Assignment 4

Question NO. 1 Using sklearn.datasets.load_diabetes apply Variance method for removing the constant column also after applying the Variance method apply multi linear regression on that data

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score

diab =load_diabetes()

X = diab.data
y = diab.target

diab.feature_names

['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']

diab_data = pd.DataFrame(diab.data,columns=diab.feature_names)
diab_data.head()

        age       sex       bmi        bp        s1        s2
s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -
0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163
0.074412
2  0.085299  0.050680  0.044451 -0.005671 -0.045599 -0.034194 -
0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -
0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596
0.008142

         s4        s5        s6
0 -0.002592  0.019908 -0.017646
1 -0.039493 -0.068330 -0.092204
2 -0.002592  0.002864 -0.025930
3  0.034309  0.022692 -0.009362
4 -0.002592 -0.031991 -0.046641

varT = VarianceThreshold()

X_filt = varT.fit_transform(X)
```

```
X_filt
```

```
array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
         0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
        -0.06832974, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
         0.00286377, -0.02593034],
       ...,
       [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
        -0.04687948,  0.01549073],
       [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
         0.04452837, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
        -0.00421986,  0.00306441]])
```

```
op = varT.fit(X)
op.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

```
X_train,X_test,y_train,y_test = train_test_split(X_filt,y, test_size
=0.25,random_state = 17)
print(X_train.shape,X_test.shape)
print(y_train.shape,y_test.shape)
model = LinearRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
print("R2 Score  ",r2_score(y_test,y_pred))
```

```
(331, 10) (111, 10)
(331,) (111,)
R2 Score   0.49062912872854303
```

Question NO.2 Using sklearn.datasets.load_wine Apply Correlation and make a heat map using seaborn and remove the highly correlated columns if exist and the apply SVM and get the best accuracy by changing the Hyperparameters

```
from sklearn.datasets import load_wine
```

```
wine = load_wine()
```

```
X2, y2 = wine.data,wine.target
wine['target'] =y2
```

```
wine.feature_names
```

```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
```

```
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']
```

```
wine_df = pd.DataFrame(wine.data,columns=wine.feature_names)
```

```
corr_matrix = wine_df.corr()
corr_matrix
```

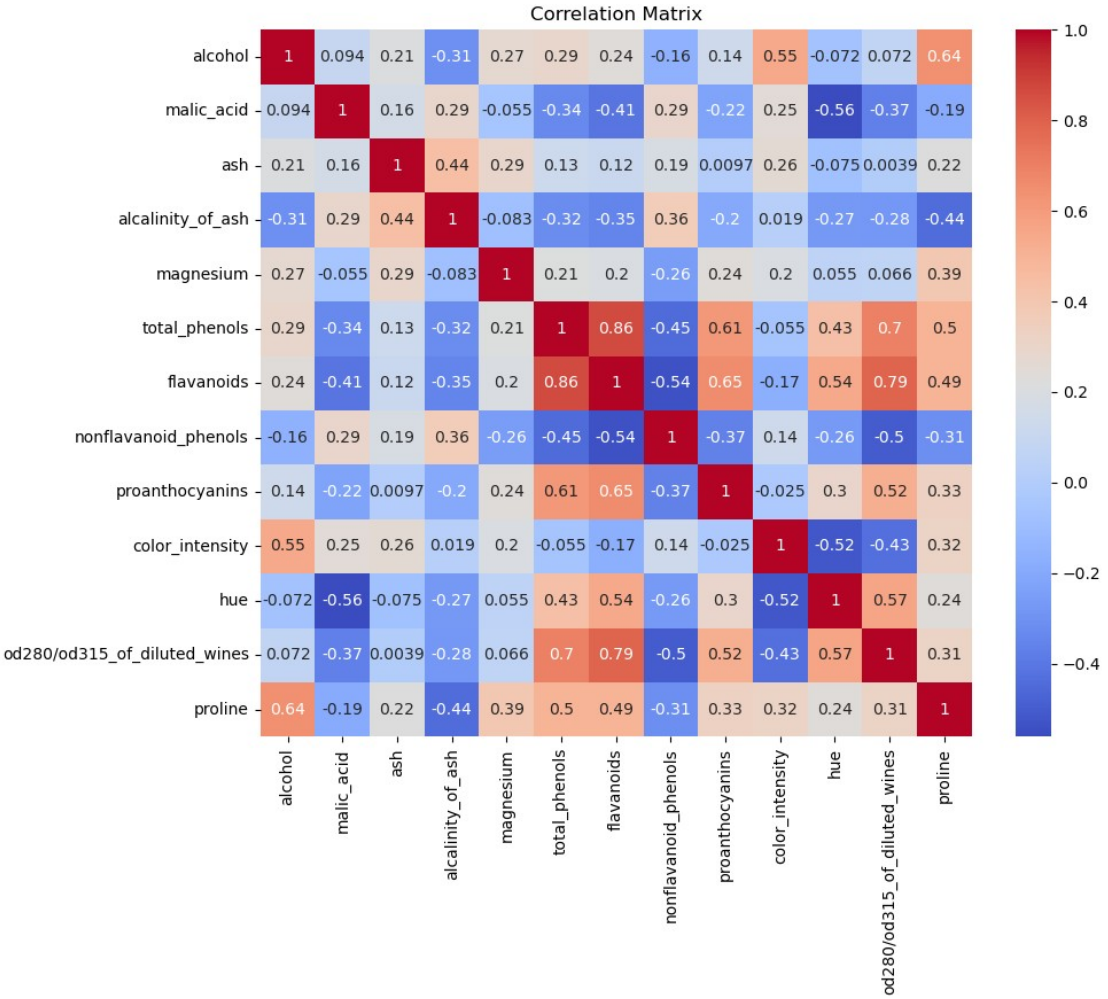
```
                              alcohol  malic_acid       ash  \
alcohol                      1.000000    0.094397  0.211545
malic_acid                   0.094397    1.000000  0.164045
ash                          0.211545    0.164045  1.000000
alcalinity_of_ash           -0.310235    0.288500  0.443367
magnesium                    0.270798   -0.054575  0.286587
total_phenols                0.289101   -0.335167  0.128980
flavanoids                   0.236815   -0.411007  0.115077
nonflavanoid_phenols        -0.155929    0.292977  0.186230
proanthocyanins              0.136698   -0.220746  0.009652
color_intensity              0.546364    0.248985  0.258887
hue                         -0.071747   -0.561296 -0.074667
od280/od315_of_diluted_wines  0.072343   -0.368710  0.003911
proline                      0.643720   -0.192011  0.223626

                             alcalinity_of_ash  magnesium
total_phenols  \
alcohol                          -0.310235   0.270798
0.289101
malic_acid                        0.288500  -0.054575      -
0.335167
ash                               0.443367   0.286587
0.128980
alcalinity_of_ash                 1.000000  -0.083333      -
0.321113
magnesium                        -0.083333   1.000000
0.214401
total_phenols                    -0.321113   0.214401
1.000000
flavanoids                       -0.351370   0.195784
0.864564
nonflavanoid_phenols              0.361922  -0.256294      -
0.449935
proanthocyanins                  -0.197327   0.236441
0.612413
color_intensity                   0.018732   0.199950      -
```

```
0.055136
hue                                -0.273955   0.055398
0.433681
od280/od315_of_diluted_wines       -0.276769   0.066004
0.699949
proline                            -0.440597   0.393351
0.498115

                               flavanoids  nonflavanoid_phenols  \
alcohol                          0.236815             -0.155929
malic_acid                      -0.411007              0.292977
ash                              0.115077              0.186230
alcalinity_of_ash               -0.351370              0.361922
magnesium                        0.195784             -0.256294
total_phenols                    0.864564             -0.449935
flavanoids                       1.000000             -0.537900
nonflavanoid_phenols            -0.537900              1.000000
proanthocyanins                  0.652692             -0.365845
color_intensity                 -0.172379              0.139057
hue                              0.543479             -0.262640
od280/od315_of_diluted_wines     0.787194             -0.503270
proline                          0.494193             -0.311385

                               proanthocyanins  color_intensity
hue  \
alcohol                              0.136698         0.546364 -
0.071747
malic_acid                          -0.220746         0.248985 -
0.561296
ash                                  0.009652         0.258887 -
0.074667
alcalinity_of_ash                   -0.197327         0.018732 -
0.273955
magnesium                            0.236441         0.199950
0.055398
total_phenols                        0.612413        -0.055136
0.433681
flavanoids                           0.652692        -0.172379
0.543479
nonflavanoid_phenols                -0.365845         0.139057 -
0.262640
proanthocyanins                      1.000000        -0.025250
0.295544
color_intensity                     -0.025250         1.000000 -
0.521813
hue                                  0.295544        -0.521813
1.000000
od280/od315_of_diluted_wines         0.519067        -0.428815
0.565468
proline                              0.330417         0.316100
```

```
0.236183
```

	od280/od315_of_diluted_wines	proline
alcohol	0.072343	0.643720
malic_acid	-0.368710	-0.192011
ash	0.003911	0.223626
alcalinity_of_ash	-0.276769	-0.440597
magnesium	0.066004	0.393351
total_phenols	0.699949	0.498115
flavanoids	0.787194	0.494193
nonflavanoid_phenols	-0.503270	-0.311385
proanthocyanins	0.519067	0.330417
color_intensity	-0.428815	0.316100
hue	0.565468	0.236183
od280/od315_of_diluted_wines	1.000000	0.312761
proline	0.312761	1.000000

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the correlation matrix heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```




```
threshold = 0.5
column_names = set()
for row in range(len(corr_matrix)):
    for col in range(row):
        if abs(corr_matrix.iloc[row][col]) > threshold:
            print(corr_matrix.iloc[row][col])
            print(corr_matrix.columns[row])
            column_names.add(corr_matrix.columns[row])
```

```
0.8645635000951147
flavanoids
-0.5378996119051984
nonflavanoid_phenols
0.6124130837800363
proanthocyanins
0.6526917686075153
proanthocyanins
0.546364195083705
color_intensity
-0.5612956886649447
```

```
hue
0.5434785664899897
hue
-0.5218131932287572
hue
0.6999493647911861
od280/od315_of_diluted_wines
0.787193901866951
od280/od315_of_diluted_wines
-0.5032695960789114
od280/od315_of_diluted_wines
0.519067095682523
od280/od315_of_diluted_wines
0.5654682931826589
od280/od315_of_diluted_wines
0.6437200371782134
proline
```

```
column_names
```

```
{'color_intensity',
 'flavanoids',
 'hue',
 'nonflavanoid_phenols',
 'od280/od315_of_diluted_wines',
 'proanthocyanins',
 'proline'}
```

```
wine_df.head()
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium
total_phenols  \
0    14.23        1.71  2.43               15.6      127.0
2.80
1    13.20        1.78  2.14               11.2      100.0
2.65
2    13.16        2.36  2.67               18.6      101.0
2.80
3    14.37        1.95  2.50               16.8      113.0
3.85
4    13.24        2.59  2.87               21.0      118.0
2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity
hue  \
0        3.06                  0.28             2.29             5.64
1.04
1        2.76                  0.26             1.28             4.38
1.05
2        3.24                  0.30             2.81             5.68
1.03
```

```
3        3.49                  0.24             2.18              7.80
0.86
4        2.69                  0.39             1.82              4.32
1.04

   od280/od315_of_diluted_wines  proline
0                          3.92   1065.0
1                          3.40   1050.0
2                          3.17   1185.0
3                          3.45   1480.0
4                          2.93    735.0
```

```
X2_filt = wine_df.drop(column_names, axis = 1)
```

```
X2_filt.head()
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium
total_phenols
0    14.23        1.71  2.43               15.6      127.0
2.80
1    13.20        1.78  2.14               11.2      100.0
2.65
2    13.16        2.36  2.67               18.6      101.0
2.80
3    14.37        1.95  2.50               16.8      113.0
3.85
4    13.24        2.59  2.87               21.0      118.0
2.80
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.svm import SVC

y2= wine['target']

X_train,X_test,y_train,y_test = train_test_split(X2_filt,y2, test_size
=0.24,random_state = 91)
print(X_train.shape,X_test.shape)
print(y_train.shape,y_test.shape)
```

```
(135, 6) (43, 6)
(135,) (43,)
```

```
classifier = SVC(kernel = 'poly')

classifier.fit(X_train,y_train)
```

```
SVC(kernel='poly')
```

```
y2_pred = classifier.predict(X_test)
```

```
print(" Initial Accuracy - ",accuracy_score(y_test,y2_pred))
print(" Initial f1 score - ",f1_score(y_test,y2_pred, average =
'weighted') )
```

```
 Initial Accuracy -  0.5581395348837209
 Initial f1 score -  0.4918297498903028
```

```
# Changing Hyperparameters so that we can get better Accuracy
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1, 1, 10],
              'kernel': ['linear', 'rbf', 'poly'],
              'gamma': ['scale', 'auto']}
grid_search = GridSearchCV(classifier, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_svm = grid_search.best_estimator_
best_predictions = best_svm.predict(X_test)
best_accuracy = accuracy_score(y_test, best_predictions)
final_svm = SVC(**grid_search.best_params_)
final_svm.fit(X_train, y_train)
final_predictions = final_svm.predict(X_test)
final_accuracy = accuracy_score(y_test, final_predictions)

print("This is the Best Accuracy after changing Hyper-parameters")
print("Best Hyperparameters:", grid_search.best_params_)
print("Final Accuracy:", final_accuracy)
print(" Final F1
Score:",f1_score(y_test,final_predictions,average="weighted"))
```

```
This is the Best Accuracy after changing Hyper-parameters
Best Hyperparameters: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Final Accuracy: 0.8837209302325582
 Final F1 Score: 0.8785295871479182
```

Question NO.3 Using sklearn.datasets.load_diabetes apply Mutual info Classification and check which are the best columns according to the target column. Then Apply decision tree on that data and try to get best accuracy by changing the hyperparameters

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.datasets import load_diabetes
from sklearn.model_selection import GridSearchCV

data = load_diabetes()
X3 = data.data
y3 = data.target

df = pd.DataFrame(X3, columns=data.feature_names)
df['target'] = y3

df.head()
```

```
        age       sex       bmi        bp        s1        s2
s3  \
```

```
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -
0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163
0.074412
2  0.085299  0.050680  0.044451 -0.005671 -0.045599 -0.034194 -
0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -
0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596
0.008142

         s4        s5        s6  target
0 -0.002592  0.019908 -0.017646   151.0
1 -0.039493 -0.068330 -0.092204    75.0
2 -0.002592  0.002864 -0.025930   141.0
3  0.034309  0.022692 -0.009362   206.0
4 -0.002592 -0.031991 -0.046641   135.0
```

```
selector = SelectKBest(score_func=mutual_info_classif, k=4)
X_new = selector.fit_transform(X3, y3)

selected_features =
df.columns[selector.get_support(indices=True)].tolist()

print("Selected Features:", selected_features)
```

```
Selected Features: ['sex', 's3', 's4', 's5']
```

```
X_filt = df[['sex', 's3', 's4', 's5']]
X_filt.head()
```

```
        sex        s3        s4        s5
0  0.050680 -0.043401 -0.002592  0.019908
1 -0.044642  0.074412 -0.039493 -0.068330
2  0.050680 -0.032356 -0.002592  0.002864
3 -0.044642 -0.036038  0.034309  0.022692
4 -0.044642  0.008142 -0.002592 -0.031991
```

```
df['target'] = y3

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import r2_score
X_train,X_test,y_train,y_test = train_test_split(X_filt,y3, test_size
=0.29,random_state = 17)
print(X_train.shape,X_test.shape)
print(y_train.shape,y_test.shape)
tree = DecisionTreeClassifier(criterion = "entropy")
tree.fit(X_train,y_train)
```

```
y_pred = tree.predict(X_test)
print("Accuracy of Decision Tree Model
",accuracy_score(y_test,y_pred))

print("r2 ",r2_score(y_test,y_pred))
print(" Initial Accuracy - ",accuracy_score(y_test,y_pred))
print(" Initial f1 score - ",f1_score(y_test,y_pred, average =
'weighted') )
```

```
(313, 4) (129, 4)
(313,) (129,)
Accuracy of Decision Tree Model  0.015503875968992248
r2  -0.3063360751715225
 Initial Accuracy -  0.015503875968992248
 Initial f1 score -  0.012919896640826873
```

```
# Define the hyperparameters to optimize
hyperparameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 8, 10],
    'min_samples_split': [2, 4, 8],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(tree, hyperparameters, cv=2)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\
_split.py:676: UserWarning: The least populated class in y has only 1
members, which is less than n_splits=2.
  warnings.warn(
```

```
print("Best columns:", selected_features)
print("Best hyperparameters:", grid_search.best_params_)
print("Best accuracy:", accuracy)
```

```
Best columns: ['sex', 's3', 's4', 's5']
Best hyperparameters: {'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 4, 'min_samples_split': 8}
Best accuracy: 0.007751937984496124
```

Question NO. 4 Using sklearn.datasets.load_boston apply Mutual info Regression and check which are the best columns according to the target column. Then Apply MultiLinear Regression on that data and try to get best accuracy by changing the hyperparameters

```
from sklearn.feature_selection import SelectKBest,
mutual_info_regression
from sklearn.datasets import load_boston

bos = load_boston()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\
deprecation.py:87: FutureWarning: Function load_boston is deprecated;
`load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can
refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use
of this
    dataset unless the purpose of the code is to study and educate
about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22,
header=None)
        data = np.hstack([raw_df.values[::2, :],
raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames
housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
```

```
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

```
bos.feature_names
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

```
X4 = bos.data
y4= bos.target
bos_data = pd.DataFrame(X4, columns=bos.feature_names)
bos_data.head()
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX
\
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0

1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0

2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0

3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0

4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0


   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

```
bos_data['target'] = y4
```

```
selector = SelectKBest(score_func=mutual_info_regression, k=4)
X_new = selector.fit_transform(X4, y4)
selected_features =
bos_data.columns[selector.get_support(indices=True)].tolist()
print("Selected Features:", selected_features)
```

```
Selected Features: ['INDUS', 'NOX', 'RM', 'LSTAT']
```

```
X_filt1 = bos_data[['INDUS', 'NOX', 'RM', 'LSTAT']]
X_filt1.head()
```

```
   INDUS    NOX     RM  LSTAT
0   2.31  0.538  6.575   4.98
1   7.07  0.469  6.421   9.14
2   7.07  0.469  7.185   4.03
3   2.18  0.458  6.998   2.94
4   2.18  0.458  7.147   5.33
```

```
X_train,X_test,y_train,y_test = train_test_split(X_filt1,y4, test_size
=0.28,random_state = 63)
print(X_train.shape,X_test.shape)
print(y_train.shape,y_test.shape)
from sklearn import linear_model
model= linear_model.LinearRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
print("Accuracy for this model",r2_score(y_test,y_pred))
```

```
(364, 4) (142, 4)
(364,) (142,)
Accuracy for this model 0.628740044008688
```

```
#change the hyperparameters to get better accuracy
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

param_grid = {
    'linearregression__normalize': [True, False],
    'linearregression__fit_intercept': [True, False]
}

pipeline = make_pipeline(StandardScaler(), LinearRegression())
grid_search = GridSearchCV(pipeline, param_grid, cv=5,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
r2_best = r2_score(y_test, y_pred_best)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())
```

```
If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
```

```
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
```

```
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:
```

```
kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
```

```
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:148: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2. Please leave the normalize parameter to
its default value to silence this warning. The default behavior of
this estimator is to not do any normalization. If normalization is
needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0
and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False),
LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in
model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
```

```
print("Best R^2 Score:", r2_best)
print("Best Model Parameters:", best_params)
```

```
Best R^2 Score: 0.628740044008688
Best Model Parameters: {'linearregression__fit_intercept': True,
'linearregression__normalize': True}
```