In [1]:
```
# 19IT016: Manav Butani
```

WEEK09 : SVM

Classifying Muffins and Cupcakes with SVM

Based on the quantity of ingredients added into the recipes. We come up with two different types of snacks one is Muffin and second one is cupcake.

Refrences: https://youtu.be/N1vOgolbjSc

Step 1: Import Packages

In [2]:
```
# Packages for analysis
import pandas as pd
import numpy as np
from sklearn import svm

# Packages for visuals
import matplotlib.pyplot as plt
import seaborn as sns

# Allows charts to appear in the notebook
%matplotlib inline
```

Step 2: Import Data

In [3]:
```
# Read in muffin and cupcake ingredient data
recipes = pd.read_csv('recipes_muffins_cupcakes.csv')
recipes
```

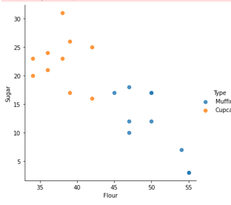
Out[3]:

	Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
0	Muffin	55	28	3	7	5	2	0	0
1	Muffin	47	24	12	6	9	1	0	0
2	Muffin	47	23	18	6	4	1	0	0
3	Muffin	45	11	17	17	8	1	0	0
4	Muffin	50	25	12	6	5	2	1	0
5	Muffin	55	27	3	7	5	2	1	0
6	Muffin	54	27	7	5	5	2	0	0
7	Muffin	47	26	10	10	4	1	0	0
8	Muffin	50	17	17	8	6	1	0	0
9	Muffin	50	17	17	11	4	1	0	0
10	Cupcake	39	0	26	19	14	1	1	0
11	Cupcake	42	21	16	10	8	3	0	0
12	Cupcake	34	17	20	20	5	2	1	0
13	Cupcake	39	13	17	19	10	1	1	0
14	Cupcake	38	15	23	15	8	0	1	0
15	Cupcake	42	18	25	9	5	1	0	0
16	Cupcake	36	14	21	14	11	2	1	0
17	Cupcake	38	15	31	8	6	1	1	0
18	Cupcake	36	16	24	12	9	1	1	0
19	Cupcake	34	17	23	11	13	0	1	0

Step 3: Prepare the Data

In [4]:
```
# Plot two ingredients
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', fit_reg=False);
```

```
C:\Users\Manav\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From versi
on 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misi
nterpretation.
  warnings.warn(
```



In [5]:
```
# Specify inputs for the model
# ingredients = recipes[['Flour', 'Milk', 'Sugar', 'Butter', 'Egg', 'Baking Powder', 'Vanilla', 'Salt']].as_matrix()
ingredients = recipes[['Flour','Sugar']].to_numpy()
type_label = np.where(recipes['Type']=='Muffin', 0, 1)
```

Step 4: Fit the Model

In [6]:
```
# Fit the SVM model
model = svm.SVC(kernel='linear')
model.fit(ingredients, type_label)
```

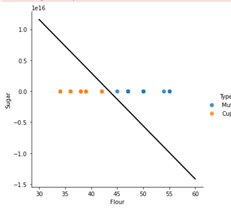
Out[6]:
```
SVC(kernel='linear')
```

Step 5: Visualize Results

In [7]:
```
# Get the separating hyperplane
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(30, 60)
yy = a * xx - (model.intercept_[0]) / w[1]

# Plot the parallels to the separating hyperplane that pass through the support vectors
b = model.support_vectors_[0]
yy_down = a * xx + (b[1] - a * b[0])
b = model.support_vectors_[-1]
yy_up = a * xx + (b[1] - a * b[0])
```

In [8]:
```
# Plot the hyperplane
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', fit_reg=False)
plt.plot(xx, yy, linewidth=2, color='black');
```

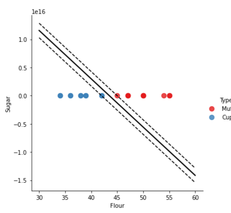
```
C:\Users\Manav\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From versi
on 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misi
nterpretation.
  warnings.warn(
```



In [9]:
```
# Look at the margins and support vectors
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', palette='Set1', fit_reg=False, scatter_kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
#plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1]);
```

```
C:\Users\Manav\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From versi
on 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misi
nterpretation.
  warnings.warn(
[<matplotlib.lines.Line2D at 0x203388938b0>]
```

Out[9]:



Step 6: Predict New Case

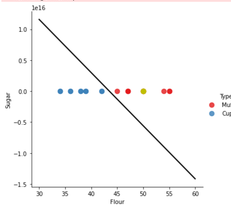
In [10]:
```
# Create a function to guess when a recipe is a muffin or a cupcake
def muffin_or_cupcake(flour, sugar):
    if(model.predict([[flour, sugar]]))==0:
        print('You\'re looking at a muffin recipe!')
    else:
        print('You\'re looking at a cupcake recipe!')
```

In [11]:
```
# Predict if 50 parts flour and 20 parts sugar
muffin_or_cupcake(50, 20)
```

```
You're looking at a muffin recipe!
```

In [12]:
```
# Plot the point to visually see where the point lies
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', palette='Set1', fit_reg=False, scatter_kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
plt.plot(50, 20, 'yo', markersize='9');
```

```
C:\Users\Manav\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From versi
on 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misi
nterpretation.
  warnings.warn(
```



In [13]:
```
# Predict if 40 parts flour and 20 parts sugar
muffin_or_cupcake(40,20)
```

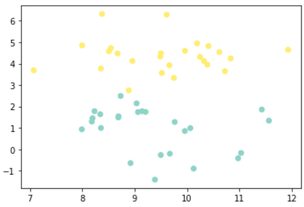
```
You're looking at a cupcake recipe!
```

WEEK09_SVM_synthetic_dataset,_iris_dataset

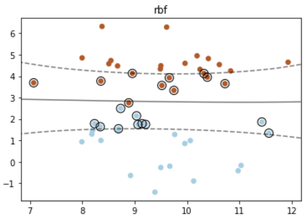
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_blobs
from sklearn import datasets
```

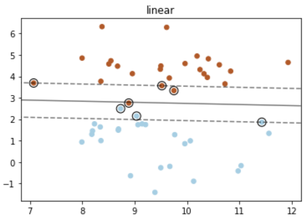
In [27]:
```
X, y = make_blobs(n_samples=50, centers=2, random_state=4)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set3)
print(X.shape,y.shape)
```

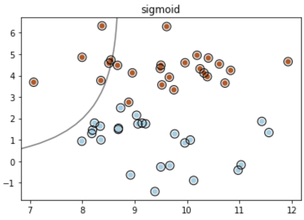
```
(50, 2) (50,)
```

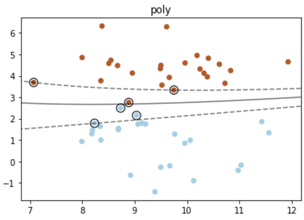


In [28]:
```
for x in {'linear', 'poly', 'rbf', 'sigmoid'}:
  # fit the model, don't regularize for illustration purposes
  clf = SVC(kernel=x, C=0.70)
  clf.fit(X, y)

  plt.title(x)
  plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
  # plot the decision function
  ax = plt.gca()
  xlim = ax.get_xlim()
  ylim = ax.get_ylim()

  # create grid to evaluate model
  xx = np.linspace(xlim[0], xlim[1], 30)
  yy = np.linspace(ylim[0], ylim[1], 30)
  YY, XX = np.meshgrid(yy, xx)
  xy = np.vstack([XX.ravel(), YY.ravel()]).T
  Z = clf.decision_function(xy).reshape(XX.shape)

  # plot decision boundary and margins
  ax.contour(
      XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-", "--"]
  )
  # plot support vectors
  ax.scatter(
      clf.support_vectors_[:, 0],
      clf.support_vectors_[:, 1],
      s=100,
      linewidth=1,
      facecolors="none",
      edgecolors="k",
  )
  plt.show()
```









In [4]:
```
# import some data to play with
iris = datasets.load_iris()
#iris
```

In [5]:
```
iris.data.shape
```

Out[5]:
```
(150, 4)
```

In [6]:
```
# Take the first two features. We could avoid this by using a two-dim dataset
X_train = iris.data[:120, :2]
y_train = iris.target[:120]
X_test = iris.data[120:, :2]
y_test = iris.target[120:]
print(X_train.shape,y_train.shape, X_test.shape,y_test.shape)
```

```
(120, 2) (120,) (30, 2) (30,)
```

In [8]:
```
from sklearn.metrics import accuracy_score
C = 1.0  # SVM regularization parameter
models = (
    SVC(kernel="linear", C=C),
    SVC(C=C, max_iter=10000),
    SVC(kernel="rbf", gamma=0.7, C=C),
    SVC(kernel="poly", degree=3, gamma="auto", C=C),
)
models = (clf.fit(X_train, y_train) for clf in models)

for model in models:
  print(accuracy_score(model.predict(X_test),y_test))
```

```
0.2
0.2
0.2
0.2
```