In [5]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("auto-mpg.csv")

# Dimension
print("Shape of the dataset:", df.shape)

# Structure
print("\nColumns and Data Types:\n", df.dtypes)

# First 10 rows
print("Shape of the dataset:", df.head(10))

# Summary
print("\nSummary:\n", df.describe(include='all'))
```

```
Shape of the dataset: (398, 9)

Columns and Data Types:
 mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
Shape of the dataset:     mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
0  18.0          8         307.0        130    3504          12.0          70
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70
5  15.0          8         429.0        198    4341          10.0          70
6  14.0          8         454.0        220    4354           9.0          70
7  14.0          8         440.0        215    4312           8.5          70
8  14.0          8         455.0        225    4425          10.0          70
9  15.0          8         390.0        190    3850           8.5          70

   origin                   car name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
5       1           ford galaxie 500
6       1           chevrolet impala
7       1          plymouth fury iii
8       1           pontiac catalina
9       1         amc ambassador dpl

Summary:
                mpg   cylinders  displacement horsepower       weight  \
count   398.000000  398.000000    398.000000        398   398.000000
unique         NaN         NaN           NaN         94          NaN
top            NaN         NaN           NaN        150          NaN
freq           NaN         NaN           NaN         22          NaN
mean     23.514573    5.454774    193.425879        NaN  2970.424623
std       7.815984    1.701004    104.269838        NaN   846.841774
min       9.000000    3.000000     68.000000        NaN  1613.000000
25%      17.500000    4.000000    104.250000        NaN  2223.750000
50%      23.000000    4.000000    148.500000        NaN  2803.500000
75%      29.000000    8.000000    262.000000        NaN  3608.000000
max      46.600000    8.000000    455.000000        NaN  5140.000000

        acceleration  model year      origin    car name
count     398.000000  398.000000  398.000000         398
unique           NaN         NaN         NaN         305
top              NaN         NaN         NaN  ford pinto
freq             NaN         NaN         NaN           6
mean       15.568090   76.010050    1.572864         NaN
std         2.757689    3.697627    0.802055         NaN
min         8.000000   70.000000    1.000000         NaN
25%        13.825000   73.000000    1.000000         NaN
50%        15.500000   76.000000    1.000000         NaN
75%        17.175000   79.000000    2.000000         NaN
max        24.800000   82.000000    3.000000         NaN
```

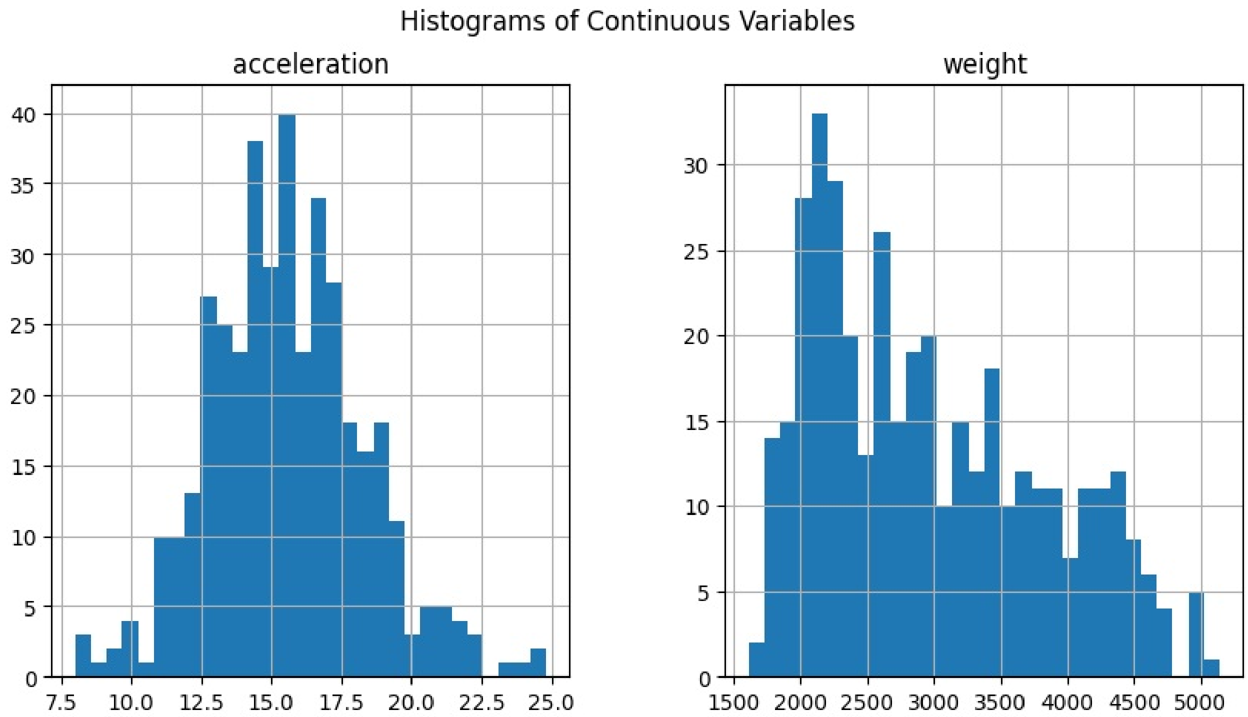
In [ ]:

```python
print(df.isnull().sum())  # Check missing values before filling

# Replace '?' in the 'horsepower' column with NaN and convert it to numeric
df['horsepower'] = df['horsepower'].replace('?', pd.NA)
df['horsepower'] = pd.to_numeric(df['horsepower'])

# Fill missing values in 'horsepower' with the mean
df['horsepower'].fillna(df['horsepower'].mean(), inplace=True)

df.fillna(df.mean(numeric_only=True), inplace=True)  # Fill missing numeric values wdf

print(df.isnull().sum())  # Check again to confirm changes
```

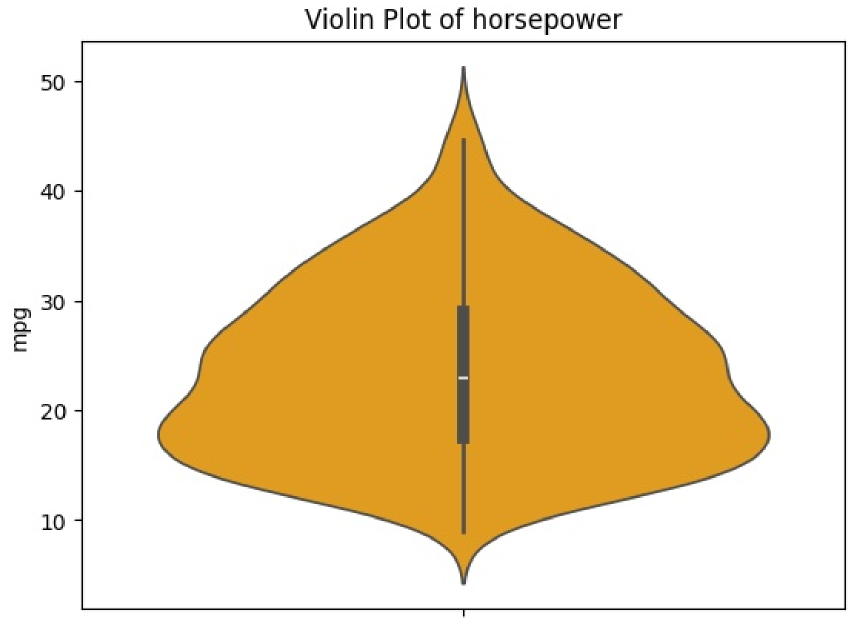
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

```
C:\Users\abhia\AppData\Local\Temp\ipykernel_7600\969698177.py:8: FutureWarning: A value is trying to be set on a
copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on w
hich we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)'
or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['horsepower'].fillna(df['horsepower'].mean(), inplace=True)
```

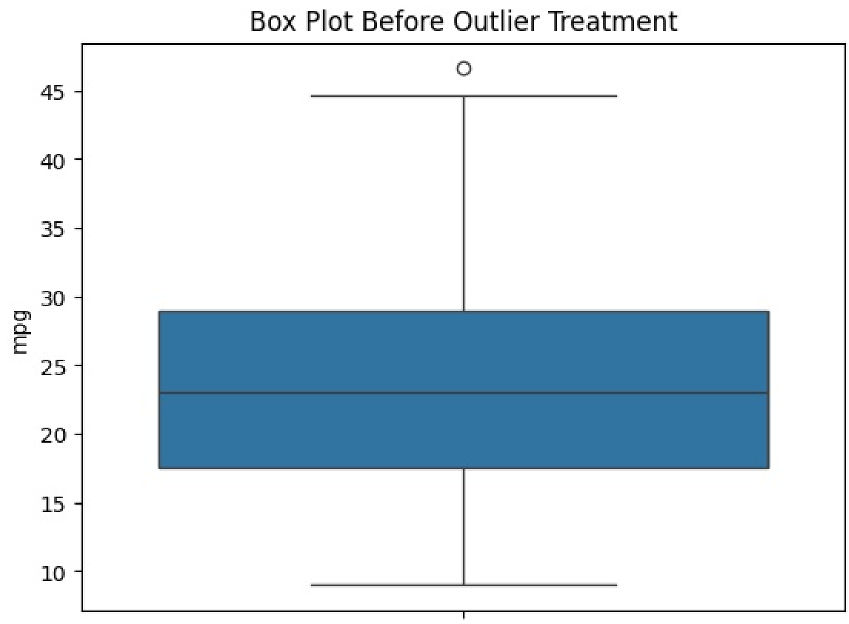
In [7]:
```
# Plot histograms
df[['acceleration', 'weight']].hist(bins=30, figsize=(10, 5))
plt.suptitle('Histograms of Continuous Variables')
plt.show()
```

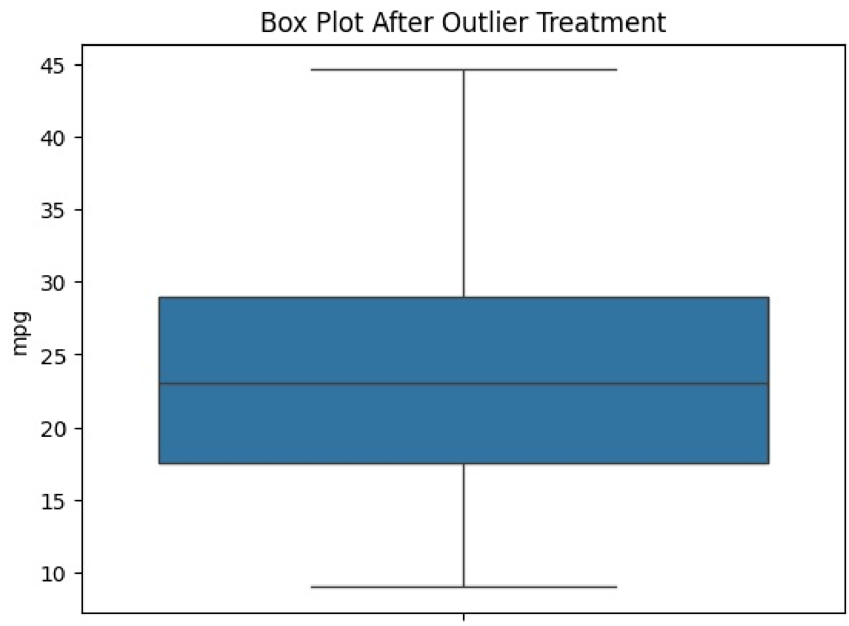


In [8]:
```
# Violin plot for a numerical column
sns.violinplot(y=df["mpg"],color="orange")
plt.title('Violin Plot of horsepower')
plt.show()
```



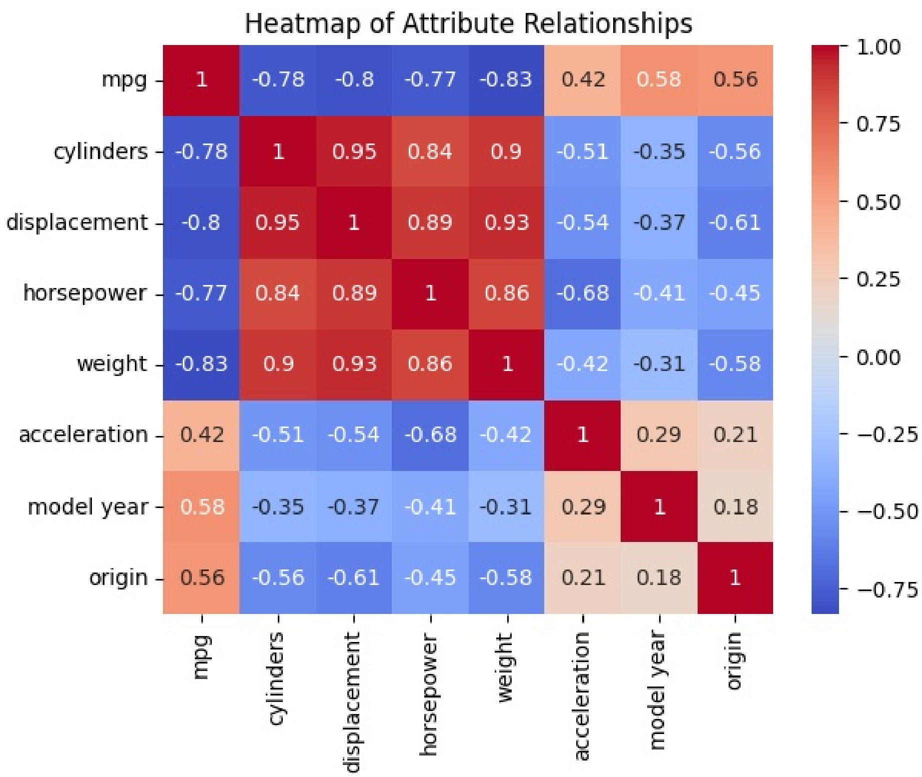
In [9]:
```
#boxplot

#Display box plot before outlier treatment
sns.boxplot(df['mpg'])
plt.title('Box Plot Before Outlier Treatment')
plt.show()

# Identify outliers using IQR (Interquartile Range)
Q1 = df['mpg'].quantile(0.25)
Q3 = df['mpg'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter the data to remove outliers
df_filtered = df[(df['mpg'] >= lower_bound) & (df['mpg'] <= upper_bound)]

# Display box plot after outlier treatment
sns.boxplot(df_filtered['mpg'])
plt.title('Box Plot After Outlier Treatment')
plt.show()
```





In [10]:

```
#heatmap

#  Select only the numeric columns for correlation
numeric_df = df.select_dtypes(include=[float, int])

# # Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# # Create the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Heatmap of Attribute Relationships')
plt.show()
```



In [11]:

```
from sklearn.preprocessing import StandardScaler
```

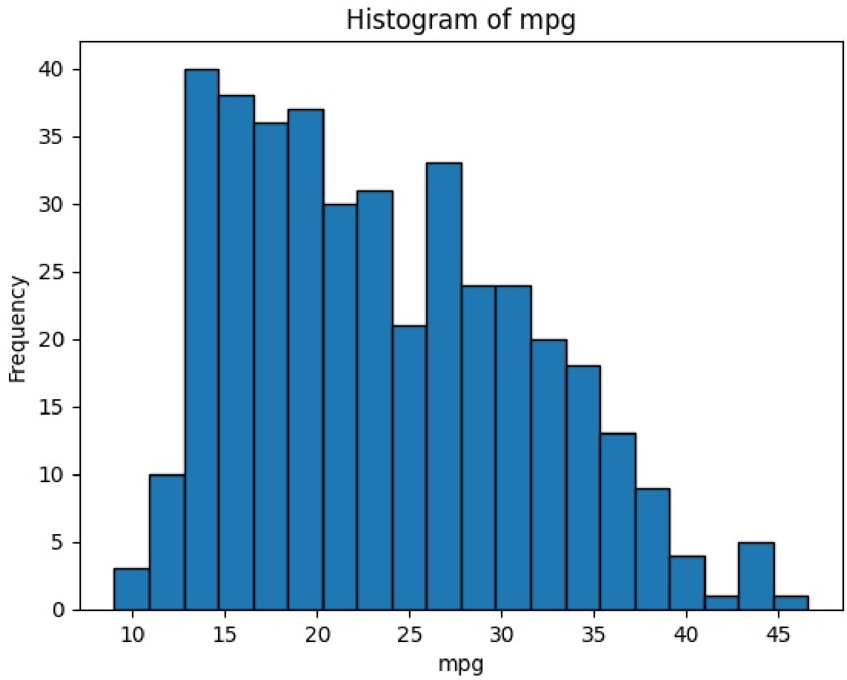
```python
continuous_columns = ['mpg', 'horsepower', 'weight', 'acceleration']

# Initialize StandardScaler
scaler = StandardScaler()

# Standardize the continuous variables
df[continuous_columns] = scaler.fit_transform(df[continuous_columns])

# Save the standardized data to a new CSV file
df.to_csv("standardized_mpg.csv", index=False)

# Print the first few rows to check the results
print("standardized data")
print(df.head())
```

```
standardized data
        mpg  cylinders  displacement  horsepower    weight  acceleration  \
0 -0.706439          8         307.0    0.669196  0.630870     -1.295498
1 -1.090751          8         350.0    1.586599  0.854333     -1.477038
2 -0.706439          8         318.0    1.193426  0.550470     -1.658577
3 -0.962647          8         304.0    1.193426  0.546923     -1.295498
4 -0.834543          8         302.0    0.931311  0.565841     -1.840117

   model year  origin                   car name
0          70       1  chevrolet chevelle malibu
1          70       1          buick skylark 320
2          70       1         plymouth satellite
3          70       1              amc rebel sst
4          70       1                ford torino
```

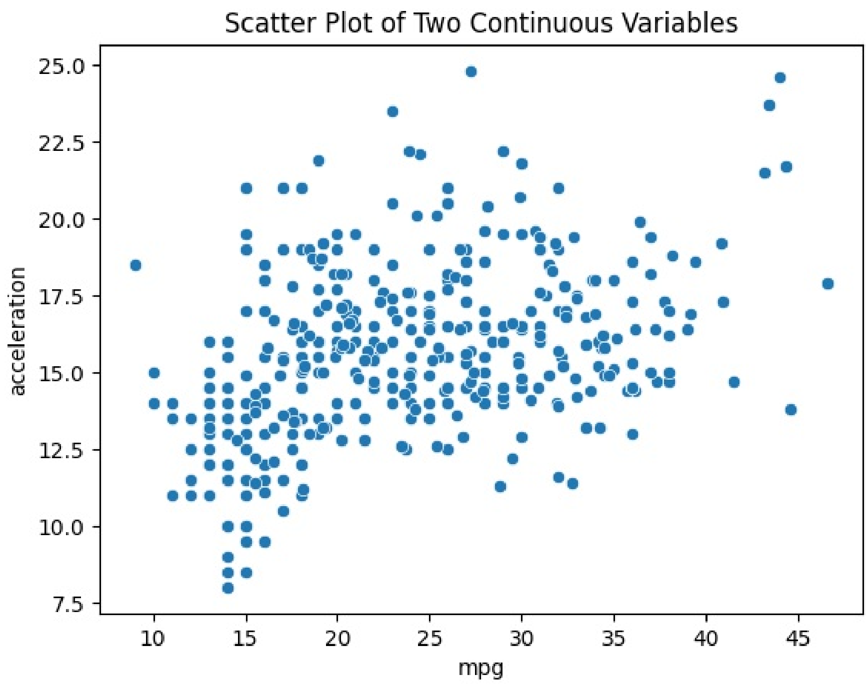
In [7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv("mpg_raw.csv")

print(df.info())
print("Before treatment\n",df.isnull().sum())

df.fillna(df.mean(numeric_only=True),inplace=True)
df.drop_duplicates(inplace=True)
print("After treatment\n",df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    object
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
None
Before treatment
 mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
name            0
dtype: int64
After treatment
 mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model_year      0
origin          0
name            0
dtype: int64
```

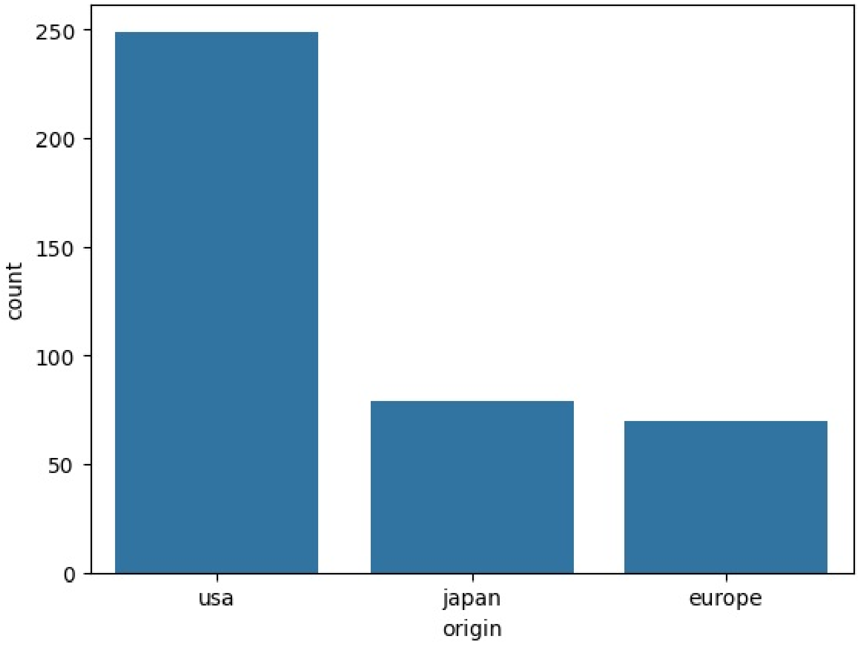
In [9]:
```
# 1. Show the distribution of continuous variables using a histogram
plt.hist(df['mpg'], bins=20, edgecolor='black')
#df.hist(figsize=(10,8),bins=20)
plt.title("Histogram of mpg")
plt.xlabel("mpg")
plt.ylabel("Frequency")
plt.show()
```



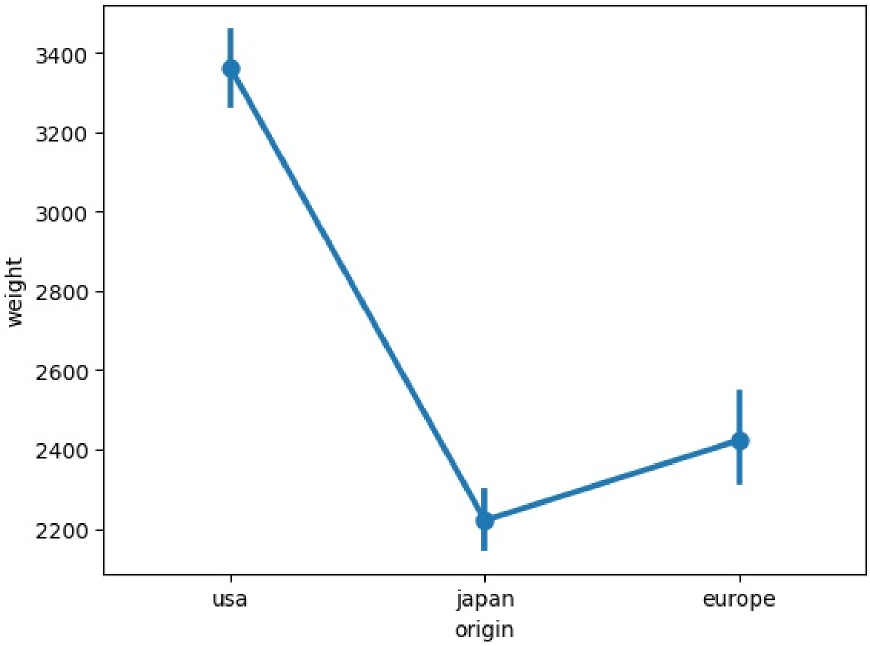

```
In [12]: # 2. Identify the relationship between two continuous variables using a scatter plot
# continous_columns=df.select_dtypes(include=["number"]).columns
sns.scatterplot(x=df["mpg"],y=df["acceleration"])
plt.title("Scatter Plot of Two Continuous Variables")


plt.show()
```




```
In [17]: # 3. Find and display the frequency of categorical values using a count plot
sns.countplot(x=df["origin"])
plt.show()
```



In [19]:
```
# 4. Apply point plots to display one continuous and one categorical variable
sns.pointplot(x=df["origin"],y=df["weight"])
plt.show()
```



In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 1: Load the dataset
df = pd.read_csv('breastcancer1 (2).csv')
df.head(20)
```

Out[1]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concav
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	
7	84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450	
8	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320	
9	84501001	M	12.46	24.04	83.97	475.9	0.11860	0.23960	
10	845636	M	16.02	23.24	102.70	797.8	0.08206	0.06669	
11	84610002	M	15.78	17.89	103.60	781.0	0.09710	0.12920	
12	846226	M	19.17	24.80	132.40	1123.0	0.09740	0.24580	
13	846381	M	15.85	23.95	103.70	782.7	0.08401	0.10020	
14	84667401	M	13.73	22.61	93.60	578.3	0.11310	0.22930	
15	84799002	M	14.54	27.54	96.73	658.8	0.11390	0.15950	
16	848406	M	14.68	20.13	94.74	684.5	0.09867	0.07200	
17	84862001	M	16.13	20.68	108.10	798.8	0.11700	0.20220	
18	849014	M	19.81	22.15	130.00	1260.0	0.09831	0.10270	
19	8510426	B	13.54	14.36	87.46	566.3	0.09779	0.08129	

20 rows × 32 columns

In [2]:
```
# Step 3: Drop the 'id' column as it's not useful
df.drop(columns=['id'], inplace=True)

df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

df.fillna(df.drop(columns=['diagnosis']).mean(), inplace=True)
```

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

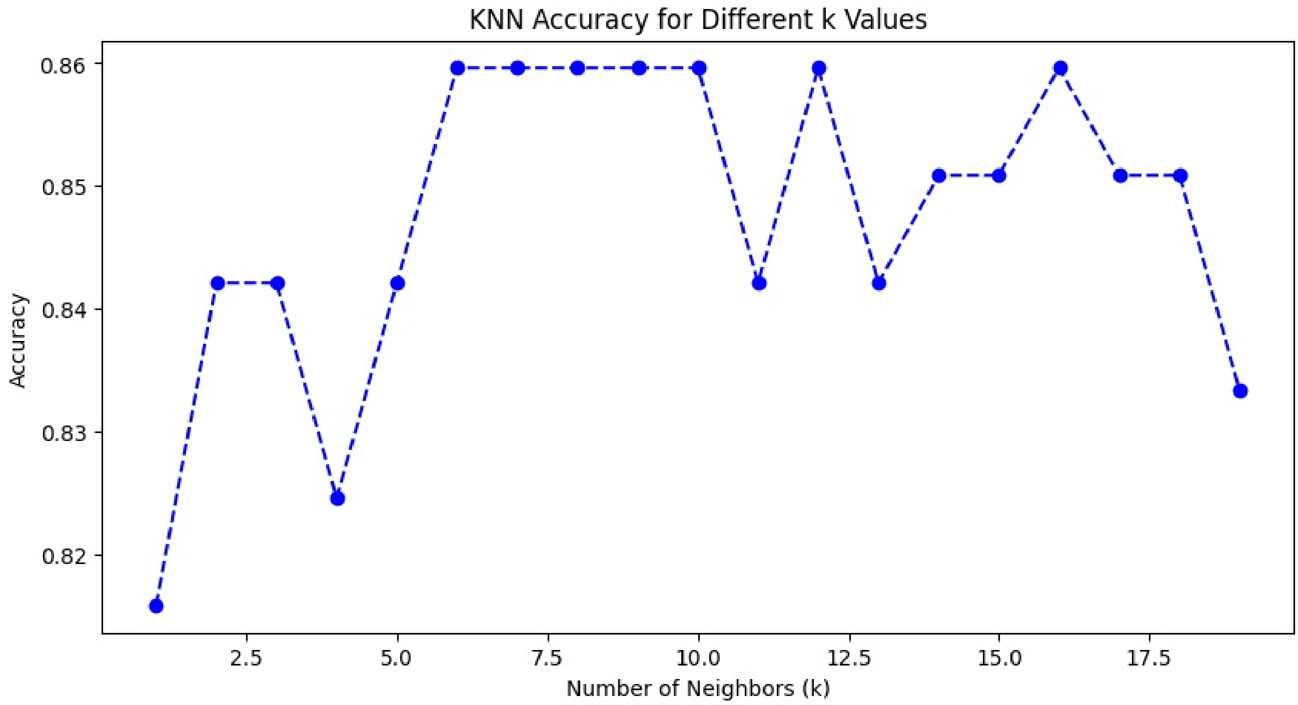
In [7]:
```
# Step 5: Feature Selection (texture_mean, radius_mean) and Target Variable (diagnosis)
X = df[['texture_mean', 'radius_mean']]
y = df['diagnosis']
```

In [8]:
```
# Step 6: Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [9]:
```
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

In [10]:
```
# Step 8: Train KNN Model with different k-values
k_values = range(1, 20)  # Testing k values from 1 to 20
accuracy_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
```

In [11]:
```
# Step 9: Plot Accuracy vs. k-values
plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracy_scores, marker='o', linestyle='dashed', color='b')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('KNN Accuracy for Different k Values')
plt.show()
```



In [12]:
```
# Step 10: Select the best k (max accuracy)
best_k = k_values[np.argmax(accuracy_scores)]
print("Best k value is:",best_k)
```

```
Best k value is: 6
```

In [13]:
```
# Step 11: Train the final KNN model with best k value
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train_scaled, y_train)
```

Out[13]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)
```

In [14]:
```
# Predictions
y_train_pred = knn_best.predict(X_train_scaled)
y_test_pred = knn_best.predict(X_test_scaled)

# Step 12: Model Performance Metrics
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Testing Accuracy: {test_accuracy:.4f}")
```

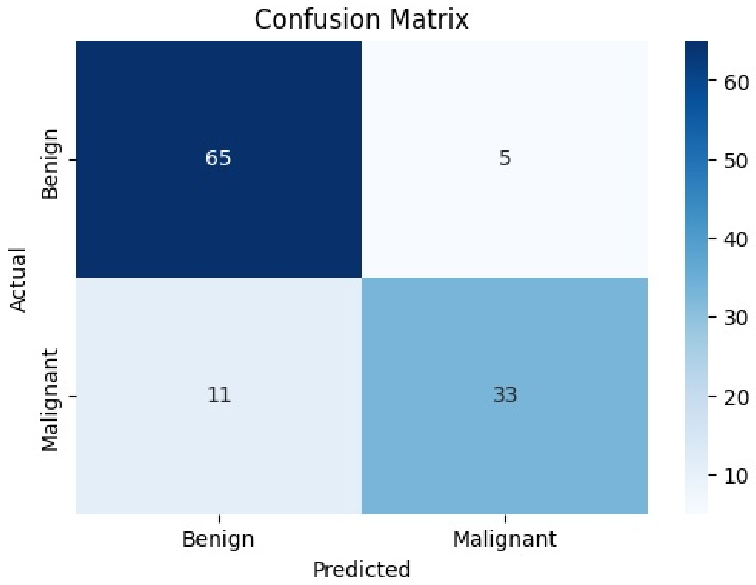
```
Training Accuracy: 0.8615
Testing Accuracy: 0.8596
```

In [15]:
```
# Step 12: Model Performance Metrics
print("\nClassification Report:\n", classification_report(y_test, y_test_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.93      0.89        70
           1       0.87      0.75      0.80        44

    accuracy                           0.86       114
   macro avg       0.86      0.84      0.85       114
weighted avg       0.86      0.86      0.86       114
```

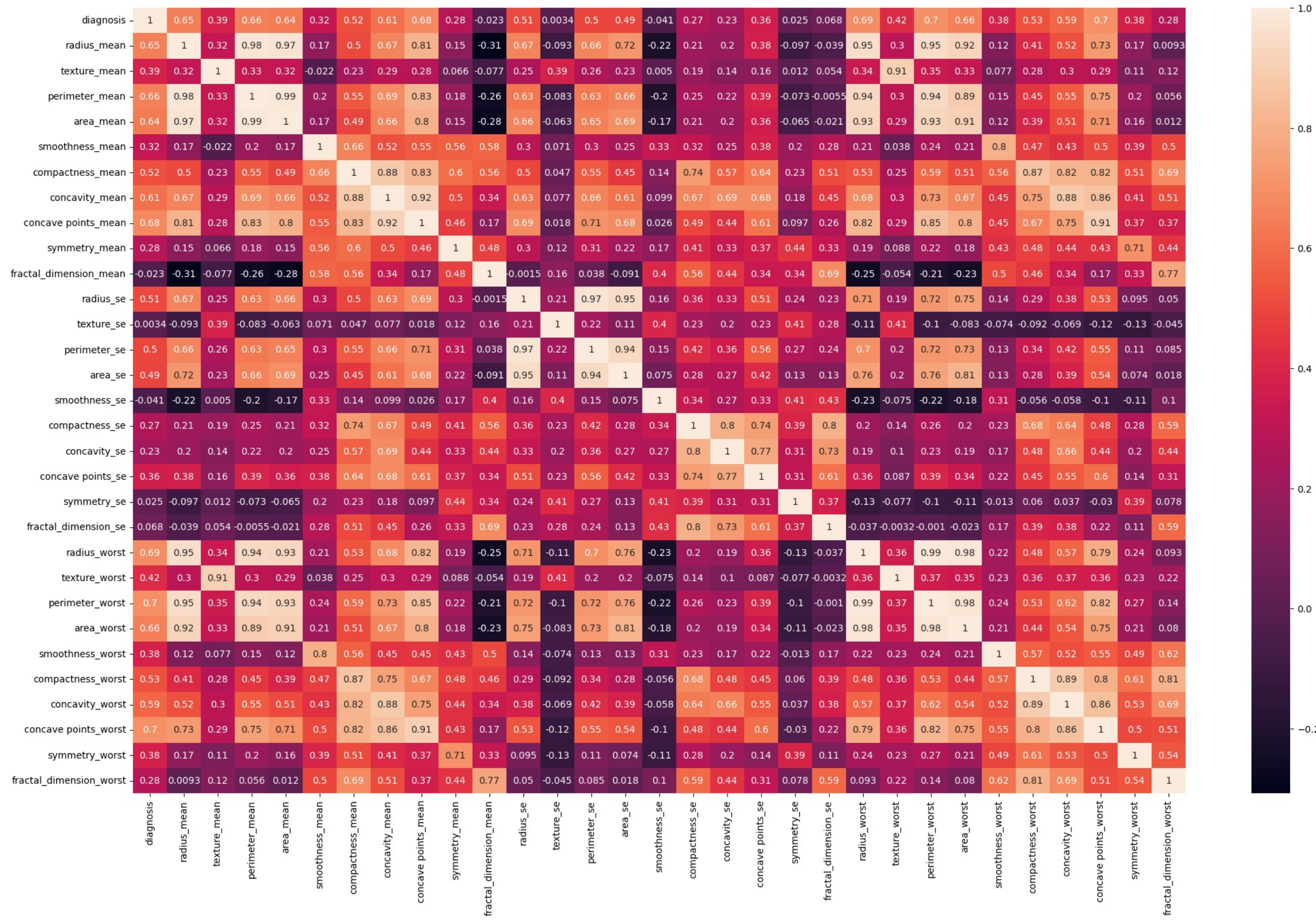
In [31]:
```
# Step 13: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)
```

In [32]:
```
# Step 14: Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Benign', 'Malignant'], yticklabels=['
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



In [47]:

```
plt.figure(figsize=(25,15))
# sns.heatmap(df.select_dtypes(exclude=['object']).corr(),annot=True)
sns.heatmap(df.corr(),annot=True)
plt.show()
```



In [102...

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.tree import DecisionTreeClassifier
%matplotlib inline
```

In [103...

```
df = pd.read_csv("student_performance_new (1).csv")
df.head(5)
```

Out[103...

	Sl.No.	USN	STUDENT NAME	Test I	Test II	Test III	Test Total	Test Result	Quiz 1	Quiz 2	Compensatory	Quiz	Quiz Result	Test + Quiz
0	1	1RV21MC001	ABHISHEK M	34	39	14	29.000000	1	4.0	6.0	10.0	16.0	1	45.000000
1	2	1RV21MC006	AJITH KUMAR K	30	33	27	30.000000	1	6.5	6.0	9.0	15.5	1	45.500000
2	3	1RV21MC009	AKASH E PUNAGIN	28	36	22	28.666667	1	7.0	5.0	NaN	12.0	1	40.666667
3	4	1RV21MC011	AMIT KUMAR	16	28	32	25.333333	1	3.0	2.0	10.0	13.0	1	38.333333
4	5	1RV21MC012	ANANDGOUDA PATIL	25	42	27	31.333333	1	6.0	5.0	10.0	16.0	1	47.333333

In [104...

```
df.isnull().sum()
df.isna().sum()
```

Out[104...

```
Sl.No.                0
USN                   0
STUDENT NAME          0
Test I                0
Test II               0
Test III              0
Test Total            0
Test Result           0
Quiz 1                0
Quiz 2                0
Compensatory          1
Quiz                  0
Quiz Result           0
Test + Quiz           0
Assignment            0
Unnamed: 15           0
Assignment Result     0
Result                0
dtype: int64
```

In [105...

```
df['Compensatory']=df['Compensatory'].fillna(df['Compensatory']).mean()


df.isna().sum()
```

Out[105...

```
Sl.No.                0
USN                   0
STUDENT NAME          0
Test I                0
Test II               0
Test III              0
Test Total            0
Test Result           0
Quiz 1                0
Quiz 2                0
Compensatory          0
Quiz                  0
Quiz Result           0
Test + Quiz           0
Assignment            0
Unnamed: 15           0
Assignment Result     0
Result                0
dtype: int64
```

In [106...

```
y=df['Result']
x=df[['Test Result ','Quiz Result ','Assignment Result ']]
```

In [107...

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

In [108…
```python
DecisionTree = DecisionTreeClassifier(criterion='gini', max_depth=3,random_state=42)
DecisionTree.fit(x_train,y_train)
```

Out[108…

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

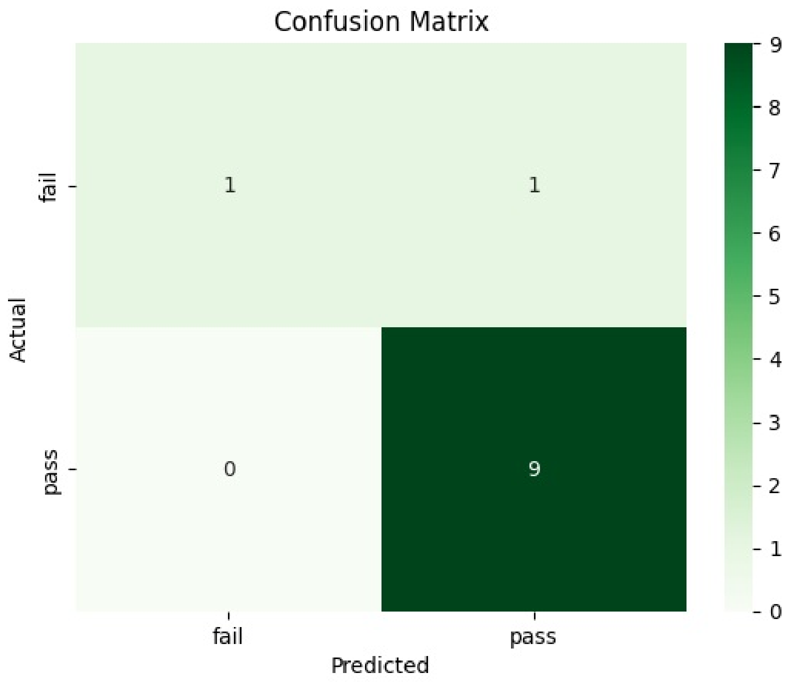
In [109…
```python
#train accuracy
y_train_pred = DecisionTree.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Train accuracy:", train_accuracy)
print(f'Train accuracy:", {train_accuracy:.4f}')
```

```
Train accuracy: 0.9069767441860465
Train accuracy:", 0.9070
```

In [110…
```python
#test accuracy
y_test_pred = DecisionTree.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test accuracy:", test_accuracy)
print(f'Test accuracy:", {test_accuracy:.4f}')
```

```
Test accuracy: 0.9090909090909091
Test accuracy:", 0.9091
```

In [111…
```python
# Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_test_pred)

sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Greens', xticklabels=['fail', 'pass'] , yticklabels=['fail'
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

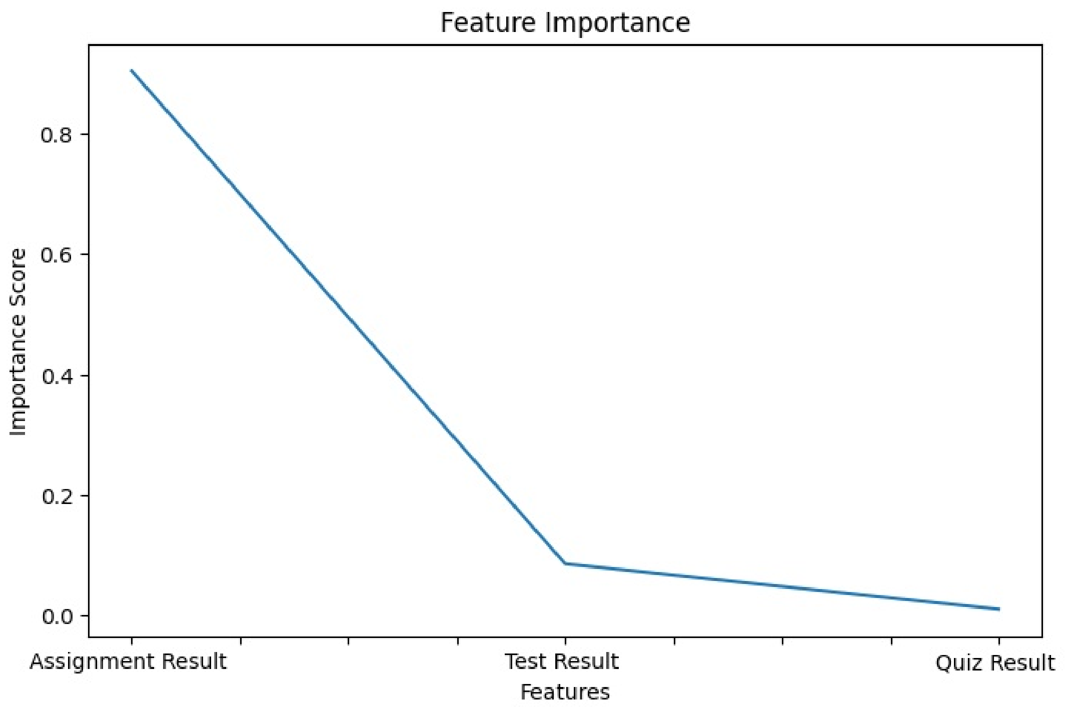


In [112…
```python
#classification report visualization

plt.figure(figsize=(10,7))
classfication_rep =classification_report(y_test,y_test_pred)
print(classfication_rep)
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.90      1.00      0.95         9

    accuracy                           0.91        11
   macro avg       0.95      0.75      0.81        11
weighted avg       0.92      0.91      0.90        11

<Figure size 1000x700 with 0 Axes>
```

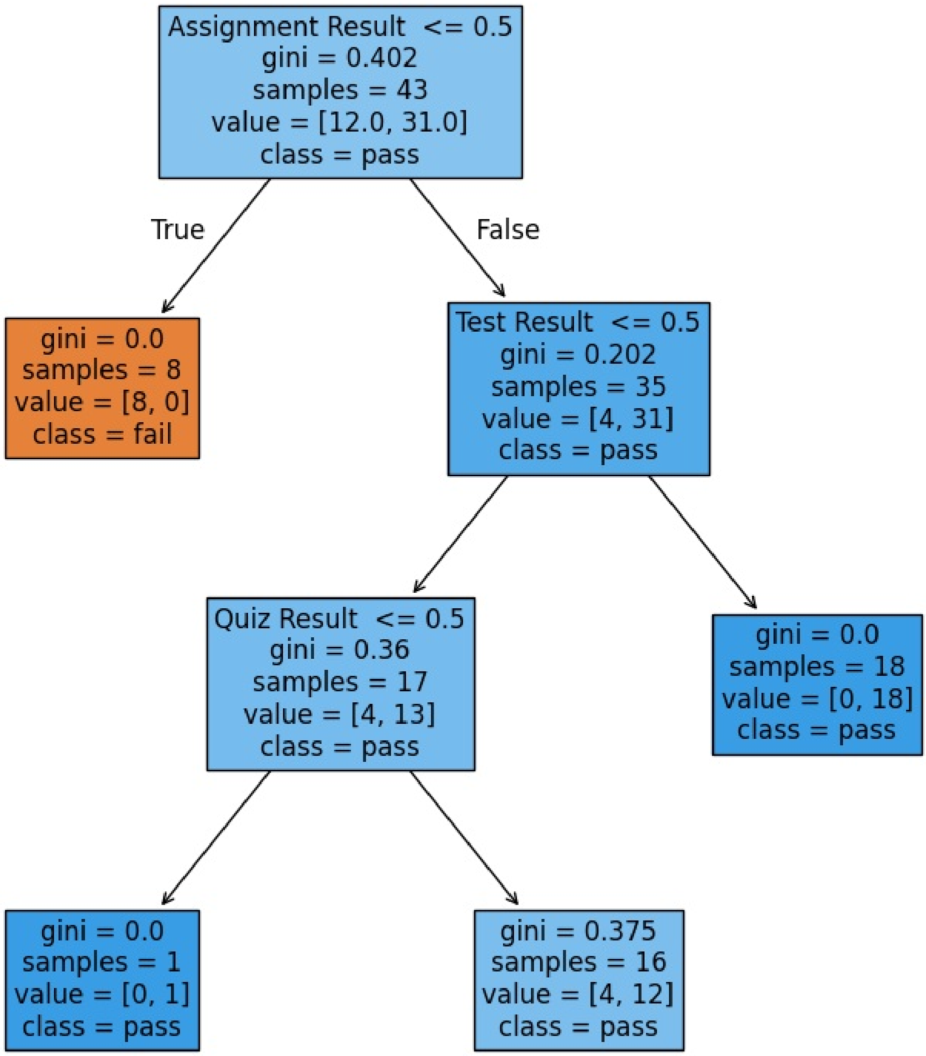
In [113…
```python
# Plot Feature Importance

feature_importance = pd.Series(DecisionTree.feature_importances_, index=x.columns).sort_values(ascending=False)
```

```python
plt.figure(figsize=(8, 5))
feature_importance.plot()  # feature_importance.plot(kind='bar', color='teal')
plt.xlabel("Features")
plt.ylabel("Importance Score")
plt.title("Feature Importance")
plt.show()
```



In [114…

```python
# Decision Tree Visualization

plt.figure(figsize=(10,10))
plot_tree(DecisionTree,feature_names=x.columns.tolist(),class_names=['fail','pass'],filled=True , fontsize=12)
plt.show()
```


Assignment Result <= 0.5
gini = 0.402
samples = 43
value = [12.0, 31.0]
class = pass
True
False
gini = 0.0
samples = 8
value = [8, 0]
class = fail
Test Result <= 0.5
gini = 0.202
samples = 35
value = [4, 31]
class = pass
Quiz Result <= 0.5
gini = 0.36
samples = 17
value = [4, 13]
class = pass
gini = 0.0
samples = 18
value = [0, 18]
class = pass
gini = 0.0
samples = 1
value = [0, 1]
class = pass
gini = 0.375
samples = 16
value = [4, 12]
class = pass

In [146…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score, roc_auc_score
```

In [147…
```
data=pd.read_excel(r'D:\random_forest_dataset.xlsx',header=1)
```

In [148…
```
data.isna().sum()
```

Out[148…
```
Sl No     0
USN       0
Name      0
Title     1
P1        0
C1        0
P2        0
C2        0
P3        0
C3        0
R1        8
T1        8
P3T       0
Total     0
Grade     8
dtype: int64
```

In [149…
```
data.columns
```

Out[149…
```
Index(['Sl No ', 'USN ', 'Name ', 'Title ', 'P1 ', 'C1', 'P2', 'C2', 'P3',
       'C3', 'R1', 'T1', 'P3T', 'Total ', 'Grade'],
      dtype='object')
```

In [150…
```
X=data.drop(columns=['Sl No ','USN ','Name ','Title ','Grade'])
```

In [151…
```
Y=data['Grade']
```

In [152…
```
X.columns
```

Out[152…
```
Index(['P1 ', 'C1', 'P2', 'C2', 'P3', 'C3', 'R1', 'T1', 'P3T', 'Total '], dtype='object')
```

In [153…
```
X.fillna(X.mean(),inplace=True)
```

In [154…
```
Y.fillna(Y.mode()[0],inplace=True)
```

In [155…
```
X.isna().sum()
```

Out[155…
```
P1        0
C1        0
P2        0
C2        0
P3        0
C3        0
R1        0
T1        0
P3T       0
Total     0
dtype: int64
```

In [156…
```
Y.isna().sum()
```

Out[156…
```
np.int64(0)
```

In [157…
```
label_encoder = LabelEncoder()
Y = label_encoder.fit_transform(Y)
```

In [158…
```
Y
```

Out[158…
```
array([0, 0, 0, 0, 0, 3, 3, 1, 1, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 2, 3, 1, 1, 0, 3, 0, 0, 3, 3, 0, 0, 0, 3, 0, 0, 0, 1, 1,
       0, 0, 3, 3, 1, 3, 3, 3, 0, 0, 1, 3, 3, 3, 0, 0, 3, 0, 3, 0, 3, 3,
       0, 3, 0, 3, 3, 0, 3, 0, 3, 0, 3, 2, 3, 2, 3, 0, 1, 2, 0, 3, 1, 3,
       0, 0, 0, 0, 3, 0, 0, 1, 0, 0, 0, 1, 1, 3, 0, 0, 3, 3, 0, 1, 3, 3,
       1, 0, 3, 3, 3, 0, 1])
```

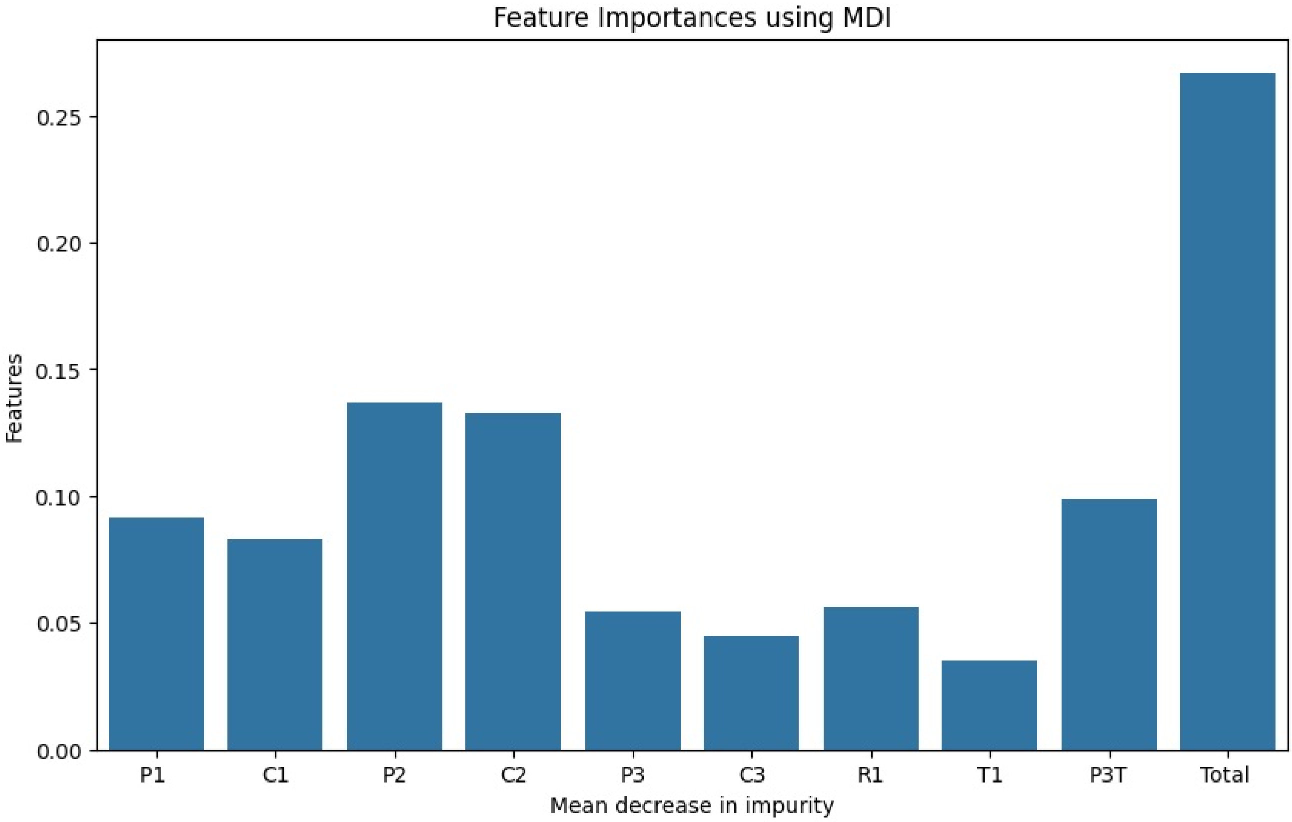
In [159…
```
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.2, random_state=42)
```

In [160…
```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(xtrain, ytrain)
```

Out[160…
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [161…
```
importances = clf.feature_importances_
feature_names = X.columns
clf_importances = pd.Series(importances, index=feature_names)
```

In [162…
```
plt.figure(figsize=(10,6))
sns.barplot(y=clf_importances, x=clf_importances.index)
plt.title("Feature Importances using MDI")
plt.xlabel("Mean decrease in impurity")
plt.ylabel("Features")
plt.show()
```



In [163…
```
param_dist = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

In [185…
```
random_search_forest = RandomizedSearchCV(RandomForestClassifier(), param_distributions=param_dist,
                                           n_iter=10, cv=5, scoring='accuracy', random_state=1)
random_search_forest.fit(xtrain, ytrain)
```

```
C:\Users\wwwsa\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\model_selection\_split.py:805:
UserWarning: The least populated class in y has only 4 members, which is less than n_splits=5.
  warnings.warn(
```

Out[185…
```
RandomizedSearchCV
best_estimator_:
RandomForestClassifier
RandomForestClassifier
```

In [186…
```
best_params = random_search_forest.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'n_estimators': 200, 'min_samples_split': 10, 'min_samples_leaf': 2, 'max_depth': 30}
```

In [187…
```
best_rf = RandomForestClassifier(**best_params, random_state=41)
```

```
best_rf.fit(xtrain, ytrain)
```

Out[187…

RandomForestClassifier

```
RandomForestClassifier(max_depth=30, min_samples_leaf=2, min_samples_split=10,
                       n_estimators=200, random_state=41)
```

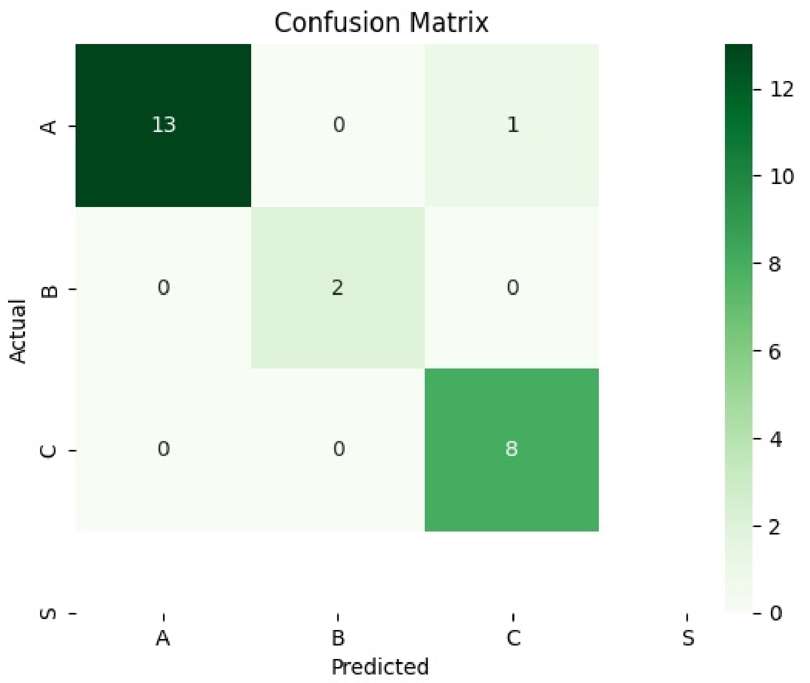
In [188…

```
y_pred = best_rf.predict(xtest)
report = classification_report(ytest, y_pred, zero_division=1)
print(report)
accuracy = accuracy_score(ytest, y_pred)*100
print(f'Accuracy: {accuracy:.2f}')
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        14
           1       1.00      1.00      1.00         2
           3       0.89      1.00      0.94         8

    accuracy                           0.96        24
   macro avg       0.96      0.98      0.97        24
weighted avg       0.96      0.96      0.96        24

Accuracy: 95.83
```

In [189…

```
conf_matrix = confusion_matrix(ytest, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap="Greens", xticklabels=label_encoder.classes_, yticklabels=lal
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [190…

```
print(conf_matrix)
```

```
[[13  0  1]
 [ 0  2  0]
 [ 0  0  8]]
```

In [191…

```
y_pred
```

Out[191…

```
array([0, 0, 0, 1, 0, 3, 3, 0, 0, 0, 0, 3, 1, 3, 0, 0, 3, 3, 3, 3, 3, 0,
       0, 0])
```

In [197…

```
ytest
```

Out[197…

```
array([0, 0, 0, 1, 0, 3, 3, 0, 0, 0, 0, 3, 1, 3, 0, 0, 3, 3, 3, 3, 0, 0,
       0, 0])
```

In [196…

```
ytrain
```

Out[196…

```
array([3, 3, 0, 0, 3, 1, 0, 3, 3, 3, 0, 3, 0, 3, 1, 3, 1, 0, 2, 3, 0, 0,
       0, 0, 0, 3, 3, 3, 1, 1, 3, 2, 1, 0, 0, 0, 2, 1, 0, 3, 0, 1, 0, 0,
       3, 0, 3, 0, 1, 3, 1, 3, 3, 3, 0, 0, 3, 2, 0, 0, 0, 1, 3, 3, 0, 0,
       3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 3, 3, 1, 1, 1, 0, 3, 0,
       0, 3, 3, 3, 0])
```

In [195…
```
Y
```

Out[195…
```
array([0, 0, 0, 0, 0, 3, 3, 1, 1, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 2, 3, 1, 1, 0, 3, 0, 0, 3, 3, 0, 0, 0, 3, 0, 0, 0, 1, 1,
       0, 0, 3, 3, 1, 3, 3, 3, 0, 0, 1, 3, 3, 3, 0, 0, 3, 0, 3, 0, 3, 3,
       0, 3, 0, 3, 3, 0, 3, 0, 3, 0, 3, 2, 3, 2, 3, 0, 1, 2, 0, 3, 1, 3,
       0, 0, 0, 0, 3, 0, 0, 1, 0, 0, 0, 1, 1, 3, 0, 0, 3, 3, 0, 1, 3, 3,
       1, 0, 3, 3, 3, 0, 1])
```

In [ ]:

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB,BernoulliNB,MultinomialNB
from sklearn.metrics import confusion_matrix,classification_report
```

In [2]:
```python
df = pd.read_csv("./Placement_Data.csv")
```

In [3]:
```python
df.head()
```

Out[3]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	stat
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Plac
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Plac
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Plac
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	N Plac
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Plac

In [4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   sl_no           215 non-null    int64  
 1   gender          215 non-null    object 
 2   ssc_p           215 non-null    float64
 3   ssc_b           215 non-null    object 
 4   hsc_p           215 non-null    float64
 5   hsc_b           215 non-null    object 
 6   hsc_s           215 non-null    object 
 7   degree_p        215 non-null    float64
 8   degree_t        215 non-null    object 
 9   workex          215 non-null    object 
 10  etest_p         215 non-null    float64
 11  specialisation  215 non-null    object 
 12  mba_p           215 non-null    float64
 13  status          215 non-null    object 
 14  salary          148 non-null    float64
dtypes: float64(6), int64(1), object(8)
memory usage: 25.3+ KB
```

In [5]:
```python
df.columns
```

Out[5]:
```
Index(['sl_no', 'gender', 'ssc_p', 'ssc_b', 'hsc_p', 'hsc_b', 'hsc_s',
       'degree_p', 'degree_t', 'workex', 'etest_p', 'specialisation', 'mba_p',
       'status', 'salary'],
      dtype='object')
```

In [6]:
```python
X= df[['ssc_p','hsc_p','degree_p','etest_p','mba_p']]
y = df['status']
```

In [7]:
```python
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

In [8]:
```python
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [9]:
```python
nb = GaussianNB()
nb.fit(X_train,y_train)
print("Naive Bayes score with Gaussian NB: ",nb.score(X_test,y_test)*100)

predictions = nb.predict(X_test)

print(classification_report(y_test,predictions))
```

```
Naive Bayes score with Gaussian NB:  72.09302325581395
              precision    recall  f1-score   support

           0       0.50      0.42      0.45        12
           1       0.79      0.84      0.81        31

    accuracy                           0.72        43
   macro avg       0.64      0.63      0.63        43
weighted avg       0.71      0.72      0.71        43
```

In [ ]:

In [ ]:

In [10]:

```
bnb = BernoulliNB(binarize=0.0)
bnb.fit(X_train,y_train)
y_pred  = bnb.predict(X_test)
print("Naive Bayes score with Bernoulli NB: ",nb.score(X_test,y_test)*100)
print(classification_report(y_test,y_pred))
```

```
Naive Bayes score with Bernoulli NB:  72.09302325581395
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        12
           1       0.72      1.00      0.84        31

    accuracy                           0.72        43
   macro avg       0.36      0.50      0.42        43
weighted avg       0.52      0.72      0.60        43
```

```
/home/harsha/Desktop/RVCE/SEM1/DataScience/labenv/lib/python3.12/site-packages/sklearn/metrics/_classification.p
y:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/harsha/Desktop/RVCE/SEM1/DataScience/labenv/lib/python3.12/site-packages/sklearn/metrics/_classification.p
y:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/harsha/Desktop/RVCE/SEM1/DataScience/labenv/lib/python3.12/site-packages/sklearn/metrics/_classification.p
y:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [11]:

```
clf = MultinomialNB()
clf.fit(X_train,y_train)
y_pred  = clf.predict(X_test)
print("NB with Multinomial NB Classifier: ",clf.score(X_test,y_test))
print(classification_report(y_test,y_pred))
```

```
NB with Multinomial NB Classifier:  0.8372093023255814
              precision    recall  f1-score   support

           0       0.86      0.50      0.63        12
           1       0.83      0.97      0.90        31

    accuracy                           0.84        43
   macro avg       0.85      0.73      0.76        43
weighted avg       0.84      0.84      0.82        43
```

In [ ]:

In [1]:
```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load dataset
df = pd.read_csv("./basket.csv")

# Fill missing values
df.fillna('', inplace=True)

# One-hot encoding
df_dum = pd.get_dummies(df)

# Apriori algorithm to find frequent itemsets
frequent_items = apriori(df_dum, min_support=0.01, use_colnames=True)
print(frequent_items)

# Generate association rules
rules = association_rules(frequent_items, metric='confidence', min_threshold=0.01)

# Sort rules by support and confidence
rules = rules.sort_values(['support', 'confidence'], ascending=[False, False])

# Display results
rules
```

```
        support                                          itemsets
0      0.024527                                          (0_beef)
1      0.010827                                       (0_berries)
2      0.016040                                  (0_bottled beer)
3      0.019849                                 (0_bottled water)
4      0.010158                                   (0_brown bread)
...         ...                                               ...
22650  0.017510  (7_, 8_, 9_, 5_, 1_shopping bags, 6_, 3_, 10_,...
22651  0.027401      (7_, 8_, 9_, 1_soda, 5_, 6_, 3_, 10_, 2_, 4_)
22652  0.015639  (7_, 8_, 9_, 5_, 6_, 1_whipped/sour cream, 3_,...
22653  0.040767  (7_, 8_, 9_, 5_, 6_, 3_, 10_, 2_, 1_whole milk...
22654  0.026599    (7_, 8_, 9_, 5_, 1_yogurt, 6_, 3_, 10_, 2_, 4_)

[22655 rows x 2 columns]
```

Out[1]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	convicti
1024	(9_)	(10_)	0.999933	0.999933	0.999933	1.000000	1.000067	1.0	6.682705e-05	
1025	(10_)	(9_)	0.999933	0.999933	0.999933	1.000000	1.000067	1.0	6.682705e-05	
1020	(8_)	(9_)	0.996592	0.999933	0.996592	1.000000	1.000067	1.0	6.660373e-05	
1022	(8_)	(10_)	0.996592	0.999933	0.996592	1.000000	1.000067	1.0	6.660373e-05	
12222	(8_, 9_)	(10_)	0.996592	0.999933	0.996592	1.000000	1.000067	1.0	6.660373e-05	
...	...	...	...	...	...	...	...	...	...	
1384209	(9_)	(7_, 8_, 5_, 0_hamburger meat, 6_, 10_, 2_, 4_)	0.999933	0.010025	0.010025	0.010025	1.000067	1.0	6.699678e-07	1.0000
1384213	(10_)	(7_, 8_, 9_, 5_, 0_hamburger meat, 6_, 2_, 4_)	0.999933	0.010025	0.010025	0.010025	1.000067	1.0	6.699678e-07	1.0000
1529546	(9_, 10_)	(7_, 8_, 5_, 0_hamburger meat, 6_, 3_, 2_, 4_)	0.999933	0.010025	0.010025	0.010025	1.000067	1.0	6.699678e-07	1.0000
1529572	(9_)	(7_, 8_, 5_, 0_hamburger meat, 6_, 3_, 10_, 2_...	0.999933	0.010025	0.010025	0.010025	1.000067	1.0	6.699678e-07	1.0000
1529577	(10_)	(7_, 8_, 9_, 5_, 0_hamburger meat, 6_, 3_, 2_,...	0.999933	0.010025	0.010025	0.010025	1.000067	1.0	6.699678e-07	1.0000

1559218 rows × 14 columns

In [2]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
```

In [3]:
```
df=pd.read_csv("Mall_Customers.csv")
df.head()
```

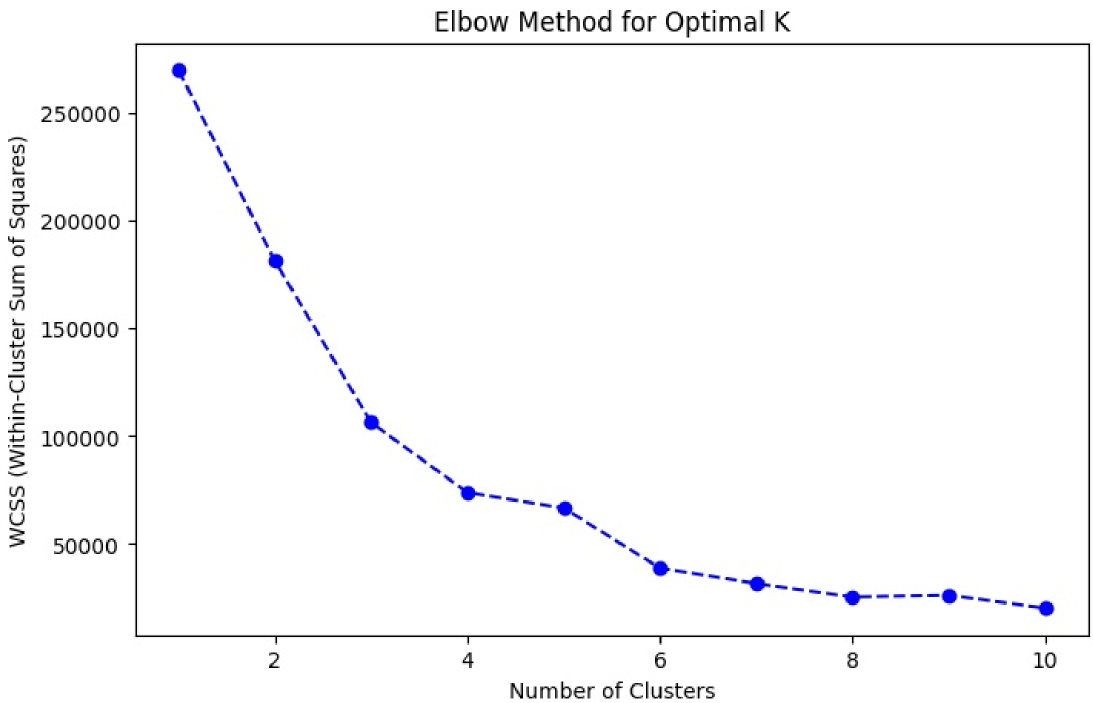
Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [4]:
```
# Selecting relevant features for clustering
X = df[['Annual Income (k$)', 'Spending Score (1-100)']]
```

In [5]:
```
# Finding the optimal number of clusters using the Elbow Method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

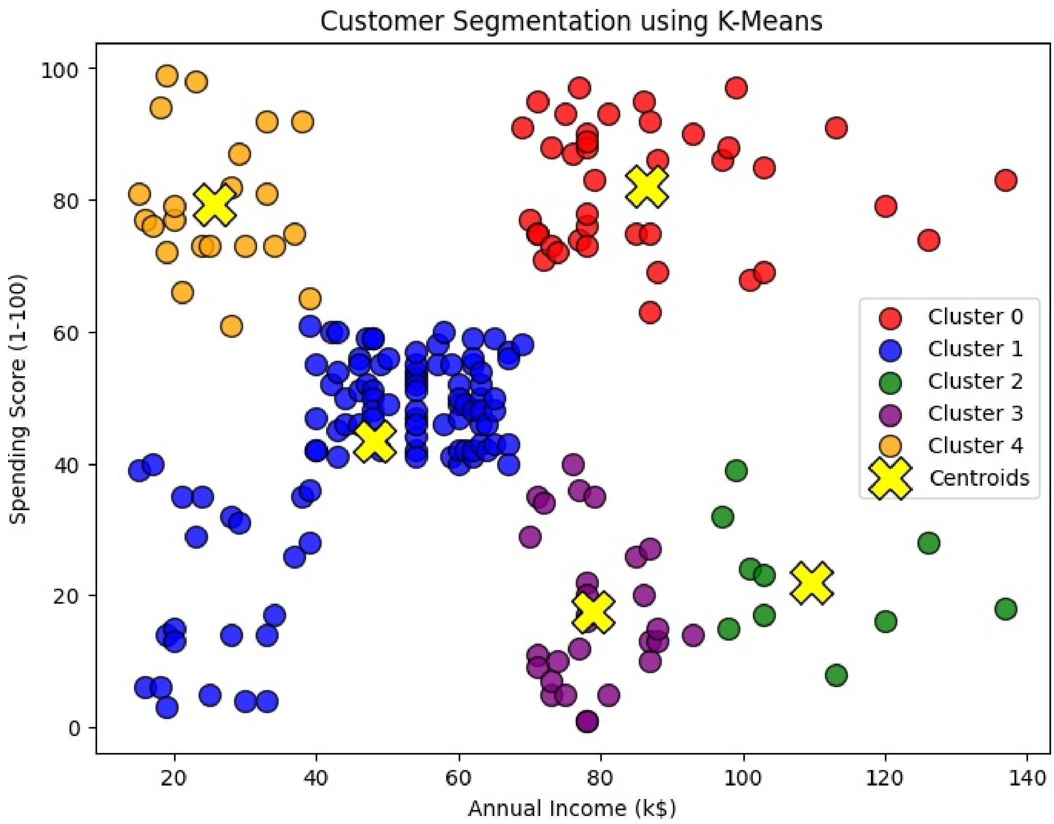
In [6]:
```
# Plotting the Elbow Method graph
plt.figure(figsize=(8,5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--', color='b')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.title('Elbow Method for Optimal K')
plt.show()
```



In [7]:
```
# Step 2: Apply K-Means with the optimal number of clusters
optimal_clusters = 5  # Based on the elbow method
kmeans = KMeans(n_clusters=optimal_clusters)
df['Cluster'] = kmeans.fit_predict(X)
```

In [8]:
```
# Step 3: Plot both clusters and centroids in the same graph
plt.figure(figsize=(8,6))
colors = ['red', 'blue', 'green', 'purple', 'orange','brown']  # Colors for clusters

# Plot each cluster
for i in range(optimal_clusters):
    plt.scatter(X[df['Cluster'] == i]['Annual Income (k$)'],
                X[df['Cluster'] == i]['Spending Score (1-100)'],
               s=100, c=colors[i], label=f'Cluster {i}', edgecolors='black', alpha=0.8)

# Plot centroids (on the same figure)
```

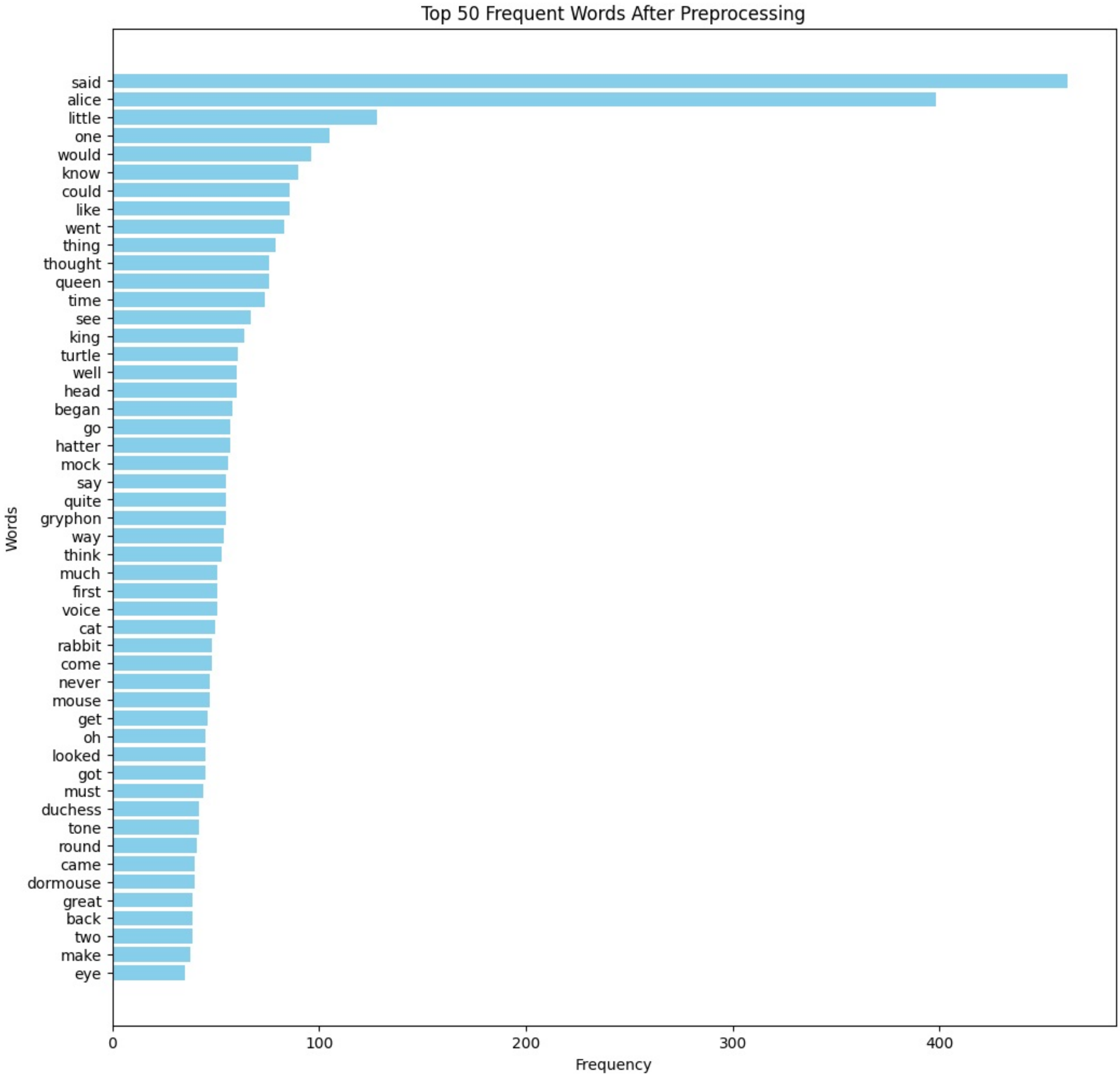
```
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=400, c='yellow', marker='X', label='Centroids', edgecolors='black')

plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Customer Segmentation using K-Means')
plt.legend()
plt.show()
```



In [16]:

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from collections import Counter
from wordcloud import WordCloud,STOPWORDS
from PIL import Image

import matplotlib.pyplot as plt
import numpy as np

# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Load the dataset
with open('alice_in_wonderland.txt') as file:
    text = file.read().lower()

# Tokenization
tokens = word_tokenize(text)

# Remove numbers, punctuation, and whitespaces
tokens = [word for word in tokens if word.isalpha()]

# Remove stop words
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word not in stop_words]

# Stemming and Lemmatization
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
tokens_stemmed = [stemmer.stem(word) for word in tokens]
tokens_lemmatized = [lemmatizer.lemmatize(word) for word in tokens]

# Find frequent words
word_freq = Counter(tokens_lemmatized)
common_words = word_freq.most_common(50)

# Plot bar chart of frequent words
plt.figure(figsize=(12, 12))
plt.barh([word[0] for word in common_words], [word[1] for word in common_words], color='skyblue')
plt.xlabel("Frequency")
plt.ylabel("Words")
plt.title("Top 50 Frequent Words After Preprocessing")
plt.gca().invert_yaxis()
plt.show()

# Generate and plot the word cloud
wordcloud = WordCloud(width=800, height=400, background_color="white", colormap="plasma",
                      max_words=100, contour_color='blue').generate_from_frequencies(word_freq)

plt.figure(figsize=(12, 12))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud After Preprocessing", fontsize=14)
plt.show()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\abhia\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\abhia\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\abhia\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

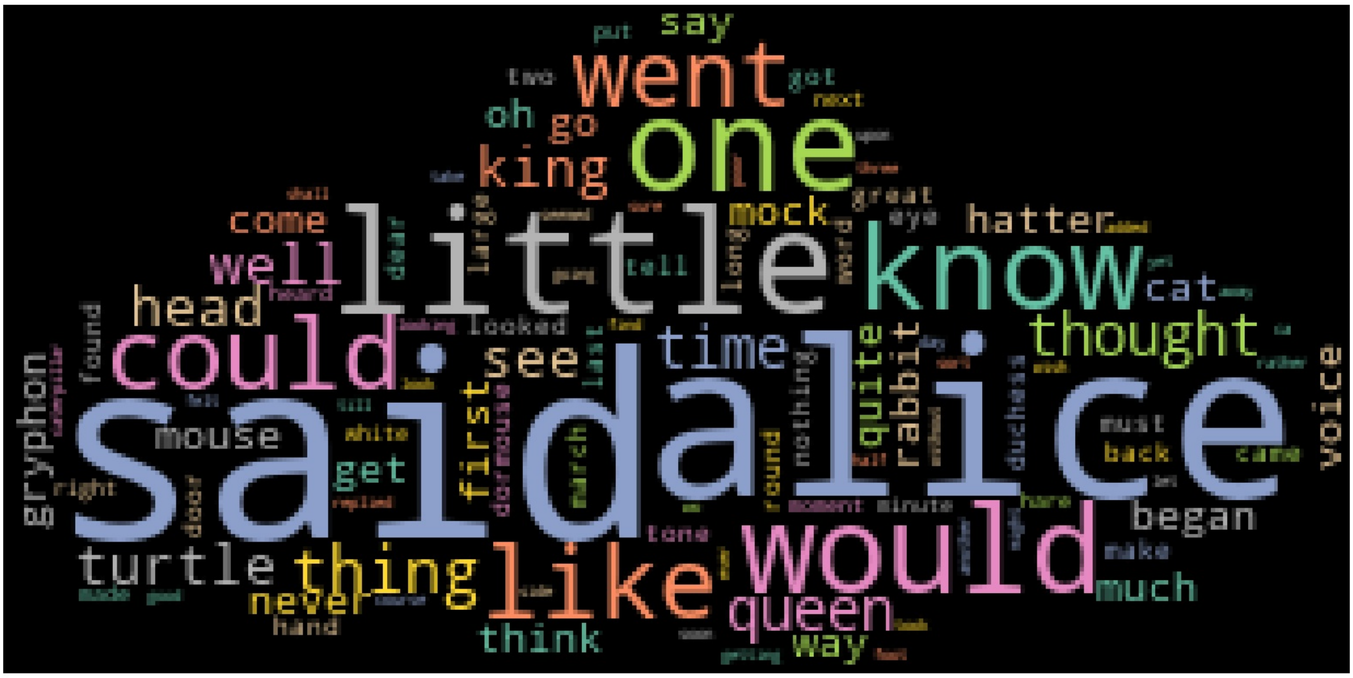

Top 50 Frequent Words After Preprocessing
said
alice
little
one
would
know
could
like
went
thing
thought
queen
time
see
king
turtle
well
head
began
go
hatter
mock
say
quite
gryphon
way
think
much
first
voice
cat
rabbit
come
never
mouse
get
oh
looked
got
must
duchess
tone
round
came
dormouse
great
back
two
make
eye
Words
0
100
200
300
400
Frequency

Word Cloud After Preprocessing


```
In [30]: car_mask = np.array(Image.open("./cloud2.jpeg"))
         car_mask = 255 - car_mask
```

```
In [31]: wordcloud2 = WordCloud(width=500,height=500,mask=car_mask,background_color='black',colormap='Set2',
                                collocations=False,stopwords=STOPWORDS).generate_from_frequencies(dict(word_freq))
         plt.figure(figsize=(20,15))
         plt.imshow(wordcloud2)
         plt.axis("off")
```

```
Out[31]: (np.float64(-0.5), np.float64(318.5), np.float64(157.5), np.float64(-0.5))
```


said
alice
little
know
one
would
could
like
went
thing
turtle
queen
time
thought
king
head
well
come
mock
hatter
cat
see
mouse
gryphon
first
rabbit
quite
voice
began
much
think
way
never
get
oh
go
say
great
nothing
duchess
must
back
came
hand
looked
found
round
tell
long
large
dear
last
march
dormouse
two
got
next
put
make
door
right
white
tone
moment
minute
eye
word
here
heard