M4L4 Solutions

CQF January 2024

Exercise 1

For the Boston housing dataset, construct a dataframe with all features and target values. Perform feature selection to choose the most appropriate features using

1. Variance Inflation Factor
2. SelectKBest
3. Recursive Feature Elimination
4. Recursive Feature Elimination Cross validation
5. Shapley Additive Explanations

Fit the regressor and compare the results. How much better does the model perform? Use Scikit-learn package to perform this task.

Solution

Features Selection

Feature selection methods are approaches to reduce the number of input variables that are believed to be most useful to a model. It is primarily focused on removing non-informative or redundant predictors from the model. We'll focus on few methods in relation to linear regression.

[1]:
```
# Data Manipulation
import pandas as pd
import numpy as np

# Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Regressor
from sklearn.linear_model import LinearRegression

# Import SHAP
import shap
```

[2]:
```
# Load boston dataset
df = pd.read_csv('https://raw.githubusercontent.com/kannansingaravelu/datasets/
↪main/boston.csv')
df.columns = df.columns.str.upper()
```

```python
# Display dataframe
display(df.head(3))

# Features
X = df.drop('MEDV', axis=1)
# Label
y = df['MEDV']
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8

        B  LSTAT  MEDV
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
```

[3]:
```python
# Feature scaling
scaler = StandardScaler()

# Pipeline
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression())
])
```

[4]:
```python
# Original model
pipe.fit(X, y)

# predict labels
y_pred = pipe.predict(X)

print(y_pred[:10])
print(f'R^2: {pipe.score(X, y):0.4}')
```

```
[30.00384338 25.02556238 30.56759672 28.60703649 27.94352423 25.25628446
 23.00180827 19.53598843 11.52363685 18.92026211]
R^2: 0.7406
```

Method 1: VIF

Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. This means that an independent variable can be predicted from another independent variable in a regression model.

Multicollinearity can be detected using various methods and one such method is Variable Inflation Factors (VIF). VIF determines the strength of the correlation between the independent variables.

It is predicted by taking a variable and regressing it against every other variable.

VIF score of an independent variable represents how well the variable is explained by other independent variables.

R^2 value is determined to find out how well an independent variable is described by the other independent variables. A high value of R^2 means that the variable is highly correlated with the other variables. This is captured by the VIF which is denoted below:

$$VIF = \frac{1}{1 - R^2}$$

- VIF starts at 1 and has no upper limit
- VIF = 1, no correlation between the independent variable and the other variables
- VIF exceeding 5 or 10 indicates high multicollinearity between this independent variable and the others

[5]:
```
# Import VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[6]:
```
# For each X, calculate VIF and save in dataframe
def vif(X):

    # perform feature scaling
    xs = scaler.fit_transform(X)

    # subsume into a dataframe
    vif = pd.DataFrame()
    vif["Features"] = X.columns
    vif["VIF Factor"] = [variance_inflation_factor(xs, i) for i in range(xs.
 ↪shape[1])]

    return vif
```

[7]:
```
# List scores
vif(X).round(2)
```

[7]:

	Features	VIF Factor
0	CRIM	1.79
1	ZN	2.30
2	INDUS	3.99
3	CHAS	1.07
4	NOX	4.39
5	RM	1.93
6	AGE	3.10
7	DIS	3.96
8	RAD	7.48
9	TAX	9.01
10	PTRATIO	1.80

```
11         B         1.35
12     LSTAT         2.94
```

[8]:
```
# Drop VIF score > 5
newX = X.drop(['TAX', 'RAD'],axis=1)
```

[9]:
```
# Scores in ascending values
vif(newX).sort_values(by="VIF Factor")
```

[9]:

	Features	VIF Factor
3	CHAS	1.057805
9	B	1.316559
0	CRIM	1.478206
8	PTRATIO	1.496077
5	RM	1.872532
1	ZN	2.154483
10	LSTAT	2.936487
6	AGE	3.075755
2	INDUS	3.179166
4	NOX	3.901348
7	DIS	3.954443

[10]:
```
# Filter first six features
X_method1 = X[['CHAS', 'B', 'CRIM', 'PTRATIO', 'RM', 'ZN']]
```

[11]:
```
# fit/train model
pipe.fit(X_method1, y)

# predict labels
y_pred = pipe.predict(X_method1)

print(y_pred[:10])
print(f'R^2: {pipe.score(X_method1, y):0.4}')
```

```
[28.57799551 24.78825544 30.16715122 28.03678666 29.13085894 23.98038835
 24.50415657 25.65810523 21.62521274 24.29070206]
R^2: 0.6273
```

Method 2: SelectKBest

Select features according to the k highest scores. Univariate feature selection works by selecting the best features based on univariate statistical tests.

[12]:
```
# Feature Selection
from sklearn.feature_selection import f_regression, SelectKBest,␣
 ↪SelectPercentile
```

[13]:
```
# SelectKBest
method2 =  SelectKBest(f_regression, k=6)
# selector1 = SelectPercentile(f_regression, percentile=25)

# Fit the model
method2.fit(X,y)
```

[13]:
```
SelectKBest(k=6, score_func=<function f_regression at 0x7f8dd8fa4310>)
```

[14]:
```
# Show selected features
method2.get_support(indices=True)
```

[14]:
```
array([ 2,  4,  5,  9, 10, 12])
```

[15]:
```
# Iterate the score
for f, s in zip(X.columns, method2.scores_):
    print(f'F-score: {s:0.4} for feature {f}')
```

```
F-score: 89.49 for feature CRIM
F-score: 75.26 for feature ZN
F-score: 154.0 for feature INDUS
F-score: 15.97 for feature CHAS
F-score: 112.6 for feature NOX
F-score: 471.8 for feature RM
F-score: 83.48 for feature AGE
F-score: 33.58 for feature DIS
F-score: 85.91 for feature RAD
F-score: 141.8 for feature TAX
F-score: 175.1 for feature PTRATIO
F-score: 63.05 for feature B
F-score: 601.6 for feature LSTAT
```

[16]:
```
# Filter six features with highest score
X_method2 = X[['INDUS', 'NOX', 'RM', 'TAX', 'PTRATIO', 'LSTAT']]
```

[17]:
```
# fit/train model
pipe.fit(X_method2, y)

# predict labels
y_pred = pipe.predict(X_method2)

print(y_pred[:10])
print(f'R^2: {pipe.score(X_method2, y):0.4}')
```

```
[30.57014999 26.10398705 32.45084162 31.0219246  30.41087418 27.1380861
 24.46192059 21.5381886  13.13415681 21.87515864]
R^2: 0.681
```

[18]:
```
# check the coefficients
pipe['regressor'].coef_
```

[18]:
```
array([ 0.59754458, -0.39395547,  3.26810054, -0.48846049, -1.9764135 ,
       -3.89469824])
```

Method 3: RFE

Feature ranking with recursive feature elimination (RFE). The goal is to select features by recursively considering smaller and smaller sets of features.

[19]:
```
# Feature Selection using RFE
from sklearn.feature_selection import RFECV, RFE
```

[20]:
```
# Method 3
method3 = RFE(LinearRegression(), n_features_to_select=6, step=1)
method3.fit(X,y)
```

[20]:
```
RFE(estimator=LinearRegression(), n_features_to_select=6)
```

[21]:
```
# Check the selected position
method3.support_
```

[21]:
```
array([False, False, False,  True,  True,  True, False,  True, False,
       False,  True, False,  True])
```

[22]:
```
# Get the feature ranking
method3.ranking_
```

[22]:
```
array([3, 5, 4, 1, 1, 1, 8, 1, 2, 6, 1, 7, 1])
```

[23]:
```
# Select Six Features
min_value = min(method3.ranking_)
col = [i for i, x in enumerate(method3.ranking_) if x == min_value]
col
```

[23]:
```
[3, 4, 5, 7, 10, 12]
```

[24]:
```
# Filter selected features
X_method3 = X[['CHAS', 'NOX', 'RM', 'DIS', 'PTRATIO', 'LSTAT']]
```

[25]:
```
# fit/train model
pipe.fit(X_method3, y)

# predict labels
y_pred = pipe.predict(X_method3)

print(y_pred[:10])
```

```
print(f'R^2: {pipe.score(X_method3, y):0.4}')
```

```
[31.0142456  25.79278275 31.84611084 29.74856961 28.99930186 26.11951419
 23.13348366 19.51563196 10.99759368 19.25866209]
R^2: 0.7158
```

[26]:
```
# check the coefficients
pipe['regressor'].coef_
```

[26]:
```
array([ 0.82321941, -2.16945087,  2.88617319, -2.40778286, -2.16874483,
       -4.06526959])
```

Method 4: RFECV

A recursive feature elimination with automatic tuning of the number of features selected with cross-validation.

[27]:
```
# Method 4
method4 = RFECV(LinearRegression(), cv=10)
method4.fit(X,y)
```

[27]:
```
RFECV(cv=10, estimator=LinearRegression())
```

[28]:
```
# Get the selected features with CV
method4.n_features_
```

[28]:
```
6
```

[29]:
```
# Get the index of the selected features
method4.get_support(indices=True)
```

[29]:
```
array([ 3,  4,  5,  7, 10, 12])
```

[30]:
```
# Check the selected position
method4.support_
```

[30]:
```
array([False, False, False,  True,  True,  True, False,  True, False,
       False,  True, False,  True])
```

[31]:
```
# Get the feature ranking
method4.ranking_
```

[31]:
```
array([3, 5, 4, 1, 1, 1, 8, 1, 2, 6, 1, 7, 1])
```

[32]:
```
# Select Six Features
min_value = min(method4.ranking_)
col = [i for i, x in enumerate(method4.ranking_) if x == min_value]
col
```

[32]: [3, 4, 5, 7, 10, 12]

[33]:
```
# Iterate to get features
for i in range(len(col)):
    print(X.columns[col[i]])
```

```
CHAS
NOX
RM
DIS
PTRATIO
LSTAT
```

[34]:
```
# Filter selected features
X_method4 = X[['CHAS', 'NOX', 'RM', 'DIS', 'PTRATIO', 'LSTAT']]
```

[35]:
```
# fit/train model
pipe.fit(X_method4, y)

# predict labels
y_pred = pipe.predict(X_method4)

print(y_pred[:10])
print(f'R^2: {pipe.score(X_method4, y):0.4}')
```

```
[31.0142456  25.79278275 31.84611084 29.74856961 28.99930186 26.11951419
 23.13348366 19.51563196 10.99759368 19.25866209]
R^2: 0.7158
```

[36]:
```
# check the coefficients
pipe['regressor'].coef_
```

[36]:
```
array([ 0.82321941, -2.16945087,  2.88617319, -2.40778286, -2.16874483,
       -4.06526959])
```

Method 5: SHAP

SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions.

Shapley values are a widely used approach from cooperative game theory that come with desirable properties and is the average marginal contribution of a feature value across all possible coalitions.

[37]:
```
# 100 instances for use as the background distribution
X100 = shap.utils.sample(X, 100, random_state=42)
pipe.fit(X, y)
```

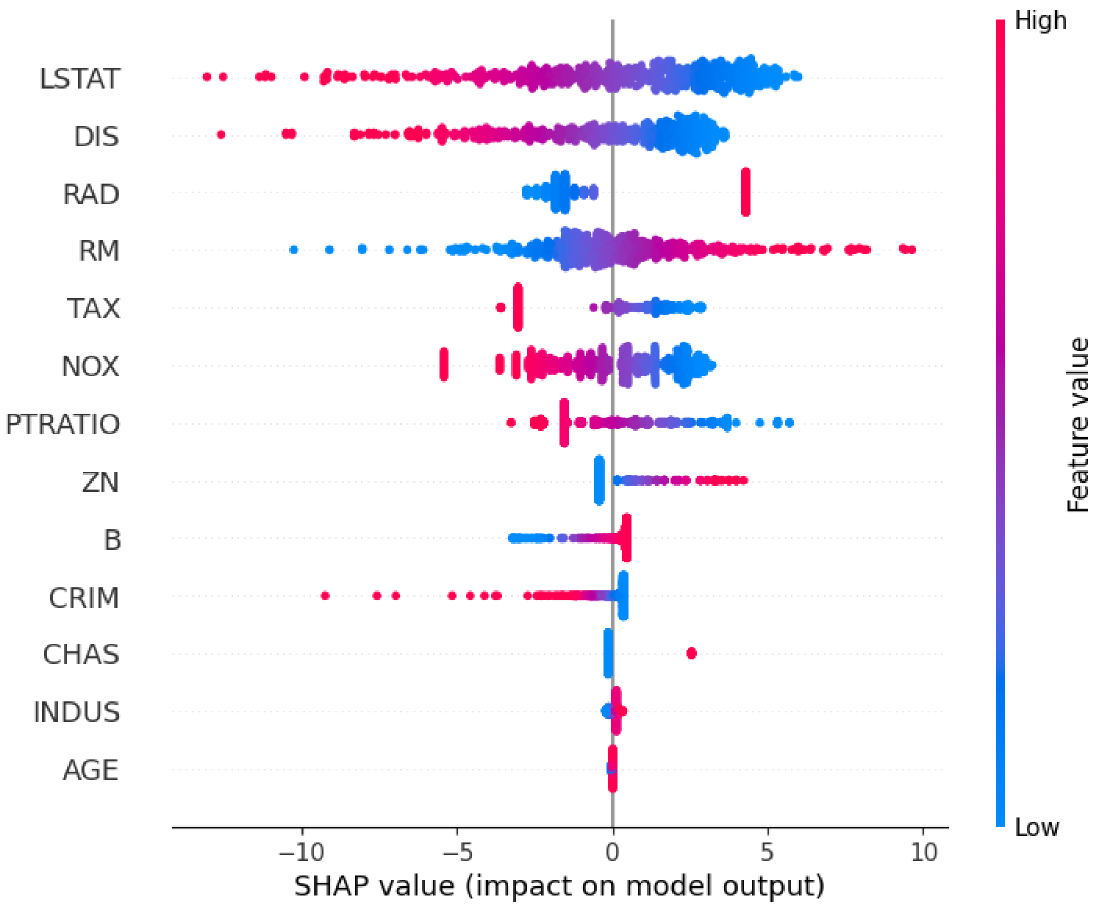
```
[37]: Pipeline(steps=[('scaler', StandardScaler()),
                      ('regressor', LinearRegression())])
```

```
[38]: # compute the SHAP values for the linear model
      explainer = shap.Explainer(pipe.predict, X100)
      shap_values = explainer(X)
```

```
Permutation explainer: 507it [00:19, 14.41it/s]
```

```
[39]: shap.plots.beeswarm(shap_values, max_display=20)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be
ignored
```



The above plot shows the feature importance of a linear model where the variables are ranked in descending order and the horizontal location shows whether the effect of that value is associated with a higher or lower prediction.

- Color shows whether that variable is high (in red) or low (in blue) for that observation.
- A high level has a high and positive impact on the quality rating.

- Impact is shown on the X-axis.

[40]:
```
# Filter selected features
X_method5 = X[['LSTAT', 'DIS', 'RM', 'PTRATIO', 'CRIM']]
```

[41]:
```
# fit/train model
pipe.fit(X_method5, y)

# predict labels

y_pred = pipe.predict(X_method5)

print(y_pred[:10])
print(f'R^2: {pipe.score(X_method5, y):0.4}')
```

```
[31.63761273 25.5236182  32.0243847  30.38600726 29.5358719  26.50795486
 23.71933951 19.96956586 10.8068779  20.1215183 ]
R^2: 0.6958
```

Comparision

Given the boston housing dataset is a processed data, there seems to be no improvement in the score on feature reduction.

Feature Selection Method	R-Square	Improvement
Original Features set	0.7406	——
VIF	0.6273	-0.15298
SelectKBest	0.6810	-0.08047
RFE	0.7158	-0.03348
RFECV	0.7158	-0.03348
SHAP	0.6958	-0.06049

Alternatively, one might split the data into train and test sets to study the impact of over or under fitting of the model. The Recursive Feature Elimination is closer to the original scoring with just six features while the impact of each of these features is explained by the SHAP values.

Exercise 2

Create a custom transformer that replaces outlier values of 1, 5, 20, 60 and 120 days SPX percentage returns. Determine the lower and upper bound of acceptable values based on the -th percentile. Compare the result with the original feature set. You can use the SPX dataset used in the Python Labs.

Solution

Custom Estimators

Scikit-learn provides dozens of machine learning models and transformers. However, our workflow sometimes requires us to specify the models or transformations; and such models or transformations should have the `fit`, and either `predict` or `transform` methods that are in compliance with `scikit-learn` so that we can leverage its functionalities such as Pipeline, GridSearchCV classes and such other features.

This is achieved with custom estimators. Scikit-learn has a base class called `BaseEstimator` that all estimators inherit and these models inherit additional classes such as `RegressorMixin`, `ClassifierMixin`, and `TransformerMixin`. We can thus customize our models by inheriting these classes that are in compliance with scikit-learn.

Transformers are estimators which implement a `transform` method. Regressors are estimators that implement a `predict` method while classifiers implement `predict` method inaddition to the probability output of the predictions using the `predict_proba` method. For this exercise, we will limit our discussion to transformers.

[42]:
```
# Import Base and Transformer classes
from sklearn.base import BaseEstimator, TransformerMixin
```

We will now specify the percentiles for the lower and upper bound and define our `fit` method that calculates values required to transform the outlier values.

[43]:
```
# Create custom transformer class by inheriting base and additional classes

class OutlierTransform(BaseEstimator, TransformerMixin):

    def __init__(self, q_lower, q_upper):
        self.q_lower = q_lower
        self.q_upper = q_upper

    def fit(self, X, y=None):
        self.lower = np.percentile(X, self.q_lower, axis=0)
        self.upper = np.percentile(X, self.q_upper, axis=0)

        return self

    def transform(self, X):
        Xt = X.copy()
        idx_lower = X < self.lower
        idx_upper = X > self.upper

        for i in range(X.shape[1]):
            Xt[idx_lower[:,i], i] = self.lower[i]
            Xt[idx_upper[:,i], i] = self.upper[i]

        return Xt
```

Fit method always returns self, which is a copy of the fitted estimator. If self is not returned, this will not be fully compatibale with scikit-learn and will not work with the pipelines.

[44]:
```python
# Load the CSV file
spx = pd.read_excel('./data/SP500.xlsx', index_col=0, parse_dates=True)['2015':]

# Calculate returns and add it to existing DataFrame as a column
rdict = {str(i)+'D_RET': spx['Adj Close'].pct_change(i) for i in␣
↪[1,5,20,60,120]}

# Convert to dataframe
rdf = pd.DataFrame(rdict).dropna()

# Check the output
rdf.head(2)
```

[44]:
```
              1D_RET    5D_RET   20D_RET   60D_RET  120D_RET
Date
2015-06-25 -0.002974 -0.008924 -0.008714  0.016645  0.021431
2015-06-26 -0.000390 -0.004028 -0.002800  0.020294  0.040043
```

[45]:
```python
# Convert to numpy array
Xnew = rdf.values
Xnew
```

[45]:
```
array([[-0.00297357, -0.00892399, -0.00871372,  0.01664507,  0.0214314 ],
       [-0.00039008, -0.00402846, -0.00279963,  0.02029434,  0.04004298],
       [-0.02086619, -0.03071823, -0.02561411, -0.00450907,  0.02747909],
       …,
       [ 0.00227564,  0.01646155,  0.04451009,  0.09358433,  0.17745381],
       [-0.00375345, -0.00216889,  0.02731537,  0.10756601,  0.18034108],
       [-0.00719003, -0.01475416,  0.01812402,  0.09442885,  0.16325198]])
```

[46]:
```python
# Descriptive Statistics
rdf.describe()
```

[46]:
```
            1D_RET       5D_RET      20D_RET      60D_RET     120D_RET
count  1401.000000  1401.000000  1401.000000  1401.000000  1401.000000
mean      0.000487     0.002374     0.009416     0.026044     0.048571
std       0.012025     0.023865     0.046613     0.071240     0.082316
min      -0.119841    -0.179666    -0.309439    -0.305884    -0.248372
25%      -0.003038    -0.006079    -0.008064    -0.005201    -0.005474
50%       0.000648     0.004295     0.015117     0.034519     0.047833
75%       0.005297     0.014062     0.032556     0.063961     0.092458
max       0.093828     0.173974     0.224841     0.391566     0.493238
```

[47]:
```python
# create, fit and transform for 5 and 95 percentile
data_transform = OutlierTransform(5,95)
data_transform.fit(Xnew)
```

[47]:
```
OutlierTransform(q_lower=5, q_upper=95)
```

[48]:
```python
Xt = data_transform.transform(Xnew)
Xt
```

[48]:
```
array([[-0.00297357, -0.00892399, -0.00871372,  0.01664507,  0.0214314 ],
       [-0.00039008, -0.00402846, -0.00279963,  0.02029434,  0.04004298],
       [-0.0175848 , -0.03071823, -0.02561411, -0.00450907,  0.02747909],
       …,
       [ 0.00227564,  0.01646155,  0.04451009,  0.09358433,  0.17745381],
       [-0.00375345, -0.00216889,  0.02731537,  0.10756601,  0.18034108],
       [-0.00719003, -0.01475416,  0.01812402,  0.09442885,  0.16325198]])
```

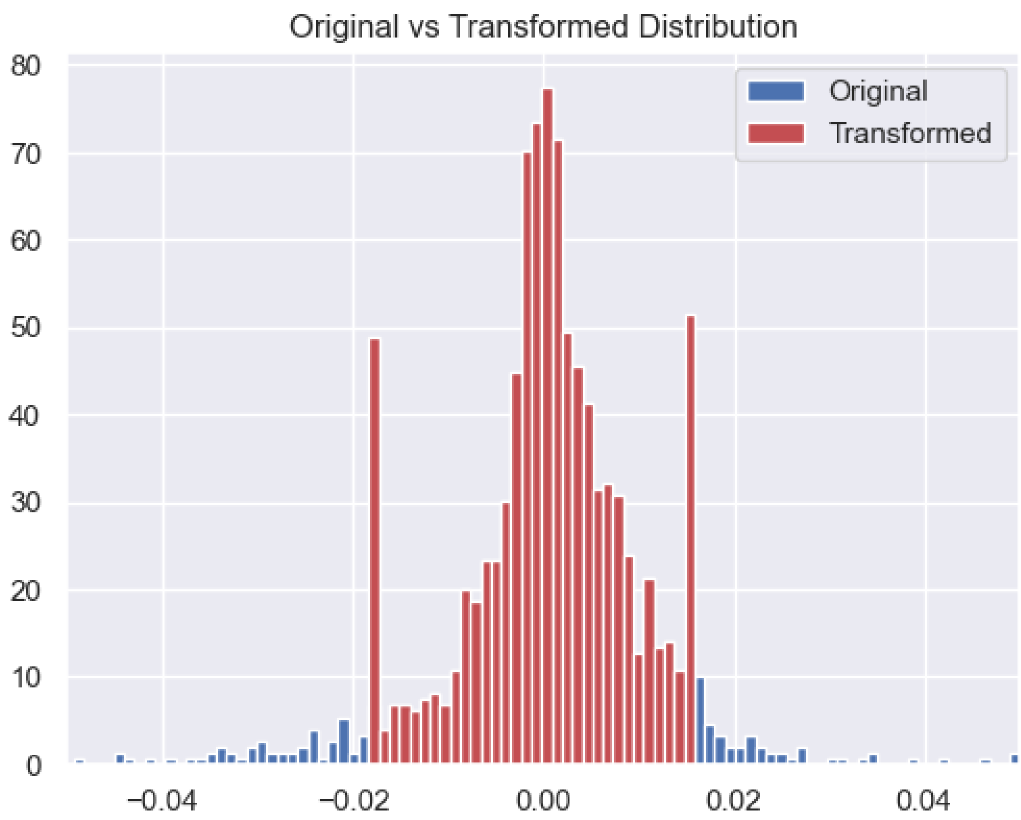
[49]:
```python
# Transformed Data
pd.DataFrame(Xt).describe()
```

[49]:

	0	1	2	3	4
count	1401.000000	1401.000000	1401.000000	1401.000000	1401.000000
mean	0.000586	0.002731	0.010536	0.026902	0.046951
std	0.007936	0.017047	0.032996	0.056641	0.070087
min	-0.017585	-0.037724	-0.062231	-0.095876	-0.080179
25%	-0.003038	-0.006079	-0.008064	-0.005201	-0.005474
50%	0.000648	0.004295	0.015117	0.034519	0.047833
75%	0.005297	0.014062	0.032556	0.063961	0.092458
max	0.015132	0.032301	0.065318	0.123099	0.182247

[50]:
```python
# Import matplotlib for visualization
import matplotlib
import matplotlib.pyplot as plt

# Plot settings
import seaborn as sns
sns.set()
```

[51]:
```python
# Plot historgram of 1day returns
_, bins, _ = plt.hist(Xnew[:,0], density=True, bins=200, alpha=1, color='b',␣
 ↪label = 'Original')
plt.hist(Xt[:,0], density=True, bins=bins, alpha=1, color='r', label =␣
 ↪'Transformed')

# Set title
plt.title('Original vs Transformed Distribution')

# Set x and y axis limits
plt.xlim(-0.05, 0.05)

# Set legends
plt.legend();
```



References

- Scikit-learn features selection
- SHAP documentation
- Scikit-learn classes reference
- Python resources

* * *