Assignment 1

สมาชิกกลุ่ม

1. จิตติณณ์ จินดานรเศรษฐ์ 6110405949 (หัวหน้ากลุ่ม)
2. รินลณี วัชรนิมมานกุล 6110401633
3. กัญญาณัฐ อินทรโชติ 6110402737
4. ณัฐณิชา คงสุนทร 6110402753
5. ธนกฤต โลจันทร์ติ 6110402770
6. นิรัติศัย คงศักดิ์ 6110406066
7. พิมพ์ลภัส ตันธนกุล 6110406171
8. สุรยุทธ์ บุญคล้าย 6110406252

การออกแบบการทดลอง

ขั้นตอนการออกแบบการทดลองมีดังนี้

1. ดูลักษณะของข้อมูลเพื่อทำการแก้ไขข้อมูลและจัดกลุ่มข้อมูล
2. แก้ไขข้อมูลที่ว่าง และทำการแก้ไขจัดกลุ่มข้อมูลที่มีความใกล้เคียงกันให้อยู่กลุ่มเดียวกัน
3. ทำการปรับตารางโดยตารางมี features ทั้งหมดที่ต้องการให้ machine เรียนรู้ และทำการทดสอบ

วิธีการ Preprocess

1. ติดตั้ง Anaconda Navigator เพื่อใช้เครื่องมือ Jupyter Notebook ในการออกแบบการทดลอง โดยเรียกใช้ Library ชื่อ Pandas ในการจัดการข้อมูล
2. ทำการ import library ที่ต้องการใช้งานเข้ามา

```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plot
        from sklearn import preprocessing
        from collections import Counter
        stdsc = preprocessing.StandardScaler()
```

3. อ่านไฟล์ indian_food.csv

```
In [2]: df = pd.read_csv('indian_food.csv')
```

4. จัดการกับข้อมูลส่วนประกอบของแต่ละเมนู โดยในขั้นตอนแรกปรับให้ตัวหนังสือชื่อส่วนประกอบ เป็นตัวเล็กทั้งหมด และในส่วนของคอลัมน์ flavour_profile, state, region ที่ข้อมูลมีค่าเป็น -1 หรือ NaN ให้ทำการ replace ให้เป็น ‘other’

In [3]:
```
df.ingredients = df.ingredients.str.lower()
df.flavor_profile = df.flavor_profile.replace('-1', 'other')
df.state = df.state.replace('-1', 'other')
df.region = df.region.replace('-1', 'Other')
df.region = df.region.fillna('Other')
```

5. ทำการเปลี่ยนแปลงชื่อส่วนผสมที่มีความคล้ายคลึงกัน/ประเภทเดียวกันให้เหมือนกันทั้งหมดเพื่อทำการจัดกลุ่มส่วนผสม โดยใช้ regular expression string เข้ามาช่วยหากลุ่มส่วนผสมที่คล้ายกันและใช้ replace method ในการเปลี่ยนแปลงค่า

In [9]:
```
df.ingredients = df.ingredients.replace({'coconut milk': 'coconut'}, regex=True)
df.ingredients = df.ingredients.replace({'mixed nuts': 'nuts'}, regex=True)
df.ingredients = df.ingredients.replace({' and ': ', '}, regex=True)
df.ingredients = df.ingredients.replace({'red pepper': 'bell pepper'}, regex=True)
df.ingredients = df.ingredients.replace({'hot water': 'water'}, regex=True)
df.ingredients = df.ingredients.replace({'greens': 'green'}, regex=True)
df.ingredients = df.ingredients.replace({'yellow mustard': 'mustard'}, regex=True)
df.ingredients = df.ingredients.replace({'mustard seed': 'mustard seeds'}, regex=True)
```

In [10]:
```
df.ingredients = df.ingredients.replace({'[^,]*milk[^,]*': ' milk'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*masala[^,]*': ' masala'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]* oil[^,]*': ' oil'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*coconut[^,]*': ' coconut'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*flour[^,]*': ' flour'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*carrot[^,]*': ' carrot'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*dal[^,]*': ' dal'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*sugar[^,]*': ' sugar'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*curry[^,]*': ' curry'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]* chilli[^,]*': ' chillies'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*potato[^,]*': ' potato'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*yogurt[^,]*': ' yogurt'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*tomato[^,]*': ' tomato'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*rice[^,]*': ' rice'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*salt[^,]*': ' salt'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*cream[^,]*': ' cream'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*rava[^,]*': ' rava'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*garlic[^,]*': ' garlic'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*ginger[^,]*': ' ginger'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*sesame[^,]*': ' sesame'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*peanut[^,]*': ' peanuts'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*chicken[^,]*': ' chicken'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*lemon[^,]*': ' lemon'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*cinnamon[^,]*': ' cinnamon'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*bell pepper[^,]*': ' bell pepper'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*pigeon peas[^,]*': ' pigeon peas'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*green peas[^,]*': ' green peas'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*tamarind[^,]*': ' tamarind'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*jaggery[^,]*': ' jaggery'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*pork[^,]*': ' pork'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*cardamom[^,]*': ' cardamom'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*rose[^,]*': ' rose'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*beans[^,]*': ' beans'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*fish fillet[^,]*': ' fish'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*daal[^,]*': ' dal'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*almond[^,]*': ' almonds'}, regex=True).str.strip()
df.ingredients = df.ingredients.replace({'[^,]*onion[^,]*': ' onions'}, regex=True).str.strip()
```

6. ทำการจัดรูปแบบข้อมูลโดยทำตารางที่มีเฉพาะคอลัมน์ชื่ออาหารและส่วนประกอบของอาหารเท่านั้น (food_ingredient) ซึ่งลักษณะของตารางนี้ในหนึ่งแถวจะเป็นการจับคู่ชื่ออาหารและส่วนประกอบหนึ่งอย่าง โดยอาหารหนึ่งชนิดจะมีจำนวนแถวเท่ากับจำนวนส่วนประกอบของอาหารนั้น ๆ

```
In [14]: df = df.set_index('name')
```

```
In [15]: food_ingredient =df.ingredients.str.split(', ', expand=True).reset_index().melt(id_vars=['name'])
         food_ingredient = food_ingredient.drop('variable', axis=1)
         food_ingredient.columns = ['name', 'ingredient']
         food_ingredient = food_ingredient.dropna()
```

```
In [16]: food_ingredient
```

Out[16]:

	name	ingredient
0	Balu shahi	flour
1	Boondi	flour
2	Gajar ka halwa	carrot
3	Ghevar	flour
4	Gulab jamun	milk
...	...	...
1823	Bandar laddu	sugar
1830	Mysore pak	ghee
2043	Ghevar	saffron
2046	Jalebi	cardamom
2298	Ghevar	cardamom

1138 rows × 2 columns

7. ทำการ pivot table โดยให้ชื่ออาหารเป็น index และนำส่วนประกอบอาหารทั้งหมดปรับเป็นคอลัมน์ (food_ingredient_hot_encode)

```
In [18]: food_ingredient_hot_encode = food_ingredient.pivot_table(index=['name'], columns=['ingredient'], aggfunc=[len], fill_value=0).droplevel(0, axis=1)
```

8. นำเอา food_ingredient_hot_encode มา join เข้ากับตาราง df กับคอลัมน์ flavour_profile, diet โดยตั้งชื่อตารางใหม่นี้ว่า food เพื่อนำมาใช้ในการวัดค่าประสิทธิภาพ โดยที่ในตาราง food จะเป็น feature ที่เราจะให้ machine นั้นใช้ในการเรียนรู้และทำนายผล ซึ่ง features (คอลัมน์) ที่ใช้คือ diet, flavour_profile และ ingredients ที่ผ่านการจัดกลุ่มแล้วทั้งหมด

```
In [19]: food = df.loc[:,['diet', 'flavor_profile']].join(food_ingredient_hot_encode)

In [20]: food
```

Out[20]:

name	diet	flavor_profile	almonds	aloo	alum powder	amaranth leaves	amchur powder	apricots	arbi ke patte	arrowroot powder	...	water	watercress	white bread slices	whole egg	whole red	whole wheat bread
Balu shahi	vegetarian	sweet	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
Boondi	vegetarian	sweet	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
Gajar ka halwa	vegetarian	sweet	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
Ghevar	vegetarian	sweet	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0
Gulab jamun	vegetarian	sweet	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Til Pitha	vegetarian	sweet	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
Bebinca	vegetarian	sweet	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
Shufta	vegetarian	sweet	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
Mawa Bati	vegetarian	sweet	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0
Pinaca	vegetarian	sweet	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

255 rows × 214 columns

การวัดค่าประสิทธิภาพ

โดยเราจะแบ่ง dataset เป็นสองส่วนคือ train dataset 90% และ test dataset 10% โดยที่ X จะเป็น feature และ y เป็นผลเฉลยดังภาพด้านล่าง

```
In [21]: from sklearn.model_selection import train_test_split

In [22]: X = pd.get_dummies(food)
         y = df.course
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=45)
```

หาค่า hyperparameter ที่เหมาะสมสำหรับ Decision tree โดยใช้ GridSearchCV โดยกำหนด max_depth และ critrerion ตามที่ต้องการดังภาพ In[24] ด้านล่าง ต่อมาทำการหาค่ากลุ่ม hyperparameter ที่ให้ผลลัพธ์ที่ดีที่สุดออกมาในภาพ In[27]

```
In [23]: from sklearn.model_selection import GridSearchCV

In [24]: param_grid = {
          'max_depth' : [2,3,4,5,6,7,8,9,10],
          'criterion' :['gini', 'entropy']
         }

In [25]: from sklearn.tree import DecisionTreeClassifier

In [26]: DTree = DecisionTreeClassifier(random_state=0)
         CV_DTree = GridSearchCV(estimator=DTree, param_grid=param_grid, cv= 2)
         CV_DTree.fit(X_train, y_train)

Out[26]: GridSearchCV(cv=2, error_score=nan,
                      estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                                       criterion='gini', max_depth=None,
                                                       max_features=None,
                                                       max_leaf_nodes=None,
                                                       min_impurity_decrease=0.0,
                                                       min_impurity_split=None,
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_leaf=0.0,
                                                       presort='deprecated',
                                                       random_state=0, splitter='best'),
                      iid='deprecated', n_jobs=None,
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring=None, verbose=0)

In [27]: CV_DTree.best_params_

Out[27]: {'criterion': 'gini', 'max_depth': 3}
```

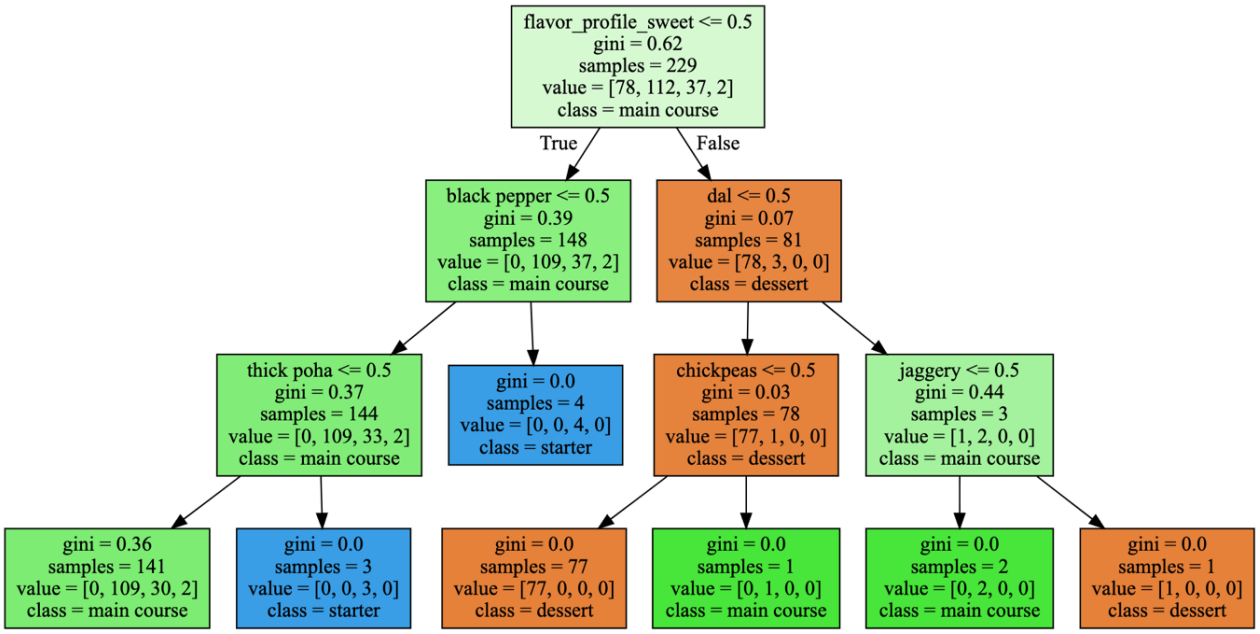
โดยเมื่อปรับค่า hyperparameter ตาม GridSearchCV จะให้ค่าประสิทธิภาพคือ 0.9230769230769231

```
In [28]: DTree = DecisionTreeClassifier(random_state=0, max_depth=3, criterion='gini')
         DTree.fit(X_train, y_train)
         DTree.score(X_test,y_test)

Out[28]: 0.9230769230769231
```

ภาพด้านล่างคือลักษณะของ Decision tree ที่ได้จากการทดลอง

```
In [29]: from graphviz import Source
         from sklearn.tree import export_graphviz

In [30]: Source(export_graphviz(DTree, out_file=None,
                         feature_names=X.columns, class_names=df.course.unique(),
                         filled=True, precision=2))
```

Out[30]:



จากการทดลองผลลัพธ์จากขั้นตอนวิธี Naïve bays

แบบวิธี gaussian ได้ค่าประสิทธิภาพคือ 0.8461538461538461

```
In [31]: from sklearn.naive_bayes import GaussianNB

         gnb = GaussianNB().fit(X_train,y_train)
         gnb.score(X_test,y_test)
```

Out[31]: 0.8461538461538461

แบบวิธี Multinomial ได้ค่าประสิทธิภาพคือ 0.8846153846153846

```
In [32]: from sklearn.naive_bayes import MultinomialNB

         mnb = MultinomialNB().fit(X_train,y_train)
         mnb.score(X_test,y_test)
```

Out[32]: 0.8846153846153846

แบบวิธี Bernoulli ได้ค่าประสิทธิภาพคือ 0.8846153846153846

In [33]:
```
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB().fit(X_train,y_train)
bnb.score(X_test,y_test)
```

Out[33]: 0.8846153846153846

<u>สรุปผลการทดลอง</u>

จากการ Preprocess ข้อมูลและผลการทดลองวัดค่าประสิทธิภาพทั้งของ Decision tree และ Naïve bays ได้ผลการทดลองว่าแบบวิธี Decision tree ได้ค่าความแม่นยำมากที่สุดเมื่อเทียบกับแบบขั้นตอนวิธี Naïve bays โดยที่ค่าประสิทธิภาพของ Decision tree ที่วัดได้คือ 0.9230769230769231 และค่าประสิทธิภาพที่วัดได้จากวิธี Naïve bays ที่ดีที่สุดที่วัดได้คือ 0.8846153846153846

โดยการปรับค่า hyperparameter ใน Decision tree นั้นได้ใช้ GridSearchCV เข้ามาช่วยในการหาค่ากลุ่ม hyperparameter ที่จะให้ผลลัพธ์ดีที่สุดออกมา