Personal Information

Name: **Roderick Majoor**

StudentID: **12852724**

Email: **roderick.majoor@student.uva.nl**

Submitted on: **22.03.2024**

Data Context

The dataset used in this study consists of pages from the collection of ledgers from the Bank of Amsterdam. The full collection can be found on the website of The Amsterdam City Archives and consists of approximately half a million pages. In the series of ledgers, the current accounts of the customers were maintained. Each customer had one or more pages, with smaller traders having a portion of a page. An alphabetical list of the traders and the page of their current account can be found in the index. The ledgers kept track of the amounts that were debited and credited from and to a customer. The ledger pages consist of several columns containing information such as the date, account holder name, account number and amount debited/credited.

Data Description

The total dataset consists of approximately half a million pages from The ledgers of the Bank of Amsterdam (Amsterdamse Wisselbank) from between 1650-1800. The pages follow a tabular structure where on each page, the debit/credit account of a client is kept track of. The columns of both the debit and credit table show a date, account name, guldens, stuivers, and penningen (currency at the time). The ground truth annotated set consist of 68 pages

Intro

Currently, a tool called Loghi is used to perform handwritten text recognition on these pages. This tool has its flaws though, and in this thesis we are trying to determine what the major errors are and how we could possibly solve them.

From the analysis show below we find the following:

1. False positives detected (typically due to marks / stripes on the image that are recognized as text)
2. Numbers are not detected as numbers but rather as text / other characters
3. Words/numbers are merged or seperated (i.e. 16 becomes 1, 6 or 2, 8 becomes 28 etc.)
4. Numbers not correctly recognized (i.e. 16 becomes 18 etc.)
5. Output not in the correct layout

Some of the code used is quite long, so instead of putting everything in this notebook, we will just import it from their scripts. The scripts can be found on the GitHub: https://github.com/roderickmajoor/Thesis/tree/main

In [1]:
```
# Imports
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import xml.etree.ElementTree as ET
import jiwer
import cv2

from compare_GT_data_textlines import main
```

Data Loading

In [2]:
```
# Path to the folder containing the images
data_dir = "/home/roderickmajoor/Desktop/Master/Thesis/GT_data"

# List to store paths of all image files
image_paths = []

# Iterate through the folder structure
for root, dirs, files in os.walk(data_dir):
    for file in files:
        # Check if the file is an image
        if file.endswith(".jpg") or file.endswith(".png"):
            # Construct the full path to the image file
            image_path = os.path.join(root, file)
            # Append the image path to the list
            image_paths.append(image_path)
```

Analysis 1: Visualizing Data

Here we show a couple of the images in our dataset. This already shows us a couple of things. First of all, the sizes of the images are not always the same. The first 3 images are larger than image 4 and 5. The first 3 images also seem to have a more 'neat' layout where text does not flow over into other text. The layout on image 4 and 5 seems a bit more

chaotic. Other noticable things are stains on some of the pages and also stripes/marks that go through some of the text which may cause problems when analysing it.

In [3]:
```
# Set the number of images to visualize
num_images_to_visualize = 5

# Visualize a sample of the images
for i in range(num_images_to_visualize):
    # Read the image using matplotlib
    image = mpimg.imread(image_paths[i])
    # Plot the image
    plt.figure(figsize=(8, 8))
    plt.imshow(image)
    plt.axis('off')  # Turn off axis
    plt.title(f"Image {i+1}")  # Set title
    plt.show()
```

Image 1

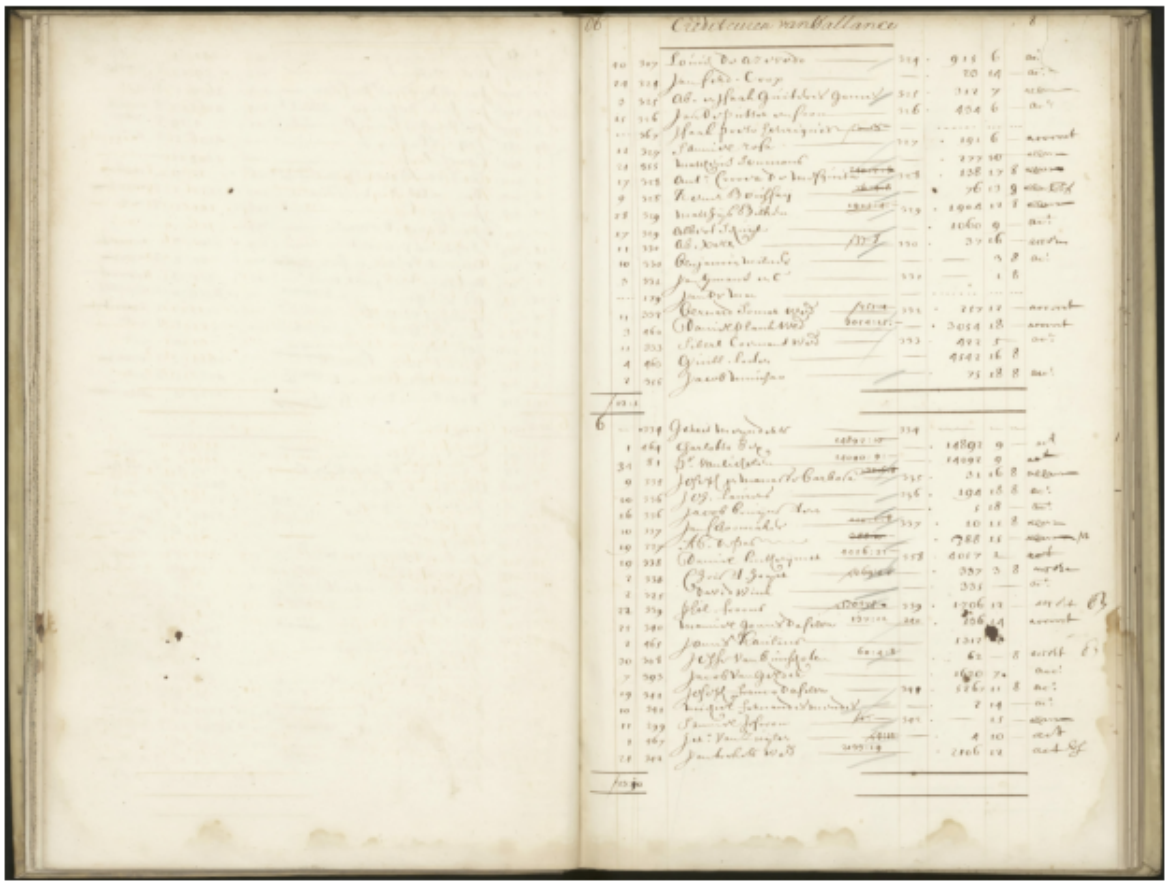
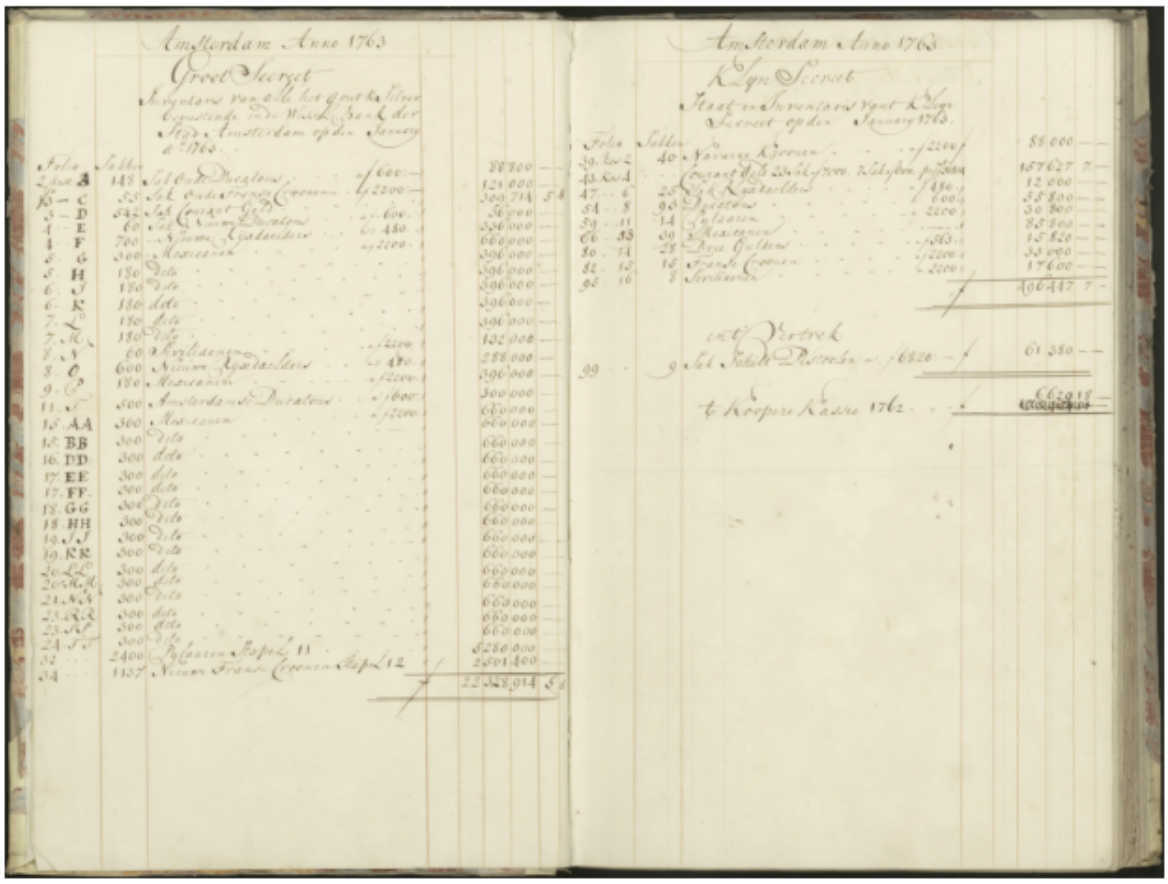


Image 2



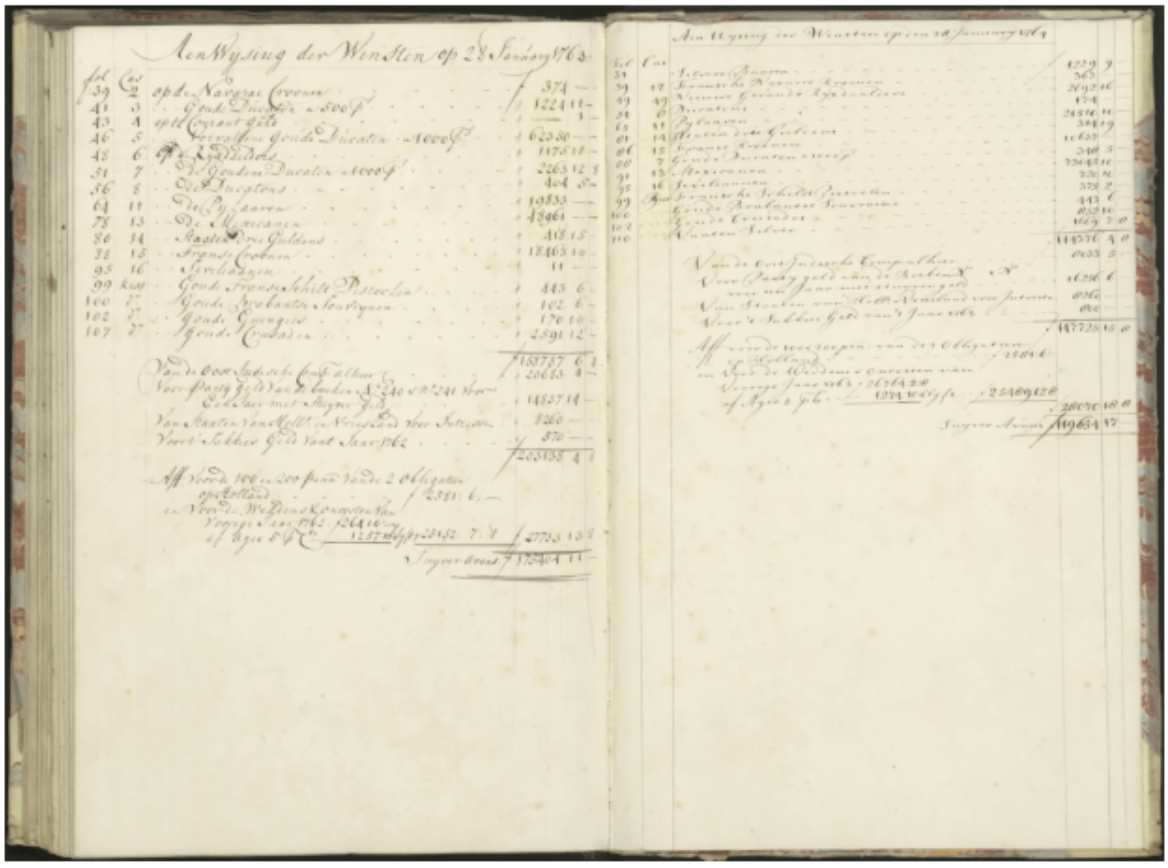
Amsterdam Anno 1763

Groot Secret

Amsterdam Anno 1763

Klyn Secret

Image 3



Aen Wysing der Wensten op 28 Januarij 1763

Aen Wysing der Wensten op 28 January 1763

Image 4

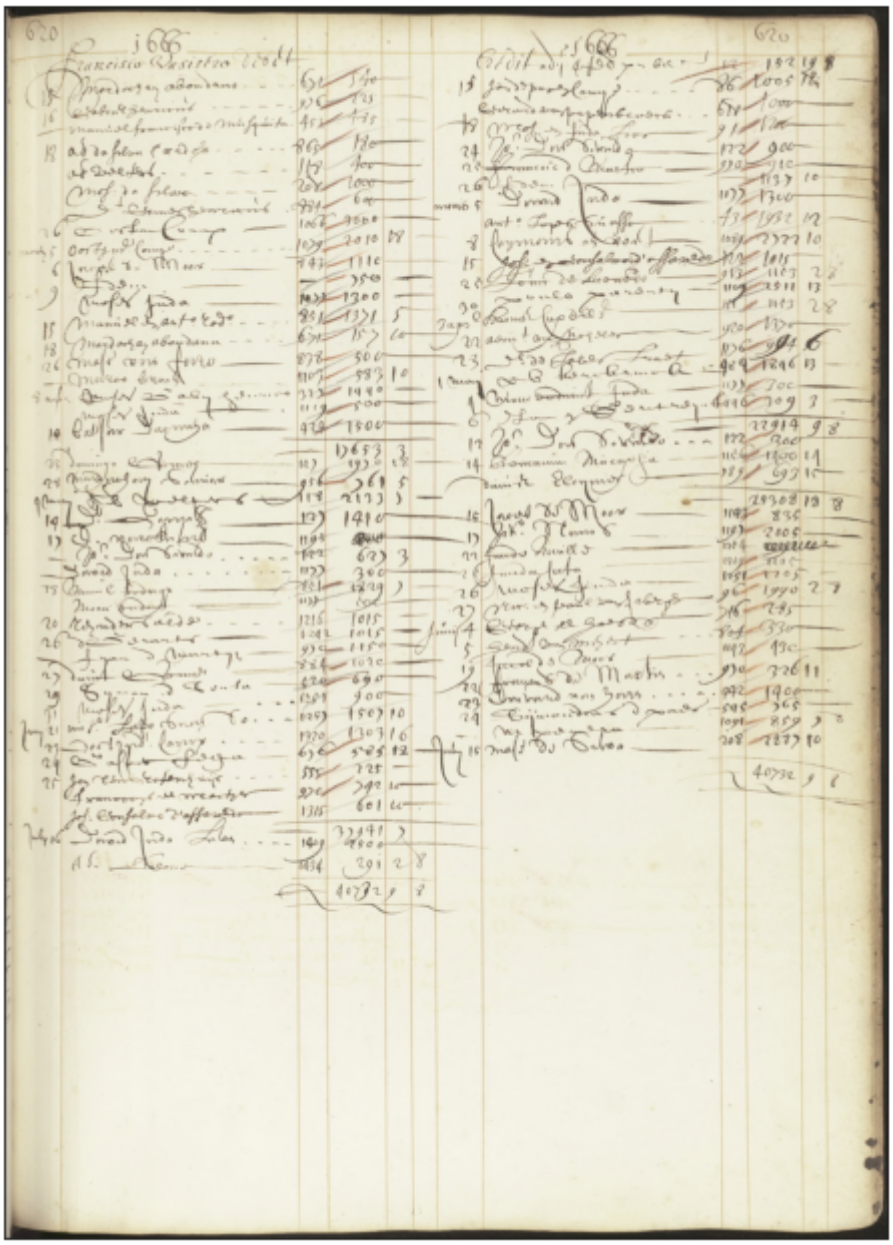
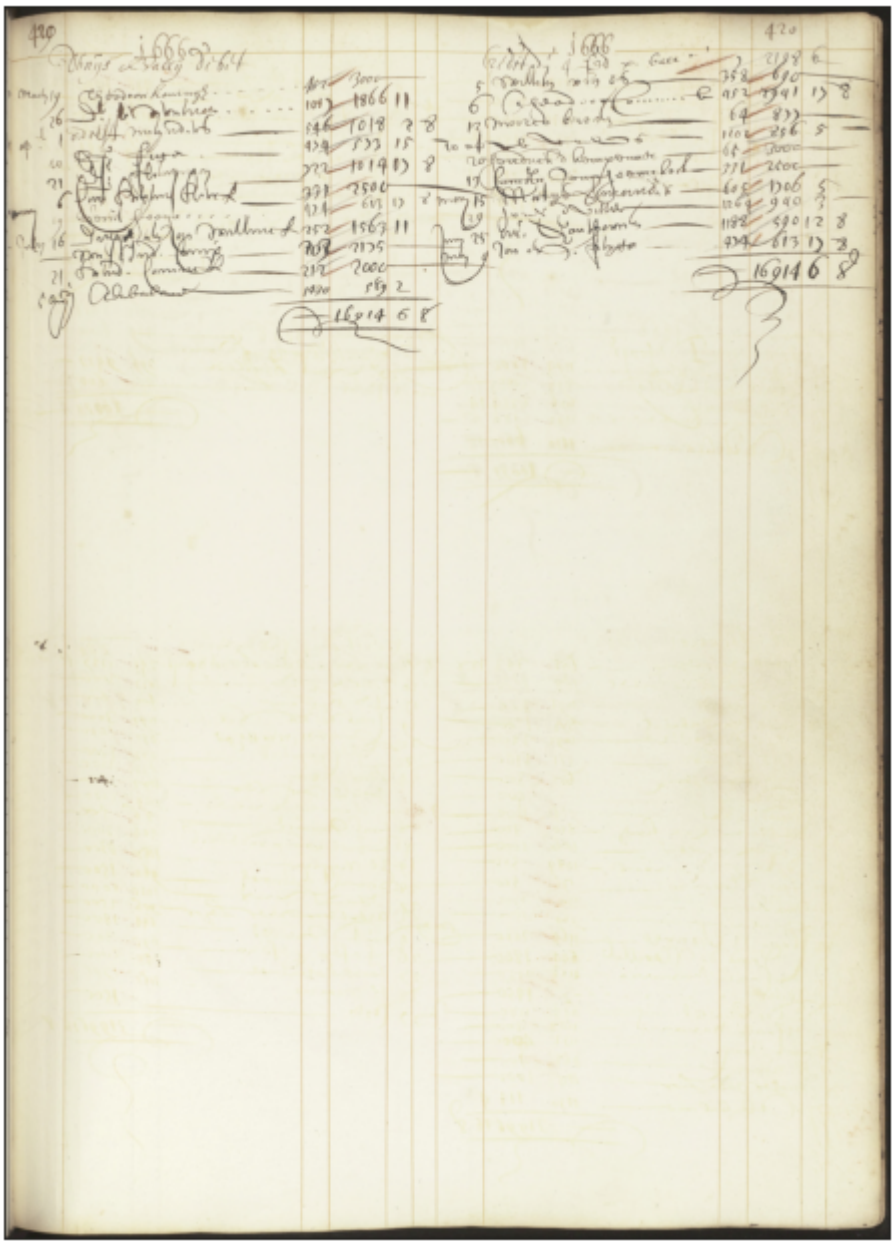


Image 5

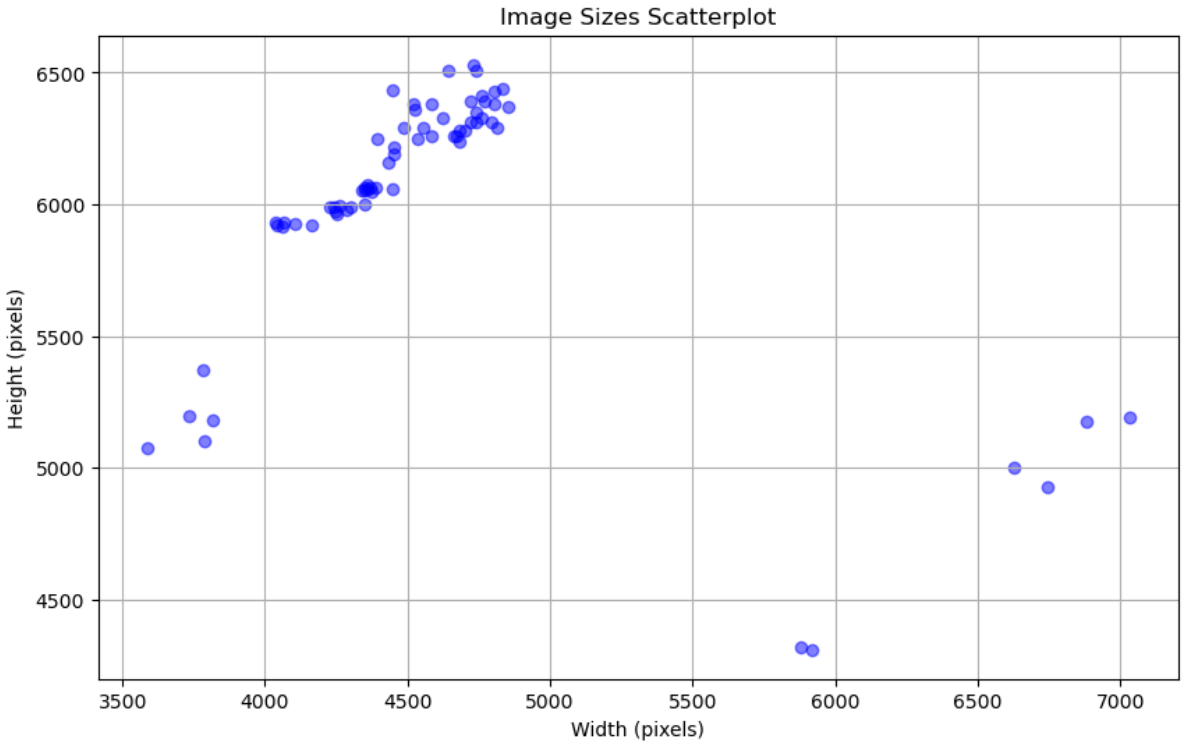


We show the length of the annotated image dataset. We show that all our images are in color (consist of 3 color channels) and we show a scatterplot of the different image sizes.

In [4]:
```
print(len(image_paths))

widths = []
heights = []
channels_images = set()

for i in range(len(image_paths)):
    img = cv2.imread(image_paths[i])

    (height, width, channels) = img.shape
    channels_images.add(channels)

    widths.append(width)
```

```
    heights.append(height)

print(channels_images)

# Create a scatterplot
plt.figure(figsize=(10, 6))
plt.scatter(widths, heights, color='blue', alpha=0.5)
plt.title('Image Sizes Scatterplot')
plt.xlabel('Width (pixels)')
plt.ylabel('Height (pixels)')
plt.grid(True)
plt.show()
```

```
68
{3}
```



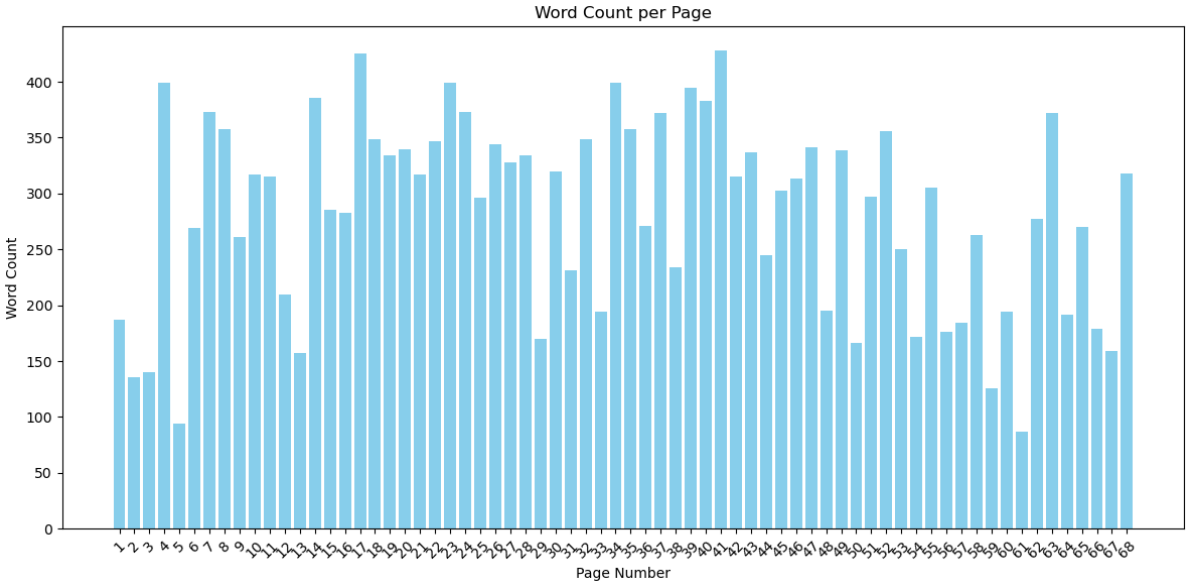
Analysis 2:

We need to understand our data. Since we are working with images, and more specifically handwritten text recognition, the important features of the images are stored in an XML file. The XML files contain the information about the images such as at what coordinates text regions, text lines, words etc. occur. This is important because the XML file tells us at what location on an image a certain feature may be.

First, let's show the amount of annotated words that we have in our ground truth dataset.

In [5]:
```
def get_all_xml_files(root_directory):
    xml_files = []
    for root, _, files in os.walk(root_directory):
        for file in files:
            if file.endswith(".xml"):
```

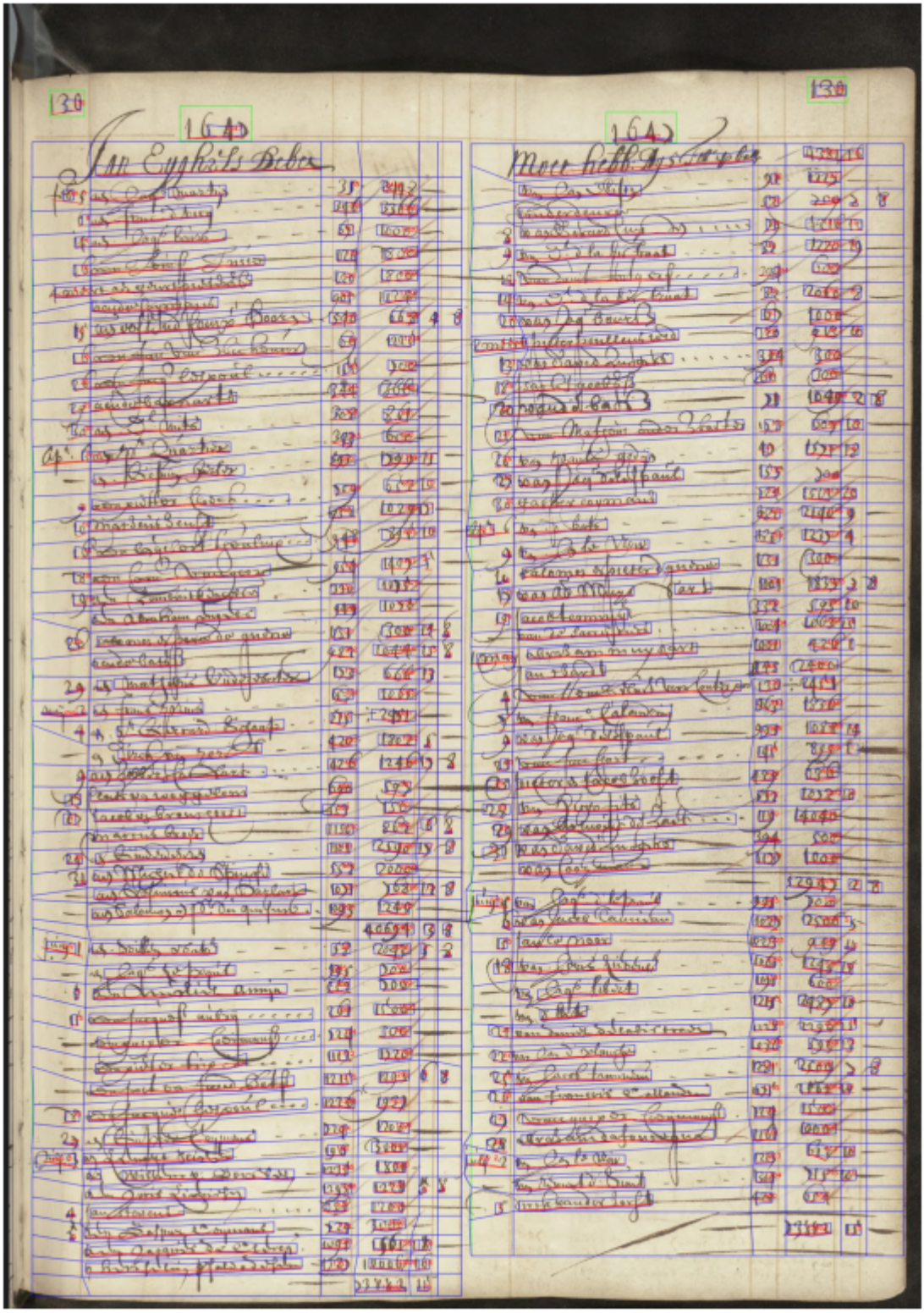
```
                xml_files.append(os.path.join(root, file))
    return xml_files

def extract_text_from_xml_gt(xml_file):
    ns = {'page': 'http://schema.primaresearch.org/PAGE/gts/pagecontent/2013

    # Parse the XML file
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Extract text content from all TextEquiv elements
    text_content = []
    for text_equiv in root.findall('.//page:TextEquiv', ns):
        text = text_equiv.find('page:Unicode', ns).text
        if text:
            text_content.append(text)

    return text_content

all_xml_GT = get_all_xml_files('/home/roderickmajoor/Desktop/Master/Thesis/G


words_per_page = []
for data_gt in all_xml_GT:
    annotated_words = extract_text_from_xml_gt(data_gt)
    words_per_page.append(len(annotated_words))


pages = list(range(1, len(words_per_page) + 1))

# Create a bar plot
plt.figure(figsize=(12, 6))
plt.bar(pages, words_per_page, color='skyblue')
plt.xlabel('Page Number')
plt.ylabel('Word Count')
plt.title('Word Count per Page')
plt.xticks(pages, rotation=45)
plt.tight_layout()
plt.show()

print("Total annotated words in dataset: " + str(sum(words_per_page)))
```



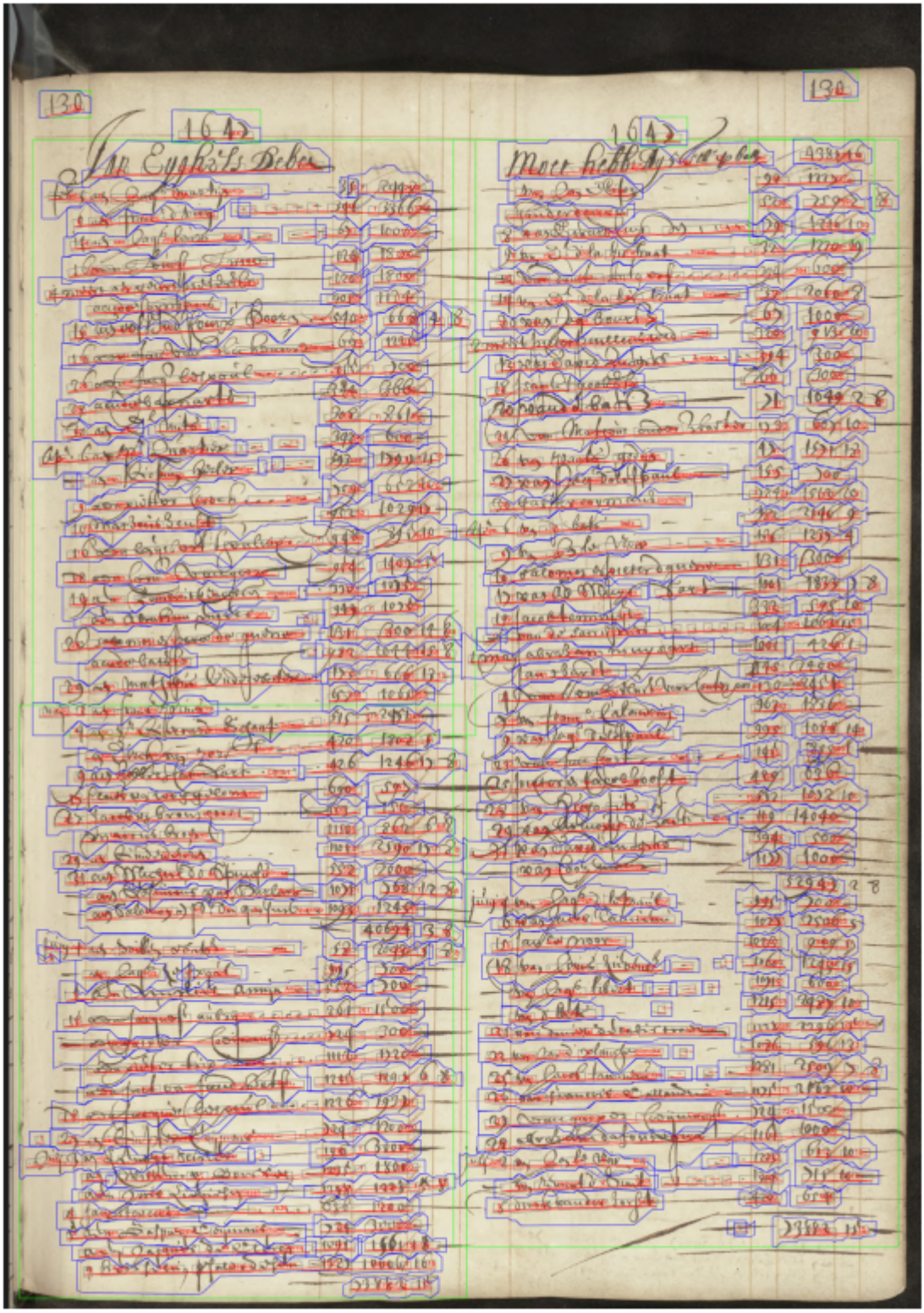

```
Total annotated words in dataset: 19191
```

To visualize these features, we plot the XML data over one of the original images so we can better understand the data. In the first cell, we do this for the ground truth XML file, which is hand annotated. In the second cell, we show this for the currently used tool to analyse these pages called Loghi. Comparing these features can give us an indication about the performance of the currently used tool.

In [6]:
```
# Cell 1: Run the GT script
%run xml_on_image_GT.py
```



In [7]:
```
# Cell 2: Run the Loghi script
%run xml_on_image_loghi.py
```



In [8]:
```
def extract_text_from_xml_gt(xml_file):
    ns = {'page': 'http://schema.primaresearch.org/PAGE/gts/pagecontent/2013

    # Parse the XML file
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Extract text content from all TextEquiv elements
```

```
    text_content = []
    for text_equiv in root.findall('.//page:TextEquiv', ns):
        text = text_equiv.find('page:Unicode', ns).text
        if text:
            text_content.append(text)

    return text_content

def extract_text_from_xml_loghi(xml_file):
    ns = {'page': 'http://schema.primaresearch.org/PAGE/gts/pagecontent/2013

    # Parse the XML file
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Extract text content from all TextEquiv elements
    text_content = []
    for word in root.findall('.//page:Word', ns):
        text_equiv = word.find('page:TextEquiv', ns)
        if text_equiv is not None:
            text = text_equiv.find('page:Unicode', ns).text
            if text:
                text_content.append(text)

    return text_content
```

Lets now show all the words in our ground truth XML file for this image. Note that in the ground truth file, the names in the credit/debit tables are not included, so we merely focus on the money amounts and dates

In [9]:
```
xml_file_path = '/home/roderickmajoor/Desktop/Master/Thesis/GT_data/55/page/
all_text_gt = extract_text_from_xml_gt(xml_file_path)
print(all_text_gt)
```

```
['130', '130', '1647', 'feb 5', '35', '2448', '6', '346', '3366', '14', '6
7', '1000', '16', '124', '1800', '4 mrt', '120', '1800', '905', '1134', '1
5', '540', '668', '4', '8', '16', '69', '1225', '26', '155', '700', '29',
'344', '366', '30', '308', '861', 'Ap 6', '343', '600', '593', '1799', '1
5', '9', '709', '652', '10', '10', '952', '1029', '17', '16', '948', '895',
'10', '18', '954', '1407', '5', '19', '330', '1075', '949', '1075', '20',
'131', '300', '14', '8', '982', '1044', '15', '8', '29', '173', '666', '1
3', 'may 2', '658', '1065', '4', '515', '2451', '420', '1802', '5', '9', '4
26', '1246', '17', '8', '17', '690', '597', '27', '553', '150', '1150', '86
2', '6', '8', '29', '1101', '2590', '17', '8', '31', '552', '2000', '1071',
'768', '12', '8', '1093', '1245', '40694', '3', '8', 'july 1', '58', '204
3', '5', '8', '995', '700', '5', '552', '700', '15', '261', '1500', '724',
'300', '1112', '1720', '1215', '1291', '6', '8', '18', '1226', '1937', '2
7', '724', '1200', 'July 3', '196', '3000', '1235', '1800', '1298', '1228',
'5', '8', '4', '636', '1200', '5', '724', '3000', '1091', '1561', '18', '13
27', '10006', '16', '73882', '15', '1647', '4381', '16', '92', '1227', '5
2', '759', '7', '8', '8', '79', '1216', '10', '9', '32', '1220', '19', '1
2', '394', '600', '14', '32', '2060', '8', '20', '67', '1000', '8 maart',
'320', '913', '10', '13', '394', '300', '18', '260', '300', '20', '71', '10
49', '2', '8', '21', '173', '607', '10', '26', '47', '1571', '18', '27', '1
55', '700', '30', '724', '1513', '10', 'Ap 6', '983', '2146', '9', '9', '18
6', '1237', '4', '10', '131', '300', '17', '1001', '1833', '7', '8', '19',
'332', '595', '10', '1004', '1063', '15', '1 may', '1001', '426', '1', '14
5', '2400', '4', '130', '2451', '7', '963', '1836', '9', '995', '1088', '1
4', '23', '145', '899', '1', '25', '429', '636', '28', '572', '1072', '10',
'29', '110', '14040', '31', '394', '500', '1177', '1000', '52947', '2',
'8', 'juny 4', '995', '700', '8', '1027', '2506', '5', '15', '1223', '949',
'17', '18', '1060', '1243', '15', '1191', '600', '1215', '2487', '10', '2
1', '1228', '2296', '15', '22', '1086', '596', '13', '25', '1281', '2509',
'7', '8', '26', '1175', '2562', '10', '27', '724', '1500', '28', '1161', '1
000', 'july 2', '1223', '613', '10', '1304', '715', '10', '5', '420', '65
4', '73882', '15']
```

Comparing this XML to the ground truth image above, we can see that the XML follows the tabular structure where its goes through the rows of the tables. Now we show the XML output for the Loghi tool

In [10]:
```
xml_file_path = '/home/roderickmajoor/Desktop/Master/Thesis/loghi/data/55/pa
all_text_loghi = extract_text_from_xml_loghi(xml_file_path)
print(all_text_loghi)
```

['130', 'Jan', 'Eijghels', 'Deben', 'feb:', '5', 'aen', 'Paqe.', 'Martijn',
'35', 'ƒ', '2448', '-', '-', '-', '-', '-', '346', 'ƒ', '3366', '6', 'aen',
'Franc.', "d'vicq", '-67', '1000', '—', '-', '14', 'aen', 'Jage.', 'bours',
'1800', '124', '160n', 'Lourisz', 'Lusse', '1800', '1120', 't', 'meert', 'a
en', 'geurt', 'Pietersz', '905', '1134', 'aende', 'Coupans', '668', '8',
'4', '15', 'aende', 'oost', "Ind'", 'Compe.', 'Hoorn', '540', '1225', '16',
'aent', 'fait', 'van', 'Wickenoort', '69', '26', 'aot', 'Jacq', 'lespoul',
'„55', 'ƒ', '100', '844', '366', '29', 'aende', 'barnarts', '308', 'ƒ', '86
1', '„', 'mits', '30', 'aen', '600', '.', '342', '—', '—', 'Quartier', 'ap
n.', 'Caextpr.', '179', '9', 'ƒ', '15', '692', '-', '-', '—', 'aen.', 'Pios
uim', 'Heller', '652/0', 'p09', '9', 'aen', 'pietter', 'Cloeck', 'f', '52',
'10291', '10,', 'Matheus', 'Senst', 'Ro10', '16', 'aer', 'hijsbert', 'theul
inx', 'ƒ', '948', '954', '1407', 'ƒ', '5', '28', 'aen', 'Corn.', 'vangoor',
'1075', '330', '19', 'aen', 'Comboit', 's', 'Jacobsen', '949', 'ƒ', '1078',
'*', '.', '.', 'aen', 'Abraham', 'engels', '131', '300', 'ƒ', '148', '26',
'salo', 'mos', 'es', 'perco', 'de', 'quena', '+', '982', '1044f158', 'aend
e', 'baths', '173', 'ƒ', '666', '13', '29', 'aen', 'mathasus', 'Ondesebote
r', '658', '1065', 'maij', '2', 'aen', 'Frans', 'sprins', '4', 'an.', 'h
r.', 'Gsercard:', 'Schaap.', '.', '.', '515.', '2451', 'ean', 'Dirck', 'va
n', 'Horst', '.', '420', 'ƒ', '1802', '5', '426', '124613', '8', '9', 'aen
d', 'robbert', 'Caffart', '690', '597', 'Een', 'Claes', 'van', 'woggelom',
'150', 'S53', '27', 'Jacobvn', 'bronchorst', '1186', '862', '68', 'Marcus',
'broes', '1101', '2590', '17', '8', '29', 'aen', 'Ginderdeuren', '552', '20
00', '31', 'aen', 'Michiel', 'de', 'Spinose', '768', 'ƒ', '128', '1071', 'a
en', 'Josumus', 'van', 'Barlaer', 'aen', 'Salomon', 'en', 'Po.', 'du', 'que
smo', '1093', 'ƒ', '1245', '40694', '38', 'Junij', 'aen', 'Willem', 'wats',
'58', '2042', '5', '58', '5.', '295', '300', 'xgut', 'of', 'aen', 'hays.',
'5552', '100', '5', 'aen', 'michiel', 'Amijn', '15', 'adnjacquesz', 'aubri
j.', '.', '.', '.', '.', '261', '1500', 'adagaiper', 'Coeijmansz.', '124',
'-', '300-', 'adrpieter', 'tripdn', 'ƒ', '1112', '1720', '1215', '129',
'&', '6', '8.', 'aen', 'port', 'van', 'fand', 'Bats.', '18', 'adnfacque',
'Gespoul', '-', '.', '.', '1226', '1931', '27', 'aen', 'Gapper', 'Coijman
s', 'f24', 'ƒ', '1200', '1', '196', 'ƒ', '3000', '-', 'Juij:', 'van', 'Hena
rt', 'Slicher', 'a', 'Willem', 'v:', 'Borssel', '1235', 'ƒ', '1800', '129
8', '1228', 'ƒ', '55', 'aen', 'Joris', 'Linniesen', '.', '636', '1200',
'4', 'Jan', 'stevens', '724', '3000-', '5', 'aen', 'Gaspur', 'Ceijmans', '1
56:l18', '191', 'aen', 'Jacques', 'de', 'Clercq', '9', 'hem', 'selven', 'pe
salvo', 'desen', '1527', '10006', 'ƒ', '16', '1785½', '15', 'Moet', 'heb
b:', 'By5', 'Feb:', '10', 'bal', 'van', 'Jan', 'Elise', 'ginderdeurg', '8',
'8van', 'Gerard', 'Luysten', '9', 'van', 'Do:', "d'le", 'histraat', '.', '3
2', '1220-19', 'th', 'van', 'danit', 'Antgers.', '.', '.', '.', '.', '.',
'394', '600', '32ƒ', '2060„8', '14', 'van.', 'D:', 'dla', 'bistraat', '67',
'1000', '20', 'van', 'Jaqe.', 'boursz', 'P20', '9', '13', '10', 'meert', 'P
ieter', 'Poullens', 'wed', '394', '300', '13', 'van', 'David', 'Zuegers',
'300', '266', '18', 'Isack', 'Jacobssz', '1049', '28', '21', '20', 'vande',
'bats', '60', '7', '10', '21', 'van', 'Mathout', 'onder', 'Waster', '173',
'47', '1571', '18', '26', 'van', 'ppauls', 'godijn', '155', '100', '27', 'v
an', 'ssaij', 'delespaul', 'f', '24', '1513', '9', '30', 'gasster', 'coyman
s', '182', '2146', '9', 'Apo.', '6', 'vn', 'd', 'bats', '186', '1237', '4',
'9', 'van', 'Iole', 'Boi', 'ƒ', '100', 'to', 'Salomes', '&', 'pieterdquen
e', '131', '18', '1001', '18337', 'dart', '17', 'van', 'ad', 'Muys', '595',
'10', '932', '19', 'Jacob', 'tenmin', 'van', 'de', 'sarresstiers.', '.',
'.', '.', '.', '.', '.', '1004', '1663', '15', '1', '1001', '4261', '(may',
'abraham', 'muygart', 'R45', '2400', 'Jan', 'Bardt', '4', 'vanAem', 'Selus
t', 'voor', 'Conten', 'por.', '130', '2451', '963', '1836', 'van', 'Franc
o.', 'Calandrinj', '995', '1088', '14', '9', 'van', '10g', 'daelefpaul', '8

```
991', '145', '-', 'ƒ', '23', 'van', 'Jan', 'Chart', '636', '429', 'V', 'poc
ctor/', 'en', 'Japob', 'hooft', 'f512.', '1012', '10', '28', 'van', 'Vugo',
'fits', '119', '14040', '29', 'pag', 'Aernoits', 'de', 'Zaet', '294', '50
0', '31se', 'an', 'David', 'Lntgers', '1000', 'pi3)', 'vande', 'Coeiman',
'5294', 'junij', '4', 'van', 'Jage.', "d'le", 'paul', '395', '700', '2506
5', 'van', 'Jacob', 'Tamieau', '1027', '1223', '949', '17', '15', 'Janle',
'noor', '1060', '—', '1242', '15', '—', 'I8', 'van', 'Joris', 'Lievensz',
'1191', '600', '—', '—', '—', 'van', 'hags.', 'libert', '2487', '10', '121
5', '—', 'van', "d'dte.", '1228', '229615-', '21', 'van', 'daniel', 'de',
'labistraet', '1086„', '596', '13', '—', '22', 'pan', 'Jan', "d'", 'planch
e', '1281', '2509', '7', '8', '—', '-', '25', 'van', 'Jacob', 'tamuneau',
'26', 'van', 'Francois', 'Callandri', '1175', '256210', '27', 'vangasper',
'Coeijmunss', '724', '150', '28', 'abraham', 'dafondeau', '116', '1000', '6
12', '10-', '1223', 'Julij', '2', 'van', 'Jan', 'le', 'Noij', '-', '71510-
', '-1', '—', '—', '-', '1304.', 'van', 'remont', "d'", 'Smit', '420', '65
4', '5', 'dirck', 'vander', 'horst', '-', '-', '17882', '15', '438116', '9
2.', '1227-', '52', '759r.', '1216', '10', '79', '47', '1647', '130']
```

Looking at the Loghi XML output and image, we can see that (besides the recognized names which we discarded in the ground truth) a lot of false positives are detected, mainly '.' and '-' characters that are typically detected due to small marks/stripes on the image. In order to get better results, we should try to pre-process our image such that these marks/stripes are not detected as text.

For now let's just get rid of the elements in the XML files that don't contain numbers. This should give us a better comparison of the numbers found and might tell us something about the order of the Loghi XML file.

In [11]:
```
filtered_list = [item for item in all_text_loghi if any(char.isdigit() for c

print(filtered_list)
```

```
['130', '5', '35', '2448', '346', '3366', '6', '-67', '1000', '14', '1800',
'124', '160n', '1800', '1120', '905', '1134', '668', '8', '4', '15', '540',
'1225', '16', '69', '26', '„55', '100', '844', '366', '29', '308', '861',
'30', '600', '342', '179', '9', '15', '692', '652/0', 'p09', '9', '52', '10
291', '10,', 'Ro10', '16', '948', '954', '1407', '5', '28', '1075', '330',
'19', '949', '1078', '131', '300', '148', '26', '982', '1044f158', '173',
'666', '13', '29', '658', '1065', '2', '4', '515.', '2451', '420', '1802',
'5', '426', '124613', '8', '9', '690', '597', '150', 'S53', '27', '1186',
'862', '68', '1101', '2590', '17', '8', '29', '552', '2000', '31', '768',
'128', '1071', '1093', '1245', '40694', '38', '58', '2042', '5', '58',
'5.', '295', '300', '5552', '100', '5', '15', '261', '1500', '124', '300-',
'1112', '1720', '1215', '129', '6', '8.', '18', '1226', '1931', '27', 'f2
4', '1200', '1', '196', '3000', '1235', '1800', '1298', '1228', '55', '63
6', '1200', '4', '724', '3000-', '5', '156:l18', '191', '9', '1527', '1000
6', '16', '1785½', '15', 'By5', '10', '8', '8van', '9', '32', '1220-19', '3
94', '600', '32ƒ', '2060„8', '14', '67', '1000', '20', 'P20', '9', '13', '1
0', '394', '300', '13', '300', '266', '18', '1049', '28', '21', '20', '60',
'7', '10', '21', '173', '47', '1571', '18', '26', '155', '100', '27', '24',
'1513', '9', '30', '182', '2146', '9', '6', '186', '1237', '4', '9', '100',
'131', '18', '1001', '18337', '17', '595', '10', '932', '19', '1004', '166
3', '15', '1', '1001', '4261', 'R45', '2400', '4', '130', '2451', '963', '1
836', '995', '1088', '14', '9', '10g', '8991', '145', '23', '636', '429',
'f512.', '1012', '10', '28', '119', '14040', '29', '294', '500', '31se', '1
000', 'pi3)', '5294', '4', '395', '700', '25065', '1027', '1223', '949', '1
7', '15', '1060', '1242', '15', 'I8', '1191', '600', '2487', '10', '1215',
'1228', '229615-', '21', '1086„', '596', '13', '22', '1281', '2509', '7',
'8', '25', '26', '1175', '256210', '27', '724', '150', '28', '116', '1000',
'612', '10-', '1223', '2', '71510-', '-1', '1304.', '420', '654', '5', '178
82', '15', '438116', '92.', '1227-', '52', '759r.', '1216', '10', '79', '4
7', '1647', '130']
```

In [12]:
```
print(len(all_text_gt))
print(len(filtered_list))
```

```
341
314
```

Now, The list of numbers in the Loghi detected text is shorter than the ground truth list. This is because some of the numbers in the ground truth are actually not recognized as numbers by Loghi. This is another error in Loghi that we should try to improve. Other things we notice are that there are some words/numbers that become merged in Loghi's detection. The last thing that we can see from the images and lists, is that Loghi is not able to capture the table structure of the page correctly thus creating a wrong order of words in the list.

Some of the things we noticed going wrong in Loghi's output:

1. False positives detected (typically due to marks / stripes on the image that are recognized as text)
2. Numbers are not detected as numbers but rather as text / other characters
3. Words/numbers are merged or seperated (i.e. 16 becomes 1, 6 or 2, 8 becomes 28 etc.)
4. Numbers not correctly recognized (i.e. 16 becomes 18 etc.)
5. Output not in the correct layout

Analysis 3:

We can try to find the Word Error Rate (WER) and Character Error Rate (CER) from the output of Loghi. However, since we have a shorter list as Loghi output than the ground truth and we know the order is not correct, we can expect a very bad score.

In [13]:
```
wer = jiwer.wer(all_text_gt[:314], filtered_list)
cer = jiwer.cer(all_text_gt[:314], filtered_list)
print("Word Error Rate:", wer)
print("Character Error Rate:", cer)
```

```
Word Error Rate: 0.9783950617283951
Character Error Rate: 1.0684150513112884
```

We will check if there are matching words in both lists. Here we do not look at word order, merely if a word occurs somewhere in both lists. This at least gives us an idea whether Loghi has some matching words with the ground truth.

In [14]:
```
def count_matching_words(list1, list2):
    # Convert lists to dictionaries to count occurrences
    dict1 = {}
    dict2 = {}

    # Count occurrences in list 1
    for word in list1:
        dict1[word] = dict1.get(word, 0) + 1

    # Count occurrences in list 2
    for word in list2:
        dict2[word] = dict2.get(word, 0) + 1

    # Find the intersection of the two dictionaries
    matching_words = set(dict1.keys()).intersection(set(dict2.keys()))

    # Count the occurrences based on the minimum count in both lists
    total_count = sum(min(dict1.get(word, 0), dict2.get(word, 0)) for word i

    return total_count
```

In [15]:
```
matching_count = count_matching_words(all_text_gt, filtered_list)
print("Number of matching words:", matching_count)
```

```
Number of matching words: 215
```

We find 215 matching words for this page, out of 314 total words in the (filtered) Loghi output. So about 68% of Loghi's output is also found in the ground truth. However, we still have no idea about the correct word order.

We can also calculate the jaccard similarity score

In [16]:
```
def jaccard_similarity(list1, list2):
    set1 = set(list1)
    set2 = set(list2)
    intersection = len(set1.intersection(set2))
    union = len(set1) + len(set2) - intersection
    return intersection / union if union != 0 else 0
```

In [17]:
```
print("Jaccard Similarity:", jaccard_similarity(all_text_gt, filtered_list))
```

```
Jaccard Similarity: 0.4708029197080292
```

Now lets show the mean metrics for all images

In [18]:
```
all_xml_GT = get_all_xml_files('/home/roderickmajoor/Desktop/Master/Thesis/G
all_xml_loghi = get_all_xml_files('/home/roderickmajoor/Desktop/Master/Thesi

list1_filenames = [os.path.basename(path) for path in all_xml_GT]
list2_filenames = [os.path.basename(path) for path in all_xml_loghi]
if list1_filenames == list2_filenames:
    print("The lists are in the same order based on filenames.")
```

```
The lists are in the same order based on filenames.
```

In [19]:
```
def compute_mean_metrics(all_xml_GT, all_xml_loghi):

    wer_total = 0
    cer_total = 0
    matching_count_total = 0
    jaccard_similarity_total = 0
    num_images = 0

    for data_gt, data_loghi in zip(all_xml_GT, all_xml_loghi):
        all_text_gt = extract_text_from_xml_gt(data_gt)
        all_text_loghi = extract_text_from_xml_loghi(data_loghi)
        filtered_list = [item for item in all_text_loghi if any(char.isdigit
        wer_total += jiwer.wer(all_text_gt[:len(filtered_list)], filtered_li
        cer_total += jiwer.cer(all_text_gt[:len(filtered_list)], filtered_li
        matching_count_total += count_matching_words(all_text_gt, filtered_l
        jaccard_similarity_total += jaccard_similarity(all_text_gt, filtered
        num_images += 1

    mean_wer = wer_total / num_images
    mean_cer = cer_total / num_images
    mean_matching_count = matching_count_total / num_images
    mean_jaccard_similarity = jaccard_similarity_total / num_images

    return mean_wer, mean_cer, mean_matching_count, mean_jaccard_similarity
```

In [20]:
```
mean_wer, mean_cer, mean_matching_count, mean_jaccard_similarity = compute_m

print("Mean Word Error Rate:", mean_wer)
print("Mean Character Error Rate:", mean_cer)
print("Mean Matching Count:", mean_matching_count)
print("Mean Jaccard Similarity:", mean_jaccard_similarity)
```

```
Mean Word Error Rate: 0.9810198160684107
Mean Character Error Rate: 1.0649680926496443
Mean Matching Count: 0.6609155616036941
Mean Jaccard Similarity: 0.4582747425528701
```

As we expected, we can see a bad score for WER and CER (mainly because of the word order) and we see that the mean matching count and jaccard similarity show that loghi does manage do get some output correct.

Analysis 4:

Now we will match the bounding boxes of the table cells in the ground truth with the bounding boxes of words found by loghi based on their intersection over union (IoU) value. The IoU value is a metric that can be used to determine the overlap between two sets (in this case, between two bounding boxes). This way, we can see the correct order. We will do this for the first image again.

In [21]:
```
data_gt, data_loghi = main()
```

In [22]:
```
print(data_gt)
print(data_loghi)
```

['130', '130', '1647', '1647', 'feb 5', '35', '2448', '6', '346', '3366',
'14', '67', '1000', '16', '124', '1800', '4 mrt', '120', '1800', '905', '11
34', '15', '540', '668', '4', '8', '16', '69', '1225', '26', '155', '700',
'29', '344', '366', '30', '308', '861', 'Ap 6', '343', '600', '593', '179
9', '15', '9', '709', '652', '10', '952', '1029', '16', '948', '895', '18',
'954', '1407', '5', '19', '330', '1075', '949', '1075', '20', '131', '300',
'14', '982', '1044', '29', '173', '666', '13', 'may 2', '658', '1065', '4',
'515', '2451', '420', '1802', '5', '9', '426', '1246', '8', '17', '690', '5
97', '27', '553', '150', '1150', '862', '6', '29', '1101', '2590', '17',
'8', '31', '552', '2000', '1071', '768', '12', '1093', '1245', '40694',
'3', 'july 1', '58', '2043', '5', '8', '995', '700', '5', '552', '700', '1
5', '261', '1500', '724', '300', '1112', '1720', '1215', '1291', '6', '8',
'18', '1226', '1937', '27', '724', '1200', 'July 3', '196', '3000', '1235',
'1800', '1298', '1228', '5', '4', '636', '5', '724', '3000', '1091', '156
1', '1327', '10006', '16', '15', '4381', '92', '1227', '52', '759', '8',
'8', '79', '1216', '10', '9', '32', '1220', '12', '394', '600', '14', '32',
'2060', '20', '67', '1000', '8 maart', '320', '913', '10', '13', '394', '30
0', '18', '260', '300', '20', '71', '1049', '8', '21', '173', '607', '10',
'26', '47', '1571', '18', '27', '155', '700', '30', '724', '1513', '10', 'A
p 6', '983', '2146', '9', '9', '186', '1237', '4', '10', '131', '300', '1
7', '1001', '1833', '7', '19', '332', '595', '10', '1004', '1063', '15', '1
may', '1001', '426', '145', '2400', '4', '130', '2451', '7', '963', '1836',
'9', '995', '1088', '14', '23', '145', '899', '25', '429', '636', '28', '57
2', '1072', '10', '29', '110', '14040', '31', '394', '500', '1177', '1000',
'juny 4', '995', '700', '8', '1027', '2506', '15', '1223', '949', '17', '1
8', '1060', '1243', '15', '1191', '600', '1215', '2487', '10', '21', '122
8', '2296', '22', '1086', '596', '13', '25', '1281', '2509', '7', '8', '2
6', '1175', '2562', '27', '724', '1500', '28', '1161', '1000', 'july 2', '1
223', '613', '10', '1304', '715', '5', '420', '654', '15']
['130', '130', '1647', '47', 'feb:', 'f', '2448', '6', '346', '3366', '14',
'-67', '1000', '160n', '124', '1800', 'meert', '1120', '1800', '905', '113
4', '15', '540', '668', '4', '8', '16', '69', '1225', '26', '„55', '100',
'29', '844', '366', '30', '308', '861', 'apn.', '342', '600', '692', '179',
'15', '9', 'p09', '652/0', '10,', '52', '10291', '16', '948', 'Ro10', '28',
'954', '1407', '5', '19', '330', '1075', '949', '1078', '26', '131', '300',
'148', '982', '1044f158', '29', '173', '666', '13', 'maij', '658', '1065',
'4', '.', '2451', '420', '1802', '5', '9', '426', '124613', '8', 'Een', '69
0', '597', '27', 'S53', '150', '1186', '862', '68', '29', '1101', '2590',
'17', '8', '31', '552', '2000', '1071', '768', '128', '1093', '1245', '4069
4', '38', 'Junij', '58', '2042', '5', '58', '295', '300', '5', '5552', '10
0', '15', '261', '1500', '124', '300-', '1112', '1720', '1215', '129', '6',
'8.', '18', '1226', '1931', '27', 'f24', '1200', 'Juij:', '196', '3000', '1
235', '1800', '1298', '1228', '55', '4', '1200', '5', '724', '3000-', '19
1', '156:l18', '1527', '10006', '16', '15', '438116', '92.', '1227-', '52',
'759r.', '8', '8van', '79', '1216', '10', '9', '32', '1220-19', 'th', '.',
'394', '14', '32ƒ', '2060„8', '20', '67', '1000', 'meert', 'P20', '13', '1
0', '13', '394', '300', '18', '266', '300', '20', '21', '1049', '28', '21',
'173', '60', '10', '26', '47', '1571', '18', '27', '155', '100', '30', '2
4', '1513', '9', 'Apo.', '182', '2146', '9', '9', '186', '1237', '4', 'to',
'131', '100', '17', '1001', '18337', '18', '19', '932', '595', '10', '100
4', '1663', '15', '(may', '1001', '4261', 'R45', '2400', '4', 'por.', '245
1', 'van', '963', '1836', '9', '995', '1088', '14', '23', '145', '8991',
'V', '429', '636', '28', 'f512.', '1012', '10', '29', '119', '14040', '31s
e', '294', '500', 'pi3)', '1000', 'junij', '395', '700', 'van', '1027', '25
065', '15', '1223', '949', '17', 'I8', '1060', '1242', '15', '1191', '600',
'1215', '2487', '10', '21', '1228', '229615-', '22', '1086„', '596', '13',

```
'25', '1281', '2509', '7', '8', '26', '1175', '256210', '27', '724', '150',
'28', '116', '1000', 'Julij', '1223', '612', '10-', '1304.', '71510-', '5',
'420', '654', '15']
```

Now we will perform the same tests as before

In [23]:
```
wer = jiwer.wer(data_gt, data_loghi)
cer = jiwer.cer(data_gt, data_loghi)
matching_count = count_matching_words(data_gt, data_loghi) / len(data_loghi)
jaccard_similarity = jaccard_similarity(data_gt, data_loghi)

print("Word Error Rate:", wer)
print("Character Error Rate:", cer)
print("Matching Count:", matching_count)
print("Jaccard Similarity:", jaccard_similarity)
```

```
Word Error Rate: 0.38198757763975155
Character Error Rate: 0.2139689578713969
Matching Count: 0.6495176848874598
Jaccard Similarity: 0.45
```

Now we have way better scores for word error and character error. However we could not match every ground truth word to a loghi word based on IoU. Only 311 out of 341 ground truth regions were matched to a loghi region. But this does show that if we can find the correct layout from the loghi output, the performance will drastically increase.

In [ ]: