Yulia Pavlova

Report. Stora Enso.

The report below is organized as follows: page 1-4 general report with figures for tasks 1-2, pages 5-11 code in python, page 12 – data science questions.

First, the table CASE_TABLE was analyzed. First by looking through the data, I have concluded that the structure of table is [invoice reference number, buyer, reference index, seller, type of goods, quantity, price, channels]. I have decided that it would be worth to look at the channels through which the deals were made and make plots demonstrating how many invoices/deals per channel have been made and how large was the deal made through each channel. I assumed that size of the deal could be adequately presented as quantity x price. The code in Python[1] is attached: there you may see how the data was aggregated w.r.t. channels, total number of invoices per channel, total spending per channel. Below you may see two figures illustrating the findings.

Figure 1. Flow of invoices over each channel

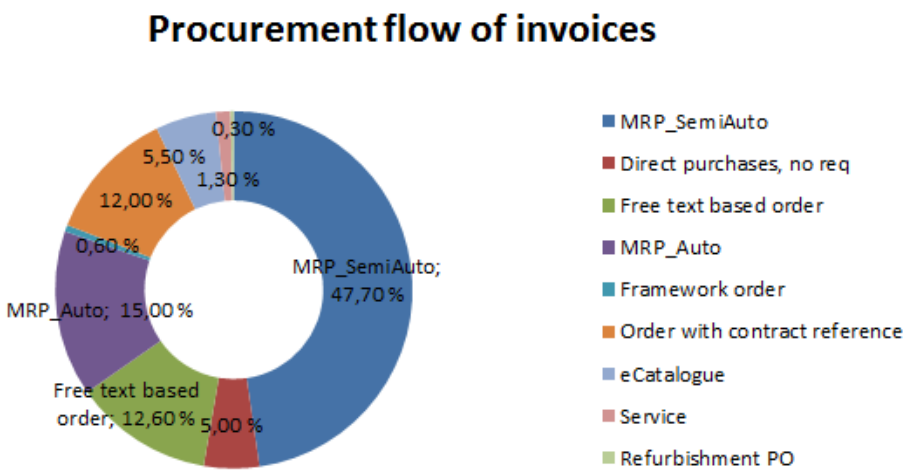


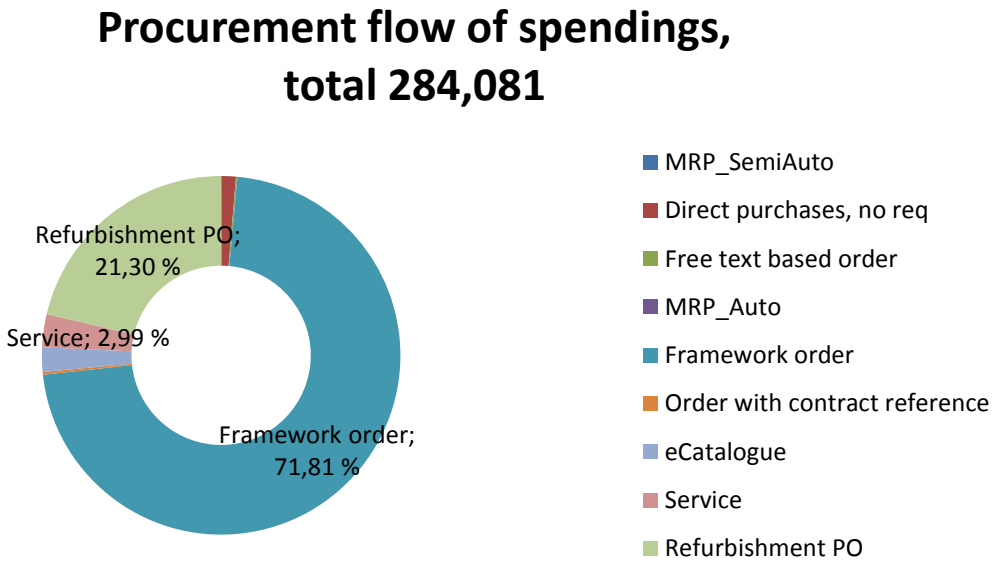
Figure [1] shows percentage of invoices/deals made through each of the channels, and we can see that 'MRP SemiAuto' is the most used channel, followed by 'MRP Auto' and 'Free text based order'. Figure [2] shows that 'Framework order' channel corresponds to the largest amount of spending, even though its relative share of invoices is rather small, only *0.6%*. Spending over the most popular channels are rather small *0.02%, 0.06%, 0.11%*, respectively. These results are produced in code ***task1a.py*** on page 11.

Secondly, I looked at the EVENT_TABLE. Up to my understanding, this table contains [invoice reference number, stages of the invoices, date and time, users]. In the attached python code, the data was aggregated w.r.t. invoices, number of stages in the process, length of the process (as a difference between the final date and the initial data). This data was further sorted according to the length of the process in seconds. There is also an output txt file attached. The first thing that came

[1] I chose Python because I realized that I do not currently have SQL server on my computer. Some procedures would have been smoother with SQL but I hope you do not mind Python this time.

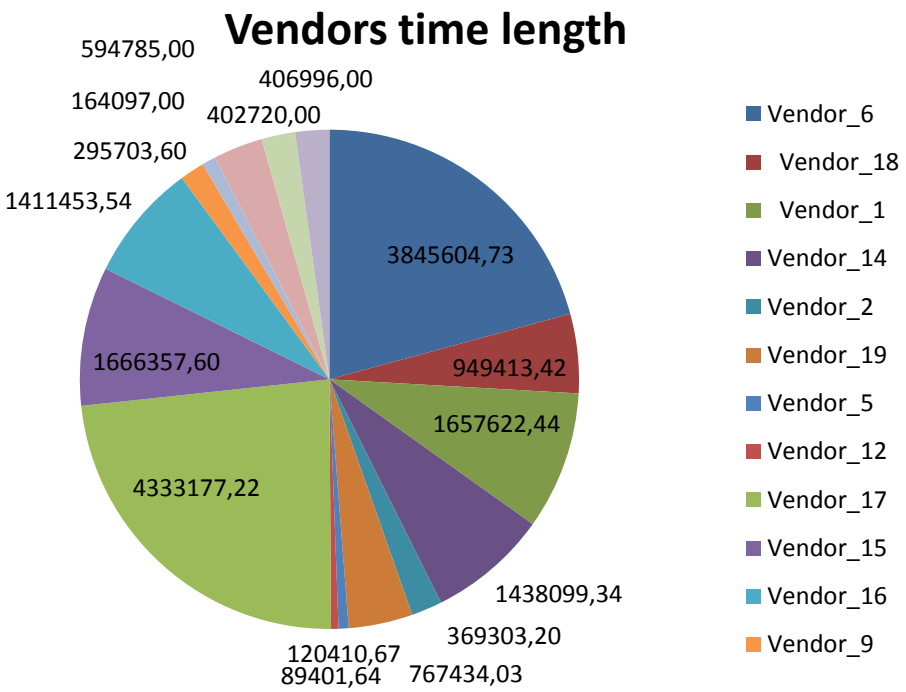
up with Python reading date and time is inconsistent formatting of dates. Secondly, dates of events of the following: invoices # 4502562904-80, # 4502565300-10, # 4502635989-10, # 4502538036-30, # 4502539822-100, # 4502493533-170, came in with peculiar closing date, which is earlier than initializing date. This should be further investigated, whether there is a mistake or has some meaning to it.

Figure 2. Flow of spending over each channel (total value is presumable monetary value, however its exact dimension should be further specified).



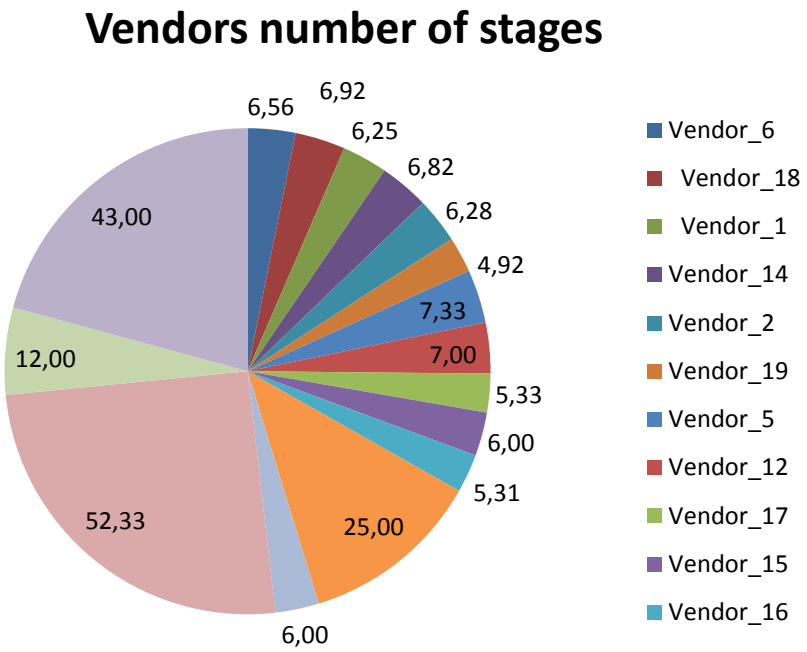
As code in ***task1b.py*** on page 4 shows, array *invoice_flow_sort* contains all invoices that are in order from the shortest process time to longest process time. Then I collected the data from both tables and combined information regarding the vendors (*vendors_vec*) and the channels (*channels_vec*). *Vendors_vec* contain lists of vendors, average number of stages of all invoices for this vendor, and average length (here in seconds) of time period from initiating the order and its final stage. It is illustrated in Fig. [3] and Fig. [4].

Figure 3. Average duration (seconds) of invoice from initiating to finalizing for each vendor



We can see in these figures that process for Vendors 6 and 17 took much more than for other vendors. At the same for Vendor 7 and Vendor 20, average numbers of stages in the process were bigger than for other vendors.

Figure 4. Average number of stages in invoice from initiating to finalizing for each vendor



Cannels_vec contain lists of channels, average number of stages of all invoices for each channel, and average length (here in seconds) of time period from initiating the order and its final stage. It is illustrated in Fig. [5] and Fig. [6].

Figure 5. Average number of stages in the invoice from initiating to finalizing for each channel

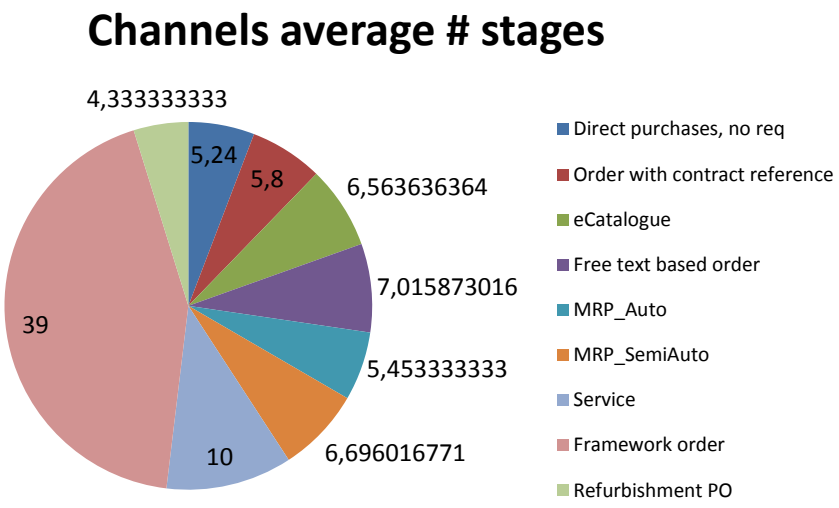
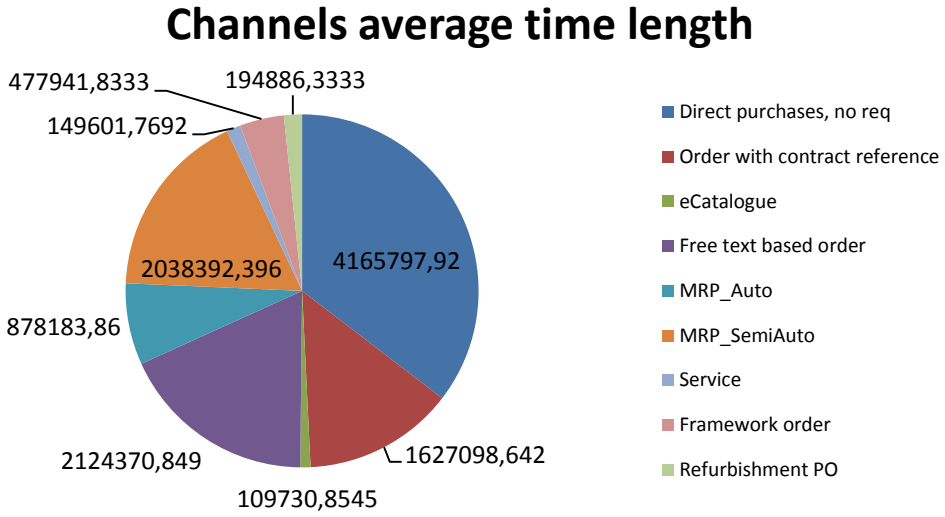


Figure 6. Average duration (seconds) of invoice from initiating to finalizing for each channel



We can see in these figures that orders through ‘Direct purchases, no req’ and ‘Free text based order’ (one of the most popular channels, Fig.[1]) took much more time than for other channels. At the same for ‘Framework order’ on average contains rather high number of stages in the process – it also corresponds to the highest amount of spending (Fig.[2]). I would suggest investigating more into why ‘Direct purchases, no req.’ take so much time and whether it is possible to process it faster. Same concerns ‘Framework order’: it seems to be a rather important channel.

As the next step, I would also suggest investigating whether the is correlation between channels and vendors, and the duration of processing the invoices and spending: this would allow saying, which type of invoices are subjected to too long processing time, perhaps using K-means cluster.

CODES: ***task1b.py***

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 13 17:43:58 2019

@author: Yulia Pavlova
"""
import csv, collections #import library
import numpy as np
import datetime
#import timestring
#import dateutil



def read_csv(file_name):  #creating function to open file for reading and creating 2D array
   array_2D = []
```

```
    with open(file_name, 'r') as csvfile:
        read = csv.reader(csvfile, delimiter=',')
        for row in read:
            array_2D.append(row)
    return array_2D


data = read_csv('EVENT_TABLE.csv') #create a table and write its dimensions
data =np.array(data)
#[nrows,ncols] = np.shape(data)

nin = len(np.unique(data[:,0])) # number of unique number of invoices
counter=collections.Counter(data[:,0]) # keys and values of these invoices
invoices = list(counter.keys())
number_stages = list(counter.values())


# calculating the time differnce between invoice initiation and finilizing
invoice_flow = []
invoice_flow.append(invoices)
invoice_flow.append(number_stages)
timelength = []

for ind in invoices:
    index_in = np.where(data[:,0] == ind)
    t, p = np.shape(np.array(index_in))
    [de, leng] = np.shape(index_in)
    #extracting the initial and the closing times, fixin the date format inconsistency
    eventd =  data[index_in[0][0], 2]
    eventd = eventd.replace('/', '')
    try:
        t1=datetime.datetime.strptime(eventd, '%Y%m%d %H:%M:%S')
    except:
```

```
        try:
            t1=datetime.datetime.strptime(eventd, '%d %b %Y %H:%M:%S')
        except:
            try:
                t1=datetime.datetime.strptime(eventd, '%d.%m.%Y %H:%M:%S')
            except:
                t1=datetime.datetime.strptime(eventd, '%Y.%m.%d %H:%M:%S')
    eventd =  data[index_in[0][leng-1], 2]
    eventd = eventd.replace('/', '')
    try:
        t2=datetime.datetime.strptime(eventd, '%Y%m%d %H:%M:%S')
    except:
        try: t2=datetime.datetime.strptime(eventd, '%d %b %Y %H:%M:%S')
        except:
            try:
                t1=datetime.datetime.strptime(eventd, '%d.%m.%Y %H:%M:%S')
            except:
                t1=datetime.datetime.strptime(eventd, '%Y.%m.%d %H:%M:%S')
    if t2-t1 < datetime.timedelta(0):
        print('Negtaive invoice proc timefor  invoice #',ind)
    td = t2-t1
    timelength.append(int(td.total_seconds()))

#sorting over the length of processing each invoice and writing into a new array
timelength=np.array(timelength)
invoice_flow.append(timelength)
invoice_flow = np.array(invoice_flow)
# the array below contains invoices, number of stages and timelength
invoice_flow_sort = invoice_flow[:,timelength.argsort()]

# open case table again
data2 = read_csv('CASE_TABLE.csv') #create a table and write its dimensions
data2 =np.array(data2)
```

```
vendor_vec = []
channel_vec = []
# extracting vendors and channels which correspond to invoices
for ind in invoice_flow_sort[0, :]:
   index_in = np.where(data2[:, 0] == ind)
   vendor_vec.append(data2[index_in[0][0],3])
   channel_vec.append(data2[index_in[0][0],7])

# appanding vendors and channels to invoice_flow_sort
lst = list(invoice_flow_sort)
lst.append(vendor_vec)
lst.append(channel_vec)
invoice_flow_sort = np.asarray(lst)

#creating output file txt
with open('output.txt', 'w') as f:
   for item in invoice_flow_sort:
      f.write("%s\n" % item)

# creating new arrays vendors with average number of stages and average length of the invoice
vendors_av = []
channels_av = []
ven_st = []
ven_len = []
chan_st = []
chan_len = []

counter=collections.Counter(invoice_flow_sort[3, :]) # keys and values of these invoices
vendors = list(counter.keys())

for ven in vendors:
   index_ven = np.where(invoice_flow_sort[3, :] == ven)
   ven_st.append(np.average(np.array(invoice_flow_sort[1, index_ven]).astype(np.float)))
```

```
    ven_len.append(np.average(np.array(invoice_flow_sort[2, index_ven]).astype(np.float)))

vendors_av.append(vendors)
vendors_av.append(ven_st)
vendors_av.append(ven_len)

counter=collections.Counter(invoice_flow_sort[4, :]) # keys and values of these invoices
channels = list(counter.keys())

for ch in channels:
    index_ch = np.where(invoice_flow_sort[4, :] == ch)
    chan_st.append(np.average(np.array(invoice_flow_sort[1, index_ch]).astype(np.float)))
    chan_len.append(np.average(np.array(invoice_flow_sort[2, index_ch]).astype(np.float)))

channels_av.append(channels)
channels_av.append(chan_st)
channels_av.append(chan_len)


with open('output_vendors.txt', 'w') as f:
    for item in vendors_av:
        f.write("%s\n" % item)


with open('output_channels.txt', 'w') as f:
    for item in channels_av:
        f.write("%s\n" % item)
```

task1a.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 13 17:43:58 2019

@author: Yulia Pavlova
"""
import csv, collections #import library
import numpy as np
def read_csv(file_name):  #creating function to open file for reading and creating 2D array
   array_2D = []
   with open(file_name, 'r') as csvfile:
      read = csv.reader(csvfile, delimiter=',')
      for row in read:
         array_2D.append(row)
   return array_2D



data = read_csv('CASE_TABLE.csv') #create a table and write its dimensions
data =np.array(data)
[nrows,ncols] = np.shape(data)
nch = len(np.unique(data[:,ncols-1])) # number of unique number of channels
counter=collections.Counter(data[:,ncols-1]) # keys and values of these channele
channels = list(counter.keys())
channels_total_invoices = list(counter.values())



#calculating proc. flow: channels, invoices, spendings
proc_flow = []
```

```
proc_flow.append(channels)

proc_flow.append(channels_total_invoices)

spends = []

for ch in channels:

  index_ch = np.where(data[:,ncols-1] == ch)

  t, p = np.shape(np.array(index_ch))

  spends_ch = 0;

  for i in (0,p-1): spends_ch = spends_ch + float(data[index_ch[0][i],5])*float(data[index_ch[0][i],6])

  spends.append(spends_ch)


proc_flow.append(spends)

print(proc_flow)

# add also users and vendors, who buys and sells most and least
```

Output text files

Vendors_ouput.txt

['Vendor_6', 'Vendor_18', 'Vendor_1', 'Vendor_14', 'Vendor_2', 'Vendor_19', 'Vendor_5', 'Vendor_12', 'Vendor_17', 'Vendor_15', 'Vendor_16', 'Vendor_9', 'Vendor_8', 'Vendor_7', 'Vendor_13', 'Vendor_20']

[6.5636363636363635, 6.9210526315789 48, 6.251247920133111, 6.8203125, 6.28, 4.916666666666667, 7.333333333333333, 7.0, 5.333333333333333, 6.0, 5.3076923076923075, 25.0, 6.0, 52.333333333333336, 12.0, 43.0]

[3845604.727272727, 949413.4210526316, 1657622.4376039933, 1438099.3359375, 369303.2, 767434.0277777778, 120410.66666666667, 89401.63636363637, 4333177.222222222, 1666357.6, 1411453.5384615385, 295703.6, 164097.0, 594785.0, 402720.0, 406996.0]

Channels_ouput.txt

['Direct purchases, no req', 'Order with contract reference', 'eCatalogue', 'Free text based order', 'MRP_Auto', 'MRP_SemiAuto', 'Service', 'Framework order', 'Refurbishment PO']

[5.24, 5.8, 6.5636363636363635, 7.015873015873016, 5.453333333333333, 6.696016771488 47, 10.0, 39.0, 4.333333333333333]

[4165797.92, 1627098.6416666666, 109730.85454545454, 2124370.8492063493, 878183.86, 2038392.3962264152, 149601.76923076922, 477941.8333333333, 194886.33333333334]

- Open Modelling, python, etc -file and answers four questions.

How to control random numbers in python?

random.seed()

What would you highlight in below classification metrics?

Recall=0,631

Precision=0,788

F1 Score=0,603

The classifier produces a bit too many false positive results (predicts positive while they are negative). Perhaps, it is good to go over all thresholds (build ROC curve) to see if we get any better F1. Is this is the best F1, then can first try to make model more sophisticated, and then do data enrichment.

What is the main benefit of serializing a model?

This is done to save the previous version of the model, so that it can be for instance trained on the new data, or so that different versions of the model can be compared.