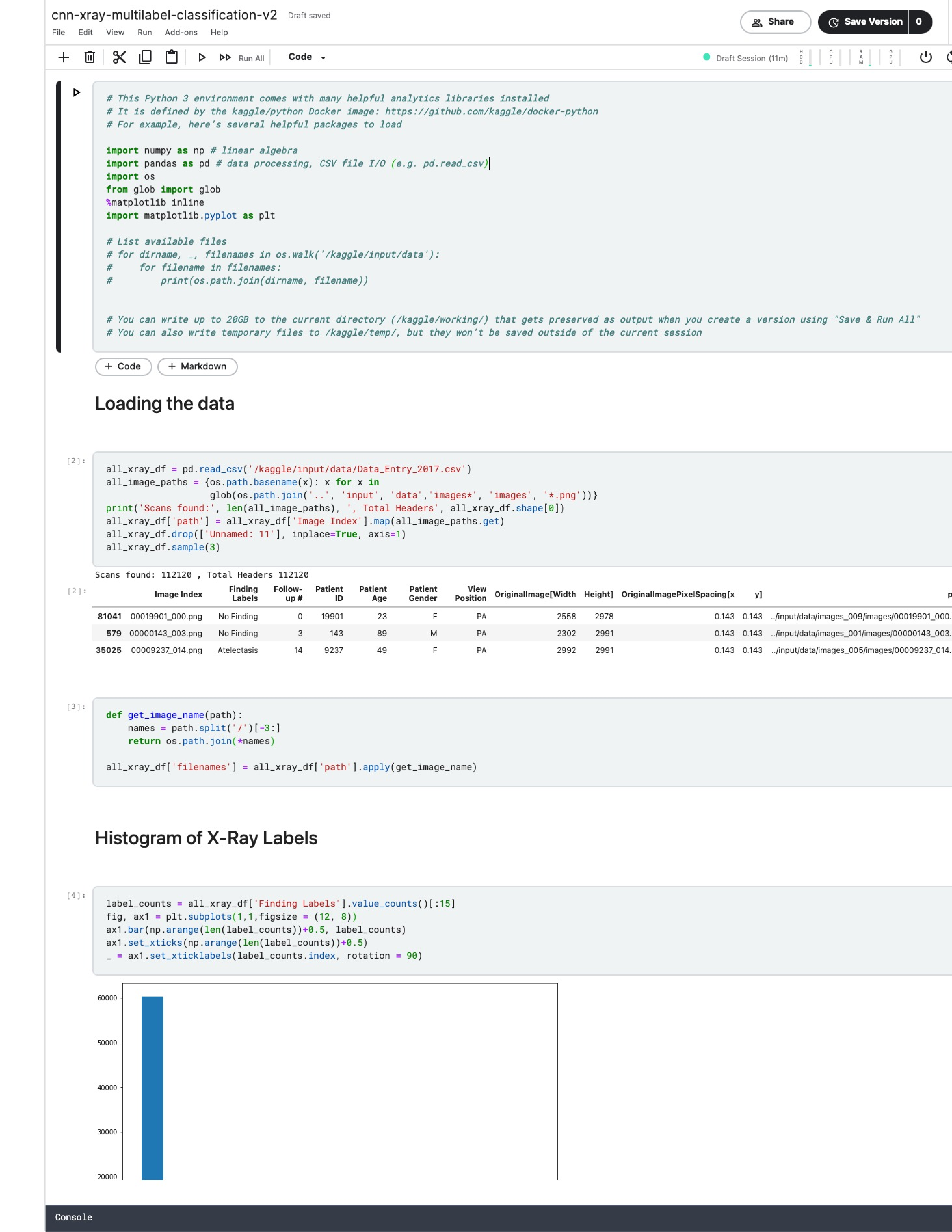


```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

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

# List available files
# for dirname, _, filenames in os.walk('/kaggle/input/data'):
#     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
```

Loading the data

[2]:

```
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
```

[2]:

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position	OriginalImage[Width	Height]	OriginalImagePixelSpacing[x	y]	p
81041	00019901_000.png	No Finding	0	19901	23	F	PA	2558	2978	0.143	0.143	../input/data/images_009/images/00019901_000.
579	00000143_003.png	No Finding	3	143	89	M	PA	2302	2991	0.143	0.143	../input/data/images_001/images/00000143_003.
35025	00009237_014.png	Atelectasis	14	9237	49	F	PA	2992	2991	0.143	0.143	../input/data/images_005/images/00009237_014.

[3]:

```
def get_image_name(path):
    names = path.split('/')[-3:]
    return os.path.join(*names)

all_xray_df['filenames'] = all_xray_df['path'].apply(get_image_name)
```

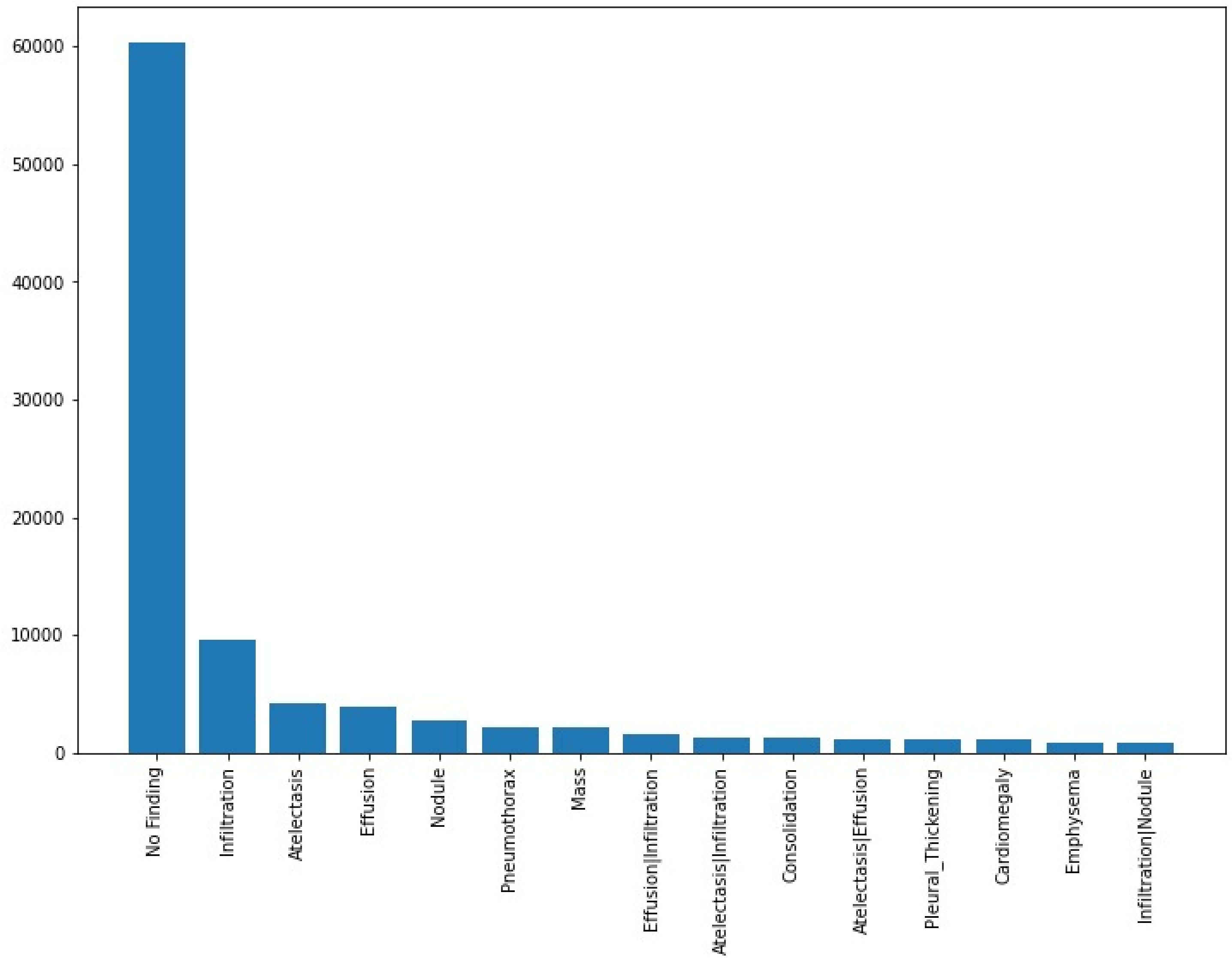
Histogram of X-Ray Labels

[4]:

```
label_counts = all_xray_df['Finding Labels'].value_counts()[:15]
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
_ = ax1.set_xticklabels(label_counts.index, rotation = 90)
```

Histogram of X-Ray Labels

[4]:
```
label_counts = all_xray_df['Finding Labels'].value_counts()[:15]
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
_ = ax1.set_xticklabels(label_counts.index, rotation = 90)
```



+ Code + Markdown

Encoding labels for multi-label classification

One image can have multiple disease labels, for such images more than one of the encoded variables can be 1, and the rest will be 0 as in one-hot encoding.

[5]:
```
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(*all_xray_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
        all_xray_df[c_label] = all_xray_df['Finding Labels'].map(lambda finding: 1.0 if c_label in finding else 0)
all_xray_df.sample(3)
```

```
All Labels (14): ['Atelectasis', 'Cardiomegaly', 'Consolidation', 'Edema', 'Effusion', 'Emphysema', 'Fibrosis', 'Hernia', 'Infiltration', 'Mass', 'Nodule', 'Pleural_Thic
g', 'Pneumonia', 'Pneumothorax']
```

[5]:

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position	OriginalImage[Width	Height]	OriginalImagePixelSpacing[x	...	Effusion	Emphysema	Fibrosis	Hernia	Infiltration
15361	00004007_001.png		1	4007	37	F	PA	2544	3056	0.139000	...	0.0	0.0	0.0	0.0	0.0
100360	00026597_008.png	Effusion|Infiltration	8	26597	45	M	PA	2021	2021	0.194311	...	1.0	0.0	0.0	0.0	1.0
42548	00010953_003.png	Effusion	3	10953	57	F	PA	2566	2991	0.143000	...	1.0	0.0	0.0	0.0	0.0

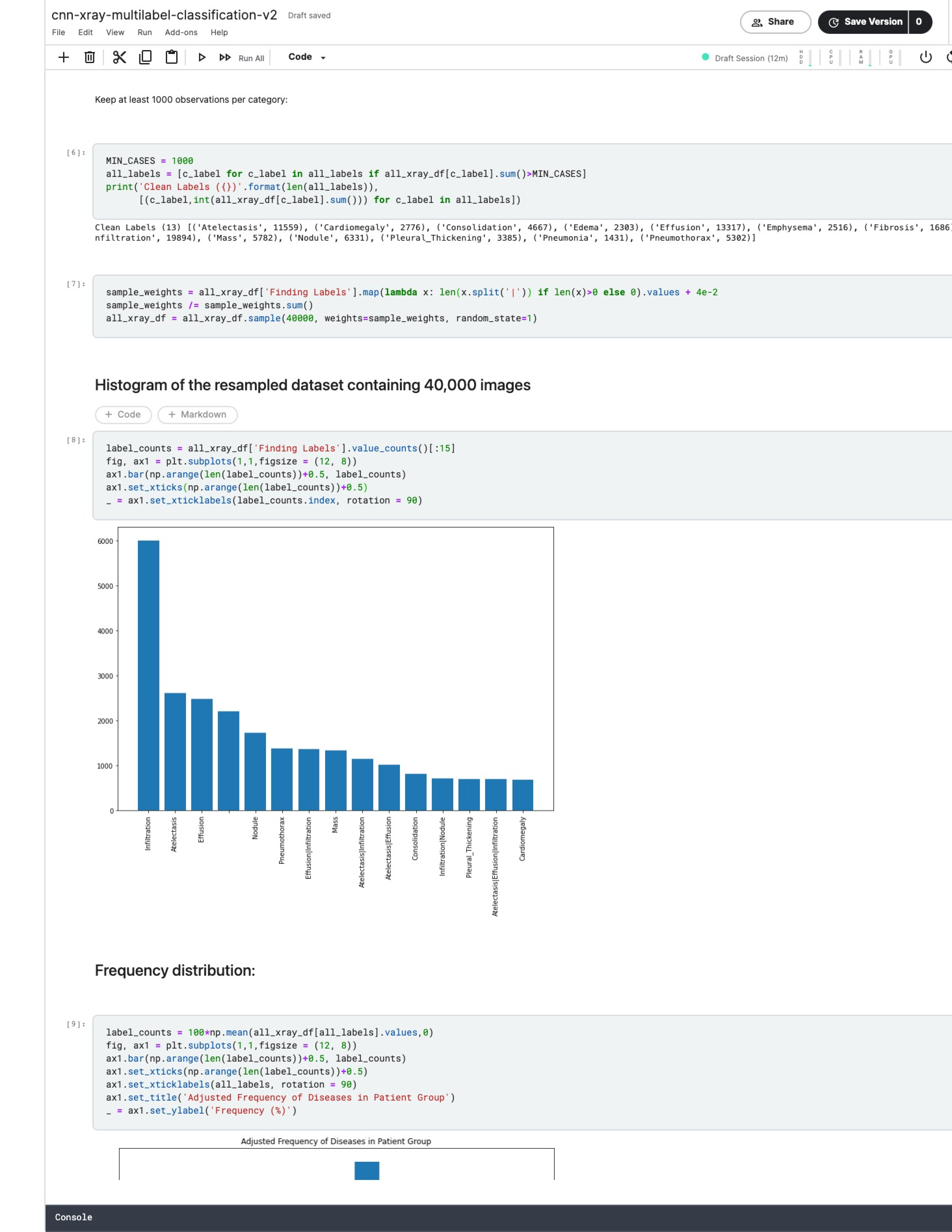
3 rows × 27 columns

Resampling the dataset to achieve more balance among categories

Keep at least 1000 observations per category:

[6]:
```
MIN_CASES = 1000
```

Console



Keep at least 1000 observations per category:

[6]:

```
MIN_CASES = 1000
all_labels = [c_label for c_label in all_labels if all_xray_df[c_label].sum()>MIN_CASES]
print('Clean Labels ({})'.format(len(all_labels)),
      [(c_label,int(all_xray_df[c_label].sum())) for c_label in all_labels])
```

```
Clean Labels (13) [('Atelectasis', 11559), ('Cardiomegaly', 2776), ('Consolidation', 4667), ('Edema', 2303), ('Effusion', 13317), ('Emphysema', 2516), ('Fibrosis', 1686)
nfiltration', 19894), ('Mass', 5782), ('Nodule', 6331), ('Pleural_Thickening', 3385), ('Pneumonia', 1431), ('Pneumothorax', 5302)]
```

[7]:

```
sample_weights = all_xray_df['Finding Labels'].map(lambda x: len(x.split('|')) if len(x)>0 else 0).values + 4e-2
sample_weights /= sample_weights.sum()
all_xray_df = all_xray_df.sample(40000, weights=sample_weights, random_state=1)
```

Histogram of the resampled dataset containing 40,000 images

[8]:

```
label_counts = all_xray_df['Finding Labels'].value_counts()[:15]
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
_ = ax1.set_xticklabels(label_counts.index, rotation = 90)
```

Frequency distribution:

[9]:

```
label_counts = 100*np.mean(all_xray_df[all_labels].values,0)
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
ax1.set_xticklabels(all_labels, rotation = 90)
ax1.set_title('Adjusted Frequency of Diseases in Patient Group')
_ = ax1.set_ylabel('Frequency (%)')
```

cnn-xray-multilabel-classification-v2 Draft saved

File Edit View Run Add-ons Help

Share Save Version 0

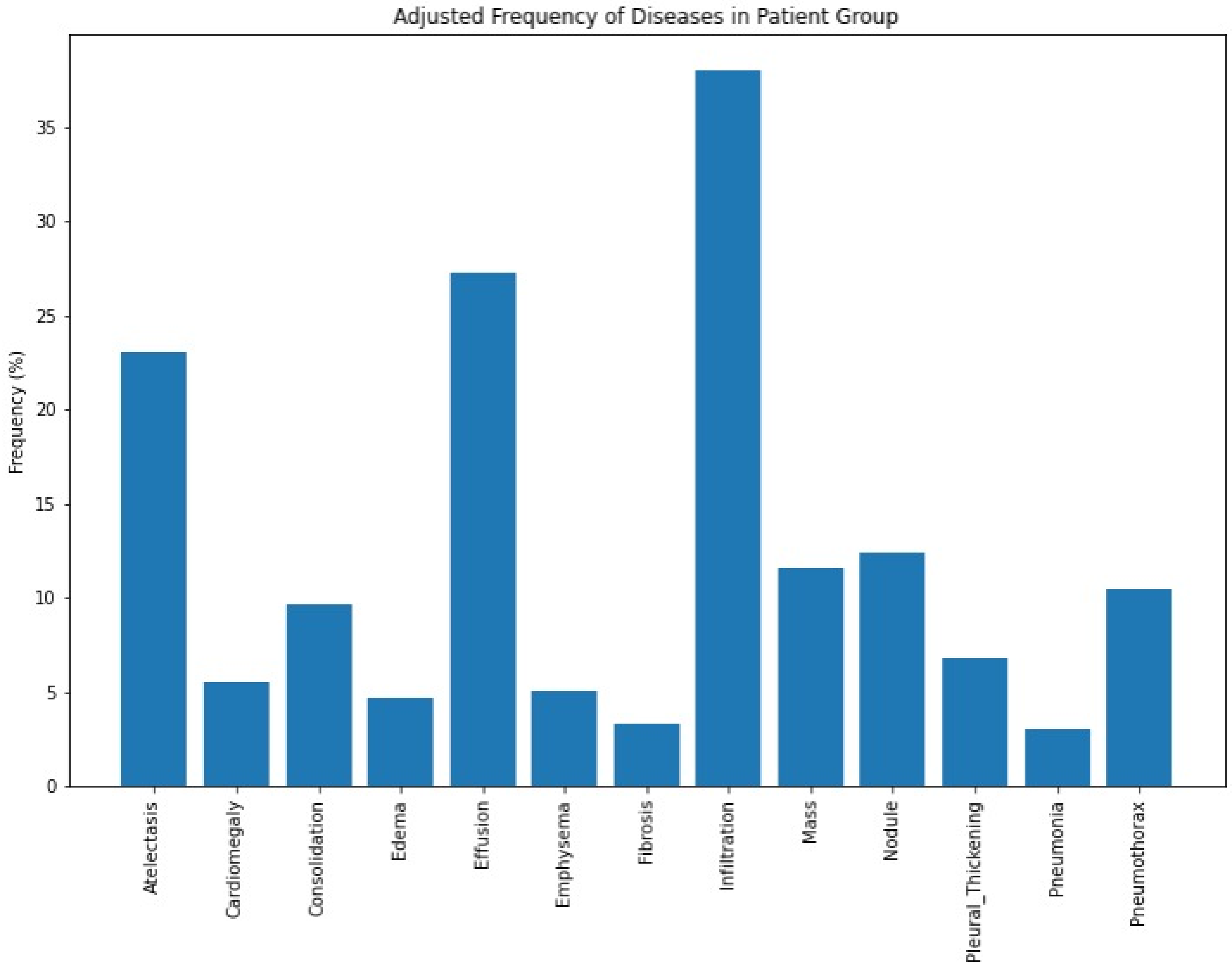
Run All Code

Draft Session (12m)

Frequency distribution:

[9]:

```python
label_counts = 100*np.mean(all_xray_df[all_labels].values,0)
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
ax1.set_xticklabels(all_labels, rotation = 90)
ax1.set_title('Adjusted Frequency of Diseases in Patient Group')
_ = ax1.set_ylabel('Frequency (%)')
```



Prepare training data

Create a vector for labels first:

[10]:

```python
all_xray_df['disease_vec'] = all_xray_df.apply(lambda x: [x[all_labels].values], 1).map(lambda x: x[0])
all_xray_df.drop(['Hernia'], axis=1, inplace=True)
```

75% for training and 25% for validation:

[11]:

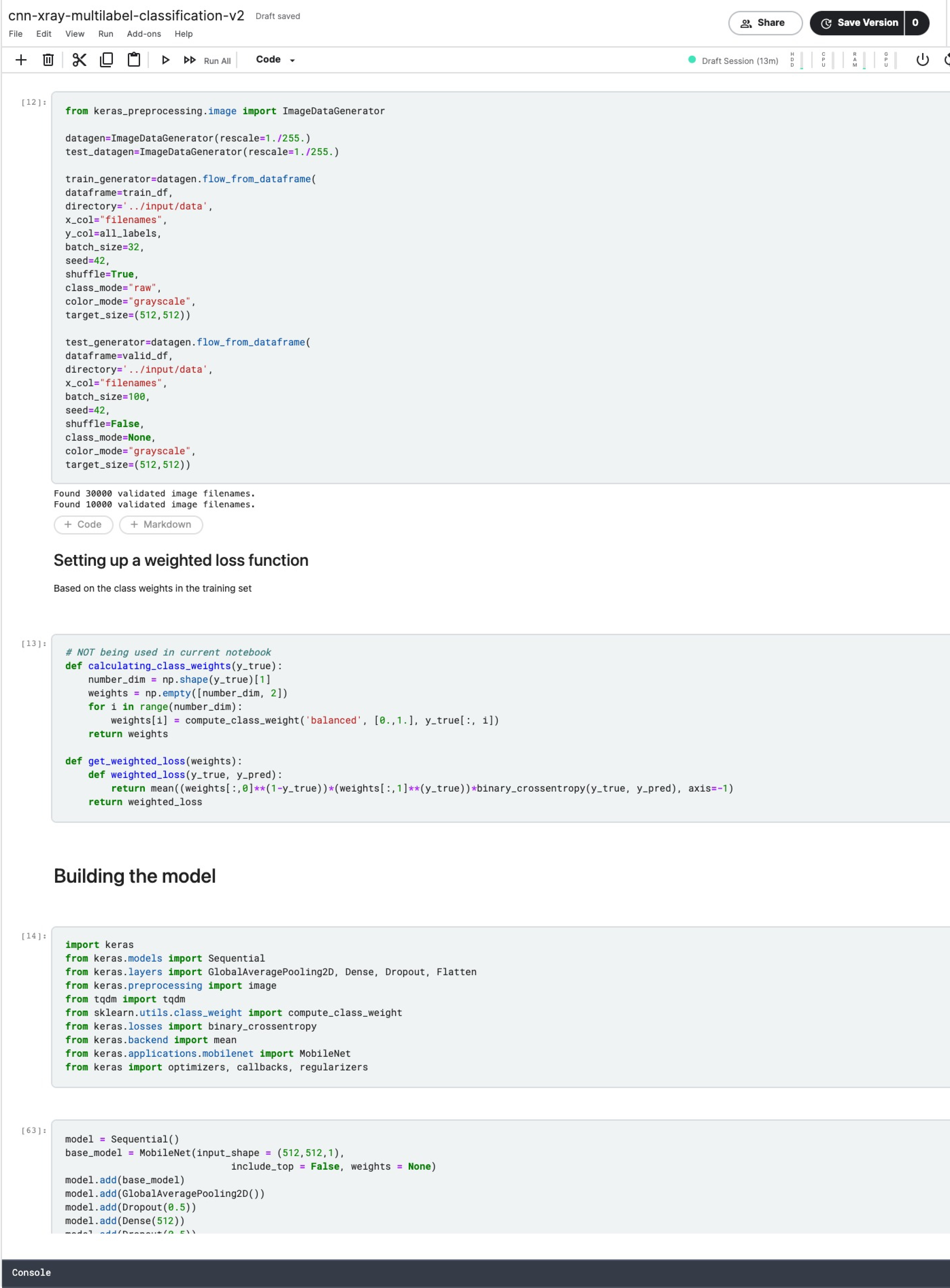
```python
from sklearn.model_selection import train_test_split
train_df, valid_df = train_test_split(all_xray_df,
                                   test_size = 0.25,
                                   random_state = 1,
                                   stratify = all_xray_df['Finding Labels'].map(lambda x: x[:4]))
print('train', train_df.shape[0], 'validation', valid_df.shape[0])
```

```
train 30000 validation 10000
```

[12]:

```python
from keras_preprocessing.image import ImageDataGenerator

datagen=ImageDataGenerator(rescale=1./255.)
test_datagen=ImageDataGenerator(rescale=1./255.)

train_generator=datagen.flow_from_dataframe(
dataframe=train_df,
directory='../input/data',
```

Console



[12]:

```
from keras_preprocessing.image import ImageDataGenerator

datagen=ImageDataGenerator(rescale=1./255.)
test_datagen=ImageDataGenerator(rescale=1./255.)

train_generator=datagen.flow_from_dataframe(
dataframe=train_df,
directory='../input/data',
x_col="filenames",
y_col=all_labels,
batch_size=32,
seed=42,
shuffle=True,
class_mode="raw",
color_mode="grayscale",
target_size=(512,512))

test_generator=datagen.flow_from_dataframe(
dataframe=valid_df,
directory='../input/data',
x_col="filenames",
batch_size=100,
seed=42,
shuffle=False,
class_mode=None,
color_mode="grayscale",
target_size=(512,512))
```

```
Found 30000 validated image filenames.
Found 10000 validated image filenames.
```

Setting up a weighted loss function

Based on the class weights in the training set

[13]:

```
# NOT being used in current notebook
def calculating_class_weights(y_true):
    number_dim = np.shape(y_true)[1]
    weights = np.empty([number_dim, 2])
    for i in range(number_dim):
        weights[i] = compute_class_weight('balanced', [0.,1.], y_true[:, i])
    return weights

def get_weighted_loss(weights):
    def weighted_loss(y_true, y_pred):
        return mean((weights[:,0]**(1-y_true))*(weights[:,1]**(y_true))*binary_crossentropy(y_true, y_pred), axis=-1)
    return weighted_loss
```

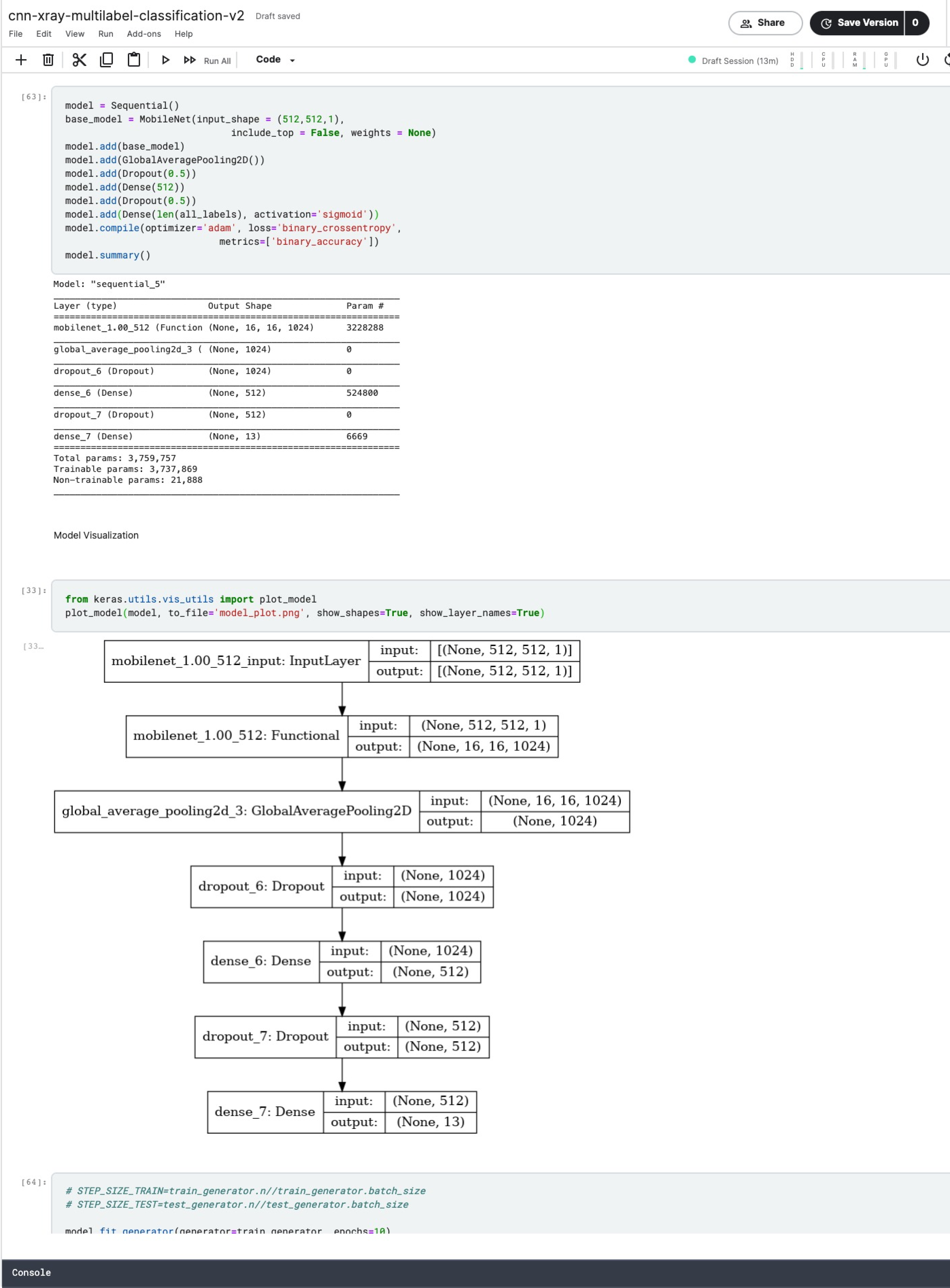
Building the model

[14]:

```
import keras
from keras.models import Sequential
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten
from keras.preprocessing import image
from tqdm import tqdm
from sklearn.utils.class_weight import compute_class_weight
from keras.losses import binary_crossentropy
from keras.backend import mean
from keras.applications.mobilenet import MobileNet
from keras import optimizers, callbacks, regularizers
```

[63]:

```
model = Sequential()
base_model = MobileNet(input_shape = (512,512,1),
                             include_top = False, weights = None)
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Dropout(0.5))
```



[63]:

```
model = Sequential()
base_model = MobileNet(input_shape = (512,512,1),
                              include_top = False, weights = None)
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Dropout(0.5))
model.add(Dense(len(all_labels), activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
                            metrics=['binary_accuracy'])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenet_1.00_512 (Function (None, 16, 16, 1024)      3228288
_________________________________________________________________
global_average_pooling2d_3 ( (None, 1024)              0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 13)                6669
=================================================================
Total params: 3,759,757
Trainable params: 3,737,869
Non-trainable params: 21,888
_________________________________________________________________
```

Model Visualization

[33]:

```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

[33…

mobilenet_1.00_512_input: InputLayer | input: [(None, 512, 512, 1)] | output: [(None, 512, 512, 1)]

mobilenet_1.00_512: Functional | input: (None, 512, 512, 1) | output: (None, 16, 16, 1024)

global_average_pooling2d_3: GlobalAveragePooling2D | input: (None, 16, 16, 1024) | output: (None, 1024)

dropout_6: Dropout | input: (None, 1024) | output: (None, 1024)

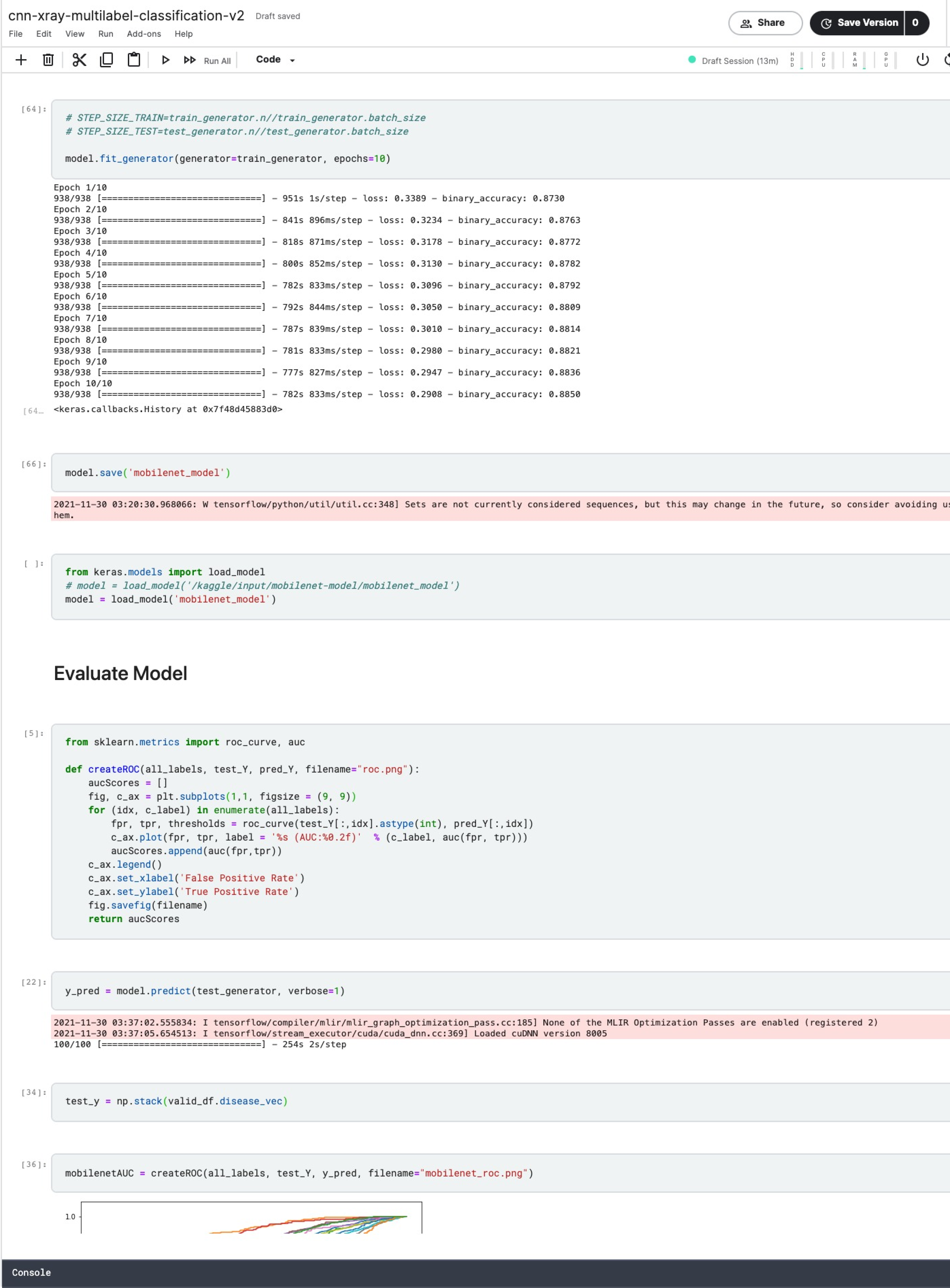
dense_6: Dense | input: (None, 1024) | output: (None, 512)

dropout_7: Dropout | input: (None, 512) | output: (None, 512)

dense_7: Dense | input: (None, 512) | output: (None, 13)

[64]:

```
# STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
# STEP_SIZE_TEST=test_generator.n//test_generator.batch_size

model.fit_generator(generator=train_generator, epochs=10)
```



[64]:
```
# STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
# STEP_SIZE_TEST=test_generator.n//test_generator.batch_size

model.fit_generator(generator=train_generator, epochs=10)
```

```
Epoch 1/10
938/938 [==============================] - 951s 1s/step - loss: 0.3389 - binary_accuracy: 0.8730
Epoch 2/10
938/938 [==============================] - 841s 896ms/step - loss: 0.3234 - binary_accuracy: 0.8763
Epoch 3/10
938/938 [==============================] - 818s 871ms/step - loss: 0.3178 - binary_accuracy: 0.8772
Epoch 4/10
938/938 [==============================] - 800s 852ms/step - loss: 0.3130 - binary_accuracy: 0.8782
Epoch 5/10
938/938 [==============================] - 782s 833ms/step - loss: 0.3096 - binary_accuracy: 0.8792
Epoch 6/10
938/938 [==============================] - 792s 844ms/step - loss: 0.3050 - binary_accuracy: 0.8809
Epoch 7/10
938/938 [==============================] - 787s 839ms/step - loss: 0.3010 - binary_accuracy: 0.8814
Epoch 8/10
938/938 [==============================] - 781s 833ms/step - loss: 0.2980 - binary_accuracy: 0.8821
Epoch 9/10
938/938 [==============================] - 777s 827ms/step - loss: 0.2947 - binary_accuracy: 0.8836
Epoch 10/10
938/938 [==============================] - 782s 833ms/step - loss: 0.2908 - binary_accuracy: 0.8850
```

[64…
```
<keras.callbacks.History at 0x7f48d45883d0>
```

[66]:
```
model.save('mobilenet_model')
```

```
2021-11-30 03:20:30.968066: W tensorflow/python/util/util.cc:348] Sets are not currently considered sequences, but this may change in the future, so consider avoiding us
hem.
```

[ ]:
```
from keras.models import load_model
# model = load_model('/kaggle/input/mobilenet-model/mobilenet_model')
model = load_model('mobilenet_model')
```

Evaluate Model

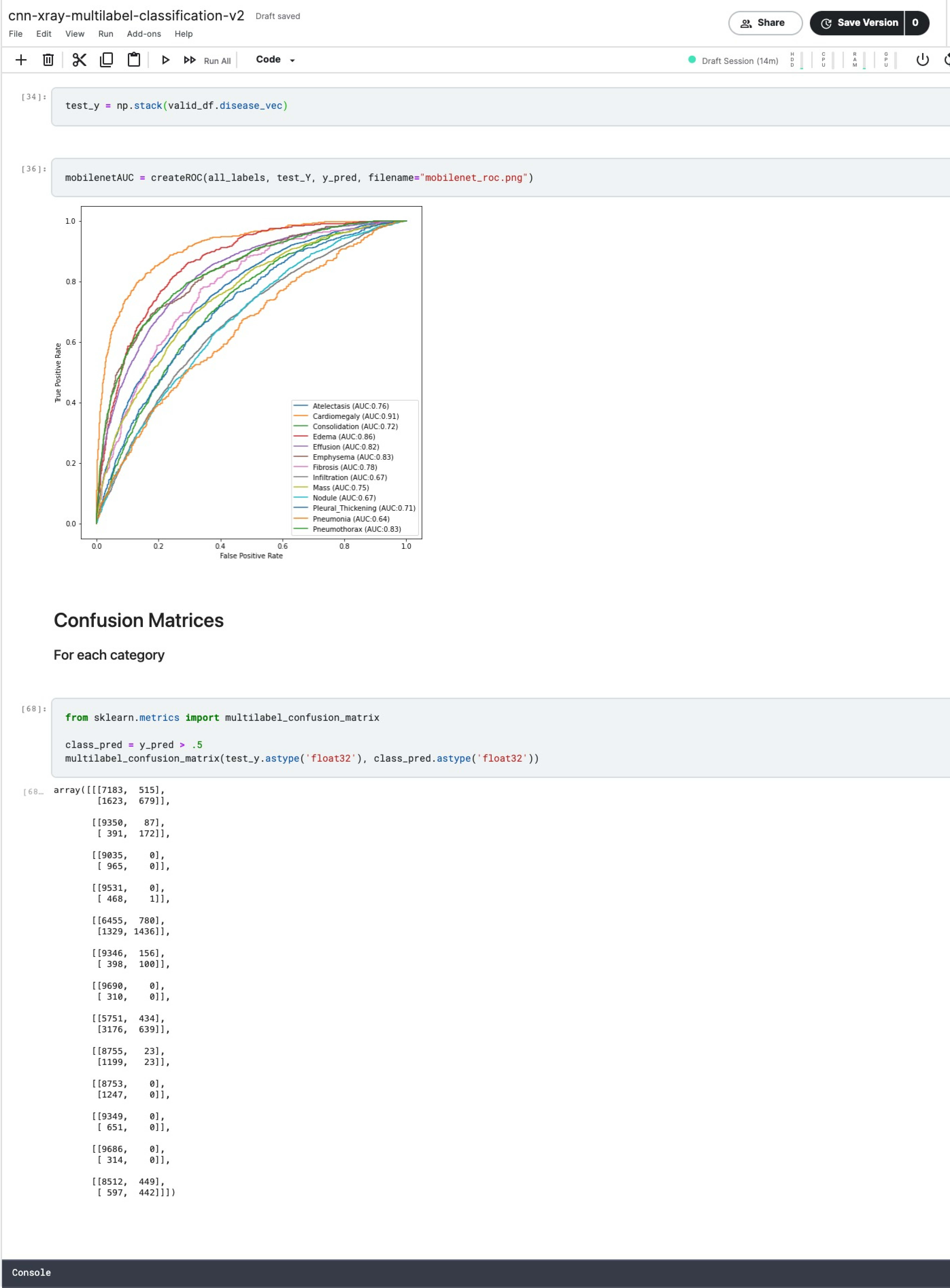
[5]:
```
from sklearn.metrics import roc_curve, auc

def createROC(all_labels, test_Y, pred_Y, filename="roc.png"):
    aucScores = []
    fig, c_ax = plt.subplots(1,1, figsize = (9, 9))
    for (idx, c_label) in enumerate(all_labels):
        fpr, tpr, thresholds = roc_curve(test_Y[:,idx].astype(int), pred_Y[:,idx])
        c_ax.plot(fpr, tpr, label = '%s (AUC:%0.2f)'  % (c_label, auc(fpr, tpr)))
        aucScores.append(auc(fpr,tpr))
    c_ax.legend()
    c_ax.set_xlabel('False Positive Rate')
    c_ax.set_ylabel('True Positive Rate')
    fig.savefig(filename)
    return aucScores
```

[22]:
```
y_pred = model.predict(test_generator, verbose=1)
```

```
2021-11-30 03:37:02.555834: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
2021-11-30 03:37:05.654513: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
100/100 [==============================] - 254s 2s/step
```

[34]:
```
test_y = np.stack(valid_df.disease_vec)
```

[36]:
```
mobilenetAUC = createROC(all_labels, test_Y, y_pred, filename="mobilenet_roc.png")
```



[34]:
```
test_y = np.stack(valid_df.disease_vec)
```

[36]:
```
mobilenetAUC = createROC(all_labels, test_Y, y_pred, filename="mobilenet_roc.png")
```

Confusion Matrices

For each category

[68]:
```
from sklearn.metrics import multilabel_confusion_matrix

class_pred = y_pred > .5
multilabel_confusion_matrix(test_y.astype('float32'), class_pred.astype('float32'))
```

[68…
```
array([[[7183,  515],
        [1623,  679]],

       [[9350,   87],
        [ 391,  172]],

       [[9035,    0],
        [ 965,    0]],

       [[9531,    0],
        [ 468,    1]],

       [[6455,  780],
        [1329, 1436]],

       [[9346,  156],
        [ 398,  100]],

       [[9690,    0],
        [ 310,    0]],

       [[5751,  434],
        [3176,  639]],

       [[8755,   23],
        [1199,   23]],

       [[8753,    0],
        [1247,    0]],

       [[9349,    0],
        [ 651,    0]],

       [[9686,    0],
        [ 314,    0]],

       [[8512,  449],
        [ 597,  442]]])
```