In [ ]:
```
from sklearn import datasets
data = datasets.load_diabetes(return_X_y=False,as_frame=True)
#Exploring the Data:
print(data.data.head())
features_name=data.feature_names
print(features_name)
```

```
        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 -0.005670 -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  0.019907 -0.017646
1 -0.039493 -0.068332 -0.092204
2 -0.002592  0.002861 -0.025930
3  0.034309  0.022688 -0.009362
4 -0.002592 -0.031988 -0.046641
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

In [ ]:
```
features=data.data
target=data.target
```

In [ ]:
```
#check number of samples and check null values:
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
dtypes: float64(10)
memory usage: 34.7 KB
```

In [ ]:
```
#check the null on the label column:
target.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 442 entries, 0 to 441
Series name: target
Non-Null Count  Dtype  
--------------  -----  
442 non-null    float64
dtypes: float64(1)
memory usage: 3.6 KB
```

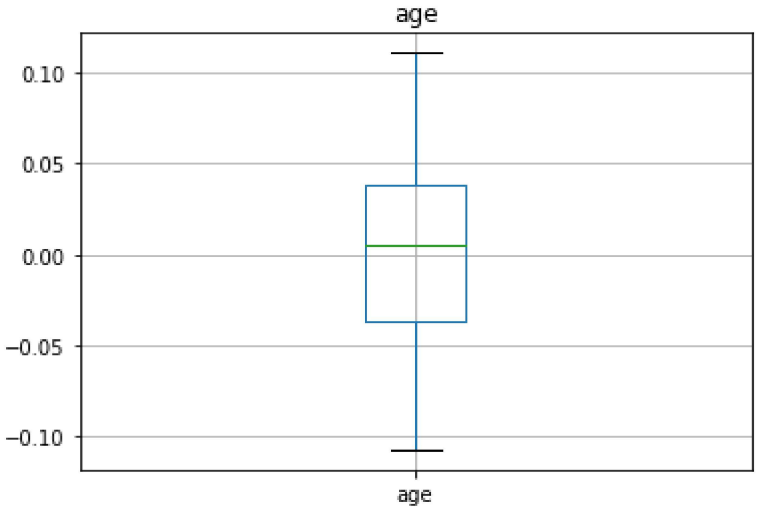
In [ ]:
```
#do satatitical analysis to find the mean , min , and max values :
```

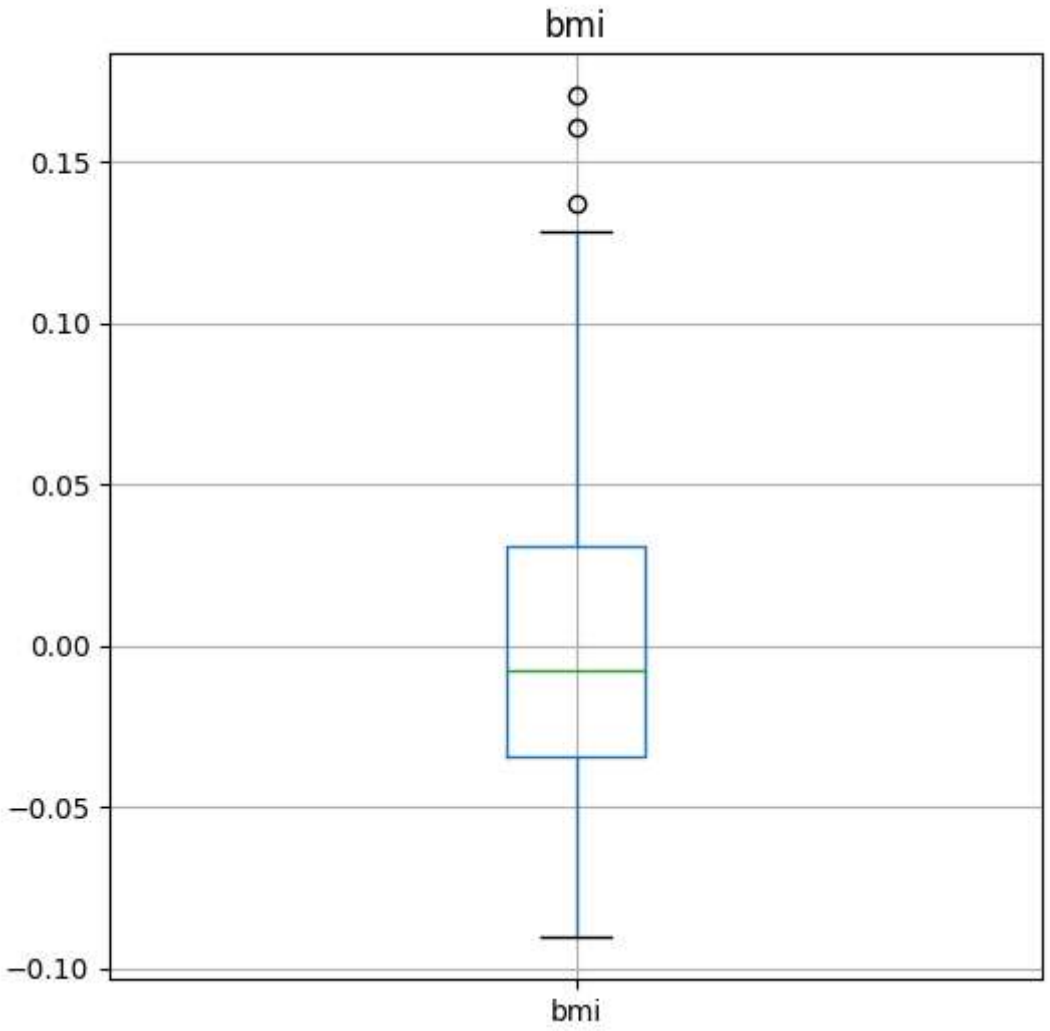
```
Diabetes=features.copy()
Diabetes['target']=target
Diabetes.describe()
```

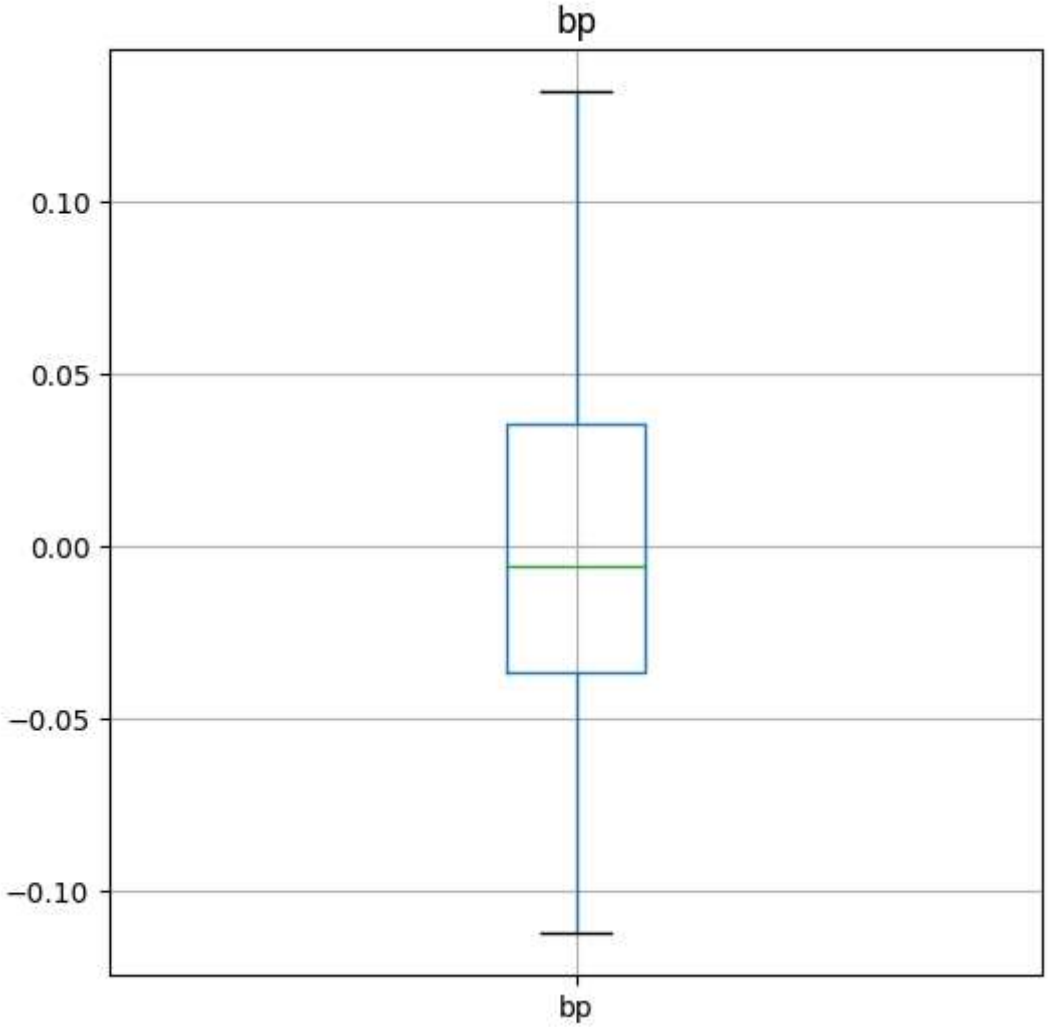
Out[ ]:

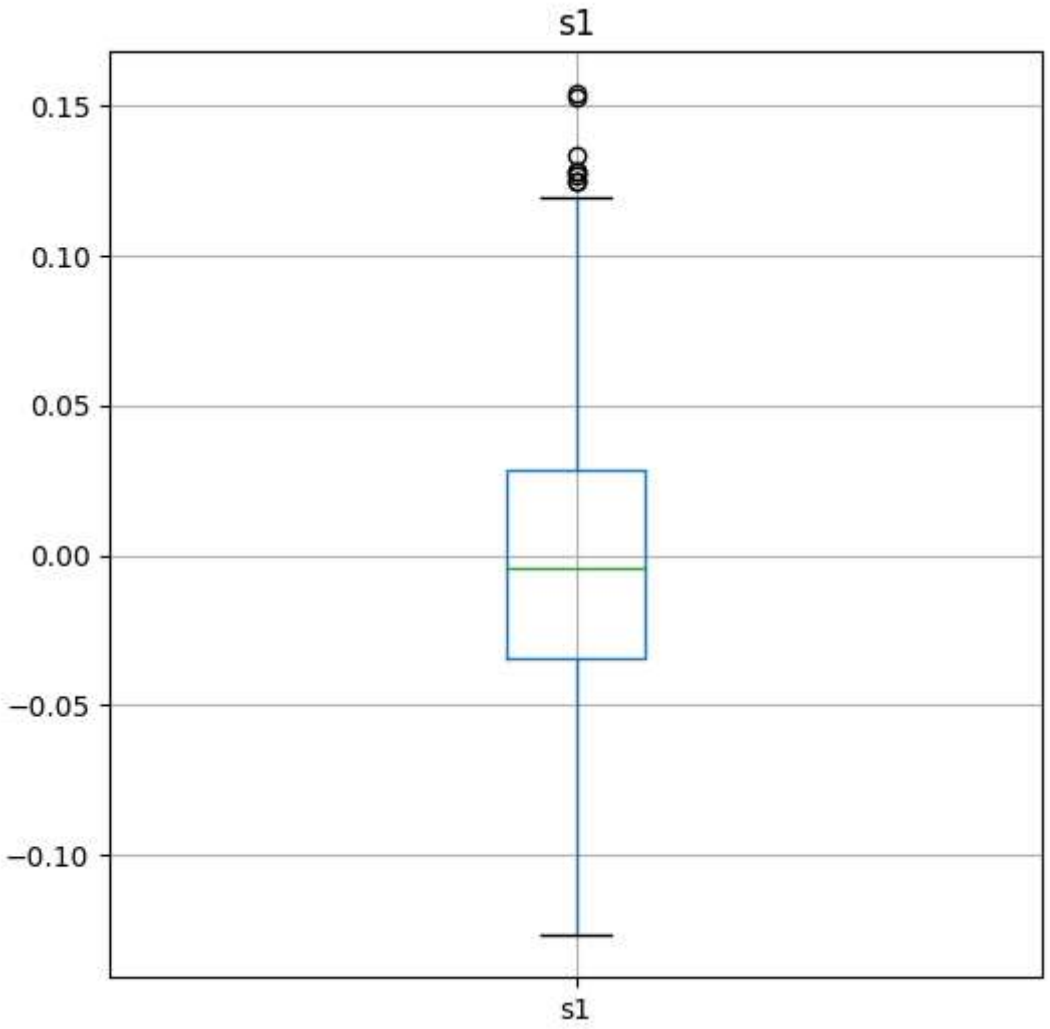
	age	sex	bmi	bp	s1	s2	
count	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.4200(
mean	-2.511817e-19	1.230790e-17	-2.245564e-16	-4.797570e-17	-1.381499e-17	3.918434e-17	-5.77
std	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.7619
min	-1.072256e-01	-4.464164e-02	-9.027530e-02	-1.123988e-01	-1.267807e-01	-1.156131e-01	-1.02
25%	-3.729927e-02	-4.464164e-02	-3.422907e-02	-3.665608e-02	-3.424784e-02	-3.035840e-02	-3.51
50%	5.383060e-03	-4.464164e-02	-7.283766e-03	-5.670422e-03	-4.320866e-03	-3.819065e-03	-6.58
75%	3.807591e-02	5.068012e-02	3.124802e-02	3.564379e-02	2.835801e-02	2.984439e-02	2.9311
max	1.107267e-01	5.068012e-02	1.705552e-01	1.320436e-01	1.539137e-01	1.987880e-01	1.8117

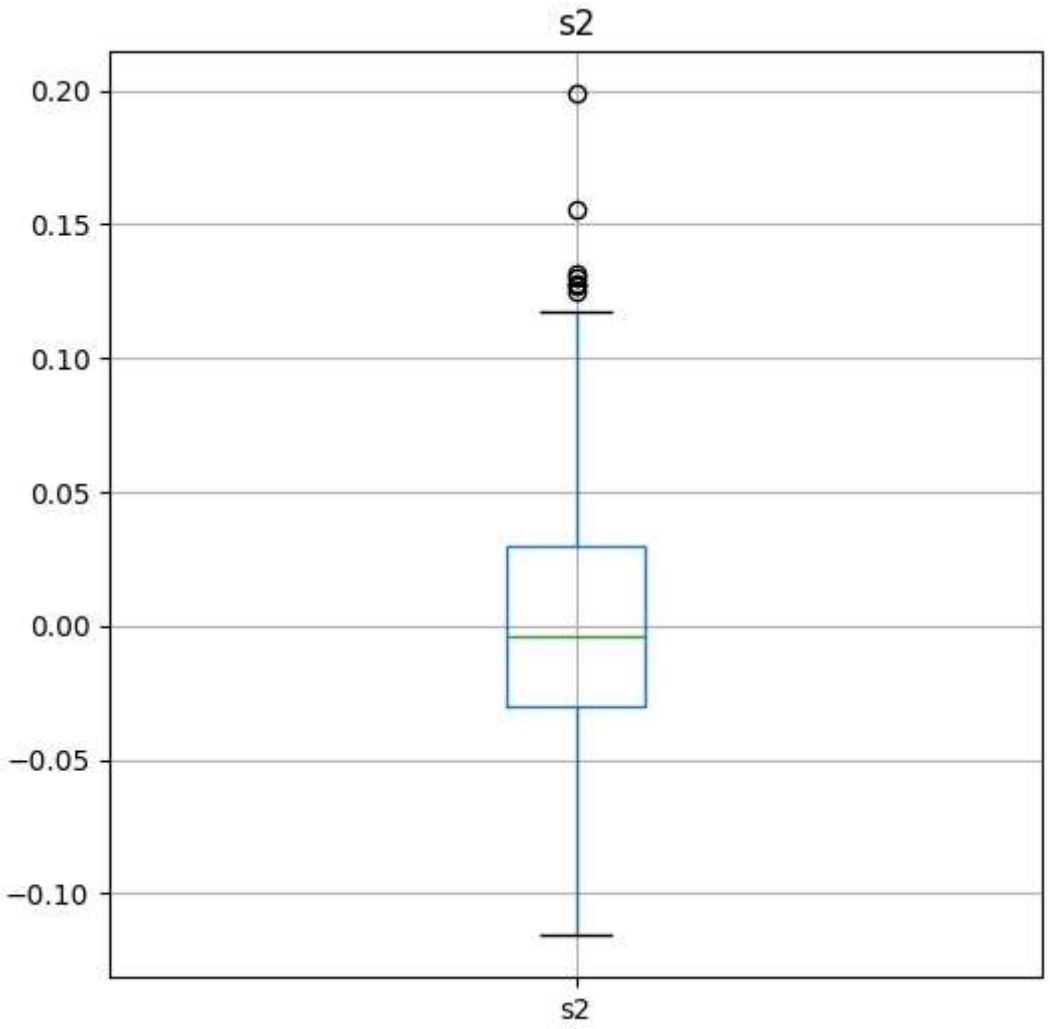
In [ ]:
```
#Check Outliers by using boxplot :
from matplotlib import pyplot as plt
num_features=['age', 'bmi', 'bp', 's1', 's2', 's3', 's5', 's6']
cat_features=['sex','s4']
type(features)
for col in num_features:
    Diabetes.boxplot(column=col, figsize=(6,6))
    plt.title(col)
    plt.show()
```

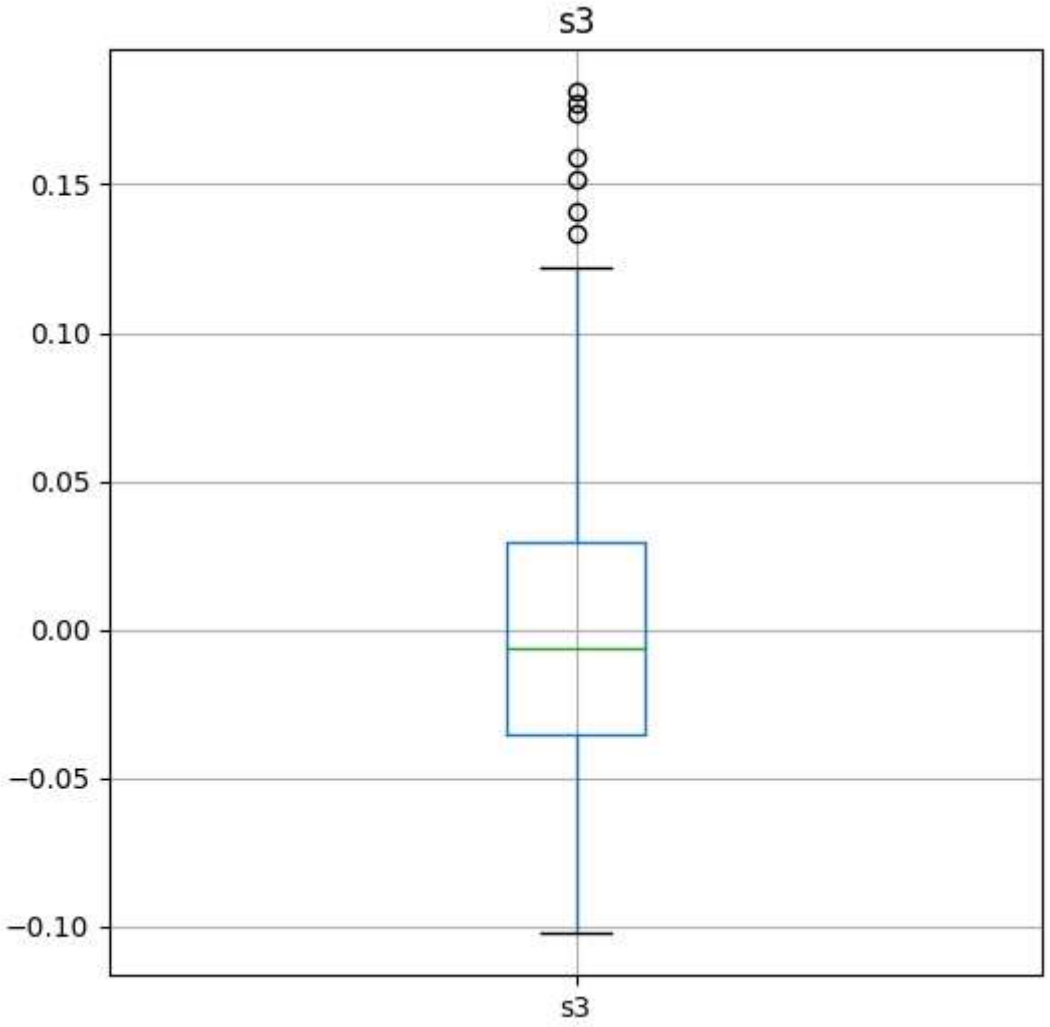


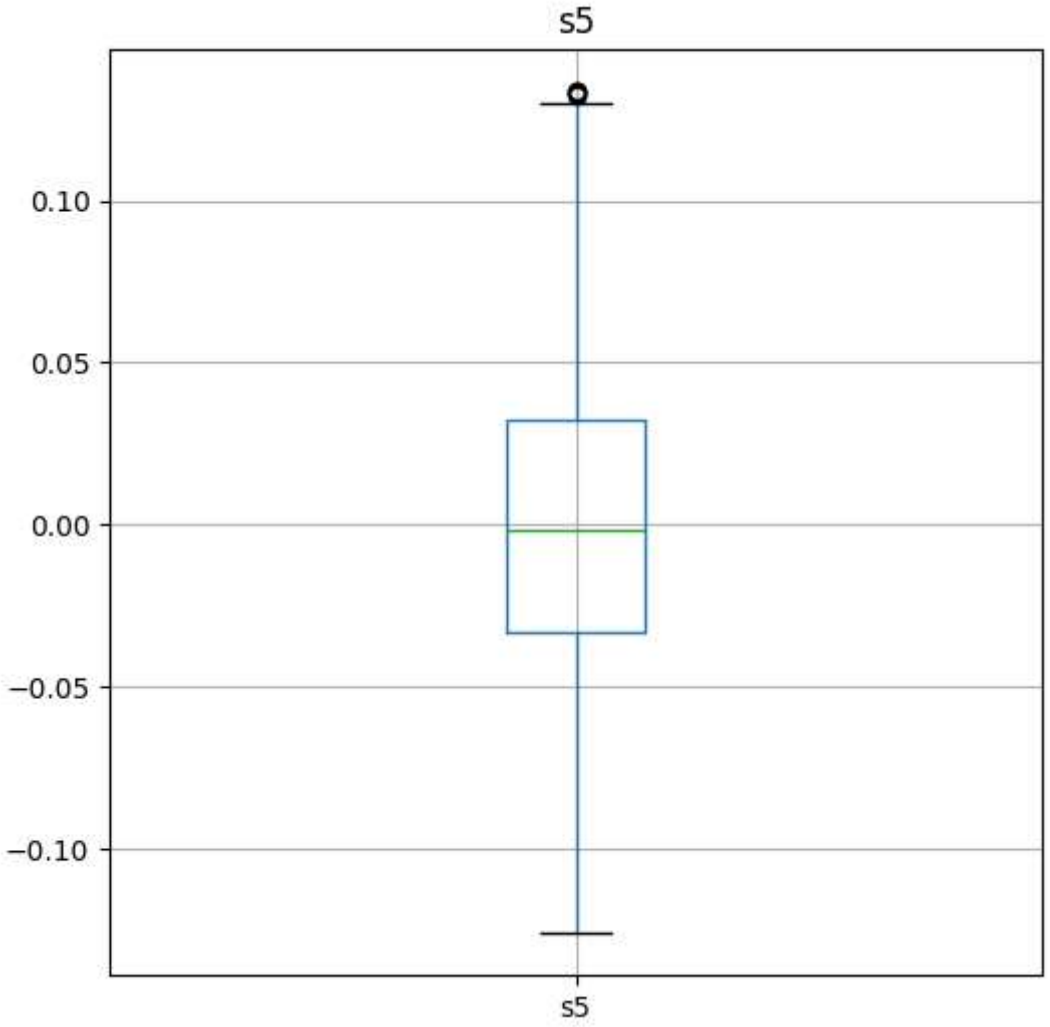

bmi
0.15
0.10
0.05
0.00
−0.05
−0.10
bmi

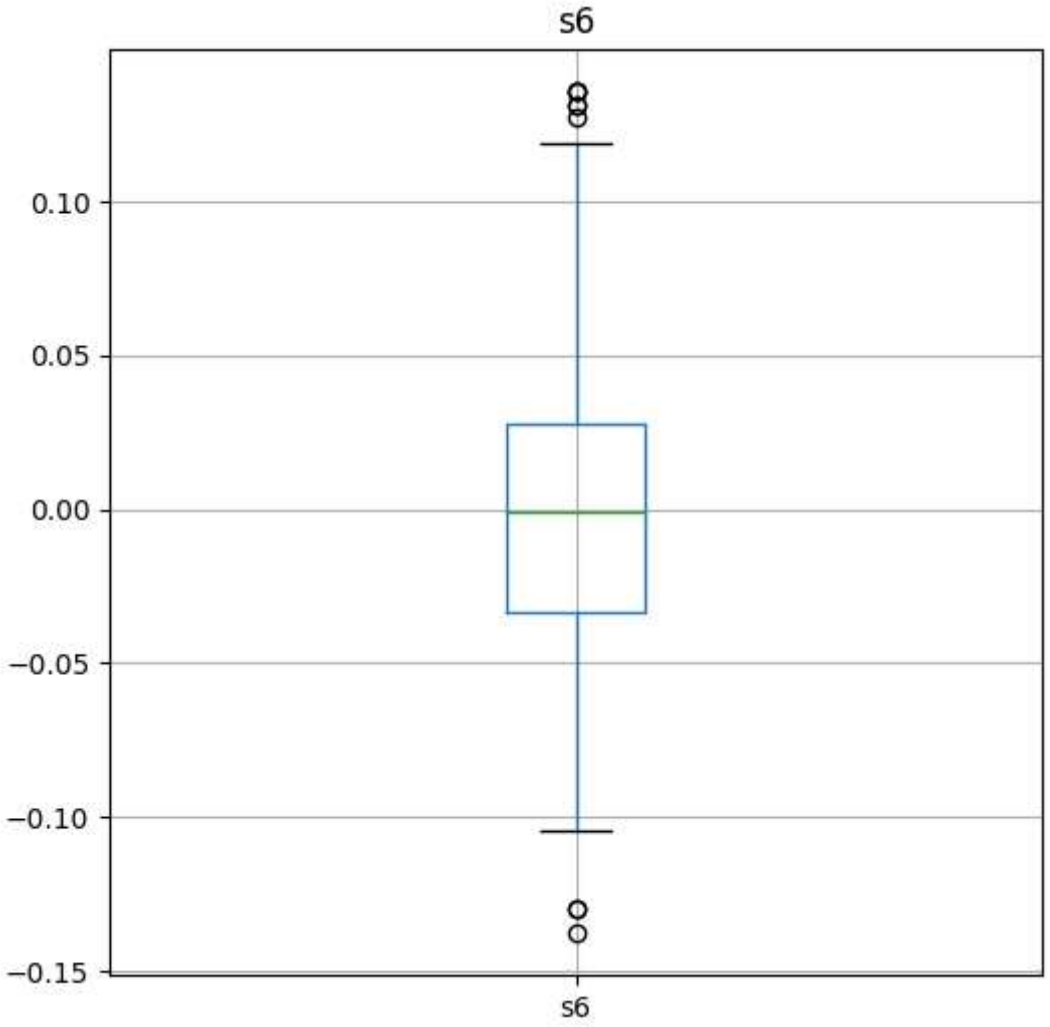

bp
0.10
0.05
0.00
−0.05
−0.10
bp


s1
0.15
0.10
0.05
0.00
−0.05
−0.10
s1

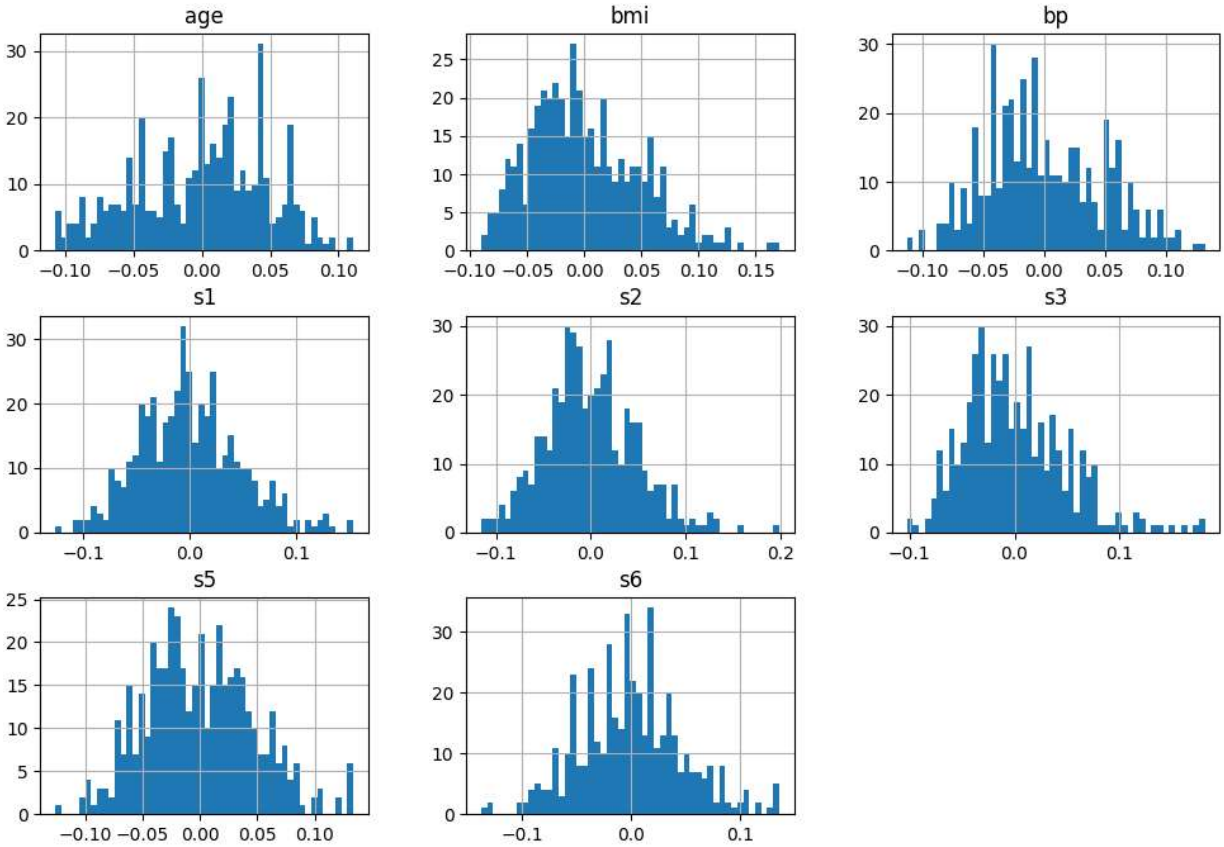

s2
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
s2


s3
0.15
0.10
0.05
0.00
−0.05
−0.10
s3

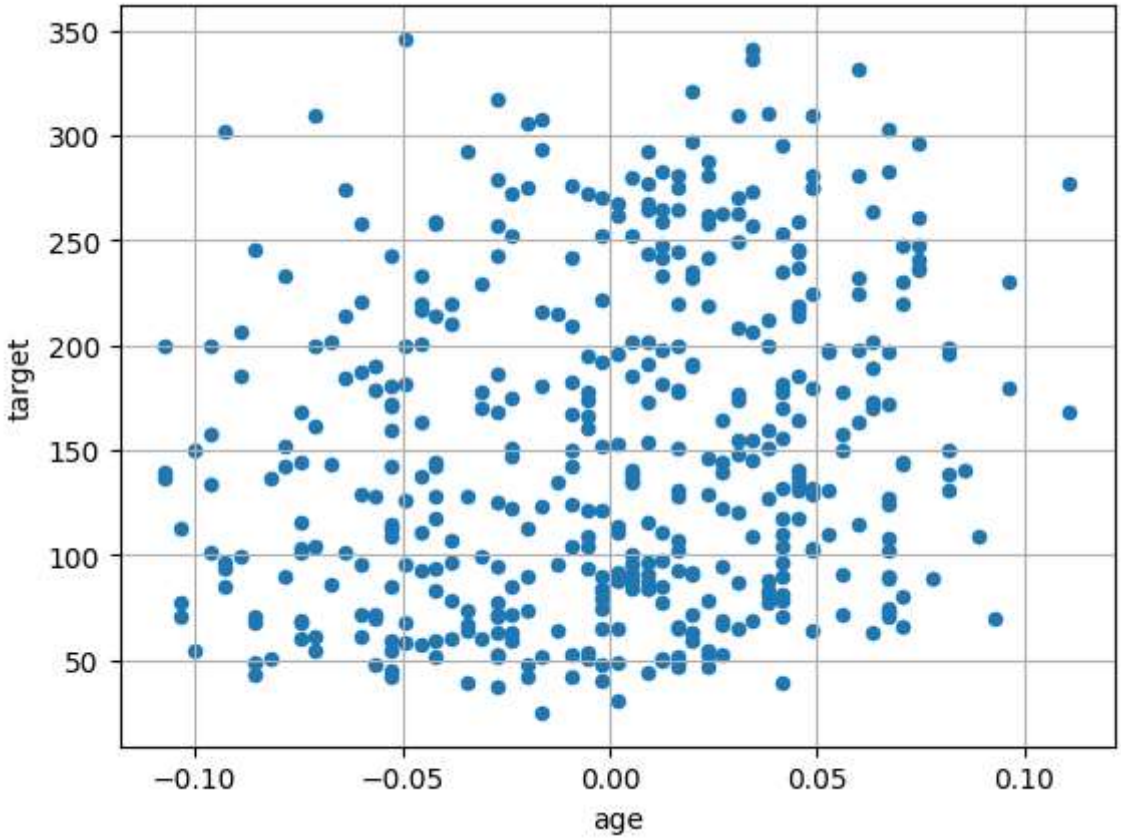

s5
0.10
0.05
0.00
−0.05
−0.10
s5

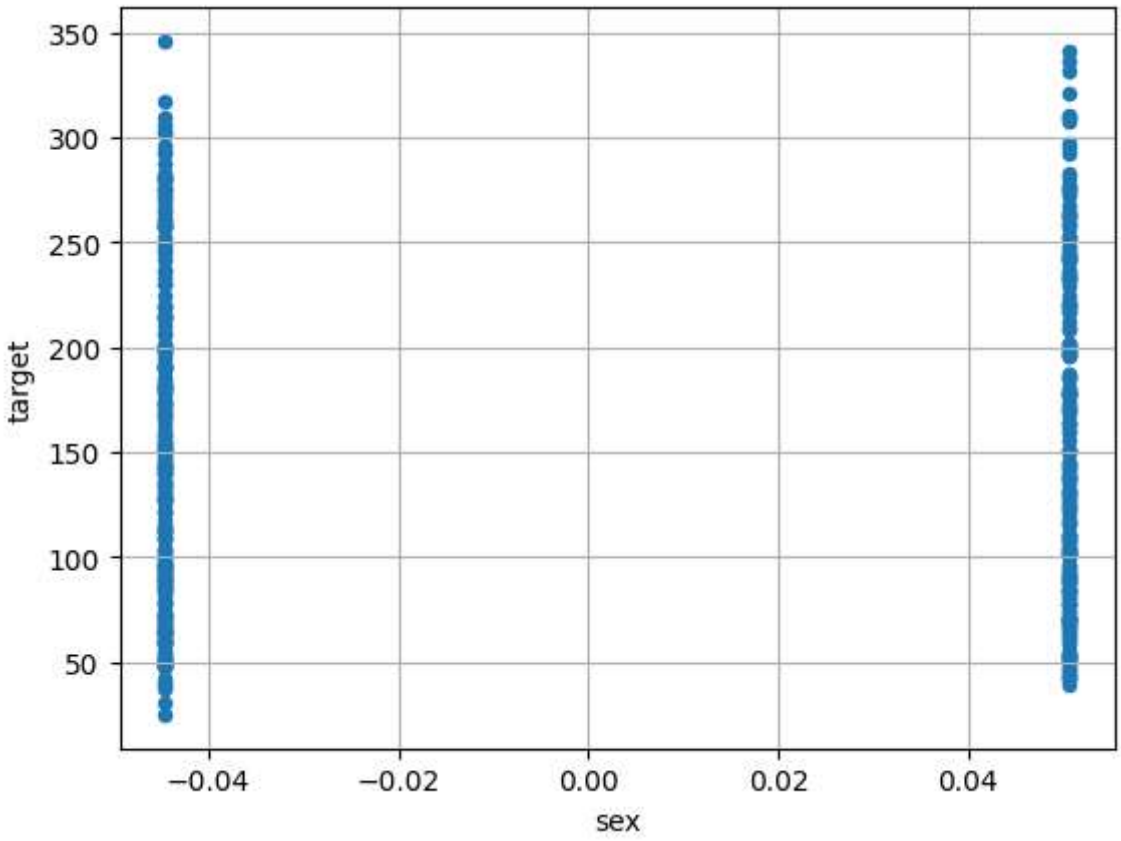


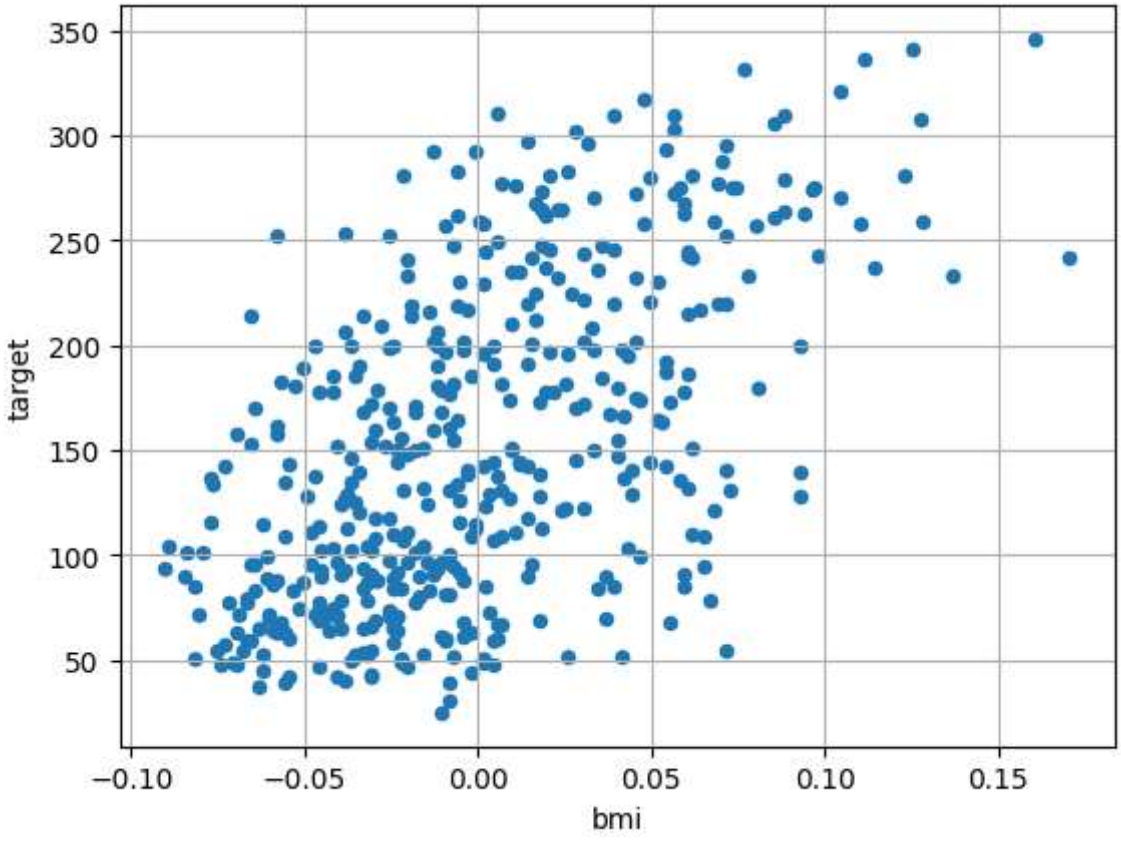
In [ ]:
```
#plot the distripution of each numarical feature :
#the data is normalized between (-0.1,0.1)
import matplotlib.pyplot as plt

Diabetes[num_features].hist(bins=50,figsize=(12,8))
plt.show()
```

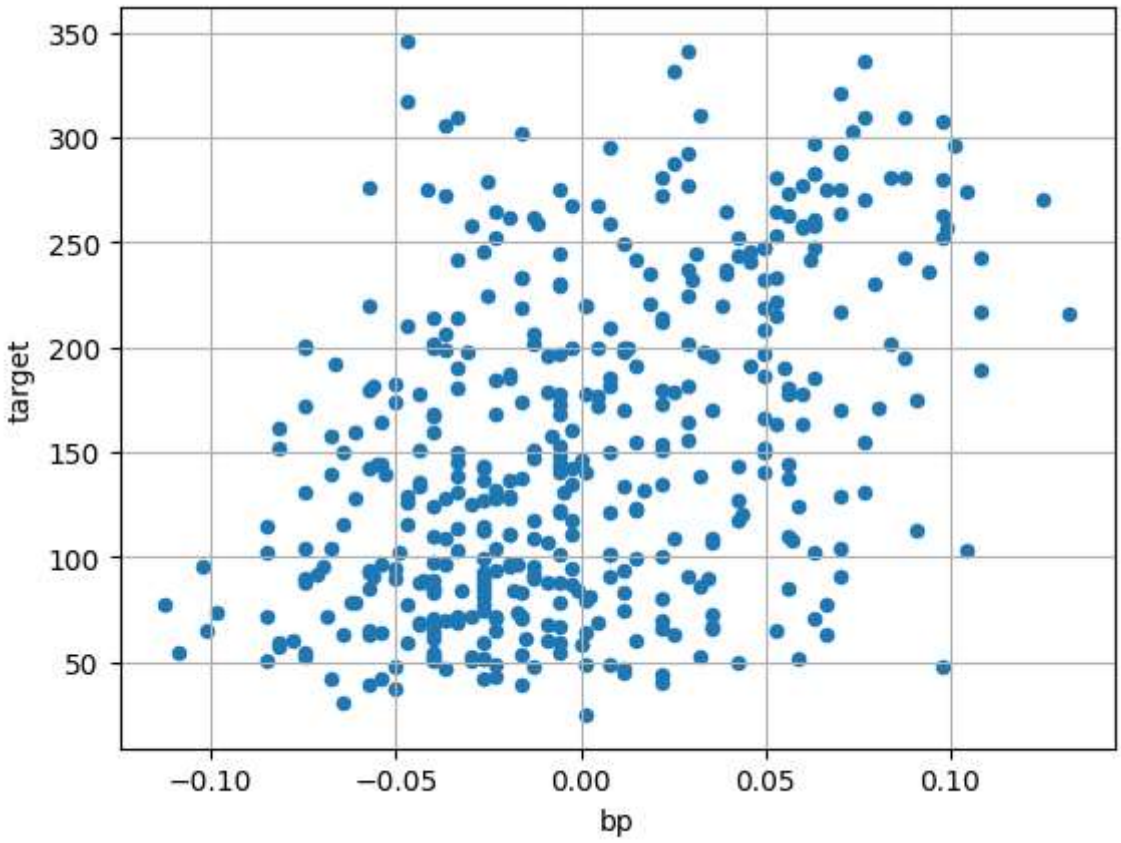


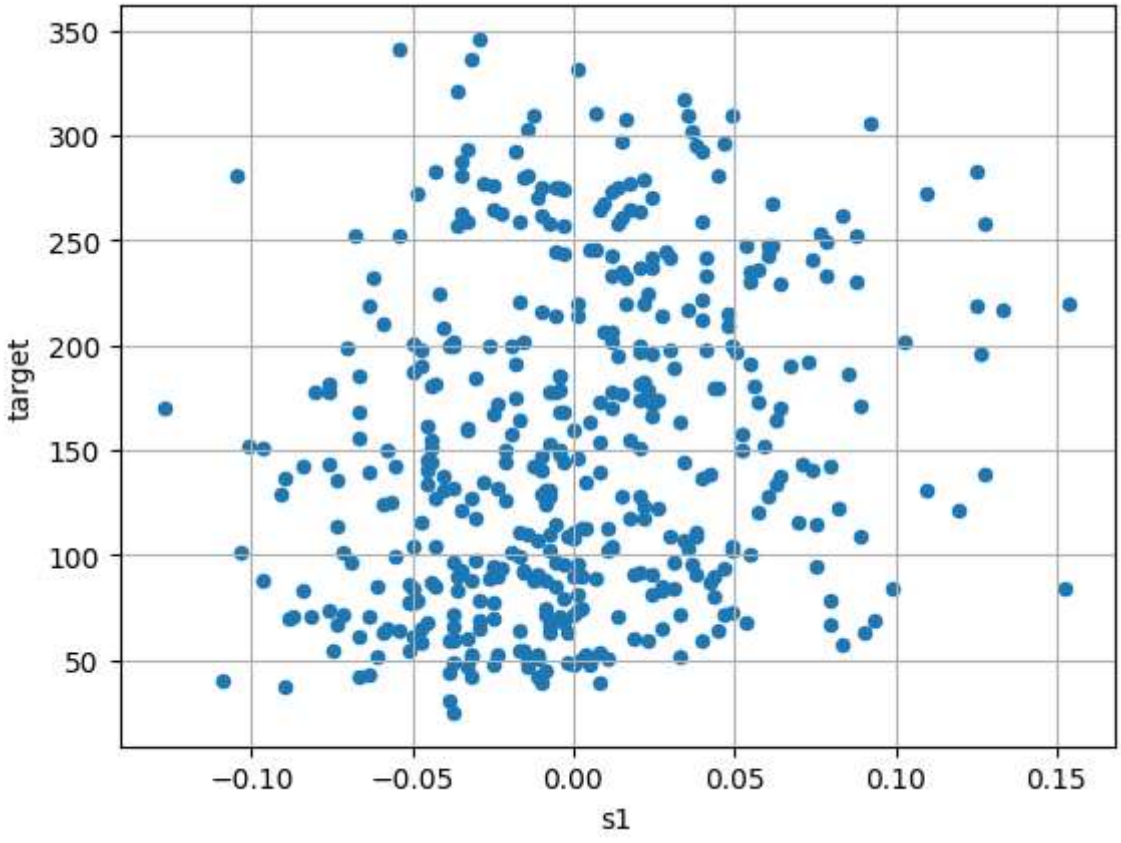
In [ ]:
```
for feature in features_name:
    Diabetes.plot(kind='scatter',x=feature,y='target',grid=True)
    plt.show()
```

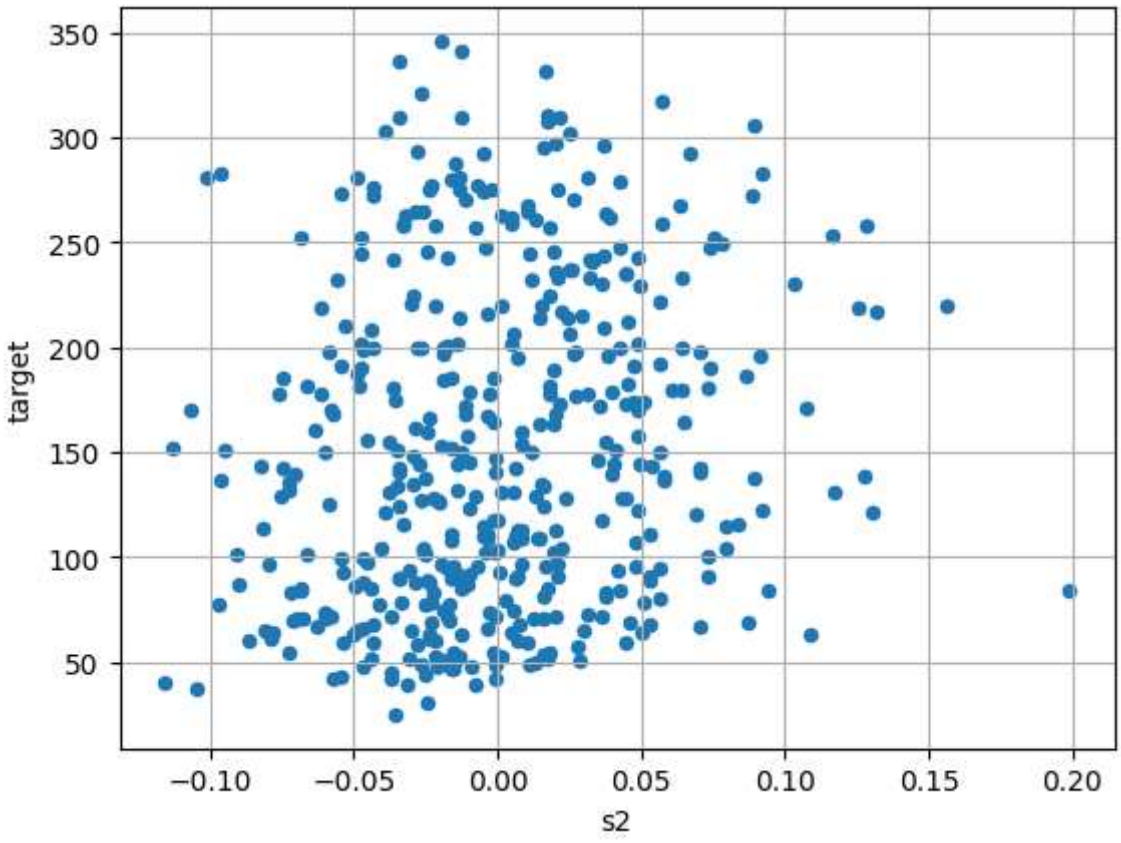


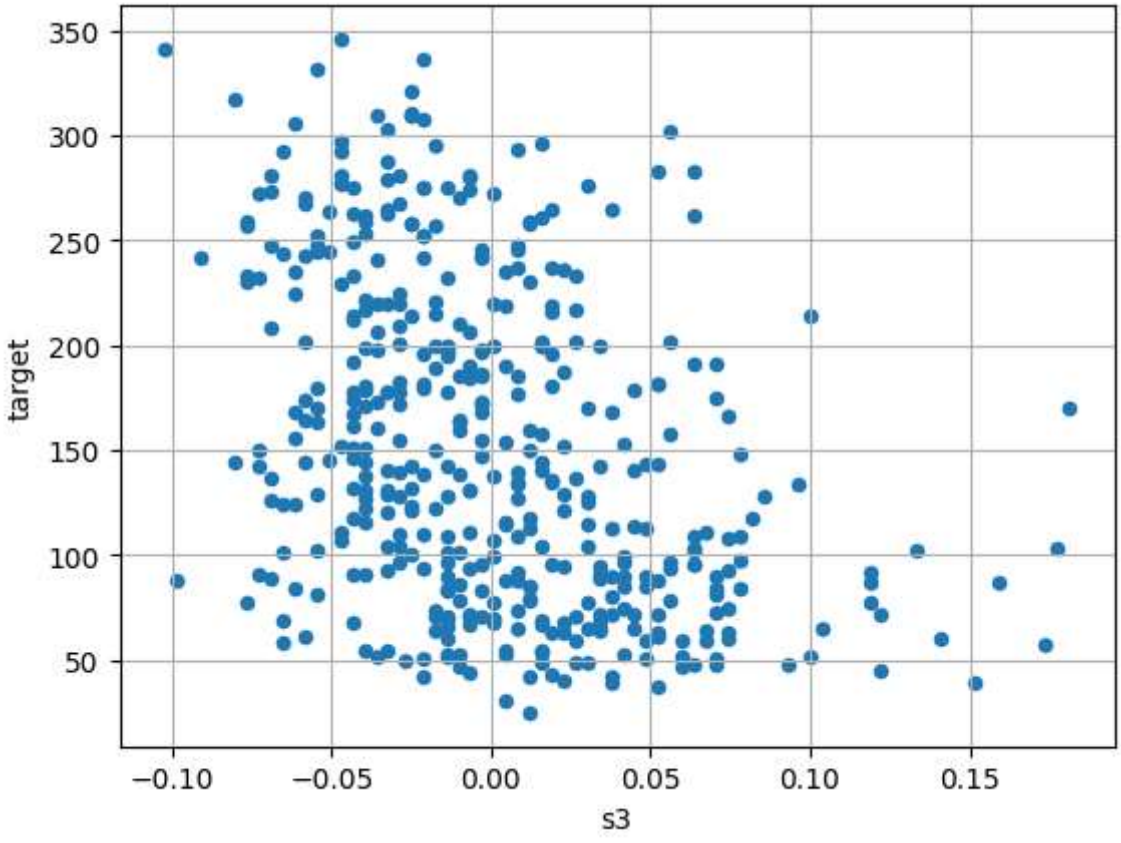

target
350
300
250
200
150
100
50
−0.04
−0.02
0.00
0.02
0.04
sex

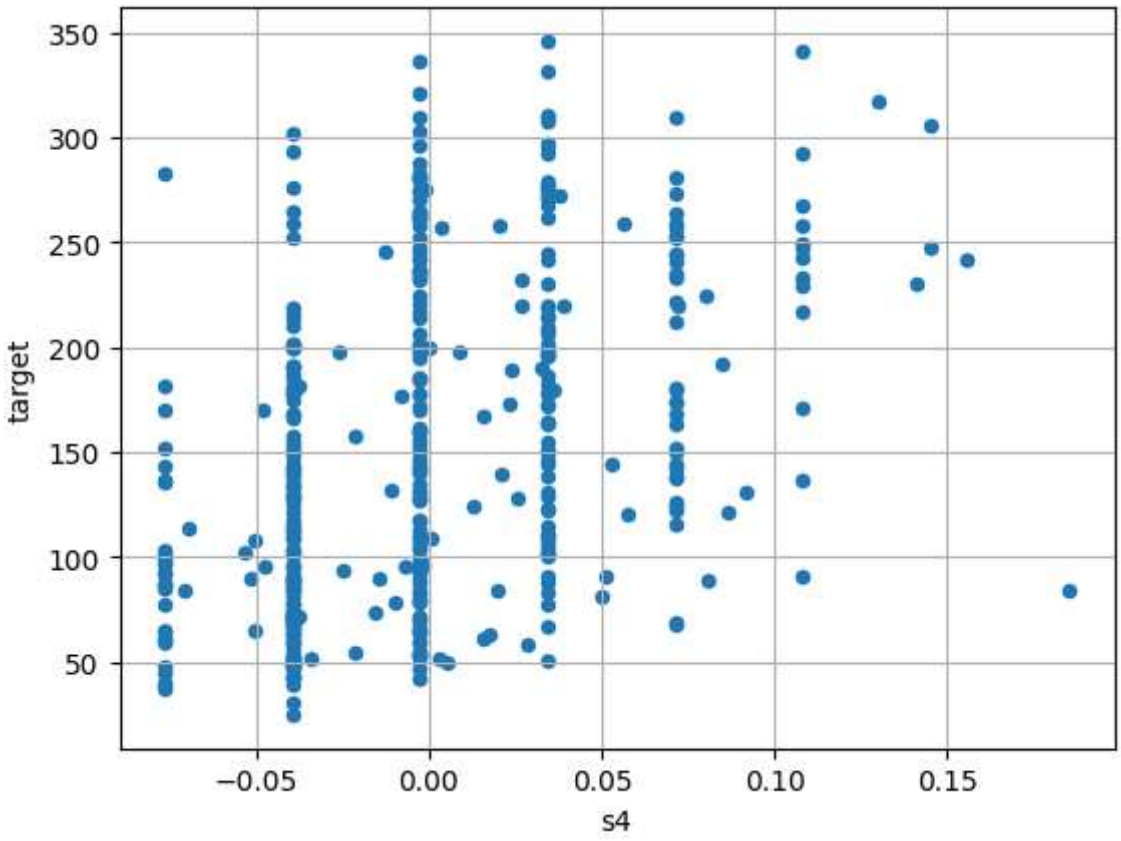

target
350
300
250
200
150
100
50
−0.10
−0.05
0.00
0.05
0.10
0.15
bmi

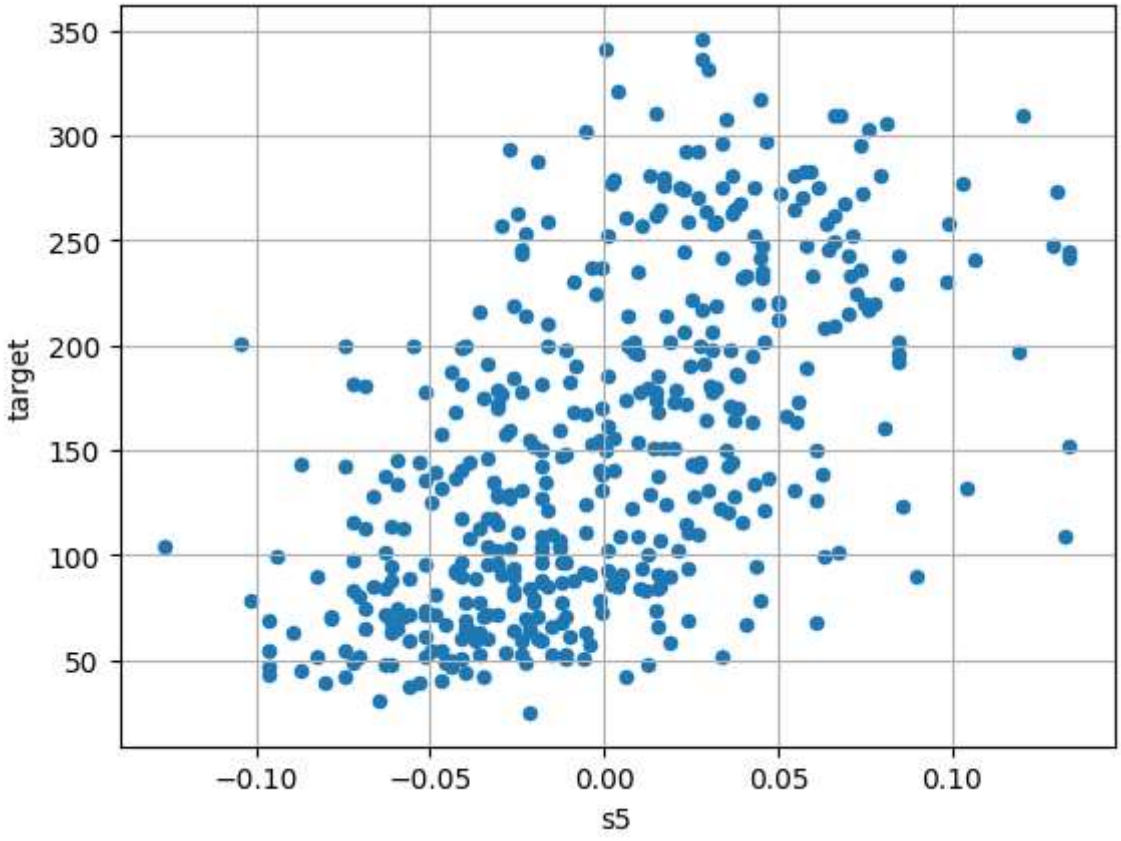

target
bp
350
300
250
200
150
100
50
−0.10
−0.05
0.00
0.05
0.10

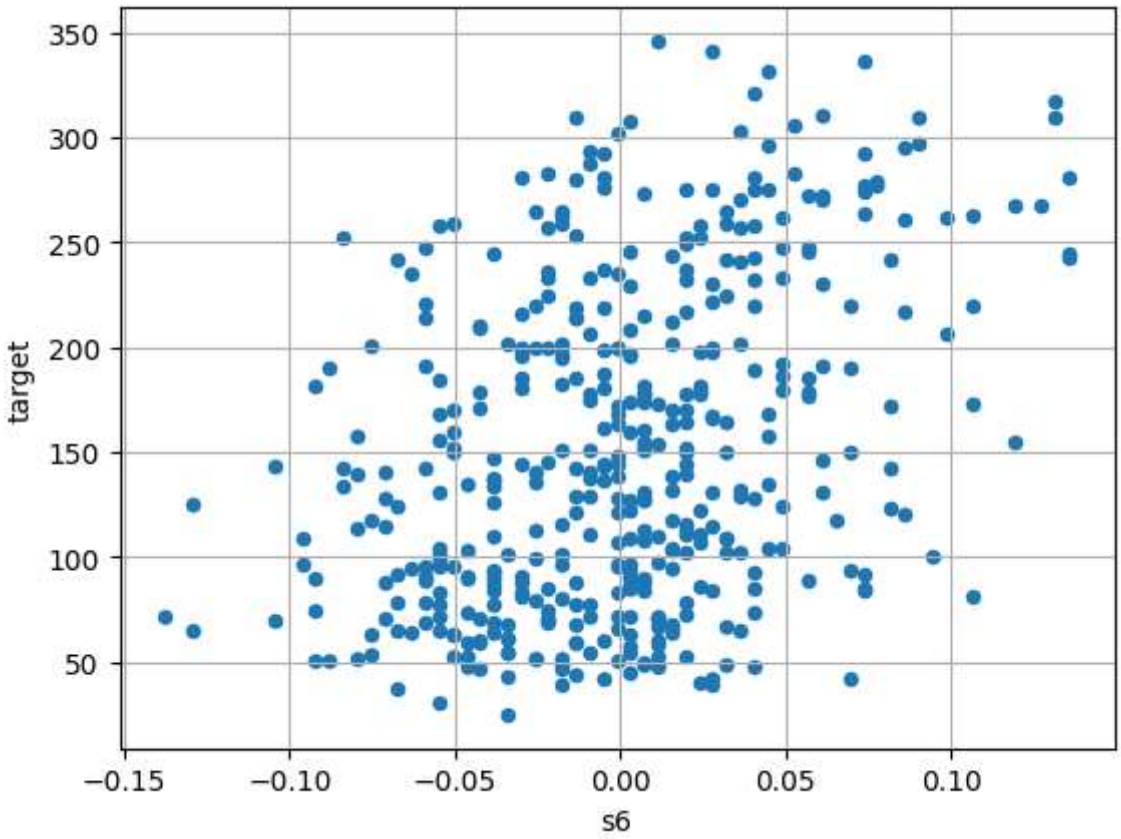

target
s1
350
300
250
200
150
100
50
−0.10
−0.05
0.00
0.05
0.10
0.15


target
350
300
250
200
150
100
50
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
s2


target
350
300
250
200
150
100
50
−0.10
−0.05
0.00
0.05
0.10
0.15
s3

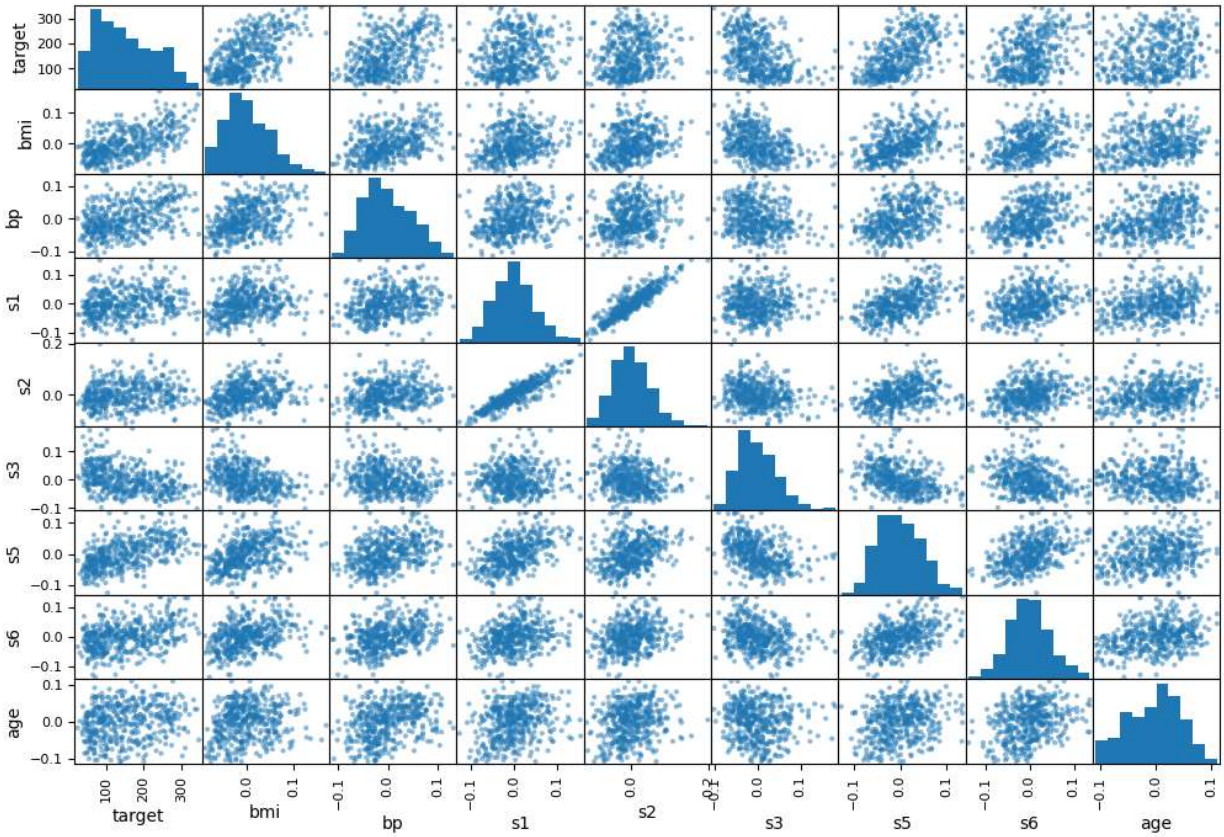

target
350
300
250
200
150
100
50
−0.05
0.00
0.05
0.10
0.15
s4


target
350
300
250
200
150
100
50
−0.10
−0.05
0.00
0.05
0.10
s5



In [ ]:

```
from pandas.plotting import scatter_matrix
attributes=['target','bmi','bp','s1','s2','s3','s5','s6','age']
scatter_matrix(Diabetes[attributes],figsize=(12,8))
plt.show()


#there is a high correlation between the two features s1, s2 (not good) to solve that
```



In [ ]:
```
correlation=Diabetes[attributes].corr()
correlation['target'].sort_values(ascending=False)
```

Out[ ]:
```
target    1.000000
bmi       0.586450
s5        0.565883
bp        0.441482
s6        0.382483
s1        0.212022
age       0.187889
s2        0.174054
s3       -0.394789
Name: target, dtype: float64
```

In [ ]:
```
correlation['s1'].sort_values(ascending=False)#there is ahigh correlation between the
#we can solve this by apllying PCA
```

Out[ ]:
```
s1        1.000000
s2        0.896663
s5        0.515503
s6        0.325717
age       0.260061
bmi       0.249777
bp        0.242464
target    0.212022
s3        0.051519
Name: s1, dtype: float64
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X=features
y=target
Xtrain,Xtest,ytrain,ytest=train_test_split( X,y, test_size=0.2,stratify=features["sex'
Xtrain.shape
```

Out[ ]:
```
(353, 10)
```

In [ ]:
```
from sklearn.decomposition import PCA
from sklearn.metrics.pairwise import rbf_kernel
#Apply PCA to solve the high correlation between features
pca = PCA(n_components=6)
Xtrain_num = pca.fit_transform(features[num_features])
```

In [ ]:
```
from sklearn.base import BaseEstimator,TransformerMixin
from sklearn .utils.validation import check_array,check_is_fitted
from sklearn.cluster import KMeans
#apply clustering similarity for the S4 feature
class ClusterSimilarity(BaseEstimator,TransformerMixin):
    def __init__(self,n_clusters=6,gamma=0.1,random_state=None):
        self.n_clusters=n_clusters
        self.gamma=gamma
        self.random_state=random_state

    def fit(self,X,sample_weight=None,y=None):
        X=check_array(X)
        self.kmeans_=KMeans(n_clusters=self.n_clusters,random_state=self.random_state)
        self.kmeans_.fit(X,sample_weight=sample_weight)
        self.n_features_in_ = X.shape[1]
        return self

    def transform(self,X):
        check_is_fitted(self)
        X=check_array(X)
        assert self.n_features_in_==X.shape[1]

        return rbf_kernel(X,self.kmeans_.cluster_centers_,gamma=self.gamma)

    def get_feature_names_out(self, names=None):
        return [f"Cluster {i} similarity" for i in range(self.n_clusters)]
```

In [ ]:
```
import numpy as np
cluster_simil = ClusterSimilarity(n_clusters=6, gamma=1., random_state=42)
similarities = cluster_simil.fit_transform(np.array(np.array(Diabetes['s4']).reshape(-
                                           sample_weight=target)
```

In [ ]:
```
from sklearn.pipeline import Pipeline,make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn.model_selection import cross_val_score
import numpy as np
from sklearn.compose import ColumnTransformer,make_column_selector
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
results=[]
reg={'LinearRegression':LinearRegression(),
    'KNeighborsRegressor':KNeighborsRegressor(),
    'DecisionTreeRegressor':DecisionTreeRegressor(random_state=42),
    'RandomForestRegressor':RandomForestRegressor(random_state=42,max_features=10,crit
    'LinearSVR':LinearSVR(random_state=42),
```

```
    'SVR':  SVR()}
num_pipline=make_pipeline(PCA(n_components=6))
s4_cluster_simil=make_pipeline(ClusterSimilarity(n_clusters=6,gamma=.1,random_state=42

sex_cluster_simil=make_pipeline(ClusterSimilarity(n_clusters=2,gamma=.1,random_state=4

preprocessing=ColumnTransformer([('num', num_pipline, num_features),
                                 ('sex',sex_cluster_simil,['sex']),
                                 ('s4',s4_cluster_simil,['s4'])])


data_prepared=preprocessing.fit_transform(Xtrain)
data_prepared.shape
```

Out[ ]:
```
(353, 14)
```

In [ ]:
```
for key in reg.keys():
    full_pipeline = Pipeline([
        ('preprocessing', preprocessing),
        ('reg',reg[key]),
    ])
    score=-cross_val_score(full_pipeline, Xtrain, ytrain, scoring="neg_root_mean_squar
    results.append((key,score.mean()/(np.max(target)-np.min(target))))
print('models scores:',results)
best_model_idx=np.array(results)[:,1].argmin()
print('best model:',results[best_model_idx][0],results[best_model_idx][1])
```

```
models scores: [('LinearRegression', 0.17655602446698188), ('KNeighborsRegressor', 0.
18574094830094015), ('DecisionTreeRegressor', 0.256972290335307), ('RandomForestRegre
ssor', 0.18142407731234306), ('LinearSVR', 0.2462042873874522), ('SVR', 0.24474757990
31947)]
best model: LinearRegression 0.17655602446698188
```

In [ ]:
```
#**Transformation Pipelines:**
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

full_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('linear_regression',LinearRegression(fit_intercept=False)),
])
full_pipeline.fit(Xtrain,ytrain)
#LinearRegression().get_params()
param_grid=[{'preprocessing__num__pca__n_components': [3,4,5,6],
             'preprocessing__s4__clustersimilarity__n_clusters':[2,3,4,5,6],
             'preprocessing__s4__clustersimilarity__gamma':[.01,.1]
           }]

grid_search=GridSearchCV(full_pipeline,param_grid,cv=10,scoring='neg_root_mean_squared
grid_search.fit(Xtrain,ytrain)
print('best estimator=',grid_search.best_estimator_)
print('best score=',grid_search.best_score_)
rmse=-cross_val_score(full_pipeline,Xtrain,ytrain,scoring='neg_root_mean_squared_error
rmse.mean().round(1)
rmse_percentage=rmse.mean().round(1)/(target.max()-target.min())
print('rmse percentage:',rmse_percentage)
```

```
best estimator= Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('pca',
                                                                   PCA(n_components=
6))]),
                                                  ['age', 'bmi', 'bp', 's1',
                                                   's2', 's3', 's5', 's6']),
                                                 ('sex',
                                                  Pipeline(steps=[('clustersimilarit
y',
                                                                   ClusterSimilarity
(n_clusters=2,

random_state=42))]),
                                                  ['sex']),
                                                 ('s4',
                                                  Pipeline(steps=[('clustersimilarit
y',
                                                                   ClusterSimilarity
(gamma=0.01,

n_clusters=2,

random_state=42))]),
                                                  ['s4'])])),
                ('linear_regression', LinearRegression(fit_intercept=False))])
best score= -54.90687388065811
rmse percentage: 0.17133956386292834
```

In [ ]:
```
preprocessing.get_feature_names_out()
```

Out[ ]:
```
array(['num__pca0', 'num__pca1', 'num__pca2', 'num__pca3', 'num__pca4',
       'num__pca5', 'sex__Cluster 0 similarity',
       'sex__Cluster 1 similarity', 's4__Cluster 0 similarity',
       's4__Cluster 1 similarity', 's4__Cluster 2 similarity',
       's4__Cluster 3 similarity', 's4__Cluster 4 similarity',
       's4__Cluster 5 similarity'], dtype=object)
```

In [ ]:
```
from sklearn.metrics import mean_squared_error

final_model=grid_search.best_estimator_

final_prediction=final_model.predict(Xtest)
final_rmse=mean_squared_error(y_pred=final_prediction,y_true=ytest,squared=False)
final_rmse
final_rmse_percentage=final_rmse.round(1)/(target.max()-target.min())
print('final rmse percentage:',final_rmse_percentage)
```

```
final rmse percentage: 0.1791277258566978
```

In [ ]:
```
#Save the final Model:
import joblib

joblib.dump(final_model,'final_model.pkl')
```

Out[ ]:
```
['final_model.pkl']
```

In [ ]:
```
import joblib
from sklearn.cluster import KMeans
from sklearn.base import BaseEstimator, TransformerMixin
```

```
from sklearn.metrics.pairwise import rbf_kernel


def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

final_model=joblib.load('final_model.pkl')

new_data=features.iloc[:5]
predictions=final_model.predict(new_data)
predictions
```

Out[ ]:
```
array([199.91999316,  61.47775717, 169.42274505, 157.28589196,
       130.89240849])
```

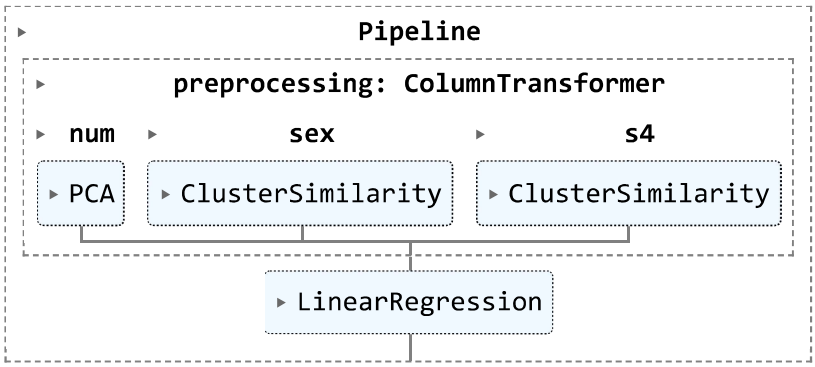
In [ ]:
```
target.iloc[:5]
```

Out[ ]:
```
0    151.0
1     75.0
2    141.0
3    206.0
4    135.0
Name: target, dtype: float64
```

In [ ]:
```
#the end :)
```

In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint


rnd_param={
    'preprocessing__num__pca__n_components': randint(low=2,high=6),
    'preprocessing__s4__clustersimilarity__n_clusters':randint(low=2,high=6),
    'preprocessing__s4__clustersimilarity__gamma':randint(low=.01,high=1)
}
rnd_search= RandomizedSearchCV(full_pipeline,param_distributions=rnd_param,n_iter=10,c
                               scoring='neg_root_mean_squared_error',random_state=42)
rnd_search.fit(Xtrain , ytrain)

rnd_search.best_estimator_
```

Out[ ]:



In [ ]:
```
rnd_search.best_score_
```

Out[ ]:
```
-55.390825497650304
```

In [ ]:
```
#**Transformation Pipelines:**
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LassoCV
import xgboost as xgb

full_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('XGB_regression',xgb.XGBRegressor(verbosity=0)),
])
full_pipeline.fit(Xtrain,ytrain)
#LinearRegression().get_params()
param_grid=[{'preprocessing__num__pca__n_components': [3,4,5,6],
             'preprocessing__s4__clustersimilarity__n_clusters':[2,3,4,5,6,7],
             'preprocessing__s4__clustersimilarity__gamma':[.01,.1]
            }
            ]

grid_search=GridSearchCV(full_pipeline,param_grid,cv=10,scoring='neg_root_mean_squared
grid_search.fit(Xtrain,ytrain)
print('best estimator=',grid_search.best_estimator_)
print('best score=',grid_search.best_score_)
rmse=-cross_val_score(full_pipeline,Xtrain,ytrain,scoring='neg_root_mean_squared_error
rmse.mean().round(1)
rmse_percentage=rmse.mean().round(1)/(target.max()-target.min())
print('rmse percentage:',rmse_percentage)
```

```
best estimator= Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('pca',
                                                                   PCA(n_components=
6))]),
                                                  ['age', 'bmi', 'bp', 's1',
                                                   's2', 's3', 's5', 's6']),
                                                 ('sex',
                                                  Pipeline(steps=[('clustersimilarit
y',
                                                                   ClusterSimilarity
(n_clusters=2,

random_state=42))]),
                                                  ['sex']),
                                                 ('s4',
                                                  Pipeline(steps=[('clustersimilarit
y',
                                                                   ClusterSimilarity
(gamma=0.01,

n_clusters=...
                              feature_types=None, gamma=0, gpu_id=-1,
                              grow_policy='depthwise', importance_type=None,
                              interaction_constraints='',
                              learning_rate=0.300000012, max_bin=256,
                              max_cat_threshold=64, max_cat_to_onehot=4,
                              max_delta_step=0, max_depth=6, max_leaves=0,
                              min_child_weight=1, missing=nan,
                              monotone_constraints='()', n_estimators=100,
                              n_jobs=0, num_parallel_tree=1, predictor='auto',
                              random_state=0, ...))])
best score= -60.77136476661784
rmse percentage: 0.19158878504672897
```

In [ ]:
```
grid_search.best_params_
```

Out[ ]:
```
{'preprocessing__num__pca__n_components': 6,
 'preprocessing__s4__clustersimilarity__gamma': 0.01,
 'preprocessing__s4__clustersimilarity__n_clusters': 3}
```

In [ ]:
```
param_grid=[{'preprocessing__num__pca__n_components': [6],
             'preprocessing__s4__clustersimilarity__n_clusters':[5],
             'preprocessing__s4__clustersimilarity__gamma':[.01]
            }
            ]

grid_search=GridSearchCV(full_pipeline,param_grid,cv=10,scoring='neg_root_mean_squared
grid_search.fit(features,target)
print('best estimator=',grid_search.best_estimator_)
print('best score=',grid_search.best_score_)
rmse=-cross_val_score(full_pipeline,features,target,scoring='neg_root_mean_squared_err
rmse.mean().round(1)
rmse_percentage=rmse.mean().round(1)/(target.max()-target.min())
print('rmse percentage:',rmse_percentage)
```

```
best estimator= Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('pca',
                                                                   PCA(n_components=
4))]),
                                                  ['age', 'bmi', 'bp', 's1',
                                                   's2', 's3', 's5', 's6']),
                                                 ('sex',
                                                  Pipeline(steps=[('clustersimilarit
y',
                                                                   ClusterSimilarity
(n_clusters=2,

random_state=42))]),
                                                  ['sex']),
                                                 ('s4',
                                                  Pipeline(steps=[('clustersimilarit
y',
                                                                   ClusterSimilarity
(gamma=0.01,

random_stat...
                              feature_types=None, gamma=0, gpu_id=-1,
                              grow_policy='depthwise', importance_type=None,
                              interaction_constraints='',
                              learning_rate=0.300000012, max_bin=256,
                              max_cat_threshold=64, max_cat_to_onehot=4,
                              max_delta_step=0, max_depth=6, max_leaves=0,
                              min_child_weight=1, missing=nan,
                              monotone_constraints='()', n_estimators=100,
                              n_jobs=0, num_parallel_tree=1, predictor='auto',
                              random_state=0, ...))])
best score= -61.73767213592574
rmse percentage: 0.19563862928348907
```

In [ ]:

```python
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import VotingRegressor

#**Transformation Pipelines:**
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LassoCV
import xgboost as xgb
results=[]
reg={'LinearRegression':LinearRegression(),
    'GradientBoostingRegressor':GradientBoostingRegressor(random_state=42),
    'RandomForestRegressor':RandomForestRegressor(random_state=42),
    }

for key in reg.keys():
    full_pipeline = Pipeline([
        ('preprocessing', preprocessing),
        ('reg',reg[key]),
    ])
    reg[key].fit(Xtrain,ytrain)
```

```
ereg = VotingRegressor([("gb", reg['GradientBoostingRegressor']), ("rf", reg['RandomFo
ereg.fit(Xtrain, ytrain)

#    score=-cross_val_score(full_pipeline, Xtrain, ytrain, scoring="neg_root_mean_squa
#    results.append((key,score.mean().round(1)))
#print('models scores:',results)
#best_model_idx=np.array(results)[:,1].argmin()
#print('best model:',results[best_model_idx][0],results[best_model_idx][1].round(1))
#
#
#param_grid=[{'preprocessing__num__pca__n_components': [3,4,5,6],
#              'preprocessing__s4_cluster__clustersimilarity__n_clusters':[2,3,4,5,6,7
#              'preprocessing__s4_cluster__clustersimilarity__gamma':[.01,.1]
#            }
#            ]
#
#grid_search=GridSearchCV(full_pipeline,param_grid,cv=10,scoring='neg_root_mean_square
#grid_search.fit(Xtrain,ytrain)
#print('best estimator=',grid_search.best_estimator_)
#print('best score=',grid_search.best_score_)
mse=-cross_val_score(ereg,Xtest,ytest,scoring='neg_root_mean_squared_error',cv=10)
rmse.mean().round(1)
rmse_percentage=rmse.mean().round(1)/(target.max()-target.min())
print('rmse percentage:',rmse_percentage)
```

```
rmse percentage: 0.19563862928348907
```

In [ ]:
```
rmse.mean()
```

Out[ ]:
```
62.84972489541224
```

In [ ]:
```
#after trying altenative models we conclude that the best model to fit the data is :
#LinearRegression with RMSE =.17
```