In [1]:

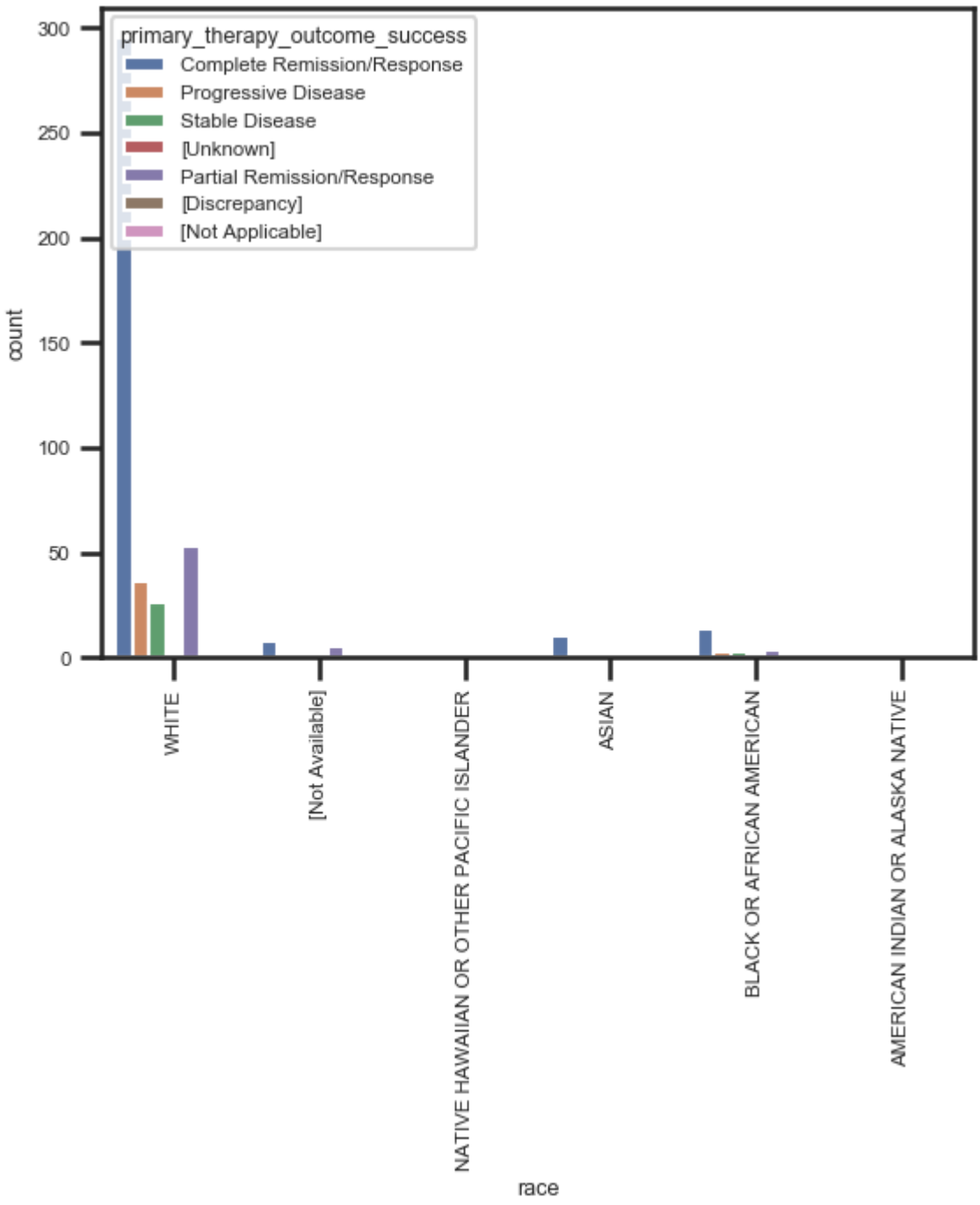
```
#imports
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import classification_report
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
import shap
from sklearn.preprocessing import LabelEncoder

#seaborn set up
sns.set(context='talk', style='white', rc={'figure.facecolor':'white'}, font_scale=0.5)
sns.set_style('ticks')

#warnings
pd.options.mode.chained_assignment = None  # default='warn'

#readins
datasetplots1 = pd.read_csv("0abf0e45-07c1-4ac2-b27c-2a4f3f460364.csv")
datasetplots2 = pd.read_csv("5ea3f385-7f0b-4332-881b-a3a30b399623.csv")
datasetplots3 = pd.read_csv("09a4994f-d50e-423c-895e-0be2d6dd5762.csv")
datasetplots4 = pd.read_csv("30c149ac-9ac2-4f51-88a3-68bb4afb50a9.csv")

#ds1 + output set up
ds1 = datasetplots1[datasetplots1['primary_therapy_outcome_success'] !="[Not Available]
output = ds1.drop(columns=['bcr_patient_uuid', 'bcr_followup_barcode', 'bcr_followup_uu
output = output.drop_duplicates(subset='bcr_patient_barcode', keep="last")

#datasetplots4 set up
mergewith = datasetplots4.drop(columns=['bcr_patient_uuid', 'form_completion_date', 'ti
mergewith = mergewith.drop_duplicates(subset = 'bcr_patient_barcode', keep = 'last')

#datasetplots2 set up
ds2 = datasetplots2.drop_duplicates(subset ='bcr_patient_barcode', keep="last")
ds2 = ds2.drop(columns=['bcr_patient_uuid', 'bcr_followup_barcode', 'days_to_new_tumor_

#dataset mergers
dffinal = output.merge(mergewith, on='bcr_patient_barcode', how='outer')
dftest = dffinal.merge(ds2, on='bcr_patient_barcode', how='outer')

#countplots - race
raceplot = sns.countplot(x='race',data=dftest, hue='primary_therapy_outcome_success')
raceplot.set_xticklabels(raceplot.get_xticklabels(),rotation = 90)
plt.show()

#countplots - stage
stageplot = sns.countplot(x='clinical_stage',data=dftest, hue='primary_therapy_outcome_
stageplot.set_xticklabels(stageplot.get_xticklabels(),rotation = 90)
```

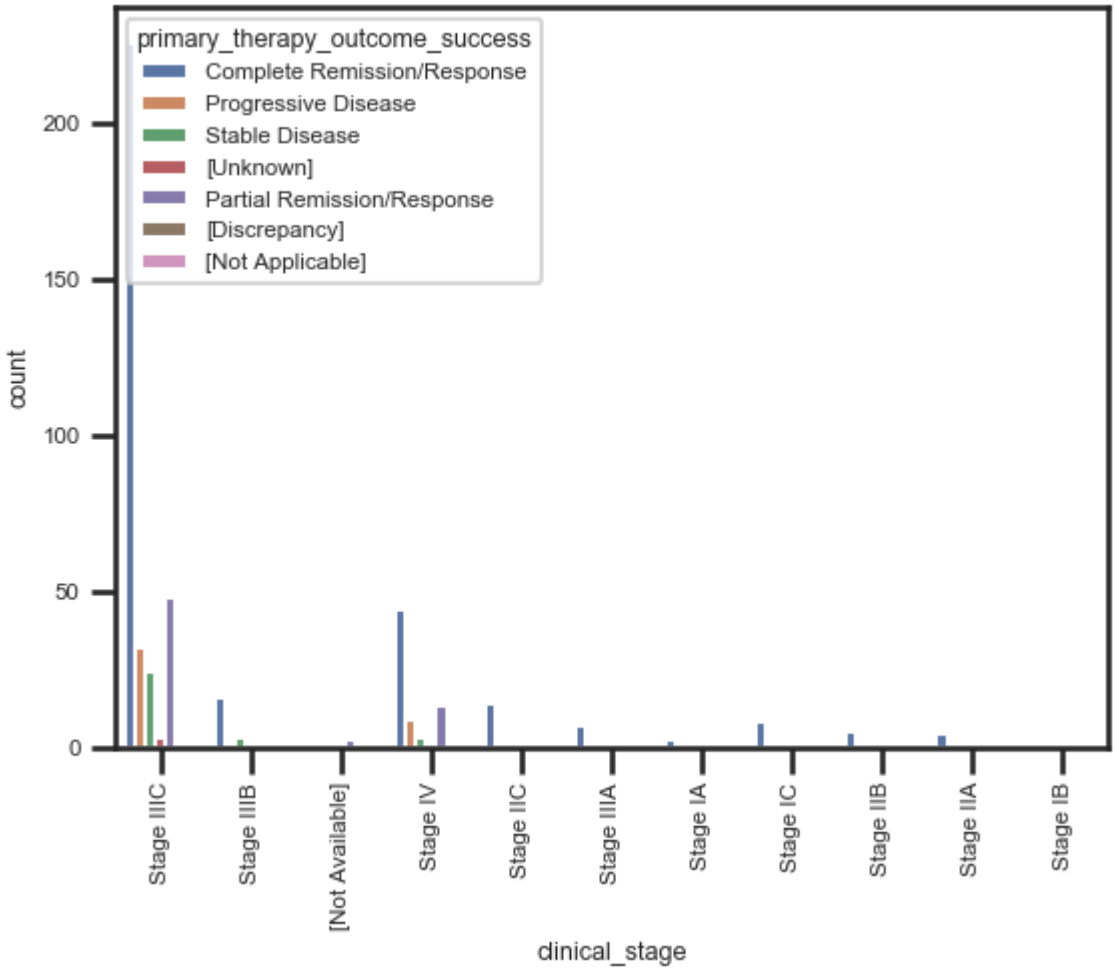
```
plt.show()

#jewish origin
dftest['jewish_origin'] = dftest['jewish_origin'].replace({"[Not Evaluated]" : "[Not Av
dftest['jewish_origin'] = dftest['jewish_origin'].replace({'[Not Available]' : 0, 'ASHK

#race
race = pd.get_dummies(dftest['race'])
dftest = dftest.drop('race', axis=1)
dftest = dftest.join(race)
dftest.rename(columns={'[Not Available]':'race_[Not Available]'}, inplace=True)

#neoadjuvant treatment
dftest['history_of_neoadjuvant_treatment'].replace({"No":0, "Yes":1}, inplace=True)

#diagnosis method
method = pd.get_dummies(dftest['initial_pathologic_diagnosis_method'])
dftest = dftest.drop('initial_pathologic_diagnosis_method', axis=1)
dftest = dftest.join(method)
dftest.rename(columns={'[Not Available]':'method_[Not Available]'}, inplace=True)

# histologic grade
grade = pd.get_dummies(dftest['neoplasm_histologic_grade'])
dftest = dftest.drop('neoplasm_histologic_grade', axis=1)
dftest = dftest.join(grade)
dftest.rename(columns={'[Not Available]':'grade_[Not Available]'}, inplace=True)
dftest

#venous invasion
venous = pd.get_dummies(dftest['venous_invasion'])
dftest = dftest.drop('venous_invasion', axis=1)
dftest = dftest.join(venous)
dftest.rename(columns={'NO':'venous_NO'}, inplace=True)
dftest.rename(columns={'YES':'venous_YES'}, inplace=True)
dftest.rename(columns={'[Not Available]':'venous_[Not Available]'}, inplace=True)
dftest.rename(columns={'[Unknown]':'venous_[Unknown]'}, inplace=True)

#Lymphatic invasion
lymphatic = pd.get_dummies(dftest['lymphatic_invasion'])
dftest = dftest.drop('lymphatic_invasion', axis=1)
dftest = dftest.join(lymphatic)
dftest.rename(columns={'NO':'lymphatic_NO'}, inplace=True)
dftest.rename(columns={'YES':'lymphatic_YES'}, inplace=True)
dftest.rename(columns={'[Not Available]':'lymphatic_[Not Available]'}, inplace=True)
dftest.rename(columns={'[Unknown]':'lymphatic_[Unknown]'}, inplace=True)

#anatomic neoplasm subdivision
subdivision = pd.get_dummies(dftest['anatomic_neoplasm_subdivision'])
dftest = dftest.drop('anatomic_neoplasm_subdivision', axis=1)
dftest = dftest.join(subdivision)
dftest.rename(columns={'[Not Available]':'subdivision_[Not Available]'}, inplace=True)

#stage
stage = pd.get_dummies(dftest['clinical_stage'])
dftest = dftest.drop('clinical_stage', axis=1)
dftest = dftest.join(stage)
dftest.rename(columns={'[Not Available]':'stage_[Not Available]'}, inplace=True)

#icd (insurance code)
icd = pd.get_dummies(dftest['icd_10'])
```

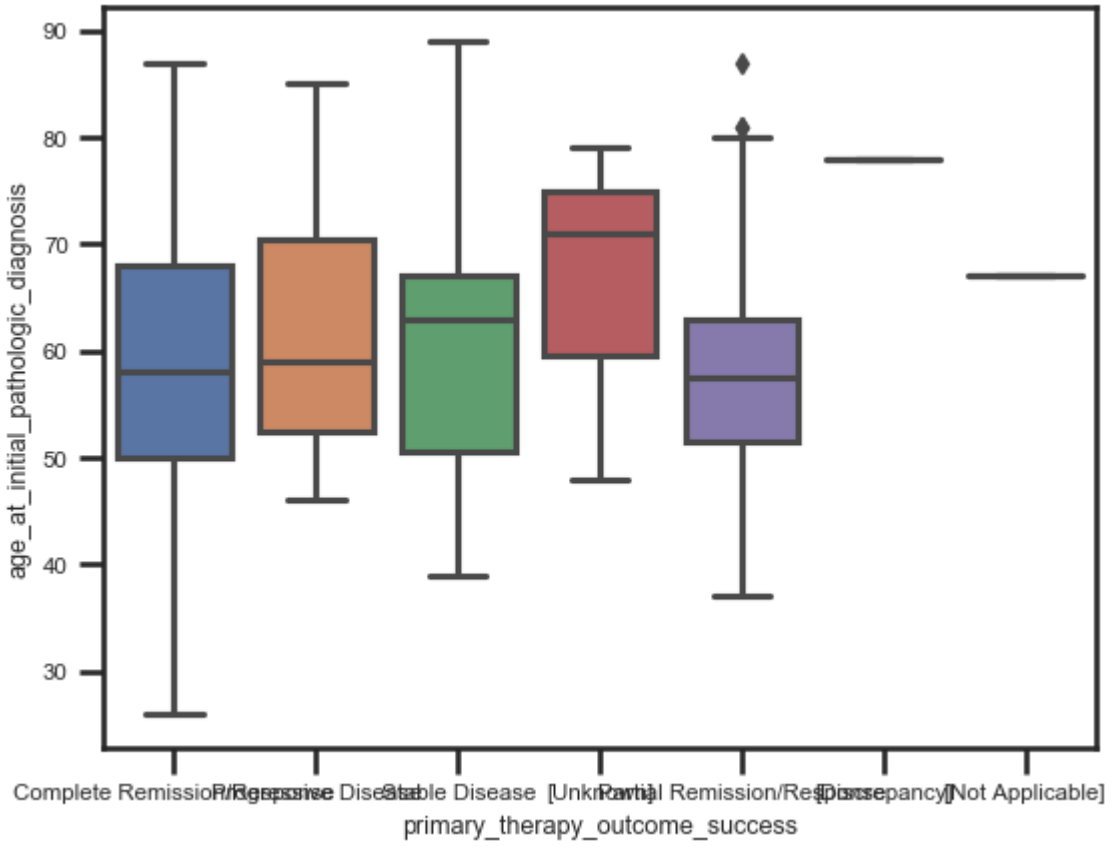
```
dftest = dftest.drop('icd_10', axis=1)
dftest = dftest.join(icd)

#tissue source site
source = pd.get_dummies(dftest['tissue_source_site'])
dftest = dftest.drop('tissue_source_site', axis=1)
dftest = dftest.join(source)

#tumor tissue site
tumor_tissue = pd.get_dummies(dftest['tumor_tissue_site'])
dftest = dftest.drop('tumor_tissue_site', axis=1)
dftest = dftest.join(tumor_tissue)

#additional radiation therapy
radiation = pd.get_dummies(dftest['additional_radiation_therapy'])
dftest = dftest.drop('additional_radiation_therapy', axis=1)
dftest = dftest.join(radiation)
dftest.rename(columns={'[Not Available]':'radiation_[Not Available]'}, inplace=True)
dftest.rename(columns={'NO':'radiation_NO'}, inplace=True)
dftest.rename(columns={'YES':'radiation_YES'}, inplace=True)
dftest

#additional pharma therapy
pharma = pd.get_dummies(dftest['additional_pharmaceutical_therapy'])
dftest = dftest.drop('additional_pharmaceutical_therapy', axis=1)
dftest = dftest.join(pharma)
dftest.rename(columns={'[Not Available]':'pharma_[Not Available]'}, inplace=True)
dftest.rename(columns={'YES':'pharma_YES'}, inplace=True)

#dropping all the ID columns (before classification)
dftest = dftest.drop(columns=['bcr_patient_barcode'])
dftest = dftest.drop(columns=['patient_id'])
dftest = dftest.drop(columns=['race_[Not Available]', 'method_[Not Available]', 'grade_
dftest = dftest.drop(columns=['lymphatic_[Not Available]', 'lymphatic_[Unknown]', 'subd
dftest = dftest.drop(columns=['radiation_[Not Available]', 'pharma_[Not Available]'])


#plots - age
alloutcomesage = sns.boxplot(x='primary_therapy_outcome_success', y='age_at_initial_pat
plt.show()

#print
dftest
```

```
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\framework\d
types.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprec
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\framework\d
types.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprec
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\framework\d
types.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprec
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\framework\d
types.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprec
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
```

```
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\framework\d
types.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprec
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\framework\d
types.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprec
ated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorboard\compat\tensorflow
_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)t
ype'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorboard\compat\tensorflow
_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)t
ype'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorboard\compat\tensorflow
_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)t
ype'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorboard\compat\tensorflow
_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)t
ype'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorboard\compat\tensorflow
_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)t
ype'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tensorboard\compat\tensorflow
_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)t
ype'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
Using TensorFlow backend.
```


primary_therapy_outcome_success
Complete Remission/Response
Progressive Disease
Stable Disease
[Unknown]
Partial Remission/Response
[Discrepancy]
[Not Applicable]
count
300
250
200
150
100
50
0
WHITE
[Not Available]
NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER
ASIAN
BLACK OR AFRICAN AMERICAN
AMERICAN INDIAN OR ALASKA NATIVE
race

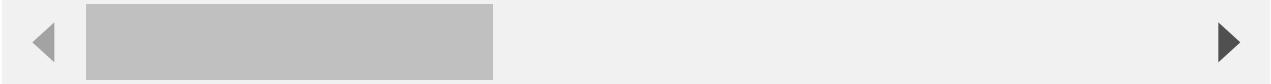

primary_therapy_outcome_success
Complete Remission/Response
Progressive Disease
Stable Disease
[Unknown]
Partial Remission/Response
[Discrepancy]
[Not Applicable]
count
200
150
100
50
0
Stage IIIC
Stage IIIB
[Not Available]
Stage IV
Stage IIC
Stage IIIA
Stage IA
Stage IC
Stage IIB
Stage IIA
Stage IB
clinical_stage


age_at_initial_pathologic_diagnosis
90
80
70
60
50
40
30
Complete Remission/Response
Progressive Disease
Stable Disease
[Unknown]
Partial Remission/Response
[Discrepancy]
[Not Applicable]
primary_therapy_outcome_success

Out[1]:

	primary_therapy_outcome_success	jewish_origin	history_of_neoadjuvant_treatment	age_at_initial_pat
0	Complete Remission/Response	0	0	
1	Complete Remission/Response	0	0	
2	Complete Remission/Response	0	0	
3	Progressive Disease	0	0	
4	Progressive Disease	0	0	
...	...	...	...	
582	NaN	0	0	
583	NaN	0	0	
584	NaN	0	0	
585	NaN	0	0	
586	NaN	0	0	

587 rows × 69 columns



In [2]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
```

In [3]:
```
#binary outcome (temporary)
dftest['success'] = np.where(dftest.primary_therapy_outcome_success.str.contains("Remis
dftest = dftest.drop("primary_therapy_outcome_success", axis=1)
agebox = sns.boxplot(x='success', y='age_at_initial_pathologic_diagnosis', data=dftest)
```

In [4]:
```
dftest['success'].value_counts()
```

Out[4]:
```
1    466
0    121
Name: success, dtype: int64
```

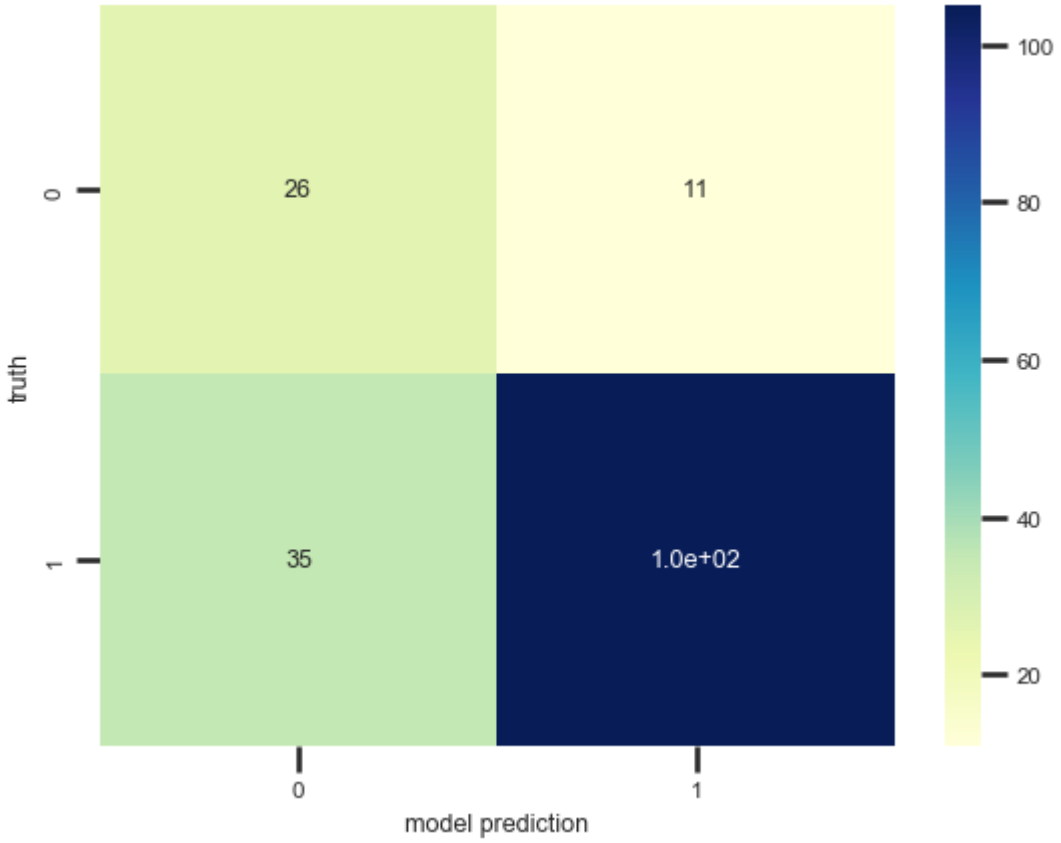
In [5]:
```
#binary outcome variable defs
Y = dftest[['success']]
X = dftest.drop('success', axis=1)

#train/test sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0
y_test["success"].value_counts(normalize=True)
```

Out[5]:
```
1    0.79096
0    0.20904
Name: success, dtype: float64
```

In [6]:
```
logit = LogisticRegression(solver="lbfgs", max_iter=1000, class_weight='balanced')
logit.fit(x_train.values, y_train.values.reshape(-1,))
y_pred = logit.predict(x_test)
accuracy_score(y_test, y_pred)
```

Out[6]:
```
0.7401129943502824
```

In [7]:
```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu")
plt.xlabel("model prediction")
plt.ylabel("truth")
plt.show()
```

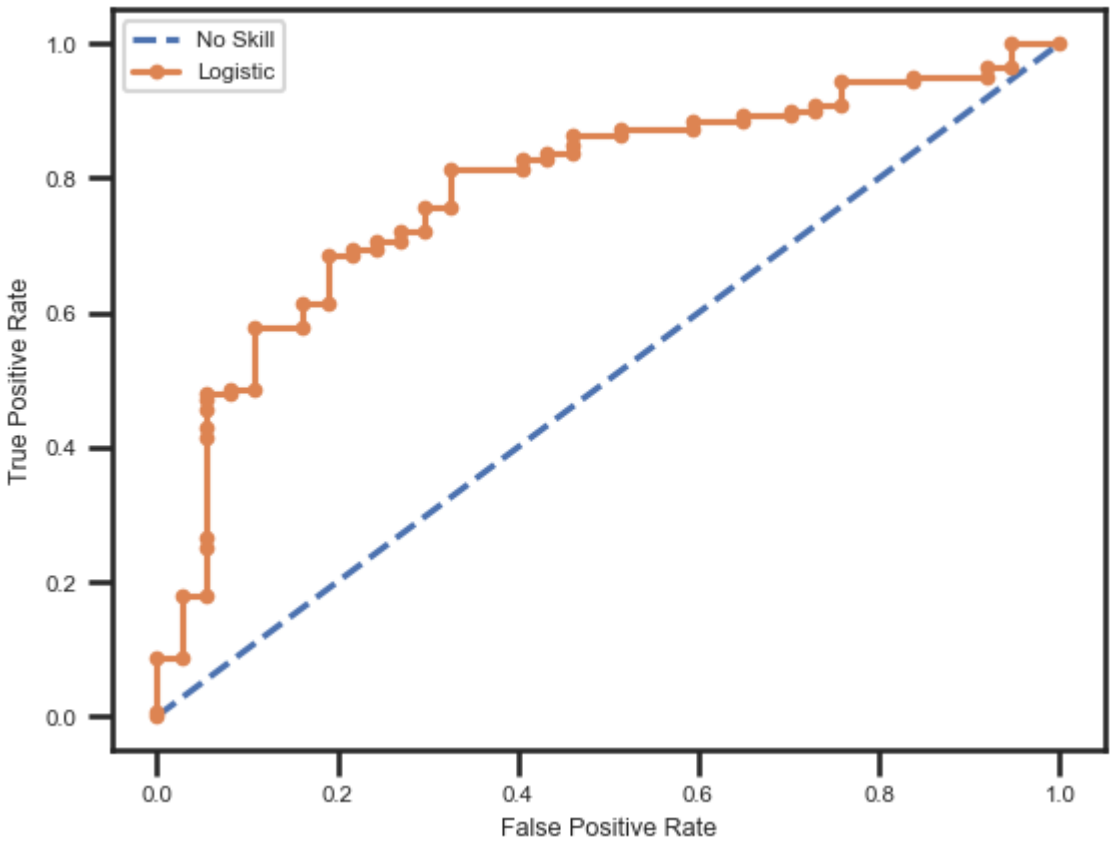


In [8]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
```

In [9]:
```
ns_probs = [0 for _ in range(len(y_test))]
# predict probabilities
lr_probs = logit.predict_proba(x_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
```

```
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.784
```



In [10]:

```
#neural net defs
model = Sequential()
model.add(Dense(6, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [11]:

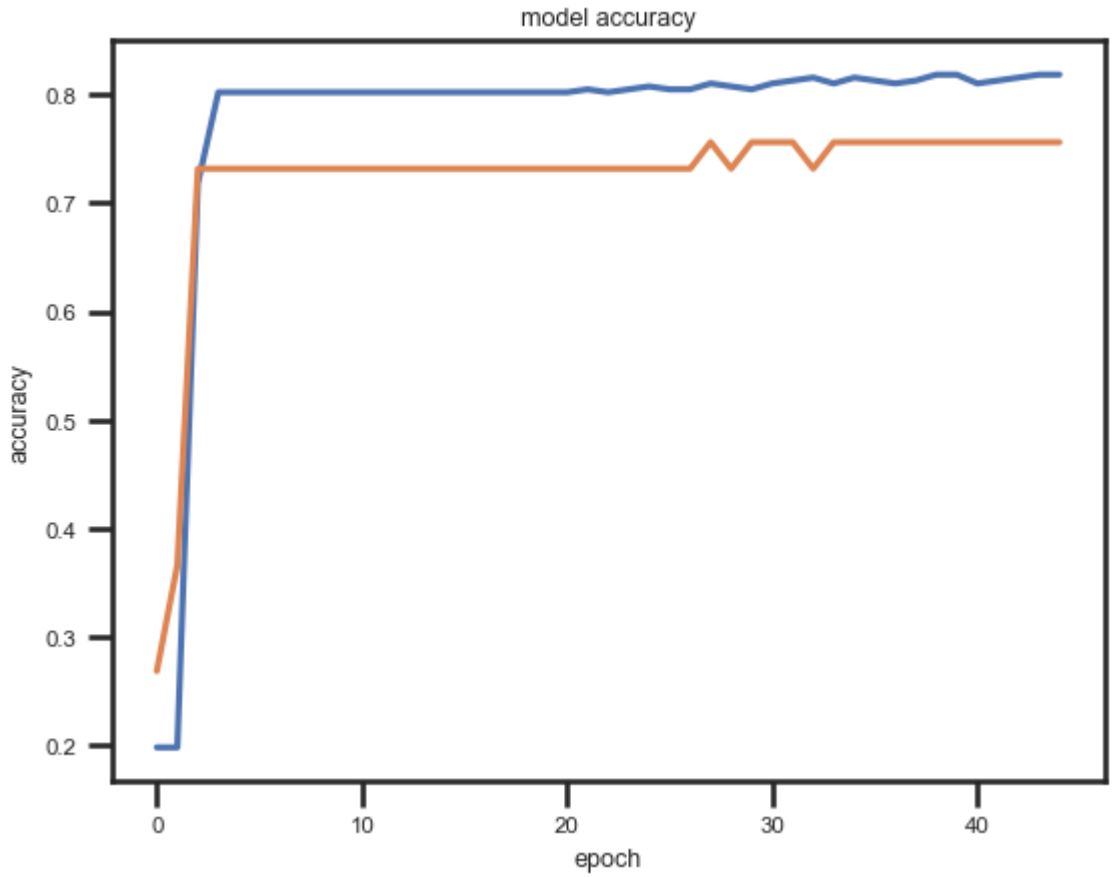
```
#neural net compile/fit
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train, validation_split = 0.1, class_weight= 'balanced',
```

```
WARNING:tensorflow:From C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\tenso
rflow\python\ops\nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tensorflow.
python.ops.array_ops) is deprecated and will be removed in a future version.
```

```
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\aparn\anaconda3\envs\tensorflow\lib\site-packages\keras
\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please u
se tf.compat.v1.global_variables instead.
```

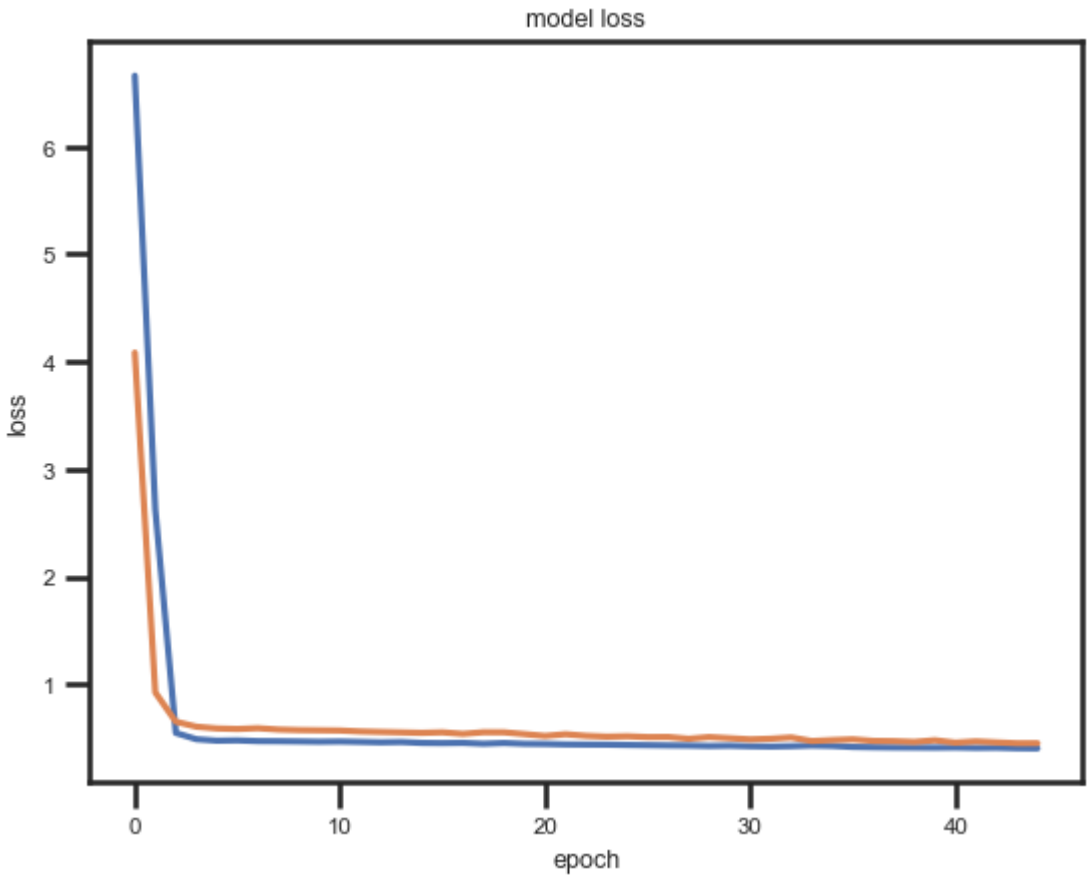
In [13]:

```
#model accuracy plot
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.show()
```



In [14]:

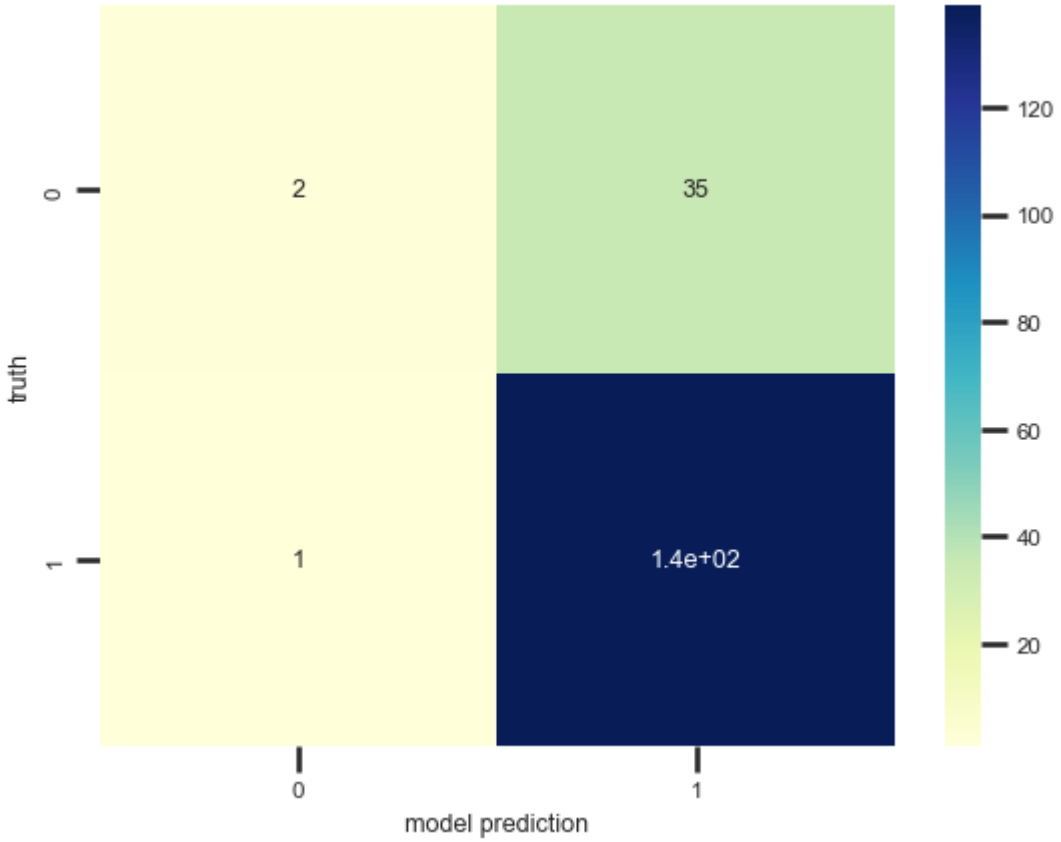
```
#model loss plot
plt.plot(history.history["loss"])
plt.plot(history.history['val_loss'])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.show()
```



In [15]:
```
#model eval
score = model.evaluate(x_test, y_test, verbose=0)
score
```

Out[15]:
```
[0.4493095005972911, 0.7966101765632629]
```

In [16]:
```
#nn cnf matrix
predictions = model.predict(x_test)
confusion_matrix = sklearn.metrics.confusion_matrix(y_test, np.rint(predictions))
sns.heatmap(confusion_matrix, annot=True, cmap="YlGnBu")
plt.xlabel("model prediction")
plt.ylabel("truth")
plt.show()
```



In [17]:
```
kfold = StratifiedKFold(n_splits=5, shuffle=True)
```

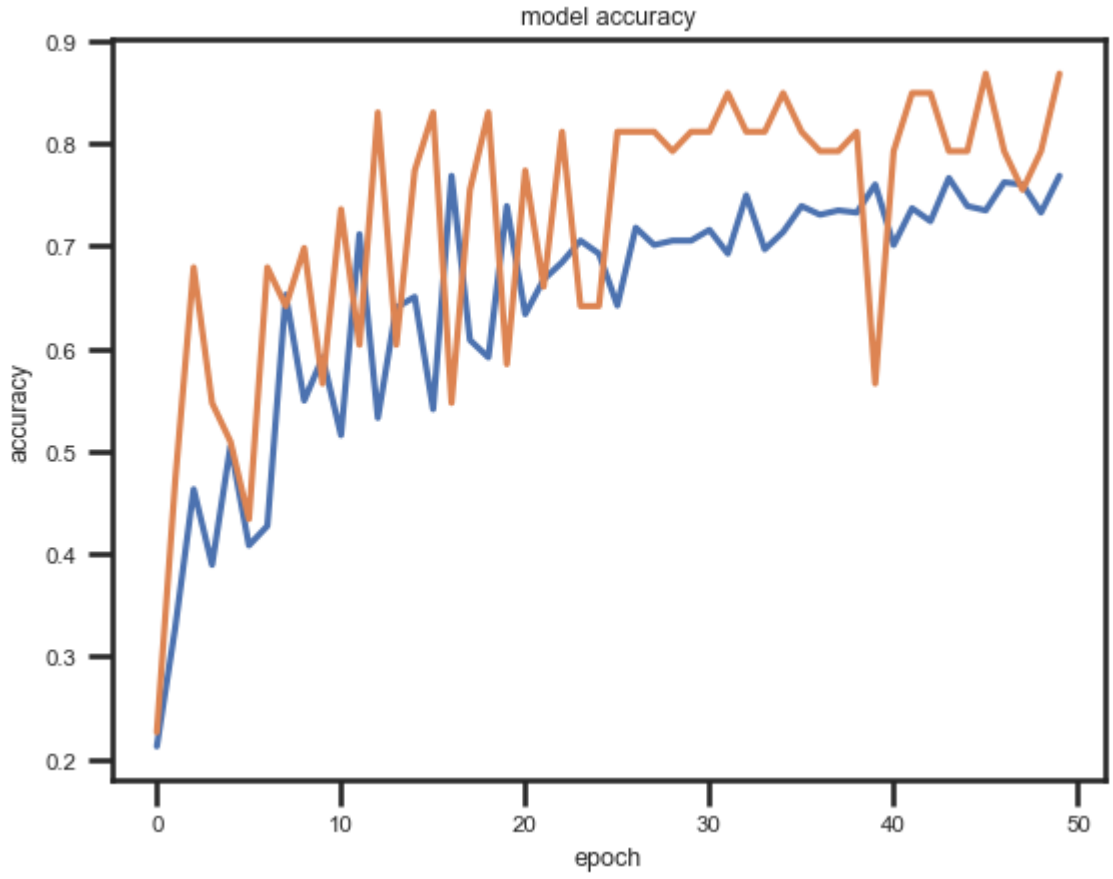
In [18]:
```
#train/test sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=0
y_test["success"].value_counts(normalize=True)
```

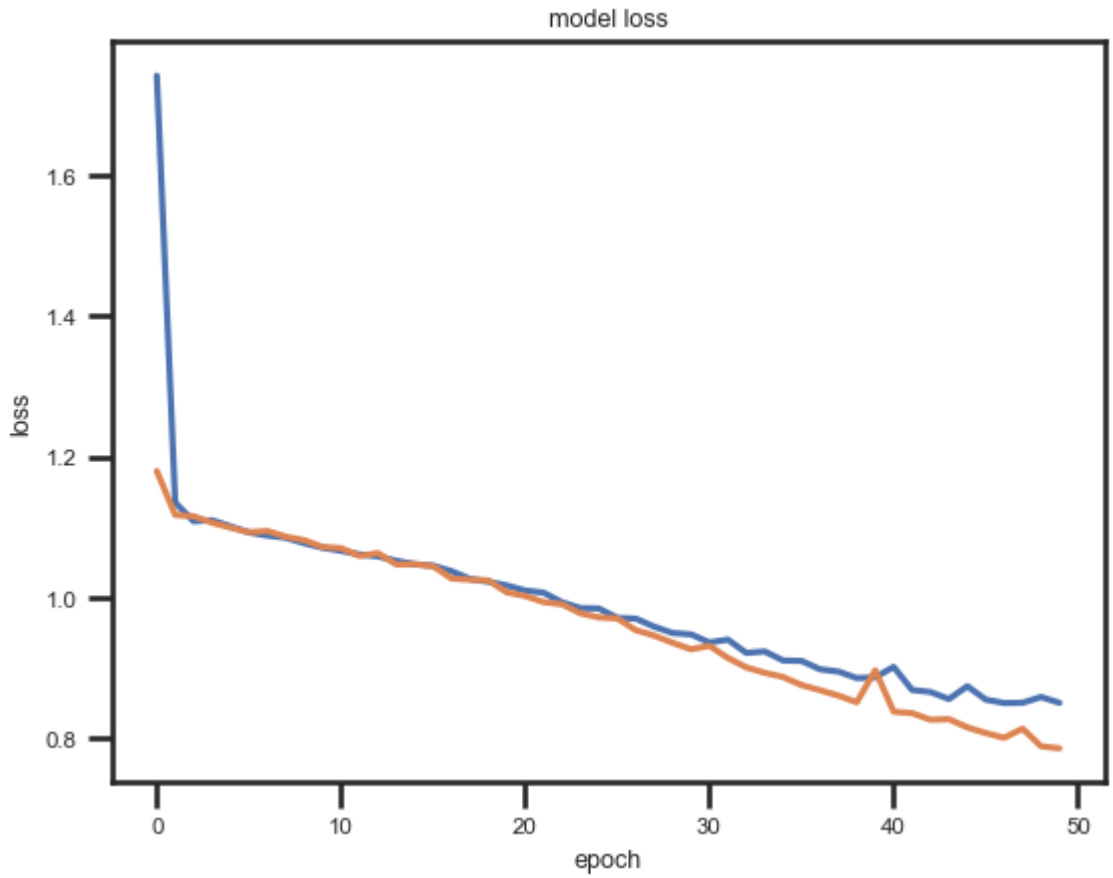
Out[18]:
```
1    0.864407
0    0.135593
Name: success, dtype: float64
```

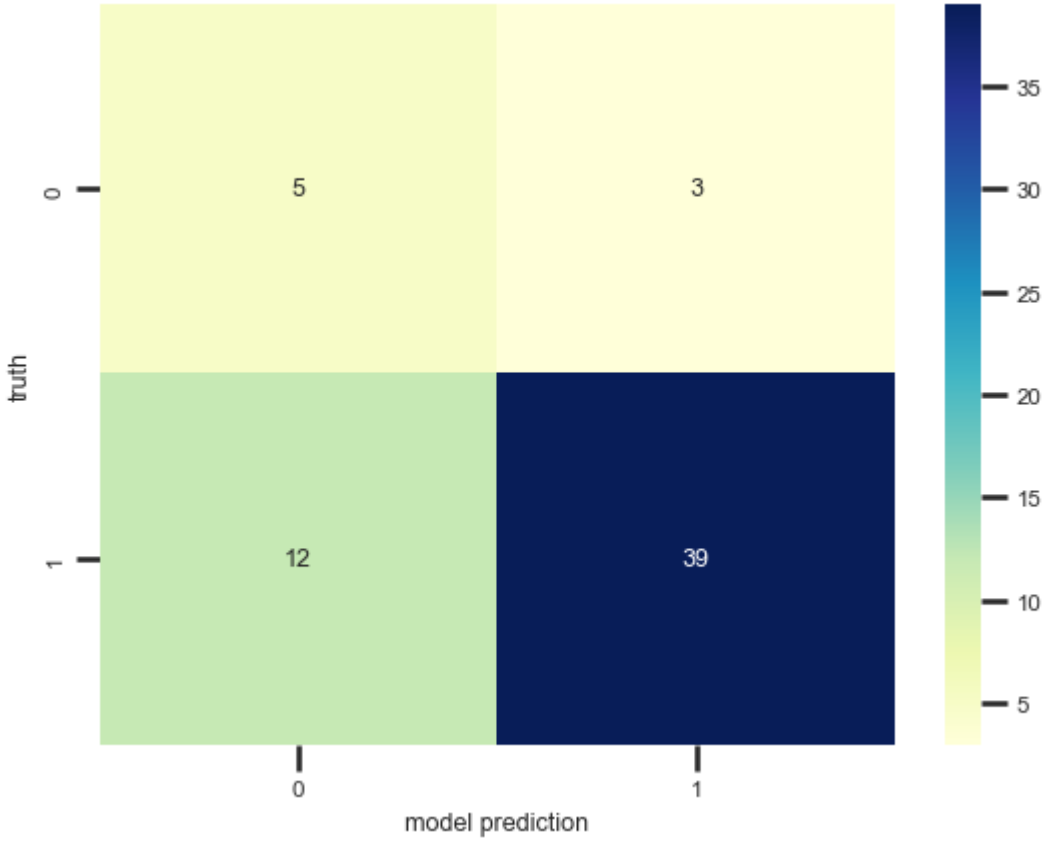
In [19]:
```
fold_no = 1

acc_per_fold = []
loss_per_fold = []

for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(Dense(8, input_dim=x_train.shape[1], activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train, y_train, validation_split = 0.1, class_weight= {0:3.85

    scores = model.evaluate(x_test, y_test, verbose=0)
    print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.m
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])
```

```
    #model accuracy plot
    plt.plot(history.history["accuracy"])
    plt.plot(history.history['val_accuracy'])
    plt.title("model accuracy")
    plt.ylabel("accuracy")
    plt.xlabel("epoch")
    plt.show()

    #model loss plot
    plt.plot(history.history["loss"])
    plt.plot(history.history['val_loss'])
    plt.title("model loss")
    plt.ylabel("loss")
    plt.xlabel("epoch")
    plt.show()

    predictions = model.predict(x_test)
    confusion_matrix = sklearn.metrics.confusion_matrix(y_test, np.rint(predictions))
    sns.heatmap(confusion_matrix, annot=True, cmap="YlGnBu")
    plt.xlabel("model prediction")
    plt.ylabel("truth")
    plt.show()


    # Increase fold number
    fold_no = fold_no + 1

print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
```

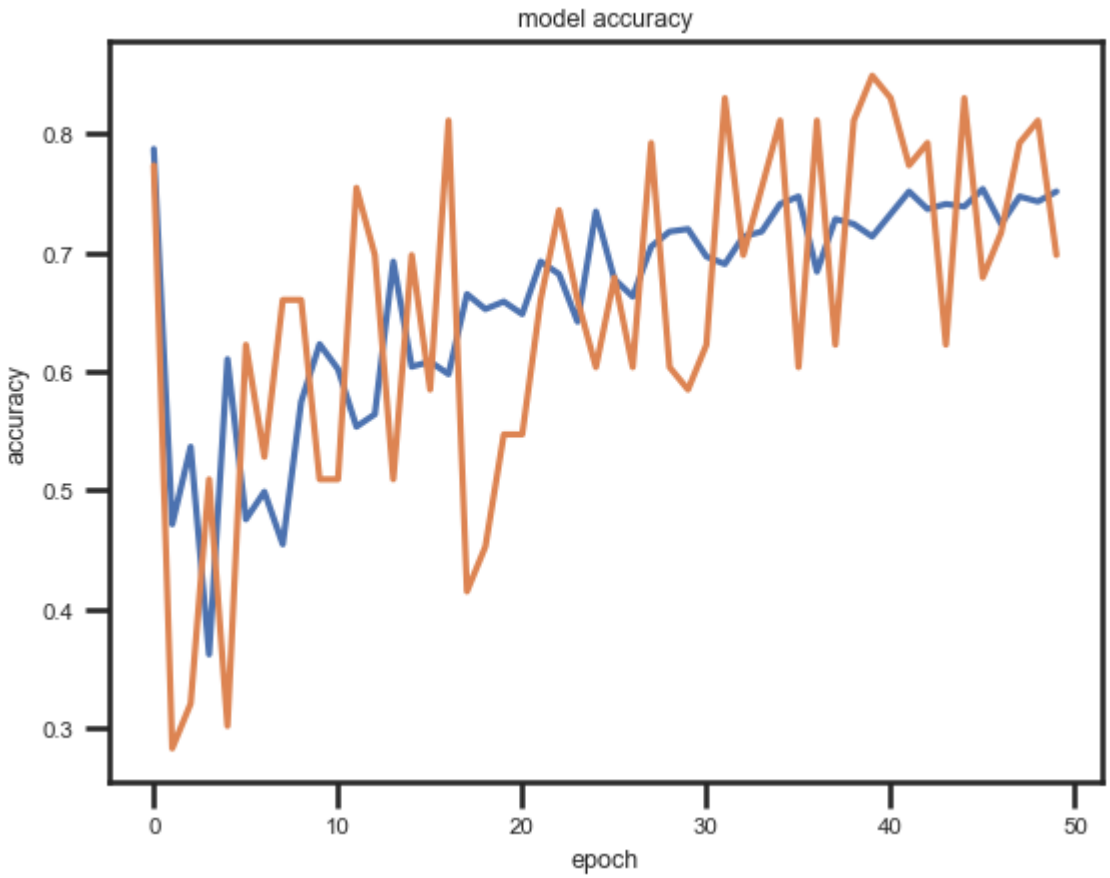
```
Score for fold 1: loss of 0.4984316664226985; accuracy of 74.57627058029175%
```

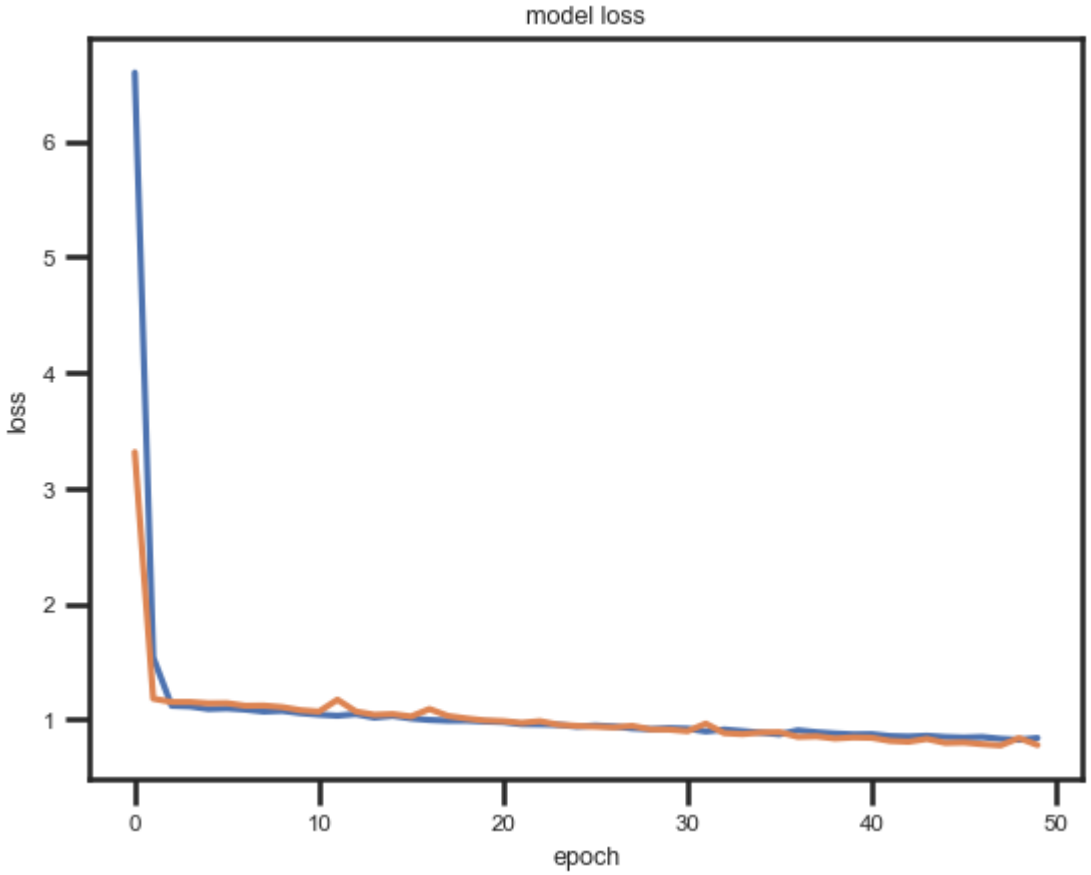


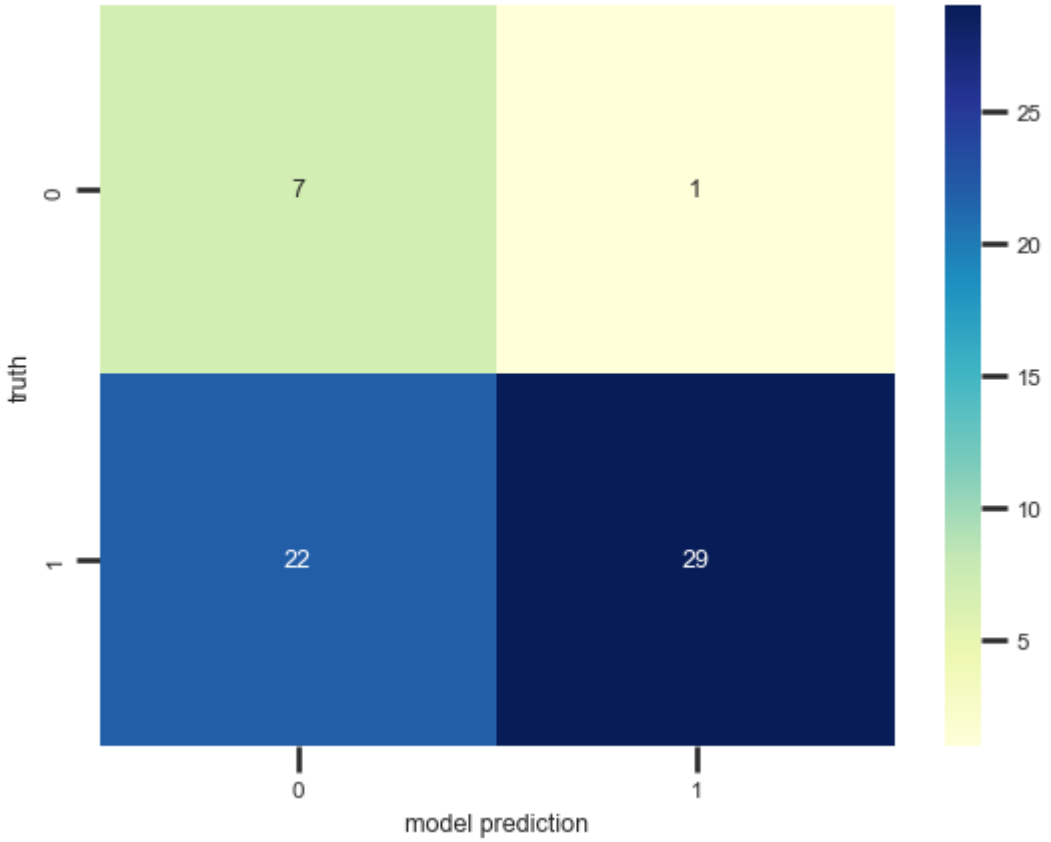




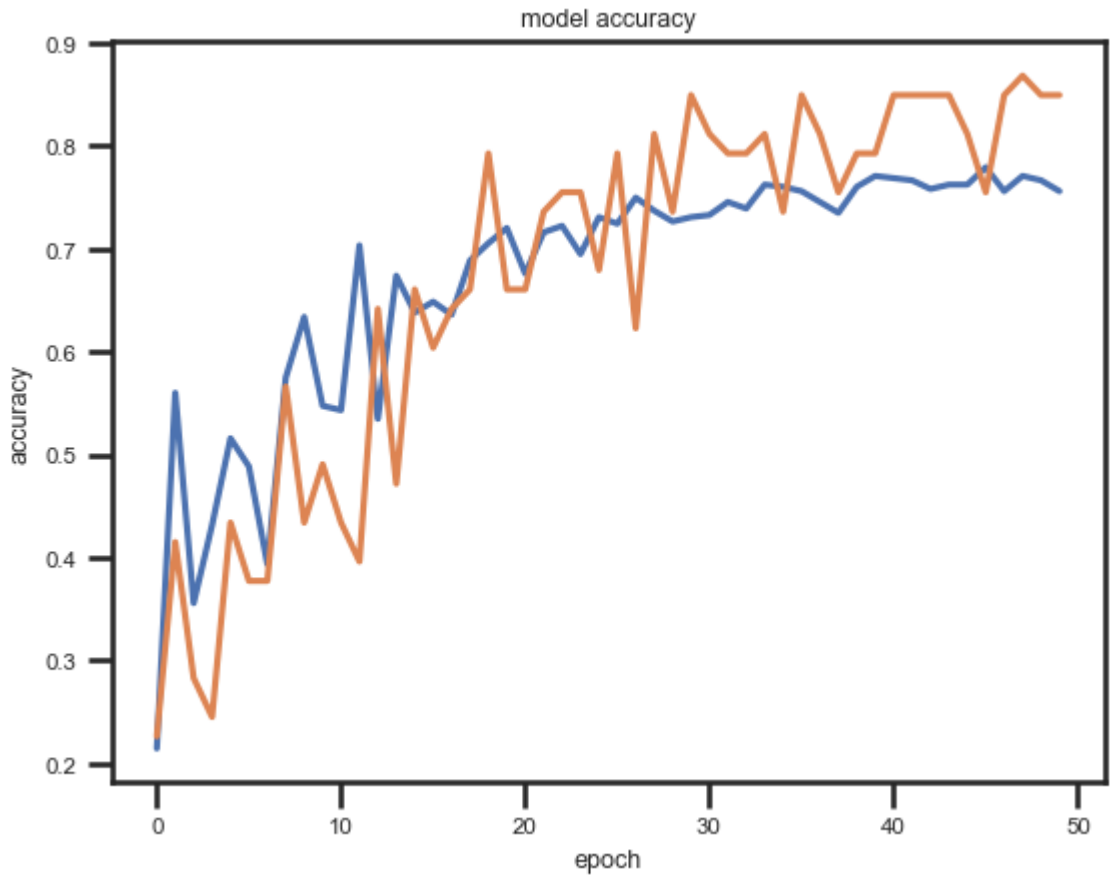
Score for fold 2: loss of 0.6575628700902907; accuracy of 61.01694703102112%

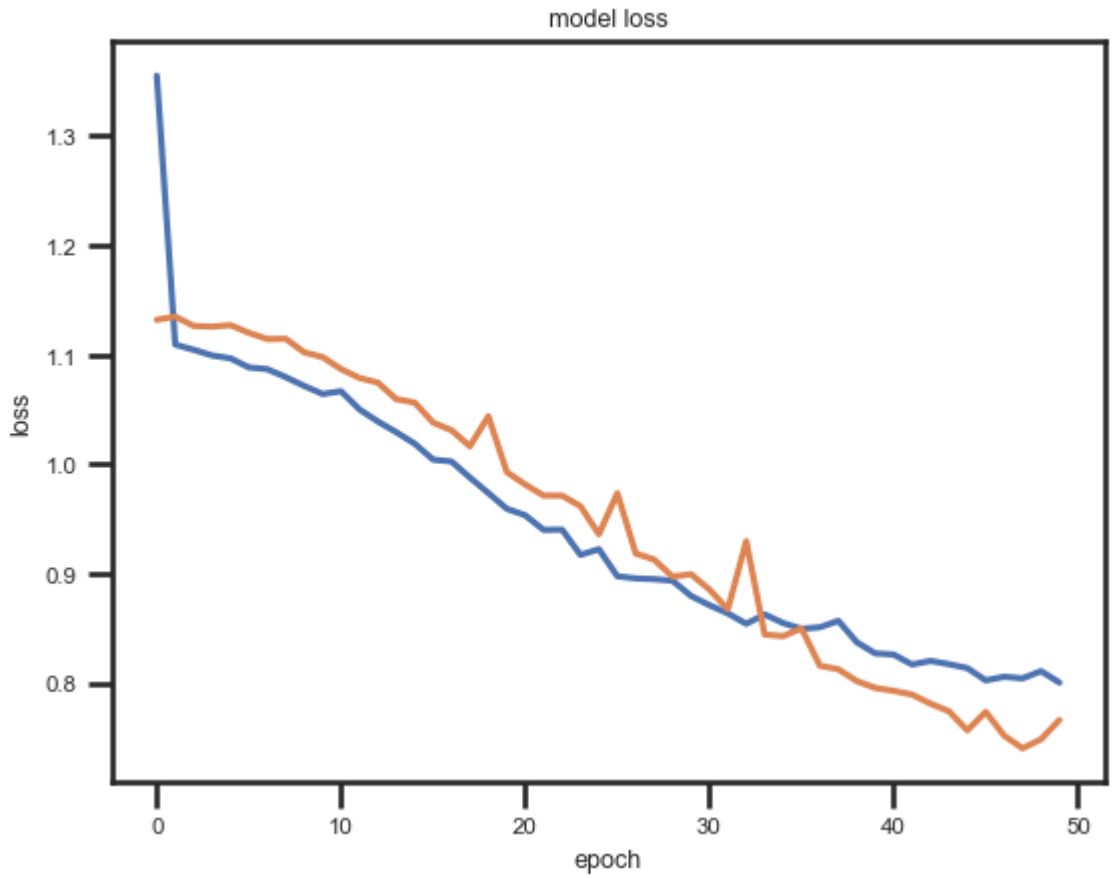

model accuracy
accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50
epoch

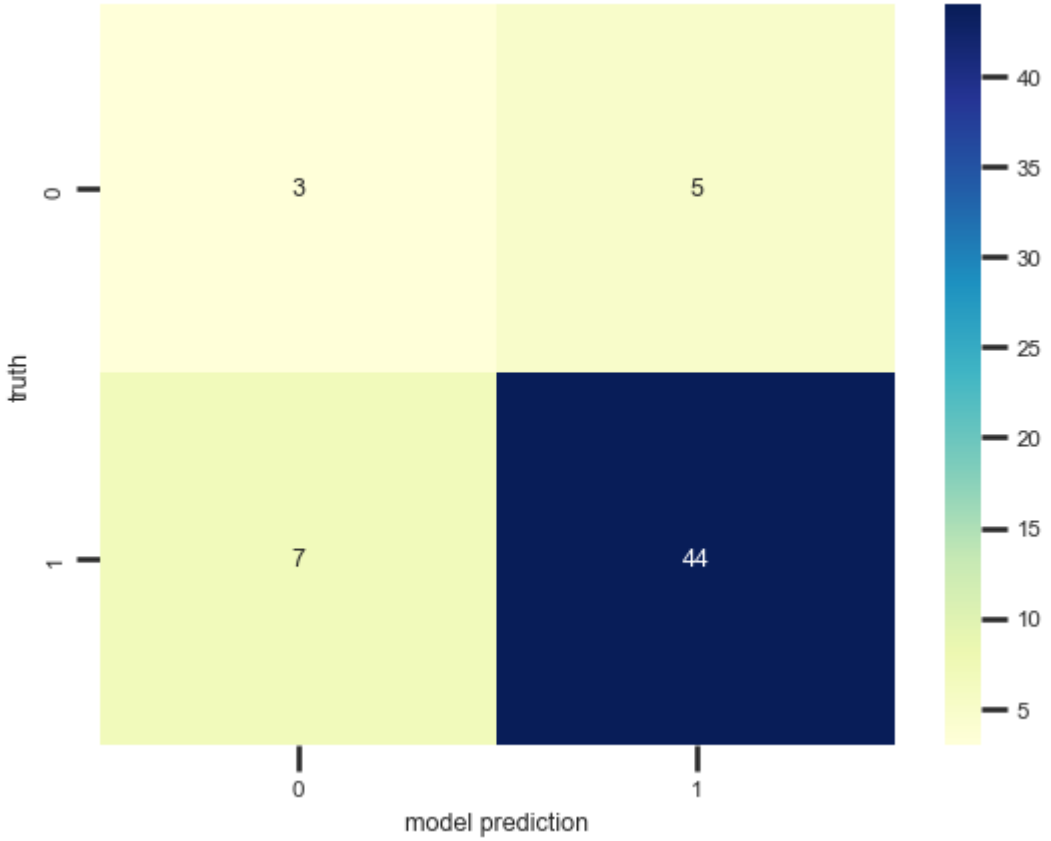

model loss
loss
6
5
4
3
2
1
0
10
20
30
40
50
epoch



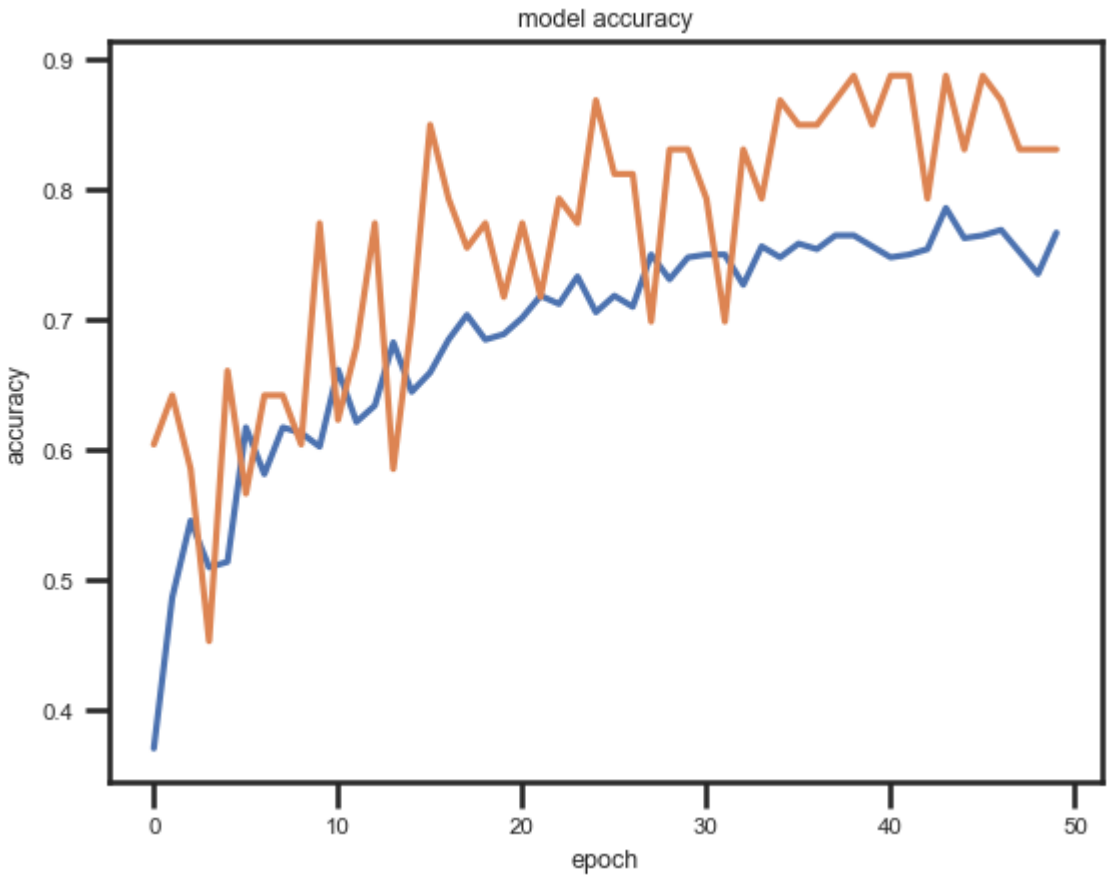

```
Score for fold 3: loss of 0.42802845724558425; accuracy of 79.6610176563263%
```

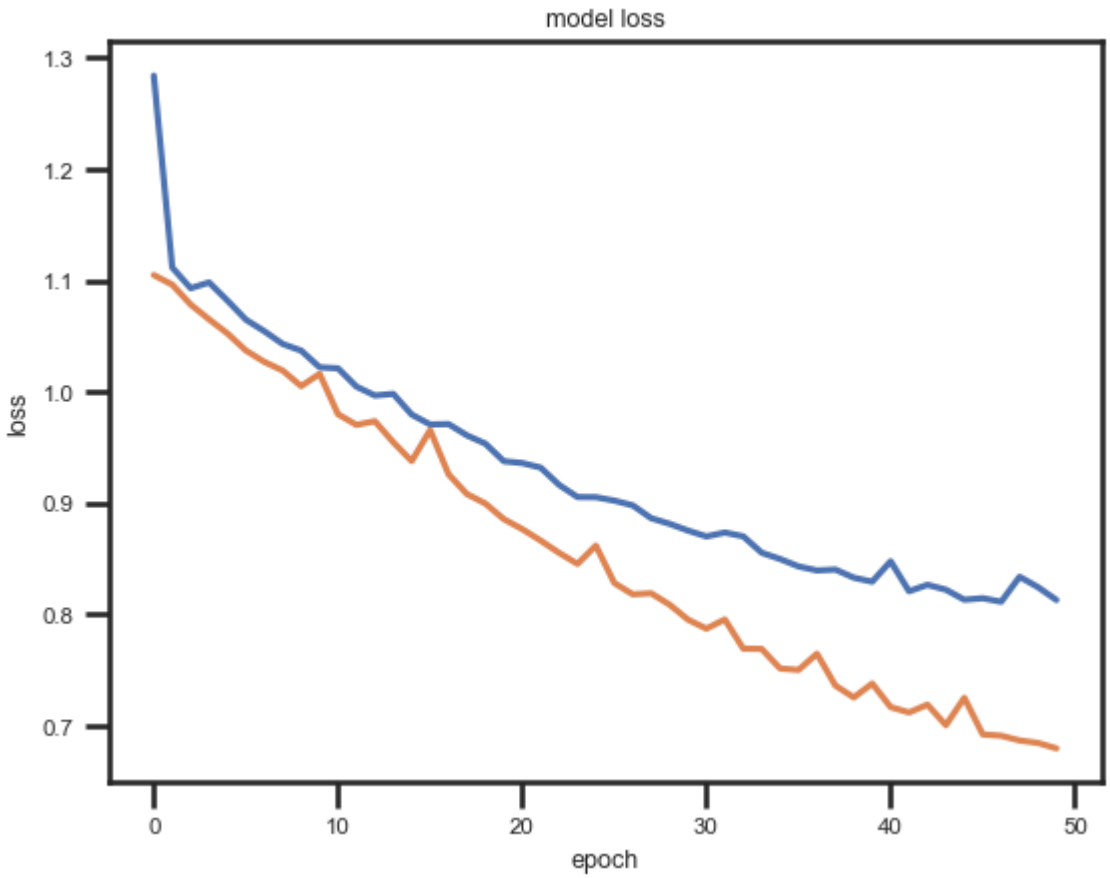


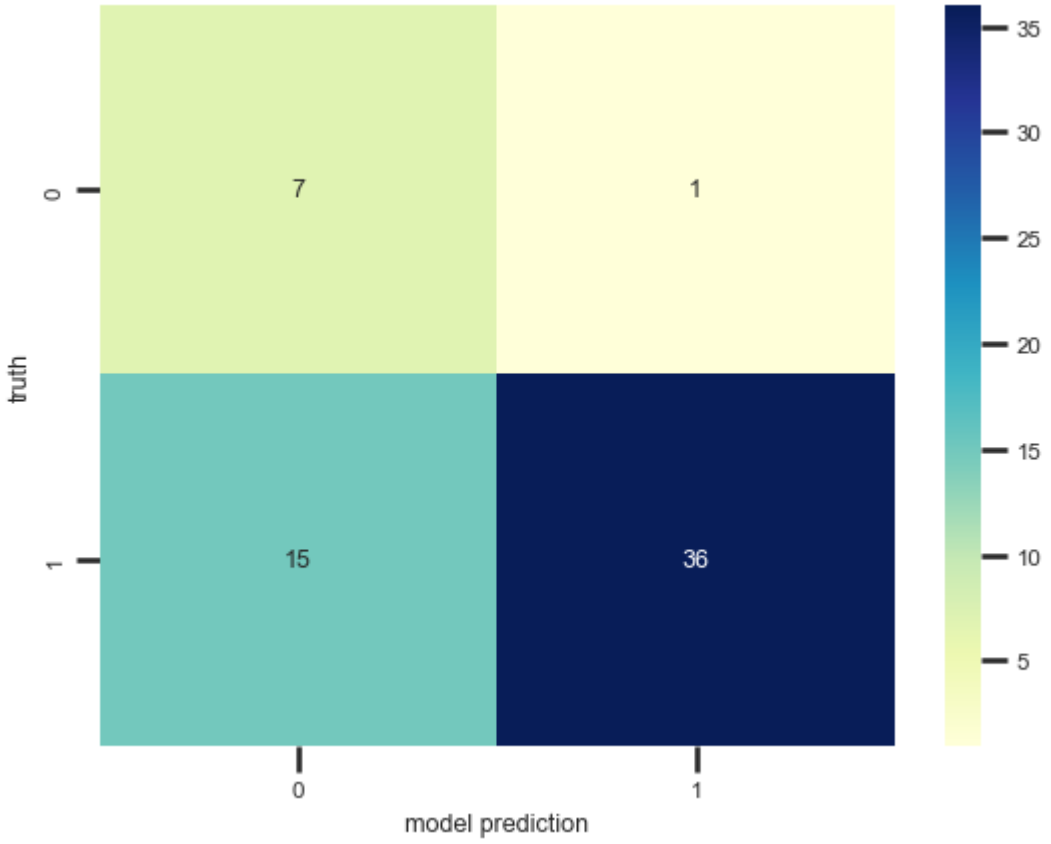




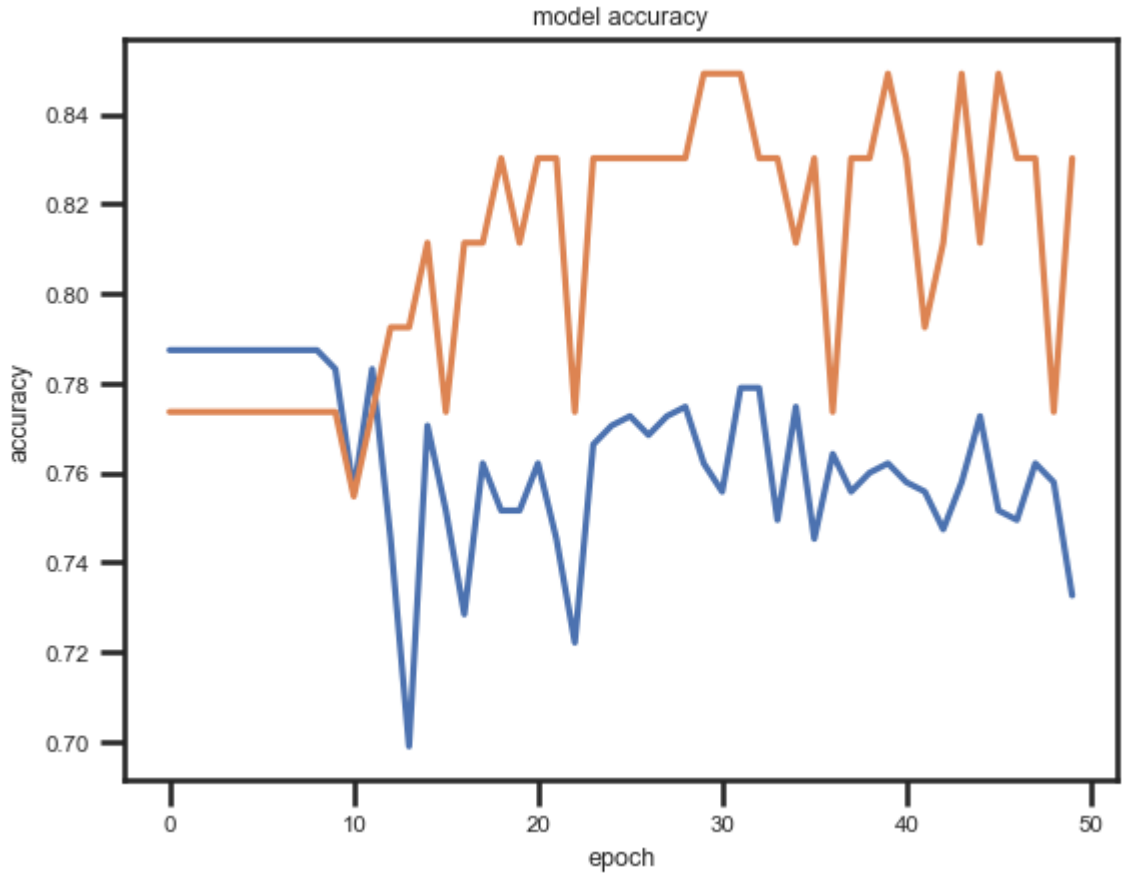
Score for fold 4: loss of 0.598055414224075; accuracy of 72.88135886192322%

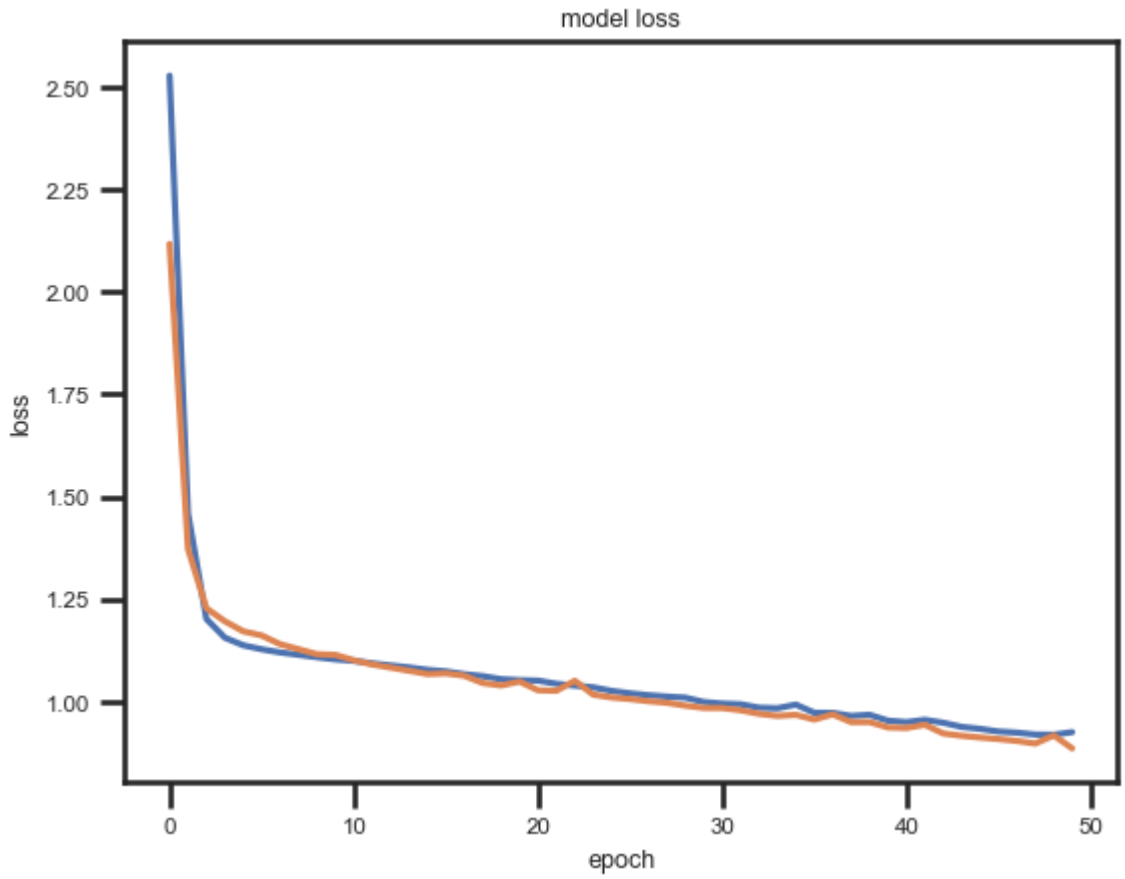

model accuracy
accuracy
epoch
0.9
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50

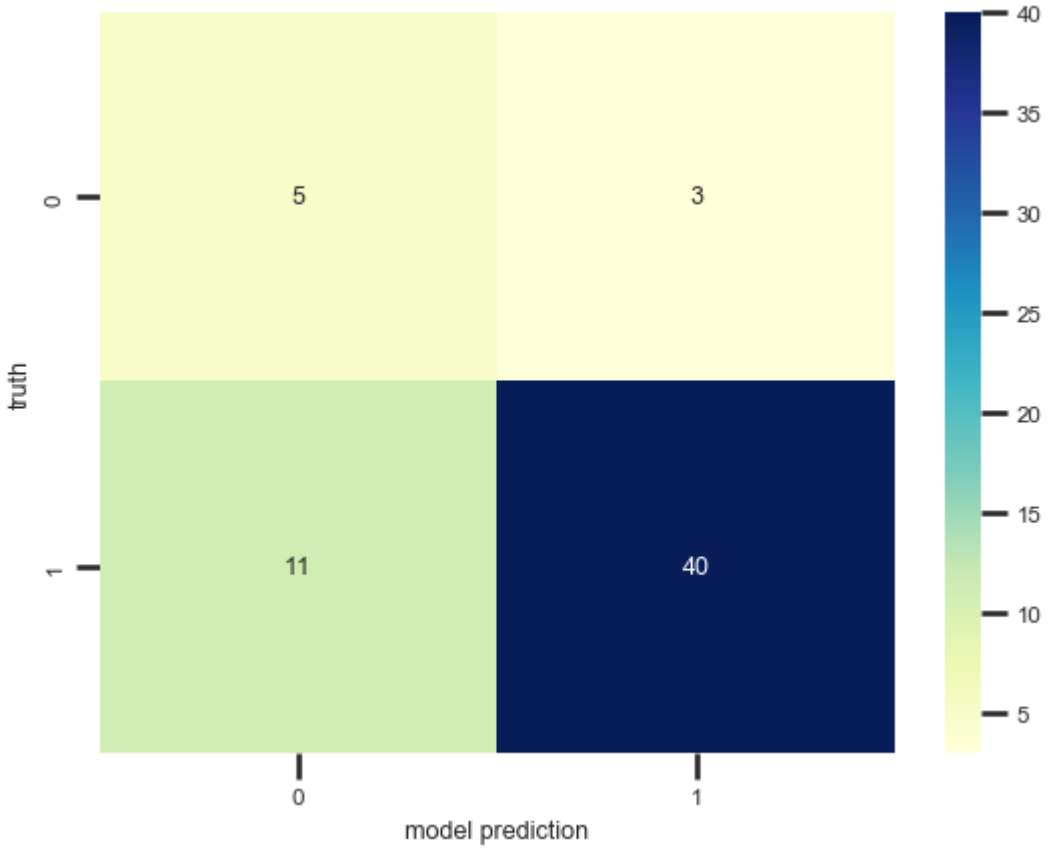

model loss
loss
epoch
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0
10
20
30
40
50




```
Score for fold 5: loss of 0.49431895098443757; accuracy of 76.27118825912476%
```








```
> Accuracy: 72.88135647773743 (+- 6.341793287973726)
> Loss: 0.5352794717934172
```

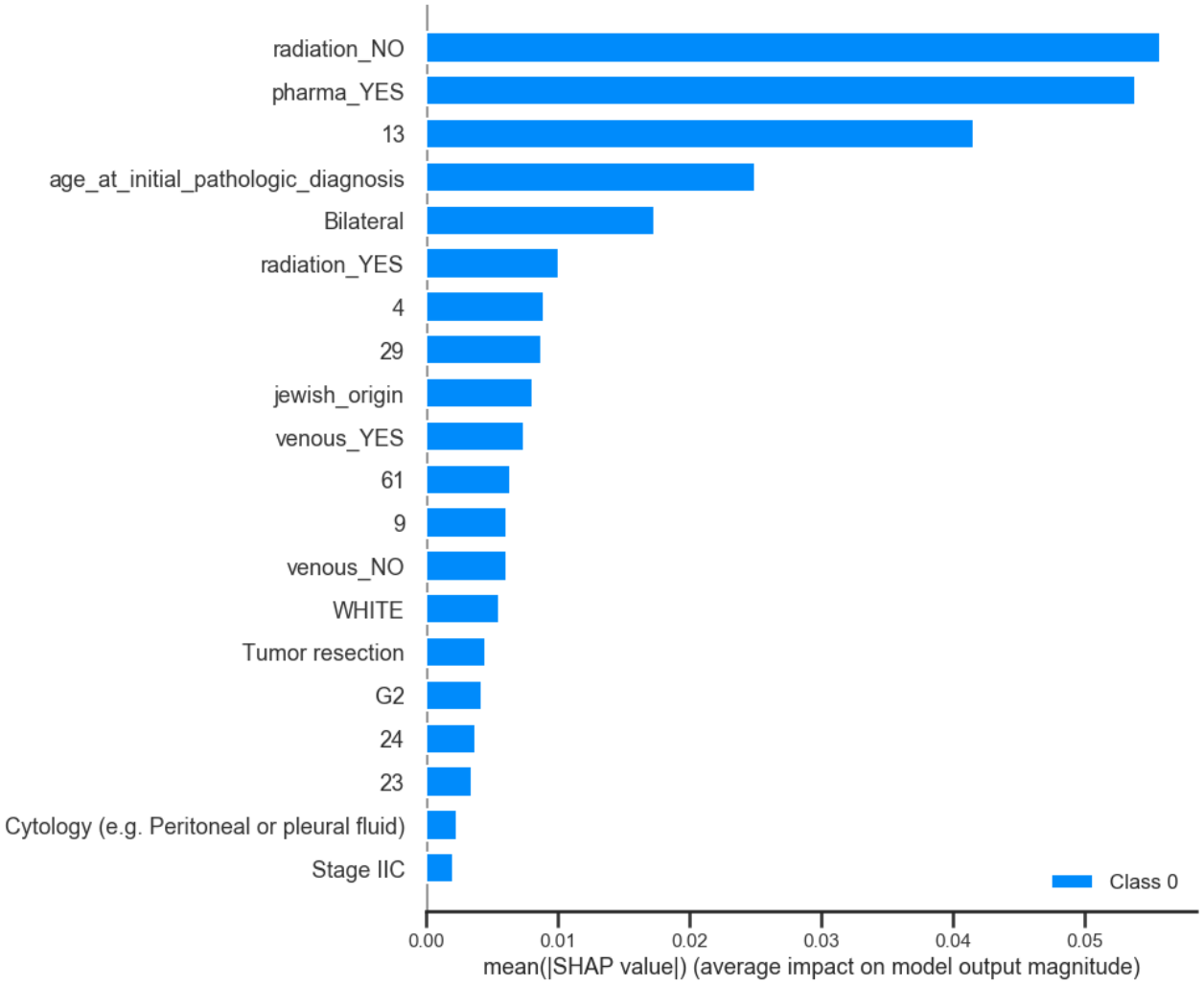
In [20]:

```
explainer = shap.KernelExplainer(model.predict, shap.sample(x_train, 100))
```

```
shap_values = explainer.shap_values(x_test, nsamples=100)
```

In [21]:

```
shap.summary_plot(shap_values, X)
```



In [22]:

```
print(shap_values)
```

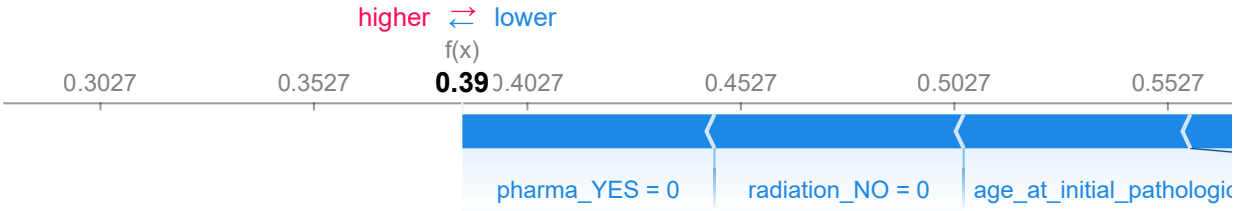
```
[array([[-0.0072755 ,  0.        , -0.05311518, ..., -0.05836914,
         0.        , -0.05905766],
       [ 0.        ,  0.        ,  0.        , ...,  0.05931298,
         0.        ,  0.04524618],
       [ 0.        ,  0.        ,  0.07482617, ...,  0.0551572 ,
        -0.0098782 ,  0.0426547 ],
       ...,
       [ 0.        ,  0.        ,  0.00670715, ...,  0.05617821,
         0.        ,  0.06110961],
       [ 0.        ,  0.        , -0.04646291, ...,  0.04586145,
         0.        ,  0.02875537],
       [ 0.        ,  0.        ,  0.        , ...,  0.06126853,
         0.        ,  0.05541258]])]
```

In [23]:

```
shap.initjs()
```

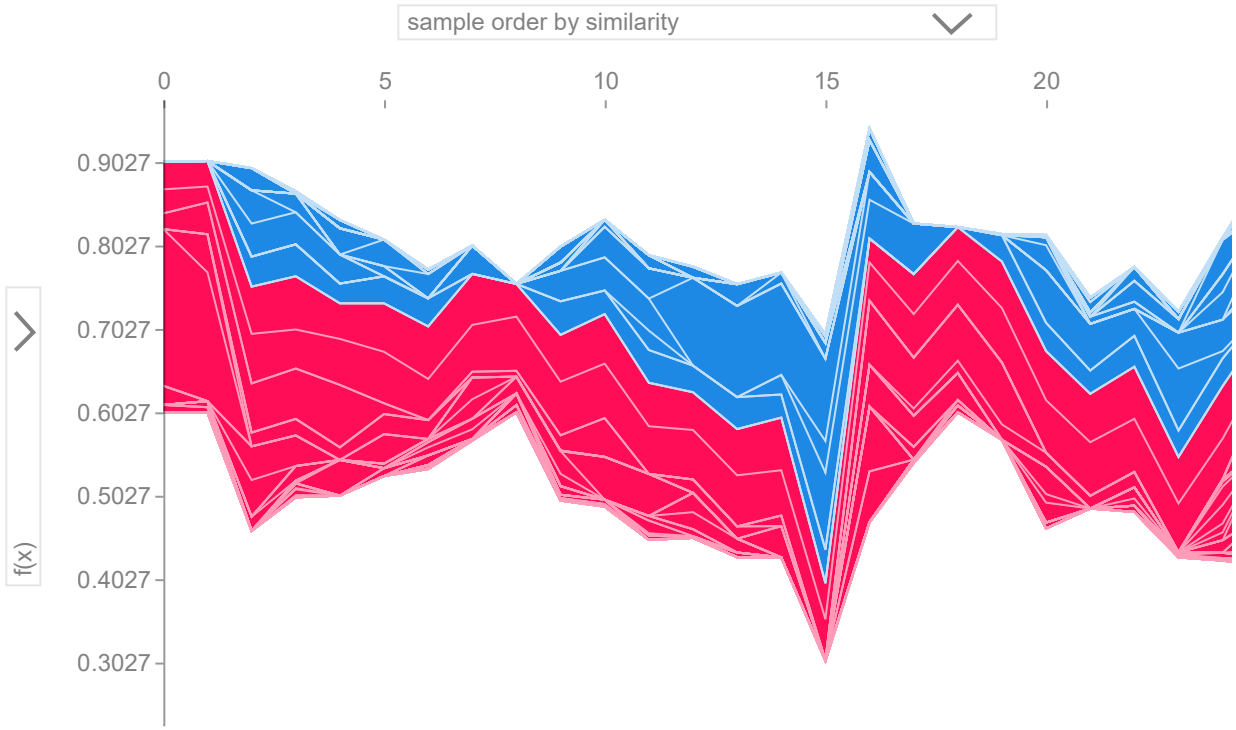
In [24]:
```
shap.force_plot(explainer.expected_value[0], shap_values[0][0,:],
                pd.DataFrame(x_test).iloc[0,:])
```

Out[24]:



In [25]:
```
shap.force_plot(explainer.expected_value[0], shap_values[0], x_test)
```

Out[25]:



In [26]:
```
from sklearn import datasets
from sklearn import svm
```