BIS634 - Assignment3

Name: Zhiyuan Cao; NetID: zc347

Exercise 1

Q1

I use the requests module as follows to acquire PMID of 1000 Alzheimers papers from 2022.

In [11]:
```
def getid_from_term(num, term):
    r = requests.get(
        "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
        f"esearch.fcgi?db=pubmed&retmode=xml&retmax={num}&term={term}"
    )
    doc = m.parseString(r.text)
    IdLists = doc.getElementsByTagName("Id")
    IdList = [IdLists[i].childNodes[0].wholeText for i in range(num)]
    return IdList

AlzheimersList = getid_from_term(1000, 'Alzheimer+AND+2022[pdat]')
print(f"The IDs for 1000 Alzheimer papers from 2022 are {AlzheimersList}")
```

```
The IDs for 1000 Alzheimer papers from 2022 are ['36309183', '36309087', '36308033', '36306920', '36306735', '36306
540', '36306459', '36306458', '36306386', '36305541', '36305459', '36305148', '36305125', '36304998', '36304823', '
36304723', '36304124', '36303331', '36302977', '36302665', '36302659', '36302488', '36302464', '36301043', '3629961
3', '36299608', '36298279', '36297317', '36297313', '36296980', '36296969', '36296692', '36296686', '36296677', '36
296574', '36296397', '36295605', '36295535', '36295014', '36294010', '36293946', '36293666', '36293539', '36293528'
, '36293516', '36293327', '36293221', '36293147', '36293049', '36292947', '36292945', '36292933', '36292931', '3629
2674', '36292623', '36292114', '36291714', '36291679', '36291666', '36291661', '36291639', '36291618', '36291595',
'36291553', '36291536', '36291224', '36291125', '36291068', '36291020', '36291017', '36290612', '36290138', '362898
78', '36289859', '36289565', '36289458', '36289390', '36289355', '36288997', '36288945', '36288546', '36288285', '3
6287840', '36287605', '36287554', '36286505', '36286438', '36286188', '36285785', '36284403', '36284365', '36284363
', '36284351', '36284252', '36284251', '36284177', '36284170', '36283631', '36282451', '36282189', '36282001', '362
81858', '36281688', '36281687', '36281686', '36281685', '36281683', '36281682', '36281681', '36281678', '36281677',
'36281676', '36281675', '36281671', '36281670', '36281668', '36281667', '36281666', '36281665', '36281664', '362816
63', '36281662', '36281661', '36281660', '36281659', '36281546', '36281465', '36281127', '36281092', '36281030', '3
6280690', '36280653', '36280370', '36280236', '36280008', '36279229', '36279227', '36279224', '36279115', '36279110
', '36278769', '36278470', '36278469', '36278440', '36278356', '36278355', '36278006', '36277564', '36277478', '362
77022', '36276653', '36276649', '36276340', '36276187', '36275801', '36275621', '36275013', '36275006', '36274975',
'36274680', '36274509', '36274383', '36274284', '36274138', '36274019', '36273804', '36273719', '36273484', '362734
56', '36273347', '36273219', '36273169', '36272739', '36272585', '36272532', '36271704', '36271678', '36271598', '3
```

Similarly I do the same for Cancer papers.

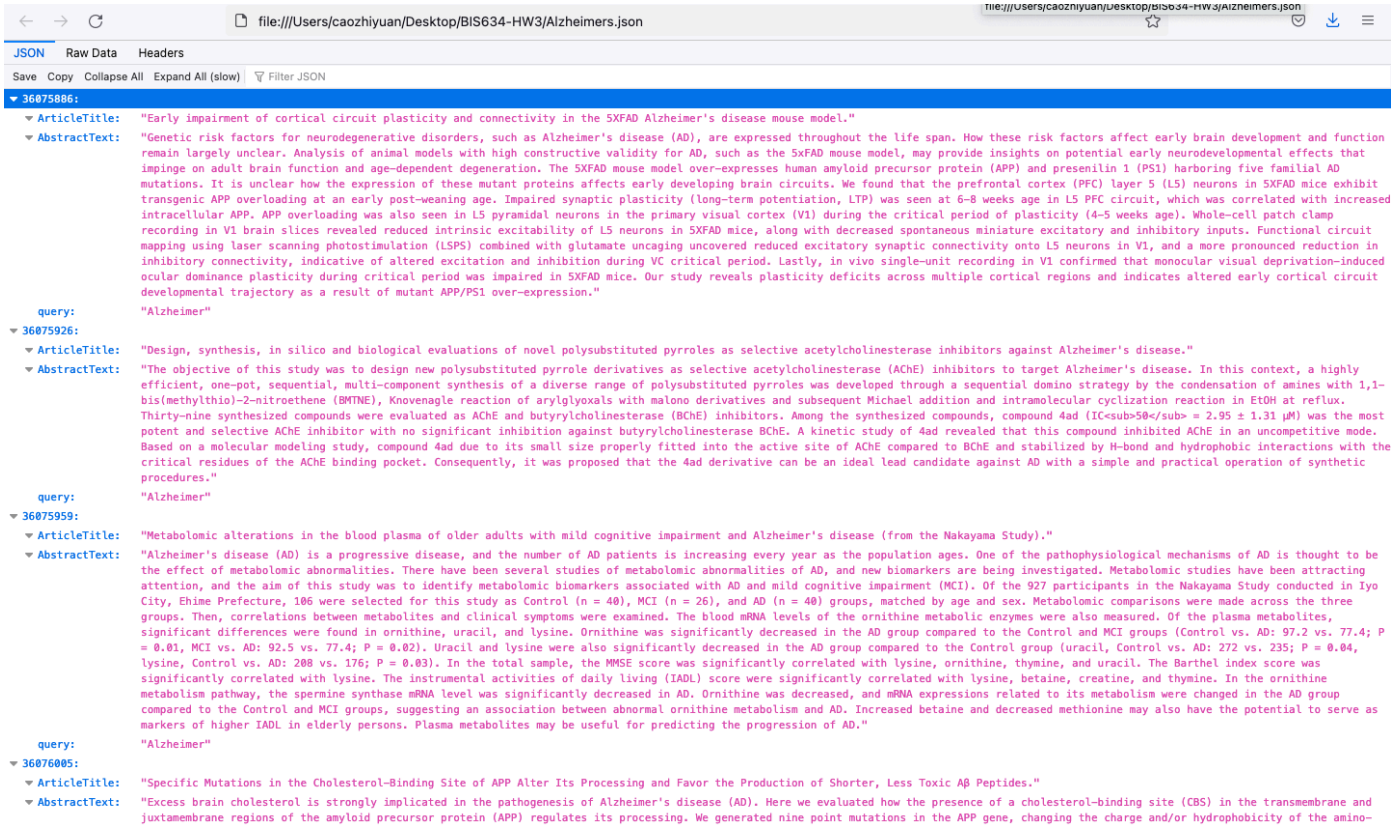
```python
def getid_from_term(num, term):
    r = requests.get(
        "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
        f"esearch.fcgi?db=pubmed&retmode=xml&retmax={num}&term={term}"
    )
    doc = m.parseString(r.text)
    IdLists = doc.getElementsByTagName("Id")
    IdList = [IdLists[i].childNodes[0].wholeText for i in range(num)]
    return IdList

CancerList = getid_from_term(1000, 'Cancer+AND+2022[pdat]')
print(f"The IDs for 1000 Cancer papers from 2022 are {CancerList}")
```

```
The IDs for 1000 Cancer papers from 2022 are ['36309847', '36309839', '36309838', '36309837', '36309831', '36309829
', '36309827', '36309820', '36309805', '36309761', '36309760', '36309745', '36309740', '36309731', '36309713', '363
09712', '36309698', '36309694', '36309693', '36309691', '36309687', '36309678', '36309672', '36309669', '36309667',
'36309666', '36309662', '36309655', '36309653', '36309646', '36309638', '36309636', '36309630', '36309626', '363096
19', '36309616', '36309615', '36309613', '36309604', '36309603', '36309602', '36309571', '36309561', '36309560', '3
6309559', '36309558', '36309551', '36309550', '36309544', '36309522', '36309516', '36309510', '36309507', '36309506
', '36309503', '36309502', '36309495', '36309486', '36309485', '36309484', '36309482', '36309474', '36309424', '363
09418', '36309416', '36309415', '36309414', '36309413', '36309406', '36309401', '36309400', '36309397', '36309395',
'36309391', '36309388', '36309387', '36309385', '36309384', '36309383', '36309382', '36309376', '36309374', '363093
71', '36309370', '36309369', '36309368', '36309366', '36309364', '36309362', '36309357', '36309356', '36309355', '3
6309354', '36309352', '36309349', '36309329', '36309321', '36309296', '36309224', '36309223', '36309222', '36309209
', '36309208', '36309204', '36309201', '36309200', '36309195', '36309193', '36309184', '36309154', '36309152', '363
09135', '36309131', '36309124', '36309122', '36309110', '36309105', '36309104', '36309102', '36309101', '36309096',
'36309086', '36309082', '36309080', '36309079', '36309077', '36309076', '36309075', '36309074', '36309073', '363090
72', '36309066', '36309037', '36309036', '36309018', '36309017', '36309008', '36308932', '36308926', '36308925', '3
6308919', '36308918', '36308915', '36308910', '36308909', '36308899', '36308884', '36308869', '36308859', '36308854
', '36308853', '36308849', '36308840', '36308836', '36308835', '36308822', '36308802', '36308778', '36308775', '363
08760', '36308746', '36308744', '36308741', '36308736', '36308711', '36308708', '36308702', '36308678', '36308675',
'36308669', '36308667', '36308666', '36308645', '36308643', '36308634', '36308630', '36308626', '36308624', '363086
```

Q2

I save two json files as instructed. See my github file for detail. Part of the json file is as follows.



For the abstract, if the article have multiple AbstractText fields, I store all the parts by simply concatenating with a space in between.

Pros:

- Easy to implement
- Whole structure less complicated

Cons:

- Abstract becomes a single paragraph

Q3

By using

```
set(AlzheimersList) & set(CancerList)
```

I find that there is no overlap in the two sets of papers that I identified.

Exercise 2

Q1

I compute the SPECTER embedding of each paper in EX1 as follows. See my code for detail.

```
In [51]: 1 embeddings = embedding_(papers)
100%|██████████| 2000/2000 [30:59<00:00,  1.08it/s]
```

Q2

I load the paper using the following code

```
f1 = open('Alzheimers.json')
f2 = open('Cancer.json')

Alzheimers = json.load(f1)
Cancer = json.load(f2)
```

Then I process using the following code

```
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 + get_abstract(paper)]
    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()]
```

Q3

Then I perform PCA using the following code

```
from sklearn import decomposition
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()]
```

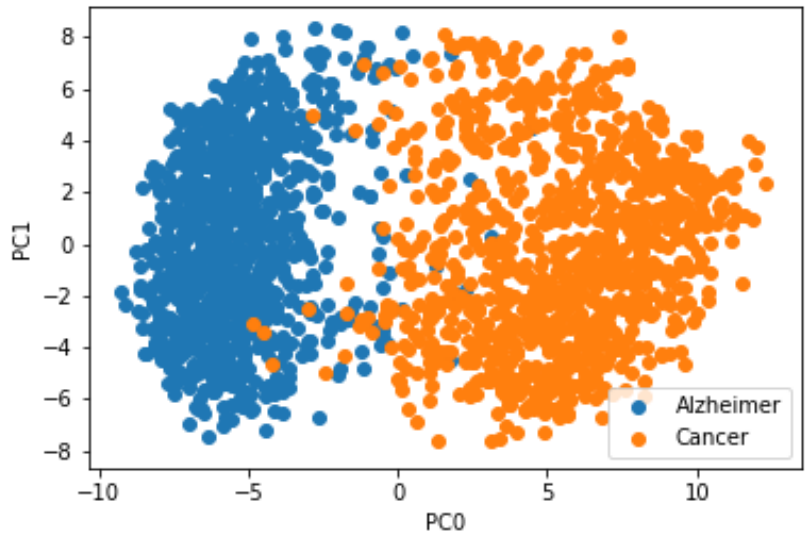
The result is

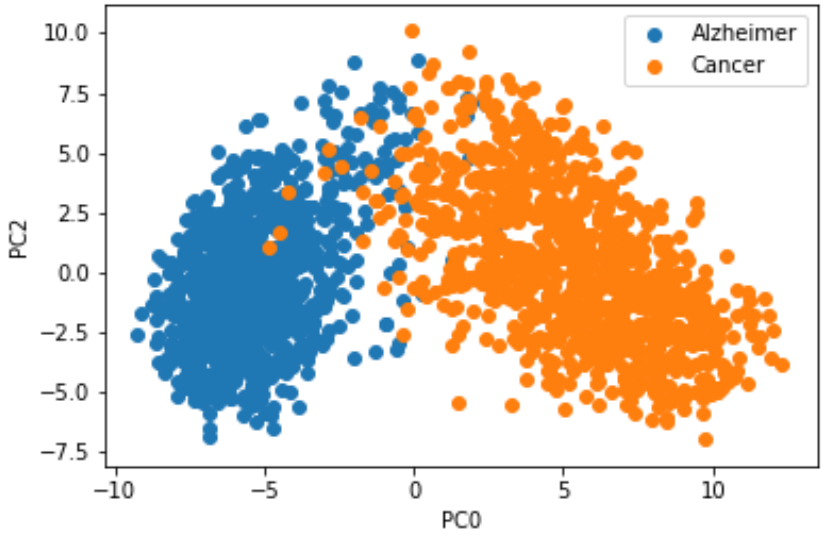
	PC0	PC1	PC2	query
0	-6.952025	2.151169	-4.292486	Alzheimer
1	-6.091127	-3.099970	0.176831	Alzheimer
2	-5.144713	-2.483867	0.643897	Alzheimer
3	-7.636140	-4.824004	2.221835	Alzheimer
4	-7.070122	-3.585453	-2.375718	Alzheimer
...	...	...	...	...
1995	4.881541	-2.351293	1.407387	Cancer
1996	1.630329	3.402599	6.384639	Cancer
1997	1.611991	-4.113490	2.973116	Cancer
1998	2.378570	-5.871854	3.056212	Cancer
1999	6.160980	-4.464087	-1.409446	Cancer

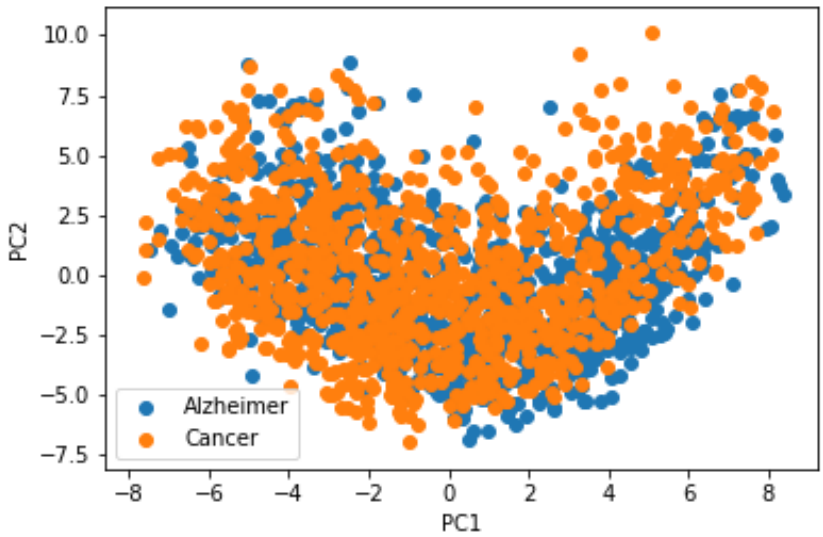
2000 rows × 4 columns

Q4

Finally, I plot three scatter plots for PC0 vs PC1, PC0 vs PC2, and PC1 vs PC2.


PC1
PC0
Alzheimer
Cancer


PC2
PC0
Alzheimer
Cancer


PC2
PC1
Alzheimer
Cancer

Comment: PC0 vs. PC1 performs the best among these three graphs because it uses the principle components with the biggest two eignevalues. The two class are separated pretty far away. PC0 vs. PC1 preform worse than PC0 vs. PC2, but better than PC1 vs. PC2. Finally, PC1 vs. PC2 is the worst, and we can hardly tell one class from another.

Exercise 3

Q1

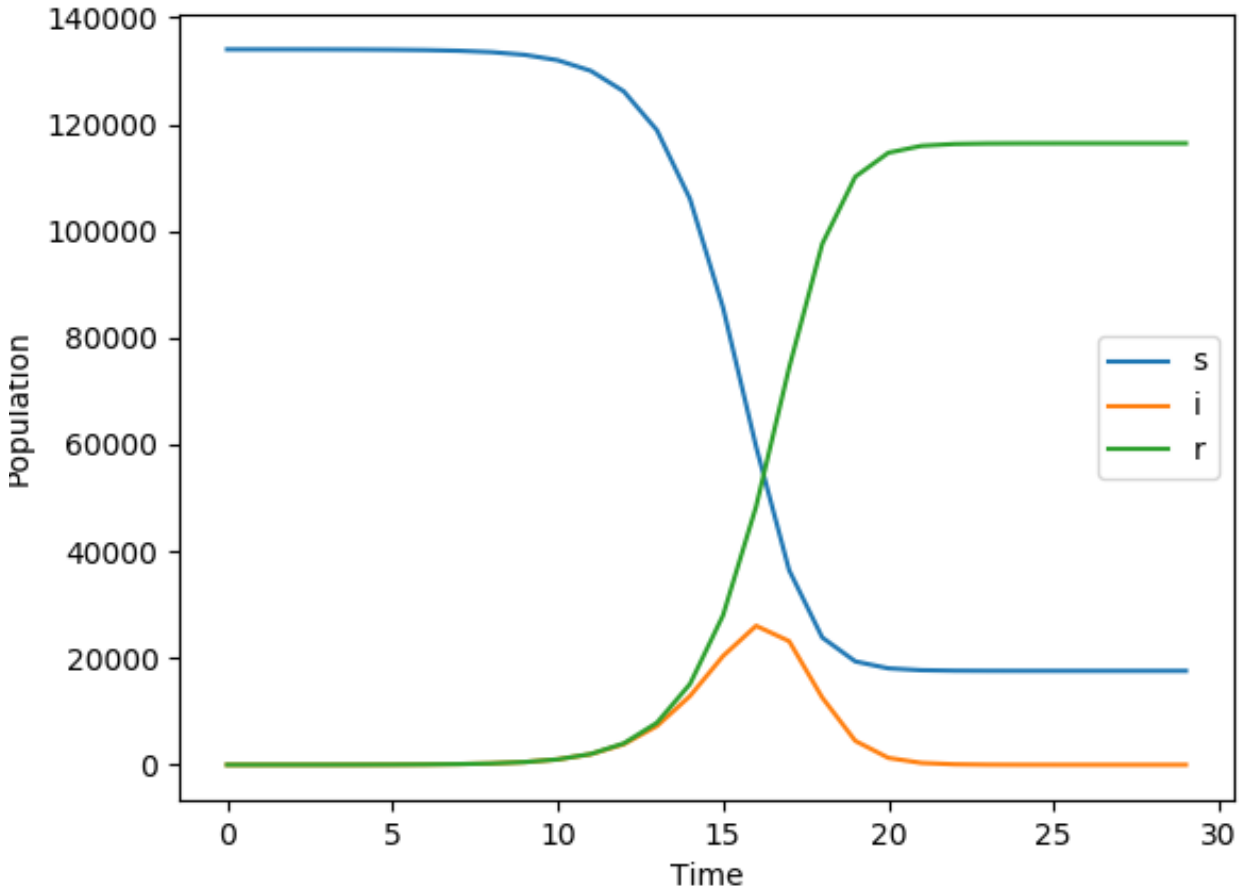
I write a class `SIR_Model()` to realize an Explicit Euler method to plot *i(t)*. There are several main functions inside the class.

- `__init__`: Set parameters of s, i, r, beta and gamma. The default values are s=133999, i=1, r=0, beta=2, gamma=1, Tmax=30.
- `update`: To update the value of s, i and r for one iteration.
- `evolve`: Evolve the population information to Tmax.
- `plot_graph`: plot the graph for t vs. population.

See my code for detailed implementation.

Q2

By these initial values, the time course of the number of infected individuals are plotted below.



Q3

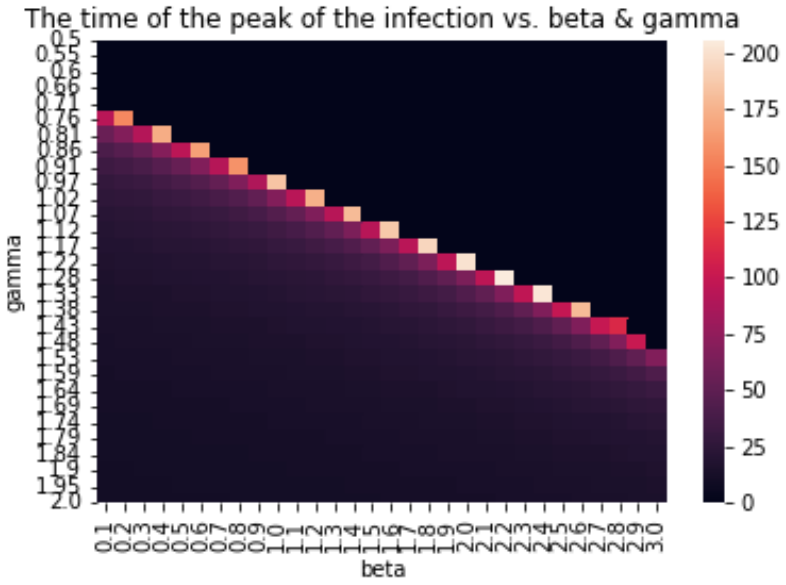
Two functions help me obtain that values.

- I write a function named `time_of_peak`. When t = 16, the number of infected people reaches its peak.
- I write another function named `iValue_of_peak`. 26033 people are infected at the peak.

See my code for detailed implementation.

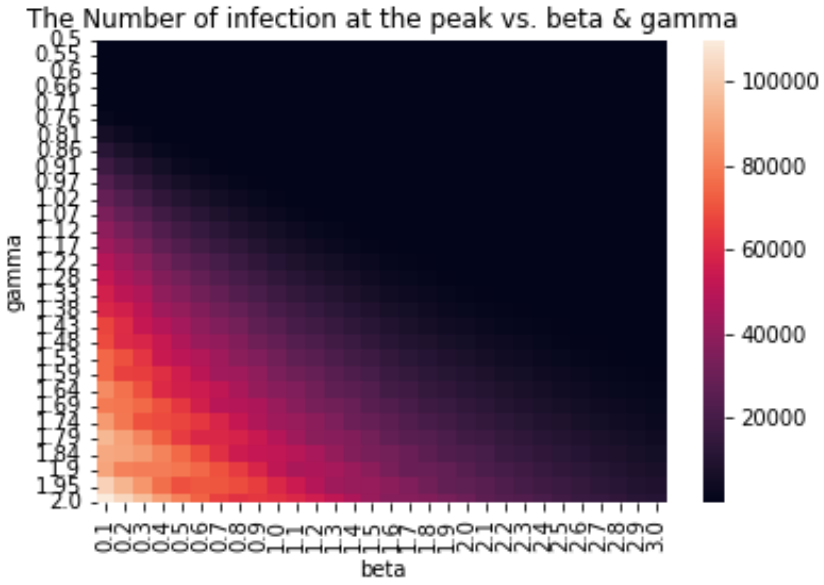
Q4

After that, I vary β and γ to be different values ranging from 0.1 to 3 and 0.5 to 2 respectively. Then I plot on a heat map how the time of the peak of the infection depends on these two variables. The result is shown below



Q5

I do the same for the number of individuals infected at peak. The result is shown below.



Exercise 4

Dataset Identification

The dataset I find: https://www.kaggle.com/datasets/harikrishnareddyb/used-car-price-predictions

It is "Used car price predictions".

Dataset Description

The data contains 8 columns, which means 8 variables. Some sample data are shown below.

Price	Year	Mileage	City	State	Vin	Make	Model
8995	2014	35725	El Paso	TX	19VDE2E53EE000083	Acura	ILX6-Speed
10888	2013	19606	Long Island City	NY	19VDE1F52DE012636	Acura	ILX5-Speed
8995	2013	48851	El Paso	TX	19VDE2E52DE000025	Acura	ILX6-Speed

Specifically,

- Price: Target Variable.
- Year: Year of the car purchased.
- Mileage: The no.of kms drove by the car.
- City: In which city it was sold.
- State: In which state it was sold.
- Vin: A unique number for a car.
- Make: Manufacturer of the car.
- Model: The model(name) of the car.

The key variables have been explicitly specified, which means we do not need to do further derive.

No variable is redundant.

Variable "State" can be predicted from variable "City".

There are 852123 data points available.

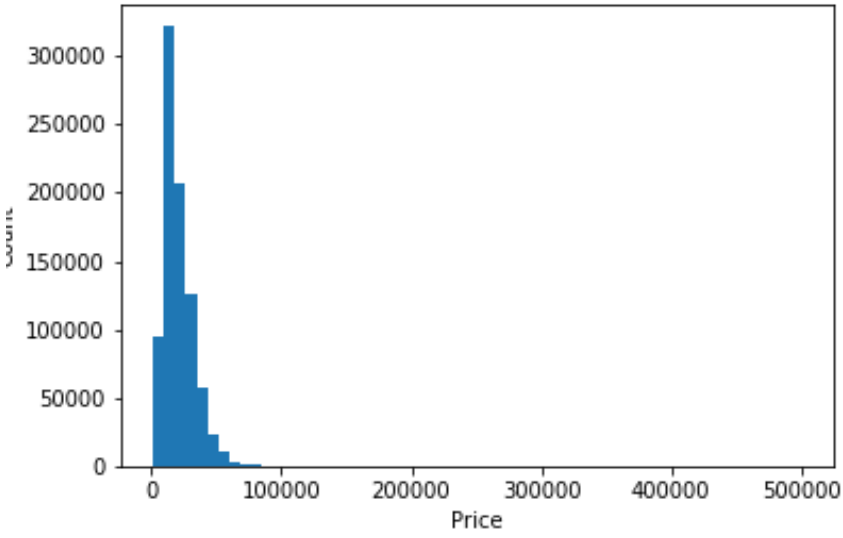
The data is in standard format.

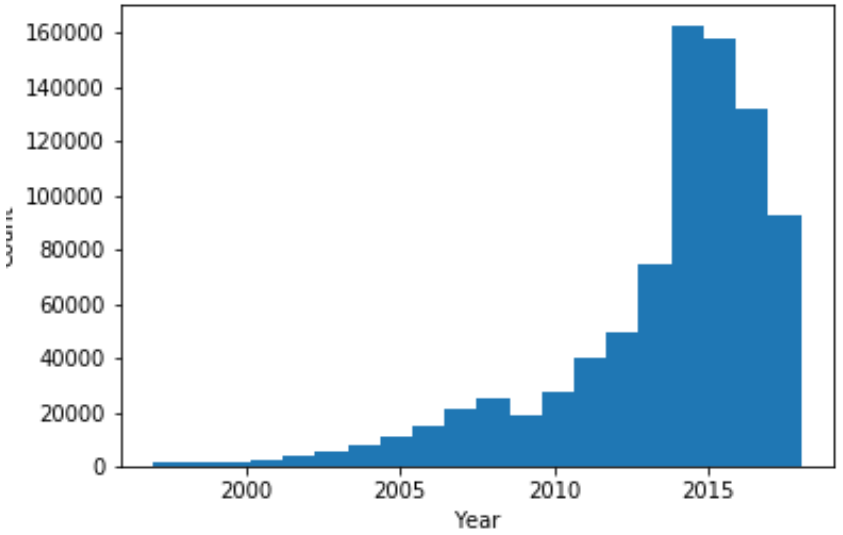
Terms of use & Restrictions

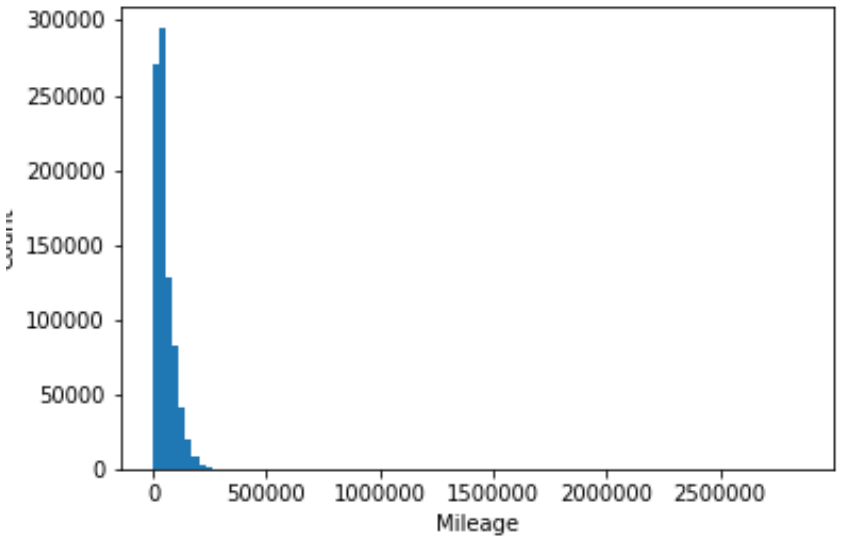
I do not have to officially apply to get access to the data. These does not exist certain type of analyses I can't do.

Data Exploration

I plot some histograms:







From these figures, I can observe that most cars have price lower than 100000. Most cars are purchased in 2014 or 2015. And the mileage tems to be less than 50000.

Data Cleaning Needs

There is no need to perform data cleaning. Because I have checked all the dataset and there is no missing data, which means the dataset has already been cleaned.

Appendix: Python Code

Exercise 1

```
from Bio import Entrez
import xml.dom.minidom as m
import requests
import json

def getid_from_term(num, term):
    r = requests.get(
        "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
        f"esearch.fcgi?db=pubmed&retmode=xml&retmax={num}&term={term}"
    )
    doc = m.parseString(r.text)
    IdLists = doc.getElementsByTagName("Id")
    IdList = [IdLists[i].childNodes[0].wholeText for i in range(num)]
    return IdList

AlzheimersList = getid_from_term(1000, 'Alzheimer+AND+2022[pdat]')
print(f"The IDs for 1000 Alzheimer papers from 2022 are {AlzheimersList}")

def getid_from_term(num, term):
    r = requests.get(
        "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
        f"esearch.fcgi?db=pubmed&retmode=xml&retmax={num}&term={term}"
    )
    doc = m.parseString(r.text)
    IdLists = doc.getElementsByTagName("Id")
    IdList = [IdLists[i].childNodes[0].wholeText for i in range(num)]
    return IdList

CancerList = getid_from_term(1000, 'Cancer+AND+2022[pdat]')
print(f"The IDs for 1000 Cancer papers from 2022 are {CancerList}")
```

```
def get_info(pmid, query):
    r = requests.get(
        "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"
        f"efetch.fcgi?db=pubmed&retmode=xml&id={pmid}"
    )
    doc = m.parseString(r.text)
```

```
    dict_articles = {}
    for i in doc.getElementsByTagName("PubmedArticle"):
        PMID = i.getElementsByTagName("PMID")[0].childNodes[0].wholeText
        title_i = i.getElementsByTagName("ArticleTitle")[0].childNodes # title
containing italics
        title = ""
        for item in title_i:
            title += item.toxml()
        try:
            abstracts = i.getElementsByTagName("Abstract")
[0].getElementsByTagName("AbstractText")
            abstract = ""
            for item in (abstracts[0].childNodes):
                abstract += item.toxml()
            # If there are more than one abstract, concancate with space between them
            if len(abstracts) > 1:
                for j in range(1,len(abstracts)):
                    abstract += " "
                    for item in (abstracts[j].childNodes):
                        abstract += item.toxml()
        except:
            abstract = ""
        dict_article = {"ArticleTitle": title,
                        "AbstractText": abstract,
                        "query": query}
        dict_articles[PMID] = dict_article
    return dict_articles

IDstr = ""
for item in AlzheimersList[:400]:
    IDstr += item
    IDstr += ","
IDstr = IDstr[:-1]
result = get_info(IDstr, "Alzheimer")

IDstr = ""
for item in AlzheimersList[400:800]:
    IDstr += item
    IDstr += ","
IDstr = IDstr[:-1]
temp = get_info(IDstr, "Alzheimer")
result.update(temp)

IDstr = ""
for item in AlzheimersList[800:]:
    IDstr += item
    IDstr += ","
IDstr = IDstr[:-1]
temp = get_info(IDstr, "Alzheimer")
```

```
result.update(temp)



with open("Alzheimers.json", "w") as outfile:
    json.dump(result, outfile)
```

```
IDstr = ""
for item in CancerList[:400]:
    IDstr += item
    IDstr += ","
IDstr = IDstr[:-1]
result = get_info(IDstr, "Cancer")

IDstr = ""
for item in CancerList[400:800]:
    IDstr += item
    IDstr += ","
IDstr = IDstr[:-1]
temp = get_info(IDstr, "Cancer")
result.update(temp)

IDstr = ""
for item in CancerList[800:]:
    IDstr += item
    IDstr += ","
IDstr = IDstr[:-1]
temp = get_info(IDstr, "Cancer")
result.update(temp)



with open("Cancer.json", "w") as outfile:
    json.dump(result, outfile)
```

Exercise 2

```
import json
import tqdm
import pandas as pd
import matplotlib.pyplot as plt

from transformers import AutoTokenizer, AutoModel

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

```
f1 = open('Alzheimers.json')
f2 = open('Cancer.json')

Alzheimers = json.load(f1)
Cancer = json.load(f2)
```

```
embeddings = embedding_(papers)
```

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

```
plt.scatter(embeddings_pca_all[embeddings_pca_all['query'] == 'Alzheimer']['PC0'],
embeddings_pca_all[embeddings_pca_all['query'] == 'Alzheimer']['PC1'],
label="Alzheimer")
plt.scatter(embeddings_pca_all[embeddings_pca_all['query'] == 'Cancer']['PC0'],
embeddings_pca_all[embeddings_pca_all['query'] == 'Cancer']['PC1'], label="Cancer")
plt.xlabel("PC0")
plt.ylabel("PC1")
plt.legend()
plt.show()
```

```
plt.scatter(embeddings_pca_all[embeddings_pca_all['query'] == 'Alzheimer']['PC0'],
embeddings_pca_all[embeddings_pca_all['query'] == 'Alzheimer']['PC2'],
label="Alzheimer")
plt.scatter(embeddings_pca_all[embeddings_pca_all['query'] == 'Cancer']['PC0'],
embeddings_pca_all[embeddings_pca_all['query'] == 'Cancer']['PC2'], label="Cancer")
plt.xlabel("PC0")
plt.ylabel("PC2")
plt.legend()
plt.show()
```

```
plt.scatter(embeddings_pca_all[embeddings_pca_all['query'] == 'Alzheimer']['PC1'],
embeddings_pca_all[embeddings_pca_all['query'] == 'Alzheimer']['PC2'],
label="Alzheimer")
plt.scatter(embeddings_pca_all[embeddings_pca_all['query'] == 'Cancer']['PC1'],
embeddings_pca_all[embeddings_pca_all['query'] == 'Cancer']['PC2'], label="Cancer")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()
```

Exercise 3

```
import matplotlib.pyplot as plt
import numpy as np
class SIR_Model():

    def __init__(self, s=133999, i=1, r=0, beta=2, gamma=1, Tmax=30):
        self._s = s
        self._i = i
        self._r = r
        self._beta = beta
        self._gamma = gamma
        self._Tmax = Tmax
        self._N = s + i + r

    def update(self):
        s_new = self._s - self._s*self._i*self._beta/self._N
        i_new = self._i + self._s*self._i*self._beta/self._N - self._gamma*self._i
        r_new = self._r + self._gamma*self._i
        self._s = s_new
        self._i = i_new
        self._r = r_new
        return self._s, self._i, self._r

    def evolve(self):
        s = [self._s,]
        i = [self._i,]
        r = [self._r,]
        t = range(self._Tmax)
        for time in range(self._Tmax-1):
            s_temp, i_temp, r_temp = self.update()
            s.append(s_temp)
            i.append(i_temp)
            r.append(r_temp)
        return s, i, r

    def time_of_peak(self):
        s, i, r = self.evolve()
        return np.argmax(i)

    def iValue_of_peak(self):
        s, i, r = self.evolve()
        return max(i)

    def plot_graph(self):
        s, i, r = self.evolve()
        t = range(self._Tmax)
        plt.plot(t,s,label='s')
```

```
        plt.plot(t,i,label='i')
        plt.plot(t,r,label='r')
        plt.xlabel("Time")
        plt.ylabel("Population")
        plt.legend()
        plt.savefig("README_img/EX3_1.png")
        plt.show()

sir = SIR_Model()
sir.plot_graph()
```

```
def heatplot_peak_of_time():
    x_lim = 30
    y_lim = 30
    peak_time = [[0]*x_lim for i in range(y_lim)]
    beta_x = np.linspace(0.1, 3, x_lim)
    gamma_y = np.linspace(0.5, 2, y_lim)
    for i in range(len(beta_x)):
        for j in range(len(gamma_y)):
            beta_x[i] = round(beta_x[i],2)
            gamma_y[i] = round(gamma_y[i],2)
            peak_time[i][j] = SIR_Model(beta=beta_x[i], gamma=gamma_y[j],
Tmax=1000).time_of_peak()
    sns.heatmap(peak_time, xticklabels=beta_x, yticklabels=gamma_y)
    plt.xlabel("beta")
    plt.ylabel("gamma")
    plt.title("The time of the peak of the infection vs. beta & gamma")
    plt.savefig("README_img/EX3_2.png")
    plt.show()

print(f"The number of infected people peak at t = {SIR_Model().time_of_peak()}, and
{round(SIR_Model().iValue_of_peak())} people are infected at the peak.")
heatplot_peak_of_time()
```

```
def heatplot_peak_of_number():
    x_lim = 30
    y_lim = 30
    peak_time = [[0]*x_lim for i in range(y_lim)]
    beta_x = np.linspace(0.1, 3, x_lim)
    gamma_y = np.linspace(0.5, 2, y_lim)
    for i in range(len(beta_x)):
        for j in range(len(gamma_y)):
            beta_x[i] = round(beta_x[i],2)
            gamma_y[i] = round(gamma_y[i],2)
            peak_time[i][j] = SIR_Model(beta=beta_x[i], gamma=gamma_y[j],
Tmax=1000).iValue_of_peak()
    sns.heatmap(peak_time, xticklabels=beta_x, yticklabels=gamma_y)
    plt.xlabel("beta")
```

```
    plt.ylabel("gamma")
    plt.title("The Number of infection at the peak vs. beta & gamma")
    plt.savefig("README_img/EX3_3.png")
    plt.show()

heatplot_peak_of_number()
```

Exercise 4

```
import pandas as pd
df = pd.read_csv("true_car_listings.csv")

import matplotlib.pyplot as plt
plt.hist(df['Price'], bins=60)
plt.xlabel("Price")
plt.ylabel("Count")
plt.savefig("README_img/EX4_1.png")
plt.show()

plt.hist(df['Year'], bins=20)
plt.xlabel("Year")
plt.ylabel("Count")
plt.savefig("README_img/EX4_2.png")
plt.show()

plt.hist(df['Mileage'], bins=100)
plt.xlabel("Mileage")
plt.ylabel("Count")
plt.savefig("README_img/EX4_3.png")
plt.show()
```