Iris\Iris_functions.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from Iris_functions import *
import seaborn as sns
from scipy.stats import gaussian_kde

# Data processing and loading functions
def remove_features(data, disabled_features):
    data.columns = data.columns.str.strip()
    data = data.drop(columns=disabled_features)
    return data

def create_train_test_data(setosa, versicolour, verginica, N_train, N_test, first_30_to_train):
    if first_30_to_train:
        train_data = pd.concat([setosa[:N_train], versicolour[:N_train], verginica[:N_train]])
        test_data = pd.concat([setosa[N_train:N_train+N_test],
versicolour[N_train:N_train+N_test], verginica[N_train:N_train+N_test]])
        train_data = train_data.values
        test_data = test_data.values
    else:
        test_data = pd.concat([setosa[:N_test], versicolour[:N_test], verginica[:N_test]])
        train_data = pd.concat([setosa[N_test:N_test+N_train],
versicolour[N_test:N_test+N_train], verginica[N_test:N_test+N_train]])
        train_data = train_data.values
        test_data = test_data.values
    return train_data, test_data

# Functoins for training algorithm 3.1 in compendium
def sigmoid(x):
    return np.array(1 / (1 + np.exp(-x)))

def grad_W_MSE_func(g_k, t_k, x_k, D):
    A = (g_k - t_k)*g_k*(1-g_k)
    A = A.reshape(3, 1)
    B = x_k
    B = B.reshape(D+1, 1)
    return A @ B.T


# Plotting functions
def plot_MSE(MSE_list):
    plt.figure()
    plt.plot(MSE_list)
    plt.xlabel("Iteration")
    plt.ylabel("MSE")
    plt.title("MSE vs iteration")
    plt.show()

def print_accuracy_for_confusion_matrix(confusion_matrix, label_names):
    accuracy = round(np.trace(confusion_matrix)/np.sum(confusion_matrix),4)
    print("Accuracy for", label_names,"data:", accuracy)
    print("Error rate for",label_names,"data:", round(1-accuracy,4))
```

```
def plot_confusion_matrix(confusion_matrix_train,confusion_matrix_test, label_names, first_30_to_train, error_rate_train, error_rate_test):
    class_labels = label_names
    df_cm_test = pd.DataFrame(confusion_matrix_test, index = [i for i in class_labels], columns = [i for i in class_labels])
    plt.figure(figsize = (10,7))

    #Confusion matrix for test data
    if first_30_to_train:
        plt.title("Confusion matrix for test data using first 30\n" + "Error rate: " + str(error_rate_test) + "%")
    else:
        plt.title("Confusion matrix for test data using last 30\n" + "Error rate: " + str(error_rate_test) + "%")

    sns.heatmap(df_cm_test, annot=True)

    # Confusion matrix for train data
    df_cm_train = pd.DataFrame(confusion_matrix_train, index = [i for i in class_labels], columns = [i for i in class_labels])
    plt.figure(figsize = (10,7))
    if first_30_to_train:
        plt.title("Confusion matrix for train data using first 30\n" + "Error rate: " + str(error_rate_train) + "%")
    else:
        plt.title("Confusion matrix for train data using last 30\n" + "Error rate: " + str(error_rate_train) + "%")
    sns.heatmap(df_cm_train, annot=True)

def plot_histograms(train_data, N_train):
    #Extract features from training data
    feature_1_class_1 = np.array(train_data[:N_train, 0])
    feature_2_class_1 = np.array(train_data[:N_train, 1])
    feature_3_class_1 = np.array(train_data[:N_train, 2])
    feature_4_class_1 = np.array(train_data[:N_train, 3])

    feature_1_class_2 = np.array(train_data[N_train:2*N_train, 0])
    feature_2_class_2 = np.array(train_data[N_train:2*N_train, 1])
    feature_3_class_2 = np.array(train_data[N_train:2*N_train, 2])
    feature_4_class_2 = np.array(train_data[N_train:2*N_train, 3])

    feature_1_class_3 = np.array(train_data[2*N_train:3*N_train, 0])
    feature_2_class_3 = np.array(train_data[2*N_train:3*N_train, 1])
    feature_3_class_3 = np.array(train_data[2*N_train:3*N_train, 2])
    feature_4_class_3 = np.array(train_data[2*N_train:3*N_train, 3])

    feature_plot_1 = [feature_1_class_1, feature_1_class_2, feature_1_class_3]
    feature_plot_2 = [feature_2_class_1, feature_2_class_2, feature_2_class_3]
    feature_plot_3 = [feature_3_class_1, feature_3_class_2, feature_3_class_3]
    feature_plot_4 = [feature_4_class_1, feature_4_class_2, feature_4_class_3]
    feature_plot_1_text = ['Setosa', 'Versicolour', 'Verginica']
    feature_plot_2_text = ['Setosa', 'Versicolour', 'Verginica']
    feature_plot_3_text = ['Setosa', 'Versicolour', 'Verginica']
    feature_plot_4_text = ['Setosa', 'Versicolour', 'Verginica']

    #Define the features and their corresponding labels
    features = [feature_plot_1, feature_plot_2, feature_plot_3, feature_plot_4]
```

```
    feature_labels = [feature_plot_1_text, feature_plot_2_text, feature_plot_3_text,
feature_plot_4_text]
    #Make list of color for each class to use in plots
    colors = ['red', 'blue', 'green']
    x_lable = ['Spetal Length', 'Spetal Width', 'Petal Length', 'Petal Width']
    #Loop through each feature
    for i, feature in enumerate(features):
        #Create a new figure for each feature
        plt.figure()

        #Loop through each class and plot histogram with probability density curve
        for j, data in enumerate(feature):
            plt.hist(data, density=True, alpha=0.5, label=feature_labels[i][j], color=colors[j])

            #Plot probability density curve
            kde = gaussian_kde(data)
            x_vals = np.linspace(min(data), max(data), 100)
            plt.plot(x_vals, kde(x_vals), color=colors[j])

        plt.title('Histogram with Probability Density Curve for ' + x_lable[i])
        plt.xlabel(x_lable[i])
        plt.ylabel('Number of occurences')
        plt.legend()
```