DATA 221 Homework 5

Chris Low

Problem 1

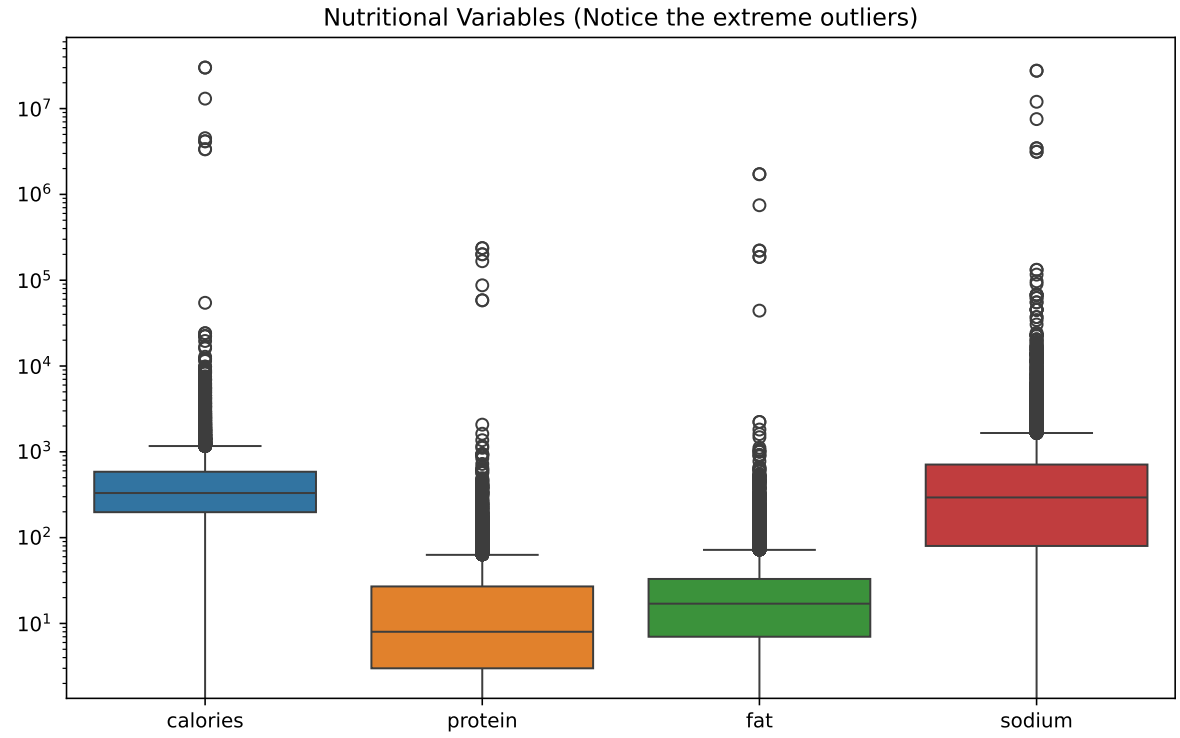
Exploratory Data Analysis

```
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()

# # Step C: Separate Features from the Outcome Variable
# # Problem 1 asks to do PCA on features. Problem 3c mentions "it will include 'cake' this t
# # which implies 'cake' should be excluded from the features in Problem 1.
# if 'cake' in epi.columns:
#     y = epi['cake'] # The outcome of interest
#     X = epi.drop(columns=['cake'])
# else:
#     X = epi.copy()

# # Step D: Scale the Features
# # This is crucial for PCA and Regularized Logistic Regression
# scaler = StandardScaler()
# X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns, index=X.index)

# print("Pre-processing complete. X_scaled is ready for PCA.")
```

```
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)

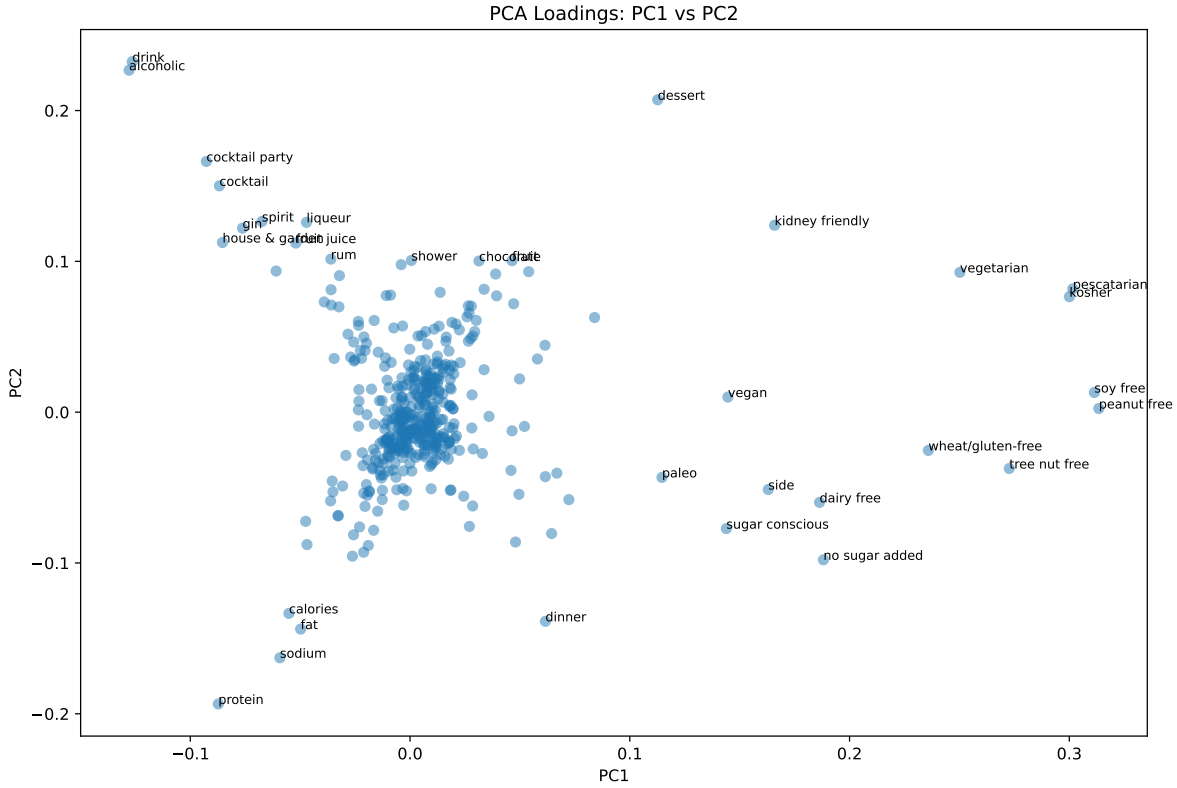
```
from sklearn.decomposition import PCA

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

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])
```

```
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]
```

1(b)

```python
# 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.