## 1.Bowley's Skewness

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

tips = sns.load\_dataset('tips')

data = tips['total\_bill']

Q1 = np.percentile(data, 25)

Q2 = np.percentile(data, 50)

Q3 = np.percentile(data, 75)

print(f"Q1: {Q1}, Q2: {Q2}, Q3: {Q3}")

B = (Q1 + Q3 - 2\*Q2) / (Q3 - Q1)

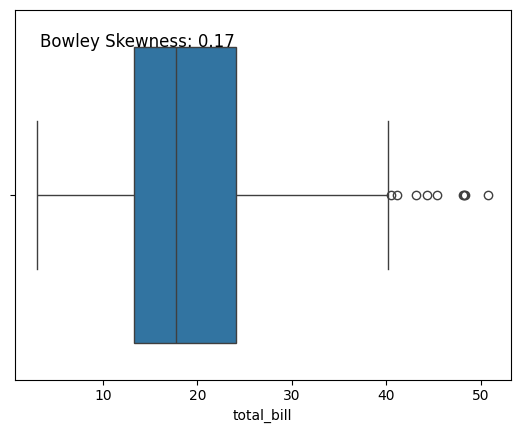
sns.boxplot(x=data)

plt.annotate(f'Bowley Skewness: {B:.2f}', xy=(0.05, 0.9), xycoords='axes fraction', fontsize=12)

plt.show()

OutPut:

Q1: 13.3475, Q2: 17.795, Q3: 24.127499999999998



## 2. Skewness and Kurtosis

import seaborn as sns

import matplotlib.pyplot as plt

from scipy.stats import skew, kurtosis

tips = sns.load\_dataset('tips')

data = tips['total\_bill']

skewness = skew(data)

kurt = kurtosis(data)

plt.hist(data, bins=30, alpha=0.5, color='b', edgecolor='black')

plt.xlabel('Total Bill')

plt.ylabel('Frequency')

plt.title('Histogram of Total Bill')

plt.annotate(f'Skewness: {skewness:.2f}', xy=(0.05, 0.9), xycoords='axes fraction', fontsize=12)

plt.annotate(f'Kurtosis: {kurt:.2f}', xy=(0.05, 0.85), xycoords='axes fraction', fontsize=12)

plt.show()

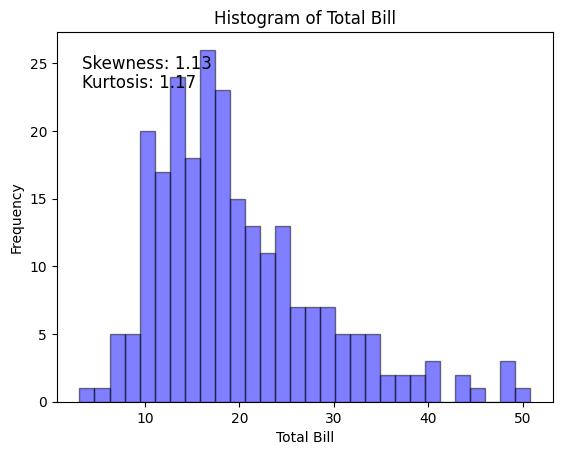
print(f"Skewness: {skewness}")

print(f"Kurtosis: {kurt}")

Output:

Skewness: 1.1262346334818638

Kurtosis: 1.1691681323851366



## 3. Univariate data Analysis

import seaborn as sns

import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 12))

sns.histplot(data=tips, x='total\_bill', bins=20, kde=True, ax=ax1)

ax1.set\_title('Histogram of Total Bill')

ax1.set\_xlabel('Total Bill ($)')

ax1.set\_ylabel('Frequency')

sns.boxplot(x='total\_bill', data=tips, ax=ax2, orient='h', palette='Set1')

ax2.set\_title('Quartiles of Total Bill')

ax2.set\_xlabel('Total Bill ($)')

ax2.set\_yticks([])

sns.kdeplot(data=tips, x='total\_bill', ax=ax3, fill=True)

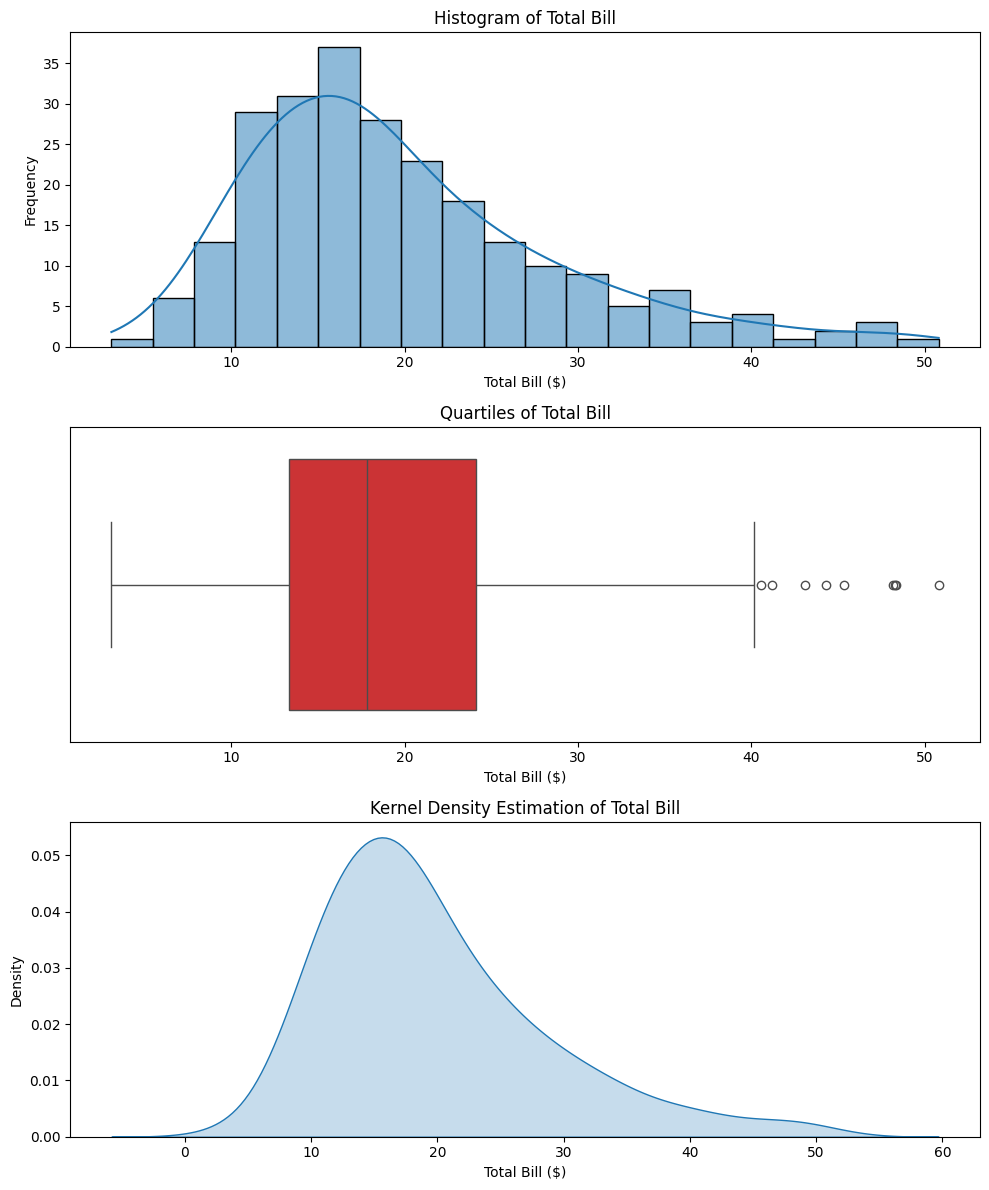
ax3.set\_title('Kernel Density Estimation of Total Bill')

ax3.set\_xlabel('Total Bill ($)')

ax3.set\_ylabel('Density')

plt.tight\_layout()

plt.show()



## 4. Multivariate Data Analysis

import seaborn as sns

import matplotlib.pyplot as plt

iris = sns.load\_dataset('iris')

sns.scatterplot(data=iris, x='sepal\_length', y='sepal\_width', hue='species')

plt.title('Scatter Plot of Sepal Length vs Sepal Width')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.show()

sns.pairplot(iris, hue='species')

plt.suptitle('Pairplot of Iris Dataset', y=1.02)

plt.show()

sns.scatterplot(data=iris, x='sepal\_length', y='sepal\_width', size='petal\_length', hue='species')

plt.title('Bubble Chart of Sepal Length vs Sepal Width (Size by Petal Length)')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.show()

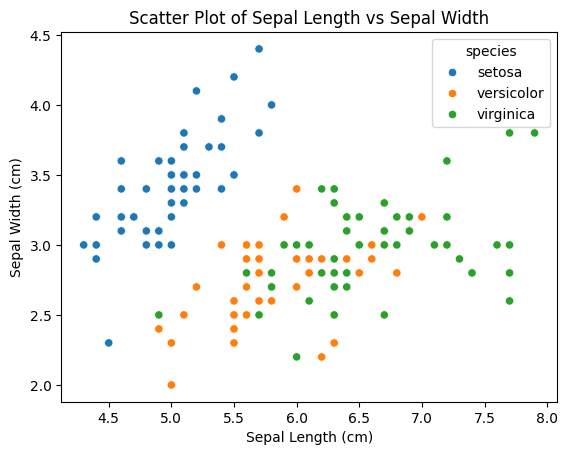
sns.kdeplot(data=iris, x='sepal\_length', y='sepal\_width', hue='species', fill=True)

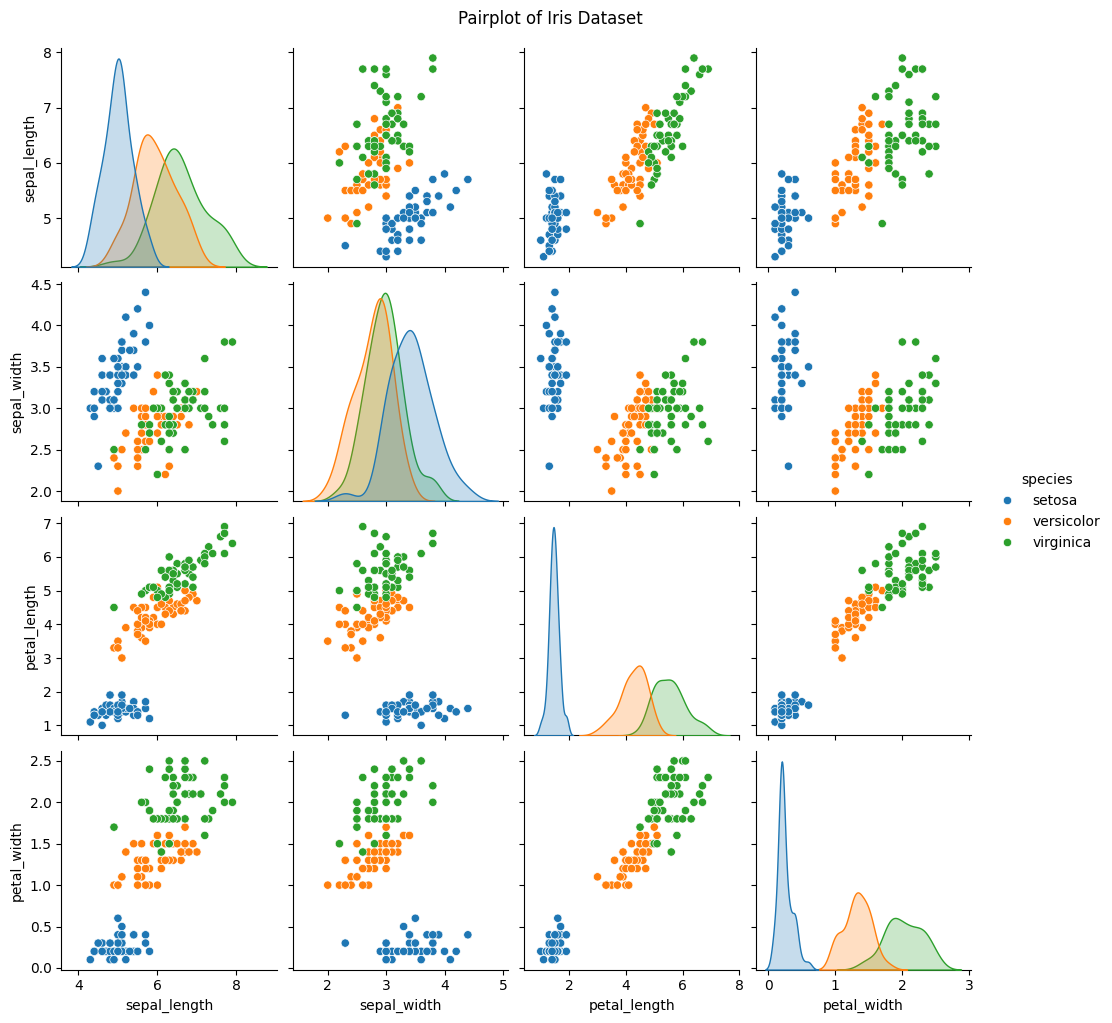
plt.title('Density Chart of Sepal Length vs Sepal Width')

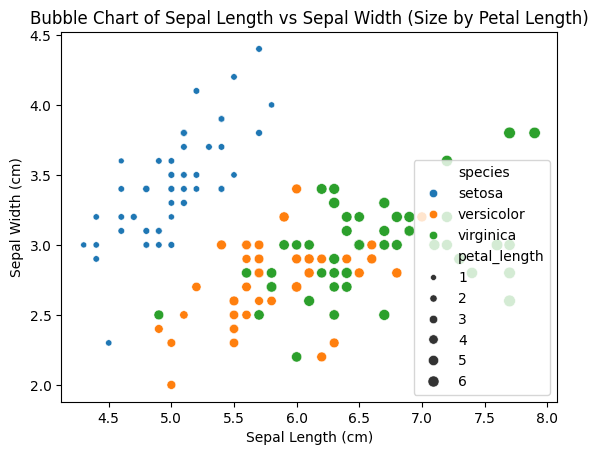
plt.xlabel('Sepal Length (cm)')

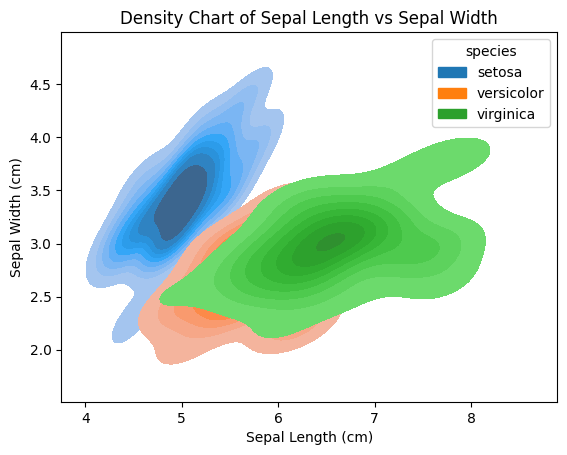
plt.ylabel('Sepal Width (cm)')

plt.show()









## 5. Normal Distribution

#The weights of apples in a certain orchard follow a normal distribution with a mean of 150 grams and a standard deviation of 20 grams.

#a) What is the probability that a randomly selected apple weighs between 140 and 160 grams?

#b) What is the probability that a randomly selected apple weighs more than 170 grams?

#c) What is the probability that a randomly selected apple weighs less than 120 grams?

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import norm

mean = 150

std\_dev = 20

x = np.linspace(50, 250, 500)

y = norm.pdf(x, mean, std\_dev)

plt.figure(figsize=(12, 6))

plt.plot(x, y, label='Normal Distribution (Mean=150, SD=20)')

plt.fill\_between(x, 0, y, where=((x >= 140) & (x <= 160)), color='orange', alpha=0.5, label='Between 140 and 160 grams')

plt.fill\_between(x, 0, y, where=(x > 160), color='green', alpha=0.5, label='More than 160 grams')

plt.fill\_between(x, 0, y, where=(x < 120), color='red', alpha=0.5, label='Less than 120 grams')

plt.xlabel('Weight (grams)')

plt.ylabel('Probability Density')

plt.title('Normal Distribution of Apple Weights')

plt.legend()

plt.grid(True)

plt.show()

prob\_between\_140\_160 = norm.cdf(160, mean, std\_dev) - norm.cdf(140, mean, std\_dev)

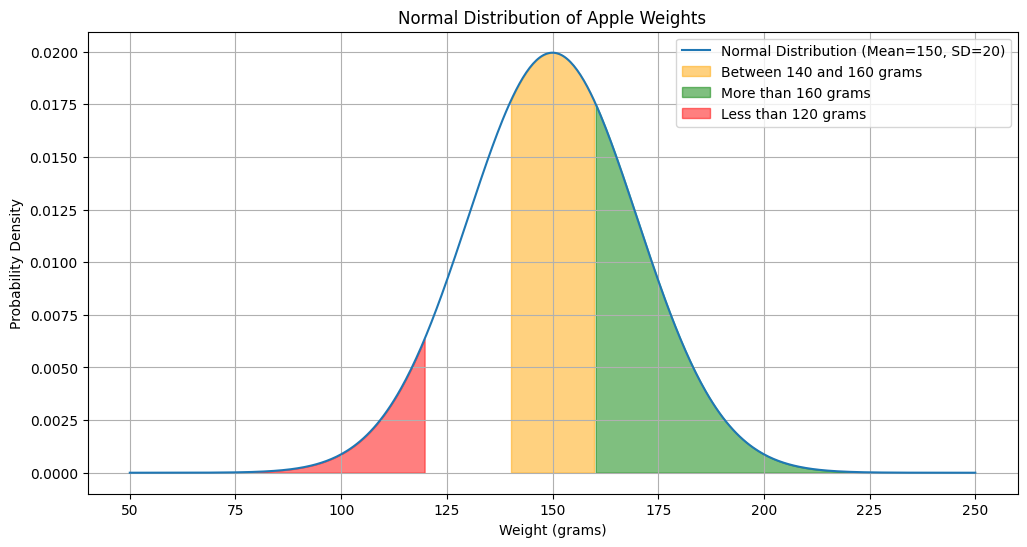
prob\_more\_than\_170 = 1 - norm.cdf(170, mean, std\_dev)

prob\_less\_than\_120 = norm.cdf(120, mean, std\_dev)

print(f"Probability of a randomly selected apple weighing between 140 and 160 grams: {prob\_between\_140\_160:.4f}")

print(f"Probability of a randomly selected apple weighing more than 170 grams: {prob\_more\_than\_170:.4f}")

print(f"Probability of a randomly selected apple weighing less than 120 grams: {prob\_less\_than\_120:.4f}")



Probability of a randomly selected apple weighing between 140 and 160 grams: 0.3829

Probability of a randomly selected apple weighing more than 170 grams: 0.1587

Probability of a randomly selected apple weighing less than 120 grams: 0.0668

## 6. Poisson Distribution

#The annual number of industrial accidents occurring in a particular manufacturing plant is known to

# follow Poisson distribution with mean 12.

#a)What is the probability of observing exactly 5 accidents at this plant during the coming year?

#b)What is the probability of observing not more than 12 accidents at this plant the coming year?

#c)What is the probability of observing at least 15 accidents at this plant during the coming year?

#d)What is the probability of observing between 10 and 15 accidents (inclusive) at this plant during the coming year?

from scipy.stats import poisson

import matplotlib.pyplot as plt

import numpy as np

mean = 12

x = np.arange(0, 30)

poisson\_dist = poisson.pmf(x, mean)

plt.figure(figsize=(12, 6))

plt.bar(x, poisson\_dist, label='Poisson Distribution (Mean=12)', alpha=0.5)

plt.fill\_between(x, 0, poisson\_dist, where=(x == 5), color='red', alpha=0.5, label='Exactly 5 accidents')

plt.fill\_between(x, 0, poisson\_dist, where=(x <= 12), color='green', alpha=0.5, label='Not more than 12 accidents')

plt.fill\_between(x, 0, poisson\_dist, where=(x >= 15), color='blue', alpha=0.5, label='At least 15 accidents')

plt.fill\_between(x, 0, poisson\_dist, where=((x >= 10) & (x <= 15)), color='orange', alpha=0.5, label='Between 10 and 15 accidents')

plt.xlabel('Number of Accidents')

plt.ylabel('Probability')

plt.title('Poisson Distribution of Accidents in a Manufacturing Plant')

plt.legend()

plt.grid(True)

plt.show()

prob\_5\_accidents = poisson.pmf(5, mean)

print(f"Probability of observing exactly 5 accidents: {prob\_5\_accidents}")

prob\_not\_more\_than\_12 = poisson.cdf(12, mean)

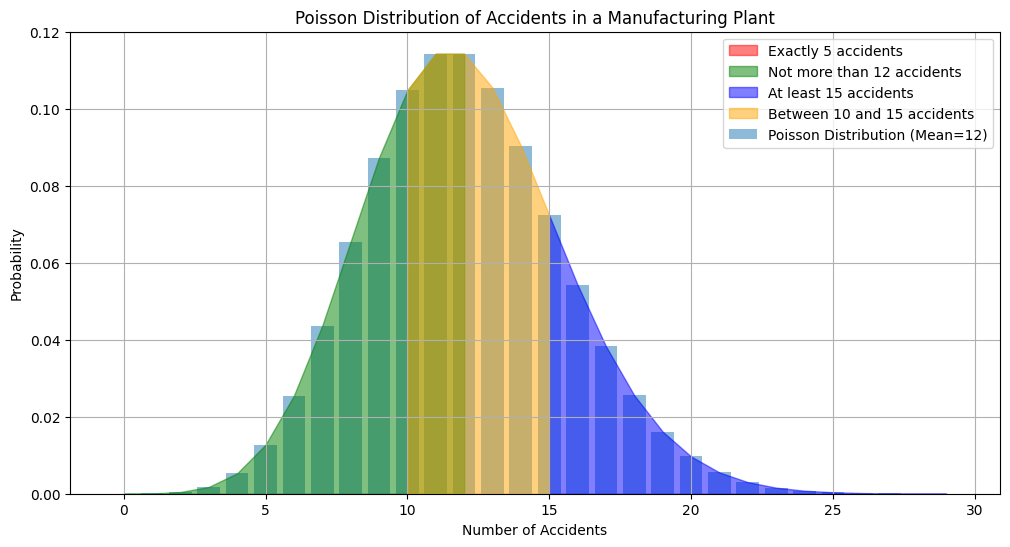
print(f"Probability of observing not more than 12 accidents: {prob\_not\_more\_than\_12}")

prob\_at\_least\_15 = 1 - poisson.cdf(14, mean)

print(f"Probability of observing at least 15 accidents: {prob\_at\_least\_15}")

prob\_between\_10\_and\_15 = poisson.cdf(15, mean) - poisson.cdf(9, mean)

print(f"Probability of observing between 10 and 15 accidents: {prob\_between\_10\_and\_15}")



Probability of observing exactly 5 accidents: 0.012740638735861376

Probability of observing not more than 12 accidents: 0.5759652485730645

Probability of observing at least 15 accidents: 0.22797546769645516

Probability of observing between 10 and 15 accidents: 0.6020234907796708

## 7. Z-Test

# Question: A soft drink company claims that the mean sugar content in its soda bottles is 40 grams.

# To test this claim, a random sample of 25 bottles is selected, and the sugar content is measured.

# The sample mean is found to be 38 grams with a standard deviation of 4 grams.

# Conduct a Z-test at a significance level of 0.05 to determine if there is enough evidence to reject the company's claim.

import matplotlib.pyplot as plt

import numpy as np

from scipy import stats

sample\_mean = 38

pop\_mean = 40

sample\_std = 4

sample\_size = 25

alpha = 0.05

z\_statistic = (sample\_mean - pop\_mean) / (sample\_std / np.sqrt(sample\_size))

p\_value = 2 \* (1 - stats.norm.cdf(np.abs(z\_statistic)))

x = np.linspace(-4, 4, 1000)

y = stats.norm.pdf(x, 0, 1)

plt.plot(x, y, 'b-', linewidth=2)

shade = np.linspace(abs(z\_statistic), 4, 300)

plt.fill\_between(shade, stats.norm.pdf(shade, 0, 1), color='red', alpha=0.5)

plt.title('Z-test for Mean Sugar Content')

plt.xlabel('Z')

plt.ylabel('Density')

plt.annotate('Critical Region\n(p < {:.3f})'.format(alpha), xy=(2, 0.1), xytext=(2.5, 0.2),

             arrowprops=dict(facecolor='black', shrink=0.05))

plt.axvline(x=abs(z\_statistic), color='black', linestyle='--', label='Z statistic')

plt.legend()

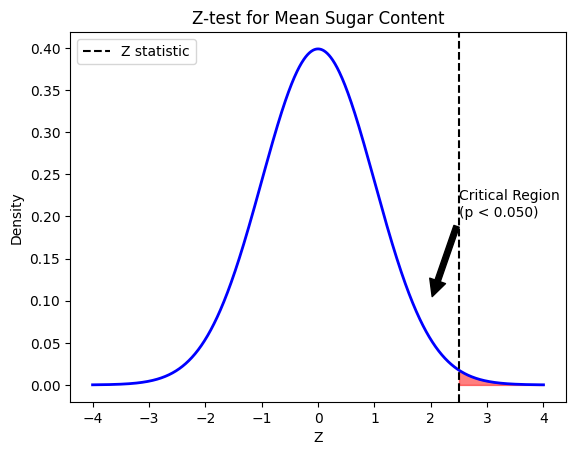
plt.show()

if p\_value < alpha:

    print("Reject the null hypothesis: There is enough evidence to conclude that the mean sugar content is not 40 grams.")

else:

    print("Fail to reject the null hypothesis: There is not enough evidence to conclude that the mean sugar content is not 40 grams.")



Reject the null hypothesis: There is enough evidence to conclude that the mean sugar content is not 40 grams.

## 8. Outlier Detection

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.neighbors import LocalOutlierFactor

iris = load\_iris()

X = iris.data

y = iris.target

lof = LocalOutlierFactor(n\_neighbors=20, contamination=0.1)

y\_pred = lof.fit\_predict(X)

plt.figure(figsize=(12, 6))

plt.scatter(X[:, 0], X[:, 1], c=y\_pred, cmap='viridis')

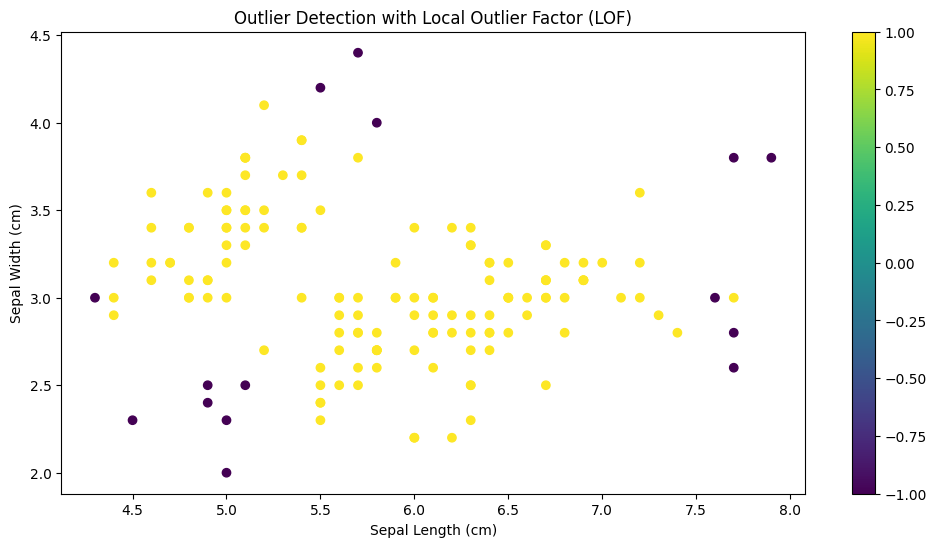
plt.colorbar()

plt.title('Outlier Detection with Local Outlier Factor (LOF)')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.show()



## 9. Performance Evaluation

from sklearn.datasets import load\_diabetes

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_absolute\_percentage\_error

diabetes = load\_diabetes()

X = diabetes.data

y = diabetes.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

mape = mean\_absolute\_percentage\_error(y\_test, y\_pred)

mase = mae / (sum(abs(y\_test[i + 1] - y\_test[i]) for i in range(len(y\_test) - 1)) / (len(y\_test) - 1))

print(f"Mean Absolute Error (MAE): {mae:.2f}")

print(f"Root Mean Square Error (RMSE): {rmse:.2f}")

print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}")

print(f"Mean Absolute Scaled Error (MASE): {mase:.2f}")

Output:

Mean Absolute Error (MAE): 42.79

Root Mean Square Error (RMSE): 53.85

Mean Absolute Percentage Error (MAPE): 0.37

Mean Absolute Scaled Error (MASE): 0.53

## 10. ARIMA

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from sklearn.metrics import mean\_squared\_error

from math import sqrt

from statsmodels.datasets import get\_rdataset

air\_passengers = get\_rdataset('AirPassengers', 'datasets').data['value']

air\_passengers.index = pd.date\_range(start='1949-01-01', periods=len(air\_passengers), freq='M')

model = ARIMA(air\_passengers, order=(1, 1, 1))

model\_fit = model.fit()

predictions = model\_fit.predict(start=len(air\_passengers), end=len(air\_passengers)+11, typ='levels')

plt.plot(air\_passengers.index, air\_passengers, label='Actual')

plt.plot(predictions.index, predictions, label='Predicted')

plt.title('ARIMA Model Forecasting')

plt.xlabel('Date')

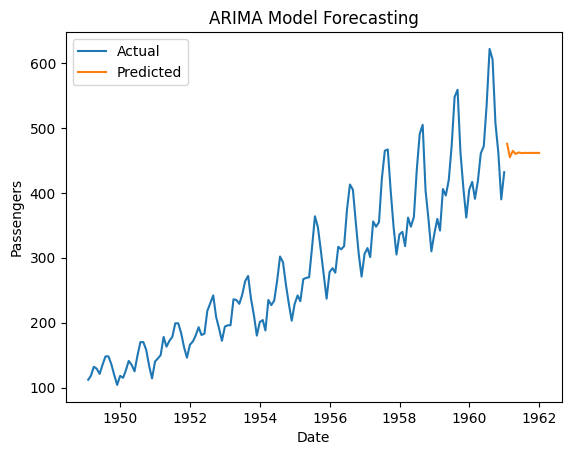
plt.ylabel('Passengers')

plt.legend()

plt.show()

rmse = sqrt(mean\_squared\_error(air\_passengers[-12:], predictions))

print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')



Root Mean Squared Error (RMSE): 76.29