I have decided to make one small change to my analysis plan. In question #1, I ask what features are most contunous feature is most important to silhouette score. I initially had planned to remove groups features such as all the features with fat in them and other such groups for various models. To expand more, one clustering model would be fit on features that do not contain the word fat (so no calories from fat, total fat, total fat DV, saturated fat, saturated fat DV and trans fat) I had planned to remove a major feature for every model. After some thought, I have realized that it would only make sense to do this method for the major nutrients which are carbohydrates, protein and fat. I had initially planned to make 12 models and have every single one missing a related group of features.

Now, I realize this isn't a good use of time and it extremely computationally efficient since I would need to do a gridsearch on every single model to find the ideal parameters. My new plan for question number 1 is to only make three models. One would be missing all features with the word fat in them, one would be missing all features with carbohydrates and sugars in them and the final would be missing all features with the word protein in them.

I still feel that this plan for analysis for my first question will still discover which of the 3 feature groups is most important for cluster cohesion.

Additionally, for my second question, the question was too broad so I am changing it to picking the elbow as portrayed in a scree plot and computing accuracy metrics for that many components instead of what I originally had which was getting as close as I could to complete variance.

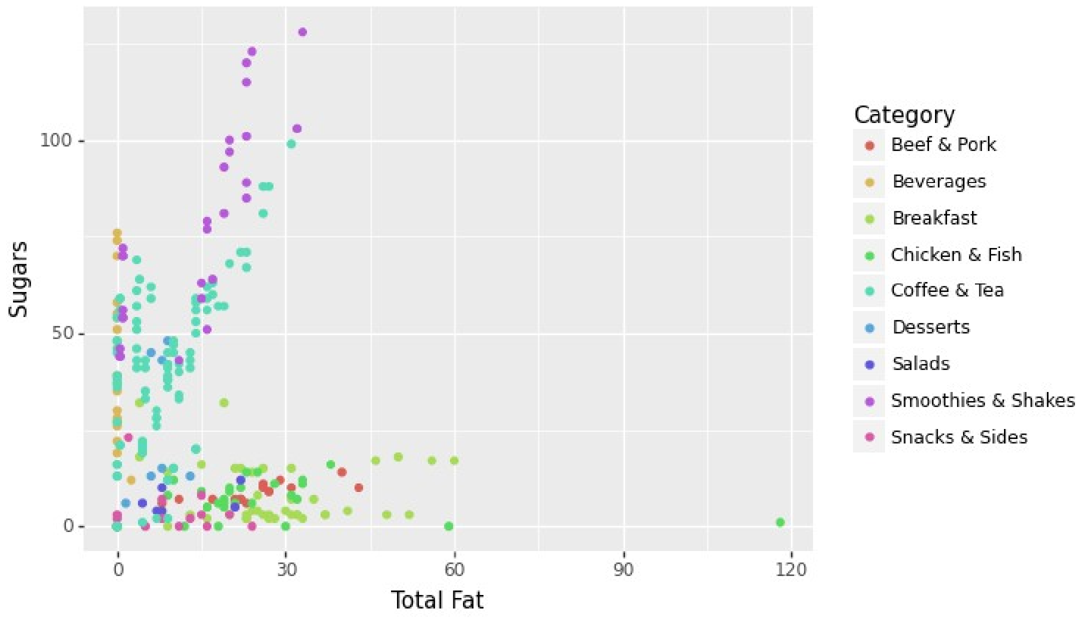
```
from plotnine import *
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split # simple TT split
cv
from sklearn.metrics import mean_squared_error, r2_score,
accuracy_score,mean_absolute_error,precision_score,recall_score,f1_sco
re #model evaluation
import numpy as np
import matplotlib.pyplot as plt
#from sklearn.metrics._plot.confusion_matrix import confusion_matrix,
plot_confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.neighbors import KNeighborsClassifier
```

```
import sklearn

sklearn.__version__
```

```
{"type":"string"}
```

```
Menu=pd.read_csv("menu.csv")
X=Menu[['Calories', 'Calories from Fat',
       'Total Fat', 'Total Fat (% Daily Value)', 'Saturated Fat',
       'Saturated Fat (% Daily Value)', 'Trans Fat', 'Cholesterol',
       'Cholesterol (% Daily Value)', 'Sodium', 'Sodium (% Daily
Value)',
       'Carbohydrates', 'Carbohydrates (% Daily Value)', 'Dietary
Fiber',
       'Dietary Fiber (% Daily Value)', 'Sugars', 'Protein',
       'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
       'Calcium (% Daily Value)', 'Iron (% Daily Value)']]
y=Menu['Category']
z=StandardScaler()
X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.2)
X_train = z.fit_transform(X_train)
X_test= z.transform(X_test)
```

```
#1

ggplot(Menu,aes(x='Total Fat',y='Sugars',color='Category'))
+geom_point()
```



```
<ggplot: (8752897132352)>
```

```
from sklearn.mixture import GaussianMixture
GM=GaussianMixture()
kparams = {"n_components":range(2,20)}
OrigX=Menu[['Calories', 'Calories from Fat',
       'Total Fat', 'Total Fat (% Daily Value)', 'Saturated Fat',
       'Saturated Fat (% Daily Value)', 'Trans Fat', 'Cholesterol',
       'Cholesterol (% Daily Value)', 'Sodium', 'Sodium (% Daily
Value)',
       'Carbohydrates', 'Carbohydrates (% Daily Value)', 'Dietary
Fiber',
       'Dietary Fiber (% Daily Value)', 'Sugars', 'Protein',
       'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
       'Calcium (% Daily Value)', 'Iron (% Daily Value)']]
gridkm1=GridSearchCV(GM,param_grid=kparams,cv=30,verbose=1,n_jobs=-1)
gridkm1.fit(OrigX)
```

```
Fitting 30 folds for each of 18 candidates, totalling 540 fits
```

```
GridSearchCV(cv=30, estimator=GaussianMixture(), n_jobs=-1,
             param_grid={'n_components': range(2, 20)}, verbose=1)
```

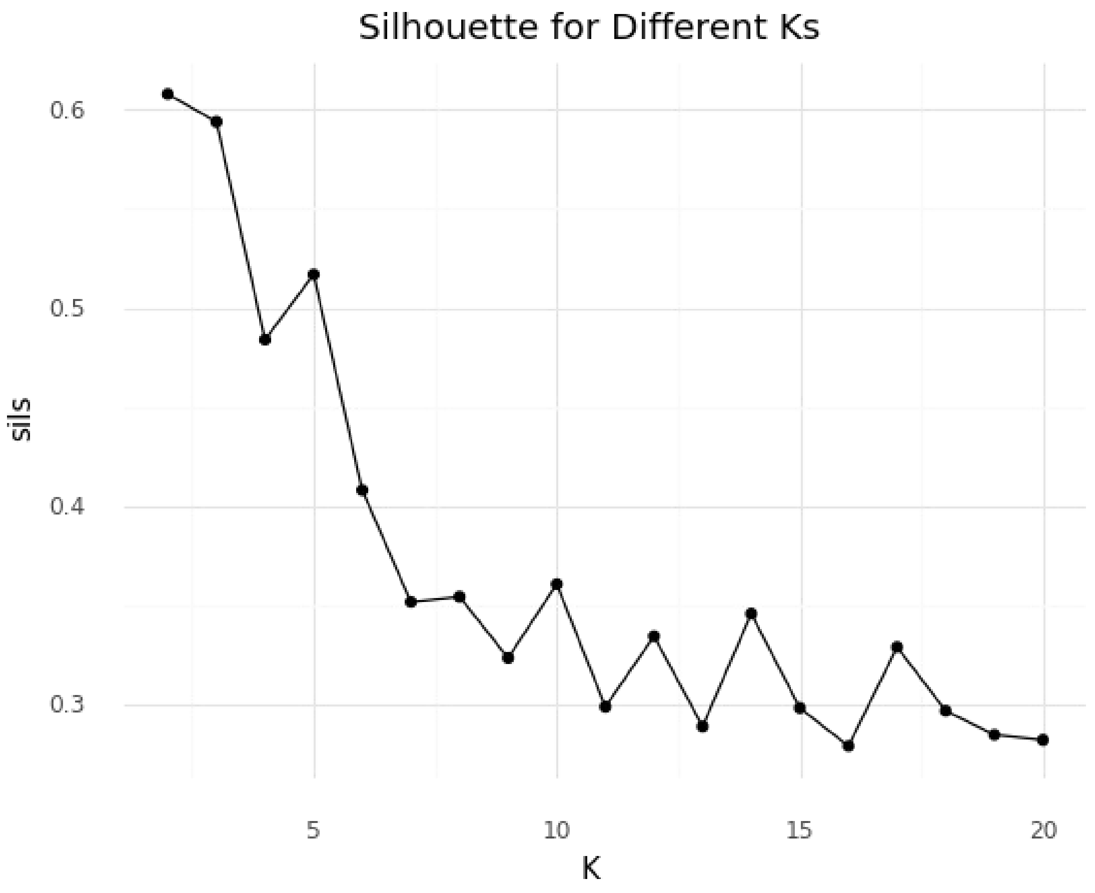
```
gridkm1.best_estimator_
```

```
GaussianMixture(n_components=2)
```

```
OriginalGM=GaussianMixture(n_components=2)
OriginalGM.fit(OrigX)
```

```
GaussianMixture(n_components=2)
```

```
ks = [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

sils = []

for k in ks:
  gmm = GaussianMixture(n_components = k)
  gmm.fit(OrigX)
  sils.append(silhouette_score(OrigX, gmm.predict(OrigX)))

sil_df = pd.DataFrame({"K": ks,"silhouette": sils})

(ggplot(sil_df, aes(x = "K", y = "sils")) + geom_point() +
geom_line() +
theme_minimal() +
labs(title = "Silhouette for Different Ks"))
```



```
<ggplot: (8752890158412)>
```

```
sil_df
```

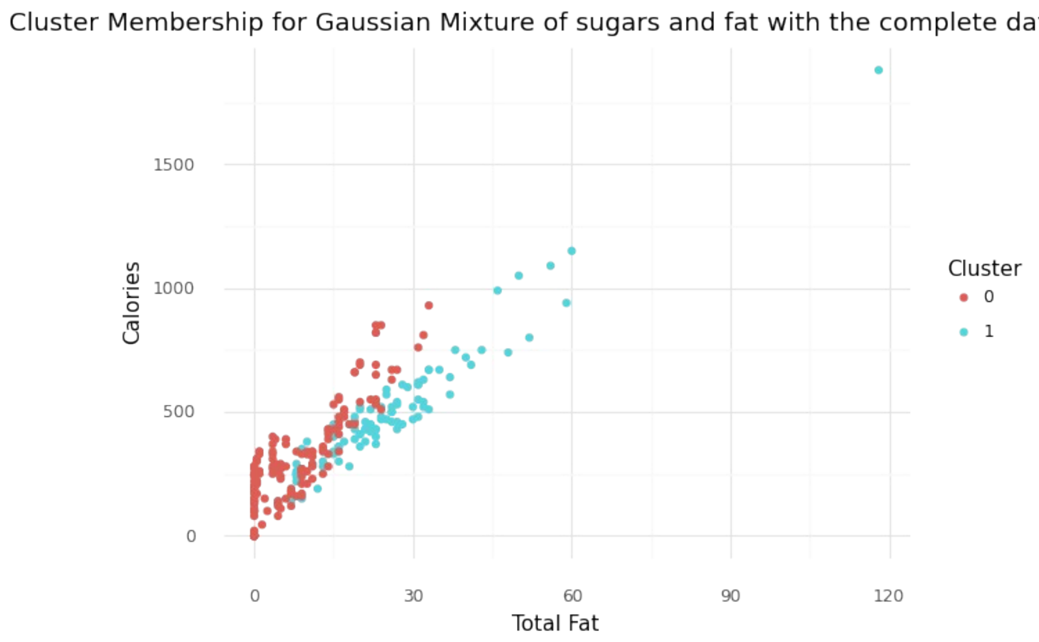
```
     K  silhouette
0    2    0.607773
1    3    0.593794
2    4    0.483900
3    5    0.516779
4    6    0.408067
5    7    0.351665
6    8    0.354298
7    9    0.323433
8   10    0.360630
9   11    0.298793
10  12    0.334389
11  13    0.288875
12  14    0.345736
13  15    0.298192
14  16    0.278984
15  17    0.328895
16  18    0.296725
```

```
17  19    0.284696
18  20    0.282058
```

```
OrigXpred=OriginalGM.predict(OrigX)
print("Silhouette score with the full dataset used in the
model:",silhouette_score(OrigX, OrigXpred))
OrigX["cluster"] = OrigXpred
```

```
Silhouette score with the full dataset used in the model:
0.6077730015116231

<ipython-input-177-137356cb1069>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
```

```
(ggplot(OrigX, aes(x='Total Fat',y='Calories', color =
"factor(cluster)")) + geom_point() +
theme_minimal() + labs(title = "Cluster Membership for Gaussian
Mixture of sugars and fat with the complete dataset") +
scale_color_discrete(name ="Cluster"))
```



```
<ggplot: (8752889869565)>
```

```
Xnocarbs=Menu[['Calories', 'Calories from Fat',
       'Total Fat', 'Total Fat (% Daily Value)', 'Saturated Fat',
       'Saturated Fat (% Daily Value)', 'Trans Fat', 'Cholesterol',
       'Cholesterol (% Daily Value)', 'Sodium', 'Sodium (% Daily
Value)', 'Dietary Fiber',
       'Dietary Fiber (% Daily Value)', 'Protein',
       'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
       'Calcium (% Daily Value)', 'Iron (% Daily Value)']]
gridnocarb=GridSearchCV(GM,param_grid=kparams,cv=40,verbose=1,n_jobs=-
1)
gridnocarb.fit(Xnocarbs)
```

```
Fitting 40 folds for each of 18 candidates, totalling 720 fits
```

```
GridSearchCV(cv=40, estimator=GaussianMixture(), n_jobs=-1,
             param_grid={'n_components': range(2, 20)}, verbose=1)
```

```
gridnocarb.best_estimator_
```

```
GaussianMixture(n_components=2)
```

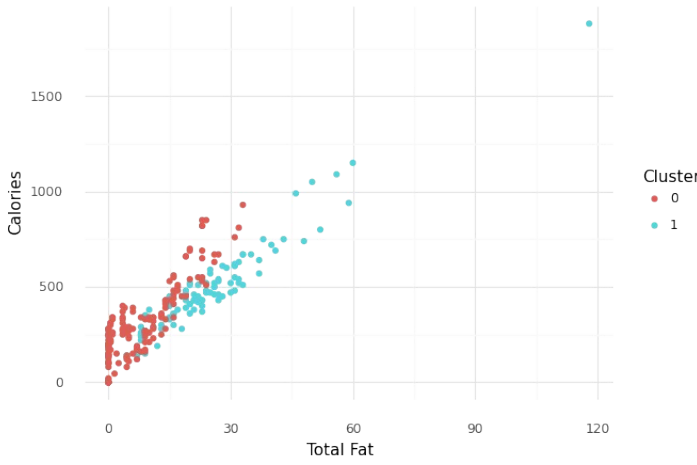
```
GMnocarb=GaussianMixture(n_components=2)
GMnocarb.fit(Xnocarbs)
nocarbpred=GMnocarb.predict(Xnocarbs)
print("Silhouette score with no features that are carbs included in
the model:",silhouette_score(Xnocarbs,nocarbpred))
Xnocarbs["cluster"] = nocarbpred
```

```
Silhouette score with no features that are carbs included in the
model: 0.601717320452297
```

```
<ipython-input-178-3cb6a202ffb8>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
```

```
(ggplot(Xnocarbs, aes(x='Total Fat',y='Calories', color =
"factor(cluster)")) + geom_point() +
theme_minimal() + labs(title = "Cluster Membership for Gaussian
Mixture of total fat and calories, with all features with the name
sugar missing") + scale_color_discrete(name ="Cluster"))
```

Cluster Membership for Gaussian Mixture of total fat and calories, with all features with the name sugar missing



```
<ggplot: (8752889955931)>
```

```
Xnofats=Menu[['Calories','Sodium', 'Sodium (% Daily Value)',
       'Carbohydrates', 'Carbohydrates (% Daily Value)', 'Dietary
Fiber',
       'Dietary Fiber (% Daily Value)', 'Sugars', 'Protein',
       'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
       'Calcium (% Daily Value)', 'Iron (% Daily Value)']]
gridnofats=GridSearchCV(GM,param_grid=kparams,cv=40,verbose=1,n_jobs=-
1)
gridnofats.fit(Xnofats)
```

```
Fitting 40 folds for each of 18 candidates, totalling 720 fits
```

```
GridSearchCV(cv=40, estimator=GaussianMixture(), n_jobs=-1,
             param_grid={'n_components': range(2, 20)}, verbose=1)
```

```
gridnofats.best_estimator_
```

```
GaussianMixture(n_components=2)
```

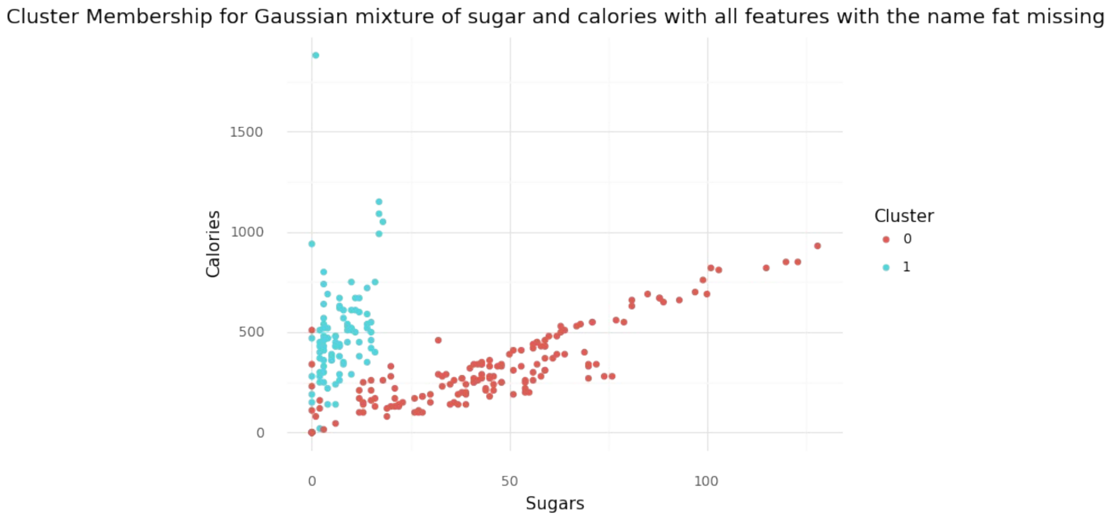
```
GMnofats=GaussianMixture(n_components=2)
GMnofats.fit(Xnofats)
nofatspred=GMnofats.predict(Xnofats)
print("Silhouette score with all features that include fat
missing:",silhouette_score(Xnofats,nofatspred))
Xnofats["cluster"] = nofatspred
```

```
Silhouette score with all features that include fat missing:
0.6021575485866818
```

```
<ipython-input-179-e613aae65a2c>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
(ggplot(Xnofats, aes(x='Sugars',y='Calories', color =
"factor(cluster)")) + geom_point() +
theme_minimal() + labs(title = "Cluster Membership for Gaussian
mixture of sugar and calories with all features with the name fat
missing") + scale_color_discrete(name ="Cluster"))
```



```
<ggplot: (8752890022440)>
```

```
Xnoprotein=Menu[['Calories', 'Calories from Fat',
       'Total Fat', 'Total Fat (% Daily Value)', 'Saturated Fat',
       'Saturated Fat (% Daily Value)', 'Trans Fat', 'Cholesterol',
       'Cholesterol (% Daily Value)', 'Sodium', 'Sodium (% Daily
Value)',
       'Carbohydrates', 'Carbohydrates (% Daily Value)', 'Dietary
Fiber',
       'Dietary Fiber (% Daily Value)', 'Sugars',
       'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
       'Calcium (% Daily Value)', 'Iron (% Daily Value)']]
gridnoprotein=GridSearchCV(GM,param_grid=kparams,cv=40,verbose=1,n_job
s=-1)
gridnoprotein.fit(Xnoprotein)
```

```
Fitting 40 folds for each of 18 candidates, totalling 720 fits
```

```
GridSearchCV(cv=40, estimator=GaussianMixture(), n_jobs=-1,
             param_grid={'n_components': range(2, 20)}, verbose=1)
```

```
gridnoprotein.best_estimator_
```

```
GaussianMixture(n_components=4)
```

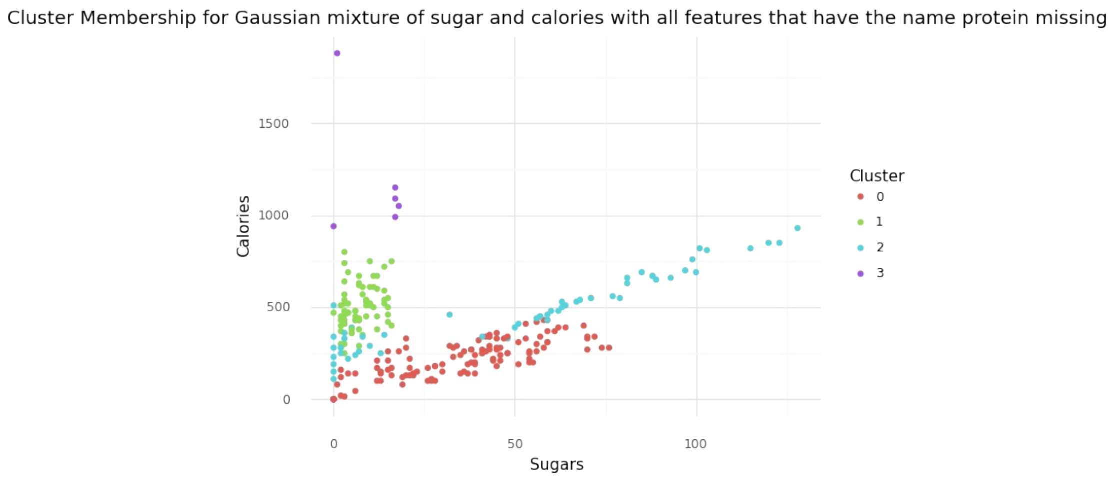
```
GMnoprotein=GaussianMixture(n_components=4)
GMnoprotein.fit(Xnoprotein)
```

```
noproteinpred=GMnoprotein.predict(Xnoprotein)
Xnoprotein["cluster"] = noproteinpred
print("Silhouette score with all features that include protein
missing:",silhouette_score(Xnoprotein,noproteinpred))
```

```
Silhouette score with all features that include protein missing:
0.4405476533016729
```

```
<ipython-input-180-9273ba9f54c7>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
```

```
(ggplot(Xnoprotein, aes(x='Sugars',y='Calories', color =
"factor(cluster)")) + geom_point() +
theme_minimal() + labs(title = "Cluster Membership for Gaussian
mixture of sugar and calories with all features that have the name
protein missing") + scale_color_discrete(name ="Cluster"))
```



```
<ggplot: (8752890100565)>
```

To answer question #1: With clustering, which group of continuous features when removed from a model would result in the highest discrepancy in silhouette score?

The metric I used to measure how well a model was performing is silhouette score. Silhouette score ranges from -1 to 1 and measures how similar a single data point (in this case a food item) to other food items in the same cluster and how similar a point is to items in other clusters as well. The closer a silhouette score is to 1, the better the clustering model performed.

The group of continuous features' whose exclusion was most detrimental on the silhouette score was protein. To answer this question, I created 4 models: One fitted on the full data and the other three fitted on 3 subsets of the original data and all were missing one major nutrient to isolate which major nutrient has the greatest impact on silhouette score. The three subsets I used are: one set with no features that included the word "fat", one set with no features that are carbohydrates and one set with no features that are protein.

The machine learning model used for all of these analyses was Gaussian Mixture. This is an unsupervised machine learning model which means there is no "correct" answer and that the model makes clusters from our data based on how similar a single data point (a McDonald's menu item) is to another menu item. Unsupervised machine learning models also have hyperparameters, which are options that we change within the model to change the silhouette score of the model. To find the best hyperparameters for the highest silhouette score, I used gridsearchCV. What gridsearchCV does it make many models based on a list of hyperparameters you give to it and based on the data you give to it as well. When it is done, you are able to access the hyperparameters that would make the model perform the best. I used this for all four models to find the best hyperparameters.

With the model on the full dataset, the silhouette score was .6. This isn't great but the point of this question isn't to test sheer model performence, but what features are most important to how the model clusters data. The model on the full dataset was created to be used as a baseline so we can compare the other models to it.

The next model built was the model missing all the features that are carbohydrates. Initially I anticipated that this would have a large effect on silhouette score, but the silhouette score stayed the same as the baseline model at .6. Sugar is a big part of the McDonald's menu as it is present in almost every drink, coffee, smoothie and milkshake. The reason removing carbohydrates from the model didn't show any change was because only two features in the full data representing carbohydrates were removed. Only carbohydrates and sugars fell into this category and as there are 23 total features, other features that are related to sugar and carbohydrates were able to make up for them being gone.

The next model was built was all features that contained the word fat missing. I for sure thought this one would be the most impactful in the silhouette score because this model removed the most features. I was incorrect in this assumption as removing all features with the name fat in them also resulted in a silhouette score of .6. What this implies is that features that include the word fat are correlated with carbohydrates and sugar because when you remove one or the other, the silhouette score stays the same. It is more than possible that when fat is missing, sugar can make it up for it and vice versa as well.

The third model was built with features that had the name protein missing. This model ended up having the highest discrepancy in silhouette score, much to my surprise. The silhouette score of this model was .44. Additionally, this model had more clusters as well than the others that were chosen by gridsearchCV. What this tells us is that of the three major nutrients I picked, protein is the most important to the silhouette score. This could potentially be because protein is not correlated with any other featurs very highly and hat when it is removed, there is nothing that can emulate its effect. Additonally, protein is very important to the classification of the data because protein is what is able to differentiate food such as burgers or fries from beverage items such as coffees or milkshakes. Without this major differentiator in food items, the model performed much worse because it was unable to cluster as effectively as the base model without any features missing and the models before it which had features missing as well.

This question is significant because the models used to answer it prove that protein is the most important of the three nutrients to predict what food category a data point belongs to. This tells us that the amount of protein is the most important data to consider for creating new food items and figuring out what category they should be put in

#2

```
LR=LogisticRegression(solver='lbfgs', max_iter=100000)
LR.fit(X_train,y_train)
trainpred=LR.predict(X_train)
testpred=LR.predict(X_test)
print("Train accuracy = ",accuracy_score(y_train, trainpred))
print("Test accuracy =" ,accuracy_score(y_test, testpred))
```

```
Train accuracy =  0.875
Test accuracy = 0.8269230769230769
```

```
pca = PCA()
pca.fit(X_train)
```

```
PCA()
```

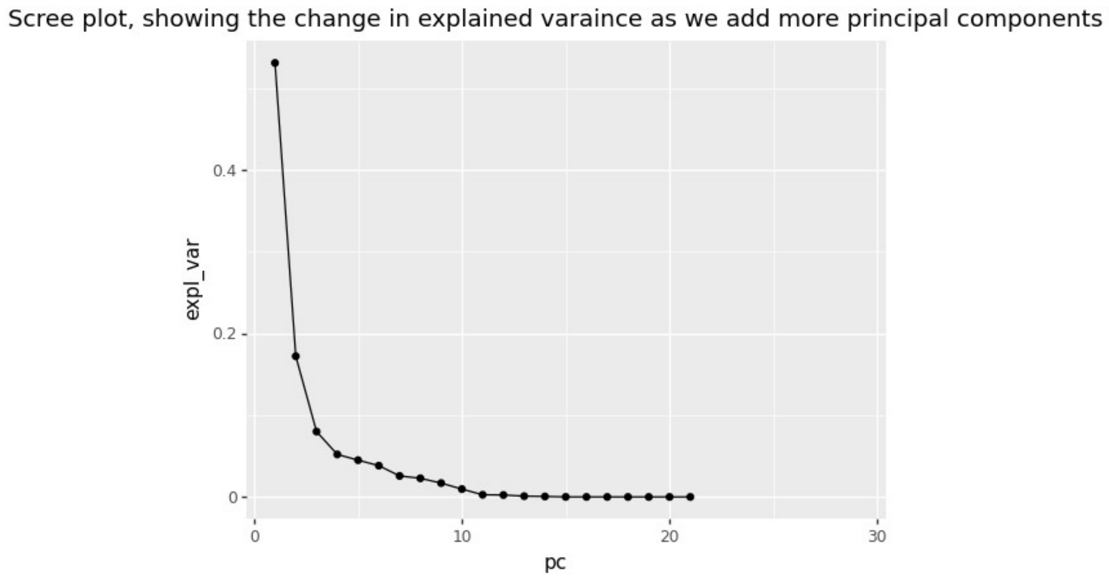
```
dictionary={"expl_var" : pca.explained_variance_ratio_, "pc":
range(1,30), "cum_var":pca.explained_variance_ratio_.cumsum()}
pcaDF = pd.DataFrame.from_dict(dictionary, orient='index')
pcaDF = pcaDF.transpose()
pcaDF
```

```
    expl_var    pc   cum_var
0   0.530889   1.0  0.530889
1   0.172149   2.0  0.703038
2   0.079959   3.0  0.782996
3   0.052008   4.0  0.835004
4   0.045005   5.0  0.880009
5   0.038264   6.0  0.918273
6   0.025706   7.0  0.943979
7   0.022812   8.0  0.966791
8   0.016954   9.0  0.983745
9   0.009590  10.0  0.993335
10  0.002670  11.0  0.996004
11  0.002355  12.0  0.998359
12  0.000970  13.0  0.999329
13  0.000576  14.0  0.999905
14  0.000032  15.0  0.999937
15  0.000022  16.0  0.999959
16  0.000013  17.0  0.999972
17  0.000010  18.0  0.999983
```

```
18  0.000008  19.0  0.999991
19  0.000006  20.0  0.999997
20  0.000003  21.0  1.000000
21       NaN  22.0       NaN
22       NaN  23.0       NaN
23       NaN  24.0       NaN
24       NaN  25.0       NaN
25       NaN  26.0       NaN
26       NaN  27.0       NaN
27       NaN  28.0       NaN
28       NaN  29.0       NaN
```

```
ggplot(pcaDF, aes(x = "pc", y = "expl_var")) + geom_line() +
geom_point()+ggtitle("Scree plot, showing the change in explained
varaince as we add more principal components")
```

```
/usr/local/lib/python3.8/dist-packages/plotnine/geoms/geom_path.py:75:
PlotnineWarning: geom_path: Removed 8 rows containing missing values.
/usr/local/lib/python3.8/dist-packages/plotnine/layer.py:401:
PlotnineWarning: geom_point : Removed 8 rows containing missing
values.
```



```
<ggplot: (8752889432546)>
```

```
pcomps7train = pca.transform(X_train)
pcomps7train = pd.DataFrame(pcomps7train[:,0:7])

pcomps7test=pca.transform(X_test)
pcomps7test=pd.DataFrame(pcomps7test[:,0:7])

LR2 = LogisticRegression(solver='lbfgs', max_iter=100000)
LR2.fit(pcomps7train, y_train)
pcatrainpred=LR2.predict(pcomps7train)
pcatestpred=LR2.predict(pcomps7test)
```

```
print("PCA Train accuracy = ",accuracy_score(y_train, pcatrainpred))
print("PCA Test accuracy =" ,accuracy_score(y_test, pcatestpred))
```

```
PCA Train accuracy =  0.7836538461538461
PCA Test accuracy = 0.7115384615384616
```

Question 2: In using PCA and scree plot and the elbow of the scree plot, how many components should be picked, what is their accuracy compared to the full data and what are the implications of that?

For this question, I am building a logistic regression and then getting accuracy metrics for that as a baseline. Then I used PCA to reduce the dimensions of my data and chose the number of components at the elbow so I do not get diminishing returns, then I make a new logistic regression that is fit upon my principal components and accuracy metrics are computed for that as well.

Logsitic Regression is a model type that is supervised and what that means is that we have access to the "correct" answers, unlike the Gaussian mixture in the last question. For this question we use a Train test split which splits up all the data into two sets: the seen data and the unseen data. The reason for this is that we show the "seen" data to the model so it can learn the patterns of the data and be able to predict what type of food a certain piece of data is. Then to see how well our model was able to learn the prediction food item, we show it pieces of data from the "unseen" data to figure out if it can generalize what it learned from the seen data to the unseen data. In machine learning, the seen data is commonly referred to as train because the model is trained with it and the unseen data is referred to as test data because we use it to test how well the model performs. With this model we used the training set to train the model and test set to test out how well the model works. To do this we used the accuracy score. What the accuracy tells us is what percent of the total predictions of the model were correct.

With the model before PCA, the train accuracy was 87.5% and the test accuracy was 82%. Since the test accuracy is a lower than the train accuracy, the model is slightly overfit. What this means is that the model can predict on the train data really well, but that it has trouble generalizing what it learned from the train data to the test data. As it is only overfit by 5%, this isn't a huge worry.

To explain PCA simply, the model first computes the covariance matrix which explains how the features are related, then it computes eigenvectors and eigen values from the covariance matrix. TThe associated math but the eigenvectors with the largest eigenvalues become the principal components of the data.

Essentially what this does is find new dimensions of the data that are combinations of the original data, while keeping the most variance. PCA aims to reduce dimensionality with the combining of dimensions so the data is less complex and so it is easier to analyze.

While using PCA, I made a scree plot which explains how much of the data's total variance is maintained through a specific number of components. In this case, we want higher variance as variance is essentially how well the patterns and trends of the original are maintained, but we do not want to choose a number of components so high that we get diminishing returns. With the scree plot, we examine the "elbow" of the graph, or where the graph begins to flatten out. Once the graph begins to flatten out is the point at which we will get diminishing returns. In this case, I selected 7 components, which holds 94% of the original data's variance. After choosing these 7 components, I made a new logistic regression model using these seven components and computed accuracy scores for the new model.

With the model using PCA's components, the train accuracy was 78% and the test accuracy was 71%. Since the test accuracy is a lower than the train accuracy, the model is slightly overfit. This is a pretty stark drop from the model before PCA. We lose about 10% accuracy on the train and test after using PCA. What this means is that we had 10% more incorrect predictions when we use the components derived from PCA compared the original data.

This most likely occurs because of the 6% variance that we are missing out on by selecting 7 components. This dataset isn't especially large and removing 6% of the total variance has a major effect because for this dataset, 6% is a lot of the total data compared to a larger dataset. Additionally, by reducing the dimensions we are most likely missing out on important data for predicting and our accuracy is lower as a result.

The answer to this question is significant because now know that we should not be using PCA with this dataset because it is on the smaller side and the reduction of dimensions that PCA does is very harmful to the predicting power of this dataset. Additionally, if in the future, McDonald's proposes changing the way they display their nutrition data by combining certain features, I can strongly reccomend that they decide not to do that based on my findings.

#3

```
params={"criterion":['gini','entropy'],
"max_depth":range(1,30),"min_samples_split":range(2,10),'min_samples_l
eaf':range(2,10)}
decision_tree=DecisionTreeClassifier()
grid=GridSearchCV(decision_tree,param_grid=params,cv=4,verbose=1,n_job
s=-1)
grid.fit(X_train,y_train)
```

```
Fitting 4 folds for each of 3712 candidates, totalling 14848 fits
```

```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 30),
                         'min_samples_leaf': range(2, 10),
                         'min_samples_split': range(2, 10)},
             verbose=1)
```

```
grid.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=6,
min_samples_leaf=2,
                       min_samples_split=7)
```

```
mainTree=DecisionTreeClassifier(criterion='entropy', max_depth=6,
min_samples_leaf=2,
                       min_samples_split=7)
mainTree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=6,
min_samples_leaf=2,
                       min_samples_split=7)
```

```
treetrainpred=mainTree.predict(X_train)
treetestpred=mainTree.predict(X_test)
```

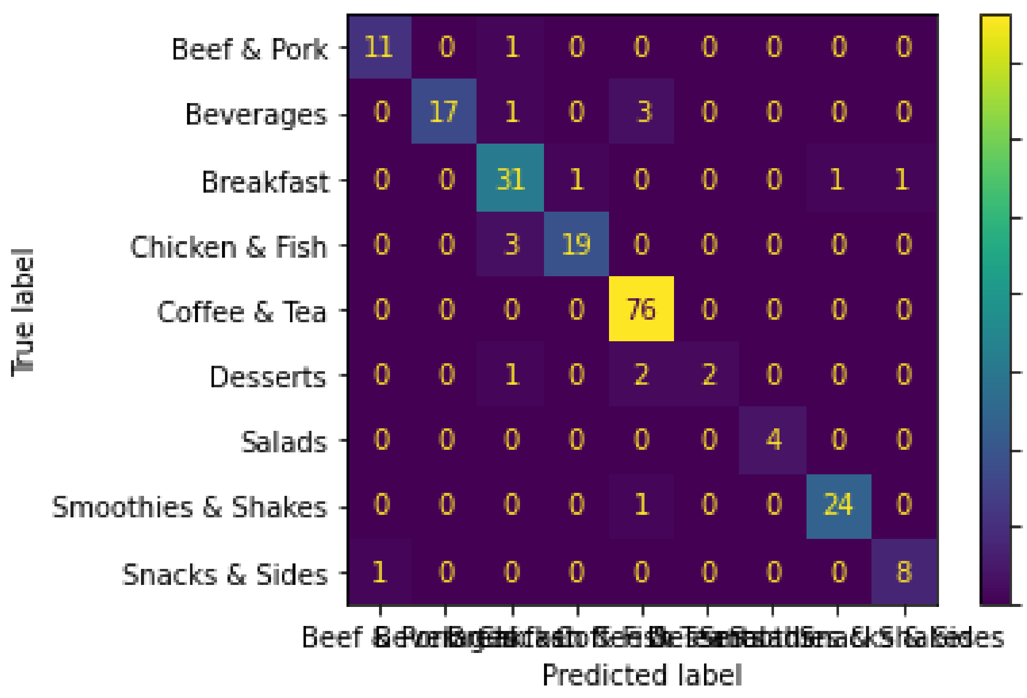
```
print("Train accuracy for the full
dataset:",accuracy_score(treetrainpred,y_train))
print("Test accuracy for the full
dataset:",accuracy_score(treetestpred,y_test))
```

```
Train accuracy for the full dataset: 0.8942307692307693
Test accuracy for the full dataset: 0.75
```

```
from sklearn.metrics._plot.confusion_matrix import
plot_confusion_matrix
plot_confusion_matrix(mainTree,X_train,y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and
will be removed in 1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5f0f435c40>
```



```
X_noDV=Menu[['Calories', 'Calories from Fat','Total Fat', 'Saturated
Fat','Trans Fat', 'Cholesterol','Sodium', 'Carbohydrates',  'Dietary
Fiber','Sugars', 'Protein']]
X_noTotal=Menu[['Total Fat (% Daily Value)', 'Saturated Fat (% Daily
Value)', 'Cholesterol (% Daily Value)',  'Sodium (% Daily
Value)','Carbohydrates (% Daily Value)', 'Dietary Fiber (% Daily
Value)','Vitamin A (% Daily Value)', 'Vitamin C (% Daily
Value)','Calcium (% Daily Value)', 'Iron (% Daily Value)']]

X_trainnoDV, X_testnoDV, y_trainnoDV, y_testnoDV =
train_test_split(X_noDV,y, test_size=0.2)
X_trainnoTotal, X_testnoTotal, y_trainnoTotal, y_testnoTotal =
train_test_split(X_noTotal,y, test_size=0.2)
decision_treenoDV=DecisionTreeClassifier()
gridnoDV=GridSearchCV(decision_treenoDV,param_grid=params,cv=5,verbose
=1,n_jobs=-1)
gridnoDV.fit(X_trainnoDV,y_trainnoDV)

Fitting 5 folds for each of 3712 candidates, totalling 18560 fits

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 30),
                         'min_samples_leaf': range(2, 10),
                         'min_samples_split': range(2, 10)},
             verbose=1)

gridnoDV.best_estimator_
treenoDV=DecisionTreeClassifier(max_depth=19, min_samples_leaf=2,
min_samples_split=3)
treenoDV.fit(X_trainnoDV,y_trainnoDV)
```

```
DecisionTreeClassifier(max_depth=19, min_samples_leaf=2,
min_samples_split=3)
```

```
noDVtrainpred=treenoDV.predict(X_trainnoDV)
noDVtestpred=treenoDV.predict(X_testnoDV)
```

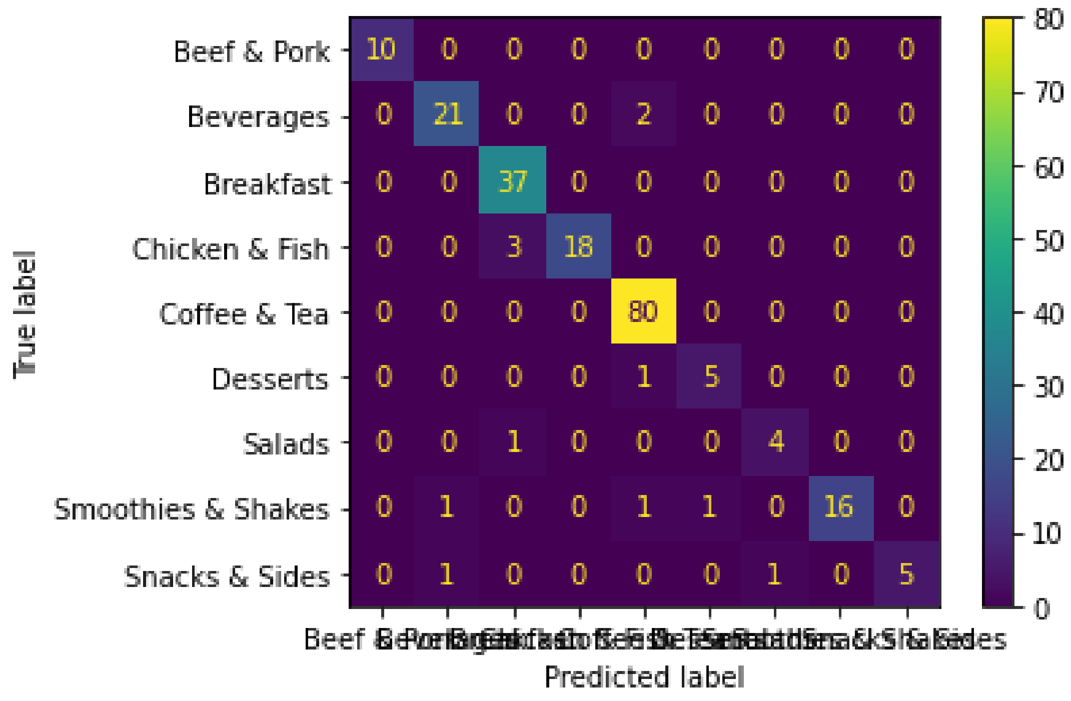
```
print("Tree train accuracy with daily value %s
excluded:",accuracy_score(noDVtrainpred,y_trainnoDV))
print("Tree test accuracy with daily value %s
excluded:",accuracy_score(noDVtestpred,y_testnoDV))
```

```
Tree train accuracy with daily value %s excluded: 0.9423076923076923
Tree test accuracy with daily value %s excluded: 0.6346153846153846
```

```
plot_confusion_matrix(treenoDV,X_trainnoDV,y_trainnoDV)
```

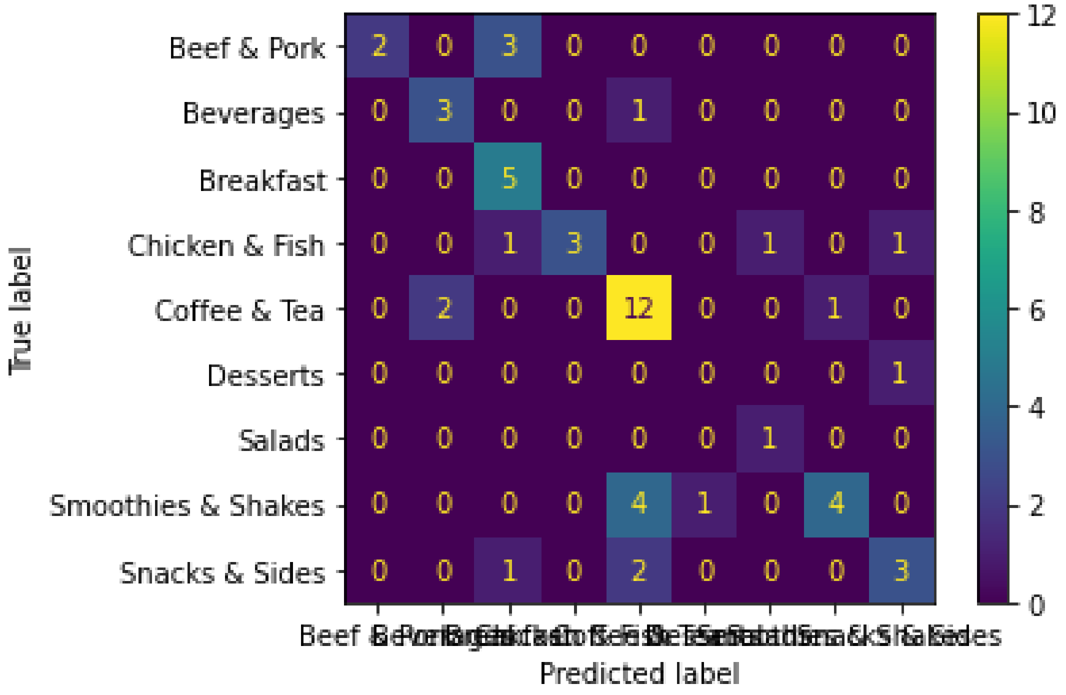
```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and
will be removed in 1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5f14e88550>
```



```
plot_confusion_matrix(treenoDV,X_testnoDV,y_testnoDV)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and
will be removed in 1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5f0f011940>
```



```
decision_treenoTotal=DecisionTreeClassifier()
gridnoTotal=GridSearchCV(decision_treenoTotal,param_grid=params,cv=5,v
erbose=1,n_jobs=-1)
gridnoTotal.fit(X_trainnoTotal,y_trainnoTotal)

Fitting 5 folds for each of 3712 candidates, totalling 18560 fits

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 30),
                         'min_samples_leaf': range(2, 10),
                         'min_samples_split': range(2, 10)},
             verbose=1)

decision_treenoTotal=DecisionTreeClassifier(criterion='entropy',
max_depth=19, min_samples_leaf=2,
                       min_samples_split=5)
decision_treenoTotal.fit(X_trainnoTotal,y_trainnoTotal)

DecisionTreeClassifier(max_depth=19, min_samples_leaf=2)
DecisionTreeClassifier(criterion='entropy', max_depth=19,
min_samples_leaf=2,
                       min_samples_split=5)
```

```
noTotaltrainpred=decision_treenoTotal.predict(X_trainnoTotal)
noTotaltestpred=decision_treenoTotal.predict(X_testnoTotal)
```

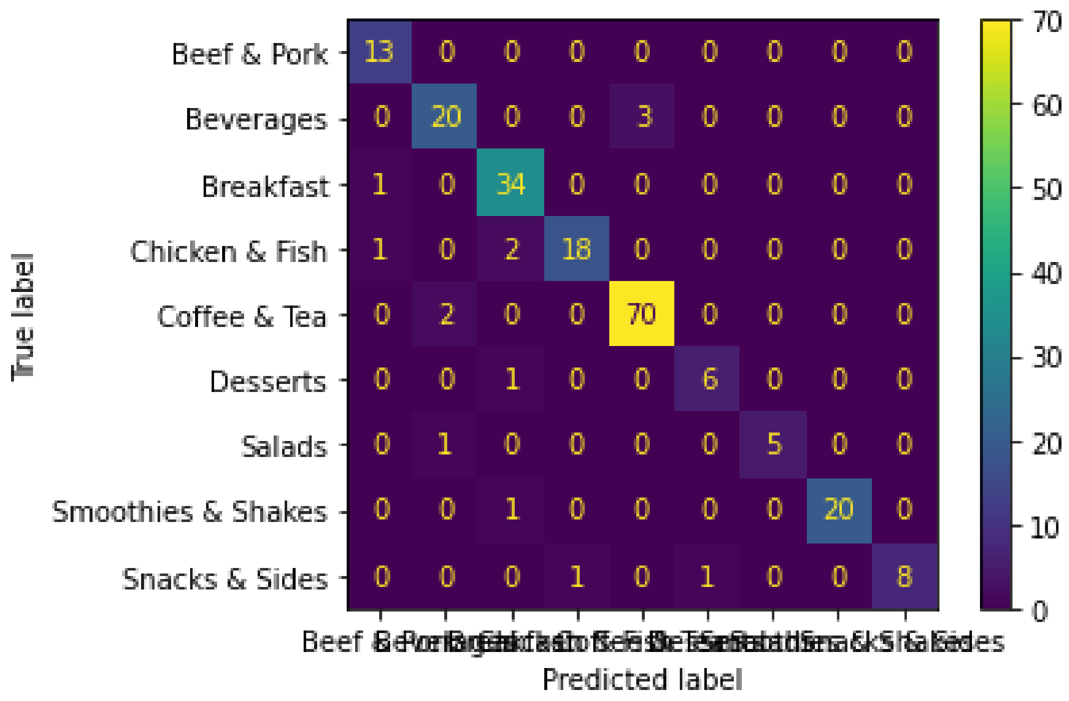
```
print("Tree train accuracy with total amounts
excluded:",accuracy_score(noTotaltrainpred,y_trainnoTotal))
print("Tree test accuracy with total amounts
excluded:",accuracy_score(noTotaltestpred,y_testnoTotal))
```

```
Tree train accuracy with total amounts excluded: 0.9326923076923077
Tree test accuracy with total amounts excluded: 0.8269230769230769
```

```
plot_confusion_matrix(decision_treenoTotal,X_trainnoTotal,y_trainnoTot
al)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and
will be removed in 1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5f0eadc1c0>
```

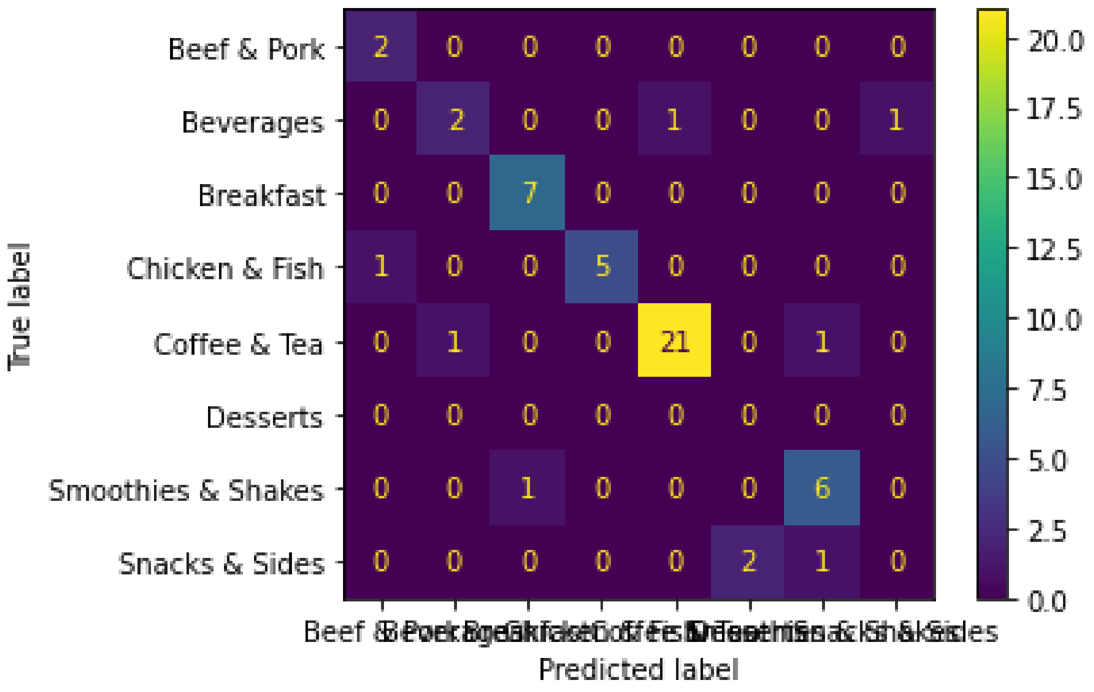


13	0	0	0	0	0	0	0	0
0	20	0	0	3	0	0	0	0
1	0	34	0	0	0	0	0	0
1	0	2	18	0	0	0	0	0
0	2	0	0	70	0	0	0	0
0	0	1	0	0	6	0	0	0
0	1	0	0	0	0	5	0	0
0	0	1	0	0	0	0	20	0
0	0	0	1	0	1	0	0	8

```
plot_confusion_matrix(decision_treenoTotal,X_testnoTotal,y_testnoTotal
)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and
will be removed in 1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5f0dafaee0>
```



Question 3: Are the actual nutrients themselves (their content in grams) or their daily values as a percent more important in predicting the category of food?

This question may seem redundant at first, but my line of thinking while creating this question and the results prove it is not. While to a human the daily values of a food item in percentages and those same nutrients as total values in grams or milligrams mean the same thing and can be converted back and forth, the machine does not know that. All the measure sees input data as is something to learn from and cannot discern that nutrients as daily value percents and the same nutrients as the total values.

To answer this question I used gridsearchCV and a decision tree. GridsearchCV works the same as mentioned above, but decision trees are different from the models explained previously

A decision tree uses train and test data just like the logistic regression model but a decision tree splits the data into smaller and smaller groups such that what falls into an individual group is all very similar to the other members of the group. With the train data, the decision tree makes these groups in order to be able to see a data point and in this case, be able to know what food category it falls into. To predict, the tree uses what it has learned from the training data to make new predictions.

To answer this question, I used three models: one tree fit on the whole data, one tree fit on the data with none of the daily value and one fit on the data with no of the total value nutrients.

The baseline model which is the tree fit on the full dataset had a training set accuracy of 89% and testing set accuracy of 75%. The tree is overfit, but as gridsearch was used to find the best parameters, this is likely the best it will get. As decision trees have more depth, they are more prone to overfitting. While this model is overfit, it is just being used a baseline to compare the other two models to.

The next model used was the model that was missing all of the features that were a daily value. Gridsearch was used for this model as well to find the best hyperparameters. This model had 94% train accuracy and 63% test accuracy. This model is severly overfit and cannot be trusted to predict on unseen data. What this implies is that the nutrients as daily value percentages is very important to prediction on unseen data. Even though the training set prediction went really well, it is insignificant as this model was built to predict on unseen data.

The third model used was the model with no total values of any nutrients. This model performed the best and had 93% training accuracy and 82% testing accuracy. This model is still overfit but it is the least overfit of all three and has the highest testing accuracy and the second highest training accuracy. What this tells us is that the total nutrient values in grams or milligrams significantly interferes with the model for two reasons: the first being that when we don't use them at all, we have significantly better model performence and when we only use the total values to predict, the model performence signficantly degrades. The third model which predicted over with no total nutrient values is the best model.

This question is significiant because it shows that the nutrition information in the total value format causes more harm to the predicting power of a model on what category a food item is and if the only concern with the data was model accuracy, then total values should be removed. Unforuntately, this is most likely illegal as the FDA or whoever the governing body over McDonald's nutrient reporting most likely mandates that it is reported in this format.