ASSIGNMENT-4

Title: Implement the non-parametric Locally Weighted Regression algorithm to fit data points. Select the appropriate data set for your experiment and draw graphs

Objective: To fit data points by assigning different weights to each point based on its proximity to the query point.

Dataset: A synthetic dataset with a sinusoidal pattern will be used to showcase the capabilities of the locally weighted regression algorithm. You can generate a dataset by using python code. np.random.seed(x)

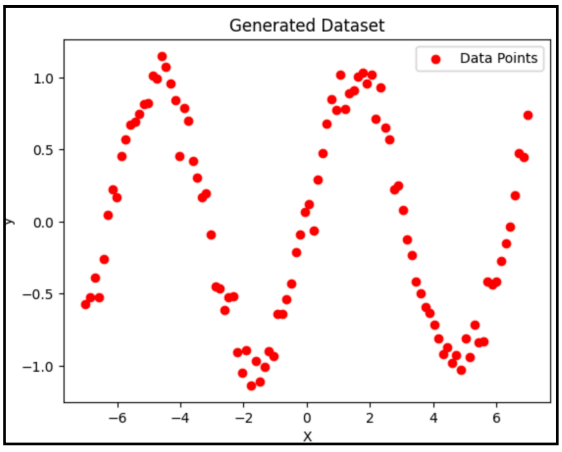
Tasks:

1) Generate a dataset.'

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(20)
X = np.linspace(-7, 7, 100)
y = np.sin(X) + np.random.normal(0, 0.1, X.shape)  # Sine wave with noise
```

2) Split eand plotting each dataset into features (X) and target variable (y).

```
plt.scatter(X, y, color='red', label='Data Points')
plt.title('Generated Dataset')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```



3) Implement the Locally Weighted Regression algorithm:

```
def get_weights(X, query_point, tau):
    m = X.shape[0]
    weights = np.eye(m)
    for i in range(m):
        diff = query_point - X[i]
        weights[i, i] = np.exp(-(diff ** 2) / (2 * tau ** 2))
    return weights
```

```
def locally_weighted_linear_regression(X, y, query_point, tau):
    X_ = np.vstack((np.ones(len(X)), X)).T
    query_point_ = np.array([1, query_point])

    weights = get_weights(X, query_point, tau)

    theta = np.linalg.pinv(X_.T @ weights @ X_) @ (X_.T @ weights @ y)
    return query_point_ @ theta

# Prediction over the entire dataset
def predict_lwlr(X, y, tau):
    y_pred = np.zeros(len(X))
    for i in range(len(X)):
        y_pred[i] = locally_weighted_linear_regression(X, y, X[i], tau)
    return y_pred
```

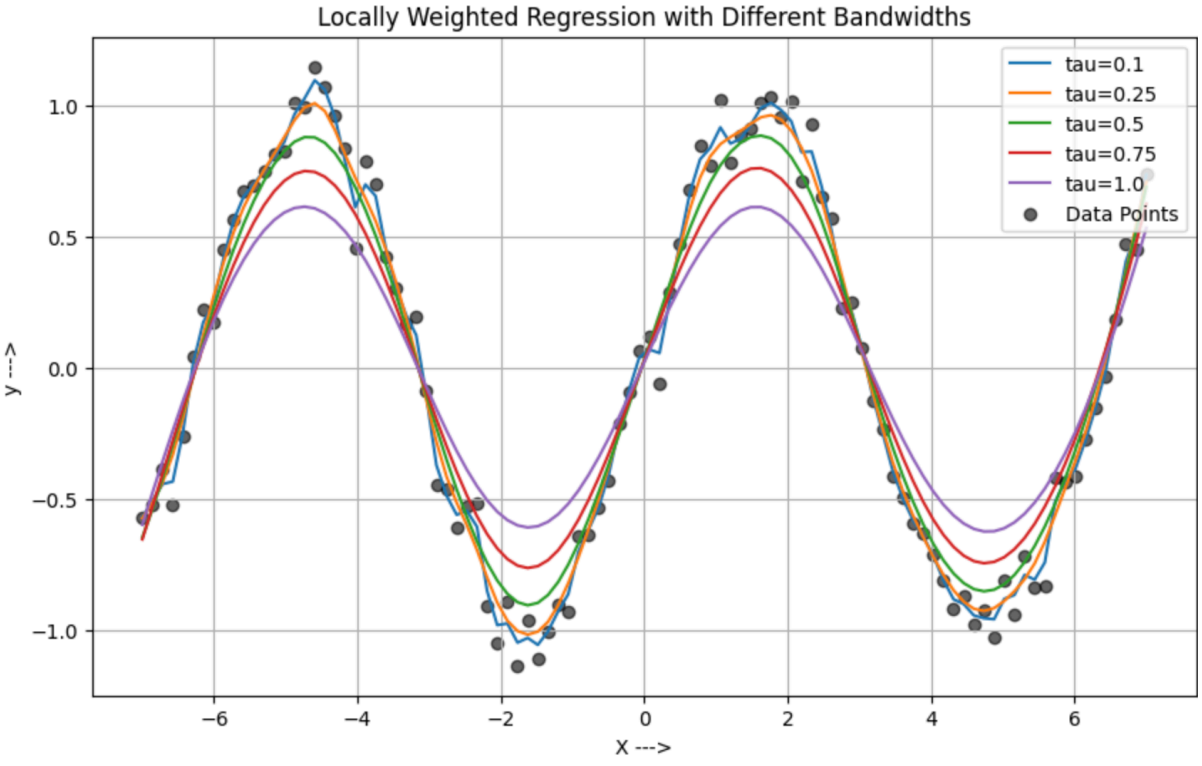
4) Experiment using multiple query points across the range of the dataset.

```
taus = [0.1, 0.25, 0.5, 0.75, 1.0]
query_points = X

plt.figure(figsize=(10, 6))
for tau in taus:
    predictions = [locally_weighted_linear_regression(X, y, qp, tau) for qp in query_points]
    plt.plot(query_points, predictions, label=f'tau={tau}')

plt.scatter(X, y, color='black', label='Data Points', alpha=0.6)

plt.title('Locally Weighted Regression with Different Bandwidths')
plt.xlabel('X --->')
plt.ylabel('y --->')
plt.legend()
plt.grid(True)
plt.show()
```

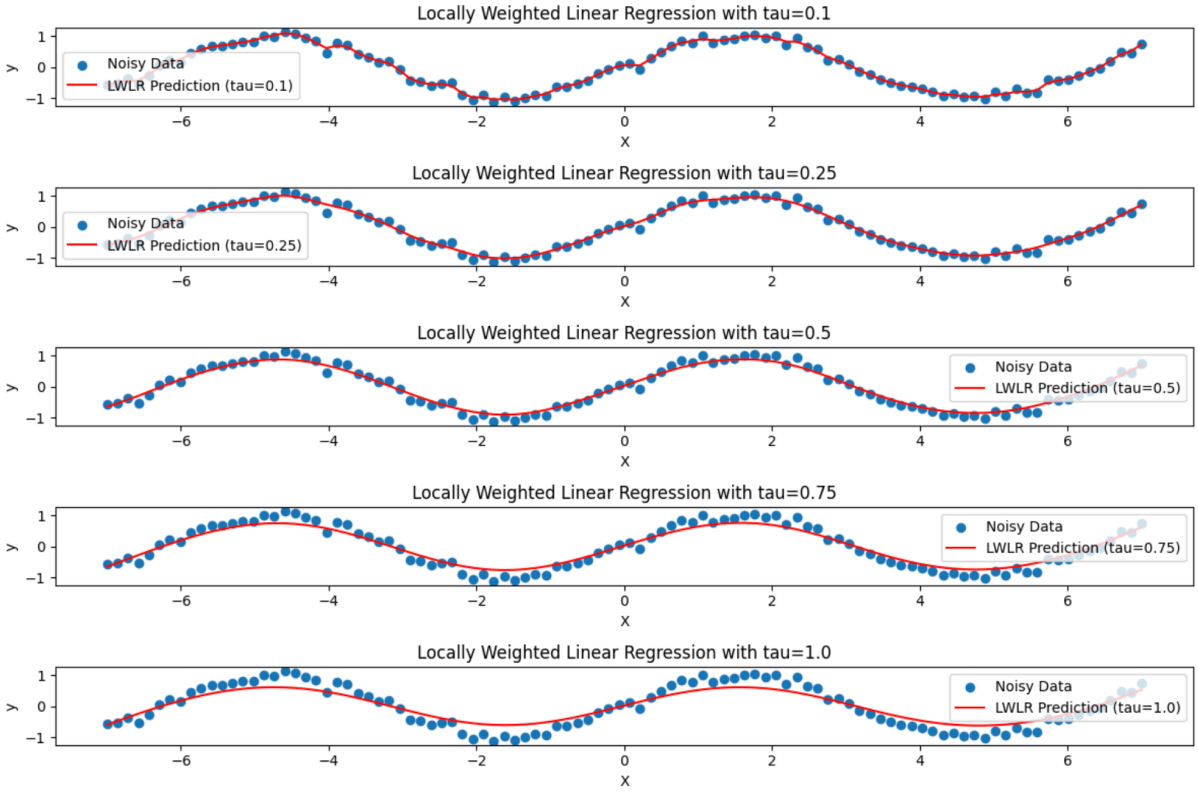


5) Create a plot with the original dataset points and the fitted curves for different query points and bandwidths.

```
taus = [0.1, 0.25, 0.5, 0.75, 1.0]
plt.figure(figsize=(12, 8))

for i, tau in enumerate(taus):
    y_pred = predict_lwlr(X, y, tau)
    plt.subplot(5, 1, i + 1)  #subplot for each tau
    plt.scatter(X, y, label='Noisy Data')
    plt.plot(X, y_pred, color='red', label=f'LWLR Prediction (tau={tau})')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.legend()
    plt.title(f"locally Weighted Linear Regression with tau={tau}')

plt.tight_layout()
plt.show()
```



Conclusion: In this experiment, Locally Weighted Regression (LWLR) effectively fits non-linear, noisy data by adjusting the influence of nearby points using the bandwidth parameter `tau`. Smaller `tau` values captured local variations more closely, while larger `tau` values provided a smoother, generalized fit. Overall, LWLR offers flexibility in modeling complex patterns, but careful selection of `tau` is essential to balance overfitting and underfitting.