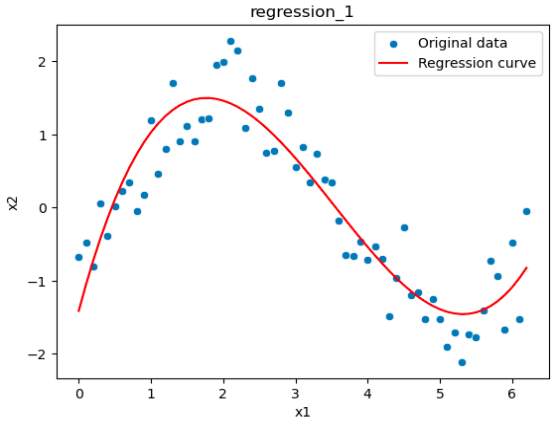
Homework 7

a) This is perhaps a non-noisy polynomial function of degree=3 (cubic).

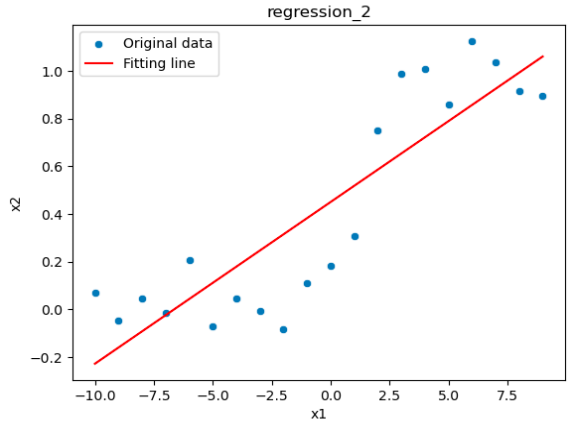



```
# For the regression_1 data, fit a polynomial regression model using
# sklearn.preprocessing.PolynomialFeatures
from sklearn import linear_model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
degree = 3
model = make_pipeline(PolynomialFeatures(degree), linear_model.LinearRegression())
model.fit(regression_1[['x1']], regression_1['x2'])

# plot the regression_1 data and the regression curve (polynomial)
sns.scatterplot(x='x1', y='x2', data=regression_1, label="Original data")
# plotting the regression curve
plt.plot(regression_1['x1'], model.predict(regression_1[['x1']]), color='red',
         label="Regression curve")
plt.title('regression_1')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```

a) I made a pipeline of two different models (linear and polynomial) to fit the whole data so that the distance between the actual data points and the predicted is minimized (Mean Squared Error).

b) The data seems to be linearly distributed with noise at start and end.

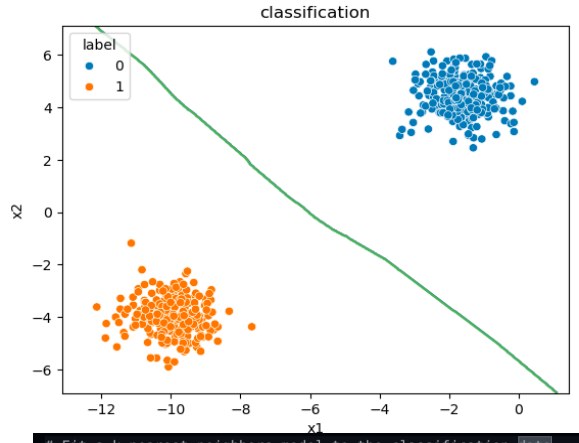



```
# Fitting a linear regression model to the regression_2 data
reg = linear_model.LinearRegression()
x1 = regression_2[['x1']]
x2 = regression_2[['x2']]
# Fitting the model
reg.fit(x1, x2)

# plot the regression_2 data and the regression line
sns.scatterplot(x='x1', y='x2', data=regression_2, label='Original data')
plt.plot(regression_2['x1'], reg.predict(regression_2[['x1']]), color='red',
         label='Fitting line')
plt.title('regression_2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(loc='best')
plt.show()
```

b) I fitted a linear model to the data as it seems somehow linearly distributed with the value of slope **0.067** and correlation coefficient of **0.873355** suggest that the data could be fitted with linear line.

c) The data is divided into two clusters of points which are extremely distinct from one another. Might be temperature readings of two sensor.




```
# Fit a k-nearest neighbors model to the classification data
from sklearn.neighbors import KNeighborsClassifier

n_neighbors=3
X = classification[['x1', 'x2']]
y = classification['label']
knn = KNeighborsClassifier(n_neighbors=n_neighbors)
knn.fit(X, y)

# plot the classification data and the decision boundary
sns.scatterplot(x='x1', y='x2', data=classification, hue=y)
# plotting the decision boundary
x_min, x_max = classification['x1'].min() - 1, classification['x1'].max() + 1
y_min, y_max = classification['x2'].min() - 1, classification['x2'].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contour(xx, yy, Z, alpha=0.4)
plt.title('classification')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

c) I fitted a KNN model as the data consists of two clusters with the labels provided, so it is a supervised problem. KNN fits small to medium sized data well also in case of binary classifications. And it requires less training time than other algorithms.

Abdallah Eid, M.N. 22942414, IdM: ez72oxib