DATA 221 Homework 5

Chris Low

Problem 1

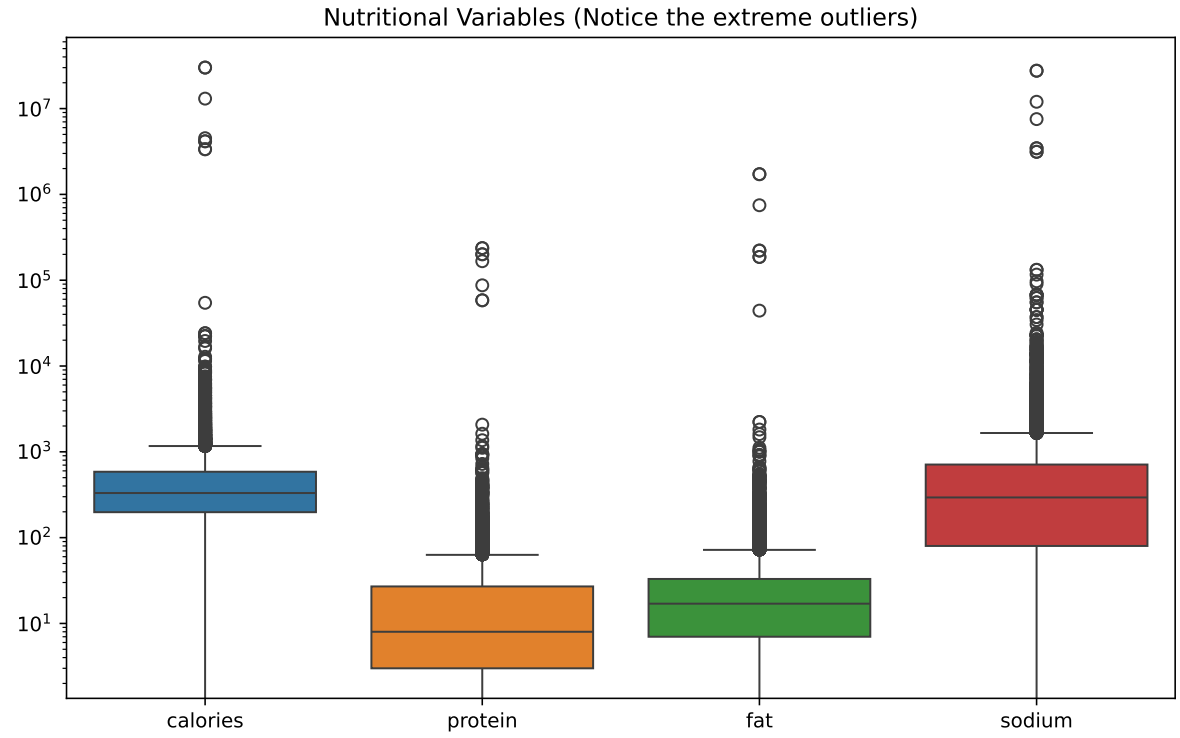
Exploratory Data Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

epi = pd.read_csv("epi_r.csv", index_col='title')

# EDA
print(f"Dataset shape: {epi.shape}")
print(epi.info())

nutritional_vars = ['calories', 'protein', 'fat', 'sodium']
print(epi[nutritional_vars].describe())

# Plotting to see the extreme outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=epi[nutritional_vars])
plt.title("Nutritional Variables (Notice the extreme outliers)")
plt.yscale('log') # Log scale: outliers are massive
plt.show()

# Check for missing values
print("Missing values before imputation:\n", epi[nutritional_vars].isna().sum())
```

```
Dataset shape: (20052, 679)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20052 entries, Lentil, Apple, and Turkey Wrap  to Baked Ham with Marmalade-
Horseradish Glaze
Columns: 679 entries, rating to turkey
dtypes: float64(679)
memory usage: 104.0+ MB
None
```

	calories	protein	fat	sodium
count	1.593500e+04	15890.000000	1.586900e+04	1.593300e+04
mean	6.322958e+03	100.160793	3.468775e+02	6.225975e+03
std	3.590460e+05	3840.318527	2.045611e+04	3.333182e+05
min	0.000000e+00	0.000000	0.000000e+00	0.000000e+00
25%	1.980000e+02	3.000000	7.000000e+00	8.000000e+01
50%	3.310000e+02	8.000000	1.700000e+01	2.940000e+02
75%	5.860000e+02	27.000000	3.300000e+01	7.110000e+02
max	3.011122e+07	236489.000000	1.722763e+06	2.767511e+07




```
Missing values before imputation:
 calories    4117
protein     4162
```

```
fat         4183
sodium      4119
dtype: int64
```

I decided to pre-process mainly nutritional variables because they are numerical and directly relevant to the analysis we will be doing (PCA and regularized logistic regression). These variables are likely to have a significant impact on the outcome variable 'cake', which is what we will be trying to predict.

The other features, such as Inredients, are not numerical and would require different pre-processing steps (like encoding) if we were to include them in the analysis. For the scope of this homework, we will focus on the nutritional variables as features for PCA and regularized logistic regression, and 'cake' as the outcome variable.

Some pre-processing steps I will take include:

1. Handling Missing Values: I will impute missing values using the median, as it is more robust to outliers than the mean.
2. Handling Outliers: I will cap the values at the 99th percentile to mitigate the influence of extreme outliers without dropping too much data.
3. Scaling the Features: I will standardize the features to have a mean of 0 and a standard deviation of 1, which is crucial for PCA and regularized logistic regression to perform well.

```
# PRE-PROCESSING

# Handle Missing Values (Median: as it is more robust to outliers than mean)
for col in nutritional_vars:
    epi[col] = epi[col].fillna(epi[col].median())

# Handle Outliers (Capping at the 99th percentile). We do this so we don't
# drop too much data, but we eliminate non-sensical values that are likely
# data entry errors or extreme outliers that could skew our analysis.
for col in nutritional_vars:
    cap_value = epi[col].quantile(0.99)
    epi[col] = np.where(epi[col] > cap_value, cap_value, epi[col])

epi[nutritional_vars].skew()

for col in nutritional_vars:
    print(f"{col} skew before log:", epi[col].skew())
```

```
    log_col = np.log1p(epi[col])
    print(f"{col} skew after log:", log_col.skew())
    print()
```

```
calories skew before log: 3.0581732543069506
calories skew after log: -0.9529220795049637

protein skew before log: 3.0273912717515508
protein skew after log: 0.18141172715576814

fat skew before log: 3.1216217591697024
fat skew after log: -0.8013486243105284

sodium skew before log: 3.4444276767393576
sodium skew after log: -0.983097857512756
```

From the above code, the nutritional variables still show strong right skew (skewness ≈ 3) after capping the 99 percentile. After applying a $log(1 + x)$ transformation, skewness decreased a lot toward zero, so this shows more symmetric distribution. This log transformation prior to scaling improves the stability of PCA.

Moving onto the next steps, where are remove binary tags that are too rare (less than 20 occurrences) (because they are not informative for modeling) and then separate the outcome variable 'cake' from the features, and finally scale the features for PCA and regularized logistic regression.

```
# Remove rare binary tags
binary_cols = [col for col in epi.columns
               if set(epi[col].unique()).issubset({0,1})]

min_count = 20
rare_cols = [col for col in binary_cols
             if epi[col].sum() < min_count]

epi = epi.drop(columns=rare_cols)

# Separate outcome (i.e. cake) from features
y = epi['cake']
X = epi.drop(columns=['cake'])

# Scale
```

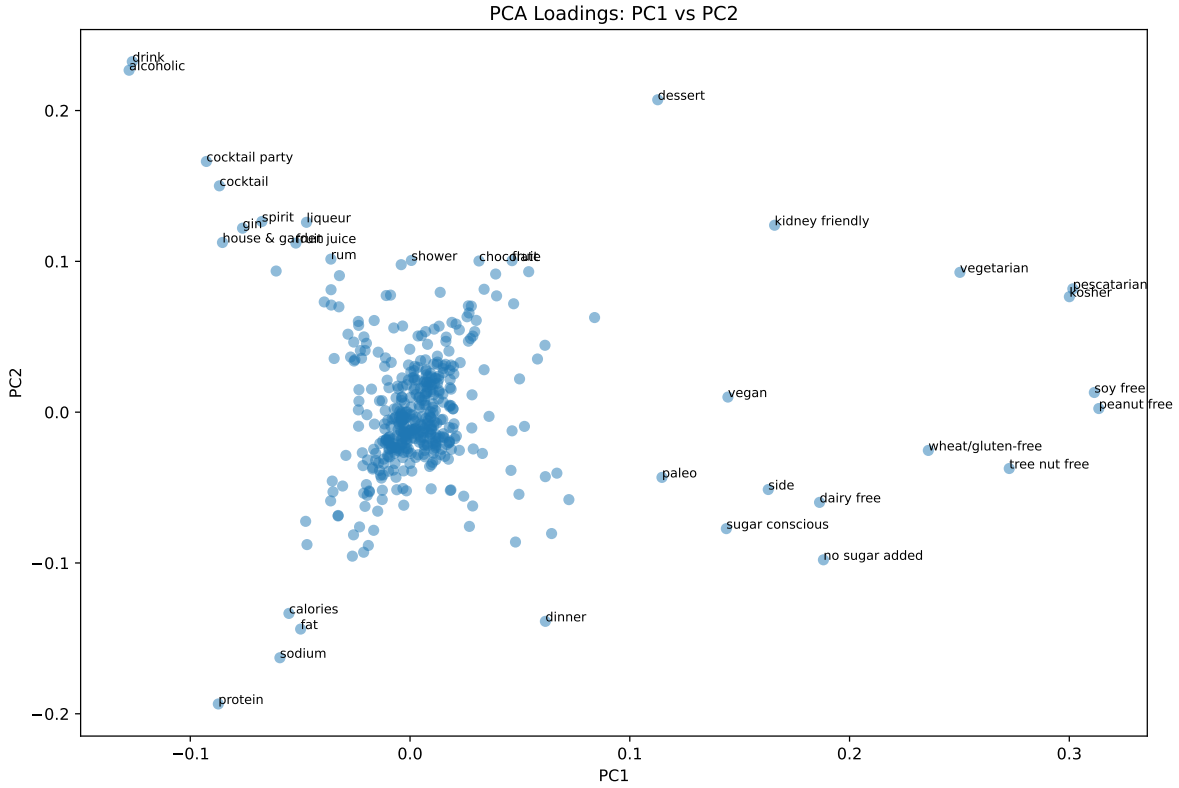
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

1(a)

```
from sklearn.decomposition import PCA

pca = PCA()
X_pca = pca.fit_transform(X_scaled)
```

1(b)

```
# Transpose because pca.components_ has shape (n_components, n_features),
# and we want (n_features, n_components) to align with our original features
loadings = pca.components_.T

pc1 = loadings[:, 0]
pc2 = loadings[:, 1]

plt.figure(figsize=(12,8))
plt.scatter(pc1, pc2, alpha=0.5)

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA Loadings: PC1 vs PC2')

# Label strongest contributors to PC1 and PC2 (absolute loading > 0.1)
for i, feature in enumerate(X.columns):
    if abs(pc1[i]) > 0.1 or abs(pc2[i]) > 0.1:
        plt.text(pc1[i], pc2[i], feature, fontsize=8)

plt.show()
```



Patterns:

In the scatter plot of PC1 vs PC2, **most ingredient tags are clustered near the origin**, which suggests they do not strongly influence the first two principal components. This makes sense because many tags are sparse and do not vary consistently across recipes, so they do not drive the main directions of variation.

The nutritional variables (calories, fat, sodium, and protein) appear grouped together in the lower-left region. This suggests that one of the components is capturing something like overall richness or heaviness, since these variables tend to increase together. We also see that **alcohol (wine, beer, liquor) tags cluster together in the upper-left area**, which may indicate that they contribute to a different dimension of variation related to alcoholic content or recipe type.

On the positive side of PC1, dietary restriction tags such as vegetarian, vegan, gluten-free, and dairy-free cluster together, which suggests that PC1 may represent a dietary or health-conscious dimension.

Dessert stands out in the upper-right area of the plot, indicating that it contributes differently from both the nutritional variables and alcohol-related tags.

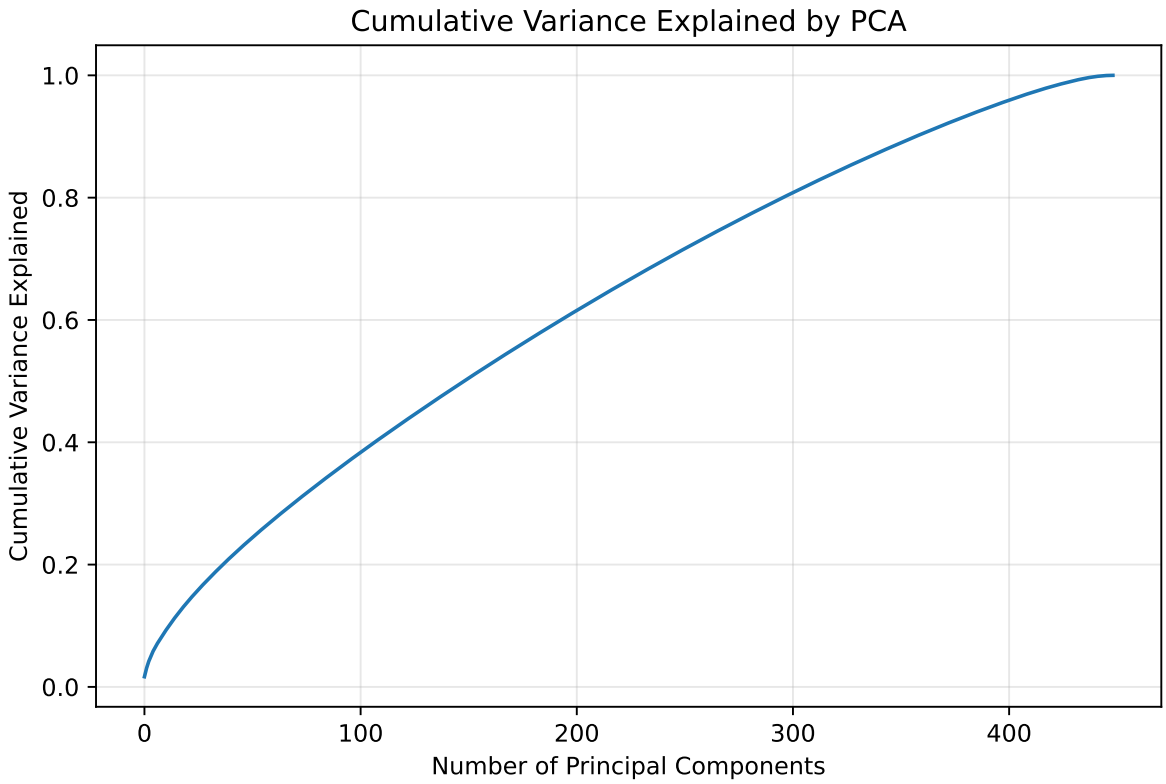
Overall, the plot shows clear groupings of related features, showing that the first two principal components capture meaningful structure in the dataset rather than random noise.

1(c)

```
explained_variance = pca.explained_variance_ratio_
cumulative_variance = explained_variance.cumsum()

print("Number of components:", len(pca.explained_variance_ratio_))
print("First 10 explained variances:")
print(explained_variance[:10])

print("Cumulative variance (first 10):")
print(cumulative_variance[:10])

print("Explained by PC1:", explained_variance[0])
print("Explained by PC2:", explained_variance[1])
print("Total explained by PC1 + PC2:", explained_variance[0] + explained_variance[1])
```

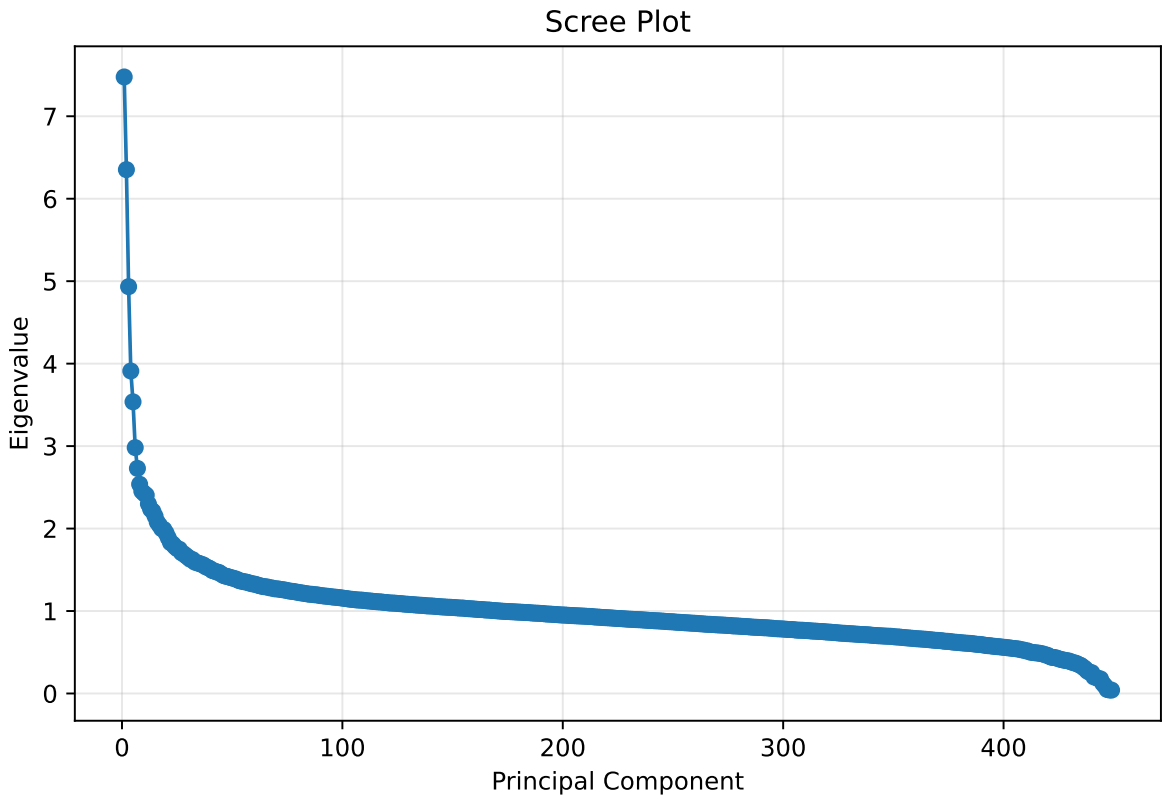
```
Number of components: 449
First 10 explained variances:
[0.01665035 0.0141477  0.01098697 0.00870957 0.0078767  0.00664027
 0.00607906 0.00565162 0.0054638  0.00540505]
Cumulative variance (first 10):
[0.01665035 0.03079805 0.04178502 0.05049459 0.05837128 0.06501155
 0.07109061 0.07674223 0.08220603 0.08761108]
Explained by PC1: 0.016650345349986145
Explained by PC2: 0.014147699940516576
Total explained by PC1 + PC2: 0.030798045290502722
```

```
plt.figure(figsize=(8,5))
plt.plot(cumulative_variance)
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Variance Explained")
plt.title("Cumulative Variance Explained by PCA")
plt.grid(alpha=0.3)
plt.show()
```



Scree plot:

```
plt.figure(figsize=(8,5))

plt.plot(range(1, len(pca.explained_variance_) + 1),
         pca.explained_variance_,
         marker='o')

plt.xlabel("Principal Component")
plt.ylabel("Eigenvalue")
plt.title("Scree Plot")
plt.grid(alpha=0.3)

plt.show()
```



Based on the scree plot, I cannot see a clear "elbow". However, there is a noticeable drop in eigenvalues after the first few components. I decided to use the KneeLocator method to identify the optimal number of components.

```
from kneed import KneeLocator

x = range(1, len(pca.explained_variance_)+1)
knee = KneeLocator(x, pca.explained_variance_, curve='convex', direction='decreasing')

print("Knee at component:", knee.knee)
```

```
Knee at component: 33
```

PC1 explains approximately 1.67% of the variance and PC2 explains approximately 1.41%, for a total of about 3.08%.

From the scree plot and the KneeLocator result, we see that the optimal number of components is around 30–40 (33 specifically). After this point, each additional component contributes only a small incremental amount of variance.

Therefore, if the goal is dimensionality reduction while retaining the main structures in the data, I would select roughly 30–40 principal components. However, to capture a large fraction of total variance (such as 80%), many more components would be required (around 200+), which may not be practical for modeling.

Problem 2

2(a)

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

from sklearn.model_selection import train_test_split

X_model = X_pca   # from Problem 1
y_model = y

X_train, X_test, y_train, y_test = train_test_split(
    X_model,
    y_model,
    test_size=0.3,
    random_state=42,
    stratify=y_model
)

C_grid = np.logspace(-3, 0, 6)   # Strong to weak regularization

auc_scores = []
coef_paths = []

for i, C in enumerate(C_grid):
    print(f"Fitting model {i+1}/{len(C_grid)} with C={C}")

    model = LogisticRegression(
        penalty="l1",
        C=C,
        solver="saga",
        max_iter=5000,
        random_state=42,
        n_jobs=-1
    )

    model.fit(X_train, y_train)
```

```
    # Compute test AUC
    y_prob = model.predict_proba(X_test)[:, 1]
    auc = roc_auc_score(y_test, y_prob)

    auc_scores.append(auc)
    coef_paths.append(model.coef_[0])
```

```
Fitting model 1/6 with C=0.001
Fitting model 2/6 with C=0.003981071705534973
Fitting model 3/6 with C=0.015848931924611134
Fitting model 4/6 with C=0.0630957344480193
Fitting model 5/6 with C=0.25118864315095796
Fitting model 6/6 with C=1.0
```

```
auc_scores = np.array(auc_scores)

best_index = np.argmax(auc_scores)
best_C = C_grid[best_index]

print(f"Optimal regularization parameter (C): {best_C:.4f}")
```

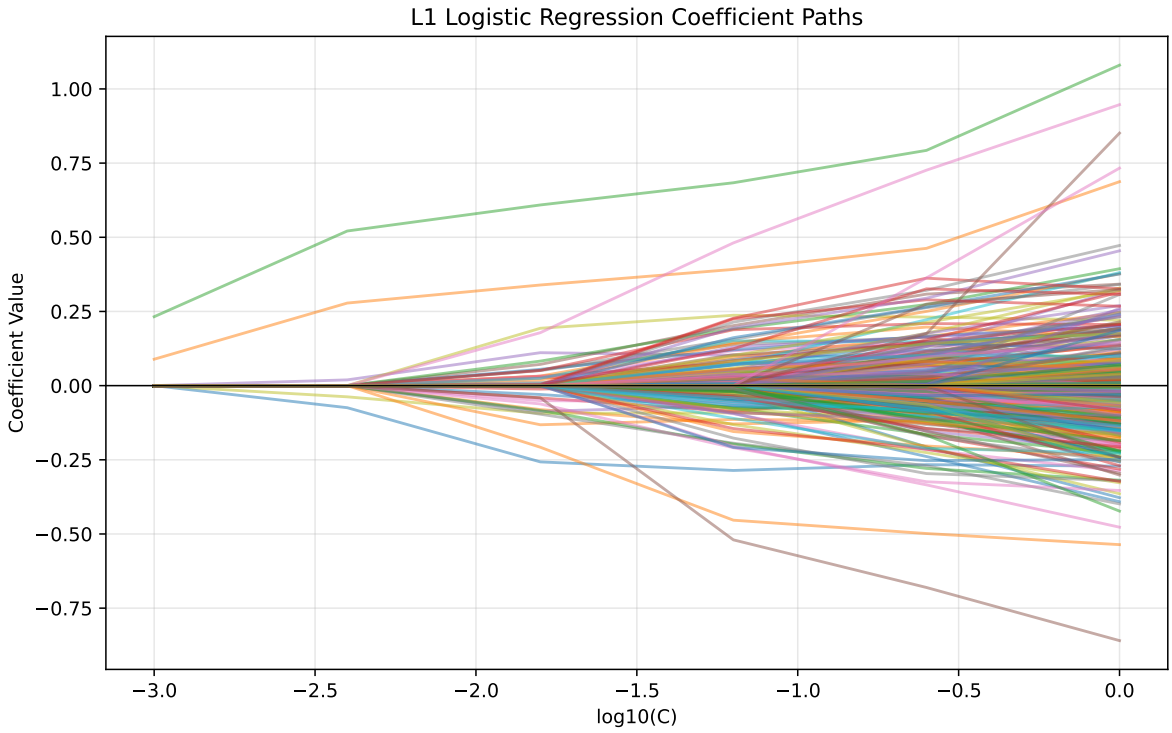
```
Optimal regularization parameter (C): 0.0631
```

```
best_AUC = auc_scores[best_index]

print(f"Test set AUC of selected classifier: {best_AUC:.4f}")
```

```
Test set AUC of selected classifier: 0.9734
```

```
coef_paths = np.array(coef_paths)

plt.figure(figsize=(10,6))

for j in range(coef_paths.shape[1]):
    plt.plot(np.log10(C_grid), coef_paths[:, j], alpha=0.5)

plt.xlabel("log10(C)")
plt.ylabel("Coefficient Value")
plt.title("L1 Logistic Regression Coefficient Paths")
```

```
plt.axhline(0, color="black", linewidth=0.8)
plt.grid(alpha=0.3)

plt.show()
```



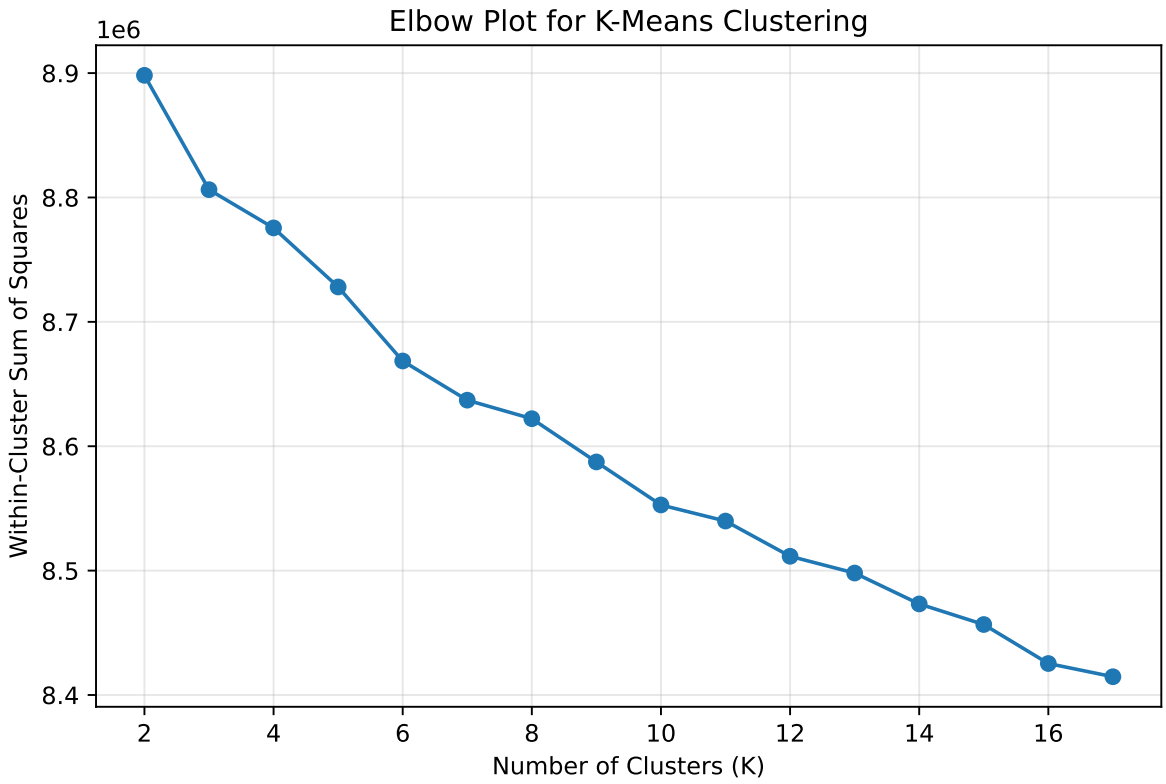
Problem 3

3(a)

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

X_full = epi.copy()

# Scale all features
scaler_full = StandardScaler()
X_full_scaled = scaler_full.fit_transform(X_full)

K_values = range(2, 18)
within_cluster_var = []

for K in K_values:
    kmeans = KMeans(
        n_clusters=K,
        random_state=42,
        n_init=20
    )

    kmeans.fit(X_full_scaled)
    within_cluster_var.append(kmeans.inertia_)  # inertia = WCSS
```

3(b)

```
# Elbow plot
plt.figure(figsize=(8,5))
plt.plot(K_values, within_cluster_var, marker='o')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Within-Cluster Sum of Squares")
plt.title("Elbow Plot for K-Means Clustering")
plt.grid(alpha=0.3)
plt.show()
```

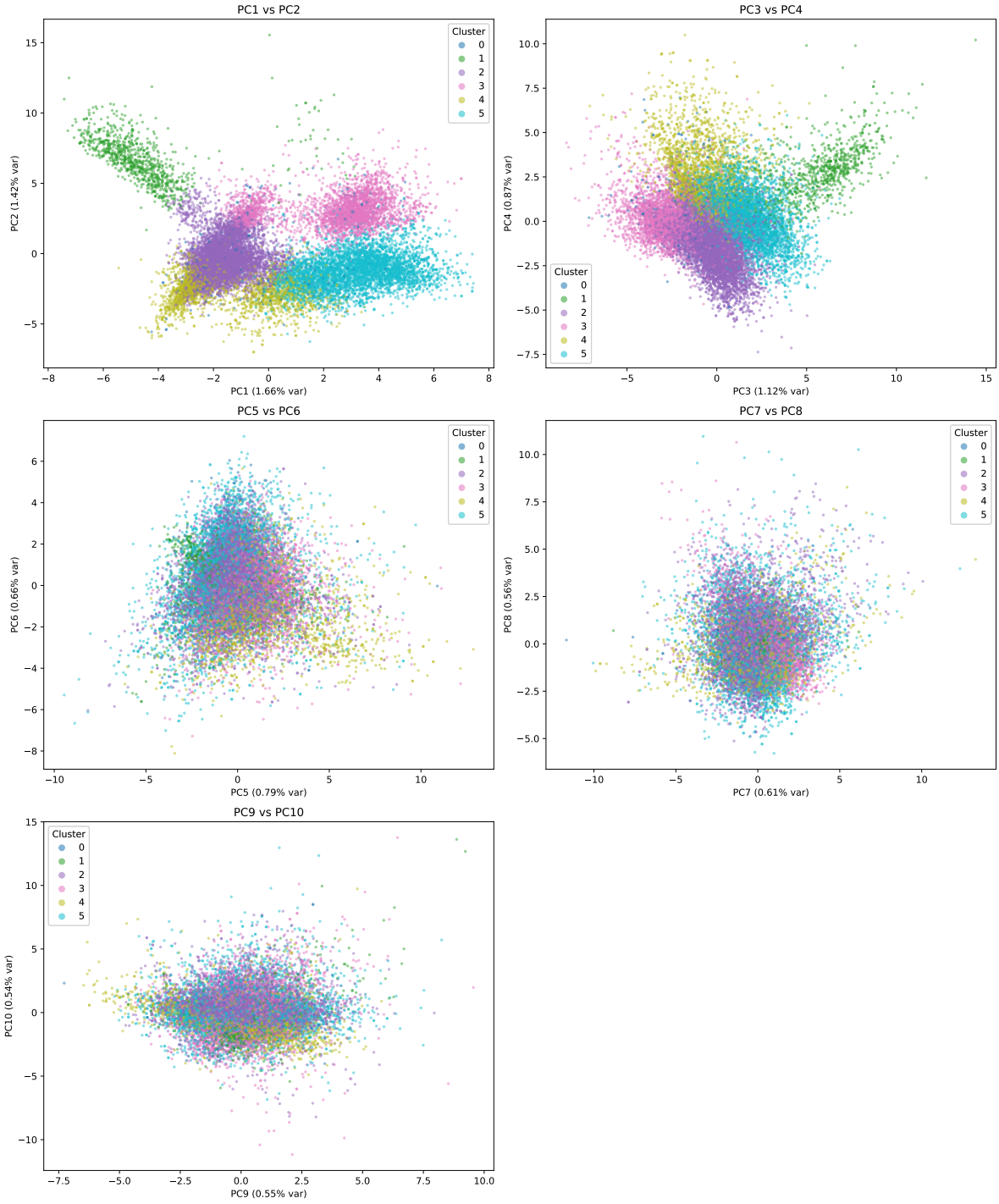



```
best_K = 6

kmeans_final = KMeans(
    n_clusters=best_K,
    random_state=42,
    n_init=20
)

cluster_labels = kmeans_final.fit_predict(X_full_scaled)
```

The elbow plot does not show a sharp bend. The within-cluster variance decreases smoothly as K increases. However, the rate of decrease slows after approximately K = 6. After this, adding more clusters results in only small reductions in within-cluster variance. So, K = 6 was selected.

3(c)

```
from sklearn.decomposition import PCA

pca_all = PCA()
X_pca_all = pca_all.fit_transform(X_full_scaled)
explained_var = pca_all.explained_variance_ratio_

pc_pairs = [(0,1), (2,3), (4,5), (6,7), (8,9)]

fig, axes = plt.subplots(3, 2, figsize=(15,18))
axes = axes.flatten()

for i, (pc_x, pc_y) in enumerate(pc_pairs):
    scatter = axes[i].scatter(
        X_pca_all[:, pc_x],
        X_pca_all[:, pc_y],
        c=cluster_labels,
        cmap="tab10",
        alpha=0.5,
        s=4
    )

    axes[i].set_xlabel(f"PC{pc_x+1} ({explained_var[pc_x]*100:.2f}% var)")
    axes[i].set_ylabel(f"PC{pc_y+1} ({explained_var[pc_y]*100:.2f}% var)")
    axes[i].set_title(f"PC{pc_x+1} vs PC{pc_y+1}")

    legend = axes[i].legend(*scatter.legend_elements(), title="Cluster")
    axes[i].add_artist(legend)

fig.delaxes(axes[5])

plt.tight_layout()
plt.show()
```



We see that PC1 explains about 1.66% of the variance and PC2 explains about 1.42%. These values are small, but that makes sense since the dataset has many features and the variance

is spread out across them.

In the PC1 vs PC2 plot, one cluster is clearly separated along PC1, but the others overlap quite a bit. This suggests that the clusters are not fully separated in just the first two components and that the structure exists across multiple dimensions.

3(d)

We are adding the cluster labels back to the original dataset and then calculating the average feature values for each cluster. This allows us to understand the characteristics of each cluster by looking at which features have higher or lower average values within that cluster compared to others.

```
epi_with_clusters = epi.copy()
epi_with_clusters["cluster"] = cluster_labels

# Average feature values per cluster
cluster_summary = epi_with_clusters.groupby("cluster").mean()

cluster_summary.head()
```

cluster	rating	calories	protein	fat	sodium	3-ingredient recipes	advance prep r
0	3.828965	535.615323	25.782097	28.327957	1025.693011	0.002688	0.008065
1	2.253121	247.304200	2.906924	6.007946	122.392009	0.000000	0.001135
2	3.640431	338.769222	12.946577	18.425099	450.720448	0.001805	0.002406
3	3.831579	434.381101	6.851128	22.242105	222.024168	0.001805	0.005414
4	4.064497	1058.276741	62.070711	66.194089	1294.760415	0.001198	0.013179

```
for c in range(best_K):
    print(f"\nCluster {c}")
    print(cluster_summary.loc[c].sort_values(ascending=False).head(10))
```

```
Cluster 0
sodium          1025.693011
calories         535.615323
fat               28.327957
protein           25.782097
rating             3.828965
```

```
ham                0.596774
bon appétit        0.521505
soy free           0.470430
peanut free        0.451613
jam or jelly       0.422043
Name: 0, dtype: float64

Cluster 1
calories          247.304200
sodium            122.392009
fat                 6.007946
protein             2.906924
rating              2.253121
drink               0.970488
alcoholic           0.921680
cocktail party      0.674234
cocktail            0.416572
house & garden      0.314415
Name: 1, dtype: float64

Cluster 2
sodium          450.720448
calories        338.769222
fat              18.425099
protein          12.946577
rating            3.640431
bon appétit       0.441102
gourmet           0.369751
quick & easy      0.317050
summer            0.199374
vegetarian        0.174227
Name: 2, dtype: float64

Cluster 3
calories       434.381101
sodium         222.024168
fat             22.242105
protein          6.851128
rating           3.831579
dessert          0.905865
peanut free      0.757895
soy free         0.757594
vegetarian       0.738045
```

```
kosher           0.726316
Name: 3, dtype: float64

Cluster 4
sodium           1294.760415
calories         1058.276741
fat                66.194089
protein            62.070711
rating              4.064497
bon appétit         0.483626
dinner              0.410144
peanut free         0.384585
tree nut free       0.370208
soy free            0.347045
Name: 4, dtype: float64

Cluster 5
sodium           434.684610
calories         328.146160
fat               17.943979
protein           11.157650
rating             3.842026
peanut free        0.974458
soy free           0.931960
tree nut free      0.891178
pescatarian        0.739214
kosher             0.715175
Name: 5, dtype: float64
```

Interpretation of Clusters:

Cluster 0 seems to represent savory main dishes, since it has relatively high sodium, calories, and protein, along with tags like ham. **Cluster 1 is clearly alcoholic drinks**, given the very high drink and alcoholic tags and low protein and fat. **Cluster 2 looks like more everyday or lighter meals**, with moderate nutrition values and tags such as "quick and easy" and "vegetarian". **Cluster 3 appears to be desserts**, since the dessert tag is very high and the nutrition profile fits baked goods. **Cluster 4** stands out as very high in calories, fat, and protein, suggesting **large or rich dinner entrées, or high-protein dishes**. Finally, **Cluster 5 seems to group together dietary-restriction recipes**, with very high peanut-free, soy-free, and pescatarian tags. Overall, most of the clusters are fairly interpretable, even though there is still some overlap between them.