Q1:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

data = {
    'Weather': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny'],
    'Temperature': ['Hot', 'Hot', 'Mild', 'Mild', 'Cool', 'Mild', 'Mild', 'Hot', 'Mild', 'Mild'],
    'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)

le_weather = LabelEncoder()
le_temperature = LabelEncoder()
le_play = LabelEncoder()

df['Weather'] = le_weather.fit_transform(df['Weather'])
df['Temperature'] = le_temperature.fit_transform(df['Temperature'])
df['Play'] = le_play.fit_transform(df['Play'])

X = df[['Weather', 'Temperature']]
y = df['Play']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

prediction = knn.predict([[le_weather.transform(['Overcast'])[0], le_temperature.transform(['Mild'])[0]]])

play_result = le_play.inverse_transform(prediction)
print(f"Prediction: The player {'can' if play_result[0] == 'Yes' else 'cannot'} play when the weather is overcast and the temperature is mild.")

y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

print("\nConfusion Matrix:")
print(cm)
```

```
Prediction: The player can play when the weather is overcast and the temperature is mild.

Confusion Matrix:
[[1 0]
 [0 1]]
```

Q2:

```python
from sklearn.neighbors import KNeighborsClassifier

X_train = [[7, 7], [7, 4], [3, 4], [1, 4]]
y_train = ['A', 'A', 'B', 'B']  # Classes

X_new = [[3, 7]]

knn = KNeighborsClassifier(n_neighbors=3)


knn.fit(X_train, y_train)

prediction = knn.predict(X_new)

print(f"The predicted class for the new tissue with X1=3 and X2=7 is: {prediction[0]}")
```

```
The predicted class for the new tissue with X1=3 and X2=7 is: B
```

Q3:

Artificial Intelligence

```python
import numpy as np
from collections import Counter

X_train = np.array([[7, 7], [7, 4], [3, 4], [1, 4]])
y_train = ['A', 'A', 'B', 'B']

X_query = np.array([3, 7])

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

distances = []
for i, x_train in enumerate(X_train):
    distance = euclidean_distance(X_query, x_train)
    distances.append((distance, y_train[i]))

distances.sort(key=lambda x: x[0])

k = 3
nearest_neighbors = distances[:k]

neighbor_classes = [neighbor[1] for neighbor in nearest_neighbors]

prediction = Counter(neighbor_classes).most_common(1)[0][0]

print(f"Distances: {distances}")
print(f"3 Nearest Neighbors: {nearest_neighbors}")
print(f"Predicted class for the query instance (3, 7): {prediction}")
```

```
Distances: [(3.0, 'B'), (3.605551275463989, 'B'), (4.0, 'A'), (5.0, 'A')]
3 Nearest Neighbors: [(3.0, 'B'), (3.605551275463989, 'B'), (4.0, 'A')]
Predicted class for the query instance (3, 7): B
```

HomeTask: