Q. Develop a MapReduce program to calculate the frequency of a given word in a given file.

Wordcount.py

```
import re
from multiprocessing import Pool

WORD_RE = re.compile(r"[\w']+")

def read_file(filename):
    with open(filename, 'r') as file:
        return file.readlines()

def mapper(line):
    word_count = {}
    for word in WORD_RE.findall(line):
        word_count[word.lower()] = word_count.get(word.lower(), 0) + 1
    return word_count

def reducer(mapped_counts):
    reduced_counts = {}
    for word_count in mapped_counts:
        for word, count in word_count.items():
            reduced_counts[word] = reduced_counts.get(word, 0) + count
    print(reduced_counts)
    return reduced_counts

def main(filename, target_word):
    lines = read_file(filename)
    with Pool() as pool:
        mapped_counts = pool.map(mapper, lines)
    reduced_counts = reducer(mapped_counts)

    # Get the frequency of the target word
    target_frequency = reduced_counts.get(target_word.lower(), 0)
```

```
    print(f"The frequency of '{target_word}' in the file is: {target_frequency}")


if __name__ == "__main__":
    filename = input("Enter the file name: ")
    target_word = input("Enter the word to find frequency: ")
    main(filename, target_word)
```

samplefile.txt

The quick brown fox jumps over the lazy dog. The lazy dog yawns and stretches. The fox looks back and smiles at the dog. Then, the fox continues its journey through the forest. The quick brown fox is a clever animal. It knows how to survive in the wild. The lazy dog, on the other hand, prefers to relax and enjoy life. Life is simple for the lazy dog. The quick brown fox and the lazy dog are good friends. They often play together in the meadow. Sometimes, they chase each other around the trees. Other times, they simply lie down and bask in the sun. But no matter what they do, they always have fun together.

Output:

```
PS D:\Learning only> & C:/ProgramData/Python310/python.exe "d:/Learning only/CL4/practical2.py"
Enter the file name: CL4\practical2.txt
Enter the word to find frequency: the
{'the': 17, 'quick': 3, 'brown': 3, 'fox': 5, 'jumps': 1, 'over': 1, 'lazy': 5, 'dog': 6, 'yawns': 1, 'and': 5, 'stretches': 1, 'looks': 1, 'back': 1, 'smile
s': 1, 'at': 1, 'then': 1, 'continues': 1, 'its': 1, 'journey': 1, 'through': 1, 'forest': 1, 'is': 2, 'a': 1, 'clever': 1, 'animal': 1, 'it': 1, 'knows': 1,
 'how': 1, 'to': 2, 'survive': 1, 'in': 3, 'wild': 1, 'on': 1, 'other': 3, 'hand': 1, 'prefers': 1, 'relax': 1, 'enjoy': 1, 'life': 2, 'simple': 1, 'for': 1,
 'are': 1, 'good': 1, 'friends': 1, 'they': 5, 'often': 1, 'play': 1, 'together': 2, 'meadow': 1, 'sometimes': 1, 'chase': 1, 'each': 1, 'around': 1, 'trees'
: 1, 'times': 1, 'simply': 1, 'lie': 1, 'down': 1, 'bask': 1, 'sun': 1, 'but': 1, 'no': 1, 'matter': 1, 'what': 1, 'do': 1, 'always': 1, 'have': 1, 'fun': 1}
The frequency of 'the' in the file is: 17
```

```
import multiprocessing

def matrix_multiply_mapper(row, col):
    result = 0
    for i in range(len(row)):
        result += row[i] * col[i]
    return result

def matrix_multiply_worker(args):
    row_index, row, columns = args
    return [(row_index, col_index, matrix_multiply_mapper(row, col))
            for col_index, col in enumerate(columns)]

def matrix_multiply_reduce(results):
    final_result = {}
    for row_index, col_index, value in results:
        if row_index not in final_result:
            final_result[row_index] = {}
        final_result[row_index][col_index] = value

    return final_result

def map_reduce_matrix_multiply(matrix1, matrix2):
    num_workers = multiprocessing.cpu_count()
    pool = multiprocessing.Pool(processes=num_workers)

    args = [(i, matrix1[i], matrix2) for i in range(len(matrix1))]
    intermediate_results = pool.map(matrix_multiply_worker, args)

    pool.close()
    pool.join()

    final_result = matrix_multiply_reduce(
        [item for sublist in intermediate_results for item in sublist])

    return final_result

if __name__ == "__main__":
    matrix1 = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]

    matrix2 = [
        [9, 8, 7],
        [6, 5, 4],
        [3, 2, 1]
    ]

    result = map_reduce_matrix_multiply(matrix1, matrix2)
    for row_index, row in result.items():
        print(row)
```

```
{0: 46, 1: 28, 2: 10}
{0: 118, 1: 73, 2: 28}
{0: 190, 1: 118, 2: 46}
```

Q. Develop a MapReduce program to find the grades of students

GradeCalculator.py

```
from mrjob.job import MRJob

class GradeCalculator(MRJob):
    def mapper(self, _, line):
        # Split the input line into name and score
        name, score = line.split(',')
        score = int(score)

        # Emit the name and score
        yield name, score

    def reducer(self, key, values):
        # Calculate the average score
        total_score = 0
        num_scores = 0
        for score in values:
            total_score += score
            num_scores += 1
        average_score = total_score / num_scores

        # Determine the grade based on the average score
        if average_score >= 90:
            grade = 'A'
        elif average_score >= 80:
            grade = 'B'
        elif average_score >= 70:
            grade = 'C'
        elif average_score >= 60:
            grade = 'D'
        else:
            grade = 'F'
```

```
        # Emit the name and grade
        yield key, grade

if __name__ == '__main__':
    GradeCalculator.run()
```

samplefile.txt

Prathamesh,95

Rohit,75

Virat,82

Sachin,68

Dhoni,88

Ashwin,73

Kuldeep,91

David,55

jadeja,50

Rahul,60

Iyer,45

Chahal,40

Output:

```
D:\Learning only\CL4>python practical4_1.py practical4.txt
No configs found; falling back on auto-configuration
No configs specified for inline runner
Creating temp directory C:\Users\PRATHA~1\AppData\Local\Temp\practical4_1.Prathamesh Patil.20240414.100510.266307
Running step 1 of 1...
job output is in C:\Users\PRATHA~1\AppData\Local\Temp\practical4_1.Prathamesh Patil.20240414.100510.266307\output
Streaming final output from C:\Users\PRATHA~1\AppData\Local\Temp\practical4_1.Prathamesh Patil.20240414.100510.266307\o
tput...
"Ashwin"        "C"
"Chahal"        "F"
"David" "F"
"Dhoni" "B"
"Iyer"  "F"
"Kuldeep"       "A"
"Prathamesh"    "A"
"Rahul" "D"
"Rohit" "C"
"Sachin"        "D"
"Virat" "B"
"jadeja"        "F"
Removing temp directory C:\Users\PRATHA~1\AppData\Local\Temp\practical4_1.Prathamesh Patil.20240414.100510.266307...
```

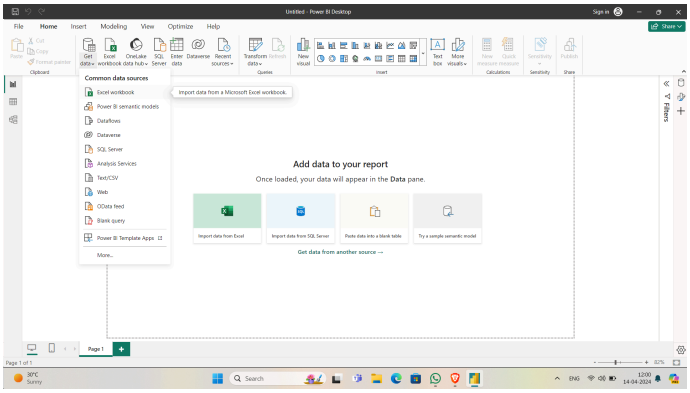
Code:

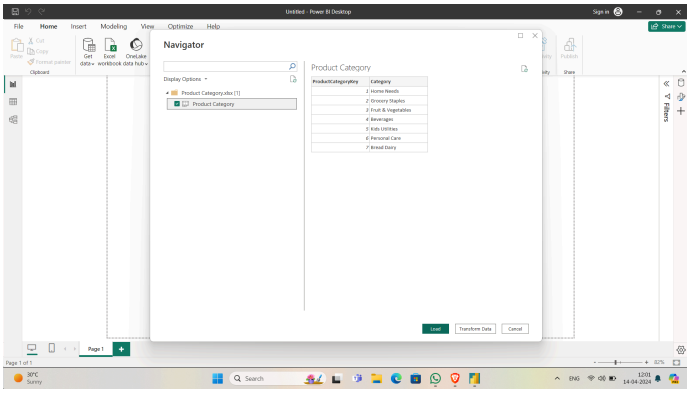
```
import pandas as pd
def map_reduce_with_pandas(input_file):
 # Load the dataset
 df = pd.read_csv(input_file)

 # Map: Filter deceased males and transform data for average age
calculation
 deceased_males = df[(df['Survived'] == 0) & (df['Sex'] == 'male')]

 # Reduce: Calculate average age of deceased males
 average_age_deceased_males = deceased_males['Age'].mean()

 # Map: Filter deceased females and transform data for count by class
 deceased_females_by_class = df[(df['Survived'] == 0) & (df['Sex'] ==
'female')]
 # Reduce: Count deceased females by class
 count_deceased_females_by_class =
deceased_females_by_class['Pclass'].value_counts()
 return average_age_deceased_males, count_deceased_females_by_class

# Example usage
input_file = r'D:\BE SEM VIII\CL_IV_Code\titanic.csv' # Update this
to the path of your Titanic dataset CSV file
average_age, female_class_count = map_reduce_with_pandas(input_file)
print(f"Average age of males who died: {average_age:.2f}")
print("Number of deceased females in each class:")
print(female_class_count)
```

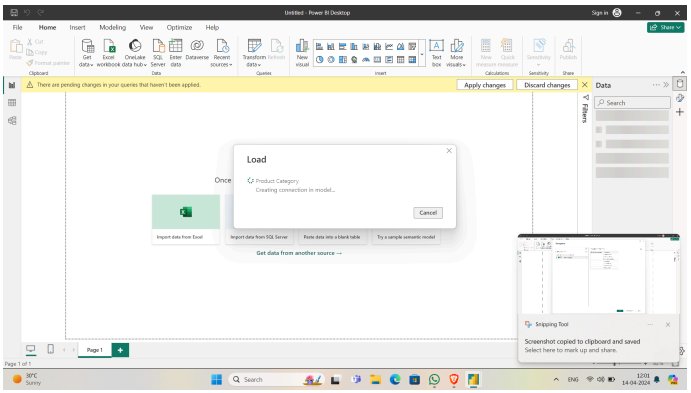
Output:

```
Average age of males who died: 31.62
Number of deceased females in each class:
Pclass
3    72
2     6
1     3
Name: count, dtype: int64
```

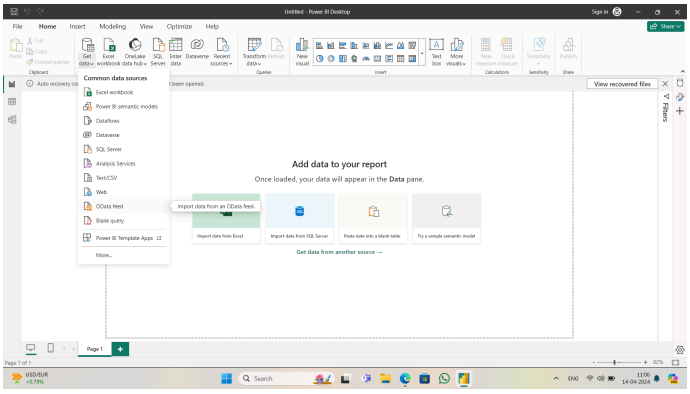
Importing Data from Excel:

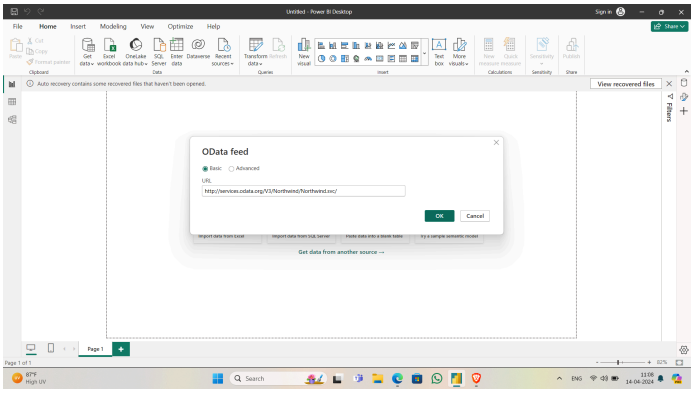


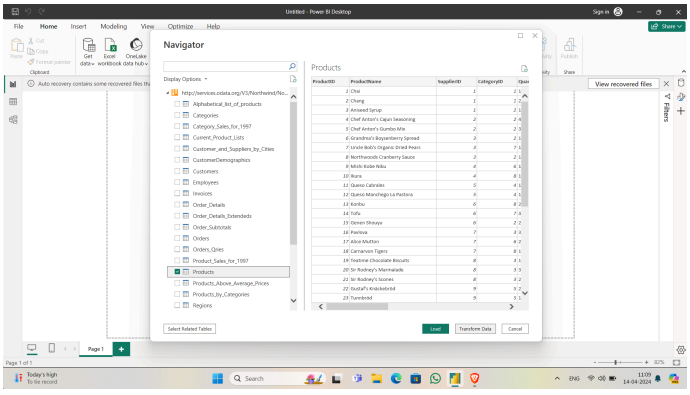


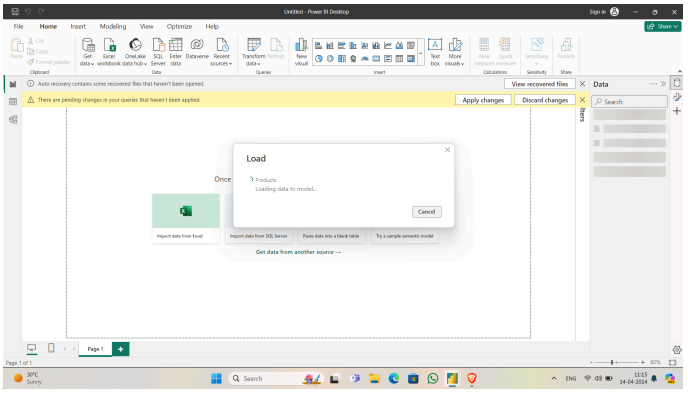


Importing Data from SQL server:

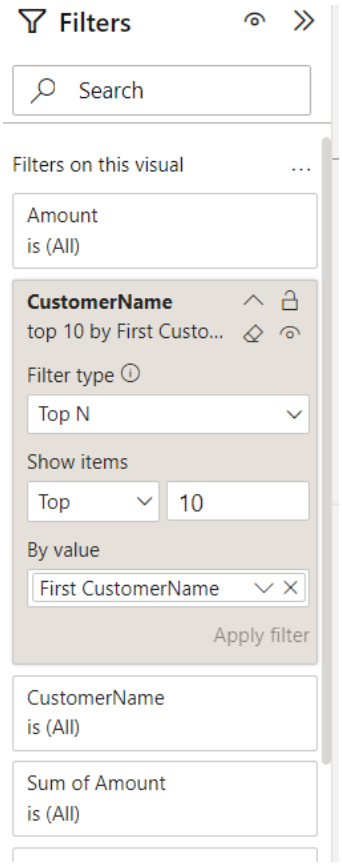


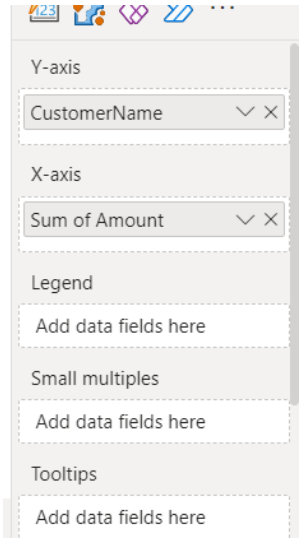


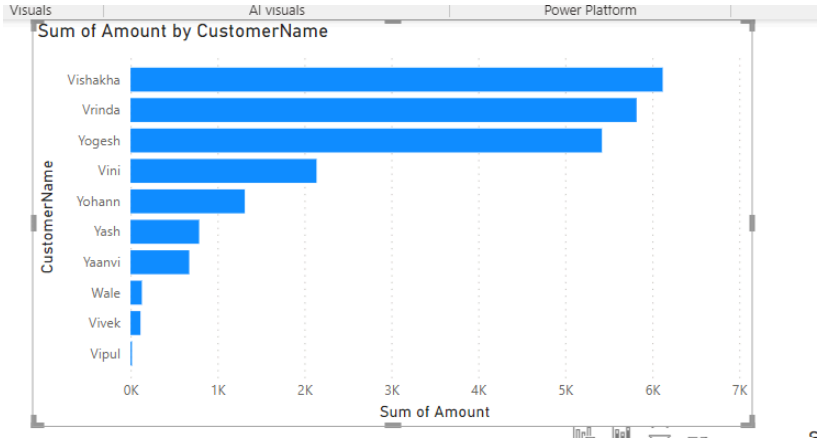




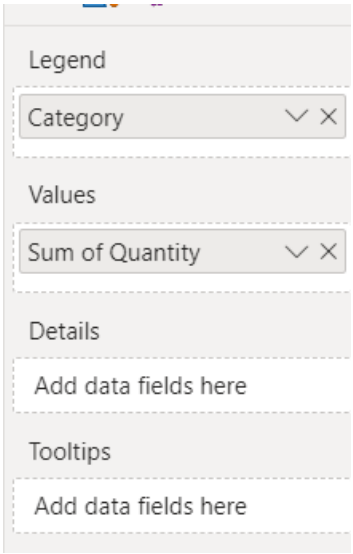
Creating Bar Chart:

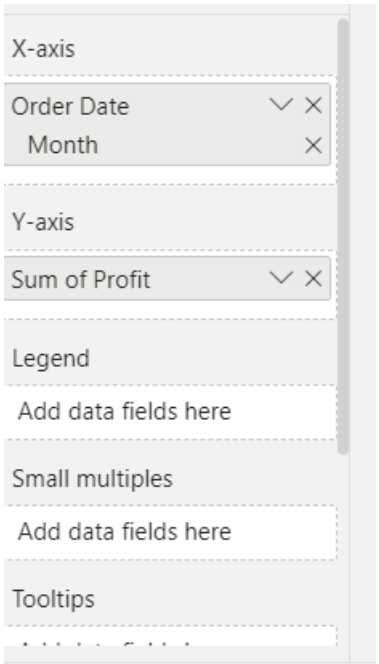


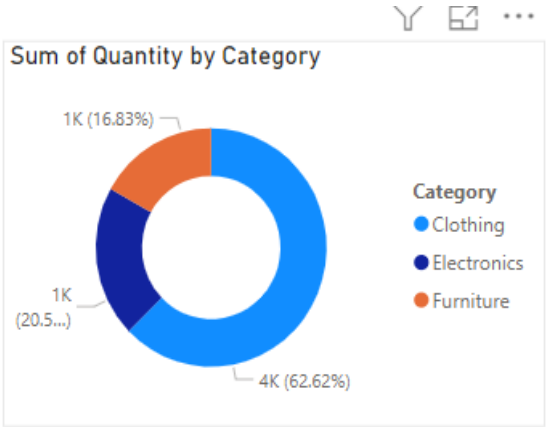




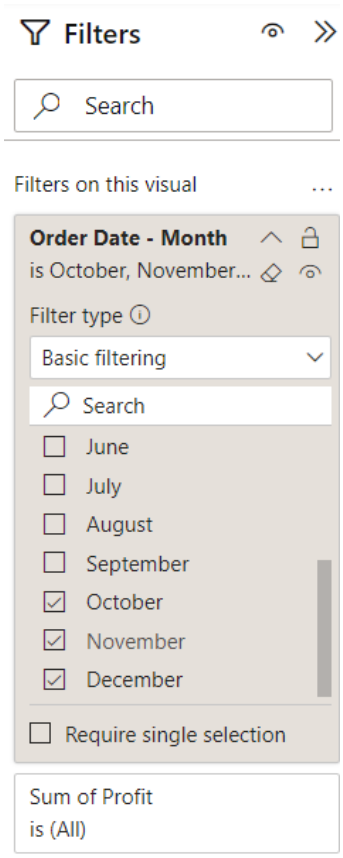
Creating Donut Chart:

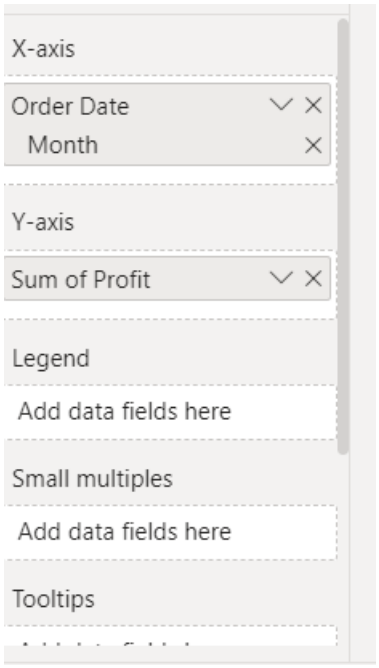


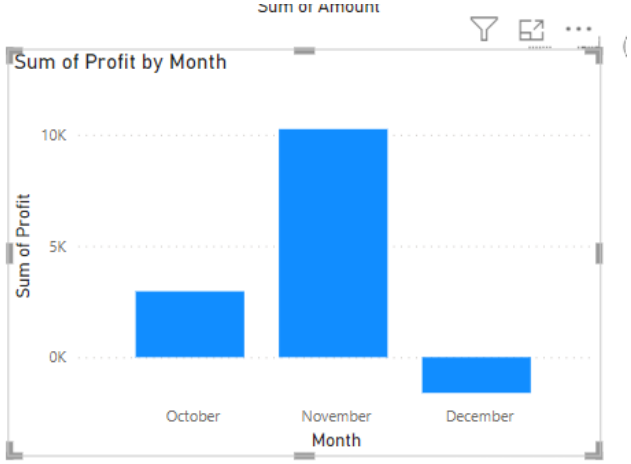




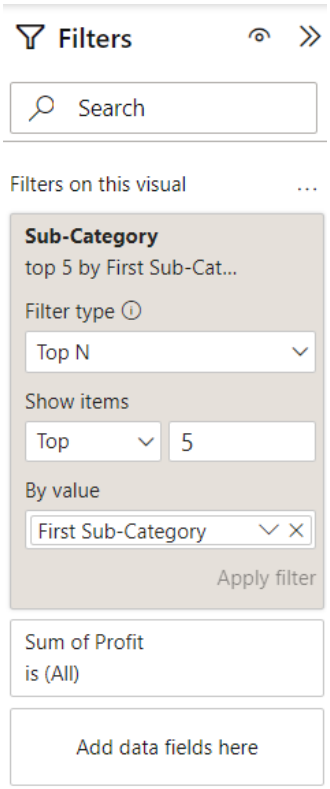
Creating Histogram:

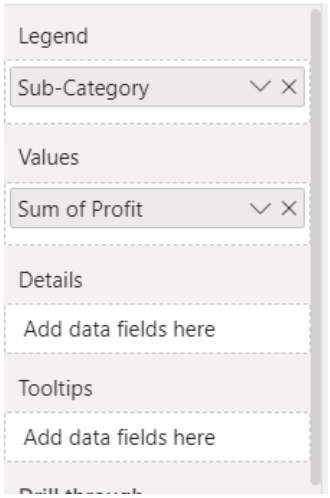


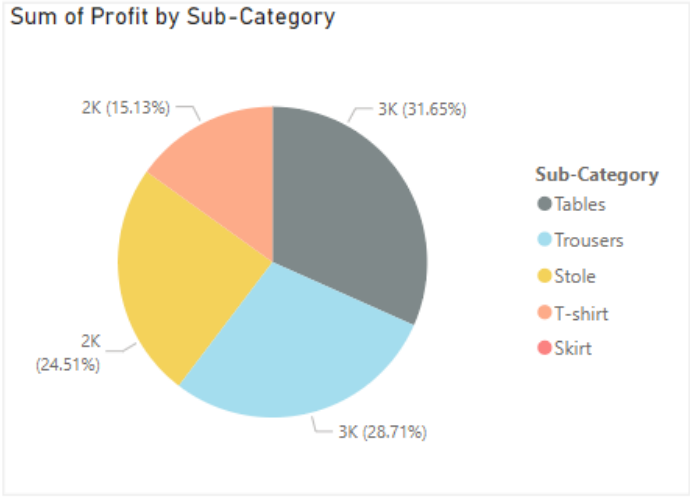




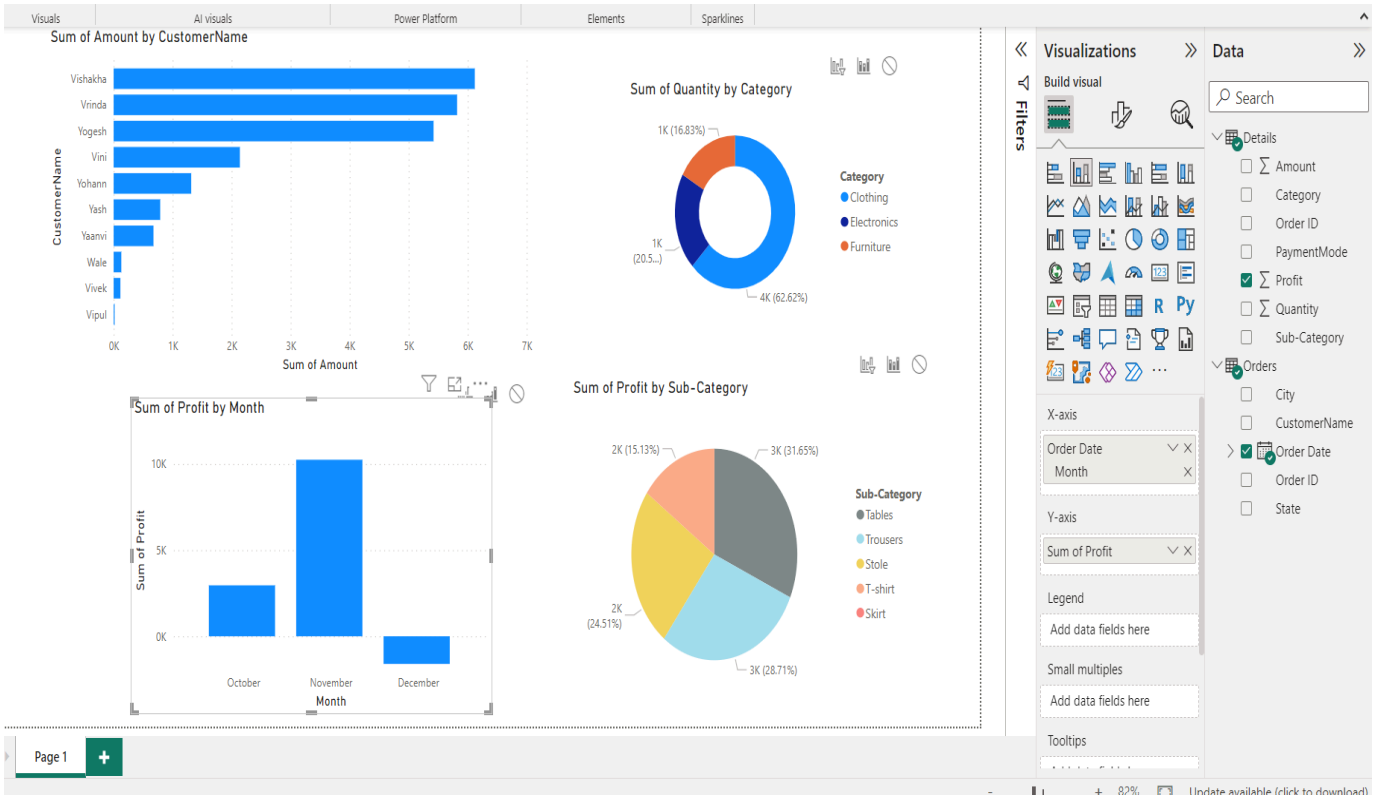
Creating Pie Chart:



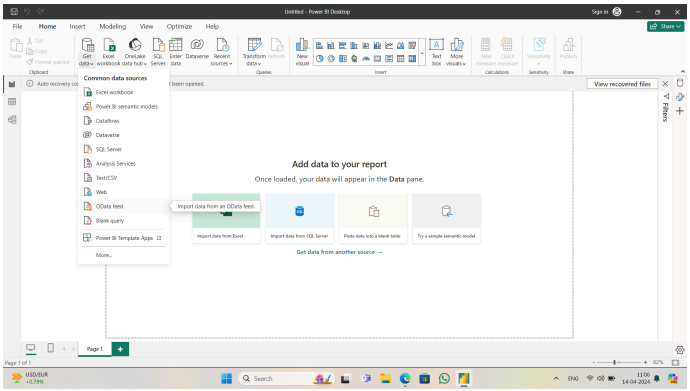


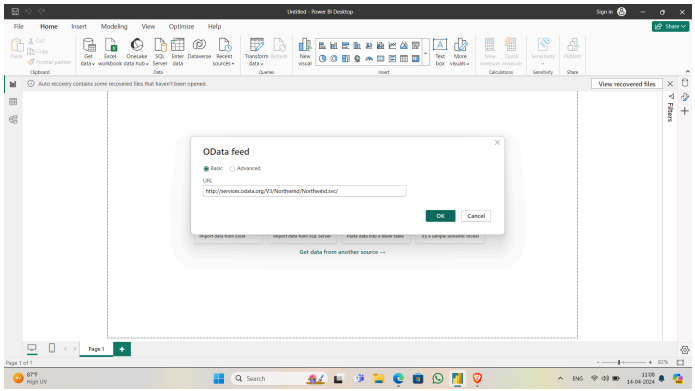


Dashboard:

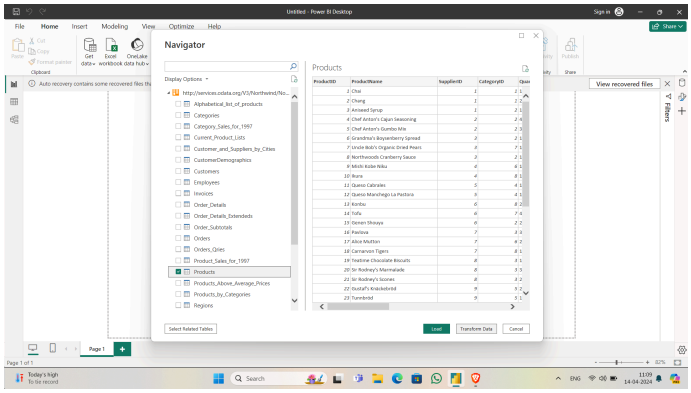

Visuals
AI visuals
Power Platform
Elements
Sparklines
Sum of Amount by CustomerName
CustomerName
Vishakha
Vrinda
Yogesh
Vini
Yohann
Yash
Yaanvi
Wale
Vivek
Vipul
0K
1K
2K
3K
4K
5K
6K
7K
Sum of Amount
Sum of Quantity by Category
1K (16.83%)
1K (20.5...)
4K (62.62%)
Category
Clothing
Electronics
Furniture
Sum of Profit by Month
Sum of Profit
10K
5K
0K
October
November
December
Month
Sum of Profit by Sub-Category
2K (15.13%)
3K (31.65%)
2K (24.51%)
3K (28.71%)
Sub-Category
Tables
Trousers
Stole
T-shirt
Skirt
Page 1
Visualizations
Build visual
Filters
X-axis
Order Date
Month
Y-axis
Sum of Profit
Legend
Add data fields here
Small multiples
Add data fields here
Tooltips
Data
Search
Details
Amount
Category
Order ID
PaymentMode
Profit
Quantity
Sub-Category
Orders
City
CustomerName
Order Date
Order ID
State
82%
Update available (click to download)

Extracting data from source:

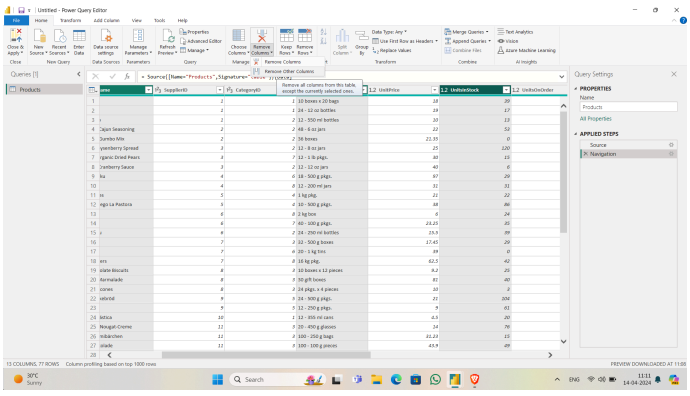


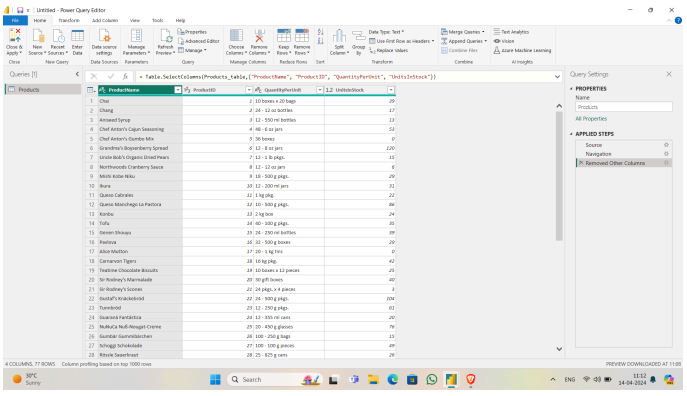


Transforming data:

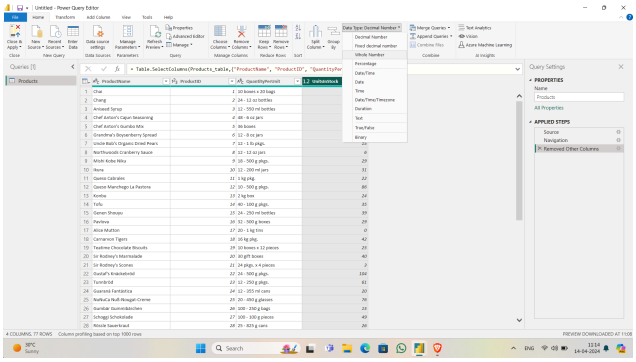


a) Removing unwanted columns

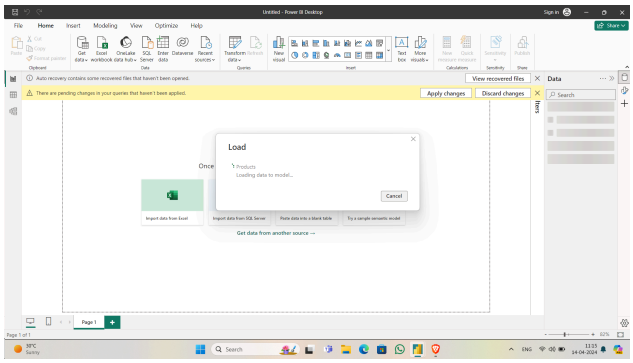




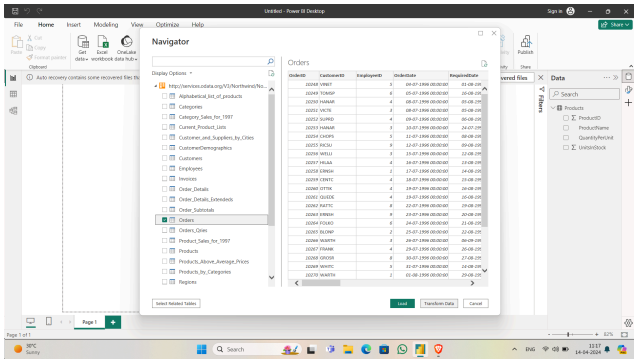
b) Changing Data Type

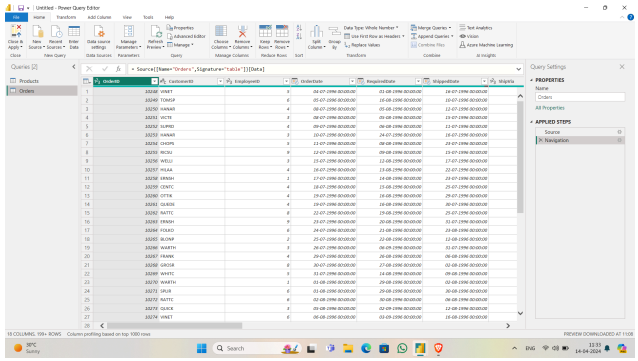


Loading the Data:



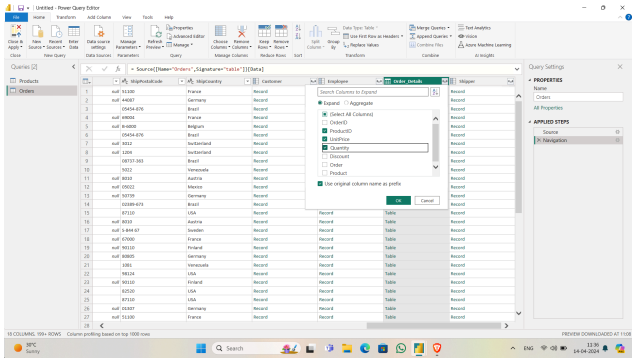
Extracting orders data:

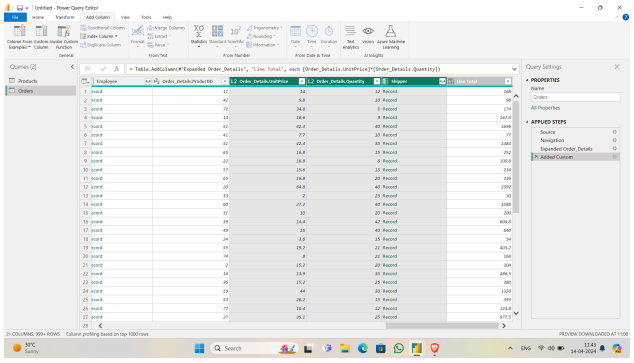




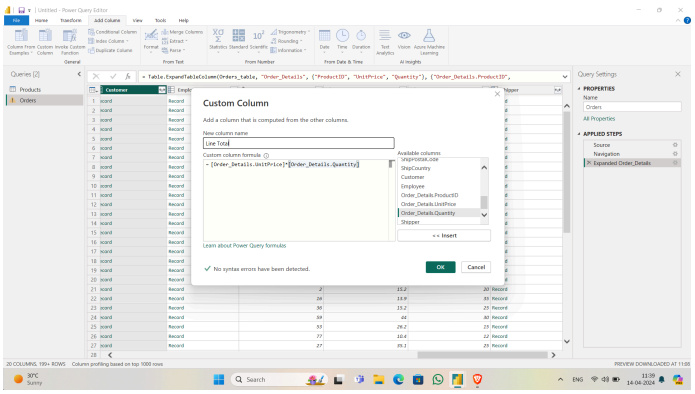
Transforming Orders Data:

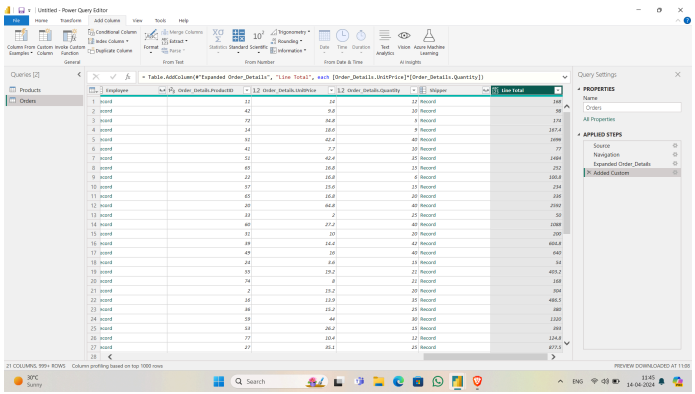
a) Expanding OrderDetails Column:



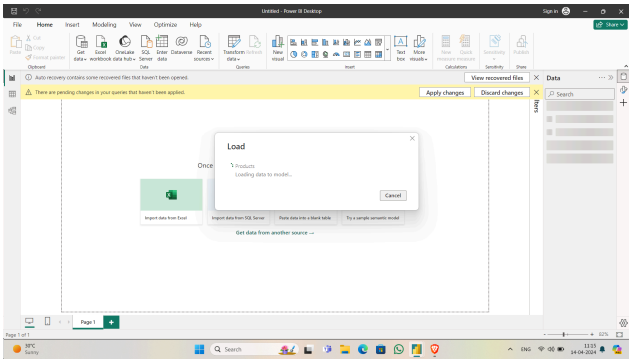


b) Creating a new Custom Columns:





Loading Data:



```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report


# Load the iris dataset
iris = load_iris()


# Features
X = iris.data


# Target variable
y = iris.target


# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Initialize and train the logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
# Predictions
y_pred_train = logreg.predict(X_train_scaled)
y_pred_test = logreg.predict(X_test_scaled)


# Model evaluation
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)


print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

```
Training Accuracy: 0.9666666666666667
Testing Accuracy: 1.0
```

```
# Additional evaluation metrics
print("Classification Report on Test Data:")
print(classification_report(y_test, y_pred_test))
```

```
Classification Report on Test Data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt


# Load the iris dataset
iris = load_iris()


# Features
X = iris.data


# Initialize KMeans with the number of clusters (3 for the iris dataset)
kmeans = KMeans(n_clusters=3)


# Fit KMeans to the data
kmeans.fit(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
```

KMeans

KMeans(n_clusters=3)

```
# Get the cluster centroids and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_


# Visualizing the clusters (assuming 2D data for simplicity)
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=200, c='red')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```