In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import normaltest, norm, f
```

In [ ]:

```
df = pd.read_csv("../dataset/banana.csv")
```

In [ ]:

```
# Implementasi fungsi statistik

def mean(data: np.ndarray) -> float:
    total = 0
    for i in data:
        total += i
    return total / len(data)

def median(data: np.ndarray) -> float:
    data = np.sort(data)
    if len(data) % 2 == 0:
        return (data[len(data) // 2] + data[len(data) // 2 - 1]) / 2
    else:
        return data[len(data) // 2]

def mode(data: np.ndarray) -> float:
    data = np.sort(data)
    max_count = 0
    max_num = 0
    current_count = 0
    current_num = 0
    for i in data:
        if i == current_num:
            current_count += 1
        else:
            if current_count > max_count:
                max_count = current_count
                max_num = current_num
            current_count = 1
            current_num = i
    return max_num

def variance(data: np.ndarray) -> float:
    mean_data = mean(data)
    total = 0
    for i in data:
        total += (i - mean_data) ** 2
    return total / (len(data) - 1)

def std_dev(data: np.ndarray) -> float:
    return variance(data) ** 0.5

def data_range(data: np.ndarray) -> float:
    return max(data) - min(data)

def percentile(data: np.ndarray, p: float) -> float:
    data = np.sort(data)
    n = len(data)
    rank = p * (n - 1) / 100
    k = int(rank)
    d = rank - k
    return data[k] + d * (data[k + 1] - data[k])

def quartile(data: np.ndarray, q: int) -> float:
    if q == 1:
        return percentile(data, 25)
    elif q == 2:
        return percentile(data, 50)
    elif q == 3:
        return percentile(data, 75)

def iqr(data: np.ndarray) -> float:
    return quartile(data, 3) - quartile(data, 1)

def skewness(data: np.ndarray) -> float:
    mean_data = mean(data)
    std_dev_data = std_dev(data)
    total = 0
    for i in data:
        total += (i - mean_data) ** 3
    return total / (len(data) * std_dev_data ** 3)

def kurtosis(data: np.ndarray) -> float:
    mean_data = mean(data)
    std_dev_data = std_dev(data)
    total = 0
    for i in data:
        total += (i - mean_data) ** 4
    return total / (len(data) * std_dev_data ** 4) - 3

def unique(data: np.ndarray) -> np.ndarray:
    unique_data = []
    for i in data:
        if i not in unique_data:
            unique_data.append(i)
    return np.array(unique_data)

def proportion(data: np.ndarray) -> np.ndarray:
    unique_data = unique(data)
    data = np.sort(data)
    result = []
    for i in unique_data:
        result.append([i, len(data[data == i]) / len(data)])

    # Sort by proportion descending
    result = np.array(result)
    result = result[result[:, 1].argsort()[::-1]]
    return result
```

1. Menulis deskripsi statistika (Descriptive Statistics) dari semua kolom pada data. Data yang bersifat numerik dapat diberikan nilai mean, median, modus, standar deviasi, variansi, range, nilai minimum, maksimum, kuartil, IQR, skewness dan kurtosis. Data dalam bentuk string dapat dicari unique values, dan proporsi nya.

Terdapat 11 kolom pada data csv, yaitu Acidity, Weight, Length, Appearance, Tannin, Ripeness, Sweetness, Country_of_Origin, Firmness, Grade, dan Price. Berikut adalah deskripsi statistika dari masing-masing kolom.

In [ ]:

```python
# Acidity, bersifat numerik
acidity = df["Acidity"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean acidity
implemented_mean_acidity = mean(acidity)
# Median acidity
implemented_median_acidity = median(acidity)
# Mode acidity
implemented_mode_acidity = mode(acidity)
# Standard deviation acidity
implemented_std_acidity = std_dev(acidity)
# Variance acidity
implemented_var_acidity = variance(acidity)
# Range acidity
implemented_range_acidity = data_range(acidity)
# Minimum acidity
implemented_min_acidity = min(acidity)
# Maximum acidity
implemented_max_acidity = max(acidity)
# Q1 acidity
implemented_q1_acidity = quartile(acidity, 1)
# Q3 acidity
implemented_q3_acidity = quartile(acidity, 3)
# IQR acidity
implemented_iqr_acidity = iqr(acidity)
# Skewness acidity
implemented_skew_acidity = skewness(acidity)
# Kurtosis acidity
implemented_kurt_acidity = kurtosis(acidity)

# Menggunakan numpy
# Mean acidity
mean_acidity = np.mean(acidity)
# Median acidity
median_acidity = np.median(acidity)
# Mode acidity
mode_acidity = df["Acidity"].mode().values[0]
# Standard deviation acidity
std_acidity = np.std(acidity)
# Variance acidity
var_acidity = np.var(acidity)
# Range acidity
range_acidity = np.ptp(acidity)
# Minimum acidity
min_acidity = np.min(acidity)
# Maximum acidity
max_acidity = np.max(acidity)
# Q1 acidity
q1_acidity = np.percentile(acidity, 25)
# Q3 acidity
q3_acidity = np.percentile(acidity, 75)
# IQR acidity
iqr_acidity = q3_acidity - q1_acidity
# Skewness acidity
skew_acidity = df["Acidity"].skew()
# Kurtosis acidity
kurt_acidity = df["Acidity"].kurt()
```

```
# Acidity dataframe
df_acidity = pd.DataFrame({
    "Implemented": [implemented_mean_acidity, implemented_median_acidity, implement
    "Numpy": [mean_acidity, median_acidity, mode_acidity, std_acidity, var_acidity,
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mir

df_acidity
```

Out[ ]:

	Implemented	Numpy
Mean	8.014830	8.014830
Median	8.005347	8.005347
Mode	4.456118	4.456118
Standard Deviation	1.105781	1.105505
Variance	1.222752	1.222141
Range	6.962518	6.962518
Minimum	4.456118	4.456118
Maximum	11.418636	11.418636
Q1	7.259942	7.259942
Q3	8.758361	8.758361
IQR	1.498418	1.498418
Skewness	0.056708	0.056793
Kurtosis	-0.152615	-0.147134

In [ ]:

```
# Weight, bersifat numerik
weight = df["Weight"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean weight
implemented_mean_weight = mean(weight)
# Median weight
implemented_median_weight = median(weight)
# Mode weight
implemented_mode_weight = mode(weight)
# Standard deviation weight
implemented_std_weight = std_dev(weight)
# Variance weight
implemented_var_weight = variance(weight)
# Range weight
implemented_range_weight = data_range(weight)
# Minimum weight
implemented_min_weight = min(weight)
# Maximum weight
implemented_max_weight = max(weight)
# Q1 weight
implemented_q1_weight = quartile(weight, 1)
# Q3 weight
implemented_q3_weight = quartile(weight, 3)
# IQR weight
implemented_iqr_weight = iqr(weight)
# Skewness weight
implemented_skew_weight = skewness(weight)
# Kurtosis weight
```

```
implemented_kurt_weight = kurtosis(weight)

# Menggunakan numpy
# Mean weight
mean_weight = np.mean(weight)
# Median weight
median_weight = np.median(weight)
# Mode weight
mode_weight = df["Weight"].mode().values[0]
# Standard deviation weight
std_weight = np.std(weight)
# Variance weight
var_weight = np.var(weight)
# Range weight
range_weight = np.ptp(weight)
# Minimum weight
min_weight = np.min(weight)
# Maximum weight
max_weight = np.max(weight)
# Q1 weight
q1_weight = np.percentile(weight, 25)
# Q3 weight
q3_weight = np.percentile(weight, 75)
# IQR weight
iqr_weight = q3_weight - q1_weight
# Skewness weight
skew_weight = df["Weight"].skew()
# Kurtosis weight
kurt_weight = df["Weight"].kurt()

# Weight dataframe
df_weight = pd.DataFrame({
    "Implemented": [implemented_mean_weight, implemented_median_weight, implementec
    "Numpy": [mean_weight, median_weight, mode_weight, std_weight, var_weight, rang
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mir

df_weight
```

Out[ ]:

	Implemented	Numpy
Mean	150.011549	150.011549
Median	150.022865	150.022865
Mode	146.060922	146.060922
Standard Deviation	1.194980	1.194681
Variance	1.427977	1.427263
Range	8.009448	8.009448
Minimum	146.060922	146.060922
Maximum	154.070370	154.070370
Q1	149.227116	149.227116
Q3	150.827613	150.827613
IQR	1.600497	1.600497
Skewness	-0.084640	-0.084767
Kurtosis	0.018885	0.024967

In [ ]:

```
# Length, bersifat numerik
length = df["Length"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean length
implemented_mean_length = mean(length)
# Median length
implemented_median_length = median(length)
# Mode length
implemented_mode_length = mode(length)
# Standard deviation length
implemented_std_length = std_dev(length)
# Variance length
implemented_var_length = variance(length)
# Range length
implemented_range_length = data_range(length)
# Minimum length
implemented_min_length = min(length)
# Maximum length
implemented_max_length = max(length)
# Q1 length
implemented_q1_length = quartile(length, 1)
# Q3 length
implemented_q3_length = quartile(length, 3)
# IQR length
implemented_iqr_length = iqr(length)
# Skewness length
implemented_skew_length = skewness(length)
# Kurtosis length
implemented_kurt_length = kurtosis(length)

# Menggunakan numpy
# Mean length
mean_length = np.mean(length)
# Median length
median_length = np.median(length)
# Mode length
mode_length = df["Length"].mode().values[0]
# Standard deviation length
std_length = np.std(length)
# Variance length
var_length = np.var(length)
# Range length
range_length = np.ptp(length)
# Minimum length
min_length = np.min(length)
# Maximum length
max_length = np.max(length)
# Q1 length
q1_length = np.percentile(length, 25)
# Q3 length
q3_length = np.percentile(length, 75)
# IQR length
iqr_length = q3_length - q1_length
# Skewness length
skew_length = df["Length"].skew()
# Kurtosis length
kurt_length = df["Length"].kurt()

# Length dataframe
df_length = pd.DataFrame({
    "Implemented": [implemented_mean_length, implemented_median_length, implementec
    "Numpy": [mean_length, median_length, mode_length, std_length, var_length, rang
```

```
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mir

df_length
```

Out[ ]:

	Implemented	Numpy
Mean	49.950434	49.950434
Median	49.923682	49.923682
Mode	46.418052	46.418052
Standard Deviation	0.894599	0.894375
Variance	0.800307	0.799907
Range	6.647099	6.647099
Minimum	46.418052	46.418052
Maximum	53.065151	53.065151
Q1	49.346508	49.346508
Q3	50.572027	50.572027
IQR	1.225519	1.225519
Skewness	0.026838	0.026878
Kurtosis	-0.059357	-0.053550

In [ ]:

```
# Appearance, bersifat numerik
appearance = df["Appearance"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean appearance
implemented_mean_appearance = mean(appearance)
# Median appearance
implemented_median_appearance = median(appearance)
# Mode appearance
implemented_mode_appearance = mode(appearance)
# Standard deviation appearance
implemented_std_appearance = std_dev(appearance)
# Variance appearance
implemented_var_appearance = variance(appearance)
# Range appearance
implemented_range_appearance = data_range(appearance)
# Minimum appearance
implemented_min_appearance = min(appearance)
# Maximum appearance
implemented_max_appearance = max(appearance)
# Q1 appearance
implemented_q1_appearance = quartile(appearance, 1)
# Q3 appearance
implemented_q3_appearance = quartile(appearance, 3)
# IQR appearance
implemented_iqr_appearance = iqr(appearance)
# Skewness appearance
implemented_skew_appearance = skewness(appearance)
# Kurtosis appearance
implemented_kurt_appearance = kurtosis(appearance)

# Menggunakan numpy
# Mean appearance
mean_appearance = np.mean(appearance)
```

```
# Median appearance
median_appearance = np.median(appearance)
# Mode appearance
mode_appearance = df["Appearance"].mode().values[0]
# Standard deviation appearance
std_appearance = np.std(appearance)
# Variance appearance
var_appearance = np.var(appearance)
# Range appearance
range_appearance = np.ptp(appearance)
# Minimum appearance
min_appearance = np.min(appearance)
# Maximum appearance
max_appearance = np.max(appearance)
# Q1 appearance
q1_appearance = np.percentile(appearance, 25)
# Q3 appearance
q3_appearance = np.percentile(appearance, 75)
# IQR appearance
iqr_appearance = q3_appearance - q1_appearance
# Skewness appearance
skew_appearance = df["Appearance"].skew()
# Kurtosis appearance
kurt_appearance = df["Appearance"].kurt()

# Appearance dataframe
df_appearance = pd.DataFrame({
    "Implemented": [implemented_mean_appearance, implemented_median_appearance, imp
    "Numpy": [mean_appearance, median_appearance, mode_appearance, std_appearance,
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mi

df_appearance
```

Out[ ]:

	Implemented	Numpy
Mean	4.965595	4.965595
Median	4.979534	4.979534
Mode	1.775864	1.775864
Standard Deviation	1.014863	1.014609
Variance	1.029946	1.029431
Range	6.458104	6.458104
Minimum	1.775864	1.775864
Maximum	8.233968	8.233968
Q1	4.258210	4.258210
Q3	5.653875	5.653875
IQR	1.395665	1.395665
Skewness	-0.035336	-0.035389
Kurtosis	-0.008176	-0.002189

In [ ]:

```
# Tannin, bersifat numerik
tannin = df["Tannin"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean tannin
```

```
implemented_mean_tannin = mean(tannin)
# Median tannin
implemented_median_tannin = median(tannin)
# Mode tannin
implemented_mode_tannin = mode(tannin)
# Standard deviation tannin
implemented_std_tannin = std_dev(tannin)
# Variance tannin
implemented_var_tannin = variance(tannin)
# Range tannin
implemented_range_tannin = data_range(tannin)
# Minimum tannin
implemented_min_tannin = min(tannin)
# Maximum tannin
implemented_max_tannin = max(tannin)
# Q1 tannin
implemented_q1_tannin = quartile(tannin, 1)
# Q3 tannin
implemented_q3_tannin = quartile(tannin, 3)
# IQR tannin
implemented_iqr_tannin = iqr(tannin)
# Skewness tannin
implemented_skew_tannin = skewness(tannin)
# Kurtosis tannin
implemented_kurt_tannin = kurtosis(tannin)

# Menggunakan numpy
# Mean tannin
mean_tannin = np.mean(tannin)
# Median tannin
median_tannin = np.median(tannin)
# Mode tannin
mode_tannin = df["Tannin"].mode().values[0]
# Standard deviation tannin
std_tannin = np.std(tannin)
# Variance tannin
var_tannin = np.var(tannin)
# Range tannin
range_tannin = np.ptp(tannin)
# Minimum tannin
min_tannin = np.min(tannin)
# Maximum tannin
max_tannin = np.max(tannin)
# Q1 tannin
q1_tannin = np.percentile(tannin, 25)
# Q3 tannin
q3_tannin = np.percentile(tannin, 75)
# IQR tannin
iqr_tannin = q3_tannin - q1_tannin
# Skewness tannin
skew_tannin = df["Tannin"].skew()
# Kurtosis tannin
kurt_tannin = df["Tannin"].kurt()

# Tannin dataframe
df_tannin = pd.DataFrame({
    "Implemented": [implemented_mean_tannin, implemented_median_tannin, implementec
    "Numpy": [mean_tannin, median_tannin, mode_tannin, std_tannin, var_tannin, rang
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mir

df_tannin
```

Out[ ]:

	Implemented	Numpy
Mean	7.965435	7.965435
Median	8.022448	8.022448
Mode	4.291274	4.291274
Standard Deviation	1.217188	1.216883
Variance	1.481546	1.480805
Range	8.124904	8.124904
Minimum	4.291274	4.291274
Maximum	12.416177	12.416177
Q1	7.167241	7.167241
Q3	8.792184	8.792184
IQR	1.624943	1.624943
Skewness	-0.066053	-0.066152
Kurtosis	0.060122	0.066349

In [ ]:

```
# Ripeness, bersifat numerik
ripeness = df["Ripeness"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean ripeness
implemented_mean_ripeness = mean(ripeness)
# Median ripeness
implemented_median_ripeness = median(ripeness)
# Mode ripeness
implemented_mode_ripeness = mode(ripeness)
# Standard deviation ripeness
implemented_std_ripeness = std_dev(ripeness)
# Variance ripeness
implemented_var_ripeness = variance(ripeness)
# Range ripeness
implemented_range_ripeness = data_range(ripeness)
# Minimum ripeness
implemented_min_ripeness = min(ripeness)
# Maximum ripeness
implemented_max_ripeness = max(ripeness)
# Q1 ripeness
implemented_q1_ripeness = quartile(ripeness, 1)
# Q3 ripeness
implemented_q3_ripeness = quartile(ripeness, 3)
# IQR ripeness
implemented_iqr_ripeness = iqr(ripeness)
# Skewness ripeness
implemented_skew_ripeness = skewness(ripeness)
# Kurtosis ripeness
implemented_kurt_ripeness = kurtosis(ripeness)

# Menggunakan numpy
# Mean ripeness
mean_ripeness = np.mean(ripeness)
# Median ripeness
median_ripeness = np.median(ripeness)
# Mode ripeness
mode_ripeness = df["Ripeness"].mode().values[0]
```

```
# Standard deviation ripeness
std_ripeness = np.std(ripeness)
# Variance ripeness
var_ripeness = np.var(ripeness)
# Range ripeness
range_ripeness = np.ptp(ripeness)
# Minimum ripeness
min_ripeness = np.min(ripeness)
# Maximum ripeness
max_ripeness = np.max(ripeness)
# Q1 ripeness
q1_ripeness = np.percentile(ripeness, 25)
# Q3 ripeness
q3_ripeness = np.percentile(ripeness, 75)
# IQR ripeness
iqr_ripeness = q3_ripeness - q1_ripeness
# Skewness ripeness
skew_ripeness = df["Ripeness"].skew()
# Kurtosis ripeness
kurt_ripeness = df["Ripeness"].kurt()

# Ripeness dataframe
df_ripeness = pd.DataFrame({
    "Implemented": [implemented_mean_ripeness, implemented_median_ripeness, impleme
    "Numpy": [mean_ripeness, median_ripeness, mode_ripeness, std_ripeness, var_ripe
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mir

df_ripeness
```

Out[ ]:

	Implemented	Numpy
Mean	6.743434	6.743434
Median	6.667618	6.667618
Mode	4.862560	4.862560
Standard Deviation	0.680320	0.680150
Variance	0.462836	0.462604
Range	4.619506	4.619506
Minimum	4.862560	4.862560
Maximum	9.482066	9.482066
Q1	6.268258	6.268258
Q3	7.164813	7.164813
IQR	0.896555	0.896555
Skewness	0.494854	0.495597
Kurtosis	0.271235	0.278203

In [ ]:

```
# Sweetness, bersifat numerik
sweetness = df["Sweetness"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean sweetness
mean_sweetness = mean(sweetness)
# Median sweetness
median_sweetness = median(sweetness)
# Mode sweetness
```

```
mode_sweetness = mode(sweetness)
# Standard deviation sweetness
std_sweetness = std_dev(sweetness)
# Variance sweetness
var_sweetness = variance(sweetness)
# Range sweetness
range_sweetness = data_range(sweetness)
# Minimum sweetness
min_sweetness = min(sweetness)
# Maximum sweetness
max_sweetness = max(sweetness)
# Q1 sweetness
q1_sweetness = quartile(sweetness, 1)
# Q3 sweetness
q3_sweetness = quartile(sweetness, 3)
# IQR sweetness
iqr_sweetness = iqr(sweetness)
# Skewness sweetness
skew_sweetness = skewness(sweetness)
# Kurtosis sweetness
kurt_sweetness = kurtosis(sweetness)

# Menggunakan numpy
# Mean sweetness
mean_sweetness_np = np.mean(sweetness)
# Median sweetness
median_sweetness_np = np.median(sweetness)
# Mode sweetness
mode_sweetness_np = df["Sweetness"].mode().values[0]
# Standard deviation sweetness
std_sweetness_np = np.std(sweetness)
# Variance sweetness
var_sweetness_np = np.var(sweetness)
# Range sweetness
range_sweetness_np = np.ptp(sweetness)
# Minimum sweetness
min_sweetness_np = np.min(sweetness)
# Maximum sweetness
max_sweetness_np = np.max(sweetness)
# Q1 sweetness
q1_sweetness_np = np.percentile(sweetness, 25)
# Q3 sweetness
q3_sweetness_np = np.percentile(sweetness, 75)
# IQR sweetness
iqr_sweetness_np = q3_sweetness_np - q1_sweetness_np
# Skewness sweetness
skew_sweetness_np = df["Sweetness"].skew()
# Kurtosis sweetness
kurt_sweetness_np = df["Sweetness"].kurt()

# Sweetness dataframe
df_sweetness = pd.DataFrame({
    "Custom": [mean_sweetness, median_sweetness, mode_sweetness, std_sweetness, var
    "Numpy": [mean_sweetness_np, median_sweetness_np, mode_sweetness_np, std_sweetn
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mi

df_sweetness
```

Out[ ]:

	Custom	Numpy
Mean	6.226319	6.226319
Median	6.312819	6.312819
Mode	3.033193	3.033193
Standard Deviation	0.662980	0.662814
Variance	0.439543	0.439323
Range	4.645496	4.645496
Minimum	3.033193	3.033193
Maximum	7.678689	7.678689
Q1	5.808028	5.808028
Q3	6.714660	6.714660
IQR	0.906632	0.906632
Skewness	-0.662696	-0.663692
Kurtosis	0.487390	0.495115

In [ ]:

```python
# Country_of_Origin, bersifat string
country_of_origin = df["Country_of_Origin"].values

# Menggunakan fungsi statistik yang telah dibuat
# Unique values country_of_origin
implemented_unique_country_of_origin = unique(country_of_origin)
# Proportion country_of_origin
implemented_proportion_country_of_origin = proportion(country_of_origin)

# Menggunakan numpy
# Unique values country_of_origin
unique_country_of_origin = df["Country_of_Origin"].unique()
# Proportion country_of_origin
proportion_country_of_origin = df["Country_of_Origin"].value_counts(normalize=True)

# Implemented
print("Unique values country_of_origin (Implemented):")
for i in implemented_unique_country_of_origin:
    print(i)
print()

# Numpy
print("Unique values country_of_origin (Numpy):")
for i in unique_country_of_origin:
    print(i)
print()

# Merge dataframe
df_country_of_origin = pd.DataFrame({
    "Implemented": implemented_proportion_country_of_origin[:, 1],
    "Numpy": proportion_country_of_origin.values
}, index=proportion_country_of_origin.index)

df_country_of_origin
```

```
Unique values country_of_origin (Implemented):
Costa Rica
Colombia
Ecuador
undefined

Unique values country_of_origin (Numpy):
Costa Rica
Colombia
Ecuador
undefined
```

Out[ ]:

	Implemented	Numpy
Country_of_Origin		
Ecuador	0.5605	0.5605
Costa Rica	0.285	0.2850
Colombia	0.153	0.1530
undefined	0.0015	0.0015

In [ ]:

```python
# Firmness, bersifat numerik
firmness = df["Firmness"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean firmness
mean_firmness = mean(firmness)
# Median firmness
median_firmness = median(firmness)
# Mode firmness
mode_firmness = mode(firmness)
# Standard deviation firmness
std_firmness = std_dev(firmness)
# Variance firmness
var_firmness = variance(firmness)
# Range firmness
range_firmness = data_range(firmness)
# Minimum firmness
min_firmness = min(firmness)
# Maximum firmness
max_firmness = max(firmness)
# Q1 firmness
q1_firmness = quartile(firmness, 1)
# Q3 firmness
q3_firmness = quartile(firmness, 3)
# IQR firmness
iqr_firmness = iqr(firmness)
# Skewness firmness
skew_firmness = skewness(firmness)
# Kurtosis firmness
kurt_firmness = kurtosis(firmness)

# Menggunakan numpy
# Mean firmness
mean_firmness_np = np.mean(firmness)
# Median firmness
median_firmness_np = np.median(firmness)
# Mode firmness
mode_firmness_np = df["Firmness"].mode().values[0]
# Standard deviation firmness
std_firmness_np = np.std(firmness)
```

```
# Variance firmness
var_firmness_np = np.var(firmness)
# Range firmness
range_firmness_np = np.ptp(firmness)
# Minimum firmness
min_firmness_np = np.min(firmness)
# Maximum firmness
max_firmness_np = np.max(firmness)
# Q1 firmness
q1_firmness_np = np.percentile(firmness, 25)
# Q3 firmness
q3_firmness_np = np.percentile(firmness, 75)
# IQR firmness
iqr_firmness_np = q3_firmness_np - q1_firmness_np
# Skewness firmness
skew_firmness_np = df["Firmness"].skew()
# Kurtosis firmness
kurt_firmness_np = df["Firmness"].kurt()

# Firmness dataframe
df_firmness = pd.DataFrame({
    "Custom": [mean_firmness, median_firmness, mode_firmness, std_firmness, var_fir
    "Numpy": [mean_firmness_np, median_firmness_np, mode_firmness_np, std_firmness_
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mir

df_firmness
```

Out[ ]:

	Custom	Numpy
Mean	0.507790	0.507790
Median	0.515483	0.515483
Mode	0.000254	0.000254
Standard Deviation	0.292226	0.292153
Variance	0.085396	0.085353
Range	1.999746	1.999746
Minimum	0.000254	0.000254
Maximum	2.000000	2.000000
Q1	0.254351	0.254351
Q3	0.758786	0.758786
IQR	0.504436	0.504436
Skewness	0.024836	0.024873
Kurtosis	-0.907732	-0.904900

In [ ]:

```
# Grade, bersifat string
grade = df["Grade"].values

# Menggunakan fungsi statistik yang telah dibuat
# Unique values grade
implemented_unique_grade = unique(grade)
# Proportion grade
implemented_proportion_grade = proportion(grade)

# Menggunakan numpy
# Unique values grade
```

```
unique_grade = df["Grade"].unique()
# Proportion grade
proportion_grade = df["Grade"].value_counts(normalize=True)

# Implemented
print("Unique values grade (Implemented):")
for i in implemented_unique_grade:
    print(i)
print()

# Numpy
print("Unique values grade (Numpy):")
for i in unique_grade:
    print(i)
print()

# Merge dataframe
df_grade = pd.DataFrame({
    "Implemented": implemented_proportion_grade[:, 1],
    "Numpy": proportion_grade.values
}, index=proportion_grade.index)

df_grade
```

```
Unique values grade (Implemented):
A
C
B

Unique values grade (Numpy):
A
C
B
```

Out[ ]:

	Implemented	Numpy
Grade		
A	0.3415	0.3415
C	0.339	0.3390
B	0.3195	0.3195

In [ ]:

```
# Price, bersifat numerik
price = df["Price"].values

# Menggunakan fungsi statistik yang telah dibuat
# Mean price
mean_price = mean(price)
# Median price
median_price = median(price)
# Mode price
mode_price = mode(price)
# Standard deviation price
std_price = std_dev(price)
# Variance price
var_price = variance(price)
# Range price
range_price = data_range(price)
# Minimum price
min_price = min(price)
# Maximum price
max_price = max(price)
```

```
# Q1 price
q1_price = quartile(price, 1)
# Q3 price
q3_price = quartile(price, 3)
# IQR price
iqr_price = iqr(price)
# Skewness price
skew_price = skewness(price)
# Kurtosis price
kurt_price = kurtosis(price)

# Menggunakan numpy
# Mean price
mean_price_np = np.mean(price)
# Median price
median_price_np = np.median(price)
# Mode price
mode_price_np = df["Price"].mode().values[0]
# Standard deviation price
std_price_np = np.std(price)
# Variance price
var_price_np = np.var(price)
# Range price
range_price_np = np.ptp(price)
# Minimum price
min_price_np = np.min(price)
# Maximum price
max_price_np = np.max(price)
# Q1 price
q1_price_np = np.percentile(price, 25)
# Q3 price
q3_price_np = np.percentile(price, 75)
# IQR price
iqr_price_np = q3_price_np - q1_price_np
# Skewness price
skew_price_np = df["Price"].skew()
# Kurtosis price
kurt_price_np = df["Price"].kurt()

# Price dataframe
df_price = pd.DataFrame({
    "Custom": [mean_price, median_price, mode_price, std_price, var_price, range_pr
    "Numpy": [mean_price_np, median_price_np, mode_price_np, std_price_np, var_pric
}, index=["Mean", "Median", "Mode", "Standard Deviation", "Variance", "Range", "Mir

df_price
```

Out[ ]:

	Custom	Numpy
Mean	19969.669241	19969.669241
Median	19999.508312	19999.508312
Mode	0.000000	0.000000
Standard Deviation	777.347464	777.153103
Variance	604269.080280	603966.945740
Range	20282.431062	20282.431062
Minimum	-1.000000	-1.000000
Maximum	20281.431062	20281.431062
Q1	19953.093529	19953.093529
Q3	20047.301949	20047.301949
IQR	94.208419	94.208419
Skewness	-25.431046	-25.469237
Kurtosis	650.345912	652.633188

2. Apakah pada data tersebut terdapat outlier? Jika ya, dapatkah anda menanganinya? Jelaskan apa yang umumnya dilakukan untuk menangani outlier.

Data memiliki outlier pada beberapa kolom. Dibawah akan dicetak semua kolom outlier dan juga masing-masing jumlahnya.

Data outlier ini dapat ditangani. Beberapa metode yang dapat digunakan untuk menangani outlier adalah dengan menghapus data outlier atau mengganti data outlier dengan nilai yang lebih masuk akal seperti mengganti dengan nilai mean, median, atau mode nya.

Dalam tugas ini, kami menangani data outlier dengan cara yang pertama, yaitu menghapus data outlier tersebut.

In [ ]:

```python
# Cari outlier untuk acidity
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR
acidity_lower_outlier_limit = quartile(acidity, 1) - 1.5 * iqr(acidity)
acidity_higher_outlier_limit = quartile(acidity, 3) + 1.5 * iqr(acidity)

print(f"Lower outlier limit: {acidity_lower_outlier_limit}")
print(f"Higher outlier limit: {acidity_higher_outlier_limit}")
print()

# Dataframe
cleaned_acidity = (df["Acidity"] >= acidity_lower_outlier_limit) & (df["Acidity"] <
df_outliers_acidity = df[(df["Acidity"] < acidity_lower_outlier_limit) | (df["Acidi

# Output
```

```
print(f"Ada {len(df_outliers_acidity)} outlier pada acidity, yaitu:")
df_outliers_acidity["Acidity"].to_frame()
```

```
Lower outlier limit: 5.012314896354701
Higher outlier limit: 11.005988281432417

Ada 12 outlier pada acidity, yaitu:
```

Out[ ]:

	Acidity
148	11.191852
209	11.119288
279	11.137342
289	11.024219
345	11.079811
349	11.418636
683	11.026875
819	4.897068
966	4.456118
1040	4.896538
1327	11.284712
1785	11.374194

In [ ]:

```
# Cari outlier untuk weight
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR
weight_lower_outlier_limit = quartile(weight, 1) - 1.5 * iqr(weight)
weight_higher_outlier_limit = quartile(weight, 3) + 1.5 * iqr(weight)

print(f"Lower outlier limit: {weight_lower_outlier_limit}")
print(f"Higher outlier limit: {weight_higher_outlier_limit}")
print()

# Dataframe
cleaned_weight = (df["Weight"] >= weight_lower_outlier_limit) & (df["Weight"] <= we
df_outliers_weight = df[(df["Weight"] < weight_lower_outlier_limit) | (df["Weight"]

# Output
print(f"Ada {len(df_outliers_weight)} outlier pada weight, yaitu:")
df_outliers_weight["Weight"].to_frame()
```

```
Lower outlier limit: 146.82637023654053
Higher outlier limit: 153.22835888037406

Ada 14 outlier pada weight, yaitu:
```

Out[ ]:

	Weight
44	146.535963
357	153.970493
386	146.376184
658	146.490788
677	146.444130
1059	154.070370
1116	146.603512
1133	146.496350
1159	146.126108
1269	153.285546
1412	146.812035
1793	146.060922
1898	146.533637
1959	153.599879

In [ ]:

```python
# Cari outlier untuk length
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR
length_lower_outlier_limit = quartile(length, 1) - 1.5 * iqr(length)
length_higher_outlier_limit = quartile(length, 3) + 1.5 * iqr(length)

print(f"Lower outlier limit: {length_lower_outlier_limit}")
print(f"Higher outlier limit: {length_higher_outlier_limit}")
print()

# Dataframe
cleaned_length = (df["Length"] >= length_lower_outlier_limit) & (df["Length"] <= le
df_outliers_length = df[(df["Length"] < length_lower_outlier_limit) | (df["Length"]

# Output
print(f"Ada {len(df_outliers_length)} outlier pada length, yaitu:")
df_outliers_length["Length"].to_frame()
```

```
Lower outlier limit: 47.5082285751469
Higher outlier limit: 52.410305852223885

Ada 12 outlier pada length, yaitu:
```

Out[ ]:

	Length
40	53.065151
446	52.413780
522	47.452026
637	52.543665
747	52.626968
792	47.313156
988	52.558423
1136	47.366597
1197	52.439588
1220	52.519990
1484	47.262146
1873	46.418052

In [ ]:

```
# Cari outlier untuk appearance
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR

appearance_lower_outlier_limit = quartile(appearance, 1) - 1.5 * iqr(appearance)
appearance_higher_outlier_limit = quartile(appearance, 3) + 1.5 * iqr(appearance)

print(f"Lower outlier limit: {appearance_lower_outlier_limit}")
print(f"Higher outlier limit: {appearance_higher_outlier_limit}")
print()

# Dataframe
cleaned_appearance = (df["Appearance"] >= appearance_lower_outlier_limit) & (df["Ap
df_outliers_appearance = df[(df["Appearance"] < appearance_lower_outlier_limit) | (

# Output
print(f"Ada {len(df_outliers_appearance)} outlier pada appearance, yaitu:")
df_outliers_appearance["Appearance"].to_frame()
```

```
Lower outlier limit: 2.1647113424403055
Higher outlier limit: 7.747373338498701

Ada 15 outlier pada appearance, yaitu:
```

Out[ ]:

	Appearance
143	8.233968
242	2.127349
328	7.927957
594	7.842696
615	2.007510
1064	7.848426
1067	7.817189
1216	1.977268
1296	8.032614
1316	1.775864
1443	1.931581
1605	2.071613
1611	7.773449
1762	1.786403
1845	1.910726

In [ ]:

```
# Cari outlier untuk tannin
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR

tannin_lower_outlier_limit = quartile(tannin, 1) - 1.5 * iqr(tannin)
tannin_higher_outlier_limit = quartile(tannin, 3) + 1.5 * iqr(tannin)

print(f"Lower outlier limit: {tannin_lower_outlier_limit}")
print(f"Higher outlier limit: {tannin_higher_outlier_limit}")
print()

# Dataframe
cleaned_tannin = (df["Tannin"] >= tannin_lower_outlier_limit) & (df["Tannin"] <= ta
df_outliers_tannin = df[(df["Tannin"] < tannin_lower_outlier_limit) | (df["Tannin"]

# Output
print(f"Ada {len(df_outliers_tannin)} outlier pada tannin, yaitu:")
df_outliers_tannin["Tannin"].to_frame()
```

```
Lower outlier limit: 4.729826963771409
Higher outlier limit: 11.229598458769104

Ada 13 outlier pada tannin, yaitu:
```

Out[ ]:

	Tannin
217	11.273264
400	4.291274
576	4.709272
581	12.090781
610	4.629238
687	11.780068
1261	11.431587
1456	12.416177
1461	11.550949
1484	11.250187
1631	4.650028
1796	11.355590
1989	11.521227

In [ ]:

```
# Cari outlier untuk ripeness
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR

ripeness_lower_outlier_limit = quartile(ripeness, 1) - 1.5 * iqr(ripeness)
ripeness_higher_outlier_limit = quartile(ripeness, 3) + 1.5 * iqr(ripeness)

print(f"Lower outlier limit: {ripeness_lower_outlier_limit}")
print(f"Higher outlier limit: {ripeness_higher_outlier_limit}")
print()

# Dataframe
cleaned_ripeness = (df["Ripeness"] >= ripeness_lower_outlier_limit) & (df["Ripeness
df_outliers_ripeness = df[(df["Ripeness"] < ripeness_lower_outlier_limit) | (df["Ri

# Output
print(f"Ada {len(df_outliers_ripeness)} outlier pada ripeness, yaitu:")
df_outliers_ripeness["Ripeness"].to_frame()
```

```
Lower outlier limit: 4.923425290238341
Higher outlier limit: 8.509645135586311

Ada 26 outlier pada ripeness, yaitu:
```

Out[ ]:

	Ripeness
233	8.767843
270	8.991369
280	8.645577
371	8.676075
427	8.628959
559	8.527220
757	8.637225
765	8.530369
822	9.482066
890	8.637212
901	8.629589
1028	4.862560
1121	9.173803
1142	8.636351
1288	8.698339
1300	9.348371
1353	8.573482
1373	9.114434
1493	8.834792
1507	4.904725
1567	8.612570
1633	8.707027
1675	4.918675
1693	8.782708
1881	9.425643
1956	8.539070

In [ ]:

```
# Cari outlier untuk sweetness
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR

sweetness_lower_outlier_limit = quartile(sweetness, 1) - 1.5 * iqr(sweetness)
sweetness_higher_outlier_limit = quartile(sweetness, 3) + 1.5 * iqr(sweetness)

print(f"Lower outlier limit: {sweetness_lower_outlier_limit}")
print(f"Higher outlier limit: {sweetness_higher_outlier_limit}")
print()

# Dataframe
cleaned_sweetness = (df["Sweetness"] >= sweetness_lower_outlier_limit) & (df["Sweet
df_outliers_sweetness = df[(df["Sweetness"] < sweetness_lower_outlier_limit) | (df[

# Output
```

```
print(f"Ada {len(df_outliers_sweetness)} outlier pada sweetness, yaitu:")
df_outliers_sweetness["Sweetness"].to_frame()
```

```
Lower outlier limit: 4.448078990732531
Higher outlier limit: 8.074608633579198

Ada 21 outlier pada sweetness, yaitu:
```

Out[ ]:

	Sweetness
29	4.025152
128	4.363350
143	4.053357
172	3.954111
186	4.411304
232	4.136793
329	4.151006
351	4.220835
418	4.179858
469	3.429437
791	4.339535
804	3.795591
1156	3.599487
1160	4.299325
1178	4.095918
1191	4.413483
1226	3.033193
1472	4.380152
1559	4.363427
1716	4.412548
1762	4.131909

In [ ]:

```
# Cari outlier untuk country_of_origin
# Outlier outlier: undefined

# Dataframe
cleaned_country_of_origin = (df["Country_of_Origin"] != "undefined")
df_outliers_country_of_origin = df[df["Country_of_Origin"] == "undefined"]

# Output
print(f"Ada {len(df_outliers_country_of_origin)} outlier pada country_of_origin, ya
df_outliers_country_of_origin["Country_of_Origin"].to_frame()
```

```
Ada 3 outlier pada country_of_origin, yaitu:
```

Out[ ]:

	Country_of_Origin
402	undefined
824	undefined
1853	undefined

In [ ]:

```
# Cari outlier untuk firmness
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR

firmness_lower_outlier_limit = quartile(firmness, 1) - 1.5 * iqr(firmness)
firmness_higher_outlier_limit = quartile(firmness, 3) + 1.5 * iqr(firmness)

print(f"Lower outlier limit: {firmness_lower_outlier_limit}")
print(f"Higher outlier limit: {firmness_higher_outlier_limit}")
print()

# Dataframe
cleaned_firmness = (df["Firmness"] >= firmness_lower_outlier_limit) & (df["Firmness
df_outliers_firmness = df[(df["Firmness"] < firmness_lower_outlier_limit) | (df["Fi

# Output
print(f"Ada {len(df_outliers_firmness)} outlier pada firmness, yaitu:")
df_outliers_firmness["Firmness"].to_frame()
```

```
Lower outlier limit: -0.5023027956582491
Higher outlier limit: 1.5154393695714752

Ada 1 outlier pada firmness, yaitu:
```

Out[ ]:

	Firmness
283	2.0

In [ ]:

```
# Cari outlier untuk grade
# Outlier outlier: undefined

# Dataframe
cleaned_grade = (df["Grade"] != "undefined")
df_outliers_grade = df[df["Grade"] == "undefined"]

# Output
print(f"Ada {len(df_outliers_grade)} outlier pada grade, yaitu:")
df_outliers_grade["Grade"].to_frame()
```

```
Ada 0 outlier pada grade, yaitu:
```

Out[ ]:

Grade

In [ ]:

```
# Cari outlier untuk price
# Outlier outlier: X < Q1 - 1.5 * IQR atau X > Q3 + 1.5 * IQR

price_lower_outlier_limit = quartile(price, 1) - 1.5 * iqr(price)
price_higher_outlier_limit = quartile(price, 3) + 1.5 * iqr(price)

print(f"Lower outlier limit: {price_lower_outlier_limit}")
print(f"Higher outlier limit: {price_higher_outlier_limit}")
print()

# Dataframe
cleaned_price = (df["Price"] >= price_lower_outlier_limit) & (df["Price"] <= price_
df_outliers_price = df[(df["Price"] < price_lower_outlier_limit) | (df["Price"] > p
# Output
print(f"Ada {len(df_outliers_price)} outlier pada price, yaitu:")
df_outliers_price["Price"].to_frame()
```

```
Lower outlier limit: 19811.780900435893
Higher outlier limit: 20188.614577845517

Ada 18 outlier pada price, yaitu:
```

Out[ ]:

	Price
53	19803.813931
378	19781.569703
402	0.000000
689	19729.904103
690	19809.257798
759	20199.676334
789	20189.020997
832	19769.470304
873	19785.810537
964	19769.450553
995	19811.228690
1012	19759.846000
1095	19809.025516
1134	19763.590653
1294	-1.000000
1364	20281.431062
1474	19786.680740
1922	0.000000

In [ ]:

```
cleaned_data = cleaned_acidity & cleaned_weight & cleaned_length & cleaned_appearar
cleaned_data = df[cleaned_data]
print(f"Data yang sudah dibersihkan dari outlier: {len(cleaned_data)}")
cleaned_data.head()
```

```
Data yang sudah dibersihkan dari outlier: 1869
```

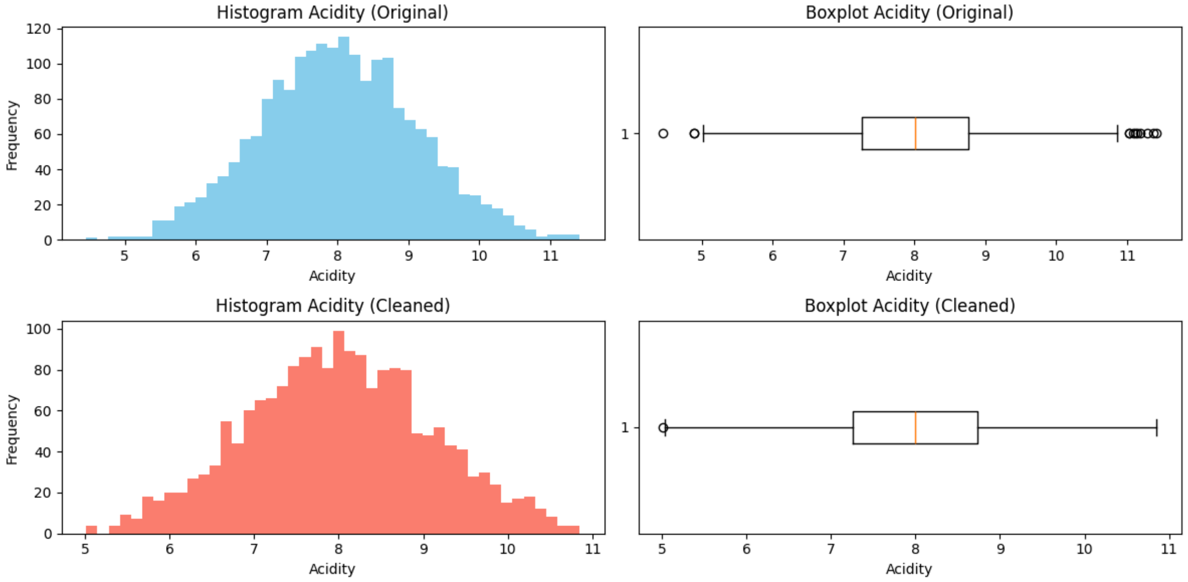
Out[ ]:

	Unnamed: 0	Acidity	Weight	Length	Appearance	Tannin	Ripeness	Sweetness	Count
0	0	5.977114	149.825704	49.249144	3.770162	8.092385	6.916558	6.763285	
1	1	8.625523	150.759254	50.048300	6.007516	7.400025	6.706338	6.481902	
2	2	8.813012	148.780694	49.865871	5.166949	6.861433	6.607327	5.702631	
3	3	7.496444	152.329626	49.676489	5.451806	7.342269	6.482970	6.265227	
4	4	6.885109	150.412228	50.526268	3.872441	7.630643	6.064423	6.856929	

3. Membuat Visualisasi plot distribusi. Berikan uraian penjelasan kondisi setiap kolom berdasarkan kedua plot tersebut. Jika numerik dapat dibuat dalam bentuk histogram dan box plot, dan jika string dengan histogram.

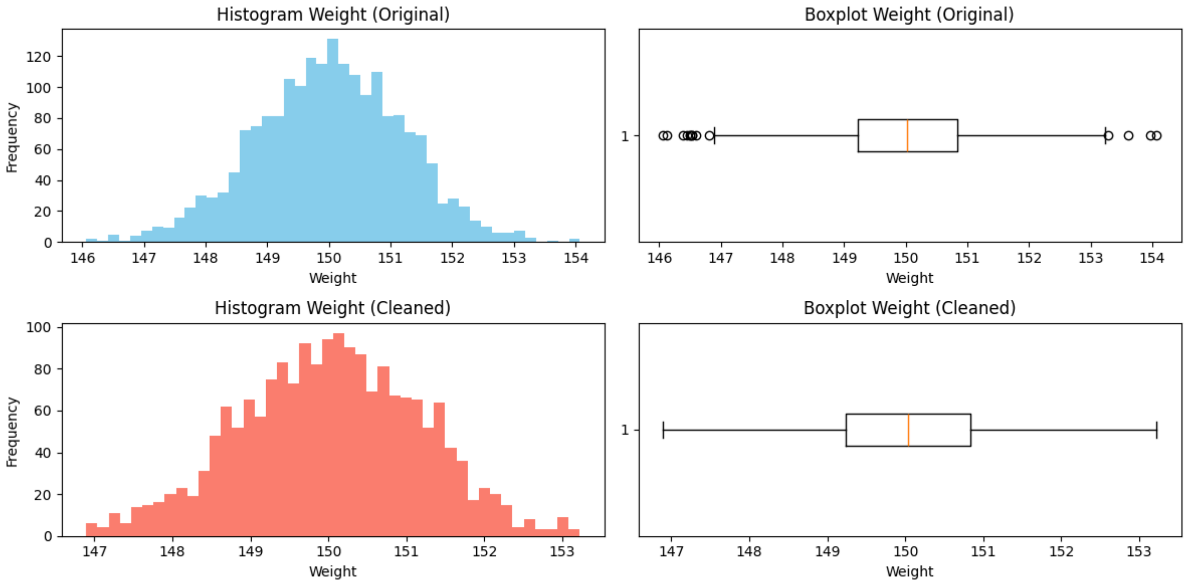
In [ ]:

```python
# Visualisasi Acidity
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram Acidity (Original)
original_acidity_bin = int(np.ceil(np.sqrt(len(acidity))))
ax1.hist(acidity, bins=original_acidity_bin, color="skyblue")
ax1.set_title("Histogram Acidity (Original)")
ax1.set_xlabel("Acidity")
ax1.set_ylabel("Frequency")

# Boxplot Acidity (Original)
ax2.boxplot(acidity, vert=False)
ax2.set_title("Boxplot Acidity (Original)")
ax2.set_xlabel("Acidity")

# Histogram Acidity (Cleaned)
cleaned_acidity = cleaned_data["Acidity"].values
cleaned_acidity_bin = int(np.ceil(np.sqrt(len(cleaned_acidity))))
ax3.hist(cleaned_acidity, bins=cleaned_acidity_bin, color="salmon")
ax3.set_title("Histogram Acidity (Cleaned)")
ax3.set_xlabel("Acidity")
ax3.set_ylabel("Frequency")

# Boxplot Acidity (Cleaned)
ax4.boxplot(cleaned_acidity, vert=False)
ax4.set_title("Boxplot Acidity (Cleaned)")
ax4.set_xlabel("Acidity")

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



In [ ]:

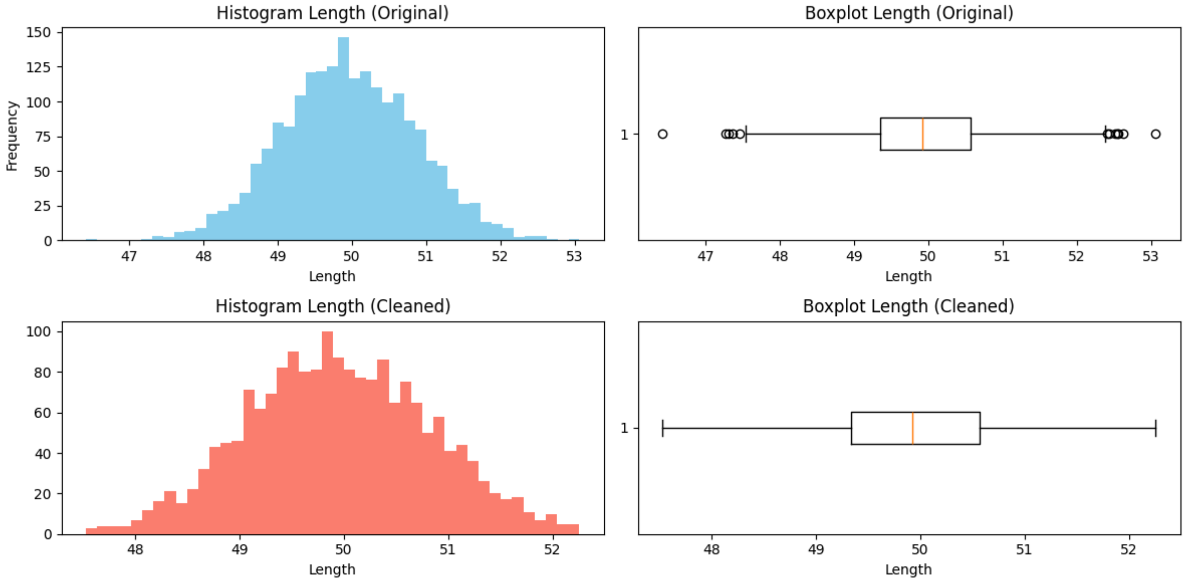
```python
# Visualisasi Weight
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_weight_bin = int(np.ceil(np.sqrt(len(weight))))
ax1.hist(weight, bins=original_weight_bin, color="skyblue")
ax1.set_title("Histogram Weight (Original)")
ax1.set_xlabel("Weight")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(weight, vert=False)
ax2.set_title("Boxplot Weight (Original)")
ax2.set_xlabel("Weight")

#  Histogram (Cleaned)
cleaned_weight = cleaned_data["Weight"].values
cleaned_weight_bin = int(np.ceil(np.sqrt(len(cleaned_weight))))
ax3.hist(cleaned_weight, bins=cleaned_weight_bin, color="salmon")
ax3.set_title("Histogram Weight (Cleaned)")
ax3.set_xlabel("Weight")
ax3.set_ylabel("Frequency")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_weight, vert=False)
ax4.set_title("Boxplot Weight (Cleaned)")
ax4.set_xlabel("Weight")

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



In [ ]:

```python
# Visualisasi Length
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_length_bin = int(np.ceil(np.sqrt(len(length))))
ax1.hist(length, bins=original_length_bin, color="skyblue")
ax1.set_title("Histogram Length (Original)")
ax1.set_xlabel("Length")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(length, vert=False)
ax2.set_title("Boxplot Length (Original)")
```

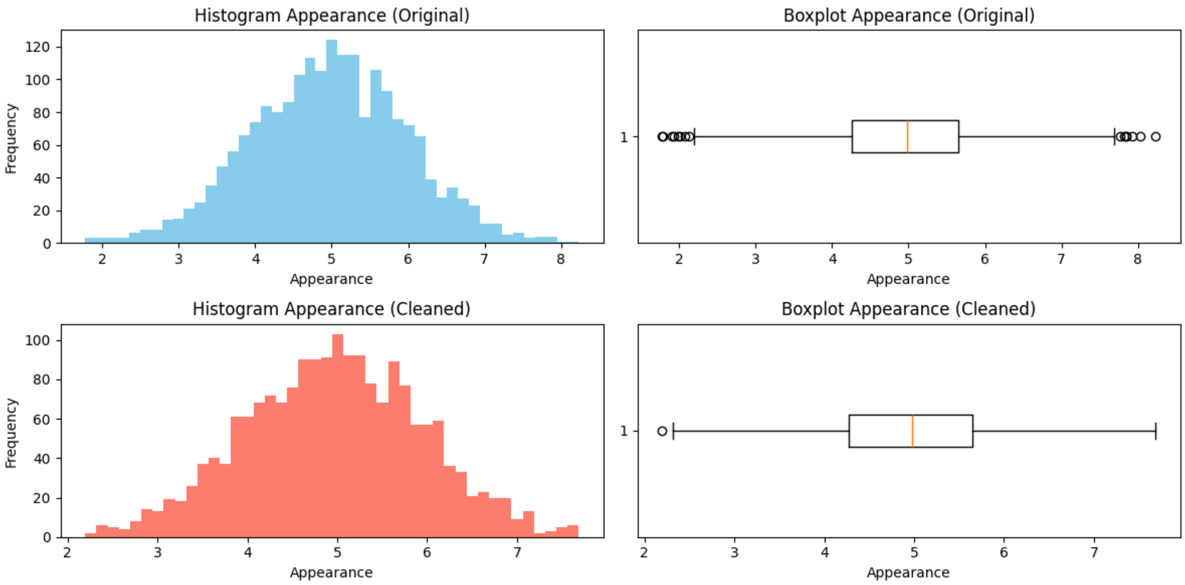
```
ax2.set_xlabel("Length")

# Histogram (Cleaned)
cleaned_length = cleaned_data["Length"].values
cleaned_length_bin = int(np.ceil(np.sqrt(len(cleaned_length))))
ax3.hist(cleaned_length, bins=cleaned_length_bin, color="salmon")
ax3.set_title("Histogram Length (Cleaned)")
ax3.set_xlabel("Length")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_length, vert=False)
ax4.set_title("Boxplot Length (Cleaned)")
ax4.set_xlabel("Length")

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



In [ ]:

```
# Visualisasi Appearance
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_appearance_bin = int(np.ceil(np.sqrt(len(appearance))))
ax1.hist(appearance, bins=original_appearance_bin, color="skyblue")
ax1.set_title("Histogram Appearance (Original)")
ax1.set_xlabel("Appearance")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(appearance, vert=False)
ax2.set_title("Boxplot Appearance (Original)")
ax2.set_xlabel("Appearance")

# Histogram (Cleaned)
cleaned_appearance = cleaned_data["Appearance"].values
cleaned_appearance_bin = int(np.ceil(np.sqrt(len(cleaned_appearance))))
ax3.hist(cleaned_appearance, bins=cleaned_appearance_bin, color="salmon")
ax3.set_title("Histogram Appearance (Cleaned)")
ax3.set_xlabel("Appearance")
ax3.set_ylabel("Frequency")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_appearance, vert=False)
ax4.set_title("Boxplot Appearance (Cleaned)")
ax4.set_xlabel("Appearance")
```

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



Histogram Appearance (Original)

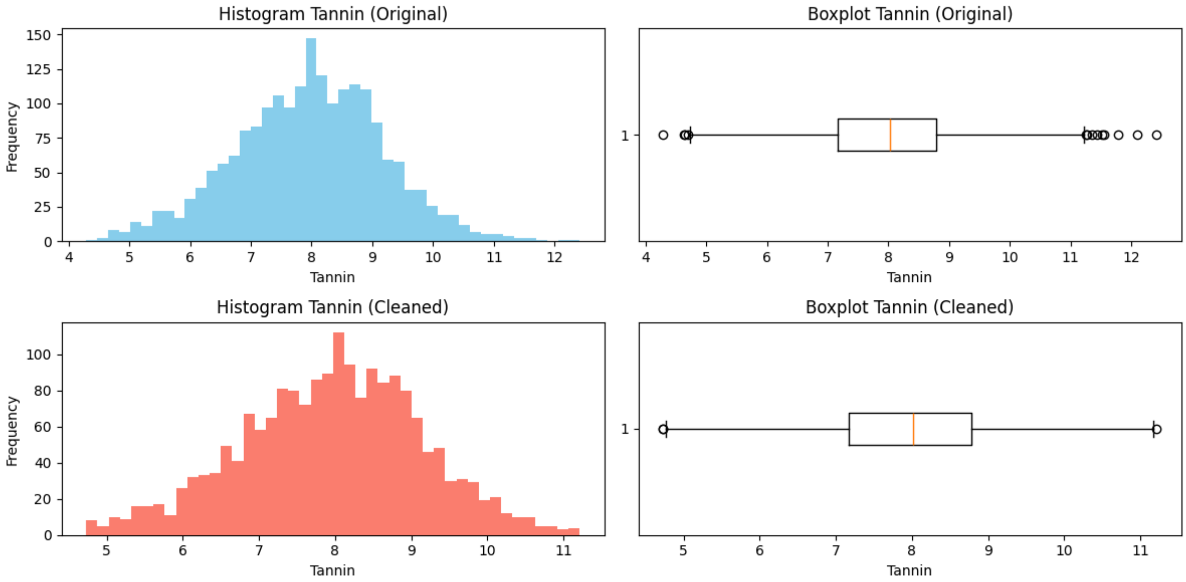
Boxplot Appearance (Original)

Histogram Appearance (Cleaned)

Boxplot Appearance (Cleaned)

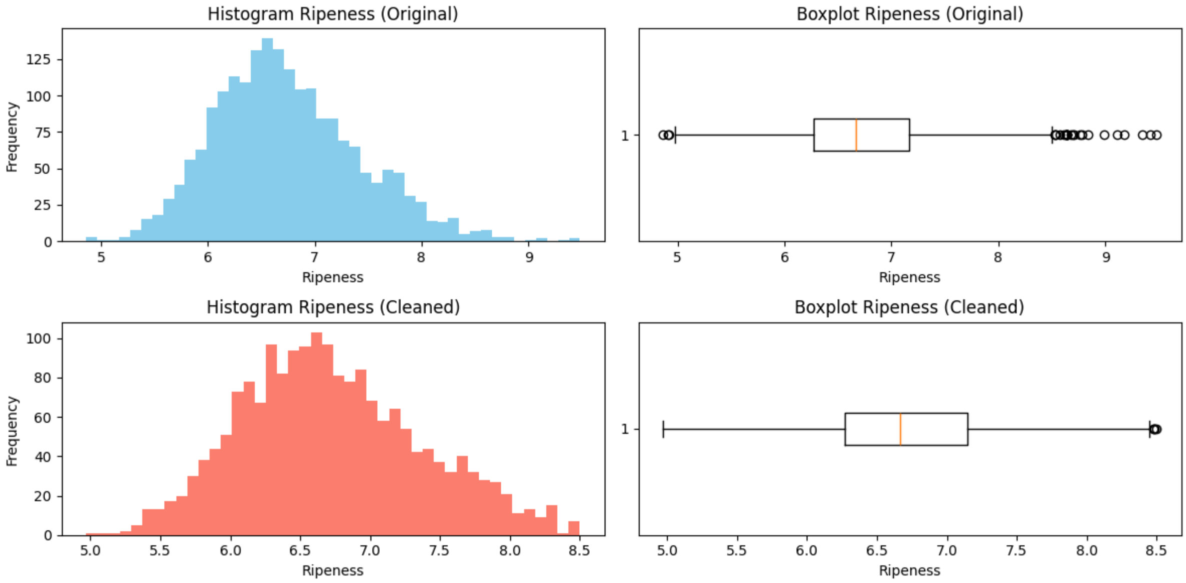
In [ ]:

```
# Visualisasi Tannin
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_tannin_bin = int(np.ceil(np.sqrt(len(tannin))))
ax1.hist(tannin, bins=original_tannin_bin, color="skyblue")
ax1.set_title("Histogram Tannin (Original)")
ax1.set_xlabel("Tannin")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(tannin, vert=False)
ax2.set_title("Boxplot Tannin (Original)")
ax2.set_xlabel("Tannin")

# Histogram (Cleaned)
cleaned_tannin = cleaned_data["Tannin"].values
cleaned_tannin_bin = int(np.ceil(np.sqrt(len(cleaned_tannin))))
ax3.hist(cleaned_tannin, bins=cleaned_acidity_bin, color="salmon")
ax3.set_title("Histogram Tannin (Cleaned)")
ax3.set_xlabel("Tannin")
ax3.set_ylabel("Frequency")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_tannin, vert=False)
ax4.set_title("Boxplot Tannin (Cleaned)")
ax4.set_xlabel("Tannin")

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



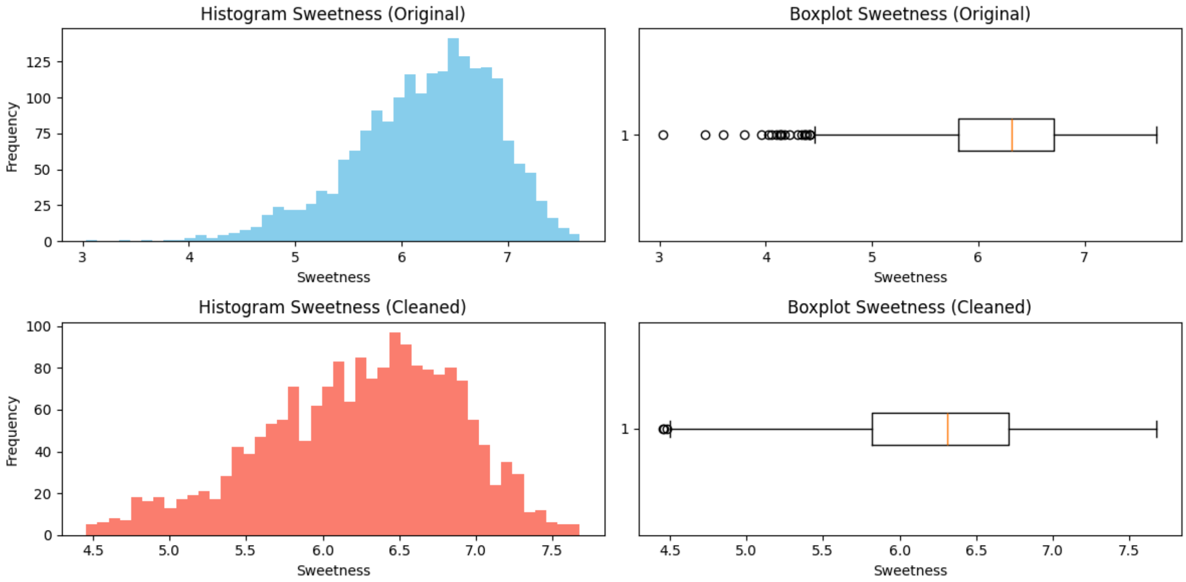
In [ ]:

```
# Visualisasi Ripeness
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_ripeness_bin = int(np.ceil(np.sqrt(len(ripeness))))
ax1.hist(ripeness, bins=original_ripeness_bin, color="skyblue")
ax1.set_title("Histogram Ripeness (Original)")
ax1.set_xlabel("Ripeness")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(ripeness, vert=False)
ax2.set_title("Boxplot Ripeness (Original)")
ax2.set_xlabel("Ripeness")

# Histogram (Cleaned)
cleaned_ripeness = cleaned_data["Ripeness"].values
cleaned_ripeness_bin = int(np.ceil(np.sqrt(len(cleaned_ripeness))))
ax3.hist(cleaned_ripeness, bins=cleaned_ripeness_bin, color="salmon")
ax3.set_title("Histogram Ripeness (Cleaned)")
ax3.set_xlabel("Ripeness")
ax3.set_ylabel("Frequency")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_ripeness, vert=False)
ax4.set_title("Boxplot Ripeness (Cleaned)")
ax4.set_xlabel("Ripeness")

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



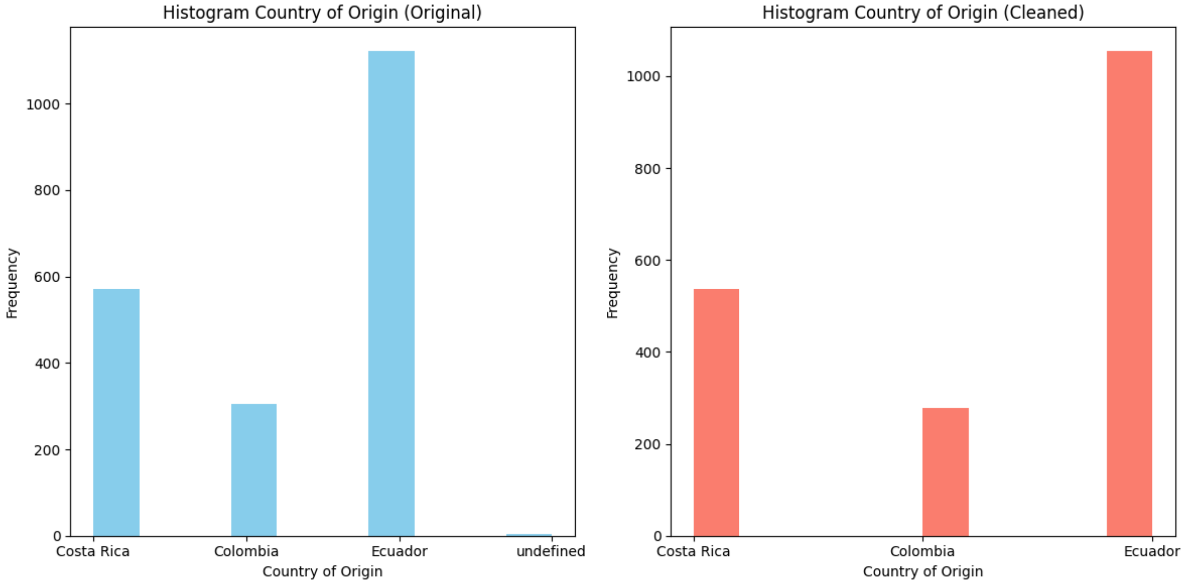
In [ ]:

```python
# Visualisasi Sweetness
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_sweetness_bin = int(np.ceil(np.sqrt(len(sweetness))))
ax1.hist(sweetness, bins=original_sweetness_bin, color="skyblue")
ax1.set_title("Histogram Sweetness (Original)")
ax1.set_xlabel("Sweetness")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(sweetness, vert=False)
ax2.set_title("Boxplot Sweetness (Original)")
ax2.set_xlabel("Sweetness")

# Histogram (Cleaned)
cleaned_sweetness = cleaned_data["Sweetness"].values
cleaned_sweetness_bin = int(np.ceil(np.sqrt(len(cleaned_sweetness))))
ax3.hist(cleaned_sweetness, bins=cleaned_sweetness_bin, color="salmon")
ax3.set_title("Histogram Sweetness (Cleaned)")
ax3.set_xlabel("Sweetness")
ax3.set_ylabel("Frequency")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_sweetness, vert=False)
ax4.set_title("Boxplot Sweetness (Cleaned)")
ax4.set_xlabel("Sweetness")

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



In [ ]:

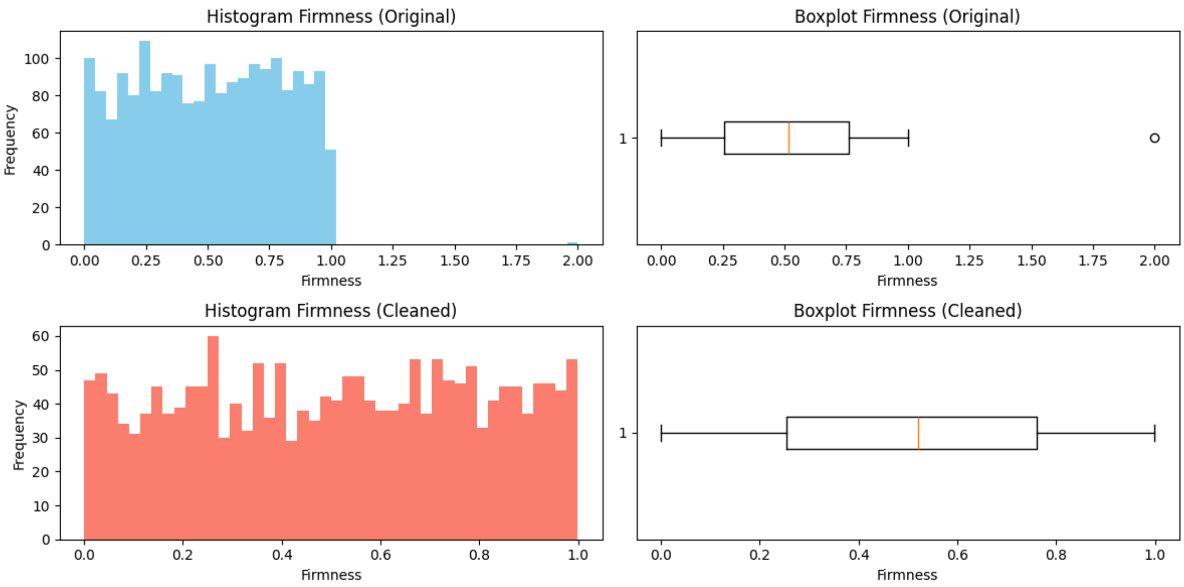
```
# Visualisasi Country_of_Origin (Cleaned)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Original bar chart
ax1.hist(country_of_origin, bins=10, color="skyblue")
ax1.set_title("Histogram Country of Origin (Original)")
ax1.set_xlabel("Country of Origin")
ax1.set_ylabel("Frequency")

# Cleaned
cleaned_country_origin = cleaned_data["Country_of_Origin"].values
ax2.hist(cleaned_country_origin, bins=10, color="salmon")
ax2.set_title("Histogram Country of Origin (Cleaned)")
ax2.set_xlabel("Country of Origin")
ax2.set_ylabel("Frequency")

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



In [ ]:

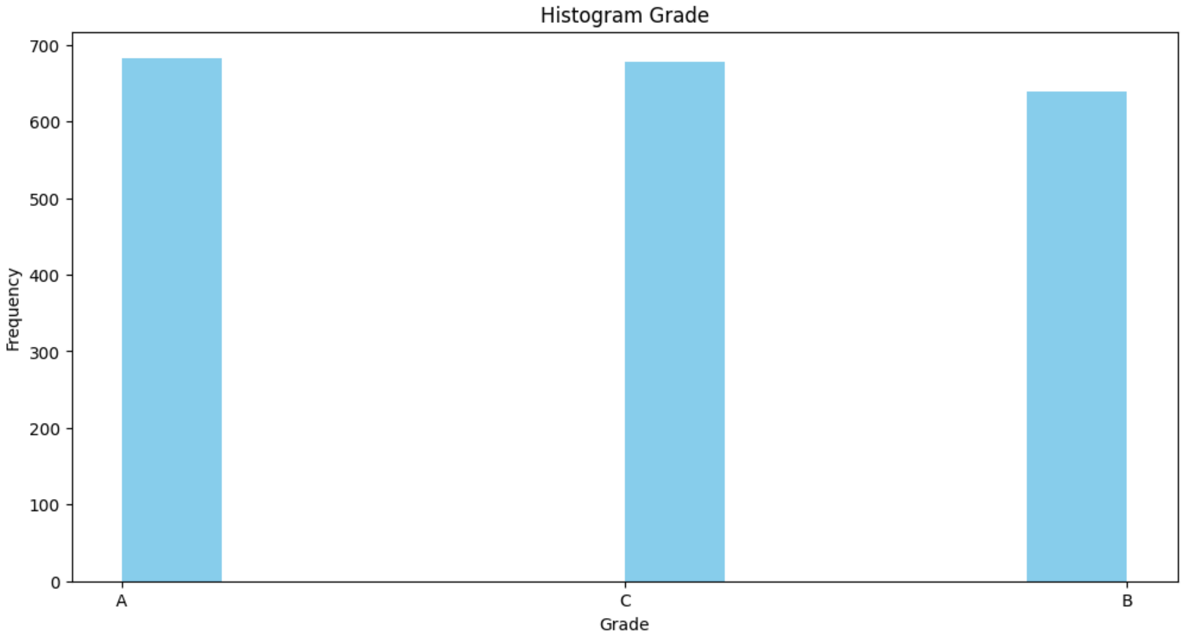
```
# Visualisasi Firmness
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_firmness_bin = int(np.ceil(np.sqrt(len(firmness))))
ax1.hist(firmness, bins=original_firmness_bin, color="skyblue")
```

```
ax1.set_title("Histogram Firmness (Original)")
ax1.set_xlabel("Firmness")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(firmness, vert=False)
ax2.set_title("Boxplot Firmness (Original)")
ax2.set_xlabel("Firmness")

# Histogram (Cleaned)
cleaned_firmness = cleaned_data["Firmness"].values
cleaned_firmness_bin = int(np.ceil(np.sqrt(len(cleaned_firmness))))
ax3.hist(cleaned_firmness, bins=cleaned_firmness_bin, color="salmon")
ax3.set_title("Histogram Firmness (Cleaned)")
ax3.set_xlabel("Firmness")
ax3.set_ylabel("Frequency")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_firmness, vert=False)
ax4.set_title("Boxplot Firmness (Cleaned)")
ax4.set_xlabel("Firmness")


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



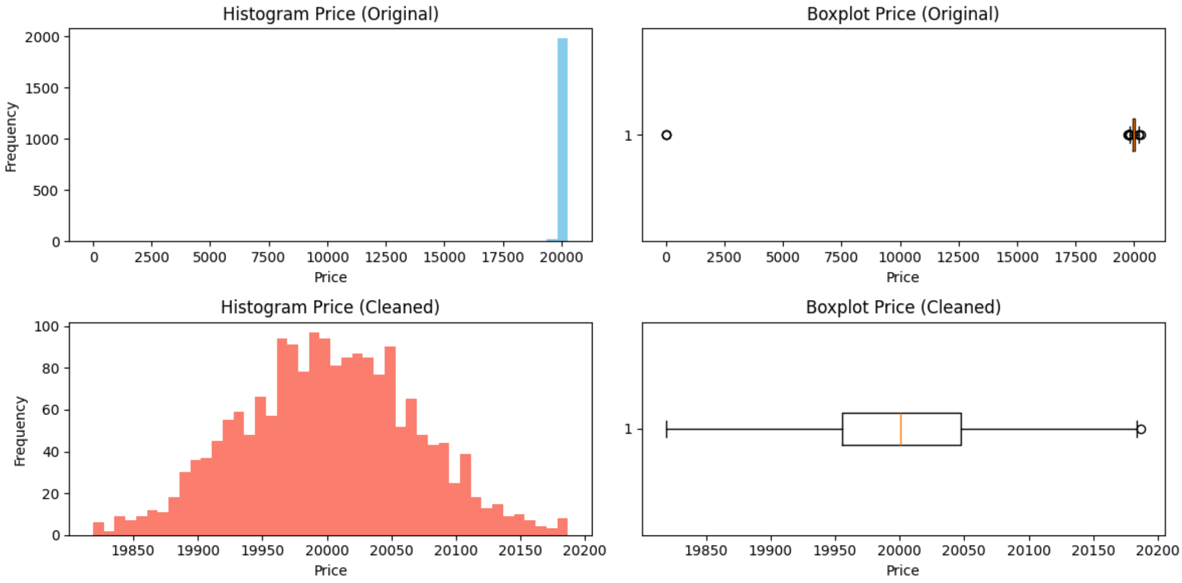
In [ ]:

```
# Visualisasi Grade
plt.figure(figsize=(12, 6))
plt.hist(grade, bins=10, color="skyblue")
plt.title("Histogram Grade")
plt.xlabel("Grade")
plt.ylabel("Frequency")
plt.show()
```



In [ ]:

```
# Visualisasi Price
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 6))

# Histogram (Original)
original_price_bin = int(np.ceil(np.sqrt(len(price))))
ax1.hist(price, bins=original_price_bin, color="skyblue")
ax1.set_title("Histogram Price (Original)")
ax1.set_xlabel("Price")
ax1.set_ylabel("Frequency")

# Boxplot (Original)
ax2.boxplot(price, vert=False)
ax2.set_title("Boxplot Price (Original)")
ax2.set_xlabel("Price")

# Histogram (Cleaned)
cleaned_price = cleaned_data["Price"].values
cleaned_price_bin = int(np.ceil(np.sqrt(len(cleaned_price))))
ax3.hist(cleaned_price, bins=cleaned_price_bin, color="salmon")
ax3.set_title("Histogram Price (Cleaned)")
ax3.set_xlabel("Price")
ax3.set_ylabel("Frequency")

# Boxplot (Cleaned)
ax4.boxplot(cleaned_price, vert=False)
ax4.set_title("Boxplot Price (Cleaned)")
ax4.set_xlabel("Price")

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



4. Menentukan distribusi setiap kolom numerik menggunakan hasil visualisasi histogram. Apakah kolom tersebut berdistribusi normal? Jika bukan, terdistribusi seperti apa kolom tersebut?

H0: Data membentuk distribusi normal

H1: Data tidak membentuk distribusi normal

Berdasarkan uji distribusi normal, dapat disimpulkan bahwa dengan threshold α = 0.05, setiap kolom pada dataset bukan merupakan distribusi normal. Hal ini ditunjukkan oleh nilai p-value yang lebih kecil dari α, yang berarti kita memiliki bukti yang cukup untuk menolak hipotesis nol.

In [ ]:

```
acidity_stats, acidity_pvalue = normaltest(cleaned_acidity)

print("Acidity:")
print(f"Statistics: {acidity_stats}")
print(f"P-value: {acidity_pvalue}")
print(f"Karena p-value {acidity_pvalue:.5f} < 0.05, maka hipotesis nol ditolak. Seh
```

```
Acidity:
Statistics: 11.795077289737968
P-value: 0.002746195870288523
Karena p-value 0.00275 < 0.05, maka hipotesis nol ditolak. Sehingga, data acidity
tidak berdistribusi normal.
```

In [ ]:

```
weight_stats, weight_pvalue = normaltest(cleaned_weight)

print("Weight:")
print(f"Statistics: {weight_stats}")
print(f"P-value: {weight_pvalue}")
print(f"Karena p-value {weight_pvalue:.5f} < 0.05, maka hipotesis nol ditolak. Sehi
Weight:
Statistics: 7.739094023965337
P-value: 0.020867820156997934
Karena p-value 0.02087 < 0.05, maka hipotesis nol ditolak. Sehingga, data weight t
idak berdistribusi normal.
```

In [ ]:

```
length_stats, length_pvalue = normaltest(cleaned_length)

print("Length:")
print(f"Statistics: {length_stats}")
print(f"P-value: {length_pvalue}")
print(f"Karena p-value {length_pvalue:.5f} < 0.05, maka hipotesis nol ditolak. Sehi
```

```
Length:
Statistics: 13.923714327468733
P-value: 0.0009473355825336166
Karena p-value 0.00095 < 0.05, maka hipotesis nol ditolak. Sehingga, data length t
idak berdistribusi normal.
```

In [ ]:

```
appearance_stats, appearance_pvalue = normaltest(cleaned_appearance)

print("Appearance:")
print(f"Statistics: {appearance_stats}")
print(f"P-value: {appearance_pvalue}")
print(f"Karena p-value {appearance_pvalue:.5f} > 0.05, maka hipotesis nol diterima.
```

```
Appearance:
Statistics: 5.470683077112884
P-value: 0.06487184708282746
Karena p-value 0.06487 > 0.05, maka hipotesis nol diterima. Sehingga, data appeara
nce berdistribusi normal.
```

In [ ]:

```
tanning_stats, tanning_pvalue = normaltest(cleaned_tannin)

print("Tannin:")
print(f"Statistics: {tanning_stats}")
print(f"P-value: {tanning_pvalue}")
print(f"Karena p-value {tanning_pvalue:.5f} < 0.05, maka hipotesis nol ditolak. Seh
```

```
Tannin:
Statistics: 8.751479562255359
P-value: 0.012578833216116046
Karena p-value 0.01258 < 0.05, maka hipotesis nol ditolak. Sehingga, data tannin t
idak berdistribusi normal.
```

In [ ]:

```
ripeness_stats, ripeness_pvalue = normaltest(cleaned_ripeness)

print("Ripeness:")
print(f"Statistics: {ripeness_stats}")
print(f"P-value: {ripeness_pvalue}")
print(f"Karena p-value {ripeness_pvalue:.1f} < 0.05, maka hipotesis nol ditolak. Se
```

```
Ripeness:
Statistics: 41.67243761835628
P-value: 8.931926639859596e-10
Karena p-value 0.0 < 0.05, maka hipotesis nol ditolak. Sehingga, data ripeness tid
ak berdistribusi normal.
```

In [ ]:

```
sweetness_stats, sweetness_pvalue = normaltest(cleaned_sweetness)

print("Sweetness:")
print(f"Statistics: {sweetness_stats}")
print(f"P-value: {sweetness_pvalue}")
print(f"Karena p-value {sweetness_pvalue:.1f} < 0.05, maka hipotesis nol ditolak. S
Sweetness:
Statistics: 60.013364863357644
P-value: 9.295299759887164e-14
Karena p-value 0.0 < 0.05, maka hipotesis nol ditolak. Sehingga, data sweetness ti
dak berdistribusi normal.
```

In [ ]:

```
firmness_stats, firmness_pvalue = normaltest(cleaned_firmness)

print("Firmness:")
print(f"Statistics: {firmness_stats}")
print(f"P-value: {firmness_pvalue}")
print(f"Karena p-value {firmness_pvalue} < 0.05, maka hipotesis nol ditolak. Sehing
```

```
Firmness:
Statistics: 1562.6266133988306
P-value: 0.0
Karena p-value 0.0 < 0.05, maka hipotesis nol ditolak. Sehingga, data firmness tid
ak berdistribusi normal.
```

In [ ]:

```
price_stats, price_pvalue = normaltest(cleaned_price)

print("Price:")
print(f"Statistics: {price_stats}")
print(f"P-value: {price_pvalue}")
print(f"Karena p-value {price_pvalue:.5f} < 0.05, maka hipotesis nol ditolak. Sehir
```

```
Price:
Statistics: 6.36440195796569
P-value: 0.04149422662127596
Karena p-value 0.04149 < 0.05, maka hipotesis nol ditolak. Sehingga, data price ti
dak berdistribusi normal.
```

5. Hipotesis 1 sampel

Perusahaan menerima beberapa keluhan bahwa buah pisang yang mereka terima akhir-akhir ini cukup asam. Dapatkah anda mengecek apakah rata-rata nilai Acidity di atas 6?

1. Penentuan hipotesis null.

   H_0: Rata-rata nilai acidity = 6

2. Penentuan hipotesis alternatif.

   H_1: Rata-rata nilai acidity > 6

3. Penentuan tingkat signifikan.

   α = 0,05

4. Penentuan uji statistik dan daerah kritis. (Nilai numerik diprint dibawah)

   Uji statistik: z = (x̄ - μ_0)/(σ/√n)

Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih random sampling sebanyak n = 100.

Daerah kritis:

$P(z > z_c) = 0{,}05$

$1 - P(z < z_c) = 0{,}05$

$P(z < z_c) = 0{,}95$

$\bar{x}_c = z_c * \sigma/\sqrt{n} + \mu_0$

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   $z_{test} = (\bar{x}_{test} - \mu_0)/(\sigma/\sqrt{n})$

   p-value = $(1 - P(z < abs(z_{test})))$

1. Pengambilan Keputusan (Keputusan diprint dibawah)

In [ ]:

```python
# Nilai2 penting
alpha = 0.05
sample_size = 100
std_cleaned_acidity = np.std(cleaned_acidity)
mean_cleaned_acidity = np.mean(cleaned_acidity)

# 4. Nilai kritis
z_critical = norm.ppf(1 - alpha)
x_critical = mean_cleaned_acidity + z_critical * std_cleaned_acidity / np.sqrt(samp
print(f"z critical: {z_critical}")
print(f"x critical: {x_critical}")

# Pengambilan sample sebanyak n = 100 dari cleaned_acidity
sample_acidity = cleaned_data.sample(n=sample_size, random_state=1)["Acidity"].valu
sample_mean = sample_acidity.mean()

# 5. Perhitungan uji statistik dan p value
z_test = (sample_mean - mean_cleaned_acidity) / (std_cleaned_acidity / np.sqrt(samp
p_value = 1 - norm.cdf(np.abs(z_test))
print(f"sample mean: {sample_mean}")
print(f"z test: {z_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata sample
else:
    print(f"Keputusan: Menolak H0 sehingga rata-rata sample acidity bukan 6.")
```

```
z critical: 1.6448536269514722
x critical: 8.178555580043403
sample mean: 8.01308781477132
z test: 0.12112974306309982
p-value: 0.4517941355330076
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata sample acidity 6.
```

Supplier menjanjikan bahwa rata-rata berat buah pisang adalah 150 gram. Pemilik mencurigai

kebenaran hal ini. Apakah rata-rata buah pisang yang mereka kirim tidak bernilai 150 gram?

1. Penentuan hipotesis null.

   H_0: Rata-rata berat buah pisang = 150

2. Penentuan hipotesis alternatif.

   H_1: Rata-rata berat buah pisang ≠ 150

3. Penentuan tingkat signifikan.

   α = 0,05

4. Penentuan uji statistik dan daerah kritis. (Nilai numerik diprint dibawah)

   Uji statistik: z = (x̄ - μ_0)/(σ/√n)

   Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih random sampling sebanyak n = 100.

   Daerah kritis:

   P(z < z_c1) = 0,05/2 = 0,025

   P(z > z_c2) = 0,05/2 = 0,025

   x̄_c1 = μ_0 + z_c1 * σ/√n

   x̄_c2 = μ_0 + z_c2 * σ/√n

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   z_test = (x̄_test - μ_0)/(σ/√n)

   p-value = 2 * (1 - P(z < abs(z_test)))

6. Pengambilan Keputusan (Keputusan diprint dibawah)

In [ ]:

```
# Nilai-nilai penting
alpha = 0.05
sample_size = 100
std_cleaned_weight = np.std(cleaned_weight)
mean_cleaned_weight = np.mean(cleaned_weight)

# 4. Nilai kritis
z_critical_1 = norm.ppf(alpha / 2)
z_critical_2 = norm.ppf(1 - alpha / 2)
x_critical_1 = mean_cleaned_weight + z_critical_1 * std_cleaned_weight / np.sqrt(sa
x_critical_2 = mean_cleaned_weight + z_critical_2 * std_cleaned_weight / np.sqrt(sa
print(f"z critical 1: {z_critical_1}")
print(f"z critical 2: {z_critical_2}")
print(f"x critical 1: {x_critical_1}")
print(f"x critical 2: {x_critical_2}")
```

```
# Pengambilan sample sebanyak n = 100 dari cleaned_weight
sample_weight = cleaned_data.sample(n=sample_size, random_state=1)["Weight"].values
sample_mean = sample_weight.mean()

# 5. Perhitungan uji statistik dan p value
z_test = (sample_mean - mean_cleaned_weight) / (std_cleaned_weight / np.sqrt(sample
p_value = 2 * (1 - norm.cdf(np.abs(z_test)))
print(f"sample mean: {sample_mean}")
print(f"z-test: {z_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata sample
else:
    print(f"Keputusan: Menolak H0 sehingga rata-rata sample weight bukan 0.5.")
```

```
z critical 1: -1.9599639845400545
z critical 2: 1.959963984540054
x critical 1: 149.79644109177
x critical 2: 150.24928908328704
sample mean: 150.23494006963475
z-test: 1.8357565219964591
p-value: 0.0663936743625464
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata sample weight 0.
5.
```

Periksalah apakah rata-rata panjang buah pisang 10 baris terakhir tidak sama dengan 49!

1. Penentuan hipotesis null.

   H_0: Rata-rata berat 10 buah pisang terakhir = 49

2. Penentuan hipotesis alternatif.

   H_1: Rata-rata berat 10 buah pisang terakhir ≠ 49

3. Penentuan tingkat signifikan.

   α = 0,05

4. Penentuan uji statistik dan daerah kritis. (Nilai numerik diprint dibawah)

   Uji statistik: z = (x̄ - μ_0)/(σ/√n)

   Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih sampling sebanyak n = 10 (10 pisang terakhir).

   Daerah kritis:

   P(z < z_c1) = 0,05/2 = 0,025

   P(z > z_c2) = 0,05/2 = 0,025

   x̄_c1 = μ_0 + z_c1 * σ/√n

$\bar{x}\_c2 = \mu\_0 + z\_c2 * \sigma/\sqrt{n}$

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   $z\_test = (\bar{x}\_test - \mu\_0)/(\sigma/\sqrt{n})$

   p-value = 2 * (1 - P(z < abs(z_test)))

6. Pengambilan Keputusan (Keputusan diprint dibawah)

In [ ]:

```
# Nilai-nilai penting
alpha = 0.05
sample_size = 10
std_cleaned_weight = np.std(cleaned_weight)
mean_cleaned_weight = np.mean(cleaned_weight)

# 4. Nilai kritis
z_critical_1 = norm.ppf(alpha / 2)
z_critical_2 = norm.ppf(1 - alpha / 2)
x_critical_1 = mean_cleaned_weight + z_critical_1 * std_cleaned_weight / np.sqrt(sa
x_critical_2 = mean_cleaned_weight + z_critical_2 * std_cleaned_weight / np.sqrt(sa
print(f"z critical 1: {z_critical_1}")
print(f"z critical 2: {z_critical_2}")
print(f"x critical 1: {x_critical_1}")
print(f"x critical 2: {x_critical_2}")

# Pengambilan sample sebanyak n = 10 dari cleaned_weight (10 terakhir)
sample_weight = cleaned_weight[-10:]
sample_mean = np.mean(sample_weight)

# 5. Perhitungan uji statistik dan p value
z_test = (sample_mean - mean_cleaned_weight) / (std_cleaned_weight / np.sqrt(sample
p_value = 2 * (1 - norm.cdf(np.abs(z_test)))
print(f"sample mean: {sample_mean}")
print(f"z-test: {z_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata berat 
else:
    print(f"Keputusan: Menolak H0 sehingga rata-rata berat 10 pisang terakhir tidak
```

```
z critical 1: -1.9599639845400545
z critical 2: 1.959963984540054
x critical 1: 149.3068495440153
x critical 2: 150.73888063104172
sample mean: 150.5784365817542
z-test: 1.5207772085179199
p-value: 0.12831575574019904
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata berat 10 pisang t
erakhir sama dengan 49.
```

```
z critical 2: 1.959963984540054
x critical 1: 149.3068495440153
x critical 2: 150.73888063104172
sample mean: 150.5784365817542
z-test: 1.5207772085179199
p-value: 0.12831575574019904
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata berat 10 pisang t
erakhir sama dengan 49.
```

Apakah proporsi nilai Tannin yang lebih besar dari 8 tidak sama dengan 55% dari total dataset?

1. Penentuan hipotesis null.

   H_0: P(x > 8) = 0,55

2. Penentuan hipotesis alternatif.

   H_1: P(x > 8) ≠ 0,55

3. Penentuan tingkat signifikan.

   α = 0,05

4. Penentuan uji statistik dan daerah kritis. (Nilai numerik diprint dibawah)

   Uji statistik: z = (x̄ - μ_0)/(σ/√n)

   Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih random sampling sebanyak n = 100 orang.

   Daerah kritis:

   P(z < z_c1) = 0,05/2 = 0,025

   P(z > z_c2) = 0,05/2 = 0,025

   x̄_c1 = μ_0 + z_c1 * σ/√n

   x̄_c2 = μ_0 + z_c2 * σ/√n

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   z_test = (x̄_test - μ_0)/(σ/√n)

   p-value = 2 * (1 - P(z < abs(z_test)))

6. Pengambilan Keputusan (Keputusan diprint dibawah)

In [ ]:

```
# Nilai-nilai penting
alpha = 0.05
sample_size = 100
std_cleaned_length = np.std(cleaned_length)
mean_cleaned_length = np.mean(cleaned_length)

# 4. Nilai kritis
z_critical_1 = norm.ppf(alpha / 2)
z_critical_2 = norm.ppf(1 - alpha / 2)
x_critical_1 = mean_cleaned_length + z_critical_1 * std_cleaned_length / np.sqrt(sa
x_critical_2 = mean_cleaned_length + z_critical_2 * std_cleaned_length / np.sqrt(sa

print(f"z critical 1: {z_critical_1}")
print(f"z critical 2: {z_critical_2}")
print(f"x critical 1: {x_critical_1}")
print(f"x critical 2: {x_critical_2}")

# Pengambilan sample sebanyak n = 100 dari cleaned_length
sample_length = cleaned_data.sample(n=sample_size, random_state=1)["Length"].values
sample_mean = sample_length.mean()

# 5. Perhitungan uji statistik dan p value
z_test = (sample_mean - mean_cleaned_length) / (std_cleaned_length / np.sqrt(sample
p_value = 2 * (1 - norm.cdf(np.abs(z_test)))

print(f"sample mean: {sample_mean}")
print(f"z-test: {z_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga proporsi nilai t
else:
    print(f"Keputusan: Menolak H0 sehingga proporsi nilai tannin yang lebih dari 8
```

```
z critical 1: -1.9599639845400545
z critical 2: 1.959963984540054
x critical 1: 49.77811091253487
x critical 2: 50.11861050435634
sample mean: 50.00144423959682
z-test: 0.6111126810580366
p-value: 0.5411249851985063
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga proporsi nilai tannin yang
lebih dari 8 adalah 0,55.
```

6. Hipotesis 2 sampel

Perusahaan ingin membandingkan kualitas buah yang diterima pada paruh awal dan paruh akhir kerjasama. Anda dapat melakukan ini dengan membagi 1 dataset menjadi 2 bagian yang sama panjang.

Anda diminta untuk memeriksa apakah rata-rata acidity dari buah pisang yang disuplai bernilai sama pada kedua kurun waktu tersebut.

1. Penentuan hipotesis null

   H_0: Rata-rata acidity pada paruh waktu awal dan paruh waktu akhir sama

2. Penentuan hipotesis alternatif

   H_1: Rata-rata acidity pada paruh waktu awal dan paruh waktu akhir berbeda

3. Penentuan tingkat signifikan

   α = 0.05

4. Penentuan uji statistik dan daerah kritis

   Uji statistik: $z = ((\bar{x}1 - \bar{x}2) - d0)/(\sqrt{}(\sigma1^2/n1 + \sigma2^2/n2))$

   Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih random sampling sebanyak n = 100.

   Daerah kritis:

   $P(z < -z\_\alpha/2)$ atau $P(z > z\_\alpha/2)$

   $P(z < -z\_\alpha/2) + P(z > z\_\alpha/2) = 2 * P(z < -z\_\alpha/2)$

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   $z = ((\bar{x}1 - \bar{x}2) - d0)/(\sqrt{}(\sigma1^2/n1 + \sigma2^2/n2))$

   p-value = $2 * P(z < abs(z\_\alpha/2))$

6. Pengambilan keputusan (Keputusan diprint dibawah)

In [ ]:

```python
# Membagi data menjadi dua kelompok, paruh awal dan paruh akhir
first_half_acidity = cleaned_acidity[:len(cleaned_acidity) // 2]
second_half_acidity = cleaned_acidity[len(cleaned_acidity) // 2:]

df_first_half_acidity = pd.DataFrame(first_half_acidity, columns=["Acidity"])
df_second_half_acidity = pd.DataFrame(second_half_acidity, columns=["Acidity"])

# Nilai2 penting
d0 = 0
alpha = 0.05
sample_size = 100


# 4. Nilai kritis two-tailed test
z_critical_1 = norm.ppf(alpha / 2)
z_critical_2 = norm.ppf(1 - alpha / 2)
print(f"z critical 1: {z_critical_1}")
print(f"z critical 2: {z_critical_2}")

# Pengambilan sample sebanyak n = 100 dari first_half_acidity
sample_first_half_acidity = df_first_half_acidity.sample(n=sample_size, random_stat
sample_mean_first_half = np.mean(sample_first_half_acidity)
sample_std_dev_first_half = np.std(sample_first_half_acidity)
sample_size_first_half = len(sample_first_half_acidity)

# Pengambilan sample sebanyak n = 100 dari second_half_acidity
sample_second_half_acidity = df_second_half_acidity.sample(n=sample_size, random_st
sample_mean_second_half = np.mean(sample_second_half_acidity)
sample_std_dev_second_half = np.std(sample_second_half_acidity)
sample_size_second_half = len(sample_second_half_acidity)

# 5. Perhitungan uji statistik dan p value
z_test = (sample_mean_first_half - sample_mean_second_half - d0) / np.sqrt((sample_
p_value = 2 * (1 - norm.cdf(np.abs(z_test)))

print(f"sample mean first half: {sample_mean_first_half}")
print(f"sample mean second half: {sample_mean_second_half}")
print(f"z-test: {z_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata acidit
else:
    print(f"Keputusan: Menolak H0 sehingga rata-rata acidity paruh pertama tidak sa
```

```
z critical 1: -1.9599639845400545
z critical 2: 1.959963984540054
sample mean first half: 7.959018365839031
sample mean second half: 8.058550900820048
z-test: -0.6825458798095351
p-value: 0.4948938422170035
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata acidity paruh per
tama sama dengan paruh kedua.
```

Bandingkanlah rata-rata appearance pada bagian awal dan akhir. Apakah rata-rata appearance pada dataset bagian awal lebih besar daripada bagian akhir sebesar 0.1 unit?

1. Penentuan hipotesis null

   H_0: Rata-rata appearance pada paruh waktu awal lebih besar daripada paruh waktu akhir sebesar 0.1 unit ($\mu1 - \mu2 = 0.1$)

2. Penentuan hipotesis alternatif

   H_1: Rata-rata appearance pada paruh waktu awal tidak lebih besar daripada paruh waktu akhir sebesar 0.1 unit ($\mu1 - \mu2 > 0.1$)

3. Penentuan tingkat signifikan

   $\alpha = 0.05$

4. Penentuan uji statistik dan daerah kritis

   Uji statistik: $z = ((\bar{x}1 - \bar{x}2) - d0)/(\sqrt{}(\sigma1\text{^}2/n1 + \sigma2\text{^}2/n2))$

   Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih random sampling sebanyak n = 100.

   Daerah kritis:

   $P(z > z\_\alpha/2) = 1 - P(z < z\_\alpha/2)$

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   $z = ((\bar{x}1 - \bar{x}2) - d0)/(\sqrt{}(\sigma1\text{^}2/n1 + \sigma2\text{^}2/n2))$

   p-value = $1 - P(z < abs(z\_\alpha/2))$

6. Pengambilan keputusan (Keputusan diprint dibawah)

In [ ]:

```
# Membagi data menjadi dua kelompok, paruh awal dan paruh akhir
first_half_appearance = cleaned_appearance[:len(cleaned_appearance) // 2]
second_half_appearance = cleaned_appearance[len(cleaned_appearance) // 2:]

df_first_half_appearance = pd.DataFrame(first_half_appearance, columns=["Appearance
df_second_half_appearance = pd.DataFrame(second_half_appearance, columns=["Appearan

# Nilai2 penting
d0 = 0.1
alpha = 0.05
sample_size = 100

# 4. Nilai kritis one-tailed test
z_critical = norm.ppf(1 - alpha)
print(f"z critical: {z_critical}")

# Pengambilan sample sebanyak n = 100 dari first_half_appearance
sample_first_half_appearance = df_first_half_appearance.sample(n=sample_size, rando
sample_mean_first_half = np.mean(sample_first_half_appearance)
sample_std_dev_first_half = np.std(sample_first_half_appearance)
sample_size_first_half = len(sample_first_half_appearance)

# Pengambilan sample sebanyak n = 100 dari second_half_appearance
sample_second_half_appearance = df_second_half_appearance.sample(n=sample_size, ran
sample_mean_second_half = np.mean(sample_second_half_appearance)
sample_std_dev_second_half = np.std(sample_second_half_appearance)
sample_size_second_half = len(sample_second_half_appearance)

# 5. Perhitungan uji statistik dan p value
z_test = (sample_mean_first_half - sample_mean_second_half - d0) / np.sqrt((sample_
p_value = 1 - norm.cdf(np.abs(z_test))

print(f"sample mean first half: {sample_mean_first_half}")
print(f"sample mean second half: {sample_mean_second_half}")
print(f"z-test: {z_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata appear
else:
    print(f"Keputusan: Menolak H0 sehingga rata-rata appearance pada paruh waktu aw
```

```
z critical: 1.6448536269514722
sample mean first half: 5.002565811219602
sample mean second half: 5.049743136003799
z-test: -1.1431676130051407
p-value: 0.1264845031300431
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata appearance pada p
aruh waktu awal lebih besar daripada paruh waktu akhir sebesar 0.1 unit.

sample mean first half: 5.002565811219602
sample mean second half: 5.049743136003799
z-test: -1.1431676130051407
p-value: 0.1264845031300431
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata appearance pada p
aruh waktu awal lebih besar daripada paruh waktu akhir sebesar 0.1 unit.
```

Apakah variansi dari panjang pisang yang dipasok suplier sama pada bagian awal dan akhir?

1. Penentuan hipotesis null

   H_0: Variansi dari panjang pisang yang dipasok suplier pada bagian awal sama dengan bagian akhir

2. Penentuan hipotesis alternatif

   H_1: Variansi dari panjang pisang yang dipasok suplier pada bagian awal tidak sama dengan bagian akhir

3. Penentuan tingkat signifikan

   α = 0.05

4. Penentuan uji statistik dan daerah kritis

   Uji statistik: f = (s_1)^2/(s_2)^2

   Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih random sampling sebanyak n = 100.

   Daerah kritis:

   P(f < f_1-α/2(v1,v2)) atau P(f > f_α/2(v1,v2))

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   f = (s_1)^2/(s_2)^2

   p-value = 2 * min(P(f < f_1-α/2(v1,v2)), f_α/2(v1,v2))

   v1 = n1 - 1

   v2 = n2 - 1

6. Pengambilan keputusan (Keputusan diprint dibawah)

In [ ]:

```python
# Membagi data menjadi dua kelompok, paruh awal dan paruh akhir
first_half_length = cleaned_length[:len(cleaned_length) // 2]
second_half_length = cleaned_length[len(cleaned_length) // 2:]

df_first_half_length = pd.DataFrame(first_half_length, columns=["Length"])
df_second_half_length = pd.DataFrame(second_half_length, columns=["Length"])

# Nilai2 penting
alpha = 0.05
sample_size = 100

# 4. Nilai kritis two-tailed test f distribution
v1 = len(first_half_length) - 1
v2 = len(second_half_length) - 1
f_critical_1 = f.ppf(alpha / 2, v1, v2)
f_critical_2 = f.ppf(1 - alpha / 2, v1, v2)
print(f"f critical 1: {f_critical_1}")
print(f"f critical 2: {f_critical_2}")
```

```
# Pengambilan sample sebanyak n = 100 dari first_half_length
sample_first_half_length = df_first_half_length.sample(n=sample_size, random_state=
sample_mean_first_half = np.mean(sample_first_half_length)
sample_std_dev_first_half = np.std(sample_first_half_length, ddof=1)
sample_size_first_half = len(sample_first_half_length)

# Pengambilan sample sebanyak n = 100 dari second_half_length
sample_second_half_length = df_second_half_length.sample(n=sample_size, random_stat
sample_mean_second_half = np.mean(sample_second_half_length)
sample_std_dev_second_half = np.std(sample_second_half_length, ddof=1)
sample_size_second_half = len(sample_second_half_length)

# 5. Perhitungan uji statistik dan p value
f_test = sample_std_dev_first_half ** 2 / sample_std_dev_second_half ** 2
p_value = 2 * min(f.cdf(f_test, v1, v2), 1 - f.cdf(f_test, v1, v2))
print(f"sample mean first half: {sample_mean_first_half}")
print(f"sample mean second half: {sample_mean_second_half}")
print(f"f-test: {f_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata length
else:
    print(f"Keputusan: Menolak H0 sehingga rata-rata length paruh pertama tidak sam
```

```
f critical 1: 0.8795200207653124
f critical 2: 1.136978696097823
sample mean first half: 49.90224035864021
sample mean second half: 49.997419103251794
f-test: 0.9538256806182229
p-value: 0.4703250615531359
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata length paruh pert
ama sama dengan paruh kedua.

f critical 2: 1.136978696097823
sample mean first half: 49.90224035864021
sample mean second half: 49.997419103251794
f-test: 0.9538256806182229
p-value: 0.4703250615531359
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga rata-rata length paruh pert
ama sama dengan paruh kedua.
```

Apakah proporsi berat pisang yang lebih dari 150 pada dataset awal lebih besar daripada proporsi di bagian dataset akhir?

1. Penentuan hipotesis null

   H_0: Proporsi berat pisang lebih dari 150 pada dataset awal sama dengan proporsi di bagian dataset akhir (p1 = p2)

2. Penentuan hipotesis alternatif

   H_1: Proporsi berat pisang lebih dari 150 pada dataset awal tidak sama dengan proporsi di bagian dataset akhir (p1 > p2)

3. Penentuan tingkat signifikan

$\alpha = 0.05$

4. Penentuan uji statistik dan daerah kritis

   Uji statistik:

   $z = (p1 - p2)/\sqrt{(pq(1/n1 + 1/n2))}$

   $p = (x1 + x2)/(n1 + n2)$

   Dengan Central Limit Theorem, distribusi sample akan mengikuti distribusi normal as long as jumlah sample yang diambil besar. Kita pilih random sampling sebanyak n = 100.

   Daerah kritis:

   $P(z > z_\alpha) = 1 - P(z < z_\alpha)$

5. Perhitungan nilai uji statistik dan p-value. (Nilai numerik diprint dibawah)

   $z = (p1 - p2)/\sqrt{(pq(1/n1 + 1/n2))}$

   $p = (x1 + x2)/(n1 + n2)$

   $\text{p-value} = 1 - P(z < \text{abs}(z_\alpha))$

6. Pengambilan keputusan (Keputusan diprint dibawah)

In [ ]:

```
# Membagi data menjadi dua kelompok, paruh awal dan paruh akhir
first_half_weight = cleaned_weight[:len(cleaned_weight) // 2]
second_half_weight = cleaned_weight[len(cleaned_weight) // 2:]

df_first_half_weight = pd.DataFrame(first_half_weight, columns=["Weight"])
df_second_half_weight = pd.DataFrame(second_half_weight, columns=["Weight"])

# Nilai2 penting
alpha = 0.05
sample_size = 100

# 4. Nilai kritis one-tailed proportion test 2 sampel
z_critical = norm.ppf(1 - alpha)
print(f"z critical: {z_critical}")

# Pengambilan sample sebanyak n = 100 dari first_half_weight
sample_first_half_weight = df_first_half_weight.sample(n=sample_size, random_state=
sample_size_first_half = len(sample_first_half_weight)

# Pengambilan sample sebanyak n = 100 dari second_half_weight
sample_second_half_weight = df_second_half_weight.sample(n=sample_size, random_stat
sample_size_second_half = len(sample_second_half_weight)

# 5. Perhitungan uji statistik dan p value
x1 = np.sum(sample_first_half_weight > 150)
x2 = np.sum(sample_second_half_weight > 150)
p1 = x1 / sample_size_first_half
p2 = x2 / sample_size_second_half
p = (x1 + x2) / (sample_size_first_half + sample_size_second_half)
q = 1 - p
z_test = (p1 - p2) / np.sqrt(p * q * (1 / sample_size_first_half + 1 / sample_size_
p_value = 1 - norm.cdf(np.abs(z_test))
print(f"sample proportion first half: {p1}")
print(f"sample proportion second half: {p2}")
print(f"z-test: {z_test}")
print(f"p-value: {p_value}")

# 6. Pengambilan keputusan
if (p_value > alpha):
    print(f"Keputusan: Tidak cukup bukti untuk menolak H0 sehingga proporsi nilai w
else:
    print(f"Keputusan: Menolak H0 sehingga proporsi nilai weight yang lebih dari 15
```

```
z critical: 1.6448536269514722
sample proportion first half: 0.47
sample proportion second half: 0.45
z-test: 0.2837521769195823
p-value: 0.38830015298058185
Keputusan: Tidak cukup bukti untuk menolak H0 sehingga proporsi nilai weight yang
lebih dari 150 pada paruh pertama sama dengan paruh kedua.
```