cnn-xray-exploration Draft saved

File Edit View Run Add-ons Help

Share Save Version

Run All Code Draft Session Starting..

[2]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
from glob import glob
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
# List available files
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))


# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

+ Code + Markdown

Loading the data

[3]:
```
all_xray_df = pd.read_csv('/kaggle/input/data/Data_Entry_2017.csv')
all_image_paths = {os.path.basename(x): x for x in
                   glob(os.path.join('..', 'input', 'data','images*', 'images', '*.png'))}
print('Scans found:', len(all_image_paths), ', Total Headers', all_xray_df.shape[0])
all_xray_df['path'] = all_xray_df['Image Index'].map(all_image_paths.get)
all_xray_df.drop(['Unnamed: 11'], inplace=True, axis=1)
all_xray_df.sample(3)
```

```
Scans found: 112120 , Total Headers 112120
```

[3]:

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position	OriginalImage[Width	Height]	OriginalImagePixelSpacing[x	y]	path
102226	00027213_056.png	No Finding	56	27213	73	F	AP	3056	2544	0.139	0.139	../input/data/images_011/images/00027213_056.png
30844	00008038_000.png	AtelectasisEffusion	0	8038	39	F	PA	2500	2048	0.168	0.168	/input/data/images_004/images/00008038_000.png

Data Exploration

First, we will begin with some exploratory data analysis to better understand the structure of the data and the characteristics of the patients.

[4]:
```
#How many x-ray images are included in this dataset?
len(np.unique(all_xray_df["Image Index"]))
#there are 112,120 unique x-ray images
```

[4]:
```
112120
```

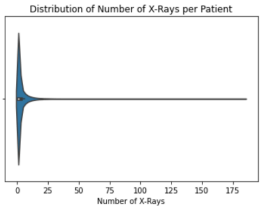
[5]:
```
#How many unique patients are represented?
total_patients = len(np.unique(all_xray_df["Patient ID"]))
print(total_patients)
#there are only 30,805 unique patients that have multiple images taken each
```

```
30805
```

[6]:
```
#what's the distribution of number of follow-ups/x-rays taken for each patient?
patient_xrays = all_xray_df.groupby(['Patient ID']).count()
#print(len(patient_xrays))
#patient_xrays.head()
```

+ Code + Markdown

[7]:
```
sns.violinplot(data = patient_xrays,
            x="Image Index").set(title='Distribution of Number of X-Rays per Patient',
                                 xlabel='Number of X-Rays');
```



+ Code + Markdown

[8]:
```
patient_xrays["Image Index"].describe()
```

[8]:
```
count    30805.000000
mean         3.639669
std          7.266734
min          1.000000
25%          1.000000
50%          1.000000
75%          3.000000
max        184.000000
Name: Image Index, dtype: float64
```

[9]:
```
#what's the percentage of patients with only 1 x-ray?
one_xray = len(patient_xrays[patient_xrays["Image Index"] == 1])
print("Patients with only 1 X-ray: ", (one_xray / total_patients)*100)

# >=5 x-rays?
five_or_less_xrays = len(patient_xrays[patient_xrays["Image Index"] <= 5])
print("Patients with 5 or fewer X-rays: ", (five_or_less_xrays / total_patients)*100)

# >=10 x-rays?
ten_or_less_xrays = len(patient_xrays[patient_xrays["Image Index"] <= 10])
print("Patients with 10 or fewer X-rays: ", (ten_or_less_xrays / total_patients)*100)

#Maximum number of X-rays?
print("Maximum number of X-rays for a patient: ", max(patient_xrays["Image Index"]))
```

```
Patients with only 1 X-ray:  56.81869826326895
Patients with 5 or fewer X-rays:  84.34994319104041
Patients with 10 or fewer X-rays:  92.77714656711574
Maximum number of X-rays for a patient:  184
```

+ Code + Markdown

The majority of patients only have one x-ray image (56.8%) and the vast majority have 5 or fewer (84.3%). Only about 7% of patients have more than 10 x-ray images included in their records here. However, as we can see, there are some outlier patients who have received dozens of x-ray images. The patient who has received the most x-rays has 184 images.

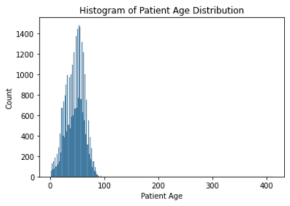
Although people come back for multiple treatments and x-rays, we want to visualize the patient's characteristics at baseline. We can assume that the vast majority will have the same sex at the beginning and end of treatment. A majority will also only have one data point with age (since only one X-ray for 56%). We will look at the distribution of changes in age later. First, let's examine demographic characteristics when they first receive an X-ray.

[10]:
```
baseline = all_xray_df.groupby("Patient ID").first()
baseline.head()
```

[10…

Patient ID	Image Index	Finding Labels	Follow-up #	Patient Age	Patient Gender	View Position	OriginalImage[Width	Height]	OriginalImagePixelSpacing[x	y]	path
1	00000001_000.png	Cardiomegaly	0	58	M	PA	2682	2749	0.143	0.143	../input/data/images_001/images/00000001_000.png
2	00000002_000.png	No Finding	0	81	M	PA	2500	2048	0.171	0.171	../input/data/images_001/images/00000002_000.png
3	00000003_000.png	Hernia	0	81	F	PA	2582	2991	0.143	0.143	../input/data/images_001/images/00000003_000.png
4	00000004_000.png	Mass|Nodule	0	82	M	AP	2500	2048	0.168	0.168	../input/data/images_001/images/00000004_000.png
5	00000005_000.png	No Finding	0	69	F	PA	2048	2500	0.168	0.168	../input/data/images_001/images/00000005_000.png

[12]:
```
sns.histplot(data=baseline, x="Patient Age").set(title = "Histogram of Patient Age Distribution");
```



There's some miscoded ages as no one can live to over 400 (yet). I will manually clean this by looking at the number of ages in the original dataset over 100 and re-code these depending on if there are other records present with their ages. Thankfully, it's very few errors given the size of this dataset.

[13]:
```
too_old = all_xray_df[all_xray_df["Patient Age"] > 100]["Patient ID"]

#thankfully only 16 patients to correct
print(len(too_old))

#original index, patient ID
print(too_old)

#to more easily find those people
old_list = too_old.tolist()
```

```
16
20852      5567
46965     11973
48284     12238
55742     13950
58650     14520
62929     15558
74884     18366
78795     19346
84810     20900
85404     21047
86264     21275
91369     22811
95794     25206
98495     26028
101194    26871
104590    27989
Name: Patient ID, dtype: int64
```

+ Code + Markdown

We then manually clean these records by looking at whether they have other X-rays that we can base their age off of. We select whatever age occured prior to this X-ray (or after if only later X-rays are present). If this is the only record for the person, we drop their record for the visualization (dropped n = 3).

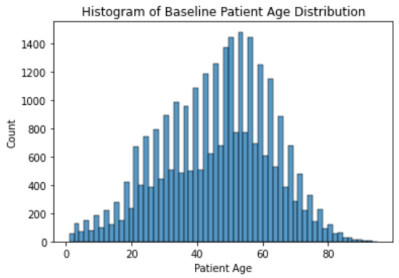
[ ]:
```
all_xray_df[all_xray_df["Patient ID"] == old_list[15]]
```

[14]:
```
all_xray_df.loc[20852, "Patient Age"] = 53
all_xray_df.loc[46965, "Patient Age"] = 58
all_xray_df.loc[48284, "Patient Age"] = 64
all_xray_df.loc[55742, "Patient Age"] = 64
all_xray_df.loc[58650, "Patient Age"] = 33
all_xray_df.loc[62929, "Patient Age"] = 46
all_xray_df.loc[74884, "Patient Age"] = 64
all_xray_df.loc[84810, "Patient Age"] = 72
all_xray_df.loc[85404, "Patient Age"] = 52
all_xray_df.loc[86264, "Patient Age"] = 22
all_xray_df.loc[91369, "Patient Age"] = 25
all_xray_df.loc[95794, "Patient Age"] = 36
all_xray_df.loc[98495, "Patient Age"] = 60

#drop - no age info
#only 3 patients
#create new dataframe to perserve these if we still want to classify
patient_age = all_xray_df.drop([78795, 101194, 104590])
```

[15]:
```
#we will re-do baseline with correct ages
baseline_age = patient_age.groupby("Patient ID").first()

#age summary statistics
baseline_age["Patient Age"].describe()
```

[15…]
```
count    30802.000000
mean        46.087559
std         16.692500
min          1.000000
25%         34.000000
50%         48.000000
75%         58.000000
max         95.000000
Name: Patient Age, dtype: float64
```

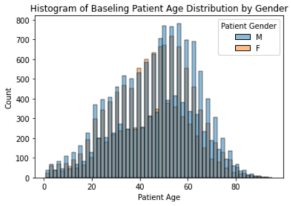
[16]:
```
#interesting that some ages are occuring more than the others right next to them
sns.histplot(data=baseline_age,
             x="Patient Age").set(title = "Histogram of Baseline Patient Age Distribution");
```



[17]:
```
#proportion male
#slightly more men in our sample than women
len(baseline_age[baseline_age["Patient Gender"] == "M"])/len(baseline_age["Patient Gender"])
```

[17…]
```
0.5398675410687618
```

[18]:
```
sns.histplot(data=baseline_age,
             x="Patient Age",
            hue = "Patient Gender").set(title = "Histogram of Baseling Patient Age Distribution by Gender");
```



The majority of patients tend to be in their mid-50s but there is a range of ages including patients who are children up to older adults. It's a little difficult to fully see the age distribution by sex. So we will now bin these ages into 10-year categories to better understand the age distribution by gender.

[19]:
```
baseline_age['Age Categories'] = np.where(
    (baseline_age['Patient Age'] >= 0) & (baseline_age['Patient Age'] <=9), "0 - 9", np.where(
    (baseline_age['Patient Age'] >= 10) & (baseline_age['Patient Age'] <=19), "10 - 19", np.where(
    (baseline_age['Patient Age'] >= 20) & (baseline_age['Patient Age'] <= 29), "20 - 29", np.where(
    (baseline_age['Patient Age'] >= 30) & (baseline_age['Patient Age'] <= 39), "30 - 39", np.where(
    (baseline_age['Patient Age'] >= 40) & (baseline_age['Patient Age'] <= 49), "40 - 49", np.where(
    (baseline_age['Patient Age'] >= 50) & (baseline_age['Patient Age'] <= 59), "50 - 59", np.where(
    (baseline_age['Patient Age'] >= 60) & (baseline_age['Patient Age'] <= 69), "60 - 69", np.where(
    (baseline_age['Patient Age'] >= 70) & (baseline_age['Patient Age'] <= 79), "70 - 79", np.where(
    (baseline_age['Patient Age'] >= 80) & (baseline_age['Patient Age'] <= 89), "80 - 89", np.where(
    (baseline_age['Patient Age'] >= 90) & (baseline_age['Patient Age'] <= 99), "90 - 99", 0)))))))))
```

+ Code + Markdown

[20]:
```
age_gender = baseline_age.groupby(['Age Categories', 'Patient Gender'])['Image Index'].count().unstack(level=-1)
age_gender["F_prop"] = age_gender["F"]/sum(age_gender["F"])
age_gender["M_prop"] = age_gender["M"]/sum(age_gender["F"])
age_gender
```

[20…

Patient Gender	F	M	F_prop	M_prop
Age Categories				
0 - 9	248	309	0.017498	0.021802
10 - 19	615	777	0.043392	0.054823
20 - 29	1725	1928	0.121710	0.136033
30 - 39	2354	2475	0.166090	0.174628
40 - 49	3095	3102	0.218373	0.218867
50 - 59	3377	3847	0.238270	0.271432
60 - 69	1984	2873	0.139984	0.202709
70 - 79	641	1129	0.045227	0.079659
80 - 89	127	182	0.008961	0.012841
90 - 99	7	7	0.000494	0.000494

[21]:
```
#40-49
print("Proportion of 40-49 year olds", (3095 + 3102) / (sum(age_gender["F"]) + sum(age_gender["M"])))

#50-59 age group
print("Proportion of 50 - 59 year olds", (3377 + 3847) / (sum(age_gender["F"]) + sum(age_gender["M"])))
```

```
Proportion of 40-49 year olds 0.2011882345302253
Proportion of 50 - 59 year olds 0.2345302253100448
```

[22]:
```
# plot the proportion of age/sex
plt.barh(age_gender.index, age_gender["F_prop"], color = 'red')
plt.barh(age_gender.index, -age_gender["M_prop"], color = 'blue')
plt.title("Population Pyramid: Proportion of Age Groups by Sex at Baseline")
plt.legend(["Female", "Male"])
plt.show()
```

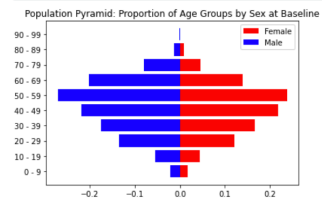


Image Characteristics

X-rays can be taken in two positions - Chest Posterior Anterior (PA) and Anterior Posterior (AP). PA is considered the "gold standard" but requires patients to be able to stand while the X-ray is taken, which may not be possible if people are experiencing extreme illnesses. We discussed the consequences of x-ray positioning in class in regards to COVID-19 x-ray image classification and outcome prediction. The algorithms were primarily learning the positioning of the person, rather than any other features of the images. However, this positioning is an indicator of outcome but perhaps doesn't add any novel information. We can look at the distribution of positions in our dataset and whether the variation in diseases differs.

[23]:
```
all_xray_df["View Position"].describe()
```

[23…
```
count     112120
unique         2
top           PA
freq       67310
Name: View Position, dtype: object
```

[24]:
```
print("Proportion of PA chest x-rays: ", len(all_xray_df[all_xray_df["View Position"] == "PA"]) / len(all_xray_df))
```

```
Proportion of PA chest x-rays:  0.6003389225829469
```

+ Code + Markdown

[25]:
```
all_xray_df['Healthy Indicator'] = np.where(
    all_xray_df['Finding Labels'] == "No Finding", 1, 0)
```

[26]:
```
pd.crosstab(all_xray_df['View Position'],
                            all_xray_df['Healthy Indicator'],
                                margins = False)
```

[26…

Healthy Indicator	0	1
View Position		
AP	23751	21059
PA	28008	39302

[27]:
```
print("The odds ratio is: ", (23751*39302)/(21059*28008))
```

```
The odds ratio is:  1.58262021464728
```

The odds of being diagnosed with any disease is 1.58 times higher for those patients that recieved a chest x-ray laying down (AP) as opposed to those that could stand up (PA).

[28]:
```
from scipy import stats

data = [
    #Unhealthy, Healthy
    [23751, 21059], #AP
    [39302, 28008] #PA
]
chi2_statistic, p_value, dof, ex = stats.chi2_contingency(data)

print("The Chi-squared statistic is: ", chi2_statistic)
print("Its accompanying p-value is: ", p_value)

#Given the very large Chi2 and very small p-value
#We reject the null hypothesis that the positioning and disease status are independent
```

```
The Chi-squared statistic is:  316.83596528203975
Its accompanying p-value is:  7.081203196565722e-71
```

[29]:
```
#are there differences in status by men/women?
pd.crosstab(all_xray_df['Patient Gender'],
                            all_xray_df['Healthy Indicator'],
                                margins = False)
```

[29…

Healthy Indicator	0	1
Patient Gender		
F	22341	26439
M	29418	33922

+ Code + Markdown

[30]:
```
print("The odds ratio is: ", (22341*33922)/(26439*29418))
#women are just slightly healthier than men
#probably because of the age distribution
#and that there are just more men in the sample (likely because they are sicker)
```

```
The odds ratio is:  0.9743744556495564
```

[31]:
```
data = [
    #Unhealthy, Healthy
    [22341, 26439], #Women
    [29418, 33922] #Men
]
chi2_statistic, p_value, dof, ex = stats.chi2_contingency(data)

print("The Chi-squared statistic is: ", chi2_statistic)
print("Its accompanying p-value is: ", p_value)

#Given the Chi2 and small p-value
#We reject the null hypothesis that sex and disease status are independent
```

```
The Chi-squared statistic is:  4.588143089352502
Its accompanying p-value is:  0.03219387481568274
```

Women are slightly healthier than men in this sample. This is probably due to there being more men in the sample to begin with and they have more x-rays (indicating they are probably unhealthier).

Let's look at the coefficients of a simple logistic regression to see the effect of these demographic and structural factors on health status.

What's the effect of sex, age, and x-ray positioning on whether a patient's x-ray is healthy/unhealthy

[32]:
```
#first convert gender to binary
all_xray_df['Male'] = np.where(
    all_xray_df['Patient Gender'] == "M", 1, 0)

#convert PA/AP to binary, want PA as reference category
all_xray_df['AP'] = np.where(
    all_xray_df['View Position'] == "AP", 1, 0)
```

[33]:

```
#this is mainly just an exploratory analysis
#although c-v and train/test split is ideal
#this just allows for basic information about the associations

import statsmodels.api as sm

y = all_xray_df["Healthy Indicator"]
log_dependents = ["Male", "Patient Age", "AP"]
x = all_xray_df[log_dependents]

logit_model = sm.Logit(y, x).fit()
print(logit_model.summary())
#all these variables have significant effects on whether a patient's x-ray is healthy/unhealthy
```

```
Optimization terminated successfully.
         Current function value: 0.687920
         Iterations 4
                           Logit Regression Results
==============================================================================
Dep. Variable:      Healthy Indicator   No. Observations:               112120
Model:                          Logit   Df Residuals:                   112117
Method:                           MLE   Df Model:                            2
Date:                Wed, 01 Dec 2021   Pseudo R-squ.:                0.003304
Time:                        23:30:00   Log-Likelihood:                -77130.
converged:                       True   LL-Null:                       -77385.
Covariance Type:            nonrobust   LLR p-value:                8.779e-112
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Male           0.1484      0.011     12.925      0.000       0.126       0.171
Patient Age    0.0029      0.000     15.426      0.000       0.003       0.003
AP            -0.3402      0.012    -28.743      0.000      -0.363      -0.317
==============================================================================
```

[34]:

```
#for fun
#let's see what the accuracy is like for a simple logistic regression to predict status
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import recall_score

scoring = ['accuracy', 'f1', 'roc_auc']
logreg = LogisticRegression()
scores = cross_validate(logreg, x, y, scoring=scoring)
sorted(scores.keys())
print("Mean test accuracy: ", scores['test_accuracy'].mean())
print("Mean test F1 Score: ", scores['test_f1'].mean())
print("Mean test AUC: ", scores['test_roc_auc'].mean())

#basically random guessing
#these are influential variables as we saw with coefficients
#but don't meaningfully explain variation
```

```
Mean test accuracy:  0.5584819835890118
Mean test F1 Score:  0.6425512923279413
Mean test AUC:  0.5810037317936233
```

X-ray Labels

The x-rays are labeled with either "No Finding" or 14 disease labels (15 classes total). The diseases include: Atelectasis, Cardiomegaly, Consolidation, Edema, Effusion, Emphysema, Fibrosis, Hernia, Infiltration, Mass, Nodule, Pleural_Thickening, Pneumonia, & Pneumothorax. The disease labels can include one label up to potentially all of them. We want to look at what diseases are occuring and when/what they co-occur with.

```
#one-hot encoding to more easily see which diseases occur with each other

one_hot_df = all_xray_df

#if it's normal, there's only one finding
#all_xray_df['Finding Labels'] = all_xray_df['Finding Labels'].map(lambda x: x.replace('No Finding', ''))

from itertools import chain
all_labels = np.unique(list(chain(*one_hot_df['Finding Labels'].map(lambda x: x.split('|')).tolist())))
all_labels = [x for x in all_labels if len(x)>0]
print('All Labels ({}): {}'.format(len(all_labels), all_labels))
for c_label in all_labels:
    if len(c_label)>1: # leave out empty labels
        one_hot_df[c_label] = one_hot_df['Finding Labels'].map(lambda finding: 1.0 if c_label in finding else 0)
one_hot_df.sample(5)
```

[36]:

```
#this gives us the total occurence of diseases even if they co-occur
total_occur = [['Atelectasis', sum(one_hot_df["Atelectasis"])],
                            ['Cardiomegaly', sum(one_hot_df["Cardiomegaly"])],
                            ['Consolidation', sum(one_hot_df["Consolidation"])],
                            ['Edema' , sum(one_hot_df["Edema"])],
                            ['Edema' , sum(one_hot_df["Edema"])],
                            ['Effusion' , sum(one_hot_df["Effusion"])],
                            ['Emphysema' , sum(one_hot_df["Emphysema"])],
                            ['Fibrosis' , sum(one_hot_df["Fibrosis"])],
                            ['Hernia' , sum(one_hot_df["Hernia"])],
                            ['Infiltration' , sum(one_hot_df["Infiltration"])],
                            ['Mass', sum(one_hot_df["Mass"])],
                            ['No Finding' , sum(one_hot_df["No Finding"])],
                            ['Nodule' , sum(one_hot_df["Nodule"])],
                            ['Pleural_Thickening' , sum(one_hot_df["Pleural_Thickening"])],
                            ['Pneumonia', sum(one_hot_df["Pneumonia"])],
                            ['Pneumothorax' , sum(one_hot_df["Pneumothorax"])]]

disease_occur = pd.DataFrame(total_occur, columns = ["Disease", "Count"])
```

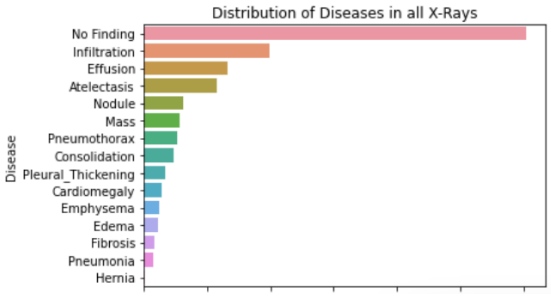
[37]:

```
disease_sorted = disease_occur.sort_values(by=['Count'], ascending = False)
disease_sorted
```

[37…

	Disease	Count
11	No Finding	60361.0
9	Infiltration	19894.0
5	Effusion	13317.0
0	Atelectasis	11559.0
12	Nodule	6331.0
10	Mass	5782.0
15	Pneumothorax	5302.0
2	Consolidation	4667.0
13	Pleural_Thickening	3385.0
1	Cardiomegaly	2776.0
6	Emphysema	2516.0
3	Edema	2303.0
4	Edema	2303.0
7	Fibrosis	1686.0
14	Pneumonia	1431.0
8	Hernia	227.0

[38]:
```
sns.barplot(y="Disease", x="Count", data=disease_sorted).set(title='Distribution of Diseases in all X-Rays');
```



Now we will see if we can create an adjacency matrix where the row and column values represent the number of times the labels appear together in an X-ray.

+ Code + Markdown

[39]:
```
disease_labels = ['Atelectasis',
                  'Cardiomegaly',
                  'Consolidation',
                  'Edema',
                  'Effusion',
                  'Emphysema',
                  'Fibrosis',
                  'Hernia',
                  'Infiltration',
                  'Mass',
                  'No Finding',
                  'Nodule',
                  'Pleural_Thickening',
                  'Pneumonia',
                  'Pneumothorax']
```

[40]:
```
label_df = one_hot_df[disease_labels]
```

[42]:
```
adjacency_df = label_df.T.dot(label_df)
```

+ Code + Markdown

[43]:
```
adjacency_df
```

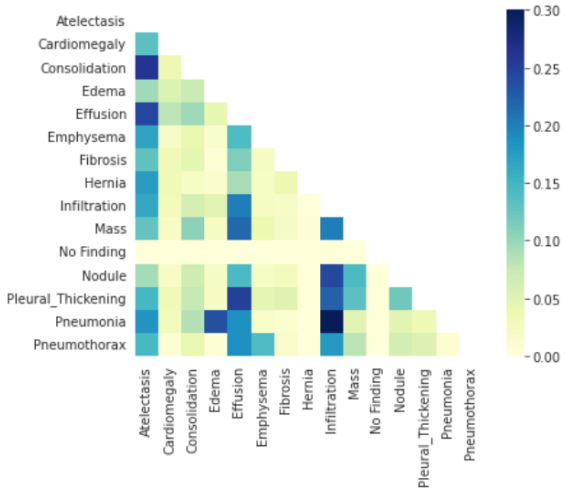
+ Code + Markdown

Now we know t Add a markdown text cell occurs in the dataset. We can now scale based on the number of times the label appears (rows divided by the diagonal of the column).

[44]:
```
adjacency_diagonal = np.diagonal(adjacency_df)
```

[45]:
```
adjacency_percentage = np.true_divide(adjacency_df, adjacency_diagonal[:, None])
```

[47]:
```
#proportion of co-occurrence of labels
mask = np.zeros_like(adjacency_percentage)

mask[np.triu_indices_from(mask)] = True

with sns.axes_style("white"):

    f, ax = plt.subplots(figsize=(7, 5))

    ax = sns.heatmap(adjacency_percentage, mask=mask, vmax=.3, square=True, cmap = "YlGnBu")
```



+ Code + Markdown

Now let's look at the distribution of each disease label by age and sex.

[50]:

```
age_disease_list = []

for l in disease_labels:
    age_count_df = one_hot_df.groupby(["Age Categories"])[l].sum()/one_hot_df[l].sum()
    age_disease_list.append(age_count_df)
```

[ ]:

```
#age_disease_list
```

+ Code + Markdown

```
fig , axs = plt.subplots(5, 3, figsize=(10,10))

fig.suptitle('Proportion of Disease Occurence by Age Categories')

counter = 0
for i in range(5):
    for j in range(3):
        axs[i, j].bar(x=age_disease_list[counter].index,
            height=age_disease_list[counter], color = "blueviolet")
        axs[i, j].set_title(disease_labels[counter])
        counter +=1

fig.tight_layout(pad=1.0)

for ax in fig.get_axes():
    ax.tick_params(labelrotation=45)
    ax.label_outer()
```