```
#import plotly.express as px
#df = px.data.iris()
#fig = px.scatter_3d(df, x='sepal_length', y='petal_length', z='petal_width',
#                    color='species')
#fig.show()
```

```
# USANDO PCA DE SCIKITLEARN
```

```
# Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Datos
# https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data
df_wine = pd.read_csv('https://bit.ly/3L1ZZI4', header=None)
df_wine.shape
```

```
(178, 14)
```

```
# Separar datos de entrenamiento y prueba
from sklearn.model_selection import train_test_split
X, y = df_wine.iloc[:,1:].values, df_wine.iloc[:,0].values
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)
# estadarizamos
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
X_train_std.shape, X_test_std.shape
```

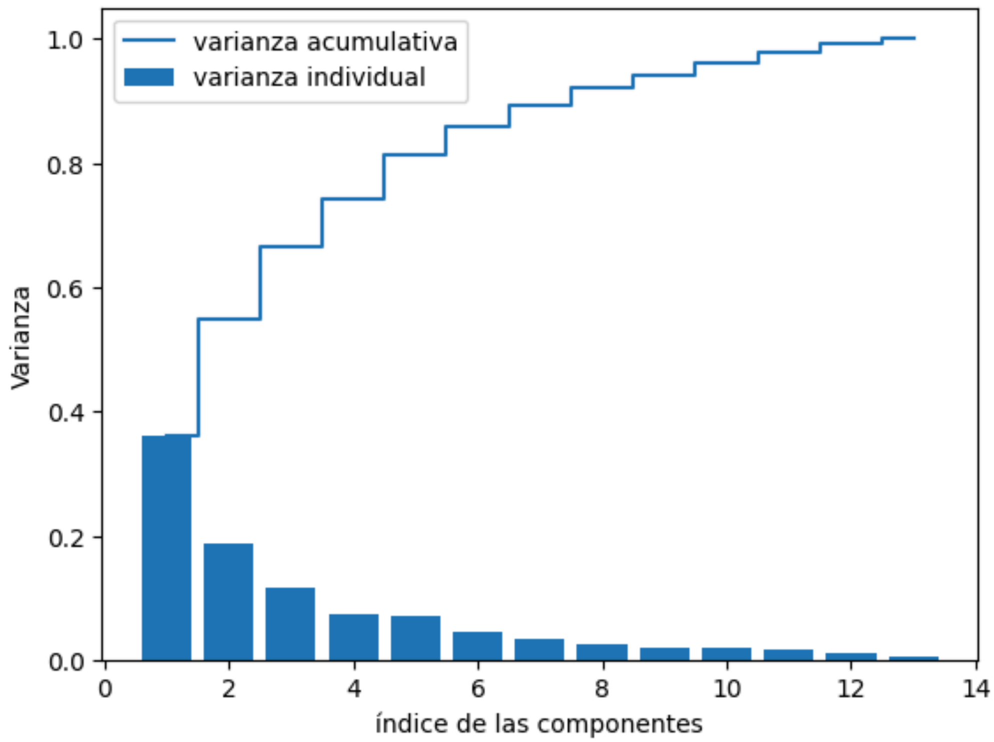
```
((124, 13), (54, 13))
```

```
# Determinar número de componentes a conservar
from sklearn.decomposition import PCA
pca = PCA(n_components=13)
pca.fit(X_train_std)
pca.explained_variance_, pca.explained_variance_ratio_
```

```
(array([4.74376552, 2.45913372, 1.5276711 , 0.99327678, 0.92313257,
        0.59663887, 0.46974164, 0.34681782, 0.28504118, 0.25665489,
        0.23096439, 0.17349645, 0.09935613]),
 array([0.36196226, 0.18763862, 0.11656548, 0.07578973, 0.07043753,
        0.04552517, 0.03584257, 0.02646315, 0.02174942, 0.01958347,
        0.01762321, 0.01323825, 0.00758114]))
```

```
# gráfica con los aportes de cada componente
import matplotlib.pyplot as plt
tot = sum(pca.explained_variance_ratio_)
var_exp = [ev/tot for ev in sorted(pca.explained_variance_ratio_,reverse=True)]
cum_var_exp = np.cumsum(var_exp)
plt.bar(range(1,14),var_exp,label='varianza individual',align='center')
plt.step(range(1,14),cum_var_exp,where='mid',label='varianza acumulativa')
plt.xlabel('índice de las componentes')
plt.ylabel('Varianza')
```

```
plt.legend(loc='best')
plt.show()
```



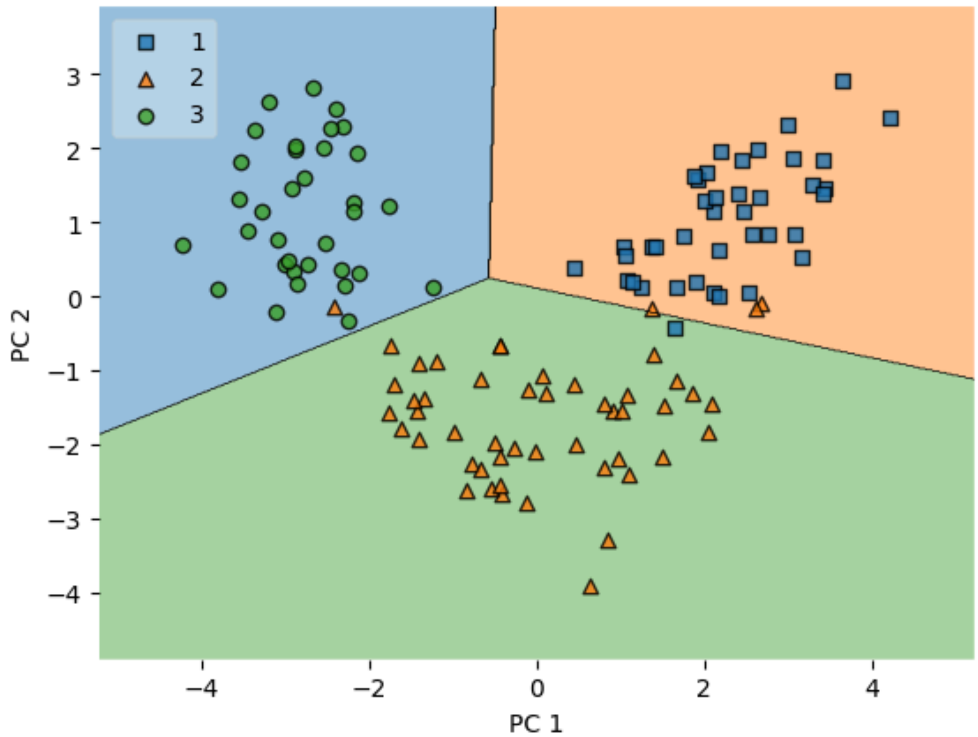

```
# Conservar 2 y revisar con regresión logística
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
lr = LogisticRegression()
```

```
# objetos de PCA y LR
# ajustar y transformar
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
lr.fit(X_train_pca, y_train)
```

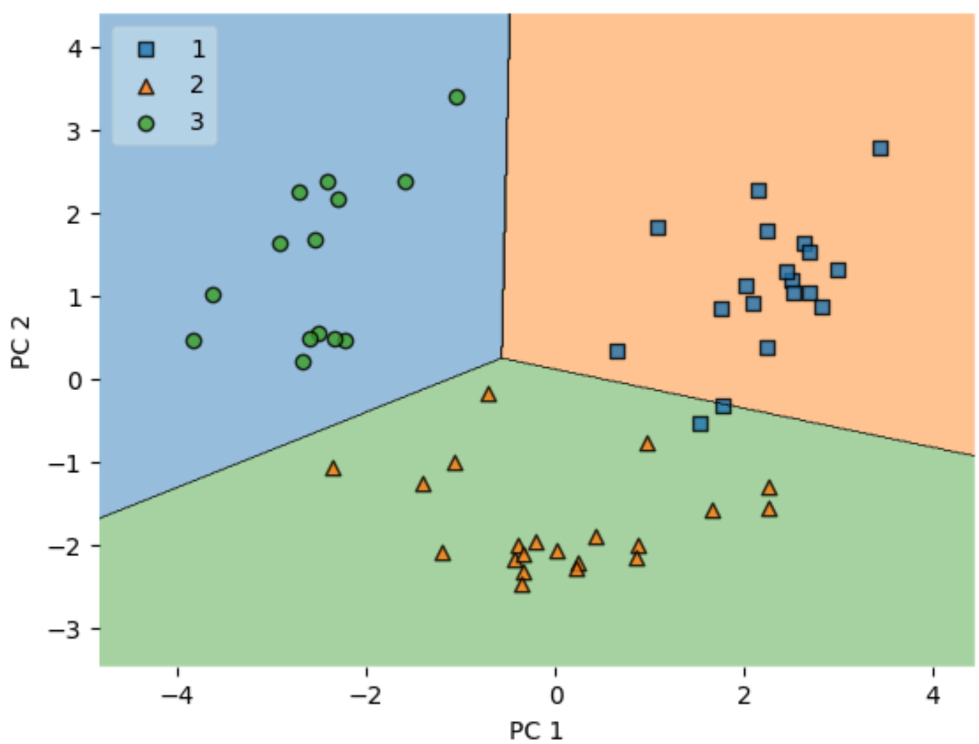
```
▾ LogisticRegression
LogisticRegression()
```

```
#!pip install mlxtend
```

```
# Grafica del conjunto de entrenamiento
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_train_pca, y_train, clf=lr, legend=2)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()
```




```
# Grafica del conjunto de prueba
plot_decision_regions(X_test_pca, y_test, clf=lr, legend=2)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()
```




```
# LDA CON PYTHON
```

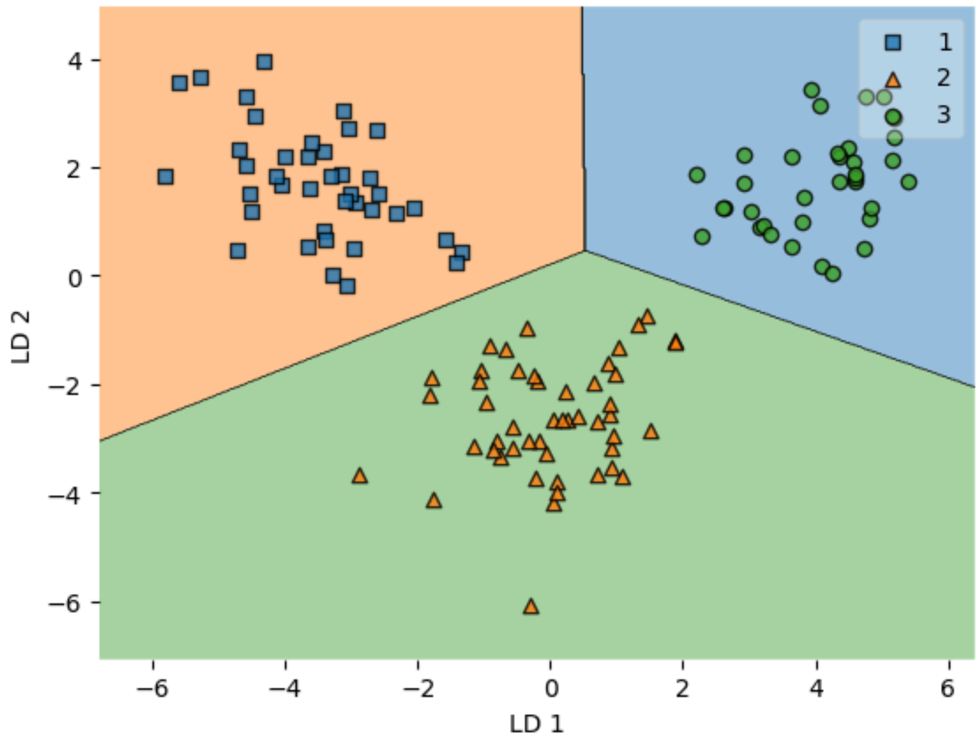
```
# Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Datos
df_wine = pd.read_csv('https://bit.ly/3L1ZZI4', header=None)
```

```
# Separar datos de entrenamiento y prueba
from sklearn.model_selection import train_test_split
X, y = df_wine.iloc[:,1:].values, df_wine.iloc[:,0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# estadarizamos
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

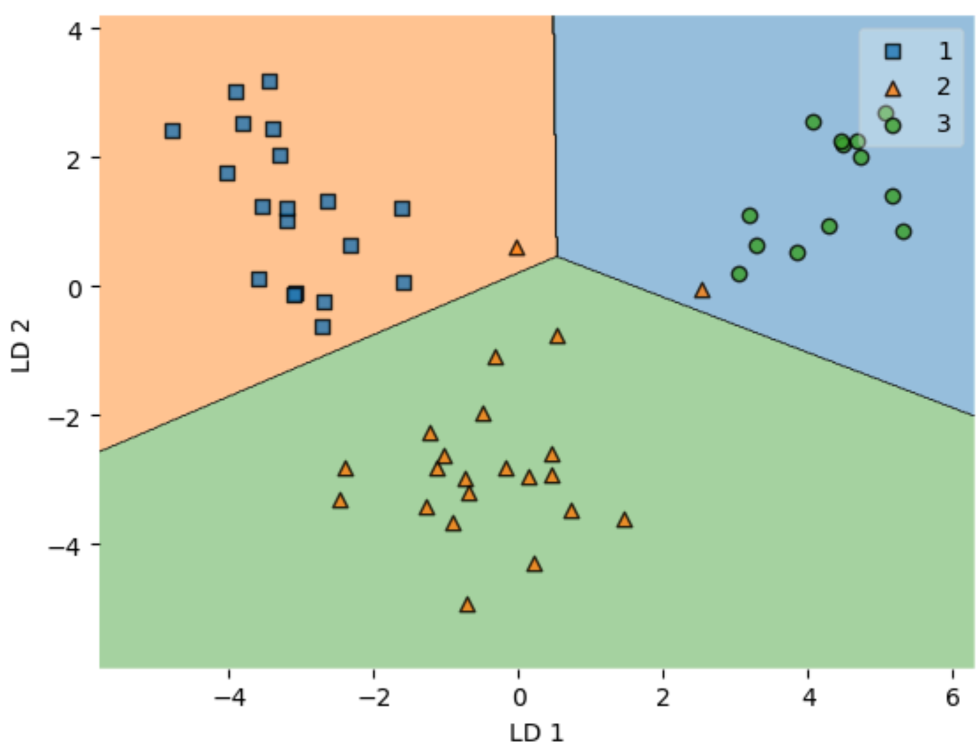
```
# Revisar con regresión logística
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
# inicializar LDA y el modelo de RL
lda = LDA(n_components=2)
lr = LogisticRegression()
# Ajustar y transformar los datos
X_train_lda = lda.fit_transform(X_train_std, y_train)
lr.fit(X_train_lda,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
# Grafica del conjunto de entrenamiento
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_train_lda, y_train, clf=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.show()
```

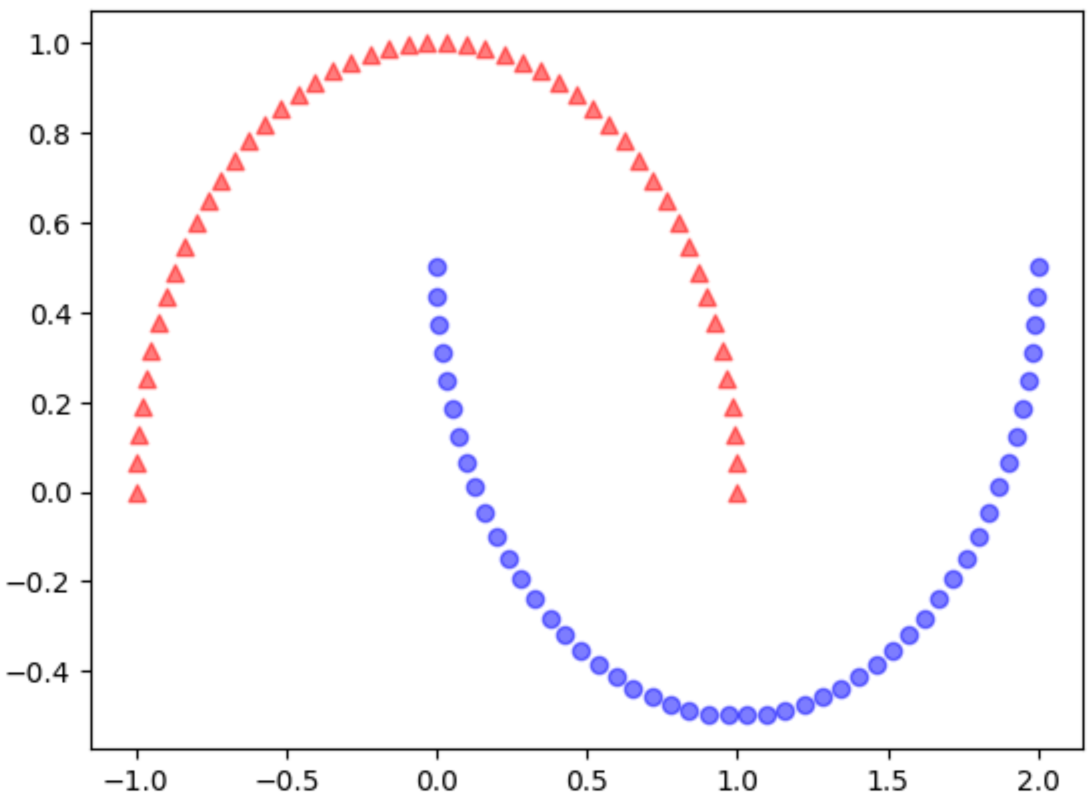



```
# Grafica del conjunto de pruebas
X_test_lda = lda.transform(X_test_std)
plot_decision_regions(X_test_lda, y_test, clf=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.show()
```



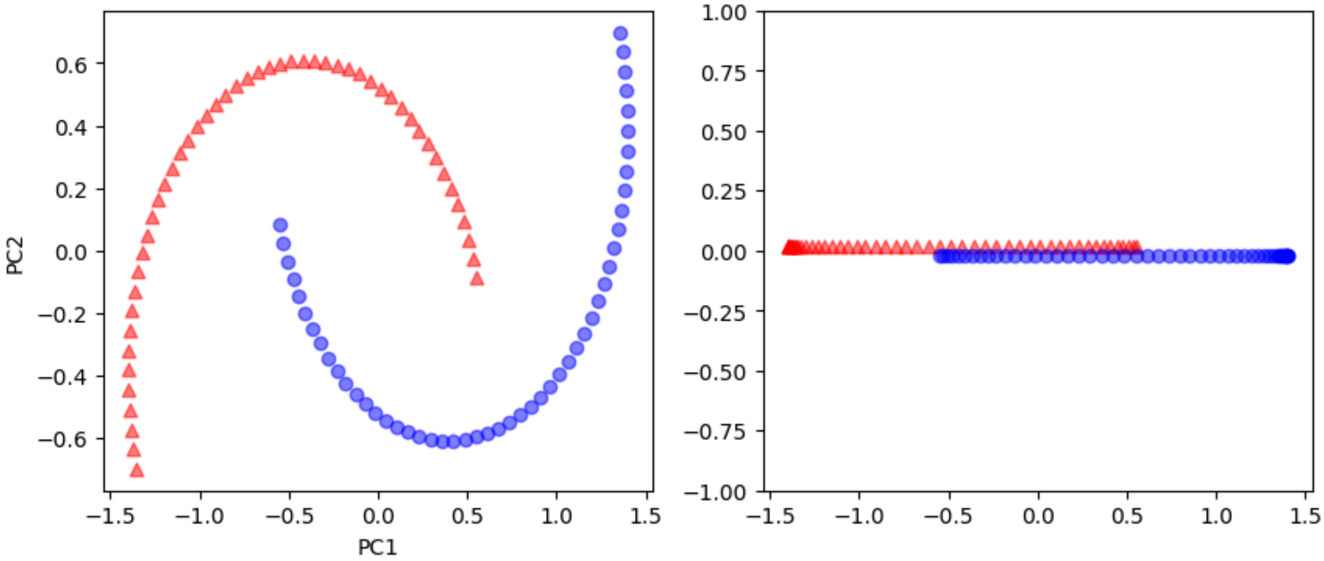
```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Datos: Separar medias lunas
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, random_state=123)
plt.scatter(X[y==0, 0], X[y==0, 1], color='red', marker='^', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', marker='o', alpha=0.5)
plt.show()
```




```
# Con PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

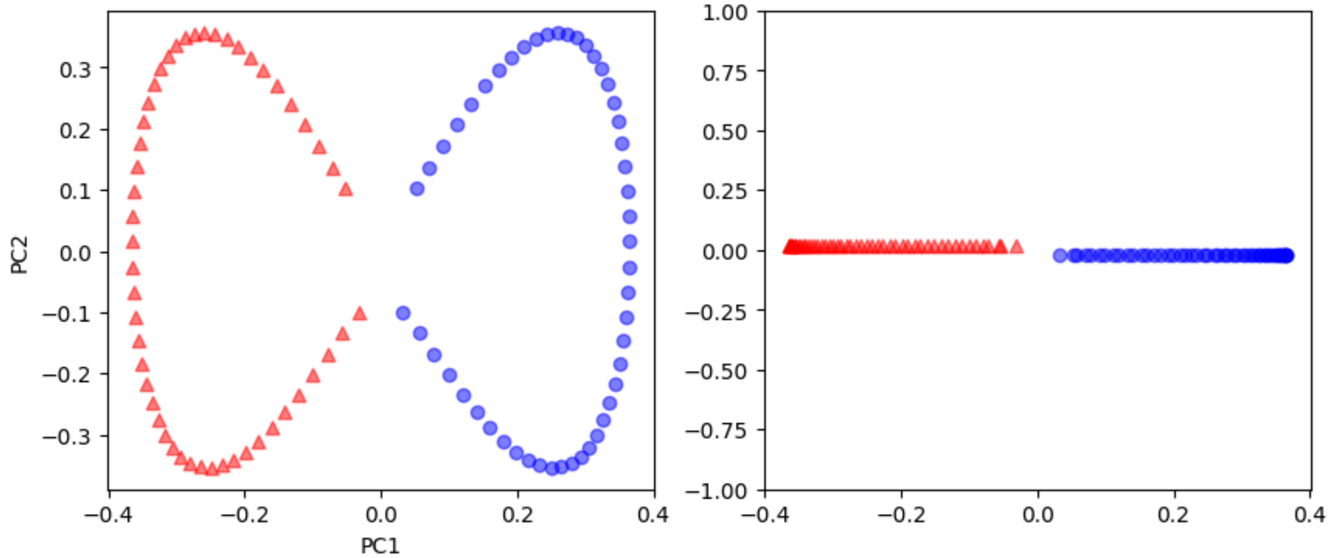
```
fig, ax = plt.subplots(nrows=1, ncols=2,figsize=(10,4))
ax[0].scatter(X_pca[y==0,0],X_pca[y==0,1],color='red',marker='^',alpha=0.5)
ax[0].scatter(X_pca[y==1,0],X_pca[y==1,1],color='blue',marker='o',alpha=0.5)
ax[1].scatter(X_pca[y==0,0],np.zeros((50,1))+0.02,color='red',marker='^',alpha=0
ax[1].scatter(X_pca[y==1,0],np.zeros((50,1))-0.02,color='blue',marker='o',alpha=
ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1,1])
ax[0].set_xlabel('PC1')
plt.show()
```



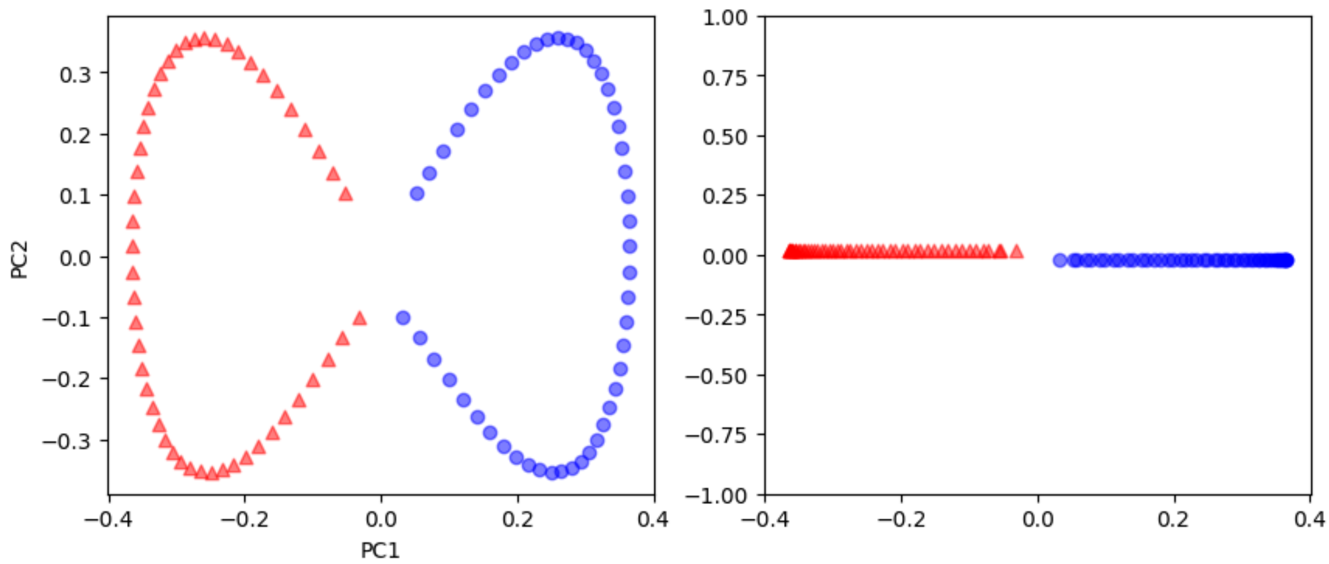

```
# Datos: Separar medias lunas
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, random_state=123)
```

```
# Con KPCA de Python
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components=2, kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X)
```

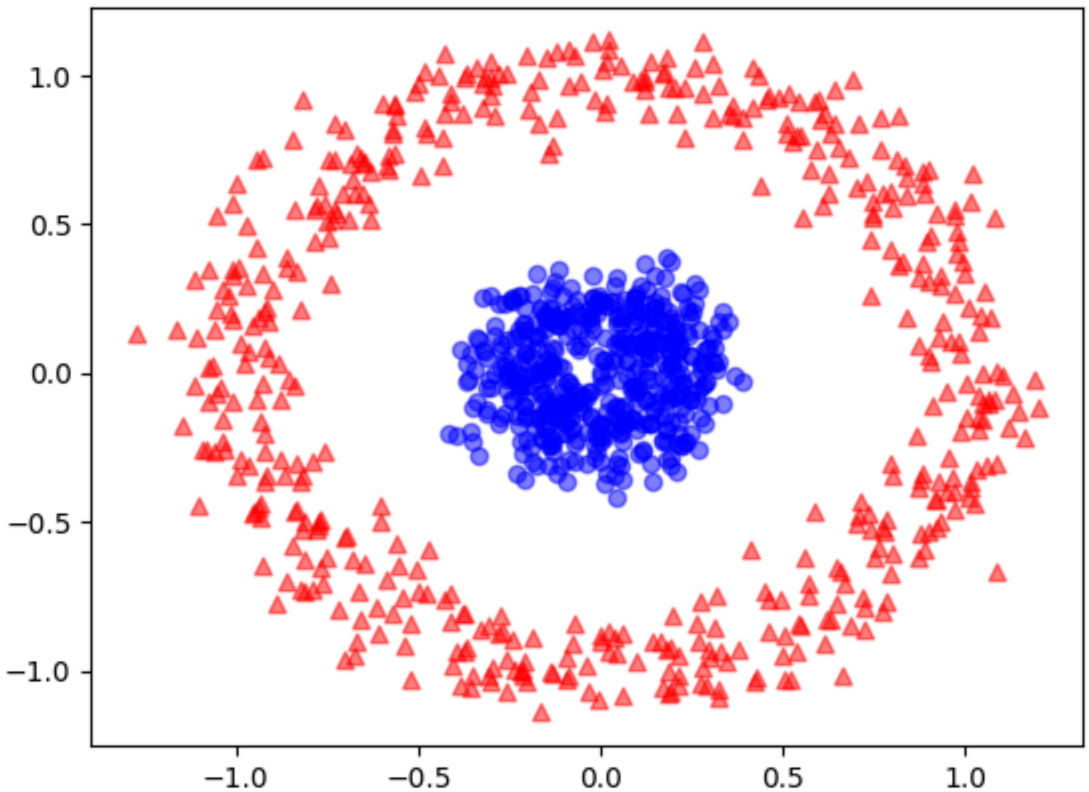
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
ax[0].scatter(X_kpca[y==0,0],X_kpca[y==0,1],color='red',marker='^',alpha=0.5)
ax[0].scatter(X_kpca[y==1,0],X_kpca[y==1,1],color='blue',marker='o',alpha=0.5)
ax[1].scatter(X_kpca[y==0,0],np.zeros((50,1))+0.02,color='red',marker='^',alpha=
ax[1].scatter(X_kpca[y==1,0],np.zeros((50,1))-0.02,color='blue',marker='o',alpha
ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1,1])
ax[0].set_xlabel('PC1')
plt.show()
```




```python
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components=2, kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X)
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
ax[0].scatter(X_kpca[y==0,0],X_kpca[y==0,1],color='red',marker='^',alpha=0.5)
ax[0].scatter(X_kpca[y==1,0],X_kpca[y==1,1],color='blue',marker='o',alpha=0.5)
ax[1].scatter(X_kpca[y==0,0],np.zeros((50,1))+0.02,color='red',marker='^',alpha=
ax[1].scatter(X_kpca[y==1,0],np.zeros((50,1))-0.02,color='blue',marker='o',alpha
ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1,1])
ax[0].set_xlabel('PC1')
plt.show()
```




```
# Datos: círculos concéntricos
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=1000, random_state=123, noise=0.1, factor=0.2)
plt.scatter(X[y==0, 0], X[y==0, 1], color='red', marker='^', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', marker='o', alpha=0.5)
plt.show()
```




```
# Intentar con PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
# Código para graficar
plt.show()
```

```
# Con KPCA de Python
from sklearn.decomposition import KernelPCA
#kpca = KernelPCA()
#X_kpca = kpca.fit_transform(X)
```

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
# Código para graficar
plt.show()
```