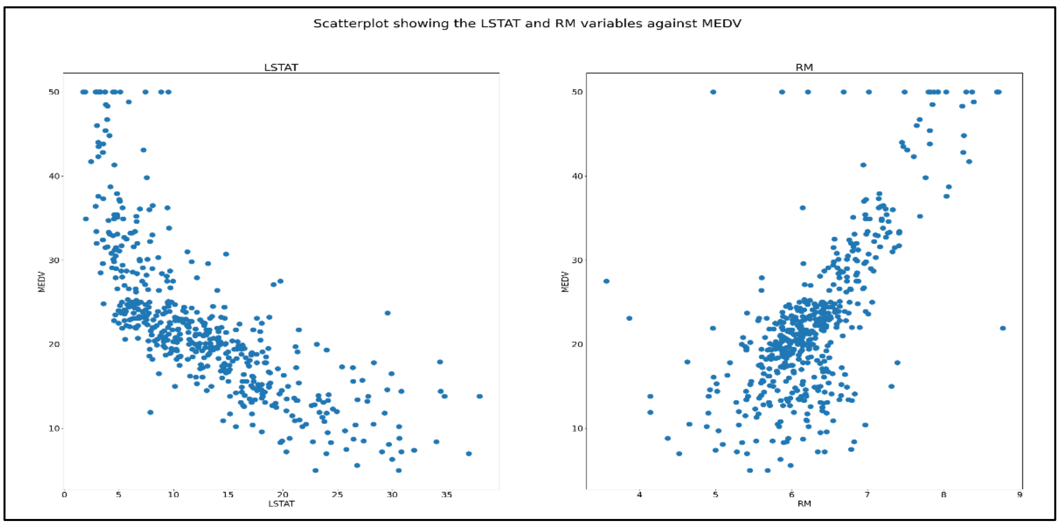

Figure 12. Scatterplots showing the *LSTAT* and *RM* variables against *MEDV*

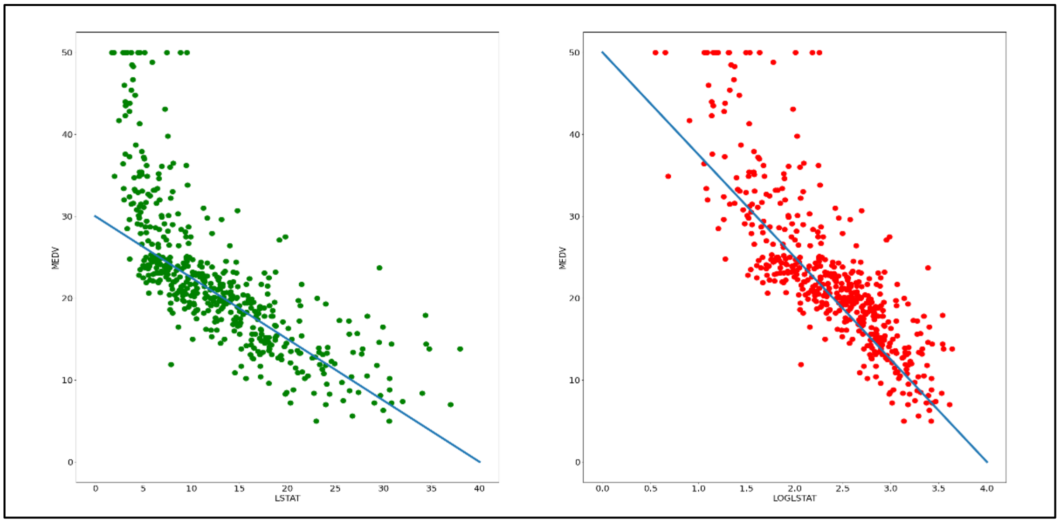

Figure 13. Performing a logarithmic transformation of the *LSTAT* variable

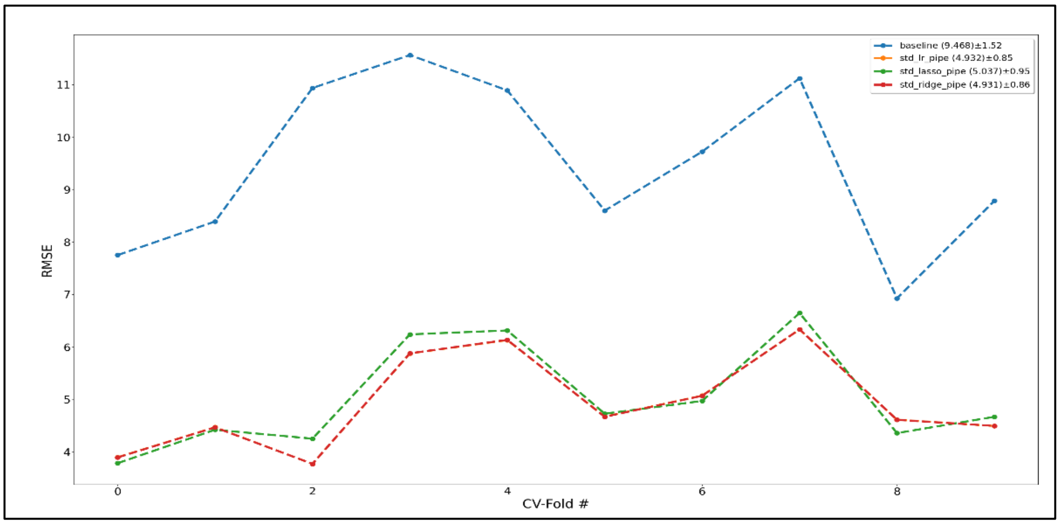

Figure 14. Model evaluation on 10-fold cross-validation

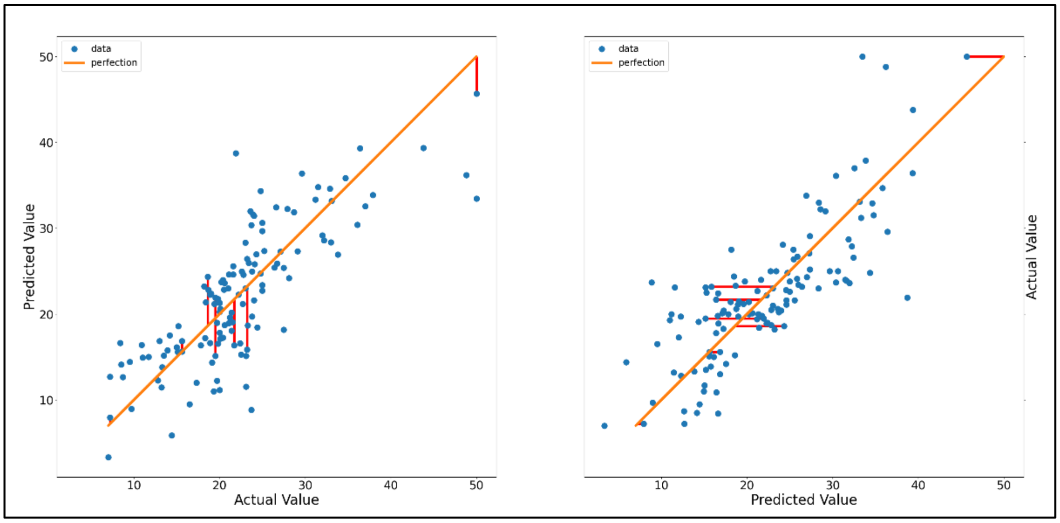

Figure 15. Regression Errors

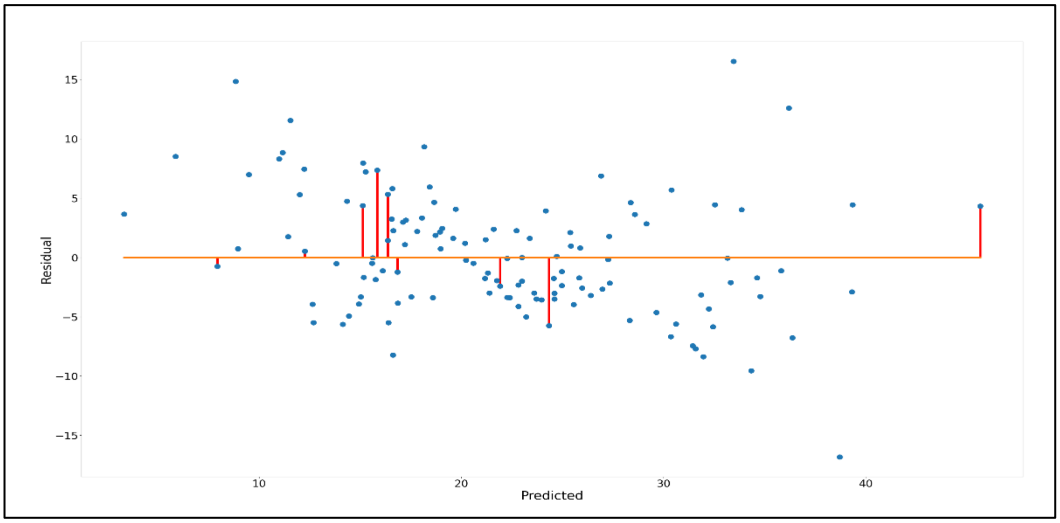

Figure 16. Residual Plot

OPTION #1: Simple Linear Regression in Scikit Learn

This paper describes a linear regression model in Python that predicts the median value of an owner-occupied home in the $1,000s from the Boston housing dataset. The dataset contains 506 rows and 14 features, totaling 7,804 values. Figure 7, the first half of the program's console output, shows each dataset feature's name and the corresponding description. Figures 1 – 6 show the code to create the program and its output. There are no null values in the dataset, and all the variables are floating-point numbers that occupy 64 bits in memory.

Exploratory Data Analysis (EDA)

The program outputs descriptive statistics for all the variables and the first and last five rows of the dataset. Figure 9 shows a histogram for all dataset variables. Agarwal (2018) writes that the target variable, *MEDV*, which indicates the median value of an owner-occupied home in the $1,000s, appears normally distributed with a few outliers. Figure 10 shows a correlation matrix and any multicollinearity that exists between variables. When performing linear regression, we seek to identify variables that strongly correlate with the target variable. The correlation matrix shows that *RM*, the average number of rooms per house, correlates positively with the target variable (r=0.7) and that *LSTAT*, the percent of the lower status of the population, correlates negatively with the target (r=-0.74). Therefore, our linear regression model will focus on these two variables and ignore the remainder since they do not strongly correlate with the target variable.

Figure 11 takes a closer look at the distribution of the *LSTAT* variable. In contrast to the distribution of the *RM* variable, which is normally distributed, the distribution of the *LSTAT* variable appears positively skewed—it has a tail on the right side. Figure 12 plots the *RM* and *LSTAT* variables against the *MEDV* variable. The data in the scatterplot on the left-hand side of the image, showing *LSTAT* against *MEDV*, appears to follow more of a curved distribution than the plot of *RM* against *MEDV*, shown on the figure's right-hand side. Therefore, Chung (2019) writes that we can transform the *LSTAT* variable logarithmically to prevent the model from underfitting.

Underfitting occurs when the model is incapable of capturing the variability of the training data (Jabbar & Khan, 2014). Chung states that by logarithmically transforming the *LSTAT* variable, we minimize the nonlinear relationship in the data and create a more accurate model. Figure 13 compares the *LSTAT* variable with its logarithmically transformed counterpart, *LOGLSTAT*, plotted against *MEDV*. We capture more data by drawing a diagonal line through the right-hand image, which contains the logarithmically transformed variable, rather than the left-hand image containing the untransformed data. Therefore, using this transformed variable to train our linear regression model allows it to capture more of the training data's variability.

Model Training, Evaluation, and Results

The program divides the training features, which contain just the *LOGLSTAT* and *RM* variables, and the target variable, *MEDV*, into 75% and 25% training and testing sets, respectively. The training set contains 379 rows, and the testing set contains 127 rows, as shown by the program's console output in Figure 8. The program then computes the root mean squared error (RMSE) for a baseline regressor that uses the target variable's mean as its prediction and three linear regression models, (a) good old-fashioned (GOF) linear regression, (b) lasso regression, and (c) ridge regression, using 10-fold cross-validation over the training data.

Figure 14 shows the RMSE for each of the models across each of the ten folds. The lower the RMSE, the better the model. The image shows that GOF linear regression ($RMSE = 4.932 \pm 0.85$) and ridge regression ($RMSE = 4.931 \pm 0.86$) performed nearly identically. Therefore, GOF linear regression was chosen to make predictions on the test data. The model achieved an RMSE of 5.05 on the test data, as shown in Figure 8, using the following equation:

$\widehat{Y}_i = -10.35 \times \ln(LSTAT_i) + 3.58 \times RM_i + 24.52$. The program outputs the independent variables and their corresponding coefficients to the console, as shown in Figure 8.

Discussion and Predictions

How can we interpret this output? The RMSE values of 4.93 and 5.05, which the GOF linear regression model achieved on the training and testing sets, respectively, can be interpreted as saying that the model is roughly $5,000 off the actual value, on average. Additionally, in the equation that generates the model's predictions, we can interpret the coefficient of the *LSTAT* variable, the percentage of the lower status of the population, as saying that a 1% increase in *LSTAT decreases* the median value of an owner-occupied home by about $\frac{10.35}{100} = .1035$ or $103.5, holding all other independent variables constant (Chung, 2019; *Interpreting Log Transformations in a Linear Model | University of Virginia Library Research Data Services + Sciences*, n.d.). Further, Chung (2019) writes that we can interpret the coefficient of the *RM* variable, the average number of rooms per home, as saying that for every one-unit increase in a house's average number of rooms, the median value increases by about $3,580, holding all other independent variables constant. Lastly, the y-intercept indicates that the starting price of a house in Boston in 1979 would be around $24,520.

The program's final output, shown in Figure 8, displays ten records from the test data set and their corresponding *LSTAT, RM,* predicted, and true values. One can plug the values from the *LSTAT* and *RM* variables into the equation above to understand how the model generates its predictions. Furthermore, Figures 15 and 16, which plot the *actual vs. predicted values* and the *predicted vs. residual values* of the test dataset, respectively, display these same ten records with their predictive errors highlighted in red. For instance, residual plots indicate what we need to do to fix our predictions (Fenner, 2019). From Figure 16, we can see that the model consistently under-predicts a few records around the $15,000 mark by about $5,000, as can be verified with the program's console output, shown in Figure 8.

Conclusion

In conclusion, this paper described a simple linear regression model in Python, developed using scikit-learn, to predict the median value of an owner-occupied home in the $1,000s from the Boston housing dataset. The paper gave a brief overview of the dataset and used graphical approaches, including heatmaps, to explore the data to determine variables strongly correlated with the target variable. The paper also discussed techniques to logarithmically transform the independent variables to avoid underfitting the model and used graphical output to verify the correctness of these techniques. The best performing linear regression model was selected to create predictions on the test data and achieved an RMSE of 5.05. Finally, the paper discussed the implications of this model and its predictions and used graphical output to support these discussions.

References

Agarwal, A. (2018, October 5). *Linear Regression on Boston Housing Dataset*. Medium. https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155

Chung, C. (2019, August 29). *Used Linear Regression to Model and Predict Housing Prices with The Classic Boston Housing Dataset*. Chris Chung. https://chrispfchung.github.io//model%20data/boston-housing-data/

Fenner, M. (2019). *Machine Learning in Python for Everyone*. Addison-Wesley.

Interpreting Log Transformations in a Linear Model | University of Virginia Library Research Data Services + Sciences. (n.d.). Retrieved September 5, 2021, from https://data.library.virginia.edu/interpreting-log-transformations-in-a-linear-model/

Jabbar, H. K., & Khan, R. Z. (2014). Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study). *Computer Science, Communication, and Instrumentation Devices*, 163–172. https://doi.org/10.3850/978-981-09-5247-1_017