**Practical No. 01**

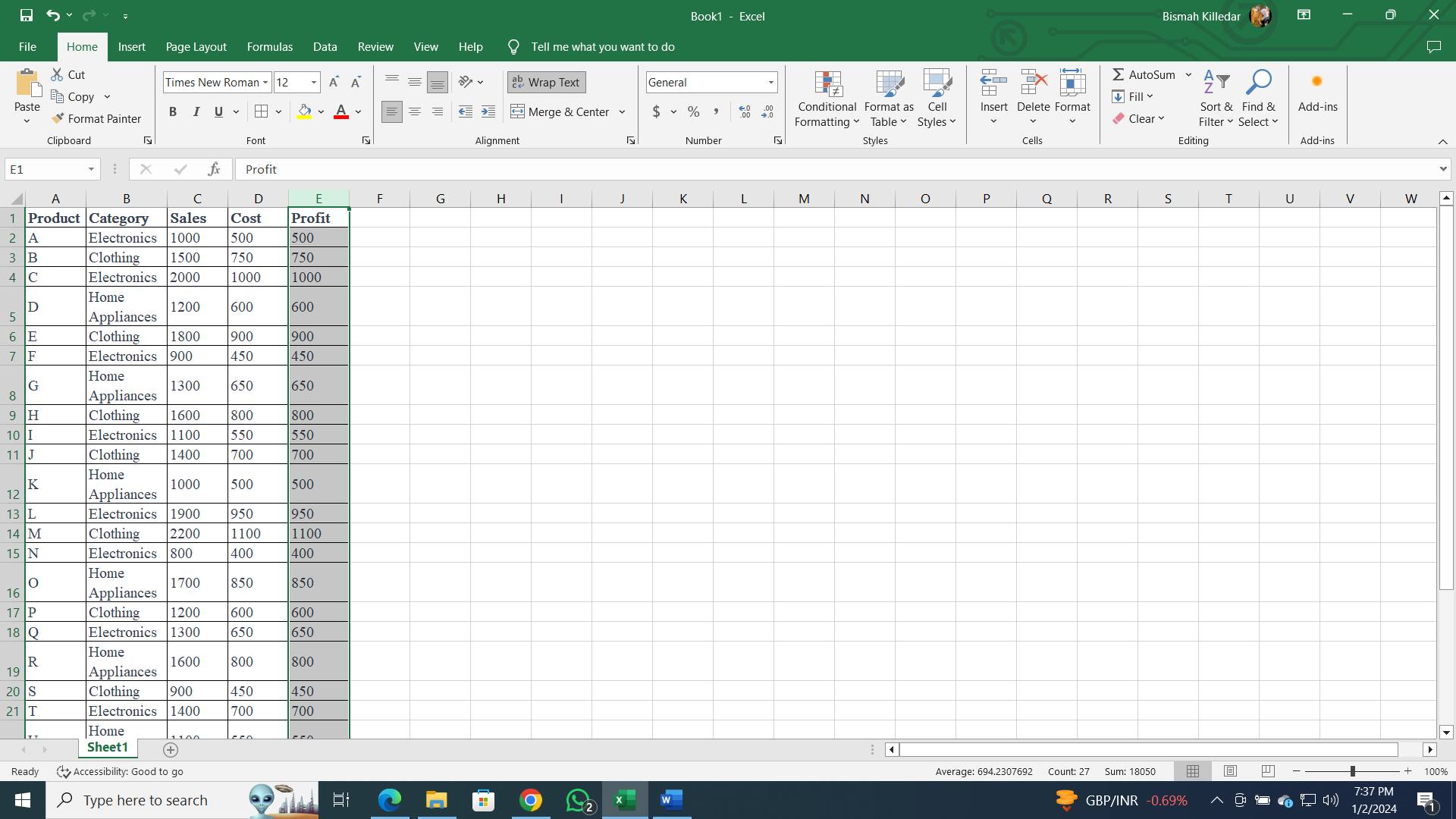
**Aim:** Introduction to Excel

* Perform conditional formatting on a dataset using various criteria.
* Create a pivot table to analyze and summarize data.
* Use VLOOKUP function to retrieve information from a different worksheet or table.
* Perform what-if analysis using Goal Seek to determine input values for desired output.
* **Perform conditional formatting on a dataset using various criteria.**

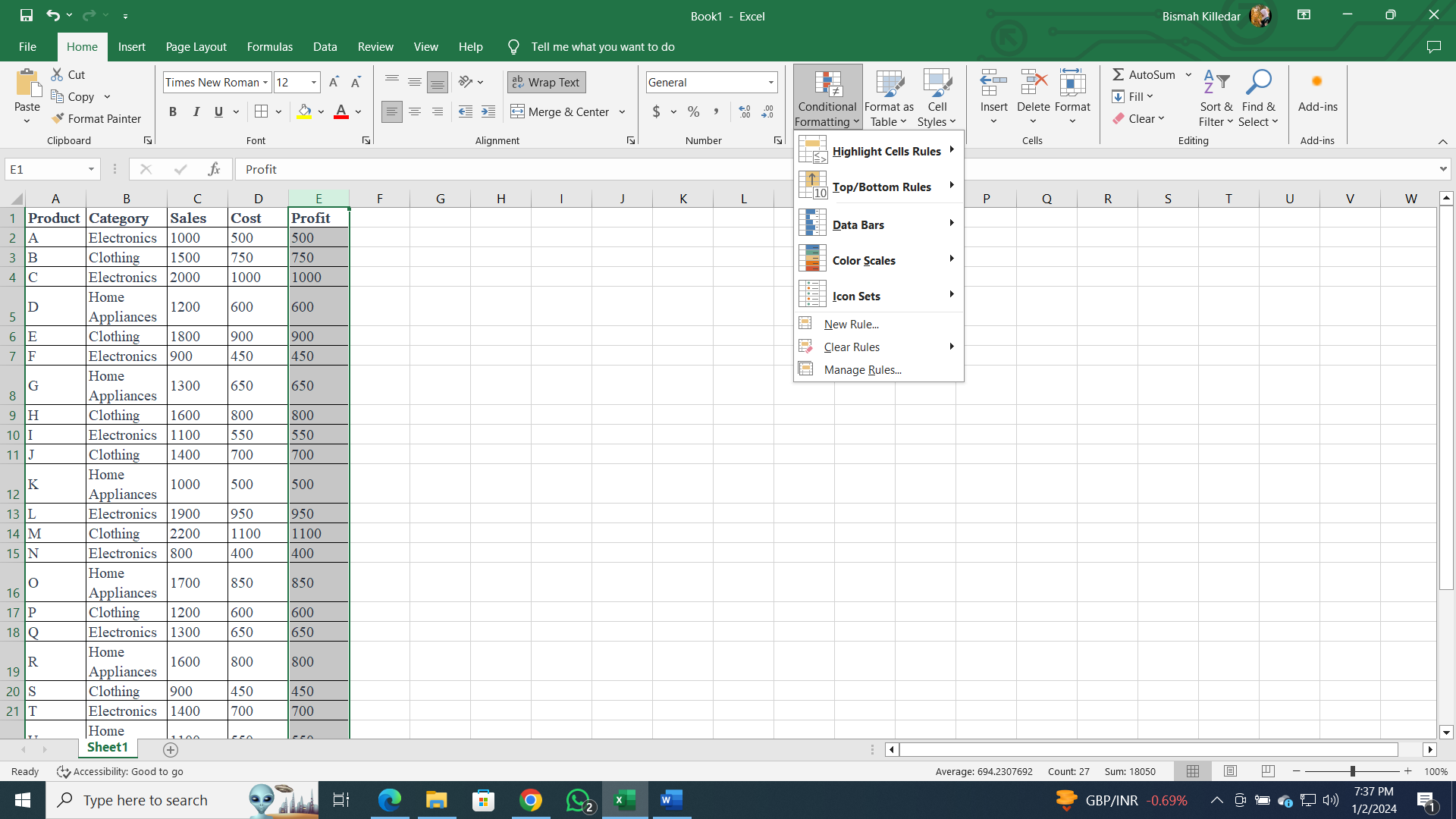
We perform conditional formatting on the "Profit" column to highlight cells with a profit greater than 800 using following steps:

**Steps:**

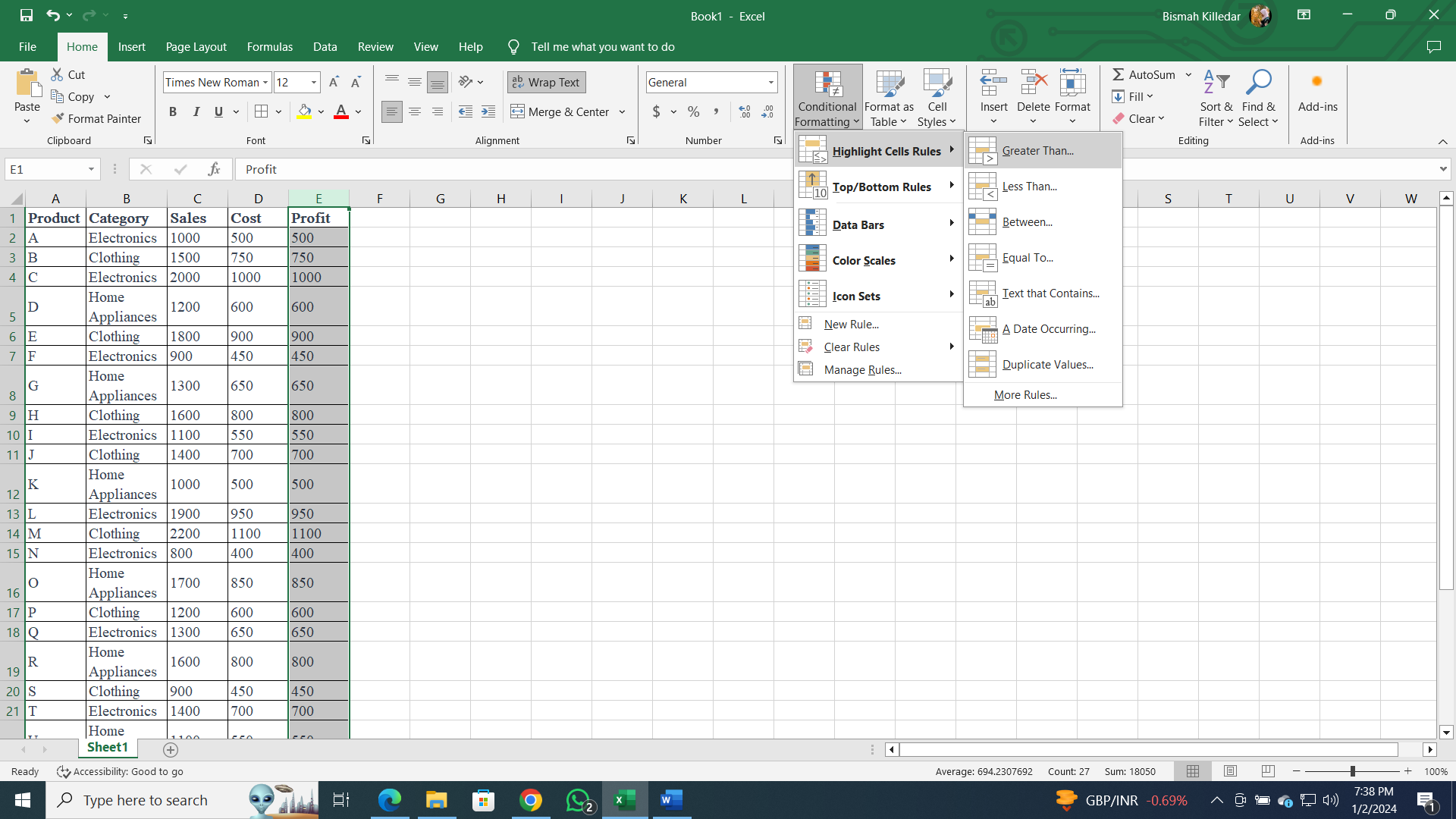
1. Select the "Profit" column (Column E).



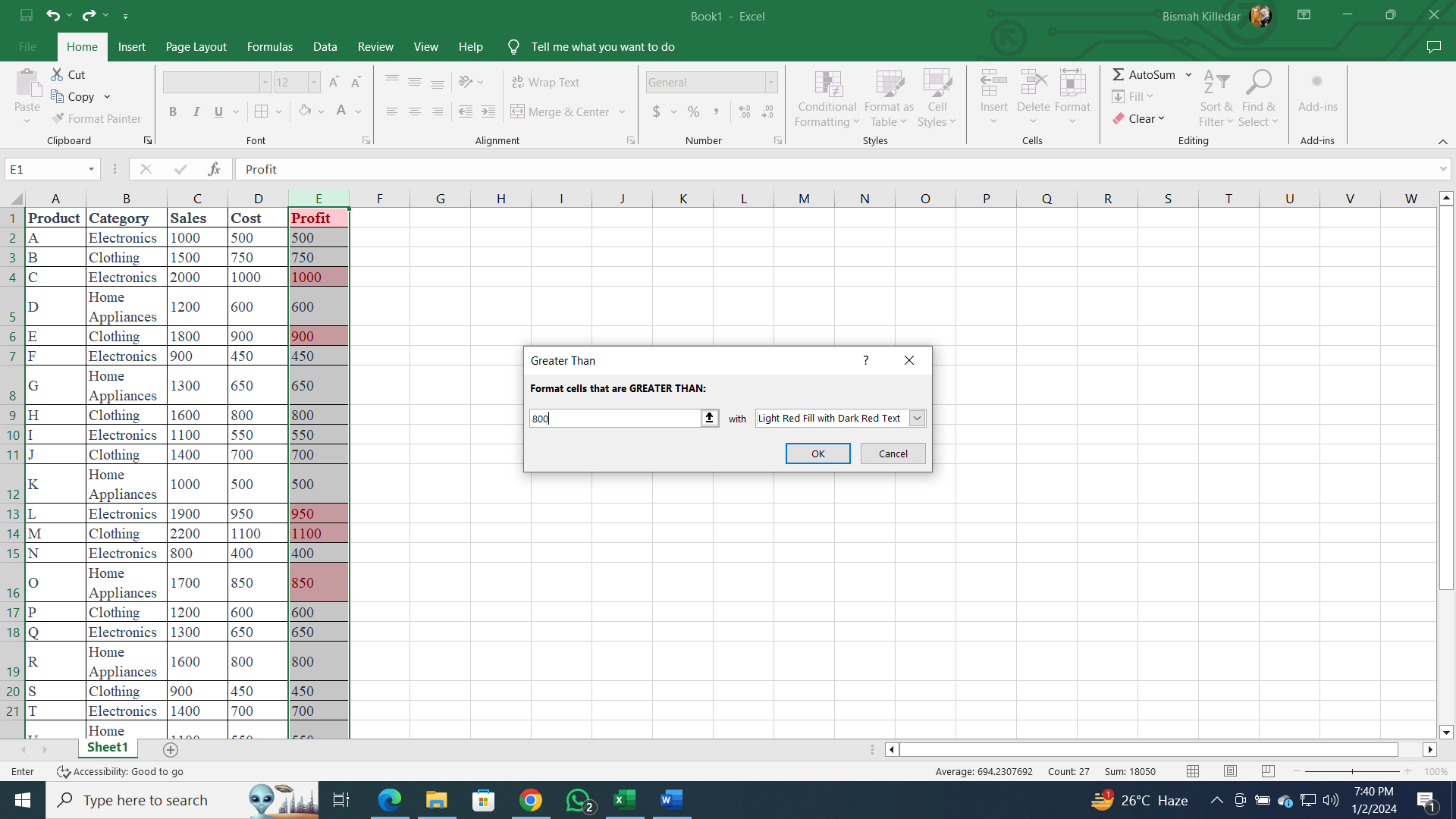
1. Go to the "Home" tab on the ribbon.
2. Click on "Conditional Formatting" in the toolbar.



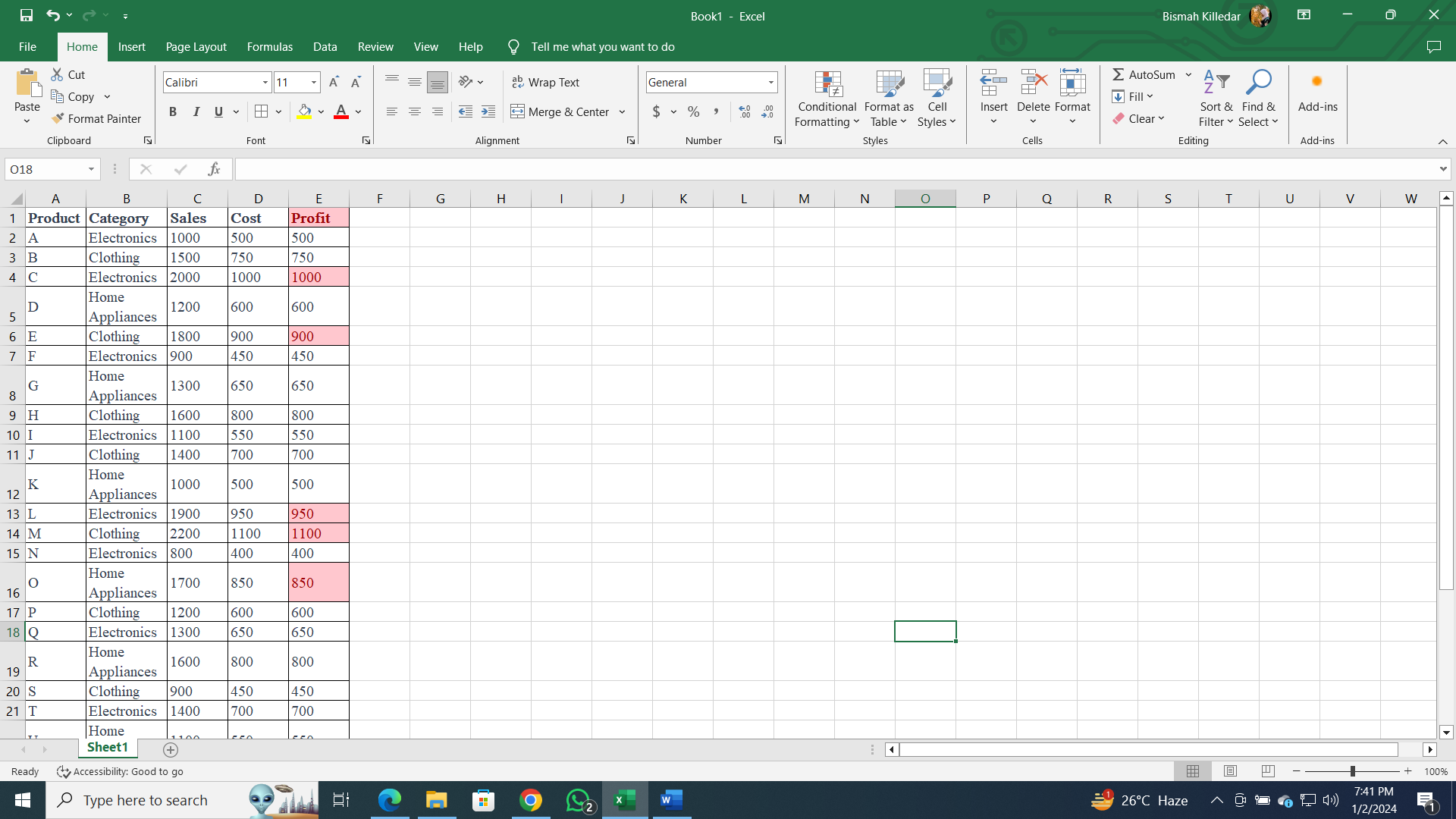
1. Choose "Highlight Cells Rules" and then "Greater Than."



1. Enter the threshold value as 800.



1. Customize the formatting options (e.g., choose a fill color).
2. Click "OK" to apply the rule.

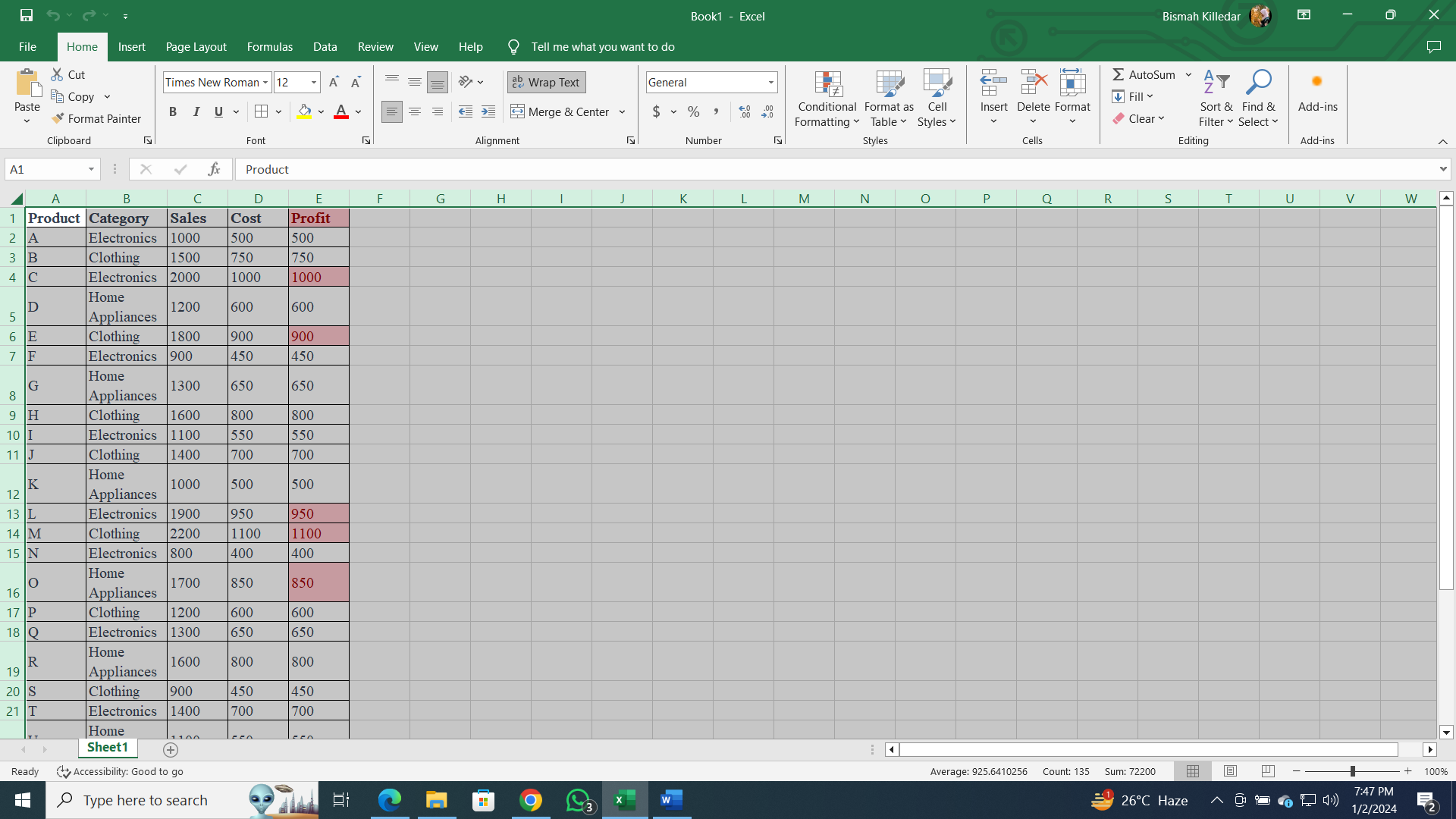


* **Create a pivot table to analyze and summarize data.**

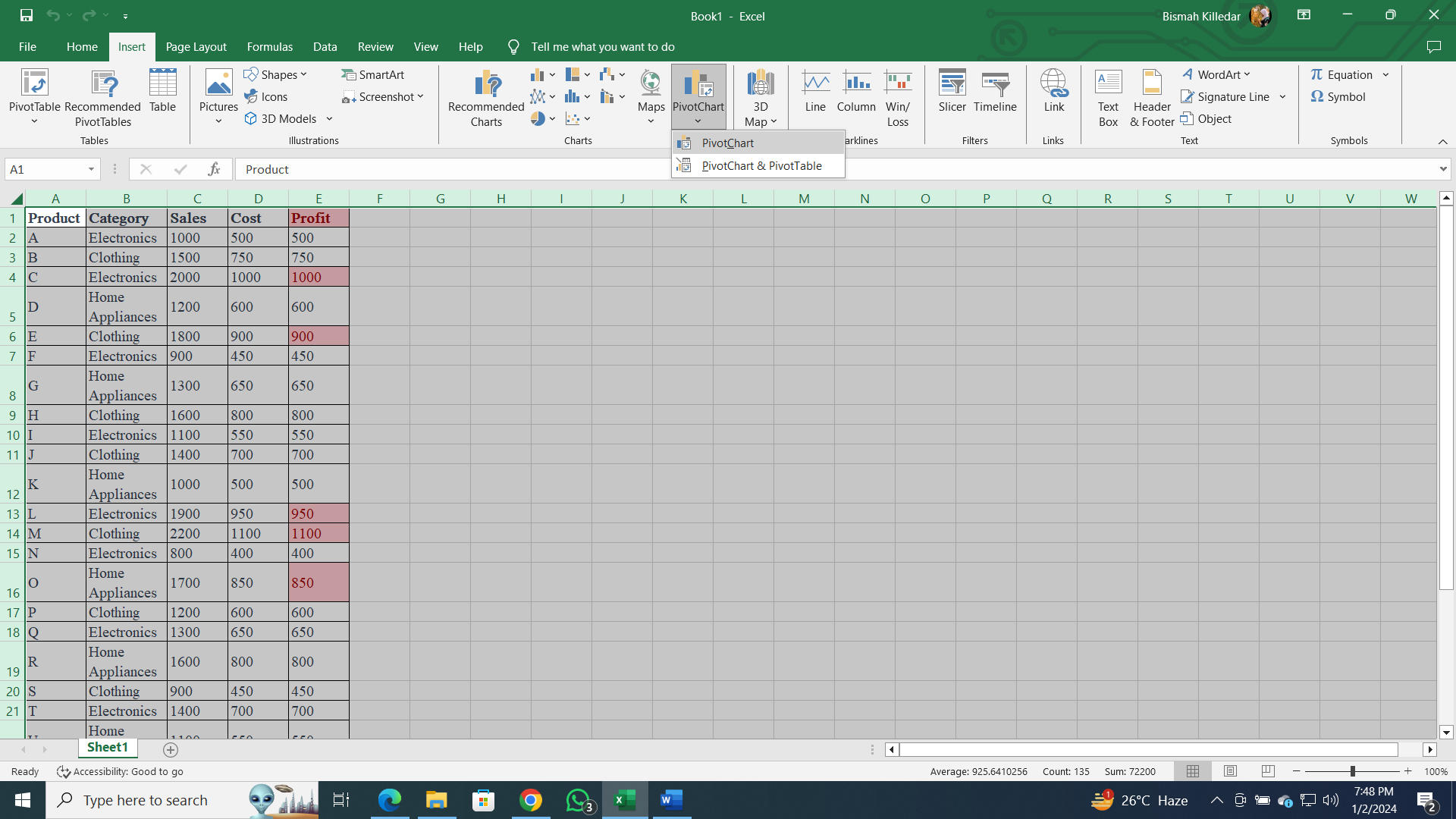
Following are the steps to create a pivot table to analyze total sales by category.

**Steps:**

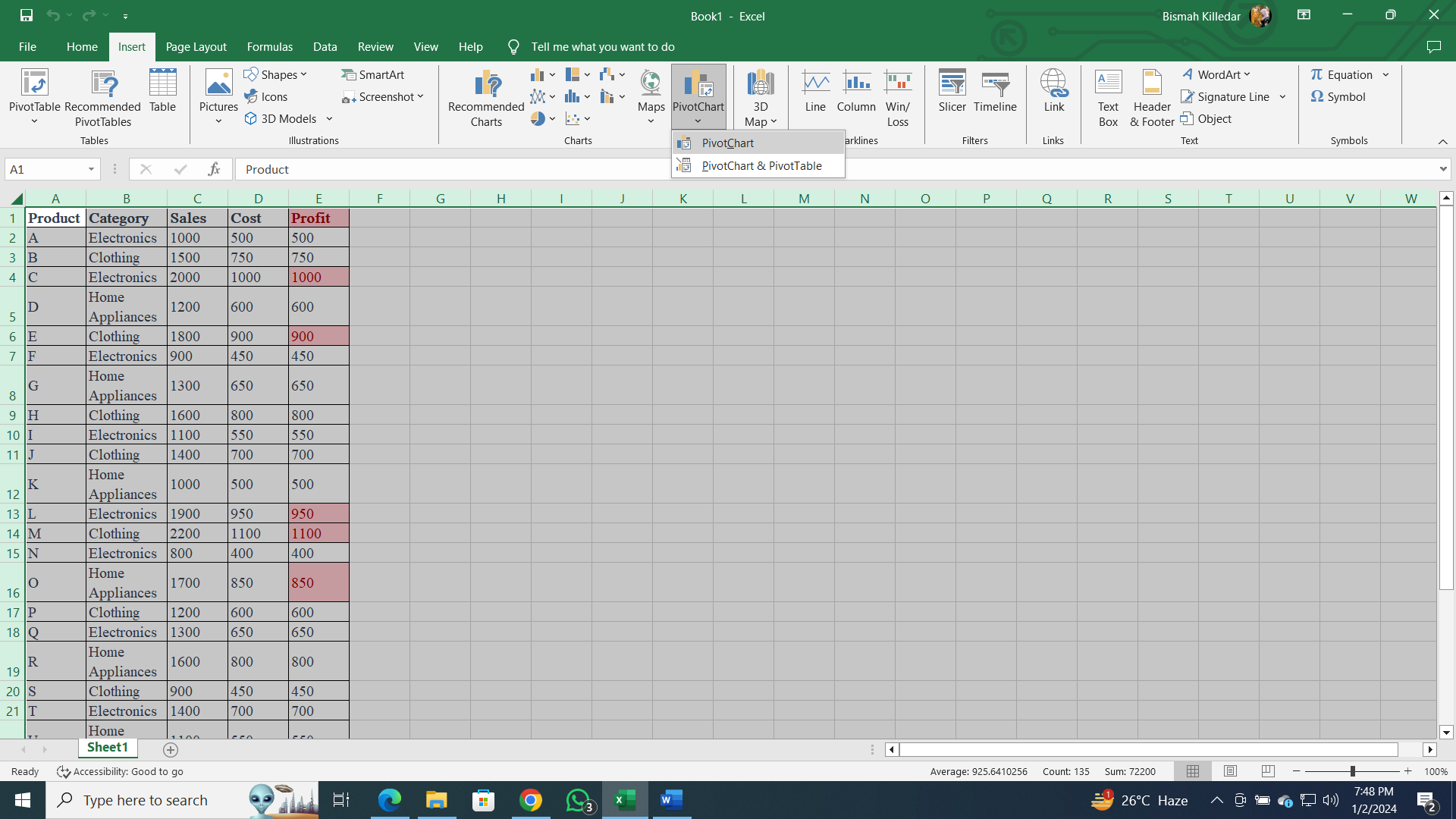
1. Select the entire dataset including headers.



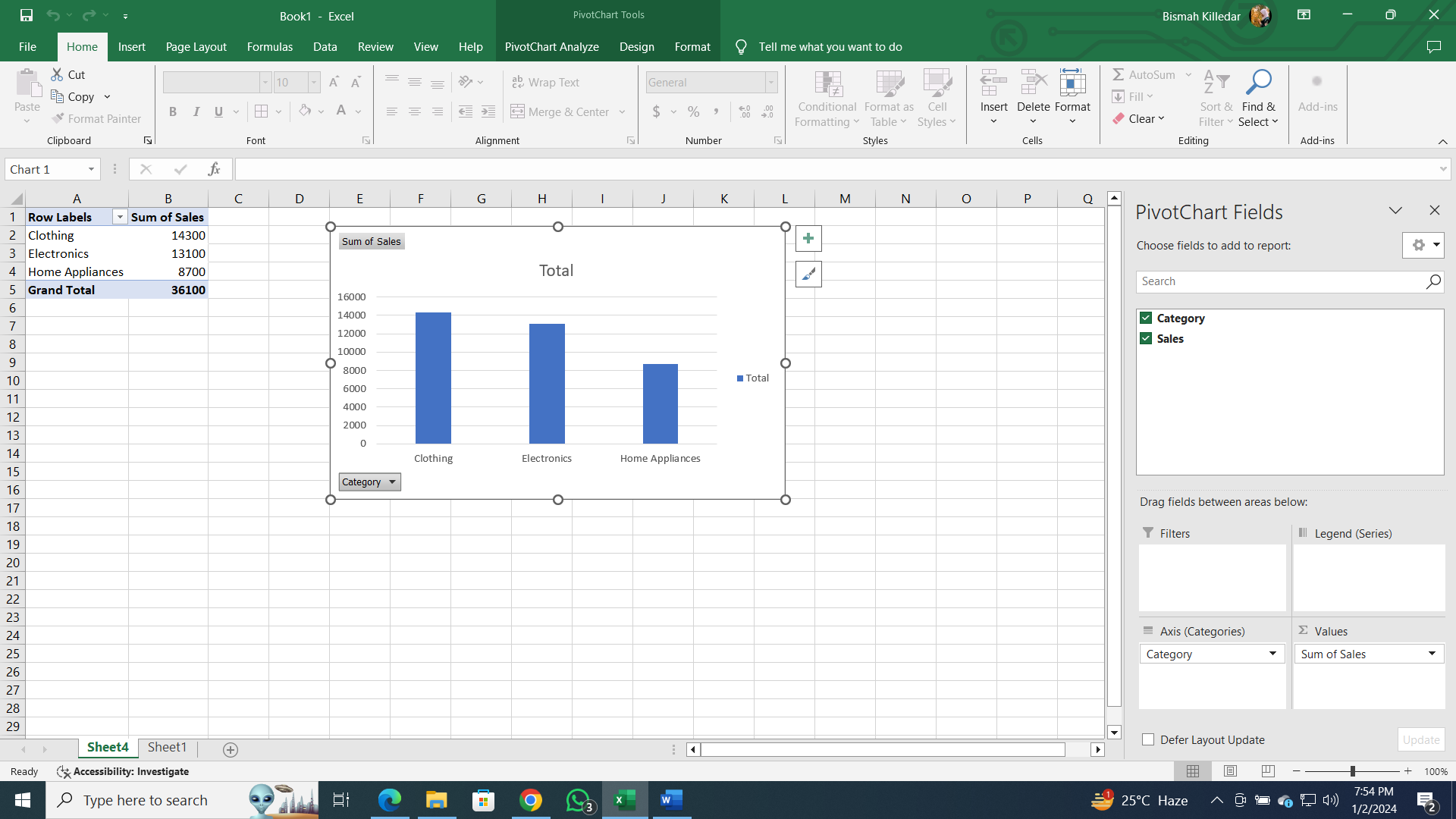
1. Go to the "Insert" tab on the ribbon.
2. Click on "PivotTable."



1. Choose where you want to place the PivotTable (e.g., new worksheet).



1. Drag "Category" to the Rows area.
2. Drag "Sales" to the Values area, choosing the sum function.



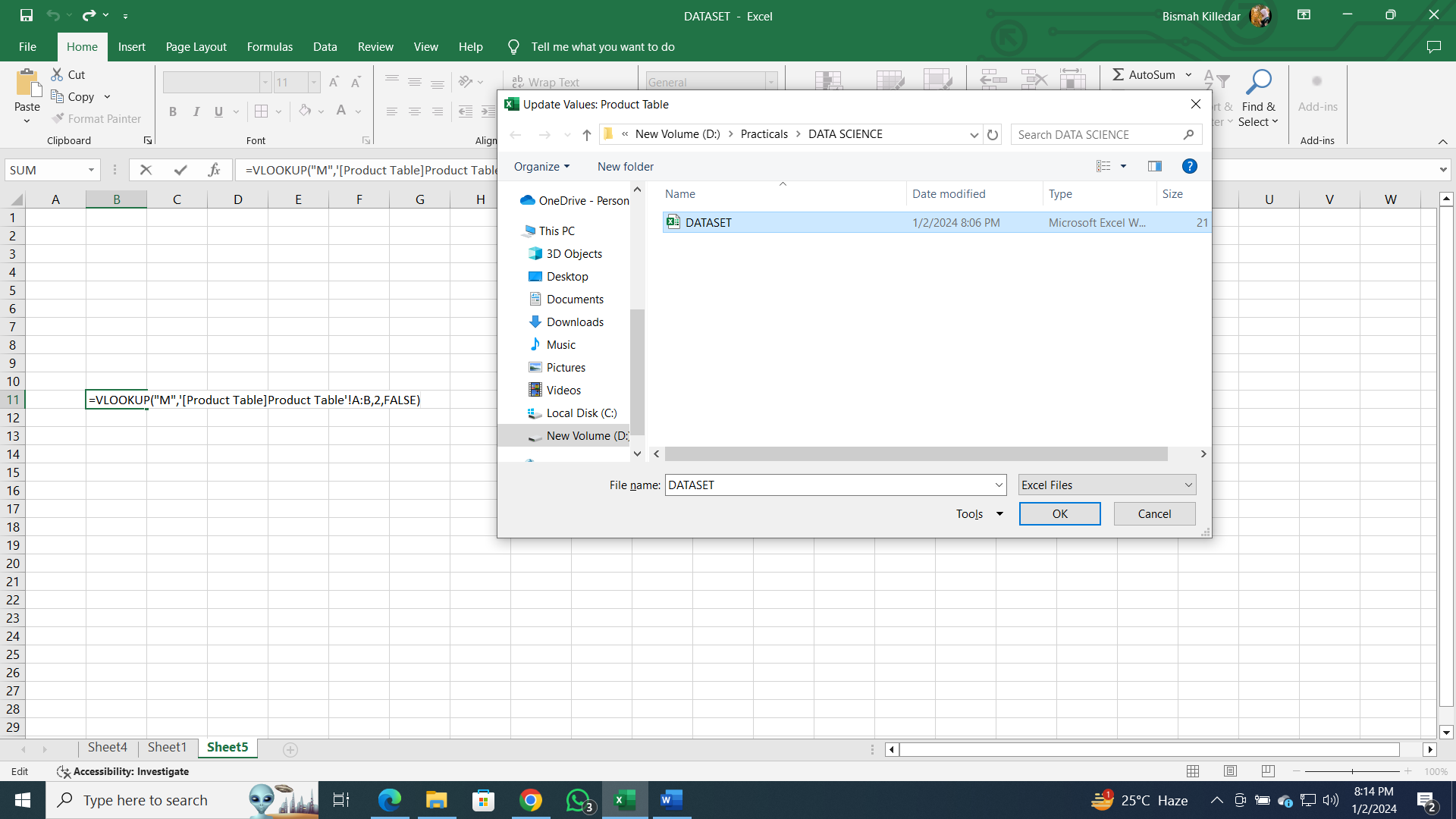
* **Use VLOOKUP function to retrieve information from a different worksheet or table.**

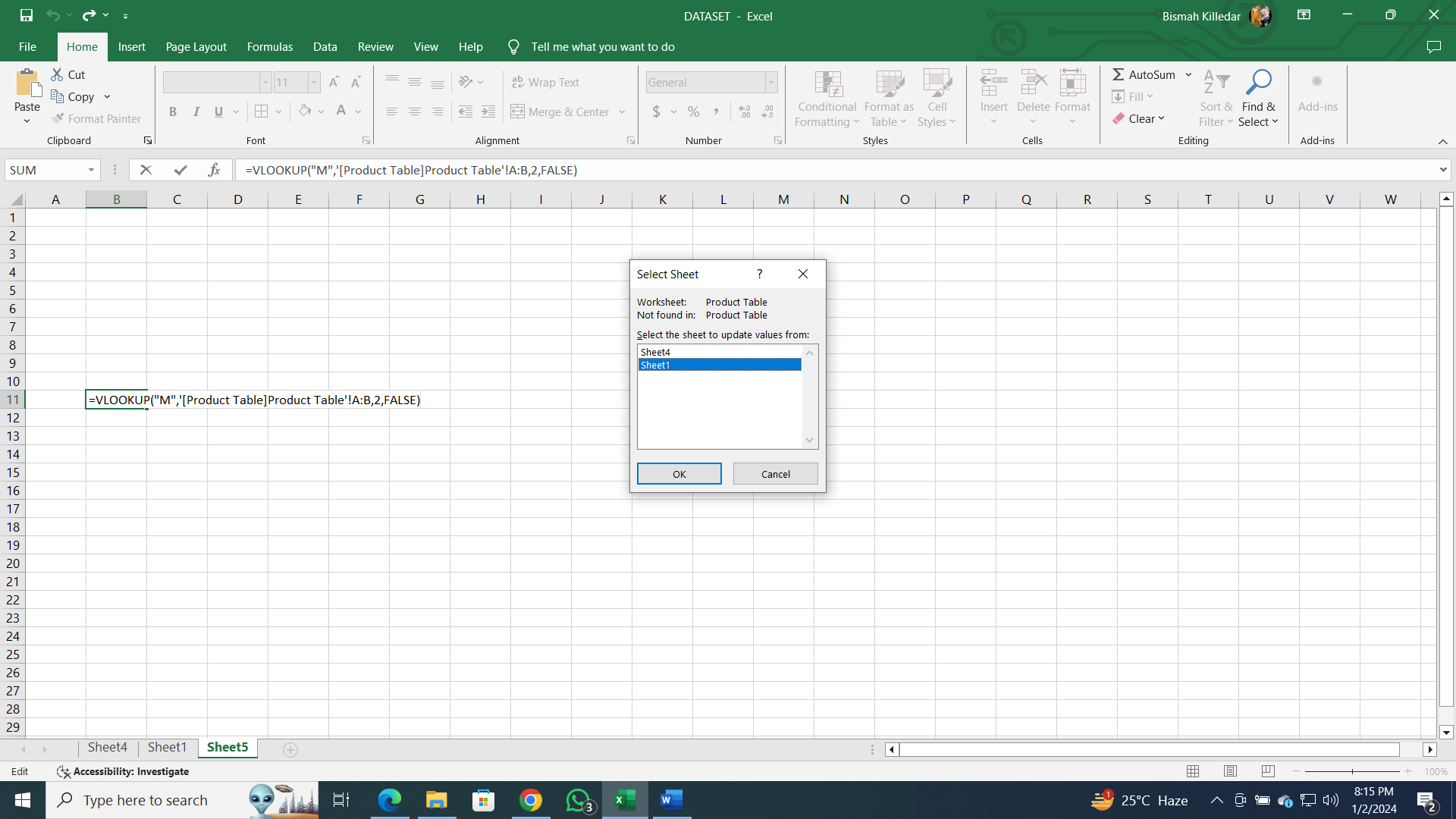
Use the VLOOKUP function to retrieve the category of "Product M" from a separate worksheet named "Product Table" using following steps:

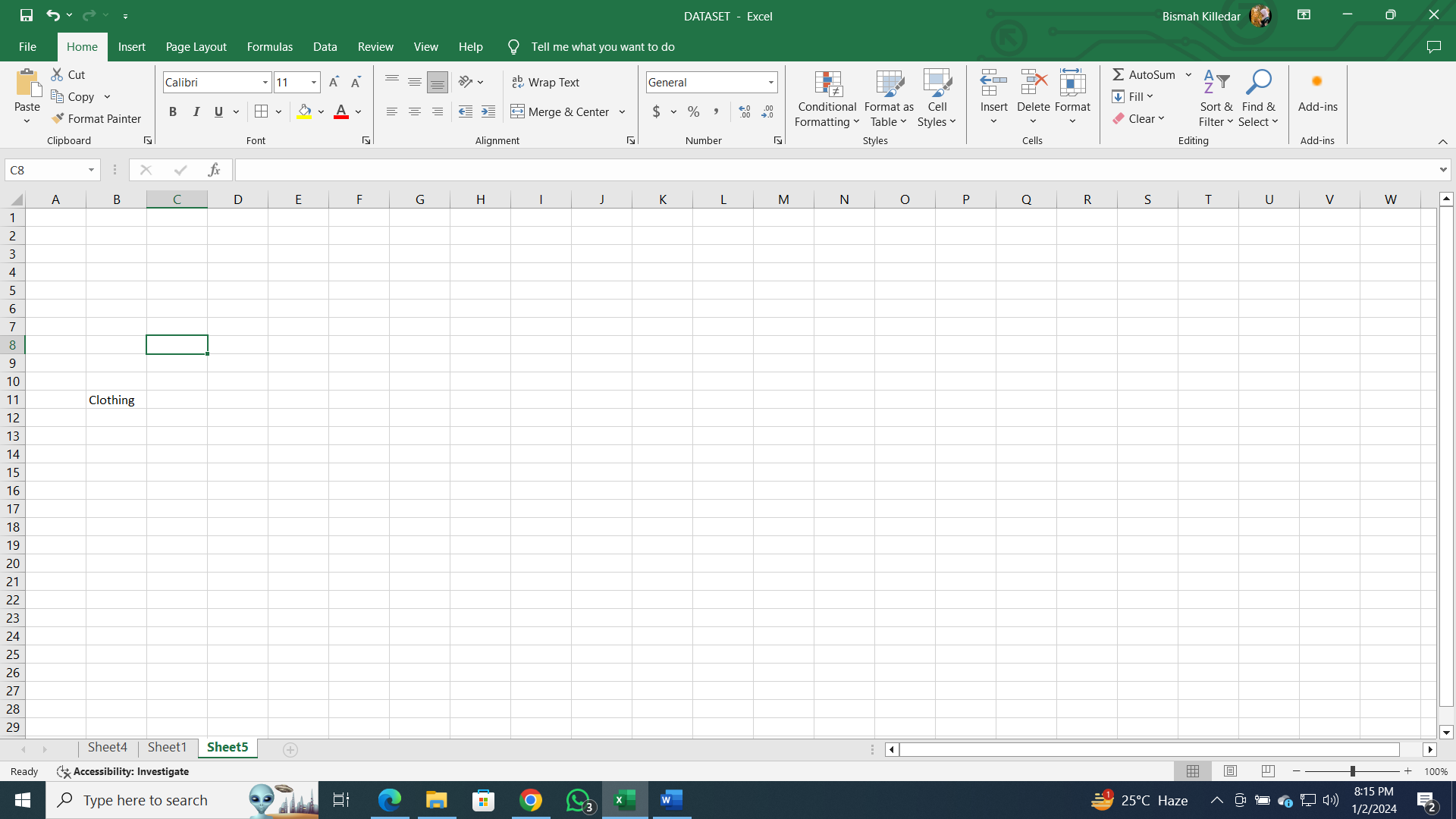
**Steps:**

1. Assuming your "Product Table" is in a different worksheet.
2. In a cell in your main dataset, enter the formula:

**=VLOOKUP("M", 'Product Table'!A:B, 2, FALSE)**





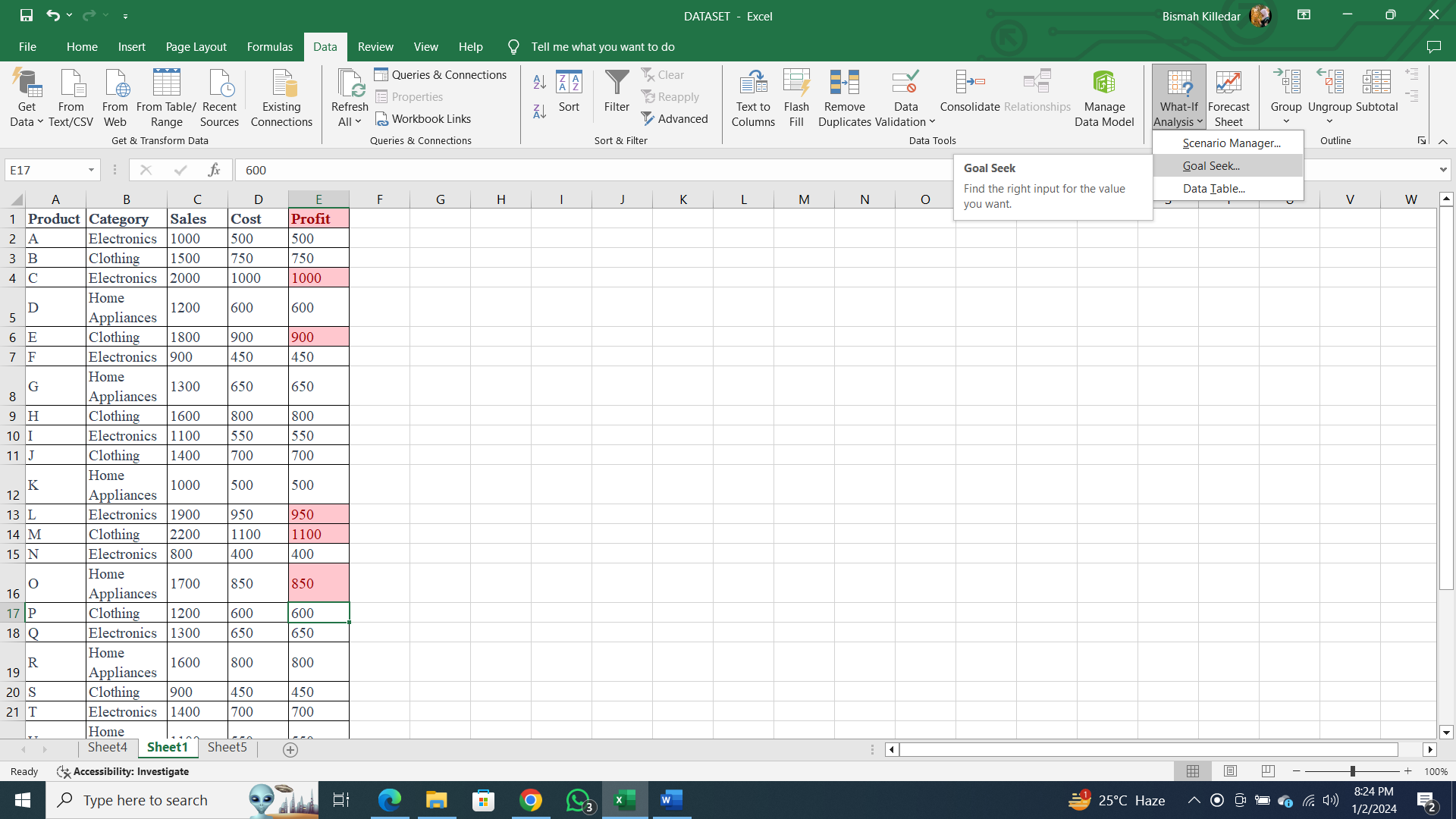


* **Perform what-if analysis using Goal Seek to determine input values for desired output.**

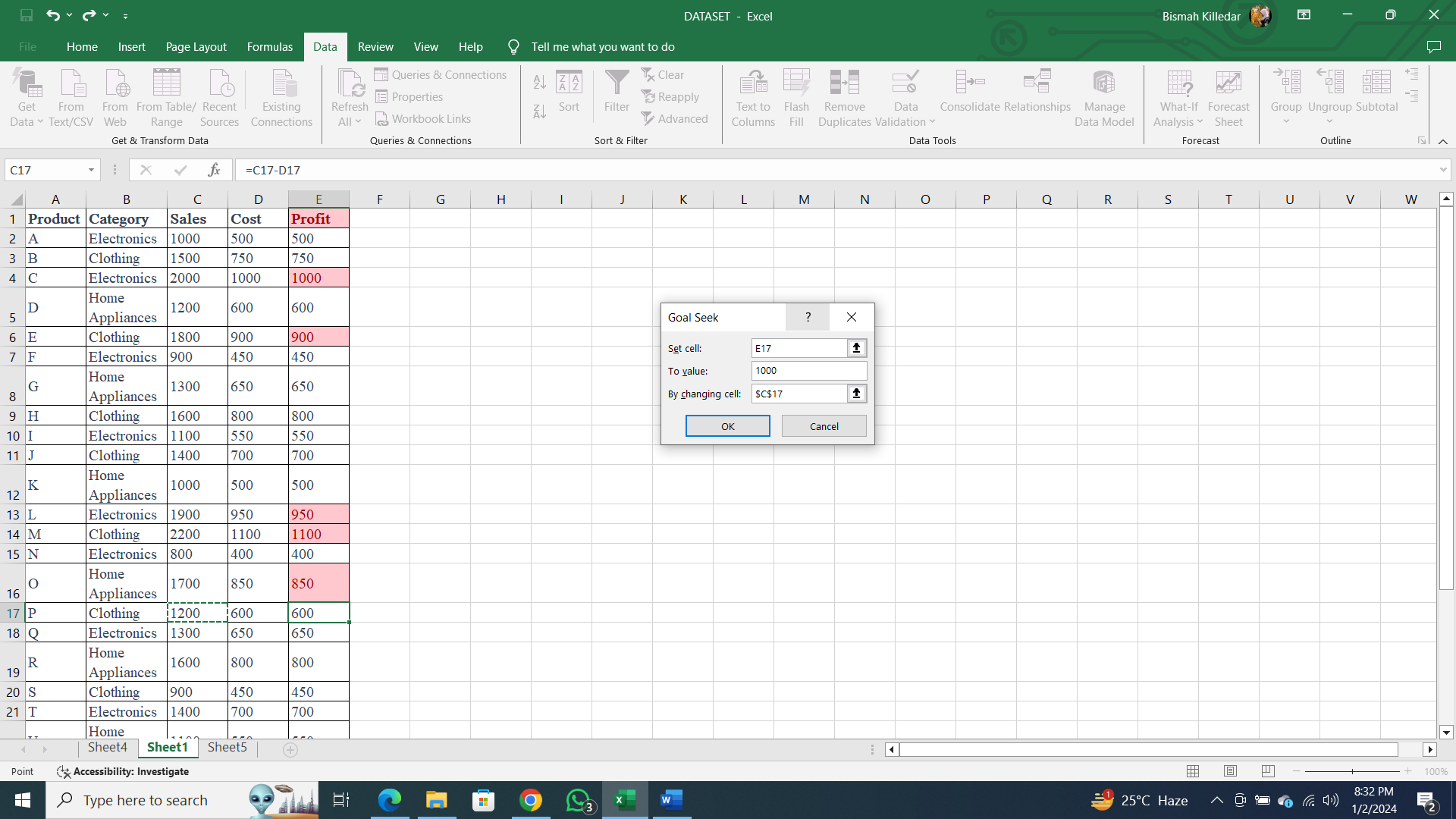
Use Goal Seek to find the required sales for "Product P" to achieve a profit of 1000 using the following steps.

**Steps:**

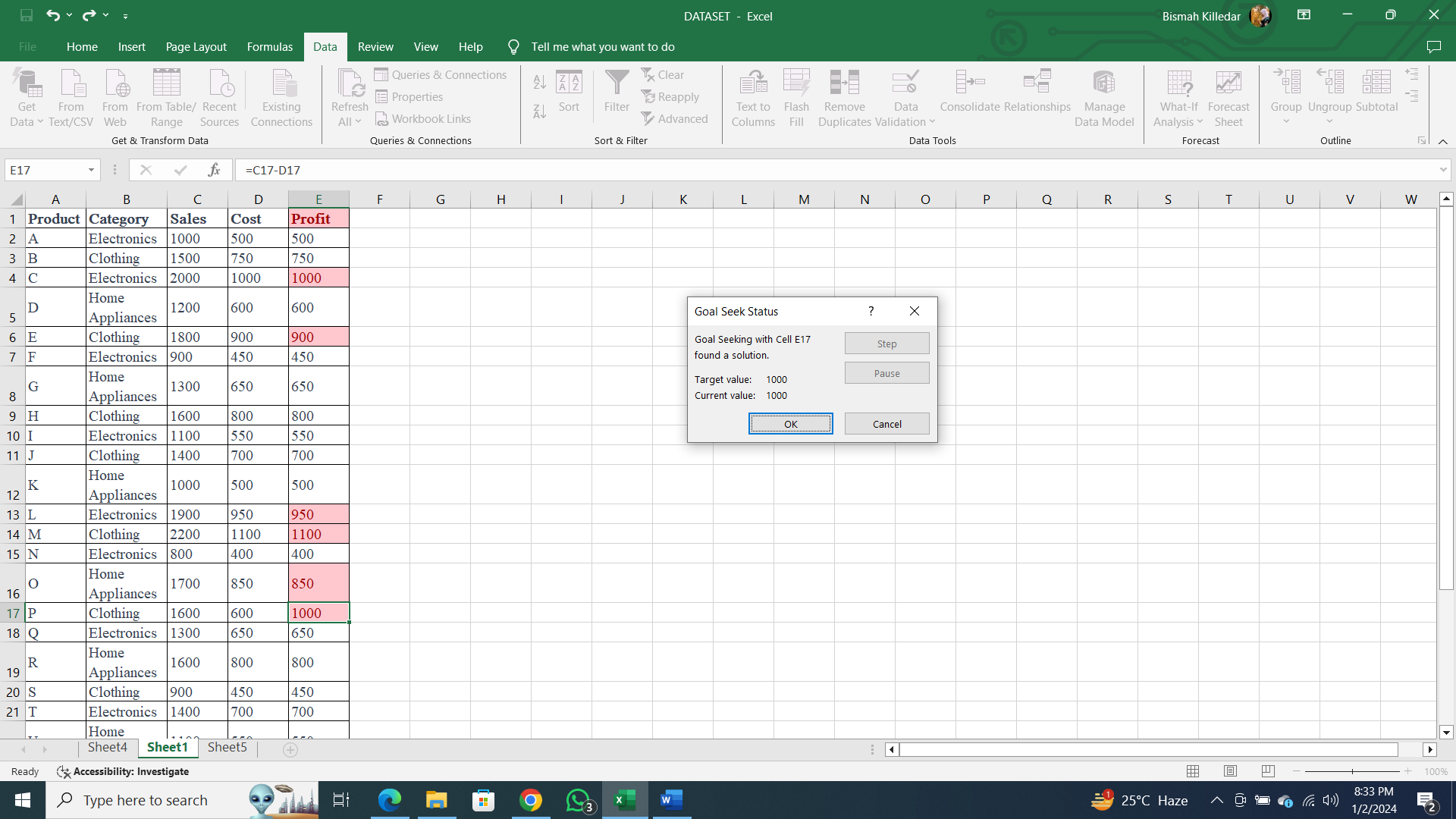
1. Identify the cell containing the formula for "Profit" for "Product P" (let's assume it's in cell E17).
2. Go to the "Data" tab on the ribbon.
3. Click on "What-If Analysis" and select "Goal Seek."



1. Set "Set cell" to the profit cell (E17), "To value" to 1000, and "By changing cell" to the sales cell (C17).



1. Click "OK" to let Excel determine the required sales.



**Practical No. 02**

**Aim:** Data Frames and Basic Data Pre-processing

* Read data from CSV and JSON files into a data frame.
* Perform basic data pre-processing tasks such as handling missing values and outliers.
* Manipulate and transform data using functions like filtering, sorting, and grouping.

**Data pre-processing:**

Data pre-processing is a crucial step in the data analysis pipeline, encompassing tasks such as reading data from various file formats, handling missing values, and managing outliers. This practical guide explores how to execute these tasks using the pandas library in Python.

**Steps:**

**Step 1: Reading from CSV and JSON Files**

1. Utilize pandas to read data from a CSV file ('DATA SET.csv') into a data frame.
2. Use pandas to read data from a JSON file ('ds.json') into a data frame.
3. Display the first few rows of each data frame to inspect the data.

**Step 2: Handling Missing Values**

1. Drop rows with missing values from the CSV data frame.
2. Fill missing values with a specific value (e.g., 0) in the JSON data frame.

**Step 3: Handling Outliers**

1. Identify outliers in the 'Sales' column of the CSV data frame.
2. Replace outliers with the median value.

**Step 4: Manipulating and Transforming Data**

1. Filter the CSV data frame to include only rows where 'Sales' is greater than 10.
2. Sort the CSV data frame based on the 'Sales' column in descending order.
3. Group the CSV data frame by the 'Category' column and calculate the mean for numeric columns ('Sales', 'Cost', 'Profit').

**Step 5: Displaying Results**

1. Display the cleaned CSV data frame after handling missing values.
2. Display the JSON data frame after filling missing values.
3. Display the filtered CSV data frame.
4. Display the sorted CSV data frame.
5. Display the grouped CSV data frame showing the mean values for numeric columns.

**Code:**

import pandas as pd

# Read data from CSV file into a data frame

csv\_file\_path = 'DATA SET.csv'

df\_csv = pd.read\_csv(csv\_file\_path)

# Read data from JSON file into a data frame

json\_file\_path = 'ds.json'

df\_json = pd.read\_json(json\_file\_path)

# Display the first few rows of each data frame to inspect the data

print("CSV Data:")

print(df\_csv.head())

print("\nJSON Data:")

print(df\_json.head())

# Handling missing values

# Drop rows with missing values

df\_csv\_cleaned = df\_csv.dropna()

# Fill missing values with a specific value (e.g., 0)

df\_json\_filled = df\_json.fillna(0)

# Handling outliers

# Assume 'Sales' is the column with outliers

# Replace outliers with the median

median\_value = df\_csv['Sales'].median()

upper\_threshold = df\_csv['Sales'].mean() + 2 \* df\_csv['Sales'].std()

lower\_threshold = df\_csv['Sales'].mean() - 2 \* df\_csv['Sales'].std()

df\_csv['Sales'] = df\_csv['Sales'].apply(lambda x: median\_value if x > upper\_threshold or x < lower\_threshold else x)

# Manipulate and transform data

# Filtering

filtered\_data = df\_csv[df\_csv['Sales'] > 10]

# Sorting

sorted\_data = df\_csv.sort\_values(by='Sales', ascending=False)

# Grouping and calculating mean for numeric columns

numeric\_columns = ['Sales', 'Cost', 'Profit']

grouped\_data = df\_csv.groupby('Category')[numeric\_columns].mean()

# Display the results

print("\nCleaned CSV Data:")

print(df\_csv\_cleaned.head())

print("\nFilled JSON Data:")

print(df\_json\_filled.head())

print("\nFiltered Data:")

print(filtered\_data.head())

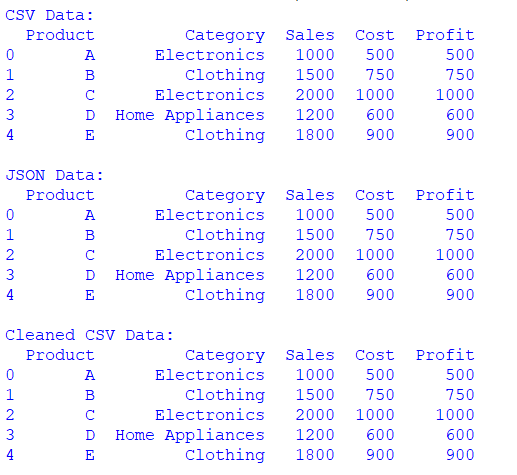
print("\nSorted Data:")

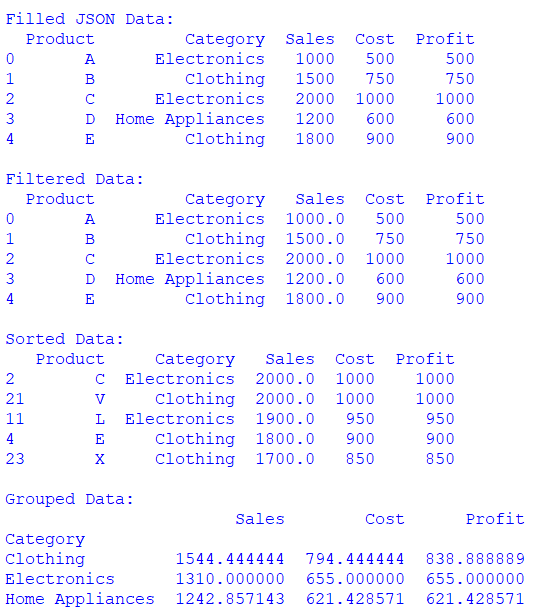
print(sorted\_data.head())

print("\nGrouped Data:")

print(grouped\_data.head())

**Output:**

****

****

**Practical No. 03**

**Aim:** PCA

(Perform on R)

data\_iris<-iris[1:4]

cov\_data<-cov(data\_iris)

cov\_data

eigen\_data<-eigen(cov\_data)

eigen\_data

pca\_data<-princomp(data\_iris,cor='False')

pca\_data

eigen\_data$values

pca\_data$sdev^2

pca\_data$loadings[,1:4]

eigen\_data$vectors

summary(pca\_data)

biplot(pca\_data)

screeplot(pca\_data,types="lines")

model2=pca\_data$loading[,1]

model2

model2\_scores<-as.matrix(data\_iris)%\*%model2

model2\_scores

**Practical NO. 4**

**Aim:** Feature Scaling and Dummification

* Apply feature-scaling techniques like standardization and normalization to numerical features.
* Perform feature dummification to convert categorical variables into numerical representations.

**Feature Scaling:**

Feature scaling is a preprocessing technique used to standardize the range of independent variables or features of the data. It is essential for certain machine learning algorithms that are sensitive to the scale of input features, ensuring that all features contribute equally to the learning process.

**Feature Dummification:**

Feature dummification or one-hot encoding is a technique used to convert categorical variables into numerical representations. This is necessary because many machine learning algorithms require numerical input, and representing categorical variables as binary vectors helps maintain their information.

**Steps:**

1. **Load and Explore Data:** Load the dataset and explore its structure, identify numeric and categorical features.
2. **Feature Scaling:** Apply standardization and normalization to numeric features.
3. **Feature Dummification:** Convert categorical variables into numerical representations using one-hot encoding.
4. **Combine Features:** Combine scaled numeric features with one-hot encoded categorical features.
5. **Display Resulting Dataset:** Display the final dataset after both feature scaling and dummification.

**Code:**

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

# Define the data

data = {

'Product': ['Apple\_Juice', 'Banana\_Smoothie', 'Orange\_Jam', 'Grape\_Jelly', 'Kiwi\_Parfait', 'Mango\_Chutney', 'Pineapple\_Sorbet', 'Strawberry\_Yogurt', 'Blueberry\_Pie', 'Cherry\_Salsa'],

'Category': ['Apple', 'Banana', 'Orange', 'Grape', 'Kiwi', 'Mango', 'Pineapple', 'Strawberry', 'Blueberry', 'Cherry'],

'Sales': [1200, 1700, 2200, 1400, 2000, 1000, 1500, 1800, 1300, 1600],

'Cost': [600, 850, 1100, 700, 1000, 500, 750, 900, 650, 800],

'Profit': [600, 850, 1100, 700, 1000, 500, 750, 900, 650, 800]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Display the original dataset

print("Original Dataset:")

print(df)

# Step 1: Feature Scaling (Standardization and Normalization)

numeric\_columns = ['Sales', 'Cost', 'Profit']

scaler\_standardization = StandardScaler()

scaler\_normalization = MinMaxScaler()

df\_scaled\_standardized= pd.DataFrame(scaler\_standardization.fit\_transform(df[numeric\_columns]), columns=numeric\_columns)

df\_scaled\_normalized= pd.DataFrame(scaler\_normalization.fit\_transform(df[numeric\_columns]), columns=numeric\_columns)

# Combine the scaled numeric features with the categorical features

df\_scaled = pd.concat([df\_scaled\_standardized, df.drop(numeric\_columns, axis=1)], axis=1)

# Display the dataset after feature scaling

print("\nDataset after Feature Scaling:")

print(df\_scaled)

# Step 2: Feature Dummification

# Identify categorical columns

categorical\_columns = ['Product', 'Category']

# Create a column transformer for dummification

preprocessor = ColumnTransformer(

transformers=[

('categorical', OneHotEncoder(), categorical\_columns)

],

remainder='passthrough'

)

# Apply the column transformer to the dataset

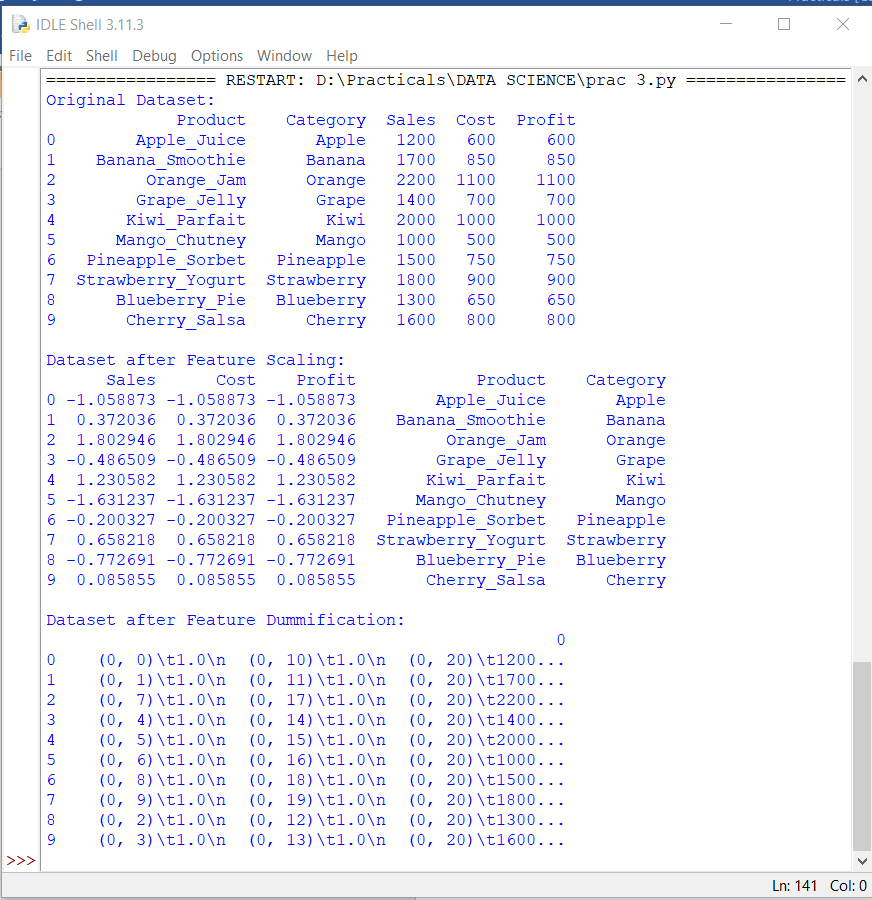
df\_dummified = pd.DataFrame(preprocessor.fit\_transform(df))

# Display the dataset after feature dummification

print("\nDataset after Feature Dummification:")

print(df\_dummified)

**Output:**

****

**Practical No. 05**

**Aim:** Hypothesis Testing

* Formulate null and alternative hypotheses for a given problem.
* Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chi-square test).
* Interpret the results and draw conclusions based on the test outcomes.

**Hypothesis Testing:**

Hypothesis testing is a statistical method used to make inferences about population parameters based on sample data. It involves the formulation of a null hypothesis (H0) and an alternative hypothesis (H1), and the collection of sample data to assess the evidence against the null hypothesis. The goal is to determine whether there is enough evidence to reject the null hypothesis in favor of the alternative hypothesis.

1. **Formulate Hypotheses:**
   * Null Hypothesis (*H*0​): The average caffeine content per serving is 80 mg (*μ*=80).
   * Alternative Hypothesis (*H*1​): The average caffeine content per serving is different from 80 mg (*μ*≠80).
2. **Statistical Test:**
   * A t-test is appropriate since you are comparing a sample mean to a known population mean, and the sample size is small.
3. **Data Collection:**
   * Randomly select 30 cans of the energy drink and measure the caffeine content in each.
4. **Conducting the Hypothesis Test:**

a. **Collect Data:**

* + Calculate the sample mean () and standard deviation (*s*) from the 30 samples.

b. **Set Significance Level (*α*):**

* + Choose a significance level (*α*=0.05,0.01,0.10).

c. **Calculate the Test Statistic (t-value):**

* + Use the formula *t*=*s*/*n*​−*μ*​.

d. **Determine Degrees of Freedom:**

* + For a one-sample t-test, degrees of freedom (*df*) is *n*−1.

e. **Find Critical Values or P-value:**

* + Use a t-table or statistical software to find the critical t-values for a two-tailed test at the chosen significance level.

f. **Make a Decision:**

* + If the t-value falls outside the critical region, reject the null hypothesis. If it falls inside, fail to reject.

g. **Interpretation:**

* + If you reject the null hypothesis, there is enough evidence to suggest that the average caffeine content per serving is different from 80 mg. If you fail to reject the null hypothesis, there is not enough evidence to suggest a difference in the average caffeine content.

1. **Conclusion:**
   * Draw conclusions about the energy drink's caffeine content, considering both statistical and practical significance. Consider decisions relevant to the context of the problem.

**Code:**

import numpy as np

from scipy import stats

import matplotlib.pyplot as plt

# Generate two samples for demonstration purposes

np.random.seed(42)

sample1 = np.random.normal(loc=10, scale=2, size=30)

sample2 = np.random.normal(loc=12, scale=2, size=30)

# Perform a two-sample t-test

t\_statistic, p\_value = stats.ttest\_ind(sample1, sample2)

# Set the significance level

alpha = 0.05

print("Results of Two-Sample t-test:")

print(f"t-statistic: {t\_statistic}")

print(f"p-value: {p\_value}")

print(f"Degrees of Freedom: {len(sample1) + len(sample2) - 2}")

# Plot the distributions

plt.figure(figsize=(10, 6))

plt.hist(sample1, alpha=0.5, label='Sample 1', color='blue')

plt.hist(sample2, alpha=0.5, label='Sample 2', color='orange')

plt.axvline(np.mean(sample1), color='blue', linestyle='dashed', linewidth=2)

plt.axvline(np.mean(sample2), color='orange', linestyle='dashed', linewidth=2)

plt.title('Distributions of Sample 1 and Sample 2')

plt.xlabel('Values')

plt.ylabel('Frequency')

plt.legend()

# Highlight the critical region if null hypothesis is rejected

if p\_value < alpha:

critical\_region = np.linspace(min(sample1.min(), sample2.min()), max(sample1.max(), sample2.max()), 1000)

plt.fill\_between(critical\_region, 0, 5, color='red', alpha=0.3, label='Critical Region')

# Show the observed t-statistic

plt.text(11, 5, f'T-statistic: {t\_statistic:.2f}', ha='center', va='center', color='black', backgroundcolor='white')

# Show the plot

plt.show()

# Draw Conclusions

# Drawing Conclusions

if p\_value < alpha:

if np.mean(sample1) > np.mean(sample2):

print("Conclusion: There is significant evidence to reject the null hypothesis.")

print("Interpretation: The mean caffeine content of Sample 1 is significantly higher than that of Sample 2.")

# Additional context and practical implications can be added here.

else:

print("Conclusion: There is significant evidence to reject the null hypothesis.")

print("Interpretation: The mean caffeine content of Sample 2 is significantly higher than that of Sample 1.")

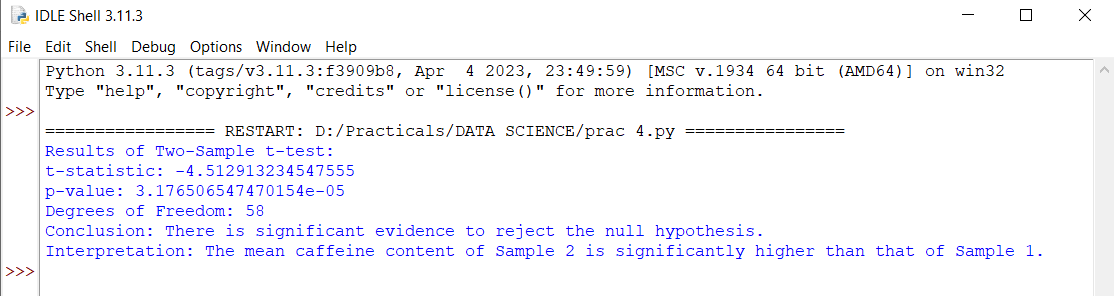
# Additional context and practical implications can be added here.

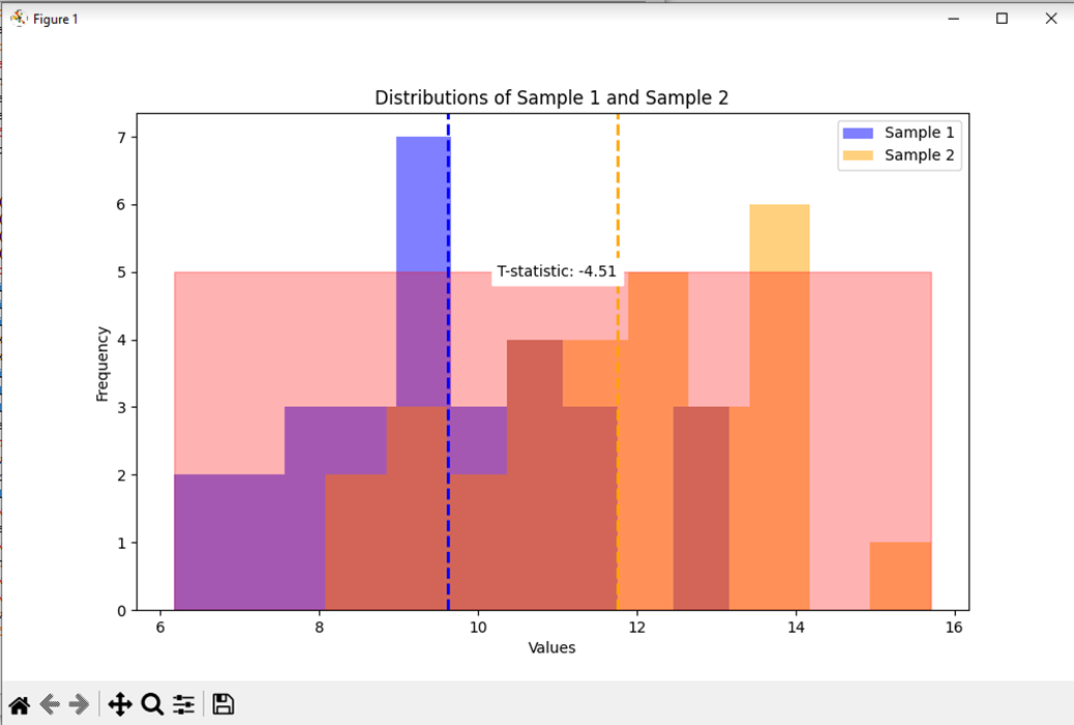
else:

print("Conclusion: Fail to reject the null hypothesis.")

print("Interpretation: There is not enough evidence to claim a significant difference between the means.")

**Output:**





**Practical No. 06**

Aim: Logistic Regression and Decision Tree

install.packages("party")

library(party)

print(head(readingskills))

input.data<-readingSkills[c(1:105),]

png(file="decisiontree.png")

output.tree<-ctree(nativeSpeaker~age+shoeSize+score,data=input.data)

plot(output.tree)

dev.off()

plot(output.tree)

**Practical No. 07**

Aim: ANOVA(Analysis of Variance) perform on R

install.packages("readr")

install.packages("ggplot2")

install.packages("multcompView")

install.packages("dplyr")

library(readr)

library(ggplot2)

library(multcompView)

library(dplyr)

radon <- data.frame(D = c(0.37,0.37,0.37,0.37,0.51,0.51,0.51,0.51,0.71,0.71,0.71,0.71,1.02,1.02,1.02,1.02,1.4,1.4,1.4,1.4,1.99,1.99,1.99,1.99),

RR = c(80,83,83,85,75,75,79,79,74,73,76,77,67,72,74,74,62,62,67,69,60,61,64,66))

str(radon)

plot(radon$D, radon$RR)

anova.rr <- aov(RR ~ as.factor(D), data = radon)

summary(anova.rr)

tukey.rr <- TukeyHSD(anova.rr)

print(tukey.rr)

**Practical No. 08**

Aim: K means Clustering(wholesale.csv)

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

data = pd.read\_csv("wholesale.csv")

categorical\_feature = ['Channel','Region']

continuous\_feature = ['Fresh','Milk','Grocery','Frozen','Detergents\_Paper','Delicassen']

for col in categorical\_feature:

dummies = pd.get\_dummies(data[col], prefix=col)

data = pd.concat([data, dummies], axis=1)

data.drop(col, axis=1, inplace=True)

mms = MinMaxScaler()

data\_transformed = mms.fit\_transform(data)

# Adjusting the range of K based on the number of samples in the dataset

K = range(1, min(15, len(data) + 1))

Sum\_of\_squared\_distances = []

for k in K:

km = KMeans(n\_clusters=k)

km = km.fit(data\_transformed)

Sum\_of\_squared\_distances.append(km.inertia\_)

plt.plot(K, Sum\_of\_squared\_distances, 'bx-')

plt.xlabel('k')

plt.ylabel('Sum\_of\_squared\_distances')

plt.title('Elbow Method for Optimal k')

plt.show()