Lab 1

1. **Write a Python program to implement Standard Scalar**.

```
import numpy as np
import pandas as pd

class StandardNorm:
  def scale(self, df):
    for i in df.columns:
      mean = df[i].mean()
      sd = df[i].std()
      df[i] = (df[i] - mean) / sd
    return df

df = pd.DataFrame(
  [[45000, 42], [32000, 26], [58000, 48], [37000, 32]], columns=["Salary", "Age"]
)
print("Original Data")
print(df)

s = StandardNorm()
df_scaled = s.scale(df)

print("\nScaled Data")
print(df_scaled)
```

Output:

```
Original Data
   Salary  Age
0   45000   42
1   32000   26
2   58000   48
3   37000   32

Scaled Data
     Salary       Age
0  0.176318  0.506803
1 -0.969750 -1.114967
2  1.322386  1.114967
3 -0.528954 -0.506803
```

2. **Write a Python program to implement Min-max Scalar.**

```
import numpy as np
import pandas as pd


class MinMaxNorm:
   def scale(self, df):
      for c in df.columns:
         min = df[c].min()
         max = df[c].max()
         df[c] = (df[c] - min) / (max - min)
      return df


df = pd.DataFrame(
   [[45000, 42], [32000, 26], [58000, 48], [37000, 32]], columns=["Salary", "Age"]
)
print("Original Data")
print(df)

s = MinMaxNorm()
df_scaled = s.scale(df)

print("\nScaled Data")
print(df_scaled)
```

Output:

```
Original Data
   Salary  Age
0   45000   42
1   32000   26
2   58000   48
3   37000   32

Scaled Data
     Salary       Age
0  0.500000  0.727273
1  0.000000  0.000000
2  1.000000  1.000000
3  0.192308  0.272727
```

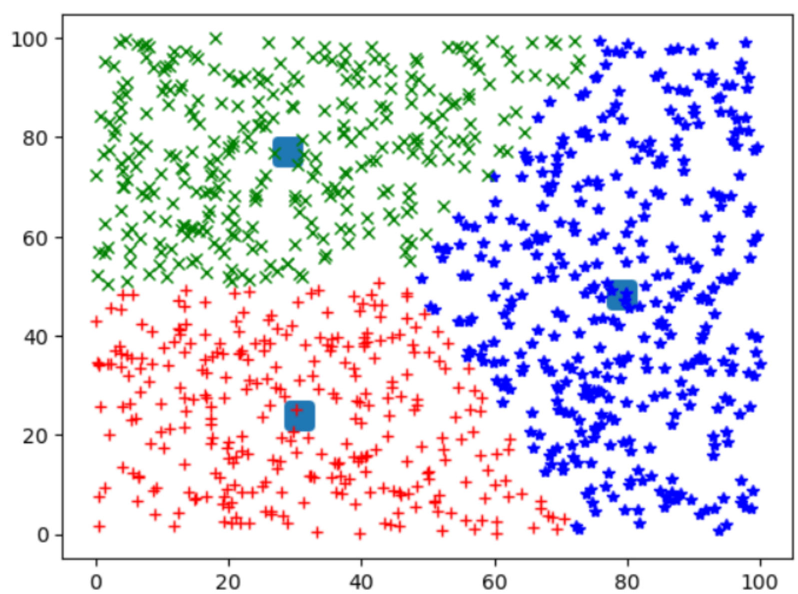
Lab2

1. **Write a python program to implement K-means Clustering algorithm**
   Generate 1000 2-D data points in the range 0-100 randomly.
   Divide data points into 3 clusters.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
data = np.random.rand(1000, 2) * 100
km = KMeans(n_clusters=3, init="random")
km.fit(data)

centers = km.cluster_centers_
labels = km.labels_

print("Cluser centers: ", *centers)
# print("Cluser Labels: ", *labels)

colors = ["r", "g", "b"]
markers = ["+", "x", "*"]

for i in range(len(data)):
  plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])
plt.scatter(centers[:, 0], centers[:, 1], marker="s", s=100, linewidths=5)
plt.show()
```

Output:

```
Cluser centers:  [30.72181821 23.86735429] [28.90842522 77.22777665] [79.15135085 48.24788716]
```



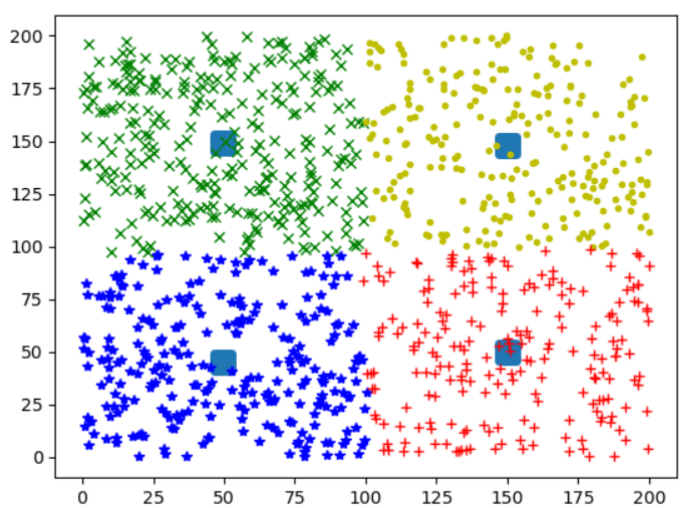
2. Write a python program to implement K-means++ Clustering algorithm.

Generate 1000 2-D data points in the range 0-200 randomly. Divide data points into 4 clusters.

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = np.random.rand(1000, 2) * 200

km = KMeans(n_clusters=4, init="k-means++")

km.fit(data)

centers = km.cluster_centers_

labels = km.labels_

print("Cluser centers: ", *centers)

# print("Cluser Labels: ", *labels)

colors = ["r", "g", "b", "y"]

markers = ["+", "x", "*", "."]

for i in range(len(data)):

    plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])

plt.scatter(centers[:, 0], centers[:, 1], marker="s", s=100, linewidths=5)

plt.show()
```

Output:

```
Cluser centers:  [150.20184899  49.64536974] [ 49.47398044 148.86694587] [49.69534141 44.95752842] [150.10489965 147.64049192]
```



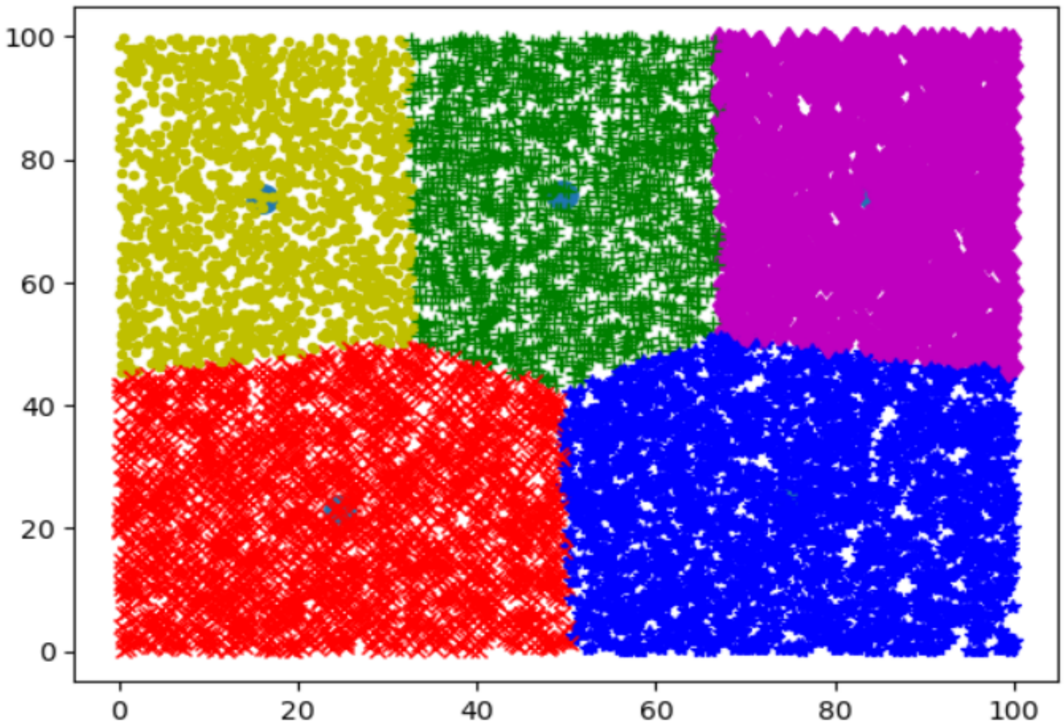
Lab3

1. **Write a python program to implement K-means Clustering algorithm**
   Generate 10000 2-D data points in the range 0-100 randomly
   Divide data points into 5 clusters
   Find time taken by the algorithm to find clusters

```
import time
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data = np.random.rand(10000, 2) * 100

km = KMeans(n_clusters=5, init="random")

t0 = time.process_time()
km.fit(data)
t1 = time.process_time()

tt = t1 - t0
print("Total Time:", tt)

centers = km.cluster_centers_
labels = km.labels_

print("Cluster Centers:", centers)
# print("Cluster Labels:", *labels)

colors = ["g", "r", "b", "y", "m"]
markers = ["+", "x", "*", ".", "d"]

for i in range(len(data)):
   plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])
plt.scatter(centers[:, 0], centers[:, 1], marker="o", s=50, linewidths=5)
plt.show()
```

Output:

```
Total Time: 0.125
Cluster Centers: [[49.38988409 74.12591804]
 [24.5502947  22.9444225 ]
 [74.99404744 24.30180105]
 [16.06925158 73.64079691]
 [84.34244621 74.37691799]]
```



2. Write a python program to implement Mini-batch K-means Clustering algorithm

Generate 10000 2-D data points in the range 0-100 randomly

Divide data points into 5 clusters

Find time taken by the algorithm to find clusters

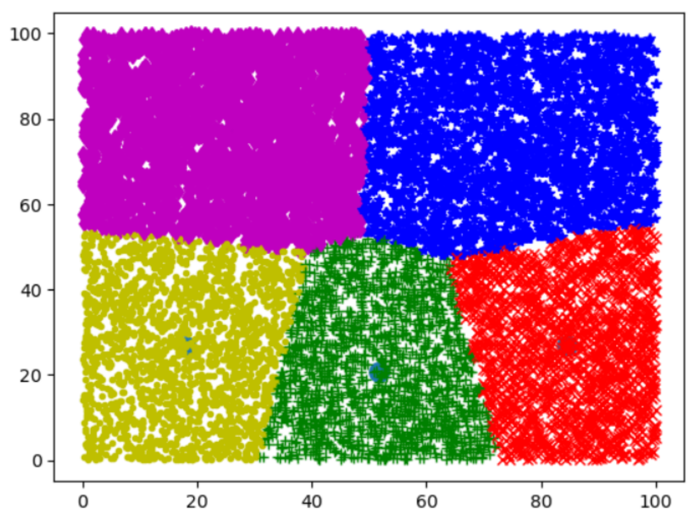
Vary the batch size from 100 to 1500, find time taken by the algorithm in each case and find best value of the batch size.

```
import time
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans
data = np.random.rand(10000, 2) * 100
mbk = MiniBatchKMeans(n_clusters=5, init="random", batch_size=500)
```

```
t0 = time.time()
mbk.fit(data)
t1 = time.time()
tt = t1 - t0
print("Total Time: ", tt)
centers = mbk.cluster_centers_
labels = mbk.labels_
print("Cluster Centers:", centers)
# print("Labels:", labels)
colors = ["g", "r", "b", "y", "m"]
markers = ["+", "x", "*", ".", "d"]
for i in range(len(data)):
    plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])
plt.scatter(centers[:, 0], centers[:, 1], marker="o", s=50, linewidths=5)
plt.show()
```

Output:

```
Total Time:  1.0215404033660889
Cluster Centers: [[51.76967451 20.69774959]
 [84.56597679 26.97321835]
 [73.0273725  74.94755   ]
 [18.03431697 26.67771513]
 [25.27104429 76.36583797]]
```

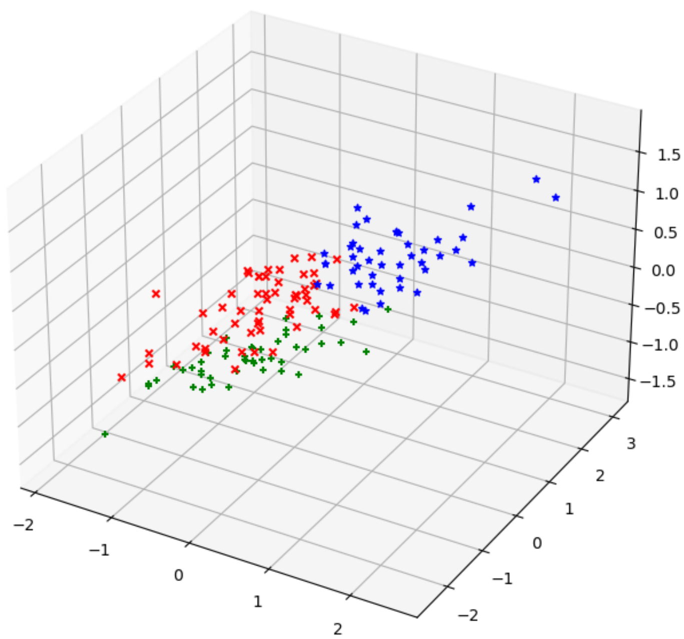


Lab4

1. **Write a python program to find clusters of Iris Dataset using KMedoids Algorithm**

```
!pip install scikit-learn-extra

from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn_extra.cluster import KMedoids
from sklearn import metrics
import matplotlib.pyplot as plt

iris_data = load_iris()

x = iris_data.data
y = iris_data.target

# print(x[:5])
# print(y[:5])
sc = StandardScaler().fit(x)
sx = sc.transform(x)
km = KMedoids(n_clusters=3)
km.fit(sx)
py = km.fit_predict(sx)
# print("Predicted: ", py)
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection="3d")
colors = ["g", "r", "b"]
markers = ["+", "x", "*"]
for i in range(len(sx)):
   ax.scatter(sx[i][0], sx[i][1], sx[i][2], color=colors[py[i]], marker=markers[py[i]])
plt.show()
ri = metrics.rand_score(y, py)
print("Rand Index:", ri)

hs = metrics.homogeneity_score(y, py)
print("Homogeniety Score:", hs)

cs = metrics.completeness_score(y, py)
print("Completeness Score:", cs)

sc = metrics.silhouette_score(sx, py, metric="euclidean")
print("Silhouette Coefficient:", sc)
```

Output:



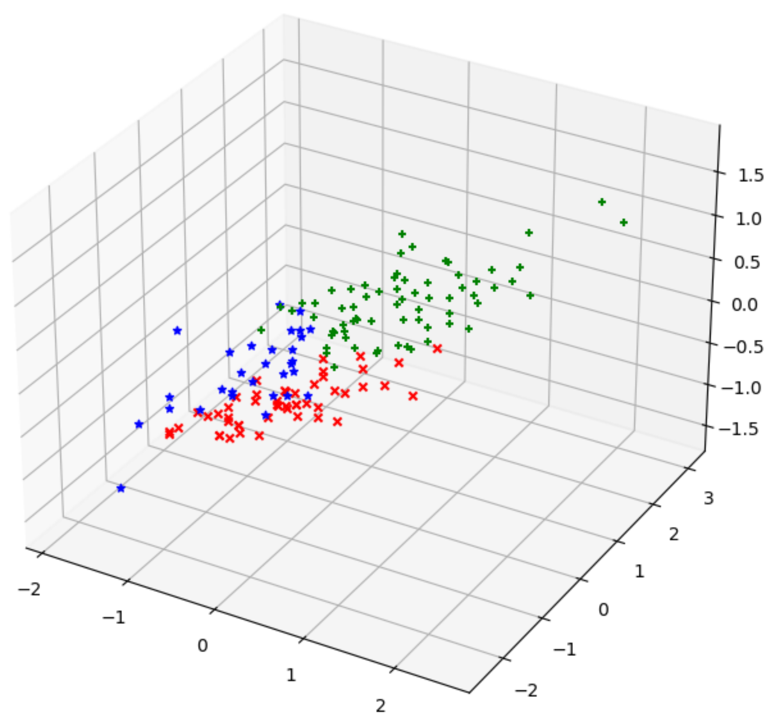

```
Rand Index: 0.8367785234899329
Homogeniety Score: 0.6672491406379297
Completeness Score: 0.6701843437329579
Silhouette Coefficient: 0.4590416105554613
```

2. **Write a python program to find clusters of Iris Dataset using Agglomerative Clustering Algorithm.**

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn import metrics
import matplotlib.pyplot as plt
iris_data = load_iris()
x = iris_data.data
y = iris_data.target
# print(x[:5])
# print(y[:5])
sc = StandardScaler().fit(x)
sx = sc.transform(x)
ac = AgglomerativeClustering(n_clusters=3)
ac.fit(sx)

py = ac.fit_predict(sx)
# print("Predicted: ", py)
```

```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection="3d")
colors = ["g", "r", "b"]
markers = ["+", "x", "*"]
for i in range(len(sx)):
    ax.scatter(sx[i][0], sx[i][1], sx[i][2], color=colors[py[i]], marker=markers[py[i]])
plt.show()

ri = metrics.rand_score(y, py)
print("Rand Index:", ri)

hs = metrics.homogeneity_score(y, py)
print("Homogeniety Score:", hs)

cs = metrics.completeness_score(y, py)
print("Completeness Score:", cs)

sc = metrics.silhouette_score(sx, py, metric="euclidean")
print("Silhouette Coefficient:", sc)
```

Output:




```
Rand Index: 0.8252348993288591
Homogeniety Score: 0.6578818079976051
Completeness Score: 0.6940248415952218
Silhouette Coefficient: 0.4466890410285909
```

Lab 5

1. **Write a python program to predict diabetes using Naive Bayes Classification.**

```
import pandas as pd
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB

dataset = pd.read_csv("Diabetes.csv")

print("Dataset Size: ", len(dataset))

split = int(len(dataset) * 0.7)
train, test = dataset.iloc[:split], dataset.iloc[split:]

p = train["Pragnency"].values
g = train["Glucose"].values
bp = train["Blod Pressure"].values
st = train["Skin Thikness"].values
ins = train["Insulin"].values
bmi = train["BMI"].values
dpf = train["DFP"].values
a = train["Age"].values
d = train["Diabetes"].values

trainfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)
traininput = list(trainfeatures)
# print(traininput)

model = GaussianNB()
model.fit(traininput, d)

p = test["Pragnency"].values
g = test["Glucose"].values
bp = test["Blod Pressure"].values
st = test["Skin Thikness"].values
ins = test["Insulin"].values
bmi = test["BMI"].values
dpf = test["DFP"].values
a = test["Age"].values
d = test["Diabetes"].values

testfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)
testinput = list(testfeatures)
```

```
predicted = model.predict(testinput)
# print('Actual Class:', *d)
# print('Predicted Class:', *predicted)

print("Confusion Matrix:")
print(metrics.confusion_matrix(d, predicted))

print("\nClassification Measures:")
print("Accuracy:", metrics.accuracy_score(d, predicted))
print("Recall:", metrics.recall_score(d, predicted))
print("Precision:", metrics.precision_score(d, predicted))
print("F1-score:", metrics.f1_score(d, predicted))
```

Output:

```
Dataset Size:  767
Confusion Matrix:
[[128  24]
 [ 30  49]]

Classification Measures:
Accuracy: 0.7662337662337663
Recall: 0.620253164556962
Precision: 0.6712328767123288
F1-score: 0.6447368421052632
```

2. **Write a python program to predict diabetes using ID3 Decision Tree Classifier.**

```
import pandas as pd
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
dataset = pd.read_csv("Diabetes.csv")
print("Dataset Size: ", len(dataset))
split = int(len(dataset) * 0.7)
train, test = dataset.iloc[:split], dataset.iloc[split:]
p = train["Pragnency"].values
g = train["Glucose"].values
bp = train["Blod Pressure"].values
st = train["Skin Thikness"].values
ins = train["Insulin"].values
bmi = train["BMI"].values
dpf = train["DFP"].values
a = train["Age"].values
d = train["Diabetes"].values
trainfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)
traininput = list(trainfeatures)
# print(traininput)
model = DecisionTreeClassifier(criterion="entropy", max_depth=4)
model.fit(traininput, d)
p = test["Pragnency"].values
```

```
g = test["Glucose"].values
bp = test["Blod Pressure"].values
st = test["Skin Thikness"].values
ins = test["Insulin"].values
bmi = test["BMI"].values
dpf = test["DFP"].values
a = test["Age"].values
d = test["Diabetes"].values
testfeatures = zip(p, g, bp, st, ins, bmi, dpf, a)
testinput = list(testfeatures)

predicted = model.predict(testinput)
# print('Actual Class:', *d)
# print('Predicted Class:', *predicted)
print("Confusion Matrix:")
print(metrics.confusion_matrix(d, predicted))
print("\nClassification Measures:")
print("Accuracy:", metrics.accuracy_score(d, predicted))
print("Recall:", metrics.recall_score(d, predicted))
print("Precision:", metrics.precision_score(d, predicted))
print("F1-score:", metrics.f1_score(d, predicted))
```

Output:

```
Dataset Size:  767
Confusion Matrix:
[[118  34]
 [ 17  62]]

Classification Measures:
Accuracy: 0.7792207792207793
Recall: 0.7848101265822784
Precision: 0.6458333333333334
F1-score: 0.7085714285714286
```

Lab6

1. Write a python program to implement Apriori algorithm to find association rules.

```
!pip install apyori
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori
dataset = pd.read_csv("store_data.csv", header=None)
# print(dataset)
records = []
for i in range(0, 7501):
  test = []
  data = dataset.iloc[i]
```

```
    data = data.dropna()
    for j in range(0, len(data)):
        test.append(str(dataset.values[i, j]))
    records.append(test)
# print(records)
association_rules = apriori(
    records, min_support=0.005, min_confidence=0.2, min_lift=3, min_length=2
)
association_results = list(association_rules)

for item in association_results:
    # print(item)
    # print(item[2])
    # print(item[2][0])
    print(list(item[2][0][0]), '->', list(item[2][0][1]))
```

Output:

```
Requirement already satisfied: apyori in e:\machinelearning\anaconda\lib\site-packages (1.1.2)
['mushroom cream sauce'] -> ['escalope']
['pasta'] -> ['escalope']
['herb & pepper'] -> ['ground beef']
['tomato sauce'] -> ['ground beef']
['whole wheat pasta'] -> ['olive oil']
['pasta'] -> ['shrimp']
['chocolate', 'frozen vegetables'] -> ['shrimp']
['spaghetti', 'frozen vegetables'] -> ['ground beef']
['shrimp', 'mineral water'] -> ['frozen vegetables']
['spaghetti', 'frozen vegetables'] -> ['olive oil']
['spaghetti', 'frozen vegetables'] -> ['shrimp']
['spaghetti', 'frozen vegetables'] -> ['tomatoes']
['spaghetti', 'grated cheese'] -> ['ground beef']
['herb & pepper', 'mineral water'] -> ['ground beef']
['spaghetti', 'herb & pepper'] -> ['ground beef']
['ground beef', 'shrimp'] -> ['spaghetti']
['spaghetti', 'milk'] -> ['olive oil']
['soup', 'mineral water'] -> ['olive oil']
['pancakes', 'spaghetti'] -> ['olive oil']
```