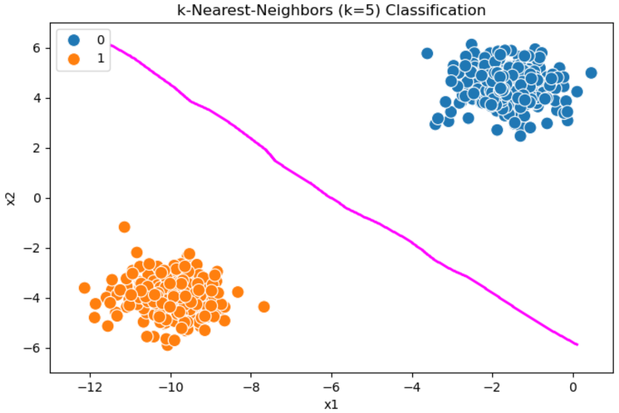
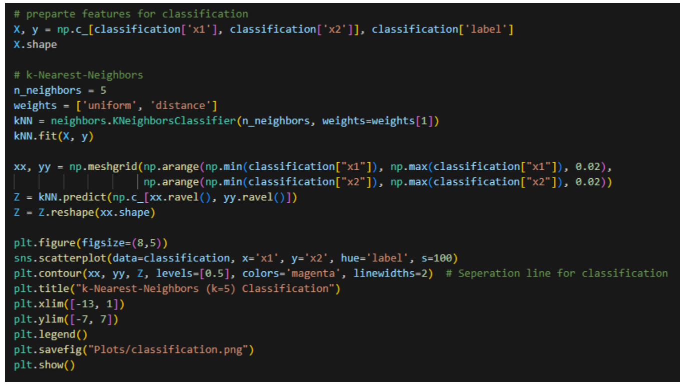
a.) It is non-noisy data with a binary classification as the output is either 0 or 1

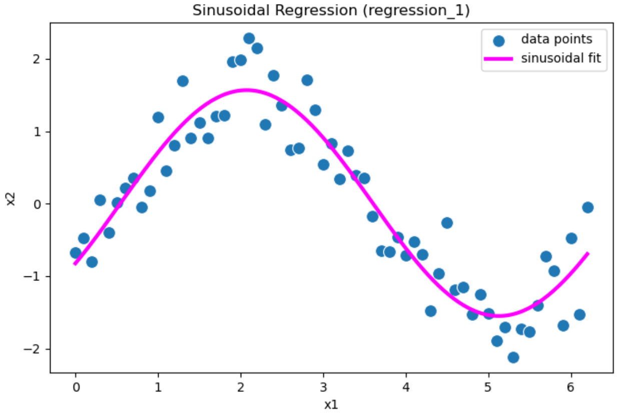

k-Nearest-Neighbors (k=5) Classification

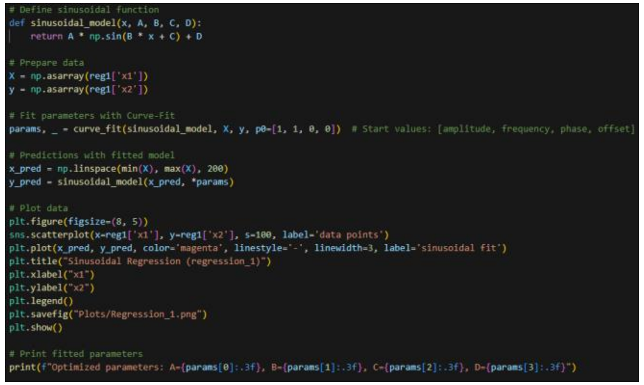


```python
# preparte features for classification
X, y = np.c_[classification['x1'], classification['x2']], classification['label']
X.shape

# k-Nearest-Neighbors
n_neighbors = 5
weights = ['uniform', 'distance']
kNN = neighbors.KNeighborsClassifier(n_neighbors, weights=weights[1])
kNN.fit(X, y)

xx, yy = np.meshgrid(np.arange(np.min(classification["x1"]), np.max(classification["x1"]), 0.02),
                     np.arange(np.min(classification["x2"]), np.max(classification["x2"]), 0.02))
Z = kNN.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(8,5))
sns.scatterplot(data=classification, x='x1', y='x2', hue='label', s=100)
plt.contour(xx, yy, Z, levels=[0.5], colors='magenta', linewidths=2)  # Seperation line for classification
plt.title("k-Nearest-Neighbors (k=5) Classification")
plt.xlim([-13, 1])
plt.ylim([-7, 7])
plt.legend()
plt.savefig("Plots/classification.png")
plt.show()
```

a.) I chose the k-NN (with k=5) method. It creates a nearly linear decision boundary, which works good as the classes are very distinct.

b.) It is noisy data that follows a sinusoidal pattern

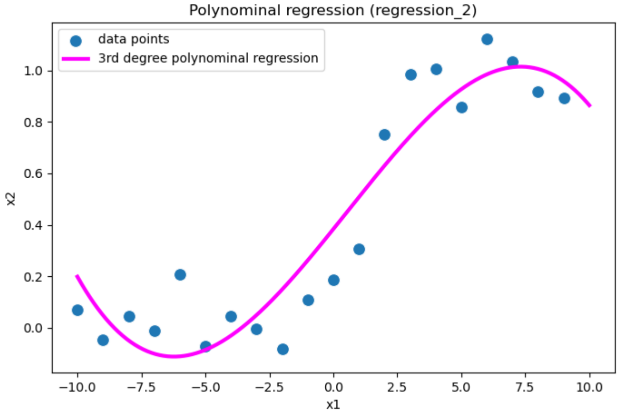

Sinusoidal Regression (regression_1)

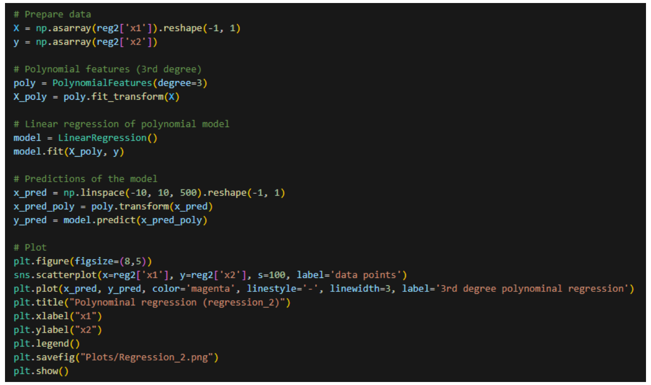


```python
# Define sinusoidal function
def sinusoidal_model(x, A, B, C, D):
    return A * np.sin(B * x + C) + D

# Prepare data
X = np.asarray(reg1['x1'])
y = np.asarray(reg1['x2'])

# Fit parameters with Curve-Fit
params, _ = curve_fit(sinusoidal_model, X, y, p0=[1, 1, 0, 0])  # Start values: [amplitude, frequency, phase, offset]

# Predictions with fitted model
x_pred = np.linspace(min(X), max(X), 200)
y_pred = sinusoidal_model(x_pred, *params)

# Plot data
plt.figure(figsize=(8, 5))
sns.scatterplot(x=reg1['x1'], y=reg1['x2'], s=100, label='data points')
plt.plot(x_pred, y_pred, color='magenta', linestyle='-', linewidth=3, label='sinusoidal fit')
plt.title("Sinusoidal Regression (regression_1)")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.savefig("Plots/Regression_1.png")
plt.show()

# Print fitted parameters
print(f"Optimized parameters: A={params[0]:.3f}, B={params[1]:.3f}, C={params[2]:.3f}, D={params[3]:.3f}")
```

b.) Because of the sinusoidal pattern of the data I chose sinusoidal regression with the general form: y=A·sin(B·x+C)+D. With A=1.559, B=1.029, C=-0.564 and D=0.007.

c.) The data is a non-linear noisy function (possible data source: oscillations)


Polynominal regression (regression_2)



```python
# Prepare data
X = np.asarray(reg2['x1']).reshape(-1, 1)
y = np.asarray(reg2['x2'])

# Polynomial features (3rd degree)
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)

# Linear regression of polynomial model
model = LinearRegression()
model.fit(X_poly, y)

# Predictions of the model
x_pred = np.linspace(-10, 10, 500).reshape(-1, 1)
x_pred_poly = poly.transform(x_pred)
y_pred = model.predict(x_pred_poly)

# Plot
plt.figure(figsize=(8,5))
sns.scatterplot(x=reg2['x1'], y=reg2['x2'], s=100, label='data points')
plt.plot(x_pred, y_pred, color='magenta', linestyle='-', linewidth=3, label='3rd degree polynominal regression')
plt.title("Polynominal regression (regression_2)")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.savefig("Plots/Regression_2.png")
plt.show()
```

c.) I chose a 3rd degree polynomal regression model. It is useful to fit the non-linear relation of tha data as seen in the plot.

Name: Constantin Wolff
Mat. Num.: 22442020
IdM: lu11synu