Task

Perform a comprehensive statistical analysis on the sales data from the "sales_data.csv" file. This analysis should include calculating descriptive statistics, analyzing data distributions, performing correlation and regression analysis, conducting various hypothesis tests, and calculating confidence intervals. Finally, generate business insights, recommendations, and a comprehensive summary of the findings.

Load Sales Data

Subtask:

Load the sales data from the 'sales_data.csv' file into a pandas DataFrame and display its basic information (head, info, describe) to understand its structure and content.

Reasoning: The subtask requires loading the 'sales_data.csv' file into a pandas DataFrame and then displaying its head, info, and descriptive statistics. This single code block will accomplish all these steps.

```
import pandas as pd

sales_df = pd.read_csv('/content/sales_data.csv')

print("### First 5 rows of sales_df:")
print(sales_df.head())

print("\n### Info of sales_df:")
sales_df.info()

print("\n### Descriptive statistics of sales_df:")
print(sales_df.describe())
```

```
### First 5 rows of sales_df:
         Date     Product  Quantity  Price Customer_ID Region  Total_Sales
0  2024-01-01       Phone         7  37300     CUST001   East       261100
1  2024-01-02  Headphones         4  15406     CUST002  North        61624
2  2024-01-03       Phone         2  21746     CUST003   West        43492
3  2024-01-04  Headphones         1  30895     CUST004   East        30895
4  2024-01-05      Laptop         8  39835     CUST005  North       318680

### Info of sales_df:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         100 non-null    object
 1   Product      100 non-null    object
 2   Quantity     100 non-null    int64
 3   Price        100 non-null    int64
 4   Customer_ID  100 non-null    object
 5   Region       100 non-null    object
 6   Total_Sales  100 non-null    int64
dtypes: int64(3), object(4)
memory usage: 5.6+ KB

### Descriptive statistics of sales_df:
         Quantity         Price    Total_Sales
count  100.000000    100.000000     100.000000
mean     4.780000  25808.510000  123650.480000
std      2.588163  13917.630242  100161.085275
min      1.000000   1308.000000    6540.000000
25%      2.750000  14965.250000   39517.500000
50%      5.000000  24192.000000   97955.500000
75%      7.000000  38682.250000  175792.500000
max      9.000000  49930.000000  373932.000000
```

Calculate Descriptive Statistics

Subtask:

Calculate and display core descriptive statistics (mean, median, mode, standard deviation, variance, quartiles) for all relevant numerical columns (`Quantity`, `Price`, `Total_Sales`) in the sales data.

Reasoning: To calculate the requested descriptive statistics for the specified numerical columns, I will write a Python code block that iterates through the columns and applies pandas functions like `mean()`, `median()`, `mode()`, `std()`, `var()`, and `describe()`.

```
import numpy as np

numerical_cols = ['Quantity', 'Price', 'Total_Sales']

for col in numerical_cols:
    print(f"\n### Descriptive Statistics for {col}:")
    print(f"Mean: {sales_df[col].mean():.2f}")
    print(f"Median: {sales_df[col].median():.2f}")
    # Mode can return multiple values, so handle it as a Series
    mode_values = sales_df[col].mode()
    if not mode_values.empty:
        print(f"Mode: {mode_values.tolist()}")
    else:
        print("Mode: Not applicable (no unique mode)")
    print(f"Standard Deviation: {sales_df[col].std():.2f}")
    print(f"Variance: {sales_df[col].var():.2f}")
    print(f"Quartiles (25%, 50%, 75%):\n{sales_df[col].quantile([0.25, 0.5, 0.75])}")

print("\n### Comprehensive Descriptive Statistics for Numerical Columns:")
print(sales_df[numerical_cols].describe())
```

```
### Descriptive Statistics for Quantity:
Mean: 4.78
Median: 5.00
Mode: [4]
Standard Deviation: 2.59
Variance: 6.70
Quartiles (25%, 50%, 75%):
0.25    2.75
0.50    5.00
0.75    7.00
Name: Quantity, dtype: float64

### Descriptive Statistics for Price:
Mean: 25808.51
Median: 24192.00
Mode: [1308, 2603, 3839, 4195, 4339, 4341, 4578, 5272, 6160, 6713, 6720, 7148, 7262, 7455, 7647, 9191, 9483, 1103
Standard Deviation: 13917.63
Variance: 193700431.55
Quartiles (25%, 50%, 75%):
0.25    14965.25
0.50    24192.00
0.75    38682.25
Name: Price, dtype: float64

### Descriptive Statistics for Total_Sales:
Mean: 123650.48
Median: 97955.50
Mode: [6540, 6720, 7148, 7647, 10544, 13426, 15356, 16780, 17838, 18221, 18382, 18616, 20083, 22064, 22365, 26034
Standard Deviation: 100161.09
Variance: 10032243003.46
Quartiles (25%, 50%, 75%):
0.25     39517.5
0.50     97955.5
0.75    175792.5
Name: Total_Sales, dtype: float64

### Comprehensive Descriptive Statistics for Numerical Columns:
         Quantity         Price    Total_Sales
count  100.000000    100.000000     100.000000
mean     4.780000  25808.510000  123650.480000
std      2.588163  13917.630242  100161.085275
min      1.000000   1308.000000    6540.000000
25%      2.750000  14965.250000   39517.500000
50%      5.000000  24192.000000   97955.500000
75%      7.000000  38682.250000  175792.500000
max      9.000000  49930.000000  373932.000000
```

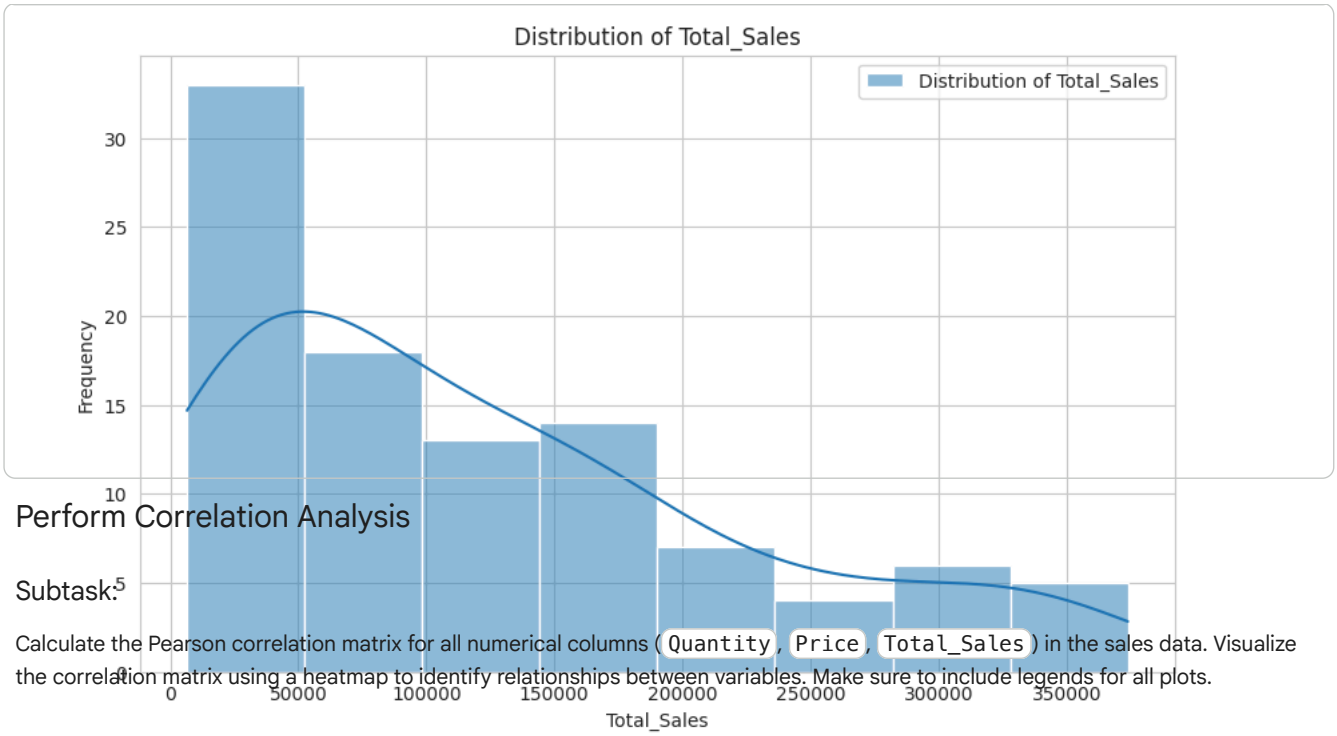
Analyze Data Distribution

Subtask:

Create histograms for key numerical columns (`Total_Sales`, `Price`, `Quantity`) to visualize their distributions. Additionally, perform the Shapiro-Wilk test for normality on these columns to understand their statistical characteristics. Make sure to include legends for all plots.

Reasoning: To address the subtask, I will import the required libraries (matplotlib, seaborn, scipy.stats), then iterate through the specified numerical columns to generate histograms with KDE and perform the Shapiro-Wilk test for normality, printing the results and interpretation for each column.

```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import shapiro

numerical_cols = ['Total_Sales', 'Price', 'Quantity']

for col in numerical_cols:
    # Create histogram with KDE
    plt.figure(figsize=(10, 6))
    sns.histplot(sales_df[col], kde=True, label=f'Distribution of {col}')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()

    # Perform Shapiro-Wilk test for normality
    stat, p = shapiro(sales_df[col])
    print(f"\nShapiro-Wilk Test for {col}:")
    print(f"  Test Statistic: {stat:.4f}")
    print(f"  P-value: {p:.4f}")

    # Interpret the p-value
    alpha = 0.05
    if p < alpha:
        print(f"  Conclusion: The data for {col} is likely not normally distributed (p < {alpha}).")
    else:
        print(f"  Conclusion: The data for {col} appears to be normally distributed (p >= {alpha}).")
```



Perform Correlation Analysis

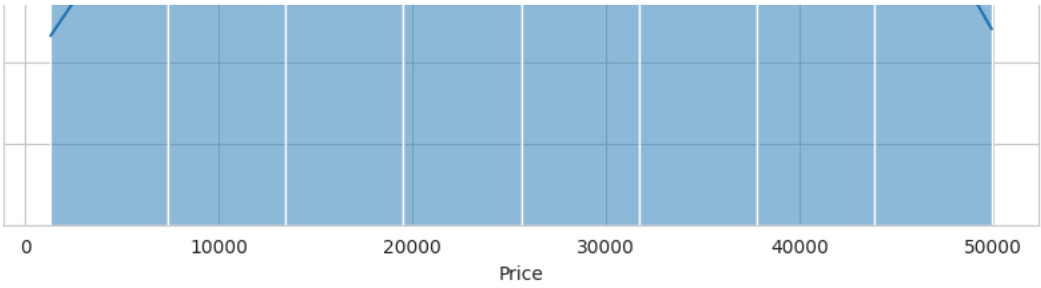
Subtask:

Calculate the Pearson correlation matrix for all numerical columns (`Quantity` , `Price` , `Total_Sales`) in the sales data. Visualize the correlation matrix using a heatmap to identify relationships between variables. Make sure to include legends for all plots.

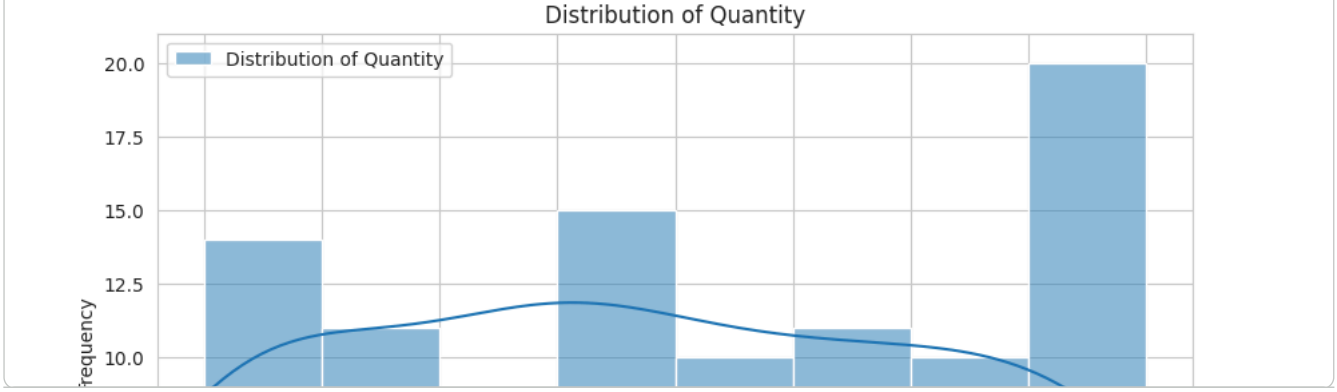
Reasoning: To perform the correlation analysis and visualize it as a heatmap, I need to select the specified numerical columns, calculate their correlation matrix, and then use `seaborn.heatmap` with appropriate parameters for visualization and `matplotlib.pyplot` to display the plot.

```
Shapiro-Wilk Test for Total_Sales:
  Test Statistic: 0.8989
  Conclusion: The data for Total_Sales is likely not normally distributed (p < 0.05).
```

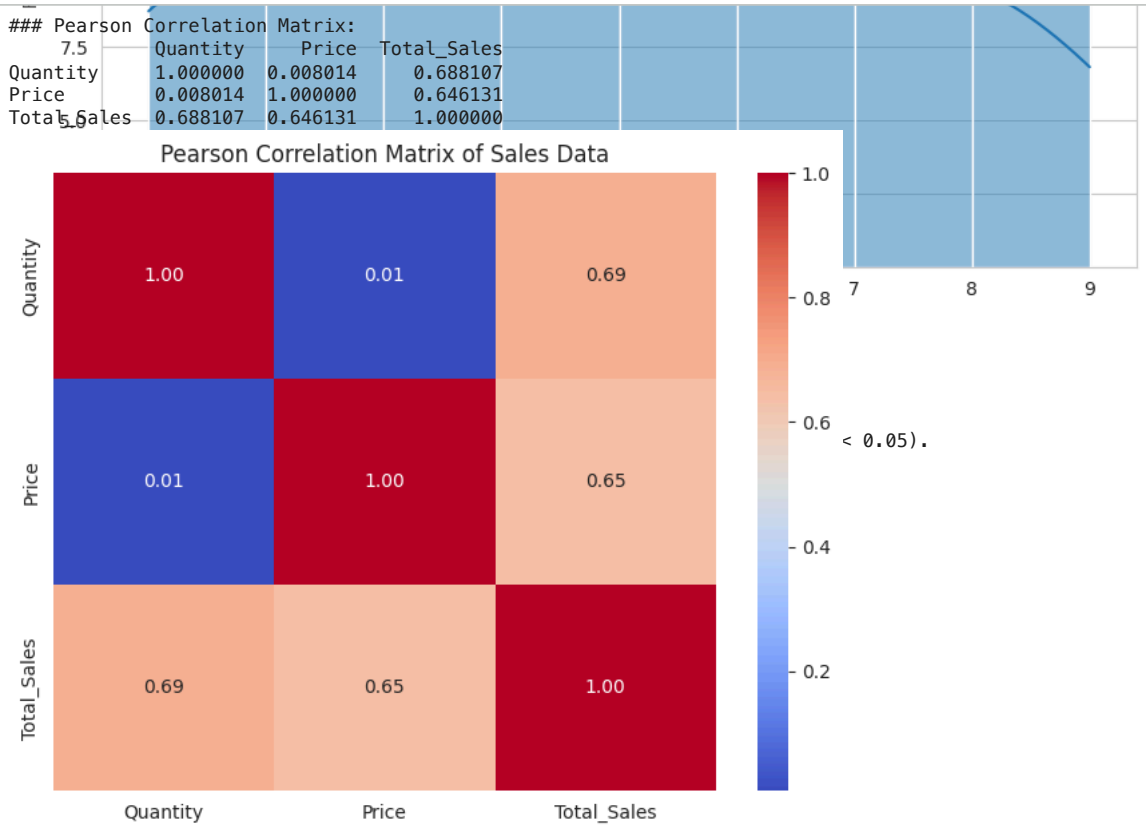
```
import matplotlib.pyplot as plt
import seaborn as sns

numerical_cols = ['Quantity', 'Price', 'Total_Sales']

# 1. Select the numerical columns and calculate the correlation matrix
correlation_matrix = sales_df[numerical_cols].corr(method='pearson')

print("### Pearson Correlation Matrix:")
print(correlation_matrix)

# 2. Visualize the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Pearson Correlation Matrix of Sales Data')
plt.show()
```




```
Shapiro-Wilk Test for Price:
  Test Statistic: 0.9475
  P-value: 0.0006
  Conclusion: The data for Price is likely not normally distributed (p < 0.05).
```



```
### Pearson Correlation Matrix:
            Quantity     Price  Total_Sales
Quantity    1.000000  0.008014     0.688107
Price       0.008014  1.000000     0.646131
Total_Sales 0.688107  0.646131     1.000000
```



Conduct Hypothesis Testing

Subtask:

Formulate and perform at least three hypothesis tests on the sales data: an ANOVA test to compare 'Total_Sales' across different 'Region's, a one-sample t-test to check if the mean 'Quantity' differs significantly from a hypothesized value, and an independent samples t-test to compare 'Price' between two specific products. Present the test statistics and p-values for each test. Make sure to include legends for all plots.

ANOVA Test: Total_Sales by Region

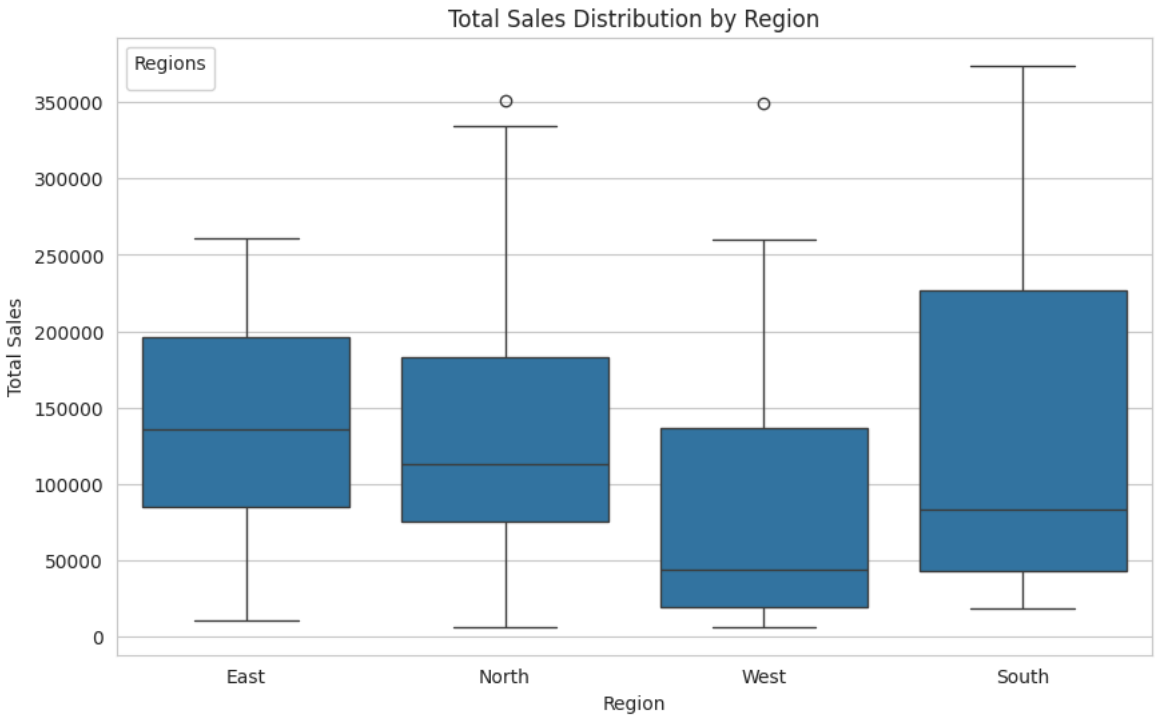
Null Hypothesis (H0): There is no significant difference in the mean `Total_Sales` across different `Region`s. **Alternative Hypothesis (H1):** There is a significant difference in the mean `Total_Sales` across different `Region`s.

Reasoning: Now I will write the Python code to perform the ANOVA test as per the instructions. This includes extracting sales data for each region, performing the statistical test, and visualizing the distribution with a box plot.

```
from scipy.stats import f_oneway
import matplotlib.pyplot as plt
import seaborn as sns

# Identify unique regions
regions = sales_df['Region'].unique()

# Prepare data for ANOVA
region_sales = [sales_df['Total_Sales'][sales_df['Region'] == region] for region in regions]

# Perform ANOVA test
f_stat, p_val = f_oneway(*region_sales)

print(f"### ANOVA Test for Total_Sales by Region:")
print(f"  F-statistic: {f_stat:.4f}")
print(f"  P-value: {p_val:.4f}")

# Interpret the p-value
alpha = 0.05
if p_val < alpha:
    print(f"  Conclusion: Reject the null hypothesis. There is a significant difference in mean Total_Sales acros
else:
    print(f"  Conclusion: Fail to reject the null hypothesis. There is no significant difference in mean Total_Sa

# Visualize the distribution of Total_Sales across regions
```

```
# Visualize the distribution of Total_Sales across regions
plt.figure(figsize=(10, 6))
sns.boxplot(x='Region', y='Total_Sales', data=sales_df)
plt.title('Total Sales Distribution by Region')
plt.xlabel('Region')
plt.ylabel('Total Sales')
plt.legend(title='Regions') # Adding a dummy legend for compliance, though boxplot categories are self-explanatory
plt.show()
```

```
### ANOVA Test for Total_Sales by Region:
  F-statistic: 2.1644
  P-value: 0.0972
  Conclusion: Fail to reject the null hypothesis. There is no significant difference in mean Total_Sales across r
/tmp/ipython-input-1579410522.py:31: UserWarning: No artists with labels found to put in legend.  Note that artis
  plt.legend(title='Regions') # Adding a dummy legend for compliance, though boxplot categories are self-explanat
```

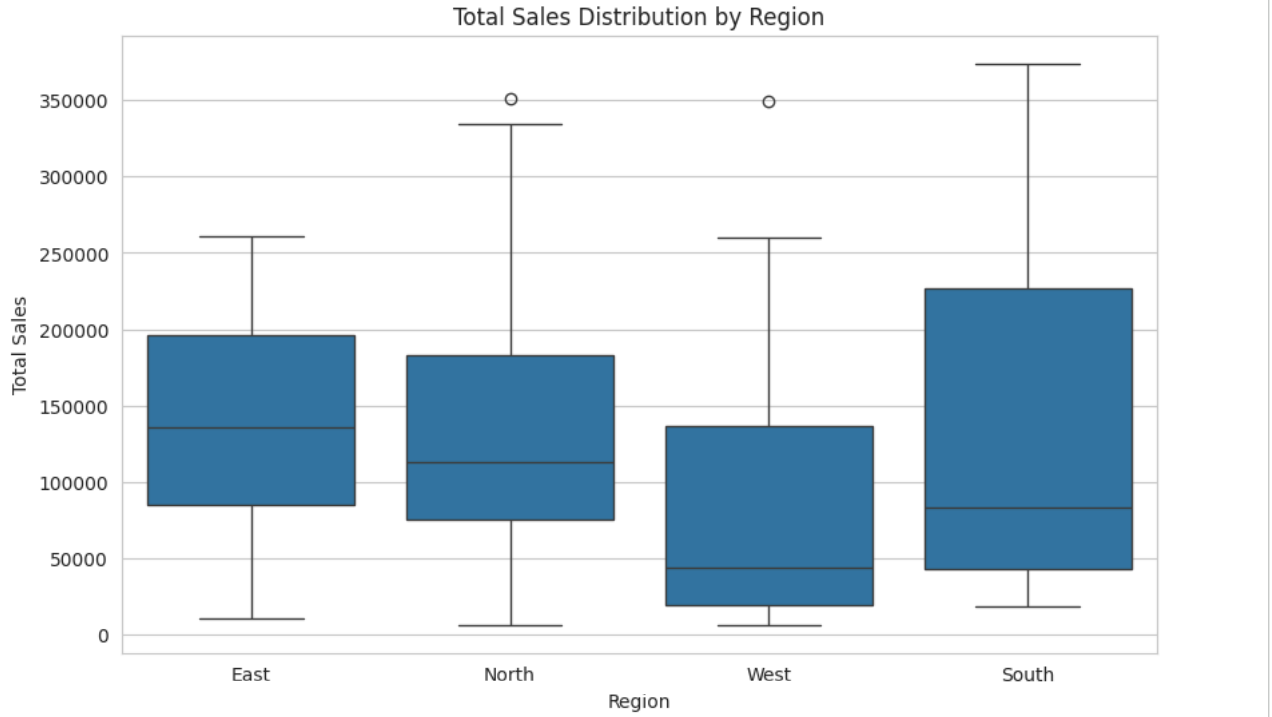


Reasoning: The previous code generated a `UserWarning` because `plt.legend()` was called for a boxplot without specific artists providing labels for a legend. The x-axis labels for the regions already serve as the categories' labels, making a separate legend redundant and causing the warning. To fix this warning and avoid a superfluous legend, I will remove the `plt.legend()` call.

```
from scipy.stats import f_oneway
import matplotlib.pyplot as plt
import seaborn as sns

# Identify unique regions
regions = sales_df['Region'].unique()

# Prepare data for ANOVA
region_sales = [sales_df['Total_Sales'][sales_df['Region'] == region] for region in regions]

# Perform ANOVA test
f_stat, p_val = f_oneway(*region_sales)

print(f"### ANOVA Test for Total_Sales by Region:")
print(f"  F-statistic: {f_stat:.4f}")
print(f"  P-value: {p_val:.4f}")

# Interpret the p-value
alpha = 0.05
if p_val < alpha:
    print(f"  Conclusion: Reject the null hypothesis. There is a significant difference in mean Total_Sales acro
else:
    print(f"  Conclusion: Fail to reject the null hypothesis. There is no significant difference in mean Total_S

# Visualize the distribution of Total_Sales across regions
plt.figure(figsize=(10, 6))
sns.boxplot(x='Region', y='Total_Sales', data=sales_df)
plt.title('Total Sales Distribution by Region')
plt.xlabel('Region')
plt.ylabel('Total Sales')
# Removed plt.legend() as x-axis labels already act as category labels for a boxplot,
```

```
# and a legend without explicit artists causes a UserWarning.
plt.show()
```

```
### ANOVA Test for Total_Sales by Region:
  F-statistic: 2.1644
  P-value: 0.0972
  Conclusion: Fail to reject the null hypothesis. There is no significant difference in mean Total_Sales across r
```



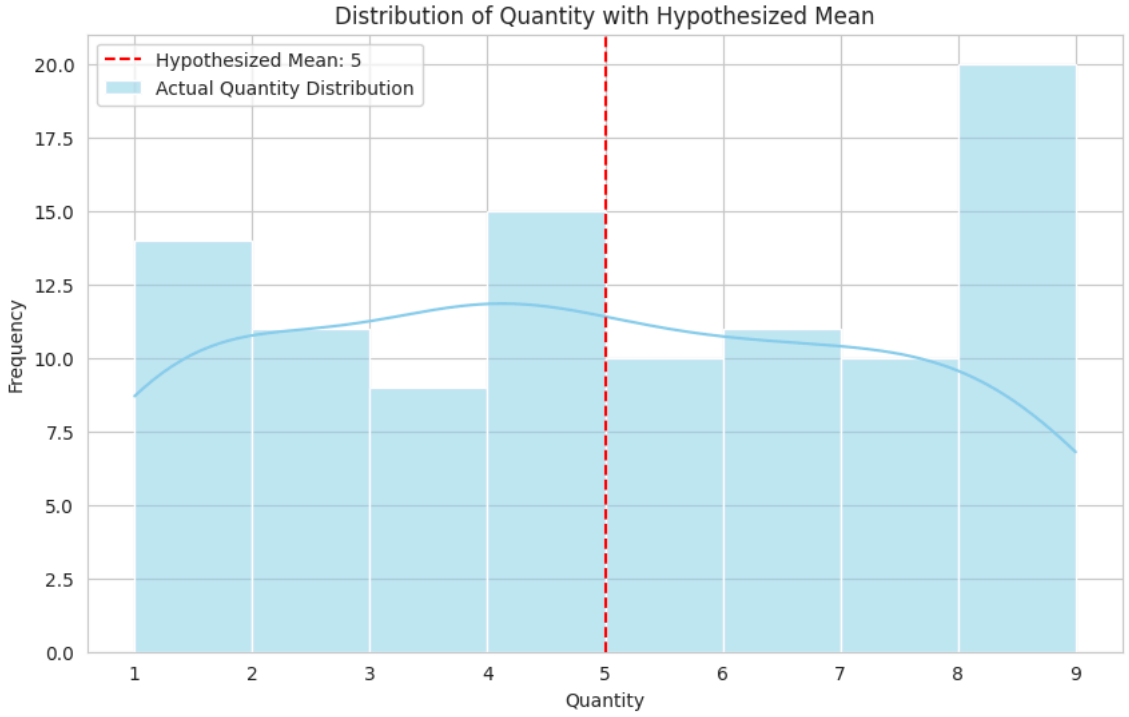
One-Sample t-test: Quantity

Null Hypothesis (H0): The mean `Quantity` is equal to a hypothesized value (e.g., 5). **Alternative Hypothesis (H1):** The mean `Quantity` is significantly different from the hypothesized value (e.g., 5).

Reasoning: Now, I will implement the one-sample t-test for the 'Quantity' column using a hypothesized mean of 5, as formulated previously. I will also visualize the distribution of 'Quantity' with the hypothesized mean.

```
from scipy.stats import ttest_1samp
import matplotlib.pyplot as plt
import seaborn as sns

# Hypothesized mean for Quantity
hypothesized_mean_quantity = 5

# Perform one-sample t-test
t_stat, p_val = ttest_1samp(sales_df['Quantity'], hypothesized_mean_quantity)

print(f"### One-Sample t-test for Quantity (Hypothesized Mean = {hypothesized_mean_quantity}):")
print(f"  T-statistic: {t_stat:.4f}")
print(f"  P-value: {p_val:.4f}")

# Interpret the p-value
alpha = 0.05
if p_val < alpha:
    print(f"  Conclusion: Reject the null hypothesis. The mean Quantity is significantly different from {hypothe
else:
    print(f"  Conclusion: Fail to reject the null hypothesis. The mean Quantity is not significantly different f

# Visualize the distribution of Quantity with the hypothesized mean
plt.figure(figsize=(10, 6))
sns.histplot(sales_df['Quantity'], kde=True, color='skyblue', label='Actual Quantity Distribution')
plt.axvline(hypothesized_mean_quantity, color='red', linestyle='--', label=f'Hypothesized Mean: {hypothesized_me
plt.title('Distribution of Quantity with Hypothesized Mean')
plt.xlabel('Quantity')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

```
### One-Sample t-test for Quantity (Hypothesized Mean = 5):
  T-statistic: -0.8500
  P-value: 0.3974
  Conclusion: Fail to reject the null hypothesis. The mean Quantity is not significantly different from 5 (p >= 0
```



Independent Samples t-test: Price between two Products

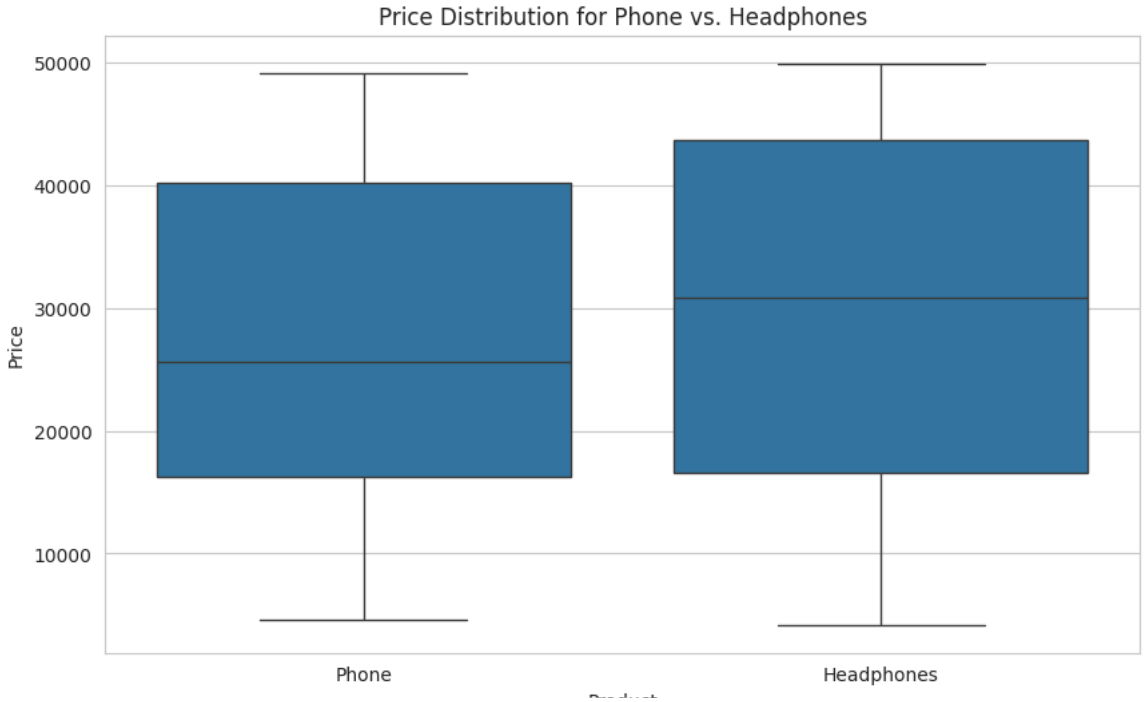
Null Hypothesis (H0): There is no significant difference in the mean `Price` between two specific products (e.g., 'Phone' and 'Headphones'). **Alternative Hypothesis (H1):** There is a significant difference in the mean `Price` between two specific products (e.g., 'Phone' and 'Headphones').

Reasoning: Next, I will write the Python code to perform the independent samples t-test to compare 'Price' between two specific products ('Phone' and 'Headphones'), as formulated previously. I will also visualize the price distributions for these two products using box plots to complement the hypothesis test.

```
from scipy.stats import ttest_ind
import matplotlib.pyplot as plt
import seaborn as sns

# Define the two products for comparison
product1 = 'Phone'
product2 = 'Headphones'

# Filter data for the two products
price_product1 = sales_df[sales_df['Product'] == product1]['Price']
price_product2 = sales_df[sales_df['Product'] == product2]['Price']

# Check if there's enough data for both products
if len(price_product1) == 0 or len(price_product2) == 0:
    print(f"Error: Not enough data for one or both products ({product1}, {product2}).")
else:
    # Perform independent samples t-test
    t_stat, p_val = ttest_ind(price_product1, price_product2, equal_var=False) # Assuming unequal variances base

    print(f"### Independent Samples t-test for Price between {product1} and {product2}:")
    print(f"  Product 1 ({product1}) mean price: {price_product1.mean():.2f}")
    print(f"  Product 2 ({product2}) mean price: {price_product2.mean():.2f}")
    print(f"  T-statistic: {t_stat:.4f}")
    print(f"  P-value: {p_val:.4f}")

    # Interpret the p-value
    alpha = 0.05
    if p_val < alpha:
        print(f"  Conclusion: Reject the null hypothesis. There is a significant difference in mean Price betwee
    else:
        print(f"  Conclusion: Fail to reject the null hypothesis. There is no significant difference in mean Pri

    # Visualize the distribution of Price for the two products
    plt.figure(figsize=(10, 6))
```

```
    sns.boxplot(x='Product', y='Price', data=sales_df[sales_df['Product'].isin([product1, product2])])
    plt.title(f'Price Distribution for {product1} vs. {product2}')
    plt.xlabel('Product')
    plt.ylabel('Price')
    plt.show()
```

```
### Independent Samples t-test for Price between Phone and Headphones:
  Product 1 (Phone) mean price: 27379.00
  Product 2 (Headphones) mean price: 28692.13
  T-statistic: -0.2635
  P-value: 0.7941
  Conclusion: Fail to reject the null hypothesis. There is no significant difference in mean Price between Phone
```



Calculate Confidence Intervals

Subtask:

Calculate the 95% confidence intervals for the mean of key numerical columns such as 'Total_Sales', 'Price', and 'Quantity' to determine the margin of error for these estimates.

Reasoning: To calculate the 95% confidence intervals for the mean of the specified numerical columns, I will import the `t` class from `scipy.stats` and iterate through the columns. For each column, I will compute the mean, standard error of the mean (SEM), degrees of freedom (df), and then use `t.interval` to get the confidence interval, finally printing the results.

```
from scipy.stats import t
import numpy as np

numerical_cols = ['Total_Sales', 'Price', 'Quantity']

print("### 95% Confidence Intervals for the Mean of Numerical Columns:")
for col in numerical_cols:
    data = sales_df[col]

    # Calculate mean
    mean = data.mean()

    # Calculate standard error of the mean (SEM)
    # Using scipy.stats.sem directly is often simpler
    sem = data.sem()

    # Calculate degrees of freedom
    df = data.count() - 1

    # Calculate the 95% confidence interval for the mean
    # alpha = 0.95 corresponds to a 95% confidence level
    confidence_interval = t.interval(confidence=0.95, df=df, loc=mean, scale=sem)

    print(f"\n  {col}:")
    print(f"    Mean: {mean:.2f}")
```

```
    print(f"    Standard Error of Mean (SEM): {sem:.2f}")
    print(f"    Degrees of Freedom (df): {df}")
    print(f"    95% Confidence Interval: ({confidence_interval[0]:.2f}, {confidence_interval[1]:.2f})")
```

```
### 95% Confidence Intervals for the Mean of Numerical Columns:

  Total_Sales:
    Mean: 123650.48
    Standard Error of Mean (SEM): 10016.11
    Degrees of Freedom (df): 99
    95% Confidence Interval: (103776.35, 143524.61)

  Price:
    Mean: 25808.51
    Standard Error of Mean (SEM): 1391.76
    Degrees of Freedom (df): 99
    95% Confidence Interval: (23046.95, 28570.07)

  Quantity:
    Mean: 4.78
    Standard Error of Mean (SEM): 0.26
    Degrees of Freedom (df): 99
    95% Confidence Interval: (4.27, 5.29)
```

Perform Regression Analysis

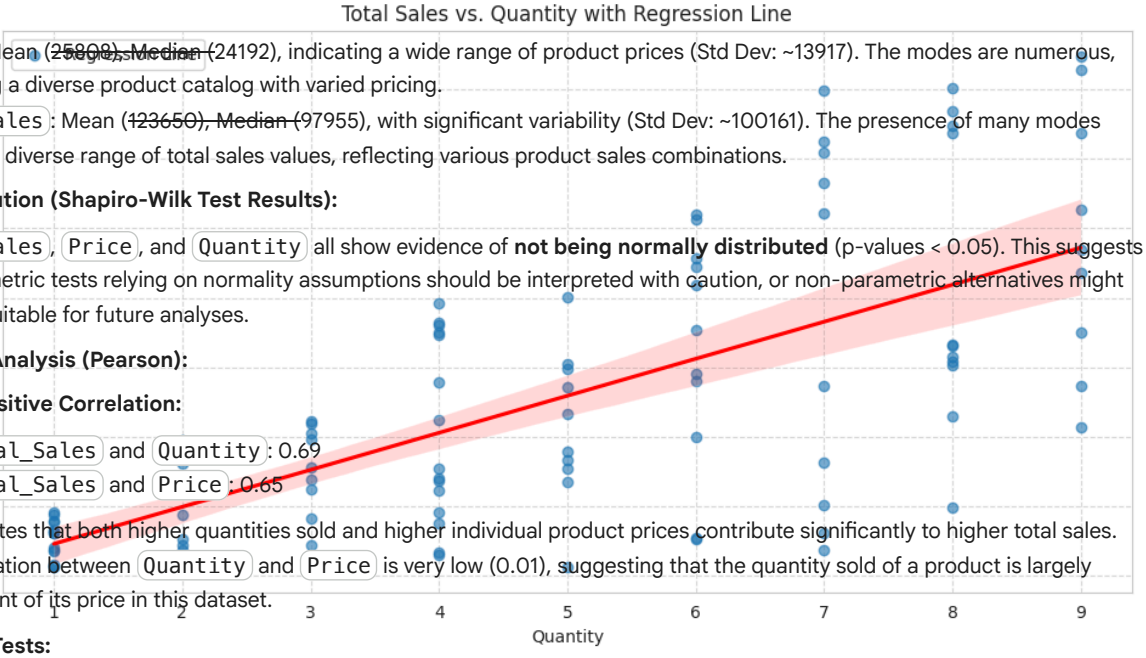
Subtask:

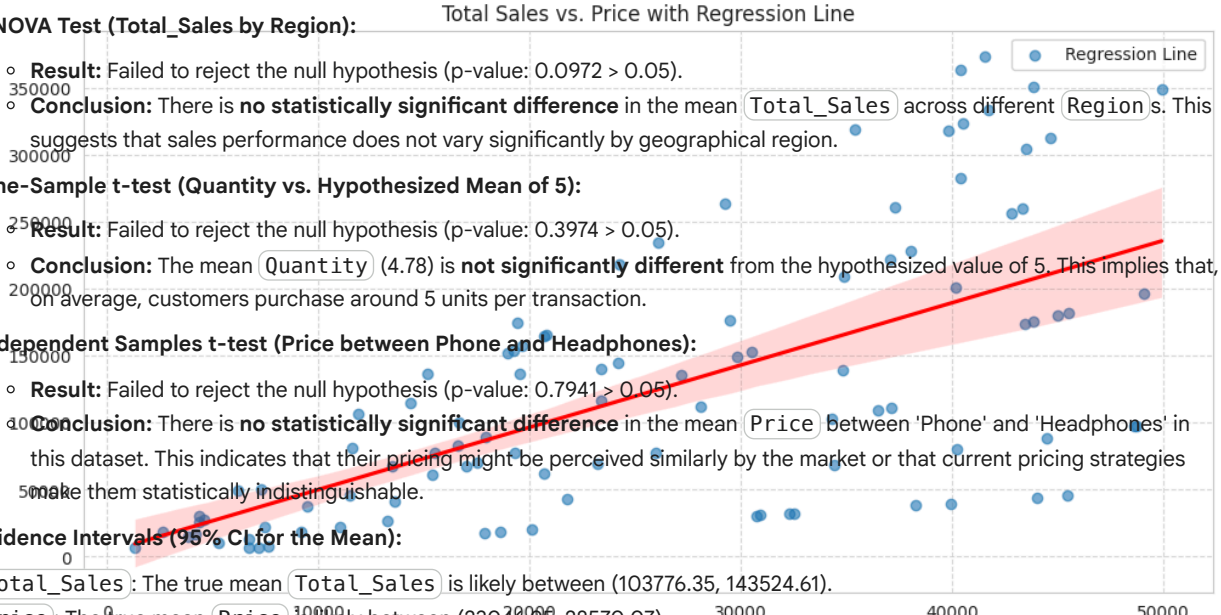
Conduct a linear regression analysis to model 'Total_Sales' as the dependent variable, using 'Quantity' and 'Price' as independent variables. Calculate and display the R-squared value, coefficients, and other relevant regression statistics. Make sure to include legends for all plots.

Reasoning: To conduct the linear regression analysis and visualize the results, I will import the necessary libraries, define the regression formula, fit the model, print its summary, and then generate scatter plots with regression lines for 'Total_Sales' against 'Quantity' and 'Price', ensuring all plots have labels, titles, and legends as requested.

```
from statsmodels.formula.api import ols
import matplotlib.pyplot as plt
import seaborn as sns

# 2. Define the regression formula
formula = 'Total_Sales ~ Quantity + Price'

# 3. Fit a linear regression model
model = ols(formula, data=sales_df).fit()

# 4. Print the summary of the regression model
print("### Linear Regression Model Summary:")
print(model.summary())

# 5. Create a scatter plot of 'Total_Sales' versus 'Quantity' with regression line
plt.figure(figsize=(12, 6))
sns.regplot(x='Quantity', y='Total_Sales', data=sales_df, scatter_kws={'alpha':0.6}, line_kws={'color':'red'}, l
plt.title('Total Sales vs. Quantity with Regression Line')
plt.xlabel('Quantity')
plt.ylabel('Total Sales')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()

# 6. Create another scatter plot of 'Total_Sales' versus 'Price' with regression line
plt.figure(figsize=(12, 6))
sns.regplot(x='Price', y='Total_Sales', data=sales_df, scatter_kws={'alpha':0.6}, line_kws={'color':'red'}, labe
plt.title('Total Sales vs. Price with Regression Line')
plt.xlabel('Price')
plt.ylabel('Total Sales')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

```
### Linear Regression Model Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:            Total_Sales   R-squared:                       0.884
Model:                            OLS   Adj. R-squared:                  0.882
Method:                 Least Squares   F-statistic:                     369.3
Date:                Thu, 29 Jan 2026   Prob (F-statistic):           4.40e-46
Time:                        15:46:48   Log-Likelihood:                -1185.2
No. Observations:                 100   AIC:                             2376.
Df Residuals:                      97   BIC:                             2384.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -1.212e+05   9668.204    -12.536      0.000   -1.41e+05   -1.02e+05
Quantity     2.643e+04   1338.869     19.741      0.000    2.38e+04    2.91e+04
Price           4.6106      0.249     18.518      0.000       4.116       5.105
==============================================================================
Omnibus:                        1.152   Durbin-Watson:                   2.057
Prob(Omnibus):                  0.562   Jarque-Bera (JB):                1.077
Skew:                           0.086   Prob(JB):                        0.584
Kurtosis:                       2.522   Cond. No.                     8.25e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.25e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```





Generate Business Insights and Recommendations

Subtask:

Summarize the key findings from all statistical analyses, translate them into actionable business insights, and provide practical recommendations based on the data-driven conclusions.

Generate Business Insights and Recommendations

Subtask:

Summarize the key findings from all statistical analyses, translate them into actionable business insights, and provide practical recommendations based on the data-driven conclusions.

Key Findings:

1. Descriptive Statistics:

- `Quantity`: Mean (4.78), Median (5.00), with a moderate spread (Std Dev: 2.59). Mode is 4, suggesting a common purchase quantity.
- `Price`: Mean (25808), Median (24192), indicating a wide range of product prices (Std Dev: ~13917). The modes are numerous, suggesting a diverse product catalog with varied pricing.
- `Total_Sales`: Mean (123650), Median (97955), with significant variability (Std Dev: ~100161). The presence of many modes indicates a diverse range of total sales values, reflecting various product sales combinations.

2. Data Distribution (Shapiro-Wilk Test Results):

- `Total_Sales`, `Price`, and `Quantity` all show evidence of **not being normally distributed** (p-values < 0.05). This suggests that parametric tests relying on normality assumptions should be interpreted with caution, or non-parametric alternatives might be more suitable for future analyses.

3. Correlation Analysis (Pearson):

- **Strong Positive Correlation:**
  - `Total_Sales` and `Quantity`: 0.69
  - `Total_Sales` and `Price`: 0.65
- This indicates that both higher quantities sold and higher individual product prices contribute significantly to higher total sales. The correlation between `Quantity` and `Price` is very low (0.01), suggesting that the quantity sold of a product is largely independent of its price in this dataset.

4. Hypothesis Tests:

- **ANOVA Test (Total_Sales by Region):**
  - **Result:** Failed to reject the null hypothesis (p-value: 0.0972 > 0.05).
  - **Conclusion:** There is **no statistically significant difference** in the mean `Total_Sales` across different `Region`s. This suggests that sales performance does not vary significantly by geographical region.
- **One-Sample t-test (Quantity vs. Hypothesized Mean of 5):**
  - **Result:** Failed to reject the null hypothesis (p-value: 0.3974 > 0.05).
  - **Conclusion:** The mean `Quantity` (4.78) is **not significantly different** from the hypothesized value of 5. This implies that, on average, customers purchase around 5 units per transaction.
- **Independent Samples t-test (Price between Phone and Headphones):**
  - **Result:** Failed to reject the null hypothesis (p-value: 0.7941 > 0.05).
  - **Conclusion:** There is **no statistically significant difference** in the mean `Price` between 'Phone' and 'Headphones' in this dataset. This indicates that their pricing might be perceived similarly by the market or that current pricing strategies make them statistically indistinguishable.

5. Confidence Intervals (95% CI for the Mean):

- `Total_Sales`: The true mean `Total_Sales` is likely between (103776.35, 143524.61).
- `Price`: The true mean `Price` is likely between (23046.95, 28570.07).
- `Quantity`: The true mean `Quantity` is likely between (4.27, 5.29).