In [16]:
```
import numpy as np
import pandas as pd
import sklearn.datasets
import seaborn as sns
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
```

In [5]:
```
df=pd.DataFrame(data=diabetes.data,columns=diabetes.feature_names)          # loading dataset
df
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641
...	...	...	...	...	...	...	...	...	...	...
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.007207
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018114	0.044485
439	0.041708	0.050680	-0.015906	0.017293	-0.037344	-0.013840	-0.024993	-0.011080	-0.046883	0.015491
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560	0.044529	-0.025930
441	-0.045472	-0.044642	-0.073030	-0.081413	0.083740	0.027809	0.173816	-0.039493	-0.004222	0.003064

442 rows × 10 columns

In [ ]:
```
#The Sklearn Diabetes Dataset include following attributes:

age: Age in years
sex: Gender of the patient
bmi: Body mass index
bp: Average blood pressure
s1: Total serum cholesterol (tc)
s2: Low-density lipoproteins (ldl)
s3: High-density lipoproteins (hdl)
s4: Total cholesterol / HDL (tch)
s5: Possibly log of serum triglycerides level (ltg)
s6: Blood sugar level (glu)
Number of Instances: 442
Number of Attributes: The first 10 columns are numeric predictive values.
Target: Column 11 represents a quantitative measure of disease progression one year after baseline.
```

Data Pre processing

In [7]:
```
df['target'] = diabetes.target
```

In [8]:
```
df.head()
```

Out[8]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0

In [9]:
```
df.shape
```

Out[9]:
```
(442, 11)
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 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
 10  target  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
age       0
sex       0
bmi       0
bp        0
s1        0
s2        0
s3        0
s4        0
s5        0
s6        0
target    0
dtype: int64
```

In [12]:
```
df.select_dtypes(include="number").columns
```

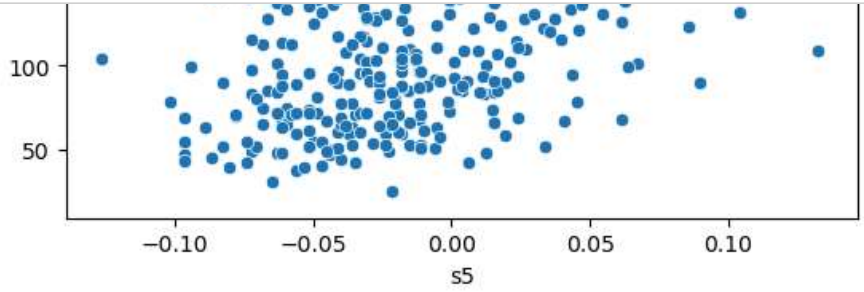
Out[12]:
```
Index(['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6',
       'target'],
      dtype='object')
```

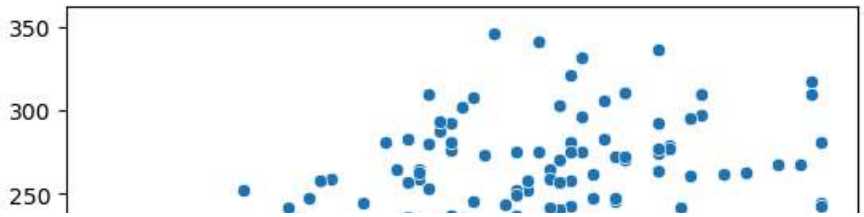
Visual representation

identifyig the relationship between target values and other features

In [32]:
```
import matplotlib.pyplot as plt
```

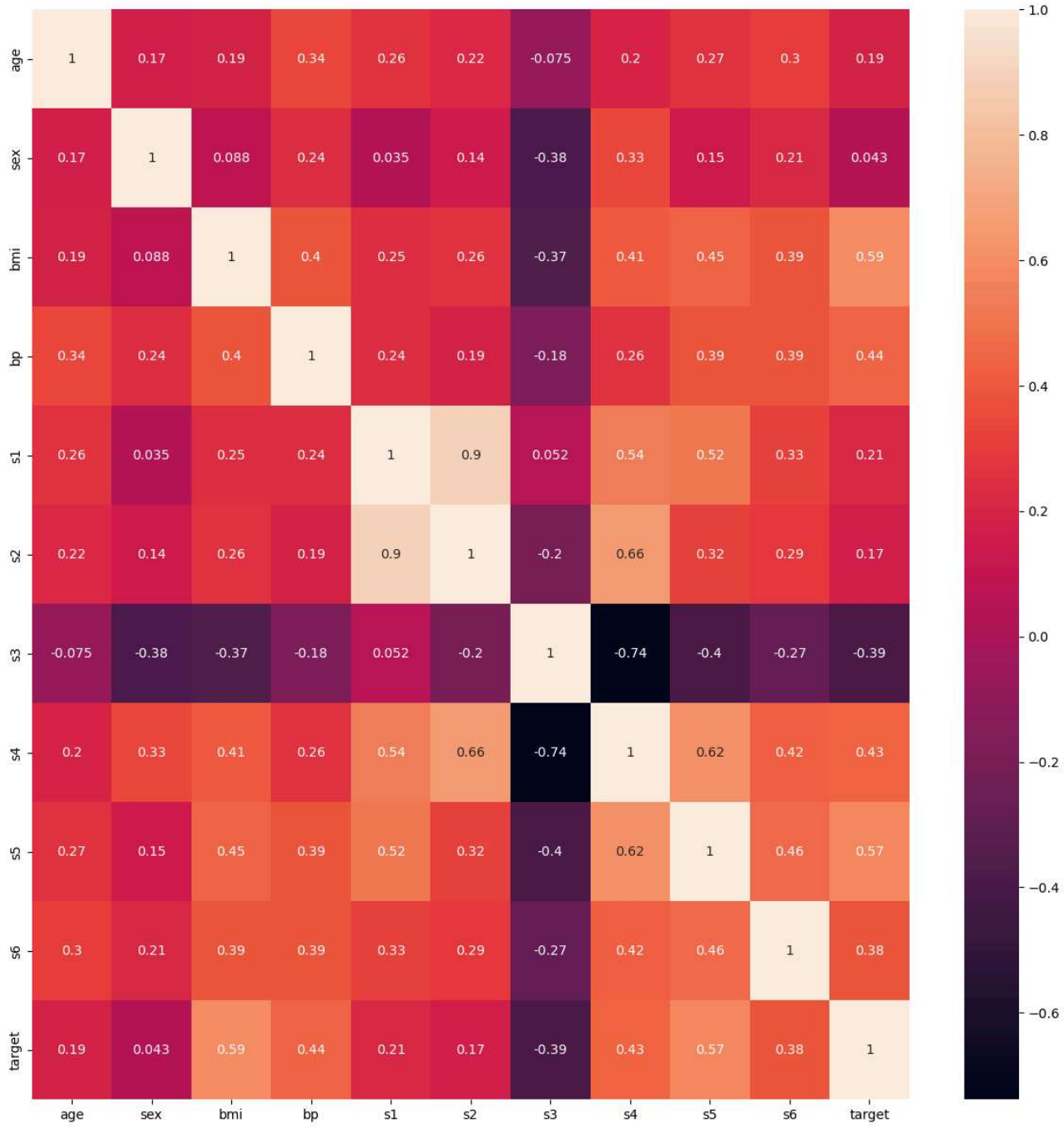
In [33]:
```
for i in ['age','sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']:
    sns.scatterplot(data=df, x=i, y='target')
    plt.show()
```





In [34]:
```
heat_map=df.select_dtypes(include="number").corr()
plt.figure(figsize=(15,15))
sns.heatmap(heat_map,annot=True)
```

Out[34]: <Axes: >



In [ ]:
```
# Corelation found using heat map
```

OUTLIER DETECTION

data has certain amount of outliers that have to be treated for better performance

In [35]:
```
from scipy import stats
```

In [36]:
```
def detect_outliers_zscore(df):
    outliers = {}
    for column in df.select_dtypes(include=['float64', 'int64']).columns:
        z_scores = stats.zscore(df[column])
        outlier_indices = df[(z_scores > 3) | (z_scores < -3)].index.tolist()
        outliers[column] = outlier_indices
    return outliers
# Find and print outliers
outliers = detect_outliers_zscore(df)
print(outliers)
valid_outliers = [index for index in outliers if index in df.index]
```

```
{'age': [], 'sex': [], 'bmi': [256, 367], 'bp': [], 's1': [123, 230], 's2': [123, 230], 's3': [58, 260,
261, 269, 441], 's4': [123, 216, 322, 336], 's5': [], 's6': [], 'target': []}
```

In [37]:
```
df_cleaned = df.drop(index=valid_outliers).reset_index(drop=True)
```

In [59]:
```
x=df_cleaned.drop(columns="target")
```

In [39]:
```
from sklearn.model_selection import train_test_split
```

In [40]:
```
y=df["target"]
```

Splittig dataset

data set need to be split into test data and train data for model training purposes

In [60]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Feature scaling

In [46]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [47]:
```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [48]:
```
from keras.models import Sequential
from keras.layers import Dense
```

model impelemtation

Artificial Neural Networks (ANNs) are computational models inspired by the way biological neural networks in the human brain function. They are used to recognize patterns and make predictions based on input data.
Input Layer: The first layer that receives input data.
Hidden Layers: Intermediate layers where the network processes inputs. A network can have one or multiple hidden layers.
Output Layer: The final layer that produces the output.

Activation Functions: Functions applied to the output of each neuron. Common activation functions include:
Sigmoid: Outputs values between 0 and 1.
ReLU (Rectified Linear Unit): Outputs the input directly if positive; otherwise, it outputs zero.
Tanh: Outputs values between -1 and 1.

In [49]:
```
model = Sequential()
```

In [50]:
```
model.add(Dense(units=11,activation="relu"))# first layer
model.add(Dense(units=7,activation="relu"))# input layer
model.add(Dense(units=5,activation="relu"))# second layer
model.add(Dense(1, activation='softmax'))# output layer
```

In [51]:
```
model.compile(optimizer='adam', loss='mean_squared_error')
```

In [52]:
```
# training the model
model_history = model.fit(x_train, y_train, epochs=50, batch_size=10, validation_split=0.2)
```

```
Epoch 1/50

C:\Users\user\anaconda3\Lib\site-packages\keras\src\ops\nn.py:545: UserWarning: You are using a softm
ax over axis -1 of a tensor of shape (None, 1). This axis has size 1. The softmax operation will alwa
ys return the value 1, which is likely not what you intended. Did you mean to use a sigmoid instead?
  warnings.warn(

29/29 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 30986.3516 - val_loss: 22223.7051
Epoch 2/50
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 31522.8145 - val_loss: 22223.7051
Epoch 3/50
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 30744.8555 - val_loss: 22223.7051
Epoch 4/50
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 32078.5801 - val_loss: 22223.7051
Epoch 5/50
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 31215.5039 - val_loss: 22223.7051
Epoch 6/50
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 29864.7090 - val_loss: 22223.7051
Epoch 7/50
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 31472.7988 - val loss: 22223.7051
```

In [53]:
```
loss=model.evaluate(x_test,y_test)
print("Loss Functions:",loss)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 28042.0352
Loss Functions: 26258.033203125
```

In [54]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 11)	121
dense_1 (Dense)	(None, 7)	84
dense_2 (Dense)	(None, 5)	40
dense_3 (Dense)	(None, 1)	6

Total params: 755 (2.95 KB)

Trainable params: 251 (1004.00 B)

Non-trainable params: 0 (0.00 B)

Optimizer params: 504 (1.97 KB)

In [55]:
```
predict=model.predict(x_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step

C:\Users\user\anaconda3\Lib\site-packages\keras\src\ops\nn.py:545: UserWarning: You are using a softmax
over axis -1 of a tensor of shape (32, 1). This axis has size 1. The softmax operation will always retur
n the value 1, which is likely not what you intended. Did you mean to use a sigmoid instead?
  warnings.warn(
C:\Users\user\anaconda3\Lib\site-packages\keras\src\ops\nn.py:545: UserWarning: You are using a softmax
over axis -1 of a tensor of shape (None, 1). This axis has size 1. The softmax operation will always ret
urn the value 1, which is likely not what you intended. Did you mean to use a sigmoid instead?
  warnings.warn(
```

In [56]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

In [57]:
```
mse=mean_squared_error(y_test, predict)
mse
```

Out[57]:
```
26258.03370786517
```

In [58]:
```
r2=r2_score(y_test, predict)
r2
```

Out[58]:
```
-3.9560742980261
```

In [62]:
```
from sklearn.linear_model import LinearRegression
```

In [63]:
```
model_1= LinearRegression()
```

In [64]:
```
model.fit(x_train, y_train)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 28002.2441
```

Out[64]:
```
<keras.src.callbacks.history.History at 0x1e55b61c7d0>
```

In [66]:
```
y_pred = model.predict(x_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

In [68]:
```
mse = mean_squared_error(y_test, y_pred)
mse
```

Out[68]:
```
26258.03370786517
```

In [69]:
```
model_1= Sequential()
```

In [70]:
```
model_1.add(Dense(units=32,activation="tanh"))
model_1.add(Dense(units=16,activation="tanh"))
model_1.add(Dense(1))
```

In [71]:
```
model_1.compile(optimizer='adam', loss='mean_squared_error')
```

In [72]:
```
model_history1 = model.fit(x_train, y_train, epochs=100, batch_size=10, validation_split=0.2)
```

```
Epoch 39/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 30917.3477 - val_loss: 22223.7051
Epoch 40/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 30154.4336 - val_loss: 22223.7051
Epoch 41/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 31184.4277 - val_loss: 22223.7051
Epoch 42/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 30065.2109 - val_loss: 22223.7051
Epoch 43/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 32010.6504 - val_loss: 22223.7051
Epoch 44/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 29347.2246 - val_loss: 22223.7051
Epoch 45/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 29562.6875 - val_loss: 22223.7051
Epoch 46/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 33937.8789 - val_loss: 22223.7051
Epoch 47/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 30630.4668 - val_loss: 22223.7051
Epoch 48/100
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 31027.3789 - val loss: 22223.7051
```

In [73]:
```
predict_2=model_1.predict(x_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

In [74]:
```
mse=mean_squared_error(y_test, predict_2)
mse
```

Out[74]:
```
26546.3101409224
```

In [82]:
```
accuracy = model.evaluate(x_test, y_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 28042.0352
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(model_history.history['accuracy'], label='Train Accuracy')
plt.plot(model_history.history['accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```