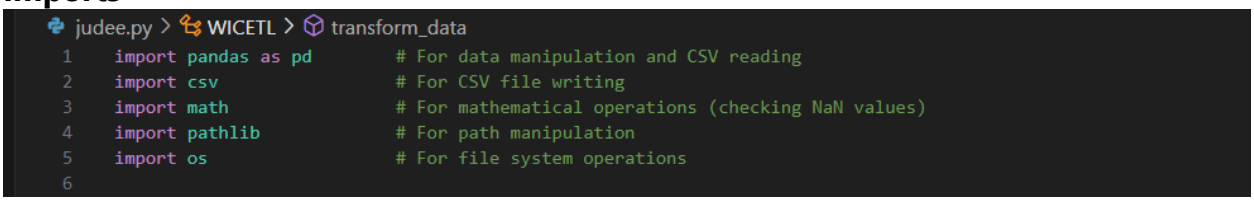
Name: Jude Anaman **Company:** Bracane Company Inc.

Github:

This program code defines a class `WICETL` designed to **extract, transform, and load (ETL)** data related to infants fully formula-fed from multiple CSV files spanning several years. The class reads data from CSV files, processes and aggregates quarterly data by state and year, appends metadata, and finally writes the transformed data into a new CSV file.

Imports



- `pandas` is used for reading CSV files into DataFrames.
- `csv` module is used for writing the transformed data back to CSV.
- `math.isnan()` helps check for missing values.
- `pathlib` and `os` help with file and directory path management.

Class Definition: `WICETL`

- **Class variables** are initialized to keep track of:
  - `column_year`: Starting year for data columns.
  - `transform`: Dictionary to hold aggregated totals across all states.
  - `data`: Dictionary to hold state-level aggregated data.
  - `path`: Directory path for output.
  - `fields`: CSV header fields.

Method: `transform_data`

```
def transform_data(self):
    for i in range(3, 7):
        df = pd.read_csv("C:/Users/Lenovo/Downloads/Openclassroom/JudeOpenclassroomProject/WICAgencies201" + str(i) + "ytd/Infants_Fully_Formula-fed.csv")
        columns = df.columns
        values = df.values.tolist()
        totalQ4 = 0
        totalQ1 = 0
        totalQ2 = 0
        totalQ3 = 0

    for value in values:
        stateQ4 = 0
        stateQ1 = 0
        stateQ2 = 0
        stateQ3 = 0
        state_column_year = self.column_year
        key = ""
        for i in range(0, 15):
            if i == 0:
                if self.data.get(value[i], None) == None:
                    self.data[value[i]] = {}
                key = value[i]
            if i > 0 and i <= 3:
                if math.isnan(value[i]) == False:
                    stateQ4 += value[i]
                    totalQ4 += value[i]
            if i > 3 and i <= 6:
                if math.isnan(value[i]) == False:
                    stateQ1 += value[i]
                    totalQ1 += value[i]
            if i > 6 and i <= 9:
                if math.isnan(value[i]) == False:
                    stateQ2 += value[i]
                    totalQ2 += value[i]
            if i > 9 and i <= 12:
                if math.isnan(value[i]) == False:
                    stateQ3 += value[i]
                    totalQ3 += value[i]
        self.data[key]["Q4 " + str(state_column_year)] = stateQ4
        state_column_year+=1
        self.data[key]["Q1 " + str(state_column_year)] = stateQ1
```

```
57                  self.data[key]["Q2 " + str(state_column_year)] = stateQ2
58                  self.data[key]["Q3 " + str(state_column_year)] = stateQ3
59                  self.transform["Q4 " + str(self.column_year)] = totalQ4
60                  self.column_year+=1
61                  self.transform["Q1 " + str(self.column_year)] = totalQ1
62                  self.transform["Q2 " + str(self.column_year)] = totalQ2
63                  self.transform["Q3 " + str(self.column_year)] = totalQ3
64
65
```

What it does:

- Loops over years 2013 to 2016 (since `i` goes from 3 to 6, appended to "`201`").
- Reads CSV files for each year.
- Converts the DataFrame to a list of lists (`values`) for processing.
- Initializes totals for each quarter (`Q1` to `Q4`).
- For each row (state data), sums quarterly values while checking for NaNs.
- Aggregates data by state and stores it in `self.data`.
- Aggregates totals across all states and stores in `self.transform`.
- Updates `column_year` to track the year for each quarter.

Important details:

- The code assumes the CSV files have a specific structure where columns 1-3 correspond to Q4, 4-6 to Q1, 7-9 to Q2, and 10-12 to Q3.
- The first column (`i == 0`) is assumed to be the state name.
- Uses `math.isnan()` to avoid adding missing values.
- The year increments after processing each file.

Method: `append_territory`

```
66      def append_territory(self):
67          for state in self.data:
68              self.data[state]["State"] = state
69
70          self.path = str(pathlib.Path().resolve())
71          self.fields = ["State"] + list(self.transform.keys())
72
```

- Adds a `"State"` key to each state's data dictionary for easier CSV writing.
- Sets the output path to the current working directory.
- Prepares the CSV header fields: `"State"` plus all keys from `self.transform` (which are quarterly totals).

Method: `write_file`

```
74      def write_file(self):
75          if os.path.exists(self.path + "/data") == False:
76              os.mkdir(self.path + "/data")
77
78
79          with open(self.path + "/data/" + "Formula_Fed-Totals.csv", 'w') as csvfile:
80              writer = csv.DictWriter(csvfile, fieldnames=self.fields)
81              writer.writeheader()
82
83          self.transform["State"] = "All States"
84          for state in self.data:
85              writer.writerow(self.data[state])
86
87          writer.writerow(self.transform)
88
```

- Checks if a `data` directory exists in the current path; creates it if not.
- Opens a CSV file for writing.
- Writes the header row.
- Writes each state's data as a row.
- Writes the aggregated totals row labeled `"All States"`.

Instantiation and Execution

```
89          wic_etl = WICETL()
90          wic_etl.transform_data()
91          wic_etl.append_territory()
92          wic_etl.write_file()
```

- Creates an instance of `WICETL`.
- Calls the methods in order to process and write the data.

Summary of Key Concepts and Patterns

- **ETL Pattern**: Extract (read CSV), Transform (aggregate data), Load (write CSV).
- **Data aggregation** by keys (states) and time periods (quarters and years).
- Use of **dictionaries** to store hierarchical data.
- Use of **pandas** for CSV reading and **csv.DictWriter** for writing.
- File and directory management with `os` and `pathlib`.