In [1]:
```
# Handling Linearly Inseparable Classes Using Kernels
```

In [2]:
```
# Importing Libraries
```

In [3]:
```
from sklearn.svm import SVC
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
import numpy as np
```

In [4]:
```
# Set randomization seed
```

In [4]:
```
np.random.seed(0)
```

In [6]:
```
# Generate two features
```

In [5]:
```
features = np.random.randn(200, 2)
```

In [8]:
```
# Use a XOR gate to generate linearly inseparable classes
```

In [6]:
```
target_xor = np.logical_xor(features[:, 0] > 0, features[:, 1] > 0)
target = np.where(target_xor, 0, 1)
```

In [7]:
```
# Create a support vector machine with a radial basis function kernel
svc = SVC(kernel="rbf", random_state=0, gamma=1, C=1)
```

In [8]:
```
# Train the classifier
model = svc.fit(features, target)
```

In [13]:
```
#Plot observations and decision boundary hyperplane
```

In [9]:
```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
```

In [11]:
```
def plot_decision_regions(X, y, classifier):
    cmap = ListedColormap(("red", "blue"))
    xx1, xx2 = np.meshgrid(np.arange(-3, 3, 0.02), np.arange(-3, 3, 0.02))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.1, cmap=cmap)
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker="+", label=cl)
```

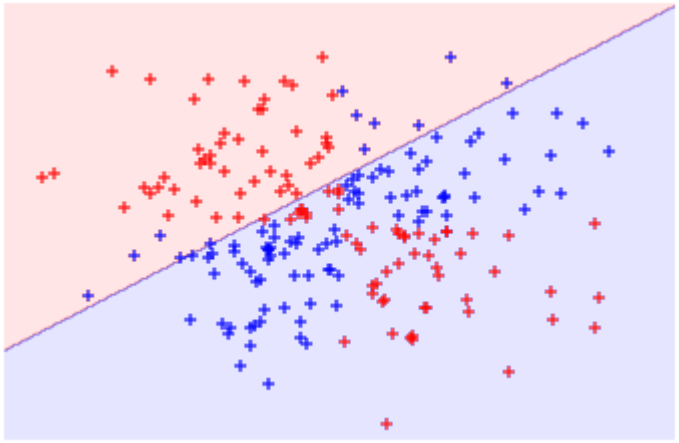
In [12]:
```
# Create support vector classifier with a linear kernel
svc_linear = SVC(kernel="linear", random_state=0, C=1)
```

In [13]:
```
# Train model
svc_linear.fit(features, target)
```

Out[13]:
```
SVC(C=1, kernel='linear', random_state=0)
```

In [20]:
```
# Plot observations and hyperplane
plot_decision_regions(features, target, classifier=svc_linear)
plt.axis("off"), plt.show();
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping wil
l have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or pr
ovide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping wil
l have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or pr
ovide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

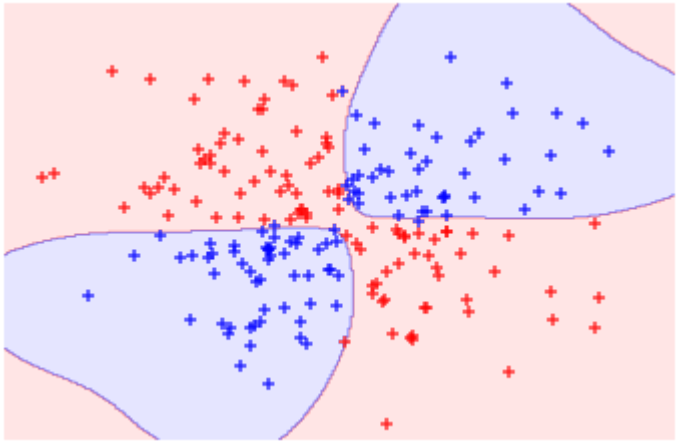


In [14]:
```
# Create a support vector machine with a radial basis function kernel
svc = SVC(kernel="rbf", random_state=0, gamma=1, C=1)
```

In [15]:
```
# Train the classifier
model = svc.fit(features, target)
```

In [23]:
```
1 plot_decision_regions(features, target, classifier=svc)
2 plt.axis("off"), plt.show();
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping wil
l have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or pr
ovide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping wil
l have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or pr
ovide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [ ]:
```
1
```