In [1]:

```
import pandas as pd
import os
import pyarrow
import numpy as np
import random
```

In [2]:

```
#os.listdir(os.getcwd()+'/negative')
```

In [5]:

```
curdir=os.getcwd()+'/negative'
curdir1=os.getcwd()+'/together2'
curdir2=os.getcwd()+'/together'
curdir3=os.getcwd()+'/working'
curdir9=os.getcwd()+'/positive'

blahs= [curdir,curdir1,curdir2,curdir3,curdir9]
```

In [ ]:

```
#This deletes the contents of all of the stuff in these folders
barf= 0
for k in blahs:
    for i,j in enumerate(os.listdir(k)):
        os.remove(os.path.join(k,j))
        barf+=1
        #print("removed:,", barf,"files")
```

What is the question I am trying to answer here?

Identify cancerous lung nodules using the Imaging Data Commons (IDC) National Lung Cancer Screening Trial (NLST) data for early detection and to prevent metastasis.

First steps Data Dictionary loading to help work with the files that are needed

In [ ]:

```
patients=pd.read_csv(os.getcwd()+'/people.csv')
```

From patient data dictionary we want the following variables for now\ `scr_res0-2` -Results of screening (for T0, T1, and T2 exams)\ `can_scr` -Result of screen associated with the first confirmed lung cancer diagnosis

The control group is probably where there was no cancer throuought the whole study at all. \

Looking through all of the items, we want

scr_res0-2 to be 1

Filtering by `patients[patients['scr_iso0']==1]` there are 20550 patients that have a negative screen. Filtering again two more times in order to make sure all 3 screens were negative leaves 11162 records.

In [ ]:

```
abc=patients[patients['scr_res0']==1]
```

```
defg=abc[abc['scr_res1']==1]
pqr=defg[defg['scr_res2']==1]
len(pqr)
```

Have a selection of patients Now to further clean the data set

In [ ]:

```
pqr['lesionsize'].notna().sum() #<- How many values are not na
```

In [ ]:

```
am=pqr[pqr['lesionsize'].notna()] #<- Select patients that are not NA values
ptf1=pqr[pqr['lesionsize'].isna()]
```

Going over all of the variables that might be important to leave out.\ `lochil` -Cancer in Left Hilum \ `loccar` -Cancer in Carina\ `loclin` -Cancer in Lingula\ `locllow` -Cancer in Left lower lobe\ `loclmsb` -Cancer in Left main stem bronchus\ `loclup` -Cancer in Left upper lobe\ `locmed` -Cancer in Mediastinum\ `locoth` -Cancer in Other Location\ `locrhil` -Cancer in Right Hilum\ `locrlow` -Cancer in Right lower lobe\ `locrmid` -Cancer in Right middle lobe\ `locrmsb` -Cancer in Right main stem bronchus\ `locrup` -Cancer in Right upper lobe\ `locunk` -Cancer in Unknown location

In [ ]:

```
pqr['loclhil'].notna().sum()
ptf1=pqr[pqr['lesionsize'].isna()]
```

In [ ]:

```
ptf1=ptf1[ptf1['loclhil'].isna()]
ptf1=ptf1[ptf1['loccar'].isna()]
ptf1=ptf1[ptf1['loclin'].isna()]
ptf1=ptf1[ptf1['locllow'].isna()]
ptf1=ptf1[ptf1['loclmsb'].isna()]
ptf1=ptf1[ptf1['loclup'].isna()]

ptf1=ptf1[ptf1['locmed'].isna()]
ptf1=ptf1[ptf1['locoth'].isna()]
ptf1=ptf1[ptf1['locrhil'].isna()]
ptf1=ptf1[ptf1['locrlow'].isna()]


ptf1=ptf1[ptf1['locrmid'].isna()]
ptf1=ptf1[ptf1['locrmsb'].isna()]
ptf1=ptf1[ptf1['locrup'].isna()]
ptf1=ptf1[ptf1['locunk'].isna()]
```

In [ ]:

```
print('There are still',len(ptf1),'samples')
```

In [ ]:

```
#values=ptf1['pid']

#with open("negative_group.txt", 'w') as output:
    #for row in values:
        #output.write(str(row) + ',')
```

Control group imaging is now downloaded.

Experimental group

In [ ]:

```
len(patients[patients['scr_iso0']!=1])
```

Many files to work with for the experimental. \ About 32,000 total but I need to include only those who are relevant.

1="Negative screen, no significant abnormalities" \ 2="Negative screen, minor abnormalities not suspicious for lung cancer"\ 3="Negative screen, significant abnormalities not suspicious for lung cancer"\ 4="Positive, Change Unspecified, nodule(s) >= 4 mm or enlarging nodule(s), mass(es), other non-specific abnormalities suspicious for lung cancer"\ 10="Inadequate Image"\ 11="Not Compliant - Left Study"\ 13="Not Expected - Cancer before screening window"\ 14="Not Expected - Death before screening window"\ 15="Not Compliant - Refused a screen"\ 17="Not Compliant - Wrong Screen"\ 23="Not Expected - Cancer in screening window"\ 24="Not Expected - Death in screening window"\ 95="Not Compliant - Erroneous Report of Lung Cancer Before Screen (LSS Only)"\ 97="Not Compliant - Form Not Submitted, Window Closed"

It looks like 4 is the category that fits best.

In [ ]:

```
len(patients[patients['scr_res0']==4])
```

The number is reduced to about 10,000.\ But when the results were further refined, there were very few experimental subjects.\ Instead Going to refine from res_0

In [ ]:

```
first_step=patients[patients['scr_res0']==4]
second_step=first_step[first_step['scr_res1']==4]
third_step=second_step[second_step['scr_res2']==4]
len(third_step)
```

In [ ]:

```
len(second_step)
```

In [ ]:

```
len(first_step)
```

Now to filter by results of the screen associated with the first confirmed diagnosis.

`can_scr` **Result of screen associated with the first confirmed lung cancer \ diagnosis Indicates whether the cancer followed a positive, negative, or missed screen, or whether it occurred after the screening years.\ 0="No Cancer"\ 1="Positive Screen"\ 2="Negative Screen"\ 3="Missed Screen"\ 4="Post Screening"**

Probably want a positive screen here, and the results are the same length (9758)

In [ ]:

```
first_step['can_scr']==1
```

Next by cancer stage

`de_stag` **combining clinical and pathologic staging information.\ .M="Missing"\ .N="Not Applicable"\ 110="Stage IA"\ 120="Stage IB"\ 210="Stage IIA"\ 220="Stage IIB"\ 310="Stage IIIA"\ 320="Stage IIIB"\ 400="Stage IV"\ 888="TNM not available"\ 900="Occult Carcinoma"\ 994="Carcinoid, cannot be assessed"\ 999="Unknown, cannot be assessed"**

In [ ]:

```
(len(first_step[first_step['de_stag']==110])+
len(first_step[first_step['de_stag']==120])+
```

```
len(first_step[first_step['de_stag']==210])+
len(first_step[first_step['de_stag']==220])+
len(first_step[first_step['de_stag']==310])+
len(first_step[first_step['de_stag']==320])+
len(first_step[first_step['de_stag']==400]))
```

Check for locations of cancers

In [ ]:

```
first_step=first_step[first_step['loclhil'].notna()]
first_step=first_step[first_step['loccar'].notna()]
first_step=first_step[first_step['loclin'].notna()]
first_step=first_step[first_step['locllow'].notna()]
first_step=first_step[first_step['loclmsb'].notna()]
first_step=first_step[first_step['loclup'].notna()]

first_step=first_step[first_step['locmed'].notna()]
first_step=first_step[first_step['locoth'].notna()]
first_step=first_step[first_step['locrhil'].notna()]
first_step=first_step[first_step['locrlow'].notna()]

first_step=first_step[first_step['locrmid'].notna()]
first_step=first_step[first_step['locrmsb'].notna()]
first_step=first_step[first_step['locrup'].notna()]
first_step=first_step[first_step['locunk'].notna()]
```

In [ ]:

```
'''values=first_step['pid']

with open("experimental_file.txt", 'w') as output:
    for row in values:
        output.write(str(row) + ',')'''
```

Looking at demographic information

In [ ]:

```
print("# of male participants:",len(first_step[first_step['gender']==1]),
      "# of female participants:",len(first_step[first_step['gender']==2]))
```

In [ ]:

```
import matplotlib.pyplot as plt
x=[1,2]
y=[498,354]
plt.bar(x,y)
plt.title('Number of male vs female participants')

plt.show()
```

In [ ]:

```
print('# of male participants:',len(ptf1[ptf1['gender']==1]),"# of female participants:",
len(ptf1[ptf1['gender']==2]))
```

In [ ]:

```
import matplotlib.pyplot as plt
x=[1,2]
y=[6430,4609]
plt.bar(x,y)
plt.title('Number of male vs female participants')

plt.show()
```

In [ ]:

```
m=patients['age'].values.tolist()
```

In [ ]:

```
m.sort()
```

In [ ]:

```
import seaborn as sns
sns.boxplot(m)
```

In [ ]:

```
sns.histplot(m,color='green').set(title='Distributions of ages for the study')
```

Now to choose patients for the cancer positive group

**By using the Abnormalities data set, specifically this column. I was able to obtain a list of slides that had possible cancerous growths. By using the abnormalities data set. I can filter by the patients who had positive nodules `sct_ab_desc` = 51 **

The information for the abnormalities data set is as follows:

`sct_ab_code`

.M="Missing"\ 51="Non-calcified nodule or mass (opacity >= 4 mm diameter)\ " 52="Non-calcified micronodule(s) (opacity < 4 mm diameter\ )" 53="Benign lung nodule(s) (benign calcificatio \ n)" 54="Atelectasis, segmental or grea \ ter" 55="Pleural thickening or effu \ sion" 56="Non-calcified hilar/mediastinal adenopathy or mass (>= 10 mm on short \ axis)" 57="Chest wall abnormality (bone destruction, metastasis\ etc.)" 58="Consol\idation" 59="E \mphysema" 60="Significant cardiovascular ab \normality" 61="Reticular/reticulonodular opacities, honeycombing, fibr \osis, scar" 62="6 or more nodules, not suspicious for cancer (opaci \ty >= 4 mm)" 63="Other potentially significant abnormality above t\he diaphragm" 64="Other potentially significant abnormality below \the diaphragm" 65="Other minor abn

51="Non-calcified nodule or mass (opacity >= 4 mm diameter)" <---

To me, an untrained person out of all of the other categories. This one says quietly, maybe cancer.ormality noted"

In [ ]:

```
abnormalities=pd.read_csv(os.getcwd()+'/nlst_780_ctab_idc_20210527.csv')
```

In [ ]:

```
larger_abnormlities=abnormalities[abnormalities['sct_ab_desc']==51]
```

In [ ]:

```
filtered_by_missing=larger_abnormlities[larger_abnormlities['sct_slice_num']!=999]
```

In [ ]:

```
filtered_by_missing=filtered_by_missing[filtered_by_missing['sct_slice_num'].notna()]
```

In [ ]:

```
#maybe merge by pid? there are images with abnormalities.

seta= set(filtered_by_missing['pid'])
setb= set(first_step['pid'])
```

In [ ]:

```
positive_patients=(seta.intersection(setb))
```

In [ ]:

```
positive_patients =pd.DataFrame(positive_patients)
```

In [ ]:

```
positive_patients
```

I downloaded the data in many ways.

I tried using the cloud services for both the Cancer imaging data commons, and Cancer Imaging Archive. But due to my own lack of skill I struggled heavily. I was not able to filter out images that were too small or too large despite my attempts. And what ended up hapenning is my file directory would get larger and larger due to composite files I did not need and became painfully slow quickly. So Instead I downloaded manually. From the Cancer imaging atlas.

In [ ]:

```
#values=positive_patients[0]

#with open("positive_patients1.csv", 'w') as output:
    #for row in values:
        #output.write(str(row) + ',')
```

In [ ]:

```
'''# formatting is not required, but makes queries easier to read!
query = """
SELECT
 *
FROM
  index
WHERE collection_id IN ('nlst')
"""
client.sql_query(query)'''
```

In [ ]:

```
'''from idc_index import index

client = index.IDCClient()

downloadDir = "C:\\Users\\amcfa\\Desktop\\Testing_download"

!rm -rf {downloadDir}
!mkdir -p {downloadDir}

selection_query = """
SELECT
  CONCAT('cp ',series_aws_url,' /content/idc_downloads') as cp_command
FROM
  index
WHERE collection_id IN ('nlst') and Modality ='CT'


"""
df = client.sql_query(selection_query)

with open('C:\\Users\\amcfa\\Desktop\\Testing_download\\download_manifest.txt', 'w') as f
:
  f.write('\n'.join(df['cp_command']))'''
```

In [ ]:

```
'''from idc_index import index
client = index.IDCClient()
```

```
# get identifiers of all collections available in IDC
all_collection_ids = client.get_collections()


for i in positive_patients:
 # download files for the specific collection, patient, study or series
 client.download_from_selection(patientId=str(i), \
  downloadDir="C:\\Users\\amcfa\\Desktop\\Testing_download")
 #for i in positive_patients:
  #client.download_from_selection(collection_id="'NLST'", patientId=str(i),\
         #downloadDir="D:\\NLST_2\\positive_images_final")'''
```

In [ ]:

```
"""from idc_index import index

client = index.IDCClient()


selection_query =

SELECT
*, CAST (instanceCount AS INT) AS INSTANCES
FROM
  index
WHERE collection_id IN ('nlst') and Modality ='CT' and BodyPartExamined IN ('CHEST') and
INSTANCES > 80 AND PatientID IN ('104592',
 '104683',
 '104778',
 '104815',
 '104999',
 '105142',
 '105144',
 '105165',
 '105340',
 '105352',
 '105514',
 '105540',
 '105543',
 '105562',
 '105742',
 '105767',
 '105771',
 '105796',
 '105974',
 '106000')



df = client.sql_query(selection_query) """
```

In [ ]:

```
""" downloadDir = os.getcwd()

!rm -rf {downloadDir}
!mkdir -p {downloadDir}

selection_query = '''
SELECT
  CONCAT('cp ',series_aws_url,' /content/idc_downloads') as cp_command,
  *, CAST (instanceCount AS INT) AS INSTANCES
FROM
  index

WHERE collection_id IN ('nlst') and Modality ='CT' and BodyPartExamined IN ('CHEST') and
INSTANCES > 80 and PatientID IN ('104592',
 '104683',
 '104778',
 '104815',
 '104999',
```

```
 '105142',
 '105144',
 '105165',
 '105340',
 '105352',
 '105514',
 '105540',
 '105543',
 '105562',
 '105742',
 '105767',
 '105771',
 '105796',
 '105974',
 '106000')

'''
df = client.sql_query(selection_query)

with open(os.getcwd()+  '\download_manifest.txt', 'w') as f:
  f.write('\n'.join(df['cp_command']))  """
```

In [ ]:

```
positive_patients=positive_patients[0]
```

In [ ]:

```
filtered_by_missing['pid']
```

In [ ]:

```
subset_df = filtered_by_missing[filtered_by_missing['pid'].isin(positive_patients)]
#We want # 51 which is correct
subset_df['sct_ab_desc'].unique()
```

`sct_ab_desc` **51="Non-calcified nodule or mass (opacity >= 4 mm diameter)"**

In [ ]:

```
#Now to simplify things I am going to replace `study_yr` 1,2,3 with T0,T1,T2
#subset_df['study_yr'] = subset_df['study_yr'].replace(0, 'T0')
#subset_df['study_yr'] = subset_df['study_yr'].replace(1, 'T1')
#subset_df['study_yr'] = subset_df['study_yr'].replace(2, 'T2')
#list_for_filtering =subset_df[['study_yr','sct_slice_num','pid']]
```

In [ ]:

```
subset_df=subset_df.replace({0: "T0", 1: "T1",2:"T2"})
```

In [ ]:

```
""" subset_df.loc[subset_df.study_yr ==0, 'study_yr'] = 'T0'
subset_df.loc[subset_df.study_yr ==1, 'study_yr'] = 'T1'
subset_df.loc[subset_df.study_yr ==2, 'study_yr'] = 'T2' """
```

In [ ]:

```
list1=['study_yr','sct_slice_num','pid']
list_for_filtering = subset_df[list1].copy()
```

In [ ]:

```
print(list_for_filtering['pid'])
```

In [ ]:

```
list_for_filtering['pid']=list_for_filtering['pid'].astype(str)
list_for_filtering['sct_slice_num']=list_for_filtering['sct_slice_num'].astype(int)
```

```
list_for_filtering['sct_slice_num']=list_for_filtering['sct_slice_num'].astype(str)
```

In [ ]:

```
m=list_for_filtering.values.tolist()
```

In [ ]:

```
#list_for_filtering.to_csv('list_for_filtering.csv',index=False,sep=',')
```

In [ ]:

```
#list_for_filtering=pd.read_csv('C:\\Users\\amcfa\\Desktop\\New folder (2)\\NLST\\list_for_filtering.csv')
```

In [ ]:

```
import sys
!{sys.executable} -m pip install pydicom
import pydicom
```

In [ ]:

```
os.getcwd()
```

In [ ]:

```
control_dir=os.path.join(os.getcwd()+"/fb6fa072-b31c-4e8d-907c-ae11b58b9578.dcm")
yeet=(pydicom.dcmread(control_dir,force=True))
```

Looking over these files. The most important things are slice numbers, Patient IDs and Clinical Trial Time Point ID.

In [ ]:

```
yeet
```

In [ ]:

```
list_a =[]  #  This code takes the dicom files and reads their data. Appending necessary information to a list

if hasattr(yeet, "SliceLocation") and yeet.SliceLocation:
 if "LOCALIZER" not in yeet.ImageType:
  tags = {'Clinical Trial Time Point ID':yeet[0x12,0x50].value,
        'Instance Number':yeet[0x20,0x13].value,
        'Patient ID ':yeet[0x10,0x20].value}

list_a.append([list(tags.values())][0])
```

In [ ]:

```
for i in m:
    if list_a[0]==i:
        print('yes')
```

The above fails it is not in the list .Testing if this works by forcing a good value.

In [ ]:

```
control_dir=os.path.join(os.getcwd()+"/Y_ffaf639d-1de8-46d8-b5a0-ecb61fb725c1.dcm")
yeet=(pydicom.dcmread(control_dir,force=True))
```

In [ ]:

```
list_a =[]  #  This code takes the dicom files and reads their data. Appending necessary information to a list
#I got this basic setup from the pydicom website. Thank you for the help! I had no idea t
```

```
hat dicom images could be indexed like that before. But I guess why
# wouldn't they be.
if hasattr(yeet, "SliceLocation") and yeet.SliceLocation:
 if "LOCALIZER" not in yeet.ImageType:
  tags = {'Clinical Trial Time Point ID':yeet[0x12,0x50].value,
        'Instance Number':yeet[0x20,0x13].value,
        'Patient ID ':yeet[0x10,0x20].value}

list_a.append([list(tags.values())][0])
```

In [ ]:

```
list_a[0]
```

In [ ]:

```
for i in m:
    if list_a[0]==i:
        print('yes, it is there')
```

So I need to match relevant information in each file. And then rename and move those files. Testing with just 1 small file. And it worked.

Maybe Don't uncomment the below code. unless you want random files everywhere.

So first going to process all of the data

and create some new directories. Just in case.

In [ ]:

```
import os
```

In [ ]:

```
os.getcwd()
```

In [ ]:

```
#os.chdir("/sbgenomics/project-files/")
```

In [ ]:

```
os.getcwd()
```

In [ ]:

```
#Finally Goes here
temp_dir="/sbgenomics/project-files/"
acessing_files1=os.listdir(temp_dir)
next_step=os.path.join(temp_dir+'/Agnes_McFarlin_Tier1_Submission')
os.listdir(next_step)
step_after =os.path.join(next_step+'/challenge_data')
os.listdir(step_after)
finally_ =os.path.join(step_after+'/Original_format_data')
os.listdir(finally_)
print(len(os.listdir(finally_)))
```

In [ ]:

```
finally_
```

In [ ]:

```
tmpy="/sbgenomics/workspace/"
os.listdir(tmpy)
```

In [ ]:

```
#This code makes a bunch of files for us. So we can work with the image files and do all
the stuff we need to.
control_dir=tmpy
newpath = control_dir +"//positive//"
if not os.path.exists(newpath):
    os.makedirs(newpath)
control_dir=tmpy
newpath = tmpy +"//working//"
if not os.path.exists(newpath):
    os.makedirs(newpath)
control_dir=tmpy
newpath = tmpy +"//negative//"
if not os.path.exists(newpath):
    os.makedirs(newpath)
newpath = control_dir +"//together//"
if not os.path.exists(newpath):
    os.makedirs(newpath)
newpath = control_dir +"//together2//"
if not os.path.exists(newpath):
    os.makedirs(newpath)
```

In [ ]:

```
x=500
```

In [ ]:

```
if x%100 ==0:
    print(x)
```

In [ ]:

```
finally_
```

In [ ]:

```
patients=pd.read_csv(os.getcwd()+"//people.csv")
abc=patients[patients['scr_res0']==1]
defg=abc[abc['scr_res1']==1]
pqr=defg[defg['scr_res2']==1]
ptf1=pqr[pqr['lesionsize'].isna()]
ptf1=ptf1[ptf1['loclhil'].isna()]
ptf1=ptf1[ptf1['loccar'].isna()]
ptf1=ptf1[ptf1['loclin'].isna()]
ptf1=ptf1[ptf1['locllow'].isna()]
ptf1=ptf1[ptf1['loclmsb'].isna()]
ptf1=ptf1[ptf1['loclup'].isna()]

ptf1=ptf1[ptf1['locmed'].isna()]
ptf1=ptf1[ptf1['locoth'].isna()]
ptf1=ptf1[ptf1['locrhil'].isna()]
ptf1=ptf1[ptf1['locrlow'].isna()]


ptf1=ptf1[ptf1['locrmid'].isna()]
ptf1=ptf1[ptf1['locrmsb'].isna()]
ptf1=ptf1[ptf1['locrup'].isna()]
ptf1=ptf1[ptf1['locunk'].isna()]
```

In [ ]:

```
abnormalities=pd.read_csv(os.getcwd()+"//nlst_780_ctab_idc_20210527.csv")
larger_abnormlities=abnormalities[abnormalities['sct_ab_desc']==51]
filtered_by_missing=larger_abnormlities[larger_abnormlities['sct_slice_num']!=999]
filtered_by_missing=filtered_by_missing[filtered_by_missing['sct_slice_num'].notna()]

seta= set(filtered_by_missing['pid'])
setb= set(ptf1['pid'])
```

In [ ]:

```
ors=(setb.difference(seta))
```

In [ ]:

```
setb.intersection(seta) #<- pretty sure this file is a mistake because you cant have.. po
sitive cancer and negative cancer at the same time?
```

In [ ]:

```
#This code Is kind of cool. I checked if it works above. And in my own notebooks.
#But what it does is it makes a set. And matches items to that set.
#So all I need are the lengths of said sets and that tells me
#if the match is correct.
# I tried doing it by creating lists and it... took a really long time.
#Because there are more negative files. So the list just builds and builds.
# this might work pretty well and faster for the testing files
#But what this should do is match all of the positive patient files. And then move them t
o a specific directory

control_dir=finally_
list_a=[]
long_list=[]
import os, random, shutil
for file in os.listdir(control_dir):  #<--- Get first directory

    abc=(os.path.join(control_dir, file)) #<- get the first file name
    #print(abc)
    yeet=(pydicom.dcmread(abc,force=True)) #<- reading those images
    if hasattr(yeet, "SliceLocation") and yeet.SliceLocation:
        if "LOCALIZER" not in yeet.ImageType:
            tags = {'Clinical Trial Time Point ID':yeet[0x12,0x50].value,
            'Instance Number':yeet[0x20,0x13].value,
            'Patient ID ':yeet[0x10,0x20].value}

            list_a.append([list(tags.values())][0])
            long_list.append([[list(tags.values())][0],file])
            x=len(long_list)
            if x % 100 ==0:
                print(x)
            if x == (len(os.listdir(finally_))):
                print("Done")
```

In [ ]:

```
yarf1=[]
for i,j in enumerate(m):
    yarf1.append(int(j[2]))
```

In [ ]:

```
argh5=[]
for i,j in enumerate(list_a):
    argh5.append(int(list_a[i][2]))
```

In [ ]:

```
a=set(argh5)
yarf2=set(yarf1)
```

In [ ]:

```
len(yarf2.intersection(seta))
```

In [ ]:

```
len(a.intersection(seta))
```

In [ ]:

```
len(a.intersection(yarf2))
```

In [ ]:

```
orsa=(a.difference(yarf2))
```

In [ ]:

```
pm=list(orsa)
```

In [ ]:

```
more_lists =[]
for i,j in enumerate(long_list):
    if int(long_list[i][0][2]) in pm:
        more_lists.append(long_list[i][1])
```

In [ ]:

```
len(more_lists)
```

In [ ]:

```
argh=[]
for i,j in enumerate(list_a):
    argh.append([j[0],int(j[1]),int(j[2])])
yarf=[]
for i,j in enumerate(m):
    yarf.append([j[0],int(j[1]),int(j[2])])
ats=(set(tuple(x) for x in argh))
r_tup=(set(tuple(x) for x in yarf))
mop=(ats.intersection(r_tup))

scarf=[]
for i,j in enumerate(long_list):
    scarf.append([long_list[i][0][0],int(long_list[i][0][1]),int(long_list[i][0][2]),lon
g_list[i][1]])
frame1=pd.DataFrame(scarf)
frame2=pd.DataFrame(mop)
a=frame1.merge(frame2)
a.sort_values(by=[2])
a=a.drop_duplicates(subset=[0,1,2])
framey1=a[3]
```

In [ ]:

```
for i,j in enumerate(framey1):
    file_again=os.path.join(finally_+"//"+j)
    moveto =tmpy
    src = finally_
    dst = tmpy +"//positive"
    try:
        shutil.copy(src+"//"+j,dst)
        X=(len(os.listdir(dst)))
        if X % 100 ==0:
            print(((X)/len(framey1))*100,"%there")
    except OSError:
        pass
```

In [ ]:

```
len(os.listdir(dst))
```

In [ ]:

```
""" barf= 0
for i,j in enumerate(os.listdir(dst)):
    if j not in b[3]:
        os.remove(os.path.join(dst,j))
```

```
        barf+=1
        print("removed:,", barf,"files") """
```

In [ ]:

```
import os
#Renamed all of the files
for filename in os.listdir(dst):
    os.rename(tmpy+"//positive//"+ filename,tmpy+"//positive//"+"_Y"+filename)
```

In [ ]:

```
tmpy
```

In [ ]:

```
"""Normally it could just be random but with so many files I chose a small subset to use"
""
source1=finally_
for file in os.listdir(finally_):
    counter = 0
    if file in more_lists:
        src = finally_
        dst = tmpy + "//negative//"
        try:
            shutil.copy(src+"//"+file,dst)
            counter +=1
            if counter == 1734:
                break
        except OSError:
            pass
```

In [ ]:

```
import os
#Renamed all of the files
for filename in os.listdir(dst):
    os.rename(tmpy+"//negative//"+ filename,tmpy+"//negative//"+"_X"+filename)
```

In [ ]:

```
for i,j in enumerate(m):
    if list_a[i]==j:
        print('yes')
```

Now that files are all renamed I need to convert them to jpg. Because DCM files are hard to work with

In [ ]:

```
!pip install dicom2jpg
```

In [ ]:

```
import dicom2jpg
```

In [ ]:

```
tmpy
```

In [ ]:

```
path1="positive/"
```

The below code would transfer all files to a new directory after transforming images to jpg

transforming images to jpg:

In [ ]:

```
#This code will move the files and delete un needed directories

control_dir=tmpy +path1
for file in os.listdir(control_dir):  #<--- Get first directory
    filename = os.fsdecode(file) #<-- decode file name
    #print(filename)
    abc=(os.path.join(control_dir, filename)) #<- get the first file name
    #print(abc)
    #qweh=(os.listdir(abc)) #<- Get the nested files within
    dicom2jpg.dicom2jpg(abc,target_root=os.getcwd()+"//together//",anonymous=False)
    rename_name=abc
    new_dir=(tmpy+"//together//")
    new_dir2=(tmpy+"//together2//")
    for files in (os.listdir(new_dir)):
        filenames = os.fsdecode(files)
        #print(filenames)
        abd=(os.path.join(new_dir, filenames))
        qwehl=(os.listdir(abd)) #<- even further
        for p,i in enumerate(qwehl):
            deml=(os.path.join(abd, i))
            #print(deml)
            qweh2=(os.listdir(deml))
            for j in qweh2:
                abq=(os.path.join(deml, j))
                #print(abq)
                qwehh=(os.listdir(abq))
                filenames2=qwehh
                for k in qwehh:
                    bbq=(os.path.join(abq, k))
                    os.rename(bbq,new_dir2+filename+".jpg")
                    os.rmdir(abq)
                    os.rmdir(deml)
                    os.rmdir(abd)
```

In [ ]:

```
path2="negative/"
```

In [ ]:

```
control_dir=tmpy +path2
for file in os.listdir(control_dir):  #<--- Get first directory
    filename = os.fsdecode(file) #<-- decode file name
    #print(filename)
    abc=(os.path.join(control_dir, filename)) #<- get the first file name
    #print(abc)
    #qweh=(os.listdir(abc)) #<- Get the nested files within
    dicom2jpg.dicom2jpg(abc,target_root=os.getcwd()+"//together//",anonymous=False)
    rename_name=abc
    new_dir=(tmpy+"//together//")
    new_dir2=(tmpy+"//together2//")
    for files in (os.listdir(new_dir)):
        filenames = os.fsdecode(files)
        #print(filenames)
        abd=(os.path.join(new_dir, filenames))
        qwehl=(os.listdir(abd)) #<- even further
        for p,i in enumerate(qwehl):
            deml=(os.path.join(abd, i))
            #print(deml)
            qweh2=(os.listdir(deml))
            for j in qweh2:
                abq=(os.path.join(deml, j))
                #print(abq)
                qwehh=(os.listdir(abq))
                filenames2=qwehh
                for k in qwehh:
                    bbq=(os.path.join(abq, k))
```

```
                    os.rename(bbq,new_dir2+filename+".jpg")
                    os.rmdir(abq)
                    os.rmdir(deml)
                    os.rmdir(abd)
```

In [ ]:

```
control_dir="/sbgenomics/output-files/"
```

In [ ]:

```
os.listdir(control_dir)
```

In [ ]:

```
control_dir="/sbgenomics/output-files/"
newpath = control_dir +"//train//"
if not os.path.exists(newpath):
    os.makedirs(newpath)

newpath = control_dir +"//val//"
if not os.path.exists(newpath):
    os.makedirs(newpath)

newpath = control_dir +"//test//"
if not os.path.exists(newpath):
    os.makedirs(newpath)
```

In [ ]:

```
n=int(len(os.listdir(tmpy+"//together2//"))*0.8)
m=n
rt=int(len(os.listdir(tmpy+"//together2//")))
pt=(int(rt*0.1))
```

In [ ]:

```
dups_no =[]
import os, random, shutil
source1=tmpy+"//together2//"
for i in range(0,(m)):
    random_file = random.choice(os.listdir(source1))
    abc=(os.path.join(source1, random_file))
    path = os.getcwd()
    moveto =control_dir +"//train//"
    src = source1
    dst = control_dir +"//train//"
    try:
        shutil.move(src+random_file,dst)

    except OSError:
        pass
```

In [ ]:

```
dups_no =[]
import os, random, shutil
source1=tmpy+"//together2//"
for i in range(0,(pt)):
    random_file = random.choice(os.listdir(source1))
    abc=(os.path.join(source1, random_file))
    path = os.getcwd()
    moveto =control_dir +"//val//"
    src = source1
    dst = control_dir +"//val//"
    try:
        shutil.move(src+random_file,dst)

    except OSError:
        pass
```

In [ ]:

```
dups_no =[]
import os, random, shutil
source1=tmpy+"//together2//"
for i in range(0,(pt)):
    random_file = random.choice(os.listdir(source1))
    abc=(os.path.join(source1, random_file))
    path = os.getcwd()
    moveto =control_dir +"//test//"
    src = source1
    dst = control_dir +"//test//"
    try:
        shutil.move(src+random_file,dst)

    except OSError:
        pass
```

In [ ]:

```
temp_dir="/sbgenomics/project-files/"
acessing_files1=os.listdir(temp_dir)
next_step=os.path.join(temp_dir+'/Agnes_McFarlin_Tier1_Submission')
os.listdir(next_step)
step_after =os.path.join(next_step+'/challenge_data')
os.listdir(step_after)
finally_ =os.path.join(step_after+'/Original_format_data')
os.listdir(finally_)
print(len(os.listdir(finally_)))
```

In [ ]:

```
temp_dir="/sbgenomics/output-files/"
acessing_files1=os.listdir(temp_dir)
print(acessing_files1)
```

In [ ]:

```
os.listdir(step_after)
```

In [ ]:

```
#long_lista.to_csv('long_lista.csv')
```

In [ ]:

```
## And that should output files into the training testing validation directories.
```

In [ ]:

```
!pip freeze > requirements.txt
```