tars-consultor-plugin

October 9, 2023

1 Consultor

This Python script defines a class called Consultor, which is used to perform queries and data manipulation on a CSV file. Here is a high-level summary of the main functionalities and structure of the code:

1.1 Importing Libraries:

The script imports necessary libraries, including pandas for data handling in DataFrame format and json for JSON-related operations.

Class Consultor: It defines a class named Consultor used for querying and

[1]:
```
from pandas.core.frame import DataFrame
from pandas import read_csv

from json import dumps
```

1.2 Class Consultor:

It defines a class named Consultor used for querying and processing data in a CSV file. The class has the following methods

- **init**(self, path: str): The class constructor takes a CSV file path as an argument and stores it in the private attribute ___path.
- get_flat_row(self, header_val: str): This method takes a header value as an argument, looks up the header in the CSV file, and returns a JSON-format representation of the corresponding row, where keys are column headers, and values are row values.
- ___get_row_df(self, header_val: str): A private method that searches and returns a DataFrame containing the row with the specified header.
- get_flat_keys(self, *args): This method takes multiple headers as arguments, filters the CSV file to include only the specified columns, and returns a list of JSON representations of the corresponding rows. Each JSON representation contains the row header and a dictionary of column values.
- ___get_keys_df(self, *args): A private method that filters the DataFrame to include only the specified columns and returns the resulting DataFrame.

[2]:
```
class Consultor:

    def __init__(self,path: str) -> None:

        #file path
        self.__path = path


        pass

    def get_flat_row_group_by(self, header1 : str, header2 : str):

        df = self.__get_row_group_by(header1,header2)

        lst = df.values.tolist()
        lst = lst[0]

        return lst[2],lst[3]


        pass


    def __get_row_group_by(self,header1 : str, header2: str):
        df = read_csv(self.__path)
        return df.loc[(df[df.columns[0]] == header1) & (df[df.columns[1]] ==␣
↪header2)]

    def get_flat_row(self, header_val: str):
        df= self.__get_row_df(header_val)

        lst = df.values.tolist()[0]

        val_dict : dict = dict()
        t_dict : dict = dict()

        for index in range(1,len(lst)-1):
            val_dict[df.columns[index]] = lst[index]
            pass

        t_dict[df.columns[0]] = lst[0]
        t_dict["Values"] = val_dict


        return str(t_dict)

    def __get_row_df(self, header_val: str):
        df = read_csv(self.__path)
        return df[df.iloc[:, 0] == header_val]

    def get_flat_keys(self,*args):
        '''
            Filter a data frame in path

            Returns
            -------
            Data frame compose by headers passed as parameters
        '''
        # get filtered data frame
        df = self.__get_keys_df(*args)

        # init head values dictionary
        val_dict : dict = dict()
        # init total dictionary
        t_dict : dict = dict()

        # init return list (all data)
        rtn_lst : list = list()

        # for each row in data frame
        for row in df.values:
            # add key - value twice
            for index in range(1,len(row)):

                val_dict[args[index-1]] = row[index]

            # add header twice
            t_dict[df.columns[0]] = row[0]
            # add values dict
            t_dict["Values"] = val_dict

            # add values dict to list
            rtn_lst.append(t_dict)

            t_dict = dict()
            val_dict = dict()

        return str(rtn_lst)

    def __get_keys_df(self,*args):

        df : DataFrame = read_csv(self.__path)
        #add Planet header to filter
        t_args = [df.columns[0]]
        #add key words
        t_args.extend(args)

        #return filter data frame
        df = read_csv(self.__path)[t_args]
        return df
```