Introduction to Data Science in Accounting – Practice Set 2

In this set we will use our newly learned Python skills to implement an IT control.

Step 1: Import an Excel spreadsheet using the Pandas library

Step 2: Writing a three-way match program

Step 3: Successfully running the three-way match on our dataset

By the end of the set, you should be able to carry out a three-way match on a spreadsheet of business transaction data.

Difficulty Level: Easy-Moderate

Step 1: Importing an Excel spreadsheet

Our first is to import an Excel spreadsheet in Python. There is a function in Python that can do this, but it is not part of standard Python. Instead, this function is part of a *library*, which is a collection of functions that others have built before us. We need to import this libary into Python first, and then we can use the function from this library to import the spreadsheet.

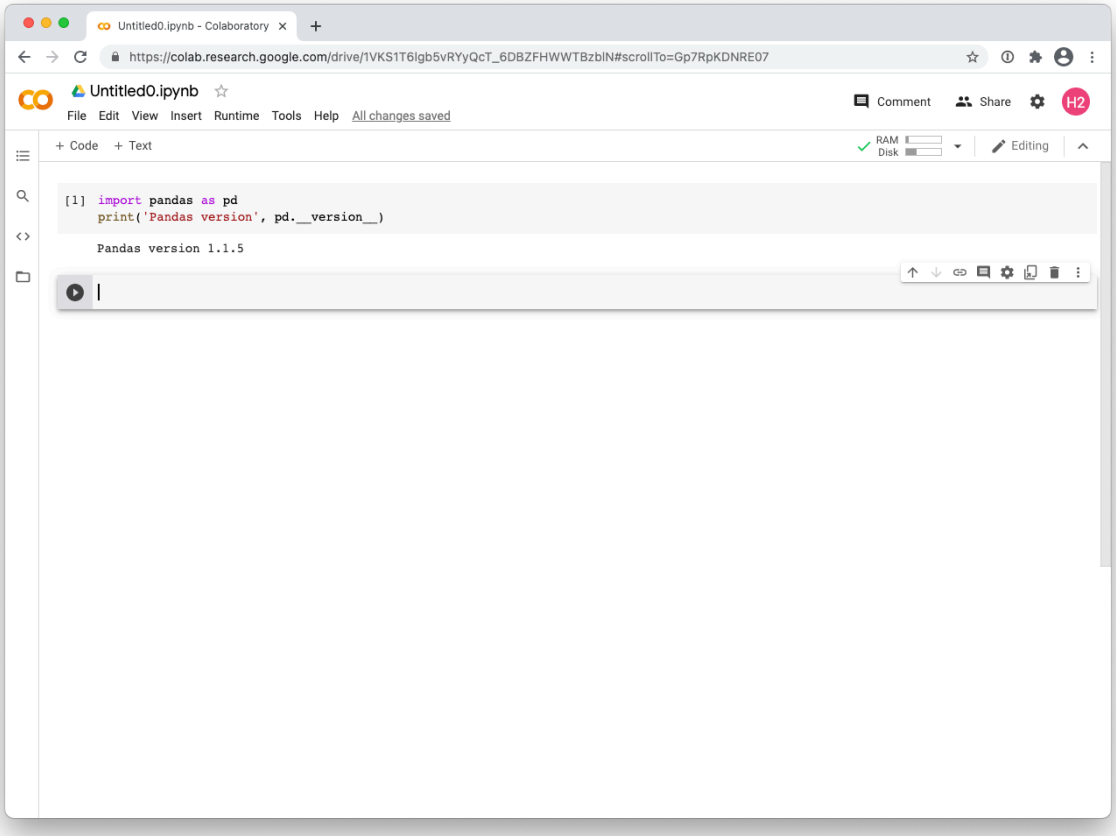
To import spreadsheets in Python we will be using a library called Pandas.

First, create a new Jupyter notebook in the Python environment of your choice (Colab or local). To import the Pandas library into Python, start with the following:

```
import pandas as pd
print('Pandas version', pd.__version__)
```

The 'pd' is a common abbreviation for 'pandas'. When we use a function from the library, we have to prefix the name of that function with the name of the library, like so: `library.function.` By using the abbreviation, we don't have to type `pandas.__version__` but instead can use `pd.__version__`.

The __version__ is a special variable that all libraries have. There are two underscore chars at the front, and two at the back. Printing out the version of the library is good practice so that others who look at the same output know can potentially reproduce the same results (results might be different when a different version of the library is used).



Now identify the last digit of your student ID. We will assume it is 0 in this exercise. Add it to the workbook like so:

```
student_id_ends_with = '0'
print('Your student ID ends with:', student_id_ends_with)
```

Replace the '0' with the last digit of your student ID, (or keep it at 0 if the last digit of your student ID is 0). We need this last digit so you can download the correct spreadsheet. There are ten spreadsheets available. Continue by entering this into seperate cells:

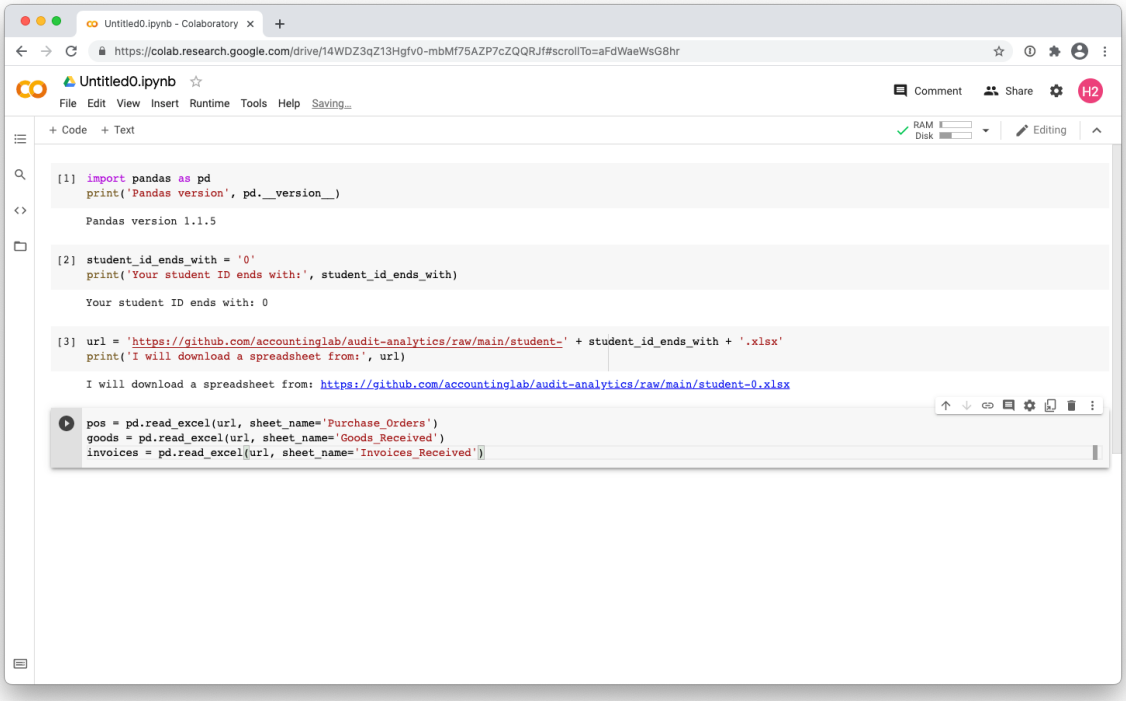
```
url = 'https://github.com/accountinglab/audit-
analytics/raw/main/student-' + student_id_ends_with + '.xlsx'
print('I will download a spreadsheet from:', url)
```

You now have a 'url' variable that has the correct link to download the spreadsheet. Colab will print it for you if you run this cell.

Importing the spreadsheets into Python is done by using the read_excel function from Pandas. We call it like this (all in one cell).

```
pos = pd.read_excel(url, sheet_name='Purchase_Orders')
goods = pd.read_excel(url, sheet_name='Goods_Received')
invoices = pd.read_excel(url, sheet_name='Invoices_Received')
```

If you have done everything correctly, your notebook will now look like this:



The variables `purchase_orders`, `goods_received`, and `invoices_received` hold *data frames*, which are the Pandas equivalent of an Excel worksheet. There are three data frames, one for every worksheet.

Let's look at some rows of these datasets. This is just for visual inspection. The Pandas method for this is `sample`(n=X), with X being the number of random rows you would like to see. Here we choose 5 rows, which will be different each time you run this code (try going back to the cell and pressing shift-Enter again).

```
[6] pos.sample(n=5)
```

od	Business_Unit_Code	Supplier_Account_ID	Entered_By	Purchase_Order_Amount_Local	Purchase_Order_Local_Curren
M2	Dairy	Violet & Co	Procurement	1835.35	U
M4	Dairy	Lavender Ltd	Procurement	1285.20	U
M1	Dairy	Lavender Ltd	Procurement	16287.82	U
M5	Dairy	Orchid & Sons	Procurement	5909.74	U
M1	Dairy	Iris & Sons	Procurement	6531.72	U

```
[7] goods.sample(n=5)
```

_Account_ID	Business_Unit_Code	Purchase_Order_ID	Purchase_Order_Date	Entered_By	Receipt_Amount_Local	Receip
Azalea & Sons	Dairy	4500000067	2020-01-10	Warehouse	1887.05	
Iris & Sons	Dairy	4500000297	2020-02-13	Warehouse	1499.30	
Hibiscus Plc	Dairy	4500000782	2020-04-24	Warehouse	5645.70	
nolia Importers	Dairy	4500000438	2020-03-05	Warehouse	3465.45	
Iris & Sons	Dairy	4500000778	2020-04-24	Warehouse	11796.36	

```
[8] invoices.sample(n=5)
```

	Invoice_ID	Invoice_Number	Invoice Date	Invoice_Fiscal_Year	Invoice_Period	Invoice_Due_Date	Purchase_Order_
160	70174	70174	2020-01-27	2020	M1	2020-04-26	45000001
351	70368	70368	2020-02-27	2020	M2	2020-05-27	45000003
130	70144	70144	2020-01-22	2020	M1	2020-04-21	45000001
963	70988	70988	2020-05-26	2020	M5	2020-08-24	45000009
119	70133	70133	2020-01-21	2020	M1	2020-04-20	45000001

If you scroll through these rows you will occasionally see a value called NaN. This means: Not a Number, and is the equivalent of a blank cell in Excel.

Slicing and dicing data frames

In data science, it is important to know how to slice and dice your data set. Slicing means selecting different rows, and dicing means selecting different columns.

Looking up a value is essentially slicing the rows and dicing the columns so that one value remains (the value to be looked up).

For our three-way match program, we are going to be looking up particular values. For example, we would like to know the purchase order amount given a purchase order ID. You can do this using the following statement:

```
po = 4500000011
pos[pos['Purchase_Order_ID']==po]['Purchase_Order_Amount_Local'].val
ues[0]
```

Let's break this down so we know what is going on:

```
po = 4500000011
```

Here we assign a value to the variable po.

```
pos[pos['Purchase_Order_ID']==po]
```

This is the slice. These are the rows of the pos table where the column 'Purchase_Order_ID is equal to po. In our case there will be only one row.

```
pos[pos['Purchase_Order_ID']==po]['Purchase_Order_Amount_Local']
```

This is the dice. From the rows where po is 4500000011, select the values from the column 'Purchase_Order_Amount_Local. This will result in a single row with a single column.

```
pos[pos['Purchase_Order_ID']==po]['Purchase_Order_Amount_Local'].values[0]
```

values[0] means that we will select the first value of the rows available (there was only one row available, so it's the first value of the first row).

Try and experiment with different values and different columns to ensure you are comfortable with slicing and dicing a Pandas data frame.

Step 2: Writing a three-way match program

We will build the three-way match program in a few steps.

The data that we need is in the invoices data frame. So we will focus on this data frame only for now.

The first thing to do is to create a loop that goes through each row of invoice data.

We'll start out with a simple loop as follows:

```
for index, invoice in invoices.iterrows():
   print(invoice['Invoice_ID'])
```

Don't forget the indent on the second line, it signifies what needs to be done with every invoice.

Make sure you have the uppercase and lowercase exactly as written. Even something as this, with an innocent typo, will not work:

```
for index, invoice in invoices.iterrows():
    print(invoice['Invoice_id'])
```

You can perhaps spot the error, `id` on the second line was lowercase and it should have been uppercase. This statement will produce an error.

The result of the correct statement will be a print-out of 1,000 invoice numbers! Not very helpful, but it was just the start and we will build on this foundation. You can go back to the cell, and amend it to look like this[1]:

```
for index, invoice in invoices.iterrows():
     invoice_amount = invoice['Invoice_Amount_Local']
     po = invoice['Purchase_Order_ID']
     po_amount = pos[pos['Purchase_Order_ID']==po]['Purchase_Order_Amount_Local'].values[0]
     is_po_ok = invoice_amount == po_amount
     if not is_po_ok:
          print('PO anomaly for invoice:', invoice['Invoice_ID'])
     goods_amount = goods[goods['Purchase_Order_ID']==po]['Receipt_Amount_Local'].values[0]
     is_goods_ok = invoice_amount == goods_amount
     if not is_goods_ok:
          print('Goods anomaly for invoice:', invoice['Invoice_ID'])
```

What have we added? First of all, we have located the invoice amount for every invoice. We also pick up the purchase order ID, and store it in the variable `po`. We then lookup the Purchase Order amount. We do this by looking at the `pos` able, and finding the purchase order amount.

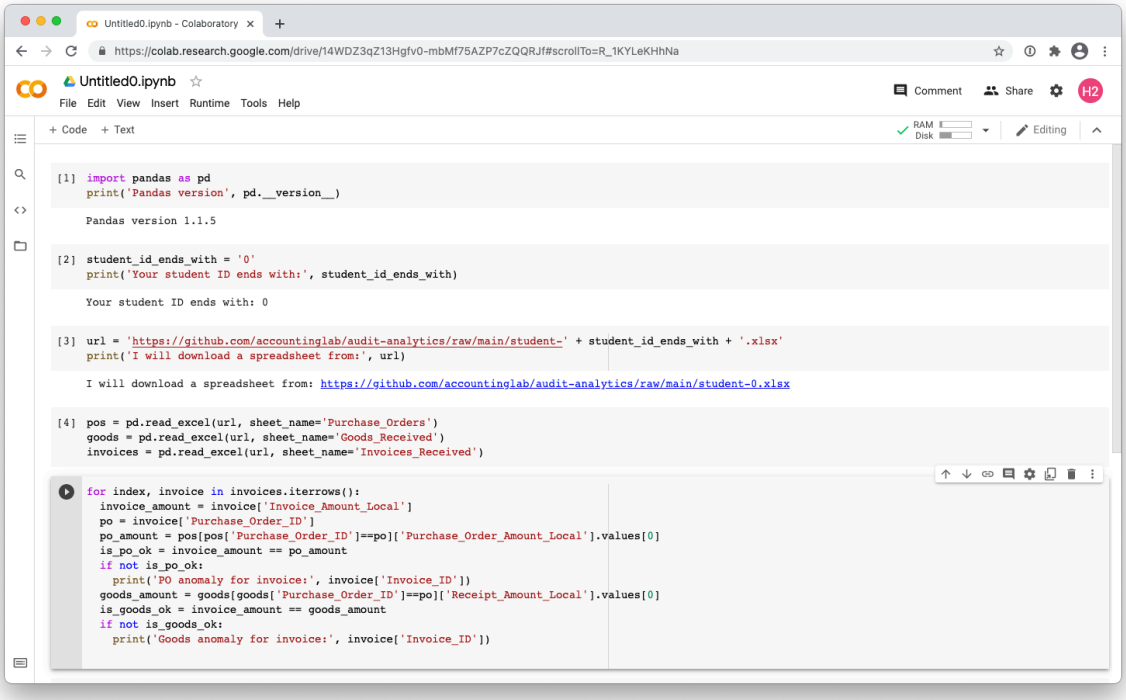
The next step is to compare the purchase order amount with the invoice amount. If we are being invoiced too much, the invoice amount should be higher than PO amount. Here we simply look if the two values are the same. If they are not the same then we print out the invoice number, and report the anomaly. This is where we introduce the if statement.

We then do the same with the goods receipt table. We pick up the amount of goods that we have received. When we are being invoiced an amount higher than the amount of goods receipt, we are again being over-invoiced. The program picks this up and reports it.

Step 2: Run the three-way match program

Your program should look like this (with the correct spreadsheet, here I have assumed your student ID ends with 0).

[1] You can avoid the look-ups (which are relatively slow) by merging the three data frames into one. However, merging data frames is beyond the scope of this exercise.



You can now run the three-way match program by pressing Shift-Enter in the notebook. If you have made a mistake, please check your code very carefully and see if you made a typing error. Forgetting even one double-quote, for example, will cause the code not to run.

Here are some common mistakes:

1. Forgetting a space
2. Adding a space that shouldn't be there
3. Uppercase should be lowercase or vice versa
4. Double quotes instead of single quotes
5. Forgetting a quote
6. Forgetting an indent (tab)
7. The indent is different for different lines, for example, you used spaces on one line and tab on another

Examine what happens if you run the three-way match program on the your dataset. Is there an anomaly? Are all invoices OK?

If all has gone well, you should see the code picking up on the unusual transactions in the data set. Each spreadsheet has at least one invoice that is not correct.

The program above is just 11 lines long and performs a complete three-way-match check in just a few seconds.