FACULTY OF ENGINEERING

# Exam 1
# Data Analysis

REII 424

by:
W Bisschoff 26217406

**North-West University Potchefstroom Campus**

Lecturer: Prof Alwyn Hoffman

November 12, 2018

# Contents

# List of Figures

# List of Tables

# 1 Question 1

## 1.1 Question 1 a, b

A folder of XML files are read into a Pandas Dataframe. The first level of fields are specified, assnap well as the fields inside the customs responses fields.

The tags are iterated through and placed in temporary dataframes to be later merged into one. Customs responses items are also iterated through and placed into another dataframe to be placed inside the upper dataframe.

This is done with the function in Listing A.1.1.

This function is called for each file and placed in a seperate process to speed up the process in Listing A.1.2.

The output can be seen in Table 1.1.

Table 1.1: File Processing Output

| | |
|---|---|
| Time | 0:08:40.084965 |
| Number of files | 179 |
| Number of entries | 48520 |

The first 5 entries of the dataframe can be seen in Fig. 1.1. There are 25 columns in total, most of which is out of frame.

| | CountryExport | CountryImport | CountryOfOrigin | CpcCode | CustomsOffice | CustomsResponses | CustomsValue | DateControlReceived | DateSubmitted | HSCod |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | TW | ZA | TW | 11 | CTN | StatusCode Messag... | 423918 | 2014-09-01 08:16:00 | 2014-09-01 08:13:19 | 54076 |
| 1 | DE | ZA | DE | 11 | DFM | StatusCode MessageText DateTimeReceived 0... | 38183 | 2014-09-01 09:08:00 | 2014-09-01 09:02:50 | 84749 |
| 2 | RO | ZA | RO | 11 | CTN | StatusCode Messag... | 125970 | 2014-09-01 09:41:00 | 2014-09-01 09:38:21 | 19021 |
| 3 | CN | ZA | CN | 11 | CTN | StatusCode Messag... | 116406 | 2014-09-01 08:18:00 | 2014-09-01 08:14:15 | 52104 |
| 4 | CN | ZA | CN | 11 | CTN | StatusCode Messag... | 120117 | 2014-09-02 08:29:00 | 2014-09-02 08:27:43 | 52104 |

5 rows × 25 columns

Figure 1.1: The Resulting Dataframe

An example of the CustomsResponses dataframe is shown in Fig. 1.2.

| | StatusCode | MessageText | DateTimeReceived |
|---|---|---|---|
| 0 | 7 | Ready - Cash Payment | 2014-09-01 08:17:00 |
| 1 | 13 | Query - Supporting documents required | 2014-09-01 13:57:00 |
| 2 | 34 | Case Closed | 2014-09-01 15:36:00 |
| 3 | 1 | Release | 2014-09-01 15:36:00 |

Figure 1.2: The First Entry's CustomsResponses DataFrame

The length of the dataframe is calculated as 48215.

The number of customs responses are calculated by applying the *len* function to the inner dataframe for every row:

```
1 csv_data['NoCustomsResponses'] = (csv_data['CustomsResponses']).apply(
    len)
```

| | NoCustomsResponses |
|---|---|
| 0 | 4 |
| 1 | 1 |
| 2 | 3 |
| 3 | 4 |
| 4 | 4 |
| 5 | 2 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 5 |

Figure 1.3: Number of Customs Responses for the First Ten Entries

## 1.2 Question 1 c

There were no duplicates found after comparing the length of the dataframe to the length of the dataframe with dropped duplicates.

The dataframe is sorted by the submitted dates. as in Fig. 1.4. Only two columns are shown.

The number of entries did not change.

```
1 xml_data = xml_data.drop_duplicates()
2 xml_data.sort_values(['DateSubmitted'])
3 xml_data[['UniqueNumber', 'DateSubmitted']].head(20)
```

| | UniqueNumber | DateSubmitted |
|---|---|---|
| 0 | 00008125 | 2014-09-01 08:13:19 |
| 1 | 00008130 | 2014-09-01 09:02:50 |
| 2 | 00008131 | 2014-09-01 09:38:21 |
| 3 | 00008126 | 2014-09-01 08:14:15 |
| 4 | 00008127 | 2014-09-02 08:27:43 |
| 5 | 00008134 | 2014-09-05 07:38:58 |
| 6 | 00008138 | 2014-09-08 08:05:07 |
| 7 | 00008142 | 2014-09-08 14:17:15 |
| 8 | 00008143 | 2014-09-08 14:48:38 |
| 9 | 00008135 | 2014-09-08 08:23:17 |
| 10 | 00008144 | 2014-09-09 08:21:45 |
| 11 | 00008145 | 2014-09-10 08:34:00 |
| 12 | 00008147 | 2014-09-11 12:03:06 |
| 13 | 00008152 | 2014-09-15 09:05:07 |
| 14 | 00008154 | 2014-09-15 12:37:27 |
| 15 | 00008158 | 2014-09-16 10:40:39 |
| 16 | 00008159 | 2014-09-16 14:01:56 |
| 17 | 00008155 | 2014-09-17 08:23:39 |
| 18 | 00008164 | 2014-09-18 12:53:02 |
| 19 | 00008149 | 2014-09-18 10:22:44 |

Figure 1.4: First 20 Entries of the Sorted Dataframe

## 1.3 Question 1 d

Categories were created for each transaction from the HS codes. A new function was created that takes a code and searches for the appropriate category within a created dictionary with the first two characters of the code.

This was also done with multi processing and the time was measured to compare the processing speed.

The first twenty results of Listing A.1.3 can be seen in Fig. 1.5.

```
Time = 0:00:1.000749
```

| | UniqueNumber | DateSubmitted | HSCode | Category |
|---|---|---|---|---|
| 0 | 00008125 | 2014-09-01 08:13:19 | 540769 | Textile |
| 1 | 00008130 | 2014-09-01 09:02:50 | 847490 | Machinery |
| 2 | 00008131 | 2014-09-01 09:38:21 | 190219 | Food |
| 3 | 00008126 | 2014-09-01 08:14:15 | 521041 | Textile |
| 4 | 00008127 | 2014-09-02 08:27:43 | 521041 | Textile |
| 5 | 00008134 | 2014-09-05 07:38:58 | 843139 | Machinery |
| 6 | 00008138 | 2014-09-08 08:05:07 | 843880 | Machinery |
| 7 | 00008142 | 2014-09-08 14:17:15 | 841899 | Machinery |
| 8 | 00008143 | 2014-09-08 14:48:38 | 841899 | Machinery |
| 9 | 00008135 | 2014-09-08 08:23:17 | 540769 | Textile |
| 10 | 00008144 | 2014-09-09 08:21:45 | 611090 | Textile |
| 11 | 00008145 | 2014-09-10 08:34:00 | 400510 | Plastic |
| 12 | 00008147 | 2014-09-11 12:03:06 | 940360 | Miscellaneous |
| 13 | 00008152 | 2014-09-15 09:05:07 | 540769 | Textile |
| 14 | 00008154 | 2014-09-15 12:37:27 | 711790 | Stone and Glass |
| 15 | 00008158 | 2014-09-16 10:40:39 | 330300 | Chemical |
| 16 | 00008159 | 2014-09-16 14:01:56 | 841899 | Machinery |
| 17 | 00008155 | 2014-09-17 08:23:39 | 620342 | Textile |
| 18 | 00008164 | 2014-09-18 12:53:02 | 030119 | Animal |
| 19 | 00008149 | 2014-09-18 10:22:44 | 851190 | Machinery |

Figure 1.5: First Twenty Entries of the Dataframe with Added Categories

## 1.4 Question 1 e

The durations were calculated with Listing A.1.4. Entries with invalid durations were removed.

The other fields were generated with Listing A.1.5. Sets of codes for each field was created. From the inner customs response dataframe, a set of codes was generated for every transaction. These codes are then compared to those for the fields to generate the output variables.

The calendar month is also calculated here for future use.

The first twenty entires with the output variables are shown in Fig. 1.6.

| | UniqueNumber | DateSubmitted | Codes | Not Stopped | Stopped | Inspected | Amended |
|---|---|---|---|---|---|---|---|
| 0 | 00008125 | 2014-09-01 08:13:19 | {1, 34, 13, 7} | False | False | False | False |
| 1 | 00008130 | 2014-09-01 09:02:50 | {1} | True | False | False | False |
| 2 | 00008131 | 2014-09-01 09:38:21 | {34, 4, 13} | False | False | False | False |
| 3 | 00008126 | 2014-09-01 08:14:15 | {1, 34, 13, 7} | False | False | False | False |
| 4 | 00008127 | 2014-09-02 08:27:43 | {1, 34, 13, 7} | False | False | False | False |
| 5 | 00008134 | 2014-09-05 07:38:58 | {1} | True | False | False | False |
| 6 | 00008138 | 2014-09-08 08:05:07 | {1} | True | False | False | False |
| 7 | 00008142 | 2014-09-08 14:17:15 | {1} | True | False | False | False |
| 8 | 00008143 | 2014-09-08 14:48:38 | {1} | True | False | False | False |
| 9 | 00008135 | 2014-09-08 08:23:17 | {1, 34, 7, 13, 31} | False | False | False | False |
| 10 | 00008144 | 2014-09-09 08:21:45 | {1, 34, 13} | False | False | False | False |
| 11 | 00008145 | 2014-09-10 08:34:00 | {1, 7} | True | False | False | False |
| 12 | 00008147 | 2014-09-11 12:03:06 | {2, 36, 7, 13, 26, 31} | False | True | True | True |
| 13 | 00008152 | 2014-09-15 09:05:07 | {1, 34, 13, 7} | False | False | False | False |
| 14 | 00008154 | 2014-09-15 12:37:27 | {2, 34, 4, 36, 7, 13} | False | True | True | False |
| 15 | 00008158 | 2014-09-16 10:40:39 | {1, 7} | True | False | False | False |
| 16 | 00008159 | 2014-09-16 14:01:56 | {1} | True | False | False | False |
| 17 | 00008155 | 2014-09-17 08:23:39 | {34, 4, 13, 7} | False | False | False | False |
| 18 | 00008164 | 2014-09-18 12:53:02 | {4} | False | False | False | False |
| 19 | 00008149 | 2014-09-18 10:22:44 | {1, 7} | True | False | False | False |

Figure 1.6: First Twenty Entries of the Dataframe with Added Output Variables

## 1.5 Question 1 f

An array with all the categories is created. Through iterations the dataframe groups by the category and counts the number of transactions as well as the number of categories. The code can be seen in Listing A.1.6.

| CustomsOffice | Transactions |
|---|---|
| BBR | 116 |
| BFN | 11 |
| CTN | 6370 |
| DBN | 10016 |
| DFM | 3791 |
| ELN | 35 |

Figure 1.7: First Six Entries of the Number of Transactions for each Customs Office

| CountryImport | Transactions |
|---|---|
| AE | 9 |
| AO | 64 |
| AR | 3 |
| AT | 2 |
| AU | 4 |
| BE | 4 |

Figure 1.8: First Six Entries of the Number of Transactions for each Country of Import

| | Categories |
|---|---|
| CustomsOffice | 25 |
| CpcCode | 26 |
| PreviousCpc | 16 |
| CountryOfOrigin | 153 |
| CountryExport | 146 |
| CountryImport | 71 |
| TransportCode | 5 |
| SPSource | 174 |
| HSCode | 2675 |

Figure 1.9: The Number of Sub Categories per Category

# 2 Question 2

## 2.1 Question 2 a, b, c

Dictionaries were created to hold the number of transactions, document requests and fractions for each category. The dataframe is manipulated to show grouped info for each calendar month for each sub-category, for each category.

This is then repeated for only transactions that requested additional documents.

These two new dataframes are then divided to create a fraction of document requests.

A rolling mean is applied to the columns in the new dataframe.

The output of Listing A.2.1 is shown in the following figures.

| | AE | AF | AG | AL | AO | AR | AT | AU | BA | BB | ... | UY | VE | VG | VN | VU | YE | ZA | ZM | ZN | ZW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **2014 09** | 21.0 | NaN | NaN | NaN | 3.0 | 5.0 | 7.0 | 33.0 | NaN | NaN | ... | 4.0 | 1.0 | 8.0 | 78.0 | NaN | NaN | 70.0 | 2.0 | NaN | 7.0 |
| **2014 10** | 19.0 | NaN | NaN | NaN | 1.0 | 3.0 | 9.0 | 16.0 | NaN | 3.0 | ... | NaN | NaN | 3.0 | 87.0 | NaN | NaN | 35.0 | NaN | 1.0 | NaN |
| **2014 11** | 15.0 | NaN | NaN | NaN | NaN | 3.0 | 9.0 | 22.0 | NaN | 1.0 | ... | 3.0 | NaN | 5.0 | 22.0 | NaN | NaN | 25.0 | NaN | NaN | NaN |
| **2014 12** | 25.0 | NaN | NaN | NaN | NaN | 4.0 | 9.0 | 8.0 | NaN | 2.0 | ... | NaN | NaN | 3.0 | 3.0 | NaN | NaN | 27.0 | 3.0 | NaN | 2.0 |
| **2015 01** | 44.0 | NaN | NaN | NaN | NaN | NaN | 8.0 | 12.0 | NaN | NaN | ... | NaN | NaN | 3.0 | 7.0 | NaN | 1.0 | 17.0 | 4.0 | NaN | 2.0 |

5 rows × 146 columns

Figure 2.1: First Five Entries of Monthly Entries by Export Country

| | AE | AF | AG | AL | AO | AR | AT | AU | BA | BB | ... | UY | VE | VG | VN | VU | YE | ZA | ZM | ZN | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2014 09 | 0.380952 | NaN | NaN | NaN | NaN | 0.200000 | NaN | 0.151515 | NaN | NaN | ... | NaN | NaN | 0.375000 | 0.089744 | NaN | NaN | 0.142857 | 1.0 | NaN | 0.857 |
| 2014 10 | 0.157895 | NaN | NaN | NaN | NaN | NaN | NaN | 0.062500 | NaN | NaN | ... | NaN | NaN | NaN | 0.022989 | NaN | NaN | 0.028571 | NaN | 1.0 | |
| 2014 11 | 0.133333 | NaN | NaN | NaN | NaN | 0.333333 | NaN | 0.090909 | NaN | 1.0 | ... | NaN | NaN | 0.400000 | NaN | NaN | NaN | NaN | NaN | NaN | |
| 2014 12 | 0.040000 | NaN | NaN | NaN | NaN | NaN | 0.111111 | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | 0.111111 | 1.0 | NaN | 1.000 |
| 2015 01 | 0.181818 | NaN | NaN | NaN | NaN | NaN | 0.125000 | 0.083333 | NaN | NaN | ... | NaN | NaN | 0.333333 | NaN | NaN | NaN | NaN | NaN | NaN | 0.500 |

5 rows × 146 columns

Figure 2.2: First Five Entries of Fractions of Document Requests by Export Country

| | AE | AF | AG | AL | AO | AR | AT | AU | BA | BB | ... | UY | VE | VG | VN | VU | YE | ZA | ZM | ZN | ZW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2014 09 | 0.38 | NaN | NaN | NaN | NaN | 0.20 | NaN | 0.15 | NaN | NaN | ... | NaN | NaN | 0.38 | 0.09 | NaN | NaN | 0.14 | 1.0 | NaN | 0.86 |
| 2014 10 | 0.27 | NaN | NaN | NaN | NaN | 0.20 | NaN | 0.11 | NaN | NaN | ... | NaN | NaN | 0.38 | 0.06 | NaN | NaN | 0.09 | 1.0 | 1.0 | 0.86 |
| 2014 11 | 0.22 | NaN | NaN | NaN | NaN | 0.27 | NaN | 0.10 | NaN | 1.0 | ... | NaN | NaN | 0.39 | 0.06 | NaN | NaN | 0.09 | 1.0 | 1.0 | 0.86 |
| 2014 12 | 0.18 | NaN | NaN | NaN | NaN | 0.27 | 0.11 | 0.10 | NaN | 1.0 | ... | NaN | NaN | 0.39 | 0.06 | NaN | NaN | 0.09 | 1.0 | 1.0 | 0.93 |
| 2015 01 | 0.18 | NaN | NaN | NaN | NaN | 0.27 | 0.12 | 0.10 | NaN | 1.0 | ... | NaN | NaN | 0.37 | 0.06 | NaN | NaN | 0.09 | 1.0 | 1.0 | 0.79 |

5 rows × 146 columns

Figure 2.3: First Five Entries of Accumulated Fractions of Document Requests by Export Country

```
['CustomsOffice',
 'CpcCode',
 'PreviousCpc',
 'CountryOfOrigin',
 'CountryExport',
 'CountryImport',
 'TransportCode',
 'SPSource',
 'HSCode']
```

Figure 2.4: List of All Categories that Declarations were Calculated For

## 2.2 Question 2 d

Accumulated fractions are allocated to each transaction per category through Listing A.2.2. A new dataframe is created with the values. This is also appended to the main dataframe.

Fig. 2.5 shows the dataframe with category values for the first ten transactions.

| | val_CustomsOffice | val_CpcCode | val_PreviousCpc | val_CountryOfOrigin | val_CountryExport | val_CountryImport | val_TransportCode | val_SPSource | val_HSC |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.352113 | 0.225593 | 0.253908 | 0.175439 | 0.178571 | 0.234483 | 0.314815 | 0.472222 | 0.833 |
| 1 | 0.046809 | 0.225593 | 0.253908 | 0.182222 | 0.172131 | 0.234483 | 0.143415 | 0.472222 | N |
| 2 | 0.352113 | 0.225593 | 0.253908 | 1.000000 | 1.000000 | 0.234483 | 0.314815 | 0.472222 | 0.571 |
| 3 | 0.352113 | 0.225593 | 0.253908 | 0.254791 | 0.278066 | 0.234483 | 0.314815 | 0.472222 | 1.000 |
| 4 | 0.352113 | 0.225593 | 0.253908 | 0.254791 | 0.278066 | 0.234483 | 0.314815 | 0.472222 | 1.000 |
| 5 | 0.352113 | 0.225593 | 0.253908 | 0.066372 | 0.079439 | 0.234483 | 0.314815 | 0.472222 | N |
| 6 | 0.046809 | 0.225593 | 0.253908 | 0.066372 | 0.079439 | 0.234483 | 0.143415 | 0.472222 | N |
| 7 | 0.046809 | 0.225593 | 0.253908 | 0.062500 | 0.111111 | 0.234483 | 0.143415 | 0.472222 | N |
| 8 | 0.046809 | 0.225593 | 0.200000 | 0.062500 | 0.111111 | 0.234483 | 0.143415 | 0.472222 | N |
| 9 | 0.352113 | 0.225593 | 0.253908 | 0.254791 | 0.178571 | 0.234483 | 0.314815 | 0.472222 | 0.833 |

Figure 2.5: First Ten Entries of Transaction Category Values

## 2.3 Question 2 e

The following code is used to calculate the Pearson correlation coefficients. The dataframe's built in *corr()* function is used.

```
1 corrcoeff = data_with_val[['Requested'] + val_names].fillna(0).corr().loc['
    Requested']
2 corrcoeff = pd.DataFrame(corrcoeff).reset_index().iloc[1:len(corrcoeff)]
```

Fig. 3.3 shows the coefficients for each category. The bar plot is shown in Fig. 2.7

| | index | Requested |
|---|---|---|
| 1 | val_CustomsOffice | 0.187068 |
| 2 | val_CpcCode | 0.155497 |
| 3 | val_PreviousCpc | 0.150715 |
| 4 | val_CountryOfOrigin | 0.247821 |
| 5 | val_CountryExport | 0.262393 |
| 6 | val_CountryImport | 0.084186 |
| 7 | val_TransportCode | 0.217531 |
| 8 | val_SPSource | 0.480162 |
| 9 | val_HSCode | 0.421181 |

Figure 2.6: Pearson Correlation Coefficients for each Category

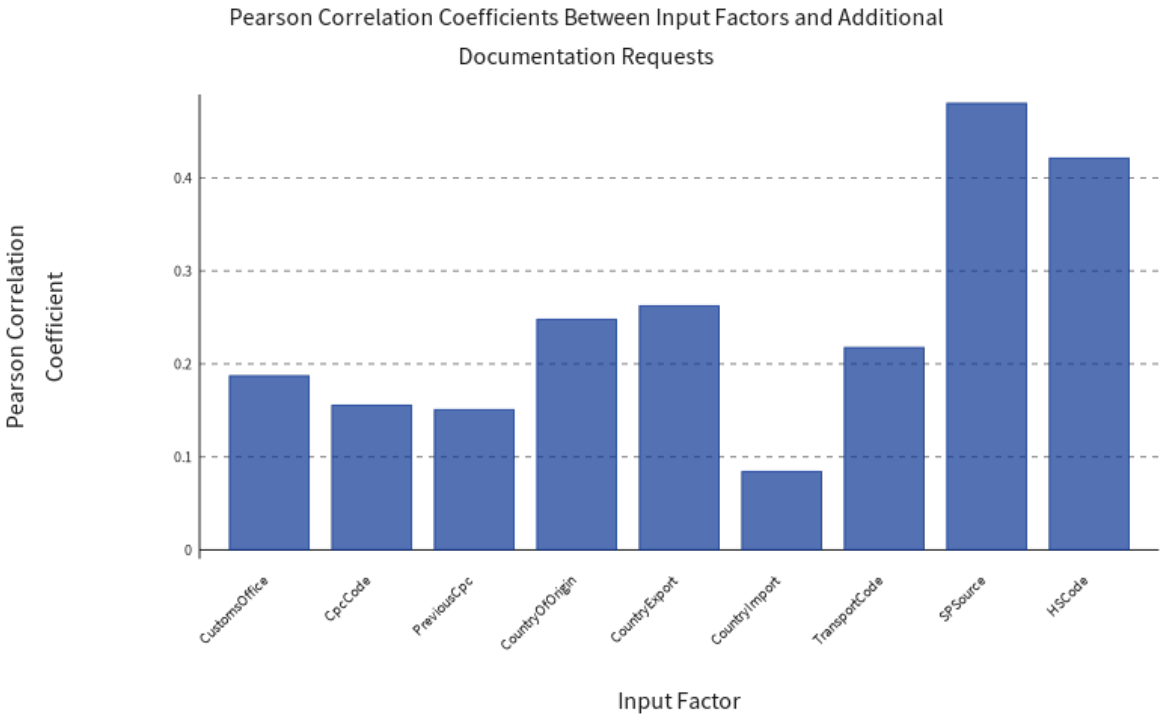


Figure 2.7: Bar Graph of Pearson Correlation Coefficients

# 3 Question 3

## 3.1 Question 3 a

Sklearn is used to split data between training sets and testing sets.

```
X_train, X_test , y_train , y_test = train_test_split(data.loc[:, "
    val_CustomsOffice":"val_HSCode"],data['Requested'],test_size=0.3)
trainindexes = list(X_train.sort_index().index.values)
traindata = data.iloc[trainindexes]
```

## 3.2 Question 3 b

To ensure a 50:50 split between transactions that request additional documents and those that do not, data needs to be duplicated. Transactions that request documents are added until the number of entries is equal or larger than the non-requests.

```
while data[data['Requested'] == 1]['Requested'].count() < norequest:
    data = data.append(request)
data = data.iloc[0:norequest*2]
data = data.reset_index(drop=True)
```

## 3.3 Question 3 c

The correlation coefficients are sorted and the largest three categories are used to plot the number of document requests against the request fraction values. The results of the following snippet is shown in the following figures.

From the graphs it seems that Export Country has the best ability to identify additional document request cases, followed by SP Source and then HS Code.

```
corrcoeff = corrcoeff.sort_values('Requested', ascending=False)
highestcorr = list(corrcoeff['index'].iloc[0:3].str[4:100])
display(highestcorr)

for cat in highestcorr:
    temp = cat_count_month_frac[cat].reset_index().melt(id_vars=['index']).
        set_index(['index', 'variable']).sort_values(['index', 'variable']).
        dropna()
    temp2 = xml_data[['Requested', 'Month', cat]].groupby(['Month', cat]).
        count()
    temp3 = pd.concat([temp, temp2], axis=1)
    temp3.plot.scatter(x='value', y = 'Requested', title=cat)
```

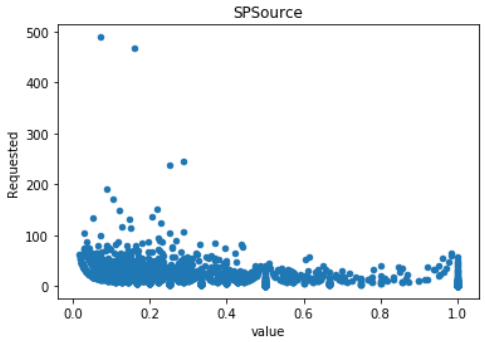


Figure 3.1: Number of Document Requests vs Fraction of Document Requests of SP Source

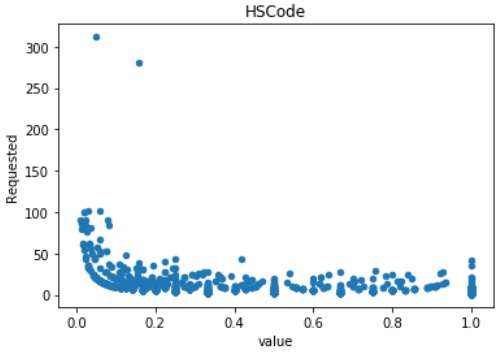


Figure 3.2: Number of Document Requests vs Fraction of Document Requests of HS Code

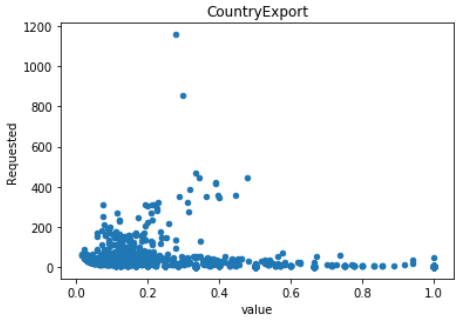


Figure 3.3: Number of Document Requests vs Fraction of Document Requests of Export Country

# 4 Question 4

## 4.1 Question 4 a

Sklearn is used to create a Linear Regression model. The regression coefficients are shown in Table 4.1.

```
X_train = X_train.fillna(0)
X_test = X_test.fillna(0)
reg = LinearRegression().fit(X_train, y_train)
reg.score(X_train, y_train)
reg.coef_
```

Table 4.1: Linear Regression Coefficients

| Category | Coefficient |
|---|---|
| CustomsOffice | 0.08299145 |
| CpcCode | 0.08299145 |
| PreviousCpc | 0.55039948 |
| CountryOfOrigin | -0.03845656 |
| CountryExport | 0.13658853 |
| CountryImport | 0.14750978 |
| TransportCode | 0.28401459 |
| SPSource | 0.50424546 |
| HSCode | 0.52598281 |
| Number of entries | 0.39340001 |

## 4.2 Question 4 b

Listing A.3.1 iterates through multiple threshold values. Predictions are made with linear regression and converted into binary values with the threshold. A dataframe is then generated that contains the fraction of correct predictions for all observations, observations where the target is 1 and observations where the target is 0, for each threshold value.

Fig. 4.1 shows the first five values of this dataframe.

| | All Observations | Target One | Target Zero | Threshold |
|---|---|---|---|---|
| **0** | 0.507808 | 0.021213 | 0.996357 | 0.00 |
| **1** | 0.508274 | 0.022516 | 0.995983 | 0.01 |
| **2** | 0.510325 | 0.026610 | 0.995983 | 0.02 |
| **3** | 0.515079 | 0.036565 | 0.995516 | 0.03 |
| **4** | 0.520347 | 0.047079 | 0.995516 | 0.04 |

Figure 4.1: Prediction Accuracy for Different Targets per Threshold Value

The plot of the above dataframe can be seen in Fig. 4.2.

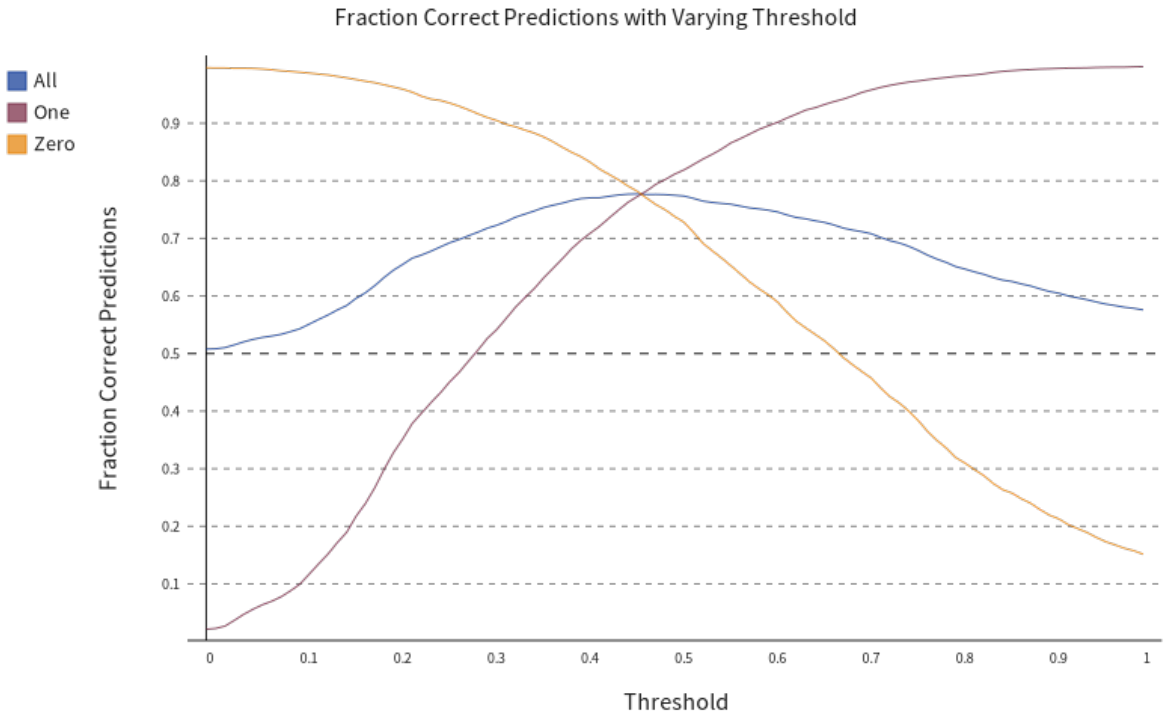


Figure 4.2: Plot of Prediction Accuracy for Different Targets per Threshold Value

## 4.3 Question 4 c

```
df_frac.loc[(df_frac['Target One']-0.5).abs().argsort()[:1]]
df_frac.loc[(df_frac['Target One']-0.8).abs().argsort()[:1]]
df_frac.loc[(df_frac['Target One']-0.9).abs().argsort()[:1]]
df_frac.loc[(df_frac['Target One']-0.95).abs().argsort()[:1]]
```

Table 4.2: Threshold Values to Predict Specific Target Percentages

| Threshold | All Observations | Target Value 1 | Target Value 0 |
|---|---|---|---|
| 0.29 | 0.710856 | 0.506048 | 0.916488 |
| 0.49 | 0.776069 | 0.802382 | 0.749650 |
| 0.61 | 0.745910 | 0.901842 | 0.589351 |
| 0.69 | 0.714446 | 0.947525 | 0.480430 |

# Appendices

## Appendix A Code

### A.1 Question 1

#### A.1.1 Read XML Function

```
def readxml(filename):
    file = os.path.join(path, filename)
    doc = et.parse(file)
    inner_data = pd.DataFrame()
    fields =  ['MRN', 'LRN', 'ISVOC', 'UniqueNumber', 'CustomsOffice', '
       CpcCode', 'PreviousCpc', 'TransportCode', 'CountryExport','
       CountryImport', 'TotalEntryLines', 'TotalWeight', 'TotalPackages', '
       ServiceProvider', 'SPSource', 'HSCode', 'HasPermit','CustomsValue',
       'TradeAgreement', 'StatsQuantity', 'StatsUOM', 'CountryOfOrigin', '
       DateSubmitted', 'DateControlReceived', 'CustomsResponses']
    cusfields = ['StatusCode', 'MessageText', 'DateTimeReceived']
    for elem in doc.findall('.//BOEHeader'):
        inner_dict = pd.Series()
        df = pd.DataFrame(columns=fields)
        i = 0
        for item in elem.findall('.//*'):
            if item.tag in fields:
                inner_dict[item.tag] = item.text
                if item.tag == 'DateSubmitted':
                    if item.text == '0000-00-00-00.00.00':
                        inner_dict[item.tag] = None
                    else:
                        inner_dict[item.tag] = dt.strptime(item.text, '%Y-%
                           m-%d-%H.%M.%S')
                if item.tag == 'DateControlReceived':
                    if item.text == '0000-00-00-00.00.00':
                        inner_dict[item.tag] = None
                    else:
                        inner_dict[item.tag] = dt.strptime(item.text, '%Y-%
                           m-%d-%H.%M.%S')
                if item.tag == 'CustomsResponses':
                    cus_data = pd.DataFrame(columns = cusfields)
                    for cus in item.findall('.//CustomsResponse'):
                        cus_dict = pd.Series()
                        for res in cus.findall('.//*'):
                            cus_dict[res.tag] = res.text

```

```
31                              if res.tag == 'StatusCode':
32                                  cus_dict[res.tag] = int(res.text)
33                              if res.tag == 'DateTimeReceived':
34                                  if item.text == '0000-00-00-00.00.00':
35                                      cus_dict[res.tag] = None
36                                  else:
37                                      cus_dict[res.tag] = dt.strptime(res.
                                            text, '%Y-%m-%d-%H.%M.%S')
38                          cus_data = cus_data.append(cus_dict, ignore_index=
                                True)
39                      inner_dict['CustomsResponses'] = cus_data.sort_values([
                            'DateTimeReceived'])
40
41          inner_data = inner_data.append(inner_dict, ignore_index=True)
42      return inner_data
```

### A.1.2 Read All Files

```
path = "/media/werner/Google Drive/Google Drive/University/Year 4/REII424 - Data Analysis/Exam 1/Small/"
xml_data = pd.DataFrame()

if __name__ == '__main__':
    filenames = os.listdir(path)
    p = mp.Pool(processes = 8)
    start = time.time()
    splitfiles = np.array_split(filenames,8)
    async_result = p.map(readxml, filenames)
    xml_data = xml_data.append(async_result, ignore_index=True, sort=True)
    p.close()
    p.join()
    end = time.time()
    m, s = divmod(end-start, 60)
    h, m = divmod(m, 60)
    print("Time = %d:%02d:%f"%(h,m,s))

print("Number of files = %d"%len(filenames))
print("Number of entries = %d"%len(xml_data))
```

### A.1.3 Categorize

```
def category(code):
    var = int(code[0:2]) if not code == None else code
    values = [-1, 5, 15, 24, 27, 38, 40, 43, 49, 63, 67, 71, 83, 85, 89,
       97, 99]
    cat = ['Animal', 'Vegetable', 'Food',  'Mineral', 'Chemical', 'Plastic'
       ,
           'Hide', 'Wood', 'Textile', 'Footwear', 'Stone and Glass', 'Metal
              ',
          'Machinery', 'Transport', 'Miscellaneous', 'Service']
    dict = {}
    for i in range(len(cat)):
        dict.update(dict.fromkeys(list(range(values[i]+1, values[i+1]+1)),
           cat[i]))

    return dict.get(var,"Other")

HS = xml_data['HSCode']
HS = HS.str.strip()
HS[HS == ""] = None
cat = []

start= time.time()
xml_data['Category'] = HS.apply(category)
end= time.time()
m, s = divmod(end-start, 60)
h, m = divmod(m, 60)
print("Time = %d:%02d:%f"%(h,m,s))
xml_data[['UniqueNumber', 'DateSubmitted', 'HSCode', 'Category']].head(20)
```

### A.1.4 Calculate Durations

```
duration = pd.DataFrame()
firstdate = pd.DataFrame()
lastdate = pd.DataFrame()
for index, row in xml_data.iterrows():
    temp = row['CustomsResponses']['DateTimeReceived'].max()
    if temp == np.nan:
        temp = row['DateControlReceived']
    lastdate = lastdate.append([temp], ignore_index=True)
    firstdate = firstdate.append([row['DateSubmitted']], ignore_index=True)

xml_data['FirstDate'] = firstdate
xml_data['LastDate'] = lastdate
for index, row in xml_data.iterrows():
    duration = duration.append([row['LastDate']-row['FirstDate']],
       ignore_index=True)

duration
xml_data['Duration'] = duration
xml_data = xml_data[~xml_data['Duration'].isnull()]
```

### A.1.5 Generate Output Variables

```
statuscodes = pd.DataFrame()
not_stopped = pd.DataFrame()
stopped = pd.DataFrame()
inspected = pd.DataFrame()
amended = pd.DataFrame()
months = pd.DataFrame()
requested = pd.DataFrame()
notstopcodes = set([2, 4, 13, 31, 36])
stopcodes = set([2])
inspectcodes = set([36])
amendcodes = set([6, 26])
requestedcodes = set([13, 31])


def processcodes(row):
    codes = set(row['CustomsResponses']['StatusCode'])
    row['Codes'] = codes
    row['Stopped'] = not not codes & stopcodes
    row['Not Stopped'] = not codes & notstopcodes
    row['Inspected'] = not not codes & inspectcodes
    row['Amended'] = not not codes & amendcodes
    row['Requested'] = not not codes & requestedcodes
    row['Month'] = date.strftime(row['FirstDate'],"%Y %m")
    return(row)

def process(df):
    res = df.apply(processcodes, axis=1)
    return res

start= time.time()

if __name__ == '__main__':
    p = mp.Pool(processes=8)
    split_dfs = np.array_split(xml_data,8)
    pool_results = p.map(process, split_dfs)
    p.close()
    p.join()

    # merging parts processed by different processes
    xml_data = pd.concat(pool_results, axis=0)


end= time.time()
m, s = divmod(end-start, 60)
h, m = divmod(m, 60)
print("Time = %d:%02d:%f"%(h,m,s))

```

```
48 xml_data = xml_data[~xml_data['Codes'].isnull()]
49 xml_data[['UniqueNumber', 'DateSubmitted', 'Codes', 'Not Stopped', 'Stopped
       ', 'Inspected', 'Amended']].head(20)
```

### A.1.6 Count Transactions per Category

```
categories = ['CustomsOffice', 'CpcCode', 'PreviousCpc', 'CountryOfOrigin',
              'CountryExport', 'CountryImport', 'TransportCode', 'SPSource'
                 , 'HSCode']
cat_count = {}
cat_amount = pd.DataFrame(columns=['Categories'])
for category in categories:
    df = pd.DataFrame()
    df['Transactions'] = xml_data.groupby([category])[category].count()
    cat_amount.loc[category] = xml_data.groupby([category])[category].count
       ().count()
    cat_count["{0}".format(category)] = df
    display(df.head(6))
```

## A.2 Question 2

### A.2.1 Fractions of Requests for Additional Documents

```
cat_count_month = {}
cat_count_month_req = {}
cat_count_month_frac = {}
for category in categories:
    monthly = xml_data.groupby([category, 'Month']).size().reset_index().
        pivot(values=0, index='Month', columns=category)
    monthly_req = xml_data[xml_data['Requested'] == True].groupby([category
        , 'Month']).size().reset_index().pivot(values=0, index='Month',
        columns=category)
    monthly_frac = (monthly_req/monthly)
    cat_count_month["{0}".format(category)] = monthly
    cat_count_month_req["{0}".format(category)] = monthly_req
    cat_count_month_frac["{0}".format(category)] = monthly_frac

cat_count_month = pd.concat(cat_count_month.values(), axis=1, keys=
    cat_count_month.keys(), sort=True)
cat_count_month_req = pd.concat(cat_count_month_req.values(), axis=1, keys=
    cat_count_month_req.keys(), sort=True)
cat_count_month_frac = pd.concat(cat_count_month_frac.values(), axis=1,
    keys=cat_count_month_frac.keys(), sort=True)
cat_count_month_frac_acc = cat_count_month_frac.rolling(100, min_periods=1)
    .mean()
display(cat_count_month['CountryExport'].head())
display(cat_count_month_req['CountryExport'].head())
display(cat_count_month_frac['CountryExport'].head())
display(cat_count_month_frac_acc['CountryExport'].head().round(2))
display(list(cat_count_month.columns.levels[0]))
```

### A.2.2 Generate Output Variables From Accumulated Document Request Fractions

```
def getval(row):
    new = pd.Series()
    for category in categories:
        if not row[category] == None:
            new["val_{0}".format(category)] = (cat_count_month_frac_acc[
                category][row[category]][date.strftime(row['DateSubmitted'],
                "%Y %m")])
        else:
            new["val_{0}".format(category)] = 0
    return(new)


def process(df):
    res = df.apply(getval, axis=1)
    return res


val_names = list("val_{0}".format(category) for category in categories)
start= time.time()
if __name__ == '__main__':
    p = mp.Pool(processes=8)
    split_dfs = np.array_split(xml_data,8)
    pool_results = p.map(process, split_dfs)
    p.close()
    p.join()

    cat_val = pd.concat(pool_results, axis=0)

data_with_val = pd.concat([xml_data, cat_val], axis=1)
end= time.time()
m, s = divmod(end-start, 60)
h, m = divmod(m, 60)
print("Time = %d:%02d:%f"%(h,m,s))
```

## A.3 Question 4

### A.3.1 Predict Values with Varying Thresholds

```
def getval(row):
    new = pd.Series()
    for category in categories:
        if not row[category] == None:
            new["val_{0}".format(category)] = (cat_count_month_frac_acc[
               category][row[category]][date.strftime(row['DateSubmitted'],
               "%Y %m")])
        else:
            new["val_{0}".format(category)] = 0
    return(new)


def process(df):
    res = df.apply(getval, axis=1)
    return res


val_names = list("val_{0}".format(category) for category in categories)
start= time.time()
if __name__ == '__main__':
    p = mp.Pool(processes=8)
    split_dfs = np.array_split(xml_data,8)
    pool_results = p.map(process, split_dfs)
    p.close()
    p.join()

    cat_val = pd.concat(pool_results, axis=0)

data_with_val = pd.concat([xml_data, cat_val], axis=1)
end= time.time()
m, s = divmod(end-start, 60)
h, m = divmod(m, 60)
print("Time = %d:%02d:%f"%(h,m,s))
```