Fiche TP 01:Linear Regression

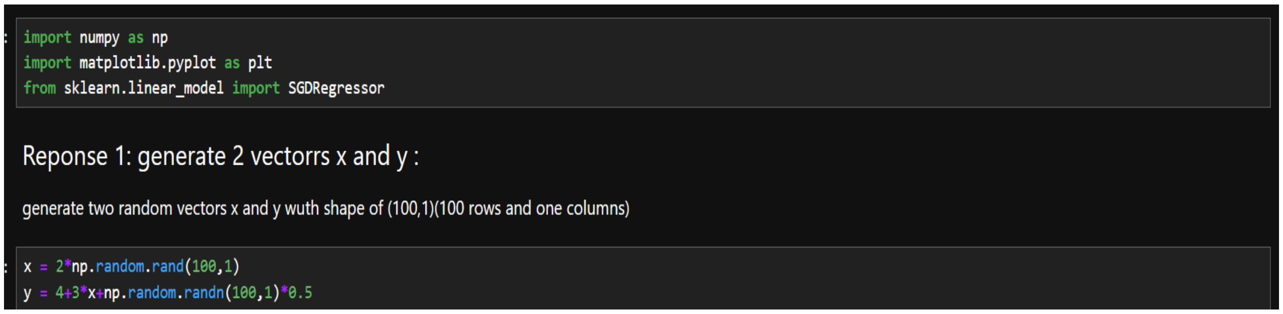
Nom: Boudelal Abderahmane Amine & Nahoui Seif

Link colab to code exo1:

[TPO1_EXO 1.ipynb - Colab]

Exercice 1 :

Reponse1 :



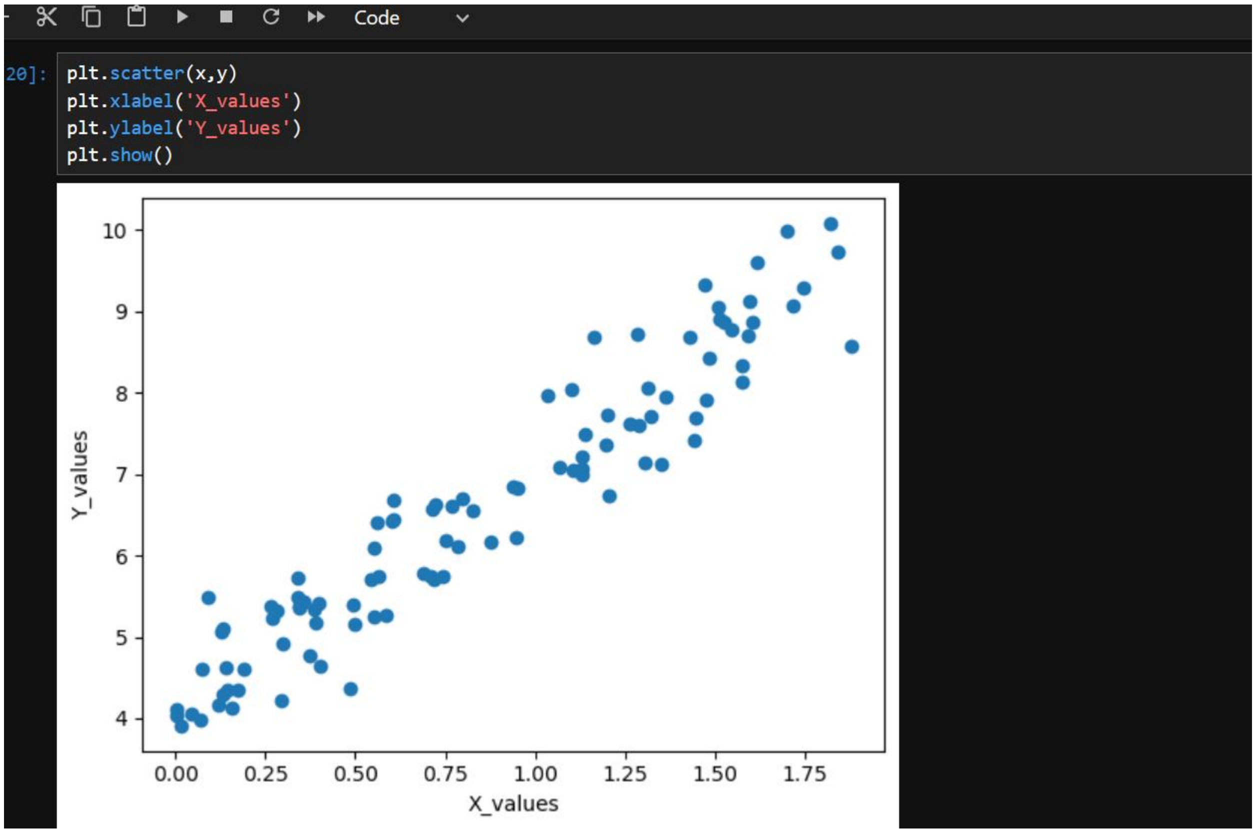
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
```

Reponse 1: generate 2 vectorrs x and y :

generate two random vectors x and y wuth shape of (100,1)(100 rows and one columns)

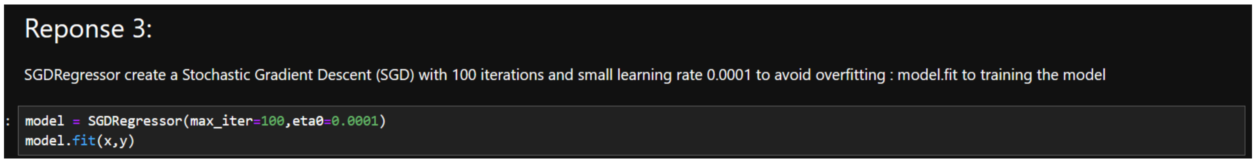
```
x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.randn(100,1)*0.5
```

Reponse 2:



```
plt.scatter(x,y)
plt.xlabel('X_values')
plt.ylabel('Y_values')
plt.show()
```

Reponse 3 :

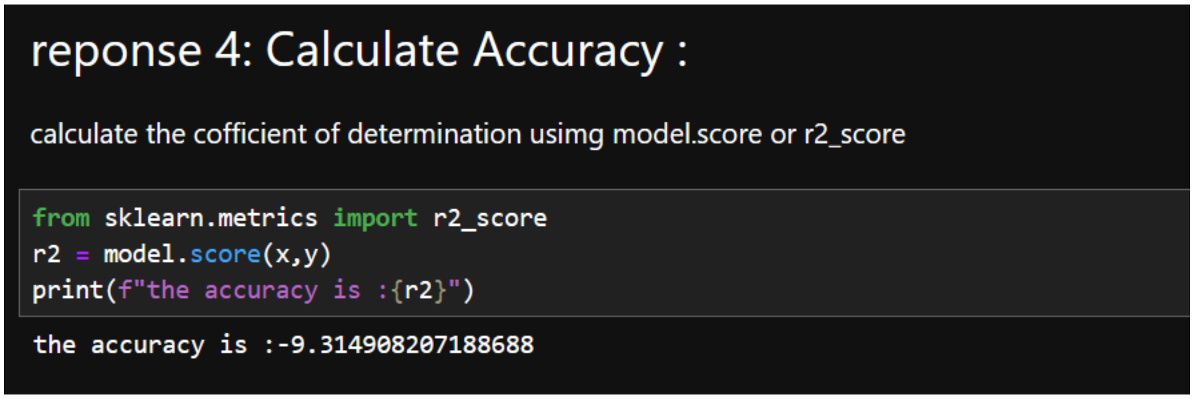


Reponse 3:

SGDRegressor create a Stochastic Gradient Descent (SGD) with 100 iterations and small learning rate 0.0001 to avoid overfitting : model.fit to training the model

```
model = SGDRegressor(max_iter=100,eta0=0.0001)
model.fit(x,y)
```

Reponse 4 :



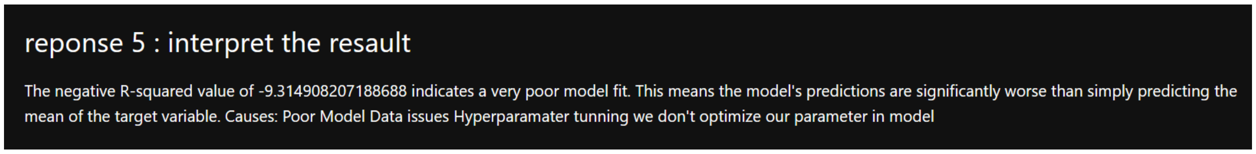
reponse 4: Calculate Accuracy :

calculate the cofficient of determination usimg model.score or r2_score

```
from sklearn.metrics import r2_score
r2 = model.score(x,y)
print(f"the accuracy is :{r2}")
```

```
the accuracy is :-9.314908207188688
```

Reponse 5:

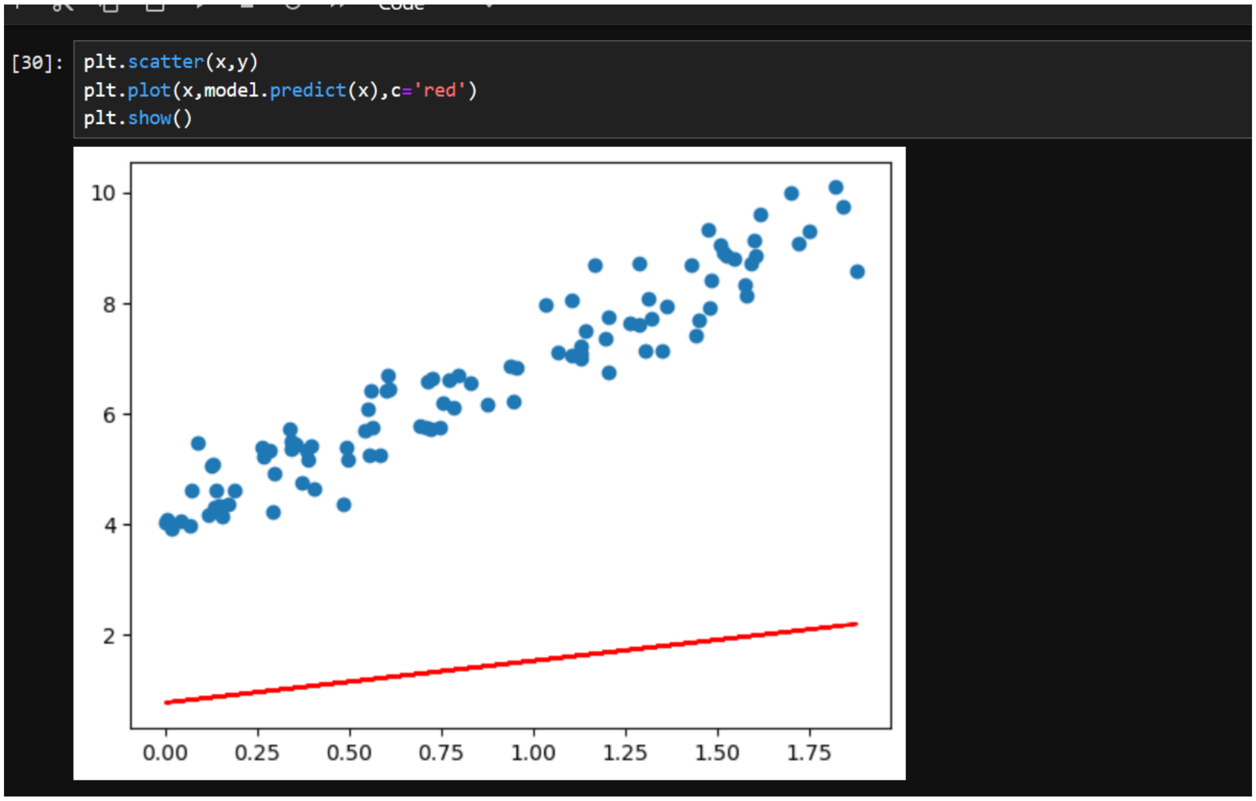


reponse 5 : interpret the resault

The negative R-squared value of -9.314908207188688 indicates a very poor model fit. This means the model's predictions are significantly worse than simply predicting the mean of the target variable. Causes: Poor Model Data issues Hyperparamater tunning we don't optimize our parameter in model

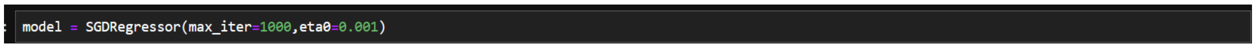
Reponse 6 :

The graph indicate an underfitting



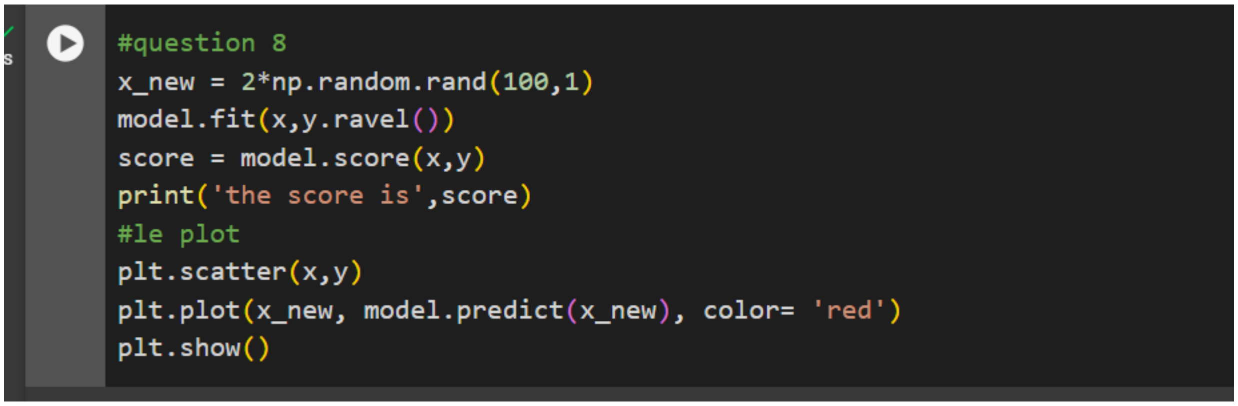
```
[30]: plt.scatter(x,y)
      plt.plot(x,model.predict(x),c='red')
      plt.show()
```

Reponse 7 :

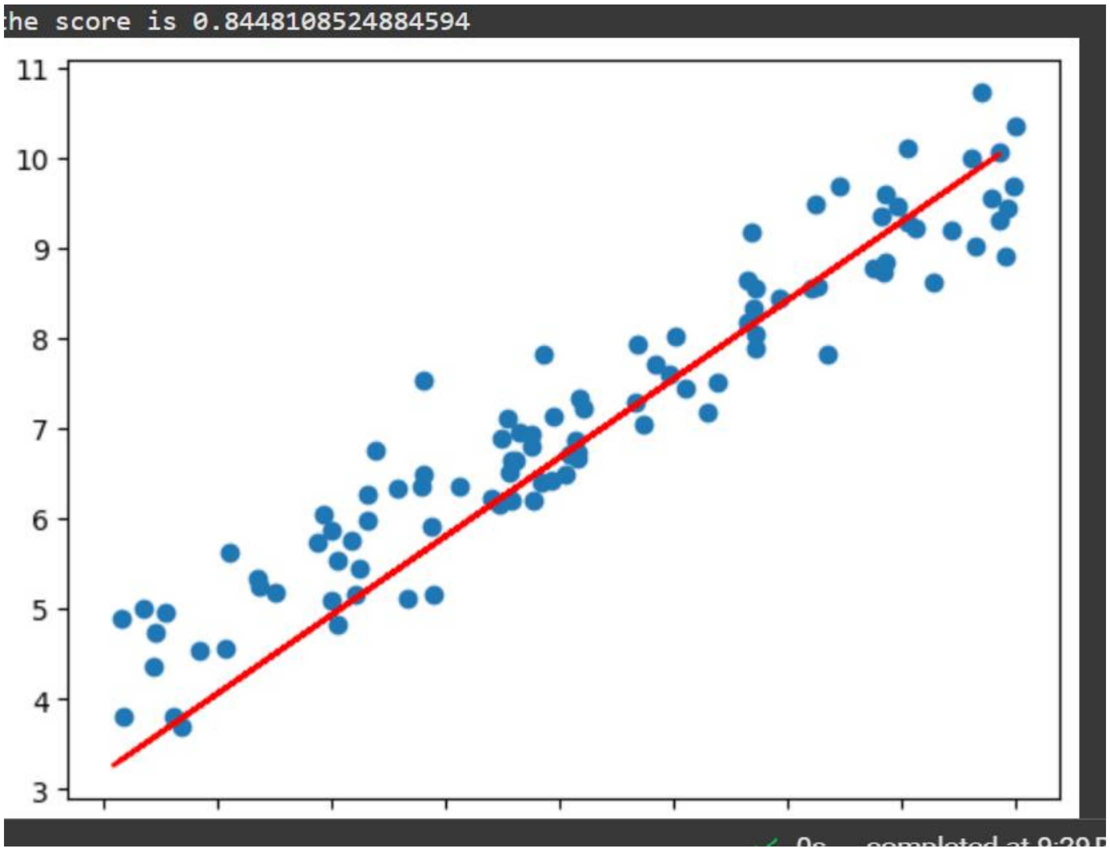


```
model = SGDRegressor(max_iter=1000,eta0=0.001)
```

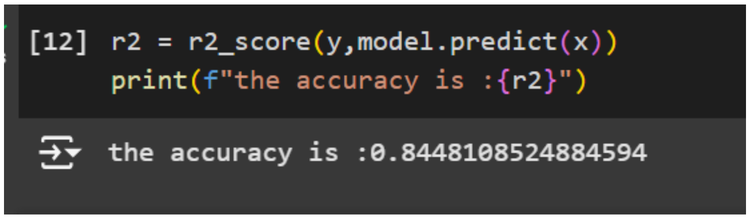
Reponse 8 :



```
#question 8
x_new = 2*np.random.rand(100,1)
model.fit(x,y.ravel())
score = model.score(x,y)
print('the score is',score)
#le plot
plt.scatter(x,y)
plt.plot(x_new, model.predict(x_new), color= 'red')
plt.show()
```

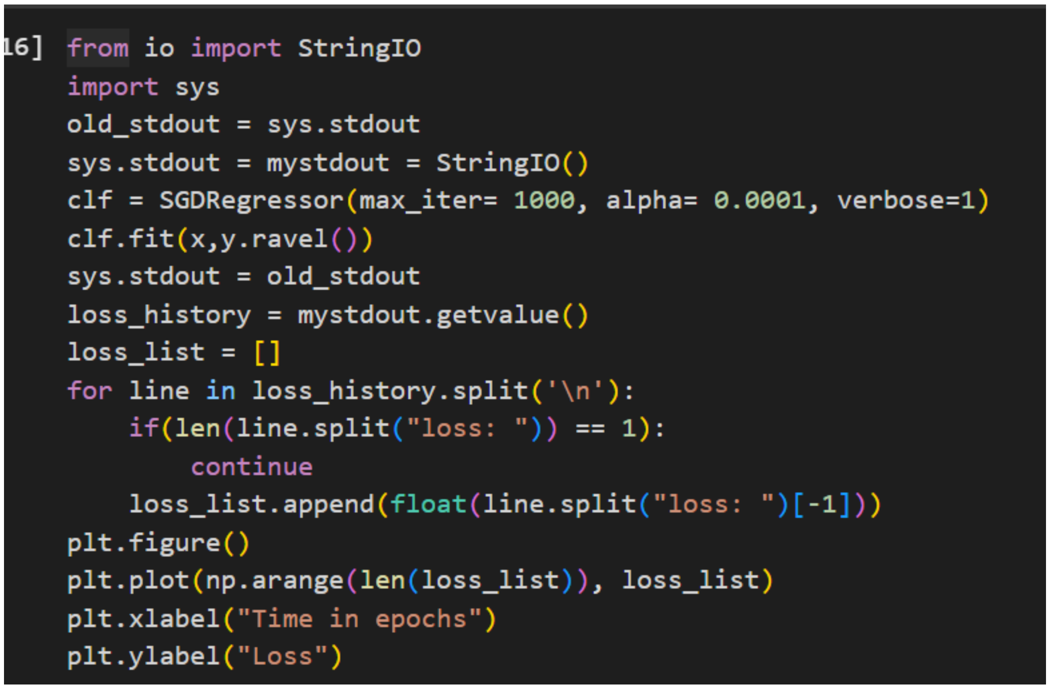


```
he score is 0.8448108524884594
```



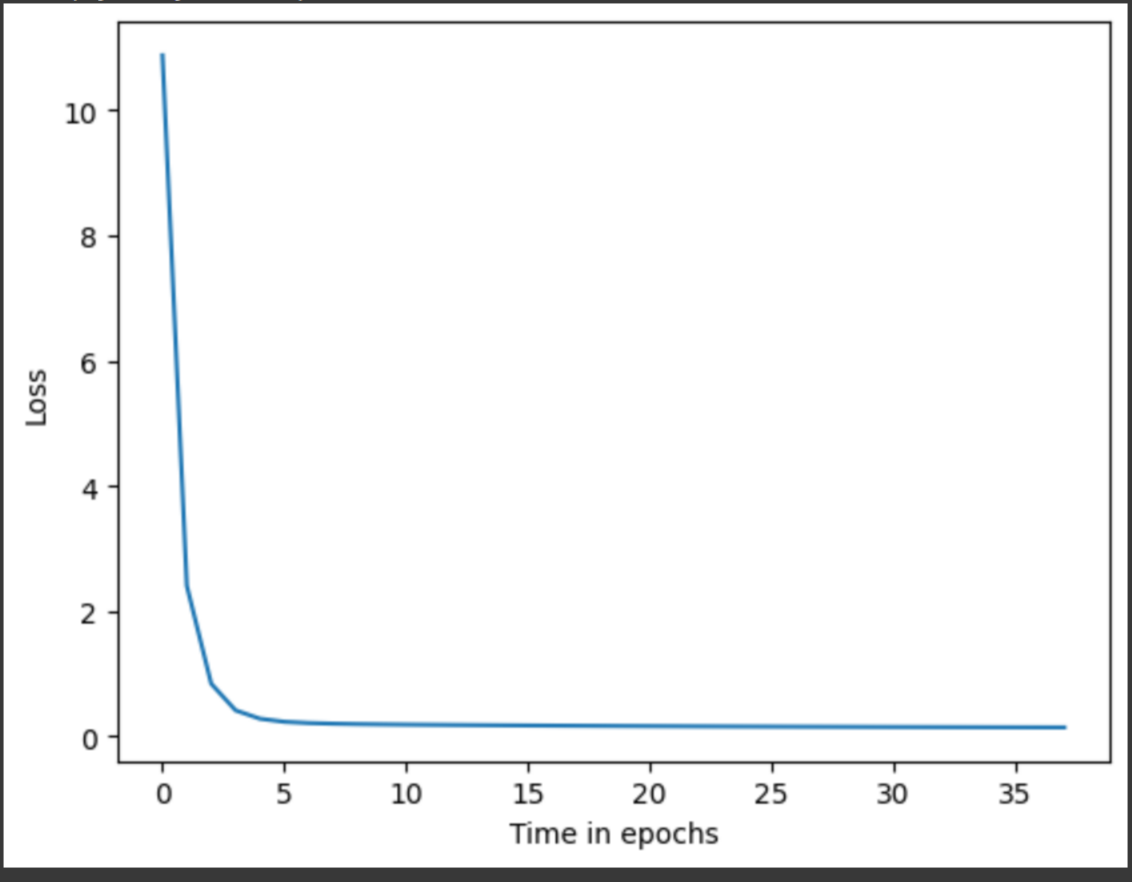
```
[12] r2 = r2_score(y,model.predict(x))
     print(f"the accuracy is :{r2}")

the accuracy is :0.8448108524884594
```

Reponse 9



```
16]  from io import StringIO
     import sys
     old_stdout = sys.stdout
     sys.stdout = mystdout = StringIO()
     clf = SGDRegressor(max_iter= 1000, alpha= 0.0001, verbose=1)
     clf.fit(x,y.ravel())
     sys.stdout = old_stdout
     loss_history = mystdout.getvalue()
     loss_list = []
     for line in loss_history.split('\n'):
         if(len(line.split("loss: ")) == 1):
             continue
         loss_list.append(float(line.split("loss: ")[-1]))
     plt.figure()
     plt.plot(np.arange(len(loss_list)), loss_list)
     plt.xlabel("Time in epochs")
     plt.ylabel("Loss")
```

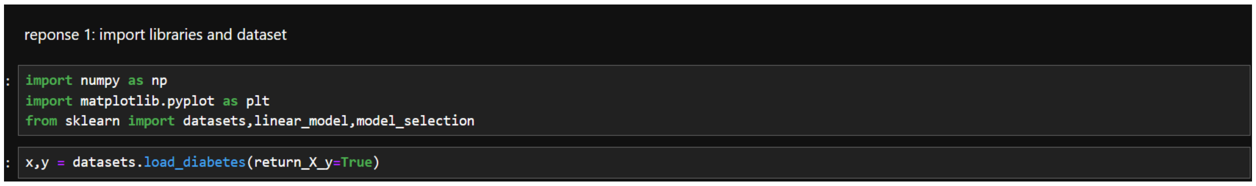
```
Text(0, 0.5, 'Loss')
```



Exercice 2 :

The link to the colab: TP01_Exo02.ipynb - Colab

Reponse 1 :

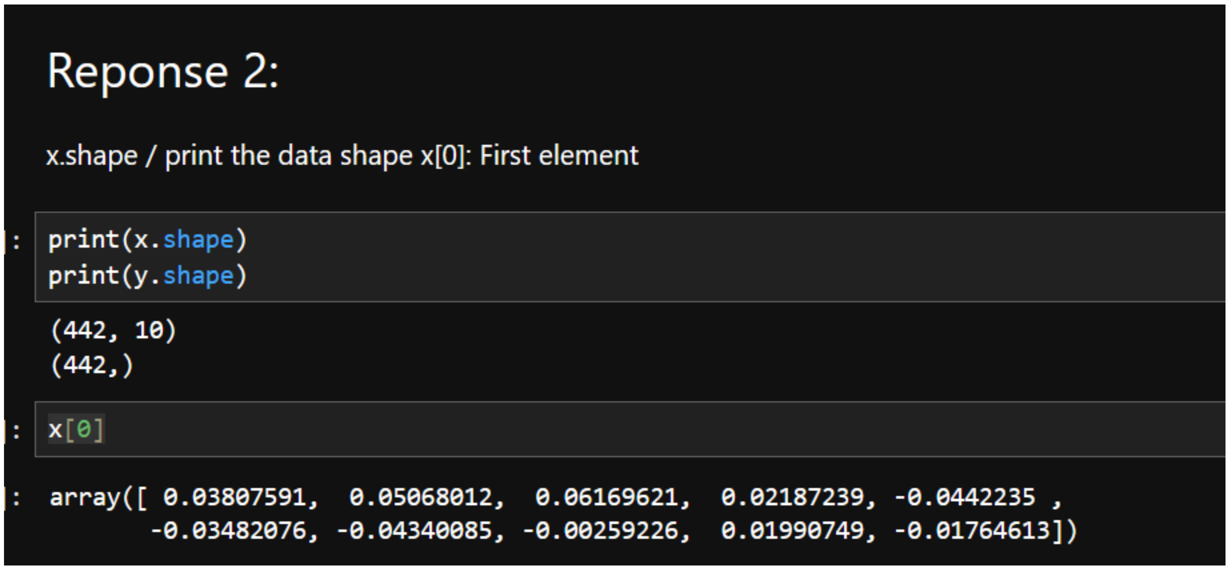


reponse 1: import libraries and dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model,model_selection
```

```
x,y = datasets.load_diabetes(return_X_y=True)
```

REPONSE 2



Reponse 2:

x.shape / print the data shape x[0]: First element

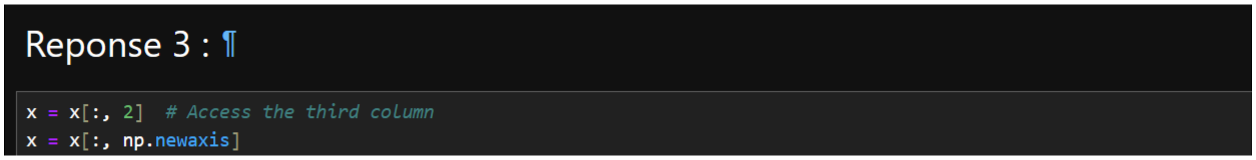
```
print(x.shape)
print(y.shape)
```

```
(442, 10)
(442,)
```

```
x[0]
```

```
array([ 0.03807591,  0.05068012,  0.06169621,  0.02187239, -0.0442235 ,
       -0.03482076, -0.04340085, -0.00259226,  0.01990749, -0.01764613])
```

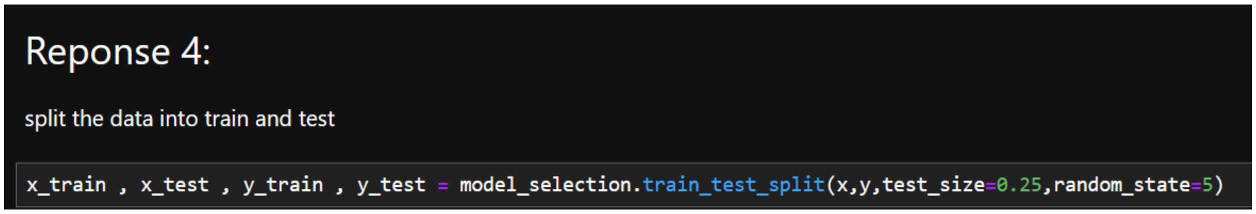
Réponse 3 :



Reponse 3 : ¶

```
x = x[:, 2]  # Access the third column
x = x[:, np.newaxis]
```

Reponse4 :

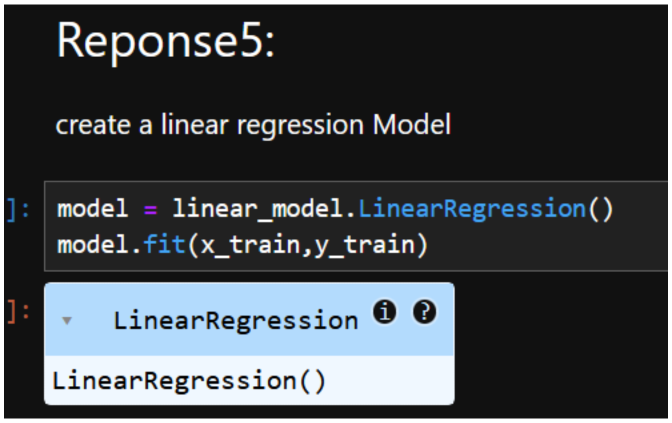


Reponse 4:

split the data into train and test

```
x_train , x_test , y_train , y_test = model_selection.train_test_split(x,y,test_size=0.25,random_state=5)
```

Reponse 5 :

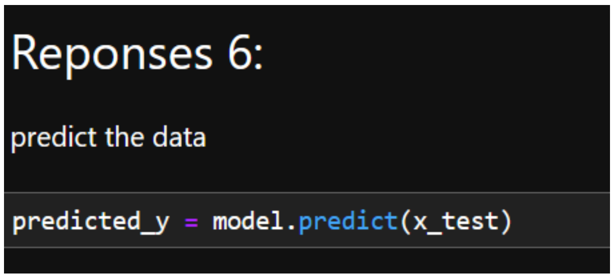

Reponse5:

create a linear regression Model

```
]: model = linear_model.LinearRegression()
   model.fit(x_train,y_train)
```

```
]: ▾ LinearRegression
   LinearRegression()
```

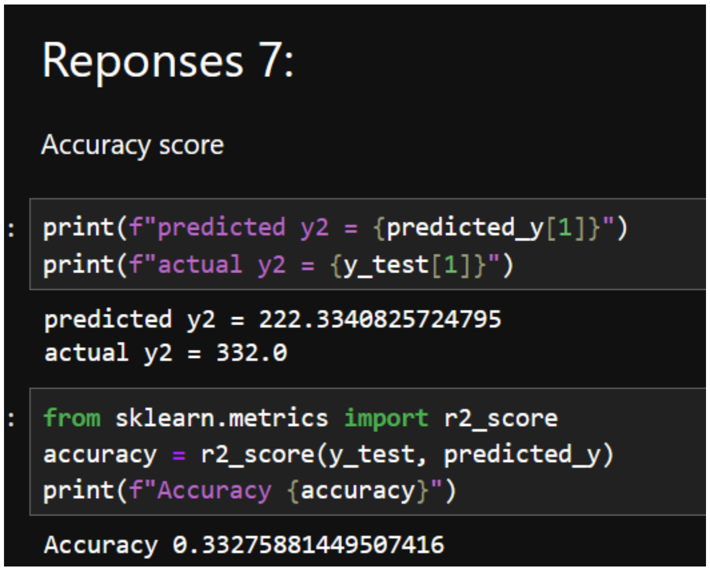
Réponses 6 :


Reponses 6:

predict the data

```
predicted_y = model.predict(x_test)
```

Réponses 7 :


Reponses 7:

Accuracy score

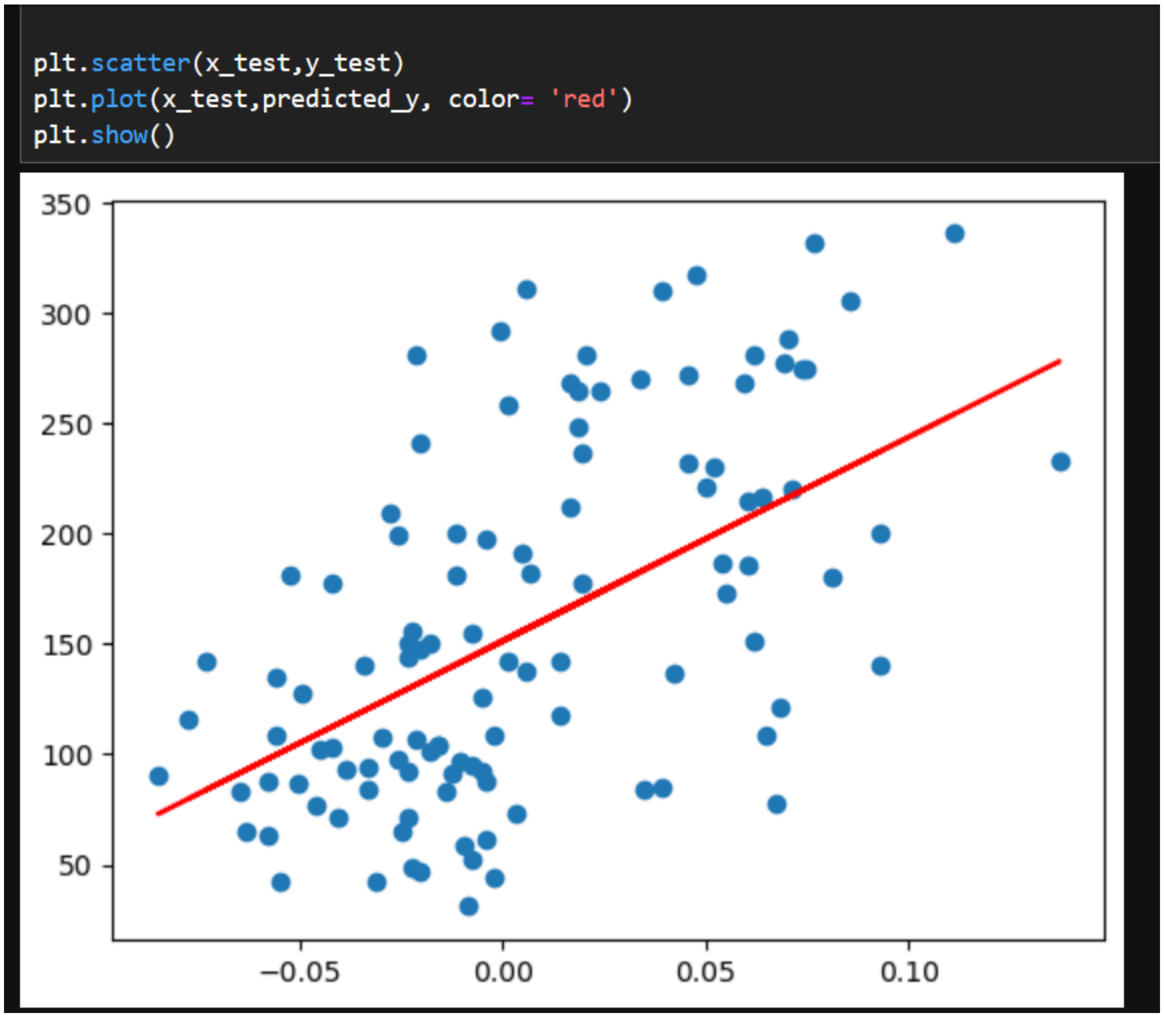
```
: print(f"predicted y2 = {predicted_y[1]}")
  print(f"actual y2 = {y_test[1]}")
```

```
predicted y2 = 222.3340825724795
actual y2 = 332.0
```

```
: from sklearn.metrics import r2_score
  accuracy = r2_score(y_test, predicted_y)
  print(f"Accuracy {accuracy}")
```

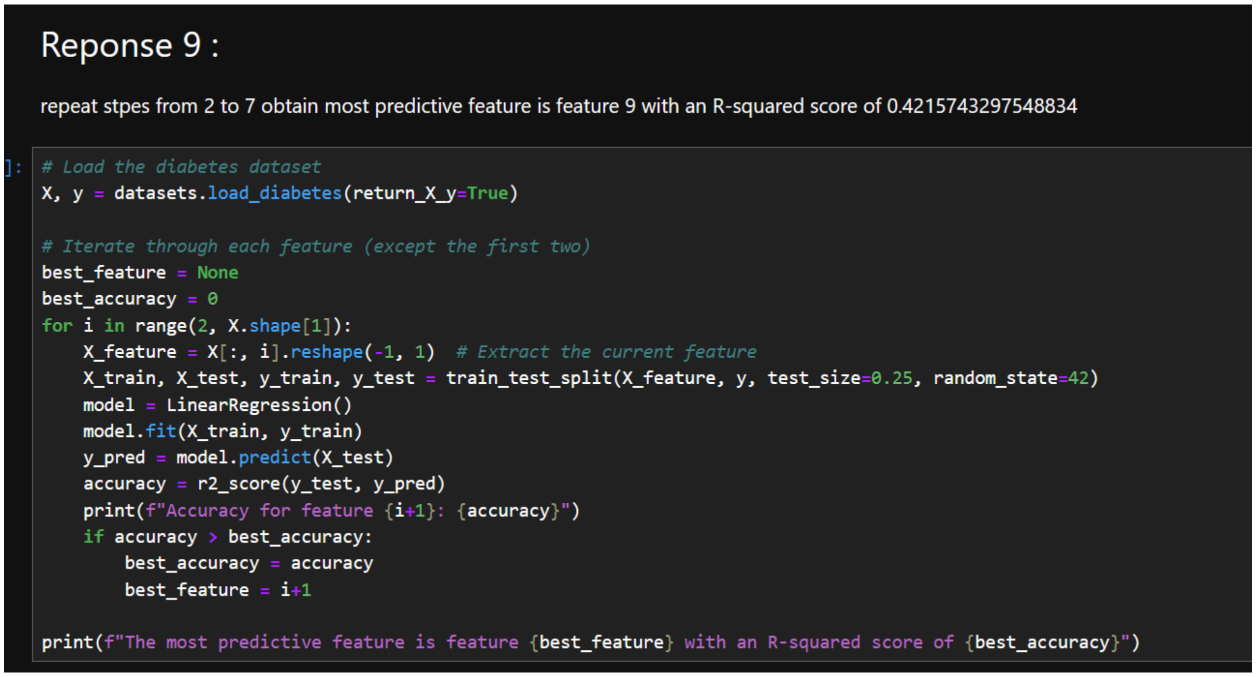
```
Accuracy 0.33275881449507416
```

Réponses 8 :



```
plt.scatter(x_test,y_test)
plt.plot(x_test,predicted_y, color= 'red')
plt.show()
```

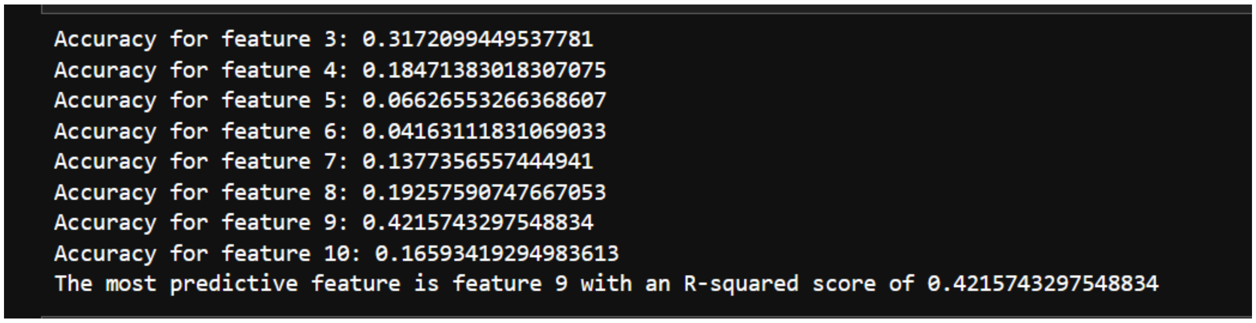
Réponses 9 :



Reponse 9 :

repeat stpes from 2 to 7 obtain most predictive feature is feature 9 with an R-squared score of 0.4215743297548834

```
# Load the diabetes dataset
X, y = datasets.load_diabetes(return_X_y=True)

# Iterate through each feature (except the first two)
best_feature = None
best_accuracy = 0
for i in range(2, X.shape[1]):
    X_feature = X[:, i].reshape(-1, 1)  # Extract the current feature
    X_train, X_test, y_train, y_test = train_test_split(X_feature, y, test_size=0.25, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = r2_score(y_test, y_pred)
    print(f"Accuracy for feature {i+1}: {accuracy}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_feature = i+1

print(f"The most predictive feature is feature {best_feature} with an R-squared score of {best_accuracy}")
```



```
Accuracy for feature 3: 0.3172099449537781
Accuracy for feature 4: 0.18471383018307075
Accuracy for feature 5: 0.06626553266368607
Accuracy for feature 6: 0.04163111831069033
Accuracy for feature 7: 0.1377356557444941
Accuracy for feature 8: 0.19257590747667053
Accuracy for feature 9: 0.4215743297548834
Accuracy for feature 10: 0.16593419294983613
The most predictive feature is feature 9 with an R-squared score of 0.4215743297548834
```