iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** IV **Month of publication:** April 2021

DOI: https://doi.org/10.22214/ijraset.2021.33872

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue IV Apr 2021- Available at www.ijraset.com

Data Cleansing on a CSV File using Python

Varun S Kulkarni[1], Shivakumara K[2], Praveenkumar A[3], Asst. Prof. Nagraj M[4]
[1, 2, 3, 4]*Computer Science Department, Visveswaraya Technological University, Belagavi, Karnataka, India.*

Abstract: Data cleansing is process of identifying and removing incorrect or corrupt data from dataset. Data is often collected (from an API of format JSON/XML) and stored in an appropriate format (commonly in CSV format). This data may contain wrong/corrupt information (information which do not satisfy the constraint). This uncertain data must be taken care before passing it to any algorithms. There are third-party software's which perform the above operations but it comes with cost and learning curve. One can implement his own algorithm which is idealistic and can be improved with time. Main advantage of implementing own algorithm is optimization to one's specific requirement.

Keywords: Data Cleansing, CSV.

I. INTRODUCTION

Data cleansing is process of cleaning the collected data. It is the primary action which is to be performed once data is collected. This step cannot be skipped as it may lead to wrong data interpretation. Algorithms can be efficient (time/complexity), but wrong data cannot yield correct result. Hence data cleansing is important (cannot be skipped/ignored).

In today's computer technology Python programming language is widely used to solve challenging problems in areas such as web scrapping, network programming, data science, machine learning, artificial intelligence and many more. Python computer language is more readable and there is less learning curve. This is the reason industries are shifting to Python.

In this context we are using Python computer language to implement the algorithm which performs data cleansing. We mainly operate on CSV (Comma Separated Value) files because data is primarily stored in CSV format and then fed to algorithms when required.

II. PROBLEM STATEMENT

CSV file contains blank lines, incorrect data-type element etc. We need to remove these inconsistent data from data set. The problem can demand solution in case of data-type incompatibility, negative value found in place of positive and so-on. These problems need to be addressed before feeding data-set to any algorithm.

Broad classification of problems is,

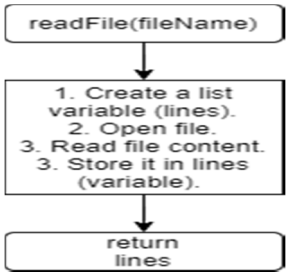
A. Removal blank lines.
B. Check constraints.

III.ALGORITHMIC APPROACH

A. Function to read a CSV file.

readFile function takes filename as an argument and returns lines after reading from file.

```
def readFile(fileName):
    actualLines = list()
    with open(fileName, "r") as csvFile:
        actualLines = list(csv.reader(csvFile))
    return actualLines
```

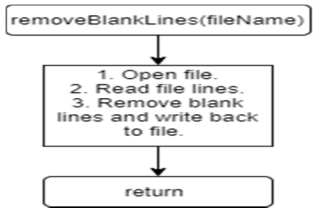


Complexity: O(1)

B. Function to write a CSV file.

writeFile function takes filename and content as arguments and write content onto file.

```
def writeFile(fileName, content):
    with open(fileName, "w") as csvFile:
        for line in content:
            temp = [str(ele).strip() for ele in line]
            csvLine = ','.join(temp) + '\n'
            csvFile.write(csvLine)
```



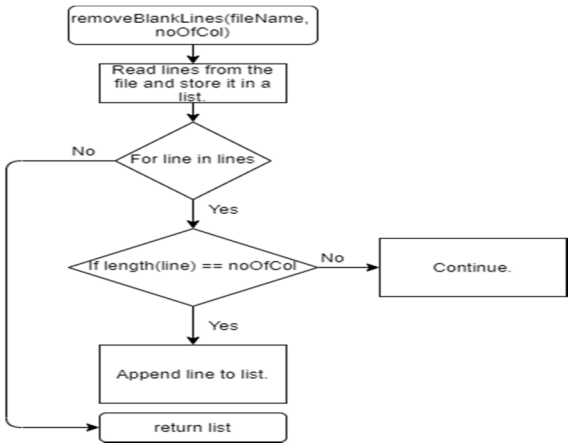
Complexity: O(n), where n is number of lines in CSV file.

C. Function to Remove Blank lines from CSV file

Arguments: fileName(name of the file), noOfCol(number of columns).

This method performs read and write operation on the same file. It checks for number of columns present via length of list (list conatins lines which are read from csv.reader() method and stored in list(actualLines)). If length of list is less than expected number of columns, it ignores the line. Lines which satisfy column constraint is saved in list(finalLines).

```
def removeBlankLines(fileName, noOfCol):
    actualLines = readFile(fileName)
    # Lines to be placed in csv file.
    finalLines = list()
    for line in actualLines:
        # Check number of columns,
        # if number of columns is equal to length
            # of line(list) append it to final lines.
        if len(line) == noOfCol:
            finalLines.append(line)
    writeFile(fileName, finalLines)
```



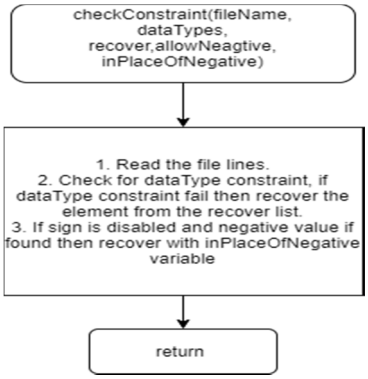
Complexity: O(n), where n is number of lines in CSV file.

D. Function To Check Constraints

Arguments: fileName (name of the file), dataTypes (type of data), recover, allowNegative (boolean value), inPlaceOfNegative (replacement for negative variable).

This function reads the file content and checks for data-type constraint as well as sign constraint. If constraint is not met then it performs a replacement on that particular element. It uses try-except block to explicitly convert element to mentioned data-type int the datatype variable. If any exception occurs, then it recover it by mentioned value in recover variable.

```
def checkConstraint(fileName, dataTypes, recover, allowNegative, inPlaceOfNegative):
    lines = readFile(fileName)
    for line in lines[1::]:
        for i in range(len(line)):
            try:
                # Try explict type conversion to check constraint.
                ele = dataTypes[i](line[i])
                if dataTypes[i] == str:
                    checkStrLength = lambda string: len(string.strip())
                    if checkStrLength(ele) == 0:
```



```
                    ele = recover[i]
                # Check sign constraint.
                if not allowNegative:
                    # Recover sign constraint.
                    ele = inPlaceOfNegative
                line[i] = ele
            except:
                # Recover data-type constraint.
                line[i] = recover[i]
        # Write final lines.
        writeFile(fileName, lines)
```

Complexity: O(n+m), where n is number of lines in CSV file and m is number of columns.

Complete implementation of the problem in object-oriented model.

```
# Implementation of cleansing using python
import csv
class Cleansing:
    # Initialization
    def __init__(self, fileNames:list(), dataTypes:list(), recover:list(), allowNegative = True, inPlaceOfNegative = "NA"):
        self.fileNames = fileNames
        self.dataTypes = dataTypes
        self.recover = recover
        # Flag for allowing sign numbers.
        self.allowNegative = allowNegative
        # If -ve value is found but -ve value is not allowed, then we need to place recover value in place of the current data.
        self.inPlaceOfNegative = inPlaceOfNegative
        for file in self.fileNames:
            # Remove blank lines.
            self.removeBlankLines(file, len(self.dataTypes))
            # Check constraints.
            self.checkConstraint(file, self.dataTypes, self.recover, self.allowNegative, self.inPlaceOfNegative)

    # Read file content into list and return it.
    def readFile(self, fileName):
        actualLines = list()
        with open(fileName, "r") as csvFile:
            actualLines = list(csv.reader(csvFile))
        return actualLines

    # Write content (list) into file.
    def writeFile(self, fileName, content):
        with open(fileName, "w") as csvFile:
            for line in content:
                temp = [str(ele).strip() for ele in line]
                csvLine = ','.join(temp) + '\n'
                csvFile.write(csvLine)
```

```
    # Lines from csv file.
    def removeBlankLines(self, fileName, noOfCol):
        actualLines = self.readFile(fileName)
        # Lines to be placed in csv file.
        finalLines = list()
        for line in actualLines:
            # Check number of columns,
            # if number of columns is equal to length of line(list) append it to final lines.
            if len(line) == noOfCol:
                finalLines.append(line)
        self.writeFile(fileName, finalLines)

    # Get lines from csv file.
    def checkConstraint(self, fileName, dataTypes, recover, allowNegative, inPlaceOfNegative):
        lines = self.readFile(fileName)
        for line in lines[1::]:
            for i in range(len(line)):
                try:
                    # Try explict type conversion to check constraint.
                    ele = dataTypes[i](line[i])
                    if dataTypes[i] == str:
                        checkStrLength = lambda string: len(string.strip())
                        if checkStrLength(ele) == 0:
                            ele = recover[i]
                    # Check sign constraint.
                    if not allowNegative:
                        # Recover sign constraint.
                        ele = inPlaceOfNegative
                    line[i] = ele
                except:
                    # Recover data-type constraint.
                    line[i] = recover[i]
        # Write final lines.
        self.writeFile(fileName, lines)
```

IV. DRY RUN

A. CSV input file: sports.csv

```
Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport
sunny,23,high,strong,cold,same,yes
rainy,34,high,strong,warm,same,yes
sunny,33,high,strong,cold,change,no
sunny,31,low,weak,cold,same,yes
rainy,30,low,weak,warm,change,no
sunny,29,,,cold,,
sunny,,low,strong,,change,yes
rainy,30,high,strong,cold,same,no
sunny,33,low,strong,warm,same,yes
rainy,33,high,weak,cold,change,no
sunny,34,low,strong,,,no
```

B. Now create instance of the Cleansing class,

```
obj = Cleansing(["D:/7_8_SEM_PROJECT/sports.csv"], [str, int, str, str, str, str, str], ["NA", "NA", "NA", "NA", "NA", "NA", "NA"])
```

C. Cleansed CSV file: sports.csv

```
Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport
sunny,23,high,strong,cold,same,yes
rainy,34,high,strong,warm,same,yes
sunny,33,high,strong,cold,change,no
sunny,31,low,weak,cold,same,yes
rainy,30,low,weak,warm,change,no
sunny,29,NA,NA,cold,NA,NA
sunny,NA,low,strong,NA,change,yes
rainy,30,high,strong,cold,same,no
sunny,33,low,strong,warm,same,yes
rainy,33,high,weak,cold,change,no
sunny,34,low,strong,NA,NA,no
```

V. CONCLUSION

Hence, we need to implement data cleansing before any implementation of analysis algorithms in any data analysis. The code can be generalized and improved over time. Implementation of code in object-oriented model enhances its usability and finally makes it efficient.

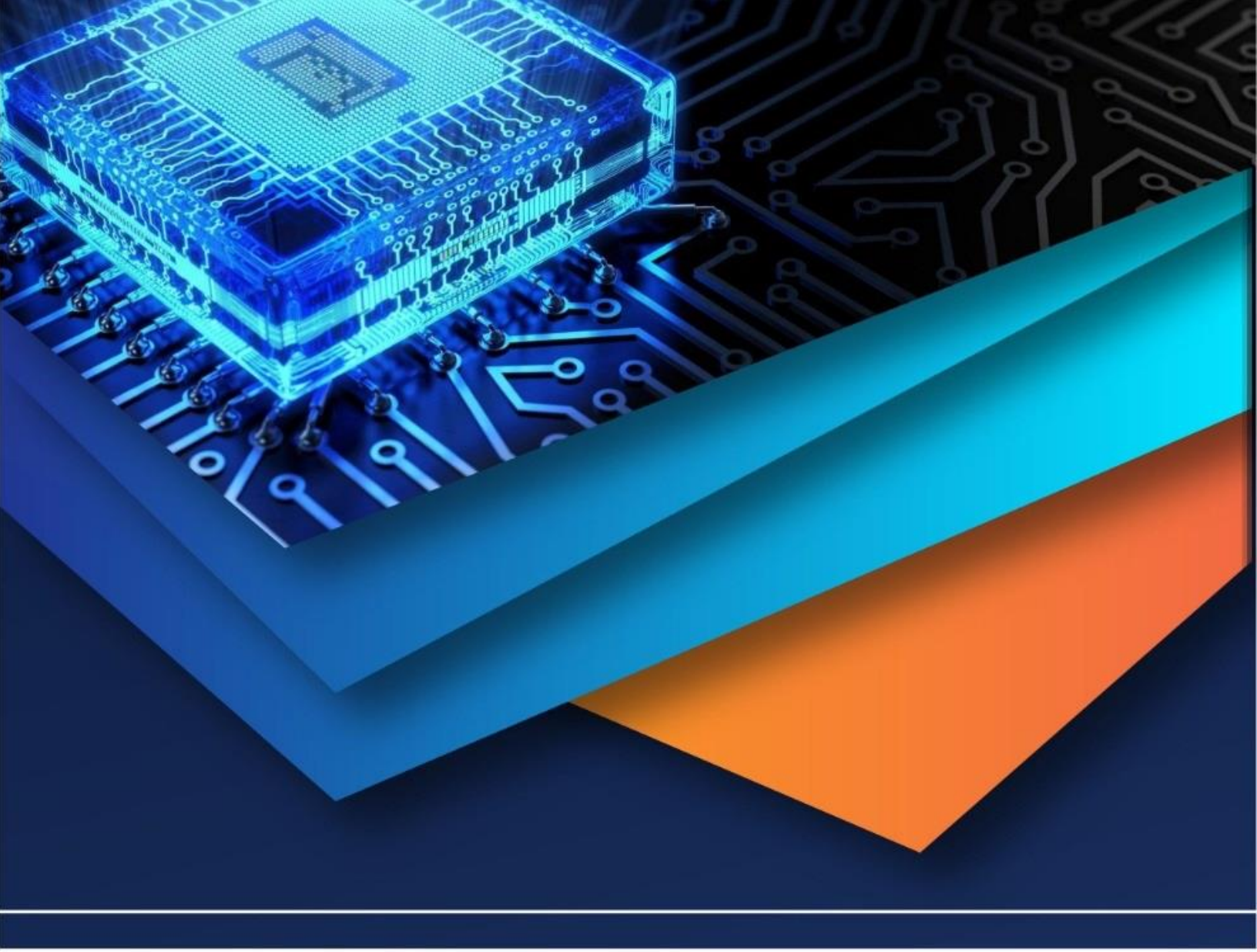
The above-mentioned code handles the blank lines, data type constraints and sign constraints of the data set (CSV file).

VI. ACKNOWLEDGMENT

We specially thank Asst.Professor M. Nagaraj for the assistance and continuous support. We also thank our project coordinator Asst.Professor D.V. Swetha Ramana and Asst.Professor Roshan D for their support and guidance.

REFERENCES

[1] Machine Learning Tom M. Mitchell.
[2] Think Python Second Edition Allen B. Downey
[3] Python for Everybody: Exploring Data in Python by Dr. Charles Russell Severance (Author)
[4] Wu, S. (2013), "A review on coarse warranty data and analysis" (PDF), Reliability Engineering and System, 114: 1–11, doi:10.1016/j.ress.2012.12.021
[5] "Data 101: What is Data Harmonization?". Datorama. 14 April 2017. Retrieved 14 August 2019.
[6] Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., Becker, B. The Data Warehouse Lifecycle Toolkit, Wiley Publishing, Inc., 2008. ISBN 978-0-470-14977-5
[7] Olson, J. E. Data Quality: The Accuracy Dimension", Morgan Kaufmann, 2002. ISBN 1-55860-891-5



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY