In [8]:

```
import numpy as np
import pandas as pd

# Sample dataset
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Over
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'M
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Norma
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes
}

df = pd.DataFrame(data)

# Features and labels
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

class NaiveBayesClassifier:
    def __init__(self):
        self.prior = {}
        self.likelihood = {}
        self.classes = None

    def fit(self, X, y):
        self.classes = np.unique(y)
        for cls in self.classes:
            X_cls = X[y == cls]
            self.prior[cls] = len(X_cls) / len(X)
            self.likelihood[cls] = {}
            for col in range(X.shape[1]):
                values, counts = np.unique(X_cls[:, col], return_counts=Tru
                self.likelihood[cls][col] = dict(zip(values, counts / len(X

    def predict(self, X):
        predictions = []
        for x in X:
            posteriors = {}
            for cls in self.classes:
                posterior = np.log(self.prior[cls])
                for col in range(len(x)):
                    if x[col] in self.likelihood[cls][col]:
                        posterior += np.log(self.likelihood[cls][col][x[col
                    else:
                        posterior += np.log(1e-6)  # Laplace smoothing
                posteriors[cls] = posterior
            predictions.append(max(posteriors, key=posteriors.get))
        return np.array(predictions)
```

```
    def predict_single(self, query):
        posteriors = {}
        for cls in self.classes:
            posterior = np.log(self.prior[cls])
            for col in range(len(query)):
                if query[col] in self.likelihood[cls][col]:
                    posterior += np.log(self.likelihood[cls][col][query[col
                else:
                    posterior += np.log(1e-6)  # Laplace smoothing
            posteriors[cls] = posterior

        # Convert log probabilities to actual probabilities
        exp_posteriors = {cls: np.exp(log_prob) for cls, log_prob in poster
        total_prob = sum(exp_posteriors.values())
        probabilities = {cls: prob / total_prob for cls, prob in exp_poster

        return max(probabilities, key=probabilities.get), probabilities

# Train the model
nb = NaiveBayesClassifier()
nb.fit(X, y)

# Define a specific query
query = ['Overcast', 'Hot', 'High', 'Strong']

# Make a prediction for the specific query
predicted_class, probabilities = nb.predict_single(query)

print(f"Predicted class for the query: {predicted_class}")
print(f"Posterior probabilities: {probabilities}")
print(f"Sum of probabilities: {sum(probabilities.values())}")
```

```
Predicted class for the query: Yes
Posterior probabilities: {'No': 9.719905522518337e-06, 'Yes': 0.9999902800944774}
Sum of probabilities: 1.0
```

In [ ]:

In [ ]: