AICPA

Audit Data Standard and Audit Data Analytics Working Group

Upgrade the Financial Statement Audit using Audit Data Analytics

I. Introduction

This document is part of a series of instructional papers meant to illustrate how the AICPA's Audit Data Standards (ADS) facilitate the use of data analytics in the financial statement audit. This paper focuses on a popular open-source programming language, Python, and how it can be used to perform certain financial statement audit procedures. More specifically, this paper will help users gain an understanding of how to use Python to do the following:

- Convert a trial balance and general ledger data set to the standardized ADS format
- Develop automated, repeatable routines to analyze the ADS standardized data set
- View, analyze, and document code and results

For further guidance, this paper can be used in conjunction with the micro learning session video "Upgrade the Financial Statement Audit with Python." To view additional micro learning session videos related to this subject matter please visit the AICPA's Audit Data Standards website.

II. Overview

March 2019

III. What is Python?

Python is an open-source programming language that was first released in 1991; in other words, the original source code is freely available and can be modified or redistributed. It's available for a variety of operating systems and can be used for general-purpose programming for both large and small projects. Python's simple coding style makes it the preferred language for those beginning to learn how to code.

Python supports many programming paradigms such as imperative, functional, and procedural. *Programming paradigms* are ways to classify programming languages based on their features.[1] Common paradigms[2] include the following:

- ***Imperative*** — allows side effects
- ***Object-oriented*** — groups code together with the state the code modifies
- ***Procedural*** — groups code into functions
- ***Declarative*** — does not state the order in which operations execute
- ***Functional*** — disallows side effects
- ***Logic*** — has a particular style of execution model coupled to a particular style of syntax and grammar
- ***Symbolic programming*** — has a particular style of syntax and grammar

Python can be used in many different areas and throughout many different industries such as data science, web development, finance, accounting and auditing, molecular biology, and application security. Specific uses include the following:

- ***Data engineering* –** Cleansing data, structuring data, and loading data
- ***Analytics* –** AI, text mining, visualizations
- ***Automation* –** Extract, transform, load (ETL), conversion, and reporting

Python also can be used to create or interact with web applications as part of web development or micro services. For the purpose of this paper, we will focus on the use of Python in the financial statement audit.

IV. Python and the Financial Statement Audit

There are a wide variety of uses for Python. When it comes to the financial statement audit, Python can help with extracting, transforming (or formatting) and loading data, as well as testing and analyzing the data and developing visualizations to help view and document results. Subsequent sections will walk through an example of how Python can be used for extracting data and transforming it into the ADS standardized format, loading the standardized data, and developing code to further analyze the ADS standardized data set.

[1] https://en.wikipedia.org/wiki/Programming_paradigm

V. Python Example

This section will walk through the process of using Python to (1) apply the ADS format to an SAP test data set and (2) develop code to further analyze the ADS standardized data set (perform journal entry testing procedures). As stated previously, this paper can be used in conjunction with the micro session video, “Upgrade the Financial Statement Audit with Python.” Please note that the routines developed here can be used on any ADS standardized data set and can be accessed on the AICPA’s Audit Data Standards webpage.

Applying the AICPA’s Audit Data Standard Format

As this example focuses on journal entry test work, the AICPA’s general ledger ADS format was used and applied to an SAP test data set. The full audit data standard document can be accessed on the AICPA’s Audit Data Standard website.

As a first step, the high-level mapping, discussed in the micro learning session “Introduction to the Audit Data Standards” and shown in figure 1, was used to develop Python code to load the SAP test data set and apply the general ledger ADS format. This mapping is important because the field names identified in figure 1 were used within the Python code to help identify the fields within the SAP test data set that would need to be reformatted.

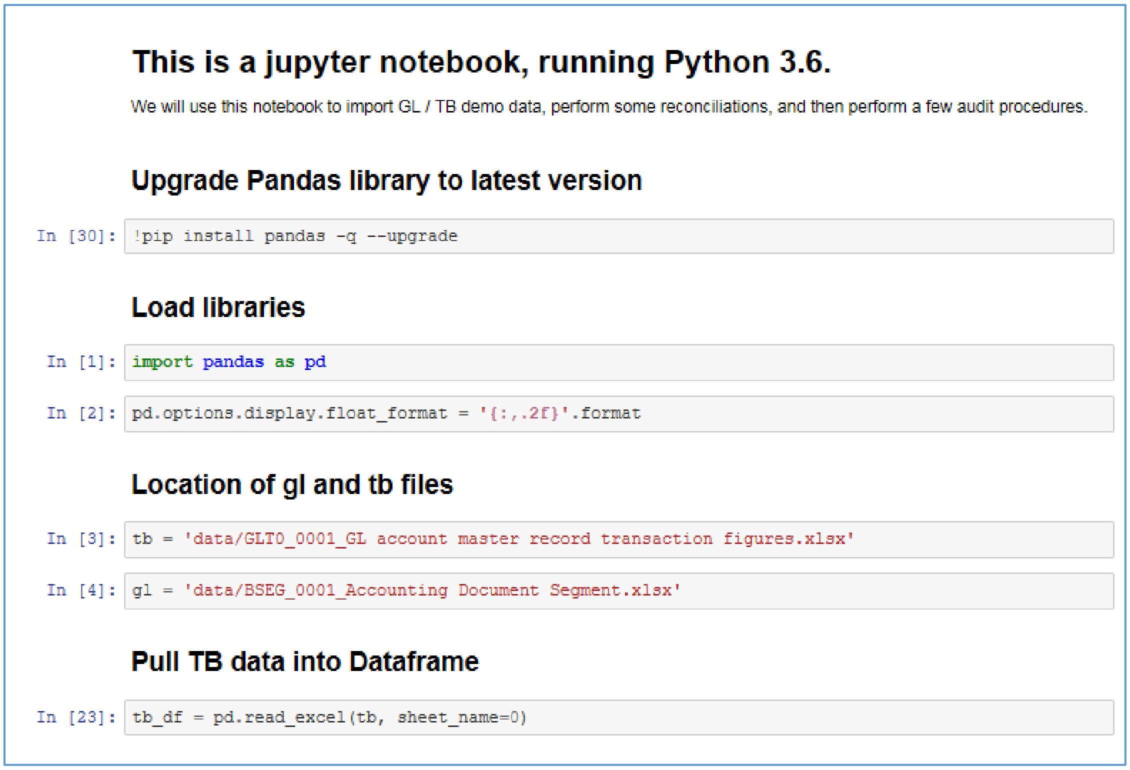
Figure 1 – High-Level Mapping of ADS Field Names to SAP Test Data Set Field Names

ADS Table	ADS Field Name	SAP Table	SAP Field Name
GL_Detail_YYYYMMDD_YYYYMMDD	Journal_ID	BSEG_0001_Accounting Document Segment	BELNR (Accounting Document Number)
GL_Detail_YYYYMMDD_YYYYMMDD	Journal_ID_Line_Number	BSEG_0001_Accounting Document Segment	BUZEI (Number of Line Item Within Accounting Document)
GL_Detail_YYYYMMDD_YYYYMMDD	JE_Header_Description	BKPF_0001_Accounting Document Header	BKTXT (Document Header Text)
GL_Detail_YYYYMMDD_YYYYMMDD	JE_Line_Description	BSEG_0001_Accounting Document Segment	SGTXT (Item Text)
GL_Detail_YYYYMMDD_YYYYMMDD	Source	BKPF_0001_Accounting Document Header	BLART (Document Type)
GL_Detail_YYYYMMDD_YYYYMMDD	Business_Unit_Code	BSEG_0001_Accounting Document Segment	GSBER (Business Area)
Chart_Of_Accounts	GL_Account_Number	SKA1_0001_GL Account Master (Chart of Accounts)	SAKNR (G/L Account Number)
Chart_Of_Accounts	GL_Account_Name	SKAT_0001_GL Account Master Record (Chart of Accounts Description)	MCOD1 (Search Term for Matchcode Search)
Chart_Of_Accounts	Account_Type	SKA1_0001_GL Account Master (Chart of Accounts)	XBILK, GVTYP,
Chart_Of_Accounts	Account_Subtype	SKA1_0001_GL Account Master (Chart of Accounts)	KTOKS (G/L Account Group)
Chart_Of_Accounts	GL_Account_Description	SKAT_0001_GL Account Master Record (Chart of Accounts Description)	TXT50 (G/L Account Long Text)
Trial_Balance_YYYYMMDD_YYYYMMDD	GL_Account_Number	GLT0_0001_GL account master record transaction figures	RACCT (Account Number)
Trial_Balance_YYYYMMDD_YYYYMMDD	Business_Unit_Code	GLT0_0001_GL account master record transaction figures	RBUSA (Business Area)
Trial_Balance_YYYYMMDD_YYYYMMDD	Fiscal_Year	GLT0_0001_GL account master record transaction figures	RYEAR (Fiscal Year)
Trial_Balance_YYYYMMDD_YYYYMMDD	Period	GLT0_0001_GL account master record transaction figures	RPMAX (Period)

Python code was developed to load the SAP test data set into Python within the Jupyter Notebook. Jupyter Notebook is an open-source web application that allows you to develop code and explore data in a format that contains live code, results, visualizations, and narrative text.

In order to apply the ADS format to the SAP test data set, the test data set was loaded into Jupyter. Figure 2 shows the code that was written to identify and load the appropriate fields from the SAP test data set into Jupyter. For this example, only selected general ledger and trial balance fields from the test data set were loaded into Jupyter.

Figure 2 – Loading SAP Test Data Set (Trial Balance and General Ledger fields only) Into Jupyter



This is a jupyter notebook, running Python 3.6.

We will use this notebook to import GL / TB demo data, perform some reconciliations, and then perform a few audit procedures.

Upgrade Pandas library to latest version

```
In [30]: !pip install pandas -q --upgrade
```

Load libraries

```
In [1]: import pandas as pd

In [2]: pd.options.display.float_format = '{:,.2f}'.format
```

Location of gl and tb files

```
In [3]: tb = 'data/GLT0_0001_GL account master record transaction figures.xlsx'

In [4]: gl = 'data/BSEG_0001_Accounting Document Segment.xlsx'
```

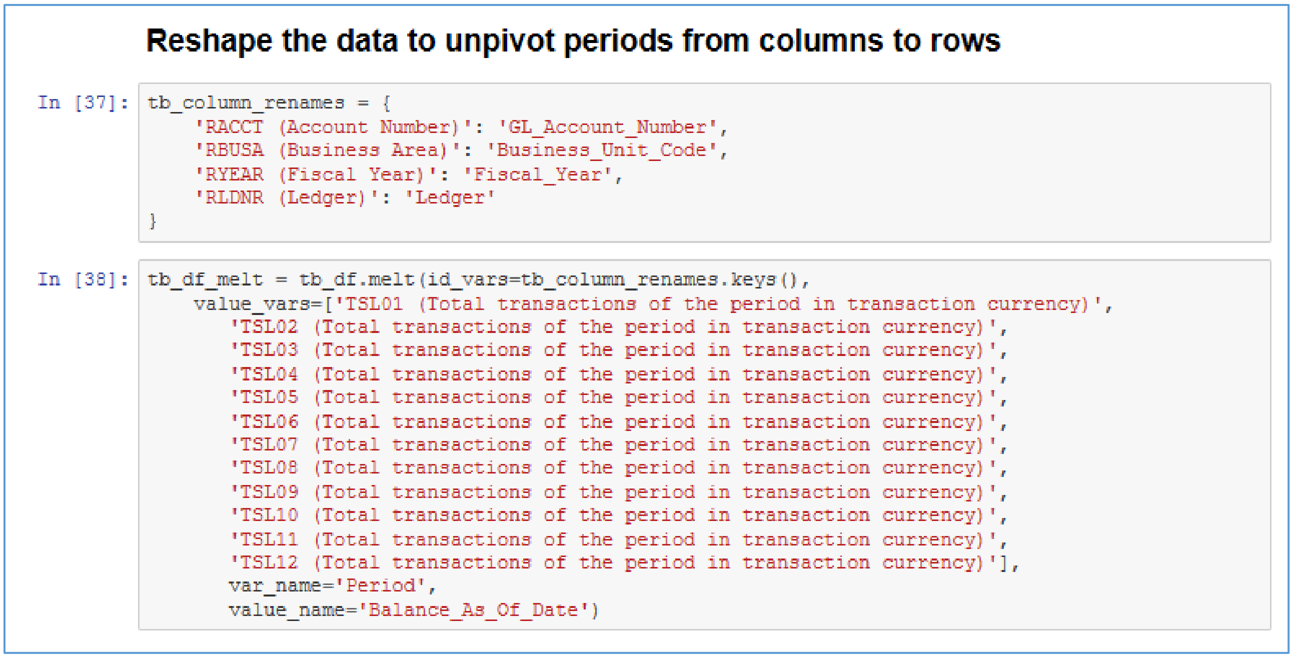
Pull TB data into Dataframe

```
In [23]: tb_df = pd.read_excel(tb, sheet_name=0)
```

As seen in figure 2, code was written to identify where the SAP test data set trial balance and general ledger files were saved (lines [3] and [4]). These files were then loaded into the Jupyter Notebook in a format called a Pandas DataFrame (line [23]). Pandas is an open-source library providing high-performance, easy-to-use data structures and data analysis tools, and DataFrame is the primary data structure used in Pandas.

Once loaded into Jupyter, code was developed to “reshape” the data into the ADS format. Figure 3 illustrates the code that was used.

Figure 3 – Python Code Developed to Reshape the SAP Trial Balance Test Data Into the ADS Format



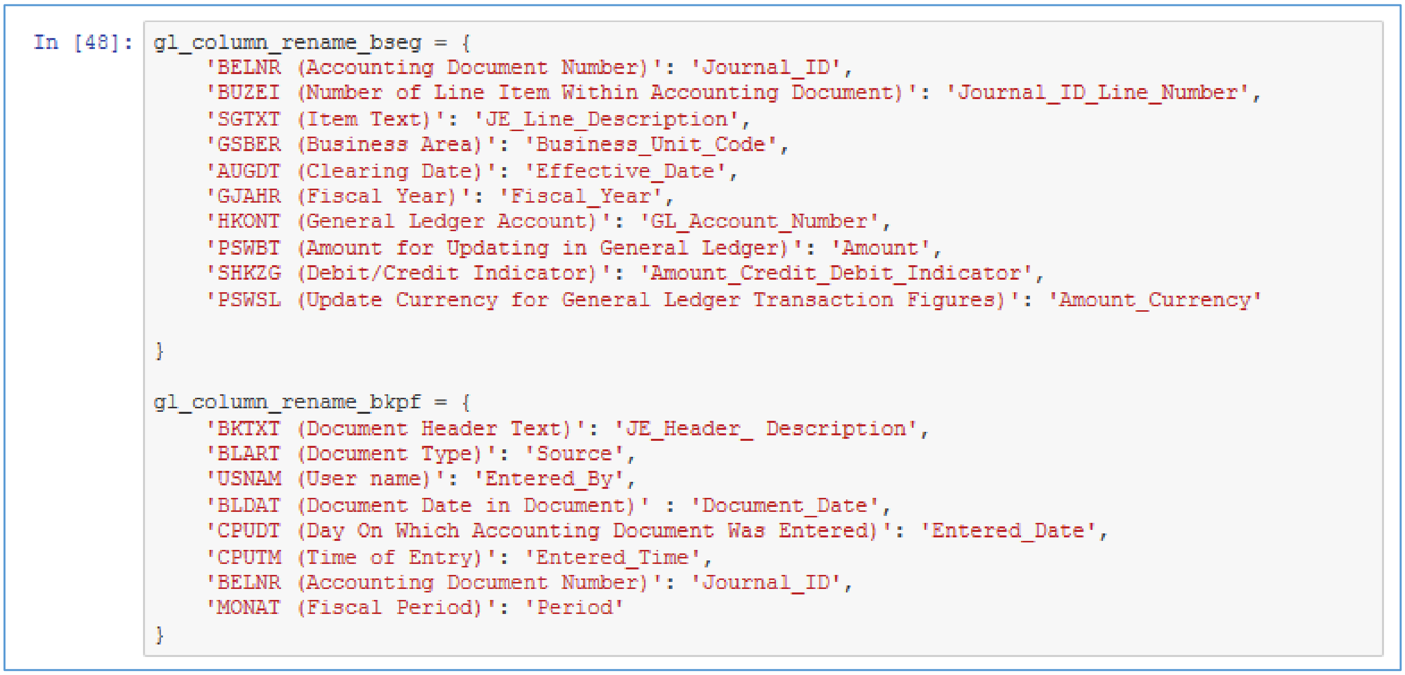
Reshape the data to unpivot periods from columns to rows

```
In [37]: tb_column_renames = {
             'RACCT (Account Number)': 'GL_Account_Number',
             'RBUSA (Business Area)': 'Business_Unit_Code',
             'RYEAR (Fiscal Year)': 'Fiscal_Year',
             'RLDNR (Ledger)': 'Ledger'
         }
```

```
In [38]: tb_df_melt = tb_df.melt(id_vars=tb_column_renames.keys(),
             value_vars=['TSL01 (Total transactions of the period in transaction currency)',
                'TSL02 (Total transactions of the period in transaction currency)',
                'TSL03 (Total transactions of the period in transaction currency)',
                'TSL04 (Total transactions of the period in transaction currency)',
                'TSL05 (Total transactions of the period in transaction currency)',
                'TSL06 (Total transactions of the period in transaction currency)',
                'TSL07 (Total transactions of the period in transaction currency)',
                'TSL08 (Total transactions of the period in transaction currency)',
                'TSL09 (Total transactions of the period in transaction currency)',
                'TSL10 (Total transactions of the period in transaction currency)',
                'TSL11 (Total transactions of the period in transaction currency)',
                'TSL12 (Total transactions of the period in transaction currency)'],
                var_name='Period',
                value_name='Balance_As_Of_Date')
```

Figure 3 illustrates the code that was written in order to begin "reshaping" the SAP trial balance data set. As can be seen in the previous chart at line [37], the SAP fields: Account Number, Business Area, Fiscal Year, and Ledger were identified to be "reshaped" into the ADS format: GL_Account_Number, Business_Unit_Code, and Fiscal_Year. Figure 4, which follows, shows additional code that was used to reformat and apply the ADS format to the SAP general ledger data set.

Figure 4 – Reshape the SAP General Ledger Test Data Into the ADS Format

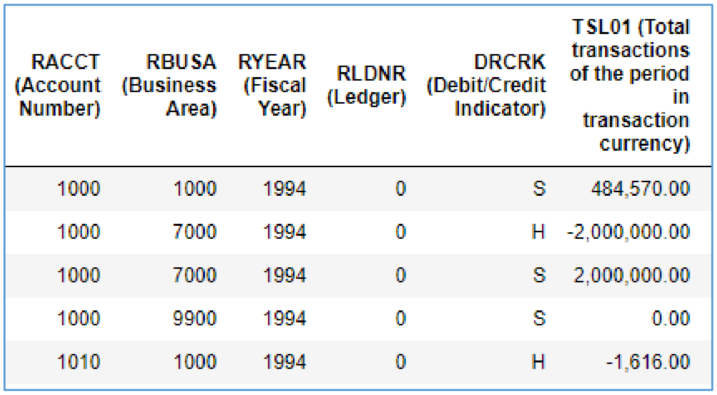


```
In [48]: gl_column_rename_bseg = {
             'BELNR (Accounting Document Number)': 'Journal_ID',
             'BUZEI (Number of Line Item Within Accounting Document)': 'Journal_ID_Line_Number',
             'SGTXT (Item Text)': 'JE_Line_Description',
             'GSBER (Business Area)': 'Business_Unit_Code',
             'AUGDT (Clearing Date)': 'Effective_Date',
             'GJAHR (Fiscal Year)': 'Fiscal_Year',
             'HKONT (General Ledger Account)': 'GL_Account_Number',
             'PSWBT (Amount for Updating in General Ledger)': 'Amount',
             'SHKZG (Debit/Credit Indicator)': 'Amount_Credit_Debit_Indicator',
             'PSWSL (Update Currency for General Ledger Transaction Figures)': 'Amount_Currency'

         }

         gl_column_rename_bkpf = {
             'BKTXT (Document Header Text)': 'JE_Header_ Description',
             'BLART (Document Type)': 'Source',
             'USNAM (User name)': 'Entered_By',
             'BLDAT (Document Date in Document)' : 'Document_Date',
             'CPUDT (Day On Which Accounting Document Was Entered)': 'Entered_Date',
             'CPUTM (Time of Entry)': 'Entered_Time',
             'BELNR (Accounting Document Number)': 'Journal_ID',
             'MONAT (Fiscal Period)': 'Period'
         }
```

Figure 5 illustrates the final output (after the preceding routines were run) in the ADS format as compared to the original SAP test data set format. It's important to note that all output can be viewed in Jupyter as well as downloaded into an Excel spreadsheet for further analysis.

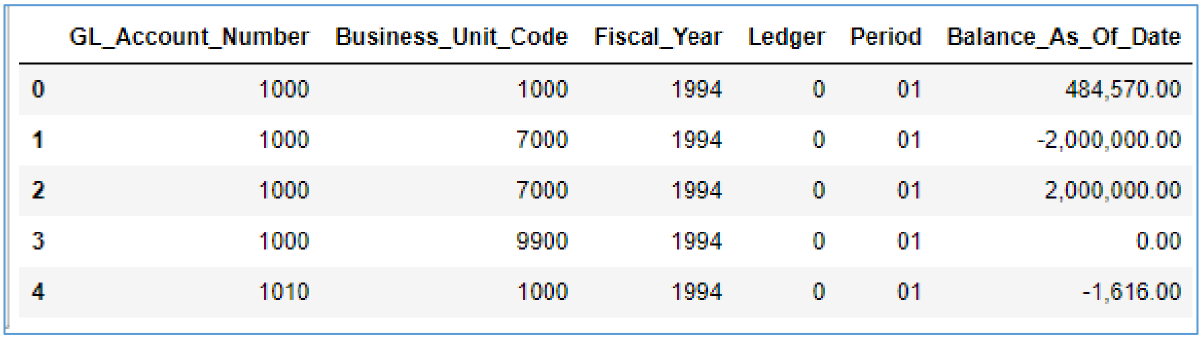
Figure 5 – Final ADS Formatted Data Set Versus Original SAP Test Data Set

Original SAP Test Data Set (Excel Format)



RACCT (Account Number)	RBUSA (Business Area)	RYEAR (Fiscal Year)	RLDNR (Ledger)	DRCRK (Debit/Credit Indicator)	TSL01 (Total transactions of the period in transaction currency)
1000	1000	1994	0	S	484,570.00
1000	7000	1994	0	H	-2,000,000.00
1000	7000	1994	0	S	2,000,000.00
1000	9900	1994	0	S	0.00
1010	1000	1994	0	H	-1,616.00

ADS Standardized Data Set (Exported From Jupyter to Excel)



	GL_Account_Number	Business_Unit_Code	Fiscal_Year	Ledger	Period	Balance_As_Of_Date
0	1000	1000	1994	0	01	484,570.00
1	1000	7000	1994	0	01	-2,000,000.00
2	1000	7000	1994	0	01	2,000,000.00
3	1000	9900	1994	0	01	0.00
4	1010	1000	1994	0	01	-1,616.00

Figures 2–5 illustrate portions of the code used to apply the ADS format to the SAP test data set. To view the full Jupyter notebook, please see appendix A.

Analyzing the ADS Standardized Data Set

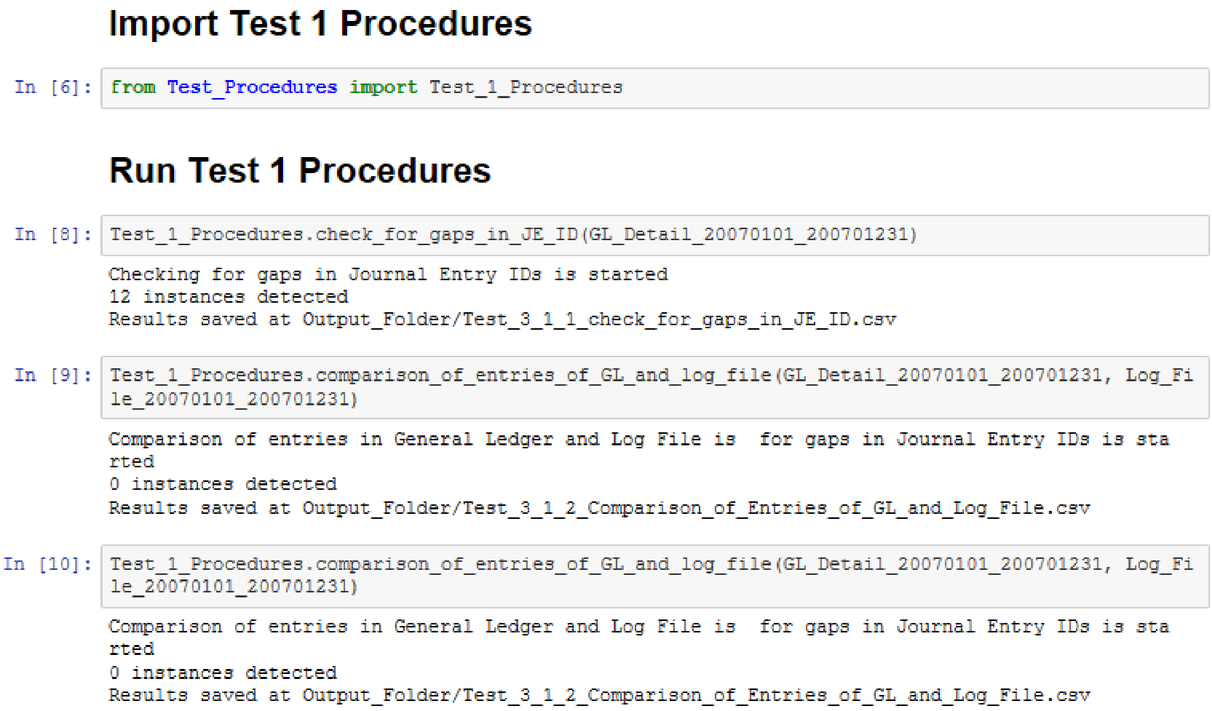
Utilizing the AICPA's Audit Data Analytics to Traditional Procedures – Mapping Document, the following journal entry audit procedures were selected to be performed over the ADS standardized data set.

Figure 6 – Audit Data Analytics to Traditional Procedures – Mapping Document

*TRADITIONAL AUDIT PROCEDURES	INDUSTRY	**** AUDIT ASSERTION OR OBJECTIVE OF THE PROCEDURE	PHASE OF AUDIT
a. Examine population for missing or incomplete journal entries.	General	Completeness and accuracy	Interim and year-end
b. Examine possible duplicate account entries.	General	Completeness	Interim and year-end
c. Examine round-dollar entries.	General	Completeness and accuracy	Interim and tear-end
d. Examine post-date entries.	General	Completeness and accuracy	Interim and year-end
e. Examine entries posted on weekends.	General	Completeness and accuracy	Interim and year-end

Figure 6 represents a portion of the AICPA's Audit Data Analytics to Traditional Procedure – Mapping Document. For each of the audit procedures noted (a–e), routines were developed and run on the ADS standardized data set. The following figures represent the code used to develop these routines. Please note that Python also allows users to save blocks of code in separate files, then load those files as libraries to be used within other files. This provides for more readable code and allows libraries that are useful in more than one situation to be used repeatedly through a simple import process. In these examples, the individual routines were written and saved as separate files, then imported into the main file. This allows Python beginners (and those who may not be familiar with coding) to more easily understand what routine is being run without having to understand all of the underlying code within each routine. Figure 7 illustrates the Python routines that were developed to cover the audit procedures noted previously, as well as some additional procedures. The routines were developed in a separate library (Test_Procedures) and able to be imported individually from that library as Test_1_Procedures and Test_2_Procedures.

Figure 7 – Test 1 and Test 2 Procedures

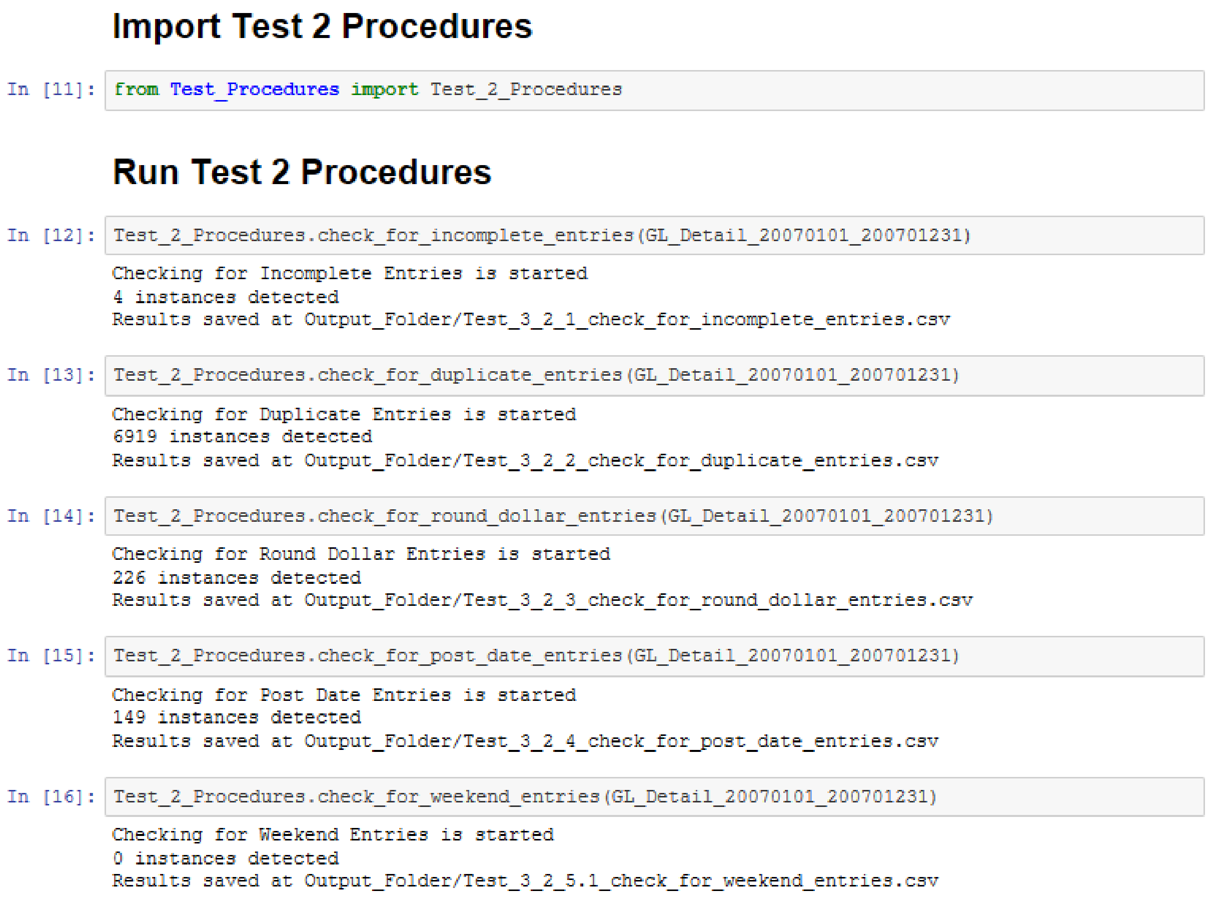


Import Test 1 Procedures

```
In [6]: from Test_Procedures import Test_1_Procedures
```

Run Test 1 Procedures

```
In [8]: Test_1_Procedures.check_for_gaps_in_JE_ID(GL_Detail_20070101_200701231)

        Checking for gaps in Journal Entry IDs is started
        12 instances detected
        Results saved at Output_Folder/Test_3_1_1_check_for_gaps_in_JE_ID.csv

In [9]: Test_1_Procedures.comparison_of_entries_of_GL_and_log_file(GL_Detail_20070101_200701231, Log_Fi
        le_20070101_200701231)

        Comparison of entries in General Ledger and Log File is  for gaps in Journal Entry IDs is sta
        rted
        0 instances detected
        Results saved at Output_Folder/Test_3_1_2_Comparison_of_Entries_of_GL_and_Log_File.csv

In [10]: Test_1_Procedures.comparison_of_entries_of_GL_and_log_file(GL_Detail_20070101_200701231, Log_Fi
         le_20070101_200701231)

         Comparison of entries in General Ledger and Log File is  for gaps in Journal Entry IDs is sta
         rted
         0 instances detected
         Results saved at Output_Folder/Test_3_1_2_Comparison_of_Entries_of_GL_and_Log_File.csv
```

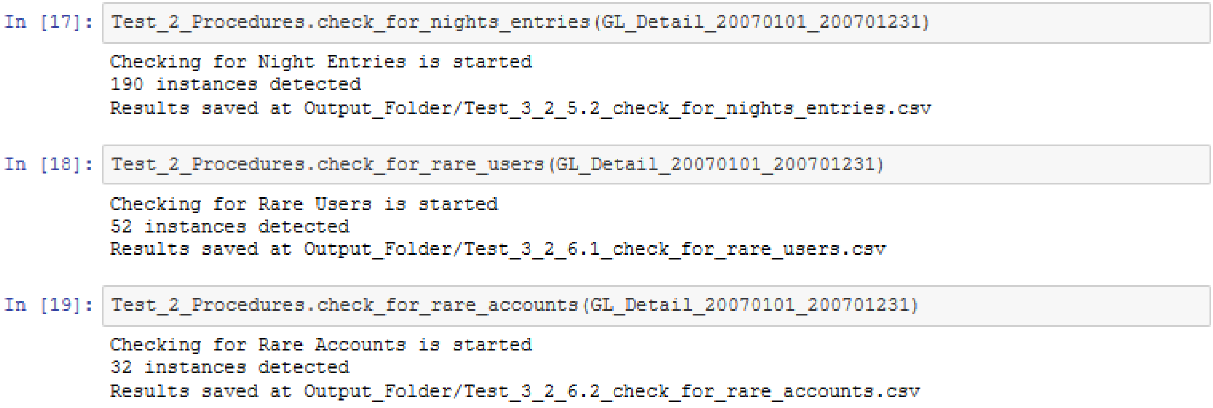


Import Test 2 Procedures

```
In [11]: from Test_Procedures import Test_2_Procedures
```

Run Test 2 Procedures

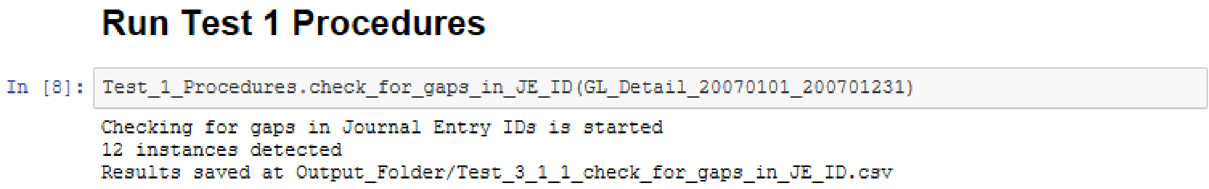
```
In [12]: Test_2_Procedures.check_for_incomplete_entries(GL_Detail_20070101_200701231)

         Checking for Incomplete Entries is started
         4 instances detected
         Results saved at Output_Folder/Test_3_2_1_check_for_incomplete_entries.csv

In [13]: Test_2_Procedures.check_for_duplicate_entries(GL_Detail_20070101_200701231)

         Checking for Duplicate Entries is started
         6919 instances detected
         Results saved at Output_Folder/Test_3_2_2_check_for_duplicate_entries.csv

In [14]: Test_2_Procedures.check_for_round_dollar_entries(GL_Detail_20070101_200701231)

         Checking for Round Dollar Entries is started
         226 instances detected
         Results saved at Output_Folder/Test_3_2_3_check_for_round_dollar_entries.csv

In [15]: Test_2_Procedures.check_for_post_date_entries(GL_Detail_20070101_200701231)

         Checking for Post Date Entries is started
         149 instances detected
         Results saved at Output_Folder/Test_3_2_4_check_for_post_date_entries.csv

In [16]: Test_2_Procedures.check_for_weekend_entries(GL_Detail_20070101_200701231)

         Checking for Weekend Entries is started
         0 instances detected
         Results saved at Output_Folder/Test_3_2_5.1_check_for_weekend_entries.csv
```



```
In [17]: Test_2_Procedures.check_for_nights_entries(GL_Detail_20070101_200701231)

         Checking for Night Entries is started
         190 instances detected
         Results saved at Output_Folder/Test_3_2_5.2_check_for_nights_entries.csv

In [18]: Test_2_Procedures.check_for_rare_users(GL_Detail_20070101_200701231)

         Checking for Rare Users is started
         52 instances detected
         Results saved at Output_Folder/Test_3_2_6.1_check_for_rare_users.csv

In [19]: Test_2_Procedures.check_for_rare_accounts(GL_Detail_20070101_200701231)

         Checking for Rare Accounts is started
         32 instances detected
         Results saved at Output_Folder/Test_3_2_6.2_check_for_rare_accounts.csv
```

Figures 8 and 9 take a deeper dive into the routine "check for gaps in journal entry IDs." Figure 8 represents the code and routine that was run, and figure 9 represents the related output. Output for each of the routines noted here can be viewed in appendix A of this paper.

Figure 8 – Routine Developed to Examine the Population for Missing or Incomplete Journal Entries

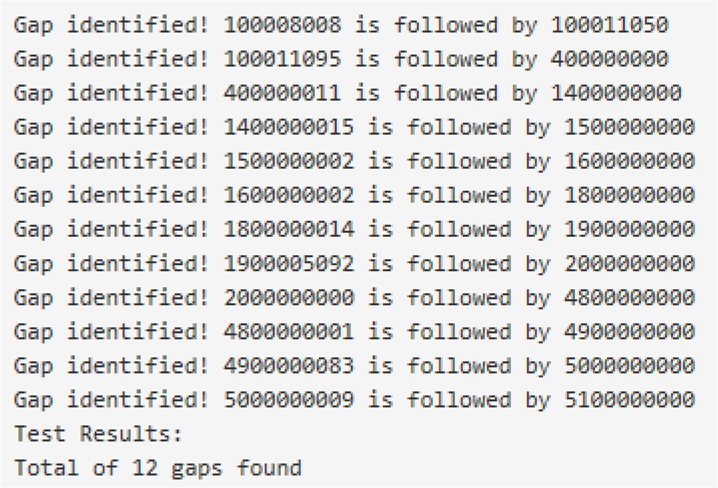


Run Test 1 Procedures

```
In [8]: Test_1_Procedures.check_for_gaps_in_JE_ID(GL_Detail_20070101_20070l231)

        Checking for gaps in Journal Entry IDs is started
        12 instances detected
        Results saved at Output_Folder/Test_3_1_1_check_for_gaps_in_JE_ID.csv
```

As noted previously, Python allows users to write code or use already written code, save as a separate file, and import the library and use a specific method such as Test_1_Procedures.check_for_gaps to run the routine. Figure 8 illustrates the routine that was created to check for missing or incomplete journal entries. This particular routine checks the population for gaps in journal entry ID number. The ADS field that is used in the coding is Journal_ID. As noted, 12 instances of gaps in IDs were noted within the population. Figure 9 illustrates the related output.

Figure 9 – Missing or Incomplete Journal Entry Output



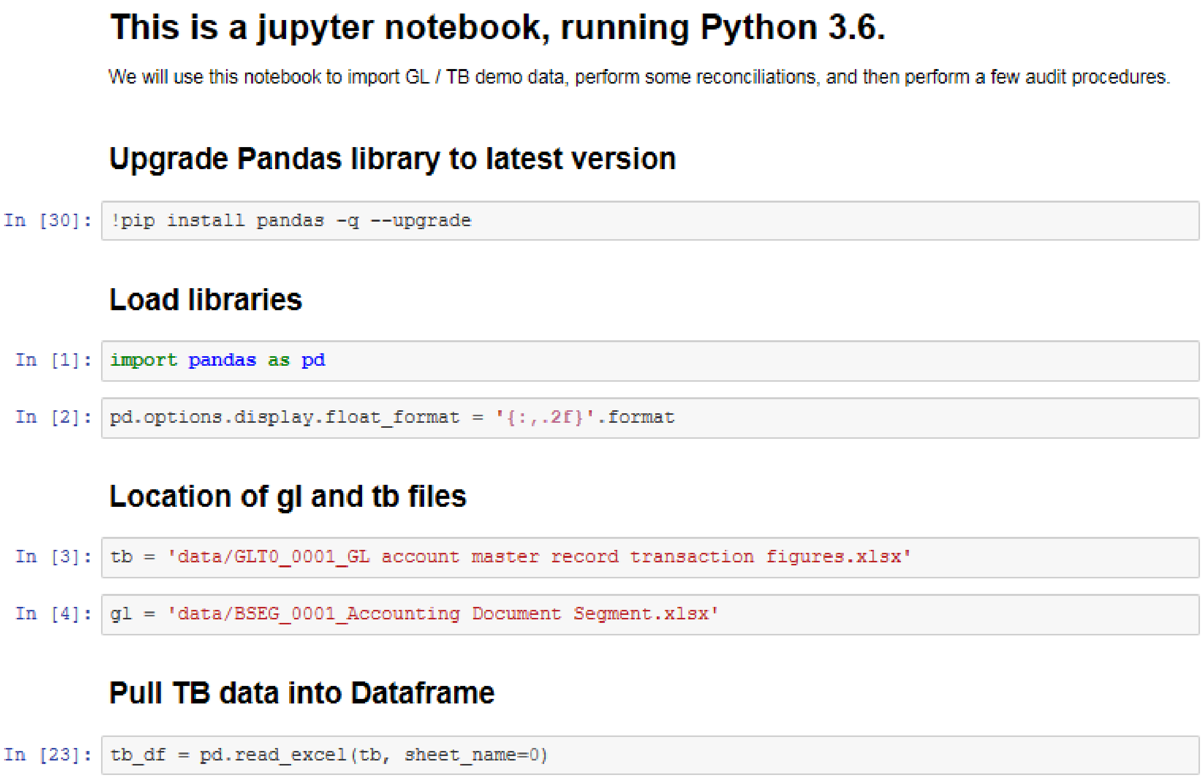
```
Gap identified! 100008008 is followed by 100011050
Gap identified! 100011095 is followed by 400000000
Gap identified! 400000011 is followed by 1400000000
Gap identified! 1400000015 is followed by 1500000000
Gap identified! 1500000002 is followed by 1600000000
Gap identified! 1600000002 is followed by 1800000000
Gap identified! 1800000014 is followed by 1900000000
Gap identified! 1900005092 is followed by 2000000000
Gap identified! 2000000000 is followed by 4800000000
Gap identified! 4800000001 is followed by 4900000000
Gap identified! 4900000083 is followed by 5000000000
Gap identified! 5000000009 is followed by 5100000000
Test Results:
Total of 12 gaps found
```

Applying audit data analytic techniques and tools to an audit, such as those that can be done using Python, can be very beneficial. It can help with the analysis of audit areas, increase your understanding of an entity and its operations, and greatly improve efficiency and accuracy. The routines that were created in this example are accessible via the AICPA's Audit Data Standards webpage. Each of the routines can be used on any AICPA ADS standardized data set, as long as the data set is properly named and contains the proper types of data in each field. Users are encouraged to visit the site and experiment more with these routines. For additional information and guidance on Audit Data Analytics and Audit Data Standards, please visit the AICPA's Audit Data Analytics website.

VI. Appendix A —Python Code

The images that follow are screenshots from the Jupyter notebook. It represents the Python code developed to (1) apply the ADS format to an SAP test data set and (2) run routines over the ADS standardized data set for further analysis. The following code can be accessed on the AICPA's Audit Data Standards webpage.

Loading and Reshaping the SAP Test Data Set



This is a jupyter notebook, running Python 3.6.

We will use this notebook to import GL / TB demo data, perform some reconciliations, and then perform a few audit procedures.

Upgrade Pandas library to latest version

```
In [30]: !pip install pandas -q --upgrade
```

Load libraries

```
In [1]: import pandas as pd
```

```
In [2]: pd.options.display.float_format = '{:,.2f}'.format
```

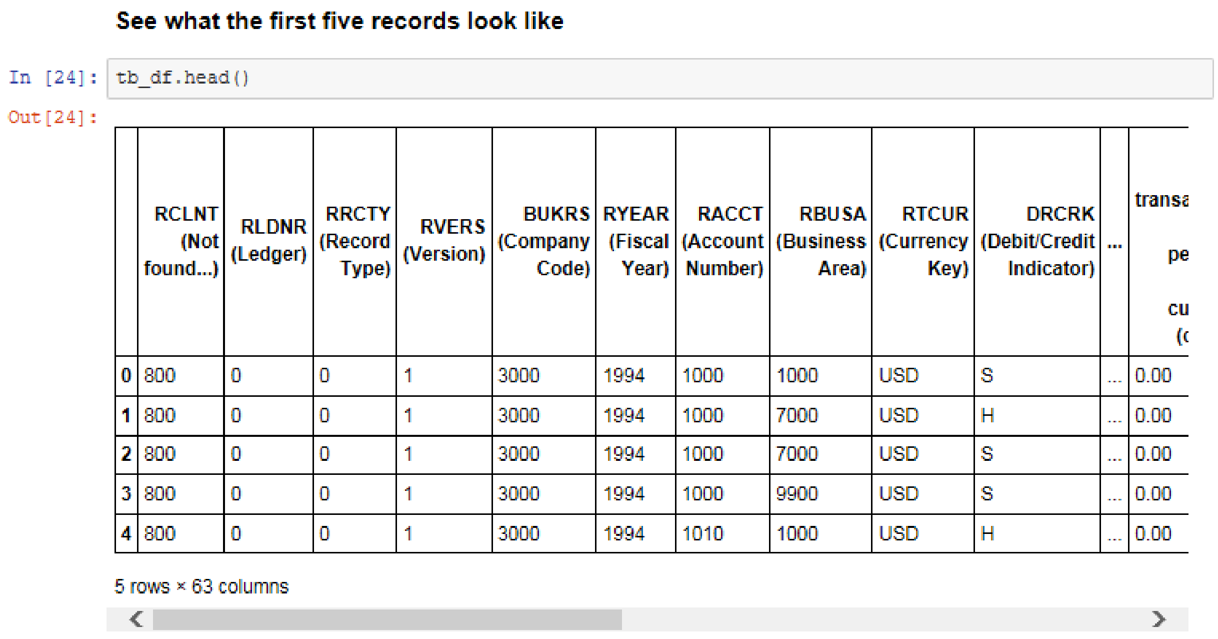
Location of gl and tb files

```
In [3]: tb = 'data/GLT0_0001_GL account master record transaction figures.xlsx'
```

```
In [4]: gl = 'data/BSEG_0001_Accounting Document Segment.xlsx'
```

Pull TB data into Dataframe

```
In [23]: tb_df = pd.read_excel(tb, sheet_name=0)
```



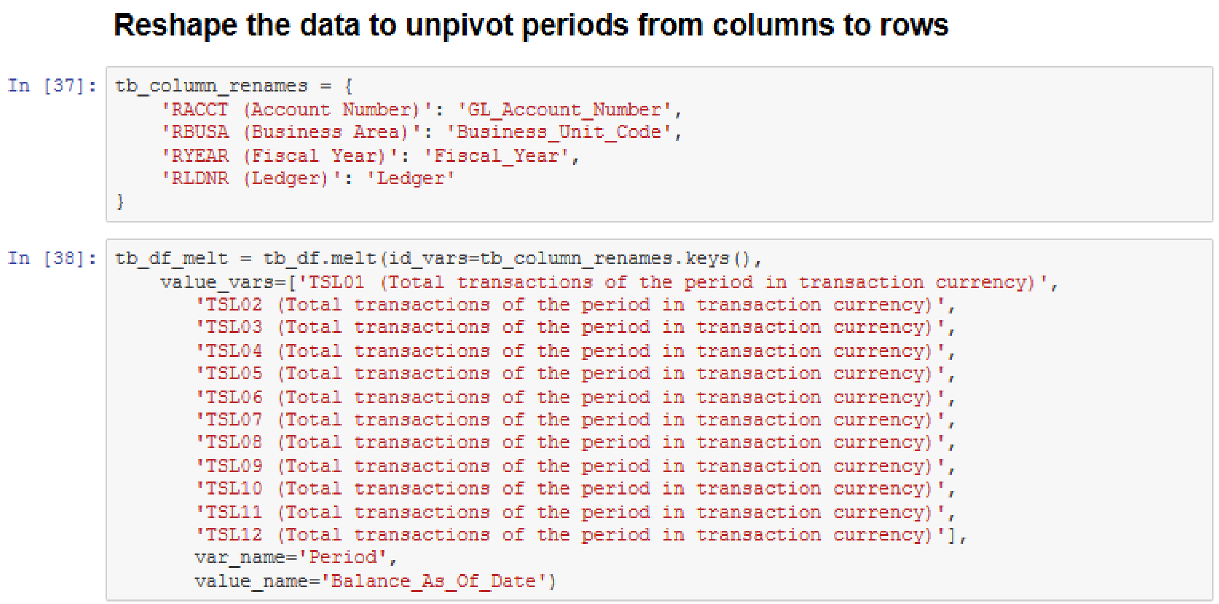
See what the first five records look like

```
In [24]: tb_df.head()
```

Out[24]:

	RCLNT (Not found...)	RLDNR (Ledger)	RRCTY (Record Type)	RVERS (Version)	BUKRS (Company Code)	RYEAR (Fiscal Year)	RACCT (Account Number)	RBUSA (Business Area)	RTCUR (Currency Key)	DRCRK (Debit/Credit Indicator)	...	transa pe cu (c
0	800	0	0	1	3000	1994	1000	1000	USD	S	...	0.00
1	800	0	0	1	3000	1994	1000	7000	USD	H	...	0.00
2	800	0	0	1	3000	1994	1000	7000	USD	S	...	0.00
3	800	0	0	1	3000	1994	1000	9900	USD	S	...	0.00
4	800	0	0	1	3000	1994	1010	1000	USD	H	...	0.00

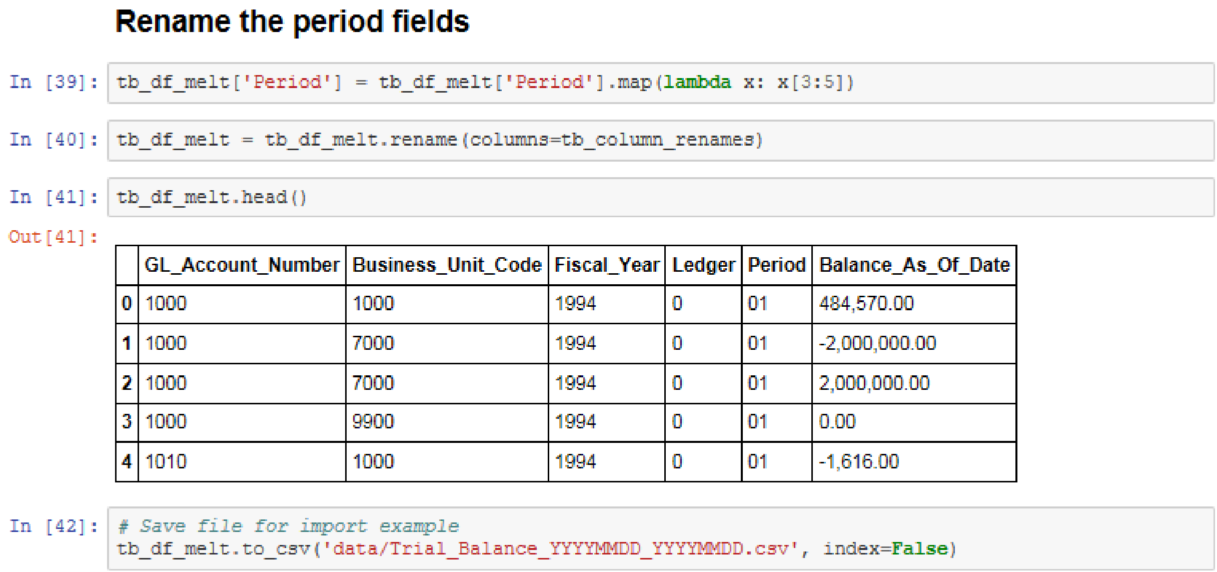
5 rows × 63 columns



Reshape the data to unpivot periods from columns to rows

```
In [37]: tb_column_renames = {
             'RACCT (Account Number)': 'GL_Account_Number',
             'RBUSA (Business Area)': 'Business_Unit_Code',
             'RYEAR (Fiscal Year)': 'Fiscal_Year',
             'RLDNR (Ledger)': 'Ledger'
         }
```

```
In [38]: tb_df_melt = tb_df.melt(id_vars=tb_column_renames.keys(),
             value_vars=['TSL01 (Total transactions of the period in transaction currency)',
                 'TSL02 (Total transactions of the period in transaction currency)',
                 'TSL03 (Total transactions of the period in transaction currency)',
                 'TSL04 (Total transactions of the period in transaction currency)',
                 'TSL05 (Total transactions of the period in transaction currency)',
                 'TSL06 (Total transactions of the period in transaction currency)',
                 'TSL07 (Total transactions of the period in transaction currency)',
                 'TSL08 (Total transactions of the period in transaction currency)',
                 'TSL09 (Total transactions of the period in transaction currency)',
                 'TSL10 (Total transactions of the period in transaction currency)',
                 'TSL11 (Total transactions of the period in transaction currency)',
                 'TSL12 (Total transactions of the period in transaction currency)'],
                 var_name='Period',
                 value_name='Balance_As_Of_Date')
```



Rename the period fields

```
In [39]: tb_df_melt['Period'] = tb_df_melt['Period'].map(lambda x: x[3:5])
```

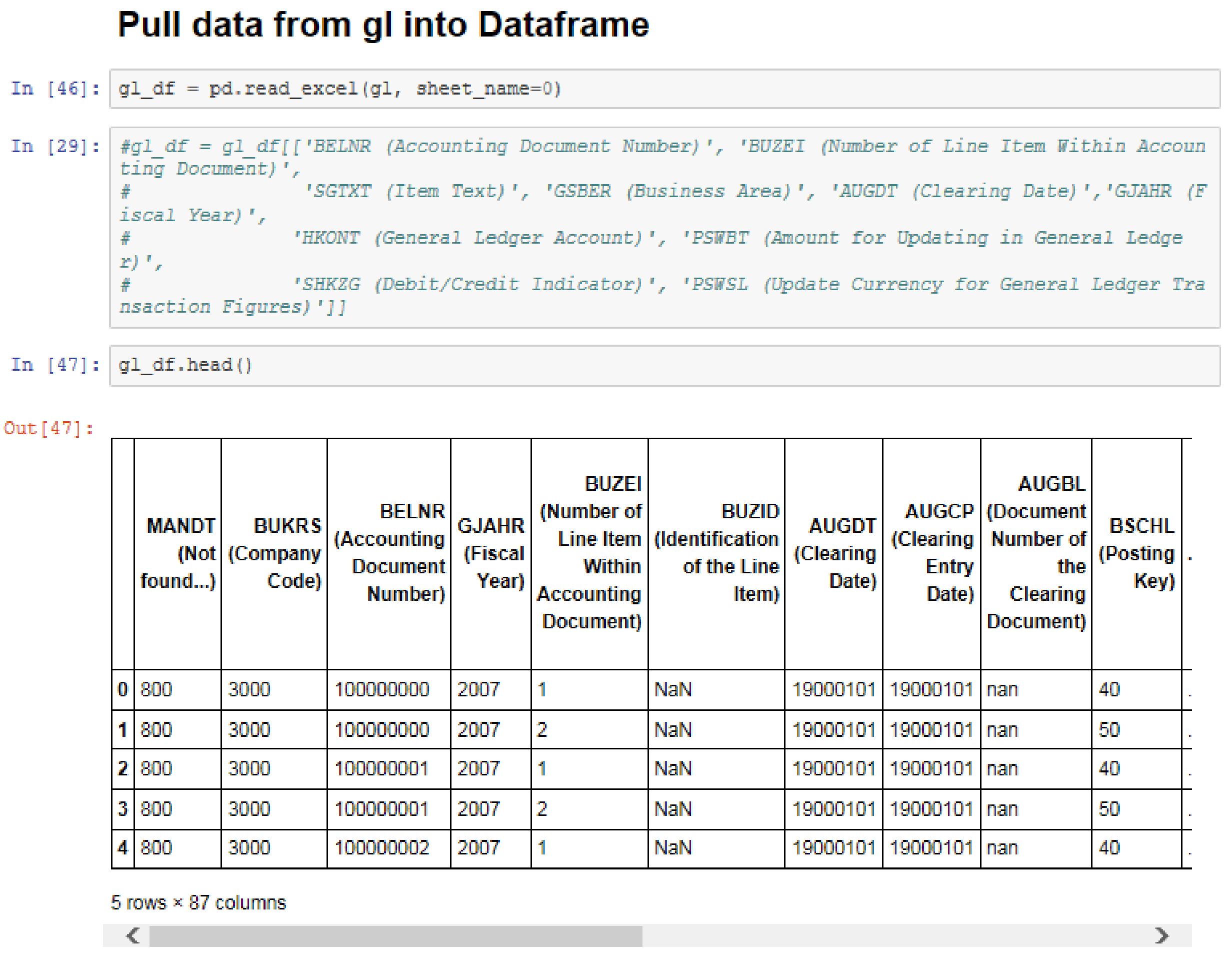
```
In [40]: tb_df_melt = tb_df_melt.rename(columns=tb_column_renames)
```

```
In [41]: tb_df_melt.head()
```

Out[41]:

	GL_Account_Number	Business_Unit_Code	Fiscal_Year	Ledger	Period	Balance_As_Of_Date
0	1000	1000	1994	0	01	484,570.00
1	1000	7000	1994	0	01	-2,000,000.00
2	1000	7000	1994	0	01	2,000,000.00
3	1000	9900	1994	0	01	0.00
4	1010	1000	1994	0	01	-1,616.00

```
In [42]: # Save file for import example
         tb_df_melt.to_csv('data/Trial_Balance_YYYYMMDD_YYYYMMDD.csv', index=False)
```



Pull data from gl into Dataframe

```
In [46]: gl_df = pd.read_excel(gl, sheet_name=0)
```

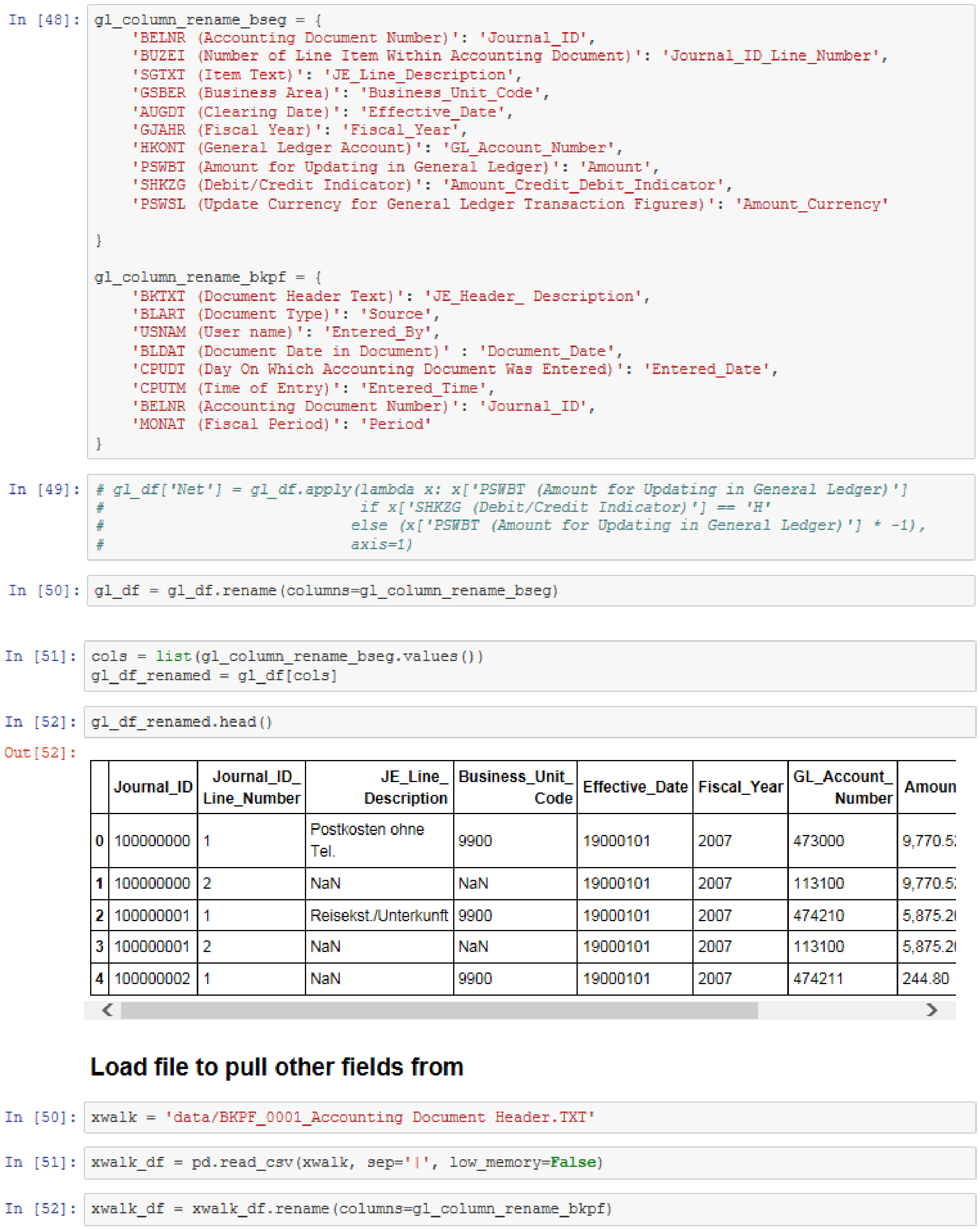
```
In [29]: #gl_df = gl_df[['BELNR (Accounting Document Number)', 'BUZEI (Number of Line Item Within Accoun
         ting Document)',
         #             'SGTXT (Item Text)', 'GSBER (Business Area)', 'AUGDT (Clearing Date)','GJAHR (F
         iscal Year)',
         #             'HKONT (General Ledger Account)', 'PSWBT (Amount for Updating in General Ledge
         r)',
         #             'SHKZG (Debit/Credit Indicator)', 'PSWSL (Update Currency for General Ledger Tra
         nsaction Figures)']]
```

```
In [47]: gl_df.head()
```

Out[47]:

	MANDT (Not found...)	BUKRS (Company Code)	BELNR (Accounting Document Number)	GJAHR (Fiscal Year)	BUZEI (Number of Line Item Within Accounting Document)	BUZID (Identification of the Line Item)	AUGDT (Clearing Date)	AUGCP (Clearing Entry Date)	AUGBL (Document Number of the Clearing Document)	BSCHL (Posting Key)	.
0	800	3000	100000000	2007	1	NaN	19000101	19000101	nan	40	.
1	800	3000	100000000	2007	2	NaN	19000101	19000101	nan	50	.
2	800	3000	100000001	2007	1	NaN	19000101	19000101	nan	40	.
3	800	3000	100000001	2007	2	NaN	19000101	19000101	nan	50	.
4	800	3000	100000002	2007	1	NaN	19000101	19000101	nan	40	.

5 rows × 87 columns



```
In [48]: gl_column_rename_bseg = {
             'BELNR (Accounting Document Number)': 'Journal_ID',
             'BUZEI (Number of Line Item Within Accounting Document)': 'Journal_ID_Line_Number',
             'SGTXT (Item Text)': 'JE_Line_Description',
             'GSBER (Business Area)': 'Business_Unit_Code',
             'AUGDT (Clearing Date)': 'Effective_Date',
             'GJAHR (Fiscal Year)': 'Fiscal_Year',
             'HKONT (General Ledger Account)': 'GL_Account_Number',
             'PSWBT (Amount for Updating in General Ledger)': 'Amount',
             'SHKZG (Debit/Credit Indicator)': 'Amount_Credit_Debit_Indicator',
             'PSWSL (Update Currency for General Ledger Transaction Figures)': 'Amount_Currency'

         }

         gl_column_rename_bkpf = {
             'BKTXT (Document Header Text)': 'JE_Header_ Description',
             'BLART (Document Type)': 'Source',
             'USNAM (User name)': 'Entered_By',
             'BLDAT (Document Date in Document)' : 'Document_Date',
             'CPUDT (Day On Which Accounting Document Was Entered)': 'Entered_Date',
             'CPUTM (Time of Entry)': 'Entered_Time',
             'BELNR (Accounting Document Number)': 'Journal_ID',
             'MONAT (Fiscal Period)': 'Period'
         }
```

```
In [49]: # gl_df['Net'] = gl_df.apply(lambda x: x['PSWBT (Amount for Updating in General Ledger)']
         #                              if x['SHKZG (Debit/Credit Indicator)'] == 'H'
         #                             else (x['PSWBT (Amount for Updating in General Ledger)'] * -1),
         #                             axis=1)
```

```
In [50]: gl_df = gl_df.rename(columns=gl_column_rename_bseg)
```

```
In [51]: cols = list(gl_column_rename_bseg.values())
         gl_df_renamed = gl_df[cols]
```

```
In [52]: gl_df_renamed.head()
```

Out[52]:

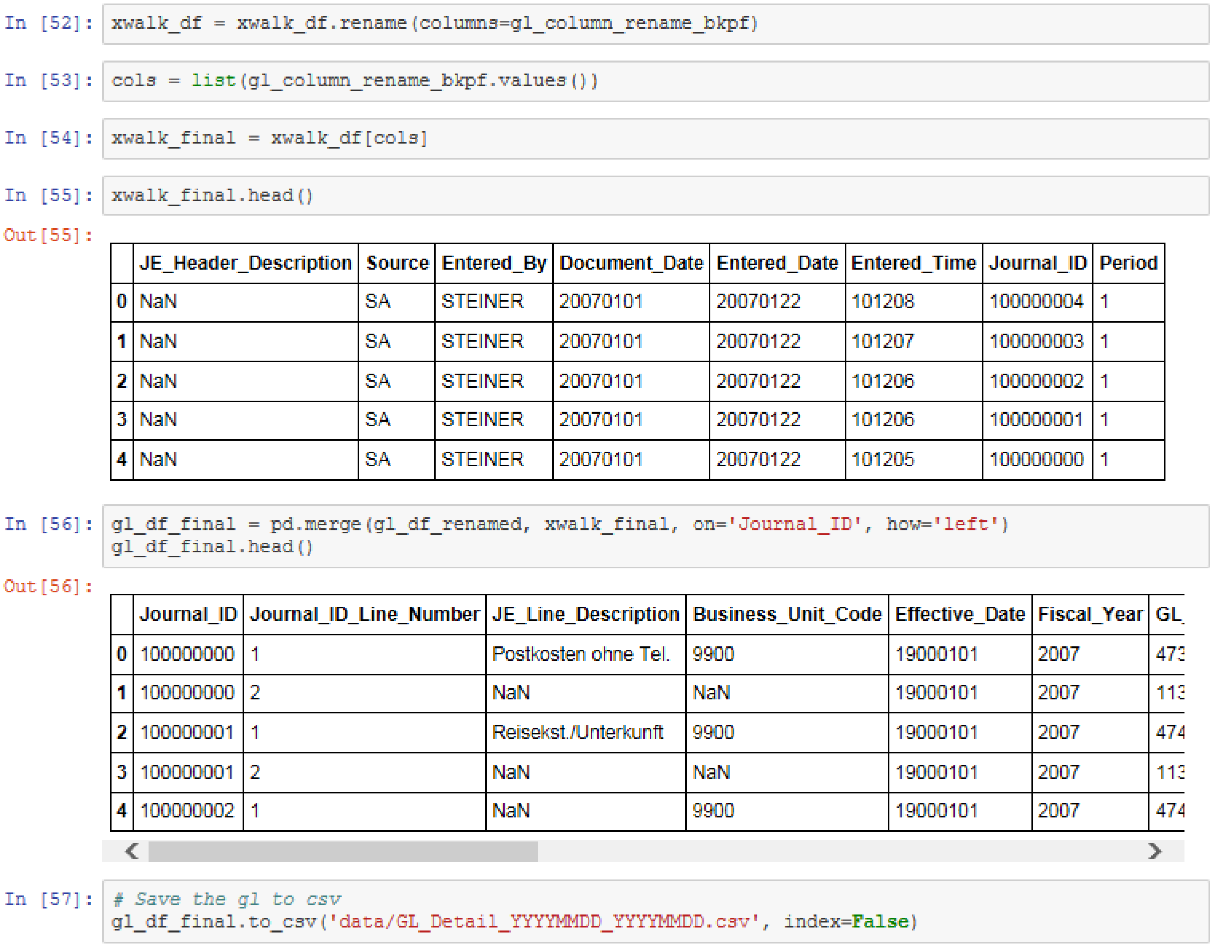
	Journal_ID	Journal_ID_ Line_Number	JE_Line_ Description	Business_Unit_ Code	Effective_Date	Fiscal_Year	GL_Account_ Number	Amoun
0	100000000	1	Postkosten ohne Tel.	9900	19000101	2007	473000	9,770.5
1	100000000	2	NaN	NaN	19000101	2007	113100	9,770.5
2	100000001	1	Reisekst./Unterkunft	9900	19000101	2007	474210	5,875.2
3	100000001	2	NaN	NaN	19000101	2007	113100	5,875.2
4	100000002	1	NaN	9900	19000101	2007	474211	244.80

Load file to pull other fields from

```
In [50]: xwalk = 'data/BKPF_0001_Accounting Document Header.TXT'
```

```
In [51]: xwalk_df = pd.read_csv(xwalk, sep='|', low_memory=False)
```

```
In [52]: xwalk_df = xwalk_df.rename(columns=gl_column_rename_bkpf)
```



```
In [52]: xwalk_df = xwalk_df.rename(columns=gl_column_rename_bkpf)
```

```
In [53]: cols = list(gl_column_rename_bkpf.values())
```

```
In [54]: xwalk_final = xwalk_df[cols]
```

```
In [55]: xwalk_final.head()
```

Out[55]:

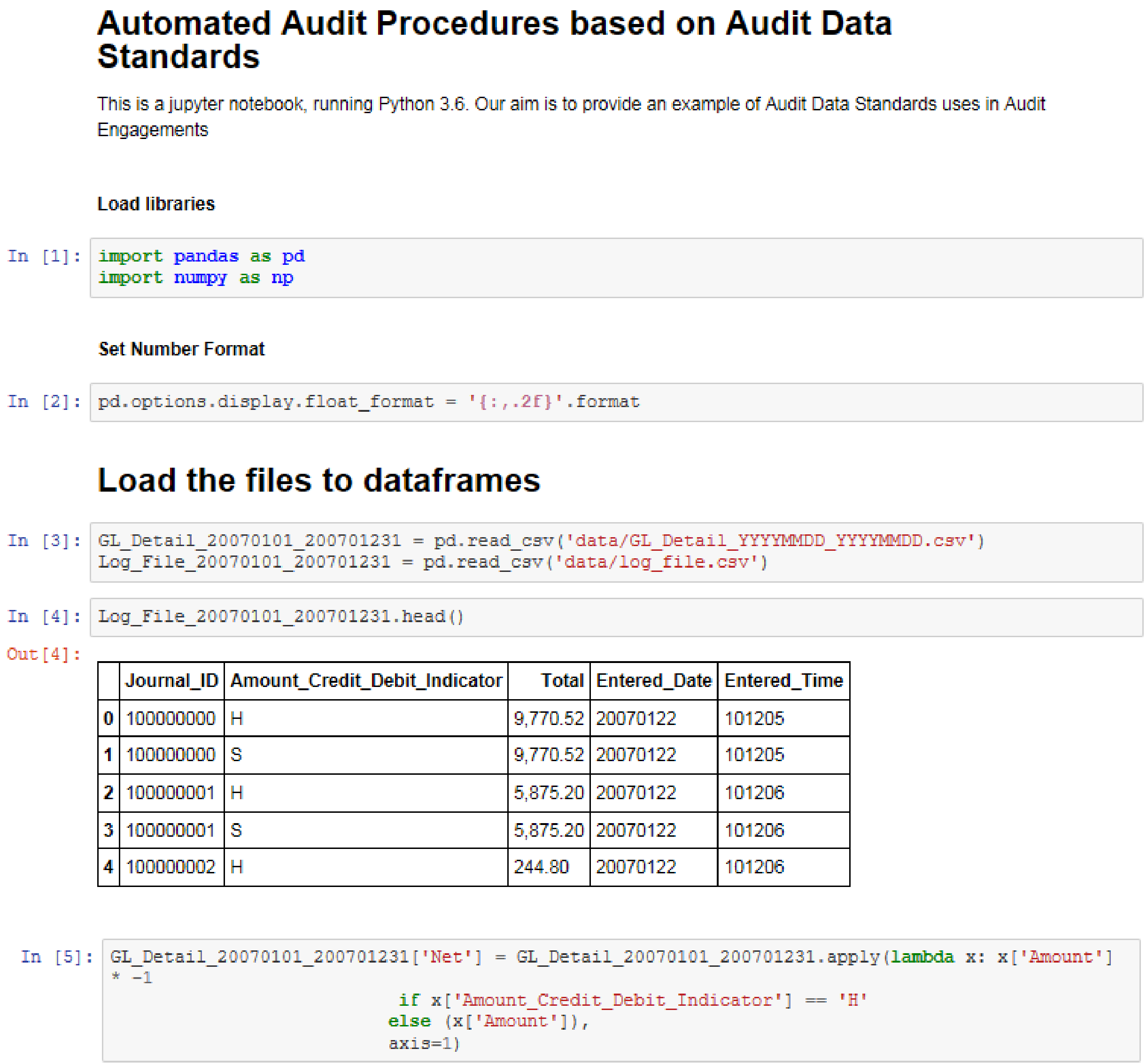
	JE_Header_Description	Source	Entered_By	Document_Date	Entered_Date	Entered_Time	Journal_ID	Period
0	NaN	SA	STEINER	20070101	20070122	101208	100000004	1
1	NaN	SA	STEINER	20070101	20070122	101207	100000003	1
2	NaN	SA	STEINER	20070101	20070122	101206	100000002	1
3	NaN	SA	STEINER	20070101	20070122	101206	100000001	1
4	NaN	SA	STEINER	20070101	20070122	101205	100000000	1

```
In [56]: gl_df_final = pd.merge(gl_df_renamed, xwalk_final, on='Journal_ID', how='left')
         gl_df_final.head()
```

Out[56]:

	Journal_ID	Journal_ID_Line_Number	JE_Line_Description	Business_Unit_Code	Effective_Date	Fiscal_Year	GL_
0	100000000	1	Postkosten ohne Tel.	9900	19000101	2007	473
1	100000000	2	NaN	NaN	19000101	2007	113
2	100000001	1	Reisekst./Unterkunft	9900	19000101	2007	474
3	100000001	2	NaN	NaN	19000101	2007	113
4	100000002	1	NaN	9900	19000101	2007	474

```
In [57]: # Save the gl to csv
         gl_df_final.to_csv('data/GL_Detail_YYYYMMDD_YYYYMMDD.csv', index=False)
```



Automated Audit Procedures based on Audit Data Standards

This is a jupyter notebook, running Python 3.6. Our aim is to provide an example of Audit Data Standards uses in Audit Engagements

Load libraries

```
In [1]: import pandas as pd
        import numpy as np
```

Set Number Format

```
In [2]: pd.options.display.float_format = '{:,.2f}'.format
```

Load the files to dataframes

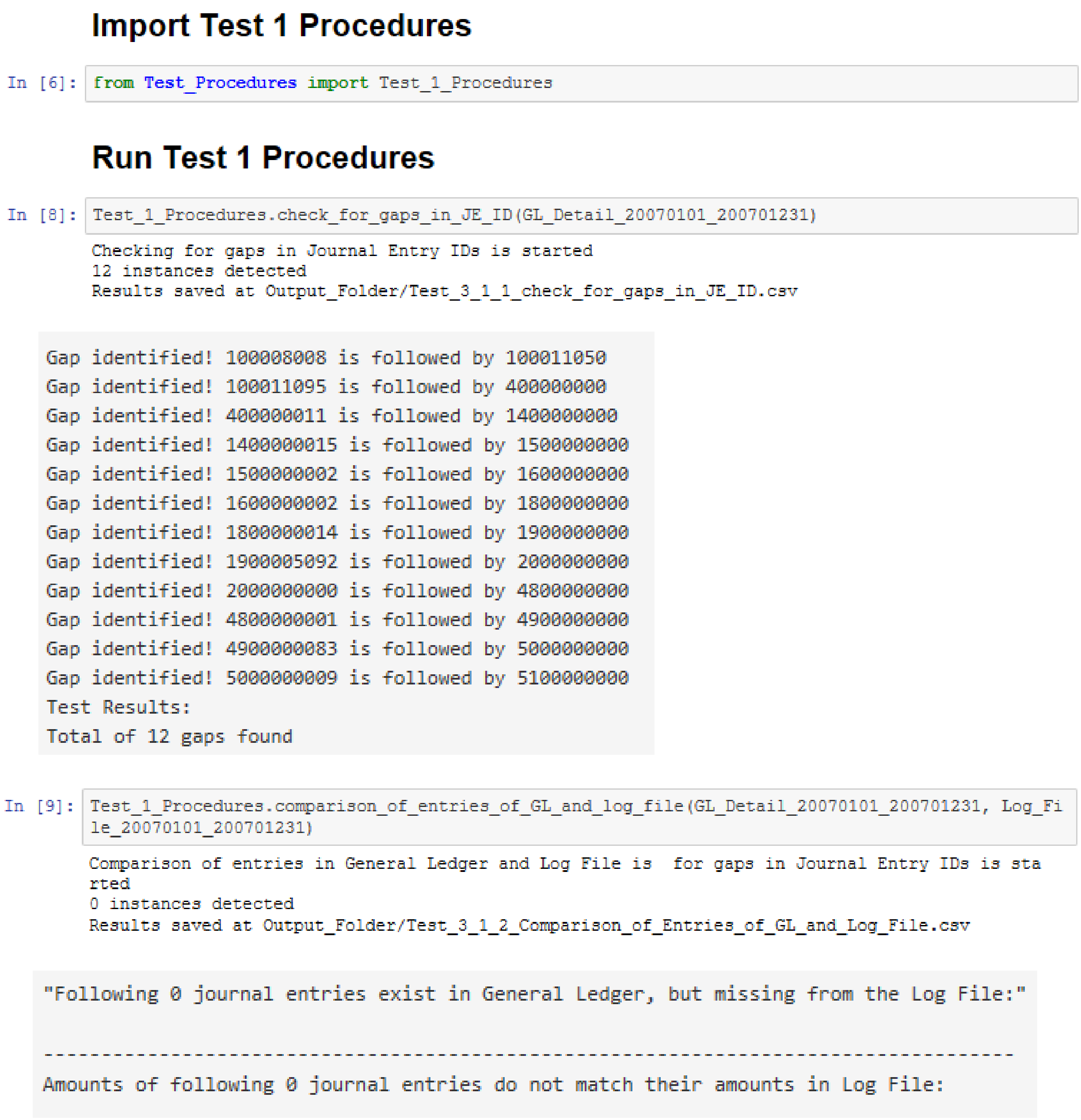
```
In [3]: GL_Detail_20070101_200701231 = pd.read_csv('data/GL_Detail_YYYYMMDD_YYYYMMDD.csv')
        Log_File_20070101_200701231 = pd.read_csv('data/log_file.csv')
```

```
In [4]: Log_File_20070101_200701231.head()
```

Out[4]:

	Journal_ID	Amount_Credit_Debit_Indicator	Total	Entered_Date	Entered_Time
0	100000000	H	9,770.52	20070122	101205
1	100000000	S	9,770.52	20070122	101205
2	100000001	H	5,875.20	20070122	101206
3	100000001	S	5,875.20	20070122	101206
4	100000002	H	244.80	20070122	101206

```
In [5]: GL_Detail_20070101_200701231['Net'] = GL_Detail_20070101_200701231.apply(lambda x: x['Amount']
        * -1
                                  if x['Amount_Credit_Debit_Indicator'] == 'H'
                                 else (x['Amount']),
                                 axis=1)
```



Import Test 1 Procedures

```
In [6]: from Test_Procedures import Test_1_Procedures
```

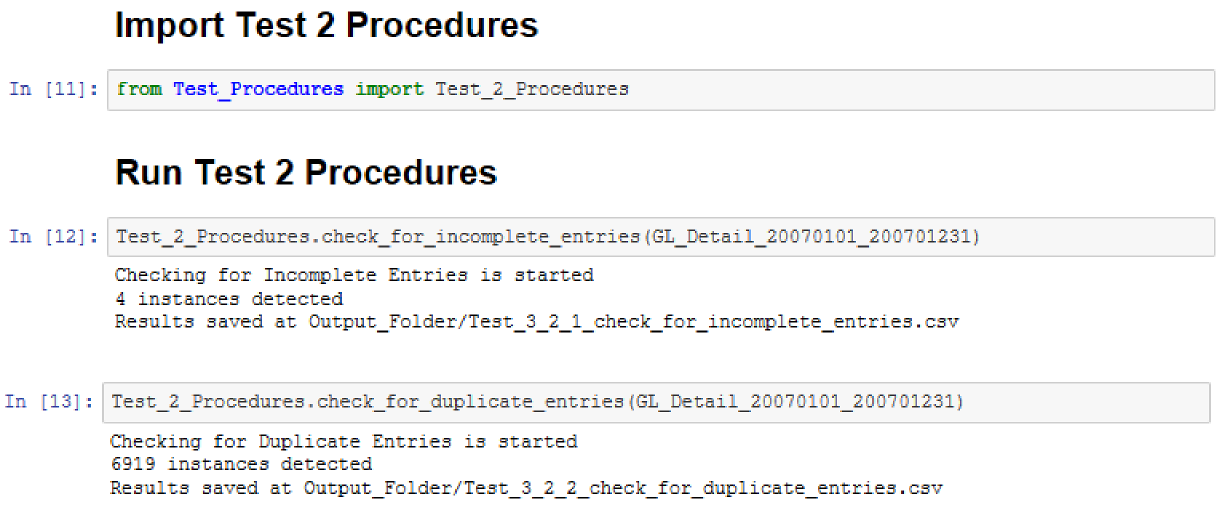
Run Test 1 Procedures

```
In [8]: Test_1_Procedures.check_for_gaps_in_JE_ID(GL_Detail_20070101_200701231)

        Checking for gaps in Journal Entry IDs is started
        12 instances detected
        Results saved at Output_Folder/Test_3_1_1_check_for_gaps_in_JE_ID.csv
```

```
Gap identified! 100008008 is followed by 100011050
Gap identified! 100011095 is followed by 400000000
Gap identified! 400000011 is followed by 1400000000
Gap identified! 1400000015 is followed by 1500000000
Gap identified! 1500000002 is followed by 1600000000
Gap identified! 1600000002 is followed by 1800000000
Gap identified! 1800000014 is followed by 1900000000
Gap identified! 1900005092 is followed by 2000000000
Gap identified! 2000000000 is followed by 4800000000
Gap identified! 4800000001 is followed by 4900000000
Gap identified! 4900000083 is followed by 5000000000
Gap identified! 5000000009 is followed by 5100000000
Test Results:
Total of 12 gaps found
```

```
In [9]: Test_1_Procedures.comparison_of_entries_of_GL_and_log_file(GL_Detail_20070101_200701231, Log_Fi
        le_20070101_200701231)

        Comparison of entries in General Ledger and Log File is  for gaps in Journal Entry IDs is sta
        rted
        0 instances detected
        Results saved at Output_Folder/Test_3_1_2_Comparison_of_Entries_of_GL_and_Log_File.csv
```

```
"Following 0 journal entries exist in General Ledger, but missing from the Log File:"


-------------------------------------------------------------------------------------
Amounts of following 0 journal entries do not match their amounts in Log File:
```



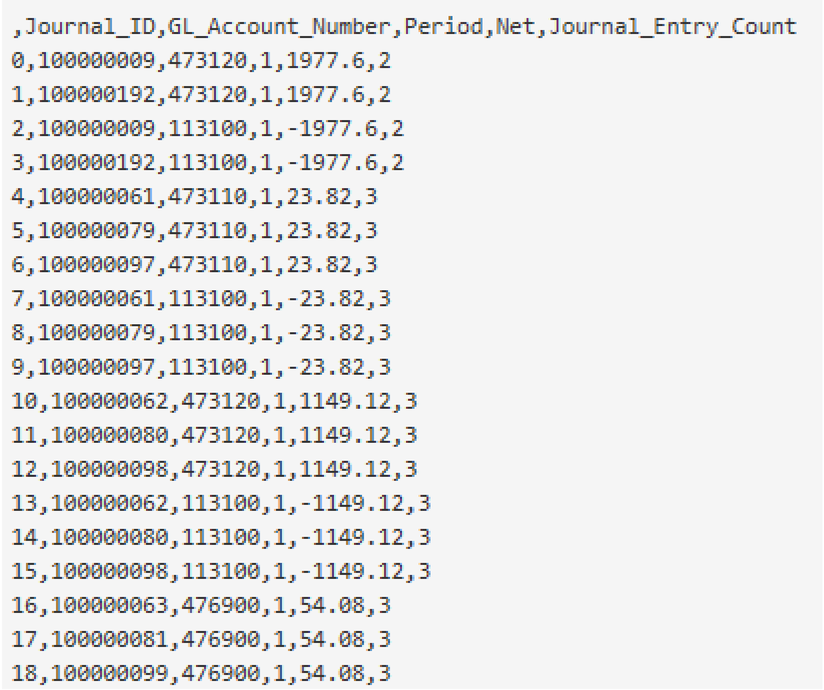
Import Test 2 Procedures

```
In [11]: from Test_Procedures import Test_2_Procedures
```

Run Test 2 Procedures

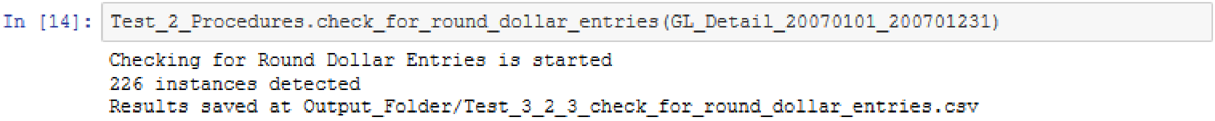
```
In [12]: Test_2_Procedures.check_for_incomplete_entries(GL_Detail_20070101_200701231)

         Checking for Incomplete Entries is started
         4 instances detected
         Results saved at Output_Folder/Test_3_2_1_check_for_incomplete_entries.csv
```

```
In [13]: Test_2_Procedures.check_for_duplicate_entries(GL_Detail_20070101_200701231)

         Checking for Duplicate Entries is started
         6919 instances detected
         Results saved at Output_Folder/Test_3_2_2_check_for_duplicate_entries.csv
```

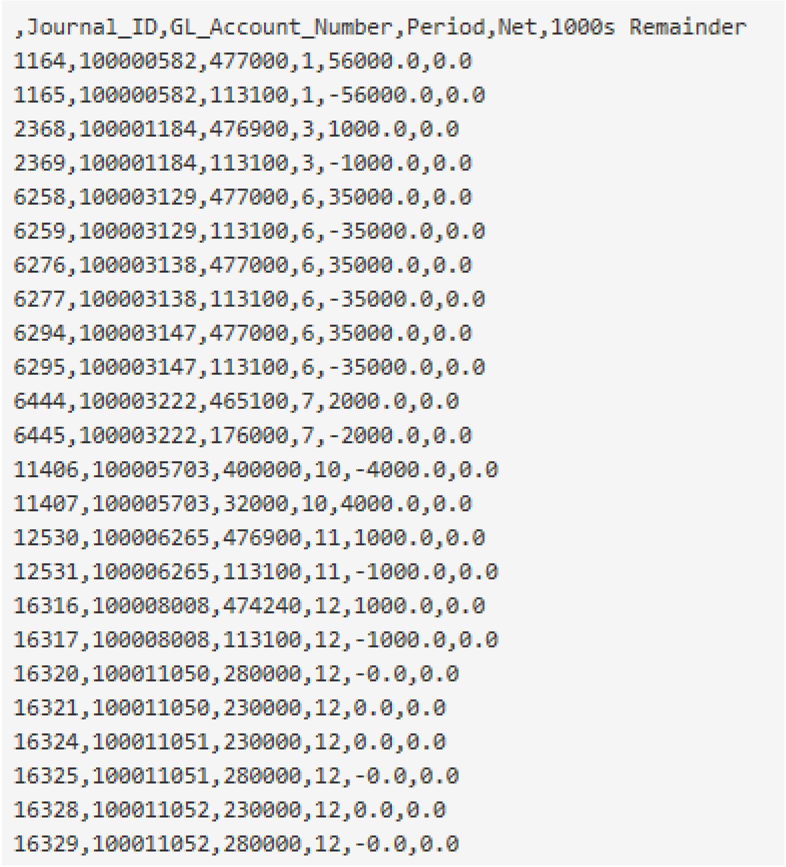


```
,Journal_ID,GL_Account_Number,Period,Net,Journal_Entry_Count
0,100000009,473120,1,1977.6,2
1,100000192,473120,1,1977.6,2
2,100000009,113100,1,-1977.6,2
3,100000192,113100,1,-1977.6,2
4,100000061,473110,1,23.82,3
5,100000079,473110,1,23.82,3
6,100000097,473110,1,23.82,3
7,100000061,113100,1,-23.82,3
8,100000079,113100,1,-23.82,3
9,100000097,113100,1,-23.82,3
10,100000062,473120,1,1149.12,3
11,100000080,473120,1,1149.12,3
12,100000098,473120,1,1149.12,3
13,100000062,113100,1,-1149.12,3
14,100000080,113100,1,-1149.12,3
15,100000098,113100,1,-1149.12,3
16,100000063,476900,1,54.08,3
17,100000081,476900,1,54.08,3
18,100000099,476900,1,54.08,3
```

(***Please note that the preceding results are a portion of the 6,919 items.)***

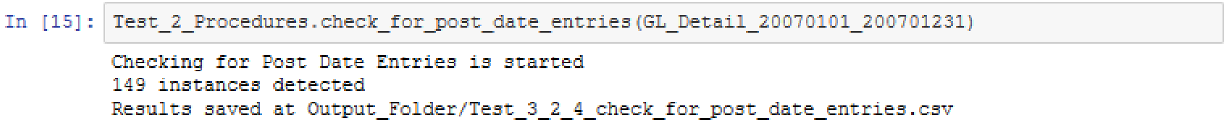


```
In [14]: Test_2_Procedures.check_for_round_dollar_entries(GL_Detail_20070101_200701231)

         Checking for Round Dollar Entries is started
         226 instances detected
         Results saved at Output_Folder/Test_3_2_3_check_for_round_dollar_entries.csv
```

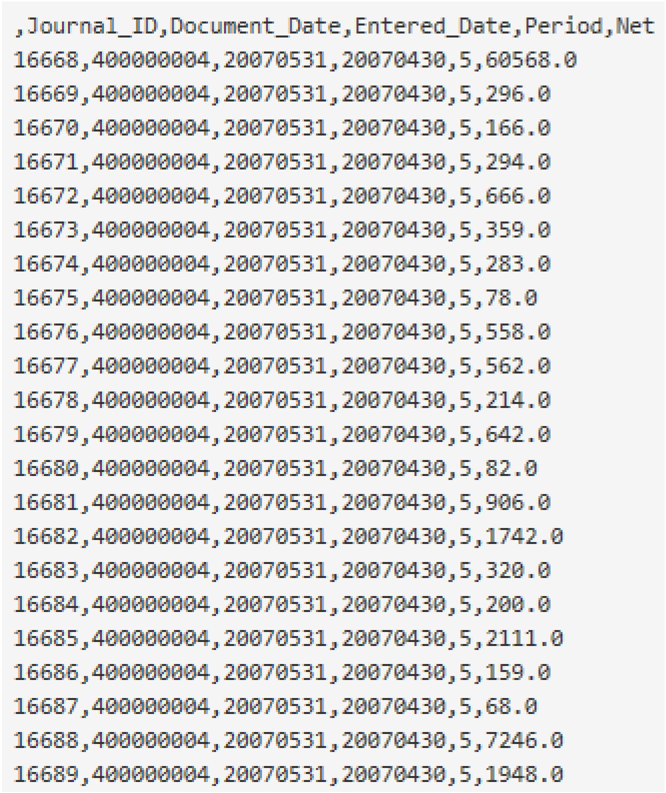


```
,Journal_ID,GL_Account_Number,Period,Net,1000s Remainder
1164,100000582,477000,1,56000.0,0.0
1165,100000582,113100,1,-56000.0,0.0
2368,100001184,476900,3,1000.0,0.0
2369,100001184,113100,3,-1000.0,0.0
6258,100003129,477000,6,35000.0,0.0
6259,100003129,113100,6,-35000.0,0.0
6276,100003138,477000,6,35000.0,0.0
6277,100003138,113100,6,-35000.0,0.0
6294,100003147,477000,6,35000.0,0.0
6295,100003147,113100,6,-35000.0,0.0
6444,100003222,465100,7,2000.0,0.0
6445,100003222,176000,7,-2000.0,0.0
11406,100005703,400000,10,-4000.0,0.0
11407,100005703,32000,10,4000.0,0.0
12530,100006265,476900,11,1000.0,0.0
12531,100006265,113100,11,-1000.0,0.0
16316,100008008,474240,12,1000.0,0.0
16317,100008008,113100,12,-1000.0,0.0
16320,100011050,280000,12,-0.0,0.0
16321,100011050,230000,12,0.0,0.0
16324,100011051,230000,12,0.0,0.0
16325,100011051,280000,12,-0.0,0.0
16328,100011052,230000,12,0.0,0.0
16329,100011052,280000,12,-0.0,0.0
```

(***Please note that the preceding results are a portion of the 226 items.)***

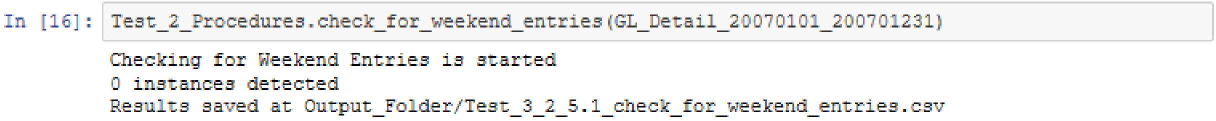


```
In [15]: Test_2_Procedures.check_for_post_date_entries(GL_Detail_20070101_200701231)

         Checking for Post Date Entries is started
         149 instances detected
         Results saved at Output_Folder/Test_3_2_4_check_for_post_date_entries.csv
```

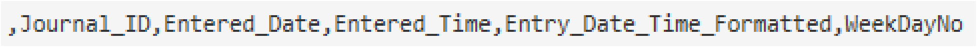


```
,Journal_ID,Document_Date,Entered_Date,Period,Net
16668,400000004,20070531,20070430,5,60568.0
16669,400000004,20070531,20070430,5,296.0
16670,400000004,20070531,20070430,5,166.0
16671,400000004,20070531,20070430,5,294.0
16672,400000004,20070531,20070430,5,666.0
16673,400000004,20070531,20070430,5,359.0
16674,400000004,20070531,20070430,5,283.0
16675,400000004,20070531,20070430,5,78.0
16676,400000004,20070531,20070430,5,558.0
16677,400000004,20070531,20070430,5,562.0
16678,400000004,20070531,20070430,5,214.0
16679,400000004,20070531,20070430,5,642.0
16680,400000004,20070531,20070430,5,82.0
16681,400000004,20070531,20070430,5,906.0
16682,400000004,20070531,20070430,5,1742.0
16683,400000004,20070531,20070430,5,320.0
16684,400000004,20070531,20070430,5,200.0
16685,400000004,20070531,20070430,5,2111.0
16686,400000004,20070531,20070430,5,159.0
16687,400000004,20070531,20070430,5,68.0
16688,400000004,20070531,20070430,5,7246.0
16689,400000004,20070531,20070430,5,1948.0
```

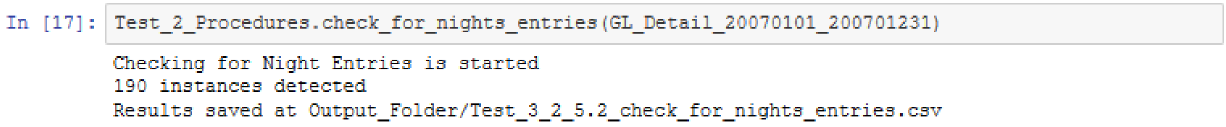
(***Please note that the preceding results are a portion of the 149 items.)***



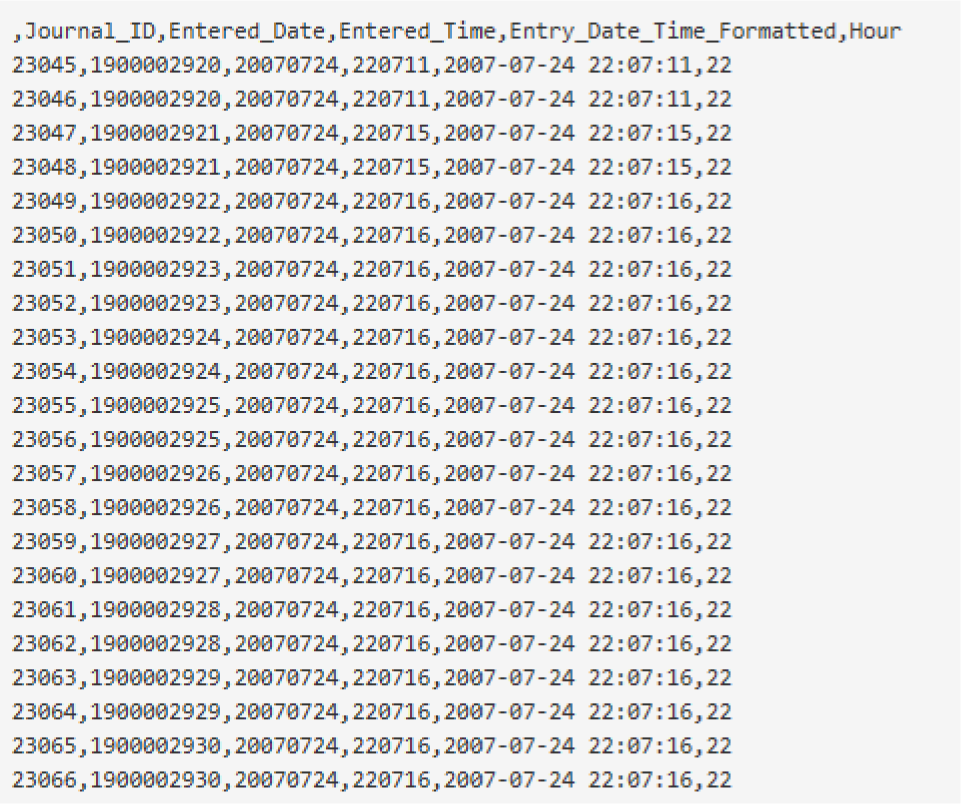
```
In [16]: Test_2_Procedures.check_for_weekend_entries(GL_Detail_20070101_200701231)

         Checking for Weekend Entries is started
         0 instances detected
         Results saved at Output_Folder/Test_3_2_5.1_check_for_weekend_entries.csv
```



```
,Journal_ID,Entered_Date,Entered_Time,Entry_Date_Time_Formatted,WeekDayNo
```



```
In [17]: Test_2_Procedures.check_for_nights_entries(GL_Detail_20070101_200701231)

         Checking for Night Entries is started
         190 instances detected
         Results saved at Output_Folder/Test_3_2_5.2_check_for_nights_entries.csv
```



```
,Journal_ID,Entered_Date,Entered_Time,Entry_Date_Time_Formatted,Hour
23045,1900002920,20070724,220711,2007-07-24 22:07:11,22
23046,1900002920,20070724,220711,2007-07-24 22:07:11,22
23047,1900002921,20070724,220715,2007-07-24 22:07:15,22
23048,1900002921,20070724,220715,2007-07-24 22:07:15,22
23049,1900002922,20070724,220716,2007-07-24 22:07:16,22
23050,1900002922,20070724,220716,2007-07-24 22:07:16,22
23051,1900002923,20070724,220716,2007-07-24 22:07:16,22
23052,1900002923,20070724,220716,2007-07-24 22:07:16,22
23053,1900002924,20070724,220716,2007-07-24 22:07:16,22
23054,1900002924,20070724,220716,2007-07-24 22:07:16,22
23055,1900002925,20070724,220716,2007-07-24 22:07:16,22
23056,1900002925,20070724,220716,2007-07-24 22:07:16,22
23057,1900002926,20070724,220716,2007-07-24 22:07:16,22
23058,1900002926,20070724,220716,2007-07-24 22:07:16,22
23059,1900002927,20070724,220716,2007-07-24 22:07:16,22
23060,1900002927,20070724,220716,2007-07-24 22:07:16,22
23061,1900002928,20070724,220716,2007-07-24 22:07:16,22
23062,1900002928,20070724,220716,2007-07-24 22:07:16,22
23063,1900002929,20070724,220716,2007-07-24 22:07:16,22
23064,1900002929,20070724,220716,2007-07-24 22:07:16,22
23065,1900002930,20070724,220716,2007-07-24 22:07:16,22
23066,1900002930,20070724,220716,2007-07-24 22:07:16,22
```

(***Please note that the preceding results are a portion of the 190 items.)***

VII. Appendix B — Helpful Resources

Python Resources:

Python

Beginner's Guide to Python

Microsoft Azure Notebooks

Jupyter

AICPA Resources:

Audit Data Analytics

Audit Data Standards

Rutgers AICPA Data Analytics Research Initiative