```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_score
import seaborn as sns
from ucimlrepo import fetch_ucirepo


def linear_data():
    # fetch dataset
    iris = fetch_ucirepo(id=53)

    # data (as pandas dataframes)
    X = iris.data.features
    y = iris.data.targets
    df_iris1 = pd.concat([X, y], axis=1)
    df_iris = df_iris1[df_iris1['class']!='Iris-virginica'].copy()

    # sns.pairplot(df_iris, hue = 'class')
    # plt.show()

    # Selecting the features
    X = df_iris[["petal length", "petal width"]]
    y = df_iris['class']

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Test different C values
    C_values = [0.01, 0.1, 1, 10, 100, 1000, 1000000]
    for C in C_values:
        clf = SVC(C=C, kernel='linear')
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        bacc = balanced_accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')
        print(f"C = {C:<7} -> Accuracy={acc:.3f} -> b_accuracy={bacc:.3f} -> f1={f1:.3f}")

    # Fit best
    clf = SVC(C=1000000, kernel='linear')
    clf.fit(X_train, y_train)

    # Plot decision boundary
    w = clf.coef_[0]
    a = -w[0] / w[1]
    x_ = np.linspace(X_train.iloc[:, 0].min() - 0.5 * X_train.iloc[:, 0].std(),
                     X_train.iloc[:, 0].max() + 0.5 * X_train.iloc[:, 0].std())
    b = clf.intercept_[0]
    y_ = a * x_ - b / w[1]
    y_minus_one = a * x_ + (-b - 1) / w[1]
    y_plus_one  = a * x_ + (-b + 1) / w[1]

    plt.plot(x_, y_, 'k-', label='Decision boundary')
    plt.plot(x_, y_minus_one, 'b--', label='Margin -1')
    plt.plot(x_, y_plus_one,  'r--', label='Margin +1')

    # Plot data points
    plt.scatter(X['petal length'][y == "Iris-setosa"], X['petal width'][y == "Iris-setosa"], c = 'b', label = "Setosa")
    plt.scatter(X['petal length'][y ==  "Iris-versicolor"], X['petal width'][y ==  "Iris-versicolor"], c = 'r', label =  "Versicolor")

    # Plot support vectors
    support_vectors = clf.support_vectors_
    plt.scatter(support_vectors[:, 0], support_vectors[:, 1], s = 100,
                facecolor = 'None', edgecolor = 'k', linestyle = 'dashed', linewidth = 2, label = 'support vectors')

    plt.xlabel('Petal length')
    plt.ylabel('Petal width')
    # plt.axhline(y = 0, color = 'k', linewidth = 0.5)
    # plt.axvline(x = 0, color = 'k', linewidth = 0.5)
    plt.legend()
    # plt.box(False)
    plt.grid(True)
    plt.show()

    # Prediction on X (check)
    df_iris['y_hat'] = clf.predict(X)
    print(df_iris.head())




def non_linear_data():
    # fetch dataset
    wine = fetch_ucirepo(id=109)
    X = wine.data.features
    y = wine.data.targets

    # Make dataset from data
    df_wine = pd.concat([X, y], axis=1)
    df_wine = df_wine[df_wine["class"] != 3].copy()

    # sns.pairplot(df_wine, hue = 'class')
    # plt.show()

    # Select the features and data
    X = df_wine[['Alcohol', 'Color_intensity', 'Proline']]
    y = df_wine['class']

    # Splitting the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Scale the data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # SVM with RBF
    clf = SVC(C = 10, kernel = 'rbf', gamma = 0.1)
    clf.fit(X_train_scaled, y_train)

    # Predict
    y_pred = clf.predict(X_test_scaled)

    # Evaluation
    print("Accuracy:", accuracy_score(y_test, y_pred))
```

```python
    print("Balanced Accuracy:", balanced_accuracy_score(y_test, y_pred))
    print("F1_score:", f1_score(y_test, y_pred, average='weighted'))


    # Test C and Gamma values
    for C in [0.1, 1, 10]:
        for gamma in [0.001, 0.01, 0.1]:
            clf = SVC(C=C, kernel='rbf', gamma=gamma)
            clf.fit(X_train_scaled, y_train)
            y_pred = clf.predict(X_test_scaled)
            acc = accuracy_score(y_test, y_pred)
            bacc = balanced_accuracy_score(y_test, y_pred)
            f1 = f1_score(y_test, y_pred, average='weighted')
            print(f"C={C}, gamma={gamma} => accuracy={acc:.3f} => b_accuracy={bacc:.3f} => f1={f1:.3f}")


    # Making the PCA
    pca = PCA(n_components = 2)
    X_PC = pca.fit_transform(X_train_scaled)

    # Plot the train_data
    plt.scatter(X_PC[y_train == 1, 0], X_PC[y_train == 1, 1], label="Class 1")
    plt.scatter(X_PC[y_train == 2, 0], X_PC[y_train == 2, 1], label="Class 2")
    plt.legend()
    plt.title("PCA of TrainingData")
    plt.show()


    # Decision boundary in PCA
    m = 100
    X_1, X_2 = np.meshgrid(np.linspace(X_PC[:, 0].min(), X_PC[:, 0].max(), m),
                           np.linspace(X_PC[:, 1].min(), X_PC[:, 1].max(), m))

    # Inverse transform to origional feature space
    X_grid_scaled = pca.inverse_transform(np.c_[X_1.ravel(), X_2.ravel()])
    y_hat = clf.decision_function(X_grid_scaled)
    y_hat_grid = y_hat.reshape(X_1.shape)

    # Plot decision boundary
    plt.figure(figsize=(10, 6), dpi=100)
    plt.contour(X_1, X_2, y_hat_grid, colors=['b', 'k', 'r'], levels=[-1, 0, 1], linestyles=['--', '-', '--'])
    plt.contourf(X_1, X_2, y_hat_grid, cmap=plt.cm.coolwarm, alpha=0.5)
    plt.colorbar(label="Decision function value")

    # Plot origional data-points in PCA-space
    for label in sorted(y_train.unique()):
        plt.scatter(X_PC[y_train == label, 0], X_PC[y_train == label, 1], label=f"Class {label}", edgecolor='white', s=60)

    # support vectors
    support_vectors_PC = pca.transform(clf.support_vectors_)
    plt.scatter(support_vectors_PC[:, 0], support_vectors_PC[:, 1],
                s=100, facecolors='none', edgecolors='black', label='Support Vectors')
    # Plot options
    plt.xlabel('PC 1')
    plt.ylabel('PC 2')
    plt.title('Decision Boundary and Support Vectors in PCA-space')
    plt.legend()
    plt.grid(True)
    plt.show()


linear_data()
non_linear_data()
```