QUESTION 1 - Entrez API

Tahir Manuel D Mello

BIS634 Assignment 3

In [1]:
```
import requests
import json
from urllib.request import urlopen
import xmltodict
import time
import xml.etree.ElementTree as ET
```

Use the requests module (or urllib) to use the Entrez API to identify the PubMed IDs for 1000 Alzheimers papers from 2022 and for 1000 cancer papers from 2022. **(9 points)**

In [2]:
```
def pubmed_ids(term, nresults):

    db = 'pubmed'
    domain = 'https://www.ncbi.nlm.nih.gov/entrez/eutils'
    queryterm = term + "+AND+2022[pdat]"
    retmode='json'
    rettype = 'abstract'

    query = f'{domain}/esearch.fcgi?db={db}&retmax={nresults}&retmode={retmode}&term={

    response = requests.get(query)
    time.sleep(1)
    pubmed_id_list = response.json()

    return pubmed_id_list
```

In [ ]:
```
alzheimers_ids = pubmed_ids("Alzheimers", 1000)
alzheimers_ids
```

In [ ]:
```
cancer_ids = pubmed_ids("cancer", 1000)
cancer_ids
```

Use the Entrez API via requests/urllib to pull the metadata for each such paper found above (both cancer and Alzheimers) (and save a JSON file storing each paper's title, abstract, and the query that found it. **(12 points)**

In [5]:
```
def pubmed_metadata(term, pubmed_ids, nresults):

    db = 'pubmed'
    domain = 'https://www.ncbi.nlm.nih.gov/entrez/eutils'
    rettype = 'abstract'
    retmode='xml'

    results = {}

    for r in range(0, nresults, 100):

        paperId = pubmed_ids["esearchresult"]["idlist"][0+r:100+r]
```

```
        query = f'{domain}/efetch.fcgi?db={db}&id={paperId}&rettype={rettype}&retmode=

        response = requests.post(query)
        time.sleep(1)

        filename = 'metadata' + term + '.xml'

        with open(filename, 'wb') as f:
                f.write(response.content)

        metadata = ET.parse(filename)
        root = metadata.getroot()

        count = 0

        for id_list in paperId:

            titles = []

            for title in root[count].iter('ArticleTitle'):
                titles.append(ET.tostring(title, method="text").decode())

            abstracts = []

            for item in root[count].iter('Abstract'):
                abstracts.append(ET.tostringlist(item, method="xml"))
            #print(abstracts)

            results[id_list] = {'ArticleTitle': titles[0],
                                'AbstractText': abstracts,
                                'query': term}

            count = count + 1


        #print(count)

    return results
```

In [ ]:
```
alzheimer_metadata =  pubmed_metadata("Alzheimers", alzheimers_ids, 1000)
alzheimer_metadata
```

In [7]:
```
len(alzheimer_metadata)
```

Out[7]:
```
1000
```

In [ ]:
```
cancer_metadata =  pubmed_metadata("cancer", cancer_ids, 1000)
cancer_metadata
```

In [9]:
```
len(cancer_metadata)
```

Out[9]:
```
1000
```

In [10]:
```
import pickle
#Also saved as a pkl binary file for convenience
f1 = open("dictionary_Alzheimers.pkl","wb")
```

```
pickle.dump(alzheimer_metadata,f1)
f1.close()

f2 = open("dictionary_cancer.pkl","wb")
pickle.dump(cancer_metadata,f2)
f2.close()
```

In [11]:

```
#I am using pkl files for convenience for this assignment but since the questions spec
#this is how it can be done. The "AbstractText" needs to be converted from bytes to st
#I had used this section in Q2 originally to preprocess my pkl dictionary there but ho
#do a JSON save and (hopefully) not lose any silly marks.

alzheimers_paper_id = alzheimers_ids["esearchresult"]["idlist"]
alzheimers_dict = alzheimer_metadata

for id_list in alzheimers_paper_id:

    if alzheimers_dict[str(id_list)]['AbstractText'] == [] or len(alzheimers_dict[str(
        alzheimers_dict[id_list]['AbstractText'] = ""
        continue

    item = alzheimers_dict[str(id_list)]['AbstractText'][0][0] #Choose saved xml formo
    tree = ET.ElementTree(ET.fromstring(item))
    root = tree.getroot()

    for child in root:
        if child.tag == 'CopyrightInformation':
            root.remove(child)

    alzheimers_dict[id_list]['AbstractText'] = ET.tostring(root, method="text").decode


cancer_paper_id = cancer_ids["esearchresult"]["idlist"]
cancer_dict = cancer_metadata

for id_list in cancer_paper_id:

    if cancer_dict[str(id_list)]['AbstractText'] == [] or len(cancer_dict[str(id_list)
        cancer_dict[id_list]['AbstractText'] = ""
        continue

    item = cancer_dict[str(id_list)]['AbstractText'][0][0] #Choose saved xml format st
    tree = ET.ElementTree(ET.fromstring(item))
    root = tree.getroot()

    for child in root:
        if child.tag == 'CopyrightInformation':
            root.remove(child)

    cancer_dict[id_list]['AbstractText'] = ET.tostring(root, method="text").decode()




with open("dictionary_Alzheimers.json", "w") as outfile1:
    json.dump(alzheimers_dict, outfile1)

with open("dictionary_cancer.json", "w") as outfile2:
    json.dump(cancer_dict, outfile2)
```

There are of course many more papers of each category, but is there any overlap in the two sets of papers that you identified? **(3 points)**

There are two papers that overlap between the two sets of papers identified.

In [12]:
```
for key in alzheimer_metadata:
    if key in cancer_metadata:
        print(key, cancer_metadata[key])
        print(key, alzheimer_metadata[key])
        print("\n")
```

36321615 {'ArticleTitle': 'Association of spermidine plasma levels with brain aging i
n a population-based study.', 'AbstractText': "Supplementation with spermidine may su
pport healthy aging, but elevated spermidine tissue levels were shown to be an indica
tor of Alzheimer's disease (AD).Data from 659 participants (age range: 21-81 years) o
f the population-based Study of Health in Pomerania TREND were included. We investiga
ted the association between spermidine plasma levels and markers of brain aging (hipp
ocampal volume, AD score, global cortical thickness [CT], and white matter hyperinten
sities [WMH]).Higher spermidine levels were significantly associated with lower hippo
campal volume (ß = -0.076; 95% confidence interval [CI]: -0.13 to -0.0
2; q = 0.026), higher AD score (ß = 0.118; 95% CI: 0.05 to 0.19;
q = 0.006), lower global CT (ß = -0.104; 95% CI: -0.17 to -0.04;
q = 0.014), but not WMH volume. Sensitivity analysis revealed no substantial cha
nges after excluding participants with cancer, depression, or hemolysis.Elevated sper
midine plasma levels are associated with advanced brain aging and might serve as pote
ntial early biomarker for AD and vascular brain pathology.", 'query': 'cancer'}
36321615 {'ArticleTitle': 'Association of spermidine plasma levels with brain aging i
n a population-based study.', 'AbstractText': "Supplementation with spermidine may su
pport healthy aging, but elevated spermidine tissue levels were shown to be an indica
tor of Alzheimer's disease (AD).Data from 659 participants (age range: 21-81 years) o
f the population-based Study of Health in Pomerania TREND were included. We investiga
ted the association between spermidine plasma levels and markers of brain aging (hipp
ocampal volume, AD score, global cortical thickness [CT], and white matter hyperinten
sities [WMH]).Higher spermidine levels were significantly associated with lower hippo
campal volume (ß = -0.076; 95% confidence interval [CI]: -0.13 to -0.0
2; q = 0.026), higher AD score (ß = 0.118; 95% CI: 0.05 to 0.19;
q = 0.006), lower global CT (ß = -0.104; 95% CI: -0.17 to -0.04;
q = 0.014), but not WMH volume. Sensitivity analysis revealed no substantial cha
nges after excluding participants with cancer, depression, or hemolysis.Elevated sper
midine plasma levels are associated with advanced brain aging and might serve as pote
ntial early biomarker for AD and vascular brain pathology.", 'query': 'Alzheimers'}

36321363 {'ArticleTitle': 'Severe cerebrovascular pathology of the first supercentena
rian to be autopsied in the world.', 'AbstractText': "We report on a 116-year-old Jap
anese woman who was the first officially documented supercentenarian to be autopsied
in the world. She lived a remarkably healthy life until suffering cerebral infarction
at 109 years of age. She became Japan's oldest person at 113 years and di
ed in 1995 from colon cancer at 116 years 175 days. Her medical records s
how the delayed onset of stroke, cancer, dementia, and heart disease and the importan
ce of appropriate medical treatment and intensive dedicated care provided during the
last stage of her life. She was the longest-lived person in Japan for 21 years
from 1993 until 2014. The neuropathological findings of her autopsied brain were brie
fly reported in the Japanese literature in 1997. In this study, we reinvestigated her
brain and spinal cord in more detail. Severe cerebrovascular lesions and cervical spo
ndylotic myelopathy were found to be the main causes of her disability. Although the
density of senile plaques was relatively high, the distribution of neurofibrillary ta
ngles was limited. Ghost tangles and argyrophilic grains were mild. The mildness of t
au pathological changes in her neurons, in other words the resistance of neurons to t
au pathology, may be a factor responsible for her longevity.", 'query': 'cancer'}
36321363 {'ArticleTitle': 'Severe cerebrovascular pathology of the first supercentena
rian to be autopsied in the world.', 'AbstractText': "We report on a 116-year-old Jap
anese woman who was the first officially documented supercentenarian to be autopsied
in the world. She lived a remarkably healthy life until suffering cerebral infarction
at 109 years of age. She became Japan's oldest person at 113 years and di
ed in 1995 from colon cancer at 116 years 175 days. Her medical records s
how the delayed onset of stroke, cancer, dementia, and heart disease and the importan
ce of appropriate medical treatment and intensive dedicated care provided during the
last stage of her life. She was the longest-lived person in Japan for 21 years
from 1993 until 2014. The neuropathological findings of her autopsied brain were brie
fly reported in the Japanese literature in 1997. In this study, we reinvestigated her

brain and spinal cord in more detail. Severe cerebrovascular lesions and cervical spo
ndylotic myelopathy were found to be the main causes of her disability. Although the
density of senile plaques was relatively high, the distribution of neurofibrillary ta
ngles was limited. Ghost tangles and argyrophilic grains were mild. The mildness of t
au pathological changes in her neurons, in other words the resistance of neurons to t
au pathology, may be a factor responsible for her longevity.", 'query': 'Alzheimers'}

Be sure to store all parts (of AbstractText). You could do this in many ways, from using a dictionary or a list or simply concatenating with a space in between. Discuss any pros or cons of your choice in your readme. **(1 point)**

I have saved my AbstractText as an XML list object.

Pros:
This retains all the labels, label names as well as all the text that they contain.
It also deals with any unwanted tags like \sup> and \sub> because it leaves them as they are to be processed later.
It can easily be converted back to XML to manipulate later.

Cons:
It saves the text as a 'bytes' object instead of a 'string' object. This is inconvenient if you want to save it as a JSON but allows you to save a .pkl file.
It captures unneccessary information like the 'CopyrightInformation' label but this can be easily removed in processing (as has been done in Q2) as the element is in XML formatting.

QUESTION 2 - SPECTER embedding and PCA

Tahir Manuel D Mello

BIS634 Assignment 3

In [1]:
```
import pickle
import xml.etree.ElementTree as ET
import seaborn as sns

#Large outputs removed for readibility of README, refer to ipynb if needed.
```

Load the papers dictionary. **(3 points)**

Keep track of which papers came from searching for Alzheimers, which came from searching for cancer. **(5 points)**

In [2]:
```
alzheimers_dict = pickle.load( open( "dictionary_Alzheimers.pkl", "rb" ) )
cancer_dict = pickle.load( open( "dictionary_cancer.pkl", "rb" ) )
```

In [ ]:
```
alzheimers_dict
```

In [ ]:
```
cancer_dict
```

In [ ]:
```
#Preprocessing to convert XML list abstract to string.

alzheimers_paper_id = list(alzheimers_dict.keys())

for id_list in alzheimers_paper_id:

    if alzheimers_dict[str(id_list)]['AbstractText'] == [] or len(alzheimers_dict[str(
        alzheimers_dict[id_list]['AbstractText'] = ""
        continue

    item = alzheimers_dict[str(id_list)]['AbstractText'][0][0] #Choose saved xml formc
    tree = ET.ElementTree(ET.fromstring(item))
    root = tree.getroot()

    for child in root:
        if child.tag == 'CopyrightInformation':
            root.remove(child)

    alzheimers_dict[id_list]['AbstractText'] = ET.tostring(root, method="text").decode

alzheimers_dict
```

In [ ]:
```
cancer_paper_id = list(cancer_dict.keys())

for id_list in cancer_paper_id:

    if cancer_dict[str(id_list)]['AbstractText'] == [] or len(cancer_dict[str(id_list)
        cancer_dict[id_list]['AbstractText'] = ""
        continue

    item = cancer_dict[str(id_list)]['AbstractText'][0][0] #Choose saved xml format st
```

```
    tree = ET.ElementTree(ET.fromstring(item))
    root = tree.getroot()

    for child in root:
        if child.tag == 'CopyrightInformation':
            root.remove(child)

    cancer_dict[id_list]['AbstractText'] = ET.tostring(root, method="text").decode()

cancer_dict
```

In [7]:
```
papers = alzheimers_dict | cancer_dict
```

In [8]:
```
len(papers) #Two overlapping papers as seen in Q1
```

Out[8]:
```
1998
```

Process your dictionary of papers to find the SPECTER embeddings. **(2 points)**

In [9]:
```
from transformers import AutoTokenizer, AutoModel

# load model and tokenizer
tokenizer = AutoTokenizer.from_pretrained('allenai/specter')
model = AutoModel.from_pretrained('allenai/specter')
```

In [10]:
```
import tqdm

# we can use a persistent dictionary (via shelve) so we can stop and restart if needed
# alternatively, do the same but with embeddings starting as an empty dictionary
embeddings = {}

for pmid, paper in tqdm.tqdm(papers.items()):
    data = [paper["ArticleTitle"] + tokenizer.sep_token + paper['AbstractText']]
    inputs = tokenizer(
        data, padding=True, truncation=True, return_tensors="pt", max_length=512
    )
    result = model(**inputs)
    # take the first token in the batch as the embedding
    embeddings[pmid] = result.last_hidden_state[:, 0, :].detach().numpy()[0]

# turn our dictionary into a list
embeddings = [embeddings[pmid] for pmid in papers.keys()]
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
1998/1998 [13:29<00:00,  2.47it/s]
```

In [ ]:
```
embeddings[0]
```

In [12]:
```
len(embeddings[0])
```

Out[12]:
```
768
```

Apply principal component analysis (PCA) to identify the first three principal components. **(5 points)**

In [13]:
```
from sklearn import decomposition
```

```
import pandas as pd

pca = decomposition.PCA(n_components=3)
embeddings_pca = pd.DataFrame(
    pca.fit_transform(embeddings),
    columns=['PC0', 'PC1', 'PC2']
)
embeddings_pca["query"] = [paper["query"] for paper in papers.values()]
```

In [14]:
```
embeddings_pca
```

Out[14]:

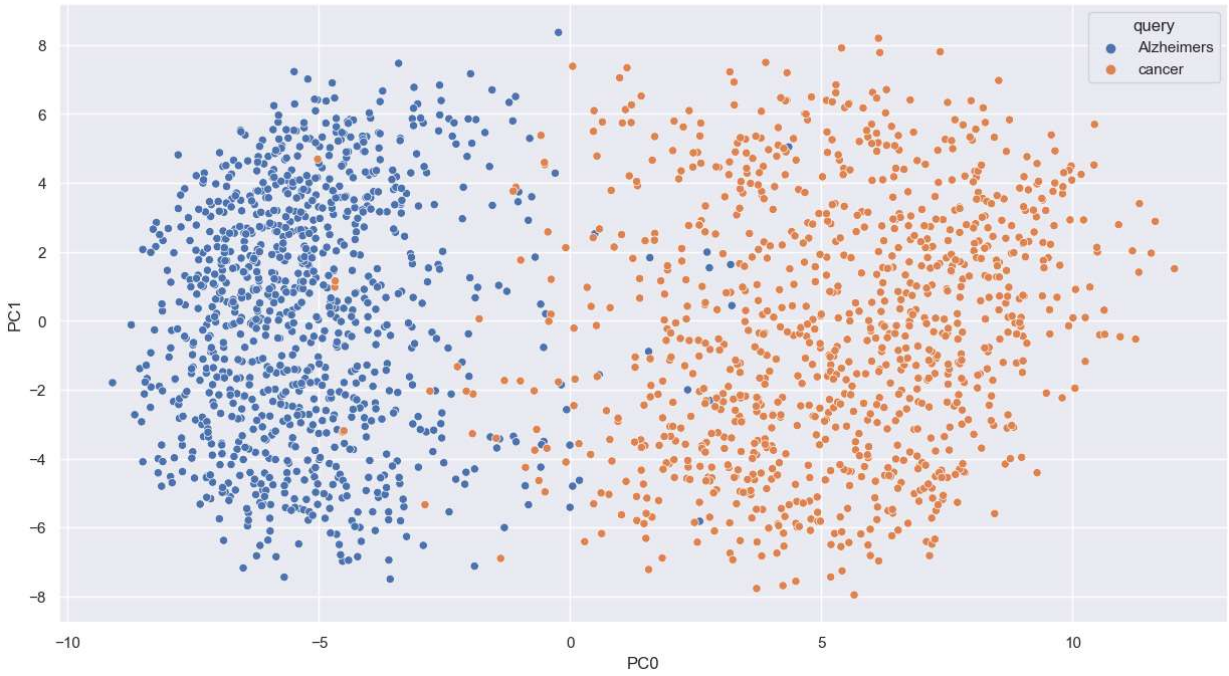
	PC0	PC1	PC2	query
0	-0.894501	-4.790774	-0.219417	Alzheimers
1	-6.128325	3.761800	0.179975	Alzheimers
2	-5.626882	4.072326	-1.862980	Alzheimers
3	-5.442851	-4.372836	-0.052957	Alzheimers
4	-5.754705	-2.198479	0.456736	Alzheimers
...	...	...	...	...
1993	2.067129	-4.032015	5.485044	cancer
1994	5.540362	6.282101	3.121495	cancer
1995	3.670037	-3.557976	2.529594	cancer
1996	2.914889	-2.243386	4.279892	cancer
1997	7.153906	-6.818355	2.493515	cancer

1998 rows × 4 columns

Plot 2D scatter plots for PC0 vs PC1, PC0 vs PC2, and PC1 vs PC2; color code these by the search query used (Alzheimers vs cancer). **(5 points)**

In [16]:
```
x = embeddings_pca['PC0']
y = embeddings_pca['PC1']
classes = embeddings_pca['query']
colours = ['chocolate', 'blue']

sns.set(rc = {'figure.figsize':(15,8)})
sns.scatterplot(x=x, y=y, hue=classes)
```

Out[16]:
```
<AxesSubplot:xlabel='PC0', ylabel='PC1'>
```



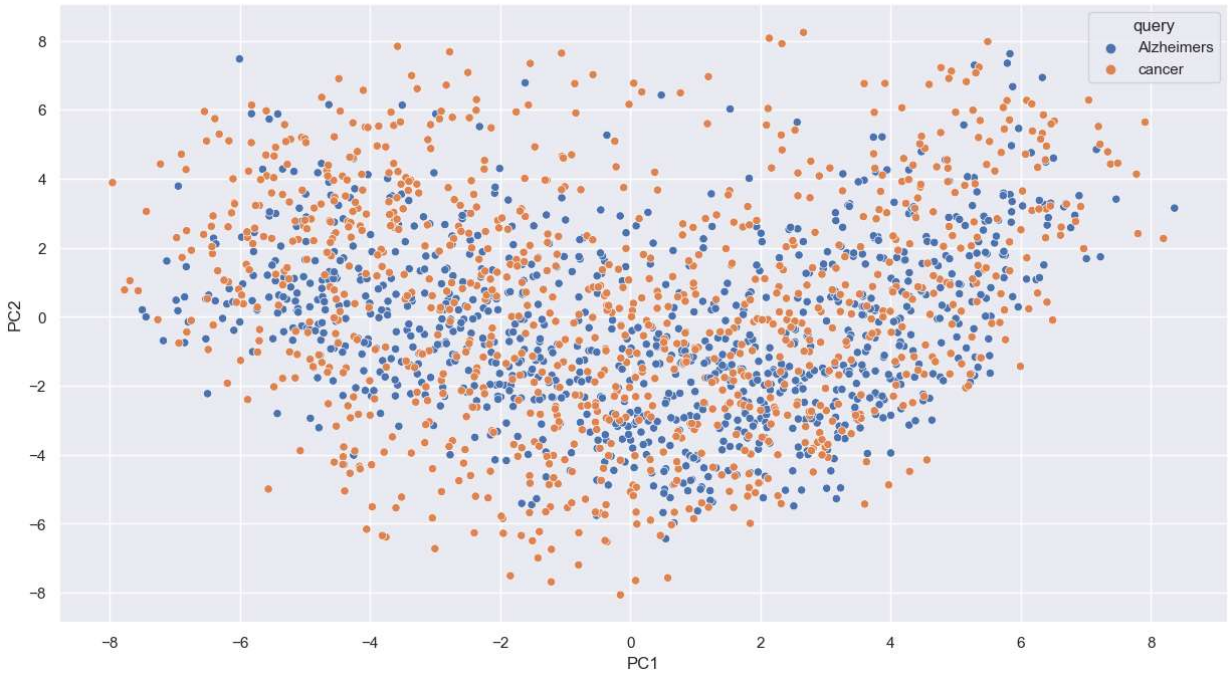
In [17]:
```
x = embeddings_pca['PC0']
y = embeddings_pca['PC2']
classes = embeddings_pca['query']
colours = ['chocolate', 'blue']

sns.set(rc = {'figure.figsize':(15,8)})
sns.scatterplot(x=x, y=y, hue=classes)
```

Out[17]:
```
<AxesSubplot:xlabel='PC0', ylabel='PC2'>
```

In [18]:
```
x = embeddings_pca['PC1']
y = embeddings_pca['PC2']
classes = embeddings_pca['query']
colours = ['chocolate', 'blue']
```

```
sns.set(rc = {'figure.figsize':(15,8)})
sns.scatterplot(x=x, y=y, hue=classes)
```

Out[18]: <AxesSubplot:xlabel='PC1', ylabel='PC2'>



Comment on the separation or lack thereof, and any take-aways from that. **(5 points)**

We process the dictionary of papers to find the SPECTER embeddings of each entry.
This embedding returns vectors of 768 dimension that express the abstract and title of each entry in the papers dicitonary,
If the papers are alike, the title and abstract will have sentences that are closer in meaning. Thus, their embedding vectors will also be closer to each other.

However, there are too many dimensions (768) to work with and visualize.
We thus reduce the dimensions with PCA while still maintaining data meaning.
This is done by reducing it and choosing the PCA compononents that express the most variance.
Thus, PC0 will explain more variance of the data than PC1 and PC2.
(The variance explained by each can be found by looking at the eigenvalues of the covariance matrix of the data.)
The number of components that are to be considered can be decided based on how much variance needs to be retained.

Thus, when we graph PC0 and PC1 or PC2, we see clear separation along the PC0 (x) axis as it explains most of the variance in the dataset. There are a few points of overlap which makes sense since there are papers working in similar areas of both conditions (cancer and Alzheimers).
For the graph of PC1 vs PC2, we don't see separation as these components explain far lesser of the variance in the dataset.

QUESTION 3 - SIR Model

Tahir Manuel D Mello

BIS634 Assignment 3

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from plotnine import ggplot, aes, geom_tile, labs, scale_fill_gradientn, theme_voi
                     element_blank, element_rect, scale_x_discrete, scale_y_discre
import plotnine
```

In [2]:
```
N = 134000
I0 = 1
R0 = 0
S0 = N - I0 - R0
beta = 2
gamma = 1
delta_t = 0.1 #days
tmax = 1000 #days
```

Provide your own implementation for Explicit Euler method to plot i(t). **(5 points)**

In [3]:
```
def deriv(N, I, R, S, beta, gamma):
    dSdt = -beta * S * I / N
    dIdt = beta * S * I / N - gamma * I
    dRdt = gamma * I
    return dIdt, dRdt, dSdt
```

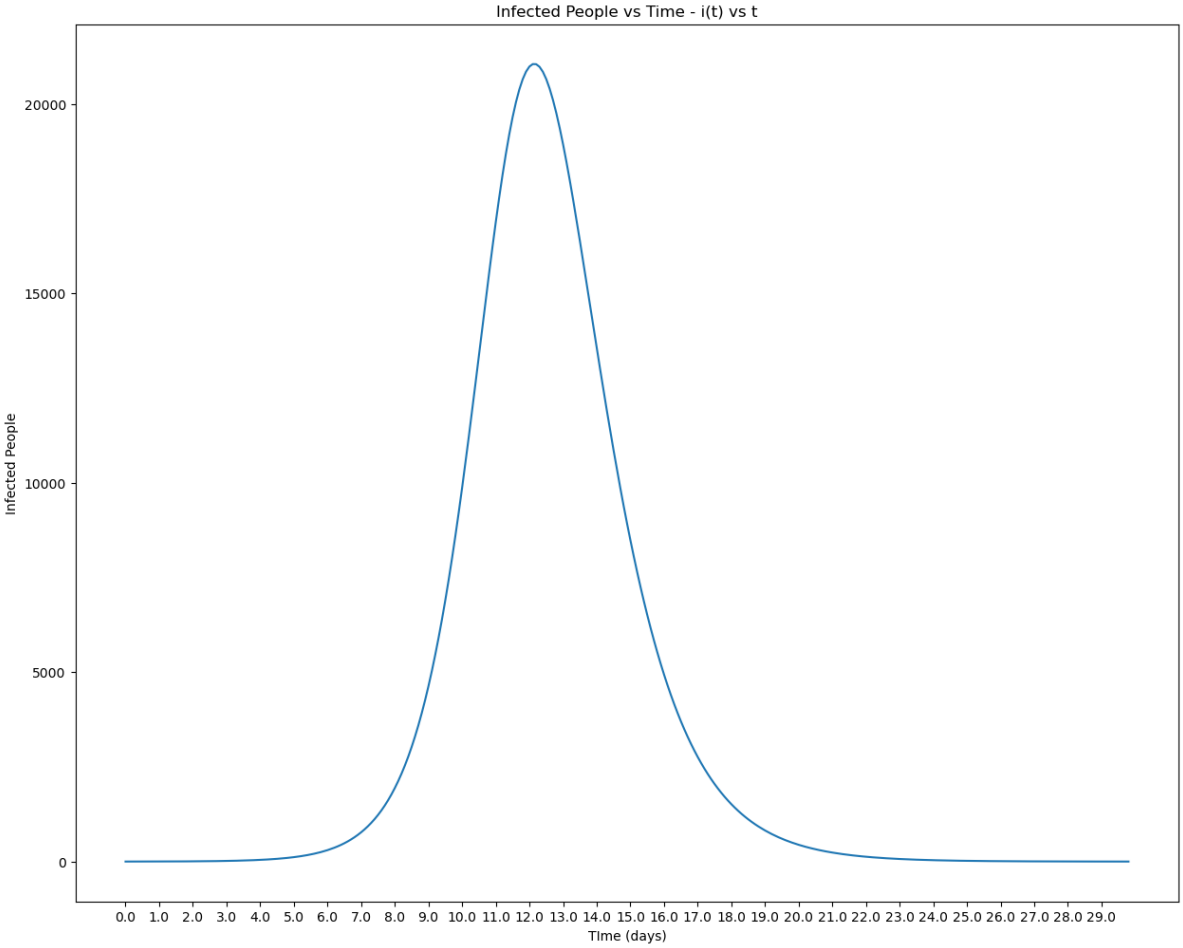
In [4]:
```
def i_plot(N, I0, R0, S0, beta, gamma, delta_t, tmax, printf, plot):

    I_set = []
    R_set = []
    S_set = []

    I_set.append(I0)
    R_set.append(R0)
    S_set.append(S0)

    I_previous, R_previous, S_previous = I0, R0, S0

    Idot_previous, Rdot_previous, Sdot_previous = deriv(N, I_previous, R_previous,

    for i in range(tmax):
        I_next = I_previous + Idot_previous*delta_t
        R_next = R_previous + Rdot_previous*delta_t
        S_next = S_previous + Sdot_previous*delta_t

        I_set.append(I_next)
        R_set.append(R_next)
        S_set.append(S_next)

        I_previous, R_previous, S_previous = I_next, R_next, S_next

        Idot_previous, Rdot_previous, Sdot_previous = deriv(N, I_previous, R_previ

        if I_next < 1:
            break
```

```
    peak = round(max(I_set))

    peakday = round(np.argmax(np.array(I_set))*delta_t)

    timescale = np.arange(0,len(I_set), 1)

    if printf == True:
        print("The peak number of infected people was " + str(peak) + "." )
        print("The peak number of infected people happened on Day " + str(peakday)

    if plot == True:
        plt.figure(figsize=(15, 12))
        plt.plot(timescale, I_set)

        plt.title('Infected People vs Time - i(t) vs t')
        plt.ylabel('Infected People')
        plt.xlabel('TIme (days)')

        plt.xticks(np.arange(0,len(I_set), 10), np.arange(0, len(I_set)*delta_t, 1

    return I_set, peak, peakday
```

For the New Haven case, plot the time course of the number of infected individuals until that number drops below 1 (at which point, we'll assume the disease has run its course). **(5 points)**

In [5]:
```
I_set_out, max_i, may_day = i_plot(N, I0, R0, S0, beta, gamma, delta_t, tmax, True
```

```
The peak number of infected people was 21050.
The peak number of infected people happened on Day 12.
```



For those parameter values, when does the number of infected people peak? **(2 points)**
How many people are infected at the peak? **(3 points)**

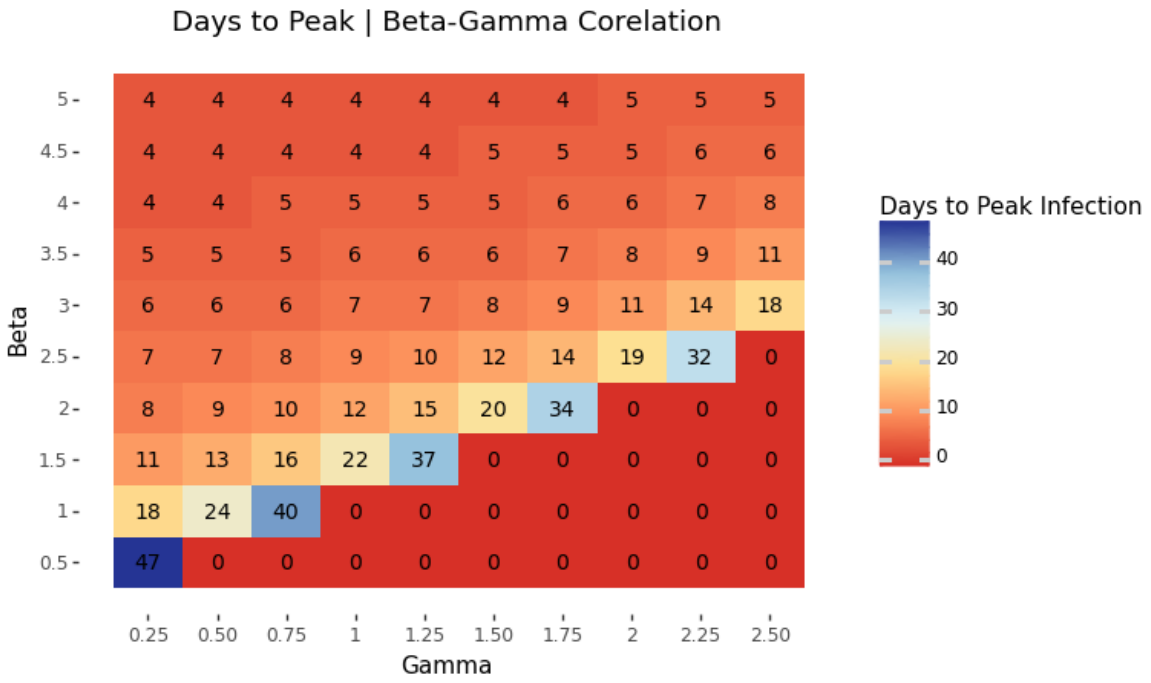
In [6]:
```
I_set_out, max_i, max_day = i_plot(N, I0, R0, S0, beta, gamma, delta_t, tmax, True
```

```
The peak number of infected people was 21050.
The peak number of infected people happened on Day 12.
```

Plot on a heat map how the time of the peak of the infection depends on gamma and beta.

(5 points)

In [7]:
```
gamma_set = np.arange(0.25,2.6, 0.25)
beta_set = np.arange(0.5,5.5,0.5)
data_out = []

for gamma in gamma_set:
    for beta in beta_set:
        I_set_out, max_i, max_day = i_plot(N, I0, R0, S0, beta, gamma, delta_t, tma
        temp = [None]*3
        temp[0] = gamma
        temp[1] = beta
        temp[2] = max_day

        data_out.append(temp)

data_out = pd.DataFrame(data_out)
data_out.columns = ['Gamma', 'Beta', 'Days to Peak Infection']

color_palette = "#d73027,#fc8d59,#fee090,#e0f3f8,#91bfdb,#253494"
color_palette_list = color_palette.split(",")

ggplot(mapping = aes(x = 'Gamma', y = 'Beta'), data = data_out) + \
    geom_tile(aes(fill = 'Days to Peak Infection')) +\
    geom_text(aes(label='Days to Peak Infection'), size=10) +\
    scale_fill_gradientn(colors=color_palette_list) +\
    labs(title = "Days to Peak | Beta-Gamma Corelation", x = "Gamma", y = "Beta")+
    scale_x_continuous(breaks = gamma_set) +\
    scale_y_continuous(breaks = beta_set) +\
    theme(panel_background=element_rect(fill='white') )
```

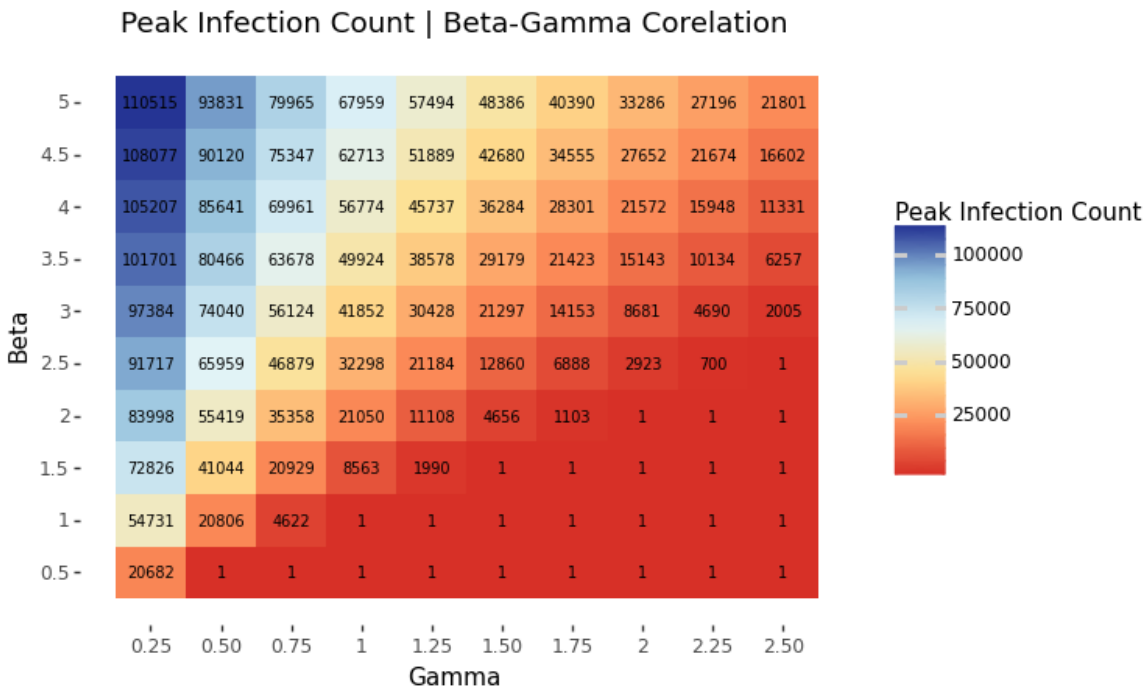


Out[7]:
```
<ggplot: (122574107115)>
```

Plot on a heat map how the number of individuals infected at peak depends on gamma and

beta. **(5 points)**

In [8]:
```
gamma_set = np.arange(0.25,2.6, 0.25)
beta_set = np.arange(0.5,5.5,0.5)
data_out = []

for gamma in gamma_set:
    for beta in beta_set:
        I_set_out, max_i, max_day = i_plot(N, I0, R0, S0, beta, gamma, delta_t, tma
        temp = [None]*3
        temp[0] = gamma
        temp[1] = beta
        temp[2] = max_i

        data_out.append(temp)

data_out = pd.DataFrame(data_out)
data_out.columns = ['Gamma', 'Beta', 'Peak Infection Count']

color_palette = "#d73027,#fc8d59,#fee090,#e0f3f8,#91bfdb,#253494"
color_palette_list = color_palette.split(",")

ggplot(mapping = aes(x = 'Gamma', y = 'Beta'), data = data_out) + \
    geom_tile(aes(fill = 'Peak Infection Count')) +\
    geom_text(aes(label='Peak Infection Count'), size=7) +\
    scale_fill_gradientn(colors=color_palette_list) +\
    labs(title = "Peak Infection Count | Beta-Gamma Corelation", x = "Gamma", y =
    scale_x_continuous(breaks = gamma_set) +\
    scale_y_continuous(breaks = beta_set) +\
    theme(panel_background=element_rect(fill='white') )
```



Out[8]:
```
<ggplot: (122574258664)>
```

QUESTION 4 - Project Dataset

Tahir Manuel D Mello

BIS634 Assignment 3

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Identify a data set online that you find interesting that could potentially be used for the final project; the main requirements is that there needs to be many (hundreds or more) data items with several identifiable variables, at least one of which could be viewed as an output variable that you could predict from the others. **(5 points)**

In [2]:
```
data = pd.read_csv('players_22.csv', low_memory=False)
```

This is a dataset titled 'FIFA 22 complete player dataset' compiled by STEFANO LEONE on Kaggle. https://www.kaggle.com/datasets/stefanoleone992/fifa-22-complete-player-dataset?select=players_22.csv

It contains all data about all soccer (referred to as football from here on for my convenience :)) players on the FIFA Football game published by EA Sports FC. The data was scraped from the publicly available website sofifa.com by Stefano Leone and uploaded to Kaggle.

It contains 110 columns of information about 19239 professional players. Apart from game stats, it also contains information about player nationality, position, age and importantly for this project, player contract information.

The output variable I am interested in is the player wage and player value. These are variables that are generally determined by various factors that may be football related (skill, position, etc.) as well as others that are not like age, contract status, maybe nationality.

I want to build a predictor of player wage and value with the other variables present in this dataset. I plan on using just the database of players from the 2022 version of the game.

This will potentially allow me to do two interesting things:

1. Find out which variables have the most effect on player wage (with the most weight in the predictor, maybe)
2. Find out which players are under/overpaid and why (maybe).

In [3]:
```
data.head()
```

Out[3]:

	sofifa_id	player_url	short_name	long_name	player_positions	overa
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	RW, ST, CF	
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Robert Lewandowski	ST	
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	ST, LW	
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Júnior	LW, CAM	
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	CM, CAM	

5 rows × 110 columns

In [4]:
```
len(data)
```

Out[4]:
```
19239
```

Describe the dataset. [Your answer should address (but not be limited to): how many variables? Are the key variables explicitly specified or are they things you would have to derive (e.g. by inferring from text)? Are any of the variables exactly derivable from other variables? (i.e. are any of them redundant?) Are there any variables that could in principle be statistically predicted from other variables? How many rows/data points are there? Is the data in a standard format? If not, how could you convert it to a standard format?] **(5 points)**

The raw data has a 110 variables. (Not all of them are useful)
The raw data has 19239 rows. Each one is a unique player (conveniently identified by the unique sofifa_id).

There is a lot of important player metadata like club, age, wage, posiitions played, information about contracts and nationality. These need to be cleaned in various ways to standardize the data.
Several of the key variables are explicitly specified (pace, shooting and other stats are defined on a scale of 0 to 99).
Some important variables have not been defined explicitly. Player positions have been specified as strings (sometimes with multiple entries if a player plays in multiple positions). These will need to be stripped and cleaned properly to convert it to standard form.
Variables might also have to be normalized since there are multiple distribution types for different variables in the data.

There are a lot of unneccesary variables like logo URLs, jersey numbers, position dependent ratings, etc. that will be discarded during data cleaning.
The total player rating is in practice statistically predicted by other ratings. However, the weighting of these ratings is not known clearly.

Describe the terms of use and identify any key restrictions. **(5 points)**

This data has no restrictions. There is no access process or permissions required.
No type of analysis is forbidden on this data.

Both Kaggle and the original source - soFIFA.com - have open public use policies for their data.

Thus, this data can be shared and distributed freely.

Do data exploration on the dataset, and present a representative set of figures that gives insight into the data. Comment on the insights gained. **(5 points)**

All the variables are not useful.
I have sliced the dataset and picked out variables in 3 categories that will be needed for this project.

In [5]:
```
player_metadata = data[['sofifa_id', 'short_name', 'long_name', 'age', 'player_positic
                        'club_name', 'value_eur', 'wage_eur','release_clause_eur', \
                        'club_joined', 'club_contract_valid_until', \
                        'league_name', 'league_level' , 'nationality_name',\
                        ]]

player_attributes = data[['sofifa_id', 'height_cm', 'weight_kg', 'preferred_foot', \
                          'weak_foot', 'skill_moves', 'international_reputation', 'boc
                          'player_tags', 'player_traits']]

player_stats = pd.concat([ data[['sofifa_id', 'overall', 'potential']], data.iloc[:,37
```

In [6]:
```
player_metadata.head()
```

Out[6]:

	sofifa_id	short_name	long_name	age	player_positions	club_name	value_eur	wage_eur	re
0	158023	L. Messi	Lionel Andrés Messi Cuccittini	34	RW, ST, CF	Paris Saint-Germain	78000000.0	320000.0	
1	188545	R. Lewandowski	Robert Lewandowski	32	ST	FC Bayern München	119500000.0	270000.0	
2	20801	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	36	ST, LW	Manchester United	45000000.0	270000.0	
3	190871	Neymar Jr	Neymar da Silva Santos Júnior	29	LW, CAM	Paris Saint-Germain	129000000.0	270000.0	
4	192985	K. De Bruyne	Kevin De Bruyne	30	CM, CAM	Manchester City	125500000.0	350000.0	

In [7]:
```
player_attributes.head()
```

Out[7]:

	sofifa_id	height_cm	weight_kg	preferred_foot	weak_foot	skill_moves	international_reputation
0	158023	170	72	Left	4	4	5
1	188545	185	81	Right	4	4	5
2	20801	187	83	Right	4	5	5
3	190871	175	68	Right	5	5	5
4	192985	181	70	Right	5	4	4

In [8]:
```
player_stats
```

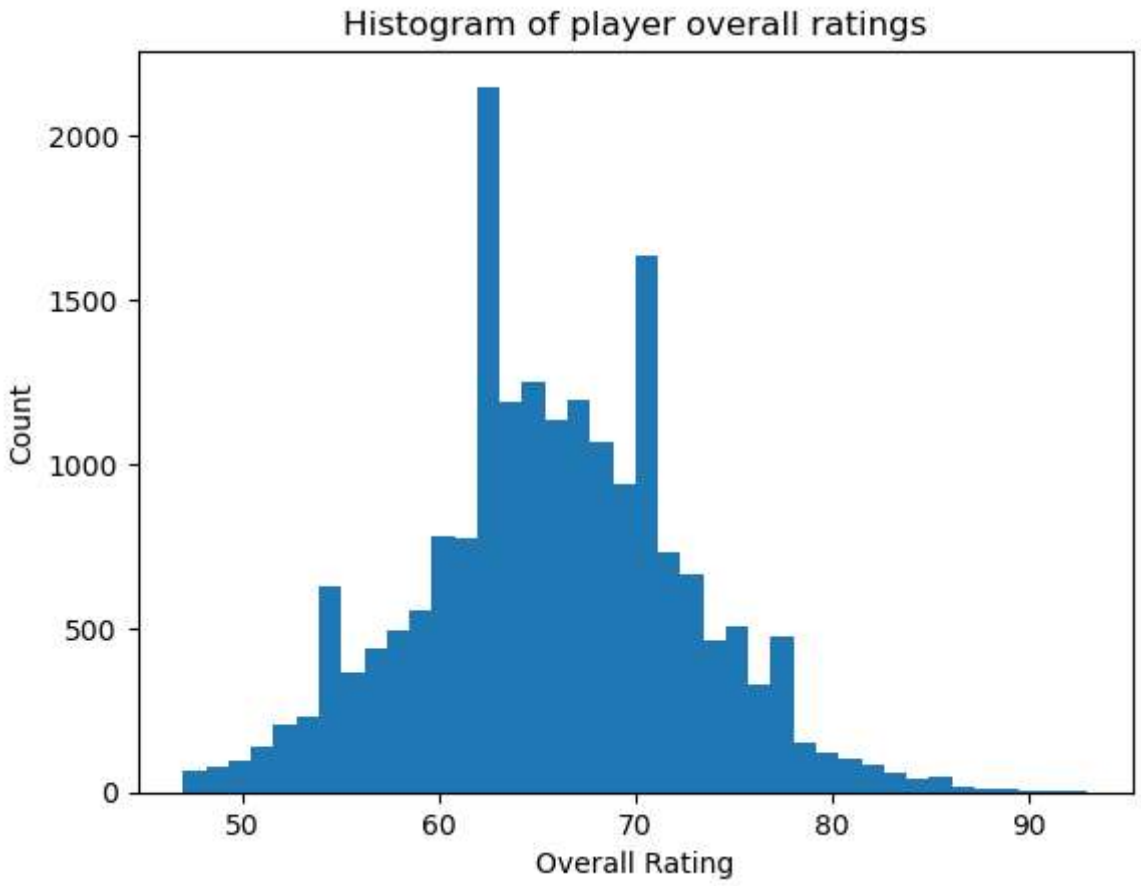
Out[8]:

	sofifa_id	overall	potential	pace	shooting	passing	dribbling	defending	physic	attacking
0	158023	93	93	85.0	92.0	91.0	95.0	34.0	65.0	
1	188545	92	92	78.0	92.0	79.0	86.0	44.0	82.0	
2	20801	91	91	87.0	94.0	80.0	88.0	34.0	75.0	
3	190871	91	91	91.0	83.0	86.0	94.0	37.0	63.0	
4	192985	91	91	76.0	86.0	93.0	88.0	64.0	78.0	
...	...	...	...	...	...	...	...	...	...	
19234	261962	47	52	58.0	35.0	46.0	48.0	42.0	49.0	
19235	262040	47	59	59.0	39.0	50.0	46.0	41.0	51.0	
19236	262760	47	55	60.0	37.0	45.0	49.0	41.0	52.0	
19237	262820	47	60	68.0	46.0	36.0	48.0	15.0	42.0	
19238	264540	47	60	68.0	38.0	45.0	48.0	36.0	48.0	

19239 rows × 44 columns

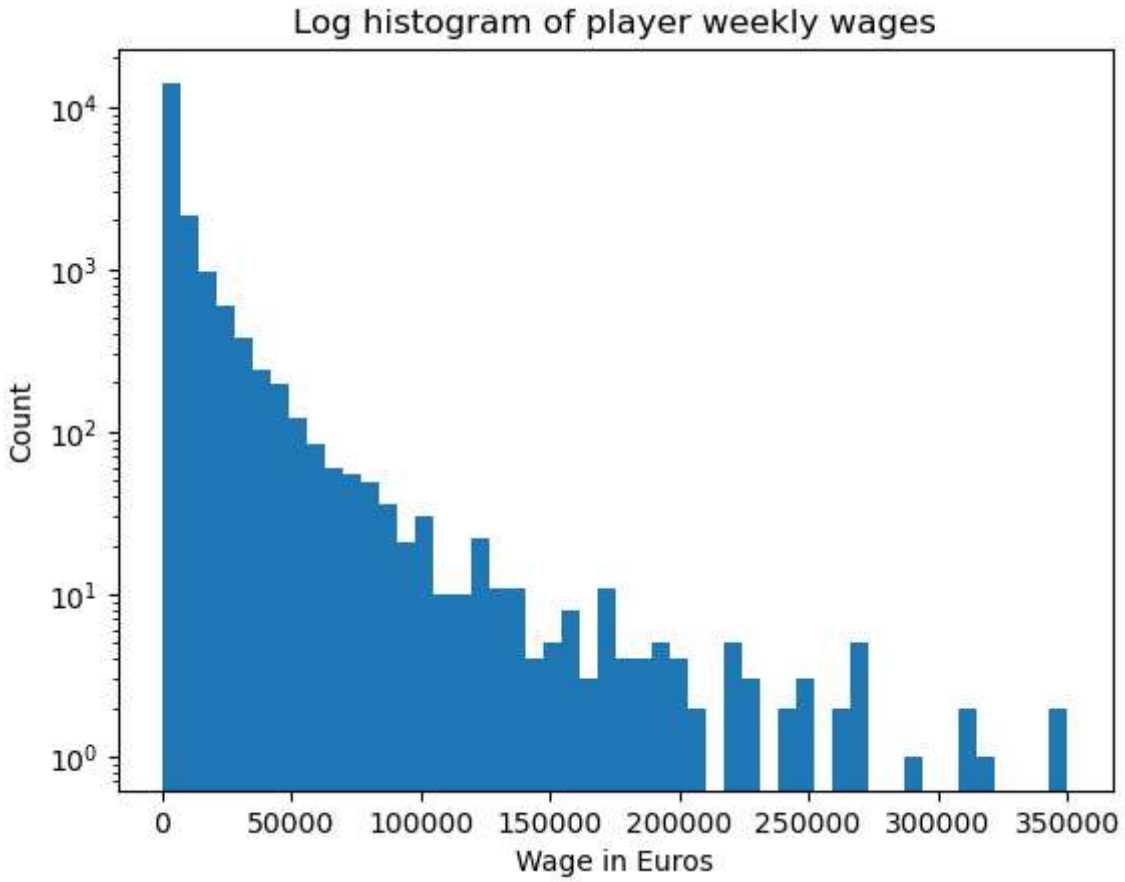
In [9]:

```
import matplotlib.pyplot as plt

plt.hist(player_stats['overall'], bins = 40)
plt.title('Histogram of player overall ratings')
plt.ylabel('Count')
plt.xlabel('Overall Rating')
plt.show()

print("The median rating of players in FIFA 22 is", int(player_stats['overall'].mediar
```



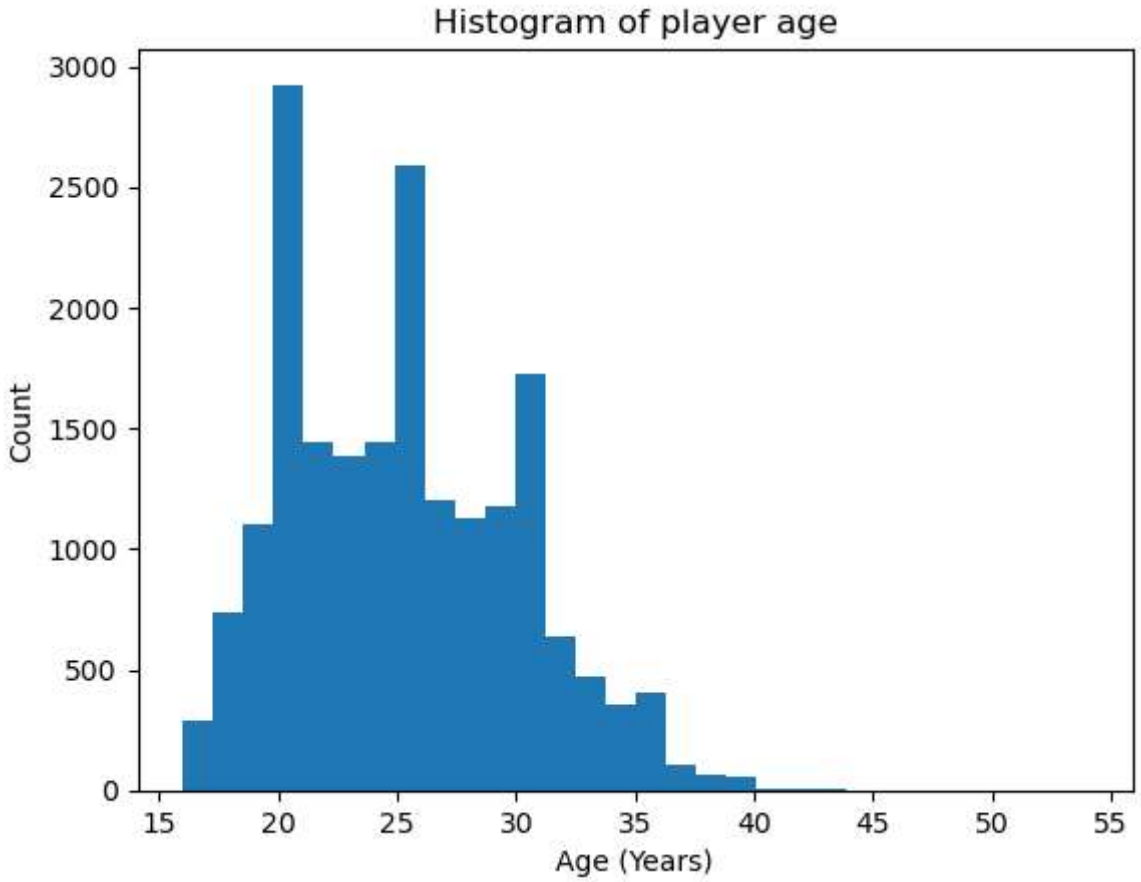
The median rating of players in FIFA 22 is 66 out of 100

In [10]:
```
plt.hist(player_metadata['wage_eur'], bins = 50, log=True)
plt.title('Log histogram of player weekly wages')
plt.ylabel('Count')
plt.xlabel('Wage in Euros')
plt.show()

print("The median weekly wage of players in FIFA 22 is", int(player_metadata['wage_eur
```



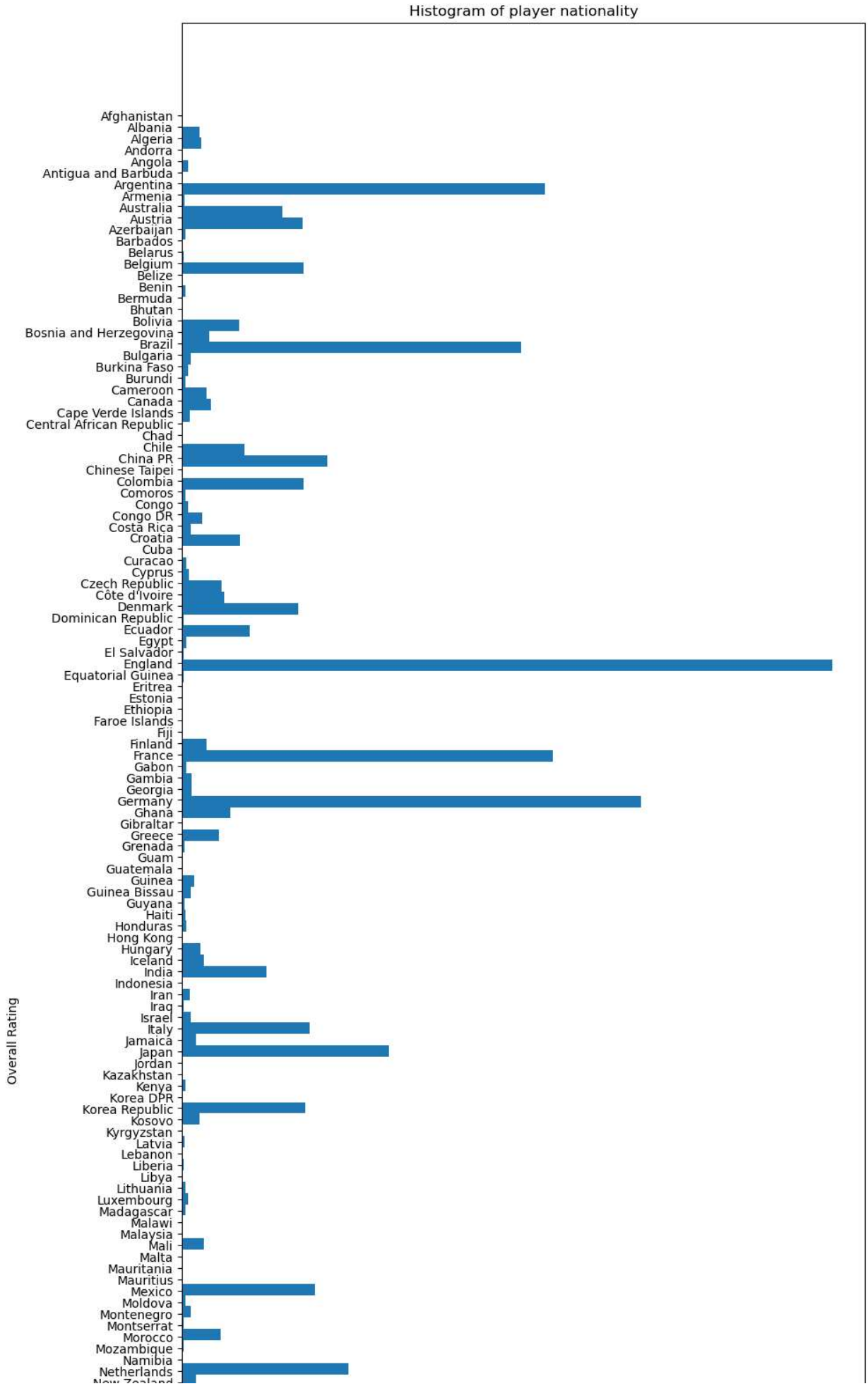
The median weekly wage of players in FIFA 22 is 3000 euros

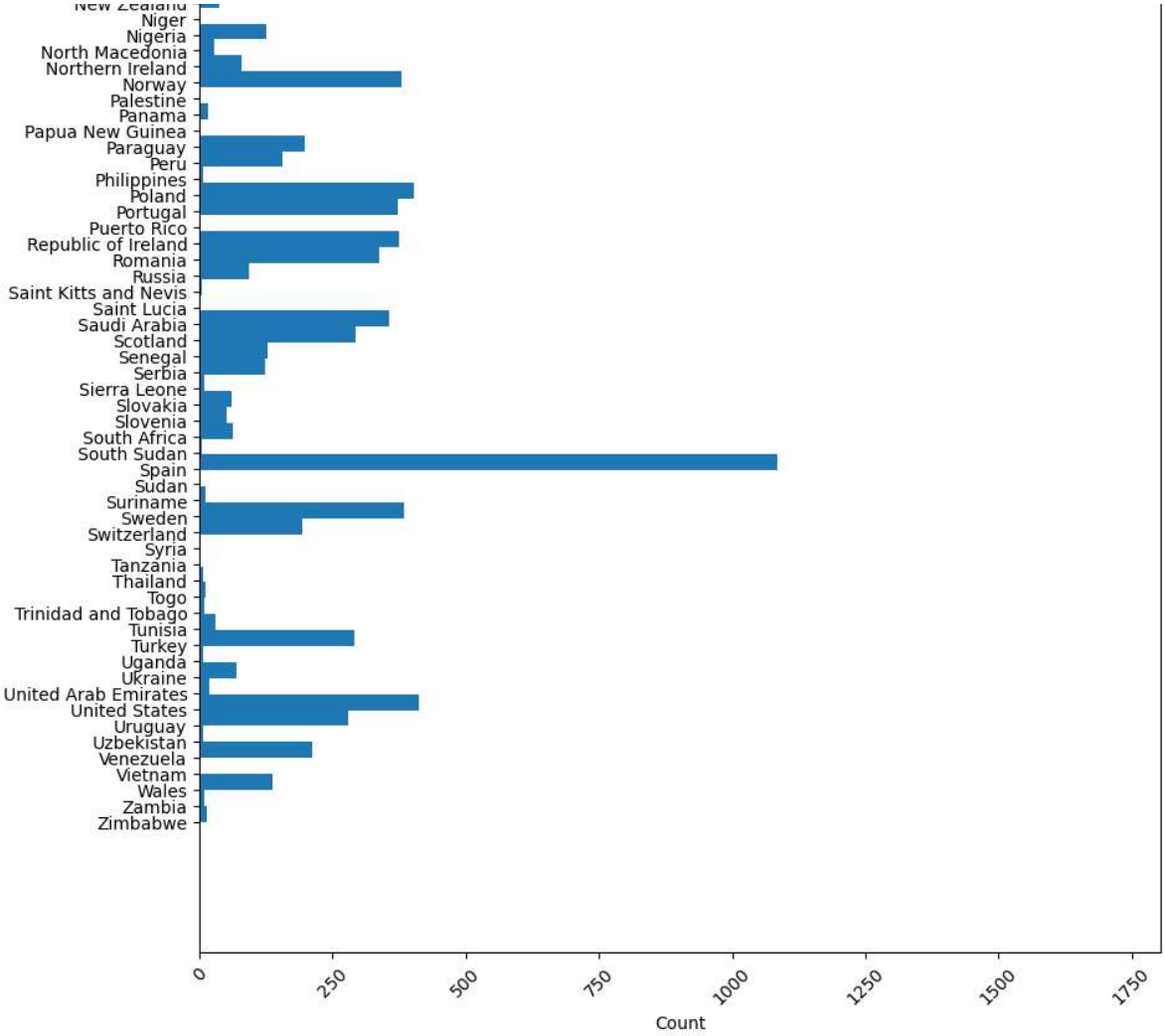
In [11]:
```
plt.hist(player_metadata['age'], bins = 30)
plt.title('Histogram of player age')
plt.ylabel('Count')
plt.xlabel('Age (Years)')
plt.show()

print("The median age of players in FIFA 22 is", int(player_metadata['age'].median()))
```



The median age of players in FIFA 22 is 25

In [12]:
```
plt.figure(figsize=(10,30))
plt.hist(sorted(player_metadata['nationality_name'], reverse=True), \
         bins = len(pd.unique(player_metadata['nationality_name'])),\
         orientation="horizontal")
plt.title('Histogram of player nationality')
plt.xlabel('Count')
plt.ylabel('Overall Rating')
plt.xticks(rotation=45)
plt.show()
```


Histogram of player nationality
Overall Rating
Afghanistan
Albania
Algeria
Andorra
Angola
Antigua and Barbuda
Argentina
Armenia
Australia
Austria
Azerbaijan
Barbados
Belarus
Belgium
Belize
Benin
Bermuda
Bhutan
Bolivia
Bosnia and Herzegovina
Brazil
Bulgaria
Burkina Faso
Burundi
Cameroon
Canada
Cape Verde Islands
Central African Republic
Chad
Chile
China PR
Chinese Taipei
Colombia
Comoros
Congo
Congo DR
Costa Rica
Croatia
Cuba
Curacao
Cyprus
Czech Republic
Côte d'Ivoire
Denmark
Dominican Republic
Ecuador
Egypt
El Salvador
England
Equatorial Guinea
Eritrea
Estonia
Ethiopia
Faroe Islands
Fiji
Finland
France
Gabon
Gambia
Georgia
Germany
Ghana
Gibraltar
Greece
Grenada
Guam
Guatemala
Guinea
Guinea Bissau
Guyana
Haiti
Honduras
Hong Kong
Hungary
Iceland
India
Indonesia
Iran
Iraq
Israel
Italy
Jamaica
Japan
Jordan
Kazakhstan
Kenya
Korea DPR
Korea Republic
Kosovo
Kyrgyzstan
Latvia
Lebanon
Liberia
Libya
Lithuania
Luxembourg
Madagascar
Malawi
Malaysia
Mali
Malta
Mauritania
Mauritius
Mexico
Moldova
Montenegro
Montserrat
Morocco
Mozambique
Namibia
Netherlands



The median player on the game is a 25 year old, English player with a 66/100 FIFA rating who earns 3000 euros per week.
It will be interesting to see how the top and bottom earners vary from this.

Identify any data cleaning needs (this includes checking for missing data) and write code to perform them. If the data does not need to be cleaned, explain how you reached this conclusion. **(5 points)**

Data cleaning required is of mainly the following types:

```
1. Changing all date/time related strings (ex. club_joined) to counts
of days/years.
2. Normalizing variables like weak foot, international reputation,
etc. (0-5 scale), height, weight (unique distributions), etc. to feed
into a predictor.
3. Splitting string categories (player_positions, player_tags,
player_traits) into separate columns with a 1 for present and 0 for
absent.
```

Another major cleaning step needed is to separate out goalkeepers from the rest.
They have very differnt stats across categories and will need to be processed separately.

There is some missing contractual data for players who do not currently have a contract with any club. These players will be removed from the dataset as they are currently not earning any weekly football related wage.

There could be more changes to make based on the predictor setup but these are what I think is needed now.

In [13]:
```
data_working = pd.concat([ player_metadata, player_stats, player_attributes ], axis=1)
len(data_working)
```

Out[13]:
```
19239
```

In [14]:
```
#Drop players with no clubs

data_dropped = data_working.dropna(subset=['club_name'])
len(data_dropped)
```

Out[14]:
```
19178
```

In [15]:
```
#Standardizing date to days since

basedate = pd.Timestamp('2022-11-05')
a = pd.to_datetime(data_dropped['club_joined'])
days = basedate - a
data_dropped['club_joined_time'] = days.astype('timedelta64[ns]')
data_dropped['club_joined_time']
```

```
C:\Users\tahir\AppData\Local\Temp\ipykernel_11496\3343727837.py:6: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  data_dropped['club_joined_time'] = days.astype('timedelta64[ns]')
```

Out[15]:
```
0        452 days
1       3049 days
2        435 days
3       1920 days
4       2624 days
           ...
19234    583 days
19235    621 days
19236    505 days
19237    505 days
19238    431 days
Name: club_joined_time, Length: 19178, dtype: timedelta64[ns]
```

The other data cleaning needed will depend on the type of predicion and processing that must be done on the data.