CMPS 340 Introduction to Data Science

Project Requirements

Dr. Ömer M. Soysal

Version 03242024

Requirements

- Modules
    - Use module_tmp.py
    - main.py
    - Config.py: Holds common configuration constants.
    - Parent class
    - Child class
    - Others as needed
- Functionalities
    - Read from a file
        - Csv file
        - Pickle file
    - Query
    - Calculate
    - Log progress and errors
    - Visualize
    - Export
        - Csv file
        - Pickle file
        - Picture file

Requirements

- Folders
  - Root
  - Input
  - Output
  - Lib
  - Doc
  - Others as needed
- Variable types
  - Immutable types: For read-only operations.
  - Mutable types: For objects to be updated.
- Checks for possible errors.
  - Log operations
  - Catch errors and continue running if not crucial.

Requirements

- Create two classes as a parent and child class.
- Parent class-1
  - Store configuration constants in a dictionary
  - Visualize data in each column using either
    - Histogram for distributions
    - Line plot to visualize numeric data
  - Query data for searching
    - A simple value (simple condition)

- Child class-1.1
  - Read data from a csv file.
    - Store into a dataframe.
  - Utilize configuration constants
  - Visualize distributions in each column using
    - Violin plot
    - Whisker-box plot
    - Scatter plot
  - Query data for searching and display
    - A set of numeric and string values
      - Using Boolean indexing

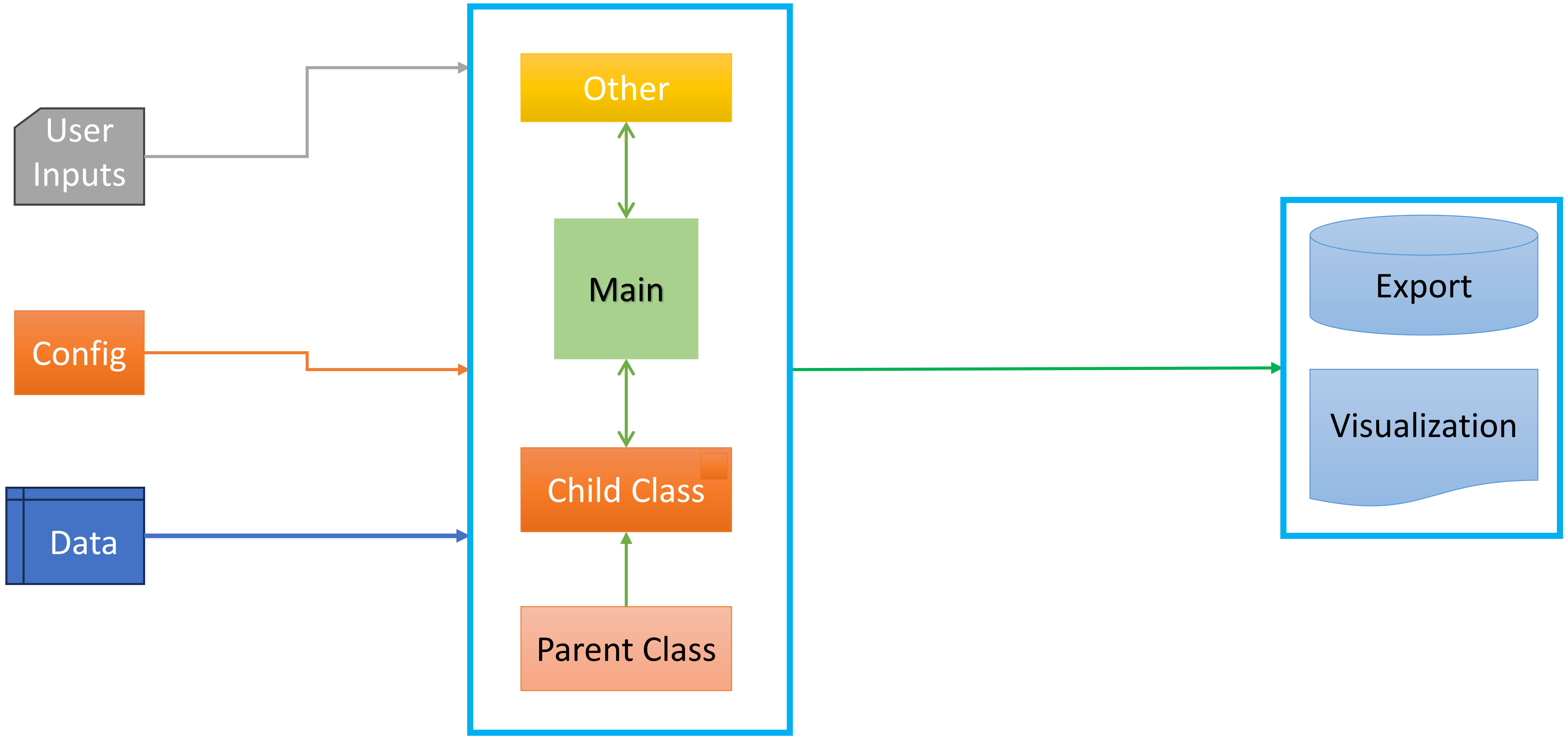
Requirements

- Create two classes as a parent and child class.
- Parent class-2
  - …. Your design ….
- Child class-2.1
  - Read data from a pickle file
  - Utilize configuration constants
  - Visualize
    - …
  - Probability: Calculate, display, export
    - joint counts
    - joint probabilities
    - conditional probabilities
    - mean, median, std
  - Vector operators: Display, export
    - Obtain position vector
    - Obtain unit vectors
    - Obtain projection vectors
    - Calculate the dot product
      - The angle between two vectors
      - Check for orthogonality
  - For a categorical attribute do the following and display
    - Obtain unique values
    - Generate permutations
    - Generate combinations

Requirements

- Utilize following variables types
  - Global
  - Nonlocal
  - Private like
- Utilize following type of function parameters
  - *arg
  - **kwarg
  - Default argument
- Utilize
  - Boolean indexing
  - Eval()
  - Lambda function
- Functionalities
  - Query DataFrame
  - Write an mxn Numpy array into a dataFrame

Module Communication Flow



Deliveries

- Project zip file with folders as instructed. Check your zip file before submission.
  - Project file name format: CS340_<S|F_YY>_<group name>.zip; S: Spring, F: Fall.
  - Do not include unnecessary files and folders such as "git", "_py_cache", etc.
- Report document
  - As a pdf file converted from PowerPoint document, in the Doc folder
  - Goal of the project
  - Module Communication Flow
  - Class diagrams
- Project manual
  - Input data format
  - Output data format
  - User manual
- Plots
- Task progress report
  - Use the template TaskProgresReport.xlsx.
  - Save in the Doc folder.
- GitHub URL
  - Use GitHub to manage the project.

Style

Coding Standards

- Write the code in aesthetically-pleasing style
- Names should be self-explanatory
  - "the main variable designator_variable group name":
  - "child_parent“; pm_single, not singlepm
  - dataDf_grpL_1 , not dataDf_grpL1; "_1" is safer for bugs.
- Comment adequately
  - Add a comment for each code block, such as a loop-block, that describe the functionality
  - Use relative path
  - Use generic coding instead of manually-entered constant values
- Plots
  - Legends should be good enough in color, line style, shape etc. to distinguish data series.
  - Properly name axes and title
- Add the symbol # at the end of EACH block
- Sort imports alphabetically
- Save data in categorized folders.
- Make code generic
- Use relative path

Performance and Safety

- Code must be efficient (data-structure, functionality).
- Avoid use of global variables. If needed, use cautiously.
  - Add the suffix "_gl" to global variables
  - "_ui" to the user interface variables
- Loops
  - Avoid if-block in a loop-block unless it is required.
  - Do not calculate a common/constant value inside a loop.
  - Avoid declarations in a loop-block unless it is required.
  - Avoid initializing variables inside a loop unless it is required.
- Initialization
  - Initialize objects with "None" (NOT zero) if their size is known instead of using append-like methods.
  - Use [None for i in Sequence] instead of [None]*len(Sequence)

Performance and Safety

- Import only the components from a package/module to be used instead of entire one.
- Prefer to use immutable types.
- Use deep-copy
- Operations with dataframe
  - Sort by the same column name, and then reset index. As an example,

```
grid_EntrpAll = x_trans.value_counts(subset=featureLst,normalize=True)
reset_index().sort_values(featureLst).reset_index()
```

- Utilize process logging
- Testing
  - Always test your code with an artificial data whose return value is known.