In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.cluster import KMeans, AgglomerativeClustering
import scipy.cluster.hierarchy as shc
import scipy.stats as stats
from sklearn import metrics
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor
```

In [ ]:
```
df = pd.read_csv('Adult_income_dataset.csv')
df.head()
```

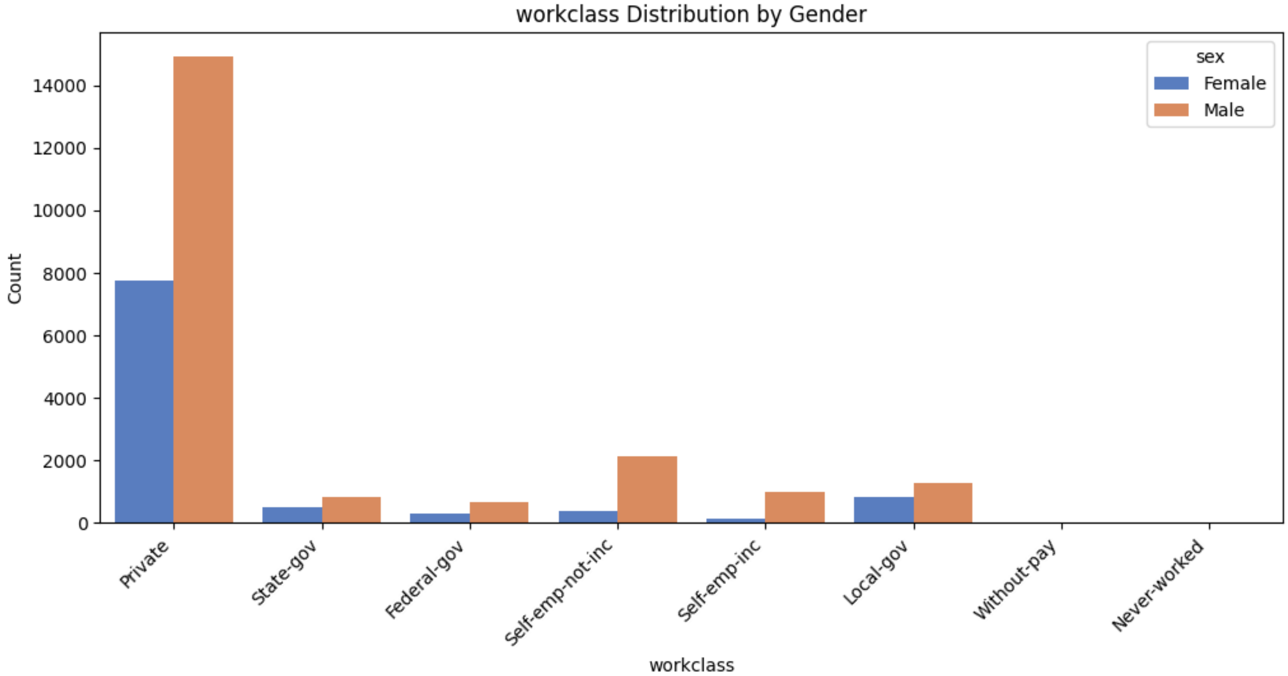
Out[ ]:

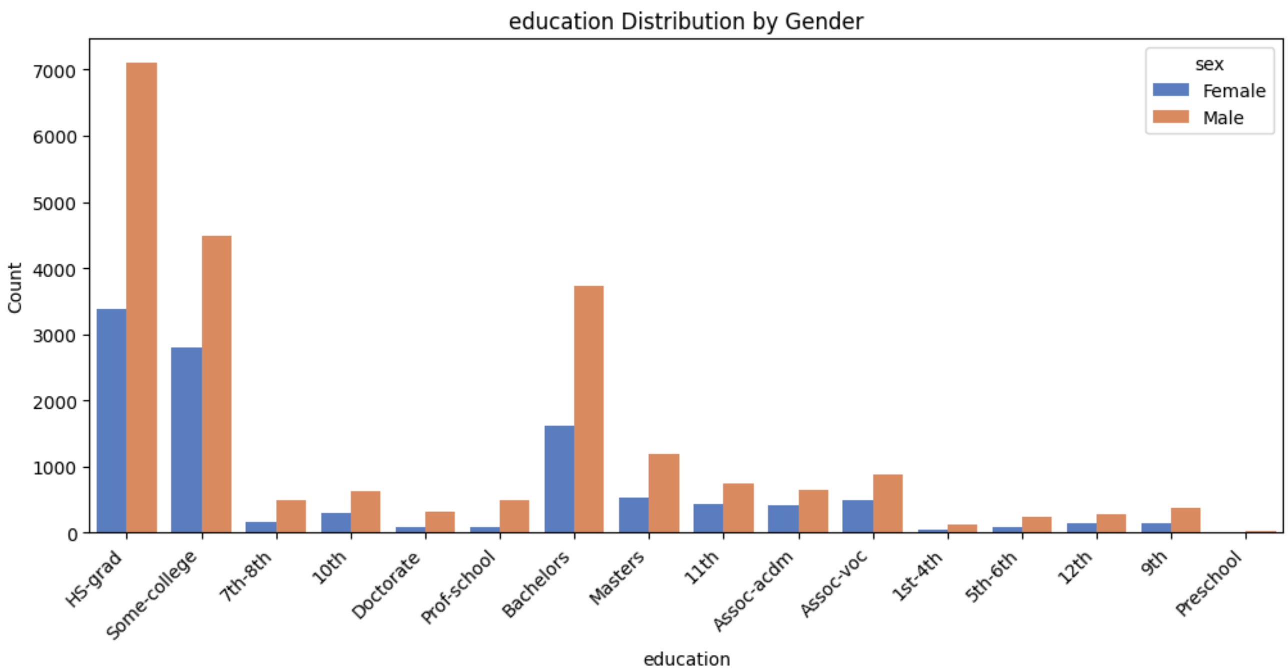
	age	workclass	Final_Weight_of_Income	education	education.num	marital.status
0	90	?	77053	HS-grad	9	Widowed
1	82	Private	132870	HS-grad	9	Widowed
2	66	?	186061	Some-college	10	Widowed
3	54	Private	140359	7th-8th	4	Divorced
4	41	Private	264663	Some-college	10	Separated

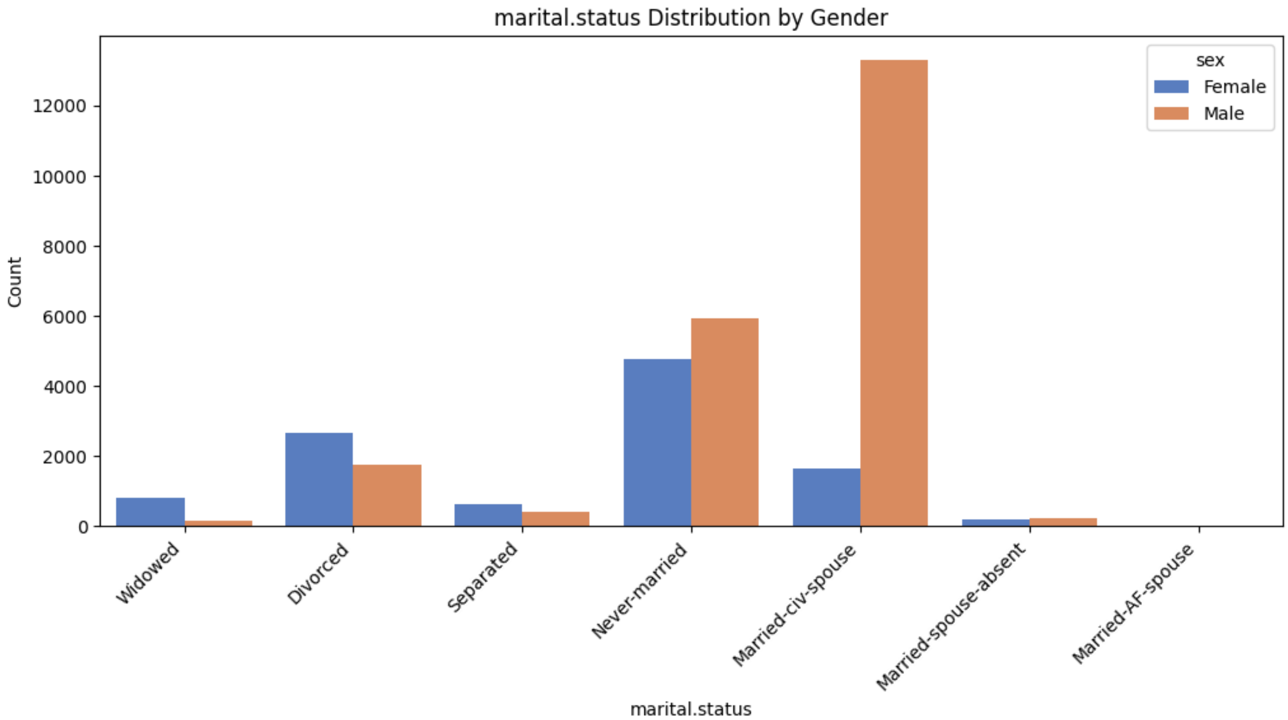
1. a)

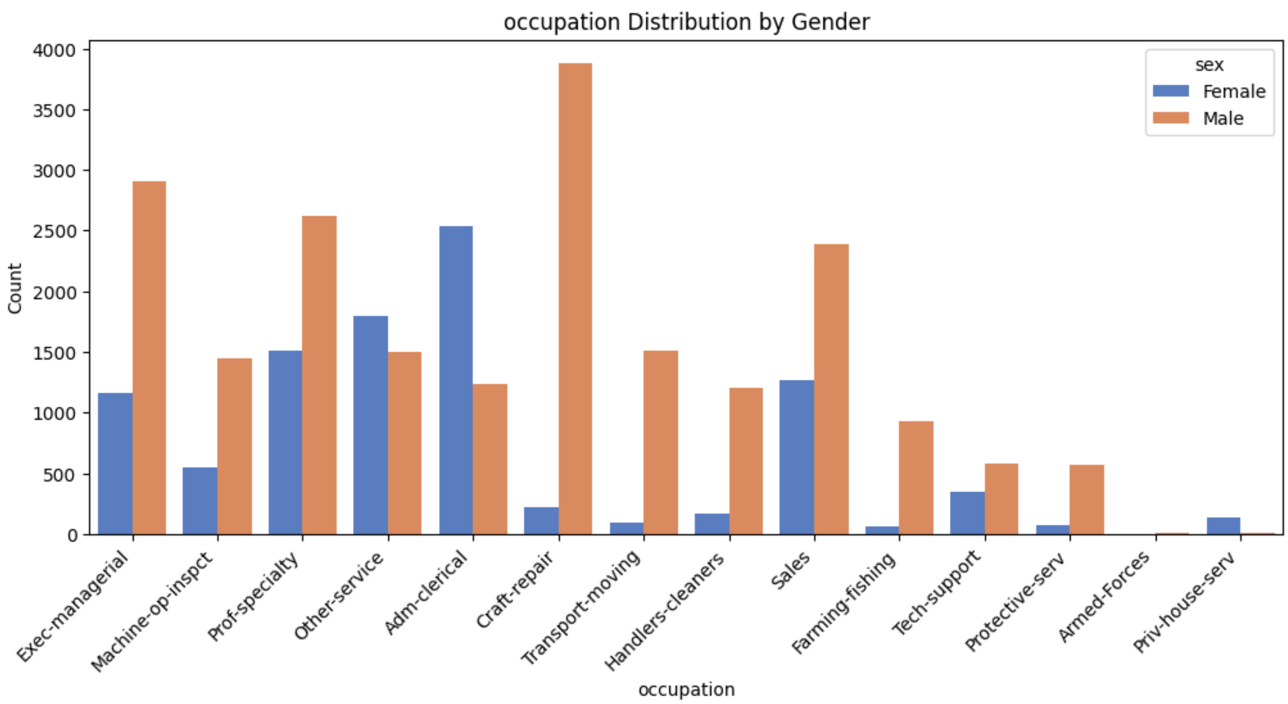
In [ ]:
```
# Replace "?" with NaN
df.replace("?", np.nan, inplace=True)

# Drop rows with NaN values
df_cleaned = df.dropna()
numerical_data = ['age','Final_Weight_of_Income', 'education.num', 'capital.
encoded_data = df.drop(columns=numerical_data).columns

for c in encoded_data:
```

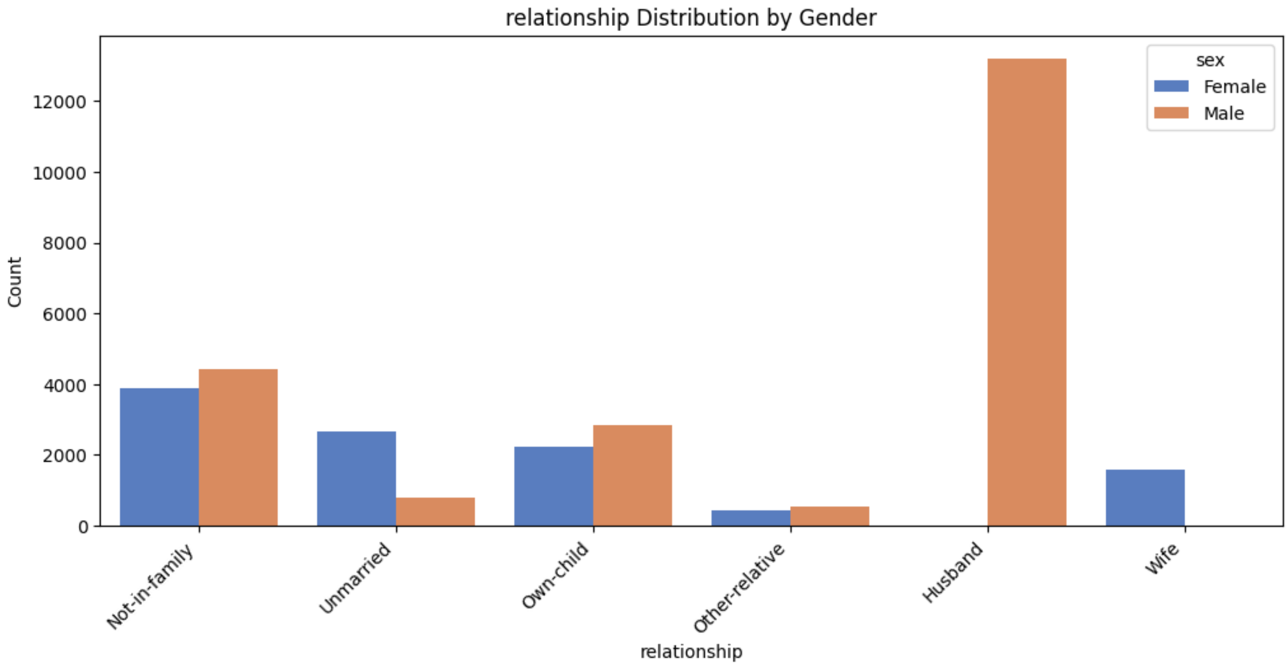
```
    plt.figure(figsize=(12,5))
    sns.countplot(data=df[encoded_data], x=c, hue=df['sex'], palette='muted'
    plt.title(f'{c} Distribution by Gender')
    plt.xlabel(c)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
```

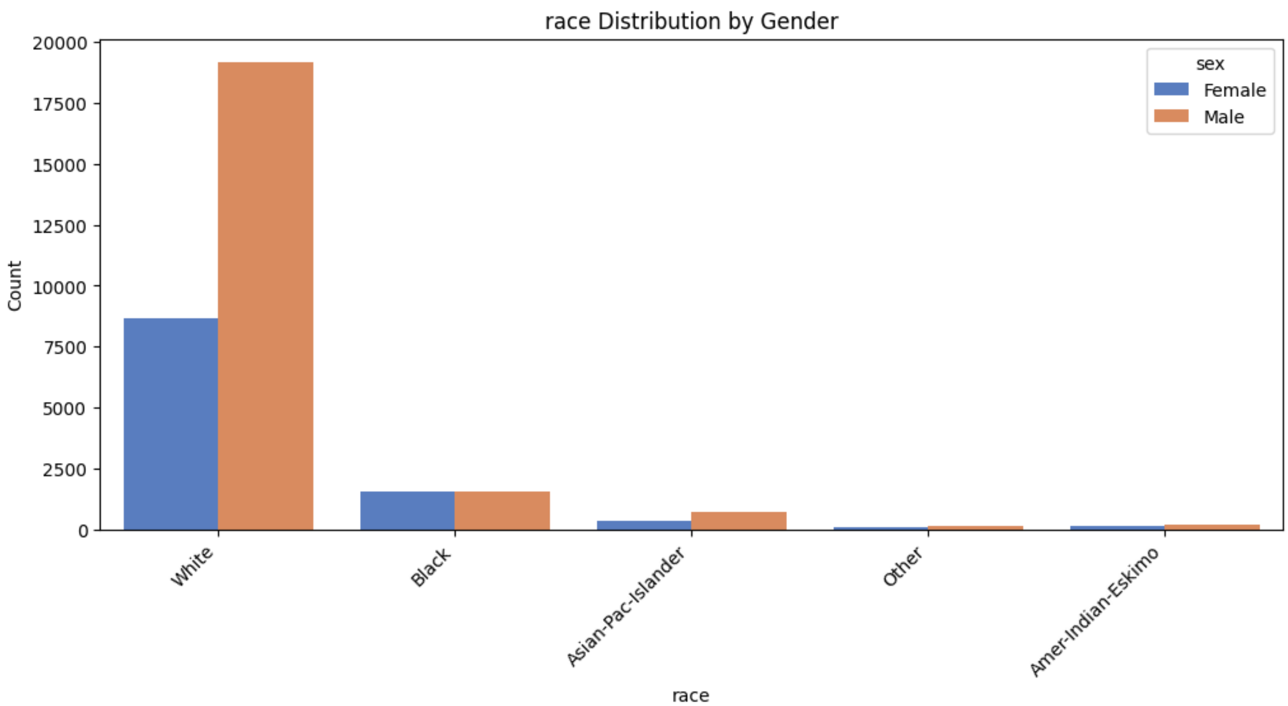


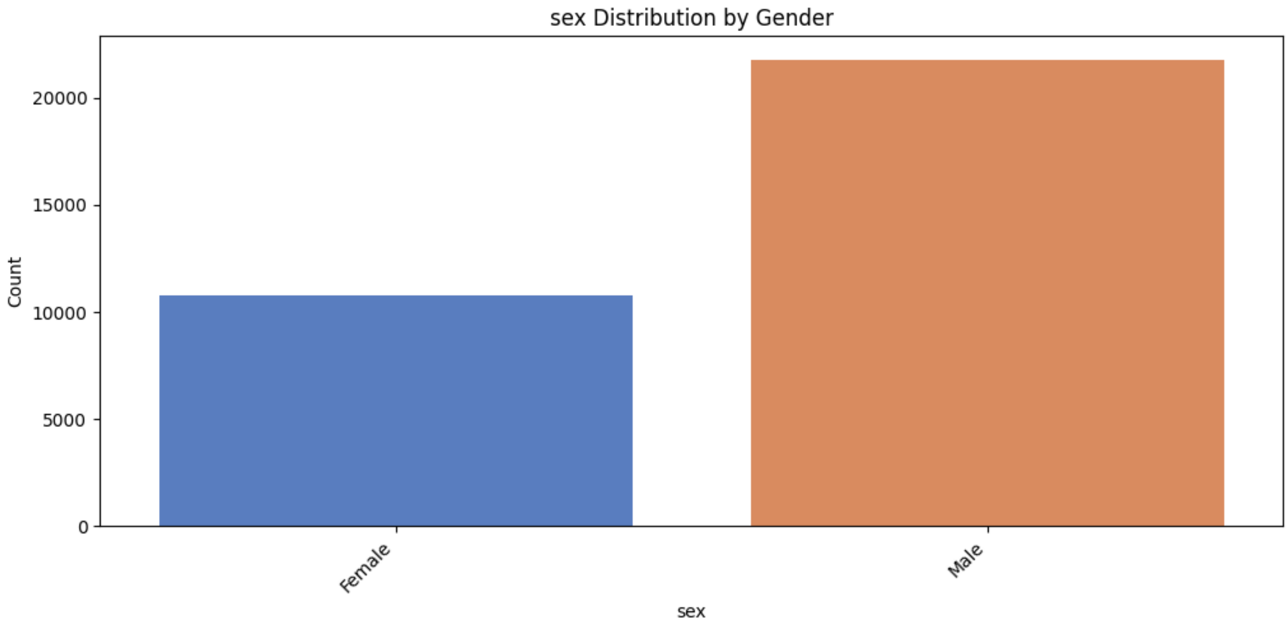


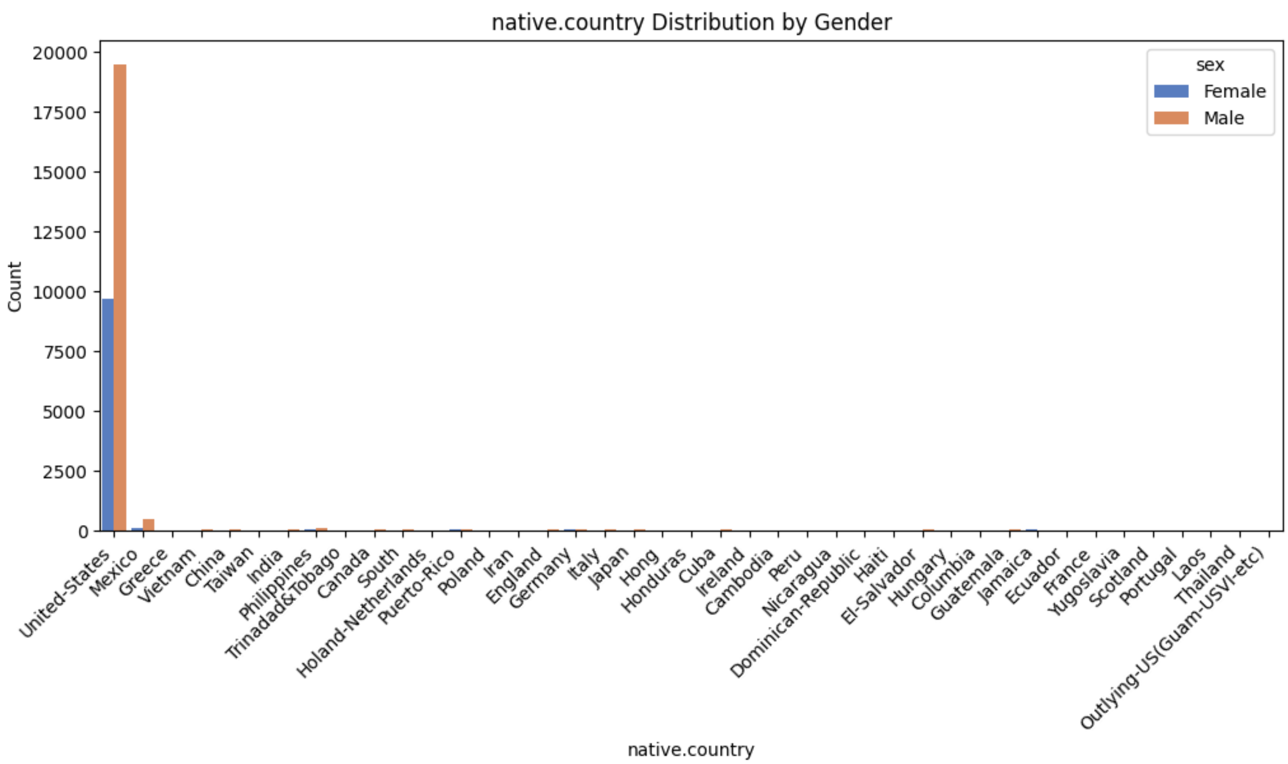

marital.status Distribution by Gender
sex
Female
Male
Count
0
2000
4000
6000
8000
10000
12000
Widowed
Divorced
Separated
Never-married
Married-civ-spouse
Married-spouse-absent
Married-AF-spouse
marital.status

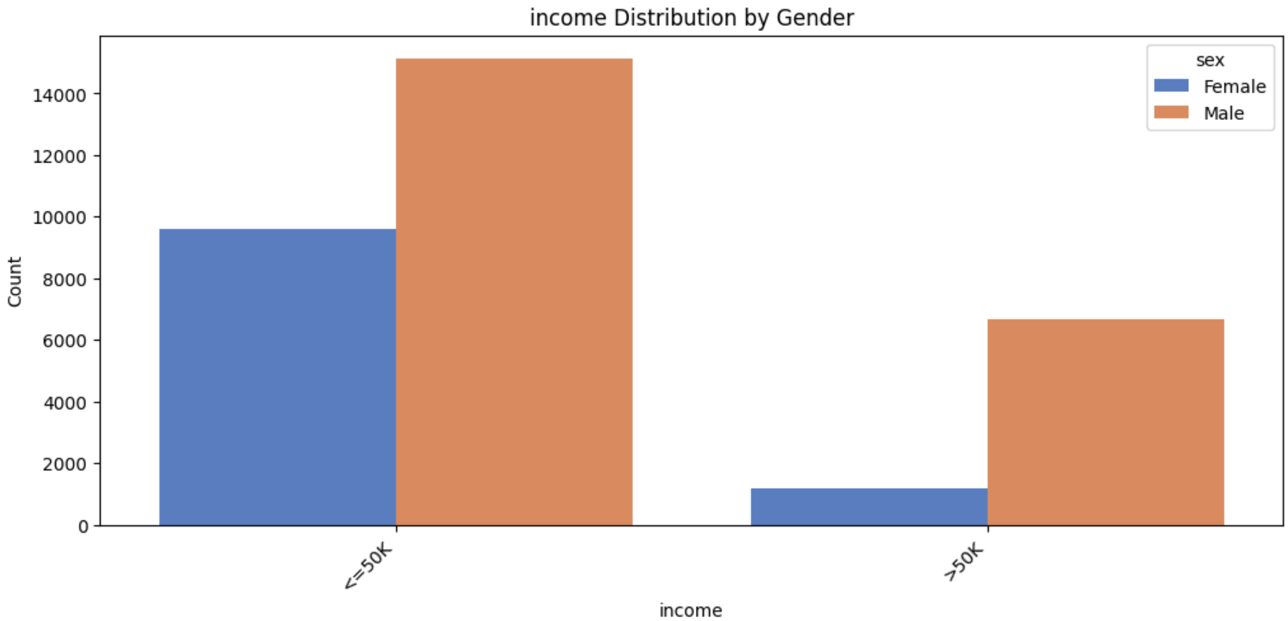

occupation Distribution by Gender
sex
Female
Male
Count
0
500
1000
1500
2000
2500
3000
3500
4000
Exec-managerial
Machine-op-inspct
Prof-specialty
Other-service
Adm-clerical
Craft-repair
Transport-moving
Handlers-cleaners
Sales
Farming-fishing
Tech-support
Protective-serv
Armed-Forces
Priv-house-serv
occupation


relationship Distribution by Gender
sex
Female
Male
Count
0
2000
4000
6000
8000
10000
12000
Not-in-family
Unmarried
Own-child
Other-relative
Husband
Wife
relationship


race Distribution by Gender
sex
Female
Male
Count
0
2500
5000
7500
10000
12500
15000
17500
20000
White
Black
Asian-Pac-Islander
Other
Amer-Indian-Eskimo
race

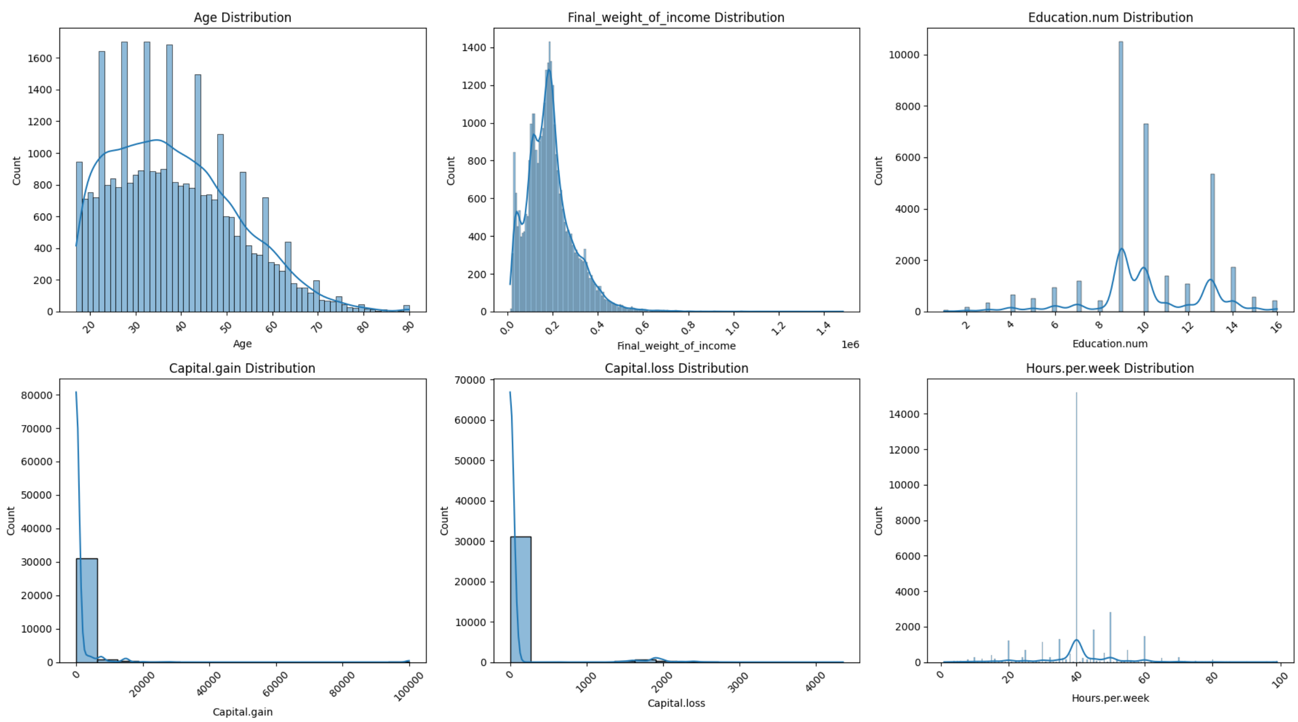

sex Distribution by Gender
Count
20000
15000
10000
5000
0
Female
Male
sex


native.country Distribution by Gender
sex
Female
Male
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
United-States
Mexico
Greece
Vietnam
China
Taiwan
India
Philippines
Trinadad&Tobago
Canada
South
Holand-Netherlands
Puerto-Rico
Poland
Iran
England
Germany
Italy
Japan
Hong
Honduras
Cuba
Ireland
Cambodia
Peru
Nicaragua
Dominican-Republic
Haiti
El-Salvador
Hungary
Columbia
Guatemala
Jamaica
Ecuador
France
Yugoslavia
Scotland
Portugal
Laos
Thailand
Outlying-US(Guam-USVI-etc)
native.country



In [ ]:
```
for col in encoded_data:
    encoder = LabelEncoder()
    df[col] = encoder.fit_transform(df[col])
```

In [ ]:
```
# Create subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))

# Plot histograms
for ax, c in zip(axs.flatten(), numerical_data):
    sns.histplot(data=df, x=c, kde=True, ax=ax)
    ax.set_title(f'{c.capitalize()} Distribution')
    ax.set_xlabel(c.capitalize())
    ax.set_ylabel('Count')
    ax.tick_params(axis='x', rotation=45)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()

# Print summary statistics
summary = df.describe(include='all')
summary
```



Out[ ]:

	age	workclass	Final_Weight_of_Income	education	educatior
count	32561.000000	32561.000000	3.256100e+04	32561.000000	32561.0C
mean	38.581647	3.376371	1.897784e+05	10.298210	10.08
std	13.640433	1.582038	1.055500e+05	3.870264	2.57
min	17.000000	0.000000	1.228500e+04	0.000000	1.0C
25%	28.000000	3.000000	1.178270e+05	9.000000	9.0C
50%	37.000000	3.000000	1.783560e+05	11.000000	10.0C
75%	48.000000	3.000000	2.370510e+05	12.000000	12.0C
max	90.000000	8.000000	1.484705e+06	15.000000	16.0C

1. c)

In [ ]:

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Selecting numeric columns for PCA
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns

# Standardizing the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df[numeric_columns])
```

```
# PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_data)

# Interpret PCA results
pca_results = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2']
explained_variance = pca.explained_variance_ratio_

print(pca_results.head())
print(f'\nExplained variance fro principal component 1 and 2:',explained_var
```

```
        PC1       PC2
0 -0.935749 -0.451300
1 -0.346099  0.730634
2  0.552984  1.145708
3  0.158072 -0.561159
4  0.273239  1.909548

Explained variance fro principal component 1 and 2: [0.15672459 0.0979157 ]
```

The first principal component expalined most of the variance in the datasets. And there are 15 prinicipal components which reflect to the 15 features in the datasets.

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(principal_components, df

from sklearn.linear_model import LogisticRegression

# Logistic Regression
model = LogisticRegression()
model.fit(X_train, y_train)

# Interpret the results
y_pred = model.predict(X_test)
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
```

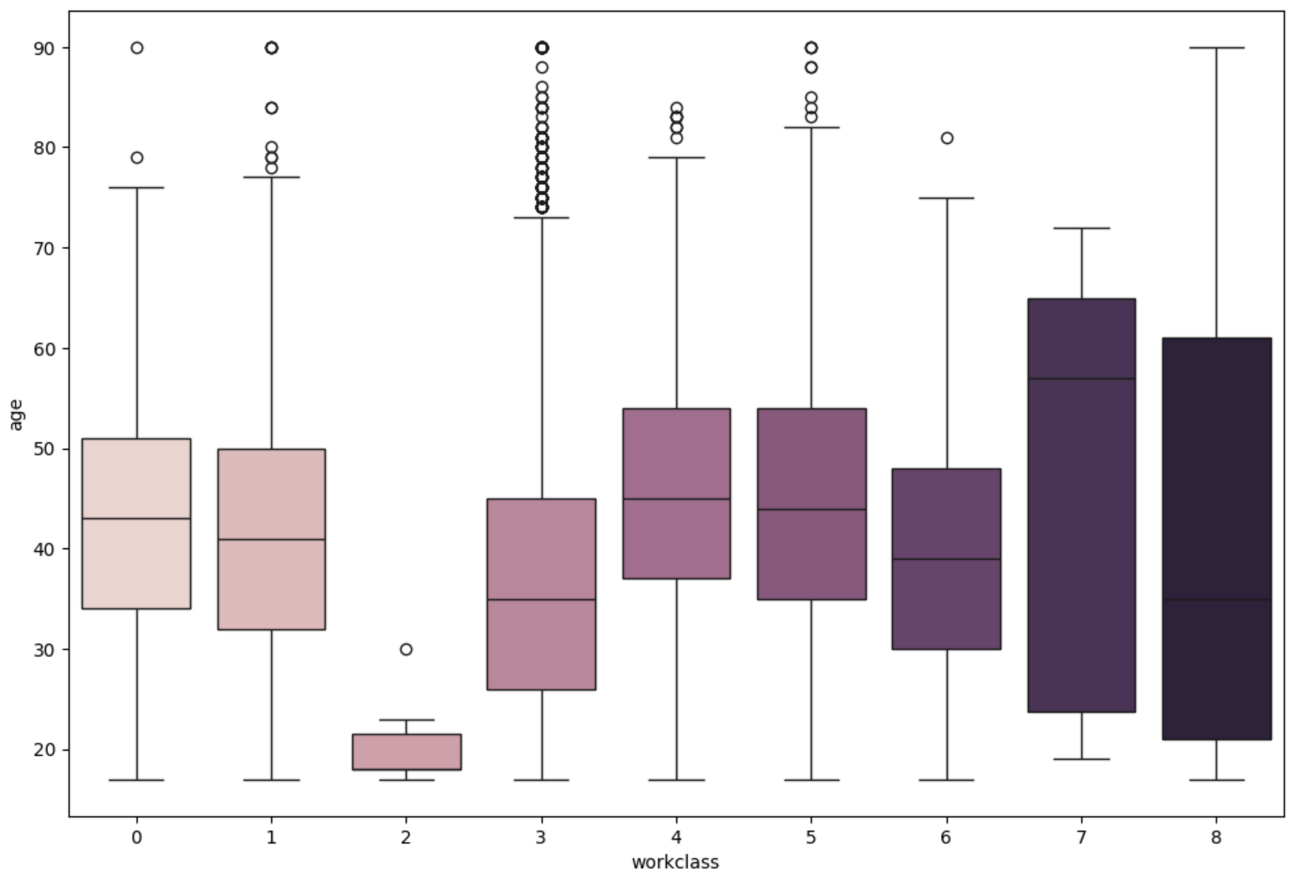
```
Coefficients: [[-2.39506159  1.41208646]]
Intercept: [-3.03354889]
```

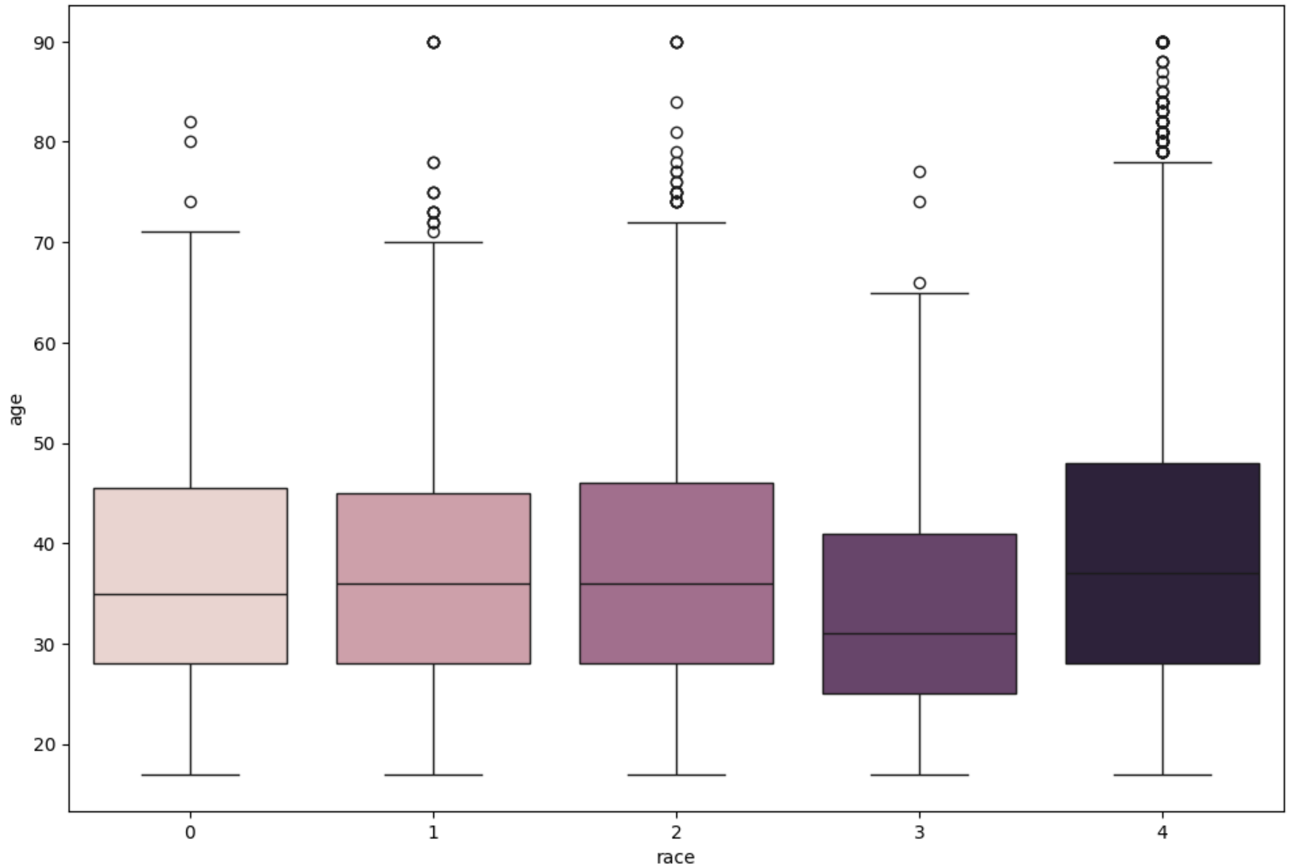
In [ ]:

```
# Model evaluation
accuracy = metrics.accuracy_score(y_test, y_pred)
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
class_report = metrics.classification_report(y_test, y_pred)

print('Accuracy:', accuracy)
print('Confusion Matrix:\n', conf_matrix)
print('Classification Report:\n', class_report)
```

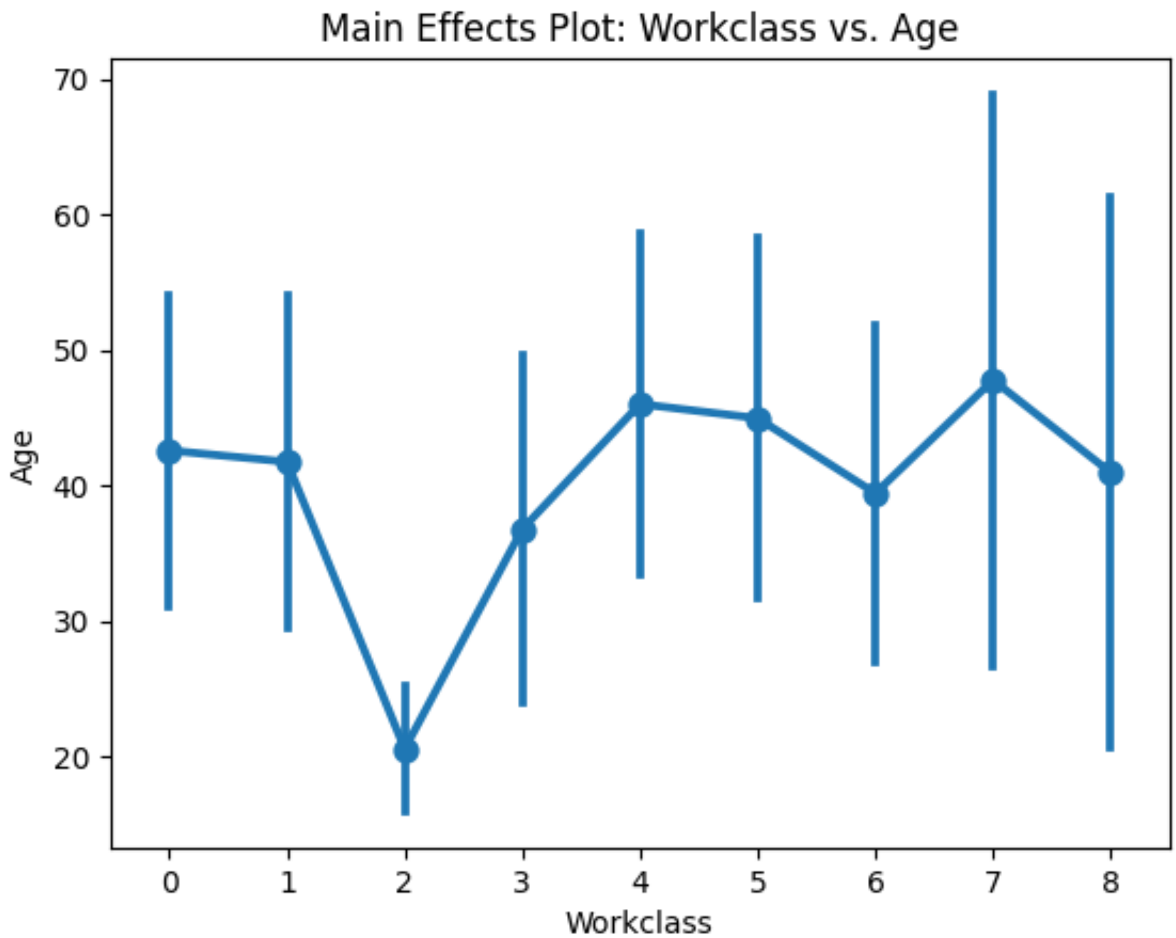
```
Accuracy:
 0.9226163058498388
Confusion Matrix:
 [[4774  202]
 [ 302 1235]]
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.96      0.95      4976
           1       0.86      0.80      0.83      1537

    accuracy                           0.92      6513
   macro avg       0.90      0.88      0.89      6513
weighted avg       0.92      0.92      0.92      6513
```

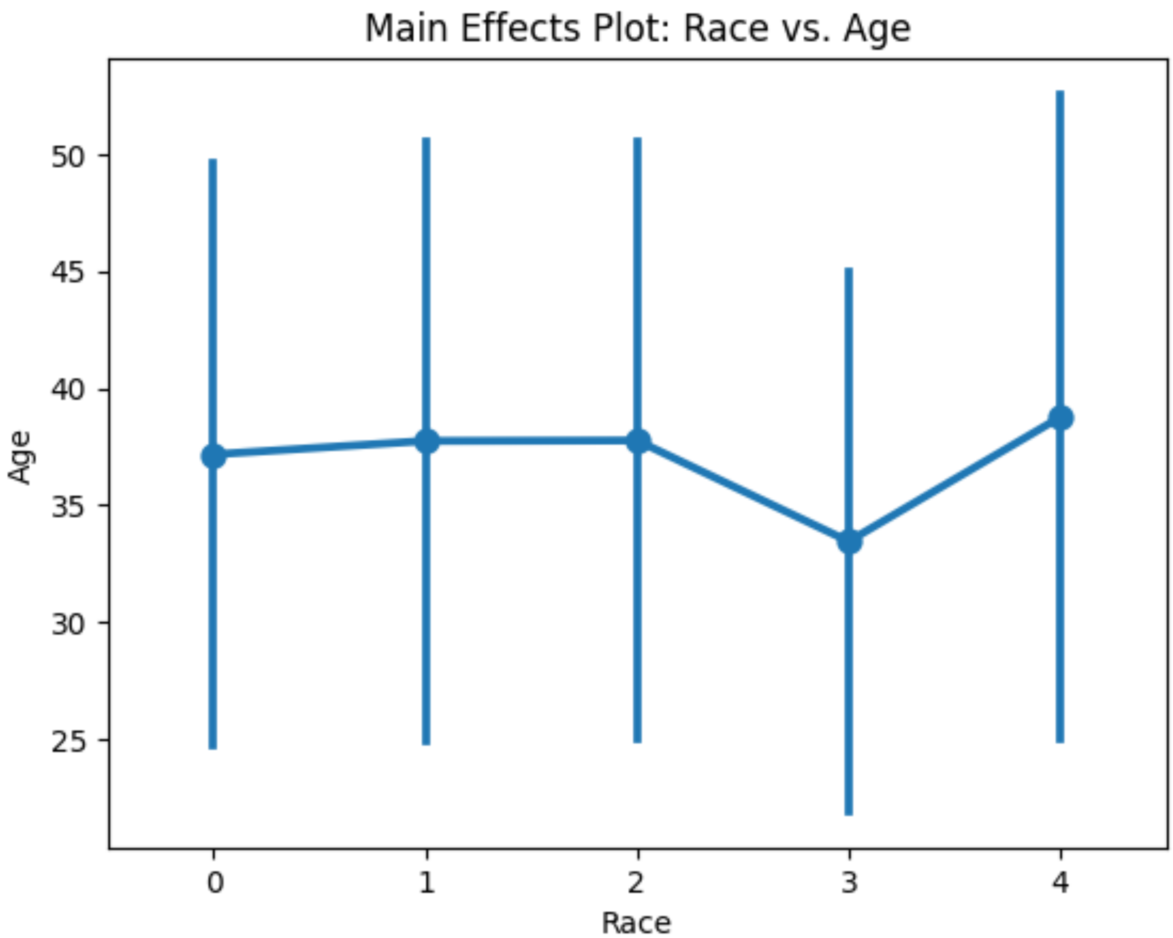
In [ ]:
```
# Boxplots
plt.figure(figsize=(12,8))
sns.boxplot(x='workclass', y='age', data=df, hue='workclass', legend=False)
plt.show()

plt.figure(figsize=(12,8))
sns.boxplot(x='race', y='age', data=df, hue='race', legend=False)
plt.show()
```





In [ ]:
```
# Main Effect Plot
sns.pointplot(x='workclass', y='age', data=df, errorbar='sd', markers='o')
plt.title('Main Effects Plot: Workclass vs. Age')
plt.xlabel('Workclass')
plt.ylabel('Age')
plt.show()

sns.pointplot(x='race', y='age', data=df, errorbar='sd', markers='o')
plt.title('Main Effects Plot: Race vs. Age')
plt.xlabel('Race')
plt.ylabel('Age')
plt.show()
```


Main Effects Plot: Workclass vs. Age
Age
70
60
50
40
30
20
Workclass
0
1
2
3
4
5
6
7
8



In [ ]:
```
# Two-Way ANOVA
model = ols('age ~ C(workclass) + C(race) + C(workclass):C(race)', data=df).
anova_table = sm.stats.anova_lm(model, typ=2)

print(f'Looking at the ANOVA table is clear to see that there is statistical
```

```
Looking at the ANOVA table is clear to see that there is statistical signifi
cance between the variables as their p-value is smaller than 0.05:
                            sum_sq       df           F        PR(>F)
C(workclass)          3.263052e+05      8.0  230.643011  0.000000e+00
C(race)               7.876841e+03      4.0   11.135209  5.022650e-09
C(workclass):C(race)  2.532156e+04     32.0    4.474524  3.441138e-02
Residual              5.751188e+06  32521.0         NaN           NaN
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packa
ges/statsmodels/base/model.py:1894: ValueWarning: covariance of constraints
does not have full rank. The number of constraints is 32, but rank is 1
  warnings.warn('covariance of constraints does not have full '
```

4.

In [ ]:
```
cereal_df = pd.read_csv('Cereal_dataset.csv')
```

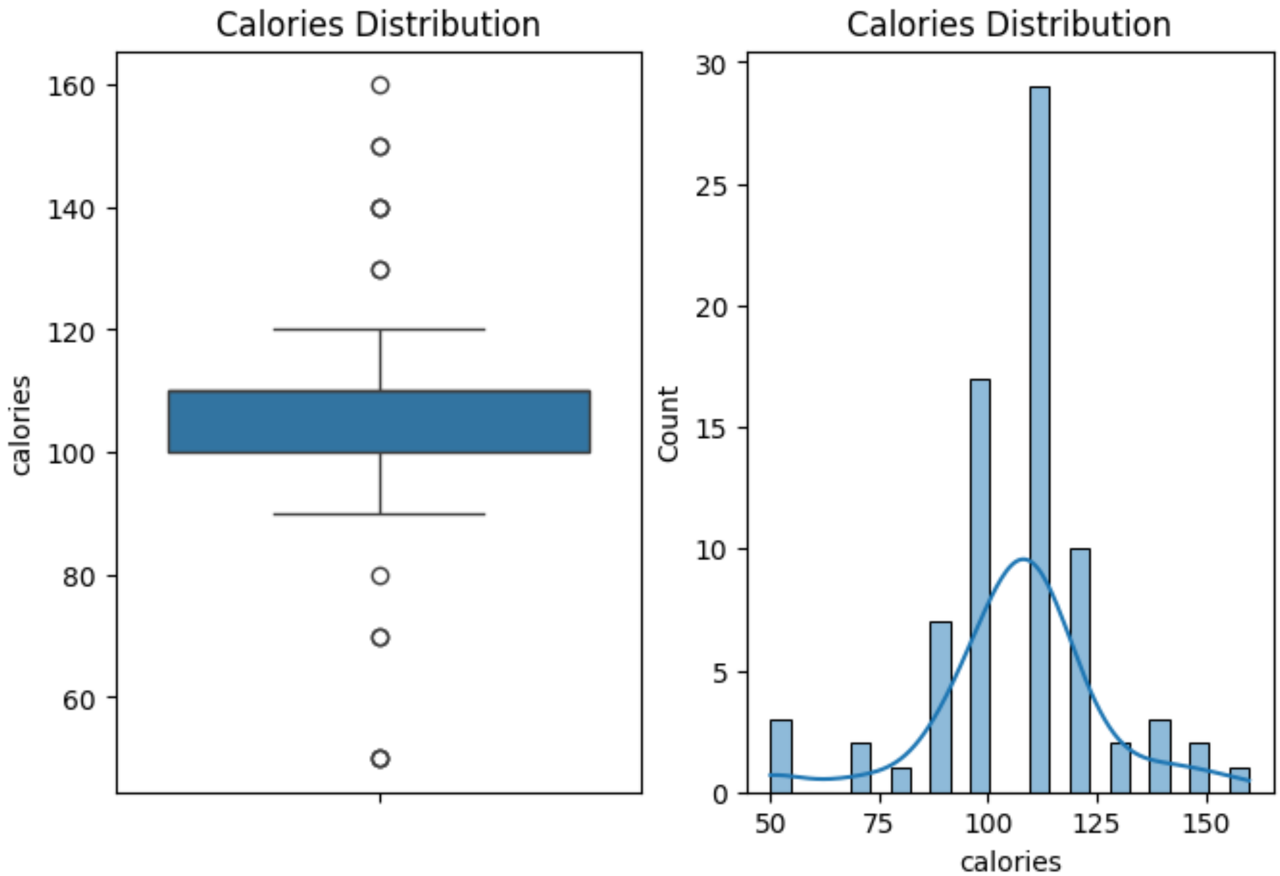
```
cereal_df.head()
```

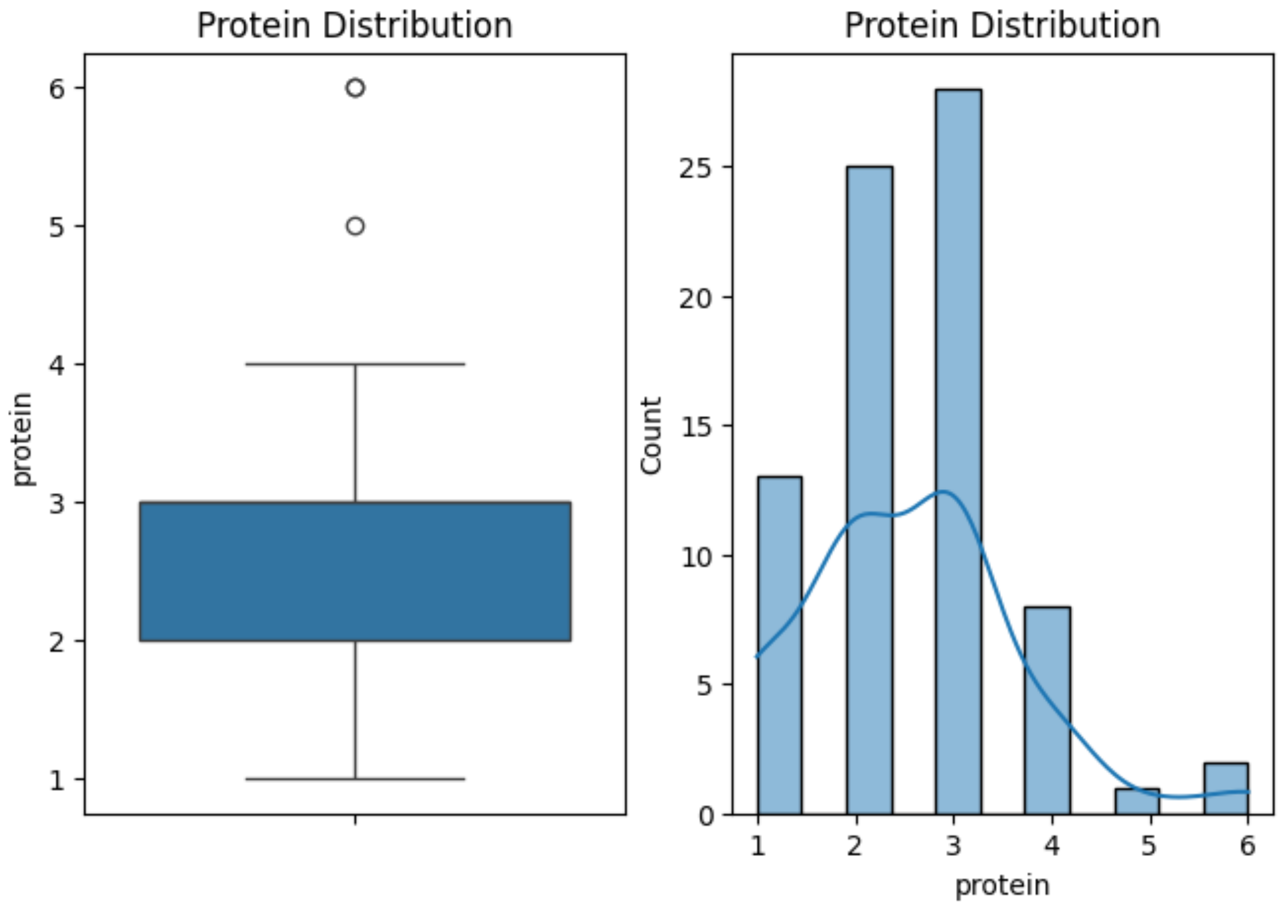
Out[ ]:

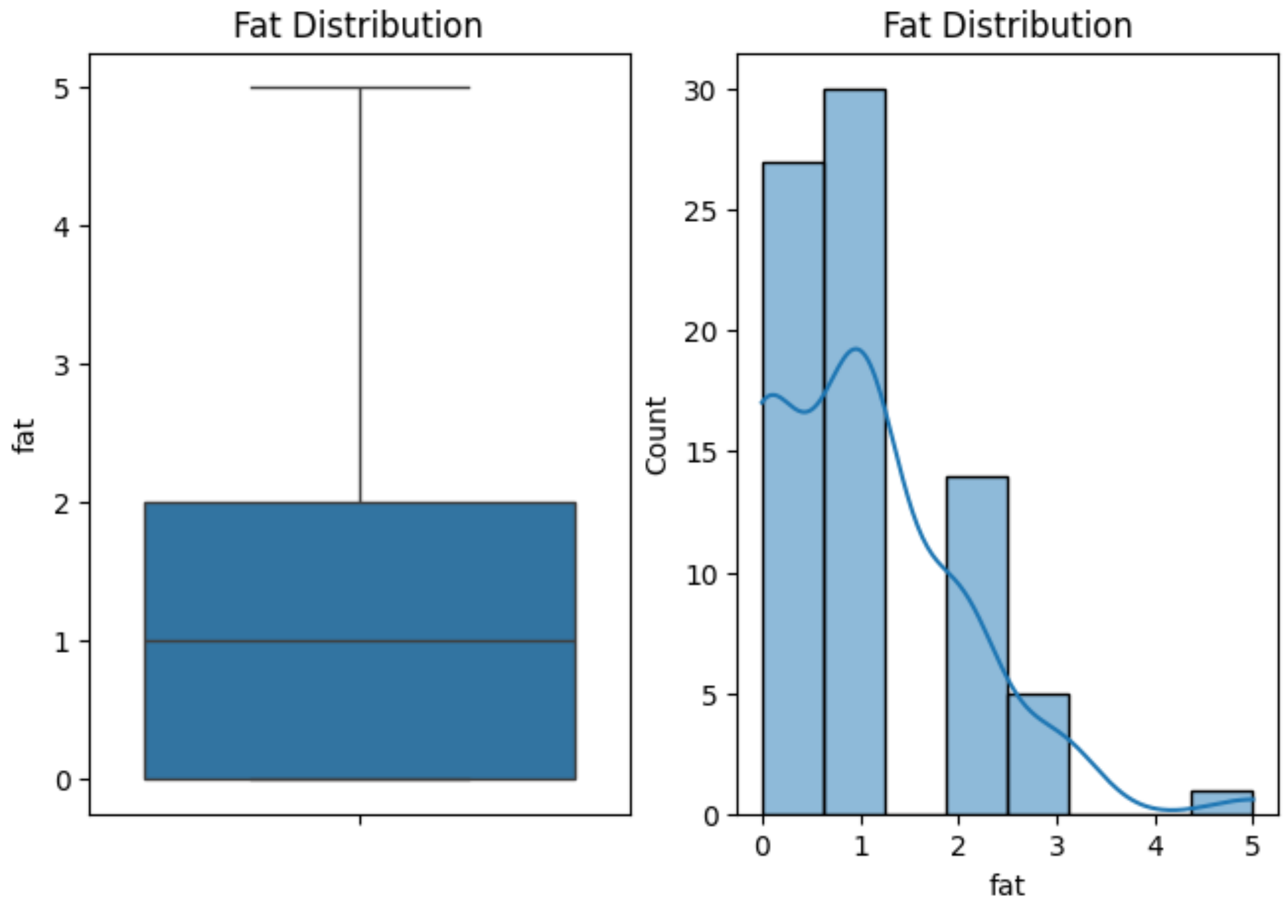
	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vi
0	100% Bran	N	C	70	4	1	130	10.0	5.0	6	280	
1	100% Natural Bran	Q	C	120	3	5	15	2.0	8.0	8	135	
2	All-Bran	K	C	70	4	1	260	9.0	7.0	5	320	
3	All-Bran with Extra Fiber	K	C	50	4	0	140	14.0	8.0	0	330	
4	Almond Delight	R	C	110	2	2	200	1.0	14.0	8	-1	

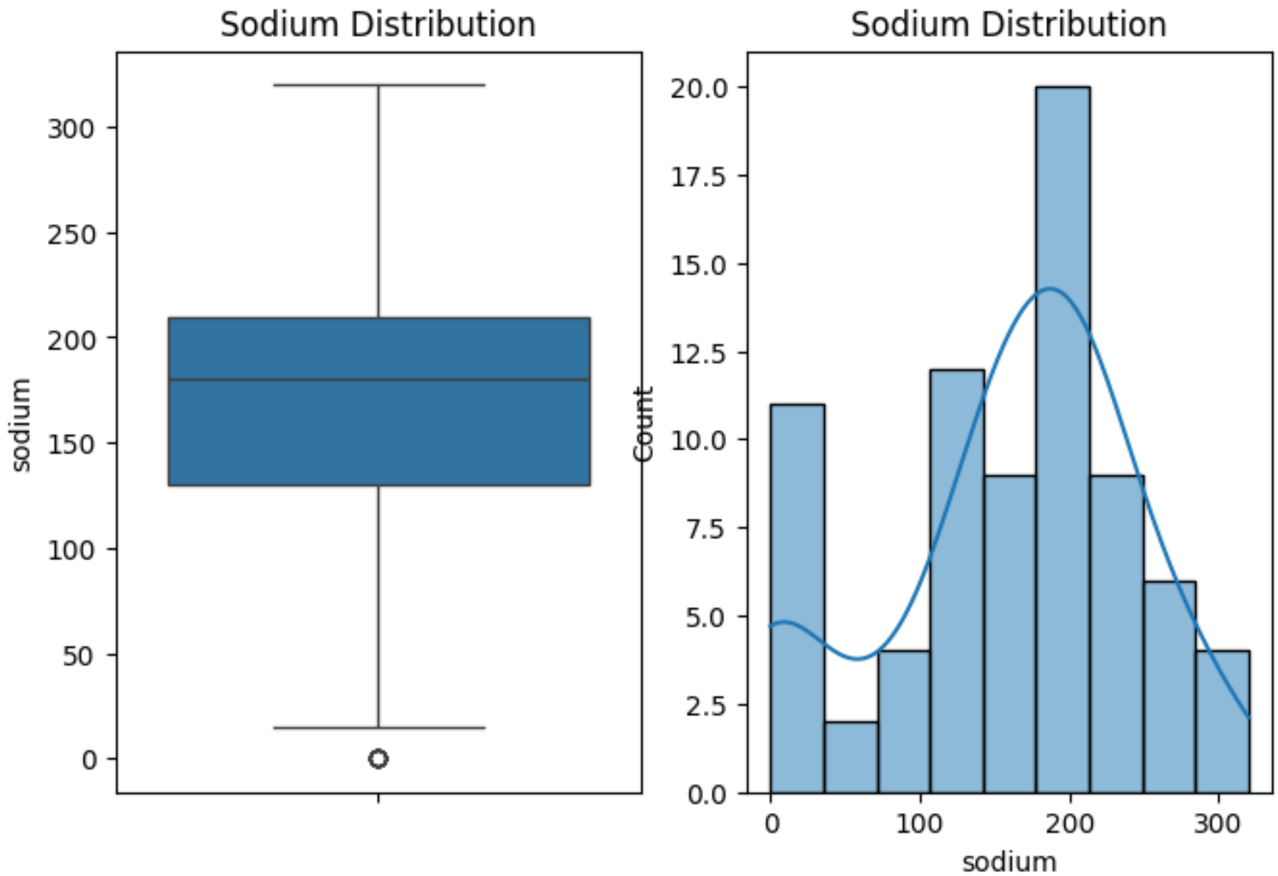
In [ ]:

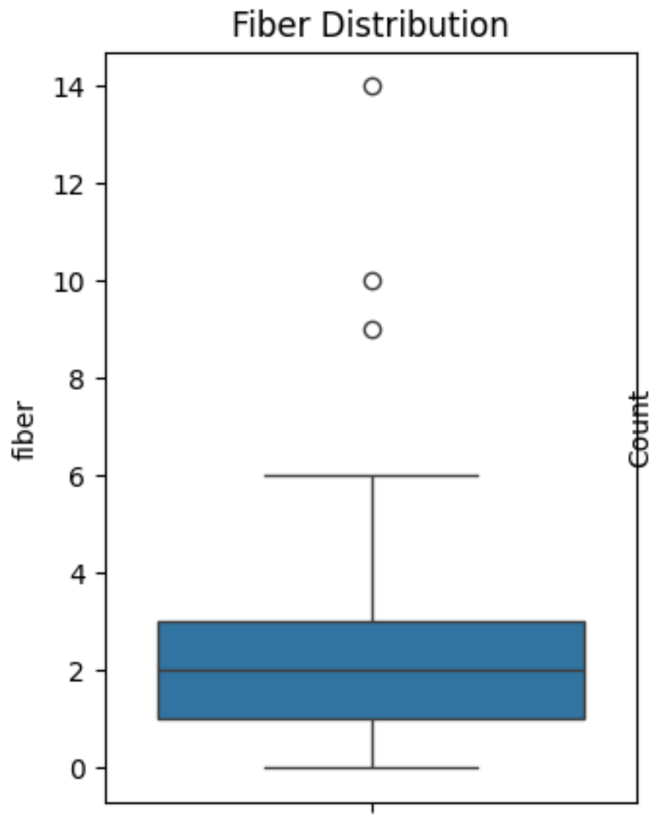
```
numeric_columns = cereal_df.drop(columns=['name', 'mfr', 'type']).columns

for col in numeric_columns:
    plt.figure(figsize=(12,5))

    plt.subplot(1,3,1)
    sns.boxplot(cereal_df[col])
    plt.title(f'{col.capitalize()} Distribution')

    plt.subplot(1,3,2)
    sns.histplot(cereal_df[col], kde=True)
    plt.title(f'{col.capitalize()} Distribution')
```

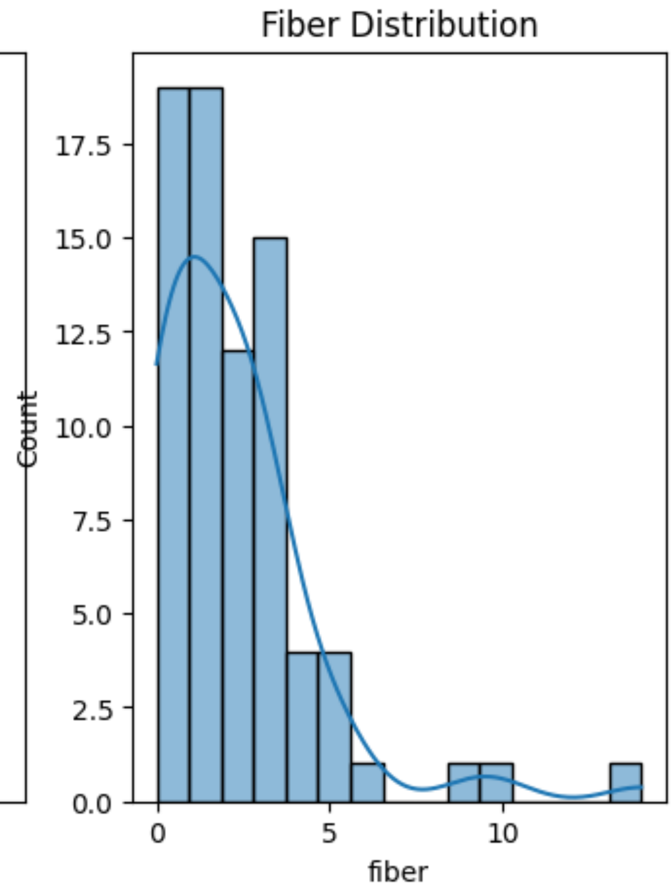

Calories Distribution
calories
160
140
120
100
80
60
Calories Distribution
Count
30
25
20
15
10
5
0
50
75
100
125
150
calories

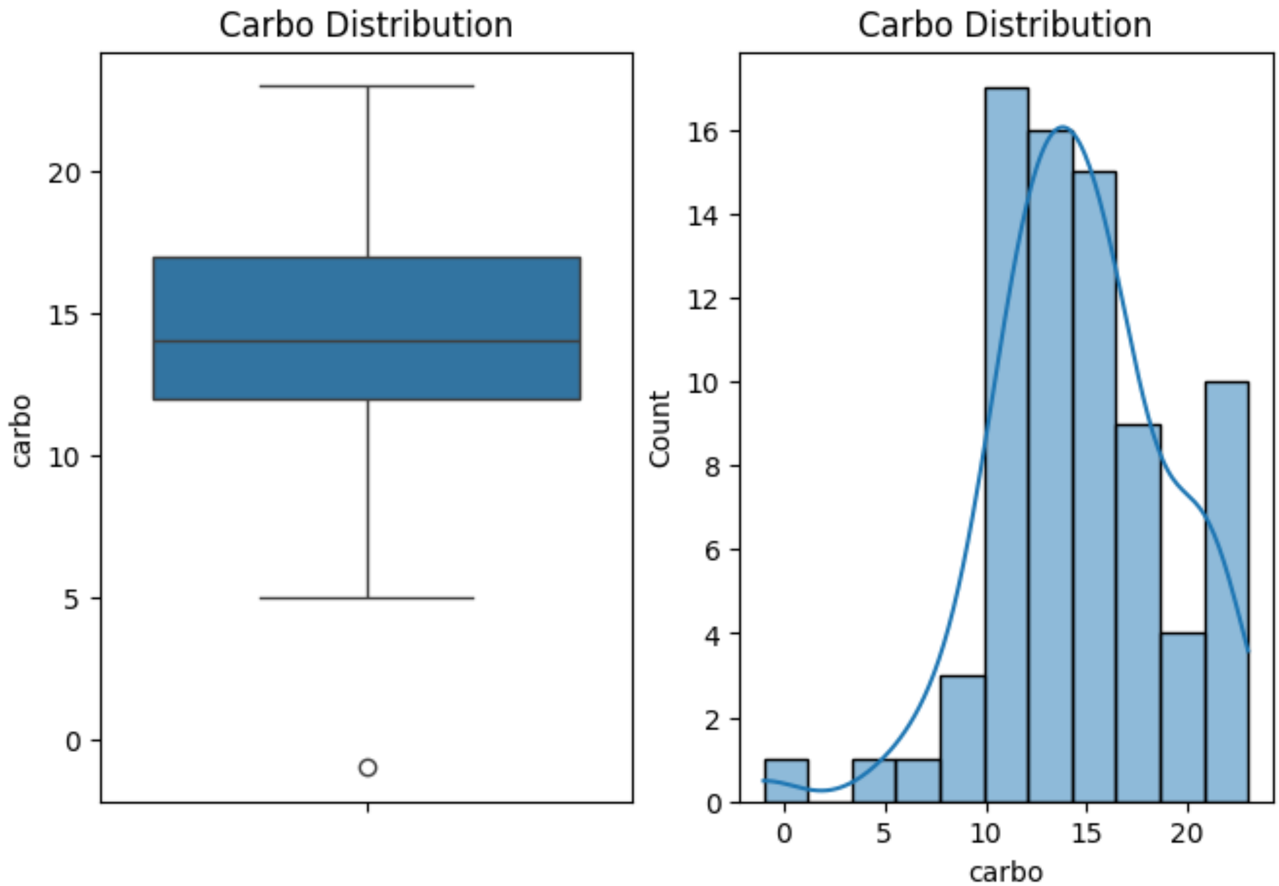

Protein Distribution
protein
1
2
3
4
5
6
Protein Distribution
Count
0
5
10
15
20
25
1
2
3
4
5
6
protein

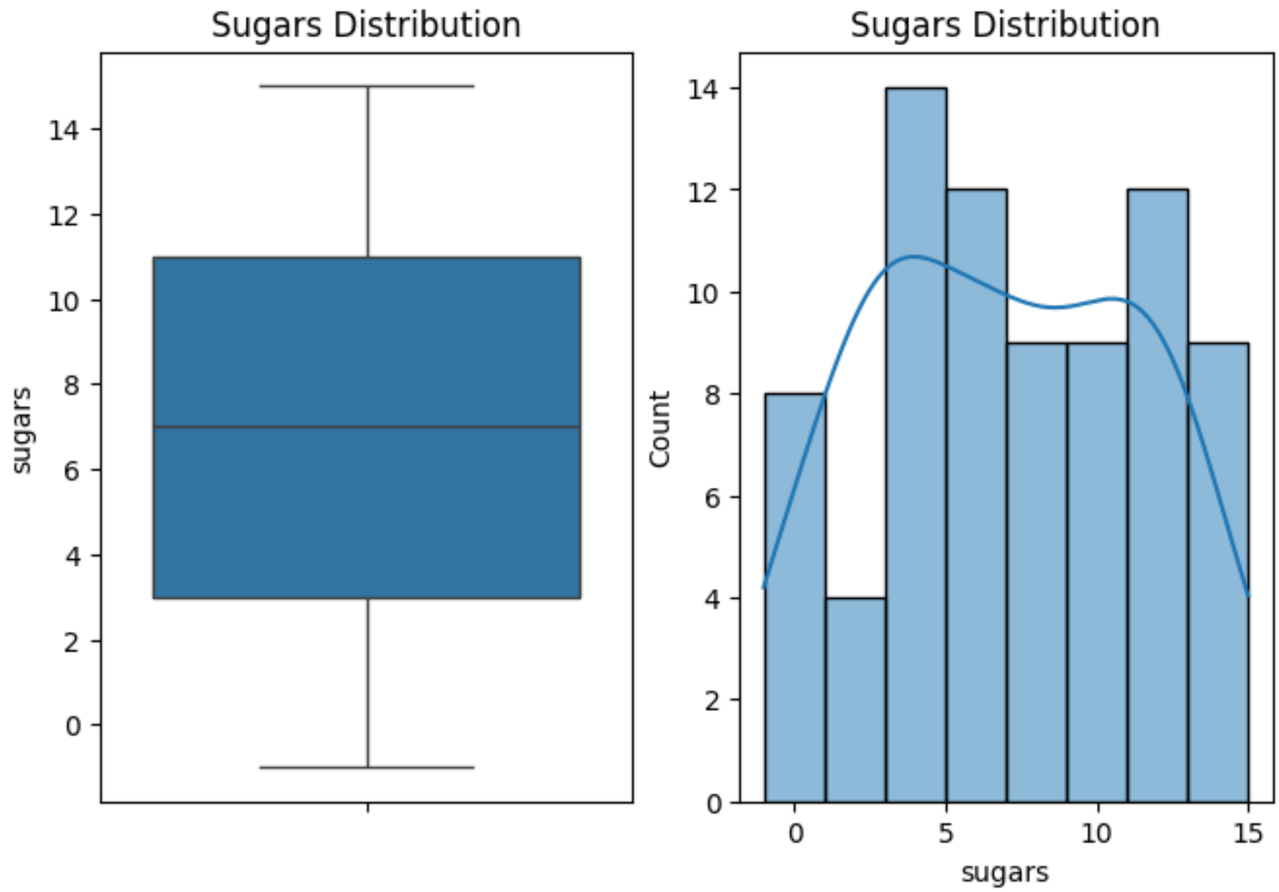

Fat Distribution
fat
0
1
2
3
4
5
Fat Distribution
Count
0
5
10
15
20
25
30
fat

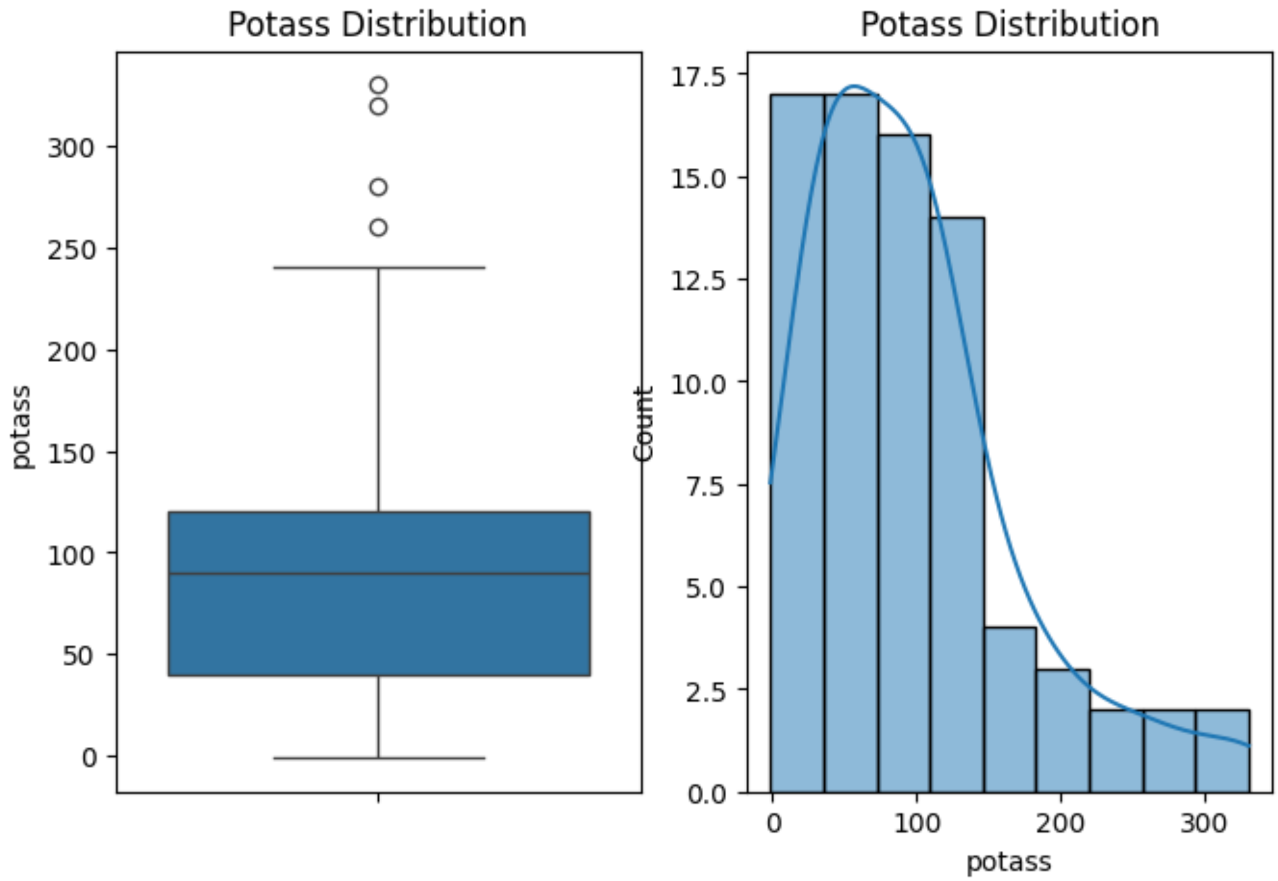

Sodium Distribution
sodium
300
250
200
150
100
50
0
Sodium Distribution
Count
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
100
200
300
sodium

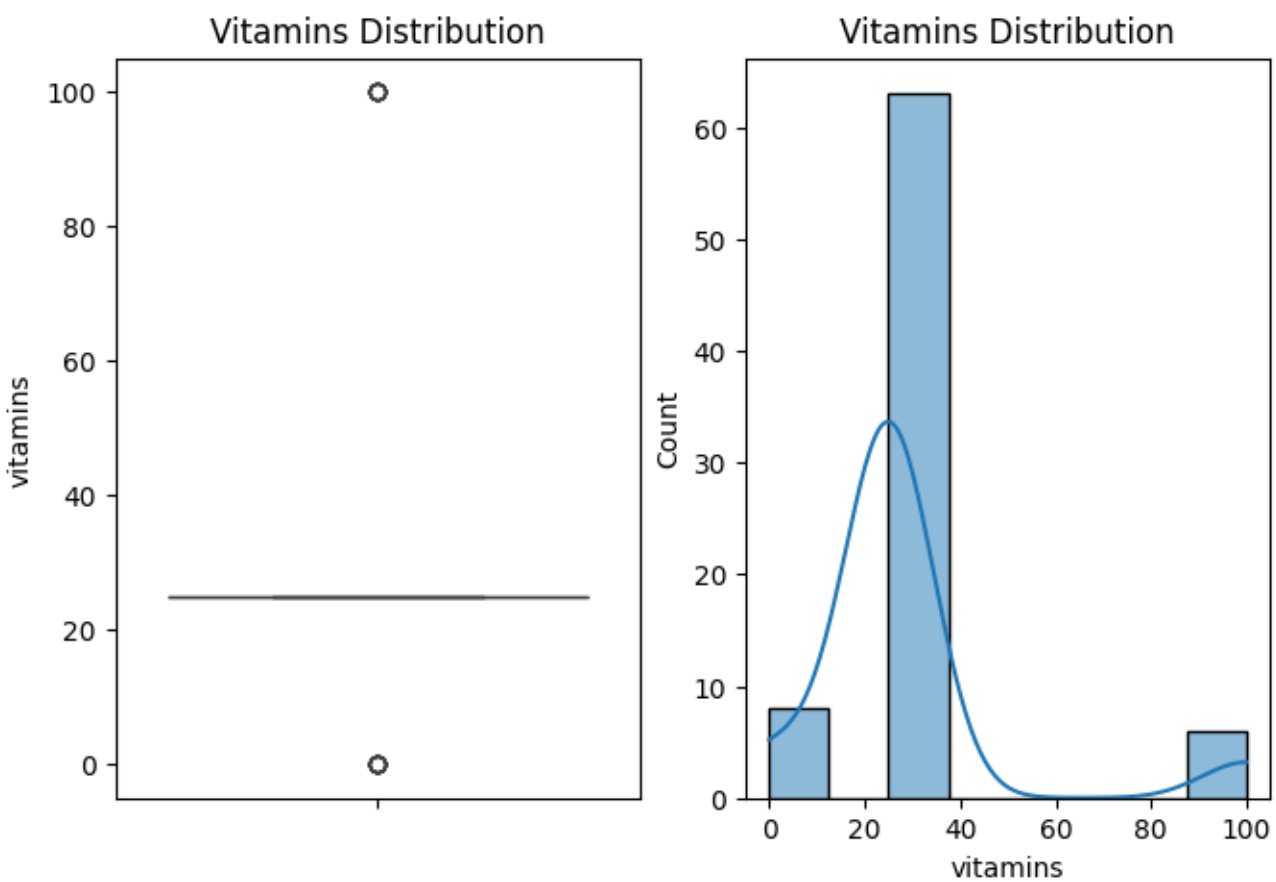

Fiber Distribution
fiber
14
12
10
8
6
4
2
0

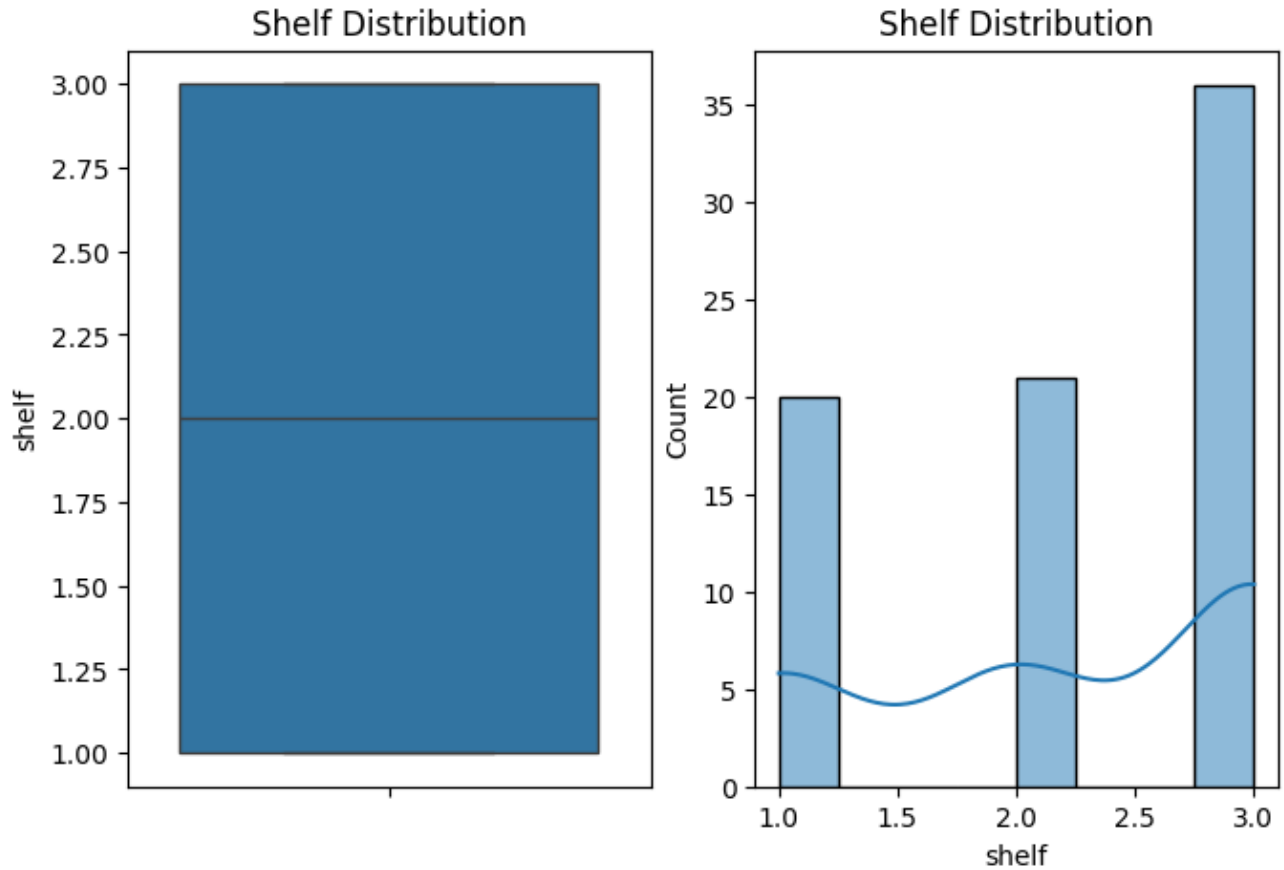

Fiber Distribution
Count
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
5
10
fiber

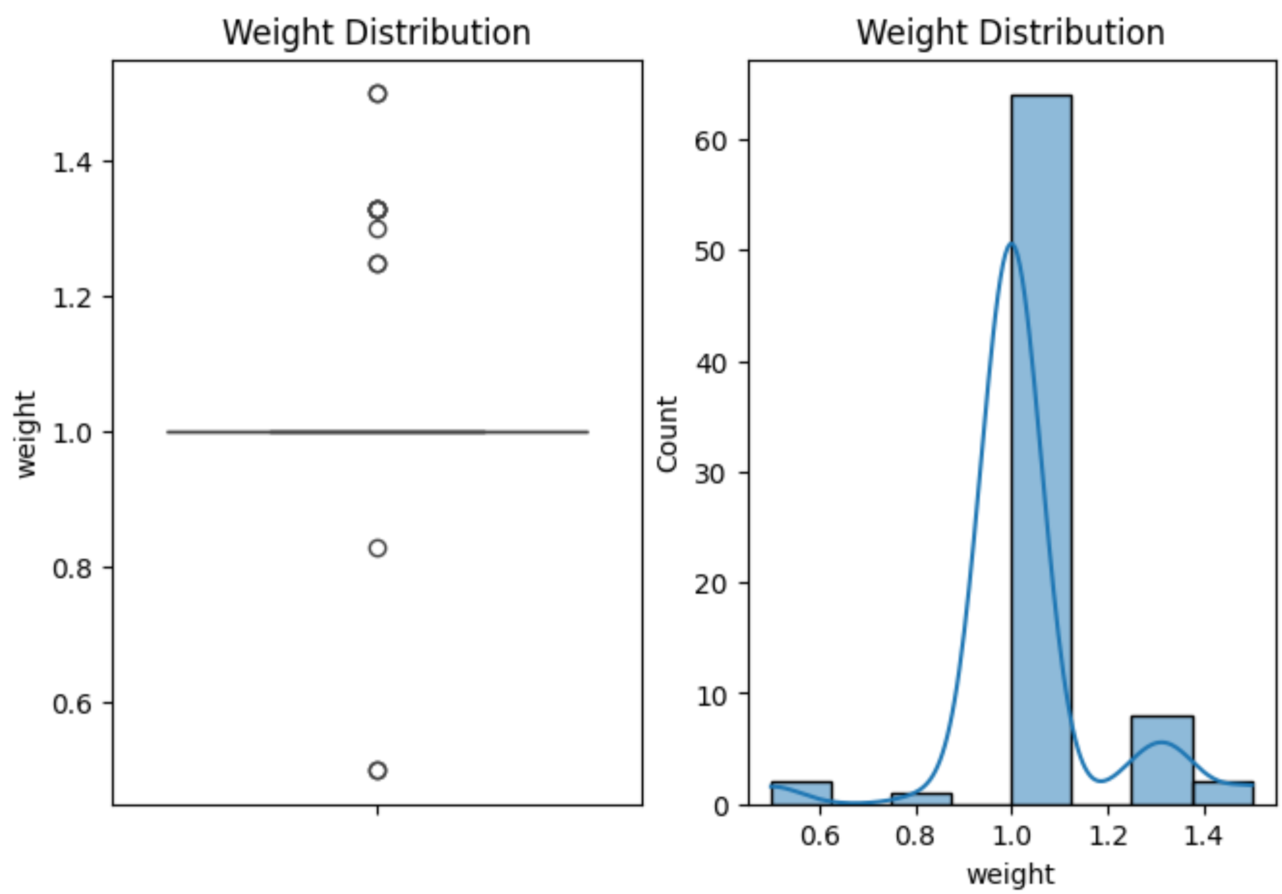

Carbo Distribution
carbo
20
15
10
5
0
Carbo Distribution
Count
16
14
12
10
8
6
4
2
0
0
5
10
15
20
carbo

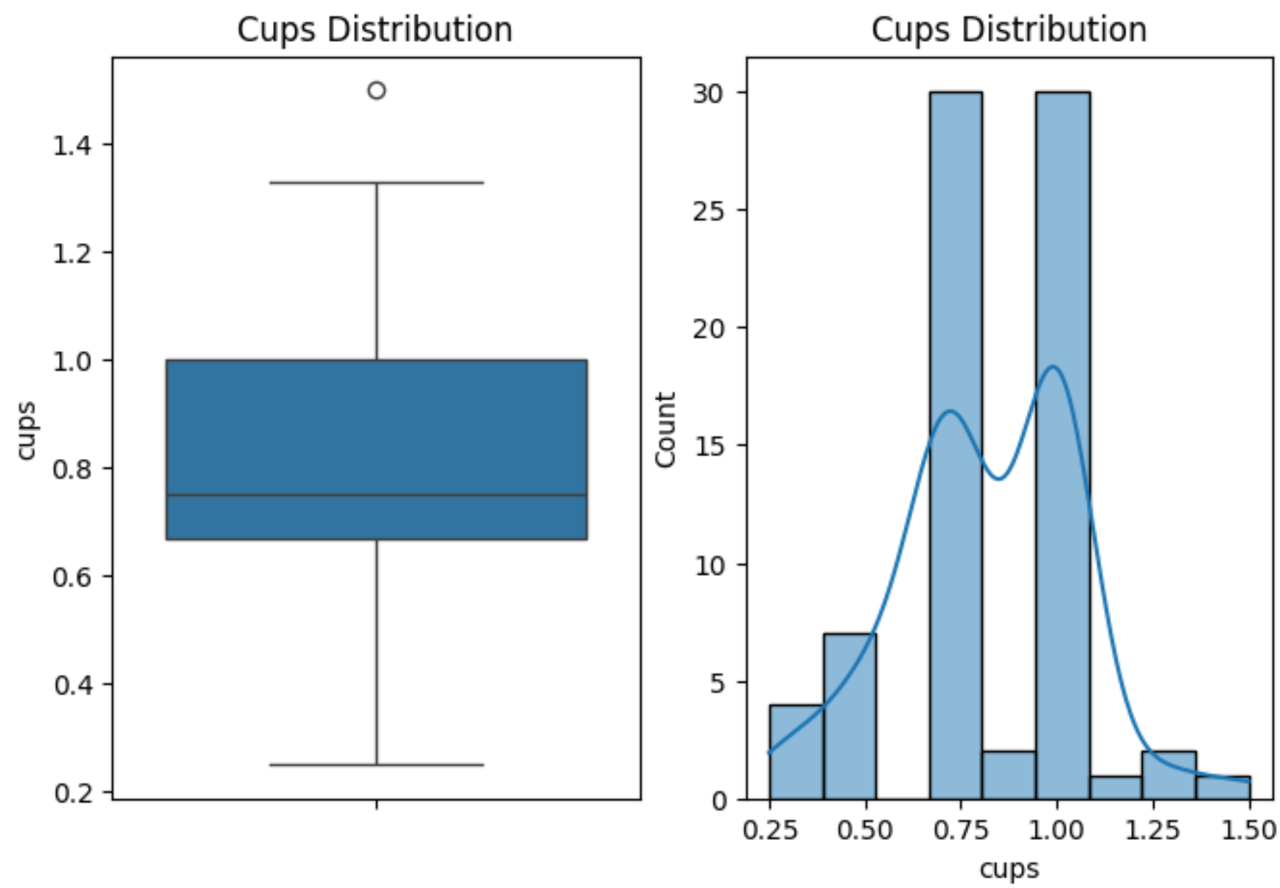

Sugars Distribution
sugars
14
12
10
8
6
4
2
0
Sugars Distribution
Count
14
12
10
8
6
4
2
0
0
5
10
15
sugars

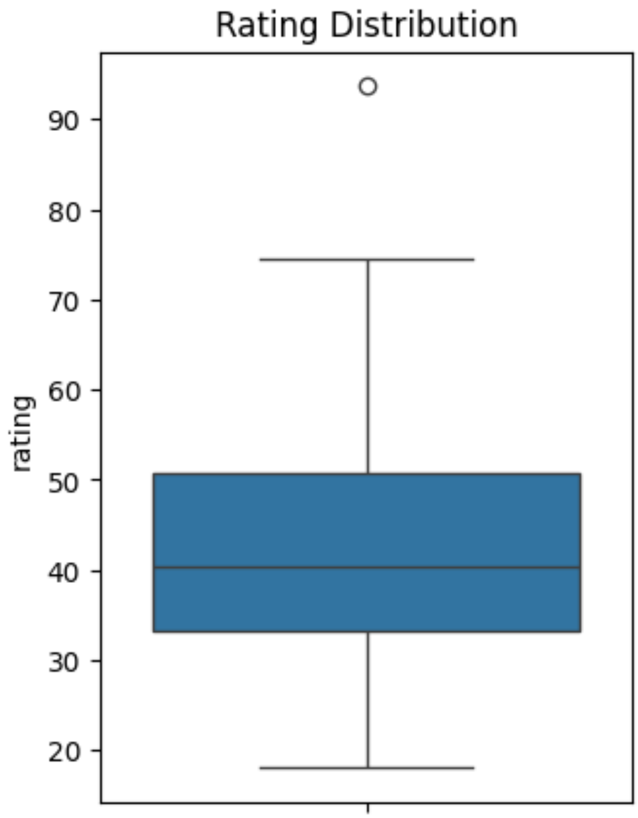

Potass Distribution
potass
0
50
100
150
200
250
300
Potass Distribution
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
100
200
300
potass

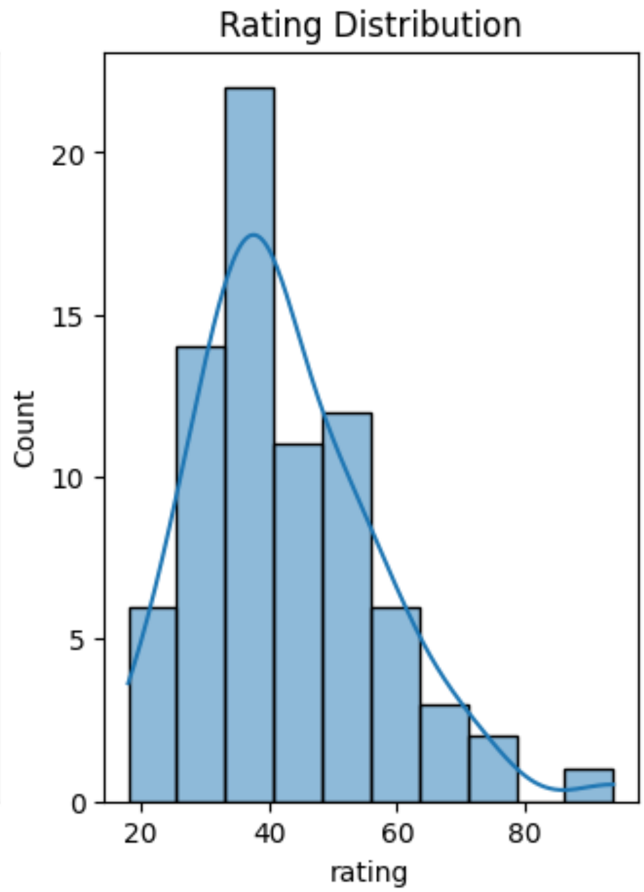

Vitamins Distribution
vitamins
0
20
40
60
80
100
Vitamins Distribution
Count
0
10
20
30
40
50
60
0
20
40
60
80
100
vitamins


Shelf Distribution
shelf
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
Shelf Distribution
Count
35
30
25
20
15
10
5
0
1.0
1.5
2.0
2.5
3.0
shelf


Weight Distribution
weight
1.4
1.2
1.0
0.8
0.6
Weight Distribution
Count
60
50
40
30
20
10
0
0.6
0.8
1.0
1.2
1.4
weight


Cups Distribution
1.4
1.2
1.0
0.8
0.6
0.4
0.2
cups
Cups Distribution
30
25
20
15
10
5
0
Count
0.25
0.50
0.75
1.00
1.25
1.50
cups





None of the variables are normally distributed and few contains outliers like Fiber, Potass, and Calories.

```
In [ ]: X = cereal_df[numeric_columns].drop(columns='rating')
y = cereal_df['rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

X_train_scaled, X_test_scaled = scaler.fit_transform(X_train), scaler.transf


X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns, ir
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X_test.columns, index

model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print(f'The predicted values are: \n{y_pred}\n')


print(f'Weight of each variable: \n{model.coef_}')
```

```
print(f'\nThe model accuracy is: {metrics.r2_score(y_test, y_pred)}')
```

```
The predicted values are:
[34.3848432  21.87129222 18.04285079 68.40297282 34.13976434 40.10596496
 31.23005449 41.50353998 59.64283667 41.01549176 59.36399352 49.78744507
 22.39651289 19.82357266 39.25919732 53.37100719]

Weight of each variable:
[-4.41842378e+00  3.62344489e+00 -1.69118070e+00 -4.66060398e+00
  8.01370306e+00  4.82065764e+00 -3.21615749e+00 -2.35051876e+00
 -1.18431757e+00 -4.79257220e-08  1.59785011e-08  1.98559348e-08]

The model accuracy is: 0.9999999999999996
```

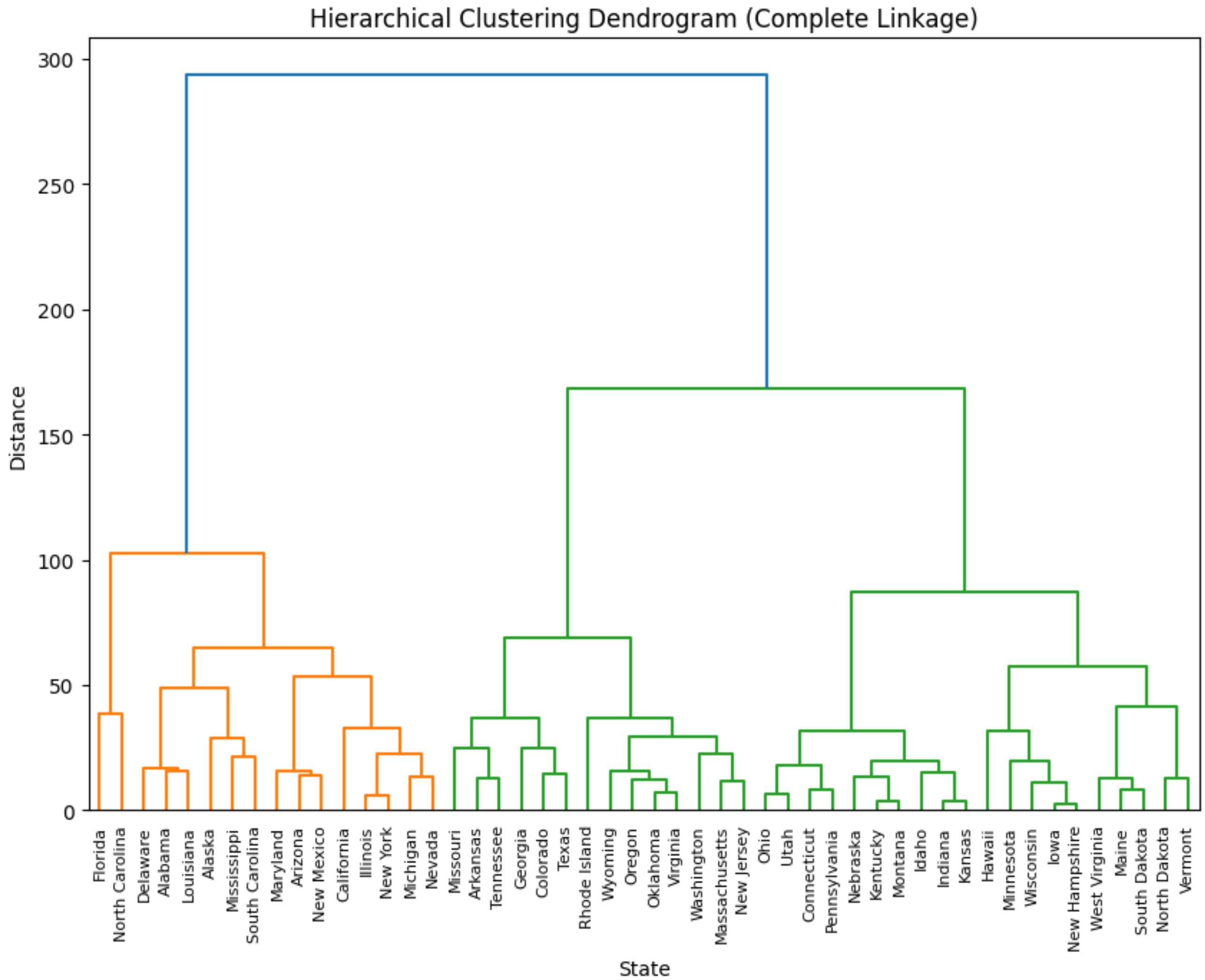
Looking at the weight of each variable it's clear to see that Protein, Fiber, Carbo are the top 3 predictors of Rating.

The model accuracy is 0.99999

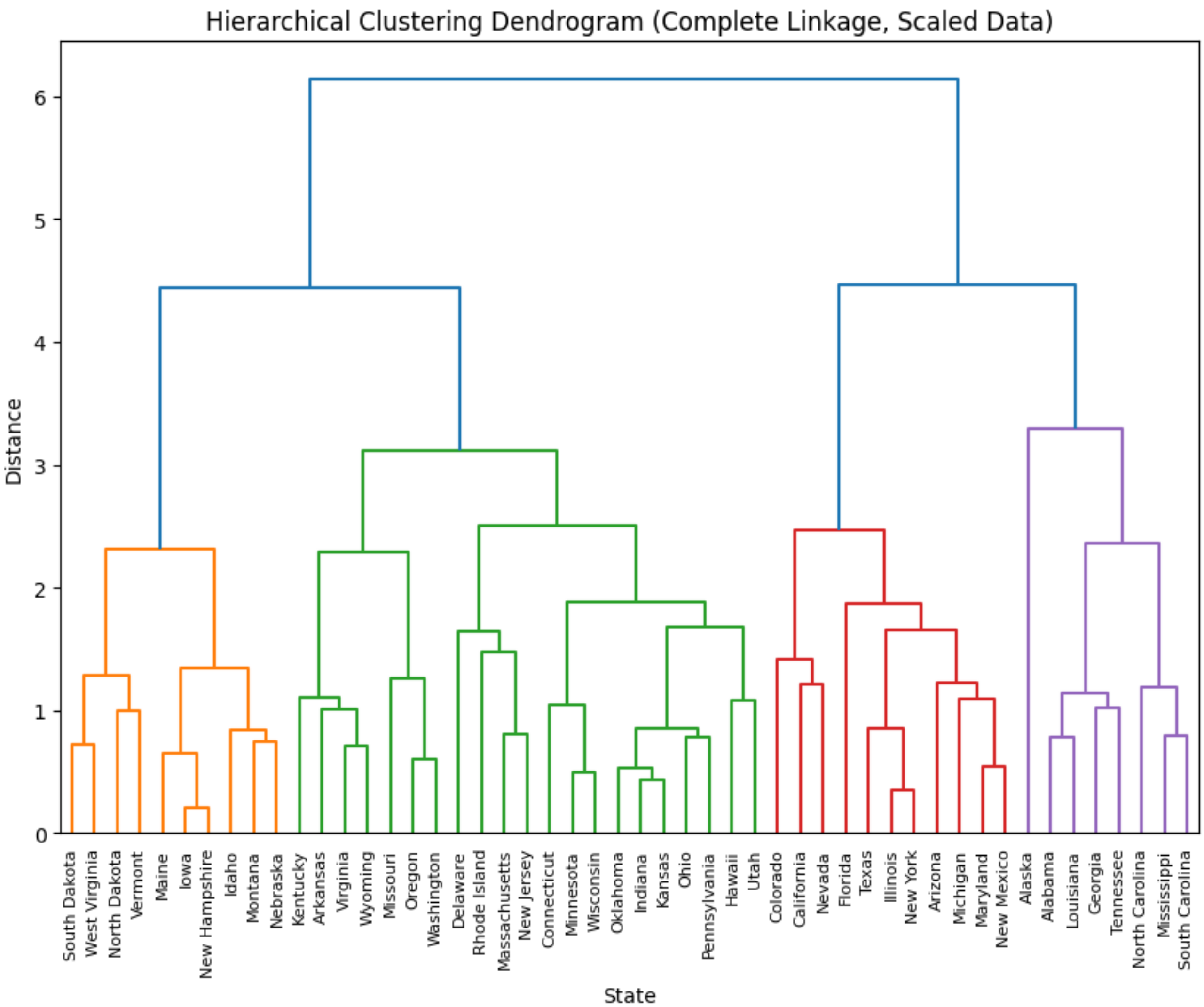
5.

In [ ]:
```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from rdatasets import data

us_arrests = data('USArrests')


us_arrests.set_index('rownames', inplace=True)

# 5(a) Perform hierarchical clustering using complete linkage and Euclidean
linked = linkage(us_arrests, method='complete', metric='euclidean')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, labels=us_arrests.index.values)
plt.title('Hierarchical Clustering Dendrogram (Complete Linkage)')
plt.xlabel('State')
plt.ylabel('Distance')
plt.show()

# 5(b) Cut the dendrogram to obtain 3 clusters
clusters = fcluster(linked, t=3, criterion='maxclust')

# Assign clusters to states
clustered_states = pd.DataFrame({'State': us_arrests.index, 'Cluster': clust
print(clustered_states.groupby('Cluster')['State'].apply(list))

# 5(c) Hierarchical clustering after scaling variables
scaler = StandardScaler()
```

```
us_arrests_scaled = scaler.fit_transform(us_arrests)
linked_scaled = linkage(us_arrests_scaled, method='complete', metric='euclic

# Plot the dendrogram for scaled data
plt.figure(figsize=(10, 7))
dendrogram(linked_scaled, labels=us_arrests.index.values)
plt.title('Hierarchical Clustering Dendrogram (Complete Linkage, Scaled Data
plt.xlabel('State')
plt.ylabel('Distance')
plt.show()
```




```
Cluster
1    [Alabama, Alaska, Arizona, California, Delawar...
2    [Arkansas, Colorado, Georgia, Massachusetts, M...
3    [Connecticut, Hawaii, Idaho, Indiana, Iowa, Ka...
Name: State, dtype: object
```



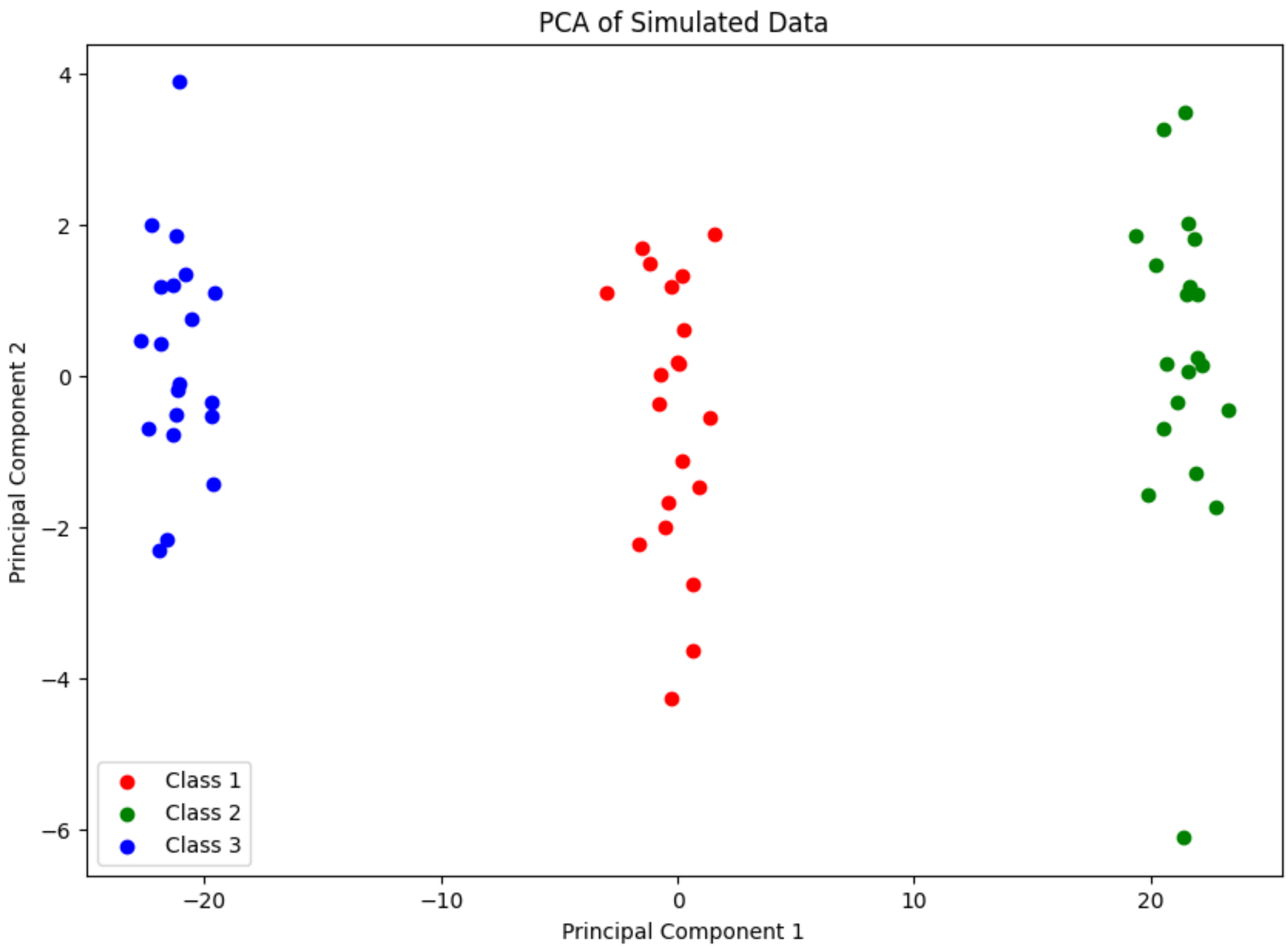
Scaling the variables standardizes them to have a mean of 0 and a standard deviation of 1. This affects the clustering results by ensuring that each variable contributes equally to the distance calculations, which is especially important when the variables are on different scales.

6.

In [ ]:
```
from sklearn.cluster import KMeans

# 6(a) Generate a simulated dataset
np.random.seed(123)
class1 = np.random.normal(0, 1, (20, 50))
class2 = np.random.normal(3, 1, (20, 50))
class3 = np.random.normal(-3, 1, (20, 50))

simulated_data = np.vstack((class1, class2, class3))
```

```
# 6(b) Perform PCA and plot the first two principal component score vectors
pca = PCA(n_components=2)
pca_result = pca.fit_transform(simulated_data)

plt.figure(figsize=(10, 7))
plt.scatter(pca_result[:20, 0], pca_result[:20, 1], color='r', label='Class
plt.scatter(pca_result[20:40, 0], pca_result[20:40, 1], color='g', label='Cl
plt.scatter(pca_result[40:, 0], pca_result[40:, 1], color='b', label='Class
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.title('PCA of Simulated Data')
plt.show()

# 6(c) Perform K-means clustering with K=3 and compare to true class labels
kmeans_3 = KMeans(n_clusters=3, random_state=123)
kmeans_3.fit(simulated_data)
labels_3 = kmeans_3.labels_

# Compare K-means clusters with true labels
from sklearn.metrics import confusion_matrix
true_labels = np.array([0]*20 + [1]*20 + [2]*20)
conf_matrix = confusion_matrix(true_labels, labels_3)
print(f'Confusion matrix: \n{conf_matrix}')

# 6(d) Perform K-means clustering with K=2
kmeans_2 = KMeans(n_clusters=2, random_state=123)
kmeans_2.fit(simulated_data)
labels_2 = kmeans_2.labels_

conf_matrix_2 = confusion_matrix(true_labels, labels_2)
print(f'Confusion Matrix with k=2 and tand the model do not quite get class

# 6(e) Perform K-means clustering with K=4
kmeans_4 = KMeans(n_clusters=4, random_state=123)
kmeans_4.fit(simulated_data)
labels_4 = kmeans_4.labels_

conf_matrix_4 = confusion_matrix(true_labels, labels_4)
print(f'Confusion Matrix with k=4 and the model is more confused on the clas

# 6(f) Perform K-means clustering with K=3 on the first two principal compor
kmeans_pca = KMeans(n_clusters=3, random_state=123)
kmeans_pca.fit(pca_result)
labels_pca = kmeans_pca.labels_

conf_matrix_pca = confusion_matrix(true_labels, labels_pca)
print(f'Confusion Matrix with k=3 before scalling the model seem to get hanc
```

```
# 6(g) Perform K-means clustering with K=3 after scaling the variables
scaler = StandardScaler()
scaled_simulated_data = scaler.fit_transform(simulated_data)
kmeans_scaled = KMeans(n_clusters=3, random_state=123)
kmeans_scaled.fit(scaled_simulated_data)
labels_scaled = kmeans_scaled.labels_

conf_matrix_scaled = confusion_matrix(true_labels, labels_scaled)
print(f'Confusion matrix after scalling the model seem to get handling on th
```



```
Confusion matrix:
[[ 0  0 20]
 [ 0 20  0]
 [20  0  0]]
Confusion Matrix with k=2 and tand the model do not quite get class 3
[[20  0  0]
 [ 0 20  0]
 [20  0  0]]
Confusion Matrix with k=4 and the model is more confused on the classes
[[ 0  0 20  0]
 [ 0 20  0  0]
 [ 4  0  0 16]
 [ 0  0  0  0]]
Confusion Matrix with k=3 before scalling the model seem to get handling on
the classes although weird looking at it positively diagonal. This should be
negatively diagonal rather
[[ 0  0 20]
 [ 0 20  0]
 [20  0  0]]
Confusion matrix after scalling the model seem to get handling on the classe
s although weird looking at it positively diagonal. This should be negativel
y diagonal rather
[[ 0  0 20]
 [ 0 20  0]
 [20  0  0]]
```

7.

In [ ]:

```
# Load the Carseats dataset
carseats = data('ISLR','Carseats')

carseats = pd.get_dummies(carseats, columns=['ShelveLoc', 'Urban', 'US'], dr

X = carseats.drop('Sales', axis=1)
y = carseats['Sales']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

rf_model = RandomForestRegressor()

rf_model.fit(X_train_scaled, y_train)

y_pred = rf_model.predict(X_test_scaled)

print(f'MSE: {metrics.mean_squared_error(y_test, y_pred)}\n')
importances = pd.DataFrame({'Features': X.columns,
```

```
                            'Importance': rf_model.feature_importances_}).sc
                            ascending=False).reset_index(drop=True)
sns.barplot(x='Importance', y='Features', data=importances)
plt.title('Feature Importance')
plt.show()
```

```
MSE: 3.1920768261249988
```