kalyan.py

```
import sys
import pandas as pd
import numpy as np
def growth_factor(pred,total):  ## to calculate growth factor (Oi_horizon / Oi_current) ==> horizon is
temp_pred nd Oi_current is temp_total
    temp_pred ,temp_total = 0 , 0
    for i in range(len(total)):
        temp_total += total[i]
        temp_pred += pred[i]
    return temp_pred / temp_total
def Uniform(mat,pred,a):
    total = [0.0 for i in range(len(mat))] #total row sum
    uniform_growth_factor = 0.0
    for i in range(len(mat)):
        for j in range(len(mat[0])):
            total[i] += mat[i][j]
    uniform_growth_factor = growth_factor(pred,total)
    for i in range(len(mat)):
        temp = 0
        for j in range(len(mat[0])):
            mat[i][j] *= uniform_growth_factor # for calculating new matrix for multiplied growth factors
            temp += mat[i][j]
```

```
        total[i] = temp
    data = {
    '1': [mat[i][0] for i in range(len(mat))] ,
    '2': [mat[i][1] for i in range(len(mat))] ,
    '3': [mat[i][2] for i in range(len(mat))] ,
    'Î£': total,
    }

# Create DataFrame
    df = pd.DataFrame(data)

# Set the index for the DataFrame to include serial numbers
    df.index = [1, 2, 3,]

# Display the DataFrame as a table
    print("\nTij_012:")
    print(df)



file_path = 'Assignment_2.xlsx'
print("You have entered Question 1",)
df = pd.read_excel(file_path, sheet_name='Uniform',header=None) #taking excel data
print(df)
```

```
a = int(input("Enter the size of matrix:"))

Tij_012 = df.iloc[:a, :a].values.tolist() #GIven Matrix


print(Tij_012)

Oih_012 = df.values[:a,a+1].tolist() #Given Oi_horizon

print(Oih_012)

Uniform(Tij_012,Oih_012,a)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\gdred\Downloads\Kalyan\kalyan.py", line 2, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

kalyan2.py

```
import sys
import pandas as pd
import numpy as np
def growth_factor(pred,total):
    temp_pred ,temp_total = 0 , 0
    for i in range(len(total)):
        temp_total += total[i]
        temp_pred += pred[i]
    return temp_pred / temp_total

def Average(mat, growth, a):
    total_row = [0.0 for _ in range(a)]
    total_col = [0.0 for _ in range(a)]
    pred_row = [0.0 for _ in range(a)]
    pred_col = [0.0 for _ in range(a)]
    # Calculate total_row and total_col
    for i in range(a):
        for j in range(a):
            total_row[i] += mat[i][j]
            total_col[j] += mat[i][j]  # Corrected to mat[i][j] for column sums

    # Calculate pred_row and pred_col based on growth factors
```

```
    for i in range(a):
        pred_row[i] = total_row[i] * growth[i]
        pred_col[i] = total_col[i] * growth[i]

    # Generate initial DataFrame
    generate(mat, total_row, growth, pred_row, total_col, pred_col)
    # Start calculating the new matrix and updating growth factors
    calculate_new_matrix(mat, growth, pred_row, pred_col)

def calculate_new_matrix(mat, growth, pred_row, pred_column):
    previous_growth = np.array(growth)
    iteration_count = 0
    a = len(growth)

    # Loop until convergence criteria is met
    while True:
        iteration_count += 1

        # Initialize the new matrix
        new_mat = np.zeros_like(mat, dtype=float)
        total_old_row = np.sum(mat,axis=1)
        f_value = growth_factor(pred_row,np.sum(mat,axis=1))
        # Update the new matrix based on the growth factors
        for i in range(a):
```

```
            for j in range(a):
                new_mat[i][j] = mat[i][j] * (previous_growth[i] + previous_growth[j]) / 2

        # Calculate total_row and total_column
        total_row = np.sum(new_mat, axis=1)
        total_column = np.sum(new_mat, axis=0)

        # Normalize total_row with pred_row and total_column with pred_column
        GF_Revised_row = pred_row / total_row
        GF_Revised_col = pred_column / total_column
        print(f_value)
        # Check for convergence
        if all(0.97 <= value <= 1.03 for value in GF_Revised_row):
            print("Convergence achieved after iterations:", iteration_count)
            break

        # Generate the DataFrame and prepare for the next iteration
        generate(new_mat, total_row, GF_Revised_row, pred_row, total_column, pred_column)
        previous_growth = GF_Revised_row.copy()
        mat = new_mat  # Update mat for next iteration

    return new_mat, total_row, total_column, growth

def generate(new_mat, total_row, GF_Revised_row, pred_row, total_column, pred_column):
```

```
# Prepare the data dictionary for DataFrame creation
data = {
    '1': [new_mat[i][0] for i in range(len(new_mat))],
    '2': [new_mat[i][1] for i in range(len(new_mat))],
    '3': [new_mat[i][2] for i in range(len(new_mat))],
    'Oi_base': total_row,
    'G-f': GF_Revised_row,
    'Oi_horizon': pred_row
}

# Create DataFrame
df = pd.DataFrame(data)

# Replace None with 0 for total_column
total_col = [x if x is not None else 0 for x in total_column]

# Add additional rows for totals and growth factors
df.loc['Dj_base'] = [total_column[i] if i < len(total_column) else "" for i in range(df.shape[1])]
df.loc['G.F'] = [GF_Revised_row[i] if i < len(GF_Revised_row) else "" for i in range(df.shape[1])]
df.loc['Dj_horizon'] = [pred_column[i] if i < len(pred_column) else "" for i in range(df.shape[1])]

# Set the index for the DataFrame
df.index = [1, 2, 3, 'Dj_base', 'G.F', 'Dj_horizon']
```

```
    # Round all numeric columns to 3 decimal places
    df = df.round(3)

    # Display the DataFrame
    print("\nTij_012:")
    print(df)



file_path = 'Assignment_2.xlsx'
print("You have entered Question 2")
df = pd.read_excel(file_path, sheet_name='Average',header=None)
print(df)
a = int(input("Enter the size of matrix:"))
Tij_012 = df.iloc[:a, :a].values.tolist()

print(Tij_012)
Oih_012 = df.values[:a,a+1].tolist()
print(Oih_012)
Average(Tij_012,Oih_012,a)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\gdred\Downloads\Kalyan\kalyan2.py", line 2, in <module>
```

```
import pandas as pd
```

```
ModuleNotFoundError: No module named 'pandas'
```

kalyan3.py

```
import sys
import pandas as pd
import numpy as np
def growth_factor(pred,total):
    temp_pred ,temp_total = 0 , 0
    for i in range(len(total)):
        temp_total += total[i]
        temp_pred += pred[i]
    return temp_pred / temp_total

def Detroit(mat, growth, a):
    total_row = [0.0 for _ in range(a)]
    total_col = [0.0 for _ in range(a)]
    pred_row = [0.0 for _ in range(a)]
    pred_col = [0.0 for _ in range(a)]
    # Calculate total_row and total_col
    for i in range(a):
        for j in range(a):
            total_row[i] += mat[i][j]
            total_col[j] += mat[i][j]  # Corrected to mat[i][j] for column sums

    # Calculate pred_row and pred_col based on growth factors
```

```
    for i in range(a):
        pred_row[i] = total_row[i] * growth[i]
        pred_col[i] = total_col[i] * growth[i]

    # Generate initial DataFrame
    generate2(mat, total_row, growth, pred_row, total_col, pred_col)
    # Start calculating the new matrix and updating growth factors
    calculate_new_matrix(mat, growth, pred_row, pred_col)

def calculate_new_matrix(mat, growth, pred_row, pred_column):
    previous_growth = np.array(growth)
    iteration_count = 0
    a = len(growth)

    # Loop until convergence criteria is met
    while True:
        iteration_count += 1

        # Initialize the new matrix
        new_mat = np.zeros_like(mat, dtype=float)
        total_old_row = np.sum(mat,axis=1)
        f_value = growth_factor(pred_row,np.sum(mat,axis=1))
        # Update the new matrix based on the growth factors
        for i in range(a):
```

```
            for j in range(a):
                new_mat[i][j] = mat[i][j] * (previous_growth[i] * previous_growth[j]) / f_value

        # Calculate total_row and total_column
        total_row = np.sum(new_mat, axis=1)
        total_column = np.sum(new_mat, axis=0)

        # Normalize total_row with pred_row and total_column with pred_column
        GF_Revised_row = pred_row / total_row
        GF_Revised_col = pred_column / total_column
        print(f_value)
        # Check for convergence
        if all(0.97 <= value <= 1.03 for value in GF_Revised_row):
            print("Convergence achieved after iterations:", iteration_count)
            break

        # Generate the DataFrame and prepare for the next iteration
        generate2(new_mat, total_row, GF_Revised_row, pred_row, total_column, pred_column)
        previous_growth = GF_Revised_row.copy()
        mat = new_mat  # Update mat for next iteration

    return new_mat, total_row, total_column, growth

def generate2(new_mat, total_row, GF_Revised_row, pred_row, total_column, pred_column):
```

```
# Prepare the data dictionary for DataFrame creation
data = {
    '1': [new_mat[i][0] for i in range(len(new_mat))],
    '2': [new_mat[i][1] for i in range(len(new_mat))],
    '3': [new_mat[i][2] for i in range(len(new_mat))],
    '4': [new_mat[i][3] for i in range(len(new_mat))],
    'Oi_base': total_row,
    'G-f': GF_Revised_row,
    'Oi_horizon': pred_row
}

# Create DataFrame
df = pd.DataFrame(data)

# Replace None with 0 for total_column
total_col = [x if x is not None else 0 for x in total_column]

# Add additional rows for totals and growth factors
df.loc['Dj_base'] = [total_column[i] if i < len(total_column) else "" for i in range(df.shape[1])]
df.loc['G.F'] = [GF_Revised_row[i] if i < len(GF_Revised_row) else "" for i in range(df.shape[1])]
df.loc['Dj_horizon'] = [pred_column[i] if i < len(pred_column) else "" for i in range(df.shape[1])]

# Set the index for the DataFrame
df.index = [1, 2, 3,4, 'Dj_base', 'G.F', 'Dj_horizon']
```

```
    # Round all numeric columns to 3 decimal places
    df = df.round(3)

    # Display the DataFrame
    print("\nTij_012:")
    print(df)
```

```
file_path = 'Assignment_2.xlsx'
print("You have entered Question 3")
df = pd.read_excel(file_path, sheet_name='Detroit',header=None)
print(df)
a = int(input("Enter the size of matrix:"))
Tij_012 = df.iloc[:a, :a].values.tolist()

print(Tij_012)
Oih_012 = df.values[:a,a+1].tolist()
print(Oih_012)
Detroit(Tij_012,Oih_012,a)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\gdred\Downloads\Kalyan\kalyan3.py", line 2, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

kalyan4.py

```
import sys
import pandas as pd
import numpy as np
def LValue(mat,total_row,growth):
    L_values = [0.0 for _ in range(len(growth))]
    for i in range(len(growth)):
        denominator=(np.array(mat[i])*np.array(growth)).sum()
        L_values[i] = total_row[i]/denominator
    return L_values
def growth_factor(pred,total):
    temp_pred ,temp_total = 0 , 0
    for i in range(len(total)):
        temp_total += total[i]
        temp_pred += pred[i]
    return temp_pred / temp_total
def Fratar(mat, growth, a):
    total_row = [0.0 for _ in range(a)]
    total_col = [0.0 for _ in range(a)]
    pred_row = [0.0 for _ in range(a)]
    pred_col = [0.0 for _ in range(a)]
    # Calculate total_row and total_col
    for i in range(a):
```

```
        for j in range(a):
            total_row[i] += mat[i][j]
            total_col[j] += mat[i][j]  # Corrected to mat[i][j] for column sums

    # Calculate pred_row and pred_col based on growth factors
    for i in range(a):
        pred_row[i] = total_row[i] * growth[i]
        pred_col[i] = total_col[i] * growth[i]

    # Generate initial DataFrame
    L = LValue(mat,total_row,growth)

    generate2(mat, total_row, growth, pred_row, total_col, pred_col,L)
    # Start calculating the new matrix and updating growth factors
    calculate_new_matrix(mat, growth, pred_row, pred_col,L)

def calculate_new_matrix(mat, growth, pred_row, pred_column,L):
    previous_growth = np.array(growth)
    iteration_count = 0
    a = len(growth)
    # Loop until convergence criteria is met
    while True:
        iteration_count += 1
```

```
# Initialize the new matrix
new_mat = np.zeros_like(mat, dtype=float)
total_old_row = np.sum(mat,axis=1)
f_value = growth_factor(pred_row,np.sum(mat,axis=1))
# Update the new matrix based on the growth factors
for i in range(a):
    for j in range(a):
        new_mat[i][j] = (mat[i][j] * (previous_growth[i] * previous_growth[j])) * (L[i]+L[j])/2

# Calculate total_row and total_column
total_row = np.sum(new_mat, axis=1)
total_column = np.sum(new_mat, axis=0)

# Normalize total_row with pred_row and total_column with pred_column
GF_Revised_row = pred_row / total_row
GF_Revised_col = pred_column / total_column
L = LValue(new_mat,total_row,GF_Revised_row)
print(f_value)
# Check for convergence
if all(0.97<=value<=1.034 for value in GF_Revised_row):
    print("Convergence achieved after iterations:", iteration_count)
    break

# Generate the DataFrame and prepare for the next iteration
```

```
        generate2(new_mat, total_row, GF_Revised_row, pred_row, total_column, pred_column,L)
        previous_growth = GF_Revised_row.copy()
        mat = new_mat  # Update mat for next iteration

    return new_mat, total_row, total_column, growth



def generate2(new_mat, total_row, GF_Revised_row, pred_row, total_column, pred_column,L):
    # Prepare the data dictionary for DataFrame creation
    data = {
        '1': [new_mat[i][0] for i in range(len(new_mat))],
        '2': [new_mat[i][1] for i in range(len(new_mat))],
        '3': [new_mat[i][2] for i in range(len(new_mat))],
        '4': [new_mat[i][3] for i in range(len(new_mat))],
        'Oi_base': total_row,
        'G-f': GF_Revised_row,
        'Oi_horizon': pred_row,
        'L':L
    }

    # Create DataFrame
    df = pd.DataFrame(data)

    # Replace None with 0 for total_column
```

```
    total_col = [x if x is not None else 0 for x in total_column]

    # Add additional rows for totals and growth factors
    df.loc['Dj_base'] = [total_column[i] if i < len(total_column) else "" for i in range(df.shape[1])]
    df.loc['G.F'] = [GF_Revised_row[i] if i < len(GF_Revised_row) else "" for i in range(df.shape[1])]
    df.loc['Dj_horizon'] = [pred_column[i] if i < len(pred_column) else "" for i in range(df.shape[1])]

    # Set the index for the DataFrame
    df.index = [1, 2, 3,4, 'Dj_base', 'G.F', 'Dj_horizon']

    # Round all numeric columns to 3 decimal places
    df = df.round(3)

    # Display the DataFrame
    print("\nTij_012:")
    print(df)



file_path = 'Assignment_2.xlsx'
sheet = ['Uniform','Average','Detroit','Fratar','Furness']
print("You have entered Question 4")
df = pd.read_excel(file_path, sheet_name="Fratar",header=None)
print(df)
a = int(input("Enter the size of matrix:"))
```

```
Tij_012 = df.iloc[:a, :a].values.tolist()


print(Tij_012)

Oih_012 = df.values[:a,a+1].tolist()

print(Oih_012)

Fratar(Tij_012,Oih_012,a)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\gdred\Downloads\Kalyan\kalyan4.py", line 2, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

kalyan5.py

```
import sys
import pandas as pd
import numpy as np
def growth_factor(pred,total):
    temp_pred ,temp_total = 0 , 0
    for i in range(len(total)):
        temp_total += total[i]
        temp_pred += pred[i]
    return temp_pred / temp_total
def Uniform(mat,growth,a):
    Oi_horizon = [0.0 for i in range(len(mat))]
    total = [0.0 for i in range(len(mat))]
    growth_factor = 0.0
    for i in range(len(mat)):
        for j in range(len(mat[0])):
            total[i] += mat[i][j]
        Oi_horizon[i] = total[i] * growth[i]

    generate_Row(mat, total,growth, Oi_horizon)
    calculate_new_matrix(mat,growth,Oi_horizon)
def calculate_new_matrix(mat, growth,Oi_horizon):
    previous_growth = np.array(growth)
```

```
iteration_count = 0
a = len(growth)

# Loop until convergence criteria is met
while True:
    iteration_count += 1

    # Initialize the new matrix
    new_mat = np.zeros_like(mat, dtype=float)
    # Update the new matrix based on the growth factors
    if iteration_count % 2:
        for i in range(a):
            for j in range(a):
                new_mat[i][j] = mat[i][j] * previous_growth[i]
        total_column = np.sum(new_mat, axis=0)

    # Normalize total_row with pred_row and total_column with pred_column

        GF_Revised = Oi_horizon / total_column
        # Check for convergence
        if all(0.97 <= value <= 1.03 for value in GF_Revised):
            print("Convergence achieved after iterations:", iteration_count)
            break
```

```
        # Generate the DataFrame and prepare for the next iteration
        generate_Column(new_mat, total_column, Oi_horizon,GF_Revised)

    else:
        for j in range(a):
            for i in range(a):
                new_mat[i][j] = mat[i][j] * previous_growth[j]
        total_row = np.sum(new_mat, axis=1)

    # Normalize total_row with pred_row and total_column with pred_column

        GF_Revised = Oi_horizon / total_row
        # Check for convergence
        if all( value == 1.0 for value in GF_Revised):
            print("Convergence achieved after iterations:", iteration_count)
            break

        # Generate the DataFrame and prepare for the next iteration
        generate_Row(new_mat, total_row, GF_Revised,Oi_horizon)
    previous_growth = GF_Revised.copy()
    mat = new_mat  # Update mat for next iteration

return new_mat, total_row, total_column, growth
```

```
def generate_Row(new_mat, total_row, GF_Revised_row, pred_row):
    # Prepare the data dictionary for DataFrame creation
    data = {
        '1': [new_mat[i][0] for i in range(len(new_mat))],
        '2': [new_mat[i][1] for i in range(len(new_mat))],
        '3': [new_mat[i][2] for i in range(len(new_mat))],
        '4': [new_mat[i][3] for i in range(len(new_mat))],
        'Oi_current': total_row,
        'Oi_Horizon': pred_row,
        'Growth': GF_Revised_row
    }

    # Create DataFrame
    df = pd.DataFrame(data)

    # Set the index for the DataFrame
    df.index = [1, 2, 3,4,]

    # Round all numeric columns to 3 decimal places
    df = df.round(3)

    # Display the DataFrame
    print("\nTij_012:")
    print(df)
```

```
def generate_Column(new_mat, total_column, Oi_horizon, GF_Revised):
    # Prepare the data dictionary for DataFrame creation
    data = {
        '1': [new_mat[i][0] for i in range(len(new_mat))],
        '2': [new_mat[i][1] for i in range(len(new_mat))],
        '3': [new_mat[i][2] for i in range(len(new_mat))],
        '4': [new_mat[i][3] for i in range(len(new_mat))],
    }

    # Create DataFrame
    df = pd.DataFrame(data)

    # Replace None with 0 for total_column
    total_col = [x if x is not None else 0 for x in total_column]

    # Add additional rows for totals and growth factors
    df.loc['Dj_Current'] = [total_column[i] if i < len(total_column) else "" for i in range(df.shape[1])]
    df.loc['Dj_horizon'] = [Oi_horizon[i] if i < len(Oi_horizon) else "" for i in range(df.shape[1])]
    df.loc['G.F'] = [GF_Revised[i] if i < len(GF_Revised) else "" for i in range(df.shape[1])]

    # Set the index for the DataFrame
    df.index = [1, 2, 3,4, 'Dj_current', 'Dj_Horizon', 'GF']
```

```
    # Round all numeric columns to 3 decimal places
    df = df.round(3)

    # Display the DataFrame
    print("\nTij_012:")
    print(df)
```

```
file_path = 'Assignment_2.xlsx'
print("You have entered Question 5")
df = pd.read_excel(file_path, sheet_name='Furness',header=None)
print(df)
a = int(input("Enter the size of matrix:"))
Tij_012 = df.iloc[:a, :a].values.tolist()

print(Tij_012)
Oih_012 = df.values[:a,a+1].tolist()
print(Oih_012)
Uniform(Tij_012,Oih_012,a)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\gdred\Downloads\Kalyan\kalyan5.py", line 2, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

pdf_python.py

```
import os
import subprocess
from fpdf import FPDF
from pygments import highlight
from pygments.lexers import PythonLexer
from pygments.formatters import HtmlFormatter


class PDFWithBorder(FPDF):
    def header(self):
        self.set_font('Arial', 'B', 12)
        self.cell(0, 10, 'NITK Surathkal', 0, 0, 'L')
        self.cell(0, 10, '242CS017', 0, 1, 'R')
        self.ln(10)

    def footer(self):
        self.set_y(-15)
        self.set_font('Arial', 'I', 8)
        self.cell(0, 10, 'Dept of Civil CV742', 0, 0, 'L')
        self.cell(0, 10, 'CV742', 0, 0, 'R')

    def chapter_title(self, title):
```

```
        self.set_font('Arial', 'B', 12)
        self.cell(0, 10, title, 0, 1, 'C')
        self.ln(10)

    def chapter_body(self, body):
        self.set_font('Arial', '', 12)
        self.multi_cell(0, 10, body)
        self.ln()

    def add_code(self, title, code):
        self.add_page()
        self.chapter_title(title)
        self.chapter_body(code)


def generate_pdf_from_files(directory):
    pdf = PDFWithBorder()

    for root, dirs, files in os.walk(directory):
        if 'Assign' in dirs:
            dirs.remove('Assign')  # Ignore 'Assign' directory

        for file in files:
            if file.endswith('.py'):
```

```
            file_path = os.path.join(root, file)
            with open(file_path, 'r') as f:
                code = f.read()
                formatted_code = highlight(code, PythonLexer(), HtmlFormatter())
                pdf.add_code(file, code)

                # Generate output by running the Python file
                try:
                                                output = subprocess.check_output(['python', file_path],
stderr=subprocess.STDOUT).decode('utf-8')
                except subprocess.CalledProcessError as e:
                    output = e.output.decode('utf-8')

                pdf.chapter_body("Output:\n" + output)

    pdf_file_path = os.path.join(directory, 'output.pdf')
    pdf.output(pdf_file_path)
    print(f'PDF generated at {pdf_file_path}')


# Specify the directory containing the Python files
directory = './'
generate_pdf_from_files(directory)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\gdred\Downloads\Kalyan\pdf_python.py", line 3, in <module>
    from fpdf import FPDF
ModuleNotFoundError: No module named 'fpdf'
```

tempCodeRunnerFile.py

```
anda
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\gdred\Downloads\Kalyan\tempCodeRunnerFile.py", line 1, in <module>
    anda
NameError: name 'anda' is not defined
```