In [ ]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn import metrics, tree
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from rdatasets import data
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
import scipy.cluster.hierarchy as shc
import scipy.stats as stats
```

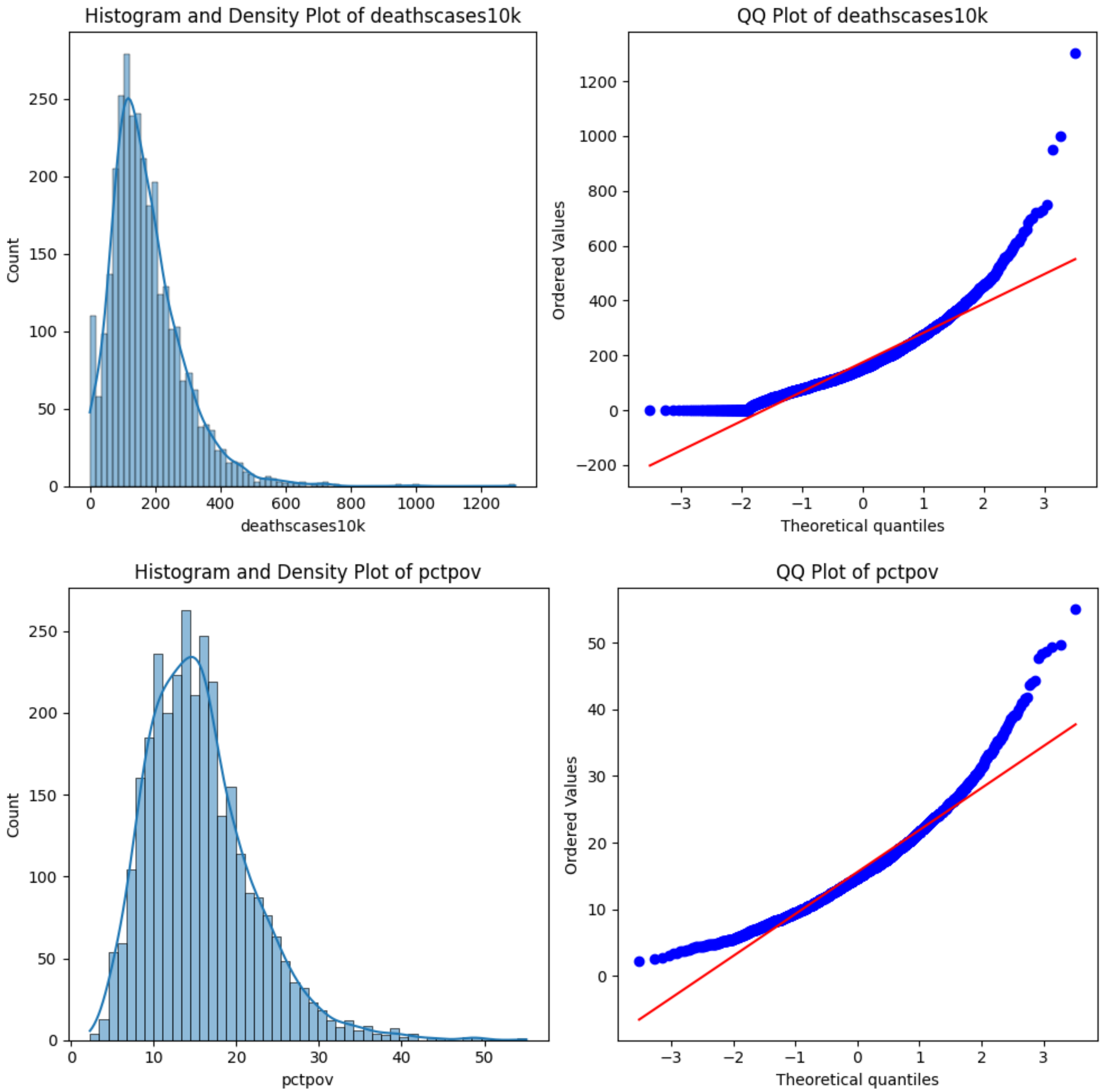
In [ ]:
```
data = pd.read_csv('HW4_Q1.csv')
data.head()
```

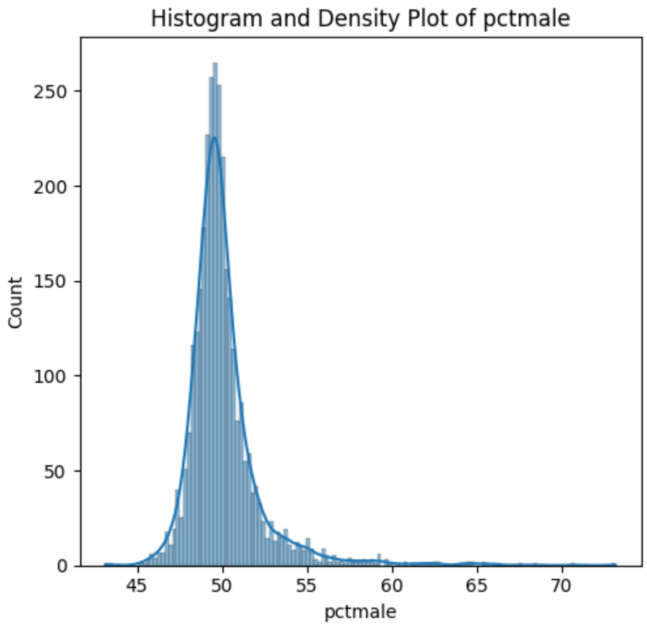
Out[ ]:

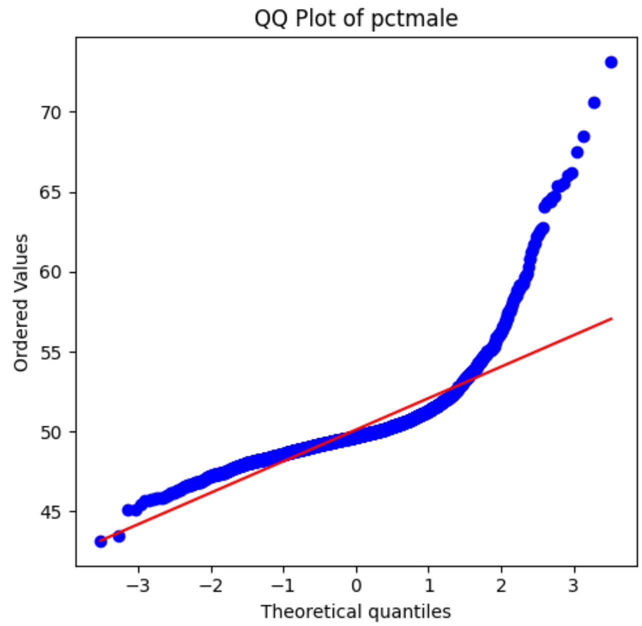
	deathscases10k	pctpov	pctmale	pctwhite	pctblack	pctasian	pcthispanic
0	114.558473	15.4	48.551285	74.308014	19.343178	1.205014	2.965774
1	118.373649	10.6	48.461623	83.111340	8.783976	1.134289	4.646779
2	211.360634	28.9	52.783248	45.641252	48.032635	0.454162	4.276355
3	250.817884	14.0	53.218750	74.589288	21.120536	0.232143	2.625000
4	135.746606	14.4	49.273859	86.886239	1.462656	0.278354	9.571231

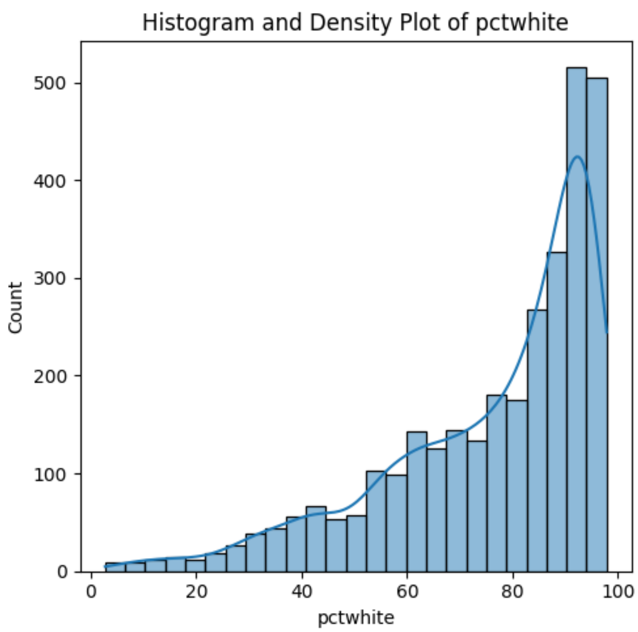
Question 1

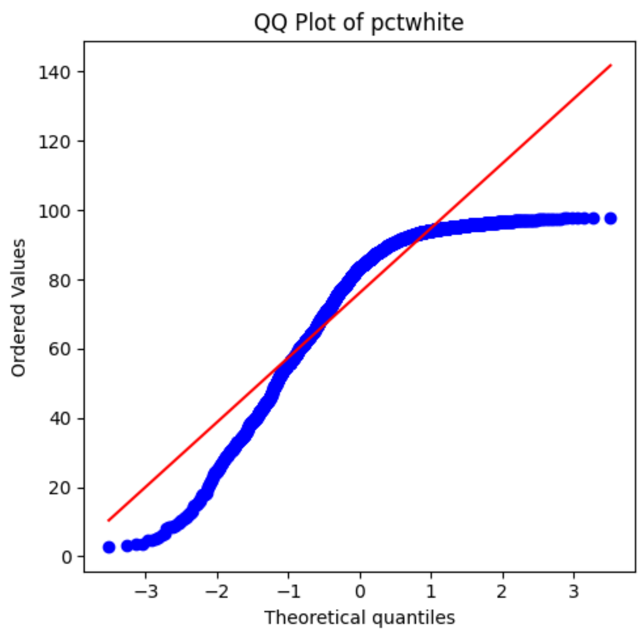
In [ ]:
```
# Create histograms, density plots, and QQ plots
columns = data.columns
for column in columns:
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    sns.histplot(data[column], kde=True)
    plt.title(f'Histogram and Density Plot of {column}')

    plt.subplot(1, 3, 2)
    stats.probplot(data[column], dist="norm", plot=plt)
    plt.title(f'QQ Plot of {column}')

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

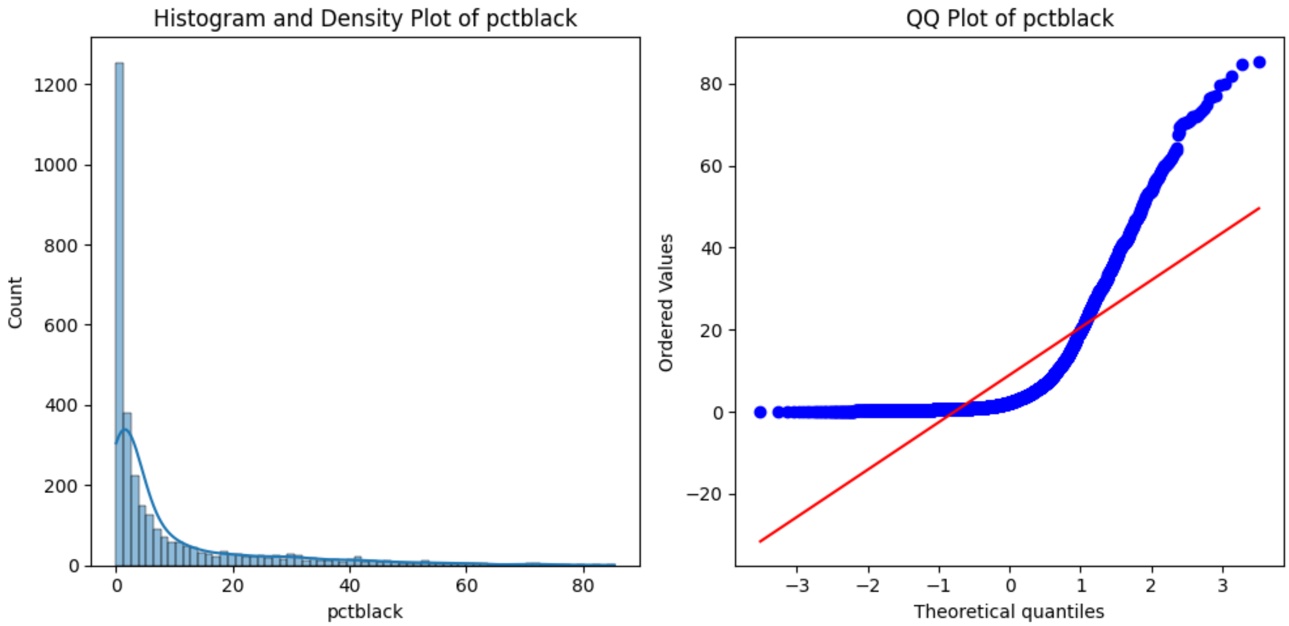

Histogram and Density Plot of deathscases10k
Count
deathscases10k
0 50 100 150 200 250
0 200 400 600 800 1000 1200
QQ Plot of deathscases10k
Ordered Values
Theoretical quantiles
−200 0 200 400 600 800 1000 1200
−3 −2 −1 0 1 2 3
Histogram and Density Plot of pctpov
Count
pctpov
0 50 100 150 200 250
0 10 20 30 40 50
QQ Plot of pctpov
Ordered Values
Theoretical quantiles
0 10 20 30 40 50
−3 −2 −1 0 1 2 3

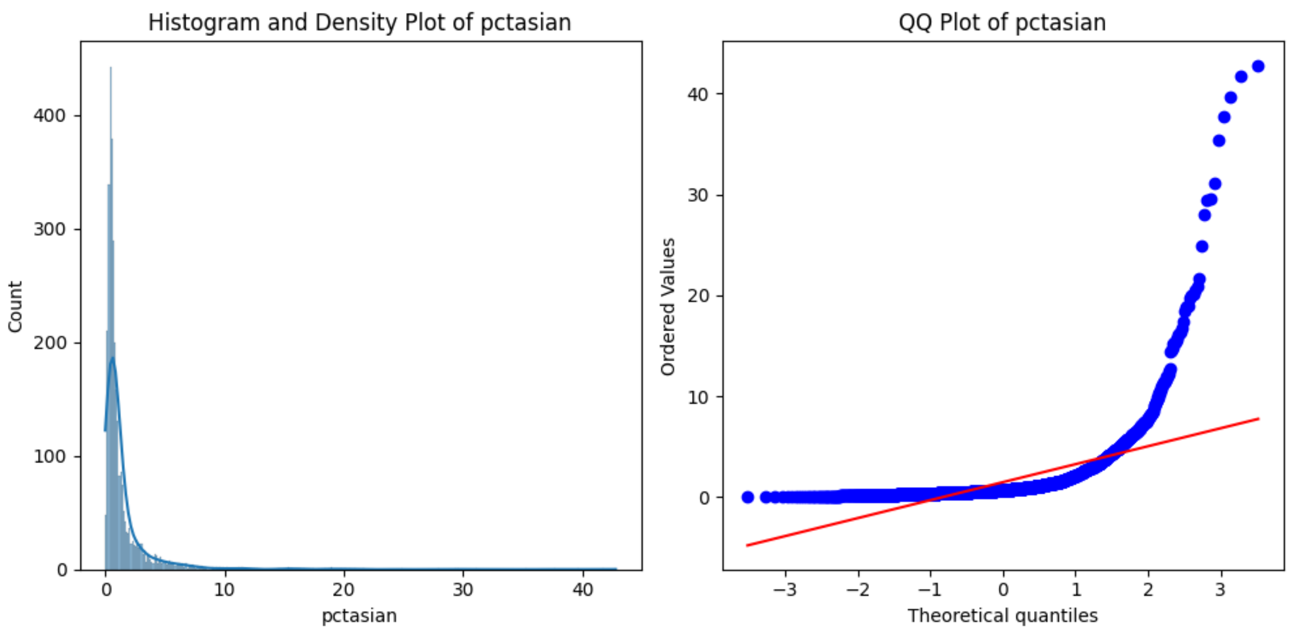

Histogram and Density Plot of pctmale
Count
pctmale

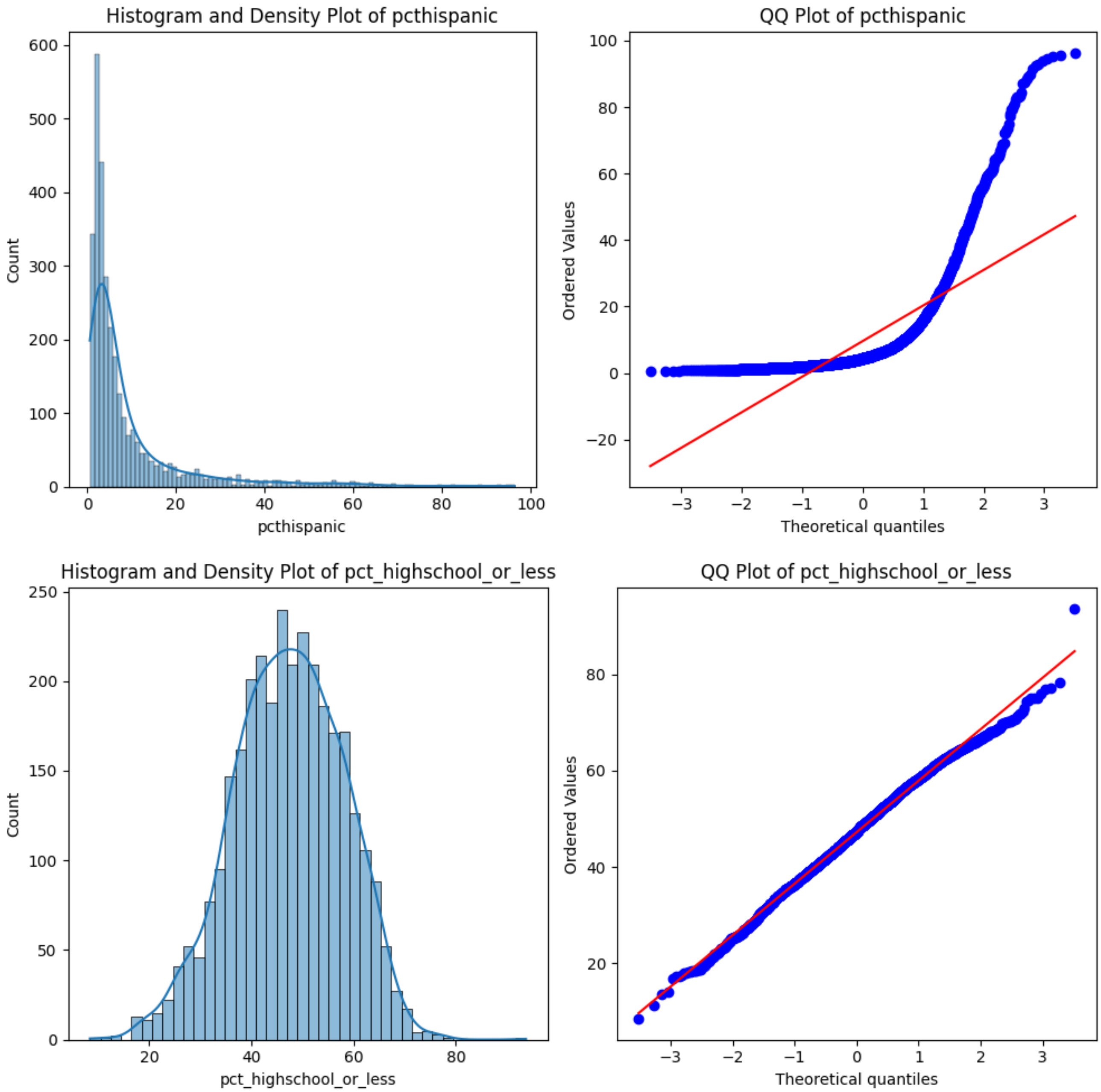

QQ Plot of pctmale
Ordered Values
Theoretical quantiles

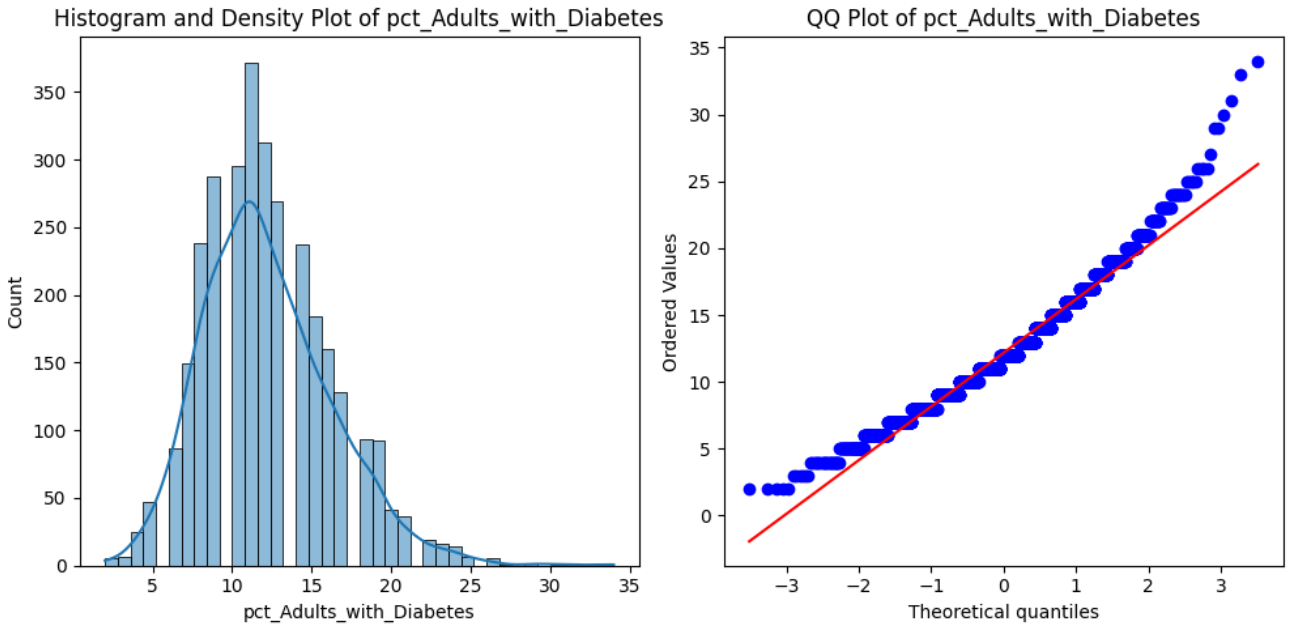

Histogram and Density Plot of pctwhite
Count
pctwhite

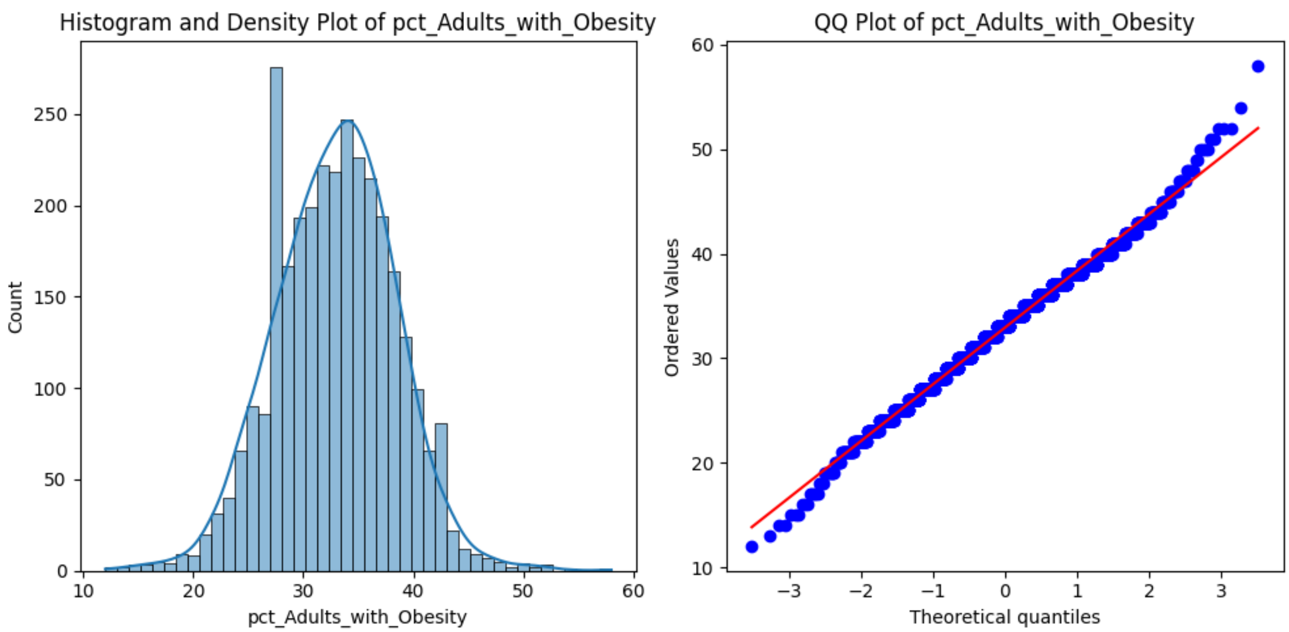

QQ Plot of pctwhite
Ordered Values
Theoretical quantiles


Histogram and Density Plot of pctblack
Count
pctblack
QQ Plot of pctblack
Ordered Values
Theoretical quantiles


Histogram and Density Plot of pctasian
Count
pctasian
QQ Plot of pctasian
Ordered Values
Theoretical quantiles


Histogram and Density Plot of pcthispanic
Count
pcthispanic
QQ Plot of pcthispanic
Ordered Values
Theoretical quantiles
Histogram and Density Plot of pct_highschool_or_less
Count
pct_highschool_or_less
QQ Plot of pct_highschool_or_less
Ordered Values
Theoretical quantiles





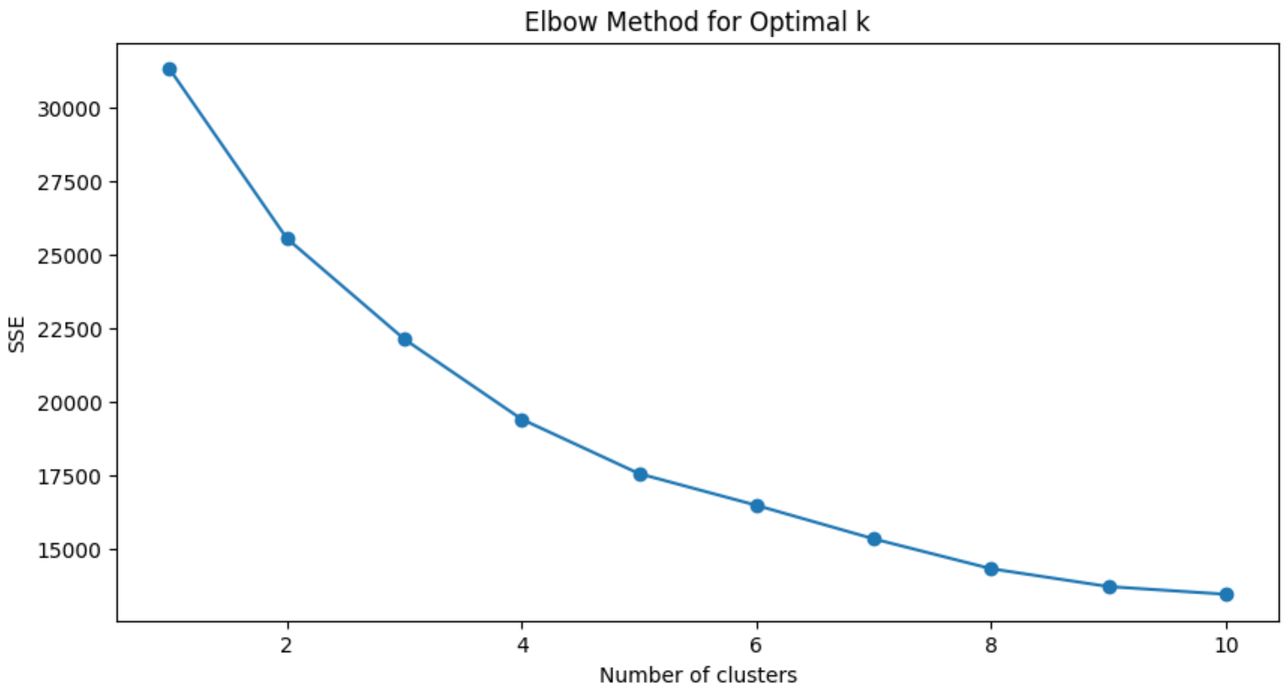
In [ ]:
```python
# Conduct Shapiro-Wilk test
from scipy.stats import shapiro

for column in columns:
    stat, p = shapiro(data[column])
    print(f'Shapiro-Wilk Test for {column}: Statistics={stat}, p-value={p}')
    if p > 0.05:
        print(f'{column} looks Gaussian (fail to reject H0)')
    else:
        print(f'{column} does not look Gaussian (reject H0)')
```

```
Shapiro-Wilk Test for deathscases10k: Statistics=0.9031476625819101, p-value
=1.0111720412623282e-40
deathscases10k does not look Gaussian (reject H0)
Shapiro-Wilk Test for pctpov: Statistics=0.9447959478964187, p-value=9.85936
221171364e-33
pctpov does not look Gaussian (reject H0)
Shapiro-Wilk Test for pctmale: Statistics=0.7481257019505474, p-value=2.1351
01179508851e-56
pctmale does not look Gaussian (reject H0)
Shapiro-Wilk Test for pctwhite: Statistics=0.8654726274449382, p-value=8.187
341655280106e-46
pctwhite does not look Gaussian (reject H0)
Shapiro-Wilk Test for pctblack: Statistics=0.6501496623094074, p-value=2.142
6347573793e-62
pctblack does not look Gaussian (reject H0)
Shapiro-Wilk Test for pctasian: Statistics=0.3967816751148091, p-value=4.988
365161483156e-73
pctasian does not look Gaussian (reject H0)
Shapiro-Wilk Test for pcthispanic: Statistics=0.6027059999285516, p-value=8.
424872373596017e-65
pcthispanic does not look Gaussian (reject H0)
Shapiro-Wilk Test for pct_highschool_or_less: Statistics=0.9966299159425304,
p-value=1.8165442978186677e-06
pct_highschool_or_less does not look Gaussian (reject H0)
Shapiro-Wilk Test for pct_Adults_with_Diabetes: Statistics=0.971680233137539
6, p-value=1.9848587260382606e-24
pct_Adults_with_Diabetes does not look Gaussian (reject H0)
Shapiro-Wilk Test for pct_Adults_with_Obesity: Statistics=0.994359159013186
4, p-value=1.2420060903838038e-09
pct_Adults_with_Obesity does not look Gaussian (reject H0)
```

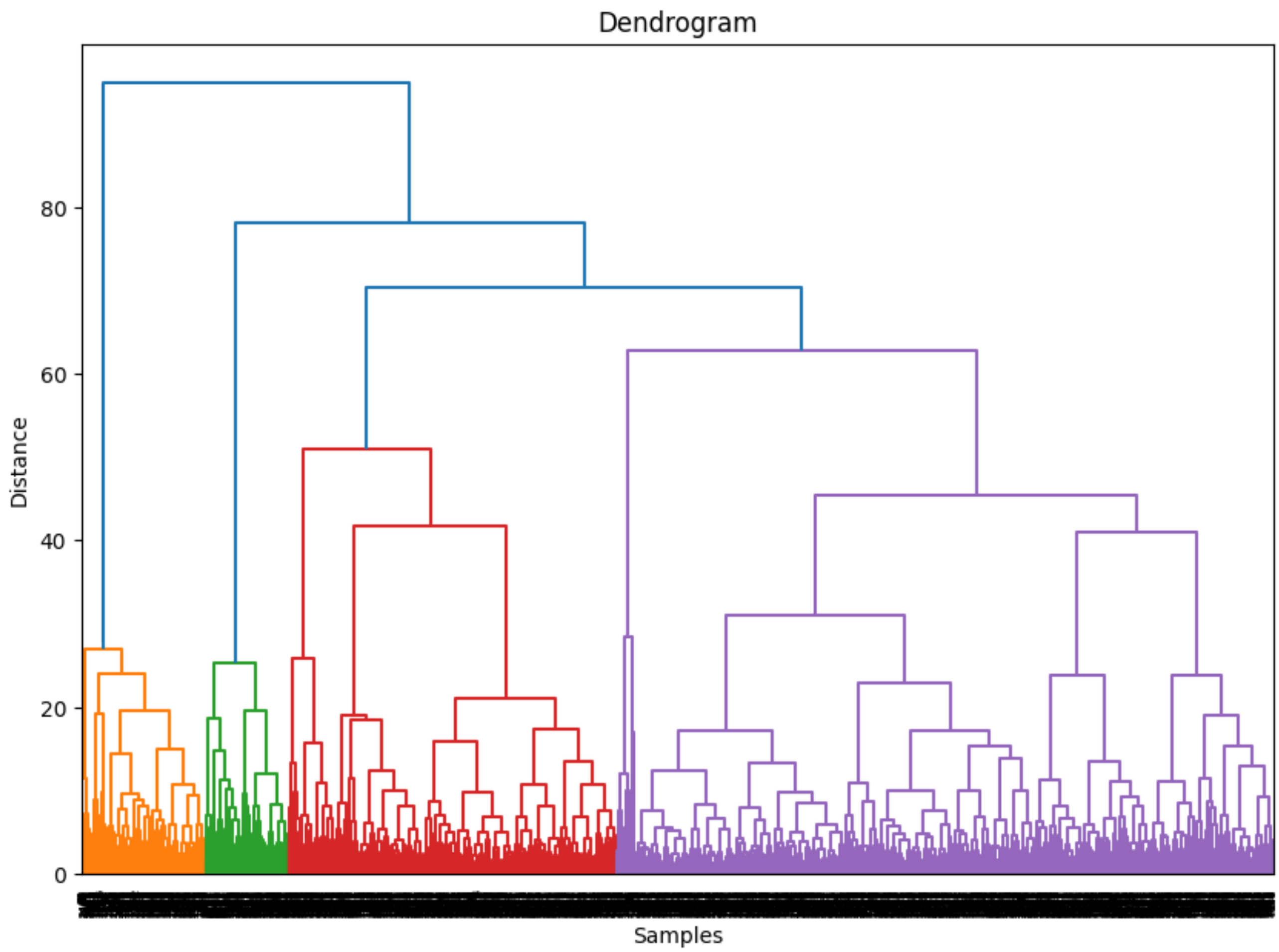
In [ ]:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data[columns])

# Determine the number of clusters using the Elbow method
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(data_scaled)
    sse.append(kmeans.inertia_)

# Plot SSE for each k
plt.figure(figsize=(10, 5))
plt.plot(range(1, 11), sse, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
```

```
plt.title('Elbow Method for Optimal k')
plt.show()

# Fit K-Means with the chosen number of clusters (let's assume k=3 for this
kmeans = KMeans(n_clusters=3, random_state=0)
clusters = kmeans.fit_predict(data_scaled)

# Add cluster labels to the original data
data['KMeans_Cluster'] = clusters
```



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

# Perform hierarchical clustering
linked = linkage(data_scaled, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title('Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()

# Decide the number of clusters (let's assume we choose 3 clusters)
hier_clusters = fcluster(linked, 3, criterion='maxclust')

# Add cluster labels to the original data
data['Hierarchical_Cluster'] = hier_clusters
```

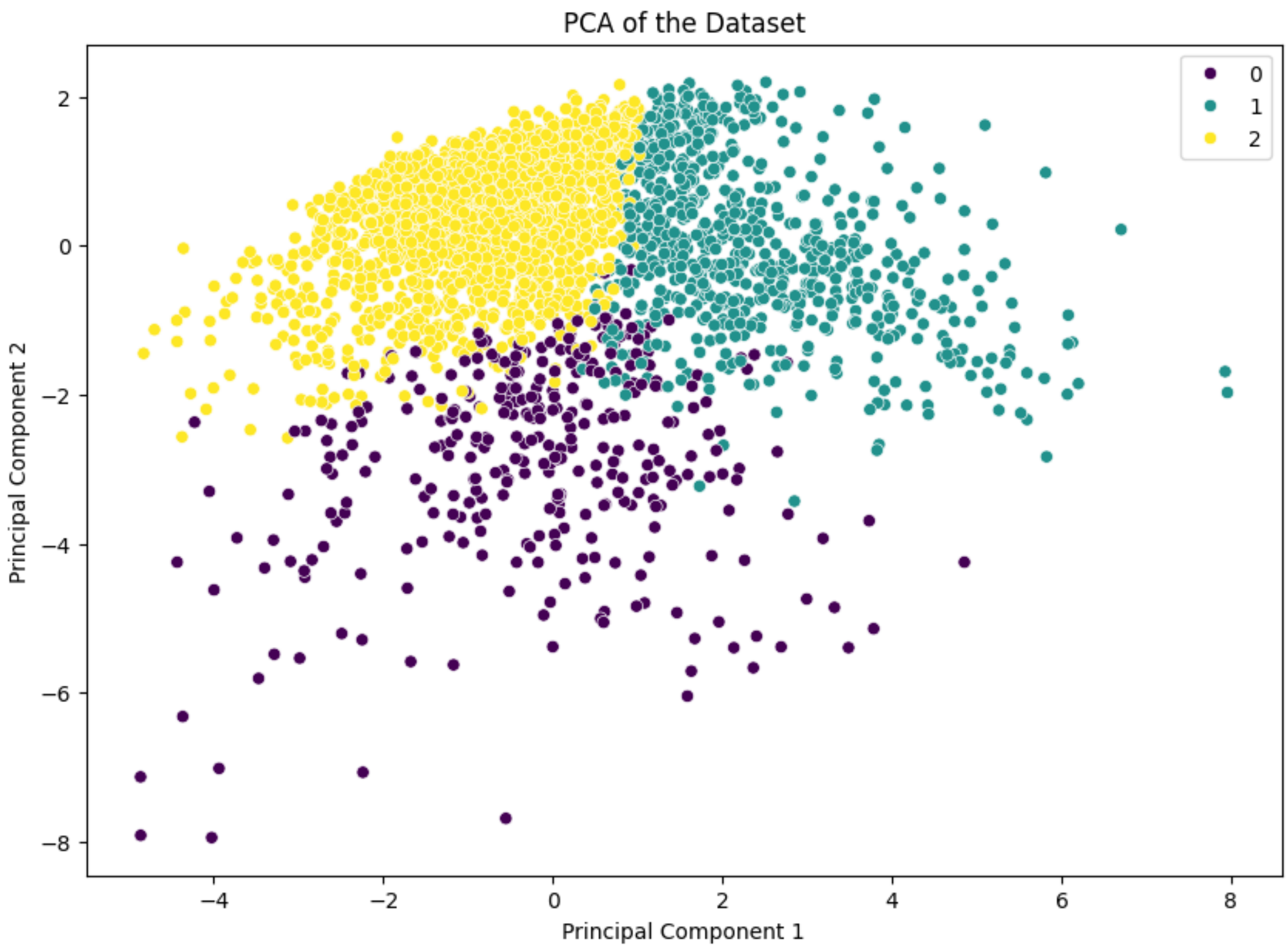


In [ ]:
```
from sklearn.metrics import adjusted_rand_score

# Compute ARI
ari = adjusted_rand_score(data['KMeans_Cluster'], data['Hierarchical_Cluster
print(f'Adjusted Rand Index (ARI) between K-Means and Hierarchical Clusterin
```

```
Adjusted Rand Index (ARI) between K-Means and Hierarchical Clustering: 0.454
39567003114595
```

In [ ]:
```
from sklearn.decomposition import PCA

# Fit PCA
pca = PCA(n_components=2)
pca_components = pca.fit_transform(data_scaled)

# Add PCA components to the original data
data['PCA1'] = pca_components[:, 0]
data['PCA2'] = pca_components[:, 1]

# Plot PCA components
plt.figure(figsize=(10, 7))
sns.scatterplot(x='PCA1', y='PCA2', hue='KMeans_Cluster', data=data, palette
plt.title('PCA of the Dataset')
```

```
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



Question 3

In [ ]:
```
# Define the dataset
data = {
    'Restaurant': ['R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R1
    'Type': ['Fast Food', 'Ethnic', 'Casual Dining', 'Casual Dining', 'Casua
    'Price': ['$', '$$', '$$', '$$$', '$', '$$', '$$', '$$', '$$$', '$$', '$
    'Neighborhood': ['Oakland', 'Squirrel Hill', 'Squirrel Hill', 'Shadyside
    'Restriction': ['Vegetarian', 'Gluten Free', 'None', 'Vegetarian', 'Vege
    'OK': [0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1]
}

df = pd.DataFrame(data)
df
```

Out[ ]:

	Restaurant	Type	Price	Neighborhood	Restriction	OK
0	R1	Fast Food	$	Oakland	Vegetarian	0
1	R2	Ethnic	$$	Squirrel Hill	Gluten Free	0
2	R3	Casual Dining	$$	Squirrel Hill	None	0
3	R4	Casual Dining	$$$	Shadyside	Vegetarian	0
4	R5	Casual Dining	$	Oakland	Vegetarian	1
5	R6	Fast Food	$$	Squirrel Hill	None	1
6	R7	Ethnic	$$	Squirrel Hill	None	1
7	R8	Casual Dining	$$	Shadyside	Gluten Free	0
8	R9	Fast Food	$$$	Oakland	None	0
9	R10	Ethnic	$$	Shadyside	Vegetarian	1
10	R11	Casual Dining	$$	Shadyside	Gluten Free	1

3. a)

In [ ]:

```
from math import log2

def entropy(probabilities):
    return -sum([p * log2(p) for p in probabilities if p > 0])

def get_entropy(column):
    elements, counts = np.unique(column, return_counts=True)
    probabilities = counts / len(column)
    return entropy(probabilities)

# Calculate the entropy of the entire dataset
entropy_dataset = get_entropy(df['OK'])
entropy_dataset
```

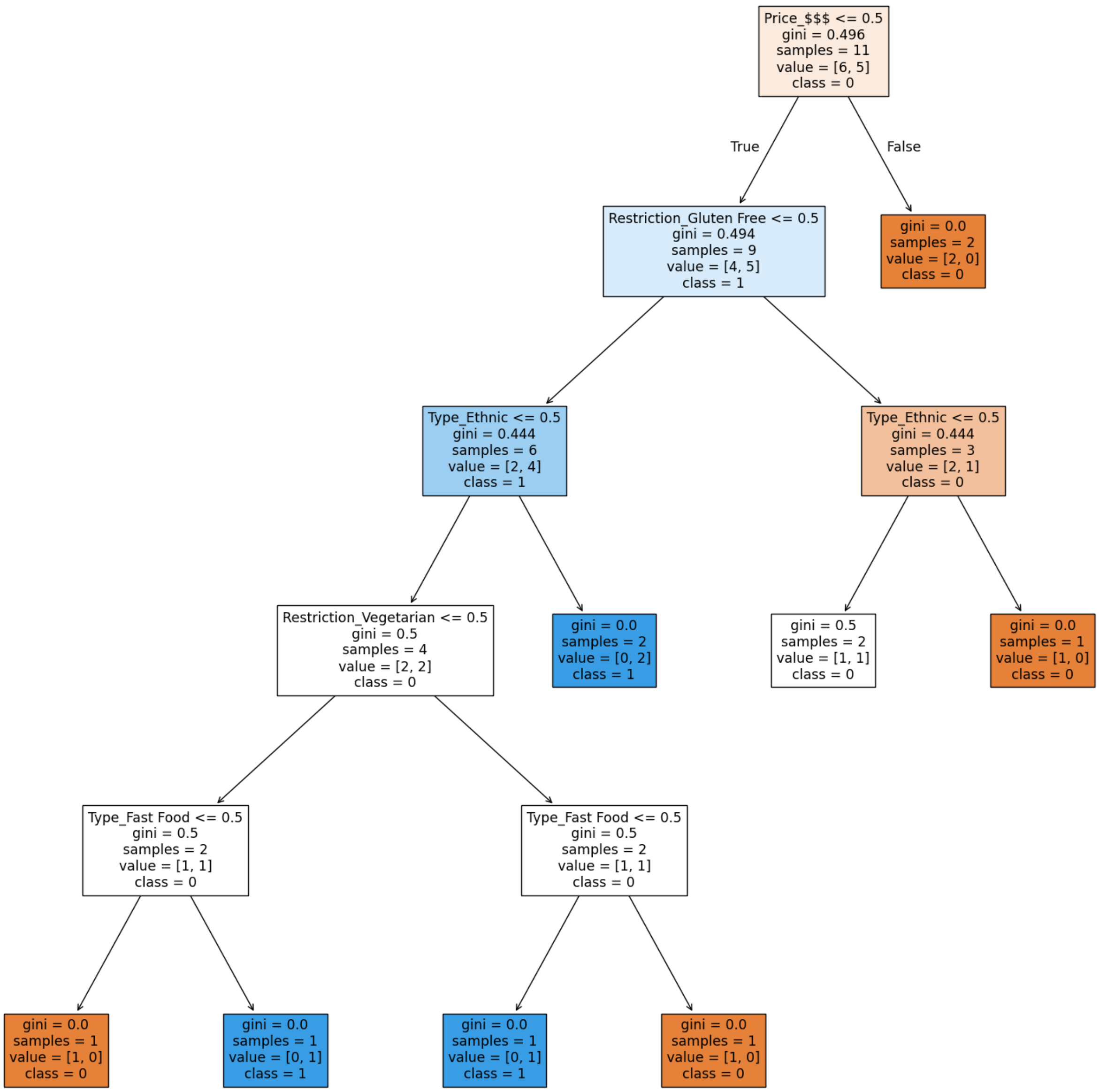
Out[ ]: 0.9940302114769565

3. b)

In [ ]:

```
dt_model = DecisionTreeClassifier(random_state=42)

df_encoded = pd.get_dummies(df.drop(columns='Restaurant'))
X = df_encoded.drop(columns='OK')
y = df_encoded['OK']
```

```
dt_model.fit(X, y)

plt.figure(figsize=(20,20))
_ = tree.plot_tree(
    dt_model,
    feature_names=X.columns,
    class_names=['0', '1'],
    filled=True
)
```



In [ ]:
```
new_data = {
    'Type': ['Fast Food', 'Ethnic', 'Ethnic', 'Casual Dining', 'Ethnic'],
    'Price': ['$', '$$', '$', '$', '$'],
```

```
    'Neighborhood': ['Squirrel Hill', 'Shadyside', 'Oakland', 'Shadyside', '
    'Restriction': ['None', 'None', 'Gluten Free', 'Vegetarian', 'Gluten Fre
}

new_df = pd.DataFrame(new_data)

new_encoded_df = pd.get_dummies(new_df)
new_encoded_df = new_encoded_df.reindex(columns=X.columns, fill_value=0)

prediction = dt_model.predict(new_encoded_df)

new_df['OK'] = prediction

new_df
```

Out[ ]:

	Type	Price	Neighborhood	Restriction	OK
0	Fast Food	$	Squirrel Hill	None	1
1	Ethnic	$$	Shadyside	None	1
2	Ethnic	$	Oakland	Gluten Free	0
3	Casual Dining	$	Shadyside	Vegetarian	1
4	Ethnic	$	Squirrel Hill	Gluten Free	0

Given these predictions i have few options but i would go Shadyside to get some ethnic food.

Question 4

In [ ]:

```
us_crime_df = pd.read_csv('HW4_UScrime_data.csv')
us_crime_df.shape
us_crime_df.head()
```

Out[ ]:

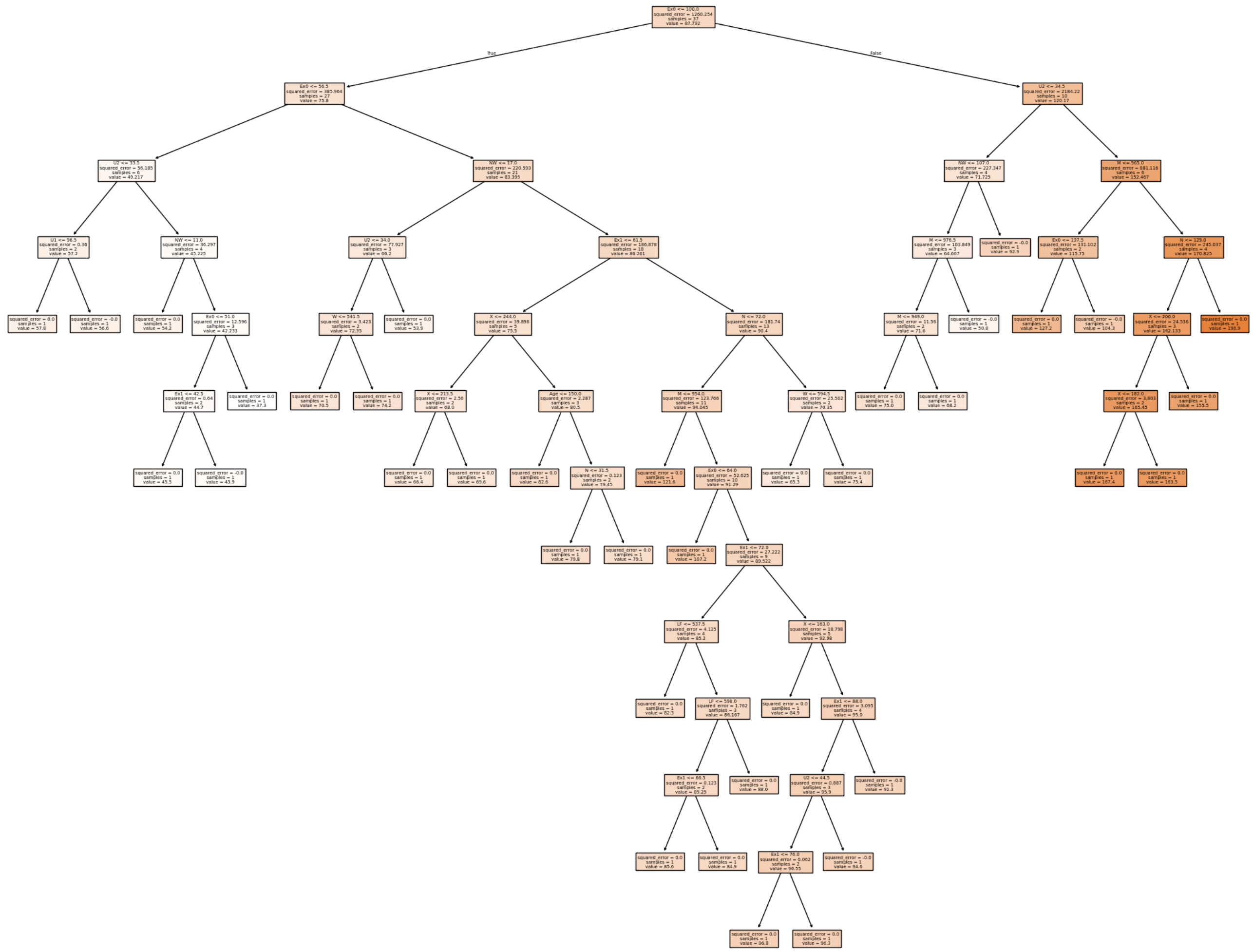
	Crime	Age	Ed	Ex0	Ex1	LF	M	N	NW	U1	U2	W	X
0	79.1	151	91	58	56	510	950	33	301	108	41	394	261
1	163.5	143	113	103	95	583	1012	13	102	96	36	557	194
2	57.8	142	89	45	44	533	969	18	219	94	33	318	250
3	196.9	136	121	149	141	577	994	157	80	102	39	673	167
4	123.4	141	121	109	101	591	985	18	30	91	20	578	174

In [ ]:

```
X = us_crime_df.drop(columns='Crime')
y = us_crime_df['Crime']
```

```
dt_reg = DecisionTreeRegressor(random_state=7)
rf_reg = RandomForestRegressor(random_state=7)

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rando

dt_reg.fit(X_train, y_train)
rf_reg.fit(X_train, y_train)

dt_reg_y_pred = dt_reg.predict(X_val)
rf_reg_y_pred = rf_reg.predict(X_val)

print('Decision Tree:', dt_reg.score(X_val, y_val))
print(f'Decision Tree Accuracy: {metrics.r2_score(y_val, dt_reg_y_pred)}')
print(f'Random Forest R-square: {metrics.r2_score(y_val, rf_reg_y_pred)}')

plt.figure(figsize=(25,20))
_ = tree.plot_tree(dt_reg,
                   feature_names=X.columns,
                   filled=True)
```

```
Decision Tree: 0.33418799066866955
Decision Tree Accuracy: 0.33418799066866955
Random Forest R-square: 0.6345368152843311
```



Question 5

In [ ]:
```
weekly = data('ISLR', 'Weekly')
weekly.set_index(['rownames']).head()
```

Out[ ]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
rownames									
1	1990	0.816	1.572	-3.936	-0.229	-3.484	0.154976	-0.270	Down
2	1990	-0.270	0.816	1.572	-3.936	-0.229	0.148574	-2.576	Down
3	1990	-2.576	-0.270	0.816	1.572	-3.936	0.159837	3.514	Up
4	1990	3.514	-2.576	-0.270	0.816	1.572	0.161630	0.712	Up
5	1990	0.712	3.514	-2.576	-0.270	0.816	0.153728	1.178	Up

In [ ]:
```
logreg = LogisticRegression(max_iter=1000, random_state=42)
rf = RandomForestClassifier(random_state=42)

X = weekly.drop(columns=['Direction', 'Year'], axis=1)
y = weekly['Direction']

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rando

logreg.fit(X_train, y_train)

rf.fit(X_train, y_train)

logreg_y_pred = logreg.predict(X_val)
rf_y_pred = rf.predict(X_val)

print(f'Logistic Regression Accuracry Score: {metrics.accuracy_score(y_val,

print('---' * 14)
print(f'Random Forest Accuracry Score: {metrics.accuracy_score(y_val, rf_y_p
```

The Random Forest Model yield the best accuracy but by a fraction of a percent.