PROJECT

KDS SAS PYTHON
Functional Design Document

Version 1.0

November 13, 2024

Version History

Date	Version #	Description	Author
12/5/2024	1.0a	Final Version	Vincent Taylor

TABLE OF CONTENTS

Introduction

This work request was submitted by XXXXX

1.1 Objectives

- Convert three SAS macro programs to Python source code.
- Produce linear and regression models
- Deliver excel spreadsheets to appropriate output folder

1.2 System Description

System/Application Name	Customer Service Experiences
Sr, Vice President	
Executive Vice President	
Contractor	Vincent Taylor

1.3 Scope and Process

Design and execute a plan to convert three SAS macro programs to Python programs using Code Convert AI code translator. Code Convert AI code converter translates source code in hunks of roughly two hundred lines, once converted can be downloaded to .txt files. The converted source code can then be assembled into a single .ipynb extension files.

- Convert AHM, ATTImpactV29 and LNImpactV1c SAS Macro programs to Python.
- Download the converted python code to .txt files.
- Assemble .txt files into three .ipynb programs AHM, ATTImpactV29 and LNImpactV1c
- Create the KDA_DRIVER_VARIABLE.xlsx spreadsheet which will house up to three sheets containing parameter variables and values.
- Write the Data Structure Python program which reads as input KDA_DRIVER_VARIABLE.xlsx then generates the JSON joutput.txt file which will be used by the AHM, ATTImpactV29 and LNImpactV1c Python programs.
- Modify and test AHM, ATTImpactV29 and LNImpactV1c Python programs. This is where most of the project time will be utilized.
- Review models, excels content generated by ATTImpactV29 and LNImpactV1c programs.
- Create a script to execute the AHM Python programs which will invoke ATTImpactV29 and LNImpactV1c.

1.4 Issues

The Code Convert AI code translator will not translate the SAS macro code with one hundred percent accuracy; therefore, the converted Python code must be reviewed line by line in most cases when testing to determine the degree of translation accuracy.

1.5 Dependencies

N/A.

2 Execution

2.1 Execution and testing of the converted Python programs will be performed using Google Cloud Colab Enterprise.

3 Environment

3.1 Tools

Google Cloud Colab Enterprise

3.2 Input and Output Datasets

Datasets:

Note: SUBSEG will equal some number. {x} will translate to a number

Datasets:

Input Datasets	Output Datasets
/content/AMH_W1_15.sav	/content/joutput.txt
gs://xxxx-cxusanalytics-dev-kda_01_test_information/Input/KDA_DRIVER_VARIABLES.xlsx	gs://xxxx-cxusanalytics-dev-kda_01_test_information/output/total-attributes.xls
	gs://xxxx-cxusanalytics-dev-kda_01_test_information/output/total1-attributes.xls
	gs://xxxx-cxusanalytics-dev-kda_01_test_information/output/total2-attributes.xls
	gs://xxxx-cxusanalytics-dev-kda_01_test_information/output/total3-attributes.xls
	gs://xxxx-cxusanalytics-dev-kda_01_test_information/output/total3-attributes_subseg{x}.xls

3.3 Special Test Needs

Refer to Testing Case Selection

4 Release Criteria

- System testing, independent testing and user acceptance testing must be carried out successfully
- Testing of all requirements must be completed

Data_Structure_Test	Completed
AHM_Test	Completed
ATTImpactv29_Test	Completed
ATTImpactv29	Fully converted code
LNImpactV1c	Fully converted code

User requirements and business rules were not available for this project as a result the Python code reflect the results of the Code Converted AI application and the developer's interpretation of the original SAS code. Data_Structure and AHM python code will be shown in this document as sizes are below 2K. Python programs will be made available in a zip folder given to Steven Ford; he will also have access to the developer's Google Colab Environment.

5 Python Source Code

DATA_STRUCTURE

```
#-------------------------------------------------------------------------------
# Program: Data_Structure
#
# Author: Vincent Taylor
#
# Creation Date: 10-2024
#
# Last Modified: 10-2024
#
# Purpose: This program reads in as input the KDA_DRIVER_VARIABLES.xlsx file
#              which houses the KDA variables and values which will be processed and used to
```

```
#              generate the JSON joutput.txt file which will be used by Python programs
#              AHM, ATTImpactV29 and LNImpactV1c
#-------------------------------------------------------------------------------

# Input File(s): gs://xxxx-cxusanalytics-dev-kda_01_test_information/Input/KDA_DRIVER_VARIABLES.xlsx
#
# Output Fies(s): /content/joutput.txt
#
# Change Version: N/A
#-------------------------------------------------------------------------------

import pandas as pd
import numpy as np
import tempfile
import sys
import json

data = {"table": " ", "weight": " ", "increase": " ", "increase1": " ", "increase2": " ", "increase3": " ", "increase4": " ",
        "increase5": " ", "seg": " ", "high": " ", "low": " ", "meanv": " ", "lscale": " ", "lscale2": " ", "lscale3": " ", "lscale4": " ",
        "lscale5": " ", "hscale": " ", "hscale2": " ", "hscale3": " ", "hscale4": " ", "hscale5": " ", "sumi": " ", "sums": " ",
        "detv1": " ", "detv2": " ", "detv3": " ", "detv4": " ", "detv5": " ", "detv6": " ", "detv7": " ", "detv8": " ", "detv9": " ",
        "detv10": " ", "detv11": " ", "detv12": " ", "detv13": " ", "detv14": " ", "detv15": " ", "detv16": " ", "detv17": " ",
        "detv18": " ", "detv19": " ", "detv20": " ", "detv21": " ", "detv22": " ", "detv23": " ", "detv24": " ", "detv25": " ",
        "detv26": " ", "detv27": " ", "detv28": " ", "detv29": " ", "detv30": " ", "detv31": " ", "detv32": " ", "detv33": " ",
        "detv34": " ", "detv35": " ", "detv36": " ", "detv37": " ", "detv38": " ", "detv39": " ", "detv40": " ", "detv41": " ",
        "detv42": " ", "detv43": " ", "detv44": " ", "detv45": " ", "detv46": " ", "detv47": " ", "detv48": " ", "detv49": " ",
        "detv50": " ", "dir": " ", "hcut": " ", "hcut2": " ", "hcut3": " ", "hcut4": " ", "hcut5": " ", "lcut": " ", "runs": " ",
        "subseg": " ", "sc1var": " ", "sc2var": " ", "sc3var": " ", "sc4var": " ", "sc5var": " ", "attdirect": " ", "sumdirect": " ",
        "attbysum": " ", "attontosum": " ", "tbinc": " ", "segment": " ", "noreg": " ", "format": " ", "lcut2": " ", "lcut3": " ",
        "lcut4": " ", "lcut5": " ", "cut1var": " ", "cut2var": " ", "cut3var": " ", "cut4var": " ", "cut5var": " ", "inc1var": ' ',
        "inc2var": " ", "inc3var": " ", "inc4var": " ", "inc5var": " ", "inc6var": " ", "inc7var": " ", "inc8var": " ", "inc9var": " ",
        "inc10var": " ", "inc11var": " ", "inc12var": " ", "inc13var": " ", "inc14var": " ", "inc15var": " ", "inc16var": " ",
        "inc17var": " ", "inc18var": " ", "inc19var": " ", "inc20var": " ", "inc21var": " ", "inc22var": " ", "inc23var": " ",
        "inc24var": " ", "inc25var": " ", "inc26var": " ", "inc27var": " ", "inc28var": " ", "inc29var": " ", "inc30var": " ",
        "inc31var": " ", "inc32var": " ", "inc33var": " ", "inc34var": " ", "inc35var": " ", "inc36var": " ", "inc37var": " ",
        "inc38var": " ", "inc39var": " ", "inc40var": " ", "inc41var": " ", "inc42var": " ", "inc43var": " ", "inc44var": " ",
        "inc45var": " ", "inc46var": " ", "inc47var": " ", "inc48var": " ", "inc49var": " ", "inc50var": " ", "regsplit": " ",
        "sigacross": " ", "siggroup": " ", "nosim": " ", "drivers": " ", "sumv1": ' ', "sumv2": " ", "sumv3": " ", "sumv4": " ",
        "sumv5": " ", "sumv6": " ", "sumv7": " ", "sumv8": " ", "sumv9": " ", "sumv10": " ", "sumv11": " ", "sumv12": " ",
        "sumv13": " ", "sumv14": " ", "sumv15": " ", "sumv16": " ", "sumv17": " ", "sumv18": " ", "sumv19": " ", "sumv20": " ",
        "sumv21": " ", "sumv22": " ", "sumv23": " ", "sumv24": " ", "sumv25": " ", "sumv26": " ", "sumv27": " ", "sumv28": " ",
        "sumv29": " ", "sumv30": " ", "sumv31": " ", "sumv32": " ", "sumv33": " ", "sumv34": " ", "sumv35": " ", "sumv36": " ",
        "sumv37": " ", "sumv38": " ", "sumv39": " ", "sumv40": " ", "sumv41": " ", "sumv42": " ", "sumv43": " ", "sumv44": " ",
        "sumv45": " ", "sumv46": " ", "sumv47": " ", "sumv48": " ", "sumv49": " ", "sumv50": " "
    }

#-------------------------------------------------------------------------
# The following variables wer created to resolve the NameError:  not defined
# error when Data_Structure is invoked
#-------------------------------------------------------------------------
global true, null, false
true = True
null = None
false = False

global file_path, sheet

pd.options.mode.copy_on_write = True # 10-25-24
```

```
#-------------------------------------------------------------------------------------------
# Houses parm variables which must be set. There are three sheets allowing for up to three
# process, meaning three python modules may run in parellel.
# IMPORTANT: If any Key:Pair value is not set with data set it to null in row / column. If the program
#            files due a dataType error manually set the format in the excel for the entire row to General.
#-------------------------------------------------------------------------------------------
file_path = "gs://xxxx-cxusanalytics-dev-kda_01_test_information/Input/KDA_DRIVER_VARIABLES.xlsx"
sheet = 'Sheet1'

def convert_single_item_list_to_string(lst):
  if len(lst) == 1:
    return str(lst[0]).replace("[","").replace("]", "").replace("'", "").replace("'", "").replace(", ", "")
  else:
    return lst #return the list unchanged if it doesn't have exactly one item

def process_excel_from_stdin(func, data_structure):
  """
  This function reads the Excel file processes the Data_Structure,
  creates a temporary file and then writes data from Data_Structure to sys.stdout
    """

  # Create a temporary file to store the data
  with tempfile.NamedTemporaryFile(suffix=".xlsx",delete=False) as temp_file:
       temp_file.write(sys.stdin.buffer.read())
       temp_file.flush

       return func(temp_file.name,data_structure)

       # Close the temporary file
       os.close(temp_file())

  return #process_excel_from_stin

def read_excel_data(path, data_structure):
    """
    Read data from an Excel file and return a Panda Data
    """
    df = pd.read_excel (file_path, sheet_name=sheet)

    df = df.replace(r'^\s*$', np.nan, regex=True)

    # Check if the first column has a value and all other are NaN
    condition = (df.iloc[:, 1:].isnull().all(axis=1)) & (df.iloc[:, 0].notnull())

    # Drop the rows that meet the above condtion
    df = df[~condition]

    #-------------------------------------------------------------------
    # Iternate (Read) through the dataframe, if the a name in the TYPE
    # column of the excel match a Key name on the dictionary populate
    # the dictionary pair value with the excel data.
    #-------------------------------------------------------------------
    for index, row in df.iterrows():
      name = row.iloc[0] # Name is in the first column

      if name in data_structure:
         for col_name, value in row.items():
```

```
            if col_name != 0 and pd.notna(value):
                data.update({name:value})

    return data_structure

def flag_sumv(path, data_structure):
    """
    This section of logic handles creation of the sumv variable which will set
    the subs NA flag variable
    """
    test_strings = []
    strings = []

    # Read data from an Excel file and return a Panda Data
    df = pd.read_excel(file_path, sheet_name=sheet)

    for i in range(1, 51):
        istring = str(i)
        row = df[df['TYPE'] == 'sumv'+istring]

        # Delete TYPE column
        del row['TYPE']

        row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

        row = row.replace(' ', "")  # replace single space with null
        row = row.replace('  ', "") # replace double space with null
        row = row.replace("'","")   # replace single quote with null

        test_strings = row.values.tolist() # Convert dataframe to list

        # Elements to remove
        to_remove = ["'"]

        suv = 'sumv'+istring

        # Using filter to remove elements
        strings = list(filter(lambda x: x not in to_remove, test_strings))

        globals()[f'sumv{i}'] = list(filter(None, strings))

        sumv_value = globals()[f'sumv{i}']

        # Convert list into a string
        sumv_str = convert_single_item_list_to_string(sumv_value)

        # Update the Dictionary
        data.update({suv:sumv_str})

        return # flag_sumv

def flag_sumi(path, data_structure):
    """
     This section of logic handles creation of the sumi variable which will set
     the isubs NA flag variable
    """

    test_strings = []
```

```
    strings = []

    # Read data from an Excel file and return a Panda Data
    df = pd.read_excel(path, sheet_name=sheet)

    # #he variable SUMI Has to be handled differently it does not conform to Key pairing
    row = df[df['TYPE'] == 'sumi']

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("'","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["'"]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    sumi = list(filter(None, strings))

    # Update the Dictionary
    data.update({'sumi':sumi})

    return # flag_sumi

def flag_sums(path, data_structure):
    """
     This section of logic handles creation of the sums variable which will set
     the means frequencies variables
    """

    test_strings = []
    strings = []

    # Read data from an Excel file and return a Panda Data
    df = pd.read_excel(path, sheet_name=sheet)

    # #he variable SUMS Has to be handled differently it does not conform to Key pairing
    row = df[df['TYPE'] == 'sums']

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("'","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list
```

```
    # Elements to remove
    to_remove = [""]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    sums = list(filter(None, strings))

    # Update the Dictionary
    data.update({'sums':sums})

    return # flag_sums

def flag_drivers(path, data_structure):
    """
     This section of logic handles creation of the drivers variable which will set
     the drivers variable
    """

    test_strings = []
    strings = []

    # Read data from an Excel file and return a Panda Data
    df = pd.read_excel(file_path, sheet_name=sheet)

    # the variable DRIVERS Has to be handled differently it does not conform to Key pairing
    row = df[df['TYPE'] == 'drivers']

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("'","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = [""]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    drivers = list(filter(None, strings))

    # Update the Dictionary
    data.update({'drivers':drivers})

    return # flag_driver

def flag_detv(path, data_structure):
    """
     This section of logic handles creation of the drivers variable which will set
     the detv variables
    """
```

```
    test_strings = []
    strings = []

    # Read data from an Excel file and return a Panda Data
    df = pd.read_excel(file_path, sheet_name=sheet)

    for i in range(1, 51):
        istring = str(i)
        row = df[df['TYPE'] == 'detv'+istring]

        # Delete TYPE column
        del row['TYPE']

        row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

        row = row.replace(' ', "")  # replace single space with null
        row = row.replace('  ', "") # replace double space with null
        row = row.replace("""","")   # replace single quote with null

        test_strings = row.values.tolist() # Convert dataframe to list

        # Elements to remove
        to_remove = ["""]

        dev = 'detv'+istring

        # Using filter to remove elements
        strings = list(filter(lambda x: x not in to_remove, test_strings))

        globals()[f'detv{i}'] = list(filter(None, strings))

        detv_value = globals()[f'detv{i}']

        # Convert list into a string
        detv_str = convert_single_item_list_to_string(detv_value)

        # Update the Dictionary
        data.update({dev:detv_str})

        return # flag_detv

def all_others(path, data_structure):
    """
     This section of logic handles creation of the all other variables which will set
     the remaining variables
    """

    test_strings = []
    strings = []

    # Read data from an Excel file and return a Panda Data55
    df = pd.read_excel(file_path, sheet_name=sheet)

    # the variable Table Has to be handled differently it does not conform to Key pairing
    row = df[df['TYPE'] == 'table']

    # Delete TYPE column
    del row['TYPE']
```

```
row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

table = list(filter(None, strings))

# Convert list into a string
table_str = convert_single_item_list_to_string(table)

# Update the Dictionary
data.update({'table':table_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'weight']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

weight = list(filter(None, strings))

# Convert list into a string
weight_str = convert_single_item_list_to_string(weight)

# Update the Dictionary
data.update({'weight':weight_str})

# the variable lscale Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'increase']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank
```

```
row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

increase = list(filter(None, strings))

# Convert list into a string
increase_str = convert_single_item_list_to_string(increase)

# Update the Dictionary
data.update({'increase':increase_str})

for i in range(2, 6):
    istring = str(i)

    row = df[df['TYPE'] == 'increase'+istring]

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("'","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["'"]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    increase = 'increase'+istring
    globals()[f'increase{i}'] = list(filter(None, strings))

    increase_value = globals()[f'increase{i}']

    # Convert list into a string
    increase_str = convert_single_item_list_to_string(increase_value)

    # Update the Dictionary
    data.update({increase:increase_str})

    # the variable Seg Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'seg']

# Delete TYPE column
```

```
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

seg = list(filter(None, strings))

# Convert list into a string
seg_str = convert_single_item_list_to_string(seg)

# Update the Dictionary
data.update({'seg':seg_str})

# the variable Low Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'high']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

high = list(filter(None, strings))

high_str = convert_single_item_list_to_string(high)

# Update the Dictionary
data.update({'high':high_str})

# the variable Low Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'low']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank
```

```
row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

low = list(filter(None, strings))

low_str = convert_single_item_list_to_string(low)

# Update the Dictionary
data.update({'low':low_str})

# the variable lscale Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'lscale']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

lscale = list(filter(None, strings))

# Convert list into a string
lscale_str = convert_single_item_list_to_string(lscale)

# Update the Dictionary
data.update({'lscale':lscale_str})

for i in range(2, 6):
    istring = str(i)

    row = df[df['TYPE'] == 'lscale'+istring]

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank
```

```
    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("""","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = [""]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    lscale = 'lscale'+istring

    globals()[f'lscale{i}'] = list(filter(None, strings))

    lscale_value = globals()[f'lscale{i}']

    # Convert list into a string
    lscale_str = convert_single_item_list_to_string(lscale_value)

    # Update the Dictionary
    data.update({lscale:lscale_str})

# the variable Lscale2 Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'hscale']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = [""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

hscale = list(filter(None, strings))

# Convert list into a string
hscale_str = convert_single_item_list_to_string(hscale)

# Update the Dictionary
data.update({'hscale':hscale_str})

for i in range(2, 6):
    istring = str(i)
    row = df[df['TYPE'] == 'hscale'+istring]

    # Delete TYPE column
    del row['TYPE']
```

```
    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("'","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["'"]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    hscale = 'hscale'+istring

    globals()[f'hscale{i}'] = list(filter(None, strings))

    hscale_value = globals()[f'hscale{i}']

    # Convert list into a string
    hscale_str = convert_single_item_list_to_string(hscale_value)

    # Update the Dictionary
    data.update({hscale:hscale_str})

# the variable hcut Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'hcut']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

hcut = list(filter(None, strings))

hcut_str = convert_single_item_list_to_string(hcut)

# Update the Dictionary
data.update({'hcut':hcut_str})

for i in range(2, 6):
    istring = str(i)
    row = df[df['TYPE'] == 'hcut'+istring]
```

```
    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("""","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = [""]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    hcut = 'hcut'+istring

    globals()[f'hcut{i}'] = list(filter(None, strings))

    hcut_value = globals()[f'hcut{i}']

    # Convert list into a string
    hcut_str = convert_single_item_list_to_string(hcut_value)

    # Update the Dictionary
    data.update({hcut:hcut_str})

# the variable hcut Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'lcut']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = [""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

lcut = list(filter(None, strings))

lcut_str = convert_single_item_list_to_string(lcut)

# Update the Dictionary
data.update({'lcut':lcut_str})

for i in range(2, 6):
    istring = str(i)
```

```
    row = df[df['TYPE'] == 'lcut'+istring]

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("""","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["""]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    lcut = 'lcut'+istring

    globals()[f'lcut{i}'] = list(filter(None, strings))

    lcut_value = globals()[f'lcut{i}']

    # Convert list into a string
    lcut_str = convert_single_item_list_to_string(lcut_value)

    # Update the Dictionary
    data.update({lcut:lcut_str})

# the variable hcut Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'runs']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

runs = list(filter(None, strings))

# Convert list into a string
runs_str = convert_single_item_list_to_string(runs)

# Update the Dictionary
data.update({'runs':runs_str})
```

```
for i in range(1, 5):
    istring = str(i)
    row = df[df['TYPE'] == 'cut'+istring+'var']

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("""","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["""]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    cut = 'cut'+istring+'var'

    globals()[f'cut{i}'+'var'] = list(filter(None, strings))

    cut_value = globals()[f'cut{i}'+'var']

    # Convert list into a string
    cut_str = convert_single_item_list_to_string(cut_value)

    # Update the Dictionary
    data.update({cut:cut_str})

for i in range(1, 6):
    istring = str(i)
    row = df[df['TYPE'] == 'sc'+istring+'var']

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("""","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["""]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    sc = 'sc'+istring+'var'

    globals()[f'sc{i}'+'var'] = list(filter(None, strings))
```

```
    sc_value = globals()[f'sc{i}'+'var']

    # Convert list into a string
    sc_str = convert_single_item_list_to_string(sc_value)

    # Update the Dictionary
    data.update({sc:sc_str})

for i in range(1, 6):
    istring = str(i)
    row = df[df['TYPE'] == 'inc'+istring+'var']

    # Delete TYPE column
    del row['TYPE']

    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("'","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["'"]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    inc = 'inc'+istring+'var'

    globals()[f'inc{i}'+'var'] = list(filter(None, strings))

    inc_value = globals()[f'inc{i}'+'var']

    # Convert list into a string
    inc_str = convert_single_item_list_to_string(inc_value)

    # Update the Dictionary
    data.update({sc:sc_str})

# Read data from an Excel file and return a Panda Data55
df = pd.read_excel(file_path, sheet_name=sheet)

# the variable Table Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'subseg']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list
```

```
# Elements to remove
to_remove = [""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

subseg = list(filter(None, strings))

subseg_str = convert_single_item_list_to_string(subseg)

# Update the Dictionary
data.update({'subseg':subseg})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'attdirect']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = [""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

attdirect = list(filter(None, strings))

attdirect_str = convert_single_item_list_to_string(attdirect)

# Update the Dictionary
data.update({'attdirect':attdirect_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'attbysum']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = [""]
```

```
# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

attbysum = list(filter(None, strings))

attbysum_str = convert_single_item_list_to_string(attbysum)

# Update the Dictionary
data.update({'attbysum':attbysum_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'attontosum']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

attontosum = list(filter(None, strings))

attontosum_str = convert_single_item_list_to_string(attontosum)

# Update the Dictionary
data.update({'attontosum':attontosum_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'ibinc']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

tbinc = list(filter(None, strings))
```

```
tbinc_str = convert_single_item_list_to_string(tbinc)

# Update the Dictionary
data.update({'tbinc':tbinc_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'meanv']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

meanv = list(filter(None, strings))

meanv_str = convert_single_item_list_to_string(meanv)

# Update the Dictionary
data.update({'meanv':meanv_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'segment']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

segment = list(filter(None, strings))

segment_str = convert_single_item_list_to_string(segment)

# Update the Dictionary
```

```
data.update({'segment':segment_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'noreg']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = [""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

noreg = list(filter(None, strings))

noreg_str = convert_single_item_list_to_string(noreg)

# Update the Dictionary
data.update({'noreg':noreg_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'regsplit']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("""","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = [""]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

regsplit = list(filter(None, strings))

regsplit_str = convert_single_item_list_to_string(regsplit)

# Update the Dictionary
data.update({'regsplit':regsplit_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'sigacross']
```

```
# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

sigacross = list(filter(None, strings))

sigacross_str = convert_single_item_list_to_string(sigacross)

# Update the Dictionary
data.update({'sigacross':sigacross_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'siggroup']

# Delete TYPE column
del row['TYPE']

row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

row = row.replace(' ', "")  # replace single space with null
row = row.replace('  ', "") # replace double space with null
row = row.replace("'","")   # replace single quote with null

test_strings = row.values.tolist() # Convert dataframe to list

# Elements to remove
to_remove = ["'"]

# Using filter to remove elements
strings = list(filter(lambda x: x not in to_remove, test_strings))

siggroup = list(filter(None, strings))

siggroup_str = convert_single_item_list_to_string(siggroup)

# Update the Dictionary
data.update({'siggroup':siggroup_str})

# the variable Weight Has to be handled differently it does not conform to Key pairing
row = df[df['TYPE'] == 'format']

# Delete TYPE column
del row['TYPE']
```

```
    row.dropna(how='all', axis=1, inplace=True) # Drop all columns with spaces or which are blank

    row = row.replace(' ', "")  # replace single space with null
    row = row.replace('  ', "") # replace double space with null
    row = row.replace("'","")   # replace single quote with null

    test_strings = row.values.tolist() # Convert dataframe to list

    # Elements to remove
    to_remove = ["'"]

    # Using filter to remove elements
    strings = list(filter(lambda x: x not in to_remove, test_strings))

    format = list(filter(None, strings))

    format_str = convert_single_item_list_to_string(format)

    # Update the Dictionary
    data.update({'format':format})

    return # all_others

def write_excel_stdout(update_data):
    """
    Write the updated data to stdout.
    """
    # Convert dictionary to a JSON string
    json_str= json.dumps(update_data)

    # Write the JSON string to stdout
    with open('/content/joutput.txt', 'w') as f:
        json.dump(json_str, f, indent=4)

    return # write_excel_stdout

if __name__ == "__main__":
  process_excel_from_stdin(read_excel_data,data)
  flag_sumv(file_path, data)
  flag_sumi(file_path, data)
  flag_sums(file_path, data)
  flag_drivers(file_path, data)
  flag_detv(file_path,data)
  all_others(file_path, data)
  write_excel_stdout(data)
  print(data)
```

AHM

```
#-------------------------------------------------------------------------------
# Program: AMH
#
# Author: Vincent Taylor
#
# Creation Date: 11-2024
#
# Last Modified: 11-2024
#
# Purpose: This program is the main python module used to build Key Driver regesssion
#                models. Its function is to read the AMH sav dataset.
#-------------------------------------------------------------------------------

# Input File(s): /content/AMH_WI_15.sav
#               /content/joutput.txt
#
# Output: Statistical measurements. See log
#-------------------------------------------------------------------------------

global true, null, false
true = True
null = None
false = False

import pandas as pd
import numpy as np
import pyreadstat
import scipy as stats
import multiprocessing
import subprocess
import time
import threading
from multiprocessing import Queue
from queue import Empty
import json
import os
import io
import sys
env = os.environ.copy()
env['PYTHONUNBUFFERED'] = '1'

#-------------------------------------------
# Allow printing of all data to screen
# Uncomment for testing
#-------------------------------------------
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.options.mode.copy_on_write = True

# Read the input dataset
#original = pd.read_sas("AMH_W1-15.sav", format='sas7bdat')

pathname = '/content/AMH_W1_15.sav'
```

```
original, meta = pyreadstat.read_sav(pathname)

# REMOVE AFTER TESTING
#print(original.columns)

# Rename columns to avoid formatting error
rawdat = original.rename(columns=lambda x: x.strip())

# Display information about the dataset
#print(rawdat.info())

# Display the first few rows of the dataset
#print(rawdat.head())

# Check for missing values
#print(rawdat.isnull().sum())

# Check for missing isna values
#print(rawdat.isna().sum())

# Create new variables
client = rawdat.copy()

# Fill columns with missing values with 0
client['BRAND_ATTR_3'] = client['BRAND_ATTR_3'].fillna(0)
client['BRAND_ATTR_4'] = client['BRAND_ATTR_4'].fillna(0)
client['BRAND_ATTR_9'] = client['BRAND_ATTR_9'].fillna(0)
client['BRAND_ATTR_10'] = client['BRAND_ATTR_10'].fillna(0)
client['BRAND_ATTR_11'] = client['BRAND_ATTR_11'].fillna(0)

#-------------------------------------------------------------------------------
# Read in the output.txt dataset and execute the Data_Structure.py program

src_path = "/content/Data_Structure.py"

command = ['python',src_path]

ipath = "/content/joutput.txt"
opath = "/content/output.txt"

F = open(opath, "w+")

try:
    proc = subprocess.Popen(command, shell=True,encoding='utf-8', stdout=F)
    proc.wait()
except Exception as ex:
    print("ERROR: CMD Failed....", ex)
F.close()
try:
    with open(ipath, 'r+') as j:
      jdata = json.loads(j.read())
except json.decoder.JSONDecodeError as e:
    print("invalid json", e)

#-------------------------------------------
# Convert jdata string to dictionary
#-------------------------------------------
```

```
cdata = json.loads(jdata)

# Create field for drivers
test_field = cdata['drivers']

# Convert list into a string
driver_str = ','.join(map(str, test_field))

# Strip brackets and and extra commas
driver_hold = driver_str.replace("[", "").replace("]", "").replace("'", "")

drivers = driver_hold

# Convert string into a list
drivers = drivers.split(", ")

print('drivers Type: ',type(drivers))
print(drivers)

# Create field for vdrivers
sumv1 = cdata.get('sumv1')
sumv2 = cdata.get('sumv2')
sumv3 = cdata.get('sumv3')
sumv4 = cdata.get('sumv4')
sumv5 = cdata.get('sumv5')
sumv6 = cdata.get('sumv6')
sumv7 = cdata.get('sumv7')
sumv8 = cdata.get('sumv8')
sumv9 = cdata.get('sumv9')
sumv10 = cdata.get('sumv10')
sumv11 = cdata.get('sumv11')
sumv12 = cdata.get('sumv12')
sumv13 = cdata.get('sumv13')
sumv14 = cdata.get('sumv14')
sumv15 = cdata.get('sumv15')
sumv16 = cdata.get('sumv16')
sumv17 = cdata.get('sumv17')
sumv18 = cdata.get('sumv18')
sumv19 = cdata.get('sumv19')
sumv20 = cdata.get('sumv20')
sumv21 = cdata.get('sumv21')
sumv22 = cdata.get('sumv22')
sumv23 = cdata.get('sumv23')
sumv24 = cdata.get('sumv24')
sumv25 = cdata.get('sumv25')
sumv26 = cdata.get('sumv26')
sumv27 = cdata.get('sumv27')
sumv28 = cdata.get('sumv28')
sumv29 = cdata.get('sumv29')
sumv30 = cdata.get('sumv30')
sumv31 = cdata.get('sumv31')
sumv32 = cdata.get('sumv32')
sumv33 = cdata.get('sumv33')
sumv34 = cdata.get('sumv34')
sumv35 = cdata.get('sumv35')
sumv36 = cdata.get('sumv36')
sumv37 = cdata.get('sumv37')
sumv38 = cdata.get('sumv38')
```

```
sumv39 = cdata.get('sumv39')
sumv40 = cdata.get('sumv40')
sumv41 = cdata.get('sumv41')
sumv42 = cdata.get('sumv42')
sumv43 = cdata.get('sumv43')
sumv44 = cdata.get('sumv44')
sumv45 = cdata.get('sumv45')
sumv46 = cdata.get('sumv46')
sumv47 = cdata.get('sumv47')
sumv48 = cdata.get('sumv48')
sumv49 = cdata.get('sumv49')
sumv50 = cdata.get('sumv50')

sumv = {f"sumv{i}": cdata.get(f"sumv{i}") for i in range(1, 51)}

sumv_str = ""

# Convert dict into a string exclude null values
def extract_non_null_values(dictionary):
  result = []
  for key, value in dictionary.items():
    if value is not None:
      result.append(value)
  return " ".join(result)

sumv_str = extract_non_null_values(sumv)

# Remove leading and trailing spaces
sumv_str = sumv_str.strip()
sumv_hold = sumv_str.replace(" ", ", ")

# print the string
print(sumv_str)

# Create field for idrivers
test_field = cdata['sumi']

# Convert list into a string
sumi_str = ','.join(map(str, test_field))

# Strip brackets and and extra commas
sumi_hold = sumi_str.replace("[", "").replace("]", "").replace("'", "")

isumi = sumi_hold

# Convert string into a list
isumi = isumi.split(", ")

print('isumi Type: ',type(isumi))
print(isumi)

# Create field for sums
test_field = cdata['sums']

# Convert list into a string
sums_str = ','.join(map(str, test_field))

# Strip brackets and and extra commas
```

```
sums_hold = sums_str.replace("[", "").replace("]", "").replace("'", "")

ssums = sums_hold

# Convert string into a list
#ssums = ssums.split(", ")

print('ssums Type: ',type(ssums))
print(ssums)

# Frequency tables

print(rawdat['OSAT'].value_counts())
print(rawdat['NPS_OVERALL'].value_counts())
print(rawdat['RENEWAL_LEASEPROP'].value_counts())

# Python3 code to demonstrate working of
# Convert key-value String to dictionary
# Using dict() + generator expression + split() + map()

# Perform concatenation
corr_vars = 'NPS_OVERALL' + ", " + driver_hold

# Convert string into a list
corr_vars = corr_vars.split(", ")

print('corr_vars Type: ',type(corr_vars))

# Perform correlation
try:
  corr_data = rawdat[corr_vars].corr()
except Exception as e:
  missing_key = e.args[0]
  print(f"Key '{missing_key}' is missing")

print(corr_data)

# Create client dataframe
client = rawdat[rawdat['Wave'].isin([13, 14, 15])].copy()

client['poids'] = 1
client['total'] = 1
client['total2'] = 1
client['total3'] = 1
client['total4'] = 1

client['v1'] = client['OSAT']
client['v2'] = client['NPS_OVERALL'] + 1
client['v3'] = client['RENEWAL_LEASEPROP']

# Rename driver attributes to v_ format
# Set DK and Refused to Missing, Skip Patterns to 0

# Create the array of drivers
for i, driver in enumerate(drivers, 4):
    client[f'v{i}'] = client[driver]
    client.loc[~client[f'v{i}'].isin(range(1, 12)), f'v{i}'] = 0
```

```
# Create NA Flags
for i in range(4, 13):
    client[f'i_v{i}'] = np.where(client[f'v{i}'] == 0, 0, 1)

# Create satisfaction variables
client['tsat'] = np.where(client['v1'] == 5, 0, np.where(client['v1'] == 4, 1, np.nan))
client['usat'] = np.where(client['v1'].isin([1, 2, 3]), 1, np.where(client['v1'] == 4, 0, np.nan))

client['tsat2'] = np.where(client['v2'].isin([10, 11]), 0, np.where(client['v2'].isin([8, 9]), 1, np.nan))
client['usat2'] = np.where(client['v2'].isin([1, 2, 3, 4, 5, 6, 7]), 1, np.where(client['v2'].isin([8, 9]), 0, np.nan))

client['tsat4'] = np.where(client['v2'] == 11, 0, np.where(client['v2'].isin([8, 9, 10]), 1, np.nan))
client['usat4'] = np.where(client['v2'].isin([1, 2, 3, 4, 5, 6, 7]), 1, np.where(client['v2'].isin([8, 9, 10]), 0, np.nan))

client['tsat3'] = np.where(client['v1'] == 5, 0, np.where(client['v1'] == 4, 1, np.nan))
client['usat3'] = np.where(client['v1'].isin([1, 2, 3]), 1, np.where(client['v1'] == 4, 0, np.nan))

# Frequency table for wave
print(client['Wave'].value_counts())

si_test = ssums + ", " + sumv_hold + ", " + sumi_hold

# Convert tuple into a string
def convertTuple(tup):
    field = ''
    for item in tup:
        field = field + item
    return field

# Execute ConvertTuple
si_test = convertTuple(si_test)

# Convert string into a list
si_test = si_test.split(", ")

# Create summary statistics
print('Display summary statistics:')
print(client[si_test].mean())
#print(client[si_test].std())              # Uncomment if needed
#print(client[si_test].median())           # Uncomment if needed
#print(client[si_test].mode())             # Uncomment if needed

# Create variable labels
var_labels = {
    'v1': 'Overall Satisfaction',
    'v2': 'NPS',
    'v3': 'Likelihood to renew lease',
    'v4': 'My rental resident manager makes me feel like a valued customer',
    'v5': 'My rental resident manager treats me fairly ',
    'v6': 'My rental resident manager makes renting easy for me',
    'v7': 'My rental resident manager is supportive and cares about my needs',
    'v8': 'My rental resident manager maintains my rental property beyond my expectations',
    'v9': 'My rental resident manager is responsive to my requests/calls',
    'v10': 'My rental resident manager is helpful with my requests/calls',
    'v11': 'My rental resident manager sets clear expectations regarding my requests',
    'v12': 'My rental resident manager follows through with resolving my requests'
}
```

```
# Frequency tables and correlations
print('Display frequency counts for V2, tsat2, usat2')
print(client[si_test[0]].value_counts())
print(client['tsat2'].value_counts())
print(client['usat2'].value_counts())

sumv_vars = si_test

# Perform correlation
try:
  corr_data = client[sumv_vars].corr()
  print('Display correlations: ')
  print(corr_data)
except Exception as e:
  missing_key = e.args[0]
  print(f"Key '{missing_key}' is missing")

# Perform frequency
print('Display frequency counts for v2, v4-v12')
print(client[sumv_vars].value_counts())

#--------------------------------------------------------------------------------
# Execute the ATTImpactV29.py program

src_path = "/content/ATTImpactV29.py"

command = ['python',src_path]

out_path = "gs://xxxx-cxusanalytics-dev-kda_01_test_information/Output/"

F = open(opath, "w+")

try:
    proc = subprocess.Popen(command, shell=True,encoding='utf-8', stdout=F)
    proc.wait()
except Exception as ex:
    print("ERROR: CMD Failed....", ex)
F.close()
```