Assignment 2:

```
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
```

Assignment 3:

```
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}
```

Assignment 4:

```
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()
```

Student grades:

```
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...
```

Assignment 5:

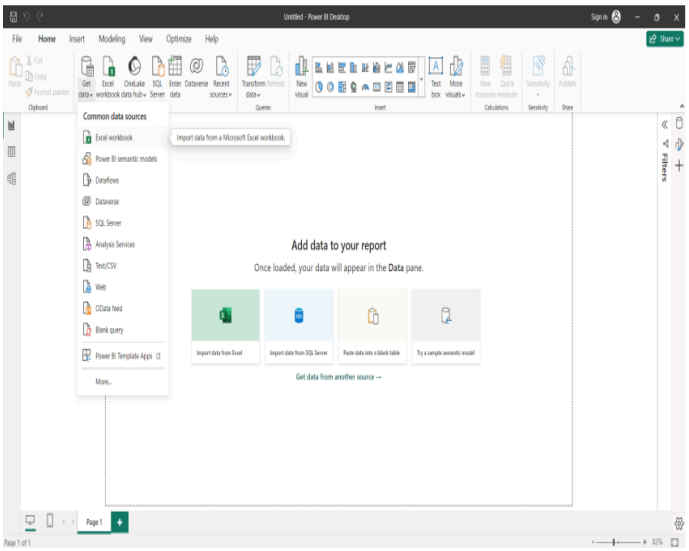
```
import pandas as pd
def map_reduce_with_pandas(input_file):
 # Load the dataset
 df = pd.read_csv(input_file)

 # 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)
```

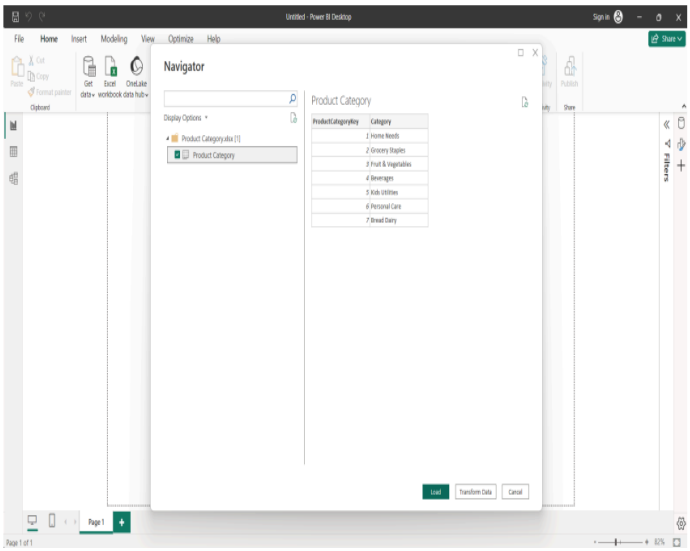
OUTPUT:

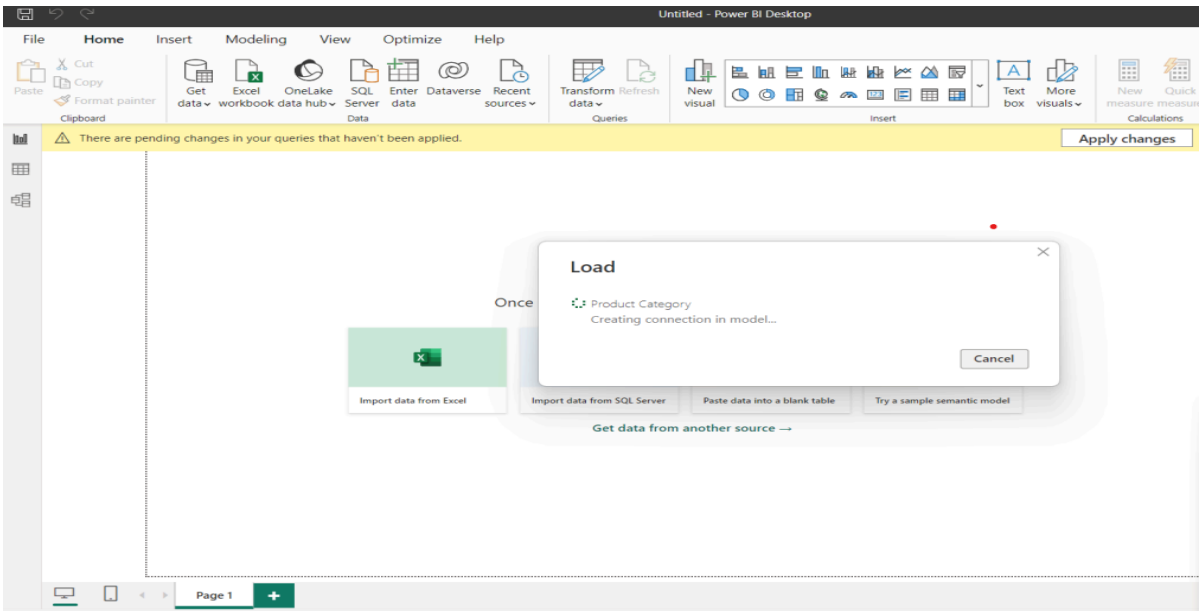
```
Average age of males who died: 31.62
Number of deceased females in each class:
Pclass
3    72
2     6
1     3
Name: count, dtype: int64
```

Assignment 6:

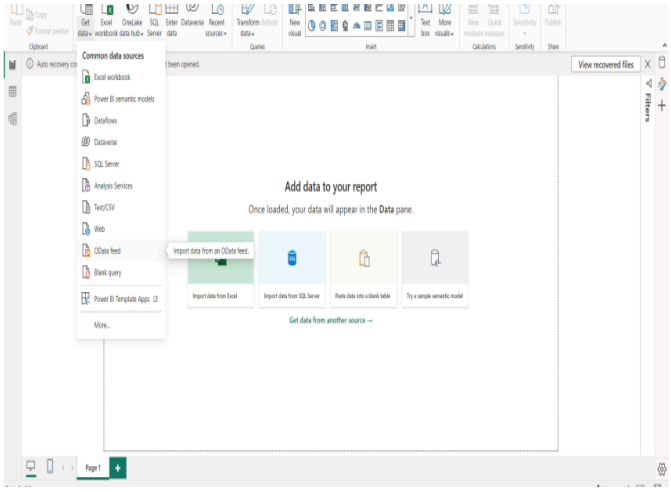
Importing Data from Excel:

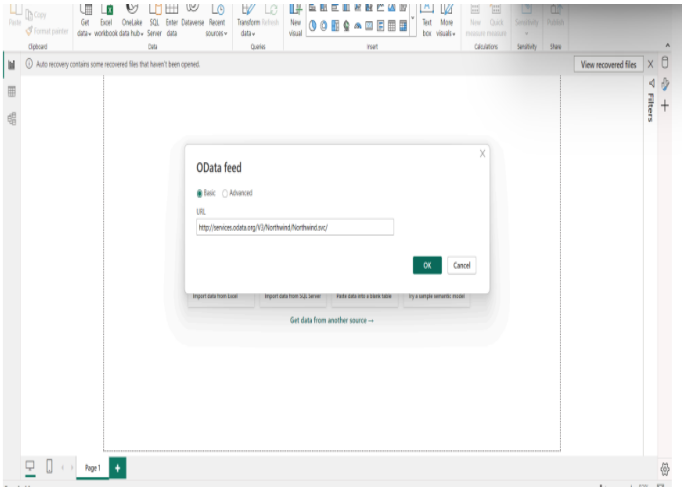


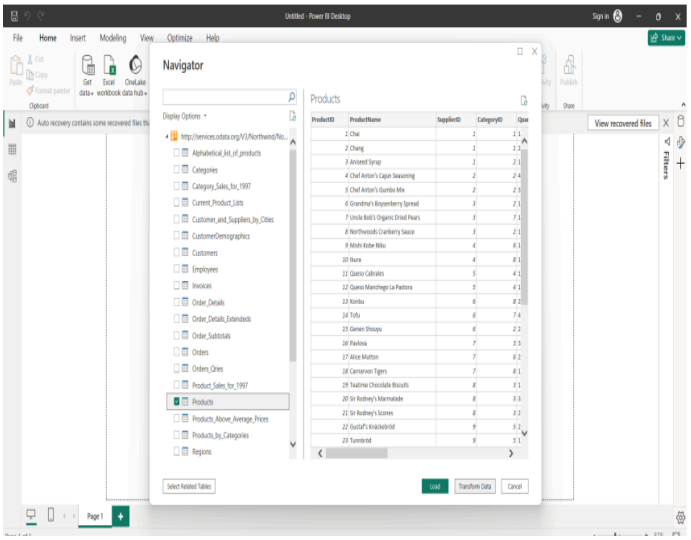


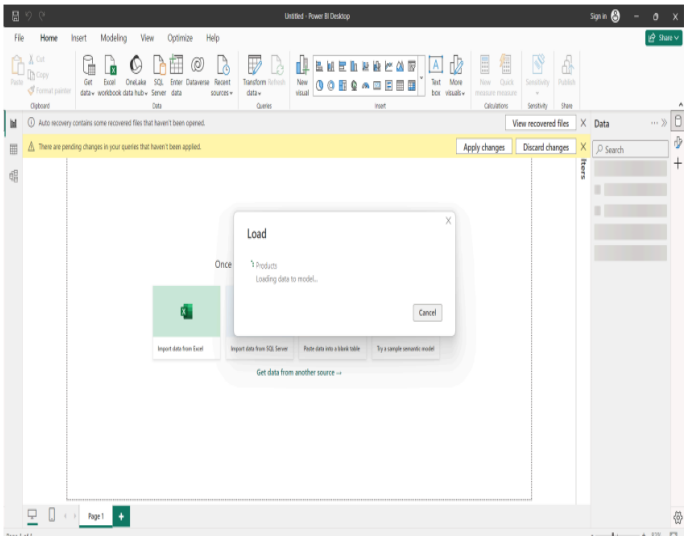


Importing Data from SQL server:



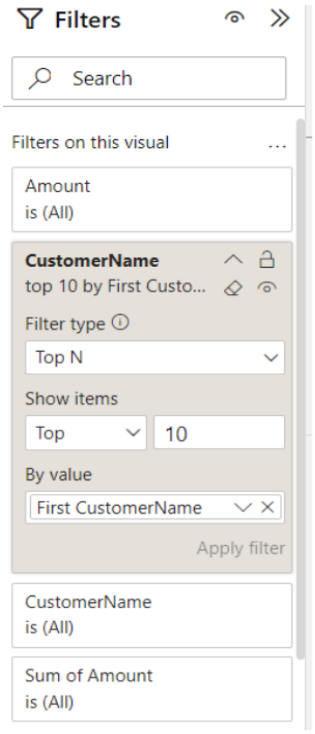


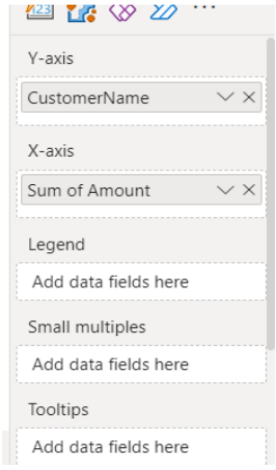


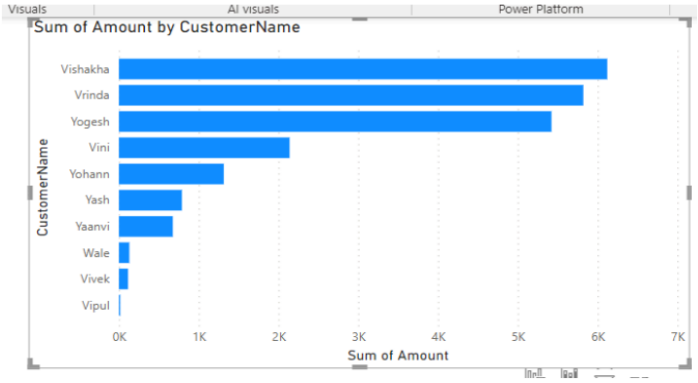


Assignment 7:

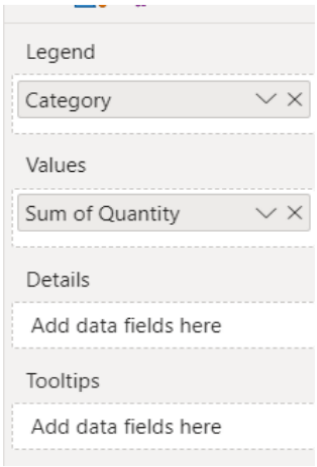
Creating Bar Chart:

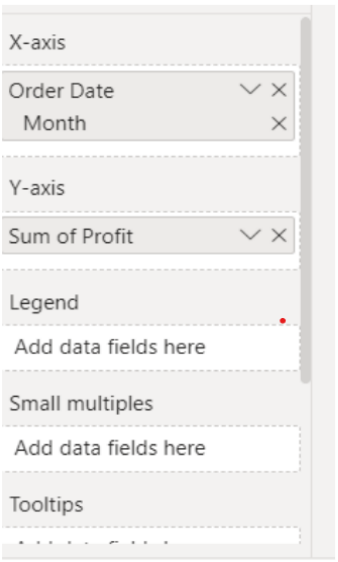


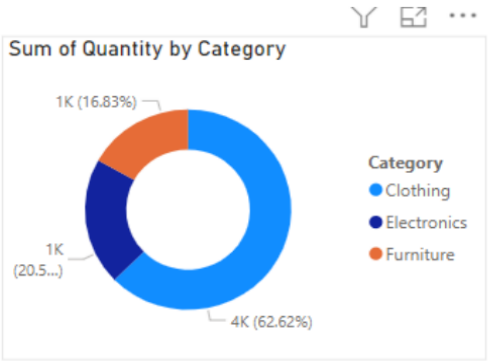




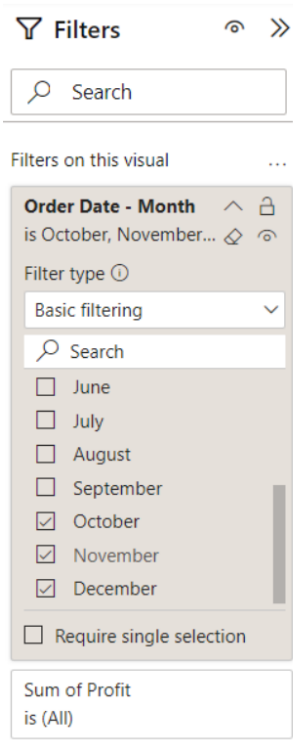
Creating Donut Chart:

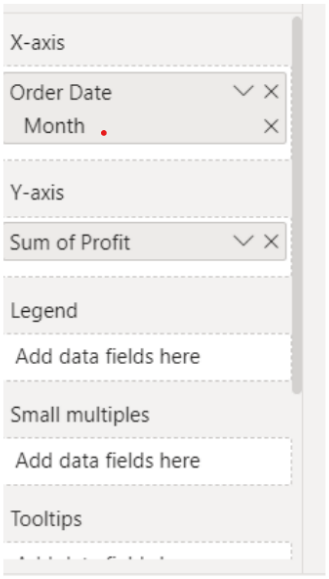


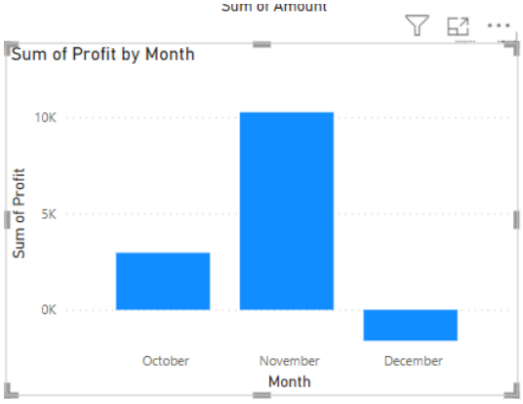




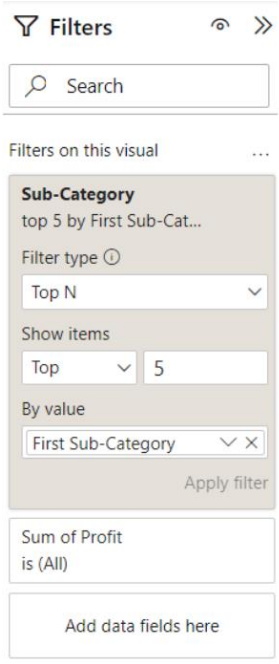
Creating Histogram:

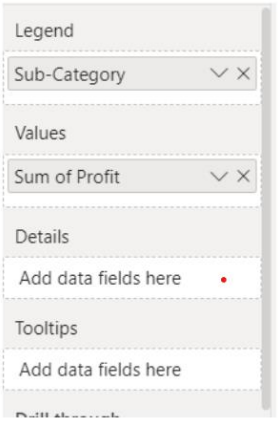


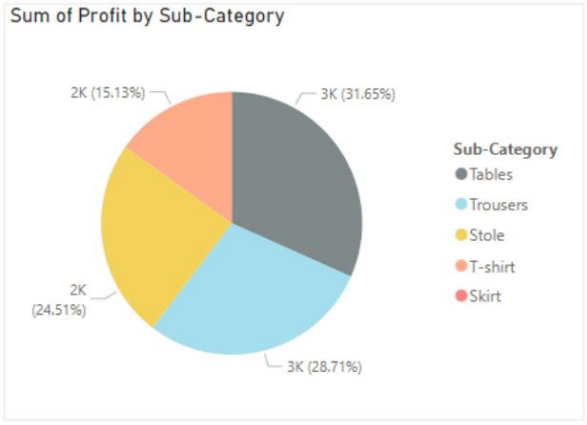




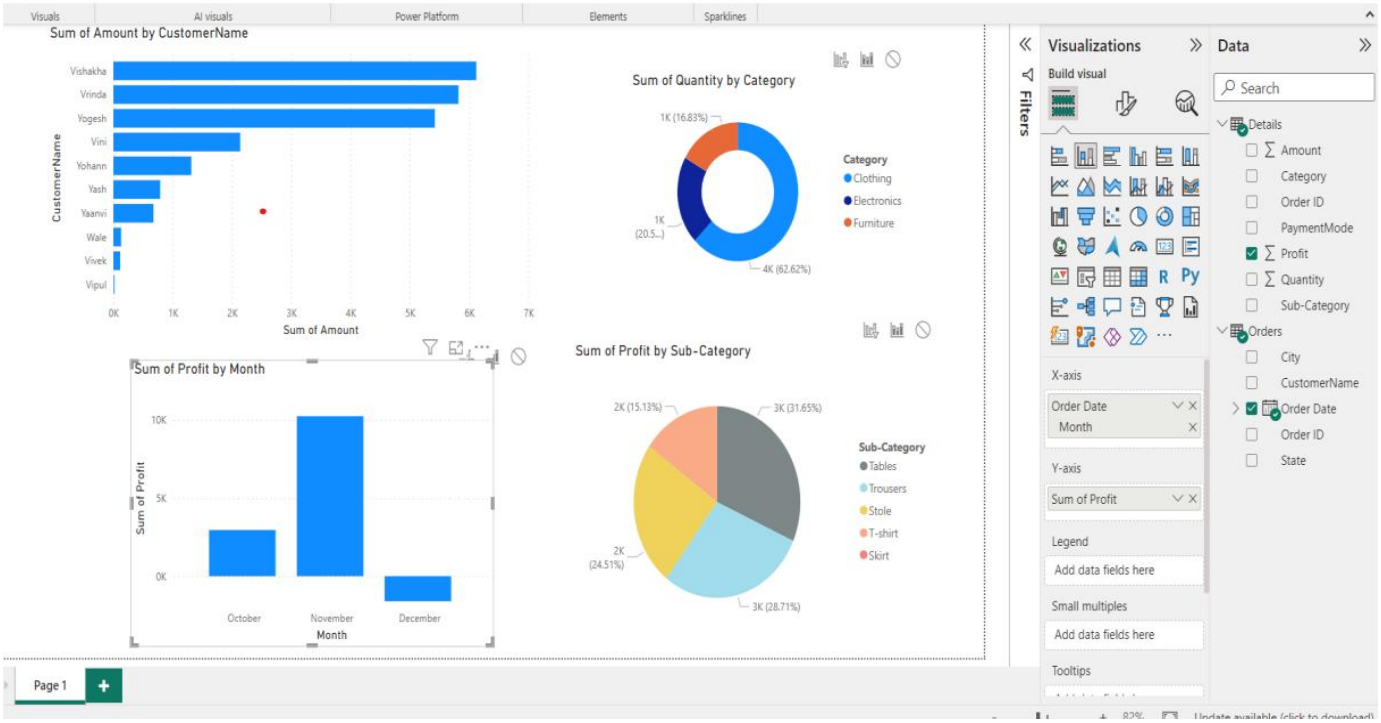
Creating Pie Chart:





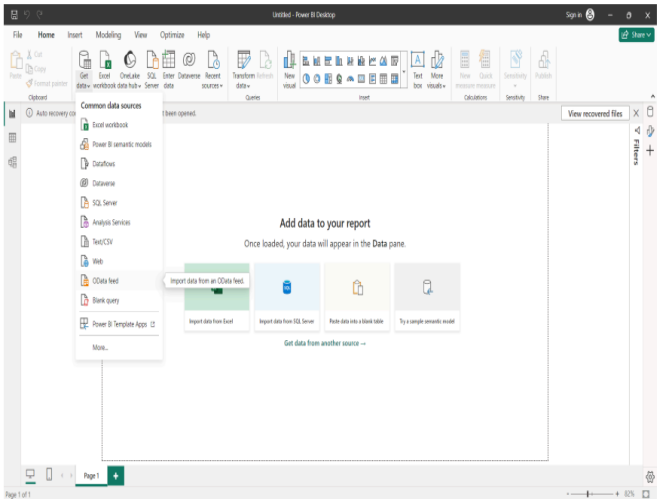


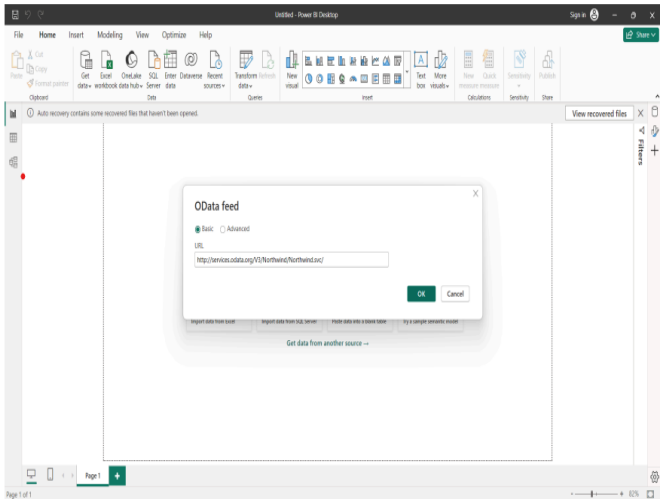
Dashboard:


Visuals
AI visuals
Power Platform
Elements
Sparklines
Sum of Amount by CustomerName
Vishakha
Vrinda
Yogesh
Vini
Yohann
Yash
Yaanvi
Wale
Vivek
Vipul
CustomerName
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
10K
5K
0K
Sum of Profit
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
Filters
Visualizations
Build visual
Data
Search
Details
Amount
Category
Order ID
PaymentMode
Profit
Quantity
Sub-Category
Orders
City
CustomerName
Order Date
Order ID
State
X-axis
Order Date
Month
Y-axis
Sum of Profit
Legend
Add data fields here
Small multiples
Add data fields here
Tooltips
Page 1

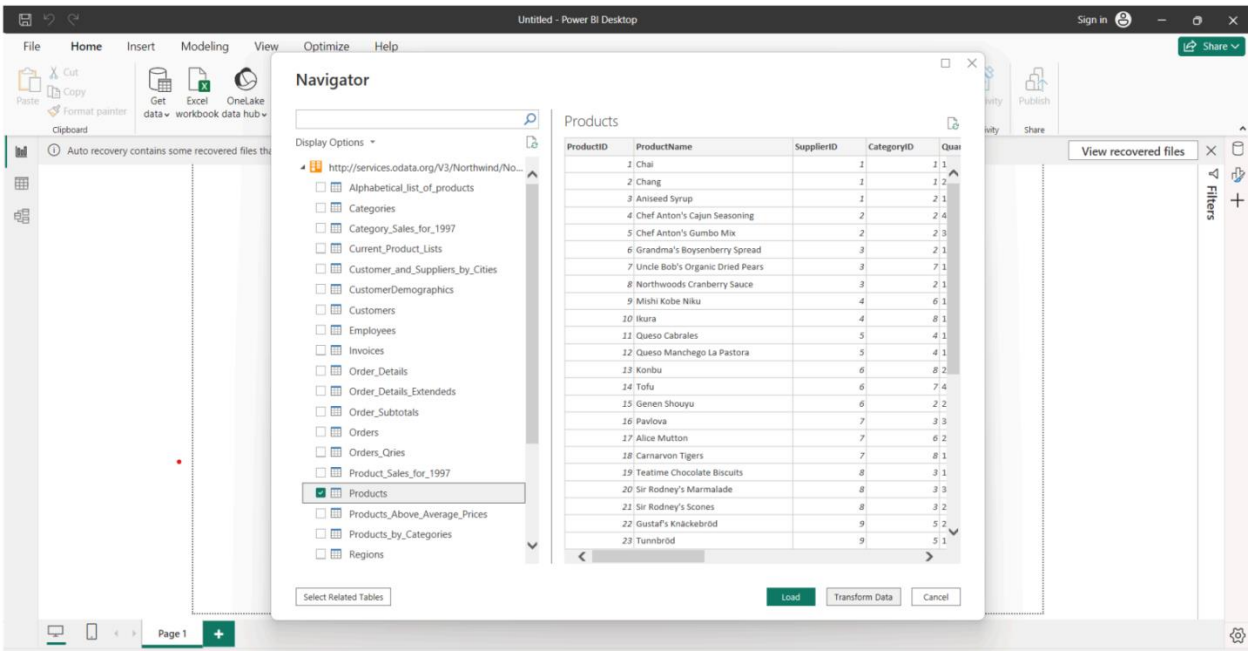
Assignment 8:

Extracting data from source:

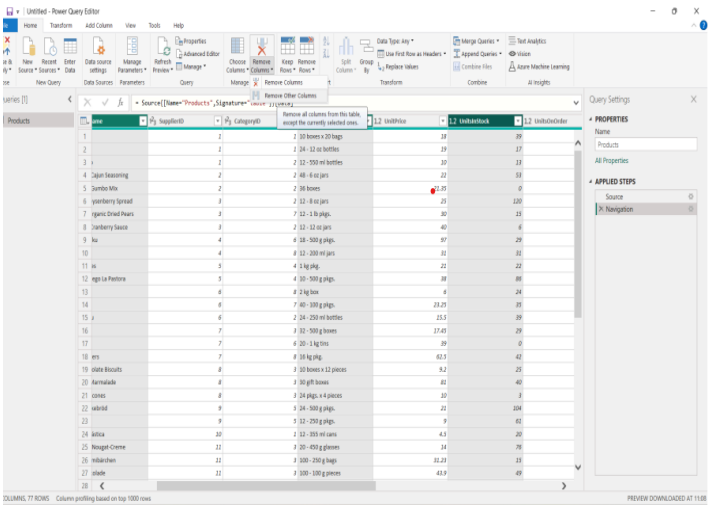


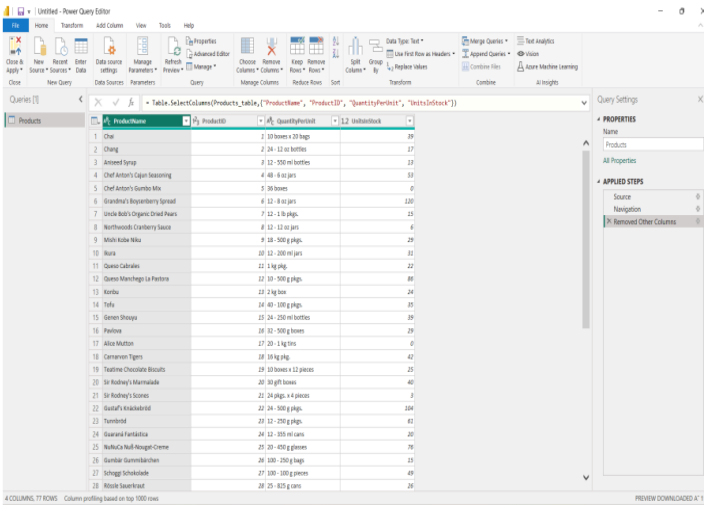


Transforming data:

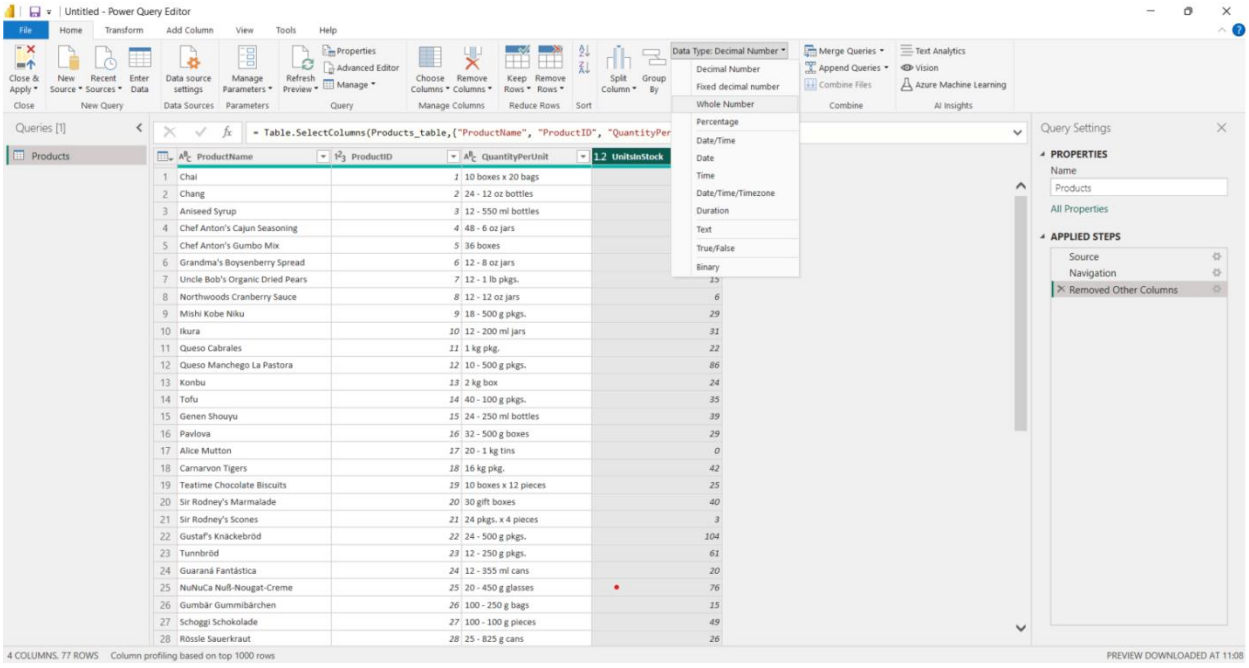


a) Removing unwanted columns

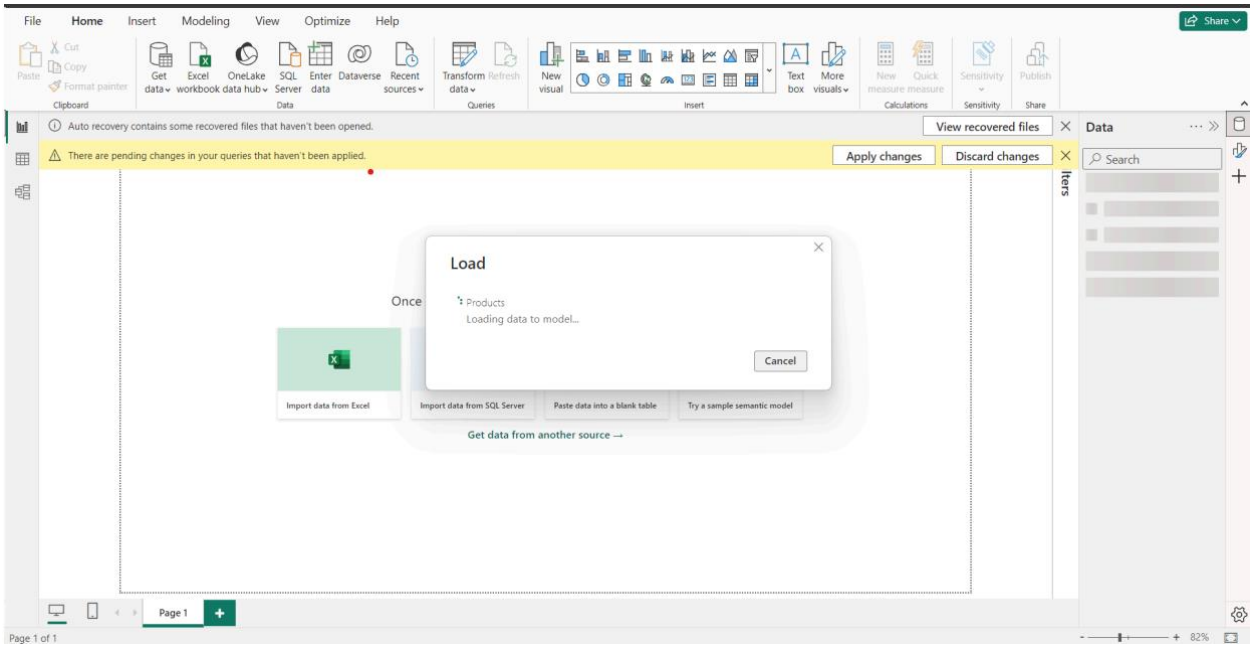




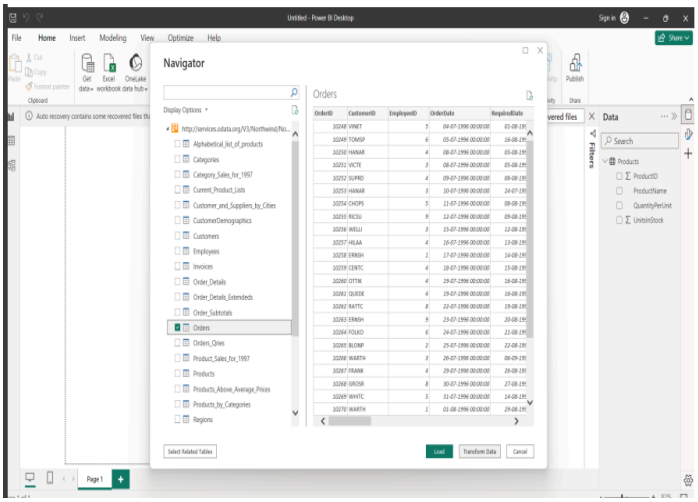
b) Changing Data Type

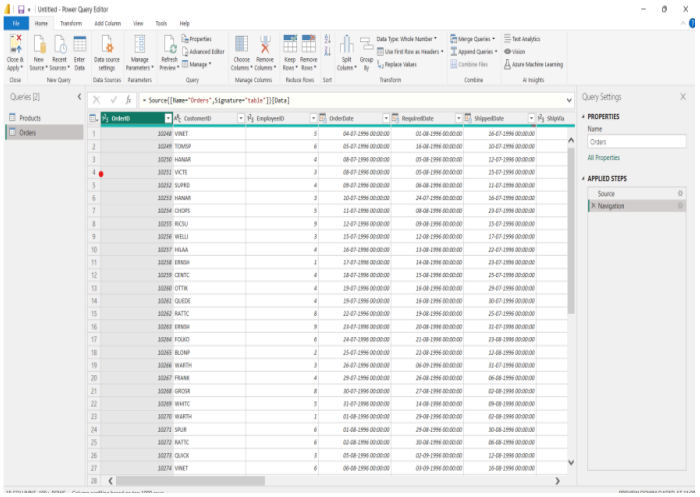


Loading the Data:



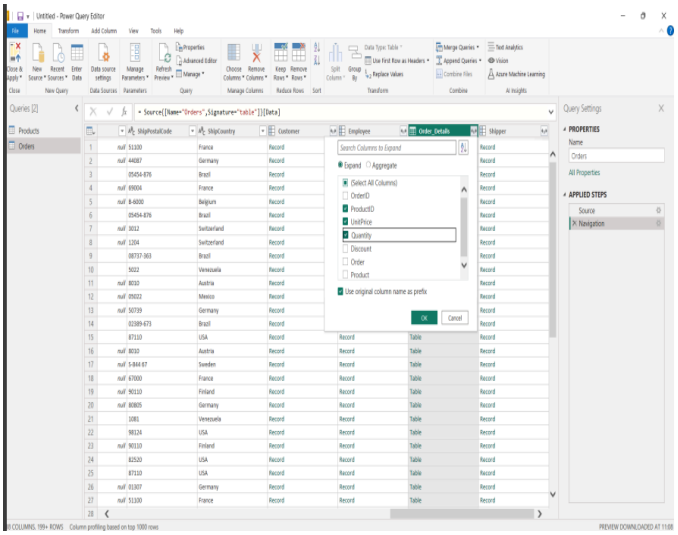
Extracting orders data:

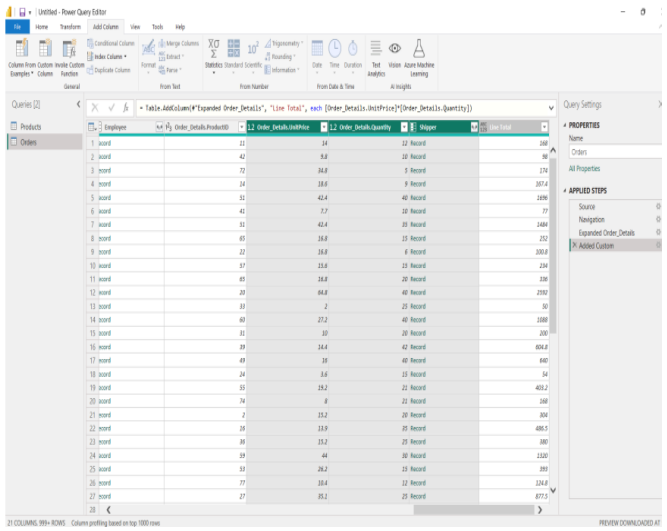




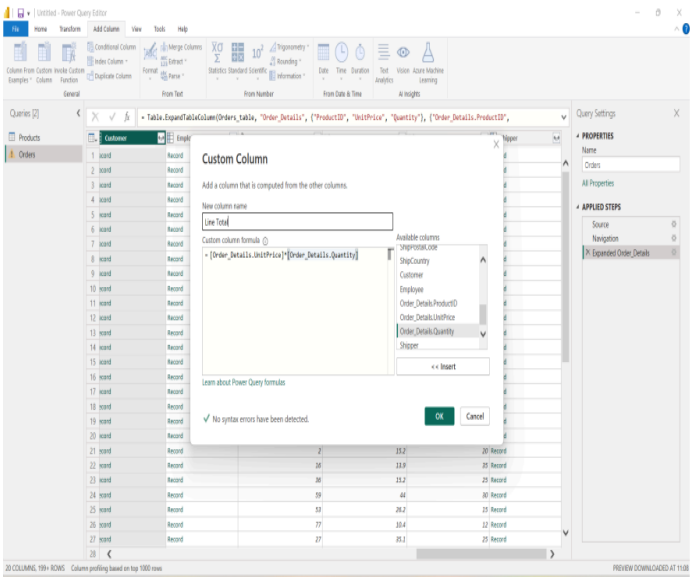
Transforming Orders Data:

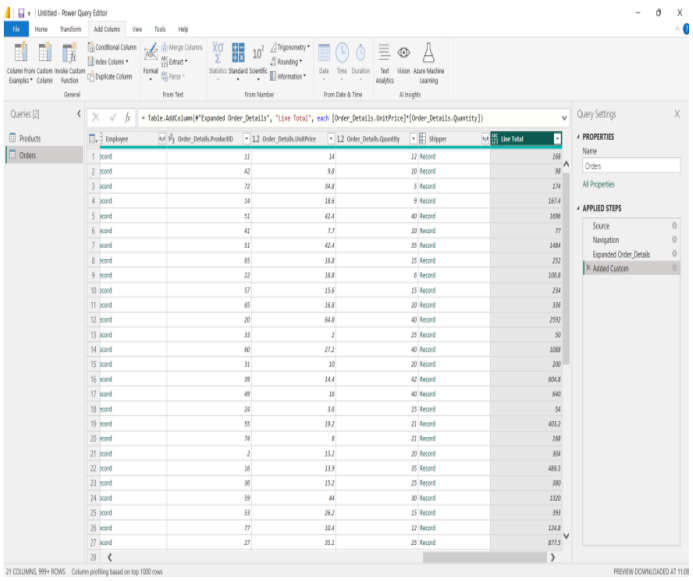
a) Expanding OrderDetails Column:



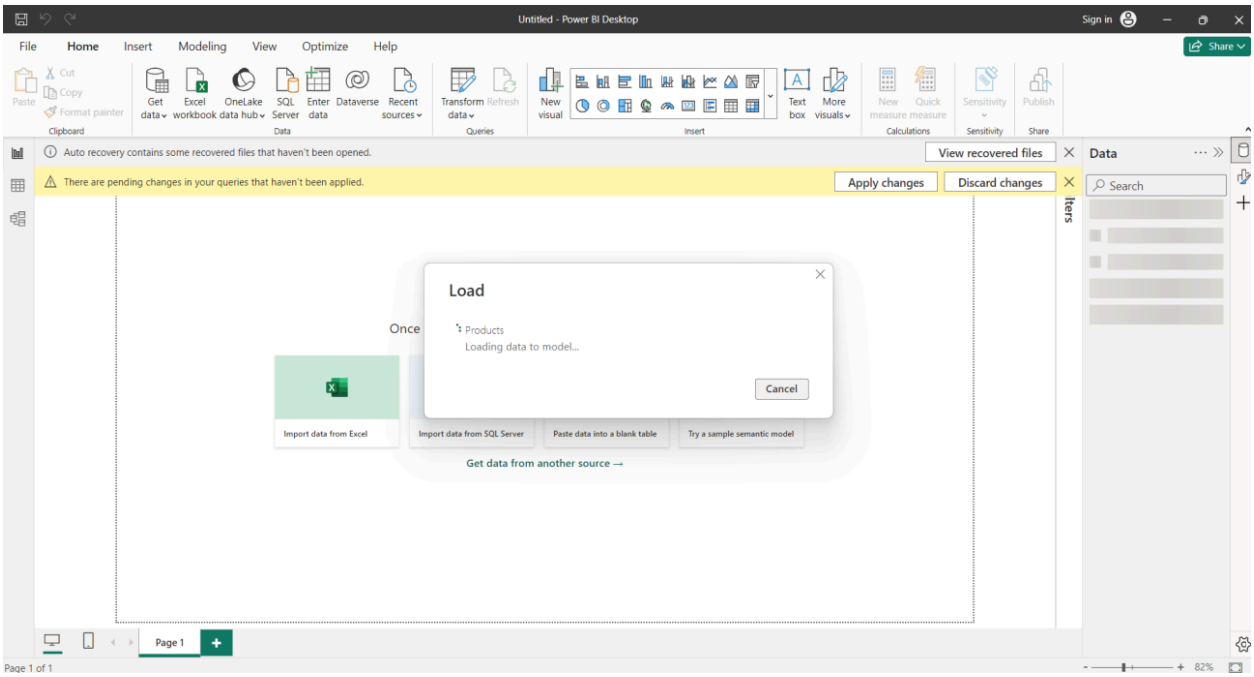


b) Creating a new Custom Columns:





Load:


Untitled - Power BI Desktop
Sign in
File
Home
Insert
Modeling
View
Optimize
Help
Share
Auto recovery contains some recovered files that haven't been opened.
View recovered files
There are pending changes in your queries that haven't been applied.
Apply changes
Discard changes
Data
Search
Load
Products
Loading data to model...
Cancel
Once
Import data from Excel
Import data from SQL Server
Paste data into a blank table
Try a sample semantic model
Get data from another source →
Page 1
Page 1 of 1
82%

Assignment 9:

```
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))
```

OUTPUT:

```
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
```

Assignment 10:

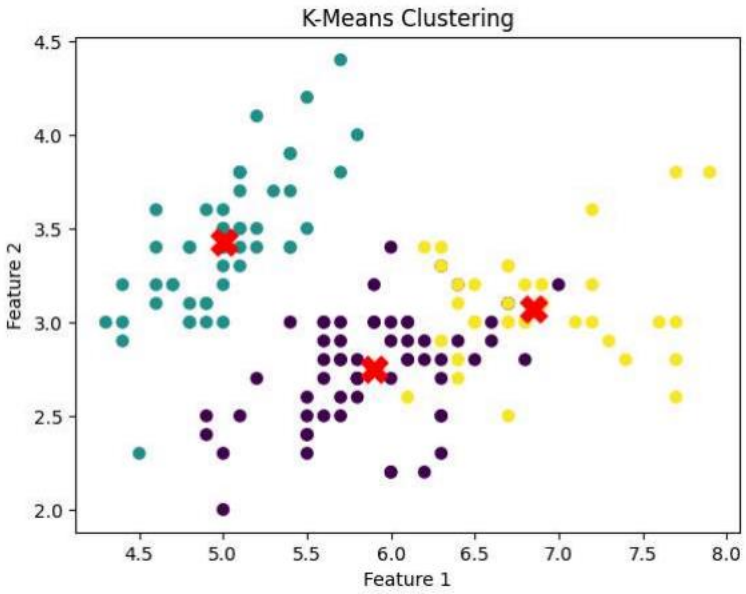
```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt


# 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(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()
```

OUTPUT:



Assignment 1:

Client:

exp 1 > client.py > ...

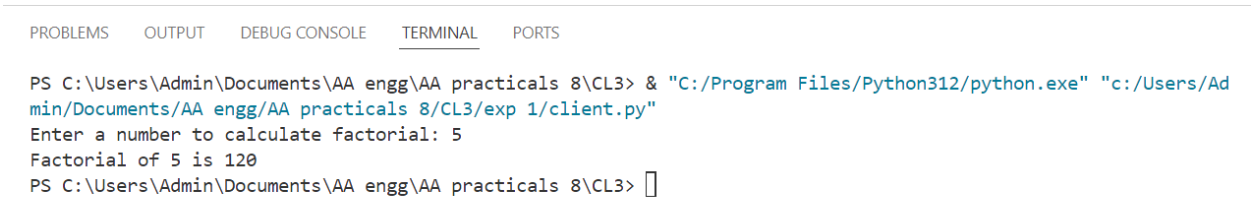
```python
import xmlrpc.client

def main():
    server = xmlrpc.client.ServerProxy("http://localhost:8000/")
    try:
        n = int(input("Enter a number to calculate factorial: "))
        result = server.factorial(n)
        print("Factorial of", n, "is", result)
    except ValueError:
        print("Please enter a valid integer.")

if __name__ == "__main__":
    main()

```

Server:

exp 1 > server.py > ...

```python
from xmlrpc.server import SimpleXMLRPCServer

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

server = SimpleXMLRPCServer(("localhost", 8000))
server.register_function(factorial, "factorial")
print("Server is ready to accept requests...")
server.serve_forever()

```

OUTPUT:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3> & "C:/Program Files/Python312/python.exe" "c:/Users/Admin/Documents/AA engg/AA practicals 8/CL3/exp 1/client.py"
Enter a number to calculate factorial: 5
Factorial of 5 is 120
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3> & "C:/Program Files/Python312/python.exe" "c:/Users/Admin/Documents/AA engg/AA practicals 8/CL3/exp 1/server.py"
Server is ready to accept requests...
127.0.0.1 - - [19/Apr/2024 11:15:52] "POST / HTTP/1.1" 200 -
```

Assignment 2:

Server:

exp 2 > server.py > ...

```python
import Pyro4

@Pyro4.expose
class StringConcatenator(object):
    def concatenate_strings(self, str1, str2):
        return str1 + str2

daemon = Pyro4.Daemon()
uri = daemon.register(StringConcatenator)

print("Ready. Object uri =", uri)
daemon.requestLoop()

```

OUTPUT:

```
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3> & "C:/Program Files/Python312/python.exe" "c:/Users/Admin/Documents/AA engg/AA practicals 8/CL3/exp 2/server.py"
Ready. Object uri = PYRO:obj_b23280566ce14b179c12378cdd471a82@localhost:59837
```

CLIENT:

exp 2 > client.py > ...

```python
import Pyro4

uri = "PYRO:obj_b23280566ce14b179c12378cdd471a82@localhost:59837"
string_concatenator = Pyro4.Proxy(uri)

str1 = "Hello, "
str2 = "World!"
result = string_concatenator.concatenate_strings(str1, str2)
print("Concatenated String:", result)

```

OUTPUT:

```
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3> & "C:/Program Files/Python312/python.exe" "c:/Users/Admin/Documents/AA engg/AA practicals 8/CL3/exp 2/client.py"
Concatenated String: Hello, World!
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3>
```

Assignment 4:

```
def fuzzy_union(set1, set2):
    result = {}
    for element in set1:
        result[element] = max(set1.get(element, 0), set2.get(element, 0))
    for element in set2:
        if element not in result:
            result[element] = set2[element]
    return result

def fuzzy_intersection(set1, set2):
    result = {}
    for element in set1:
        if element in set2:
            result[element] = min(set1[element], set2[element])
    return result

def fuzzy_complement(set1):
    result = {}
    for element in set1:
        result[element] = 1 - set1[element]
    return result

def fuzzy_difference(set1, set2):
    complement_set2 = fuzzy_complement(set2)
    return fuzzy_intersection(set1, complement_set2)

def cartesian_product(set1, set2):
    result = {}
    for element1 in set1:
        for element2 in set2:
            result[(element1, element2)] = min(set1[element1], set2[element2])
    return result

def max_min_composition(relation1, relation2):
    result = {}
    for (a, b) in relation1:
        for (c, d) in relation2:
            if b == c:
                result[(a, d)] = max(result.get((a, d), 0), min(relation1[(a, b)], relation2[(c, d)]))
    return result
# Example usage
set1 = {'a': 0.7, 'b': 0.5, 'c': 0.3}
set2 = {'b': 0.6, 'c': 0.4, 'd': 0.2}

print("Union:", fuzzy_union(set1, set2))
print("Intersection:", fuzzy_intersection(set1, set2))
print("Complement of set1:", fuzzy_complement(set1))
print("Difference of set1 and set2:", fuzzy_difference(set1, set2))

relation1 = {('x', 'y'): 0.8, ('y', 'z'): 0.5}
relation2 = {('y', 'w'): 0.7, ('w', 'z'): 0.4}

print("Cartesian product of relation1 and relation2:")
print(cartesian_product(relation1, relation2))

print("Max-min composition of relation1 and relation2:")
print(max_min_composition(relation1, relation2))
```

OUTPUT:

```
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3> & "C:/Program Files/Python312/python.exe" "c:/Users/Admin/Documents/AA engg/AA practicals 8/CL3/exp 4/exp4.py"
Union: {'a': 0.7, 'b': 0.6, 'c': 0.4, 'd': 0.2}
Intersection: {'b': 0.5, 'c': 0.3}
Complement of set1: {'a': 0.3000000000000004, 'b': 0.5, 'c': 0.7}
Difference of set1 and set2: {'b': 0.4, 'c': 0.3}
Cartesian product of relation1 and relation2:
{(('x', 'y'), ('y', 'w')): 0.7, (('x', 'y'), ('w', 'z')): 0.4, (('y', 'z'), ('y', 'w')): 0.5, (('y', 'z'), ('w', 'z')): 0.4}
Max-min composition of relation1 and relation2:
{('x', 'w'): 0.7}
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3>
```

Assignment 5:

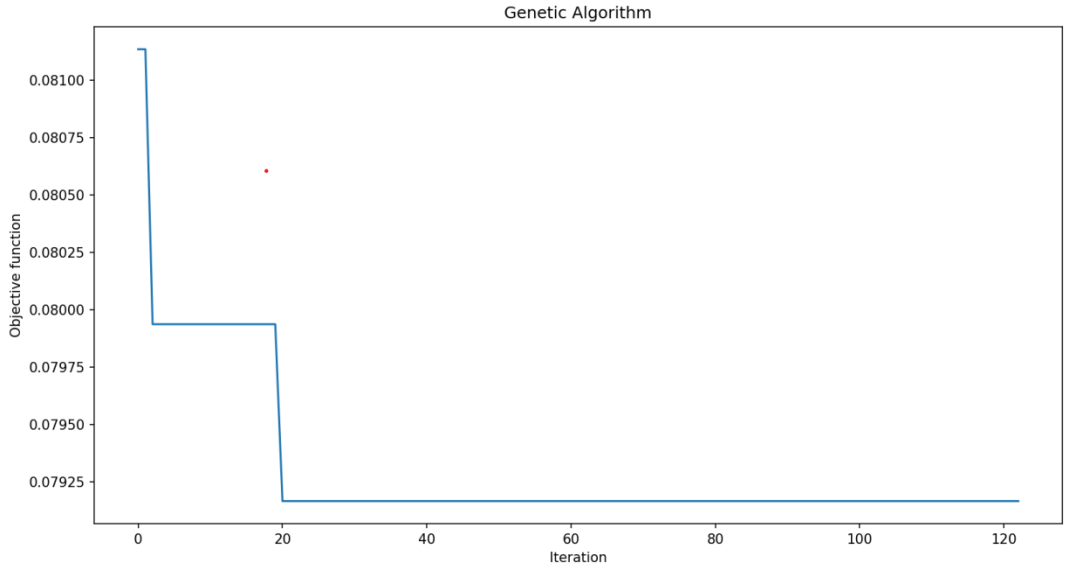
exp 5 > exp5.py > ...

```python
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from geneticalgorithm import geneticalgorithm as ga

# Generate synthetic dataset for spray drying of coconut milk
num_samples = 1000
X = np.random.rand(num_samples, 5)  # Features
y = np.random.rand(num_samples)      # Target

# Split 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)

# Define the neural network model
def create_model(layers, activation):
    activation_functions = ['identity', 'logistic', 'tanh', 'relu']
    chosen_activation = activation_functions[int(round(activation * (len(activation_functions) - 1)))]
    layers = [int(round(layer)) for layer in layers]  # Ensure layers are integers
    model = MLPRegressor(hidden_layer_sizes=tuple(layers), activation=chosen_activation, random_state=42)
    return model

# Define the objective function to be minimized
def objective_function(params):
    layers = params[0:3]  # Number of neurons in each hidden layer
    activation = params[3]  # Activation function

    model = create_model(layers, activation)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)

    return mse

# Define the bounds and constraints for the genetic algorithm parameters
varbound = np.array([[10, 100], [10, 100], [10, 100], [0, 1]])  # Bounds for neurons in each layer and activation function
vartype = np.array([['int'], ['int'], ['int'], ['real']])  # Variable type: integer for neurons, real for activation function

# Initialize and run the genetic algorithm with default values for 'max_iteration_without_improv', 'crossover_type', 'elit_ratio', 'crossover_probability', 'mutation_probability', and 'parents_portion'
algorithm_param = {'max_num_iteration': 1000, 'population_size': 100}
if 'max_iteration_without_improv' not in algorithm_param:
    algorithm_param['max_iteration_without_improv'] = 100
if 'crossover_type' not in algorithm_param:
    algorithm_param['crossover_type'] = 'uniform'
if 'elit_ratio' not in algorithm_param:
    algorithm_param['elit_ratio'] = 0.02
if 'crossover_probability' not in algorithm_param:
    algorithm_param['crossover_probability'] = 0.9
if 'mutation_probability' not in algorithm_param:
    algorithm_param['mutation_probability'] = 0.1
if 'parents_portion' not in algorithm_param:
    algorithm_param['parents_portion'] = 0.5

model = ga(function=objective_function, dimension=4, variable_type_mixed=vartype, variable_boundaries=varbound, algorithm_parameters=algorithm_param)
model.run()

# Get the best parameters and minimum MSE
best_params = model.output_dict['variable']
best_mse = model.output_dict['function']

print("Best parameters found:", best_params)
print("Minimum MSE:", best_mse)
```

OUTPUT:

```
C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
_____ 0.1% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
_____ 0.3% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
_____ 0.4% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
_____ 0.5% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
_____ 1.0% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
|_____ 1.1% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
|_____ 1.2% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
|_____ 1.3% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
|_____ 1.6% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
||||_____ 8.7% GA is running...C:\Users\Admin\AppData\Roaming\Python\Python312\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
 The best solution found:
 [30.         10.         10.          0.69409117]

 Objective function:
 0.07916595901418105
```

Figure 1



Assignment 6:

exp 6 > exp6.py > ...

```python
import numpy as np

class ClonalSelectionAlgorithm:
    def __init__(self, objective_function, num_variables, num_population=100, num_clones=10, mutation_rate=0.1, max_generations=100):
        self.objective_function = objective_function
        self.num_variables = num_variables
        self.num_population = num_population
        self.num_clones = num_clones
        self.mutation_rate = mutation_rate
        self.max_generations = max_generations

    def initialize_population(self):
        return np.random.rand(self.num_population, self.num_variables)

    def clone(self, population, fitness):
        num_selected = int(self.num_clones / 2)
        sorted_indices = np.argsort(fitness)
        selected_population = population[sorted_indices[:num_selected]]
        return np.repeat(selected_population, self.num_clones // num_selected, axis=0)

    def mutate(self, clones):
        mutation_mask = np.random.rand(*clones.shape) < self.mutation_rate
        mutations = np.random.rand(*clones.shape) * 0.1  # Adjust mutation range based on problem domain
        clones[mutation_mask] += mutations[mutation_mask]
        return clones

    def select(self, clones, population, fitness):
        merged_population = np.vstack((population, clones))
        merged_fitness = np.concatenate((fitness, self.objective_function(clones)))
        sorted_indices = np.argsort(merged_fitness)
        selected_population = merged_population[sorted_indices[:self.num_population]]
        selected_fitness = merged_fitness[sorted_indices[:self.num_population]]
        return selected_population, selected_fitness

        # Ensure the size of merged_fitness is not greater than num_population
        if len(merged_fitness) > self.num_population:
            merged_fitness = merged_fitness[:self.num_population]

        return merged_population[sorted_indices[:self.num_population]], merged_fitness

    def optimize(self):
        population = self.initialize_population()
        best_solution = None
        best_fitness = np.inf

        for generation in range(self.max_generations):
            fitness = self.objective_function(population)
            if np.min(fitness) < best_fitness:
                best_solution = population[np.argmin(fitness)]
                best_fitness = np.min(fitness)

            clones = self.clone(population, fitness)
            clones = self.mutate(clones)
            population, fitness = self.select(clones, population, fitness)

        return best_solution, best_fitness

# Example usage:
def objective_function(x):
    return np.sum(x**2, axis=1)  # Minimize sum of squares

num_variables = 5
csa = ClonalSelectionAlgorithm(objective_function, num_variables)
best_solution, best_fitness = csa.optimize()
print("Best solution:", best_solution)
print("Best fitness:", best_fitness)
```

OUTPUT:

```
/CL3/exp 6/exp6.py"
Best solution: [0.19807388 0.12044344 0.30868019 0.07941894 0.1289313 ]
Best fitness: 0.17195398869316517
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3>
```

Assignment 7:

```python
import numpy as np

# Generate dummy data for demonstration purposes (replace this with your actual data)
def generate_dummy_data(samples=100, features=10):
    data = np.random.rand(samples, features)
    labels = np.random.randint(0, 2, size=samples)
    return data, labels

# Define the AIRS algorithm
class AIRS:
    def __init__(self, num_detectors=10, hypermutation_rate=0.1):
        self.num_detectors = num_detectors
        self.hypermutation_rate = hypermutation_rate

    def train(self, X, y):
        self.detectors = X[np.random.choice(len(X), self.num_detectors, replace=False)]

    def predict(self, X):
        predictions = []
        for sample in X:
            distances = np.linalg.norm(self.detectors - sample, axis=1)
            prediction = np.argmax(distances)  # Assuming detectors have class labels
            predictions.append(prediction)
        return predictions

# Generate dummy data
data, labels = generate_dummy_data()
```

```python

# Split data into training and testing sets
split_ratio = 0.8
split_index = int(split_ratio * len(data))
train_data, test_data = data[:split_index], data[split_index:]
train_labels, test_labels = labels[:split_index], labels[split_index:]

# Initialize and train AIRS
airs = AIRS(num_detectors=10, hypermutation_rate=0.1)
airs.train(train_data, train_labels)

# Test AIRS on the test set
predictions = airs.predict(test_data)

# Evaluate accuracy
accuracy = np.mean(predictions == test_labels)
print(f"Accuracy: {accuracy}")

```

OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3> & "C:/Program Files/Python312/python.exe" "c:/Users/Admin/Documents/AA engg/AA practicals 8
/CL3/exp 7/exp7.py"
Accuracy: 0.15
PS C:\Users\Admin\Documents\AA engg\AA practicals 8\CL3>
```

Assignment 8:

exp 8 > exp8.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

```
import random
from deap import base, creator, tools, algorithms
```

[2] ✓ 0.5s

```
def eval_func(individual):
    return sum(x ** 2 for x in individual),

# DEAP setup
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
toolbox = base.Toolbox()
toolbox.register("attr_float", random.uniform, -5.0, 5.0)  # Example: Float values between -5 and 5
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_float, n=3)  # Example: 3-dimensional individual
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
# Evaluation function and genetic operators
toolbox.register("evaluate", eval_func)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
```

[3] ✓ 0.0s

```
# Create population
population = toolbox.population(n=50)
```

[4] ✓ 0.0s

```
# Genetic Algorithm parameters
generations = 20
```

[5] ✓ 0.0s

```
# Run the algorithm
for gen in range(generations):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.1)
    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
        population = toolbox.select(offspring, k=len(population))
```

[6] ✓ 0.1s

```
# Get the best individual after generations
best_ind = tools.selBest(population, k=1)[0]
best_fitness = best_ind.fitness.values[0]
print("Best individual:", best_ind)
print("Best fitness:", best_fitness)
```

[7] ✓ 0.0s

OUTPUT:

```
Best individual: [0.016304314150622588, -0.0014802558794005313, -0.001293290973998092]
Best fitness: 0.00026969441893411674
```

Assignment 9:

```
import csv
from functools import reduce
from collections import defaultdict

# Define mapper function to emit (year, temperature) pairs
def mapper(row):
    year = row["Date/Time"].split("-")[0]  # Extract year from
"Date/Time" column
    temperature = float(row["Temp_C"])  # Convert temperature to
float
    return (year, temperature)

# Define reducer function to calculate sum and count of temperatures
for each year
def reducer(accumulated, current):
    accumulated[current[0]][0] += current[1]
    accumulated[current[0]][1] += 1
    return accumulated

# Read the weather dataset
weather_data = []
with open("weather_data.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        weather_data.append(row)

# Map phase
mapped_data = map(mapper, weather_data)

# Reduce phase
reduced_data = reduce(reducer, mapped_data, defaultdict(lambda: [0,
0]))

# Calculate average temperature for each year
avg_temp_per_year = {year: total_temp / count for year, (total_temp,
count) in reduced_data.items()}

# Find coolest and hottest year
coolest_year = min(avg_temp_per_year.items(), key=lambda x: x[1])
hottest_year = max(avg_temp_per_year.items(), key=lambda x: x[1])

print("Coolest Year:", coolest_year[0], "Average Temperature:",
coolest_year[1])
print("Hottest Year:", hottest_year[0], "Average Temperature:",
hottest year[1])
```

OUTPUT:

```
Coolest Year: 1/15/2012 8:00 Average Temperature: -23.3
Hottest Year: 6/21/2012 15:00 Average Temperature: 33.0
```

Assignment 10:

```
import numpy as np
import random

# Define the distance matrix (distances between cities)
# Replace this with your distance matrix or generate one based on
your problem
# Example distance matrix (replace this with your actual data)
distance_matrix = np.array([
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
])

# Parameters for Ant Colony Optimization
num_ants = 10
num_iterations = 50
evaporation_rate = 0.5
pheromone_constant = 1.0
heuristic_constant = 1.0

# Initialize pheromone matrix and visibility matrix
num_cities = len(distance_matrix)
pheromone = np.ones((num_cities, num_cities))  # Pheromone matrix
visibility = 1 / distance_matrix  # Visibility matrix (inverse of
distance)

# ACO algorithm
for iteration in range(num_iterations):
    ant_routes = []
    for ant in range(num_ants):
        current_city = random.randint(0, num_cities - 1)
        visited_cities = [current_city]
        route = [current_city]

        while len(visited_cities) < num_cities:
            probabilities = []
            for city in range(num_cities):
                if city not in visited_cities:
                    pheromone_value = pheromone[current_city][city]
                    visibility_value = visibility[current_city][city]
                    probability = (pheromone_value **
pheromone_constant) * (visibility_value ** heuristic_constant)
                    probabilities.append((city, probability))
```

```
            probabilities = sorted(probabilities, key=lambda x: x[1],
reverse=True)
            selected_city = probabilities[0][0]
            route.append(selected_city)
            visited_cities.append(selected_city)
            current_city = selected_city

        ant_routes.append(route)

    # Update pheromone levels
    delta_pheromone = np.zeros((num_cities, num_cities))

    for ant, route in enumerate(ant_routes):
        for i in range(len(route) - 1):
            city_a = route[i]
            city_b = route[i + 1]
            delta_pheromone[city_a][city_b] += 1 /
distance_matrix[city_a][city_b]
            delta_pheromone[city_b][city_a] += 1 /
distance_matrix[city_a][city_b]

    pheromone = (1 - evaporation_rate) * pheromone + delta_pheromone

# Find the best route
best_route_index =
np.argmax([sum(distance_matrix[cities[i]][cities[(i + 1) %
num_cities]] for i in range(num_cities)) for cities in ant_routes])
best_route = ant_routes[best_route_index]
shortest_distance = sum(distance_matrix[best_route[i]][best_route[(i
+ 1) % num_cities]] for i in range(num_cities))

print("Best route:", best_route)
print("Shortest distance:", shortest_distance)
```

OUTPUT:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\TSP.py"
d:\BE SEM VIII\CL_III_Code\TSP.py:24: RuntimeWarning: divide by zero encountered in divide
  visibility = 1 / distance_matrix  # Visibility matrix (inverse of distance)
Best route: [0, 1, 3, 2]
Shortest distance: 80
PS D:\BE SEM VIII>
```