In [1]:
```
# required packages & models
import os
import sys
import pickle
import warnings
warnings.filterwarnings('ignore')
import csv
import sklearn
import shap
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
from termcolor import colored as cl #text customization

# Model packages
import xgboost
import lightgbm as lgb
from sklearn import *
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree._classes import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import lightgbm as lgb
import catboost as cgb
from sklearn import tree
from shap.plots import waterfall

#import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score


shap.initjs() # load JS visualization code to notebook. SHAP plots won't be displayed without this
```

- Laid out a rough outline of how SHAP would be computed, I thought I would give SHAP methods a try
- Earlier methods work and prove that the model is unpickled and can be used

Things to do:

- Still need to figure out saving results into a file (pickle.dump()), create and save into designated folder
- Figure out how to work TreeExplainer, expected_value function
- Find file with the feature names for corresponding dataset to load into program under 'Load Metadata" section
- Figure out how to display other shap plots such as waterfall, force plot, etc

Notes

- Most of the program is hardcoded to specifically load one of the trained models after running STREAMLINE
- Was able to prove that the model can be unpickled and used for .predict() and .predict*proba*()
- Was able to use model to create SHAP explainers, calculate shap_values for CV0 testing dataset, and display plots
- However, still need to refine the SHAP methods as there were some issues for Decision Tree Classifier
- Was able to display Decision Tree prediction using TreeExplainer or even Explainer....I might be doing something wrong

Run Parameters

In [2]:
```
experiment_path = "/Users/jessicakim/Desktop/STREAMLINE/DemoData/Output/hcc_demo/"

# hardcoded pathways for CVDataset0
train_file_path = '/Users/jessicakim/Desktop/STREAMLINE/DemoData/Output/hcc_demo/hcc-data_example/CVDatasets/hcc-data_
test_file_path = '/Users/jessicakim/Desktop/STREAMLINE/DemoData/Output/hcc_demo/hcc-data_example/CVDatasets/hcc-data_e
```

Load Metadata and Other Necessary Variables

In [3]:
```
jupyterRun = 'True'
# Loading necessary variables specified earlier in the pipeline from metadatafor dataPrep()
file = open(experiment_path + "metadata.pickle", 'rb')
metadata = pickle.load(file)
# file.close()
# print(metadata)

class_label = metadata['Class Label']
instance_label = metadata['Instance Label']
cv_partitions = int(metadata['CV Partitions'])
```

```
# unpickle and load in feature_names found in 'categorical_variables.pickle'
feature_names_file = experiment_path + 'hcc-data_example/exploratory/categorical_variables.pickle'
file = open(feature_names_file , 'rb')
feature_names= pickle.load(file)
file.close()
print('Checking for feature names...\n',feature_names)



alg_file = open(experiment_path + '/' + "algInfo.pickle", 'rb')
algInfo = pickle.load(alg_file)
alg_file.close()
algorithms = []

abbrev = {}
for key in algInfo:  # pickling specific model while also checking for corresponding algInfo
    if key in ["Decision Tree"]: # If that algorithm was used
        algorithms.append(key)
print('\nChecking for algorithms used in STREAMLINE...\n',algorithms)
```

```
Checking for feature names...
 ['Gender', 'Symptoms ', 'Alcohol', 'Hepatitis B Surface Antigen', 'Hepatitis B e Antigen', 'Hepatitis B Core Antibod
y', 'Hepatitis C Virus Antibody', 'Cirrhosis', 'Endemic Countries', 'Smoking', 'Diabetes', 'Obesity', 'Hemochromatosi
s', 'Arterial Hypertension', 'Chronic Renal Insufficiency', 'Human Immunodeficiency Virus', 'Nonalcoholic Steatohepati
tis', 'Esophageal Varices', 'Splenomegaly', 'Portal Hypertension', 'Portal Vein Thrombosis', 'Liver Metastasis', 'Radi
ological Hallmark', 'Performance Status*', 'Encephalopathy degree*', 'Ascites degree*', 'Number of Nodules']

Checking for algorithms used in STREAMLINE...
 ['Decision Tree']
```

load_model(): Load One Trained Model at a Time

In [4]:
```
def load_model():
  # this method will load the pickled model that is chosen by user (hardcoded for time being)
  # should return model object


    model_file = experiment_path + '/hcc-data_example/models/pickledModels/DT_0.pickle'
    file = open(model_file, 'rb')
    trained_model = pickle.load(file)
    file.close()

    return trained_model
```

In [5]:
```
#test load_model() method
load_model()
```

Out[5]:
```
DecisionTreeClassifier(max_depth=17, min_samples_leaf=35, min_samples_split=45,
                       random_state=42)
```

dataPrep(): Loading target CV Training & Testing Sets

In [6]:
```
def dataPrep(train_file_path,instance_label,class_label, test_file_path):
    # Loads target cv training dataset, separates class from features and removes instance labels

    train = pd.read_csv(train_file_path)
    if instance_label != 'None':
        train = train.drop(instance_label,axis=1)
    trainX = train.drop(class_label,axis=1).values
    trainY = train[class_label].values
    del train #memory cleanup


    test = pd.read_csv(test_file_path)
    if instance_label != 'None':
        test = test.drop(instance_label,axis=1)
    testX = pd.DataFrame(test.drop(class_label,axis=1).values)
    testY = pd.DataFrame(test[class_label].values)
    del test #memory cleanup



    return trainX, trainY, testX, testY
```

In [ ]:
```
#test data_prep() method
trainX, trainY,testX, testY= dataPrep(train_file_path,instance_label,class_label, test_file_path)
print('\nChecking testX for CV0 values...\n', testX)
```

SHAP: get_explainer()

- will check if explainer is one of the available ML in STREAMLINE
- if algorithm name matches ['list model names'], create explainers
- return explainer based on given model from parameter

Types of SHAP Explainers

.Explainer()

- Uses Shapley values to explain any machine learning model or python function.
- This is the primary explainer interface for the SHAP library
- It takes any combinationof a model and masker and returns a callable subclass object that implements the particular estimation algorithm that was chosen.

.TreeExplainer()

- Uses Tree SHAP algorithms to explain the output of ensemble tree models.
- Tree SHAP is a fast and exact method to estimate SHAP values for tree models and ensembles of trees, under several different possible assumptions about feature dependence.
- It depends on fast C++implementations either inside an externel model package or in the local compiled C extention.

.LinearExplainer()

- Computes SHAP values for a linear model, optionally accounting for inter-feature correlations.
- This computes the SHAP values for a linear model and can account for the correlations among the input features.
- Assuming features are independent leads to interventional SHAP values which for a linear model are coef[i] * (x[i] - X.mean(0)[i]) for the ith feature.
- If instead we accountfor correlations then we prevent any problems arising from colinearity and share credit among correlated features.
- Accounting for correlations can be computationally challenging, but LinearExplainer uses sampling to estimate a transform that can then be applied to explain any prediction of the model.

In [8]:
```
def get_explainer(model, algorithms, trainX):

    explainer = None
    trained_model = model

#     print(model) # check if model is loaded into method
#     print(algorithms)
    if algorithms[0] in ["Naive Bayes"]:  # checking if algorithms list matches list (temporarily hardcoded)
        explainer = shap.Explainer(trained_model.predict, trainX)


    # dont use model.predict for Linear Explainer (only for Explainer)
    # ^^^ You get a class method error when creating shap plots and values
    if algorithms[0] in ["Logistic Regression"]:
        explainer = shap.LinearExplainer(trained_model, trainX)

#     if algorithms[0] in ['Decision Tree']:
#         explainer = shap.Explainer(trained_model, trainX)  # have not seen examples for Decision Tree

    if algorithms[0] in ['Decision Tree', 'Random Forest', "Extreme Gradient Boosting", "Light Gradient Boosting","Cat
        explainer = shap.TreeExplainer(trained_model)

    return explainer
```

SHAP: compute_shapValues()

In [9]:
```
def compute_shapValues(model, algorithms, explainer, trainX, trainY, testX, testY):
  # this method will calculate shapley values
  # this includes creating expected_values and shap_values
  # returns shap_values (will be called by shap_summary)


    max_evals = max(500, (2 * len(testX)) + 1)   # declares number of permutations for shap.Explainer()
    shap_values = None

    if algorithms[0] in ["Naive Bayes"]:
        shap_values= explainer(testX)  # permutation object cannot use .expected_value function like LR
        print(shap_values)

    if algorithms[0] in ["Logistic Regression"]:
        shap_values = explainer.shap_values(testX)
        print(shap_values)

    if algorithms[0] in ['Decision Tree', 'Random Forest', "Extreme Gradient Boosting", "Light Gradient Boosting","Cat
#         shap_values= explainer.shap_values(testX)
#         i think shap_values() only works for TreeExplainer and LinearExplainer...Explainer for NB is considered a
#         permutation object
        shap_values = explainer.shap_values(testX, approximate=False, check_additivity=False)

        print(shap_values)
#         shap_values = explainer.shap_values(trainX)  --> .shap_values doesnt work for decision tree??????
```

```
    return shap_values
```

SHAP: shap_summary()

NOTES

- XGBOOST MODEL IS COMPATIBLE WITH ALL OF THE LISTED SHAP PLOTS
- RF MODEL NEEDED IT'S OWN IF-STATEMENT FOR NOW BUT WILL CONDENSE FOR CLARITY ADN EFFICIENCY
- STILL NEED TO WORK ON LIGHTGBM, CATBOOST
- GO BACK TO FIX DECISION TREE

FIXES

- Go back to double check shap plot compatibility for global and local importance for linear models
- Work through the DecisionTreeClassifier and compare to other codes out there (if possible)

UPDATES 7/29/22

- ALL given SHAP plots seems to work for NB() when not in a defined function block and if-statement
- Bar, scatter, waterfall, and beeswarm plots don't work for LR(), other plots work fine on LinearExplainer() and shap_values = explainer.shap_values(data)

Plot Types for SHAP v0.41.0

Waterfall

- Plots an explantion of a single prediction as a waterfall plot

Summary (type: violin & bar)

- Summary plots of SHAP values across a whole dataset

Dependence

- Plots the value of the feature on the x-axis and the SHAP value of the same feature on the y-axis
- This shows how the model depends on the given feature, and is like a richer extenstion of the classical parital dependence plots.
- Vertical dispersion of the data points represents interaction effects.
- Grey ticks along the y-axis are data points where the feature's value was NaN.

Force

- Visualize cumulative SHAP values with an additive force layout.

Beeswarm

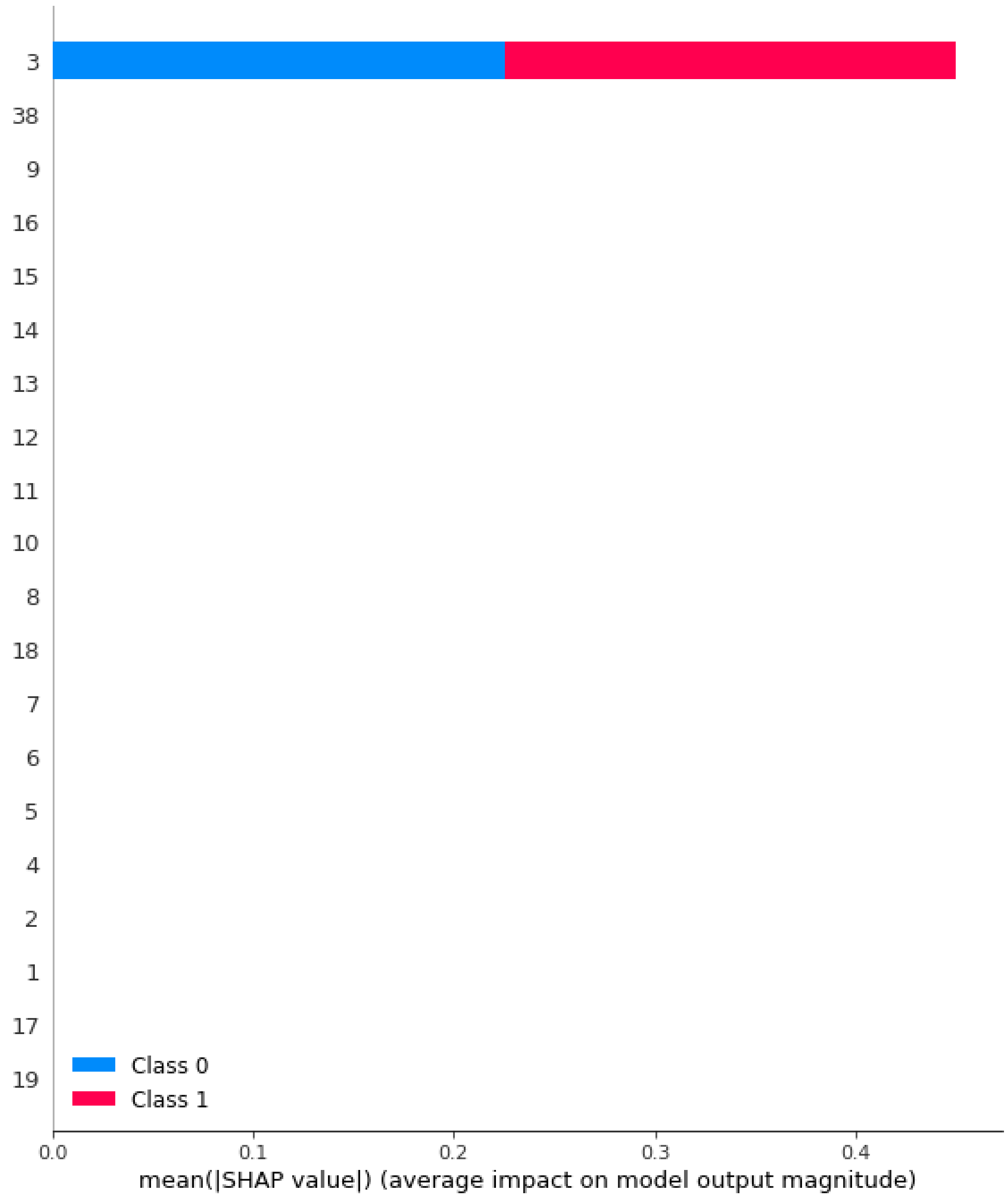
- Summary plots of SHAP values across a whole dataset
- Designed to display an information-dense summary of how the top features in a dataset impact the model's output.

In [13]:
```
# def shap_summary(algorithms, shap_values, explainer, trainX, testX):
#     # retrieve shap_values from previous method
#     # this method will return and display different types of shap plots



#     # checks algorithm in given list to execute shap summaries
#     if algorithms[0] in ["Naive Bayes"]:
#         print('Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
#         shap.summary_plot(shap_values, testX, plot_type='violin')

#         # print('SHAP Bar Plot for Summary Plot for SHAP Values in Class 0 & 1 in Test Set:\n')
#         # shap.plots.bar(shap_values[0]) # doesnt work but should for this...attribute error

#         print('SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set: \n')
#         shap.plots.beeswarm(shap_values, max_display=5)  #max_display allows user to choose # of features to display


#         # print('Waterfall Plot for SHAP Values in Class 0 in Test Set: \n')
#         # shap.plots.waterfall(shap_values[0])  # should work for this model

#         print('\nForce Plot for SHAP Values for Class - in Test Set: \n')
#         shap.force_plot(shap_values[0], testX.iloc[0], feature_names=None, show=True)

#         print('\nForce Plot for SHAP Values for Class 1 in Test Set: \n')
#         shap.force_plot(shap_values[1], testX.iloc[1], feature_names=None, show=True)




# #     # scatter, bar, waterfall, beeswarm plots should work for this model
```

```
# #     # waterfall plot also doesnt work...i get "AttributeError: 'numpy.ndarray' object has no attribute 'base_value
# #       Bar plot should work for this model if using .Explainer() and shap_values = explainer(data)-->
# #       not explainer.shap_values
#     if algorithms[0] in ["Logistic Regression"]:
#         expected_value = explainer.expected_value
#         print(expected_value)


#         print('Summary Plot for SHAP Values in Test Set: \n')
#         shap.summary_plot(shap_values, testX, plot_type='violin')

#         print('SHAP Bar Plot for SHAP Values Test Set: \n')
#         shap.summary_plot(shap_values, testX, plot_type="bar")

#         print('SHAP Decision Plot for SHAP Values in Test Set: \n')
#         shap.decision_plot(expected_value, shap_values)

#         print('SHAP Decision Plot for Single-Prediction in Test Set: \n')
#         shap.decision_plot(expected_value, shap_values[54])

#         print('\nForce Plot for SHAP Values in  Whole Test Set:  \n')
#         shap.force_plot(expected_value, shap_values, testX)

#         print('\nForce Plot for Single-Prediction in Test Set:  \n')
#         shap.force_plot(expected_value, shap_values[54], testX.iloc[54])

#         print('SHAP Bar Plot  for SHAP Values in Test Set:\n')
#         shap.plots.bar(shap_values[0])

        # waterfall plot works for DT() if it uses .Explainer() and shap_vales = explainer(data)
        # instead of using TreeExplainer but other plots listed here work
#     if algorithms[0] in ['Decision Tree']:
expected_value = explainer.expected_value
print(expected_value)

print('Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
#       #tree.tree_plot(testX)  ---> helps display Decision Tree
shap.summary_plot(shap_values, testX, plot_type='bar')

print('\nDecision Plot for SHAP Values from Class 0 in Test Set: \n')
shap.decision_plot(expected_value[0], shap_values[0], feature_names=None)

print('\nDecision Plot for SHAP Values from Class 1 in Test Set: \n')
shap.decision_plot(expected_value[1], shap_values[1], feature_names=None)

print('\nForce Plot for SHAP Values from Class 0 in Test Set: \n')
shap.force_plot(expected_value[0], shap_values[0], feature_names)

print('\nForce Plot for SHAP Values from Class 0 in Test Set: \n')
shap.force_plot(expected_value[1], shap_values[1], feature_names)


#     # RF NOT APPLICABLE TO ANY OTHER SHAP PLOT THAN THE ONES LISTED
#     # WILL CONSIDER USING MULTIOUTPUT SHAP PLOTS B/C RANDOMFOREST IS MULTIOUTPUT
#     if algorithms[0] in ['Random Forest']:
#         expected_value = explainer.expected_value
#         print(expected_value)

#         print('Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
#         shap.summary_plot(shap_values, testX)

#         print('Summary Plot for SHAP Values from Class 0 in Test Set: \n')
#         shap.summary_plot(shap_values[0], testX, plot_type='violin')

#         print('\nDecision Plot for SHAP Values from Class 0 in Test Set: \n')
#         shap.decision_plot(expected_value[0], shap_values[0], feature_names=None)

#         print('\nDecision Plot for SHAP Values from Class 1 in Test Set: \n')
#         shap.decision_plot(expected_value[1], shap_values[1], feature_names=None)

#         print('Dependence Plots for Top 5 Features in Test Set')
#         print('\n This displays SHAP Values from Class 0')
#         top_features = [3, 1, 2, 23, 32]
#         for feature in top_features:
#             shap.dependence_plot(feature, shap_values[0], testX, interaction_index=None)

#         print('\nForce Plot for SHAP Values from Class 0 in Test Set:  --> MAY NOT WORK FOR THIS MODEL\n')
#         shap.force_plot(expected_value[0], shap_values[0], testX.columns.values,  matplotlib = False, show = False)




#     # NOTES:
#     #       XGBOOST MODEL IS COMPATIBLE WITH ALL OF THE LISTED SHAP PLOTS
#     #       RF MODEL NEEDED IT'S OWN IF-STATEMENT FOR NOW BUT WILL CONDENSE FOR CLARITY ADN EFFICIENCY
#     #       STILL NEED TO WORK ON LIGHTGBM, CATBOOST
#     #       GO BACK TO FIX DECISION TREE
#     if algorithms[0] in ["Extreme Gradient Boosting", "Light Gradient Boosting","Category Gradient Boosting"]:
#         expected_value = explainer.expected_value
#         print(expected_value)

#         print('Summary Plot for Top 5 Features in Class 0 & 1 in Test Set: \n')
#         shap.summary_plot(shap_values, plot_type='violin', max_display=5)
```

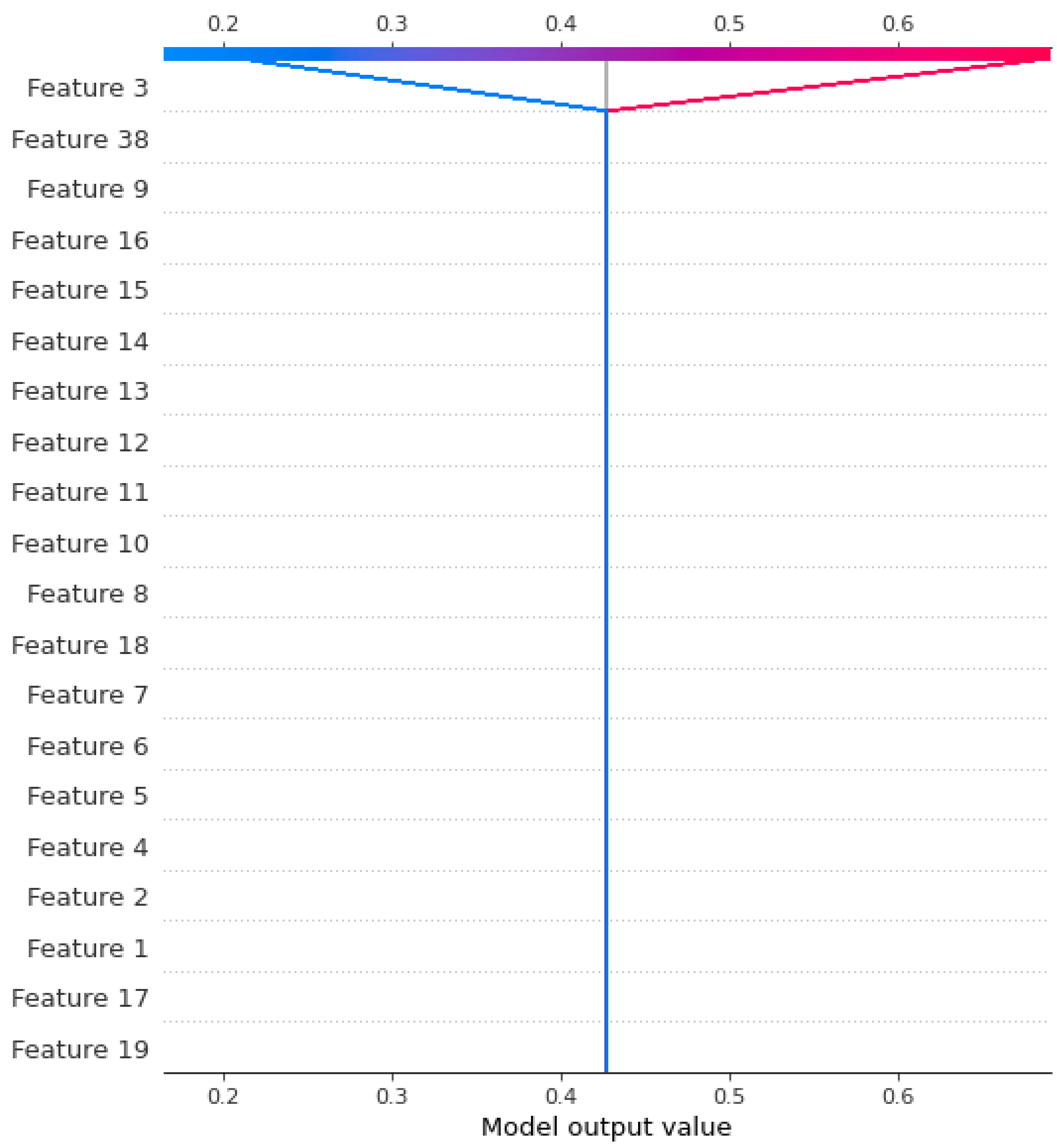
```
#         print('Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
# #         #tree.tree_plot(testX)  ---> helps display Decision Tree
#         shap.summary_plot(shap_values, testX, plot_type='bar')

#         print('\nDecision Plot for SHAP Values from Class 0 in Test Set: \n')
#         shap.decision_plot(expected_value[0], shap_values[0], feature_names=None)

#         print('\nDecision Plot for SHAP Values from Class 1 in Test Set: \n')
#         shap.decision_plot(expected_value[1], shap_values[1], feature_names=None)

# #         print('\nDecision Plot Summary for SHAP Values in Class 0 & 1 in Test Set: \n')
# #         shap.decision_plot(expected_value, shap_values, feature_names=None)


#         print('\nForce Plot for SHAP Values from Class 0 in Test Set: \n')
#         shap.force_plot(expected_value[0], shap_values[0], feature_names)

















#     return [shap_summary, shap_beeswarm, shap_bar]
```

```
[0.57272727 0.42727273]
Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set:
```




```
Decision Plot for SHAP Values from Class 0 in Test Set:
```

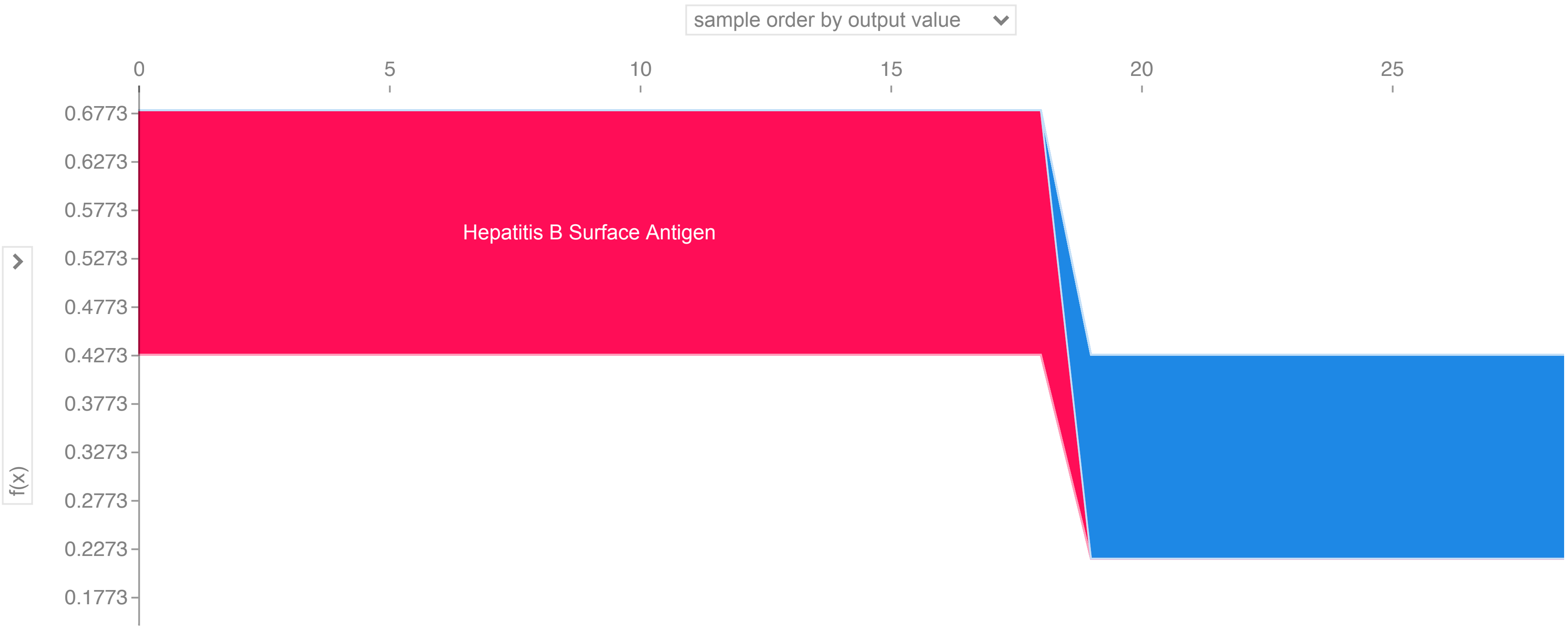

```
Decision Plot for SHAP Values from Class 1 in Test Set:
```




```
Force Plot for SHAP Values from Class 0 in Test Set:
```

```
Force Plot for SHAP Values from Class 0 in Test Set:
```

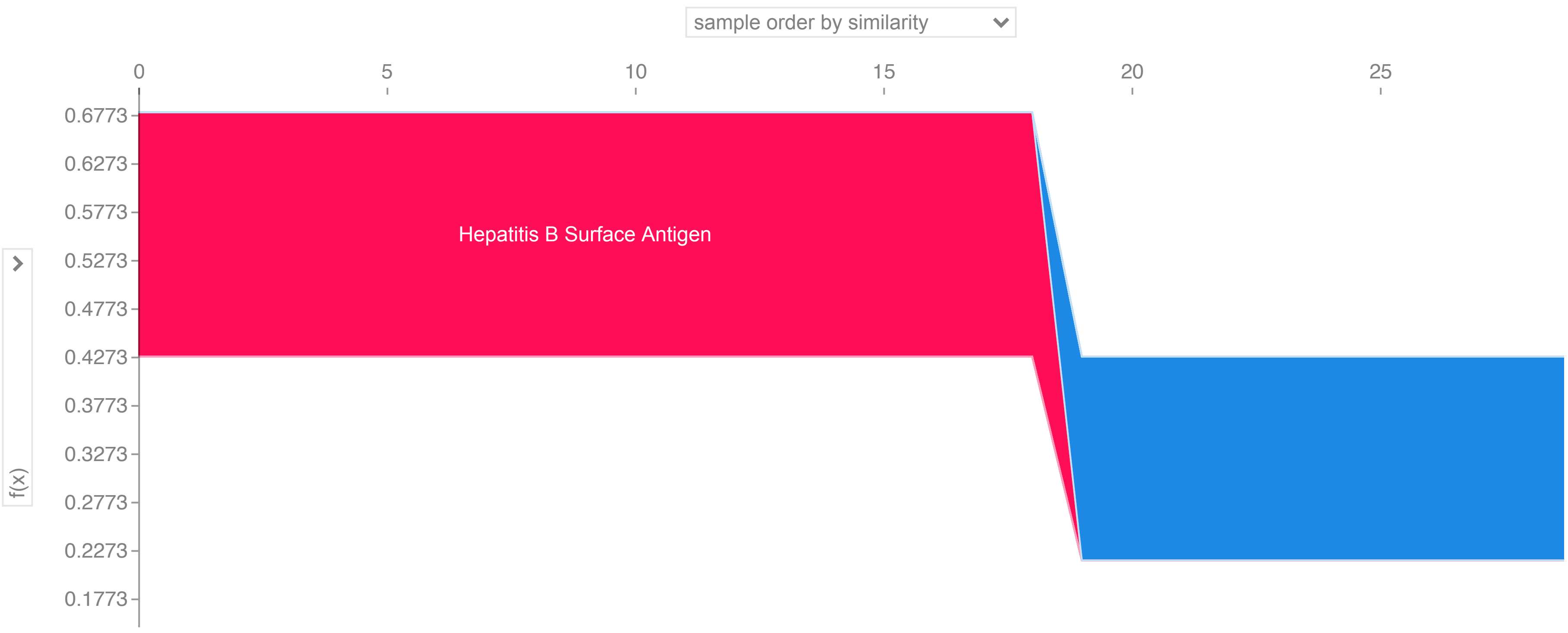
Out[13]:



In [14]:
```
print('\nForce Plot for SHAP Values from Class 0 in Test Set: \n')
shap.force_plot(expected_value[1], shap_values[1], feature_names)
```

```
Force Plot for SHAP Values from Class 0 in Test Set:
```

Out[14]:



Testing All Functions

In [ ]:
```
# testing all methods
model = load_model()  # load Logistic Regression model and algorithms list
print(model)
print(algorithms) # print to make sure variables are separated




y_pred = model.predict(testX)  # calculate model prediction for trainX of CV0
probas_= model.predict_proba(testX)  # calculate model prediction probabilities for trainX of CV0
print('\nPredict_proba_ values: \n', probas_)
print('\n.Predict() values: \n',y_pred)  # print results to show model is being loaded and being used




explainer = get_explainer(model, algorithms, trainX)
print('\nChecking if explainer for model exists...\n', explainer)  # print explainer to check if explainer exists


print('\nChecking if shap values for model is returned...\n')
shap_values = compute_shapValues(model, algorithms, explainer, trainX, trainY, testX, testY)
print('\nChecking if shap plots are returned and consistent...\n')
shap_summary(algorithms, shap_values, explainer, trainX, testX)  # retrieve shap summary plots
```

In [12]:
```
# metrics_file = experiment_path + '/hcc-data_example/model_evaluation/pickled_metrics/DT_CV_0_metrics.pickle'
# file = open(metrics_file, 'rb')
# metrics = pickle.load(file)
# file.close()

# print(metrics)
```

Exception: waterfall_plot requires a scalar expected_value of the model output as the first parameter, but you have passed an array as the first parameter! Try shap.waterfall_plot(explainer.expected_value[0], shap_values[0], X[0]) or for multi-output models try shap.waterfall_plot(explainer.expected_value[0], shap_values[0][0], X[0]).

shap.force_plot(expected_value[0], shap_values[0], testX.columns.values, matplotlib = True, show = False) Error: matplotlib = True is not yet supported for force plots with multiple samples!

In [ ]:

In [ ]: