CrossValidation2

April 3, 2023

Cross Validation2

When performing supervised machine learning analysis, it is common to withhold a portion of the data to test the final model's performance. This model testing is performed on the 'unseen' data, which the model was not trained on. This withholding of a portion of the dataset for testing is called Cross-Validation. Cross-Validation can also be used to select hyper-parameters and test the final model. In this section, we will focus on the test data only.

Cross-Validation also helps avoid over-fitting; a complex model could repeat the labels of the samples that it has just seen and, therefore, would have a perfect score but would fail to predict anything useful on the 'unseen' data. Furthermore, a complex model could just be modeling noise.

Cross validation method involves dividing the dataset into 3 parts:

training set - is a portion of the data used for training the model

validation set - is a portion of the data used to optimize the hyper-parameters of the model. This will be illustrated in the next lab

test set - is a portion of the data used to evaluate if the model generalizes enough to work on the data it was not trained on

Scikit Learn library contains many methods that can perform the splitting of the data into training, testing and validation sets. The most popular methods that we will cover now:

train_test_split - creates a single split into train and test sets

K-fold - creates number of k-fold splits, allowing cross validation

cross_val_score - evaluates model's score through cross validation

[1]:
```python
# Surpress warnings:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

[2]:
```python
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pylab as plt
%matplotlib inline
```

```python
from sklearn.preprocessing import StandardScaler, Normalizer
from sklearn.model_selection import train_test_split, cross_val_score, KFold,␣
↪GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
```

[3]:
```python
# load the data

url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBM-ML240EN-SkillsNetwork/labs/encoded_car_data.csv'
data = pd.read_csv(url)
data.head()
```

[3]:
```
   diesel  gas  std  turbo  convertible  hardtop  hatchback  sedan  wagon  \
0     0.0  1.0  1.0    0.0          1.0      0.0        0.0    0.0    0.0
1     0.0  1.0  1.0    0.0          1.0      0.0        0.0    0.0    0.0
2     0.0  1.0  1.0    0.0          0.0      0.0        1.0    0.0    0.0
3     0.0  1.0  1.0    0.0          0.0      0.0        0.0    1.0    0.0
4     0.0  1.0  1.0    0.0          0.0      0.0        0.0    1.0    0.0

   4wd  …  wheelbase  curbweight  enginesize  boreratio  horsepower  \
0  0.0  …       88.6      2548.0       130.0       3.47       111.0
1  0.0  …       88.6      2548.0       130.0       3.47       111.0
2  0.0  …       94.5      2823.0       152.0       2.68       154.0
3  0.0  …       99.8      2337.0       109.0       3.19       102.0
4  1.0  …       99.4      2824.0       136.0       3.19       115.0

   carlength  carwidth  citympg  highwaympg    price
0      168.8      64.1     21.0        27.0  13495.0
1      168.8      64.1     21.0        27.0  16500.0
2      171.2      65.5     19.0        26.0  16500.0
3      176.6      66.2     24.0        30.0  13950.0
4      176.6      66.4     18.0        22.0  17450.0

[5 rows x 36 columns]
```

[4]:
```python
data.dtypes.value_counts()
```

[4]:
```
float64    36
dtype: int64
```

[5]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
```

```
Data columns (total 36 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   diesel        205 non-null    float64
 1   gas           205 non-null    float64
 2   std           205 non-null    float64
 3   turbo         205 non-null    float64
 4   convertible   205 non-null    float64
 5   hardtop       205 non-null    float64
 6   hatchback     205 non-null    float64
 7   sedan         205 non-null    float64
 8   wagon         205 non-null    float64
 9   4wd           205 non-null    float64
 10  fwd           205 non-null    float64
 11  rwd           205 non-null    float64
 12  dohc          205 non-null    float64
 13  dohcv         205 non-null    float64
 14  l             205 non-null    float64
 15  ohc           205 non-null    float64
 16  ohcf          205 non-null    float64
 17  ohcv          205 non-null    float64
 18  rotor         205 non-null    float64
 19  eight         205 non-null    float64
 20  five          205 non-null    float64
 21  four          205 non-null    float64
 22  six           205 non-null    float64
 23  three         205 non-null    float64
 24  twelve        205 non-null    float64
 25  two           205 non-null    float64
 26  wheelbase     205 non-null    float64
 27  curbweight    205 non-null    float64
 28  enginesize    205 non-null    float64
 29  boreratio     205 non-null    float64
 30  horsepower    205 non-null    float64
 31  carlength     205 non-null    float64
 32  carwidth      205 non-null    float64
 33  citympg       205 non-null    float64
 34  highwaympg    205 non-null    float64
 35  price         205 non-null    float64
dtypes: float64(36)
memory usage: 57.8 KB
```

→ As we see from above, we now have only numeric parameters.

[6]:
```python
# split our data into X features and y target.
X = data.drop(columns=['price'])
y = data['price'].copy()
```

[7]:
```python
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,␣
 ↪random_state=42)
# model:
lr = LinearRegression()
# model fit:
lr.fit(X_train, y_train)
# predicteion:
predicted =lr.predict(X_test)
```

Now, let's check some evaluation statistics, such as the coefficient of determination, R^2, using the built-in method `score` or `r2_score`, and the Root Mean Square Error, RMSE, for which we can use the `mean_squared_error` method, MSE.

The R^2 statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively.

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

[8]:
```python
lr.score(X_train,y_train)
```

[8]:
```
0.9261981895971949
```

Using the training data only, the R^2 is ~ 0.93. So, almost 93% of variability in the training data is explained by our model.

[9]:
```python
lr.score(X_test,y_test)
```

[9]:
```
0.8500551011878996
```

[10]:
```python
# using r2_score with the same result:
print(r2_score(y_true=y_test, y_pred=predicted))
```

```
0.8500551011878996
```

Using the test data R^2, we get ~0.85, not as good as the previous score.

Now, let's calculate the RMSE. The smaller the RMSE number the better our model is. We apply `mean_squared_error` to our `y_test` and our predicted data. Then, we take a square root of our MSE, using `np.sqrt()` function.

[11]:
```python
mse = mean_squared_error(y_true=y_test, y_pred=predicted)
rmse = np.sqrt(mse)
rmse
```

[11]:
```
3223.172216951375
```

Prediction Example

Let's select some random data, using `iloc` and see some predicted versus actual values for the car prices.

[12]:
```
some_data = X.iloc[:3]
some_labels = y.iloc[:3]

print("Predictions:", lr.predict(some_data))
print("Labels:", list(some_labels))
```

```
Predictions: [13136.5 13136.5 15331.5]
Labels: [13495.0, 16500.0, 16500.0]
```

[13]:
```
predicted = lr.predict(X_test)
predicted
```

[13]:
```
array([27673.  , 21306.5 , 10731.75, 12098.5 , 26182.5 ,  5422.75,
        8206.75,  7521.25, 10133.25,  9568.25, 17289.25,  7369.25,
       16499.5 , 10778.75, 39805.5 ,  5530.25,  2453.  , 15548.5 ,
       10740.75, 11435.25, 10798.  , 15860.5 , 10356.  ,  3644.75,
        6992.5 , 28662.75, 18291.  , 15265.75,  4490.5 , 16435.25,
       26810.75,  5815.5 ,  4933.75, 16826.5 ,  8276.5 , 30296.75,
       12268.5 , 12934.25,  6289.75, 15975.5 ,  8184.  , 15277.75,
       14764.25,  5273.5 ,  6240.5 ,  9097.  ,  5815.5 ,  7237.25,
       16745.  , 15406.75,  5050.5 , 21925.  ,  7226.75, 11396.  ,
        4428.25, 16473.5 , 18310.  , 12934.25, 29684.25,  5446.25,
        9815.5 , 17210.5 ])
```

We can also use the pipeline to run operations on our data. For example we can standardize our data then perform linear regression by applying the method fit.

[14]:
```
pipe = Pipeline([('ss',StandardScaler() ),('lr', LinearRegression())])
pipe.fit(X_train,y_train)
pipe
```

[14]:
```
Pipeline(steps=[('ss', StandardScaler()), ('lr', LinearRegression())])
```

[15]:
```
# Let's calculate the R squared
pipe.score(X_train,y_train)
```

[15]:
```
0.9262722224294516
```

→ Using the training data only, the R squared is ~ 0.93. Now, let's check the R squared on the test set.

[16]:
```
pipe.score(X_test,y_test)
```

[16]:
```
0.8419174509630335
```

→ The R squared is much lower. This value provides more accurate evaluation of our model since we test our model on the 'unseen' data set. In case if the R squared is negative, it is because the model is too complex and the data is overfitting. For more information, please, visit this documentation on overfitting.

This will make more sense when we explore polynomial regression.

Exercise 1

Create a pipeline object called pipe1, replace standardization with normalization. Calculate the R^2 using the built-in method `score` and for RMSE, using `mean_squared_error` method.

```
[17]: pipe1 = Pipeline([('norm', Normalizer() ),('lr', LinearRegression())])
      pipe1.fit(X_train,y_train)

      # Calculate R2 and RMSE on the test set
      y_pred = pipe1.predict(X_test)
      r2 = pipe1.score(X_test, y_test)
      rmse = mean_squared_error(y_test, y_pred, squared=False)

      print("R2:", r2)
      print("RMSE:", rmse)
```

```
R2: -1.346604622143935e+18
RMSE: 9659123380530.787
```

Note, you can also use `normalize` by setting the `LinearRegression(normalize=True)`:

```
pipe1 = Pipeline([
    ('norm', Normalizer()),
    ('lr', LinearRegression(normalize=True))
])
# rest of the code is the same
```

One feature

We can use the test data to select a feature with the best performance. We have a list of features:

```
[19]: features=list(X)
      print(features, sep=' | ')
```

```
['diesel', 'gas', 'std', 'turbo', 'convertible', 'hardtop', 'hatchback',
'sedan', 'wagon', '4wd', 'fwd', 'rwd', 'dohc', 'dohcv', 'l', 'ohc', 'ohcf',
'ohcv', 'rotor', 'eight', 'five', 'four', 'six', 'three', 'twelve', 'two',
'wheelbase', 'curbweight', 'enginesize', 'boreratio', 'horsepower', 'carlength',
'carwidth', 'citympg', 'highwaympg']
```

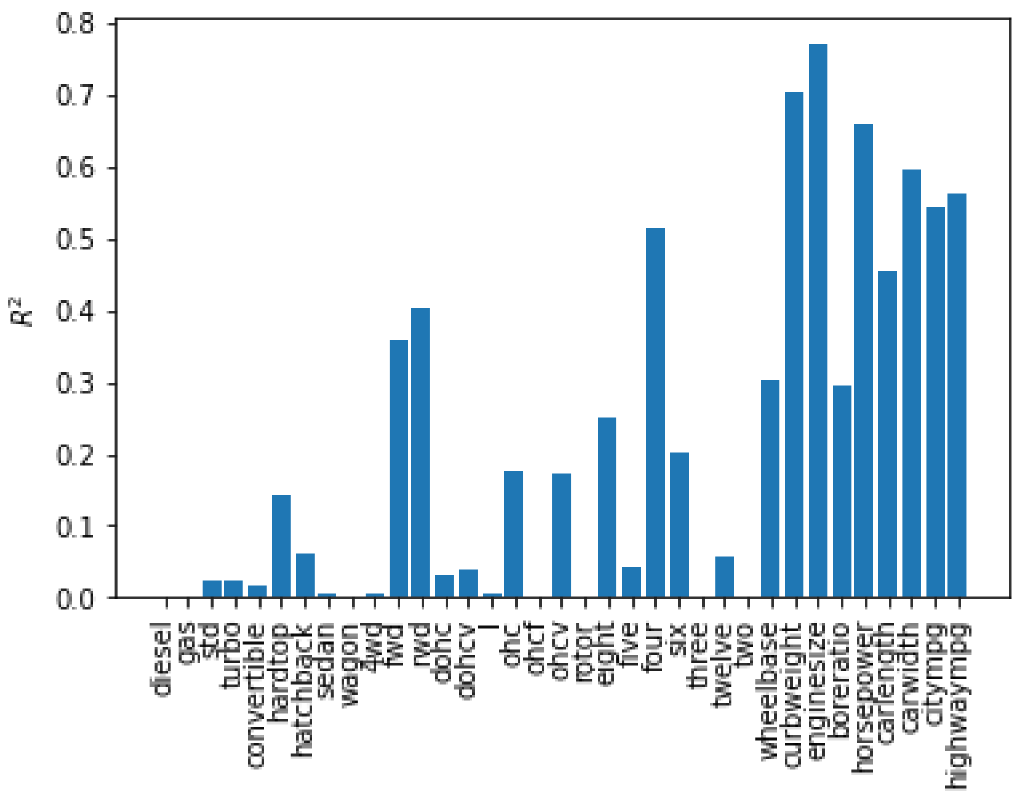
We can train a linear regression model using each feature and use the test data to obtain the best feature.

```
[20]: R_2=[]
      pipe = Pipeline([('ss',StandardScaler() ),('lr', LinearRegression())])

      for feature in features:
```

```
    pipe.fit(X_train[[feature]], y_train)
    R_2.append(pipe.score(X_train[[feature]], y_train))
```

We can plot the R^2 for each feature:

[21]:
```
plt.bar(features,R_2)
plt.xticks(rotation=90)
plt.ylabel("$R^2$")
plt.show()
```



Now, we select the feature that works best, using `argmax()` function.

[22]:
```
best=features[np.argmax(R_2)]
best
```

[22]:
```
'enginesize'
```

So, ‘enginesize’ is the feature that produces the highest R^2. We then train the feature that works best using all the data.

[23]:
```
pipe.fit(X[[best]],y)
```

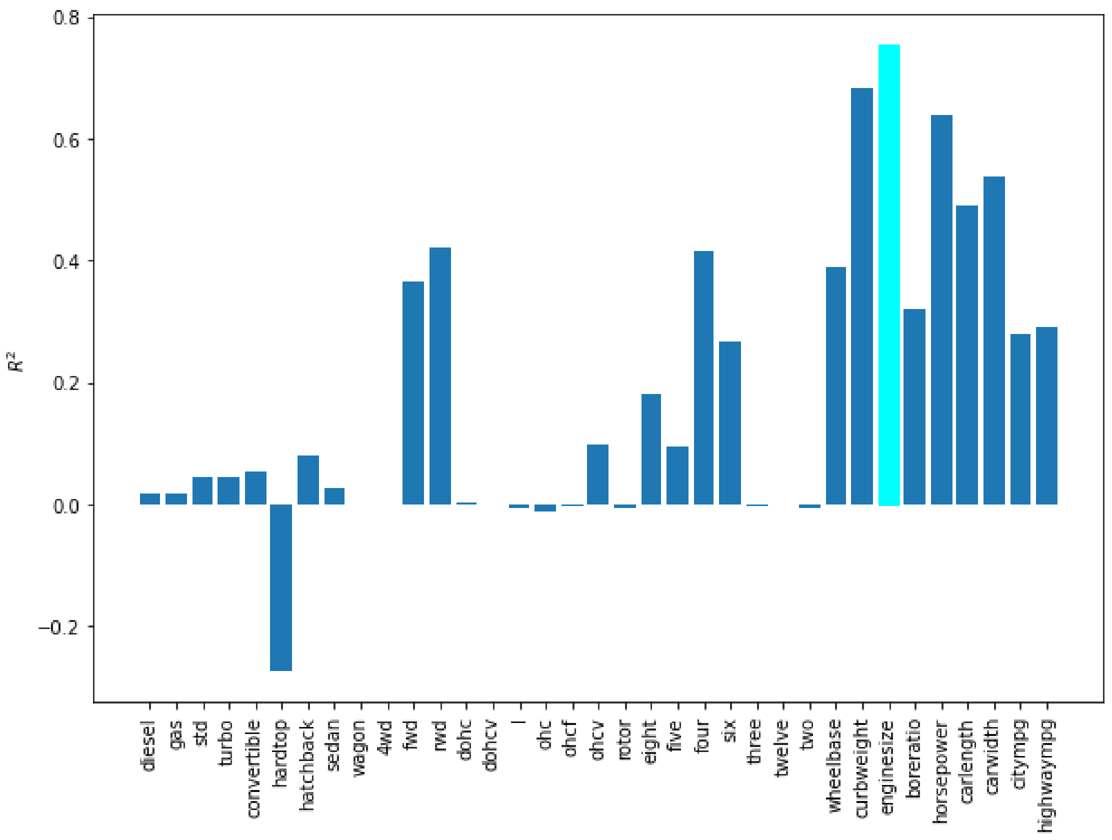
[23]:
```
Pipeline(steps=[('ss', StandardScaler()), ('lr', LinearRegression())])
```

Exercise 2

Find the best feature using the test data, without standardization.

[37]:
```python
R_2=[]

for feature in features:
    lr.fit(X_train[[feature]], y_train)
    R_2.append(lr.score(X_test[[feature]],y_test))

best = features[np.argmax(R_2)]
print("\nBest feature using test data w/o standardization:\t", best, "\n")

best_index = np.argmax(R_2)

fig, ax = plt.subplots(figsize=(10, 7))
bars = ax.bar(features, R_2)
plt.xticks(rotation=90)
plt.ylabel("$R^2$")

# Conditional formatting for best bar
for i in range(len(bars)):
    if R_2[i] == R_2[best_index]:
        bars[i].set_color('cyan')

plt.show()
```

```
Best feature using test data w/o standardization:        enginesize
```



K Fold Cross Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=5 becoming 5-fold cross-validation, as shown in the Diagram below. In this case, we would use K-1 (or 4 folds) for testing a 1 fold for training. K-fold is also used for hyper-parameters selection that we will discuss later.

Cross Validation Score

Now, let's use *Scikit-Learn's K-fold cross-validation* method to see whether we can assess the performance of our model. The *K-fold cross-validation* method splits the training set into the number of folds (n_splits), as now in the Diagram above, if we have K folds, K-1 is used for training and one fold is used for testing. The input parameters are as follows:

estimatorestimator: The object to use to `fit` the data. X: array-like of shape (n_samples, n_features). The data to fit. Can be for example a list, or an array. y: array-like of shape (n_samples,) or (n_samples, n_outputs), default=None. The target variable to try to predict in the case of supervised learning. scoring: A str or a scorer callable object/ function with signature scorer (estimator, X, y) which should return only a single value. See model evaluation

```
[38]: N=len(X)
      N
```

```
[38]: 205
```

```
[39]: # create a Linear Regression object
      lr = LinearRegression()
      # calculate cross validation scores based on our testing sets
      scores = cross_val_score(lr, X, y, scoring ="r2", cv=3)
      scores
```

```
[39]: array([ 0.81508001,  0.7238302 , -0.0913317 ])
```

We can calculate mean and standard deviation using the following function of the `scores`:

```
[40]: def display_scores(scores, print_=False):

          print("Scores:", scores)
          print("Mean:", scores.mean())
          print("Standard deviation:", scores.std())

      display_scores(scores)
```

```
Scores: [ 0.81508001  0.7238302  -0.0913317 ]
Mean: 0.4825261694103302
Standard deviation: 0.40748519272512795
```

The larger the fold, the better the model performance is, as we are using more samples for training; the variance also decreases. Cross Validation Scores are RMSE values for training the data on each of our folds, in our case cv = 3, so we get 3 scores, 1 for each fold.

Exercise 3

Compute the cross validation scores for 5 folds, using the linear regression object `lr` and `neg_mean_squared_error` method for scoring.

```
[44]: lr = LinearRegression()
      scores5 = cross_val_score(lr, X, y, scoring='neg_mean_squared_error', cv=5)
      scores5 = np.sqrt(-scores5)

      display_scores(scores5)
```

```
Scores: [ 3155.3045326  11345.67407101  3025.79039538  5629.2248323
  5549.16298134]
Mean: 5741.031362524296
Standard deviation: 3017.2786021293505
```

K Fold

In many cases, we would like to train models that are not available in Scikit-learn or are too large to fit in the memory. We can create a `KFold` object that Provides train/test indices to split data into train/test sets in an iterative manner.

`n_splitsint`: A number of folds. Must be at least 2. Changed in version 0.22: n_splits default value changed from 3 to 5. `shuffle`: Indicates whether to shuffle the data before splitting into batches. Note, the samples within each split will not be shuffled. `random_state`: the random state.

```
[45]: # We create the  `KFold` object `kf`, setting the number of splits to 2.
n_splits=2
kf = KFold(n_splits = n_splits)
```

We train the model using the `split(X,y)` method. It provides the train/test indices for `X` and `y`. Half the data is used for training in the first iteration, and the rest is used for testing and displaying the indexes for each set. For the second iteration, the data used for training is used for testing, and the testing data is used for training. We store the R^2 for each iteration in the array `R_2`. The `np.zeros()` function returns a new array of given shape and type, filled with zeros. Then, we calculate the R^2 for each of the X_train and X_test splits.

```
[46]: y = data['price'].copy()
X = data.drop(columns=['price'])
R_2 = np.zeros((n_splits,1))
pipe = Pipeline([('ss',StandardScaler() ),('lr', LinearRegression())])
n = 0

for k,(train_index, test_index) in enumerate(kf.split(X,y)):
    print("TRAIN:", train_index)
    print("TEST:", test_index)
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]

    y_train, y_test = y[train_index], y[test_index]
    pipe.fit(X_train, y_train)
    n += 1
    R_2[k]=pipe.score(X_test, y_test)
```

```
TRAIN: [103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138
 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156
 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174
 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
 193 194 195 196 197 198 199 200 201 202 203 204]
TEST: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102]
TRAIN: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102]
TEST: [103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138
 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156
 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174
 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
 193 194 195 196 197 198 199 200 201 202 203 204]
```

We can calculate the average R^2:

[47]:
```
R_2.mean()
```

[47]:
```
0.39256530180315163
```

If we set the number of splits to three, we see 2/3's of the data is used for training.

[48]:
```
n_splits = 3
kf = KFold(n_splits = n_splits)
y = data['price'].copy()
X = data.drop(columns=['price'])
R_2 = np.zeros((n_splits, 1))
pipe = Pipeline([('ss',StandardScaler() ),('lr', LinearRegression())])
n = 0
for k,(train_index, test_index) in enumerate(kf.split(X,y)):
    print("TRAIN:", train_index)
    print("TEST:", test_index)
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]

    y_train, y_test = y[train_index], y[test_index]
    pipe.fit(X_train, y_train)
    n += 1
    R_2[k] = pipe.score(X_test, y_test)


R_2.mean()
```

```
TRAIN: [ 69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86
  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104
 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158
 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176
 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194
 195 196 197 198 199 200 201 202 203 204]
```

```
TEST: [ 0   1   2   3   4   5   6   7   8   9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68]
TRAIN: [   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 137 138 139
 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193
 194 195 196 197 198 199 200 201 202 203 204]
TEST: [ 69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86
  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104
 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 123 124 125 126 127 128 129 130 131 132 133 134 135 136]
TRAIN: [   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136]
TEST: [137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154
 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172
 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190
 191 192 193 194 195 196 197 198 199 200 201 202 203 204]
```

[48]:
```
0.47719341451693237
```

Exercise 4

It many applications, it is useful to randomly select samples for K fold cross validation. In this Exercise, randomly select samples by setting `shuffle` to `True` in the `KFold` constructor. Use all the parameters, as above.

[49]:
```python
n_splits = 3
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
y = data['price'].copy()
X = data.drop(columns=['price'])
R_2 = np.zeros((n_splits, 1))
pipe = Pipeline([('ss', StandardScaler()), ('lr', LinearRegression())])
n = 0
for k, (train_index, test_index) in enumerate(kf.split(X, y)):
    print("TRAIN:", train_index)
    print("TEST:", test_index)
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    pipe.fit(X_train, y_train)
    n += 1
    R_2[k] = pipe.score(X_test, y_test)

R_2.mean()
```

```
TRAIN: [  0   1   2   3   4   6   7   8  10  11  13  14  17  20  21  22  23  26
  27  28  32  33  34  36  37  39  40  41  42  43  44  46  47  48  49  50
  51  52  53  54  57  58  59  61  62  63  64  70  71  72  74  77  78  80
  81  83  85  87  88  89  91  92  94  99 102 103 105 106 107 108 109 110
 113 114 115 116 117 118 119 121 122 123 124 126 127 129 130 131 133 134
 135 136 137 138 140 141 142 148 149 150 151 155 156 157 158 160 161 163
 165 166 169 170 171 172 176 179 180 181 183 184 185 186 187 188 189 190
 192 193 196 197 198 199 200 201 203 204]
TEST: [  5   9  12  15  16  18  19  24  25  29  30  31  35  38  45  55  56  60
  65  66  67  68  69  73  75  76  79  82  84  86  90  93  95  96  97  98
 100 101 104 111 112 120 125 128 132 139 143 144 145 146 147 152 153 154
 159 162 164 167 168 173 174 175 177 178 182 191 194 195 202]
TRAIN: [  1   3   5   7   8   9  12  13  14  15  16  17  18  19  20  21  24  25
  29  30  31  34  35  37  38  43  45  48  49  50  52  53  54  55  56  57
  58  59  60  63  65  66  67  68  69  71  72  73  74  75  76  79  80  82
  83  84  86  87  88  89  90  91  92  93  95  96  97  98  99 100 101 102
 103 104 105 106 107 110 111 112 116 120 121 125 128 129 130 131 132 133
 134 139 142 143 144 145 146 147 149 151 152 153 154 157 158 159 160 161
 162 164 167 168 169 171 173 174 175 176 177 178 179 180 181 182 185 186
 188 190 191 193 194 195 197 198 201 202 204]
TEST: [  0   2   4   6  10  11  22  23  26  27  28  32  33  36  39  40  41  42
  44  46  47  51  61  62  64  70  77  78  81  85  94 108 109 113 114 115
 117 118 119 122 123 124 126 127 135 136 137 138 140 141 148 150 155 156
 163 165 166 170 172 183 184 187 189 192 196 199 200 203]
TRAIN: [  0   2   4   5   6   9  10  11  12  15  16  18  19  22  23  24  25  26
  27  28  29  30  31  32  33  35  36  38  39  40  41  42  44  45  46  47
  51  55  56  60  61  62  64  65  66  67  68  69  70  73  75  76  77  78
  79  81  82  84  85  86  90  93  94  95  96  97  98 100 101 104 108 109
 111 112 113 114 115 117 118 119 120 122 123 124 125 126 127 128 132 135
 136 137 138 139 140 141 143 144 145 146 147 148 150 152 153 154 155 156
 159 162 163 164 165 166 167 168 170 172 173 174 175 177 178 182 183 184
 187 189 191 192 194 195 196 199 200 202 203]
TEST: [  1   3   7   8  13  14  17  20  21  34  37  43  48  49  50  52  53  54
  57  58  59  63  71  72  74  80  83  87  88  89  91  92  99 102 103 105
 106 107 110 116 121 129 130 131 133 134 142 149 151 157 158 160 161 169
 171 176 179 180 181 185 186 188 190 193 197 198 201 204]
```

[49]: 0.6904823744622521

[ ]: