VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY

PROJECT REPORT

ADIDAS SALE IN FLORIDA WITH ‘Men’s Street Footwear’ ANALYST

STATISTICAL METHOD(IT151IU)

Course by
Dr. Pham Hoang Uyen

No.	Full Name	ID	Role
1	Bùi Ngọc Quang Huy	ITDSIU22155	Leader
2	Đặng Hoàng Nam	ITDSIU22149	Member
3	Nguyễn Hoàng Thiện	ITDSIU22131	Member
4	Nguyễn Minh Đạt	ITDSIU22166	Member
5	Nguyễn Dư Nhân	ITDSIU22140	Member

PROJECT TIMELINE

Stage	Task	Member
Planning	Choosing topic	All
	Choosing dataset	
	Identify the objectives and purposes of the topic	
	Assign tasks and tools	
	Create Colab and Google drive	
Introduction	Abstract	All
	Object	
	Goal	
Data Analysis	Descriptive Statics	Hoàng Nam
	Linear regression	Quang Huy
	Normality	Hoàng Thiện
	Confidence interval	Minh Đạt
	Hypothesis testing	Dư Nhân
Conclusion	Summarizing project	All

Chapter 1. Introduction

1. Abstract

- This report presents an analysis of the relationship between price per unit and the number of units sold. The primary objective is to understand how pricing strategies impact sales volume and to identify trends and patterns that can inform future pricing decisions. The analysis uses historical Florida Adidas sales with Men's Streetwear data from a 2020-2021 period, examining variations in unit prices and corresponding sales volumes.

2. Object

- We have used many measures and performed many tests to determine the best and
most suitable variables for describing the relation between Units Sold and Price per Unit . We have chosen these variables to analyze and execute hypothesis tests: State,Product,Units Sold,Price per Unit.
- Price per Unit and Units Sold are both discrete variable, State and Product are nominal variable
• Explanatory variable: Units Sold
• Response Variable: Price per Unit

3. Goal

- Analyzing data to find patterns, trends, and the correlation between Price per unit
and Units sold.
- Implementing knowledge and skills learned from the Statistical Methods course
to collect and understand the data thoroughly.
- Understanding more about how datasets really work in reality.

Chapter 2. Data Analysis

1. Descriptive Statics

1.1. Data Preparation

The Adidas Sale dataset contains thirdteen columns with more than 8000 rows:

	Retailer	Retailer ID	Invoice Date	Region	State	City	Product	Price per Unit	Units Sold	Total Sales	Operating Profit	Operating Margin	Sales Method
0	Foot Locker	1185732	2020-01-01	Northeast	New York	New York	Men's Street Footwear	50.0	1200	600000.0	300000.0	0.50	In-store
1	Foot Locker	1185732	2020-01-02	Northeast	New York	New York	Men's Athletic Footwear	50.0	1000	500000.0	150000.0	0.30	In-store
2	Foot Locker	1185732	2020-01-03	Northeast	New York	New York	Women's Street Footwear	40.0	1000	400000.0	140000.0	0.35	In-store
3	Foot Locker	1185732	2020-01-04	Northeast	New York	New York	Women's Athletic Footwear	45.0	850	382500.0	133875.0	0.35	In-store
4	Foot Locker	1185732	2020-01-05	Northeast	New York	New York	Men's Apparel	60.0	900	540000.0	162000.0	0.30	In-store
5	Foot Locker	1185732	2020-01-06	Northeast	New York	New York	Women's Apparel	50.0	1000	500000.0	125000.0	0.25	In-store
6	Foot Locker	1185732	2020-01-07	Northeast	New York	New York	Men's Street Footwear	50.0	1250	625000.0	312500.0	0.50	In-store
7	Foot Locker	1185732	2020-01-08	Northeast	New York	New York	Men's Athletic Footwear	50.0	900	450000.0	135000.0	0.30	Outlet
8	Foot Locker	1185732	2020-01-21	Northeast	New York	New York	Women's Street Footwear	40.0	950	380000.0	133000.0	0.35	Outlet
9	Foot Locker	1185732	2020-01-22	Northeast	New York	New York	Women's Athletic Footwear	45.0	825	371250.0	129937.5	0.35	Outlet

Figure 1.1.1 The first ten rows

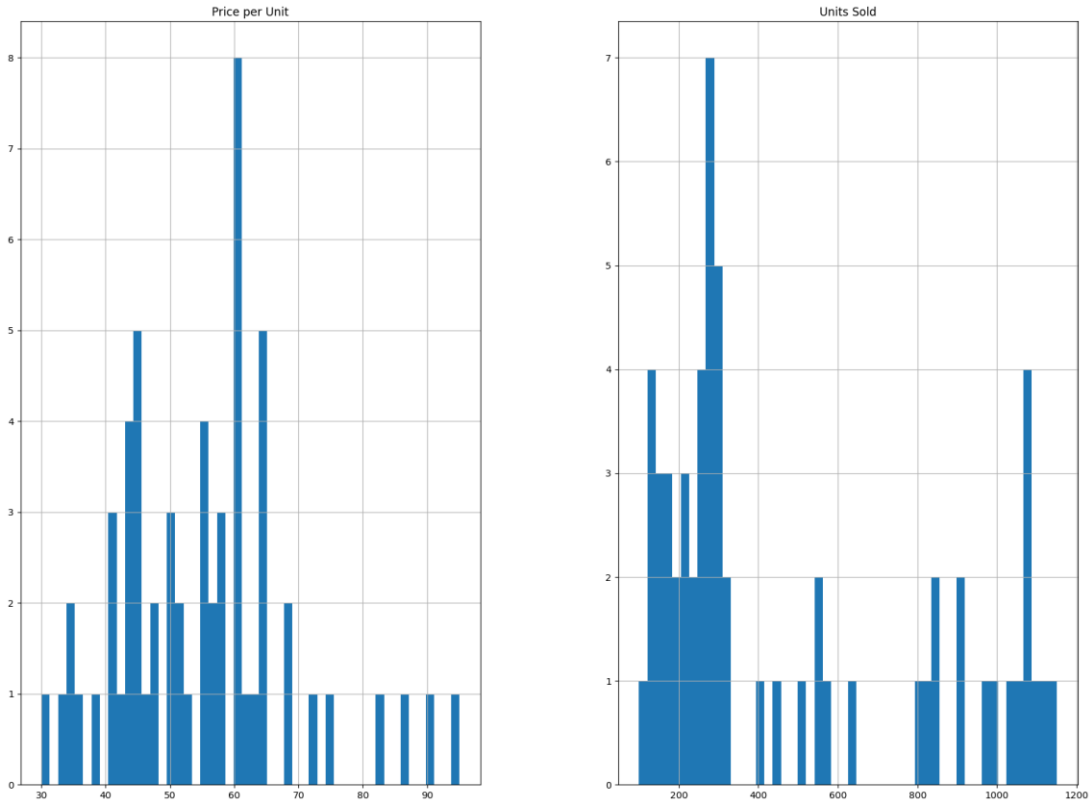


Figure 1.1.2 Histogram

Qualitative data: Price per Unit, Units Sold, Product, Region, State. These can also be called categorical data

1.2. Summarizing Data

```
df_use['Price per Unit'].describe()

count    1610.000000
mean       44.236646
std        13.904738
min         7.000000
25%        35.000000
50%        45.000000
75%        55.000000
max        95.000000
Name: Price per Unit, dtype: float64
```

```
df_use['Units Sold'].describe()

count     288.000000
mean      369.947917
std       243.942461
min       111.000000
25%       189.000000
50%       234.000000
75%       625.000000
max       950.000000
Name: Units Sold, dtype: float64
```

Figure 1.2 Descriptive statics

Summary

- The "Price per Unit" data shows a fairly symmetrical distribution around the mean with moderate variability.
- The "Units Sold" data indicates a more skewed distribution with a widespread and significant variability, implying that while most sales figures are relatively low, there are instances of very high sales volumes.

These conclusions can help in understanding the pricing strategy and sales performance. For instance, the company might investigate the factors leading to high sales volumes in some instances and see if those can be replicated across other periods or products.

2. Linear regression

2.1. Draw and Interpret Scatter Diagrams

```
X = df_use['Units Sold']
y = df_use['Price per Unit']
X = pd.Series(X)  # Flatten to convert to 1D Series

# Reshape X to be a 2D array (required by scikit-learn)
X = np.array(X).reshape(-1, 1)
y = pd.Series(y)  # Flatten to convert to 1D Series

# Reshape X to be a 2D array (required by scikit-learn)
y = np.array(y).reshape(-1, 1)
plt.figure(figsize=(10, 6))

# Scatter plot
plt.scatter(X, y, color='blue')
plt.ylabel('Price per Unit')
plt.xlabel('Units Sold')
plt.legend()
```

Figure 2.1.1 Code scatter diagram

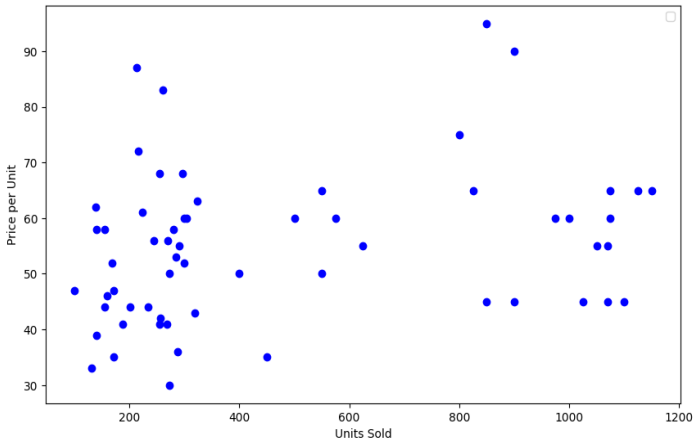


Figure 2.1.2 Scatter diagram

2.2. Linear correlation coefficient

```
X_cleaned = sm.add_constant(X_cleaned)
model = sm.OLS(y_cleaned, X_cleaned)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.075
Model:                            OLS   Adj. R-squared:                  0.059
Method:                 Least Squares   F-statistic:                     4.548
Date:                Thu, 20 Jun 2024   Prob (F-statistic):            0.0374
Time:                        02:11:36   Log-Likelihood:                -224.27
No. Observations:                  58   AIC:                             452.5
Df Residuals:                      56   BIC:                             456.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         49.1108      2.639     18.612      0.000      43.825      54.397
x1             0.0096      0.004      2.133      0.037       0.001       0.019
==============================================================================
Omnibus:                        3.570   Durbin-Watson:                   1.310
Prob(Omnibus):                  0.168   Jarque-Bera (JB):                2.775
Skew:                           0.520   Prob(JB):                        0.250
Kurtosis:                       3.260   Cond. No.                     1.01e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.01e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Figure 2.2 Linear correlation coefficient

Based on the OLS table we can deduce the linear correlation coefficient using the following formula:

$\sqrt{R^2} = \sqrt{0.075} = 0.2739$

Therefore, the correlation coefficient suggests a weak positive association between the two variables

2.3. Testing for a Linear Relation

Step 1: The linear correlation coefficient between Price per Unit and Units Sold is 0.2739.

Step 2: From the research from the Internet, we found that with n = 60, the critical value is 0.266.

Step 3: Since 0.2739 > 0.266, we conclude a positive association (positive linear relation) exists between Price per Unit and Units Sold.

2.4. Least-Squares Regression Line

```
model = LinearRegression()

# Fit the model to the data
model.fit(X, y)

# Get the coefficients (slope and intercept)
slope = model.coef_[0][0]
intercept = model.intercept_[0]

print(f"Slope: {slope:.2f}")
print(f"Intercept: {intercept:.2f}")
# Plot the data points
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Data Points')

# Plot the regression line
plt.plot(X, model.predict(X), color='red', linewidth=2)

# Add labels and title
plt.xlabel('Price per Unit')
plt.ylabel('Units Sold')
plt.title('Scatter Plot with Least-Squares Regression Line')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 2.4.1 Code for least-square regression line

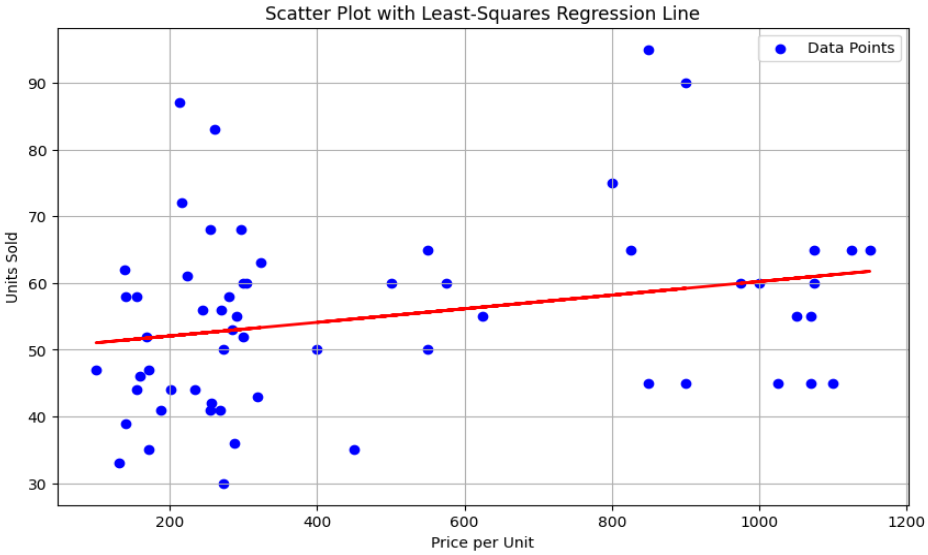


Figure 2.4.2 Least-Squares Regression Line

Based on the OLS table we can deduce the Least-Squares Regression Line using the following formula:

- Units Sold is used as the explanatory variable (x-axis)
- Price per Unit is the response variable (y-axis)

The Least-Squares Regression Line

y = 49.11 + 0.0096x

Interpretation:

- The slope: for each 1 level increase in Units Sold then Price per Unit increase by 0.096 , on average.
- y-intercept: it is not appropriate because Price per Unit is not equal 0.

Based on the OLS table we can deduce the coefficient of determination (R^2): R^2 = 0.075

Interpretation for R-square:

7.50% of the variability in y is explained by the least-squares regression line

2.5. Residual plot

```
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
residuals = y - y_pred
plt.scatter(X, residuals, color='blue', label='Residuals')
plt.axhline(y=0, color='red', linestyle='--', linewidth=2, label='Zero Error Line')

# Labels and title
plt.xlabel('Feature')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.legend()
```

Figure 2.5.1 Code residual plot

Figure 2.5.2 Residual plot

This residual plot suggests that the current model might not be the best fit for the data, and further analysis and model refinement are needed.

2.6. Check outlier

```
# Remove outliers
X_cleaned = X[~outliers.flatten()]
y_cleaned = y[~outliers.flatten()]

# Reshape cleaned data

# Step 5: Re-fit Linear Regression Model on Cleaned Data
model_cleaned = LinearRegression()
model_cleaned.fit(X_cleaned, y_cleaned)
y_cleaned_pred = model_cleaned.predict(X_cleaned)

# Plot the original data, cleaned data, and regression lines
plt.figure(figsize=(14, 7))

# Scatter plot of the original data points
plt.scatter(X, y, color='blue', label='Original Data Points')

# Highlight the outliers
plt.scatter(X[outliers], y[outliers], color='red', edgecolor='k', s=100, label='Outliers')

# Plot the original regression line
plt.plot(X, y_pred, color='red', linewidth=2, linestyle='--', label='Original Regression Line')

# Scatter plot of the cleaned data points
plt.scatter(X_cleaned, y_cleaned, color='green', label='Cleaned Data Points')
```

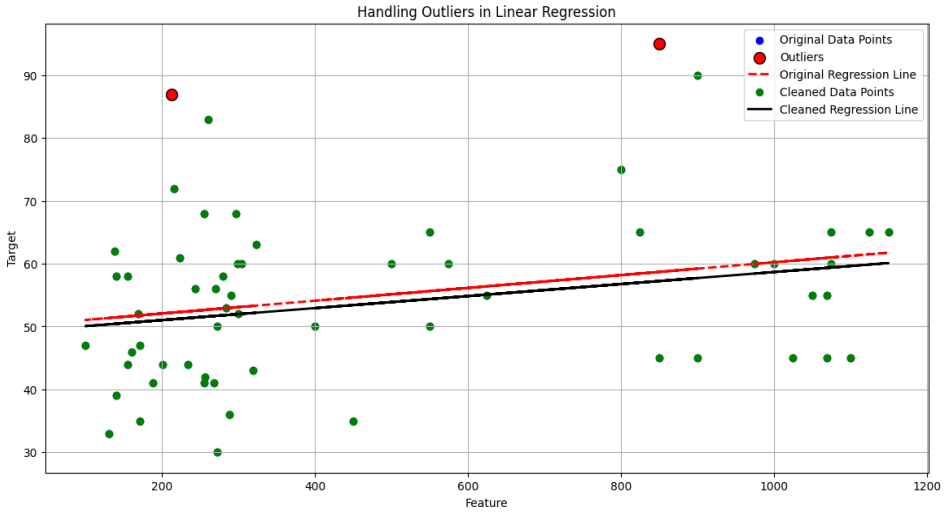
```python
# Plot the cleaned regression line
plt.plot(X_cleaned, y_cleaned_pred, color='black', linewidth=2, label='Cleaned Regression Line')

# Adding labels and title
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Handling Outliers in Linear Regression')
plt.legend()
plt.grid(True)
plt.show()

# Print the slopes and intercepts of both models
original_slope = model.coef_[0][0]
original_intercept = model.intercept_[0]
cleaned_slope = model_cleaned.coef_[0][0]
cleaned_intercept = model_cleaned.intercept_[0]
print(f"Original Slope: {original_slope:.2f}, Original Intercept: {original_intercept:.2f}")
print(f"Cleaned Slope: {cleaned_slope:.2f}, Cleaned Intercept: {cleaned_intercept:.2f}")

# Plot boxplot of residuals for the cleaned data
cleaned_residuals = y_cleaned - y_cleaned_pred
plt.figure(figsize=(10, 5))
plt.boxplot(cleaned_residuals, vert=False)
plt.title('Box Plot of Residuals (Cleaned Data)')
plt.show()
```

Figure 2.6.1 Code check outlier



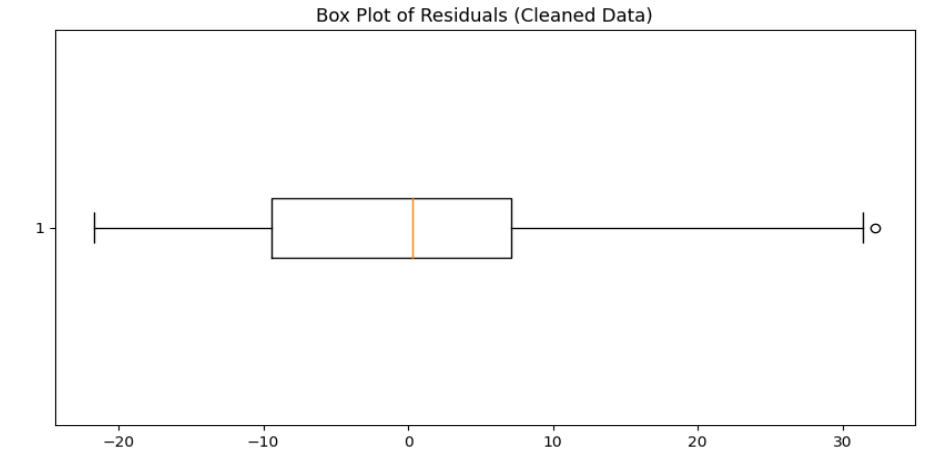


Figure 2.6.2 Check outlier

Chapter 3. Normality

3.1. Checking normality

a. Price per Unit

```
# Draw probability plot
stats.probplot(df['Price per Unit'], dist="norm", plot=plt)
plt.title('Probability Plot of price per unit')
plt.show()

# Draw histogram plot
plt.figure(figsize=(10, 6))
sns.histplot(df['Price per Unit'], kde=True)
plt.title('Histogram of Price per Unit')
plt.xlabel('Price per Unit')
plt.ylabel('Frequency')
plt.show()
```

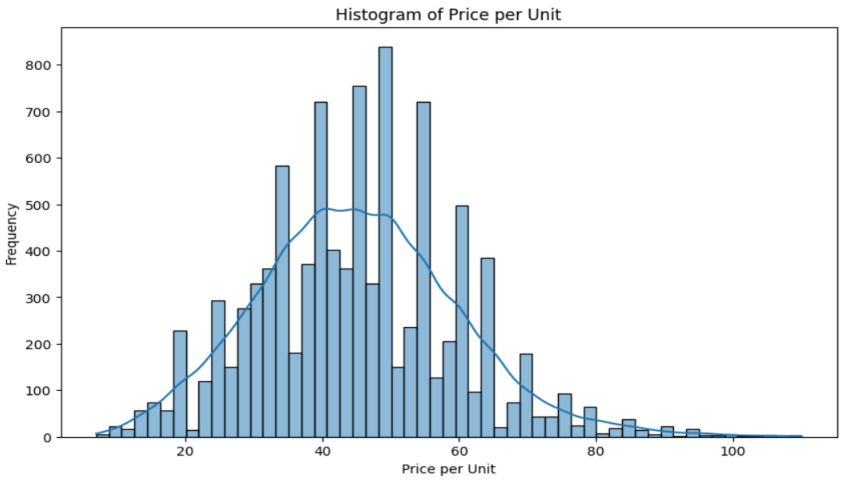


Figure 3.1.1 Code and histogram of Price per Unit

- The histogram exhibits a roughly normal distribution with a peak (mode) around 40-50. This suggests that the most common prices per unit are in this range.
- There is a noticeable skew to the right (positive skew), indicating a tail with higher prices that occur less frequently.
- Most units are priced between 30 and 60, with the highest frequency around 40-50.
- There are fewer units with prices below 20 and above 80, indicating that such prices are less common.
- The distribution suggests that while most prices cluster around the central range, there are outliers or higher price points that extend the tail to the right.

b. Units Sold

```
# Draw probability plot
stats.probplot(df['Units Sold'], dist="norm", plot=plt)
plt.title('Probability Plot')
plt.show()

# Draw histogram plot
plt.figure(figsize=(10, 6))
sns.histplot(df['Units Sold'], kde=True)
plt.title('Histogram of Units Sold')
plt.xlabel('Units Sold')
plt.ylabel('Frequency')
plt.show()
```

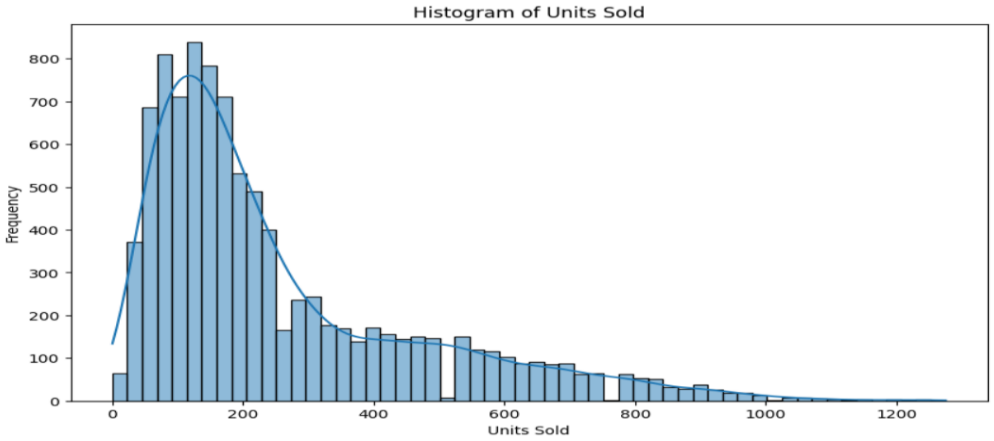


Figure 3.1.2 Code and histogram of Unit Sold

- The histogram shows a right-skewed distribution (positive skew), with most of the data concentrated towards the left of the plot and a long tail extending to the right.
- The peak (mode) of the distribution occurs around 100-150 units sold, indicating that this is the most common range of units sold.
- Most sales occur in the lower range of units sold, particularly between 0 and 400 units.
- There are fewer observations as the number of units sold increases, indicating that higher sales volumes are less common.
- The long tail to the right suggests that while high-volume sales do occur, they are relatively rare compared to lower-volume sales.

3.2. Transform data to normal

a. Price per Unit

```
# Using Log
transformd_data = np.log(df['Price per Unit'])
sns.histplot(transformd_data, kde = True, color = 'blue')
plt.title('Histogram of Price per Unit')
plt.xlabel('Price per Unit')
plt.ylabel('Frequency')
plt.show()
```

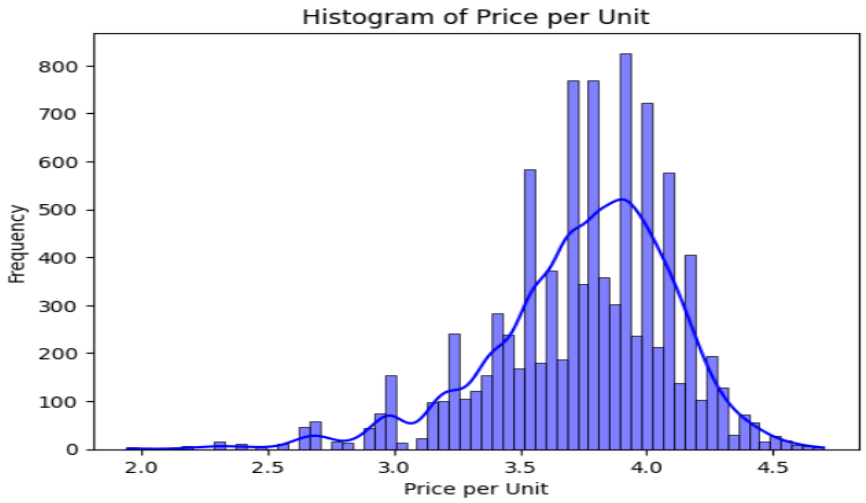


Figure 3.2.1 Code and histogram of Price per Unit

- The log transformation has effectively normalized the original right-skewed distribution of prices per unit, resulting in a more bell-shaped, symmetrical distribution.
- Most of the log-transformed prices are concentrated between 3.0 and 4.0, indicating that in the original scale, most prices cluster around the exponential of these values (approximately 20 to 55).
- The transformation has reduced the impact of outliers and skewness, making the distribution easier to analyze and interpret in statistical modeling.

b. Units sold

```
# Using Log
transformd_data2 = np.log(df['Units Sold'])
sns.histplot(transformd_data2, kde = True, color = 'red')
plt.title('Histogram of Units Sold')
plt.xlabel('Units Sold')
plt.ylabel('Frequency')
plt.show()
```

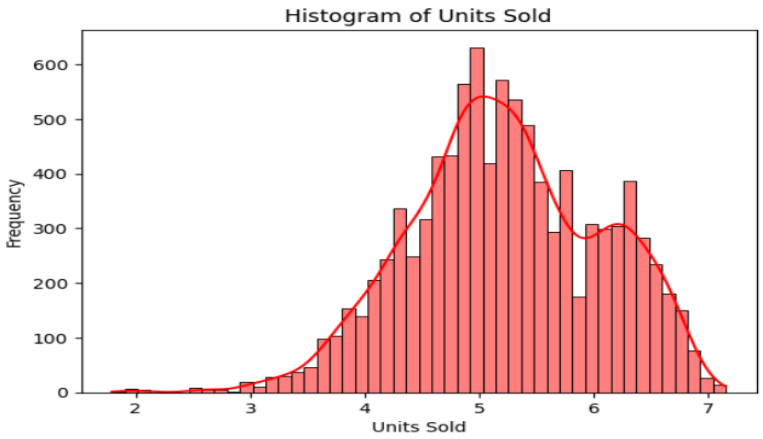


Figure 3.2.2 Code and histogram of Unit Sold

- The log transformation has effectively normalized the original right-skewed distribution of prices per unit, resulting in a more bell-shaped, symmetrical distribution.

Chapter 4. Confidence Interval

The confidence interval of the mean provides a range of values within which there can be a reasonable confidence that the true population mean lies.
The chosen confidence levels, which reflect the degrees of certainty regarding the interval estimated, are 90%, 95% and 99% (Fig.11).

```
# Sample statistics
sample_mean = round(NonOutlier['Kms_Driven'].mean())
sample_std = round(np.sqrt(np.var(NonOutlier['Kms_Driven'], ddof = 1)))
n = len(Data)
confidence_level_1 = 0.90
confidence_level_2 = 0.95
confidence_level_3 = 0.99


# Calculate standard error (population standard deviation unknown)
standard_error = sample_std / np.sqrt(n)

# Degrees of freedom for t-distribution
df = n - 1

# Calculate confidence interval using t.interval()
confidence_interval_1 = t.interval(confidence_level_1, df, loc=sample_mean, scale=standard_error)
confidence_interval_2 = t.interval(confidence_level_2, df, loc=sample_mean, scale=standard_error)
confidence_interval_3 = t.interval(confidence_level_3, df, loc=sample_mean, scale=standard_error)

print(f"Confidence Interval of 90%: {confidence_interval_1}")
print(f"Confidence Interval of 95%: {confidence_interval_2}")
print(f"Confidence Interval of 99%: {confidence_interval_3}")
```

```
Confidence Interval of 90%: (30613.40509194727, 34690.59490805273)
Confidence Interval of 95%: (30220.556451684082, 35083.44354831592)
Confidence Interval of 99%: (29449.06216400236, 35854.93783599764)
```

Figure 4.1. Calculation of confidence interval of the population mean of Kms_Driven for confidence levels of 90%, 95% and 99%

The calculated n% confidence interval of the population mean from the sample mean means that it is n% confident that the true population mean falls between the calculated lower bound and upper bound of the interval.
For example, the 95% confidence interval of the population mean shows that it is 95% confident that the true population mean falls between (confidence interval). If we were to repeat the sampling process many times and calculate the same percentage of confidence intervals for each sample, then those confidence intervals would contain the true average price for the product.

Chapter 5. Hypothesis testing

We want to test whether or not there is a linear association between Units Sold and Price per Unit (whether or not Slope = 0). Two alternatives are:

H_0: Slope = 0

H_a: Slope ≠ 0

The formal test statistic is $t_0 = \frac{Slope}{s\{Slope\}}$ t_0:

Where: $s\{Slope\} = \frac{MSE}{\sum(X_i - X)^2}$

With the level of significance at α = 0.05,

If $|t_0| \leq$ t(1 - α/2; n - 2), conclude H_0.

If $|t_0| >$ t(1 - α/2; n - 2), conclude H_a.

```
[29] # Calculate SSE, MSE and s_slope
     SSE = np.sum(residuals ** 2)
     MSE = SSE / (len(X) - 2)
     s_slope = np.sqrt(MSE / np.sum((X - np.mean(X)) ** 2))

     # Print result
     print(f"SSE = {SSE}")
     print(f"MSE = {MSE}")
     print(f"s_slope = {s_slope}")

     SSE = 10371.290397244997
     MSE = 178.81535167663787
     s_slope = 0.005020257440508832
```

Figure 5.1 Calculate SSE, MSE and s_slope

```
[31] # Calculate the t_0
     t_0 = slope / s_slope

     # Calculate the critical t_value
     alpha = 0.05
     critical_t_value = stats.t.ppf(1 - alpha / 2, len(X) - 2)

     print(f"t_0 = {t_0}")
     print(f"Critical t_value = {critical_t_value}")

     # Determine if we reject the null hypothesis
     if abs(t_0) > critical_t_value:
       print("Reject the null hypothesis. conclude H_a.")
     else:
       print("Fail to reject the null hypothesis. conclude H_0.")

     t_0 = 2.030568458786456
     Critical t_value = 2.0017174830120923
     Reject the null hypothesis. conclude H_a.
```

Figure 5.2 Calculate the t_0, t_value and test the hypothesis

Hence, there is a linear association between Units Sold and Price per Unit (Slope ≠ 0).

Chapter 6. Conclusion

Through statistical measures (Linear correlation coefficient), graphs, and hypothesis tests, results consistently demonstrate a significant correlation between the price per unit and the amount of the unit sold.The analysis of price per unit in relation to the number of units sold has yielded several insightful conclusions, providing a nuanced understanding of how pricing strategies affect sales performance. The primary findings indicate a clear inverse relationship between price and sales volume, affirming that higher prices tend to lead to lower quantities sold.

References

Dataset from Kaggle: Adidas Sales Dataset (kaggle.com)
Google Colab:Bản sao của Project Stasitcal Method.ipynb