Review of Transcription Factors in Autism Spectrum Disorder via NLP Analyses with Systematic Search-driven Data

By. Young Beum Cho

Goal

Visualization and clustering of Gene_DOID associations

Setup | Data Collection

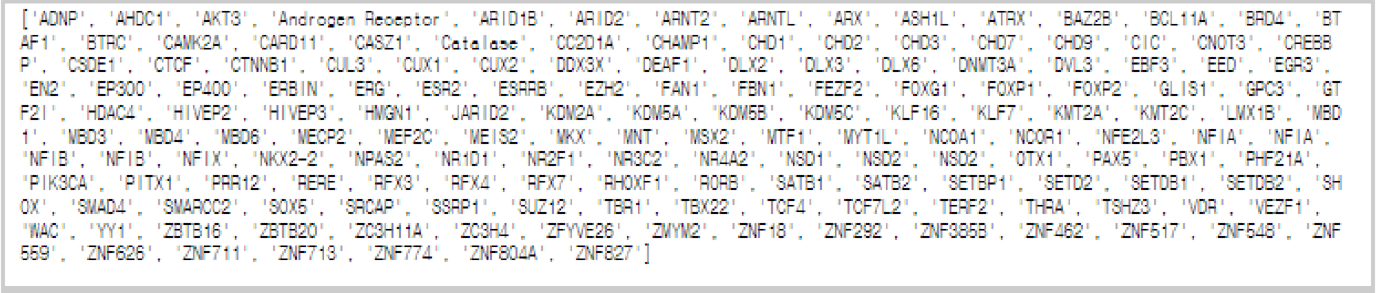
Selenium WebDriver is a tool commonly used for automating web browsers. It allows developers and testers to interact with web pages using a programming language, such as Python or Java.

To install the web driver, I followed these steps:
1) Open the Chrome browser.
2) Click on the three vertical dots located in the upper-right corner of the browser window.
3) In the drop-down menu, hover over "Help" and click "About Google Chrome."
4) The "About Google Chrome" page will open and display the current version of your Chrome browser. The browser will also automatically check for updates and install them if available.

Then, I went to “https://chromedriver.chromium.org/downloads” to download the driver that matches my version and OS, and placed the file in the same directory as the project folder that contains the ipynb file.

To collect data for this project, I used a combination of Python libraries and web scraping techniques. The main goal was to gather information about the associations between transcription factors and Autism Spectrum Disorder (ASD).

We started by creating a list of transcription factors (TFs) potentially related to ASD. This list was derived from the data from Kroll Lab (Department of Developmental Biology), The list contains a total of 162 unique TFs.



```
['ADNP', 'AHDC1', 'AKT3', 'Androgen Receptor', 'ARID1B', 'ARID2', 'ARNT2', 'ARNTL', 'ARX', 'ASH1L', 'ATRX', 'BAZ2B', 'BCL11A', 'BRD4', 'BT
AF1', 'BTRC', 'CAMK2A', 'CARD11', 'CASZ1', 'Catalase', 'CC2D1A', 'CHAMP1', 'CHD1', 'CHD2', 'CHD3', 'CHD7', 'CHD9', 'CIC', 'CNOT3', 'CREBB
P', 'CSDE1', 'CTCF', 'CTNNB1', 'CUL3', 'CUX1', 'CUX2', 'DDX3X', 'DEAF1', 'DLX2', 'DLX3', 'DLX6', 'DNMT3A', 'DVL3', 'EBF3', 'EED', 'EGR3',
'EN2', 'EP300', 'EP400', 'ERBIN', 'ERG', 'ESR2', 'ESRRB', 'EZH2', 'FAN1', 'FBN1', 'FEZF2', 'FOXG1', 'FOXP1', 'FOXP2', 'GLIS1', 'GPC3', 'GT
F2I', 'HDAC4', 'HIVEP2', 'HIVEP3', 'HMGN1', 'JARID2', 'KDM2A', 'KDM5A', 'KDM5B', 'KDM5C', 'KLF16', 'KLF7', 'KMT2A', 'KMT2C', 'LMX1B', 'MBD
1', 'MBD3', 'MBD4', 'MBD6', 'MECP2', 'MEF2C', 'MEIS2', 'MKX', 'MNT', 'MSX2', 'MTF1', 'MYT1L', 'NCOA1', 'NCOR1', 'NFE2L3', 'NFIA', 'NFIA',
'NFIB', 'NFIB', 'NFIX', 'NKX2-2', 'NPAS2', 'NR1D1', 'NR2F1', 'NR3C2', 'NR4A2', 'NSD1', 'NSD2', 'NSD2', 'OTX1', 'PAX5', 'PBX1', 'PHF21A',
'PIK3CA', 'PITX1', 'PRR12', 'RERE', 'RFX3', 'RFX4', 'RFX7', 'RHOXF1', 'RORB', 'SATB1', 'SATB2', 'SETBP1', 'SETD2', 'SETDB1', 'SETDB2', 'SH
OX', 'SMAD4', 'SMARCC2', 'SOX5', 'SRCAP', 'SSRP1', 'SUZ12', 'TBR1', 'TBX22', 'TCF4', 'TCF7L2', 'TERF2', 'THRA', 'TSHZ3', 'VDR', 'VEZF1',
'WAC', 'YY1', 'ZBTB16', 'ZBTB20', 'ZC3H11A', 'ZC3H4', 'ZFYVE26', 'ZMYM2', 'ZNF18', 'ZNF292', 'ZNF385B', 'ZNF462', 'ZNF517', 'ZNF548', 'ZNF
559', 'ZNF626', 'ZNF711', 'ZNF713', 'ZNF774', 'ZNF804A', 'ZNF827']
```

Figure1: 162 Transcription Factors

Next, I used the Selenium WebDriver to automate the process of searching for relevant articles on the PubMed website (https://pubmed.ncbi.nlm.nih.gov). The WebDriver allowed us to programmatically interact with the web page and perform actions such as entering search terms, clicking buttons, and navigating through search results.

We defined a set of search terms related to each transcription factor in the list, focusing on terms associated with the brain, autism, stem cells, and mouse while excluding terms related to cancer and tumors. These search terms were then entered into the PubMed search bar to retrieve articles related to each transcription factor.

Once the search results were obtained, I used BeautifulSoup, a Python library for web scraping, to extract relevant information from the web pages, such as article titles, abstracts, and URLs. I then stored this information in a DataFrame for further processing and analysis.

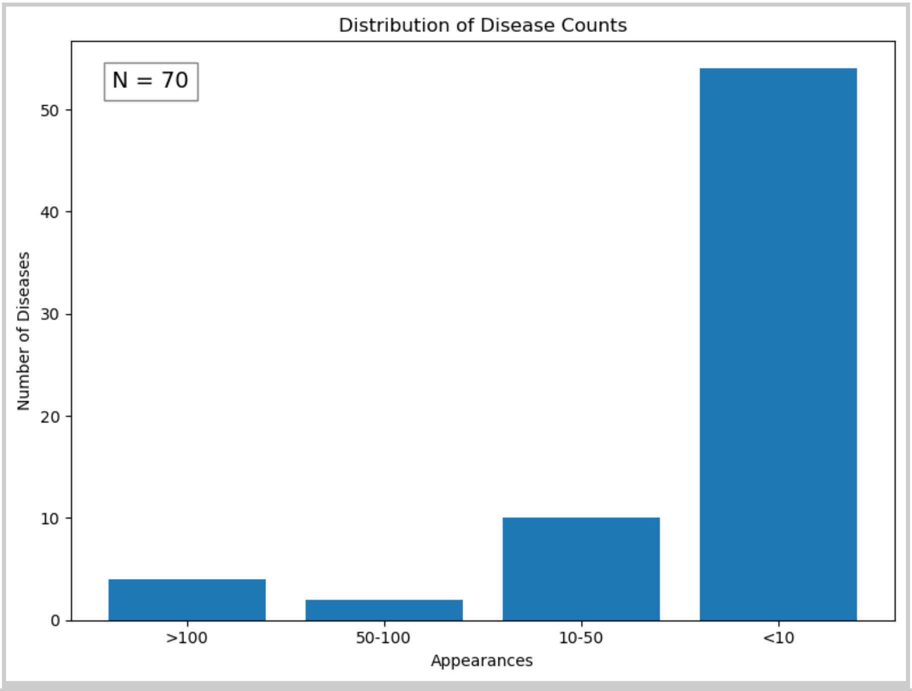
To ensure that I did not miss any relevant articles, I checked for duplicates in the search results and maintained a record of unique articles for each transcription factor.

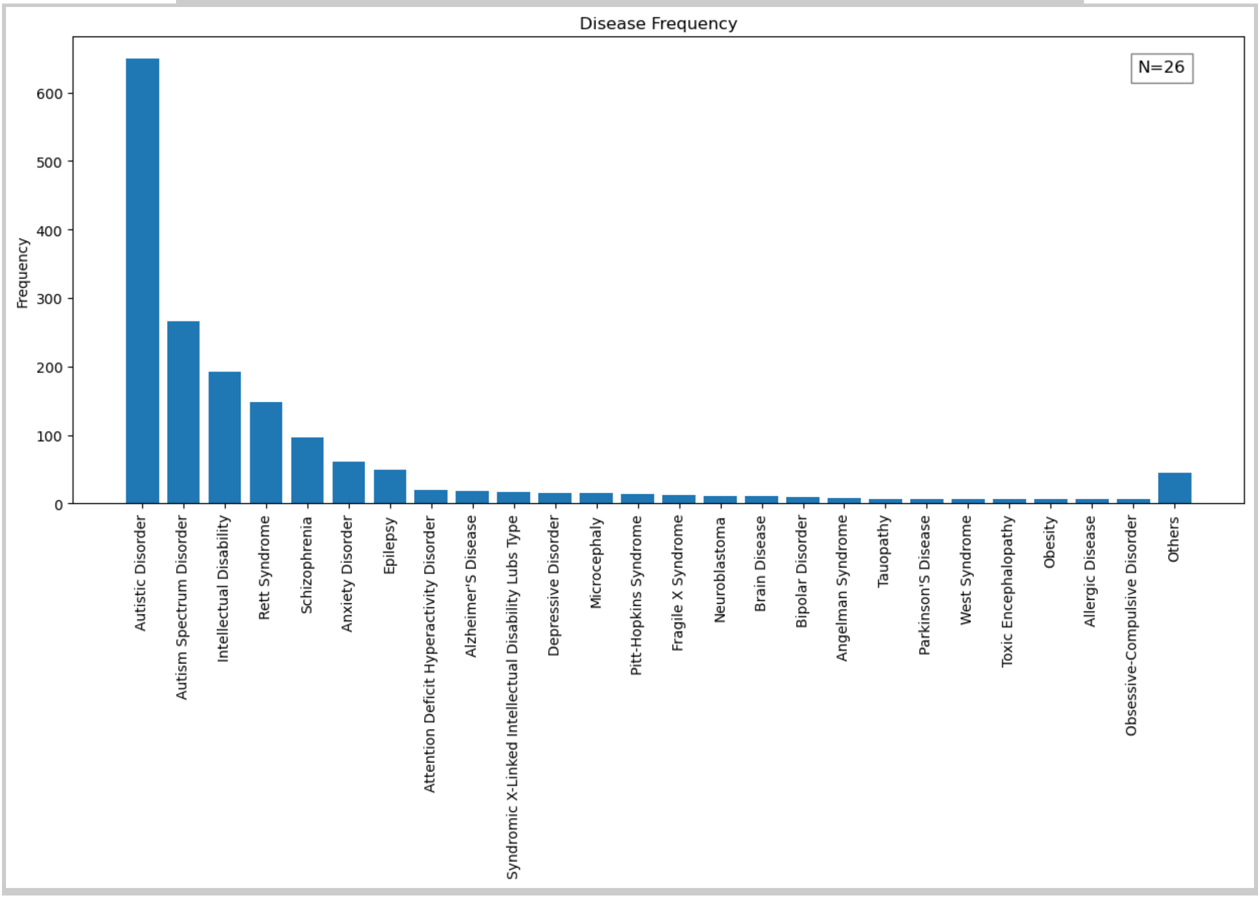
Data Preprocessing

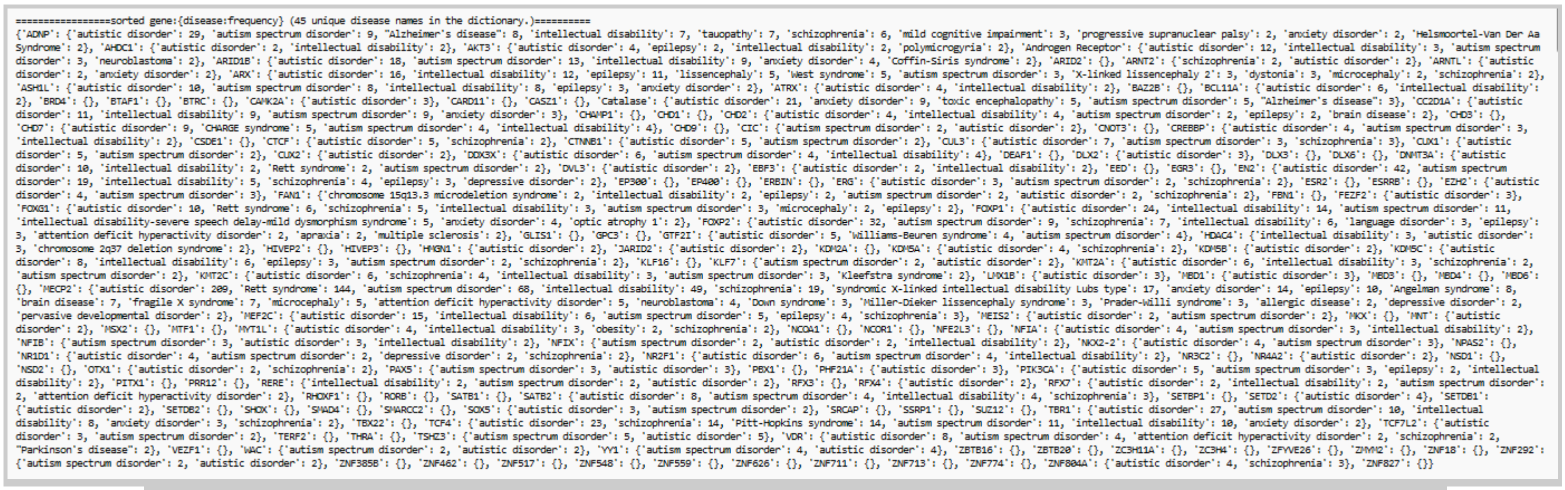
During the data preprocessing phase, I focused on cleaning and organizing the collected data to make it suitable for further analysis and visualization. The first step was to parse the text from the article titles and abstracts using Natural Language Processing (NLP) techniques. To gain deeper insights into the relationships between transcription factors and diseases, I used the BioPortal Annotator API to identify preferred labels for disease ontology (DOID) terms present in the text. This allowed me to create a dictionary of transcription factors, each associated with a nested dictionary of diseases and their frequencies in the collected data. I filtered out any diseases with a frequency of 1 to reduce noise and sorted the nested dictionaries based on frequency in descending order.

Using Bioportal, I also employed Gene Ontology (GO) to detect additional keywords related to specific biological processes mentioned in the text. This provided a more comprehensive understanding of the connections between transcription factors and their potential roles in ASD development.

With the dictionaries generated and the data preprocessed, I was now ready to move on to the visualization and analysis stages of the project.


Distribution of Disease Counts
N = 70
Number of Diseases
0
10
20
30
40
50
>100
50-100
10-50
<10
Appearances


Disease Frequency
N=26
Frequency
0
100
200
300
400
500
600
Autistic Disorder
Autism Spectrum Disorder
Intellectual Disability
Rett Syndrome
Schizophrenia
Anxiety Disorder
Epilepsy
Attention Deficit Hyperactivity Disorder
Alzheimer'S Disease
Syndromic X-Linked Intellectual Disability Lubs Type
Depressive Disorder
Microcephaly
Pitt-Hopkins Syndrome
Fragile X Syndrome
Neuroblastoma
Brain Disease
Bipolar Disorder
Angelman Syndrome
Tauopathy
Parkinson'S Disease
West Syndrome
Toxic Encephalopathy
Obesity
Allergic Disease
Obsessive-Compulsive Disorder
Others



```
================sorted gene:{disease:frequency} (45 unique disease names in the dictionary.)==========
{'ADNP': {'autistic disorder': 29, 'autism spectrum disorder': 9, "Alzheimer's disease": 8, 'intellectual disability': 7, 'tauopathy': 7, 'schizophrenia': 6, 'mild cognitive impairment': 3, 'progressive supranuclear palsy': 2, 'anxiety disorder': 2, 'Helsmoortel-Van Der Aa
Syndrome': 2}, 'AHDC1': {'autistic disorder': 2, 'intellectual disability': 2}, 'AKT3': {'autistic disorder': 4, 'epilepsy': 2, 'intellectual disability': 2, 'polymicrogyria': 2}, 'Androgen Receptor': {'autistic disorder': 12, 'intellectual disability': 3, 'autism spectrum
disorder': 3, 'neuroblastoma': 2}, 'ARID1B': {'autistic disorder': 18, 'autism spectrum disorder': 13, 'intellectual disability': 9, 'anxiety disorder': 4, 'Coffin-Siris syndrome': 2}, 'ARID2': {}, 'ARNT2': {'schizophrenia': 2, 'autistic disorder': 2}, 'ARNTL': {'autistic
disorder': 2, 'anxiety disorder': 2}, 'ARX': {'autistic disorder': 16, 'intellectual disability': 12, 'epilepsy': 11, 'lissencephaly': 5, 'West syndrome': 5, 'autism spectrum disorder': 3, 'X-linked lissencephaly 2': 3, 'dystonia': 3, 'microcephaly': 2, 'schizophrenia': 2},
'ASH1L': {'autistic disorder': 10, 'autism spectrum disorder': 8, 'intellectual disability': 8, 'epilepsy': 3, 'anxiety disorder': 2}, 'ATRX': {'autistic disorder': 4, 'intellectual disability': 2}, 'BAZ2B': {}, 'BCL11A': {'autistic disorder': 6, 'intellectual disability':
2}, 'BRD4': {}, 'BTAF1': {}, 'BTRC': {}, 'CAMK2A': {'autistic disorder': 3}, 'CARD11': {}, 'CASZ1': {}, 'Catalase': {'autistic disorder': 21, 'anxiety disorder': 9, 'toxic encephalopathy': 5, 'autism spectrum disorder': 5, "Alzheimer's disease": 3}, 'CC2D1A': {'autistic
disorder': 11, 'intellectual disability': 9, 'autism spectrum disorder': 9, 'anxiety disorder': 3}, 'CHAMP1': {}, 'CHD1': {}, 'CHD2': {'autistic disorder': 4, 'intellectual disability': 4, 'autism spectrum disorder': 2, 'epilepsy': 2, 'brain disease': 2}, 'CHD3': {},
'CHD7': {'autistic disorder': 9, 'CHARGE syndrome': 5, 'autism spectrum disorder': 4, 'intellectual disability': 4}, 'CHD9': {}, 'CIC': {'autism spectrum disorder': 2, 'autistic disorder': 2}, 'CNOT3': {}, 'CREBBP': {'autistic disorder': 4, 'autism spectrum disorder': 3,
'intellectual disability': 2}, 'CSDE1': {}, 'CTCF': {'autistic disorder': 5, 'schizophrenia': 2}, 'CTNNB1': {'autistic disorder': 5, 'autism spectrum disorder': 2}, 'CUL3': {'autistic disorder': 7, 'autism spectrum disorder': 3, 'schizophrenia': 3}, 'CUX1': {'autistic
disorder': 5, 'autism spectrum disorder': 2}, 'CUX2': {'autistic disorder': 2}, 'DDX3X': {'autistic disorder': 6, 'autism spectrum disorder': 4, 'intellectual disability': 4}, 'DEAF1': {}, 'DLX2': {'autistic disorder': 3}, 'DLX3': {}, 'DLX6': {}, 'DNMT3A': {'autistic
disorder': 10, 'intellectual disability': 2, 'Rett syndrome': 2, 'autism spectrum disorder': 2}, 'DVL3': {'autistic disorder': 2}, 'EBF3': {'autistic disorder': 2, 'intellectual disability': 2}, 'EED': {}, 'EGR3': {}, 'EN2': {'autistic disorder': 42, 'autism spectrum
disorder': 19, 'intellectual disability': 5, 'schizophrenia': 4, 'epilepsy': 3, 'depressive disorder': 2}, 'EP300': {}, 'EP400': {}, 'ERBIN': {}, 'ERG': {'autistic disorder': 3, 'autism spectrum disorder': 2, 'schizophrenia': 2}, 'ESR2': {}, 'ESRRB': {}, 'EZH2': {'autistic
disorder': 4, 'autism spectrum disorder': 3}, 'FAN1': {'chromosome 15q13.3 microdeletion syndrome': 2, 'intellectual disability': 2, 'epilepsy': 2, 'autism spectrum disorder': 2, 'autistic disorder': 2, 'schizophrenia': 2}, 'FBN1': {}, 'FEZF2': {'autistic disorder': 3},
'FOXG1': {'autistic disorder': 10, 'Rett syndrome': 6, 'schizophrenia': 5, 'intellectual disability': 3, 'autism spectrum disorder': 3, 'microcephaly': 2, 'epilepsy': 2}, 'FOXP1': {'autistic disorder': 24, 'intellectual disability': 14, 'autism spectrum disorder': 11,
'intellectual disability-severe speech delay-mild dysmorphism syndrome': 5, 'anxiety disorder': 4, 'optic atrophy 1': 2}, 'FOXP2': {'autistic disorder': 32, 'autism spectrum disorder': 9, 'schizophrenia': 7, 'intellectual disability': 6, 'language disorder': 3, 'epilepsy':
3, 'attention deficit hyperactivity disorder': 2, 'apraxia': 2, 'multiple sclerosis': 2}, 'GLIS1': {}, 'GPC3': {}, 'GTF2I': {'autistic disorder': 5, 'Williams-Beuren syndrome': 4, 'autism spectrum disorder': 4}, 'HDAC4': {'intellectual disability': 3, 'autistic disorder':
3, 'chromosome 2q37 deletion syndrome': 2}, 'HIVEP2': {}, 'HIVEP3': {}, 'HMGN1': {'autistic disorder': 2}, 'JARID2': {'autistic disorder': 2}, 'KDM2A': {}, 'KDM5A': {'autistic disorder': 4, 'schizophrenia': 2}, 'KDM5B': {'autistic disorder': 2}, 'KDM5C': {'autistic
disorder': 8, 'intellectual disability': 6, 'epilepsy': 3, 'autism spectrum disorder': 2, 'schizophrenia': 2}, 'KLF16': {}, 'KLF7': {'autism spectrum disorder': 2, 'autistic disorder': 2}, 'KMT2A': {'autistic disorder': 6, 'intellectual disability': 3, 'schizophrenia': 2,
'autism spectrum disorder': 2}, 'KMT2C': {'autistic disorder': 6, 'schizophrenia': 4, 'intellectual disability': 3, 'autism spectrum disorder': 3, 'Kleefstra syndrome': 2}, 'LMX1B': {'autistic disorder': 3}, 'MBD1': {'autistic disorder': 3}, 'MBD3': {}, 'MBD4': {}, 'MBD6':
{}, 'MECP2': {'autistic disorder': 209, 'Rett syndrome': 144, 'autism spectrum disorder': 68, 'intellectual disability': 49, 'schizophrenia': 19, 'syndromic X-linked intellectual disability Lubs type': 17, 'anxiety disorder': 14, 'epilepsy': 10, 'Angelman syndrome': 8,
'brain disease': 7, 'fragile X syndrome': 7, 'microcephaly': 5, 'attention deficit hyperactivity disorder': 5, 'neuroblastoma': 4, 'Down syndrome': 3, 'Miller-Dieker lissencephaly syndrome': 3, 'Prader-Willi syndrome': 3, 'allergic disease': 2, 'depressive disorder': 2,
'pervasive developmental disorder': 2}, 'MEF2C': {'autistic disorder': 15, 'intellectual disability': 6, 'autism spectrum disorder': 5, 'epilepsy': 4, 'schizophrenia': 3}, 'MEIS2': {'autistic disorder': 2, 'autism spectrum disorder': 2}, 'MKX': {}, 'MNT': {'autistic
disorder': 2}, 'MSX2': {}, 'MTF1': {}, 'MYT1L': {'autistic disorder': 4, 'intellectual disability': 3, 'obesity': 2, 'schizophrenia': 2}, 'NCOA1': {}, 'NCOR1': {}, 'NFE2L3': {}, 'NFIA': {'autistic disorder': 4, 'autism spectrum disorder': 3, 'intellectual disability': 2},
'NFIB': {'autism spectrum disorder': 3, 'autistic disorder': 3, 'intellectual disability': 2}, 'NFIX': {'autism spectrum disorder': 2, 'autistic disorder': 2, 'intellectual disability': 2}, 'NKX2-2': {'autistic disorder': 4, 'autism spectrum disorder': 3}, 'NPAS2': {},
'NR1D1': {'autistic disorder': 4, 'autism spectrum disorder': 2, 'depressive disorder': 2, 'schizophrenia': 2}, 'NR2F1': {'autistic disorder': 6, 'autism spectrum disorder': 4, 'intellectual disability': 2}, 'NR3C2': {}, 'NR4A2': {'autistic disorder': 2}, 'NSD1': {},
'NSD2': {}, 'OTX1': {'autistic disorder': 2, 'schizophrenia': 2}, 'PAX5': {'autism spectrum disorder': 3, 'autistic disorder': 3}, 'PBX1': {}, 'PHF21A': {'autistic disorder': 3}, 'PIK3CA': {'autistic disorder': 5, 'autism spectrum disorder': 3, 'epilepsy': 2, 'intellectual
disability': 2}, 'PITX1': {}, 'PRR12': {}, 'RERE': {'intellectual disability': 2, 'autism spectrum disorder': 2, 'autistic disorder': 2}, 'RFX3': {}, 'RFX4': {'autistic disorder': 2}, 'RFX7': {'autistic disorder': 2, 'intellectual disability': 2, 'autism spectrum disorder':
2, 'attention deficit hyperactivity disorder': 2}, 'RHOXF1': {}, 'RORB': {}, 'SATB1': {}, 'SATB2': {'autistic disorder': 8, 'autism spectrum disorder': 4, 'intellectual disability': 4, 'schizophrenia': 3}, 'SETBP1': {}, 'SETD2': {'autistic disorder': 4}, 'SETDB1':
{'autistic disorder': 2}, 'SETDB2': {}, 'SHOX': {}, 'SMAD4': {}, 'SMARCC2': {}, 'SOX5': {'autistic disorder': 3, 'autism spectrum disorder': 2}, 'SRCAP': {}, 'SSRP1': {}, 'SUZ12': {}, 'TBR1': {'autistic disorder': 27, 'autism spectrum disorder': 10, 'intellectual
disability': 8, 'anxiety disorder': 3, 'schizophrenia': 2}, 'TBX22': {}, 'TCF4': {'autistic disorder': 23, 'schizophrenia': 14, 'Pitt-Hopkins syndrome': 14, 'autism spectrum disorder': 11, 'intellectual disability': 10, 'anxiety disorder': 2}, 'TCF7L2': {'autistic
disorder': 3, 'autism spectrum disorder': 2}, 'TERF2': {}, 'THRA': {}, 'TSHZ3': {'autism spectrum disorder': 5, 'autistic disorder': 5}, 'VDR': {'autistic disorder': 8, 'autism spectrum disorder': 4, 'attention deficit hyperactivity disorder': 2, 'schizophrenia': 2,
"Parkinson's disease": 2}, 'VEZF1': {}, 'WAC': {'autism spectrum disorder': 2, 'autistic disorder': 2}, 'YY1': {'autism spectrum disorder': 4, 'autistic disorder': 4}, 'ZBTB16': {}, 'ZBTB20': {}, 'ZC3H11A': {}, 'ZC3H4': {}, 'ZFYVE26': {}, 'ZMYM2': {}, 'ZNF18': {}, 'ZNF292':
{'autism spectrum disorder': 2, 'autistic disorder': 2}, 'ZNF385B': {}, 'ZNF462': {}, 'ZNF517': {}, 'ZNF548': {}, 'ZNF559': {}, 'ZNF626': {}, 'ZNF711': {}, 'ZNF713': {}, 'ZNF774': {}, 'ZNF804A': {'autistic disorder': 4, 'schizophrenia': 3}, 'ZNF827': {}}
```

Figure2: Disease Count, Main Diseases Identified, {TF:{DOID:Frequency}} Dictionary

Visualization - Heatmap (Full 9000KB heatmap that shows everything is attached in canvas submission. Google doc errors in attaching larch scale image)

To generate the heatmap, I first converted the nested dictionary containing the gene-disease associations into a pandas DataFrame and transposed it, allowing me to visualize the relationships more effectively. I then filled any missing values with 0, ensuring that the heatmap would display all values correctly.

To create a custom color mapping for the heatmap, I defined a list of colors ranging from "floralwhite" to "crimson" and their corresponding values. I then used the LinearSegmentedColormap function from the matplotlib library to create the custom colormap.

Next, I utilized the seaborn library to create the heatmap with the given DataFrame and the custom colormap. I specified the figsize parameter to determine the size of the heatmap, with larger values producing a bigger heatmap with more visible labels. The figsize parameter directly affects the number of labels displayed on the x-axis and y-axis, as well as the overall readability of the heatmap.

For the report, I used a smaller figsize value to generate a more compact heatmap that fits within the page limits. However, for the full-sized heatmap, I used a larger figsize value (100, 100) to ensure that all labels and data points were easily readable. The full-sized heatmap, generated through the provided code, was attached to the Canvas submission for a more comprehensive view of the gene-disease associations.

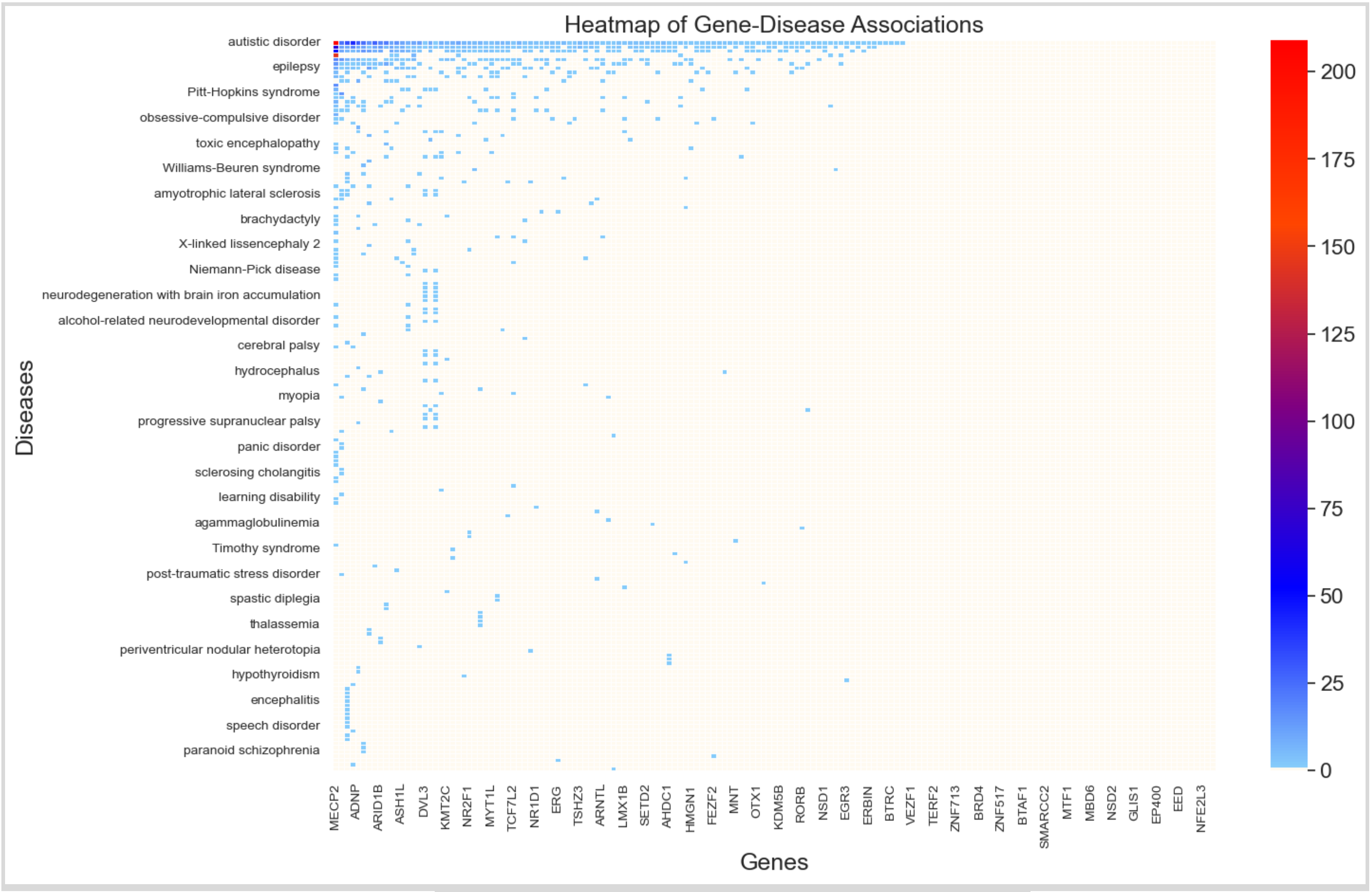


Figure3: {TF:{DOID:Frequency}} Heatmap

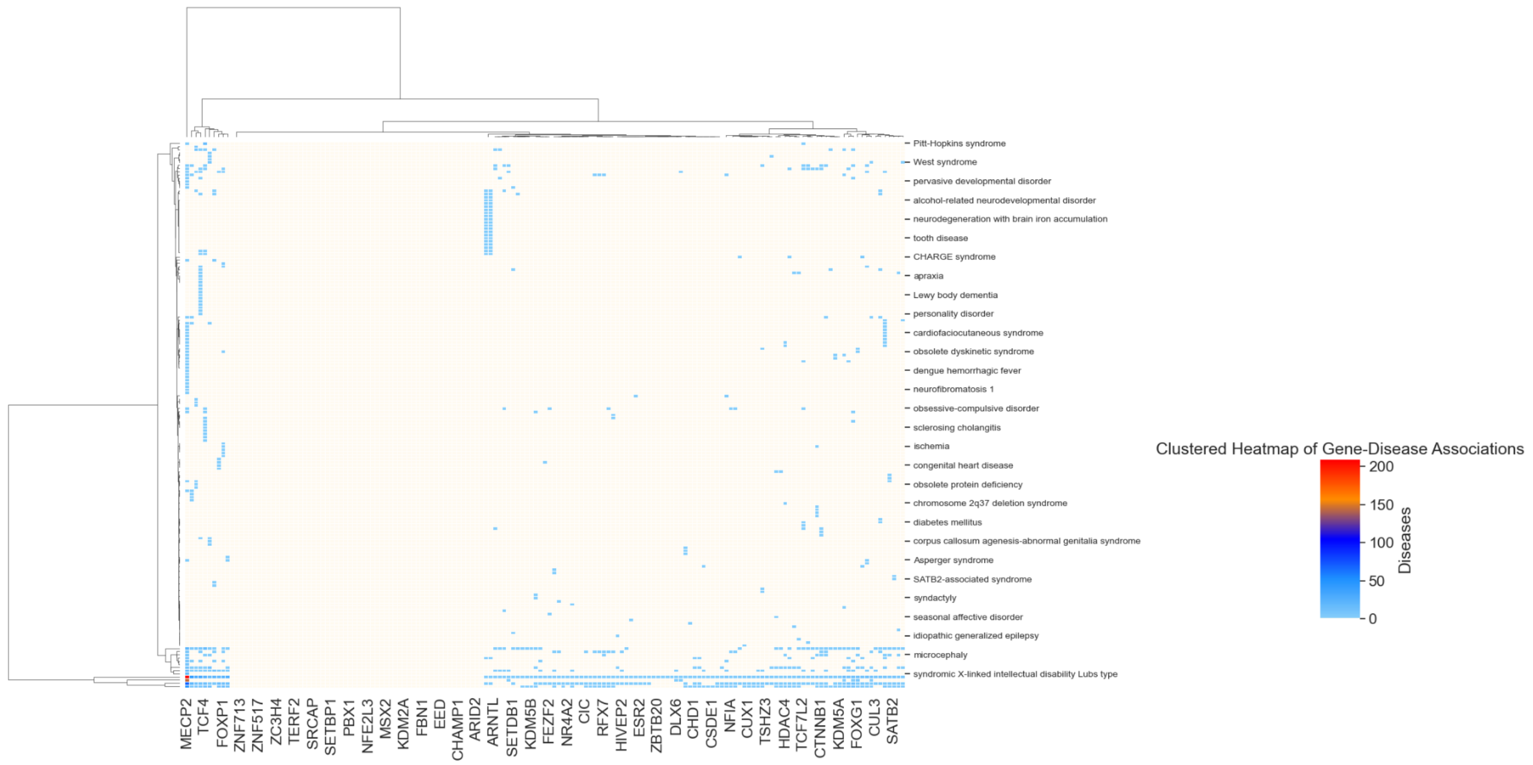
Hierarchical Clustering

To perform hierarchical clustering on the gene-disease associations, I used the seaborn library's clustermap function. This function not only generates a heatmap but also clusters the data based on similarity, displaying dendrograms for both genes and diseases.

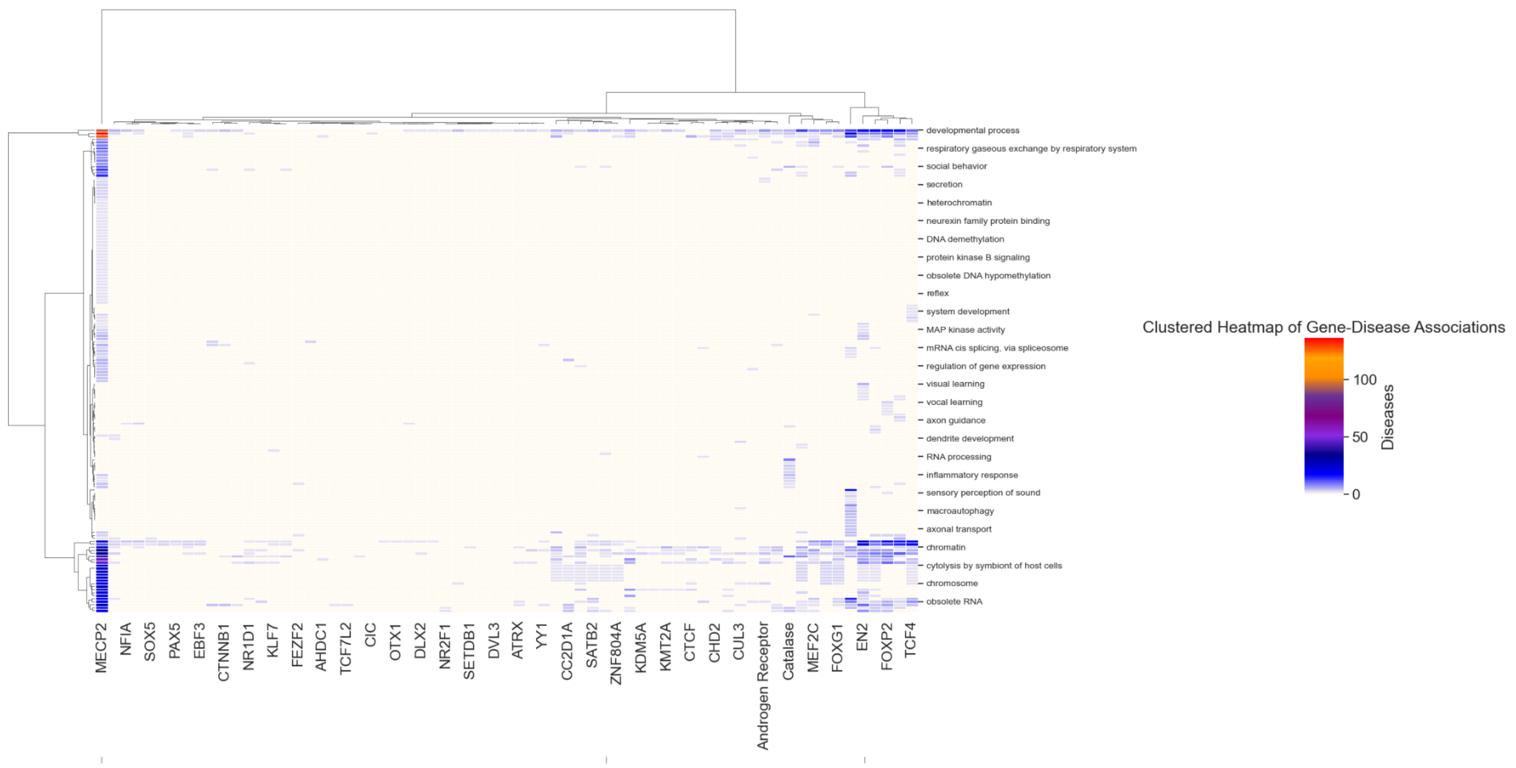
In the code provided, I first created a DataFrame from the dictionary and filled any missing values with 0. Then, I defined a custom colormap using the LinearSegmentedColormap function from the matplotlib library.

The clustermap function takes the DataFrame, the custom colormap, and various other parameters such as linewidths, figsize, and method. I set the figsize parameter to (20, 12) to create a reasonably sized heatmap that fits within the page limits. However, due to the hierarchical clustering and the figsize parameter, some labels on the x-axis and y-axis may disappear. This results in some labels being

omitted to avoid overlapping and maintain readability. To adjust the y-axis labels, I used the set_yticklabels function and set the rotation, font size, and label coordinates accordingly.
The clustered heatmap effectively visualizes the gene-disease associations while grouping similar genes and diseases together. This enables a more intuitive understanding of the relationships between genes and diseases, which can be beneficial for further analysis and modeling.



.



Figurer4: {TF:{DOID:Frequency}}, {TF:{GO:Frequency}} Hierarchical Clustering

Additionally, for further exploration, I attached the hierarchical clustering result generated for Gene ontology association with genes.

Discussion | Results

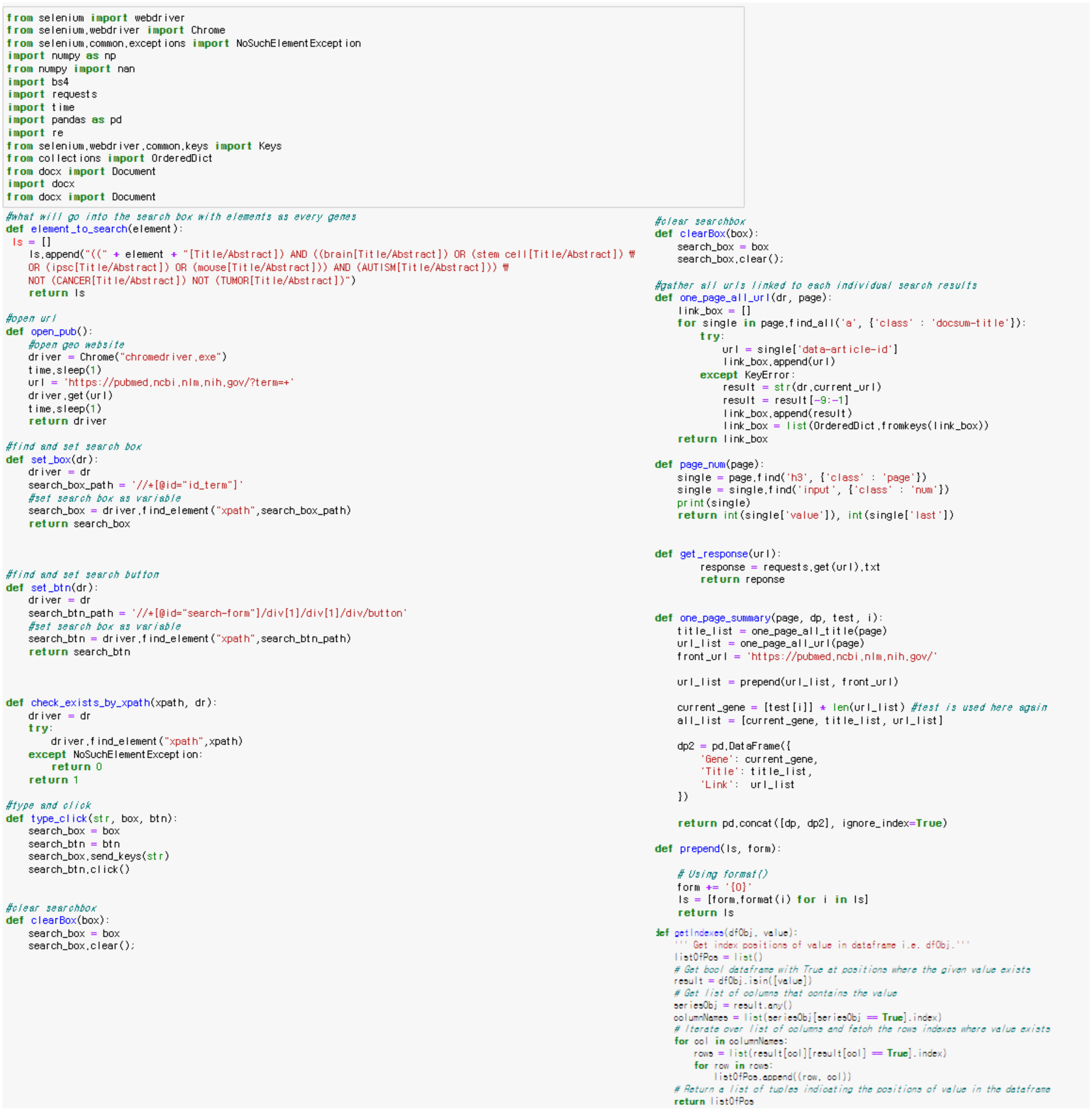
In this study, I aimed to explore the associations between transcription factors and Autism Spectrum Disorder (ASD) by employing Natural Language Processing (NLP) techniques and machine learning models. Despite the limitation, our research yielded valuable insights and demonstrated the potential of using NLP and machine learning methods to analyze biomedical literature.

Though the validity can be questioned, our findings from the heatmap and hierarchical clustering visualizations showed that certain transcription factors shared common disease associations, suggesting a possible functional relationship between them. Moreover, the hierarchical clustering revealed groups of transcription factors that are potentially involved in similar biological processes, indicating a shared role in ASD development.

In addition to refining the dataset, future research should explore the use of more advanced machine learning techniques, such as deep learning, to enhance the predictive capabilities of the models. Integrating domain knowledge with these models could help uncover novel gene-disease associations and provide a deeper understanding of the underlying molecular mechanisms involved in ASD development.

In conclusion, our study demonstrates the potential of using NLP techniques and machine learning models to analyze biomedical literature and generate predictive models for gene-disease associations. By refining the dataset and incorporating more advanced techniques, this approach can be further developed to uncover novel insights into the complex relationships between genes, diseases, and their potential functional roles in the development of ASD and other disorders.

Code

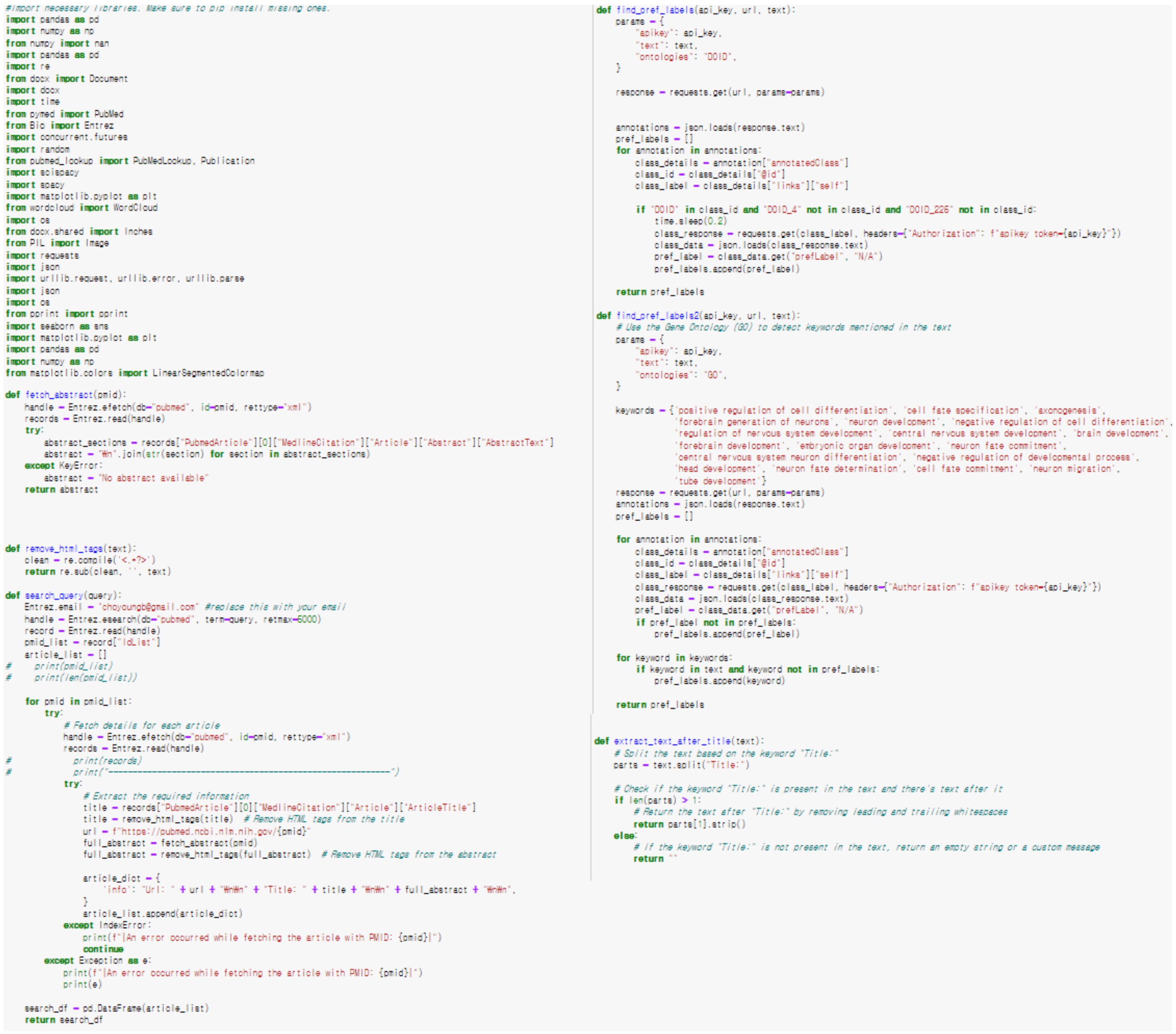


```python
from selenium import webdriver
from selenium.webdriver import Chrome
from selenium.common.exceptions import NoSuchElementException
import numpy as np
from numpy import nan
import bs4
import requests
import time
import pandas as pd
import re
from selenium.webdriver.common.keys import Keys
from collections import OrderedDict
from docx import Document
import docx
from docx import Document
```

```python
#what will go into the search box with elements as every genes
def element_to_search(element):
 ls = []
    ls.append("((" + element + "[Title/Abstract]) AND ((brain[Title/Abstract]) OR (stem cell[Title/Abstract]) ₩
    OR (ipsc[Title/Abstract]) OR (mouse[Title/Abstract])) AND (AUTISM[Title/Abstract])) ₩
    NOT (CANCER[Title/Abstract]) NOT (TUMOR[Title/Abstract])")
    return ls

#open url
def open_pub():
    #open geo website
    driver = Chrome("chromedriver.exe")
    time.sleep(1)
    url = 'https://pubmed.ncbi.nlm.nih.gov/?term=+'
    driver.get(url)
    time.sleep(1)
    return driver

#find and set search box
def set_box(dr):
    driver = dr
    search_box_path = '//*[@id="id_term"]'
    #set search box as variable
    search_box = driver.find_element("xpath",search_box_path)
    return search_box



#find and set search button
def set_btn(dr):
    driver = dr
    search_btn_path = '//*[@id="search-form"]/div[1]/div[1]/div/button'
    #set search box as variable
    search_btn = driver.find_element("xpath",search_btn_path)
    return search_btn



def check_exists_by_xpath(xpath, dr):
    driver = dr
    try:
        driver.find_element("xpath",xpath)
    except NoSuchElementException:
        return 0
    return 1

#type and click
def type_click(str, box, btn):
    search_box = box
    search_btn = btn
    search_box.send_keys(str)
    search_btn.click()


#clear searchbox
def clearBox(box):
    search_box = box
    search_box.clear();
```

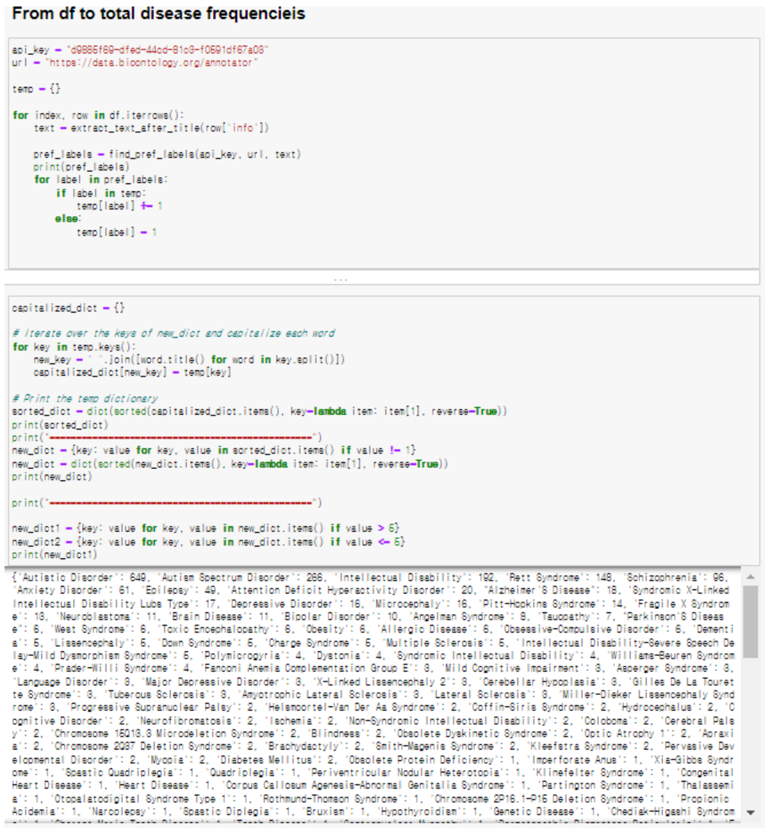
```python
#clear searchbox
def clearBox(box):
    search_box = box
    search_box.clear();

#gather all urls linked to each individual search results
def one_page_all_url(dr, page):
    link_box = []
    for single in page.find_all('a', {'class' : 'docsum-title'}):
        try:
            url = single['data-article-id']
            link_box.append(url)
        except KeyError:
            result = str(dr.current_url)
            result = result[-9:-1]
            link_box.append(result)
            link_box = list(OrderedDict.fromkeys(link_box))
    return link_box

def page_num(page):
    single = page.find('h3', {'class' : 'page'})
    single = single.find('input', {'class' : 'num'})
    print(single)
    return int(single['value']), int(single['last'])


def get_response(url):
        response = requests.get(url).txt
        return reponse


def one_page_summary(page, dp, test, i):
    title_list = one_page_all_title(page)
    url_list = one_page_all_url(page)
    front_url = 'https://pubmed.ncbi.nlm.nih.gov/'

    url_list = prepend(url_list, front_url)

    current_gene = [test[i]] * len(url_list) #test is used here again
    all_list = [current_gene, title_list, url_list]

    dp2 = pd.DataFrame({
        'Gene': current_gene,
        'Title': title_list,
        'Link':  url_list
    })

    return pd.concat([dp, dp2], ignore_index=True)

def prepend(ls, form):

    # Using format()
    form += '{0}'
    ls = [form.format(i) for i in ls]
    return ls

def getIndexes(dfObj, value):
    ''' Get index positions of value in dataframe i.e. dfObj.'''
    listOfPos = list()
    # Get bool dataframe with True at positions where the given value exists
    result = dfObj.isin([value])
    # Get list of columns that contains the value
    seriesObj = result.any()
    columnNames = list(seriesObj[seriesObj == True].index)
    # Iterate over list of columns and fetch the rows indexes where value exists
    for col in columnNames:
        rows = list(result[col][result[col] == True].index)
        for row in rows:
            listOfPos.append((row, col))
    # Return a list of tuples indicating the positions of value in the dataframe
    return listOfPos
```

```python
tf_dict = {}
docls = pd.DataFrame(columns = ['TF'])

#test for searches
test = tf  #  [tf[5], tf[6]] is a 'RFX4', 'RFX7'
link_box = []
#open pub website
driver = open_pub()
url = 'https://pubmed.ncbi.nlm.nih.gov/?term=+'
doc = Document()
doc.save('test.docx')
links = []
# open an existing document
doc = docx.Document('./test.docx')

for i in range(0, len(test)):
    box = set_box(driver)
    btn = set_btn(driver)
    time.sleep(1.5)
    ls = element_to_search(test[i])
    link_box = []


    #search using permutations derived from element_to search for ls
    for j in range(0, len(ls)):
        type_click(ls[j], box, btn)
        time.sleep(2)
        src = driver.page_source
        page = bs4.BeautifulSoup(src)

        link_box.append(one_page_all_url(driver, page))
        driver.get(url)
        box = set_box(driver)
        btn = set_btn(driver)

    # Deletes empty list inside link_box
    link_box = [x for x in link_box if x]

    # for every list in link box
    for ls in link_box:
        #for every link in lists of linkbox
        for n in ls:
            current = "https://pubmed.ncbi.nlm.nih.gov/" + str(n)

            #if entering current url is in the record (tf_dict)
            #if value is in tf_dict, skip
            #if value is not in tf_dict, add tf name to the tf_dict as a value to the url key
            #Then, update docls so that tf string is appended with a new
            if(current in tf_dict) == True:

                #URL history exists and the same TF exists -> duplicated link & TF
                if(str(test[i]) in tf_dict[current]) == True:
                    continue

                #URL history exists but new TF -> add TF name to history and to the docls
                else:
                    tf_dict[current].append(str(test[i]))
                    tf_temp = docls[docls['TF'].str.contains(current)].index.values[0]
                    docls['TF'][tf_temp-1] = docls['TF'][tf_temp-1] + ", " + str(test[i])



            #if entering current url is not in the record
            #leave the record of access to the tf_dict
            #update the docls, tfs and url+abstract
            else:
                #temp list with length 2 to save current tf at [0] and abstract at [1]
                temp = [test[i]]
                #append new url to tf_dict history
                tf_dict[current] = [str(test[i])]


                #update temp, then append to docls
                driver.get(current)
                time.sleep(2)
                src = driver.page_source
                page = bs4.BeautifulSoup(src)

                try:
                    abstract = page.find('div', {'class' : 'abstract-content selected'}).text
                    title = page.find('h1', {'class' : 'heading-title'}).text
                    toadd = "Title- " + title  + "URL- " + current +  abstract
                    temp.append(toadd)

                except AttributeError:
                    title = page.find('h1', {'class' : 'heading-title'}).text
                    title = title.replace('\n', '')
                    title = title.replace('  ', '')
                    toadd = "Title- " + title + "URL- " + current
                    temp.append(toadd)

                #docls (dataframe) updated
                docls = docls.append(pd.DataFrame(temp, columns=["TF"]), ignore_index=True)


# By now, docls is complete in the order of tf list - abstract - tflist - (...)

# First row are table headers!
table = doc.add_table(rows=(docls.shape[0]), cols=docls.shape[1])

for i, column in enumerate(docls) :
    for row in range(docls.shape[0]) :
        table.cell(row, i).text = str(docls[column][row])

doc.save('./test.docx')
```



```python
#import necessary libraries. Make sure to pip install missing ones.
import pandas as pd
import numpy as np
from numpy import nan
import pandas as pd
import re
from docx import Document
import docx
import time
from pymed import PubMed
from Bio import Entrez
import concurrent.futures
import random
from pubmed_lookup import PubMedLookup, Publication
import scispacy
import spacy
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import os
from docx.shared import Inches
from PIL import Image
import requests
import json
import urllib.request, urllib.error, urllib.parse
import json
import os
from pprint import pprint
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from matplotlib.colors import LinearSegmentedColormap

def fetch_abstract(pmid):
    handle = Entrez.efetch(db="pubmed", id=pmid, rettype="xml")
    records = Entrez.read(handle)
    try:
        abstract_sections = records["PubmedArticle"][0]["MedlineCitation"]["Article"]["Abstract"]["AbstractText"]
        abstract = "₩n".join(str(section) for section in abstract_sections)
    except KeyError:
        abstract = "No abstract available"
    return abstract



def remove_html_tags(text):
    clean = re.compile('<.*?>')
    return re.sub(clean, '', text)

def search_query(query):
    Entrez.email = "choyoungb@gmail.com" #replace this with your email
    handle = Entrez.esearch(db="pubmed", term=query, retmax=5000)
    record = Entrez.read(handle)
    pmid_list = record["IdList"]
    article_list = []
#    print(pmid_list)
#    print(len(pmid_list))

    for pmid in pmid_list:
        try:
            # Fetch details for each article
            handle = Entrez.efetch(db="pubmed", id=pmid, rettype="xml")
            records = Entrez.read(handle)
#             print(records)
#             print("----------------------------------------------------------")
            try:
                # Extract the required information
                title = records["PubmedArticle"][0]["MedlineCitation"]["Article"]["ArticleTitle"]
                title = remove_html_tags(title)  # Remove HTML tags from the title
                url = f"https://pubmed.ncbi.nlm.nih.gov/{pmid}"
                full_abstract = fetch_abstract(pmid)
                full_abstract = remove_html_tags(full_abstract)  # Remove HTML tags from the abstract

                article_dict = {
                    'info': "Url: " + url + "₩n₩n" + "Title: " + title + "₩n₩n" + full_abstract + "₩n₩n",
                }
                article_list.append(article_dict)
            except IndexError:
                print(f"|An error occurred while fetching the article with PMID: {pmid}|")
                continue
        except Exception as e:
            print(f"|An error occurred while fetching the article with PMID: {pmid}|")
            print(e)

    search_df = pd.DataFrame(article_list)
    return search_df
```

```python
def find_pref_labels(api_key, url, text):
    params = {
        "apikey": api_key,
        "text": text,
        "ontologies": "DOID",
    }

    response = requests.get(url, params=params)


    annotations = json.loads(response.text)
    pref_labels = []
    for annotation in annotations:
        class_details = annotation["annotatedClass"]
        class_id = class_details["@id"]
        class_label = class_details["links"]["self"]

        if "DOID" in class_id and "DOID_4" not in class_id and "DOID_225" not in class_id:
            time.sleep(0.2)
            class_response = requests.get(class_label, headers={"Authorization": f"apikey token={api_key}"})
            class_data = json.loads(class_response.text)
            pref_label = class_data.get("prefLabel", "N/A")
            pref_labels.append(pref_label)

    return pref_labels

def find_pref_labels2(api_key, url, text):
    # Use the Gene Ontology (GO) to detect keywords mentioned in the text
    params = {
        "apikey": api_key,
        "text": text,
        "ontologies": "GO",
    }

    keywords = {'positive regulation of cell differentiation', 'cell fate specification', 'axonogenesis',
                'forebrain generation of neurons', 'neuron development', 'negative regulation of cell differentiation',
                'regulation of nervous system development', 'central nervous system development', 'brain development',
                'forebrain development', 'embryonic organ development', 'neuron fate commitment',
                'central nervous system neuron differentiation', 'negative regulation of developmental process',
                'head development', 'neuron fate determination', 'cell fate commitment', 'neuron migration',
                'tube development'}
    response = requests.get(url, params=params)
    annotations = json.loads(response.text)
    pref_labels = []

    for annotation in annotations:
        class_details = annotation["annotatedClass"]
        class_id = class_details["@id"]
        class_label = class_details["links"]["self"]
        class_response = requests.get(class_label, headers={"Authorization": f"apikey token={api_key}"})
        class_data = json.loads(class_response.text)
        pref_label = class_data.get("prefLabel", "N/A")
        if pref_label not in pref_labels:
            pref_labels.append(pref_label)

    for keyword in keywords:
        if keyword in text and keyword not in pref_labels:
            pref_labels.append(keyword)

    return pref_labels


def extract_text_after_title(text):
    # Split the text based on the keyword "Title:"
    parts = text.split("Title:")

    # Check if the keyword "Title:" is present in the text and there's text after it
    if len(parts) > 1:
        # Return the text after "Title:" by removing leading and trailing whitespaces
        return parts[1].strip()
    else:
        # If the keyword "Title:" is not present in the text, return an empty string or a custom message
        return ""
```

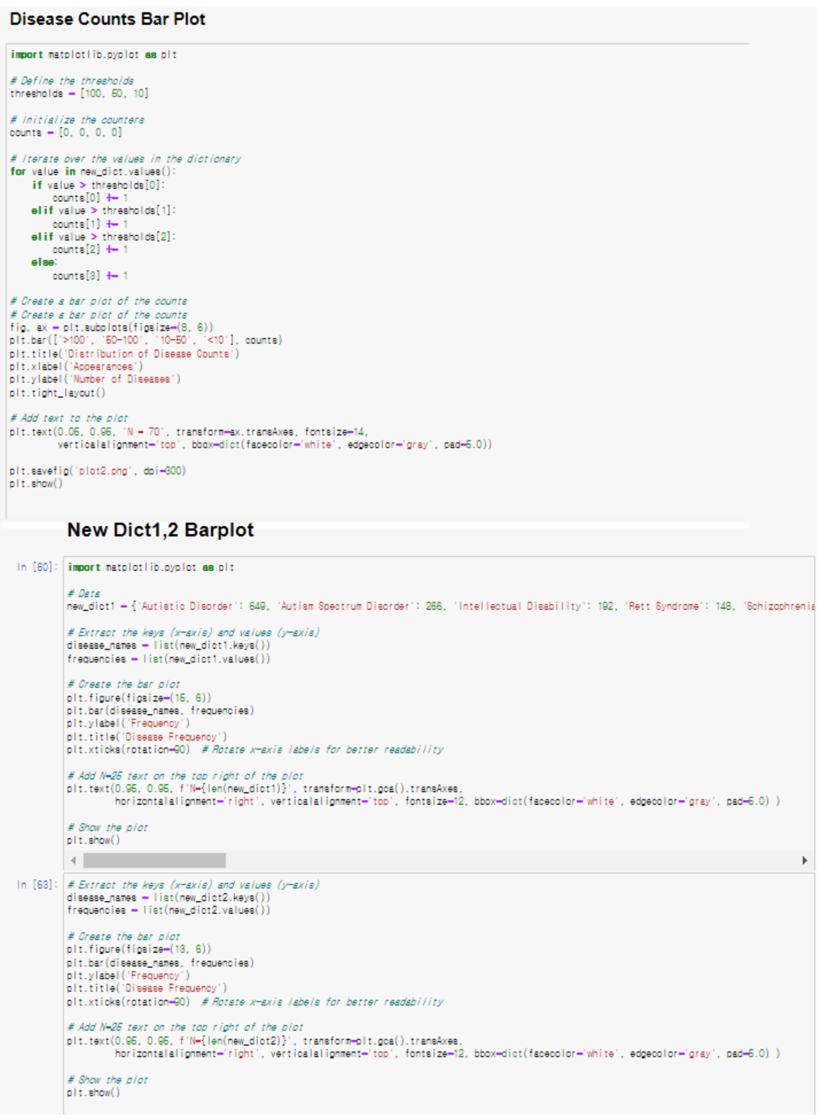


From df to total disease frequencieis

```python
api_key = "d9886f69-dfed-44cd-81c8-f0591df67a08"
url = "https://data.bioontology.org/annotator"

temp = {}

for index, row in df.iterrows():
    text = extract_text_after_title(row['info'])

    pref_labels = find_pref_labels(api_key, url, text)
    print(pref_labels)
    for label in pref_labels:
        if label in temp:
            temp[label] += 1
        else:
            temp[label] = 1
```

```python
capitalized_dict = {}

# Iterate over the keys of new_dict and capitalize each word
for key in temp.keys():
    new_key = ' '.join([word.title() for word in key.split()])
    capitalized_dict[new_key] = temp[key]

# Print the temp dictionary
sorted_dict = dict(sorted(capitalized_dict.items(), key=lambda item: item[1], reverse=True))
print(sorted_dict)
print("------------------------------------------------")
new_dict = {key: value for key, value in sorted_dict.items() if value != 1}
new_dict = dict(sorted(new_dict.items(), key=lambda item: item[1], reverse=True))
print(new_dict)

print("------------------------------------------------")

new_dict1 = {key: value for key, value in new_dict.items() if value > 5}
new_dict2 = {key: value for key, value in new_dict.items() if value <= 5}
print(new_dict1)
```

```
{'Autistic Disorder': 649, 'Autism Spectrum Disorder': 266, 'Intellectual Disability': 192, 'Rett Syndrome': 148, 'Schizophrenia': 96,
'Anxiety Disorder': 61, 'Epilepsy': 49, 'Attention Deficit Hyperactivity Disorder': 20, "Alzheimer'S Disease": 18, 'Syndromic X-Linked
Intellectual Disability Lubs Type': 17, 'Depressive Disorder': 16, 'Microcephaly': 16, 'Pitt-Hopkins Syndrome': 14, 'Fragile X Syndrom
e': 13, 'Neuroblastoma': 11, 'Brain Disease': 11, 'Bipolar Disorder': 10, 'Angelman Syndrome': 8, 'Tauopathy': 7, "Parkinson'S Diseas
e": 6, 'West Syndrome': 6, 'Toxic Encephalopathy': 6, 'Obesity': 6, 'Allergic Disease': 6, 'Obsessive-Compulsive Disorder': 6, 'Dementi
a': 5, 'Lissencephaly': 5, 'Down Syndrome': 5, 'Charge Syndrome': 5, 'Multiple Sclerosis': 5, 'Intellectual Disability-Severe Speech De
lay-Mild Dysmorphism Syndrome': 5, 'Polymicrogyria': 4, 'Dystonia': 4, 'Syndromic Intellectual Disability': 4, 'Williams-Beuren Syndrom
e': 4, 'Prader-Willi Syndrome': 4, 'Fanconi Anemia Complementation Group E': 3, 'Mild Cognitive Impairment': 3, 'Asperger Syndrome': 3,
'Language Disorder': 3, 'Major Depressive Disorder': 3, 'X-Linked Lissencephaly 2': 3, 'Cerebellar Hypoplasia': 3, 'Gilles De La Touret
te Syndrome': 3, 'Tuberous Sclerosis': 3, 'Amyotrophic Lateral Sclerosis': 3, 'Lateral Sclerosis': 3, 'Miller-Dieker Lissencephaly Synd
rome': 3, 'Progressive Supranuclear Palsy': 2, 'Helsmoortel-Van Der Aa Syndrome': 2, 'Coffin-Siris Syndrome': 2, 'Hydrocephalus': 2, 'C
ognitive Disorder': 2, 'Neurofibromatosis': 2, 'Ischemia': 2, 'Non-Syndromic Intellectual Disability': 2, 'Coloboma': 2, 'Cerebral Pals
y': 2, 'Chromosome 15Q13.3 Microdeletion Syndrome': 2, 'Blindness': 2, 'Obsolete Dyskinetic Syndrome': 2, 'Optic Atrophy 1': 2, 'Apraxi
a': 2, 'Chromosome 2Q37 Deletion Syndrome': 2, 'Brachydactyly': 2, 'Smith-Magenis Syndrome': 2, 'Kleefstra Syndrome': 2, 'Pervasive Dev
elopmental Disorder': 2, 'Myopia': 2, 'Diabetes Mellitus': 2, 'Obsolete Protein Deficiency': 1, 'Imperforate Anus': 1, 'Xia-Gibbs Syndr
ome': 1, 'Spastic Quadriplegia': 1, 'Quadriplegia': 1, 'Periventricular Nodular Heterotopia': 1, 'Klinefelter Syndrome': 1, 'Congenital
Heart Disease': 1, 'Heart Disease': 1, 'Corpus Callosum Agenesis-Abnormal Genitalia Syndrome': 1, 'Partington Syndrome': 1, 'Thalassemi
a': 1, 'Otopalatodigital Syndrome Type 1': 1, 'Rothmund-Thomson Syndrome': 1, 'Chromosome 2P16.1-P15 Deletion Syndrome': 1, 'Propionic
Acidemia': 1, 'Narcolepsy': 1, 'Spastic Diplegia': 1, 'Bruxism': 1, 'Hypothyroidism': 1, 'Genetic Disease': 1, 'Chediak-Higashi Syndrom
```



Disease Counts Bar Plot

```python
import matplotlib.pyplot as plt

# Define the thresholds
thresholds = [100, 50, 10]

# Initialize the counters
counts = [0, 0, 0, 0]

# Iterate over the values in the dictionary
for value in new_dict.values():
    if value > thresholds[0]:
        counts[0] += 1
    elif value > thresholds[1]:
        counts[1] += 1
    elif value > thresholds[2]:
        counts[2] += 1
    else:
        counts[3] += 1

# Create a bar plot of the counts
# Create a bar plot of the counts
fig, ax = plt.subplots(figsize=(8, 6))
plt.bar(['>100', '50-100', '10-50', '<10'], counts)
plt.title('Distribution of Disease Counts')
plt.xlabel('Appearances')
plt.ylabel('Number of Diseases')
plt.tight_layout()

# Add text to the plot
plt.text(0.05, 0.95, 'N = 70', transform=ax.transAxes, fontsize=14,
        verticalalignment='top', bbox=dict(facecolor='white', edgecolor='gray', pad=5.0))

plt.savefig('plot2.png', dpi=300)
plt.show()
```

New Dict1,2 Barplot

```python
In [60]: import matplotlib.pyplot as plt

         # Data
         new_dict1 = {'Autistic Disorder': 649, 'Autism Spectrum Disorder': 266, 'Intellectual Disability': 192, 'Rett Syndrome': 148, 'Schizophrenia

         # Extract the keys (x-axis) and values (y-axis)
         disease_names = list(new_dict1.keys())
         frequencies = list(new_dict1.values())

         # Create the bar plot
         plt.figure(figsize=(15, 6))
         plt.bar(disease_names, frequencies)
         plt.ylabel('Frequency')
         plt.title('Disease Frequency')
         plt.xticks(rotation=90)  # Rotate x-axis labels for better readability

         # Add N=25 text on the top right of the plot
         plt.text(0.95, 0.95, f'N={len(new_dict1)}', transform=plt.gca().transAxes,
                 horizontalalignment='right', verticalalignment='top', fontsize=12, bbox=dict(facecolor='white', edgecolor='gray', pad=5.0) )

         # Show the plot
         plt.show()
```

```python
In [63]: # Extract the keys (x-axis) and values (y-axis)
         disease_names = list(new_dict2.keys())
         frequencies = list(new_dict2.values())

         # Create the bar plot
         plt.figure(figsize=(13, 6))
         plt.bar(disease_names, frequencies)
         plt.ylabel('Frequency')
         plt.title('Disease Frequency')
         plt.xticks(rotation=90)  # Rotate x-axis labels for better readability

         # Add N=25 text on the top right of the plot
         plt.text(0.95, 0.95, f'N={len(new_dict2)}', transform=plt.gca().transAxes,
                 horizontalalignment='right', verticalalignment='top', fontsize=12, bbox=dict(facecolor='white', edgecolor='gray', pad=5.0) )

         # Show the plot
         plt.show()
```

```python
disease_Gene_df = pd.DataFrame(sorted_disease_Gene)
disease_Gene_df.fillna(0, inplace=True)

cmap_list = [
    (0, "floralwhite"),
    (1e-6, "lightskyblue"),
    (0.25, "blue"),
    (0.5, "purple"),
    (0.75, "orangered"),
    (1, "red"),
]

cmap_custom = LinearSegmentedColormap.from_list("custom", cmap_list)

# Sort rows and columns by their sum
sorted_columns = disease_Gene_df.sum(axis=0).sort_values(ascending=False).index
sorted_rows = disease_Gene_df.sum(axis=1).sort_values(ascending=False).index
disease_Gene_df_sorted = disease_Gene_df.loc[sorted_rows, sorted_columns]

# Plot the heatmap
plt.figure(figsize=(15,10))
ax = sns.heatmap(
    disease_Gene_df_sorted,
    cmap=cmap_custom,
    linewidths=0.5,
    annot=False,
    cbar=True,
    norm=plt.Normalize(vmin=0, vmax=np.max(disease_Gene_df_sorted.values))
)

ax.set_yticklabels(ax.get_yticklabels(), rotation=0, fontsize=10)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90, fontsize=10)

plt.xlabel("Genes")
plt.ylabel("Diseases")
plt.title("Heatmap of Gene-Disease Associations")
plt.show()
```

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Merge ontology_Gene and disease_Gene dictionaries based on gene names
data = {gene: {**ontology_Gene.get(gene, {}), **disease_Gene.get(gene, {})}
        for gene in set(ontology_Gene.keys()) | set(disease_Gene.keys())}

# Convert data to pandas DataFrame and fill missing values with zeros
df = pd.DataFrame.from_dict(data, orient='index').fillna(0)

# Remove target variable from the feature set
df_features = df.drop(columns=['autism spectrum disorder'])

# Standardize the data
scaler = StandardScaler()
X = scaler.fit_transform(df_features.values)

# Define target variable
y = np.array(df['autism spectrum disorder'] > 0, dtype=int)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameters for logistic regression model
param_grid = {'C': np.logspace(-3, 3, 7)}

# Perform nested cross-validation to find best hyperparameters and evaluate model performance
clf = LogisticRegression(solver='lbfgs', max_iter=1000, random_state=42)
grid_search = GridSearchCV(clf, param_grid=param_grid, cv=5)
nested_score = cross_val_score(grid_search, X=X_train, y=y_train, cv=5)
grid_search.fit(X_train, y_train)
test_score = grid_search.score(X_test, y_test)

# Plot nested cross-validation and test set scores for each hyperparameter
results = grid_search.cv_results_
plt.figure(figsize=(12, 6))
plt.title("GridSearchCV Results")
plt.xlabel("Hyperparameter C")
plt.ylabel("Score")
plt.xscale("log")
plt.plot(param_grid['C'], results['mean_test_score'], label='Nested CV Scores')
plt.plot(param_grid['C'], [test_score] * len(param_grid['C']), label='Test Set Score')
plt.legend()
plt.show()

# Plot feature coefficients for best model
fig, ax = plt.subplots(figsize=(40, 5))
coef = grid_search.best_estimator_.coef_[0]
features = df_features.columns
plt.bar(features, coef)
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Coefficient')
plt.show()

print("Best hyperparameters: ", grid_search.best_params_)
print("Test set score: ", test_score)
print("Best cross-validation score: {:.2f} (std = {:.2f})".format(
    grid_search.best_score_, results['std_test_score'][grid_search.best_index_]))
cv_results_df = pd.DataFrame.from_dict(results)
print(cv_results_df[['params', 'mean_test_score', 'std_test_score']])
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from matplotlib.colors import LinearSegmentedColormap

disease_Gene_df = pd.DataFrame(sorted_disease_Gene)
disease_Gene_df.fillna(0, inplace=True)

cmap_list = [
    (0, "floralwhite"),
    (1e-6, "lightskyblue"),
    (0.25, "dodgerblue"),
    (0.5, "blue"),
    (0.75, "darkorange"),
    (1, "red"),
]

cmap_custom = LinearSegmentedColormap.from_list("custom", cmap_list)

g = sns.clustermap(
    disease_Gene_df,
    cmap=cmap_custom,
    linewidths=0.5,
    annot=False,
    cbar=True,
    figsize=(100,100),
    norm=plt.Normalize(vmin=0, vmax=np.max(disease_Gene_df.values)),
#     dendrogram_ratio=(.1, .2),
    cbar_pos=(1.01, .2, .03, .2),
    method='ward'
)

# Adjust the y-axis labels
g.ax_heatmap.set_yticklabels(g.ax_heatmap.get_yticklabels(), rotation=0, fontsize=10)
g.ax_heatmap.yaxis.set_label_coords(-0.15, 0.5)

plt.xlabel("Genes")
plt.ylabel("Diseases")
plt.title("Clustered Heatmap of Gene-Disease Associations")
plt.show()
```