**Practical - 1**

**Aim: Introduction to Excel**

* **Perform conditional formatting on a dataset using various criteria.**
* **Create a pivot table to analyze and summarize data.**
* **Use the 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.**

**Requirements : Platform : MS Excel**

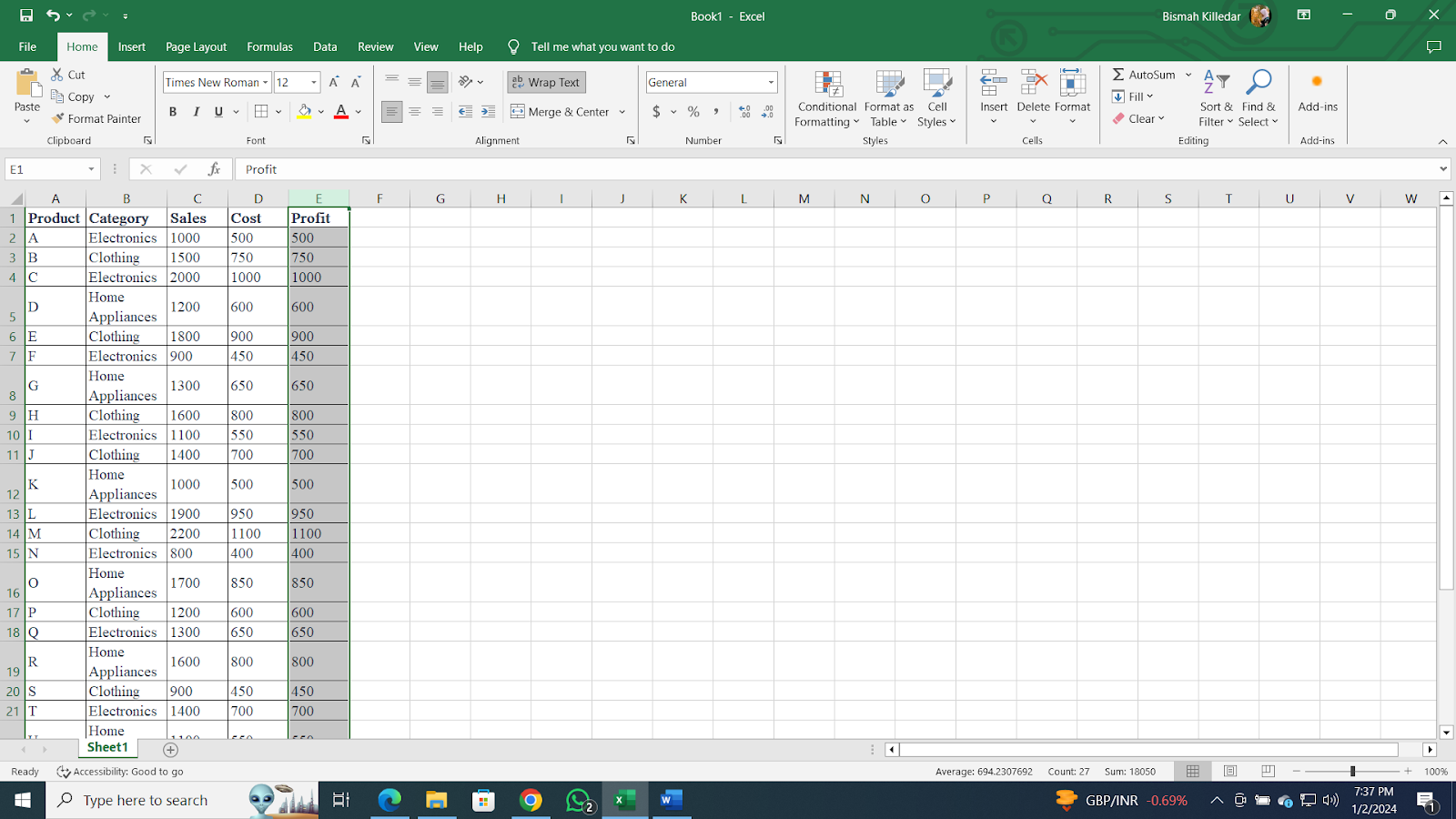
**Dataset: Pivot Table.xlsx, Use conditional formatting.xlsx, VLOOKUP.xlsx**

* **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:

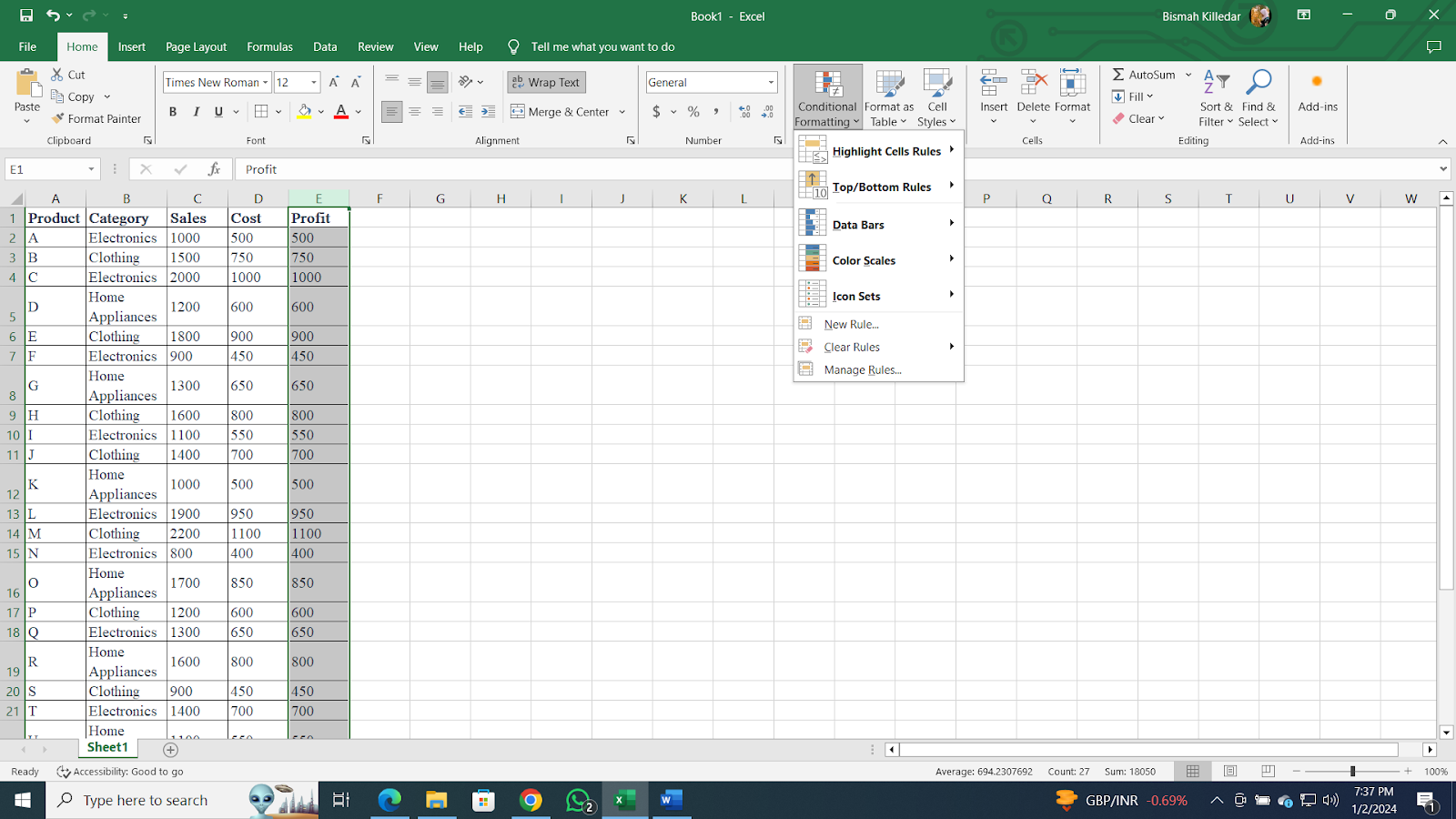
**Steps:**

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

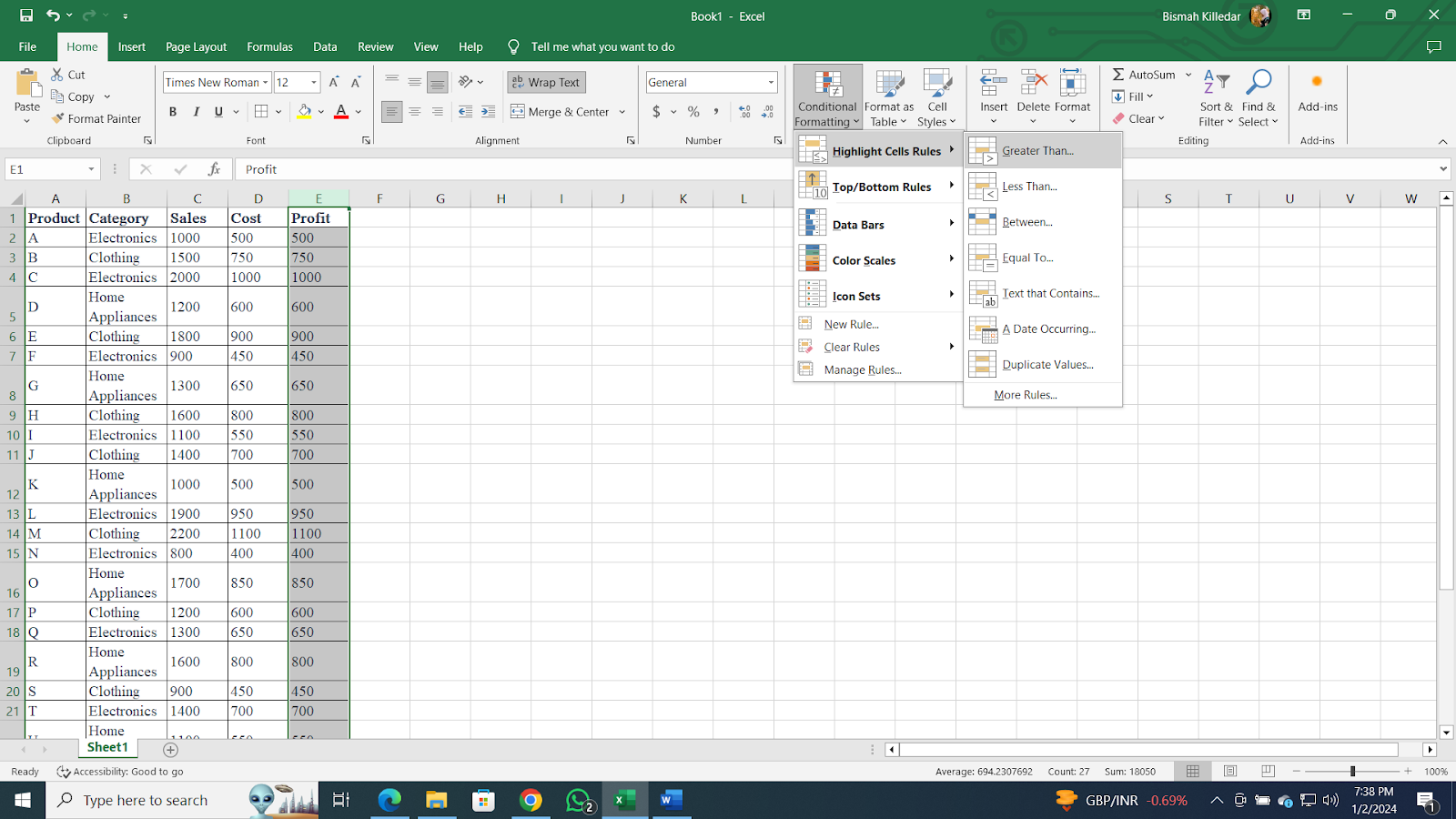


2. Go to the "Home" tab on the ribbon.

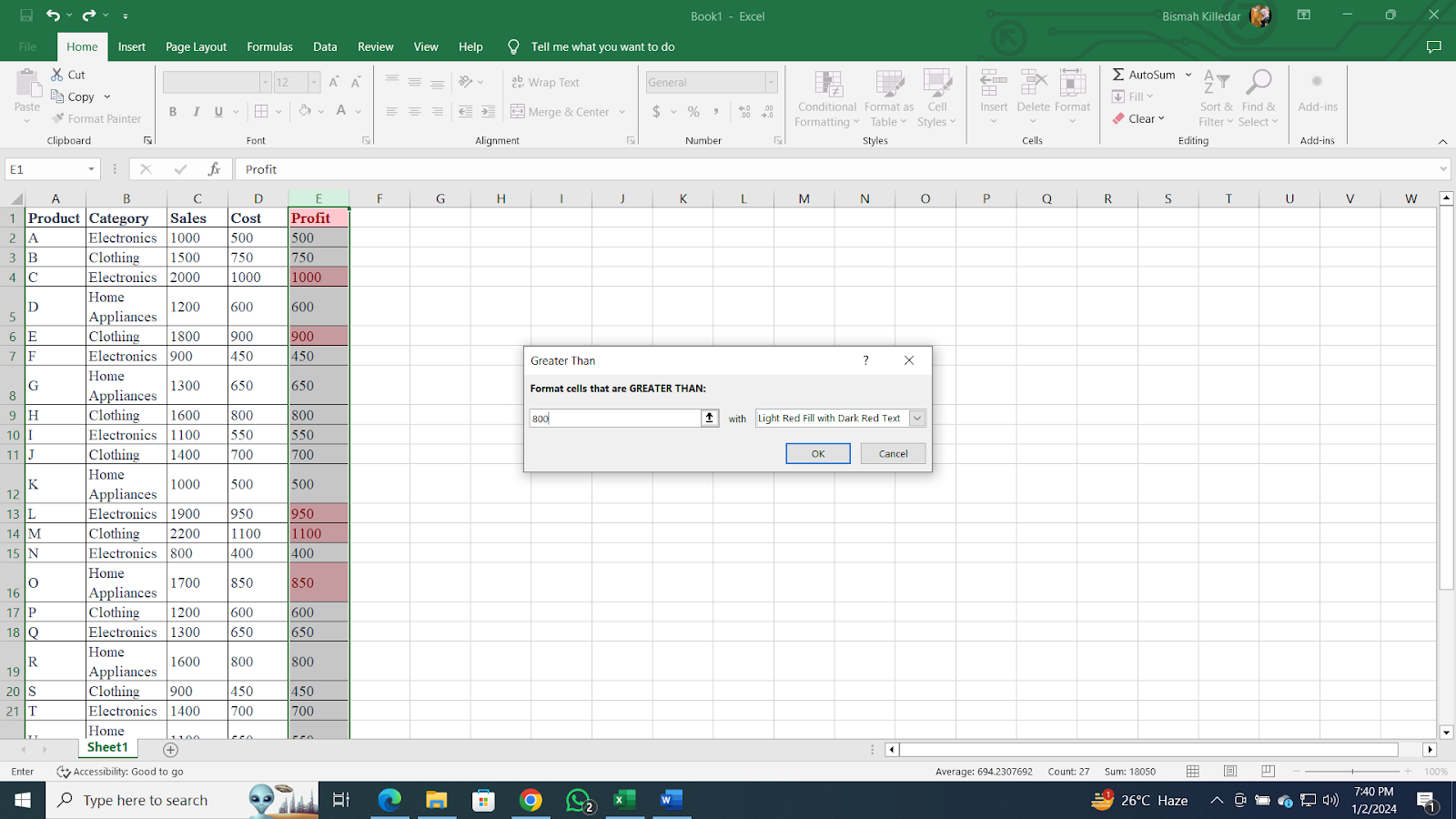
3. Click on "Conditional Formatting" in the toolbar.



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

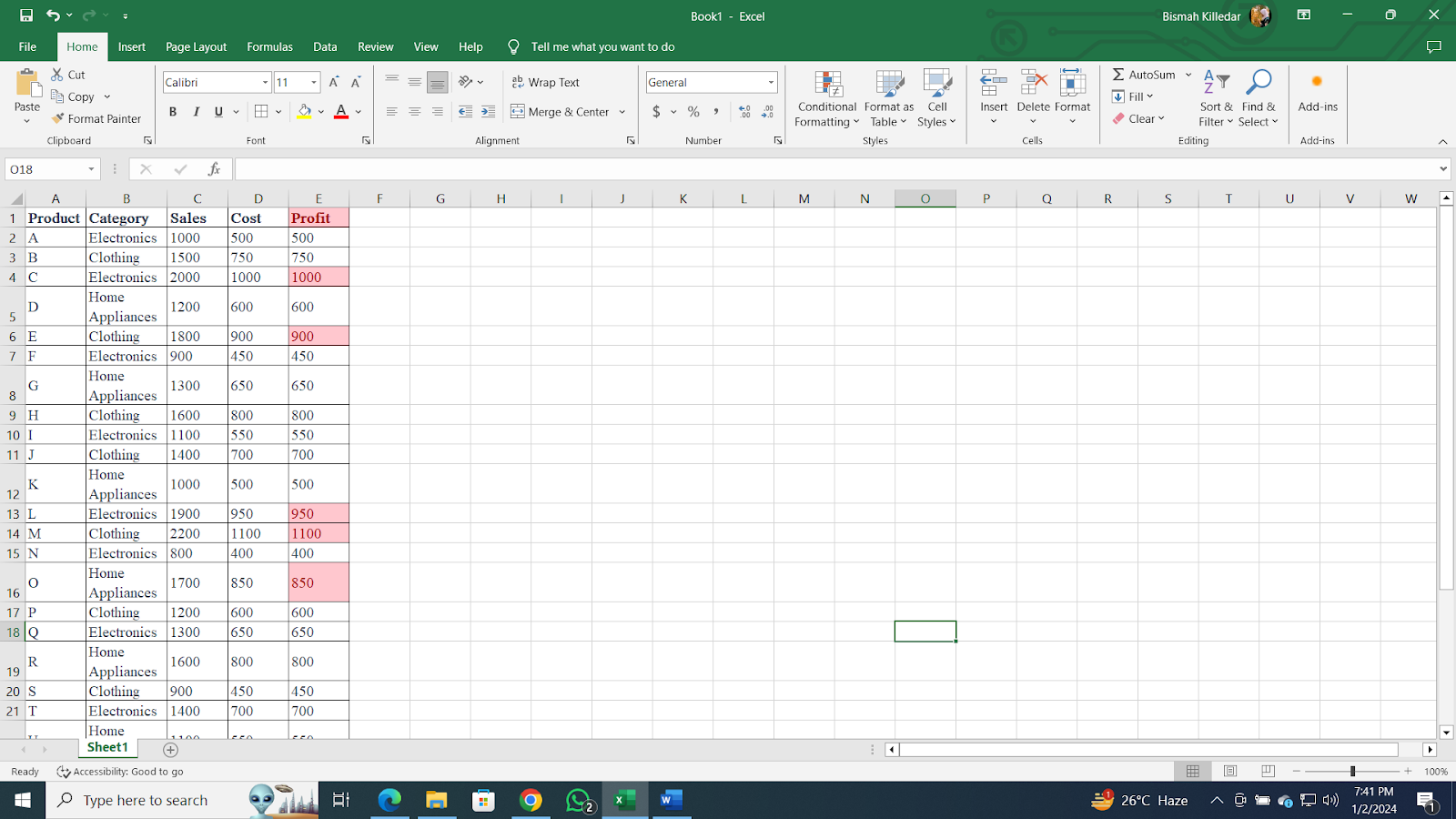


5. Enter the threshold value as 800.



6. Customize the formatting options (e.g., choose a fill color).

7. Click "OK" to apply the rule.

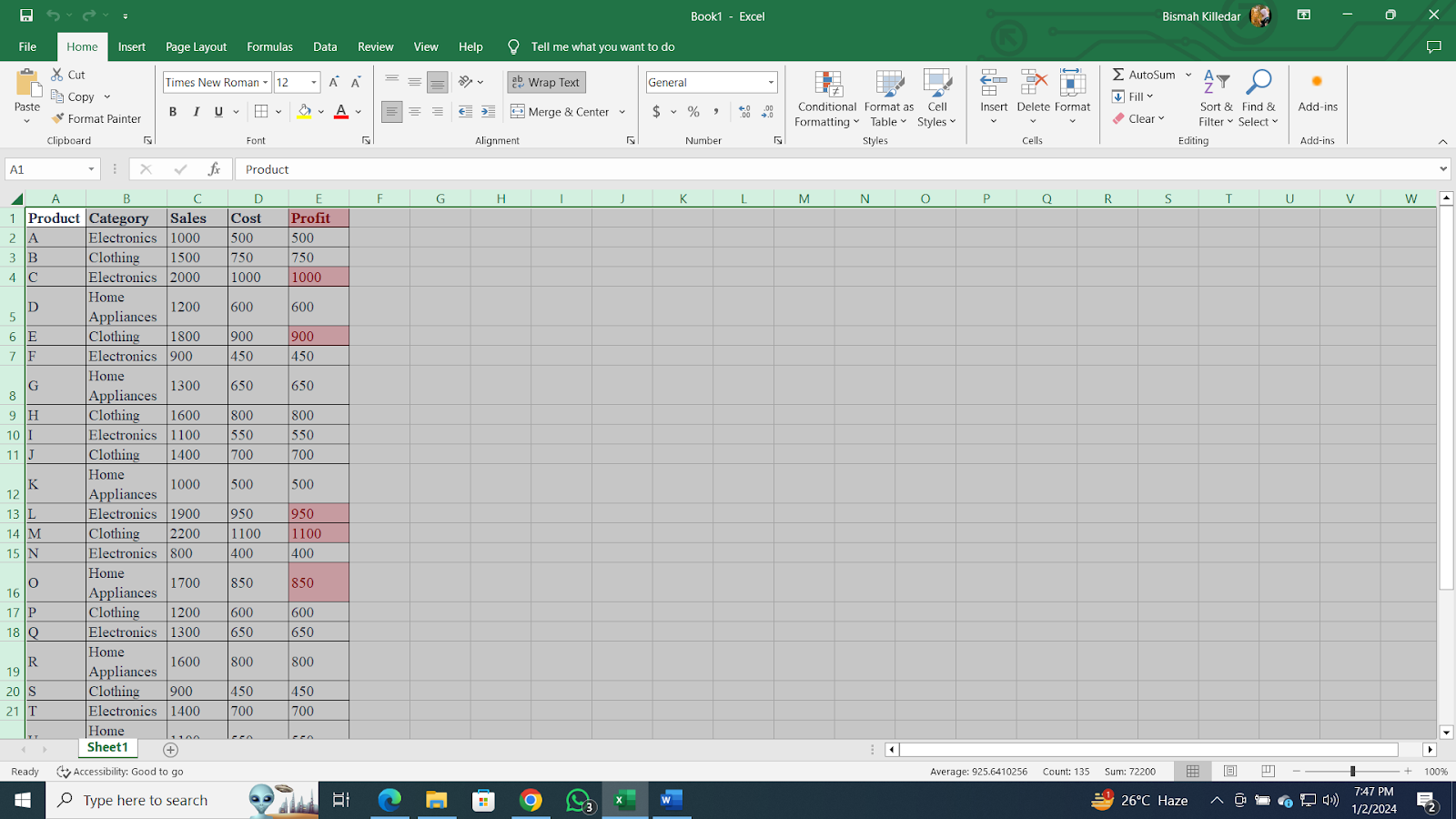


* **Create a pivot table to analyze and summarize data.**

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

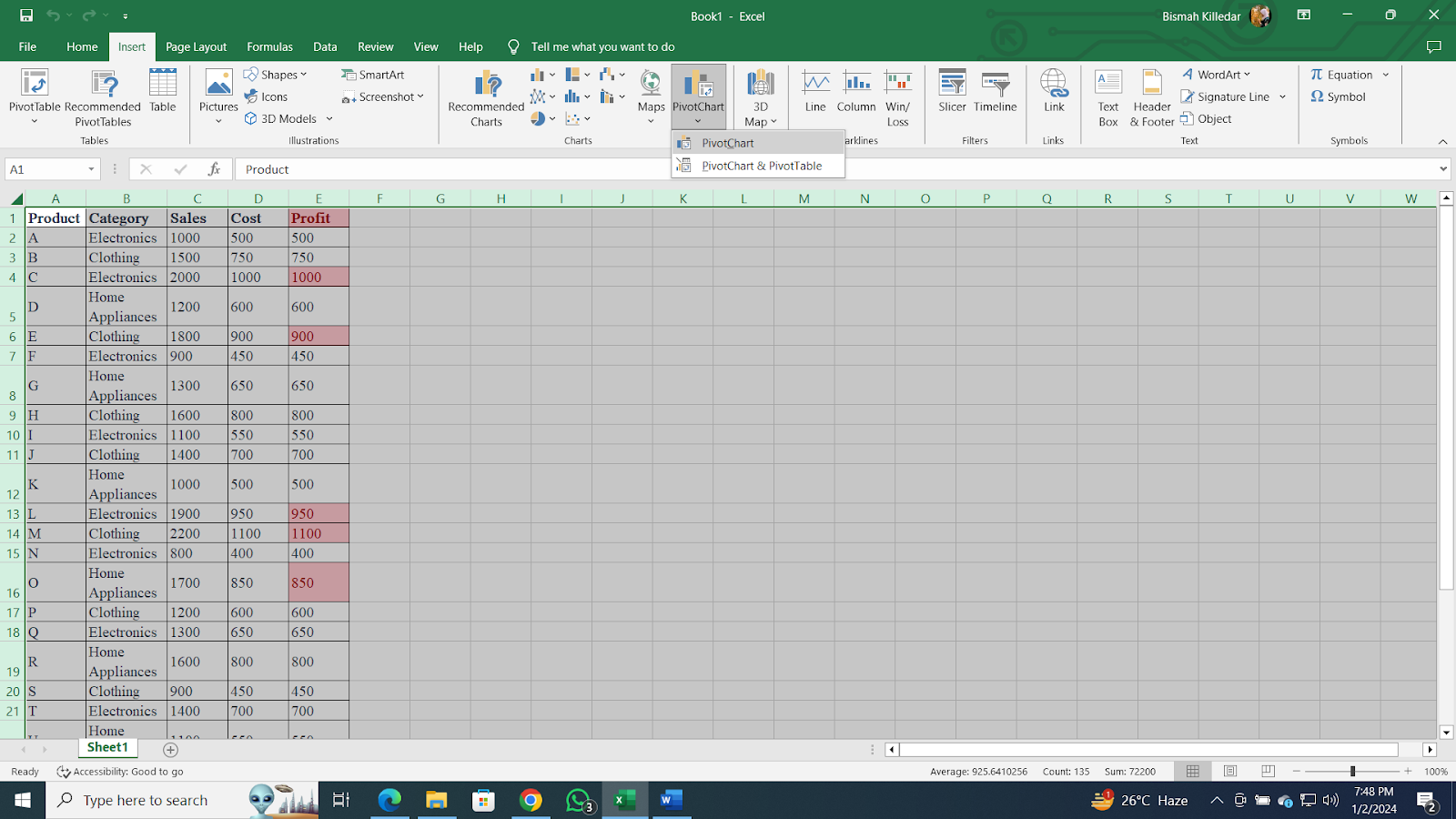
**Steps:**

1. Select the entire dataset including headers.

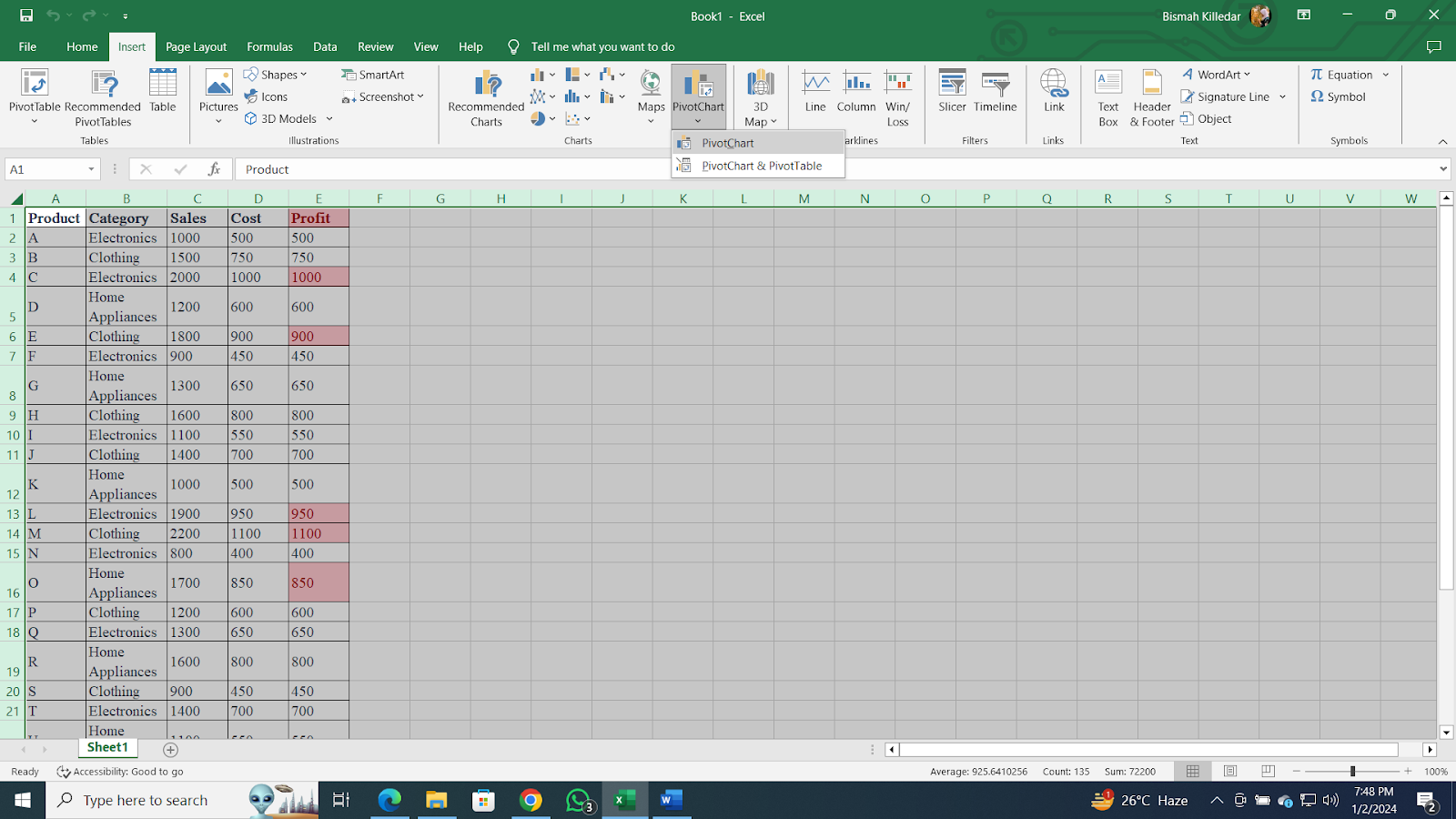


2. Go to the "Insert" tab on the ribbon.

3. Click on "PivotTable."

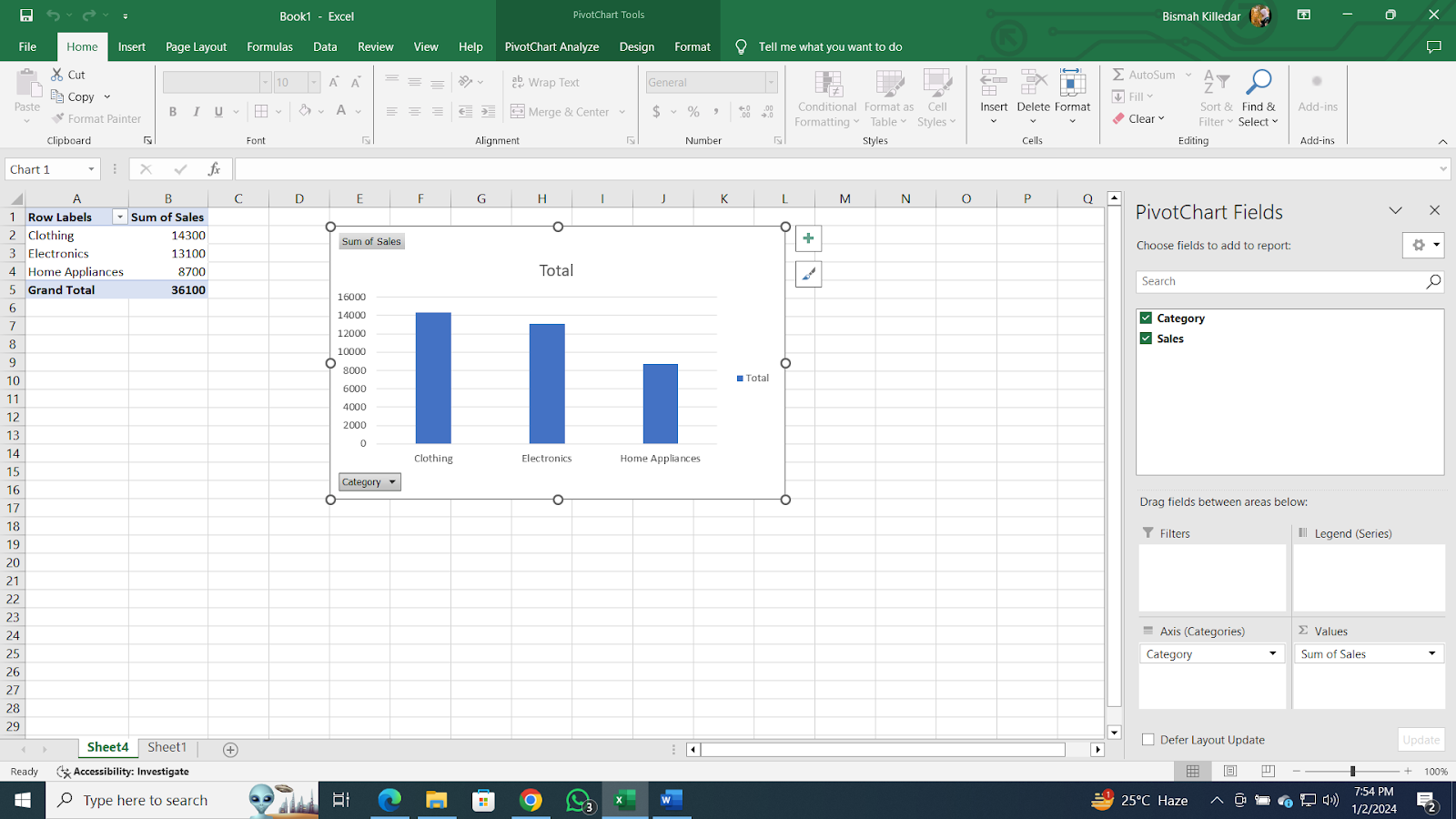


4. Choose where you want to place the PivotTable (e.g., new worksheet).



5. Drag "Category" to the Rows area.

6. Drag "Sales" to the Values area, choosing the sum function.



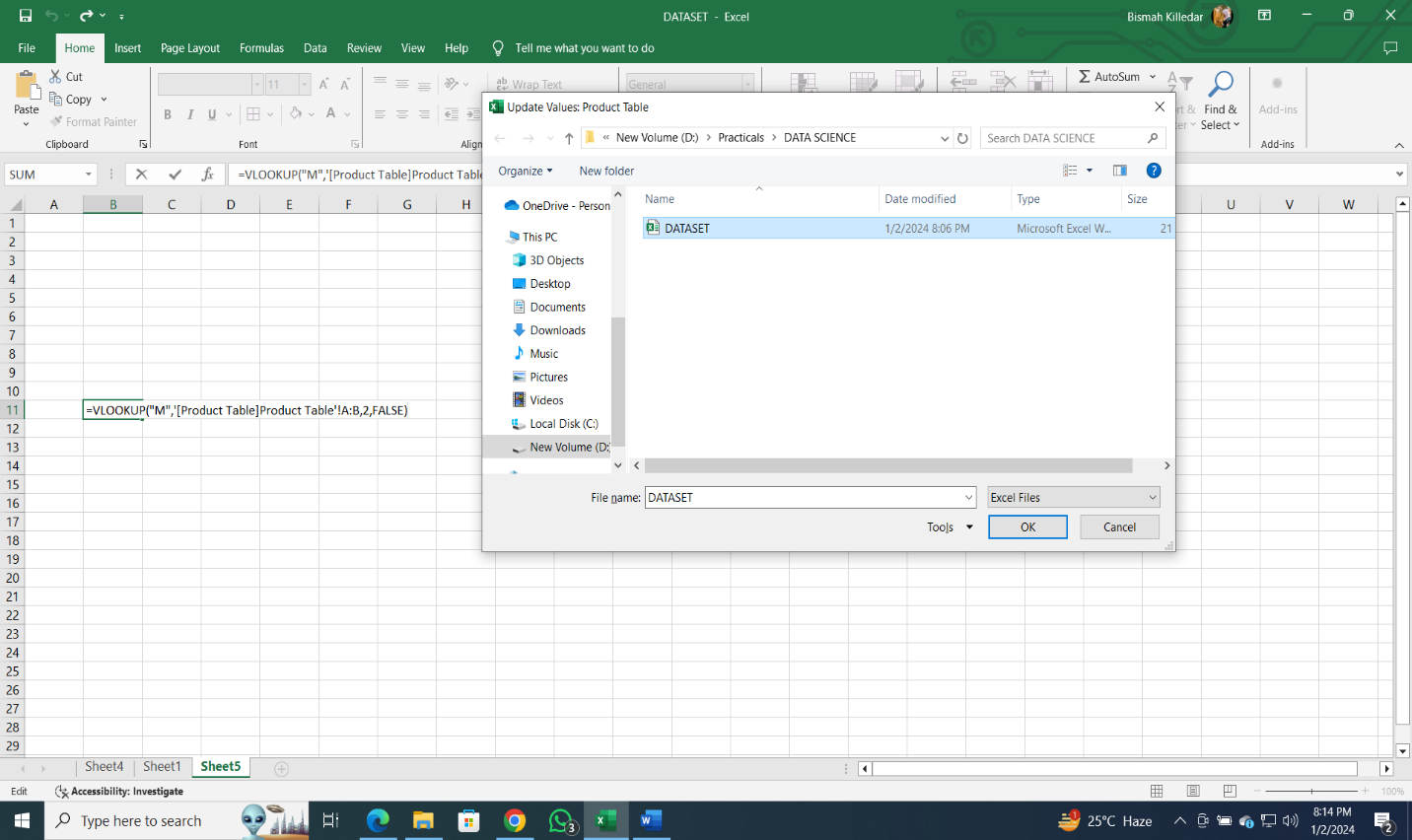
* **Use VLOOKUP function to retrieve information from a different worksheet or table.**

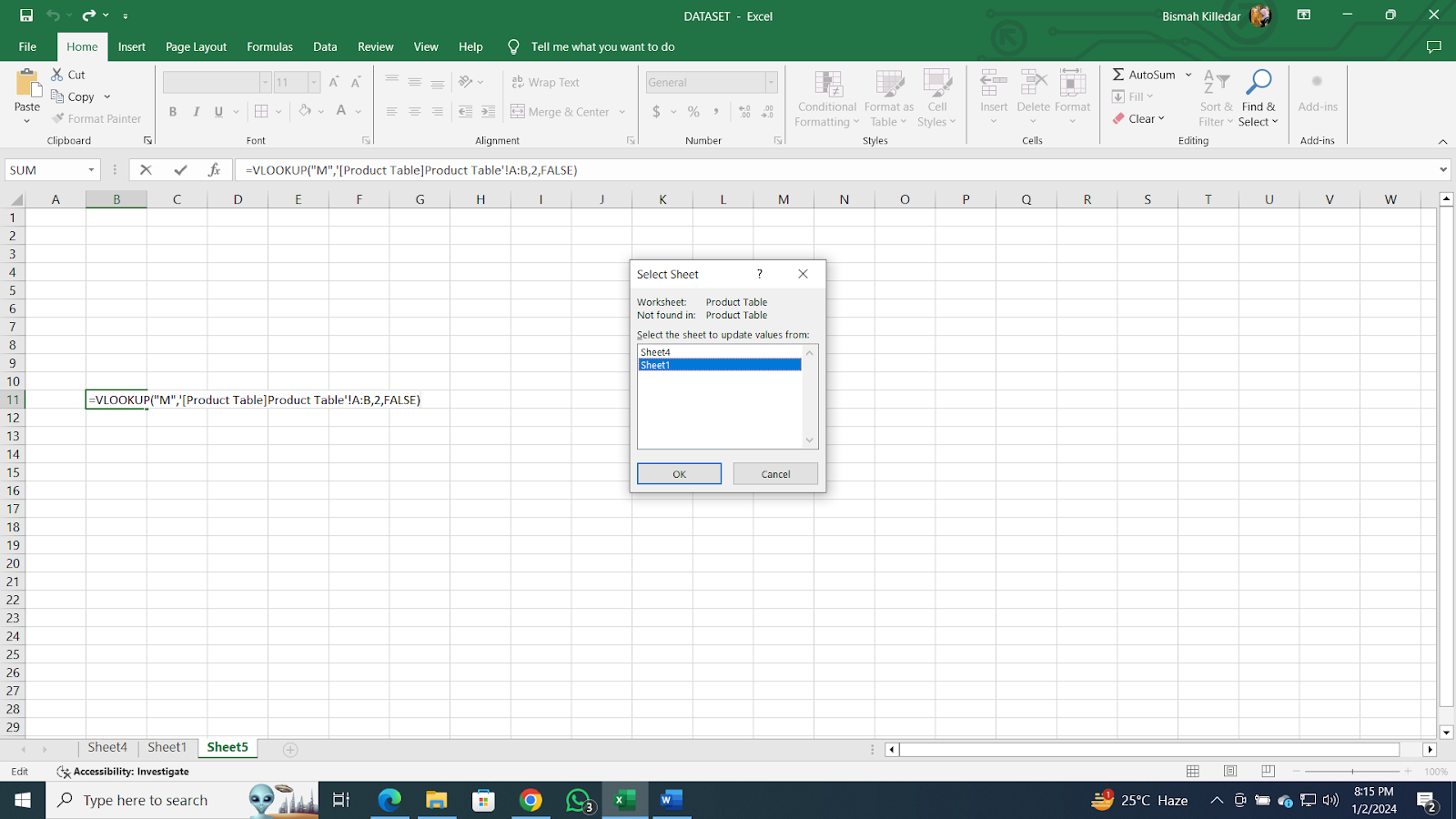
Use the VLOOKUP function to retrieve the category of "Product M" from a separate worksheet named "Product Table" using following steps:

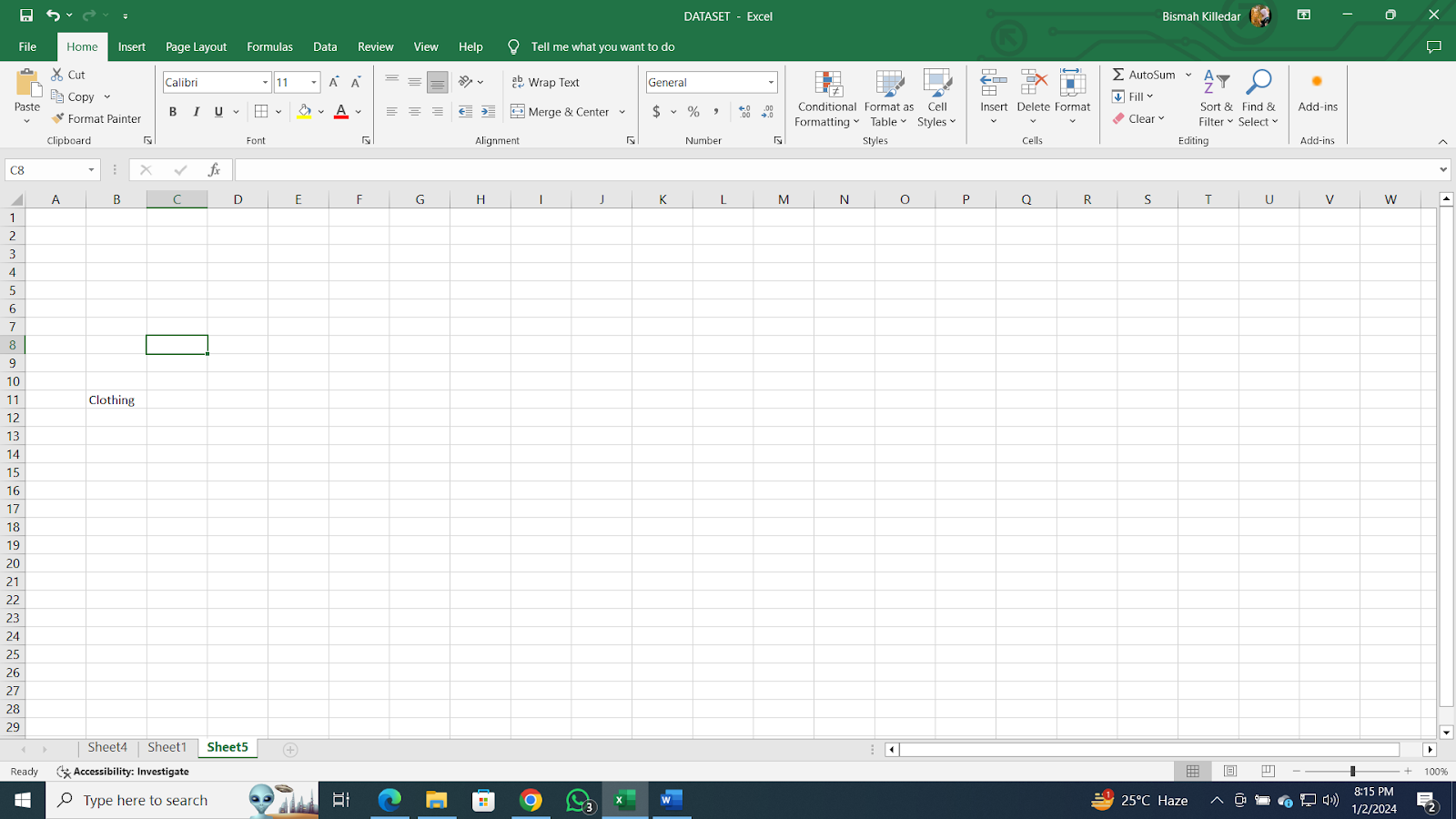
**Steps:**

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)**





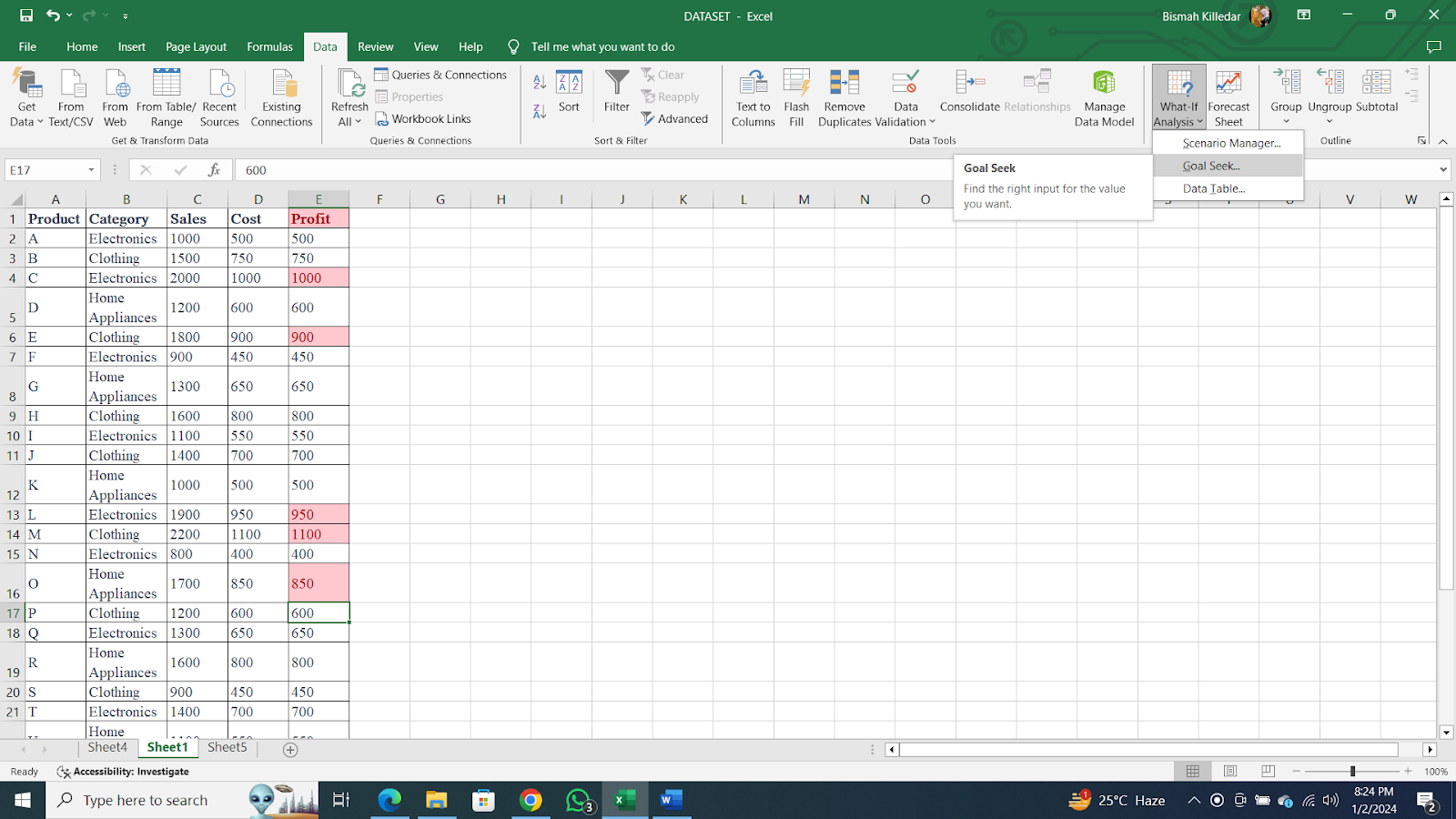


* **Perform what-if analysis using Goal Seek to determine input values for desired output.**

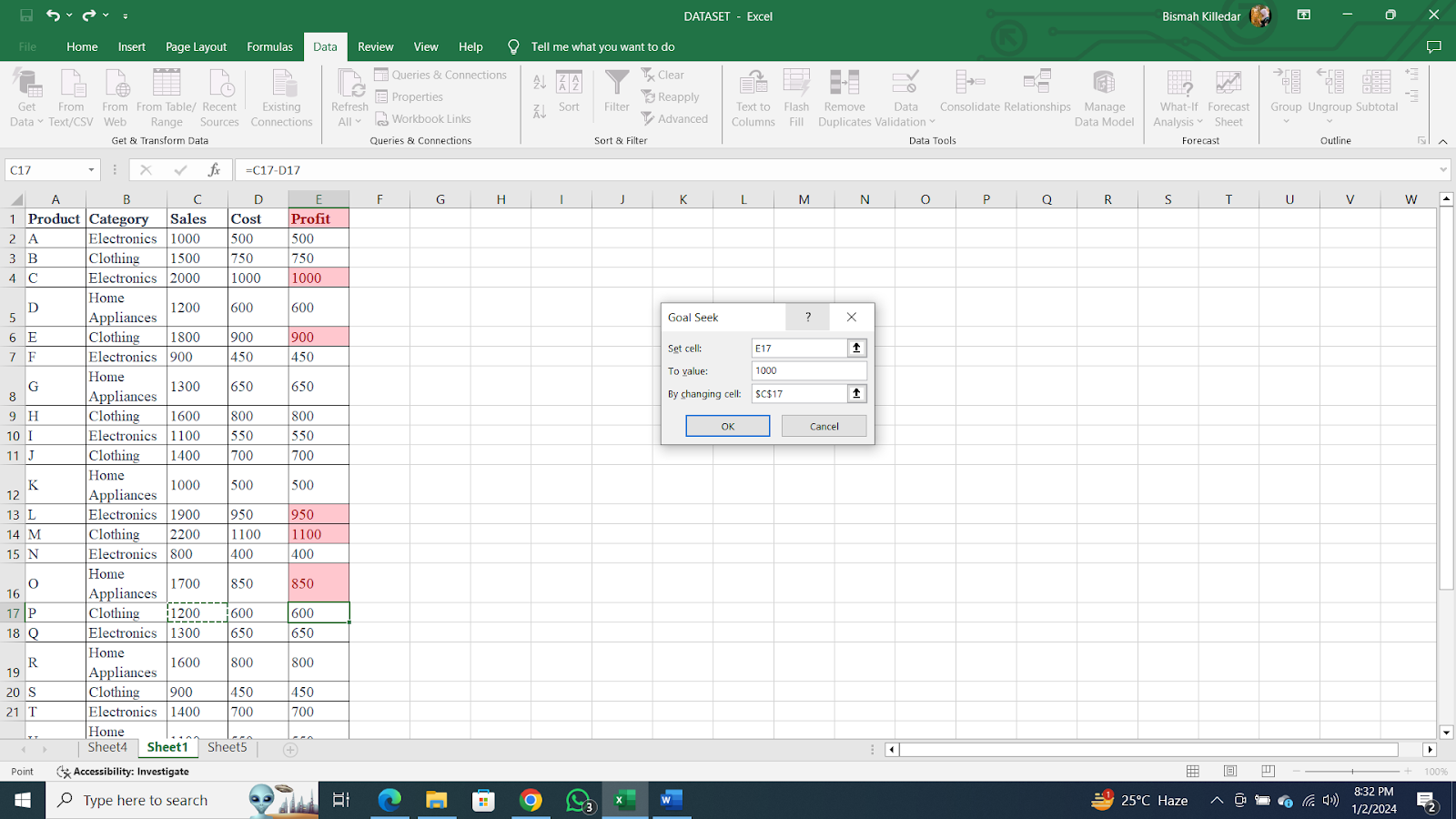
      Use Goal Seek to find the required sales for "Product P" to achieve a profit of 1000 using the following steps.

**Steps:**

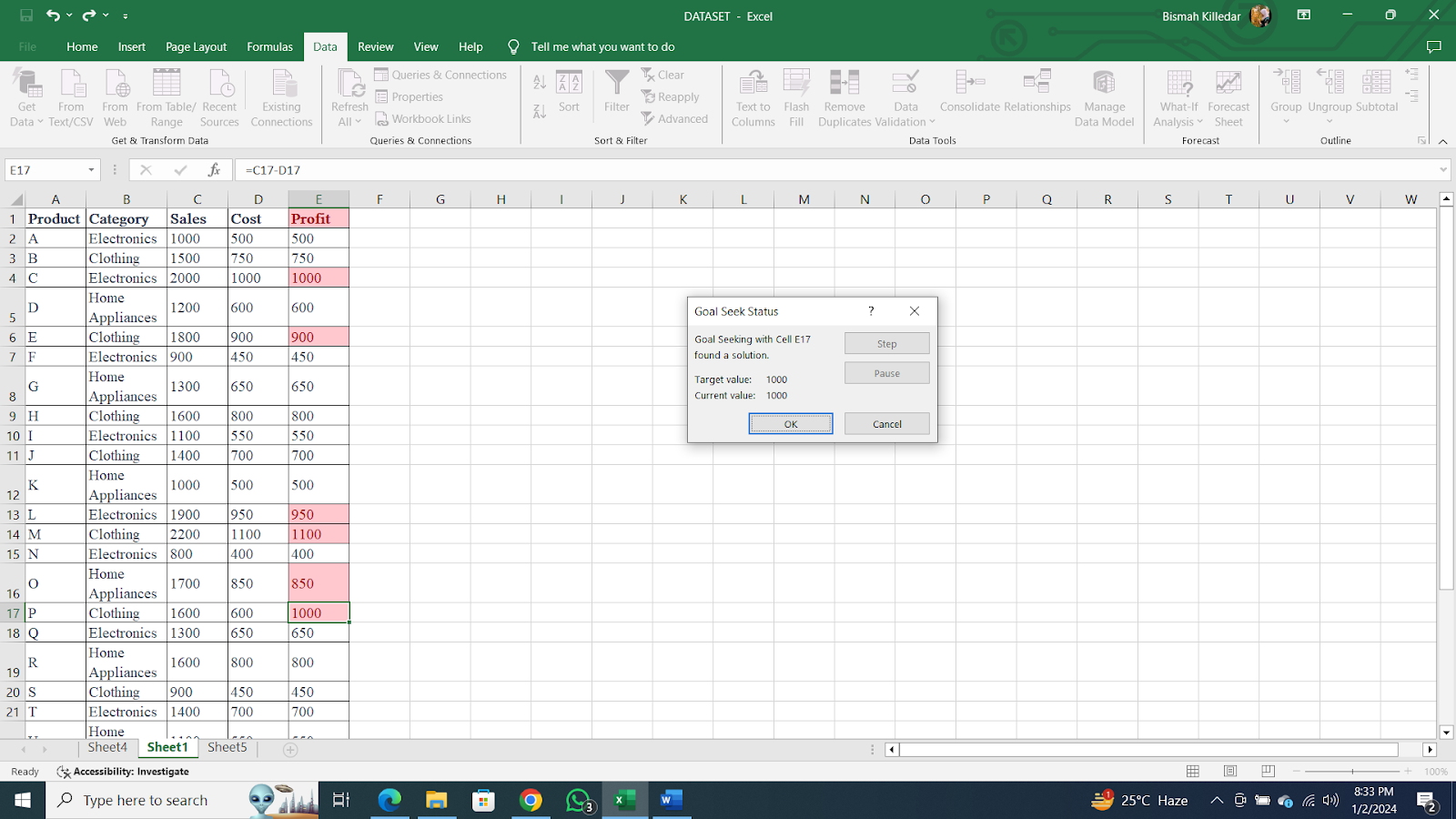
1. Identify the cell containing the formula for "Profit" for "Product P" (let's assume it's in cell E17).
2. Go to the "Data" tab on the ribbon.
3. Click on "What-If Analysis" and select "Goal Seek."



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



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



**Practical -2**

**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.**

**Requirements: Dataset : companydata.csv,**

**Platform: Google colab**

**Dataset:**

| **Job Position** | **Years of Experience** | **Salary (in USD per year)** |
| --- | --- | --- |
| **CEO** | **5** | **100000** |
| **Senior manager** | **4** | **80000** |
| **Junior manager** | **3** | **NaN** |
| **Employee** | **NaN** | **40000** |
| **Assistant staff** | **1** | **20000** |

**Code & Output:**

1. **Read data from CSV and JSON files into a data frame.**

import pandas as pd

#To create data set

#df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1], 'Salary':[100000,80000,None,40000, 20000]})

#Read the data

df= pd.read\_csv("companydata.csv")

df



**2. Perform basic data pre-processing tasks such as handling missing values and outliers**

# Handling missing values

df['Years of Experience'].fillna(df['Years of Experience'].median(), inplace=True)

df['Salary'].fillna(df['Salary'].mean(), inplace=True)

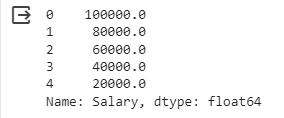
df



# Handling outliers in the 'Salary' column using winsorizing

df['Salary'] = winsorize(df['Salary'], limits=[0.05, 0.05])

df['Salary']



**3. Manipulate and transform data using functions like filtering, sorting, and grouping.**

# Filtering: Selecting rows where 'Years of Experience' is not null

df\_filtered = df[df['Years of Experience'].notnull()]

# Sorting: Sorting the DataFrame by 'Salary' in descending order

df\_sorted = df.sort\_values(by='Salary', ascending=False)

# Grouping: Calculating the average salary for each job position

df\_grouped = df.groupby('Job Position')['Salary'].mean().reset\_index()

# Viewing the results

print("Original DataFrame:")

print(df.head())

print("\nFiltered DataFrame (Rows with non-null 'Years of Experience'):")

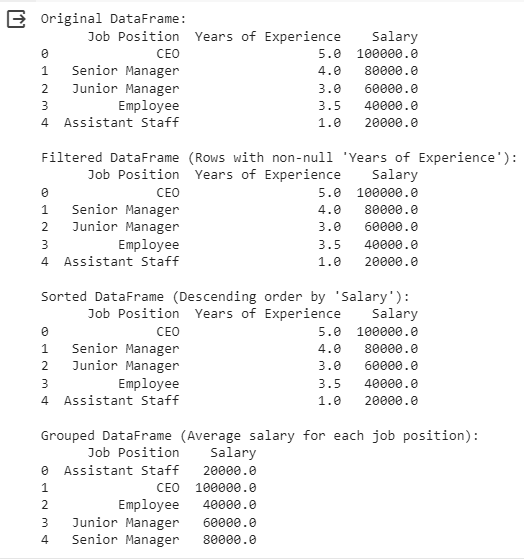
print(df\_filtered)

print("\nSorted DataFrame (Descending order by 'Salary'):")

print(df\_sorted)

print("\nGrouped DataFrame (Average salary for each job position):")

print(df\_grouped)



**Practical - 3**

**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

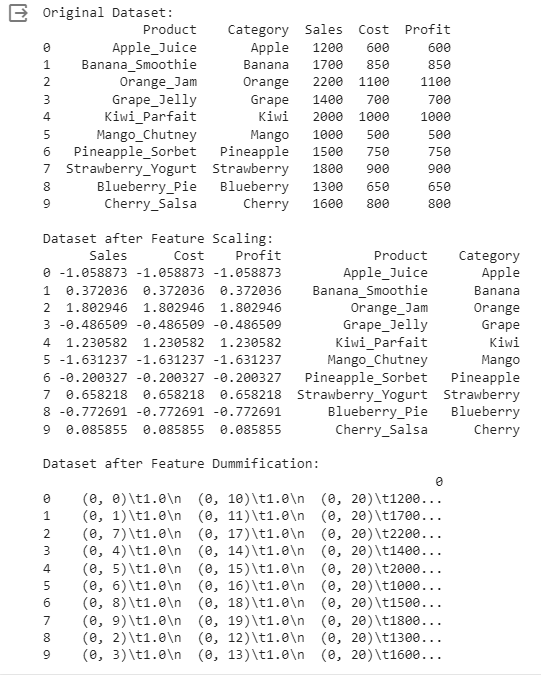
df\_dummified = pd.DataFrame(preprocessor.fit\_transform(df))

# Display the dataset after feature dummification

print("\nDataset after Feature Dummification:")

print(df\_dummified)

**Output:**

****

**Practical - 4**

**Aim: Hypothesis Testing**

* **Formulate null and alternative hypotheses for a given problem.**
* **Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chisquare 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.

5. **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

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.")

    else:

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

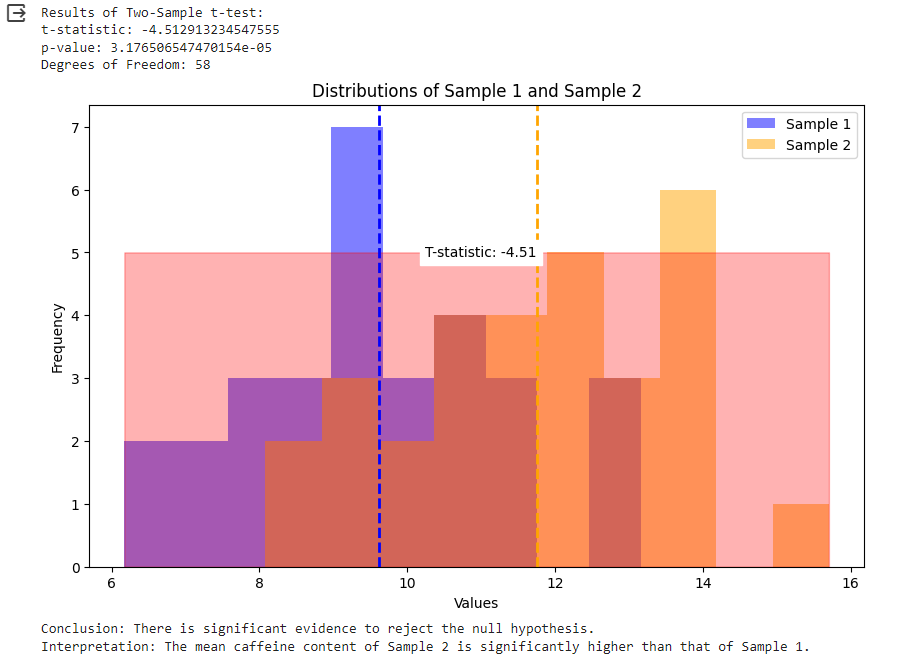
        print("Interpretation: The mean caffeine content of Sample 2 is significantly higher than that of Sample 1.")

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 - 5**

**Aim: ANOVA (Analysis of Variance)**

* **Perform one-way ANOVA to compare means across multiple groups.**
* **Conduct post-hoc tests to identify significant differences between group means.**

**Requirements: RStudio, WholesaleCustomersData.csv**

**Steps:**

**Download the “WholesaleCustomersData.csv” from Kaggle**

**Open RStudio >  Create A New Project > Import Dataset > From Text (base)**

**In the Console, Type the following Command-**

**WholesaleCustomerData$Region <- factor(WholesaleCustomerData$Region)**

**Here, “WholesaleCustomerData” is the name of the dataset**

**# Perform the ANOVA**

**one.way <- aov(Grocery ~ Region, data = WholesaleCustomerData)**

**# Check the summary**

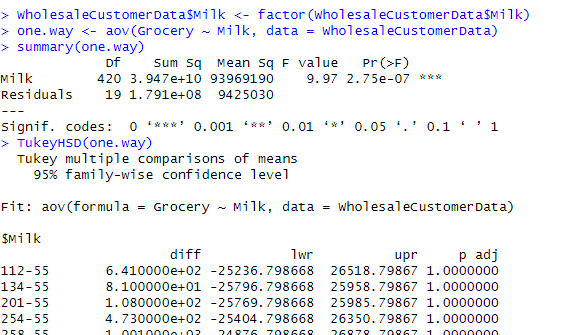
**summary(one.way)**

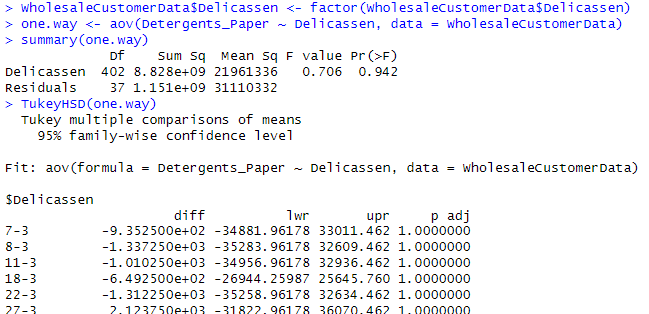
**# Try Tukey's HSD test**

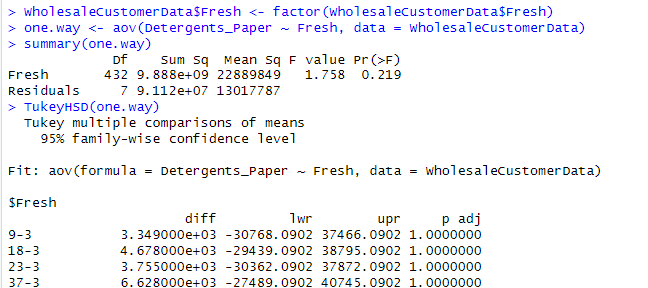
**TukeyHSD(one.way)**

**---------------------------------------------------------------------------------------------------**

**Comparing Multiple Groups**

****

****

****

**Practical - 6**

**Aim: Regression and Its Types**

* **Implement simple linear regression using a dataset.**
* **Explore and interpret the regression model coefficients and goodness-of-fit measures.**
* **Extend the analysis to multiple linear regression and assess the impact of additional predictors**

**Requirements: RStudio, LungCap.xls**

**Download the “LungCap.xls” from Kaggle**

**Open RStudio >  Create A New Project > Import Dataset > From Excel > Browse > Import**

**In the Console, Type the following Command-**

>LungCap <- read\_excel("C:/Users/admin/Downloads/LungCap.xls")

> View(LungCap)

> attach(LungCap)

> names(LungCap)

> class(`LungCap(cc)`)

> class(`Age( years)`)

> class(`Height(inches)`)

> class(Smoke)

> class(Gender)

> class(Caesarean)

> plot(`Age( years)`,`LungCap(cc)`, main = "Scatterplot")

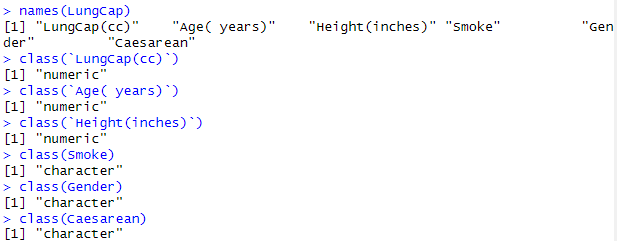
> cor(`Age( years)`, LungCap$`LungCap(cc)`)

> mod<-lm(LungCap$`LungCap(cc)`~ `Age( years)`)

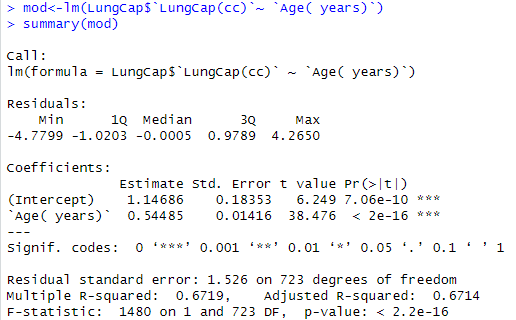
> summary(mod)

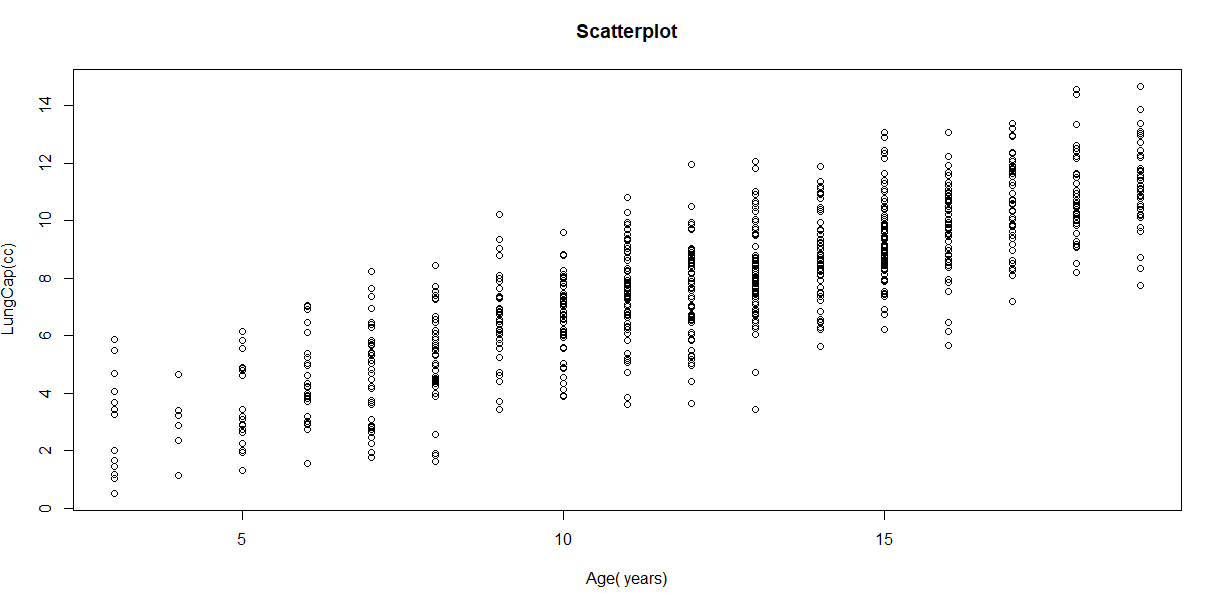
**Outputs:**

**https://lh7-us.googleusercontent.com/3eHhzi-kbxmnFA2olfd4OKfH5ncTUTzjEI2v9zhxxPcQf8wJMbhgwJBnIGGoddVmdJiN51MRwSPTHxP-3MBqacI-cVbnsCsxrsoNU7GW7idWA3ieHUlFCRIFaEk6imT1VAqti_YBw9VkgYYMMNDYpHI**

****

**https://lh7-us.googleusercontent.com/F9QmKWVjS2M8MqfAgccLMAYGt0LaZdVY7zu4m5k7lp2fK7ii_tSJwIFeePDXwhAjO7A8KpraFf3gyfyYcgcSyJ0GdUyIbzpVm9hDqOqHfQqA9z2X-p6WLkpiEMVdzeMKBvzLnMT2RGJyLJ5oKqWyo28**

****

****

**Practical 7**

**Aim: Logistic Regression and Decision Tree**

* **Build a logistic regression model to predict a binary outcome.**
* **Evaluate the model's performance using classification metrics (e.g., accuracy, precision, recall).**
* **Construct a decision tree model and interpret the decision rules for classification**

**Requirements: RStudio, Iris.csv**

**Steps: Download the “Iris.csv” from Kaggle**

**Open RStudio >  Create A New Project > Import Dataset > From Text(Base) > Browse > Import**

**In the Console, Type the following Command-**

> Iris <- read.csv("C:/Users/admin/Downloads/Iris.csv")

> View(Iris)

> library(datasets)

> ir\_data<-iris

> head(ir\_data)

> str(ir\_data)

> levels(ir\_data$Species)

> sum(is.na(ir\_data))

> ir\_data<-ir\_data[1:100,]

> set.seed(100)

> samp <- sample(1:100, 80)

> ir\_test<-ir\_data[samp,]

> ir\_ctrl<-ir\_data[samp,]

> install.packages("ggplot2")

> library(ggplot2)

> install.packages("GGally")

> library(GGally)

> ggpairs(ir\_test)

> y<-ir\_test$Species; x<-ir\_test$Sepal.Length

> glfit<-glm(y~x,family = 'binomial')

> summary(glfit)

> newdata<-data.frame(x=ir\_ctrl$Sepal.Length)

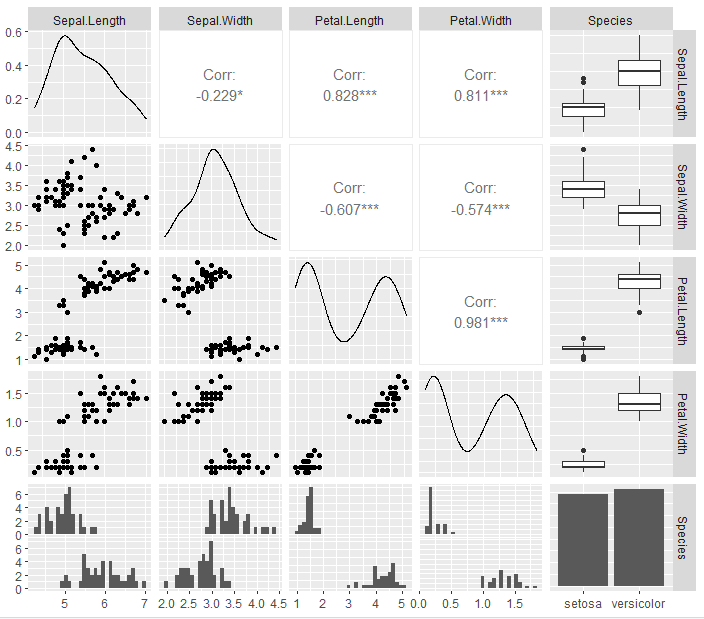
> predicted\_val<-predict(glfit,newdata, type = "response")

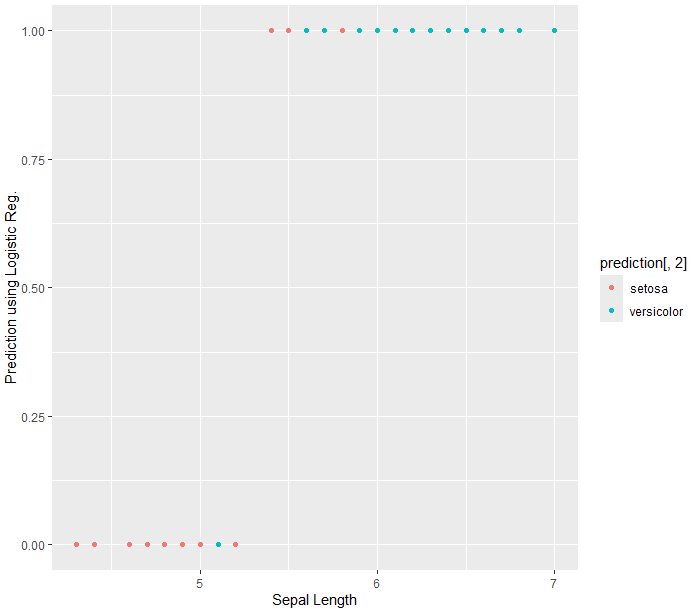
> prediction<-data.frame(ir\_ctrl$Sepal.Length,ir\_ctrl$Species,predicted\_val)

> prediction

> qplot(prediction[,1],round(prediction[,3]),col=prediction[,2],xlab='Sepal Length', ylab='Prediction using Logistic Reg.')

**Outputs:**





**Practical - 8**

**Aim: K-Means Clustering**

* **Apply the K-Means algorithm to group similar data points into clusters.**
* **Determine the optimal number of clusters using elbow method or silhouette analysis.**
* **Visualize the clustering results and analyze the cluster characteristics.**

**Requirements: RStudio, Iris.csv**

**Steps: Download the “WholesaleCustomerData.csv” from Kaggle**

**Open RStudio >  Create A New Project > Import Dataset > From Text(Base) > Browse > Import**

**In the Console, Type the following Command-**

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

data= pd.read\_csv('WholesaleCustomerData.csv')

data.head()

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

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

data[continuous\_features].describe()

for col in categorical\_features:

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

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

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

data.head()

mms = MinMaxScaler()

mms.fit(data)

data\_transformed = mms.transform(data)

from sklearn.cluster import KMeans

import numpy as np

Sum\_of\_squared\_distances = []

K = range(1, 15)

for k in K:

    km = KMeans(n\_clusters=k)

    km = km.fit(data\_transformed)

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

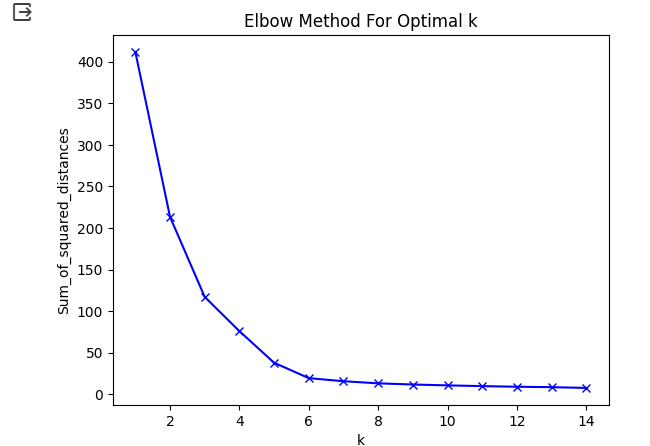
plt.plot(K, Sum\_of\_squared\_distances, 'bx-')

plt.xlabel('k')

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

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

plt.show()



**Practical - 9**

**Aim: Principal Component Analysis (PCA)**

* **Perform PCA on a dataset to reduce dimensionality.**
* **Evaluate the explained variance and select the appropriate number of principal components.**
* **Visualize the data in the reduced-dimensional space**

**Requirements: RStudio, Iris.csv**

**Steps: Download the “Iris.csv” from Kaggle**

**Open RStudio >  Create A New Project > Import Dataset > From Text(Base) > Browse > Import**

**In the Console, Type the following Command-**

> data\_iris <- iris[1:4]

> cov\_data <- cov(data\_iris)

> Eigen\_data <- eigen(cov\_data)

> PCA\_data <- princomp(data\_iris,cor = "False")

> Eigen\_data$values

> PCA\_data$sdev^2

> PCA\_data$loadings[,1:4]

> Eigen\_data$vectors

> summary(PCA\_data)

> biplot(PCA\_data)

> screeplot(PCA\_data,type = 'lines')

> model2 = PCA\_data$loadings[,1]

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

> library(class)

> install.packages("e1071")

> library(e1071)

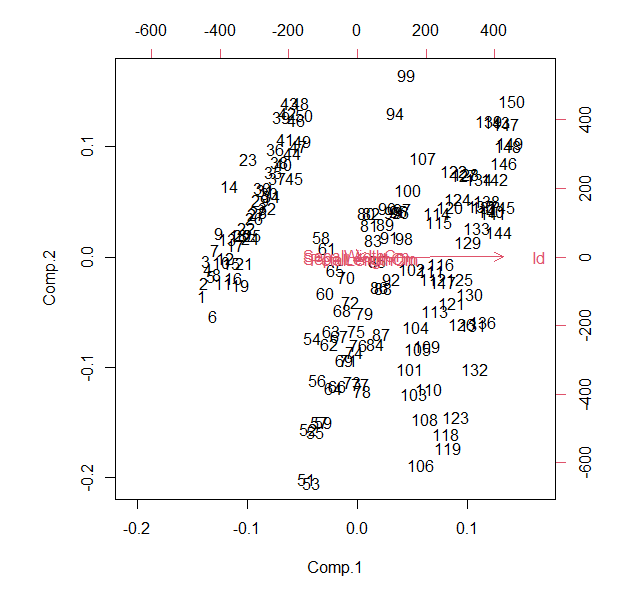
> mod1 <- naiveBayes(iris[,1:4],iris[,5])

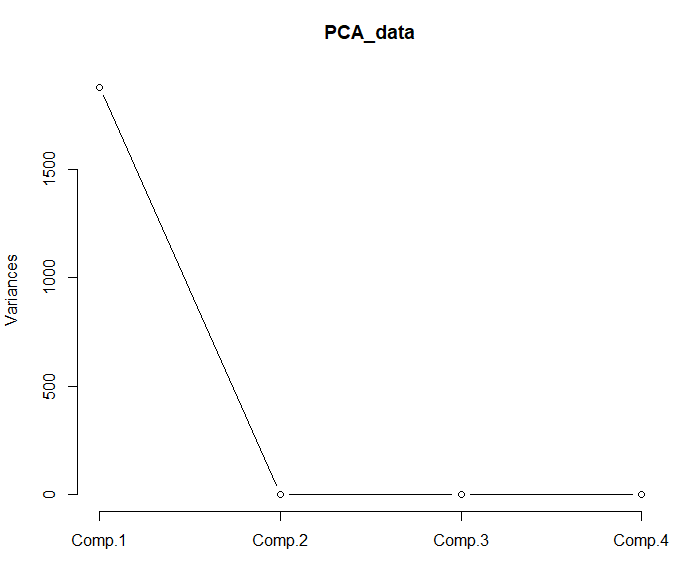
> mod2 <- naiveBayes(model2\_scores,iris[,5])

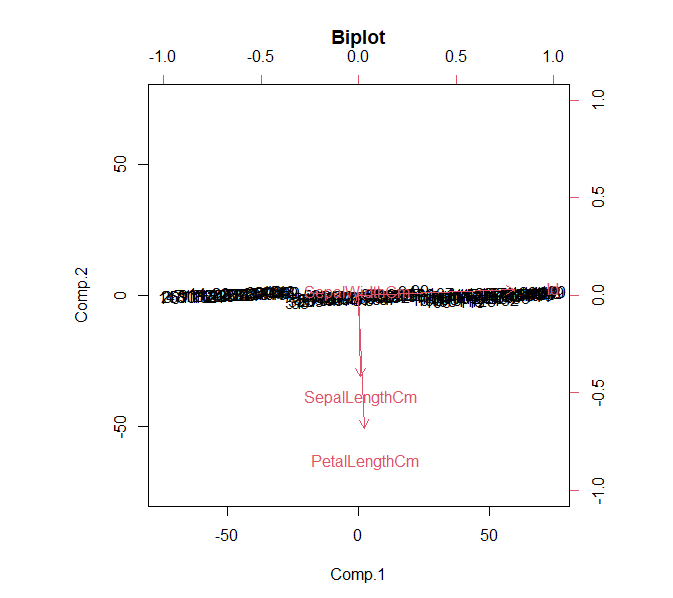
> table(predict(mod1,iris[,1:4]),iris[,5])

> table(predict(mod2,model2\_scores),iris[,5])

> biplot(PCA\_data, main = "Biplot", scale = 0)





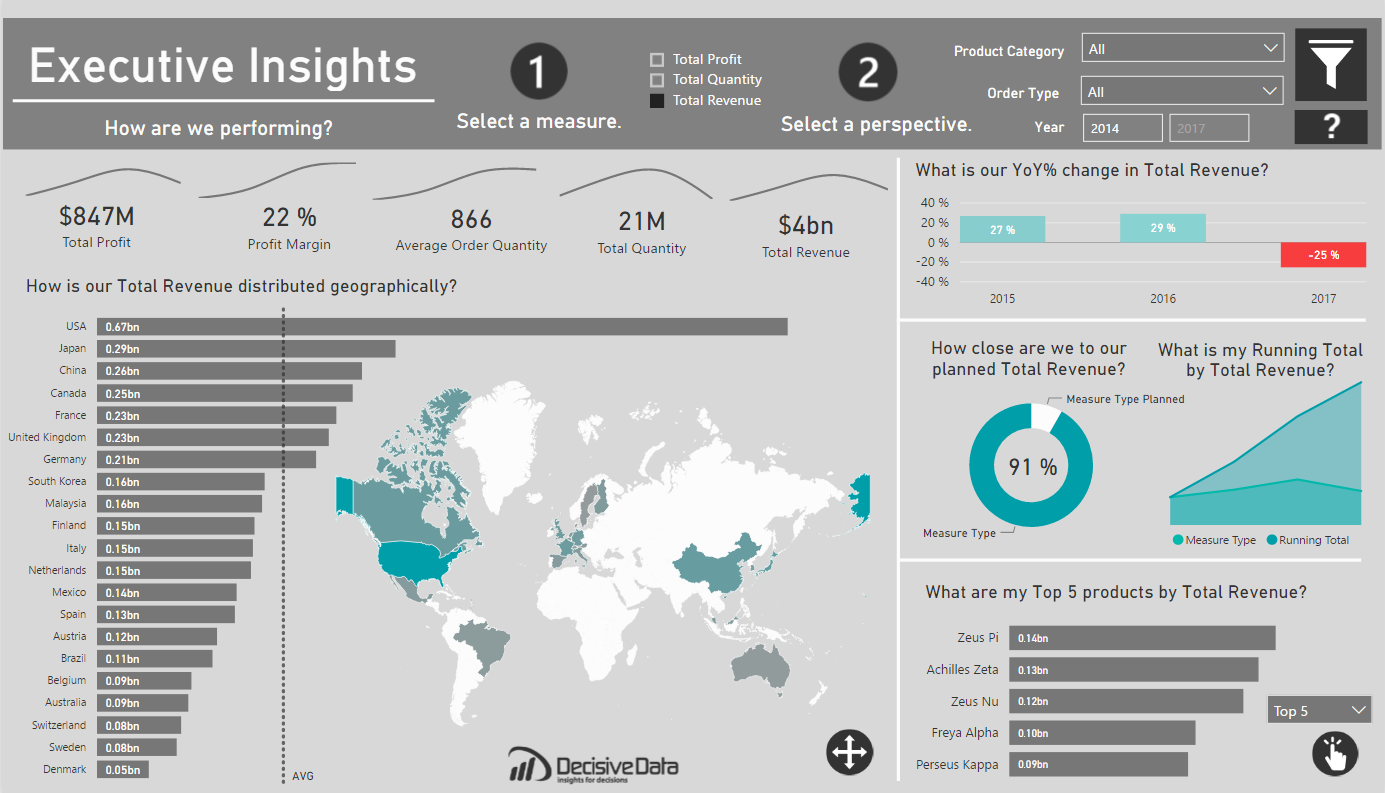


**Practical 10**

**Aim: Data Visualization and Storytelling**

* **Create meaningful visualizations using data visualization tools**
* **Combine multiple visualizations to tell a compelling data story.**
* **Present the findings and insights in a clear and concise manner**

**Executive Insights by Decisive Data**

****

**Summary**

Often asking, “How are we performing?” can be a question that cascades into a series of further questions, spinoffs and investigative research. This is especially true for globally minded companies. I wanted to create a report that preemptively addressed this kind of exploration. This report is meant to provide data-driven decision making, while emphasizing user-flexibility and visual analysis. I was able to achieve my goal to empower the user by leveraging dynamic visuals. Thus, this dashboard can scale as the needs of the global business changes.

**Approach**

A dashboard is most valuable when you immediately understand what you can do. I looked into what attributes that can be influenced by the company:

* *What we are selling (products)*
* *When we are selling (year)*
* *Where we are selling (country)*
* *How we are selling (order type)*

The focus here is to view the business from multiple angles for these attributes, providing a holistic approach to the business, through dynamic parameters. I was able to provide deeper analysis with Top/Bottom products, YoY growth and Running totals to name a few. I leveraged bar charts, line charts, donut charts and custom visuals for visual analysis.

**Pushing the Boundaries**

There were two things I wished to achieve with this dashboard, attractive mapping and next-level user interactivity.

I wanted to have a map that was design oriented and subtle, like an art piece, meant to invite the user into the dashboard. *If users are going to be using a report all the time, why not make it pleasing to the eye?* I found that design piece with the new “Shape Map” feature, where one can import custom TopoJSON files. I edited and imported a custom world map from <http://mapstarter.com/>. The simple map was meant to ground the user geographically and complement the adjacent bar chart, which holds the same information.

Moreover, I am a firm believer that interactivity is empowering. Giving users the tools to investigate data on their own terms is liberating. As such, I wanted to provide *two* layers of dynamic parameters. This was done with a Top 5/Bottom 5 measure that reflects whatever measure the user has selected at the top of the dashboard. This was executed within DAX. The question I wanted to be able to answer was “What is my [Top/Bottom] products by [Value]?” in one, simple, clean visual.

For the first part of this method of creating dynamic measures, I was inspired by Sam McKay’s wonderful blog post on [Dynamic Visuals](https://community.powerbi.com/t5/Community-Blog/Want-to-create-dynamic-visuals-that-make-your-analysis-really/ba-p/162282), within Power BI. This provided me the groundwork for my dynamic values, and how I would approach the rest of this problem.

Now came the real challenge. I had to build two ranks on a value that could change with a click. To do this I set my dynamic parameter as two rank fields, one Ascending and the other to Descending, for the respective ends of the product performance. These were calculated across my desired field, Products.

Next, I needed to filter each of these Ranks to only keep the Top 5 Products. This can be down with two IF statements, keeping the Rank with only five values or else appearing as BLANK(). Once the IFs are built, you can then build another selector table with McKay’s method with a “Top 5” and “Bottom 5”, which can eventually be used with a SWITCH function to include only the [Filtered Rank DESC] or the [Filtered Rank ASC].

The result means that instead of **six** individual bar charts, one for a Top or Bottom, across three measures, Quantity, Revenue and Profit, you end up with **one**, dynamic visual. The result empowers users and saves valuable real estate for other interesting insights.

Quality over quantity.

**Wrapping it up**

The dynamic parameters and tables worked very well, but it took a little research to have the two dynamic parameters work in conjunction, as I had yet to see any specific topic utilize such before in Power BI. Another aspect that comes to mind, is whether one could make the number of values, in this case 5, also a parameter, so you could select how many TopN you are seeing. Food for thought. In the end, it’s great to know that great visuals can be created just with DAX, determination and some trial and error.

Overall, with the combination of the custom visuals, and dynamic parameters, I created a clean dashboard for analytical insights that’s also a pleasure to interact with.