==
====

Step by step explanation of code

1. Set the random seed:

```
np.random.seed(42)
```

- **Purpose:** Ensures that the random numbers generated by NumPy are reproducible, meaning every time you run this code, the same random numbers will be generated.

2. Create a synthetic dataset:

```
data = {
    'product_id': range(1, 21),
    'product_name': [f'Product {i}' for i in range(1, 21)],
    'category': np.random.choice(['Electronics', 'Clothing', 'Home', 'Sports'], 20),
    'units_sold': np.random.poisson(lam=20, size=20),  # Poisson distribution for sales
    'sale_date': pd.date_range(start='2023-01-01', periods=20, freq='D')
}
```

- **product_id:** A range of integers from 1 to 20, representing the unique product IDs.
- **product_name:** A list comprehension generates product names like "Product 1", "Product 2", etc., for 20 products.
- **category:** Uses np.random.choice() to randomly assign each product to one of four categories: 'Electronics', 'Clothing', 'Home', or 'Sports'. It picks one category for each of the 20 products.
- **units_sold:** Simulates the number of units sold for each product using a Poisson distribution with a mean (lam) of 20. This is commonly used to model count data, such as sales over time. The result is a random array of 20 integers, each representing units sold.
- **sale_date:** Generates a sequence of 20 consecutive dates starting from '2023-01-01', using the pd.date_range() function with a daily frequency (freq='D').

3. Create a Pandas DataFrame:

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

- The dictionary data is converted into a Pandas DataFrame, sales_data. Each key in the dictionary becomes a column, and the values become rows.

4. Display the first few rows:

```
sales_data.head()
```

- This function outputs the first five rows of the DataFrame to give a preview of the data.

==
====

INFERENTIAL STATS→

1. Calculate the sample mean and standard deviation:

```
mean_sales = sales_data['units_sold'].mean()
std_deviation_sales = sales_data['units_sold'].std()
```

- **mean_sales:** The average of the units_sold column in the dataset. This is the point estimate for the population mean.
- **std_deviation_sales:** The standard deviation of the units_sold column, which gives us a measure of the spread or variability in the number of units sold.

2. Define parameters for the confidence interval:

```
confidence_level = 0.95
degrees_freedom = len(sales_data['units_sold']) - 1
sample_standard_error = std_deviation_sales / np.sqrt(len(sales_data['units_sold']))
```

- **confidence_level:** The desired level of confidence for the interval. Here, it's 95%, which means we want to be 95% confident that the true mean lies within the interval.
- **degrees_freedom:** Calculated as the number of observations minus 1. It is needed because the t-distribution, used to calculate the confidence interval, depends on the degrees of freedom.
- **sample_standard_error:** The standard error of the mean, calculated as the sample standard deviation divided by the square root of the sample size (number of observations). This represents the variability in the sample mean.

3. Calculate the t-score for the confidence level:

```
t_score = stats.t.ppf((1 + confidence_level) / 2, degrees_freedom)
```

- **t_score:** This is the t-statistic (critical value) from the t-distribution, which depends on the confidence level and degrees of freedom. It determines the number of standard errors we need to add/subtract from the sample mean to create the confidence interval.

4. Calculate the margin of error:

```
margin_of_error = t_score * sample_standard_error
```

- **margin_of_error:** This is the range within which we expect the true population mean to fall. It's calculated by multiplying the t-score by the standard error of the sample mean.

5. Compute the confidence interval:

```
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
```

- **confidence_interval:** The range of values within which the true mean of units sold is likely to fall, with 95% confidence. It is computed by subtracting and adding the margin of error to the sample mean.

Output:

(17.25, 20.35)

This means that we are 95% confident that the true mean number of units sold lies between approximately 17.25 and 20.35.

===

1. Hypotheses:

- **Null Hypothesis (H□):** The mean units sold is equal to 20. This is the claim we are testing.

H0:μ=20H□: \mu = 20H0:μ=20

- **Alternative Hypothesis (H□):** The mean units sold is not equal to 20. This is the claim we try to find evidence for.

H1:μ≠20H□: \mu \neq 20H1:μ□=20

2. Perform the t-test:

```
t_statistic, p_value = stats.ttest_1samp(sales_data['units_sold'], 20)
```

- **stats.ttest_1samp():** This function performs a one-sample t-test. It tests whether the mean of the sample (sales_data['units_sold']) is statistically different from a given value (20 in this case).
- **Inputs:**
  - sales_data['units_sold']: The sample data representing units sold.
  - 20: The population mean we are testing against (i.e., the value in the null hypothesis).
- **Outputs:**
  - **t_statistic:** The t-statistic measures how many standard deviations the sample mean is from the hypothesized population mean (20). If the t-statistic is large, it suggests that the sample mean is far from 20, providing evidence against the null hypothesis.
  - **p_value:** The probability of observing the sample data, or something more extreme, assuming the null hypothesis is true. A low p-value indicates that the sample data is unlikely under the null hypothesis, suggesting we should reject the null.

3. Print the results of the t-test:

```
print(f"T-statistic: {t_statistic}, P-value: {p_value}")
```

- This prints the calculated t-statistic and the p-value.

4. Decision Rule:

```
if p_value < 0.05:
    print("Reject the null hypothesis: The mean units sold is significantly different from 20.")
else:
    print("Fail to reject the null hypothesis: The mean units sold is not significantly different from 20.")
```

- **p-value < 0.05:** If the p-value is less than the significance level (usually 0.05), we **reject the null hypothesis**, meaning there is enough evidence to say that the mean units sold is significantly different from 20.
- **p-value ≥ 0.05:** If the p-value is greater than or equal to 0.05, we **fail to reject the null hypothesis**, meaning there is not enough evidence to conclude that the mean units sold is different from 20.

Interpretation:

- If the p-value is low (less than 0.05), the test provides strong evidence against the null hypothesis, and we conclude that the mean units sold is significantly different from 20.
- If the p-value is high (greater than or equal to 0.05), we conclude that there isn't enough evidence to suggest that the mean units sold differs from 20.

==