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 [13]:
```
def save_shap(abbrev, shap_values, original_headers, cvCount, dataset): # 'df' parameter is the dataframe returned fro
    '''Create a new dataframe that stores the model's absolute mean SHAP feature importance values over each CV
           and combines with features from original dataset'''

    FI_all = []
    temp_list = []
    shap_vals = []



    headers = pd.read_csv(f'{experiment_path}/hcc-data_example/CVDatasets/hcc-data_example_CV_{cvCount}_{dataset}.csv'
    if instance_label != 'None':
        headers.remove(instance_label)
    headers.remove(class_label)

    if abbrev in ['NB']:
        vals = (np.mean(np.abs(shap_values.values), axis=0))
        shap_vals = np.array(vals)
        for name in original_headers:
            if name in headers:
                index = headers.index(name)
                print(f'{name} is {name}\n')
#                 print(f'Index is {index}')
#                 print(f'Shap value is {shap_vals[index]}')
#                 print(f'{name} value is {shap_vals[index]}')
                temp_list.append(shap_vals[index])
            else:
                temp_list.append(0.0)
        FI_all.append(temp_list)
#         temp_dr = pd.DataFrame(data=FI_all, columns=original_headers)
        dr = pd.DataFrame(data=FI_all, columns=original_headers)
#         dr.append(temp_dr, ignore_index = True)
#         del temp_dr
```

```
        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, columns=original_headers, index=False)

#     elif abbrev in ['LR','LGB', 'XGB', 'CGB']:
#         vals = (np.mean(np.abs(shap_values), axis=0))
#         shap_vals = np.array(vals)
#         for name in original_headers:
#             if name in headers:
#                 index = headers.index(name)

# #                     print(f'{each} is {each}\n')
# #                     print(f'Index is {index}')
# #                     print(f'Shap value is {shap_vals[index]}')
# #                 print(f'{name} value is {shap_vals[index]}')
#                 temp_list.append(shap_vals[index])
#             else:
#                 temp_list.append(0.0)
#         FI_all.append(temp_list)
#     print(FI_all)
```

In [14]:

```
# # # ^^^^^

# 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=',').col
# 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, feat_order_map, 0)
```

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_values, original_headers, cvCount, 'Test')
                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([[ 0.005     , -0.0225    , -0.00916667, ..., -0.00333333,
         0.00583333, -0.02666667],
       [ 0.0075    ,  0.02833333,  0.07666667, ...,  0.0025    ,
         0.00916667, -0.01833333],
       [ 0.01      ,  0.02666667,  0.01916667, ...,  0.00333333,
         0.0475    ,  0.01      ],
       ...,
       [-0.00083333, -0.0175    , -0.015     , ...,  0.        ,
         0.005     , -0.02583333],
       [-0.00166667, -0.00583333, -0.01333333, ...,  0.00166667,
        -0.02333333, -0.0325    ],
       [ 0.00416667,  0.01916667, -0.015     , ...,  0.00333333,
        -0.04666667,  0.45      ]])

.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...

Symptoms  is Symptoms

Hepatitis B Surface Antigen is Hepatitis B Surface Antigen

Hepatitis B e Antigen is Hepatitis B e Antigen

Hepatitis C Virus Antibody is Hepatitis C Virus Antibody

Cirrhosis is Cirrhosis

Endemic Countries is Endemic Countries

Smoking is Smoking

Diabetes is Diabetes

Obesity is Obesity

Hemochromatosis is Hemochromatosis

Arterial Hypertension is Arterial Hypertension

Chronic Renal Insufficiency is Chronic Renal Insufficiency

Esophageal Varices is Esophageal Varices

Splenomegaly is Splenomegaly

Portal Hypertension is Portal Hypertension

Portal Vein Thrombosis is Portal Vein Thrombosis

Liver Metastasis is Liver Metastasis

Packs of cigarets per year is Packs of cigarets per year

Performance Status* is Performance Status*

Encephalopathy degree* is Encephalopathy degree*

Ascites degree* is Ascites degree*

International Normalised Ratio* is International Normalised Ratio*

Alpha-Fetoprotein (ng/mL) is Alpha-Fetoprotein (ng/mL)

Haemoglobin (g/dL) is Haemoglobin (g/dL)

Mean Corpuscular Volume is Mean Corpuscular Volume

Leukocytes(G/L) is Leukocytes(G/L)

Albumin (mg/dL) is Albumin (mg/dL)

Total Bilirubin(mg/dL) is Total Bilirubin(mg/dL)

Alanine transaminase (U/L) is Alanine transaminase (U/L)

Aspartate transaminase (U/L) is Aspartate transaminase (U/L)

Gamma glutamyl transferase (U/L) is Gamma glutamyl transferase (U/L)

Alkaline phosphatase (U/L) is Alkaline phosphatase (U/L)

Creatinine (mg/dL) is Creatinine (mg/dL)

Number of Nodules is Number of Nodules

Major dimension of nodule (cm) is Major dimension of nodule (cm)

Direct Bilirubin (mg/dL) is Direct Bilirubin (mg/dL)

Iron is Iron

Oxygen Saturation (%) is Oxygen Saturation (%)

Ferritin (ng/mL) is Ferritin (ng/mL)
```

	Gender	Symptoms	Alcohol	Hepatitis B Surface Antigen	Hepatitis B e Antigen	Hepatitis B Core Antibody	Hepatitis C Virus Antibody	Cirrhosis	Endemic Countries	Smoking	...	Gamma glutamyl transferase (U/L)	Alkaline phosphatase (U/L)	To Prote (g/
0	0.0	0.023	0.0	0.013924	0.0	0.0	0.012212	0.008561	0.017515	0.001864	...	0.012455	0.015712	

1 rows × 49 columns

```
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([[ 0.0075    ,  0.0225    , -0.1225    , ...,  0.10416667,
        -0.00833333, -0.00833333],
       [ 0.035     , -0.00083333, -0.01583333, ..., -0.23      ,
        -0.03833333, -0.01666667],
       [ 0.0175    ,  0.005     ,  0.0275    , ..., -0.055     ,
        -0.00916667, -0.00916667],
       ...,
       [-0.00083333, -0.005     ,  0.01916667, ..., -0.1475    ,
        -0.03416667, -0.01166667],
       [ 0.00416667, -0.00083333,  0.005     , ...,  0.01416667,
        -0.00083333, -0.00333333],
       [ 0.04      ,  0.015     , -0.055     , ...,  0.1125    ,
        -0.01583333, -0.00583333]])

.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...

Symptoms  is Symptoms

Alcohol is Alcohol

Hepatitis B Surface Antigen is Hepatitis B Surface Antigen

Hepatitis B e Antigen is Hepatitis B e Antigen

Hepatitis B Core Antibody is Hepatitis B Core Antibody

Hepatitis C Virus Antibody is Hepatitis C Virus Antibody

Cirrhosis is Cirrhosis

Endemic Countries is Endemic Countries

Smoking is Smoking

Diabetes is Diabetes

Obesity is Obesity

Nonalcoholic Steatohepatitis is Nonalcoholic Steatohepatitis

Esophageal Varices is Esophageal Varices

Portal Hypertension is Portal Hypertension

Portal Vein Thrombosis is Portal Vein Thrombosis

Liver Metastasis is Liver Metastasis

Age at diagnosis is Age at diagnosis

Grams of Alcohol per day is Grams of Alcohol per day

Packs of cigarets per year is Packs of cigarets per year

Performance Status* is Performance Status*

Ascites degree* is Ascites degree*

International Normalised Ratio* is International Normalised Ratio*

Alpha-Fetoprotein (ng/mL) is Alpha-Fetoprotein (ng/mL)

Haemoglobin (g/dL) is Haemoglobin (g/dL)

Mean Corpuscular Volume is Mean Corpuscular Volume

Leukocytes(G/L) is Leukocytes(G/L)

Platelets is Platelets

Albumin (mg/dL) is Albumin (mg/dL)

Total Bilirubin(mg/dL) is Total Bilirubin(mg/dL)

Alanine transaminase (U/L) is Alanine transaminase (U/L)

Aspartate transaminase (U/L) is Aspartate transaminase (U/L)

Gamma glutamyl transferase (U/L) is Gamma glutamyl transferase (U/L)

Alkaline phosphatase (U/L) is Alkaline phosphatase (U/L)

Total Proteins (g/dL) is Total Proteins (g/dL)

Creatinine (mg/dL) is Creatinine (mg/dL)

Number of Nodules is Number of Nodules

Major dimension of nodule (cm) is Major dimension of nodule (cm)

Direct Bilirubin (mg/dL) is Direct Bilirubin (mg/dL)

Iron is Iron

Oxygen Saturation (%) is Oxygen Saturation (%)

Ferritin (ng/mL) is Ferritin (ng/mL)
```

	Gender	Symptoms	Alcohol	Hepatitis B Surface Antigen	Hepatitis B e Antigen	Hepatitis B Core Antibody	Hepatitis C Virus Antibody	Cirrhosis	Endemic Countries	Smoking	...	Gamma glutamyl transferase (U/L)	Alkaline phosphatase (U/L)	Pro (g
0	0.0	0.065591	0.006788	0.014561	0.002545	0.005561	0.008667	0.0	0.10053	0.005212	...	0.005576	0.049682	0.01

1 rows × 49 columns

```
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.        , -0.0025    ],
       [ 0.00666667,  0.        ,  0.        , ...,  0.00583333,
         0.        ,  0.        ],
       [ 0.        ,  0.00416667, -0.00916667, ..., -0.01083333,
         0.        ,  0.00333333],
       ...,
       [-0.00083333,  0.00333333,  0.        , ..., -0.0025    ,
        -0.0025    ,  0.0125    ],
       [-0.00333333, -0.05666667,  0.0075    , ..., -0.00833333,
        -0.0525    ,  0.01666667],
       [ 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...

Symptoms  is Symptoms

Alcohol is Alcohol

Hepatitis B e Antigen is Hepatitis B e Antigen

Hepatitis B Core Antibody is Hepatitis B Core Antibody

Hepatitis C Virus Antibody is Hepatitis C Virus Antibody

Endemic Countries is Endemic Countries

Diabetes is Diabetes

Obesity is Obesity

Arterial Hypertension is Arterial Hypertension

Chronic Renal Insufficiency is Chronic Renal Insufficiency

Human Immunodeficiency Virus is Human Immunodeficiency Virus

Portal Hypertension is Portal Hypertension

Portal Vein Thrombosis is Portal Vein Thrombosis

Liver Metastasis is Liver Metastasis

Age at diagnosis is Age at diagnosis

Packs of cigarets per year is Packs of cigarets per year

Performance Status* is Performance Status*

Encephalopathy degree* is Encephalopathy degree*

Ascites degree* is Ascites degree*

International Normalised Ratio* is International Normalised Ratio*

Alpha-Fetoprotein (ng/mL) is Alpha-Fetoprotein (ng/mL)

Haemoglobin (g/dL) is Haemoglobin (g/dL)

Mean Corpuscular Volume is Mean Corpuscular Volume

Leukocytes(G/L) is Leukocytes(G/L)

Platelets is Platelets

Albumin (mg/dL) is Albumin (mg/dL)

Aspartate transaminase (U/L) is Aspartate transaminase (U/L)

Gamma glutamyl transferase (U/L) is Gamma glutamyl transferase (U/L)

Alkaline phosphatase (U/L) is Alkaline phosphatase (U/L)

Creatinine (mg/dL) is Creatinine (mg/dL)

Number of Nodules is Number of Nodules

Major dimension of nodule (cm) is Major dimension of nodule (cm)

Direct Bilirubin (mg/dL) is Direct Bilirubin (mg/dL)

Iron is Iron

Oxygen Saturation (%) is Oxygen Saturation (%)

Ferritin (ng/mL) is Ferritin (ng/mL)
```

	Gender	Symptoms	Alcohol	Hepatitis B Surface Antigen	Hepatitis B e Antigen	Hepatitis B Core Antibody	Hepatitis C Virus Antibody	Cirrhosis	Endemic Countries	Smoking	...	Gamma glutamyl transferase (U/L)	Alkaline phosphatase (U/L)	To Prote (g/
0	0.0	0.001136	0.00053	0.0	0.009182	0.003758	0.000924	0.0	0.000197	0.0	...	0.000955	0.003273	

1 rows × 49 columns

```
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 0x7fb9b33366a0>

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:

SHAP Bar Plot for SHAP Values Test Set:

SHAP Decision Plot for SHAP Values in Test Set:

SHAP Decision Plot for Single-Prediction in Test Set:


Checking feature importance for LR0...

None
```

```
LR1 In CV1...

Checking if correct model is loaded...
 LogisticRegression(C=0.06359900885943309, max_iter=48.076782938152924,
                   random_state=42, solver='sag')
           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 LR1...
<shap.explainers._linear.Linear object at 0x7fb9b3f32640>

Checking shap values for LR1...

[[-0.19905492  0.03850256 -0.30345684 ...  0.14567922 -0.09989041
  -0.01025168]
 [ 0.18762272  0.10977384 -0.00291614 ... -0.39387344 -0.10644632
  -0.02397595]
 [ 0.10356236  0.0250188   0.09475959 ... -0.39387344 -0.05072119
  -0.01968712]
 ...
 [-0.13180663 -0.09826126 -0.0279612  ... -0.39387344 -0.09005657
  -0.02654925]
 [-0.13180663 -0.13486004  0.02212892 ...  0.14567922 -0.063833
  -0.02654925]
 [ 0.20443478  0.07895383 -0.17823155 ...  0.14567922 -0.09333451
  -0.02912255]]

Checking shap plots for LR1...

Expected value for LR: -0.6091565598361125
Summary Plot for SHAP Values in Test Set:

SHAP Bar Plot for SHAP Values Test Set:

SHAP Decision Plot for SHAP Values in Test Set:

SHAP Decision Plot for Single-Prediction in Test Set:


Checking feature importance for LR1...

None
```

```
LR2 In CV2...

Checking if correct model is loaded...
 LogisticRegression(C=0.0006580360277501316, class_weight='balanced', dual=True,
                   max_iter=112.07606211860569, random_state=42,
                   solver='liblinear')
           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 LR2...
<shap.explainers._linear.Linear object at 0x7fb9a0550b80>

Checking shap values for LR2...

[[-0.00209083  0.00438031  0.00118267 ... -0.00203892 -0.00439287
  -0.00869176]
 [-0.00982644  0.00438031  0.00118267 ...  0.00524293 -0.00439287
   0.00355015]
 [-0.00275388  0.00044561 -0.00473069 ... -0.00203892 -0.00439287
   0.00355015]
 ...
 [-0.00275388  0.01749597  0.00118267 ... -0.00203892 -0.00439287
   0.00355015]
 [-0.00319591 -0.00873536  0.00118267 ... -0.00203892 -0.00439287
   0.00355015]
 [-0.00010167  0.01880754  0.00118267 ... -0.00203892 -0.00439287
   0.00355015]]

Checking shap plots for LR2...

Expected value for LR: -0.006133751932115765
Summary Plot for SHAP Values in Test Set:

SHAP Bar Plot for SHAP Values Test Set:

SHAP Decision Plot for SHAP Values in Test Set:

SHAP Decision Plot for Single-Prediction in Test Set:


Checking feature importance for LR2...

None
```

```
DT
DT0 In CV0...

Checking if correct model is loaded...
 DecisionTreeClassifier(max_depth=17, min_samples_leaf=35, min_samples_split=45,
                       random_state=42)
           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 DT0...
<shap.explainers._tree.Tree object at 0x7fb9c631a100>

Checking shap values for DT0...

[array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]]), array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])]

Checking shap plots for DT0...

Expected value for DT: [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:


Checking feature importance for DT0...

None
```

```
DT1 In CV1...

Checking if correct model is loaded...
 DecisionTreeClassifier(criterion='entropy', max_depth=21, min_samples_leaf=3,
                       min_samples_split=23, random_state=42,
                       splitter='random')
           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 DT1...
<shap.explainers._tree.Tree object at 0x7fb9b6b94a60>

Checking shap values for DT1...

[array([[ 0.        ,  0.        ,  0.        , ..., -0.02795699,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.07004662,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.07004662,
          0.        ,  0.        ],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  0.08666667,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ..., -0.11345397,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ..., -0.02795699,
          0.        ,  0.        ]]), array([[ 0.        ,  0.        ,  0.        , ...,  0.02795699,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ..., -0.07004662,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ..., -0.07004662,
          0.        ,  0.        ],
       ...,
       [ 0.        ,  0.        ,  0.        , ..., -0.08666667,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.11345397,
          0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.02795699,
          0.        ,  0.        ]])]

Checking shap plots for DT1...

Expected value for DT: [0.63636364 0.36363636]
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:


Checking feature importance for DT1...

None
```

```
DT2 In CV2...

Checking if correct model is loaded...
 DecisionTreeClassifier(class_weight='balanced', max_depth=29,
                       min_samples_leaf=30, min_samples_split=45,
                       random_state=42)
           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 DT2...
<shap.explainers._tree.Tree object at 0x7fb9b6b56d30>

Checking shap values for DT2...

[array([[ 0.        , -0.14385676,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.0462963 ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.0462963 ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        , -0.0462963 ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.05769231,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.14385676,  0.        , ...,  0.        ,
         0.        ,  0.        ]]), array([[ 0.        ,  0.14385676,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.0462963 ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.0462963 ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        ,  0.0462963 ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.05769231,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.14385676,  0.        , ...,  0.        ,
         0.        ,  0.        ]])]

Checking shap plots for DT2...

Expected value for DT: [0.5 0.5]
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:


Checking feature importance for DT2...

None
```

```
RF
RF0 In CV0...

Checking if correct model is loaded...
 RandomForestClassifier(criterion='entropy', max_depth=1, max_features=None,
                       min_samples_leaf=17, min_samples_split=41,
                       n_estimators=960, random_state=42)
           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 RF0...
<shap.explainers._tree.Tree object at 0x7fb9c7f795e0>

Checking shap values for RF0...

[array([[ 0.        ,  0.04181896,  0.00365806, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.01898534, -0.01999966, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.01898534, -0.01840236, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        ,  0.04012049,  0.02614831, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.01898534, -0.00631506, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.01898534, -0.01061794, ...,  0.        ,
         0.        ,  0.        ]]), array([[ 0.        , -0.04181896, -0.00365806, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.01898534,  0.01999966, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.01898534,  0.01840236, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        , -0.04012049, -0.02614831, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.01898534,  0.00631506, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.01898534,  0.01061794, ...,  0.        ,
         0.        ,  0.        ]])]

Checking shap plots for RF0...

Expected value for RF: [0.5728125 0.4271875]
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:


Checking feature importance for RF0...
```

```
None
```

```
RF1 In CV1...

Checking if correct model is loaded...
 RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=2, max_features='log2', min_samples_leaf=9,
                       min_samples_split=31, n_estimators=207, random_state=42)
           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 RF1...
<shap.explainers._tree.Tree object at 0x7fb9c74f0dc0>

Checking shap values for RF1...

[array([[ 3.95998286e-03,  8.78166228e-05,  5.42231605e-02, ...,
        -4.99207302e-03,  7.28955367e-03,  2.80322166e-03],
       [-3.19678857e-03,  8.79064012e-04,  1.60776918e-03, ...,
         1.80742612e-02,  5.37725417e-03, -1.63881455e-03],
       [-2.91886671e-03, -4.37616095e-04, -2.66173097e-02, ...,
         2.10142415e-02,  3.35299303e-03,  5.93880632e-04],
       ...,
       [ 2.62833254e-03, -1.78372567e-04,  1.24812736e-02, ...,
         1.73079030e-02,  4.94635866e-03, -1.53940799e-03],
       [ 2.33387141e-03, -4.42763000e-04, -2.58700195e-02, ...,
        -6.86380765e-03,  4.07652440e-03, -1.52713434e-03],
       [-5.01472471e-03,  1.13411715e-03,  5.63221558e-02, ...,
        -5.51702530e-03,  5.22884231e-03, -4.18441541e-04]]), array([[-3.95998286e-03, -8.78166228e-05, -5.42231605e-0
2, ...,
         4.99207302e-03, -7.28955367e-03, -2.80322166e-03],
       [ 3.19678857e-03, -8.79064012e-04, -1.60776918e-03, ...,
        -1.80742612e-02, -5.37725417e-03,  1.63881455e-03],
       [ 2.91886671e-03,  4.37616095e-04,  2.66173097e-02, ...,
        -2.10142415e-02, -3.35299303e-03, -5.93880632e-04],
       ...,
       [-2.62833254e-03,  1.78372567e-04, -1.24812736e-02, ...,
        -1.73079030e-02, -4.94635866e-03,  1.53940799e-03],
       [-2.33387141e-03,  4.42763000e-04,  2.58700195e-02, ...,
         6.86380765e-03, -4.07652440e-03,  1.52713434e-03],
       [ 5.01472471e-03, -1.13411715e-03, -5.63221558e-02, ...,
         5.51702530e-03, -5.22884231e-03,  4.18441541e-04]])]

Checking shap plots for RF1...

Expected value for RF: [0.49673858 0.50326142]
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:


Checking feature importance for RF1...
```

```
None
RF2 In CV2...

Checking if correct model is loaded...
 RandomForestClassifier(max_depth=11, max_features=None, min_samples_leaf=14,
                       min_samples_split=27, n_estimators=10, random_state=42)
           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 RF2...
<shap.explainers._tree.Tree object at 0x7fb9c773fa30>

Checking shap values for RF2...

[array([[ 0.01794027, -0.01378879,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.00711062, -0.00448347,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.00711062,  0.00318182,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.00711062, -0.00448347,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.01210694,  0.00318182,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.01794027, -0.01378879,  0.        , ...,  0.        ,
         0.        ,  0.        ]]), array([[-0.01794027,  0.01378879,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.00711062,  0.00448347,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.00711062, -0.00318182,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-0.00711062,  0.00448347,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.01210694, -0.00318182,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.01794027,  0.01378879,  0.        , ...,  0.        ,
         0.        ,  0.        ]])]

Checking shap plots for RF2...

Expected value for RF: [0.61909091 0.38090909]
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:


Checking feature importance for RF2...

None
```

```
XGB
XGB0 In CV0...

Checking if correct model is loaded...
 XGBClassifier(alpha=0.0002575842389979265, base_score=0.5, booster='gbtree',
              callbacks=None, colsample_bylevel=1, colsample_bynode=1,
              colsample_bytree=0.9181376162919086, early_stopping_rounds=None,
              enable_categorical=False, eta=5.623331491160975e-07,
              eval_metric=None, gamma=0.0002786718840103683, gpu_id=-1,
              grow_policy='lossguide', importance_type=None,
              interaction_constraints='', learning_rate=5.62333128e-07,
              max_bin=256, max_cat_to_onehot=4, max_delta_step=0, max_depth=27,
              max_leaves=0, min_child_weight=0.20525460238584922,
              min_samples_leaf=27, min_samples_split=37, missing=nan,
              monotone_constraints='()', n_estimators=164, n_jobs=1, nthread=1, ...)
            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 XGB0...
<shap.explainers._tree.Tree object at 0x7fb9c78890d0>

Checking shap values for XGB0...

[[-3.0952360e-06 -3.7619997e-05 -2.3696571e-06 ... -1.2713954e-07
   1.3318062e-07 -2.2155659e-06]
 [-1.1003718e-05  1.4350392e-05  2.1088497e-06 ...  6.4105130e-08
   4.7997673e-07  2.5433717e-06]
 [-3.2040905e-06  1.4657915e-05  1.6045622e-05 ...  2.1734811e-08
   1.6514244e-07 -8.6456166e-06]
 ...
 [ 1.4764141e-06 -2.2817851e-05 -1.9707142e-05 ...  4.7425072e-08
   1.5751922e-07  6.8362704e-07]
 [ 1.7882336e-05  9.4924808e-06  9.1785603e-07 ...  2.1734811e-08
  -1.7996480e-06  7.2638748e-09]
 [-3.0676049e-06  1.4631669e-05  5.5639280e-06 ...  2.1734811e-08
  -1.6272652e-06  2.0639069e-07]]

Checking shap plots for XGB0...

Expected value for XGB: 1.0591810450932826e-06
Summary Plot for SHAP Values in Test Set:

SHAP Bar Plot for SHAP Values Test Set:

SHAP Decision Plot for SHAP Values in Test Set:

SHAP Decision Plot for Single-Prediction in Test Set:


Checking feature importance for XGB0...

None
```

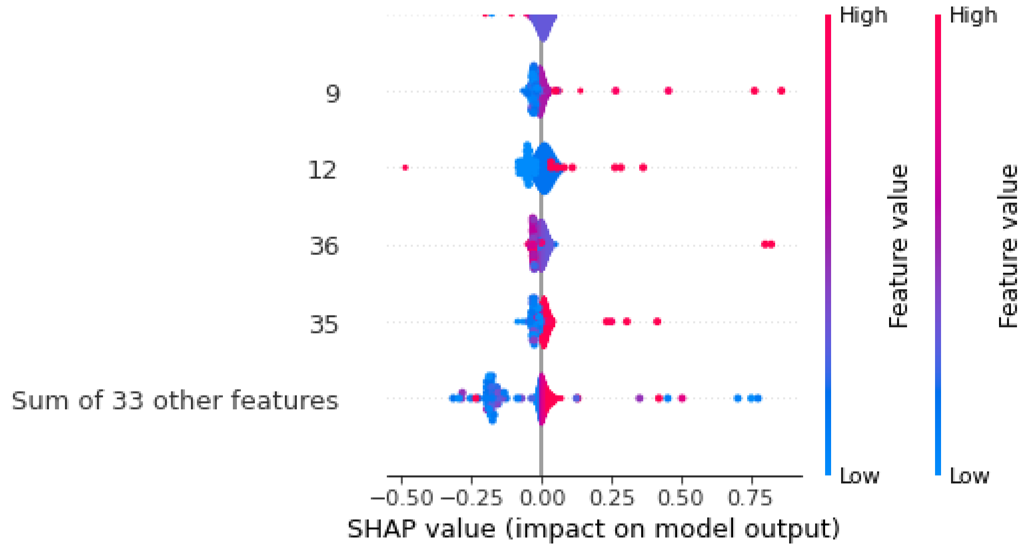
```
XGB1 In CV1...

Checking if correct model is loaded...
 XGBClassifier(alpha=0.00029260435288728723, base_score=0.5, booster='gbtree',
              callbacks=None, colsample_bylevel=1, colsample_bynode=1,
              colsample_bytree=0.5441411005619007, early_stopping_rounds=None,
              enable_categorical=False, eta=0.05120369776687421,
              eval_metric=None, gamma=0.4526660690706259, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.0512036979,
              max_bin=256, max_cat_to_onehot=4, max_delta_step=0, max_depth=18,
              max_leaves=0, min_child_weight=0.12415100550271539,
              min_samples_leaf=9, min_samples_split=27, missing=nan,
              monotone_constraints='()', n_estimators=464, n_jobs=1, nthread=1, ...)
           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 XGB1...
<shap.explainers._tree.Tree object at 0x7fb9602ff940>

Checking shap values for XGB1...

[[-0.45429307 -0.06374221 -0.94646686 ...  0.2016137  -0.11681356
  -0.14071724]
 [ 0.5519699   0.13245122 -0.10298917 ... -0.83697766 -0.3408652
  -0.07272914]
 [ 0.21548487 -0.07396804  0.14945313 ... -0.72553927  0.11201834
   0.11979318]
 ...
 [-0.237244   -0.04841679 -0.3906867  ... -0.8205372  -0.11913119
  -0.15298116]
 [-0.33601356 -0.06487641  0.14479543 ...  0.26082134  0.02685894
  -0.13012125]
 [ 0.47213364 -0.15212956 -0.5969934  ...  0.20478216 -0.17158583
  -0.18285887]]

Checking shap plots for XGB1...

Expected value for XGB: 1.5605792999267578
Summary Plot for SHAP Values in Test Set:

SHAP Bar Plot for SHAP Values Test Set:

SHAP Decision Plot for SHAP Values in Test Set:

SHAP Decision Plot for Single-Prediction in Test Set:


Checking feature importance for XGB1...

None
```

```
XGB2 In CV2...

Checking if correct model is loaded...
 XGBClassifier(alpha=5.77534955247629e-07, base_score=0.5, booster='gbtree',
              callbacks=None, colsample_bylevel=1, colsample_bynode=1,
              colsample_bytree=0.41771820514444086, early_stopping_rounds=None,
              enable_categorical=False, eta=8.67291826605322e-06,
              eval_metric=None, gamma=0.07212410933578818, gpu_id=-1,
              grow_policy='lossguide', importance_type=None,
              interaction_constraints='', learning_rate=8.67291874e-06,
              max_bin=256, max_cat_to_onehot=4, max_delta_step=0, max_depth=22,
              max_leaves=0, min_child_weight=6.66045104839759,
              min_samples_leaf=11, min_samples_split=39, missing=nan,
              monotone_constraints='()', n_estimators=884, n_jobs=1, nthread=1, ...)
           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 XGB2...
<shap.explainers._tree.Tree object at 0x7fb960bf3700>

Checking shap values for XGB2...

[[-1.8894803e-04  2.3255567e-04  0.0000000e+00 ...  0.0000000e+00
  -8.9294117e-06 -7.9064384e-06]
 [-1.9001741e-04  1.9959988e-04  0.0000000e+00 ...  0.0000000e+00
  -1.0370755e-05  3.5941350e-06]
 [-1.6012945e-04  1.6149672e-04  0.0000000e+00 ...  0.0000000e+00
  -1.5008896e-05  3.5941350e-06]
 ...
 [-1.7405280e-04  2.5253580e-04  0.0000000e+00 ...  0.0000000e+00
  -1.0370755e-05  3.5941350e-06]
 [-1.8333328e-04 -4.4896262e-04  0.0000000e+00 ...  0.0000000e+00
  -1.5008896e-05  3.5941350e-06]
 [-1.6480772e-04  2.5592663e-04  0.0000000e+00 ...  0.0000000e+00
  -1.3567553e-05  3.5941350e-06]]

Checking shap plots for XGB2...

Expected value for XGB: -8.377160702366382e-05
Summary Plot for SHAP Values in Test Set:

SHAP Bar Plot for SHAP Values Test Set:

SHAP Decision Plot for SHAP Values in Test Set:

SHAP Decision Plot for Single-Prediction in Test Set:


Checking feature importance for XGB2...

None
```

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

Checking if correct model is loaded...
 GaussianNB()
            0         1         2         3         4         5         6  \
0   -0.331668  1.939494  0.584349  2.245540 -0.138787 -0.641236  0.166887
1   -0.952737  0.170028 -1.711307 -0.578134 -0.143023 -0.641236 -0.915738
2   -0.214169 -0.405048  0.584349  0.685530  0.096434 -0.641236 -0.047382
3   -0.684167  0.022573  0.584349 -0.449531 -0.142841  2.180204 -0.126324
4   -0.230954  1.703565  0.584349 -0.639640 -0.142962 -0.641236 -0.521031
..        ...       ...       ...       ...       ...       ...       ...
105  1.195826 -0.714705 -1.711307  2.474789  0.666324  2.180204  1.892322
106  0.104759  1.202217 -1.711307 -0.880072 -0.142794  0.769484 -0.408257
107  0.171901 -1.746893  0.584349  1.311771  0.113007 -0.641236  0.324770
108  5.912592  1.202217  0.584349 -0.298562 -0.142852 -0.641236  1.441228
109 -0.516310 -1.894348  0.584349 -0.656414 -0.143004  0.769484 -0.261652

            7         8         9  ...        27        28        29  \
0   -0.397360  0.349927 -0.453232  ... -0.240192  1.115648  0.062698
1    2.516611 -2.857738  0.646464  ... -0.240192 -1.095547 -0.793494
2   -0.397360  0.349927  0.770026  ... -0.240192  1.115648  0.146057
3   -0.397360  0.349927 -0.218466  ... -0.240192  1.115648  0.052102
4   -0.397360  0.349927 -0.650931  ... -0.240192 -0.542748  0.020783
..        ...       ...       ...  ...       ...       ...       ...
105 -0.397360  0.349927  4.625141  ... -0.240192  1.115648 -1.316706
106 -0.397360  0.349927 -0.379096  ... -0.240192  0.010051  0.083420
107 -0.397360  0.349927  3.550157  ... -0.240192  1.115648 -0.137463
108 -0.397360  0.349927 -0.218466  ...  4.163332  1.115648  0.678469
109 -0.397360  0.349927 -0.638575  ... -0.240192  0.010051  0.741106

           30        31        32        33        34        35        36
0    0.398006 -0.858225 -0.542326  0.816497  0.626422 -0.406355 -0.039166
1   -0.456207  0.015893 -0.542326 -1.224745 -1.596367 -0.438538 -0.182909
2   -0.456207 -0.858225 -0.542326  0.816497 -1.596367 -0.164983 -0.137990
3   -0.087673 -0.858225  1.843909  0.816497  0.626422  0.012023 -0.227829
4   -0.071888 -0.858225 -0.542326  0.816497  0.626422 -0.309806 -0.093070
..        ...       ...       ...       ...       ...       ...       ...
105  0.557426  2.638246 -0.542326  0.816497  0.626422  5.981948 -0.227829
106  0.199352 -0.858225 -0.542326  0.816497 -1.596367 -0.454629 -0.120022
107  0.616729 -0.858225 -0.542326  0.816497  0.626422  4.050974 -0.200877
108 -0.436341 -0.858225 -0.542326  0.816497  0.626422 -0.470720 -0.021199
109 -0.020512  0.890011 -0.542326  0.816497  0.626422 -0.406355 -0.344620

[110 rows x 37 columns]

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

Checking shap values for NB0...

.values =
array([[ 0.0075    , -0.02083333,  0.00166667, ...,  0.00916667,
         -0.03      , -0.02916667],
       [ 0.00166667,  0.0075    ,  0.        , ...,  0.02      ,
         -0.03666667, -0.0275    ],
       [ 0.01      , -0.01      ,  0.        , ...,  0.0575    ,
         -0.00833333, -0.0275    ],
       ...,
       [ 0.00166667, -0.01      ,  0.        , ...,  0.00416667,
         -0.02166667, -0.03083333],
       [-0.0025    , -0.0075    ,  0.        , ..., -0.0125    ,
         -0.0275    , -0.03166667],
       [ 0.00333333,  0.02416667,  0.        , ..., -0.06083333,
          0.41583333, -0.0225    ]])

.base_values =
array([0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32,
       0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32,
       0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32,
       0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32,
       0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32, 0.32])

.data =
array([[ 0.0376161,  1.0547612, -1.7113069, ...,  0.6264224, -0.4385377,
        -0.1829094],
       [-0.331668 , -0.2723382, -1.7113069, ...,  0.6264224, -0.3902633,
        -0.3086845],
       [-0.096669 ,  0.3174837, -1.7113069, ...,  0.6264224, -0.486812 ,
        -0.1829094],
       ...,
       [-0.667381 ,  0.6123947,  0.5843487, ...,  0.6264224, -0.3741719,
        -0.1290058],
       [-0.9191656,  0.3174837,  0.5843487, ..., -1.5963668, -0.2615318,
        -0.1379898],
       [ 0.13833  , -0.346066 ,  0.5843487, ..., -1.5963668,  1.154515 ,
        -0.1379898]])

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...

Symptoms  is Symptoms
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [15], in <cell line: 7>()
     59 shap_fi_df.to_csv(shapFI_path, header=True, index=True)
     61 # create masterList of SHAP Values (not FI) for each model
---> 62 save = save_shap(abbrev[algorithm], shap_values, original_headers, cvCount, 'Test')
     63 display(save)

Input In [13], in save_shap(abbrev, shap_values, original_headers, cvCount, dataset)
     22             print(f'{name} is {name}\n')
     23 #             print(f'Index is {index}')
     24 #             print(f'Shap value is {shap_vals[index]}')
     25 #             print(f'{name} value is {shap_vals[index]}')
---> 26             temp_list.append(shap_vals[index])
     27         else:
     28             temp_list.append(0.0)

IndexError: index 37 is out of bounds for axis 0 with size 37
```



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 [ ]: