SALE ANALYST FROM ADIDAS SALES

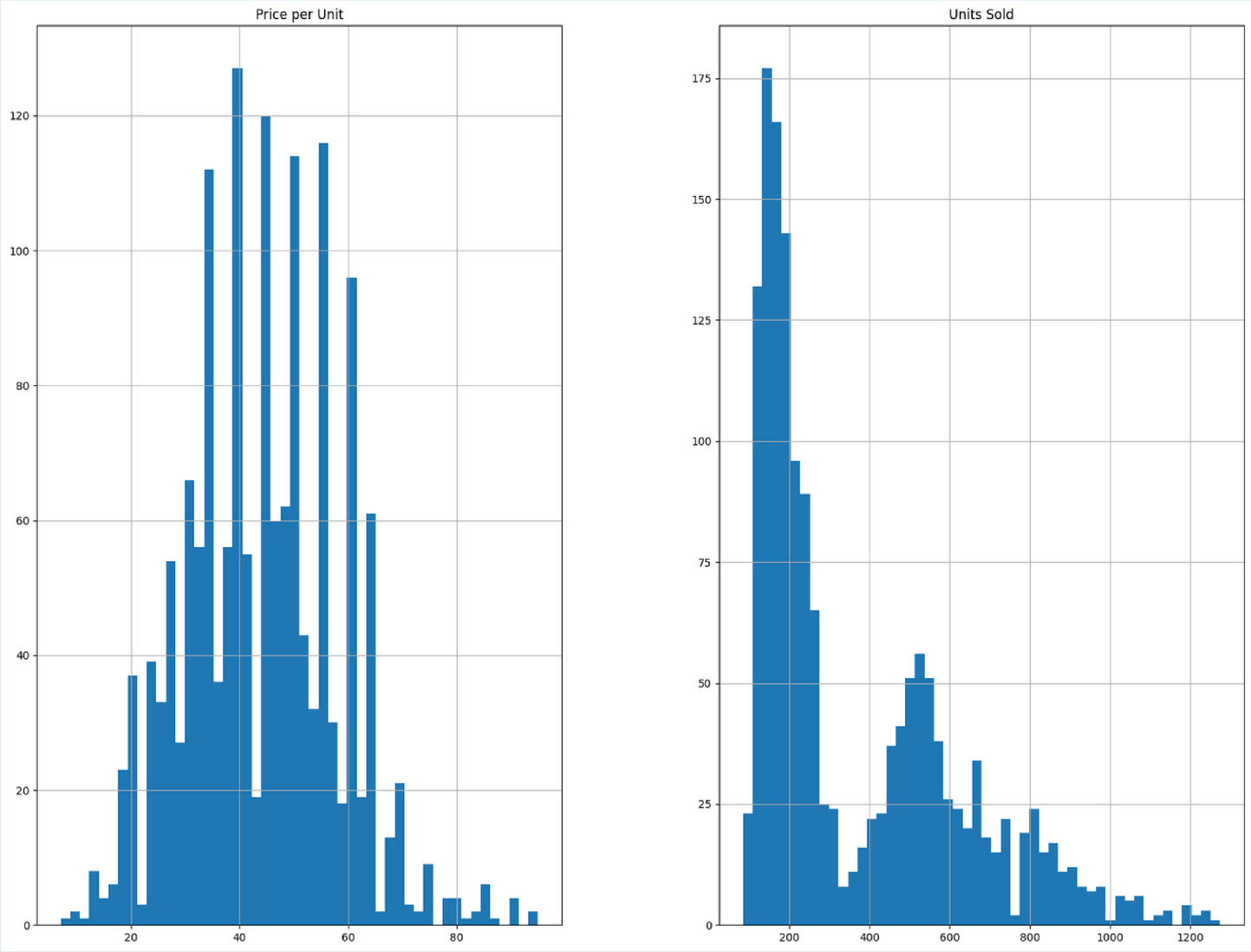
Group 1

Members

1. BÙI NGỌC QUANG HUY - ITDSIU22155
2. ĐẶNG HOÀNG NAM - ITDSIU22149
3. NGUYỄN MINH ĐẠT - ITDSIU22166
4. NGUYỄN HOÀNG THIỆN - ITDSIU22131
5. NGUYỄN DƯ NHÂN - ITDSIU22140

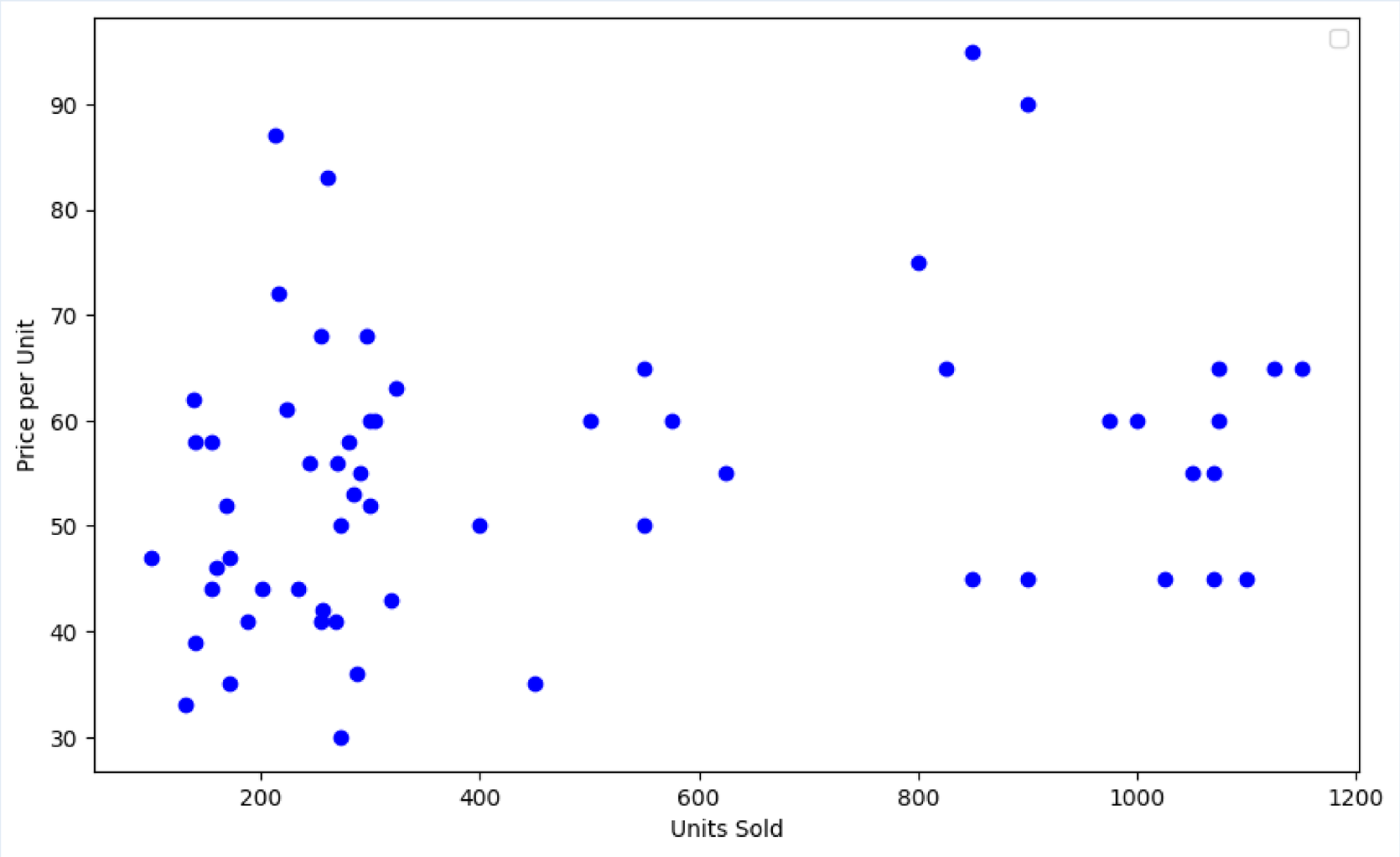
Table of content

1.DATASET


Price per Unit
0
20
40
60
80
100
120
20
40
60
80
Units Sold
0
25
50
75
100
125
150
175
200
400
600
800
1000
1200

2.LINEAR REGRESSION

Scatter diagram



2.LINEAR REGRESSION

Linear correlation coefficient

OLS Regression Results

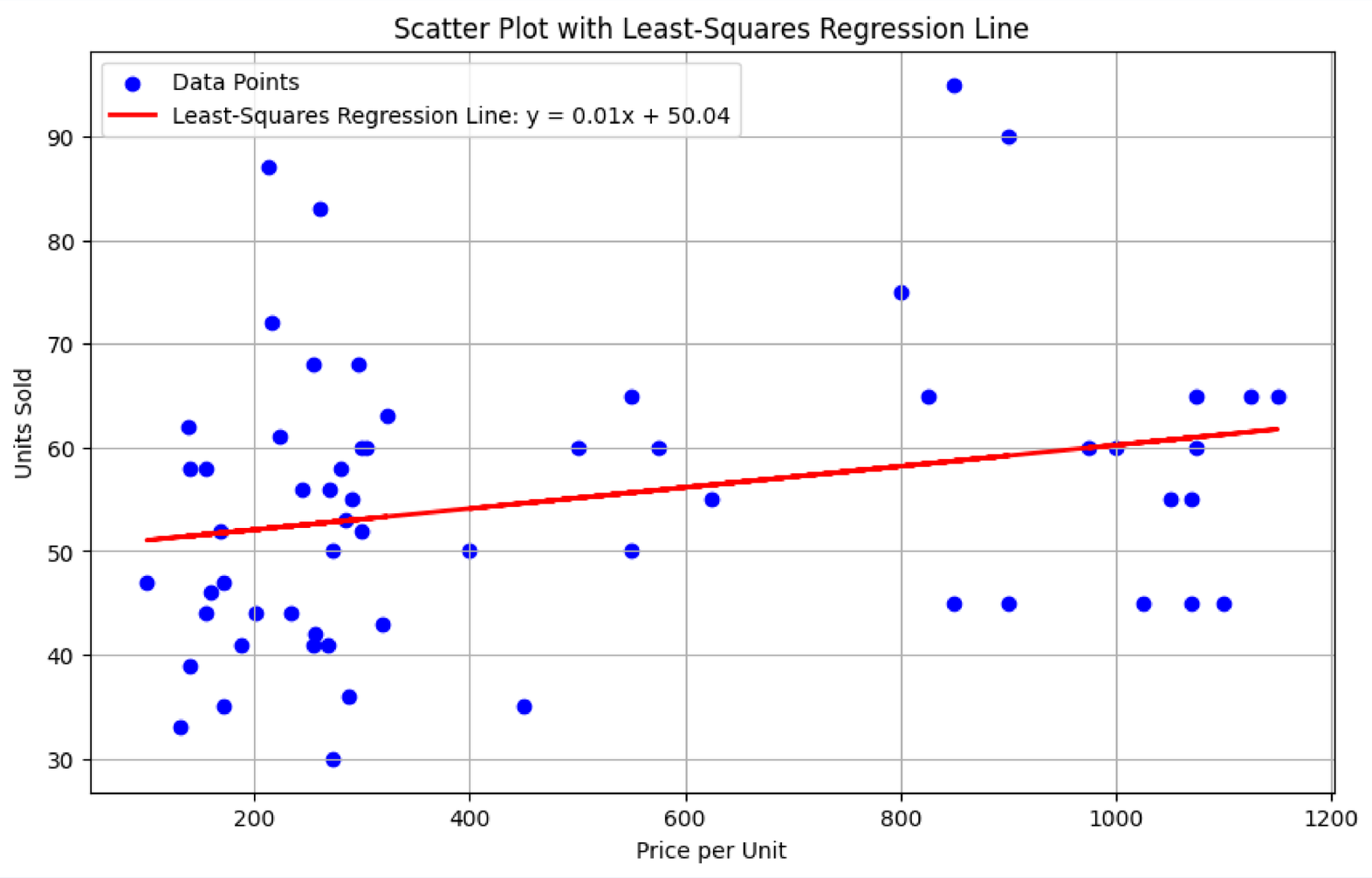
Dep. Variable:	Price per Unit	**R-squared (uncentered):**	0.680
Model:	OLS	**Adj. R-squared (uncentered):**	0.674
Method:	Least Squares	**F-statistic:**	125.3
Date:	Wed, 19 Jun 2024	**Prob (F-statistic):**	3.17e-16
Time:	16:16:08	**Log-Likelihood:**	-293.10
No. Observations:	60	**AIC:**	588.2
Df Residuals:	59	**BIC:**	590.3
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Units Sold	0.0791	0.007	11.195	0.000	0.065	0.093

Omnibus:	4.326	**Durbin-Watson:**	0.160
Prob(Omnibus):	0.115	**Jarque-Bera (JB):**	4.135
Skew:	-0.589	**Prob(JB):**	0.127
Kurtosis:	2.484	**Cond. No.**	1.00

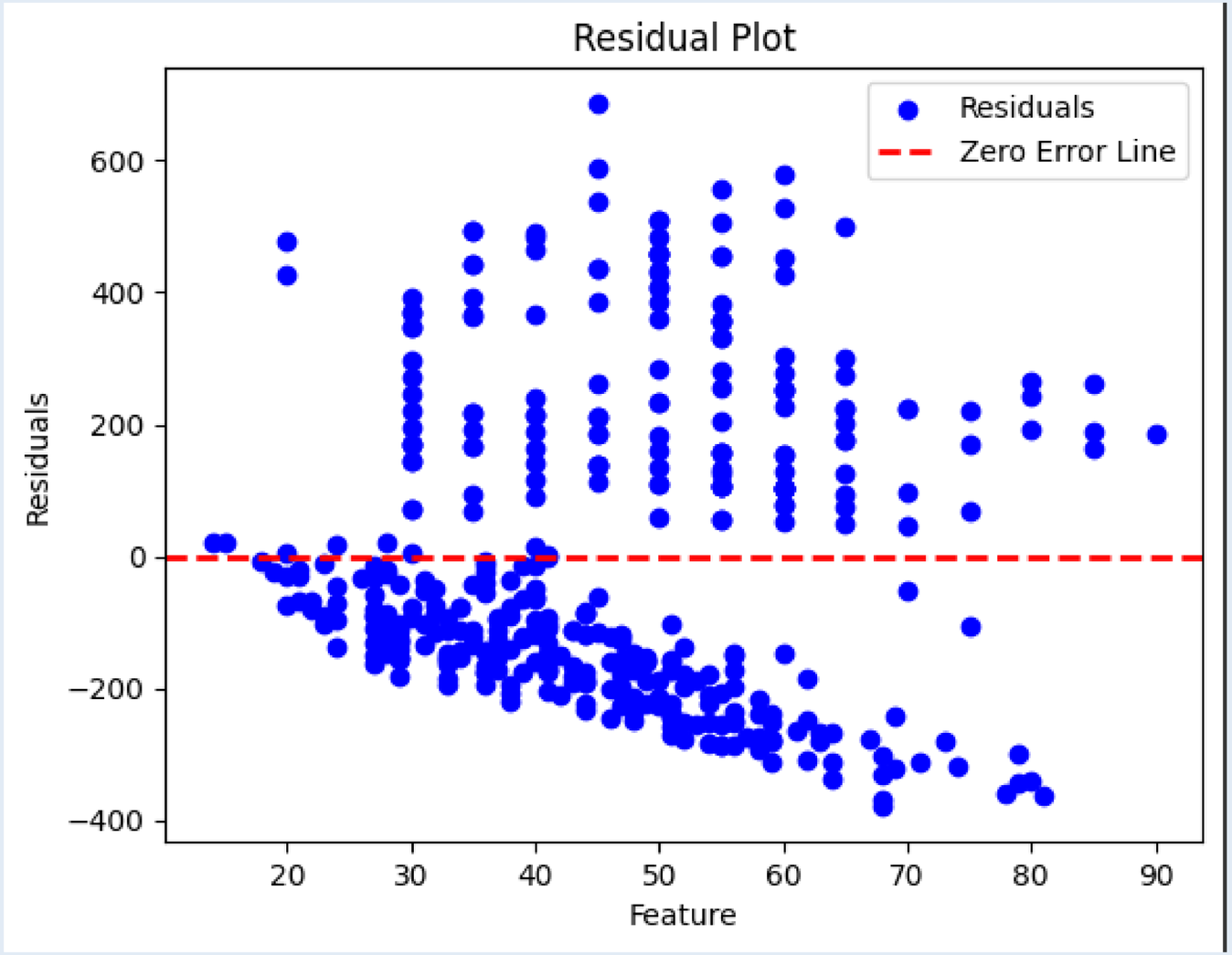
2.LINEAR REGRESSION

Least-Square Regression Line



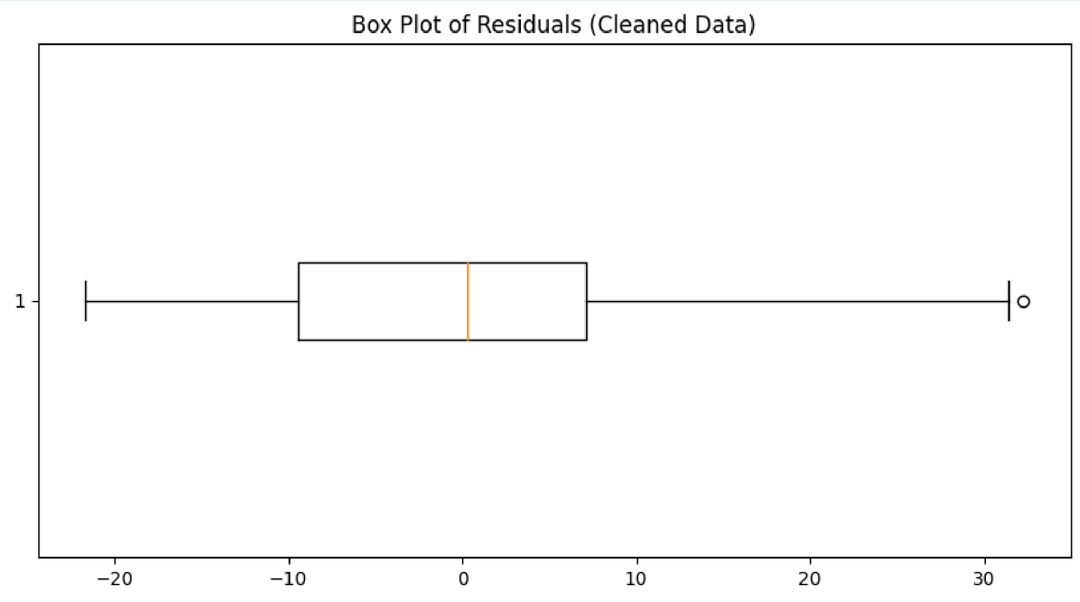
2.LINEAR REGRESSION

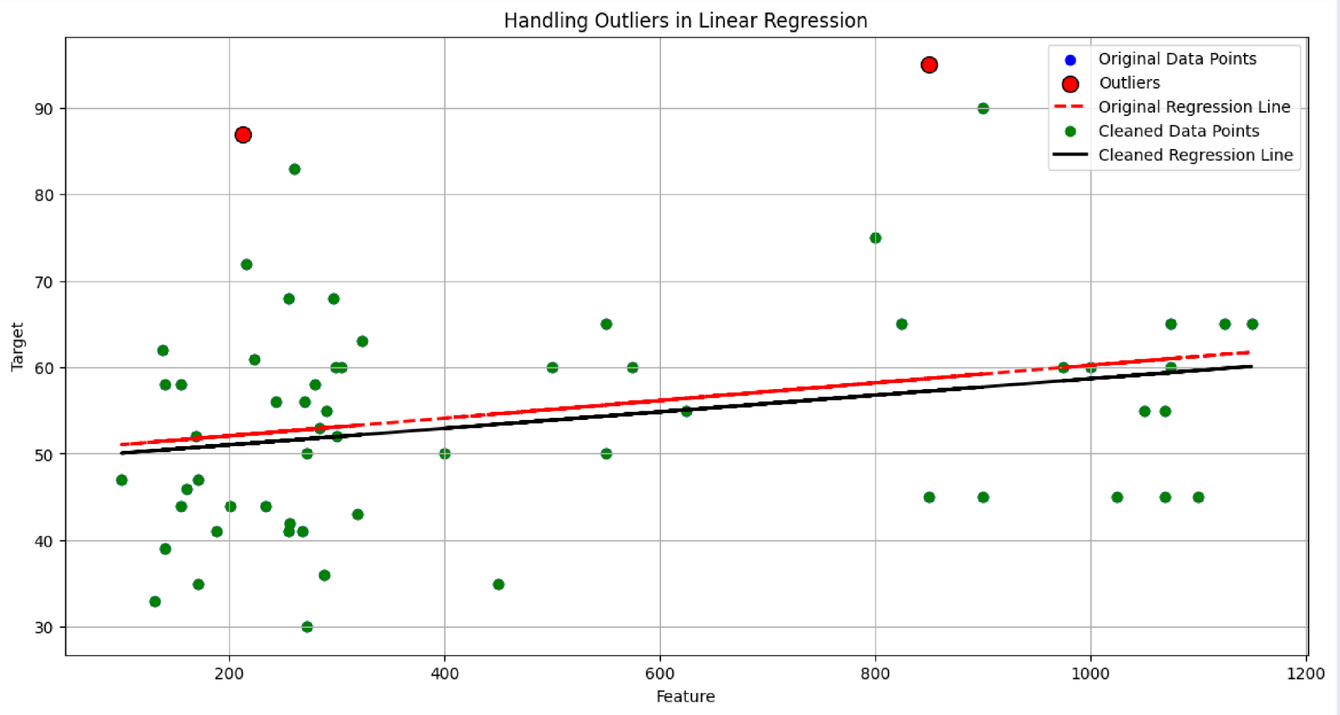
Residual plot



2.LINEAR REGRESSION

Outlier dealing





3. DESCRIPTIVE STATIC

Price per unit

```
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
```

Unit sold

```
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
```

4. Normality

Price per unit

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

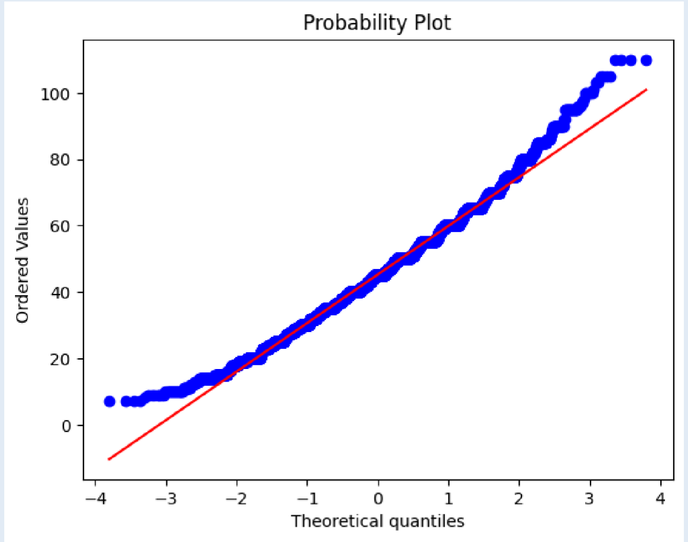
Unit sold

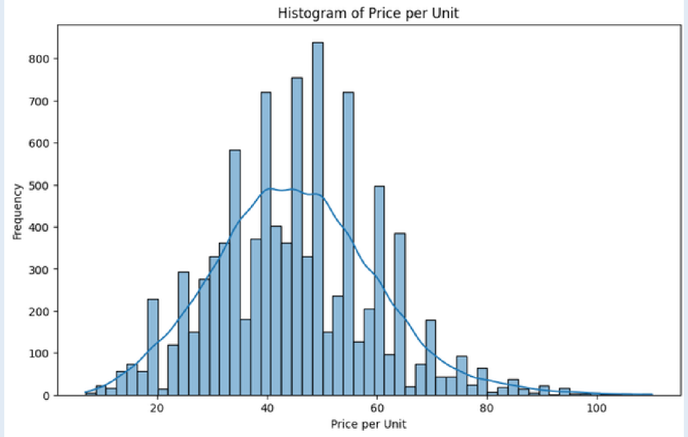
```
# Draw probability plot
stats.probplot(df['Units Sold'], dist="norm", plot=plt)
plt.title('Probability Plot')
plt.show()

# 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()
```

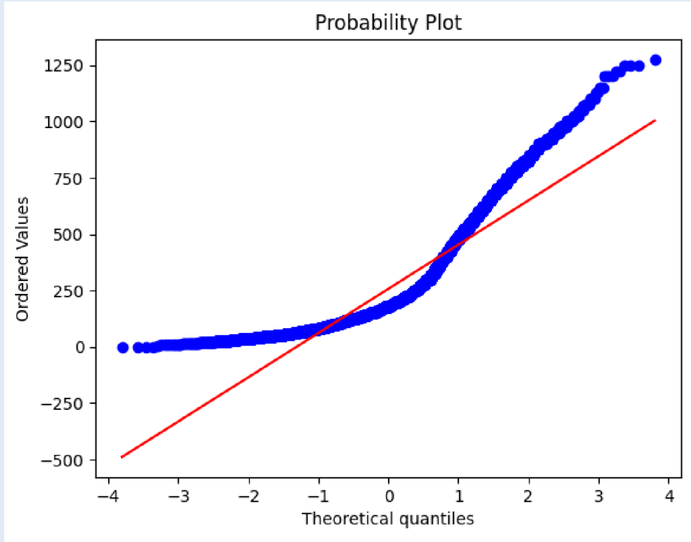
4. Normality

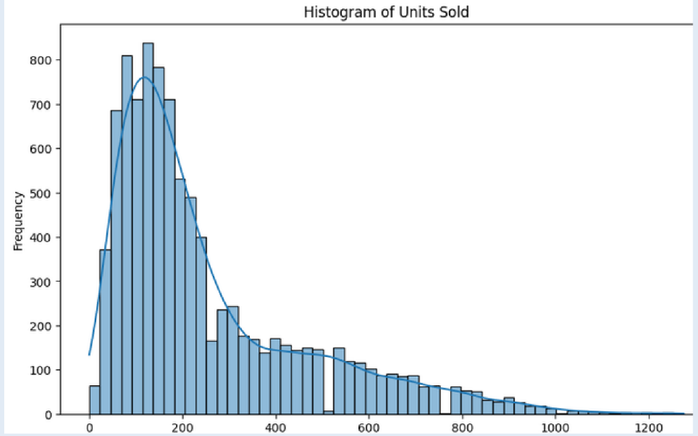
Price per unit





Unit sold

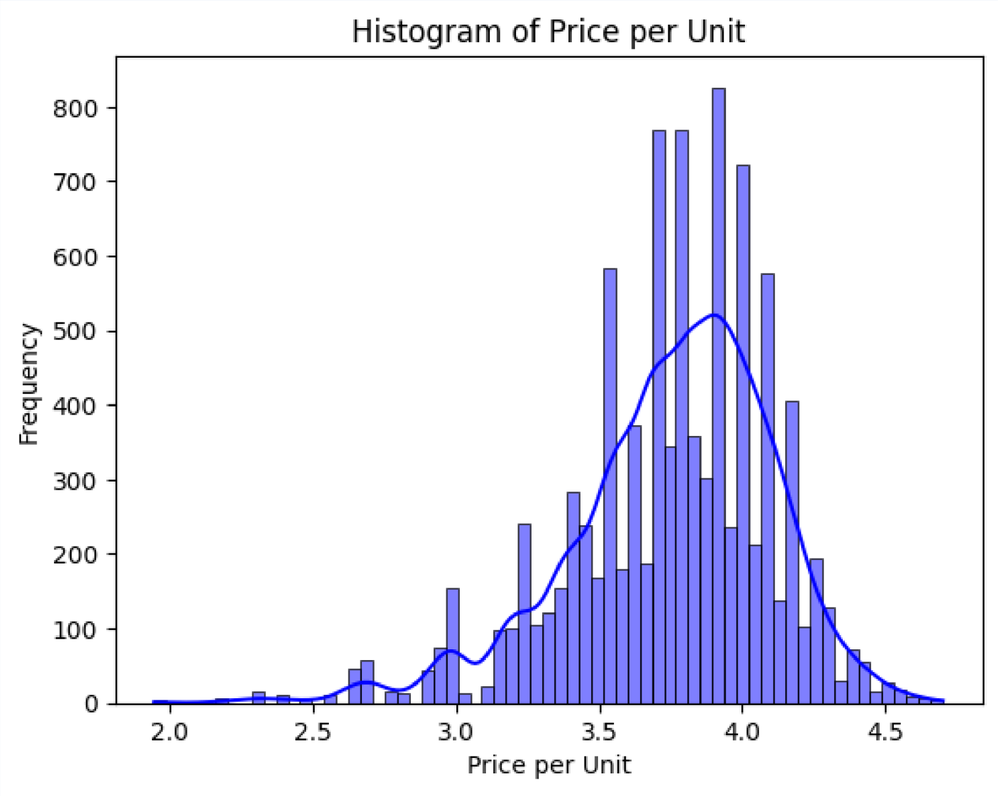




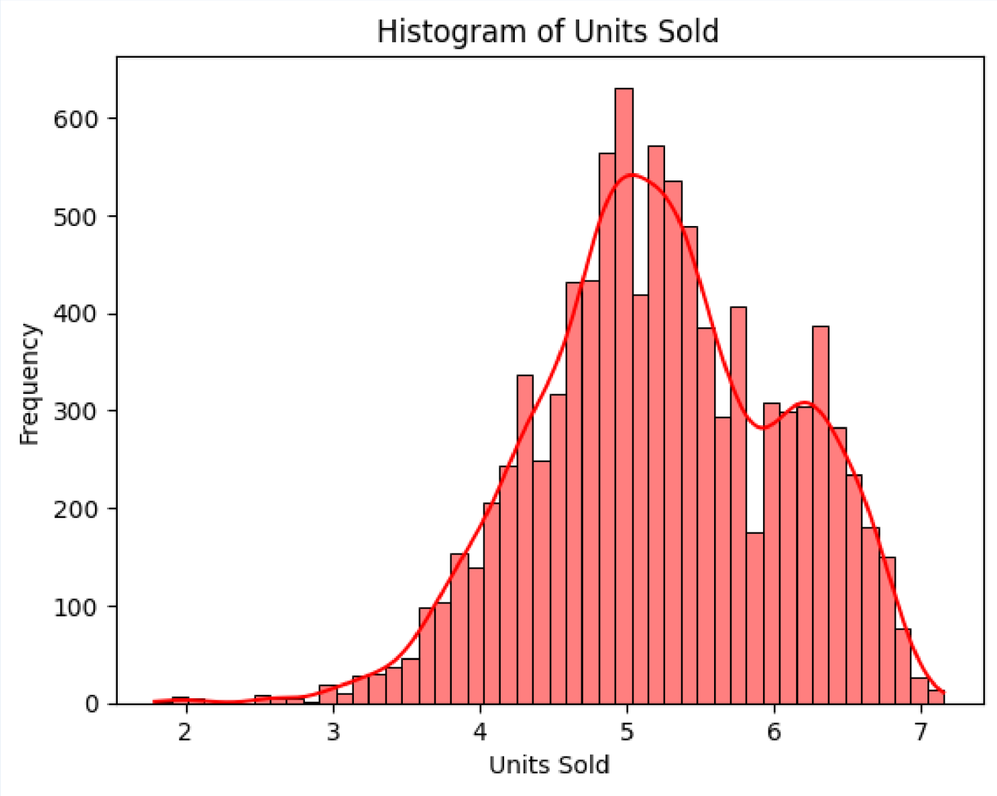
4. Normality

Transform data to normal distribution

Price per unit



Unit sold



5. Confident interval

```python
#Construct confidence Interval
def confidence_interval(data, confidence=0.95):
  # Convert data to a numpy arr
  data = np.array(data)

  #Calculate Sample mean and SD error
  mean = np.mean(data)
  std_error = stats.sem(data)

  #Get the critical value based on the sample size:
  n = len(data)
  if n>30:
    critical_value = stats.norm.ppf((1 + confidence) / 2)
  else:
    critical_value = stats.t.ppf((1 + confidence) / 2, n - 1)

  #Calculate the margin of error
  margin_of_error = critical_value * std_error

  #Calculate the confidence interval
  lower_bound = mean - margin_of_error
  upper_bound = mean + margin_of_error

  return lower_bound, upper_bound
```

```python
data_Price = df_use['Price per Unit']

#calculate the confidence interval
confidence_lvl_1 = 0.90
confidence_lvl_2 = 0.95
confidence_lvl_3 = 0.99

ci_1 = confidence_interval(data_Price, confidence_lvl_1)
ci_2 = confidence_interval(data_Price, confidence_lvl_2)
ci_3 = confidence_interval(data_Price, confidence_lvl_3)
print(f"The {int (confidence_lvl_1 * 100)}% Confidence Interval is: {ci_1}")
print(f"The {int (confidence_lvl_2 * 100)}% Confidence Interval is: {ci_2}")
print(f"The {int (confidence_lvl_3 * 100)}% Confidence Interval is: {ci_3}")
```

```
The 90% Confidence Interval is: (43.666642997473666, 44.80664892799218)
The 95% Confidence Interval is: (43.557445537428364, 44.91584638803748)
The 99% Confidence Interval is: (43.34402529608689, 45.129266629378954)
```

6.HYPOTHESIS TESING

```python
# 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
```

```python
# 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.
```

Thanks for listening!