Goals

- 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
- Be able to iterate through each trained model to it's respective CV dataset, create shap values, generate shap plots
- Be able to store each CV shap values for each model and store in csv file as a DataFrame

```
LR_shap_all_CVs.csv ==>
                        LR_0 --> CV0
                        LR_1 --> CV1
                        LR_2 --> CV2
```

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 **resolved**
- Was able to prove that the model can be unpickled and used for .predict() and .predict*proba*() **resolved**
- Was able to use model to create SHAP explainers, calculate shap_values for CV0 testing dataset, and display plots **resolved**
- However, still need to refine the SHAP methods as there were some issues for Decision Tree Classifier **resolved**
- Was able to display Decision Tree prediction using TreeExplainer or even Explainer....I might be doing something wrong **resolved**
- XGBOOST MODEL IS COMPATIBLE WITH ALL OF THE LISTED SHAP PLOTS **resolved**
- RF MODEL NEEDED IT'S OWN IF-STATEMENT FOR NOW BUT WILL CONDENSE FOR CLARITY ADN EFFICIENCY **resolved**
- STILL NEED TO WORK ON LIGHTGBM, CATBOOST **resolved**
- GO BACK TO FIX DECISION TREE **resolved**

Fix

- Go back to double check shap plot compatibility for global and local importance for linear models **resolved**
- Work through the DecisionTreeClassifier and compare to other codes out there (if possible) **resolved**
- Currently unsure if creating dataframe for each model's shap_values shuold be done in compute_shap_values() or within the nested for-loop in testing cell
- Feature names when displaying shap plots

Updates (refer to 'Next Steps' for more updates)

7/29/22

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

8/02/22

- Plots and shap_values for each trained model in each CV work
- Will focus on section called **'Next Steps'**
  - refer to bottom of Notebook for more details
  - Currently unsure if creating dataframe for each model's shap_values shuold be done in compute_shap_values() or within the nested for-loop in testing cell

In [1]:
```
# required packages & models
import os
import sys
import glob
import pickle
import warnings
warnings.filterwarnings('ignore')
import csv
import sklearn
import shap
import numpy as np
import numpy.typing as npt
```

```
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import itertools
from itertools import chain
from fpdf import FPDF
import collections
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

# Jupyter Notebook Hack: This code ensures that the results of multiple commands within a given cell are all displayed
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"


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

Run Parameters

In [2]:
```
dataset_path = "/Users/jessicakim/Desktop/STREAMLINE/DemoData"
experiment_path = "/Users/jessicakim/Desktop/STREAMLINE/DemoData/Output/hcc_demo"
targetDataName = 'None'

# hardcoded pathways for CVDataset0
# train_file_path = '/hcc-data_example/CVDatasets/'
# test_file_path = '/hcc-data_example/CVDatasets/'
```

Check for Analyzed Datasets and Remove Unecessary Files

In [3]:
```
datasets = os.listdir(experiment_path)
experiment_name = experiment_path.split('/')[-1] #Name of experiment folder

datasets.remove('metadata.csv')
datasets.remove('metadata.pickle')
datasets.remove('algInfo.pickle')

try:
    datasets.remove('jobsCompleted')
except:
    pass
try:
    datasets.remove('UsefulNotebooks')
except:
    pass
try:
    datasets.remove('logs')
    datasets.remove('jobs')
except:
    pass
try:
    datasets.remove('DatasetComparisons') #If it has been run previously (overwrite)
except:
    pass
try:
    datasets.remove('KeyFileCopy') #If it has been run previously (overwrite)
except:
    pass
try:
    datasets.remove('.DS_Store') #If it has been run previously (overwrite)
except:
    pass
try:
    datasets.remove(experiment_name+'_ML_Pipeline_Report.pdf') #If it has been run previously (overwrite)
except:
    pass
```

```
datasets = sorted(datasets) #ensures consistent ordering of datasets
print("Analyzed Datasets: "+str(datasets))
```

```
Analyzed Datasets: ['hcc-data_example', 'hcc-data_example_no_covariates']
```

Load Metadata and Other Necessary Variables

In [4]:

```
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 from original dataset
original_headers = pd.read_csv(experiment_path+"/hcc-data_example/exploratory/OriginalFeatureNames.csv",sep=',').colum
print(original_headers)
# feat_order_map = {feat:i for i, feat in enumerate(original_headers)}
# feat_order = pd.DataFrame.from_dict(feat_order_map, orient ='index')
# print(type(feat_order))
# print(feat_order)
```

```
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 algInfo[key][0]: # If that algorithm was used
        algorithms.append(key)
        abbrev[key] = (algInfo[key][1])



print('\nChecking for algorithms used in STREAMLINE...\n',algorithms)
print('\nChecking for abbrev for algorithms used in STREAMLINE...\n', abbrev)
```

```
['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', 'Age at diagnosis', 'Grams of Alcohol per day', 'Packs of cigarets per year', 'Performance Status
*', 'Encephalopathy degree*', 'Ascites degree*', 'International Normalised Ratio*', 'Alpha-Fetoprotein (ng/mL)', 'Haem
oglobin (g/dL)', 'Mean Corpuscular Volume', 'Leukocytes(G/L)', 'Platelets', 'Albumin (mg/dL)', 'Total Bilirubin(mg/d
L)', 'Alanine transaminase (U/L)', 'Aspartate transaminase (U/L)', 'Gamma glutamyl transferase (U/L)', 'Alkaline phosp
hatase (U/L)', 'Total Proteins (g/dL)', 'Creatinine (mg/dL)', 'Number of Nodules', 'Major dimension of nodule (cm)',
'Direct Bilirubin (mg/dL)', 'Iron', 'Oxygen Saturation (%)', 'Ferritin (ng/mL)']

Checking for algorithms used in STREAMLINE...
 ['Naive Bayes', 'Logistic Regression', 'Decision Tree', 'Random Forest', 'Extreme Gradient Boosting']

Checking for abbrev for algorithms used in STREAMLINE...
 {'Naive Bayes': 'NB', 'Logistic Regression': 'LR', 'Decision Tree': 'DT', 'Random Forest': 'RF', 'Extreme Gradient Bo
osting': 'XGB'}
```

Get Feature Names From Target Dataset

In [5]:

```
# user can choose which csv dataset file to use if more than one was analyzed
target_dataset = '/hcc-data_example.csv'    # default is 'None'
orig_dataset = dataset_path + '/' + target_dataset
# print(orig_dataset)

# feature_names = pd.read_csv(orig_dataset)
# if instance_label != 'None':
#     feature_names = feature_names.drop(instance_label,axis=1)
#     feature_names = feature_names.drop(class_label, axis= 1).columns
# print(feature_names)
```

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)

# get feature names from dataset
trainFeat = list(train.drop(class_label, axis=1).columns)  #note: datatype --> list
set(itertools.chain(*trainFeat))

trainX = pd.DataFrame(train.drop(class_label, axis=1).values)
trainY = pd.DataFrame(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)

# get feature names from dataset
testFeat = list(test.drop(class_label, axis=1).columns)
set(itertools.chain(*testFeat))

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, trainFeat, testFeat
```

In [7]:
```
# # train_path = f"{experiment_path}/{each}/CVDatasets/{each}_CV_{str(cvCount)}_Train.csv"
# # test_path =f"{experiment_path}/{each}/CVDatasets/{each}_CV_{str(cvCount)}_Test.csv"

# train_path = experiment_path + '/hcc-data_example/CVDatasets/hcc-data_example_CV_0_Train.csv'
# test_path_ = experiment_path + '/hcc-data_example/CVDatasets/hcc-data_example_CV_0_Test.csv'
# trainX, trainY,testX, testY, trainFeat, testFeat= dataPrep(train_path,instance_label,class_label, test_path)
# # print(sets)
```

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, abbrev, trainX):

    '''Pass loaded model and abbrev to match appropriate SHAP explainer'''

    '''Must always use training dataset as background data in order to
        evaluate SHAP values for either testing (usually)or training set'''

    explainer = None
    trained_model = model
```

```
    if abbrev in ["NB"]:
        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 abbrev in ["LR"]:
        explainer = shap.LinearExplainer(trained_model, trainX)


    if abbrev in ['DT', 'RF', "XGB", "LGB","CGB"]:
        explainer = shap.TreeExplainer(trained_model)



    return explainer
```

SHAP: compute_shapValues()

NOTES

- Parameter 'X' in this context refers to whatever training or testing dataset that was passed in from the whole run from below
- Mentioned earlier, default run uses training dataset as background data and creates shap values using testing data
- The same follows for feature_names --> either train_feat or test_feat (default) will be passed

In [9]:
```
def compute_shapValues(model, abbrev, explainer, X):

    '''This method will calculate shapley values and store these as a Pandas DataFrame for conversion to csv file
       This includes creating expected_values and shap_values --> returns shap_values (will be called by shap_summary)
    '''

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

    if abbrev in ["NB"]:
        shap_values= explainer(X)  # permutation object cannot use .expected_value function like LR
        print(shap_values)



    if abbrev in ["LR"]:
        shap_values = explainer.shap_values(X)
        print(shap_values)


    # i think shap_values() only works for TreeExplainer and LinearExplainer...Explainer for NB is considered a
    # permutation object
    if abbrev in ['DT', 'RF', "XGB", "LGB","CGB"]:
        shap_values = explainer.shap_values(X, approximate=False, check_additivity=False)
        print(shap_values)



    return shap_values
```

Testing cell below confirms that shap_values are calculated in order based on data features passed in

shap_value feature importance method just orders features & values based on importance but feature:value remains the same

SHAP: shap_summary()

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 [10]:
```
def shap_summary(abbrev, feature_names,shap_values, explainer, X, cvCount):
    '''Retrieve shap_values from previous method;
            this method will return and display different types of shap plots

        Figures for each model CV is saved as a png which will be merged to a
            final summary report for each model
    '''

    save_path = experiment_path + '/hcc-data_example/model_evaluation/shap_values/testResults/shapFigures/'
    # checks algorithm in given list to execute shap summaries
    if abbrev in ["NB"]:

        print('Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
        shap.summary_plot(shap_values, X, feature_names, plot_type='violin', show=False)
            # print('SHAP Bar Plot for Summary Plot for SHAP Values in Class 0 & 1 in Test Set:\n')
#         shap.plots.bar(shap_values.values) # 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, show=False)  #max_display allows user to choose # of features

            # print('Waterfall Plot for SHAP Values in Class 0 in Test Set: \n')
#         shap.waterfall_plot(shap_values, max_display=5, show=True)  # should work for this model


    # #     # 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_v
    # #       Bar plot should work for this model if using .Explainer() and shap_values = explainer(data)-->
    # #       not explainer.shap_values
    elif abbrev in ["LR", 'XGB']:

        expected_value = explainer.expected_value
        print('Expected value for {}: {}'.format(abbrev, expected_value))

        print('Summary Plot for SHAP Values in Test Set: \n')
        shap.summary_plot(shap_values, X, feature_names, plot_type='violin', show=False)
        plt.savefig(save_path+abbrev+'_'+str(cvCount)+'shapSummaryPlot.png', bbox_inches='tight')
        plt.close()

        print('SHAP Bar Plot for SHAP Values Test Set: \n')
        shap.summary_plot(shap_values, X, feature_names, plot_type="bar", show=False)
        plt.savefig(save_path+abbrev+'_'+str(cvCount)+'shapSummaryBarPlot.png', bbox_inches='tight')
        plt.close()


        print('SHAP Decision Plot for SHAP Values in Test Set: \n')
        shap.decision_plot(expected_value, shap_values, feature_names, show=False)
        plt.savefig(save_path+abbrev+'_'+str(cvCount)+'shapDecisionPlot.png', bbox_inches='tight')
        plt.close()


        print('SHAP Decision Plot for Single-Prediction in Test Set: \n')
        shap.decision_plot(expected_value, shap_values[54], feature_names, show=False)
        plt.savefig(save_path+abbrev+'_'+str(cvCount)+'shapDecisionPlot_singlePredict.png', bbox_inches='tight')
        plt.close()



            # waterfall plot works for DT() if it uses .Explainer() and shap_vales = explainer(data)
            # instead of using TreeExplainer but other plots listed here work
    elif abbrev in ['DT', 'RF', 'LGB','CGB']:
        expected_value = explainer.expected_value
        print('Expected value for {}: {}'.format(abbrev, 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, X, feature_names, plot_type='bar', class_names=['0', '1'], show=False)


        print('\nDecision Plot for SHAP Values from Class 0 in Test Set: \n')
        shap.decision_plot(expected_value[0], shap_values[0], feature_names=feature_names, show=False)

        print('\nDecision Plot for SHAP Values from Class 1 in Test Set: \n')
        shap.decision_plot(expected_value[1], shap_values[1], feature_names=feature_names, show=False)
```

This cell below might be to create summary reports

In [ ]:
```
```

In [11]:
```
# def run_force_plot(model, abbrev, explainer, shap_values, trainX, testX, run = True):
#     if abbrev in ['NB']:
```

```
#         print('\nForce Plot for {} SHAP Values from Class 0 in Test Set: \n'.format(abbrev))
#         shap.force_plot(shap_values[0], testX.iloc[0], feature_names=feature_names, show=True)

#         print('\nForce Plot for {} SHAP Values from Class 0 in Test Set: \n'.format(abbrev))
#         shap.force_plot(shap_values[1], testX.iloc[1], feature_names=feature_names, show=True)


#     elif abbrev in ['LR', 'XGB']:
#         print('\nChecking if shap plots are returned and consistent...\n')
#         summary = shap_summary(algorithms, shap_values, explainer, trainX, testX)  # retrieve shap summary plots

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


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

#         print('\nForce Plot for {} SHAP Values from Class 1 in Test Set: \n'.format(abbrev))
#         shap.force_plot(explainer.expected_value[1], shap_values[1], feature_names=feature_names)
```

^^^ fix later ... may want to keep this to create force plots and save results

In [12]:
```
def shap_feature_ranking(abbrev, shap_values, X, feature_names):  # 'X' and 'feature_names' argument is whichever test

    '''Calculate the average of the absolute SHAP values for each feature and use it to show
       which features were the most important when making a prediction'''

    if abbrev in ['NB']:
        feature_order = np.argsort(np.mean(np.abs(shap_values.values), axis=0))
        df = pd.DataFrame({"Features": [feature_names[i] for i in feature_order][::-1],"Importance": [ np.mean(np.abs(

    elif abbrev in ['LR','LGB', 'XGB', 'CGB']: #LR cant use shap_values.values
        feature_order = np.argsort(np.mean(np.abs(shap_values), axis=0))
        df = pd.DataFrame({"Features": [feature_names[i] for i in feature_order][::-1],"Importance": [ np.mean(np.abs(


    else: # For multiclass models (can be used for NB)..Loops through Class 0 and Class 1
         #  Sums up the shap average values form both classes to get the shap average for the whole CV for the model

        c_idxs = []
        columns = feature_names
        for column in range(0, (len(columns))):
            if isinstance(shap_values, list):
                c_idxs.append(X.columns.get_loc(column))
                means = [np.abs(shap_values[class_][:, c_idxs]).mean(axis=0) for class_ in range(len(shap_values))]
                shap_means = np.sum(np.column_stack(means), 1)
            else:                                   # Else there is only one 2D array of shap values
                assert len(shap_values.shape) == 2, 'Expected two-dimensional shap values array.'
                shap_means = np.abs(shap_values).mean(axis=0)
        df = pd.DataFrame({'Features': feature_names, 'Importance': shap_means}).sort_values(by='Importance', ascendin
        df.index += 1

    return df
```

In [126…
```
def save_shap(abbrev, shap_values, test_feat, original_headers, cvCount): # 'df' parameter is the dataframe returned f
    '''Create a new dataframe that stores the model's shap feature importance values over each CV
            and combines with features from original dataset'''

    FI_all = []
    temp_list = []
#     df = pd.DataFrame(shap_values.values)
#     df
    shap_vals = [] # store shap_values.values in a new list by creating each row in original shap_values as a list
    for row in range(0, len(shap_values.values)):
        shap_vals.append(list(shap_values.values[row]))

    print(len(shap_vals))

#     for i in shap_vals:
#         for val in i:
#             print(f'{val}\n')

    headers = pd.read_csv(experiment_path +  '/hcc-data_example/CVDatasets/hcc-data_example_CV_0_Test.csv').columns.va
    if instance_label != 'None':
        headers.remove(instance_label)
    headers.remove(class_label)


#     count = 0
    if abbrev in ['NB']:
        for each in original_headers:
            for each in headers:
                if each in headers:
                    index = headers.index(each)
#                     print(f'{each} is {each}\n')
#                     print(f'Index is {index}')
#                     print(f'Shap value is {shap_vals[index]}')
# #                     print(f'{each} value is {shap_vals[index]}')
```

```
                    temp_list.append(shap_vals[index])
                else:
                    temp_list.append(0)

            FI_all.append(temp_list)

#     for row in temp_list:
#         print(f'{row}\n')

#     else:
#         for each in original_headers:
#             if each in test_feat:
#                 index = test_feat.index(each)
#                 temp_list.append(shap_values[index])
# #                 temp_list.append(shap_values[index][index])
#             else:
#                 temp_list.append(0)
#         FI_all.append(temp_list)



    dr = pd.DataFrame(FI_all)
    display(dr)
#     if not os.path.exists(experiment_path+'/hcc-data_example/model_evaluation/shap_values/testResults/'):
#         os.mkdir(experiment_path+'/hcc-data_example/model_evaluation/shap_values/testResults/')
#     file_path = experiment_path+'/hcc-data_example/model_evaluation/shap_values/testResults/'+abbrev+"_FI.csv"
#     dr.to_csv(file_path, header=original_headers, index=False)
```

In [127…

```
# # ^^^^^
result_file = experiment_path+ '/hcc-data_example/models/pickledModels/NB_0.pickle'
file = open(result_file, 'rb')
model = pickle.load(file)
file.close()
print('\nChecking if correct model is loaded...\n', model)

train_path = experiment_path +  '/hcc-data_example/CVDatasets/hcc-data_example_CV_0_Train.csv'
test_path = experiment_path +  '/hcc-data_example/CVDatasets/hcc-data_example_CV_0_Test.csv'
trainX, trainY,testX, testY, trainFeat, testFeat = dataPrep(train_path,instance_label,class_label, test_path)


explainer = get_explainer(model, 'NB', trainX)
shap_values = compute_shapValues(model, 'NB', explainer, testX)
# print(shap_values.values)

original_headers = pd.read_csv(experiment_path+"/hcc-data_example/exploratory/OriginalFeatureNames.csv",sep=',').colum
# # feat_order_map = {feat:i for i, feat in enumerate(original_headers)}
# # feat_order = pd.DataFrame.from_dict(feat_order_map, orient ='index')
# # # print(type(feat_order))
# # # print(feat_order)
# # print(shap_values.values[0][:])

# # #path to save SHAP FI value results
filepath = experiment_path+"/hcc-data_example/model_evaluation/shap_values/testResults/"
# for cvCount in range(0, cv_partitions):

save_shap('NB', shap_values, testFeat, original_headers, 0)
```

```
Checking if correct model is loaded...
 GaussianNB()
.values =
array([[ 0.        , -0.03083333, -0.02      , ..., -0.00583333,
         0.00333333, -0.01916667],
       [ 0.00333333,  0.01333333,  0.0575    , ...,  0.00583333,
         0.01583333, -0.02916667],
       [ 0.0025    ,  0.01083333,  0.02833333, ...,  0.0025    ,
         0.03166667, -0.0075    ],
       ...,
       [-0.00416667, -0.00666667, -0.01083333, ...,  0.00083333,
         0.00333333, -0.02583333],
       [-0.00083333, -0.00166667, -0.00833333, ...,  0.00083333,
        -0.0275    , -0.0275    ],
       [ 0.01166667,  0.02583333, -0.0075    , ...,  0.        ,
        -0.02666667,  0.4725    ]])

.base_values =
array([0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33])

.data =
array([[ 0.0368995,  1.0551286, -0.5852099, ..., -1.2247449,  0.6264224,
        -0.4383745],
       [-0.3323658, -0.2719725,  3.8874108, ...,  0.8164966,  0.6264224,
        -0.3900995],
       [-0.0973788,  0.3178502,  0.1974987, ...,  0.8164966,  0.6264224,
        -0.4866495],
       ...,
       [-0.6680615,  0.6127615, -0.708207 , ...,  0.8164966,  0.6264224,
        -0.3740079],
       [-0.9198333,  0.3178502, -0.484576 , ...,  0.8164966, -1.5963668,
        -0.2613663],
       [ 0.1376082, -0.3457004, -0.3615789, ...,  0.8164966, -1.5963668,
         1.1546993]])
55
```

	0	1	2	3	4	
0	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
1	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
2	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
3	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
4	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
5	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
6	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
7	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
8	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
9	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
10	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
11	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
12	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
13	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
14	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
15	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
16	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
17	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
18	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
19	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
20	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
21	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
22	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
23	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
24	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
25	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008

	0	1	2	3	4	
26	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
27	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
28	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
29	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
30	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
31	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
32	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
33	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
34	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
35	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
36	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
37	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
38	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
39	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
40	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
41	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
42	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
43	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
44	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
45	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
46	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
47	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008
48	[0.0, -0.030833333333333338, -0.02000000000000...	[0.003333333333333336, 0.01333333333333333, 0....	[0.0025000000000000044, 0.010833333333333342, ...	[-0.001666666666666668, -0.004166666666666669,...	[0.0, 0.001666666666666666, -0.003333333333333...	[0.0041 0.0008

49 rows × 1911 columns

In [ ]:

Next Steps

- Saving shap figures per model in each cv
- Make sure you can loop through each pickled model, load it, create shap values and display plots
- Be able to load one model at a time, create shapley values for each CV train and test set, store shap scores in a dataframe
- Make sure to load original dataset features so that each csv file is the same length as the original dataset
  - This means when a CV dataset is missing a feature, we make sure to assign a shap score of 0
  - each new csv file for loading shap scores of each trained model must include all features

```
LR_shap_all_CVs.csv ==>
                        LR_0 --> CV0
                        LR_1 --> CV1
                        LR_2 --> CV2
```

- Save dataframe for each model in a csv file

More Updates/Fixes

8/02/22

- Currently unsure if creating DataFrame for each model's shap_values should be done in compute_shap_values() or within the nested for-loop in testing cell

8/04/22

- Can create DataFrames for each CV but feature names most likely are not matching actual values (double check it)
- Difficult looping through to merge Dataframes for all CVs features...tried temporary variable
- Must also consider that shap_values array are returned in order of features from test/train set it was passed from...not based on feature order in test/train set **FIXED on 8/05/22**
  - Consider mapping out and ordering the values to avoid shuffling of names and values **FIXED on 8/05/22**

8/05/22

- Saving feature importance scores for each cv
- Created two different runs, one for actual test (default) and another if the user chooses to run it on the training sets for comparison

8/08/22

- Iterating through multiclass shap values for Decision Tree poses issue?...ideally we'd want to get the shap absolute average for both classes 0 and 1...same might be for XGB and any other model that has multiclass output **FIXED on 8/08/22**
  - Figured out that when running the loop in shap_feature_ranking() for Decision Tree, both classes 0 and 1 are accounted for. The shap absolute averages are summed up automatically to get the overall CV feature importances for the model (i double checked this myself through creating a loop that would output two different csv files for each class it iterated through)
- **Current issue:** Figuring out how to save multiple figures for each model when calling shap_summary()...for now, I can only save each figure individually through each CV...if model NB has 2 plot function calls & iterate through 3 CVs --> total 6 shap plots for **ONE** model .....
  - **POSSIBLE FIX** merge all images onto one pdf per model which would entail different shap summaries **OR** create the master list of feature impmortance of all CVs for each model and create shap summaries for those

Run SHAP for Testing Datasets

Loop through each hcc_demo dataset to unpickle and load trained models to create Shapley values and plots Default run

- The default setting runs explainer and shap values for the TESTING datasets for each model and CV
- User has the option below to run the loop for training sets as well

In [15]:
```
# testing all methods
run_force_plots = False # parameter in run_force_plot(); set to True if user wants to display force plots for trained
run_test = True
save_path = experiment_path + '/hcc-data_example/model_evaluation/shap_values/testResults/shapFigures/'


if run_test == True:
    for each in datasets:
        print("----------------------------------------")
        print(each)
        print("----------------------------------------")
        full_path = experiment_path+'/' + each
        filepath = f"/{full_path}/model_evaluation/shap_values/testResults/" #path to save SHAP FI value results

        #Make folder in experiment folder/datafolder to store all shap_values per algorithm/CV combination
        if not os.path.exists(full_path+'/model_evaluation/shap_values/testResults'):
            os.mkdir(full_path+'/model_evaluation/shap_values/testResults')
```

```
    for algorithm in algorithms: #loop through algorithms
        print(abbrev[algorithm])


        for cvCount in range(0,cv_partitions): #loop through cv's
            print(f"{abbrev[algorithm]}{cvCount} In CV{cvCount}...")

            # unpickle and load model
            result_file = f"/{full_path}/models/pickledModels/{abbrev[algorithm]}_{str(cvCount)}.pickle"
            file = open(result_file, 'rb')
            model = pickle.load(file)
            file.close()
            print('\nChecking if correct model is loaded...\n', model)


            # Load CV datasets, paths to datasets updates with each iteration
            train_path = f"/{experiment_path}/{each}/CVDatasets/{each}_CV_{str(cvCount)}_Train.csv"
            test_path =f"/{experiment_path}/{each}/CVDatasets/{each}_CV_{str(cvCount)}_Test.csv"
            trainX, trainY,testX, testY, trainFeat, testFeat = dataPrep(train_path,instance_label,class_label, tes
            print(trainX)

            # shap computation and plots
            # Sanity check: print explainer to check if explainer exists
            explainer = get_explainer(model, abbrev[algorithm], trainX)  #explainer must always use training set
            print(f"\nChecking explainer for {abbrev[algorithm]}{cvCount}...\n{explainer}")

            print(f"\nChecking shap values for {abbrev[algorithm]}{cvCount}...\n")
            shap_values = compute_shapValues(model, abbrev[algorithm], explainer, testX)

            print(f"\nChecking shap plots for {abbrev[algorithm]}{cvCount}...\n")
            shap_summary(abbrev[algorithm], testFeat, shap_values, explainer, testX, cvCount)



            #save SHAP FI results for each model per CV
            print('\nChecking feature importance for {}{}...\n'.format(abbrev[algorithm], cvCount))
            shap_fi_df = shap_feature_ranking(abbrev[algorithm], shap_values, testX, testFeat) # can either choose
            shapFI_path = f"{filepath}{abbrev[algorithm]}_{str(cvCount)}_shapFIValues_Test.csv"
            shap_fi_df.to_csv(shapFI_path, header=True, index=True)

            # create masterList of SHAP Values (not FI) for each model
            save = save_shap(abbrev[algorithm], shap_fi_df, testFeat, original_headers, cvCount)
            display(save)
                 # create new folder to save summary plots for each model per CV
#             if not os.path.exists(experiment_path+'/hcc-data_example/model_evaluation/shap_values/testResults/sh
#                 os.mkdir(full_path+'/model_evaluation/shap_values/testResults/shapFigures')

#                 filepath2 = full_path+"/model_evaluation/shap_values/testResults/shapFigures"+ abbrev[algorithm]
#                 summary.to_pdf(filepath2, header=True, index=True)

            # only runs force plots if run = True
#             if run_force_plots == True:
#                 if abbrev[algorithm] in ['NB']:

#                     print('\nForce Plot for {}{} SHAP Values in Test Set: \n'.format(abbrev[algorithm], cvCount)
#                     shap.force_plot(shap_values, feature_names = testFeat)

#                     print('\nSingle-Prediction Force Plot for {}{} SHAP Values in Test Set: \n'.format(abbrev[al
#                     shap.force_plot(shap_values[42], testX.iloc[42], feature_names=testFeat, show=True)
#                     break

#                 elif abbrev[algorithm] in ['LR', 'XGB', 'LGB', 'CBG']:  #need to test out LGB and CBG for this

#                     print('\nForce Plot for {}{} SHAP Values in Whole Test Set:  \n'.format(abbrev[algorithm], c
#                     shap.force_plot(explainer.expected_value, shap_values, testX, feature_names=testFeat)

#                     print('\nSingle-Prediction Force Plot for {}{} SHAP Values in Test Set: \n'.format(abbrev[al
#                     shap.force_plot(explainer.expected_value, shap_values[42], testX.iloc[42], feature_names=tes
#                     break


#                 else:
#                    # Decision Tree has multiclass output so needed to create two separate function calls
#                    # Decision Tree doesn't work when just using shap_values as a parameter
#                     print('\nForce Plot for {}{} SHAP Values from Class 0 in Test Set: \n'.format(abbrev[algorit
#                     shap.force_plot(explainer.expected_value[0], shap_values[0], feature_names=testFeat)

#                     print('\nForce Plot for {}{} SHAP Values from Class 1 in Test Set: \n'.format(abbrev[algorit
#                     shap.force_plot(explainer.expected_value[1], shap_values[1], feature_names=testFeat)
#                     break
```

```
----------------------------------------
hcc-data_example
----------------------------------------
NB
NB0 In CV0...

Checking if correct model is loaded...
 GaussianNB()
           0         1         2         3         4         5         6   \
0   -0.332366  1.939863  2.243723 -0.138787 -0.785905 -0.641236  0.166013
1   -0.953403  0.170395 -0.579619 -0.143023 -0.785905 -0.641236 -0.916566
2   -0.214872 -0.404683  0.683896  0.096434 -0.785905 -0.641236 -0.048248
3   -0.684846  0.022939 -0.451031 -0.142841  1.272418  2.180204 -0.127186
4   -0.231657  1.703933 -0.641118 -0.142962 -0.785905 -0.641236 -0.521876
..        ...       ...       ...       ...       ...       ...       ...
105  1.195050 -0.714340  2.472945  0.666323 -0.785905  2.180204  1.891372
106  0.104039  1.202584 -0.881521 -0.142794 -0.785905  0.769484 -0.409107
107  0.171178 -1.746529  1.310063  0.113007 -0.785905 -0.641236  0.323889
108  5.911575  1.202584 -0.300080 -0.142852 -0.785905 -0.641236  1.440297
109 -0.516998 -1.893985 -0.657890 -0.143004  1.272418  0.769484 -0.262508

           7         8         9   ...        29        30        31  \
0   -0.397360  0.349927 -0.453155  ... -0.366088  0.073460  0.398006
1    2.516611 -2.857738  0.646551  ... -0.366088 -0.814822 -0.456207
2   -0.397360  0.349927  0.770113  ... -0.366088  0.144312 -0.456207
3   -0.397360  0.349927 -0.218386  ... -0.366088  0.048398 -0.087673
4   -0.397360  0.349927 -0.650855  ... -0.366088  0.016427 -0.071887
..        ...       ...       ...  ...       ...       ...       ...
105 -0.397360  0.349927  4.625261  ... -0.366088 -1.314443  0.557424
106 -0.397360  0.349927 -0.379017  ... -0.366088  0.080369  0.199352
107 -0.397360  0.349927  3.550268  ... -0.366088 -0.113873  0.616728
108 -0.397360  0.349927 -0.218386  ... -0.366088  0.687821 -0.436341
109 -0.397360  0.349927 -0.638498  ... -0.366088  0.751763 -0.020512

           32        33        34        35        36        37        38
0   -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.406191
1    0.015893 -1.671258 -0.542326 -1.349264 -1.224745 -1.596367 -0.438375
2   -0.858225  0.598352 -0.542326 -1.349264  0.816497 -1.596367 -0.164816
3   -0.858225  0.598352  1.843909  0.741145  0.816497  0.626422  0.012192
4   -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.309641
..        ...       ...       ...       ...       ...       ...       ...
105  2.638246  0.598352 -0.542326  0.741145  0.816497  0.626422  5.982196
106 -0.858225  0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.454466
107 -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422  4.051197
108 -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.470558
109  0.890011  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.406191

[110 rows x 39 columns]

Checking explainer for NB0...
shap.explainers.Permutation()

Checking shap values for NB0...

.values =
array([[ 5.00000000e-03, -3.41666667e-02, -1.66666667e-02, ...,
        -2.50000000e-03,  4.16666667e-03, -2.91666667e-02],
       [ 5.00000000e-03,  1.41666667e-02,  4.33333333e-02, ...,
         1.66666667e-03,  1.75000000e-02, -2.41666667e-02],
       [ 5.00000000e-03,  2.25000000e-02,  3.75000000e-02, ...,
         5.00000000e-03,  3.41666667e-02, -4.16666667e-03],
       ...,
       [-1.15648232e-18, -1.33333333e-02, -5.83333333e-03, ...,
         8.33333333e-04,  5.83333333e-03, -2.66666667e-02],
       [-5.00000000e-03, -6.66666667e-03, -4.16666667e-03, ...,
         1.66666667e-03, -3.33333333e-02, -2.41666667e-02],
       [ 6.66666667e-03,  1.66666667e-02, -3.16666667e-02, ...,
         2.50000000e-03, -6.25000000e-02,  4.76666667e-01]])

.base_values =
array([0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33,
       0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33, 0.33])

.data =
array([[ 0.0368995,  1.0551286, -0.5852099, ..., -1.2247449,  0.6264224,
        -0.4383745],
       [-0.3323658, -0.2719725,  3.8874108, ...,  0.8164966,  0.6264224,
        -0.3900995],
       [-0.0973788,  0.3178502,  0.1974987, ...,  0.8164966,  0.6264224,
        -0.4866495],
       ...,
       [-0.6680615,  0.6127615, -0.708207 , ...,  0.8164966,  0.6264224,
        -0.3740079],
       [-0.9198333,  0.3178502, -0.484576 , ...,  0.8164966, -1.5963668,
        -0.2613663],
       [ 0.1376082, -0.3457004, -0.3615789, ...,  0.8164966, -1.5963668,
         1.1546993]])

Checking shap plots for NB0...
```

```
Summary Plot for SHAP Values in Class 0 & 1 in Test Set:

SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set:


Checking feature importance for NB0...

None
```

```
NB1 In CV1...

Checking if correct model is loaded...
 GaussianNB()
           0         1         2         3         4         5         6   \
0    0.445520  0.029220  0.973355 -1.596367 -0.525451 -0.196345 -0.703975
1    0.954496 -0.331786 -0.419335 -1.596367  4.612012 -0.194581 -0.703975
2    0.445520 -0.102055  0.199638 -1.596367  0.373605 -0.155787 -0.703975
3    1.099918 -0.430243  0.632919  0.626422 -0.409858 -0.196293 -0.703975
4   -1.735809 -0.725612 -0.574078  0.626422 -0.185094 -0.196298  0.730048
..        ...       ...       ...       ...       ...       ...       ...
105 -0.136168  1.161468 -0.883565 -1.596367  2.987289  0.596804  2.164072
106 -0.935988  0.094858  1.128098 -1.596367 -0.865808 -0.195411  0.730048
107 -0.935988  0.160495 -1.966768  0.626422  1.651549  0.855003 -0.703975
108 -1.081410  5.772506  1.128098  0.626422 -0.197938 -0.195648 -0.703975
109 -0.063457 -0.512290 -2.121511  0.626422 -0.608935 -0.196271  0.730048

           7         8         9   ...        31        32        33  \
0   -0.495110  0.333333 -0.391324  ... -0.242237 -0.436536 -0.081923
1   -0.845905 -3.000000  0.111232  ... -1.603781 -0.436536 -0.081923
2   -0.420699  0.333333  0.295845  ... -0.402694  0.200345  1.556541
3   -0.505740  0.333333 -0.104149  ...  0.155698  0.118621 -0.901155
4   -0.441960  0.333333 -0.411836  ... -1.104832 -0.436536 -0.081923
..        ...       ...       ...  ...       ...       ...       ...
105  1.779742  0.333333  3.711176  ... -0.326554  0.357715  2.375773
106 -0.388809  0.333333 -0.442605  ... -0.126465  0.216324 -0.901155
107  0.302151  0.333333  2.818882  ...  0.098775  0.370289 -0.901155
108  1.354536  0.333333 -0.309274  ...  0.718488 -0.416753 -0.901155
109 -0.250617  0.333333 -0.657986  ...  0.807431  0.032484  0.737309

           34        35        36        37       38        39        40
0    1.583650 -1.711307 -0.514167 -1.376494  0.68313 -0.450501 -0.213575
1    0.585718 -1.711307 -0.514167 -1.376494  0.68313 -0.402982 -0.317347
2    0.915224  0.584349 -0.514167  0.726483  0.68313 -0.498019 -0.213575
3    0.425672  0.584349 -0.514167  0.726483  0.68313 -0.425157 -0.068302
4   -1.050881  0.584349  1.944893 -1.376494 -1.46385 -0.054513 -0.346996
..        ...       ...       ...       ...      ...       ...       ...
105  2.769871  0.584349 -0.514167  0.726483  0.68313  5.869463 -0.250636
106  0.284455  0.584349 -0.514167  0.726483 -1.46385 -0.466340 -0.161688
107  2.826358  0.584349 -0.514167  0.726483  0.68313  3.968722 -0.228399
108  0.086752  0.584349 -0.514167  0.726483  0.68313 -0.482179 -0.080153
109  0.114995  0.584349 -0.514167  0.726483  0.68313 -0.418821 -0.346996

[110 rows x 41 columns]

Checking explainer for NB1...
shap.explainers.Permutation()

Checking shap values for NB1...

.values =
array([[ 4.62592927e-18,  1.16666667e-02, -4.00000000e-02, ...,
         7.91666667e-02, -2.00000000e-02, -1.00000000e-02],
       [ 2.41666667e-02, -1.16666667e-02, -1.91666667e-02, ...,
        -2.39166667e-01, -2.16666667e-02, -2.00000000e-02],
       [ 1.91666667e-02,  7.50000000e-03,  4.25000000e-02, ...,
        -6.08333333e-02, -1.00000000e-02, -5.00000000e-03],
       ...,
       [ 4.16666667e-03,  9.16666667e-03,  3.33333333e-03, ...,
        -1.87500000e-01, -4.25000000e-02, -1.00000000e-02],
       [-8.33333333e-04, -7.50000000e-03,  1.33333333e-02, ...,
         1.66666667e-02, -1.66666667e-03,  0.00000000e+00],
       [ 4.25000000e-02,  5.00000000e-03, -3.83333333e-02, ...,
         1.03333333e-01, -1.16666667e-02, -1.08333333e-02]])

.base_values =
array([0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49,
       0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49,
       0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49,
       0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49,
       0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49])

.data =
array([[-0.7178553, -0.3317865,  1.9018142, ...,  0.6831301, -0.4188214,
        -0.0949777],
       [ 0.9544964, -0.9389338,  0.0448948, ..., -1.4638501, -0.4505005,
        -0.2135745],
       [ 0.5909417, -0.2169208, -0.558604 , ..., -1.4638501, -0.1812288,
        -0.176513 ],
       ...,
       [-0.4270115,  0.83328  ,  0.1996381, ..., -1.4638501, -0.3713029,
        -0.2358114],
       [-0.4270115,  1.1450584, -0.1098485, ...,  0.6831301, -0.2445869,
        -0.2358114],
       [ 1.0272073, -0.6763836,  1.1280978, ...,  0.6831301, -0.3871424,
        -0.2580483]])

Checking shap plots for NB1...

Summary Plot for SHAP Values in Class 0 & 1 in Test Set:

SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set:
```

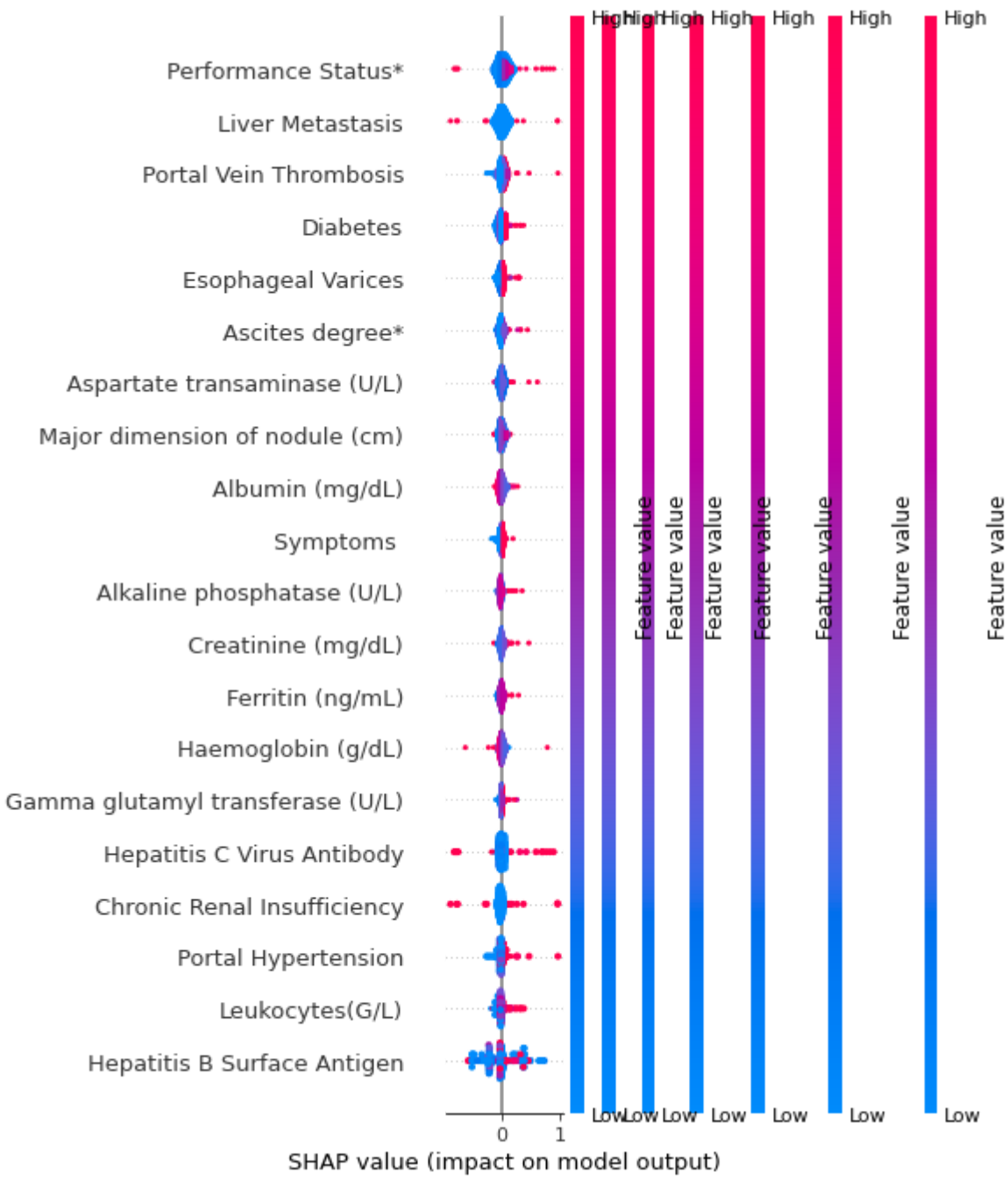
```
Checking feature importance for NB1...

None
```

```
NB2 In CV2...

Checking if correct model is loaded...
 GaussianNB()
           0         1         2         3          4         5         6   \
0    0.473919  0.898611 -1.753304 -0.506142  -0.148119  1.349264 -0.578557
1    1.050407 -0.418470 -1.753304  4.455667  -0.147698  1.349264 -0.578557
2    0.473919  0.166899 -1.753304  0.362175  -0.138449 -0.741145 -0.578557
3    1.215118  0.576658  0.570352 -0.394501  -0.148107 -0.741145 -0.578557
4   -1.996747 -0.564813  0.570352 -0.177422  -0.148108 -0.741145  1.012474
..        ...       ...       ...       ...        ...       ...       ...
105  0.309208  0.752268  0.570352 -0.741827  -0.148077 -0.741145 -0.578557
106 -0.679059 -0.711155  0.570352  0.188512  -0.142429  1.349264  2.603506
107 -0.514348  0.166899  0.570352  1.416559  -0.148116 -0.741145 -0.578557
108 -0.514348 -0.125786  0.570352  1.317323  10.087351 -0.741145  1.012474
109  1.132763  1.044953  0.570352  0.107882  -0.146856  1.349264 -0.578557

           7         8         9   ...        26        27        28  \
0   -0.537954 -0.333333 -0.277483  ...  0.157661  1.434889  2.857738
1   -0.976222 -0.333333  0.213452  ... -0.959673  0.282788 -0.349927
2   -0.444988  3.000000  0.393795  ... -0.134974 -0.869313 -0.349927
3   -0.551235 -0.333333  0.003051  ... -0.161578 -0.869313 -0.349927
4   -0.471550 -0.333333 -0.297521  ... -0.746848 -0.293262 -0.349927
..        ...       ...       ...  ...       ...       ...       ...
105 -0.179371 -0.333333 -0.297521  ...  1.288296 -0.869313 -0.349927
106 -0.498112 -0.333333 -0.307540  ... -0.267990 -0.293262 -0.349927
107  0.431547 -0.333333 -0.227388  ...  1.008963 -0.293262 -0.349927
108 -0.086405 -0.333333 -0.457827  ...  0.915852  1.434889 -0.349927
109 -0.816852  3.000000  2.036924  ...  0.051248 -0.869313 -0.349927

           29        30        31        32        33        34        35
0   -0.005476 -0.864572  0.023440  1.858575 -1.596367 -0.528270  0.755929
1    0.352675 -0.864572  0.023440  0.772520 -1.596367 -0.528270  0.755929
2   -0.205760 -0.197692  1.742347  1.131123  0.626422 -0.528270  0.755929
3   -0.900039 -0.143627 -0.836014  0.598342  0.626422 -0.528270  0.755929
4   -0.931297 -0.864572  0.023440 -1.008599  0.626422  1.892969 -1.322876
..        ...       ...       ...       ...       ...       ...       ...
105  0.044478  0.012506 -0.836014 -1.008261  0.626422 -0.528270 -1.322876
106 -0.153468  2.190446 -0.836014  0.393426  0.626422 -0.528270  0.755929
107  0.037083 -0.864572 -0.836014 -0.016406 -1.596367 -0.528270 -1.322876
108 -4.376995  1.375775  1.742347  3.426182  0.626422  1.892969  0.755929
109 -0.421194 -0.864572  1.742347 -1.009142  0.626422 -0.528270  0.755929

[110 rows x 36 columns]

Checking explainer for NB2...
shap.explainers.Permutation()

Checking shap values for NB2...

.values =
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.005     , -0.00083333,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.00166667,  0.00583333,  0.        , ..., -0.00083333,
        -0.0075    ,  0.00416667],
       ...,
       [-0.005     ,  0.02916667,  0.00083333, ..., -0.00833333,
        -0.00166667,  0.01166667],
       [ 0.0025    , -0.045     ,  0.0075    , ..., -0.00833333,
        -0.045     ,  0.02583333],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ]])

.base_values =
array([0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
       0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
       0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
       0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
       0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05])

.data =
array([[-0.8437697, -0.4184704,  0.5703518, ...,  0.6264224, -0.5282705,
        -1.3228757],
       [-3.7262129, -0.4184704,  0.5703518, ..., -1.5963668, -0.5282705,
         0.7559289],
       [-1.0908363,  0.0205566, -1.7533038, ...,  0.6264224, -0.5282705,
         0.7559289],
       ...,
       [-1.0908363, -1.8818937,  0.5703518, ...,  0.6264224, -0.5282705,
         0.7559289],
       [-1.2555473,  1.0449529,  0.5703518, ...,  0.6264224, -0.5282705,
         0.7559289],
       [-0.1025701, -2.028236 ,  0.5703518, ...,  0.6264224, -0.5282705,
         0.7559289]])

Checking shap plots for NB2...

Summary Plot for SHAP Values in Class 0 & 1 in Test Set:

SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set:
```

```
Checking feature importance for NB2...

None
LR
LR0 In CV0...

Checking if correct model is loaded...
 LogisticRegression(C=0.006606805070193189, dual=True,
                    max_iter=193.8544995971634, random_state=42,
                    solver='liblinear')
           0         1         2         3         4         5         6   \
0   -0.332366  1.939863  2.243723 -0.138787 -0.785905 -0.641236  0.166013
1   -0.953403  0.170395 -0.579619 -0.143023 -0.785905 -0.641236 -0.916566
2   -0.214872 -0.404683  0.683896  0.096434 -0.785905 -0.641236 -0.048248
3   -0.684846  0.022939 -0.451031 -0.142841  1.272418  2.180204 -0.127186
4   -0.231657  1.703933 -0.641118 -0.142962 -0.785905 -0.641236 -0.521876
..        ...       ...       ...       ...       ...       ...       ...
105  1.195050 -0.714340  2.472945  0.666323 -0.785905  2.180204  1.891372
106  0.104039  1.202584 -0.881521 -0.142794 -0.785905  0.769484 -0.409107
107  0.171178 -1.746529  1.310063  0.113007 -0.785905 -0.641236  0.323889
108  5.911575  1.202584 -0.300080 -0.142852 -0.785905 -0.641236  1.440297
109 -0.516998 -1.893985 -0.657890 -0.143004  1.272418  0.769484 -0.262508

           7         8         9   ...        29        30        31  \
0   -0.397360  0.349927 -0.453155  ... -0.366088  0.073460  0.398006
1    2.516611 -2.857738  0.646551  ... -0.366088 -0.814822 -0.456207
2   -0.397360  0.349927  0.770113  ... -0.366088  0.144312 -0.456207
3   -0.397360  0.349927 -0.218386  ... -0.366088  0.048398 -0.087673
4   -0.397360  0.349927 -0.650855  ... -0.366088  0.016427 -0.071887
..        ...       ...       ...  ...       ...       ...       ...
105 -0.397360  0.349927  4.625261  ... -0.366088 -1.314443  0.557424
106 -0.397360  0.349927 -0.379017  ... -0.366088  0.080369  0.199352
107 -0.397360  0.349927  3.550268  ... -0.366088 -0.113873  0.616728
108 -0.397360  0.349927 -0.218386  ... -0.366088  0.687821 -0.436341
109 -0.397360  0.349927 -0.638498  ... -0.366088  0.751763 -0.020512

           32        33        34        35        36        37        38
0   -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.406191
1    0.015893 -1.671258 -0.542326 -1.349264 -1.224745 -1.596367 -0.438375
2   -0.858225  0.598352 -0.542326 -1.349264  0.816497 -1.596367 -0.164816
3   -0.858225  0.598352  1.843909  0.741145  0.816497  0.626422  0.012192
4   -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.309641
..        ...       ...       ...       ...       ...       ...       ...
105  2.638246  0.598352 -0.542326  0.741145  0.816497  0.626422  5.982196
106 -0.858225  0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.454466
107 -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422  4.051197
108 -0.858225  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.470558
109  0.890011  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.406191

[110 rows x 39 columns]

Checking explainer for LR0...
<shap.explainers._linear.Linear object at 0x7f81bcb37250>

Checking shap values for LR0...

[[-1.54247265e-04 -5.20008470e-02 -4.40485958e-02 ... -1.47969848e-03
   2.31157863e-02 -2.02239904e-02]
 [ 6.85778818e-04  1.94894821e-02  2.13041189e-01 ...  1.21066239e-03
   2.31157863e-02 -1.80842415e-02]
 [ 1.51216786e-04 -1.22839969e-02  9.42115197e-04 ...  1.21066239e-03
   2.31157863e-02 -2.23637392e-02]
 ...
 [ 1.44943883e-03 -2.81707337e-02 -5.11185666e-02 ...  1.21066239e-03
   2.31157863e-02 -1.73709948e-02]
 [ 2.02218390e-03 -1.22839969e-02 -3.82640794e-02 ...  1.21066239e-03
  -6.24982370e-02 -1.23782505e-02]
 [-3.83345247e-04  2.34611703e-02 -3.11941086e-02 ...  1.21066239e-03
  -6.24982370e-02  5.03876684e-02]]

Checking shap plots for LR0...

Expected value for LR: -0.023696555525940875
Summary Plot for SHAP Values in Test Set:
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [15], in <cell line: 7>()
     48 shap_values = compute_shapValues(model, abbrev[algorithm], explainer, testX)
     50 print(f"\nChecking shap plots for {abbrev[algorithm]}{cvCount}...\n")
---> 51 shap_summary(abbrev[algorithm], testFeat, shap_values, explainer, testX, cvCount)
     55 #save SHAP FI results for each model per CV
     56 print('\nChecking feature importance for {}{}...\n'.format(abbrev[algorithm], cvCount))

Input In [10], in shap_summary(abbrev, feature_names, shap_values, explainer, X, cvCount)
     33 print('Expected value for {}: {}'.format(abbrev, expected_value))
     35 print('Summary Plot for SHAP Values in Test Set: \n')
---> 36 shap.summary_plot(shap_values, X, feature_names, plot_type='violin', show=False)
     37 plt.savefig(save_path+abbrev+'_'+str(cvCount)+'shapSummaryPlot.png', bbox_inches='tight')
     38 plt.close()

File ~/opt/anaconda3/lib/python3.9/site-packages/shap/plots/_beeswarm.py:890, in summary_legacy(shap_values, features,
feature_names, max_display, plot_type, color, axis_color, title, alpha, show, sort, color_bar, plot_size, layered_viol
in_max_num_bins, class_names, class_inds, color_bar_label, cmap, auto_size_plot, use_log_scale)
    888 else:
    889     pl.xlabel(labels['VALUE'], fontsize=13)
--> 890 pl.tight_layout()
    891 if show:
    892     pl.show()

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/pyplot.py:2302, in tight_layout(pad, h_pad, w_pad, rect)
   2300 @_copy_docstring_and_deprecators(Figure.tight_layout)
   2301 def tight_layout(*, pad=1.08, h_pad=None, w_pad=None, rect=None):
-> 2302     return gcf().tight_layout(pad=pad, h_pad=h_pad, w_pad=w_pad, rect=rect)

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/figure.py:3197, in Figure.tight_layout(self, pad, h_pad, w
_pad, rect)
   3195 renderer = _get_renderer(self)
   3196 with getattr(renderer, "_draw_disabled", nullcontext)():
-> 3197     kwargs = get tight layout figure(
   3198         self, self.axes, subplotspec list, renderer,
   3199         pad=pad, h_pad=h_pad, w_pad=w_pad, rect=rect)
   3200 if kwargs:
   3201     self.subplots_adjust(**kwargs)

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/tight_layout.py:320, in get_tight_layout_figure(fig, axes_
list, subplotspec_list, renderer, pad, h_pad, w_pad, rect)
    315         return {}
    316     span_pairs.append((
    317         slice(ss.rowspan.start * div_row, ss.rowspan.stop * div_row),
    318         slice(ss.colspan.start * div_col, ss.colspan.stop * div_col)))
--> 320 kwargs =  auto adjust subplotpars(fig, renderer,
    321                                     shape=(max nrows, max ncols),
    322                                     span pairs=span pairs,
    323                                     subplot_list=subplot_list,
    324                                     ax bbox list=ax bbox list,
    325                                     pad=pad, h_pad=h_pad, w_pad=w_pad)
    327 # kwargs can be none if tight_layout fails...
    328 if rect is not None and kwargs is not None:
    329     # if rect is given, the whole subplots area (including
    330     # labels) will fit into the rect instead of the
   (...)
    334     # auto_adjust_subplotpars twice, where the second run
    335     # with adjusted rect parameters.

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/tight_layout.py:82, in _auto_adjust_subplotpars(fig, rende
rer, shape, span_pairs, subplot_list, ax_bbox_list, pad, h_pad, w_pad, rect)
     80 if ax.get_visible():
     81     try:
---> 82         bb += [ax.get_tightbbox(renderer, for_layout_only=True)]
     83     except TypeError:
     84         bb += [ax.get_tightbbox(renderer)]

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axes/_base.py:4666, in _AxesBase.get_tightbbox(self, rende
rer, call_axes_locator, bbox_extra_artists, for_layout_only)
   4662     if np.all(clip_extent.extents == axbbox.extents):
   4663         # clip extent is inside the Axes bbox so don't check
   4664         # this artist
   4665         continue
-> 4666 bbox = a.get_tightbbox(renderer)
   4667 if (bbox is not None
   4668         and 0 < bbox.width < np.inf
   4669         and 0 < bbox.height < np.inf):
   4670     bb.append(bbox)

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/artist.py:355, in Artist.get_tightbbox(self, renderer)
    340 def get_tightbbox(self, renderer):
    341     """
    342     Like `.Artist.get_window_extent`, but includes any clipping.
    343
   (...)
    353         The enclosing bounding box (in figure pixel coordinates).
    354     """
--> 355     bbox = self.get_window_extent(renderer)
    356     if self.get_clip_on():
    357         clip_box = self.get_clip_box()
```

```
File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py:321, in Collection.get_window_extent(self,
 renderer)
    318 def get_window_extent(self, renderer):
    319     # TODO: check to ensure that this does not fail for
    320     # cases other than scatter plot legend
--> 321     return self.get_datalim(transforms.IdentityTransform())

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/collections.py:292, in Collection.get_datalim(self, transD
ata)
    289     # get_path_collection_extents handles nan but not masked arrays
    291 if len(paths) and len(offsets):
--> 292     if any(transform.contains_branch_seperately(transData)):
    293         # collections that are just in data units (like quiver)
    294         # can properly have the axes limits set by their shape +
    295         # offset.  LineCollections that have no offsets can
    296         # also use this algorithm (like streamplot).
    297         return mpath.get_path_collection_extents(
    298             transform.get_affine() - transData, paths,
    299             self.get_transforms(),
    300             transOffset.transform_non_affine(offsets),
    301             transOffset.get_affine().frozen())
    302     if hasOffsets:
    303         # this is for collections that have their paths (shapes)
    304         # in physical, axes-relative, or figure-relative units
    305         # (i.e. like scatter). We can't uniquely set limits based on
    306         # those shapes, so we just set the limits based on their
    307         # location.

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:1424, in Transform.contains_branch_seperatel
y(self, other_transform)
   1420     raise ValueError('contains_branch_seperately only supports '
   1421                      'transforms with 2 output dimensions')
   1422 # for a non-blended transform each separate dimension is the same, so
   1423 # just return the appropriate shape.
-> 1424 return [self.contains_branch(other_transform)] * 2

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:1403, in Transform.contains_branch(self, oth
er)
   1400     return False
   1402 # check that a subtree is equal to other (starting from self)
-> 1403 for _, sub_tree in self._iter_break_from_left_to_right():
   1404     if sub_tree == other:
   1405         return True

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:2404, in CompositeGenericTransform._iter_bre
ak_from_left_to_right(self)
   2402 def _iter_break_from_left_to_right(self):
   2403     for left, right in self._a._iter_break_from_left_to_right():
-> 2404         yield left, right + self._b
   2405     for left, right in self._b._iter_break_from_left_to_right():
   2406         yield self._a + left, right

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:1355, in Transform.__add__(self, other)
   1348 def __add__(self, other):
   1349     """
   1350     Compose two transforms together so that *self* is followed by *other*.
   1351
   1352     ``A + B`` returns a transform ``C`` so that
   1353     ``C.transform(x) == B.transform(A.transform(x))``.
   1354     """
-> 1355     return (composite_transform_factory(self, other)
   1356             if isinstance(other, Transform) else
   1357             NotImplemented)

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:2532, in composite_transform_factory(a, b)
   2530 elif isinstance(a, Affine2D) and isinstance(b, Affine2D):
   2531     return CompositeAffine2D(a, b)
-> 2532 return CompositeGenericTransform(a, b)

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:2366, in CompositeGenericTransform.__init__
(self, a, b, **kwargs)
   2363 self.input_dims = a.input_dims
   2364 self.output_dims = b.output_dims
-> 2366 super().__init__(**kwargs)
   2367 self._a = a
   2368 self._b = b

File ~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/transforms.py:120, in TransformNode.__init__(self, shortha
nd_name)
    116 self._parents = {}
    118 # TransformNodes start out as invalid until their values are
    119 # computed for the first time.
--> 120 self._invalid = 1
    121 self._shorthand_name = shorthand_name or ''

KeyboardInterrupt:
```



Run SHAP for Training Sets

Optional

- This runs on training CV Datasets that were partiioned during STREAMLINE
- User can set run_train to 'True' for comparison between training and testing sets

In [ ]:
```
run_force_plots = True # parameter in run_force_plot(); set to True if user wants to display force plots for trained m
run_train = False # user can change to True to run shap values for training sets

if run_train == True:
    for each in datasets:
        print("----------------------------------------")
        print(each)
        print("----------------------------------------")
        full_path = experiment_path+'/' + each

        #Make folder in experiment folder/datafolder to store all shap_values per algorithm/CV combination
        if not os.path.exists(full_path+'/model_evaluation/shap_values/trainResults'):
            os.mkdir(full_path+'/model_evaluation/shap_values/trainResults')

        original_headers = pd.read_csv(full_path+"/exploratory/OriginalFeatureNames.csv",sep=',').columns.values.tolis
        feat_order_map = {feat:i for i, feat in enumerate(original_headers)}
        print(feat_order_map)

        for algorithm in algorithms: #loop through algorithms
            print(abbrev[algorithm])


            for cvCount in range(0,cv_partitions): #loop through cv's
                print('{}{} In CV{}...'.format(abbrev[algorithm], cvCount, cvCount))

                # unpickle and load model
                result_file = full_path+ '/models/pickledModels/' + abbrev[algorithm]+ "_" + str(cvCount)+".pickle"
                file = open(result_file, 'rb')
                model = pickle.load(file)
                file.close()
                print('\nChecking if correct model is loaded...\n', model)


                # Load CV datasets, paths to datasets updates with each iteration
                train_path = f"{experiment_path}/{each}/CVDatasets/{each}_CV_{str(cvCount)}_Train.csv"
                test_path =f"{experiment_path}/{each}/CVDatasets/{each}_CV_{str(cvCount)}_Test.csv"
                trainX, trainY,testX, testY, train_feat, test_feat = dataPrep(train_path,instance_label,class_label, t
```

```
                # shap computation and plots
                explainer = get_explainer(model, abbrev[algorithm], trainX)
                print('\nChecking explainer for {}{}...\n{}'.format(abbrev[algorithm], cvCount, explainer))  # print e

                print('\nChecking shap values for {}{}...\n'.format(abbrev[algorithm], cvCount))
                shap_values = compute_shapValues(model, abbrev[algorithm], explainer, trainX)

                print('\nChecking shap plots for {}{}...\n'.format(abbrev[algorithm], cvCount))
                shap_summary(abbrev[algorithm], train_feat, shap_values, explainer, trainX)


                #save SHAP FI results
                print('\nChecking feature importance for {}{}...\n'.format(abbrev[algorithm], cvCount))
                shap_fi_df = shap_feature_ranking(abbrev[algorithm], shap_values, trainX, train_feat) # can either cho

                filepath = full_path+"/model_evaluation/shap_values/trainResults/"+ abbrev[algorithm] + '_' + str(cvCo
                shap_fi_df.to_csv(filepath, header=True, index=True)


                # only runs force plots if run = True
                if run_force_plots == True:
                    if abbrev[algorithm] in ['NB']:

                        print('\nForce Plot for {}{} SHAP Values in Train Set: \n'.format(abbrev[algorithm], cvCount))
                        shap.force_plot(shap_values, trainX, feature_names=train_feat)

                        print('\nSingle-Prediction Force Plot for {}{} SHAP Values in Train Set: \n'.format(abbrev[alg
                        shap.force_plot(shap_values[42], trainX.iloc[42], feature_names=train_feat, show=False)
# #                         plt.savefig(full_path+'/model_evaluation/'+abbrev[algorithm]+"_shapFP.png",dpi=300)  FIXME
                        break


                    elif abbrev[algorithm] in ['LR', 'XGB', 'LGB', 'CBG']:  #need to test out LGB and CBG for this

                        print('\nForce Plot for {}{} SHAP Values in Whole Train Set:  \n'.format(abbrev[algorithm], cv
                        shap.force_plot(explainer.expected_value, shap_values, trainX, feature_names=train_feat)

                        print('\nSingle-Prediction Force Plot for {}{} SHAP Values in Train Set: \n'.format(abbrev[alg
                        shap.force_plot(explainer.expected_value, shap_values[42], trainX.iloc[42], feature_names=trai
                        break


                    else:
#
                        print('\nForce Plot for {}{} SHAP Values from Class 0 in Train Set: \n'.format(abbrev[algorith
                        shap.force_plot(explainer.expected_value[0], shap_values[0], feature_names=train_feat)

                        print('\nForce Plot for {}{} SHAP Values from Class 1 in Train Set: \n'.format(abbrev[algorith
                        shap.force_plot(explainer.expected_value[1], shap_values[1], feature_names=train_feat)
                        break
```

In [ ]:

In [ ]: