day-1 q7

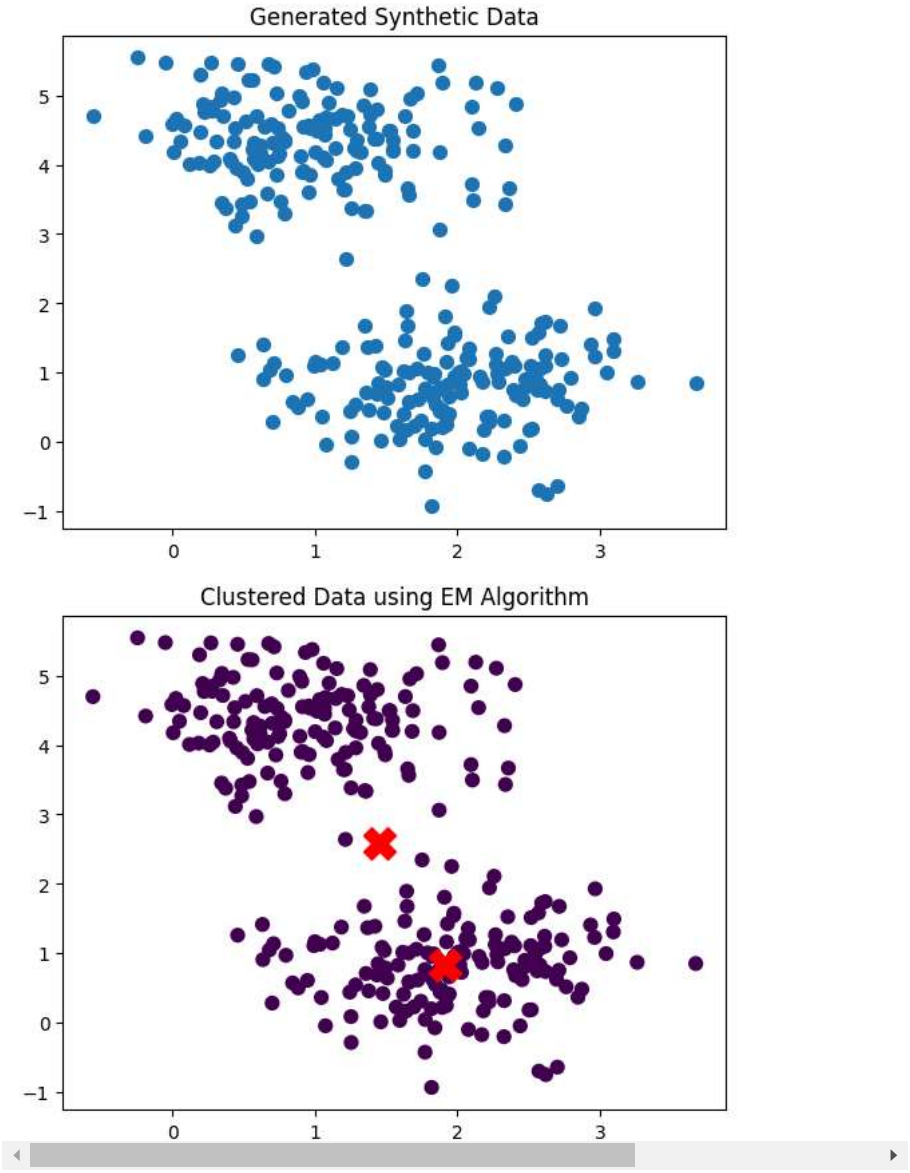
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

# Load the dataset
iris = load_iris()
X = iris.data
y = (iris.target == 2).astype(int)  # Convert to binary classification problem

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model
model = LogisticRegression(solver='liblinear')
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred_prob = model.predict_proba(X_test)[:, 1]

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"ROC-AUC: {roc_auc:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)

# Visualize the confusion matrix
```

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
ROC-AUC: 1.00
Confusion Matrix:
[[32  0]
 [ 0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        32
           1       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

day-1 q8

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
np.random.seed(42)
X, _ = make_blobs(n_samples=300, centers=2, cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50, cmap='viridis')
plt.title("Generated Synthetic Data")
plt.show()
class GaussianMixture:
      def __init__(self, n_components, max_iter=100, tol=1e-4):
              self.n_components = n_components
              self.max_iter = max_iter

              self.tol = tol
      def initialize_parameters(self, X):
                n_samples, n_features = X.shape
                self.weights = np.ones(self.n_components) / self.n_components
                self.means = X[np.random.choice(n_samples, self.n_components, False)]
                self.covariances = [np.cov(X, rowvar=False)] * self.n_components
      def gaussian(self, X, mean, covariance):
                n_features = X.shape[1]
                diff = (X - mean).T
                return np.exp(-0.5 * np.sum(diff.T @ np.linalg.inv(covariance) * diff.T, axis=1)) / \
                       np.sqrt((2 * np.pi) ** n_features * np.linalg.det(covariance))
      def e_step(self, X):
                self.responsibilities = np.zeros((X.shape[0], self.n_components))
                for k in range(self.n_components):
                    self.responsibilities[:, k] = self.weights[k] * self.gaussian(X, self.means[k], self.covariances[k])
                    self.responsibilities /= self.responsibilities.sum(1)[:, np.newaxis]
      def m_step(self, X):
                Nk = self.responsibilities.sum(axis=0)
                self.weights = Nk / X.shape[0]
                self.means = (self.responsibilities.T @ X) / Nk[:, np.newaxis]
                for k in range(self.n_components):
                    diff = X - self.means[k]
                    self.covariances[k] = (self.responsibilities[:, k][:, np.newaxis] * diff).T @ diff / Nk[k]
      def fit(self, X):
                self.initialize_parameters(X)
                log_likelihood = []
                for i in range(self.max_iter):
                    self.e_step(X)
                    self.m_step(X)
                    log_likelihood.append(np.sum(np.log(np.sum([self.weights[k] * self.gaussian(X, self.means[k], self.covariances[k])
                                                        for k in range(self.n_components)], axis=0))))
                    if len(log_likelihood) > 1 and np.abs(log_likelihood[-1] - log_likelihood[-2]) < self.tol:
                       break
      def predict(self, X):
                self.e_step(X)
                return np.argmax(self.responsibilities, axis=1)
gmm = GaussianMixture(n_components=2)
gmm.fit(X)
predictions = gmm.predict(X)

# Plot the clustered data
plt.scatter(X[:, 0], X[:, 1], c=predictions, s=50, cmap='viridis')
plt.scatter(gmm.means[:, 0], gmm.means[:, 1], s=300, c='red', marker='X')
plt.title("Clustered Data using EM Algorithm")
plt.show()
```

```
<ipython-input-7-30cf597cbb4c>:6: UserWarning: No data for colormapping provided via 'c'
  plt.scatter(X[:, 0], X[:, 1], s=50, cmap='viridis')
```




day2-q2

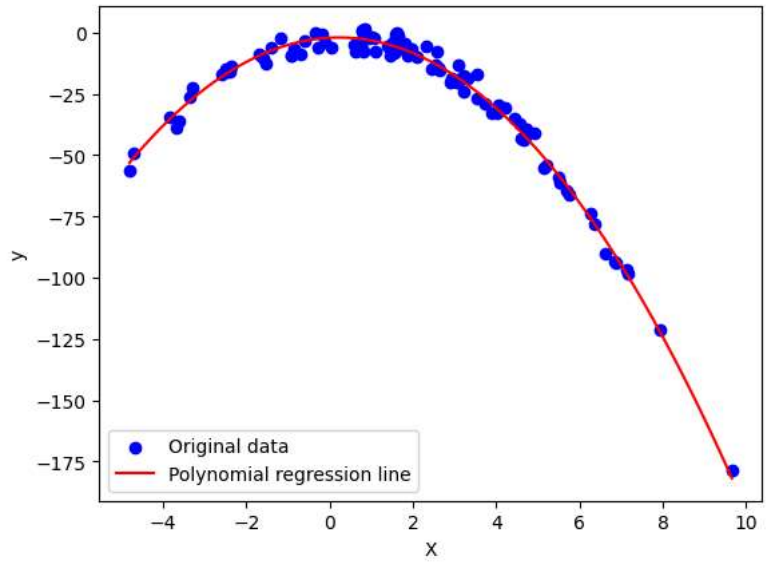
```
import pandas as pd

# Given dataset
data = {
    'Origin': ['Japan', 'Japan', 'Japan', 'USA', 'Japan'],
    'Manufacturer': ['Honda', 'Toyota', 'Toyota', 'Chrysler', 'Honda'],
    'Color': ['Blue', 'Green', 'Blue', 'Red', 'White'],
    'Decade': ['1980', '1970', '1990', '1980', '1980'],
    'Type': ['Economy', 'Sports', 'Economy', 'Economy', 'Economy'],
    'Example Type': ['Positive', 'Negative', 'Positive', 'Negative', 'Positive']
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Find-S Algorithm
def find_s_algorithm(data):
# Initialize hypothesis with the most specific hypothesis
    hypothesis = ['0'] * (len(data.columns) - 1)

    # Iterate over each training example
    for i in range(len(data)):
    # Check if the example is positive
       if data.iloc[i,-1] == 'Positive':
                                                                        # Update hypothesis
           for j in range(len(hypothesis)):
               if hypothesis[j] == '0':  # Initialize to the first positive example
                  hypothesis[j] = data.iloc[i, j]
               elif hypothesis[j] != data.iloc[i, j]:  # Generalize
                  hypothesis[j] = '?'

    return hypothesis


hypothesis = find_s_algorithm(df)
print("The most specific hypothesis is:", hypothesis)
```

```
The most specific hypothesis is: ['Japan', '?', '?', '?', 'Economy']
```

day2-q3

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Generate a synthetic dataset
np.random.seed(0)
X = 2 - 3 * np.random.normal(0, 1, 100)
y = X - 2 * (X ** 2) + np.random.normal(-3, 3, 100)

X = X[:, np.newaxis]

# Step 2: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Step 3: Preprocess the data using PolynomialFeatures
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Step 4: Fit the polynomial regression model
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Step 5: Make predictions and evaluate the model
y_train_pred = model.predict(X_train_poly)
y_test_pred = model.predict(X_test_poly)

# Calculate performance metrics
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# Print performance metrics
print(f'Training MSE: {mse_train:.2f}')
print(f'Test MSE: {mse_test:.2f}')
print(f'Training R^2: {r2_train:.2f}')
print(f'Test R^2: {r2_test:.2f}')

# Step 6: Visualize the results
plt.scatter(X, y, color='blue', label='Original data')
X_plot = np.linspace(min(X), max(X), 100)
X_plot_poly = poly.transform(X_plot)
y_plot = model.predict(X_plot_poly)
plt.plot(X_plot, y_plot, color='red', label='Polynomial regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
Training MSE: 9.73
Test MSE: 8.70
Training R^2: 0.99
Test R^2: 0.99
```

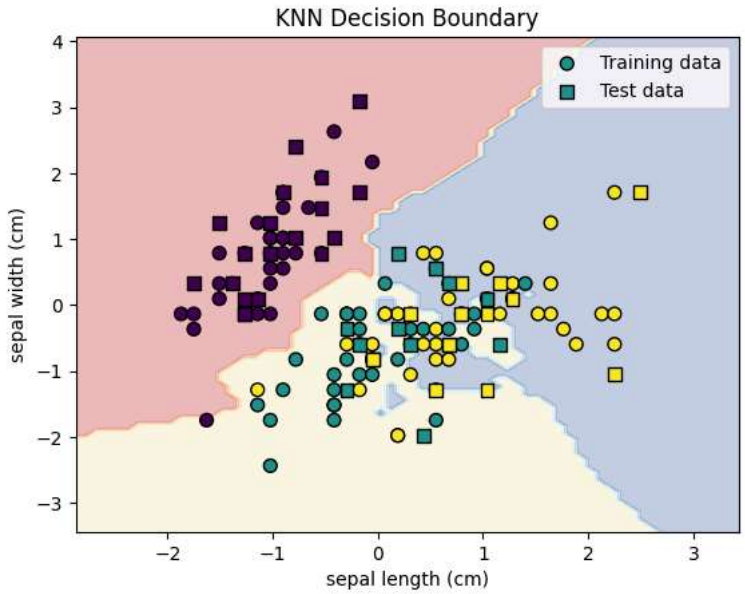



day2-q4

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Print dataset information
print(f'Dataset shape: {X.shape}')
print(f'Target shape: {y.shape}')

# Step 2: Preprocess the data by standardizing the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Print the shapes of the training and testing sets
print(f'Training set shape: {X_train.shape}')
print(f'Testing set shape: {X_test.shape}')

# Step 4: Apply the KNN algorithm
knn = KNeighborsClassifier(n_neighbors=3)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

# Step 5: Evaluate the model's performance
# Make predictions on the test data
y_pred = knn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print classification report
print('Classification Report:')
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Print confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))

# Step 6: Visualize the results
# For visualization, select the first two features
X_vis = X[:, :2]
X_train_vis, X_test_vis, y_train_vis, y_test_vis = train_test_split(X_vis, y, test_size=0.3, random_state=42)

# Fit the classifier to the subset data
knn_vis = KNeighborsClassifier(n_neighbors=3)
knn_vis.fit(X_train_vis, y_train_vis)

# Create a mesh grid
x_min, x_max = X_vis[:, 0].min() - 1, X_vis[:, 0].max() + 1
y_min, y_max = X_vis[:, 1].min() - 1, X_vis[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))

# Predict the class for each point in the mesh grid
Z = knn_vis.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.RdYlBu)

# Plot the training points
plt.scatter(X_train_vis[:, 0], X_train_vis[:, 1], c=y_train_vis, marker='o', edgecolor='k', s=50, label='Training data')
# Plot the testing points
plt.scatter(X_test_vis[:, 0], X_test_vis[:, 1], c=y_test_vis, marker='s', edgecolor='k', s=50, label='Test data')

plt.xlabel(iris.feature_names[0])
```

```
plt.ylabel(iris.feature_names[1])
plt.title('KNN Decision Boundary')
plt.legend()
plt.show()
```

```
Dataset shape: (150, 4)
Target shape: (150,)
Training set shape: (105, 4)
Testing set shape: (45, 4)
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```




day2-q5

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Predefined dataset
data = {
    'Age': [25, 30, 35, 40, 45],
    'Annual_Income': [50000, 60000, 80000, 100000, 120000],
    'Employment_Years': [3, 5, 10, 15, 20],
    'Debt_Amount': [5000, 10000, 20000, 30000, 40000],
    'Credit_History_Length': [5, 10, 15, 20, 25],
    'Number_of_Credit_Cards': [2, 3, 4, 5, 6],
    'Credit_Score': [650, 700, 750, 800, 850]
}

df = pd.DataFrame(data)

                            # Standardize features
scaler = StandardScaler()
features = df.drop('Credit_Score', axis=1)
scaled_features = scaler.fit_transform(features)

# Create a DataFrame for the scaled features
scaled_features_df = pd.DataFrame(scaled_features, columns=features.columns)
scaled features df['Credit Score'] = df['Credit Score']
```