In [1]:
```
# required packages & models
import os
import sys
import pickle
import warnings
warnings.filterwarnings('ignore')
import csv
import sklearn
import shap
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 import tree
from sklearn.datasets import load_iris

#import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import numpy as np
import pandas as pd
from termcolor import colored as cl #text customization
shap.initjs() # load JS visualization code to notebook. SHAP plots won't be displayed without this
```

js

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

Things to do:

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

Notes

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

Run Parameters

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

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

Load Metadata and Other Necessary Variables

In [47]:
```
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'])

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


abbrev = {}
for key in algInfo:  # pickling specific model while also checking for corresponding algInfo
    if key in ["Logistic Regression"]: # If that algorithm was used
```

```
        algorithms.append(key)
print(algorithms)
```

```
['Logistic Regression']
```

load_model(): Load One Trained Model at a Time

In [48]:
```
def load_model():
  # this method will load the pickled model that is chosen by user (hardcoded for time being)
  # should return model object


    model_file = experiment_path + '/hcc-data_example/models/pickledModels/LR_0.pickle'
    file = open(model_file, 'rb')
    trained_model = pickle.load(file)
    file.close()

    return trained_model
```

In [49]:
```
#test load_model() method
load_model()
```

Out[49]:
```
LogisticRegression(C=7.666887654082441e-05, class_weight='balanced',
                   max_iter=446.50817382752405, random_state=42, solver='sag')
```

dataPrep(): Loading target CV Training & Testing Sets

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

    train = pd.read_csv(train_file_path)
    if instance_label != 'None':
        train = train.drop(instance_label,axis=1)
    trainX = train.drop(class_label,axis=1).values



    feature_names = []  # create list of feature names from CV0 training set
    for feature in train.columns.tolist():
        feature_names.append(feature)
    print('\nChecking feature names in columns from X-variable: \n', feature_names)

    trainY = train[class_label].values
    del train #memory cleanup


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



    return trainX, trainY, testX, testY, feature_names
```

In [7]:
```
#test data_prep() method
trainX, trainY,testX, testY, feature_names = dataPrep(train_file_path,instance_label,class_label, test_file_path)
print('\nChecking testX for CV0 values...\n', testX)
```

```
Checking feature names in columns from X-variable:
 ['Class', 'Alanine transaminase (U/L)', 'Albumin (mg/dL)', 'Alkaline phosphatase (U/L)', 'Alpha-Fetoprotein (ng/mL)',
 'Arterial Hypertension', 'Ascites degree*', 'Aspartate transaminase (U/L)', 'Chronic Renal Insufficiency', 'Cirrhosi
s', 'Creatinine (mg/dL)', 'Diabetes', 'Direct Bilirubin (mg/dL)', 'Encephalopathy degree*', 'Endemic Countries', 'Esop
hageal Varices', 'Ferritin (ng/mL)', 'Gamma glutamyl transferase (U/L)', 'Haemoglobin (g/dL)', 'Hemochromatosis', 'Hep
atitis B Surface Antigen', 'Hepatitis B e Antigen', 'Hepatitis C Virus Antibody', 'International Normalised Ratio*',
 'Iron', 'Leukocytes(G/L)', 'Liver Metastasis', 'Major dimension of nodule (cm)', 'Mean Corpuscular Volume', 'Number of
Nodules', 'Obesity', 'Oxygen Saturation (%)', 'Packs of cigarets per year', 'Performance Status*', 'Portal Hypertensio
n', 'Portal Vein Thrombosis', 'Smoking', 'Splenomegaly', 'Symptoms ', 'Total Bilirubin(mg/dL)']

Checking testX for CV0 values...
           0         1         2         3         4         5         6   \
0   0.036900  1.055129 -0.585210 -0.143022  1.272418 -0.641236 -0.521876
1  -0.332366 -0.271973  3.887411 -0.142592  1.272418 -0.641236 -0.894012
2  -0.097379  0.317850  0.197499 -0.133145 -0.785905 -0.641236 -0.442938
3  -0.433074  0.730726 -0.484576 -0.143009 -0.785905 -0.641236 -0.533153
4  -0.735201 -0.419428 -0.288899 -0.143010 -0.785905  0.769484 -0.465492
5  -0.684846  0.170395 -0.389533 -0.142973 -0.785905 -0.641236 -0.815074
6  -0.130948  0.612761  0.616807 -0.142846 -0.785905 -0.641236 -0.499322
7  -0.869479  1.350040 -0.266536 -0.142986  1.272418 -0.641236 -0.848905
8  -0.550568  1.202584 -0.551665 -0.135894  1.272418 -0.641236 -0.149740
9  -0.718416  0.022939 -0.965383 -0.143021  1.272418  0.769484 -0.747413
10 -0.063809  0.317850 -0.199446 -0.141651  1.272418 -0.641236  0.380273
11  0.607582  1.055129 -0.797659 -0.143020  1.272418 -0.641236 -0.273785
12 -0.550568  2.234774 -0.892703 -0.143021 -0.785905 -0.641236 -0.781244
13 -0.030240 -0.832304  0.504991 -0.139486 -0.785905 -0.641236 -0.544430
14  1.111126 -0.154008 -0.456622 -0.013254 -0.785905  0.769484  0.899008
15 -0.500214 -1.599074 -0.562847 -0.143019  1.272418  0.769484 -0.273785
16  0.825785  0.907673 -0.618755 -0.140311 -0.785905 -0.641236  0.335165
17  1.077556  0.465306 -0.411896 -0.142540  1.272418 -0.641236  0.786240
18  0.859354 -1.525346 -0.093222 -0.142235 -0.785905 -0.641236  0.955392
19  2.336415  1.055129 -0.803250 -0.142898 -0.785905 -0.641236  0.673471
20  1.329328  1.055129 -0.663481 -0.142754  1.272418 -0.641236  2.342446
21  2.134998 -0.566884 -0.752933 -0.142996 -0.785905 -0.641236  3.413747
22  0.372595  1.497496  0.046548  0.439090 -0.785905  0.769484  2.658198
23 -0.567353  0.022939 -0.434259 -0.142488 -0.785905 -0.641236 -0.645921
24 -0.516998  0.612761 -0.372760 -0.142996  1.272418 -0.641236 -0.679752
25 -0.147733 -0.566884  0.365222 -0.124241 -0.785905 -0.641236 -0.149740
26 -0.265227 -0.419428  0.029775 -0.066079 -0.785905 -0.641236  0.098351
27  0.422950  0.170395 -0.350397 -0.143008 -0.785905 -0.641236 -0.149740
28 -0.617707  0.907673 -0.283308 -0.142327 -0.785905  2.180204 -0.431661
29 -0.835909  1.350040 -0.484576 -0.142933 -0.785905 -0.641236 -0.781244
30  0.892924 -0.419428 -0.115585 -0.142800 -0.785905  0.769484  0.752409
31 -0.550568  1.202584 -0.663481 -0.142933 -0.785905 -0.641236 -0.555706
32 -0.567353 -1.009251 -0.277717 -0.124339 -0.785905  0.769484 -0.566983
33 -0.433074  0.612761  0.471447 -0.142812  1.272418 -0.641236 -0.273785
34  0.490089  1.202584 -0.831204 -0.143007 -0.785905 -0.641236 -0.330169
35 -0.466644  1.792407 -0.663481 -0.142921 -0.785905 -0.641236 -0.555706
36 -0.416290  1.055129 -0.635527 -0.143007 -0.785905 -0.641236 -0.397830
37  0.187963 -0.419428  0.404357 -0.086100 -0.785905 -0.641236  0.109628
38 -0.802340 -0.419428 -0.059677  2.290938 -0.785905  0.769484 -0.612091
39 -0.567353 -0.714340 -0.115585 -0.143022 -0.785905  2.180204 -0.510599
40 -0.147733 -1.746529  1.377152 -0.219638 -0.785905  2.180204 -0.533153
41 -0.433074  1.055129 -0.389533 -0.142794 -0.785905 -0.641236  0.222397
42  0.187963  0.170395  0.415539 -0.141997 -0.785905 -0.641236 -0.454215
43 -0.684846 -0.419428 -1.265719  0.050422  1.272418  0.769484 -0.206124
44 -0.365935  1.350040 -0.831204 -0.124535 -0.785905 -0.641236 -0.758690
45 -0.852694  0.465306 -0.792069 -0.142993 -0.785905  0.769484 -0.330169
46  0.087254 -2.188896  0.197499 -0.142967 -0.785905  0.769484 -0.149740
47 -0.231657  1.644951 -0.730570 -0.143014 -0.785905 -0.641236 -0.578260
48 -0.751985  0.745472 -0.316853 -0.143025 -0.785905  0.769484 -0.566983
49  0.042126  0.071298  0.001823 -0.000002  1.272418 -0.641236  0.018662
50 -0.617707  0.170395 -0.434259 -0.143013  1.272418  2.180204 -0.420384
51  3.259578 -1.009251 -0.903884 -0.142988 -0.785905  0.769484  2.669475
52 -0.668061  0.612761 -0.708207 -0.143022  1.272418 -0.641236 -0.679752
53 -0.919833  0.317850 -0.484576 -0.142924 -0.785905 -0.641236 -0.713582
54  0.137608 -0.345700 -0.361579 -0.142869  1.272418  0.769484  0.199843

          7         8         9   ...        29        30        31        32  \
0  -0.397360  0.349927 -0.317236  ...  2.731582  0.131983 -0.456207  0.015893
1  -0.397360 -2.857738  0.288220  ... -0.366088 -0.955219 -0.456207  0.015893
2   2.516611  0.349927  0.510632  ... -0.366088  0.093059  0.183378  1.764128
3  -0.397360  0.349927  0.028739  ... -0.366088  0.588800  0.070747 -0.858225
4  -0.397360  0.349927 -0.341949  ... -0.366088 -0.622996 -0.456207  0.015893
5  -0.397360  0.349927 -0.453155  ... -0.366088  0.125907 -0.297284  0.890011
6  -0.397360  0.349927  0.065807  ... -0.366088  0.157871 -0.456207  1.764128
7   2.516611  0.349927  4.785892  ... -0.366088  0.527965  1.093295  0.015893
8  -0.397360  0.349927 -0.020686  ... -0.366088 -0.271313 -0.091604 -0.858225
9  -0.397360  0.349927 -0.280168  ... -0.366088 -0.846793 -0.257553  0.890011
10 -0.397360  0.349927 -0.218386  ...  2.731582  0.329133 -0.217822  0.890011
11 -0.397360  0.349927 -0.033043  ... -0.366088  0.066771 -0.065333 -0.858225
12 -0.397360  0.349927 -0.440799  ... -0.366088 -0.168704 -0.456207 -0.858225
13 -0.397360  0.349927 -0.539649  ... -0.366088 -0.039772 -0.456207 -0.858225
14 -0.397360  0.349927  2.660618  ... -0.366088  1.103445 -0.121973  2.638246
15 -0.397360  0.349927 -0.243099  ... -0.366088  1.806810 -0.456207  2.638246
16 -0.397360  0.349927 -0.477867  ... -0.366088  0.078580 -0.089353 -0.858225
17 -0.397360  0.349927  8.060296  ... -0.366088  0.272196 -0.456207 -0.858225
18 -0.397360  0.349927 -0.341949  ... -0.366088  2.989742 -0.456207 -0.858225
19  2.516611  0.349927 -0.218386  ...  2.731582  0.096309 -0.456207 -0.858225
20  2.516611  0.349927  0.893675  ... -0.366088 -0.559053  0.139755 -0.858225
21 -0.397360  0.349927 -0.094824  ... -0.366088  0.591907  0.278813 -0.858225
22 -0.397360  0.349927 -0.341949  ... -0.366088 -0.032923 -0.284234 -0.858225
23 -0.397360  0.349927 -0.465511  ... -0.366088  0.387780 -0.158226 -0.858225
```

```
24 -0.397360 -2.857738 -0.453155  ... -0.366088 -1.038620 -0.456207 -0.858225
25 -0.397360  0.349927 -0.502580  ... -0.366088 -0.175400  0.884708  0.015893
26 -0.397360  0.349927 -0.354305  ... -0.366088 -0.047515 -0.456207  0.890011
27 -0.397360  0.349927 -0.465511  ... -0.366088  1.742868 -0.106537  0.015893
28 -0.397360  0.349927 -0.465511  ... -0.366088 -0.686938 -0.456207  0.890011
29 -0.397360 -2.857738 -0.280168  ... -0.366088 -0.047515 -0.456207 -0.858225
30 -0.397360  0.349927 -0.477867  ... -0.366088  1.998637  0.338409 -0.858225
31 -0.397360  0.349927 -0.070111  ... -0.366088  0.272252 -0.093523 -0.858225
32 -0.397360  0.349927 -0.564361  ...  2.731582 -0.165429  0.139755  0.890011
33 -0.397360 -2.857738 -0.082468  ... -0.366088 -0.239342  0.537064  0.015893
34 -0.397360  0.349927 -0.070111  ...  2.731582  0.549514 -0.456207 -0.858225
35 -0.397360  0.349927 -0.341949  ... -0.366088  0.847676  0.338409 -0.858225
36 -0.397360  0.349927  0.016382  ... -0.366088  0.454056 -0.102416 -0.858225
37 -0.397360  0.349927 -0.687924  ... -0.366088 -0.239342 -0.058899  1.764128
38 -0.397360  0.349927 -0.465511  ... -0.366088 -0.495111  0.336176  2.638246
39 -0.397360  0.349927 -0.250530  ... -0.366088  1.167388 -0.268835  0.890011
40 -0.397360  0.349927  1.993381  ... -0.366088  0.276223  0.537064  0.015893
41 -0.397360  0.349927 -0.243099  ... -0.366088  0.228510 -0.307216  0.015893
42 -0.397360  0.349927 -0.589074  ... -0.366088 -0.263724 -0.456207 -0.858225
43 -0.397360  0.349927  0.522988  ... -0.366088 -0.559053  0.497333  0.890011
44 -0.397360  0.349927 -0.094824  ... -0.366088  0.140340 -0.456207 -0.858225
45  2.516611  0.349927  1.981025  ... -0.366088 -0.942707 -0.456207  1.764128
46 -0.397360  0.349927 -0.107180  ... -0.366088 -0.399198 -0.456207  1.764128
47 -0.397360  0.349927 -0.403730  ... -0.366088 -0.207371  0.477467 -0.858225
48 -0.397360  0.349927 -0.514936  ... -0.366088  0.306720 -0.456207  0.890011
49 -0.397360  0.349927  0.026322  ... -0.366088 -0.136319  0.000017 -0.858225
50 -0.397360  0.349927 -0.465511  ...  2.731582  0.149268 -0.456207  0.890011
51 -0.397360  0.349927  0.152301  ... -0.366088  0.246126 -0.456207  0.890011
52 -0.397360  0.349927  0.028739  ... -0.366088 -0.162587 -0.158226 -0.858225
53 -0.397360  0.349927 -0.490224  ...  2.731582  0.703114  0.735718 -0.858225
54 -0.397360  0.349927 -0.008330  ... -0.366088  1.646954 -0.268802  0.890011

          33        34        35        36        37        38
0  -1.671258 -0.542326 -1.349264 -1.224745  0.626422 -0.438375
1  -1.671258 -0.542326 -1.349264  0.816497  0.626422 -0.390099
2   0.598352 -0.542326  0.741145  0.816497  0.626422 -0.486650
3   0.598352 -0.542326  0.741145  0.816497  0.626422 -0.412628
4   0.598352  1.843909 -1.349264  0.816497 -1.596367 -0.036083
5   0.598352  1.843909  0.741145  0.816497  0.626422 -0.422283
6   0.598352 -0.542326 -1.349264 -1.224745  0.626422 -0.148725
7  -1.671258 -0.542326  0.741145 -1.224745  0.626422 -0.390099
8   0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.374008
9   0.598352 -0.542326  0.741145 -1.224745  0.626422 -0.454466
10  0.598352 -0.542326  0.741145 -1.224745  0.626422 -0.454466
11 -1.671258 -0.542326  0.741145  0.816497  0.626422 -0.357916
12 -1.671258 -0.542326 -1.349264 -1.224745 -1.596367 -0.390099
13  0.598352 -0.542326 -1.349264  0.816497 -1.596367 -0.052175
14  0.598352 -0.542326  0.741145  0.816497  0.626422  2.522490
15  0.598352 -0.542326 -1.349264  0.816497  0.626422  0.720225
16  0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.148725
17  0.598352 -0.542326 -1.349264 -1.224745 -1.596367 -0.325733
18 -1.671258 -0.542326 -1.349264 -1.224745 -1.596367 -0.097231
19  0.598352 -0.542326 -1.349264 -1.224745  0.626422 -0.309641
20 -1.671258 -0.542326  0.741145 -1.224745  0.626422 -0.422283
21  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.293550
22  0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.052175
23 -1.671258 -0.542326  0.741145 -1.224745 -1.596367 -0.197000
24  0.598352 -0.542326 -1.349264  0.816497 -1.596367 -0.309641
25  0.598352  1.843909  0.741145  0.816497 -1.596367 -0.357916
26  0.598352  1.843909 -1.349264  0.816497 -1.596367 -0.277458
27  0.598352 -0.542326  0.741145  0.816497 -1.596367  0.205292
28  0.598352 -0.542326 -1.349264  0.816497  0.626422  0.157017
29 -1.671258 -0.542326 -1.349264 -1.224745  0.626422 -0.406191
30  0.598352 -0.542326  0.741145  0.816497  0.626422  0.060467
31  0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.261366
32  0.598352 -0.542326  0.741145  0.816497 -1.596367  0.253567
33 -1.671258 -0.542326  0.741145 -1.224745  0.626422 -0.374008
34  0.598352 -0.542326 -1.349264 -1.224745 -1.596367  0.736316
35  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.197000
36  0.598352 -0.542326  0.741145 -1.224745  0.626422 -0.390099
37  0.598352  1.843909  0.741145  0.816497  0.626422 -0.019991
38  0.598352  1.843909  0.741145 -1.224745  0.626422 -0.293550
39  0.598352  1.843909  0.741145  0.816497  0.626422  0.012192
40  0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.293550
41  0.598352  1.843909  0.741145  0.816497  0.626422 -0.325733
42 -1.671258  1.843909 -1.349264 -1.224745  0.626422 -0.454466
43  0.598352  1.843909  0.741145  0.816497  0.626422 -0.164816
44 -1.671258 -0.542326 -1.349264 -1.224745 -1.596367 -0.422283
45  0.598352 -0.542326 -1.349264  0.816497  0.626422 -0.454466
46  0.598352 -0.542326 -1.349264  0.816497  0.626422  0.140925
47 -1.671258 -0.542326  0.741145 -1.224745 -1.596367 -0.374008
48  0.598352 -0.542326 -1.349264 -1.224745  0.626422 -0.390099
49 -1.671258 -0.542326  0.741145 -1.224745  0.626422 -0.015921
50 -1.671258  1.843909 -1.349264 -1.224745  0.626422 -0.438375
51  0.598352 -0.542326 -1.349264  0.816497 -1.596367  0.993783
52  0.598352 -0.542326  0.741145  0.816497  0.626422 -0.374008
53  0.598352 -0.542326  0.741145  0.816497 -1.596367 -0.261366
54  0.598352 -0.542326  0.741145  0.816497 -1.596367  1.154699

[55 rows x 39 columns]
```

SHAP: get_explainer()

In [37]:
```
def get_explainer(model, algorithms, trainX):
    # 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

    explainer = None
    trained_model = model

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

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

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

    return explainer
```

SHAP: compute_shapValues()

In [38]:
```
def compute_shapValues(model, algorithms, explainer, trainX, trainY, testX, testY):
  # this method will calculate shapley values
  # this includes creating expected_values and shap_values
  # returns shap_values (will be called by shap_summary)


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

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




    if algorithms[0] in ['Decision Tree', 'Random Forest', "Extreme Gradient Boosting", "Light Gradient Boosting","Cat
        shap_values= explainer.shap_values(testX)  # i think shap_values() only works for TreeExplainer...not sure yet
#         shap_values = explainer.shap_values(trainX)  --> .shap_values doesnt work for decision tree??????




    return shap_values
```

SHAP: shap_summary()

In [50]:
```
def shap_summary(algorithms, shap_values, explainer, trainX, testX):
    # retrieve shap_values from previous method
    # this method will return and display different types of shap plots

   # checks algorithm in given list to execute shap summaries
    if algorithms[0] in ["Naive Bayes", "Logistic Regression"]:
        print('SHAP summary for Test Set\n')
        shap_summary = shap.summary_plot(shap_values, testX, plot_type='violin')
        print('SHAP Beeswarm Plot for Test Set\n')
        shap_beeswarm = shap.plots.beeswarm(shap_values)
        print('SHAP Bar Plot for Test Set\n')
        shap_bar = shap.summary_plot(shap_values, testX, plot_type="bar")


    if algorithms[0] in ['Decision Tree', 'Random Forest', "Extreme Gradient Boosting", "Light Gradient Boosting","Cat
        print('SHAP summary for Test Set\n')
        #tree.tree_plot(testX)  ---> helps display Decision Tree
        shap_summary = shap.summary_plot(shap_values, testX)


    return [shap_summary, shap_beeswarm, shap_bar]
```

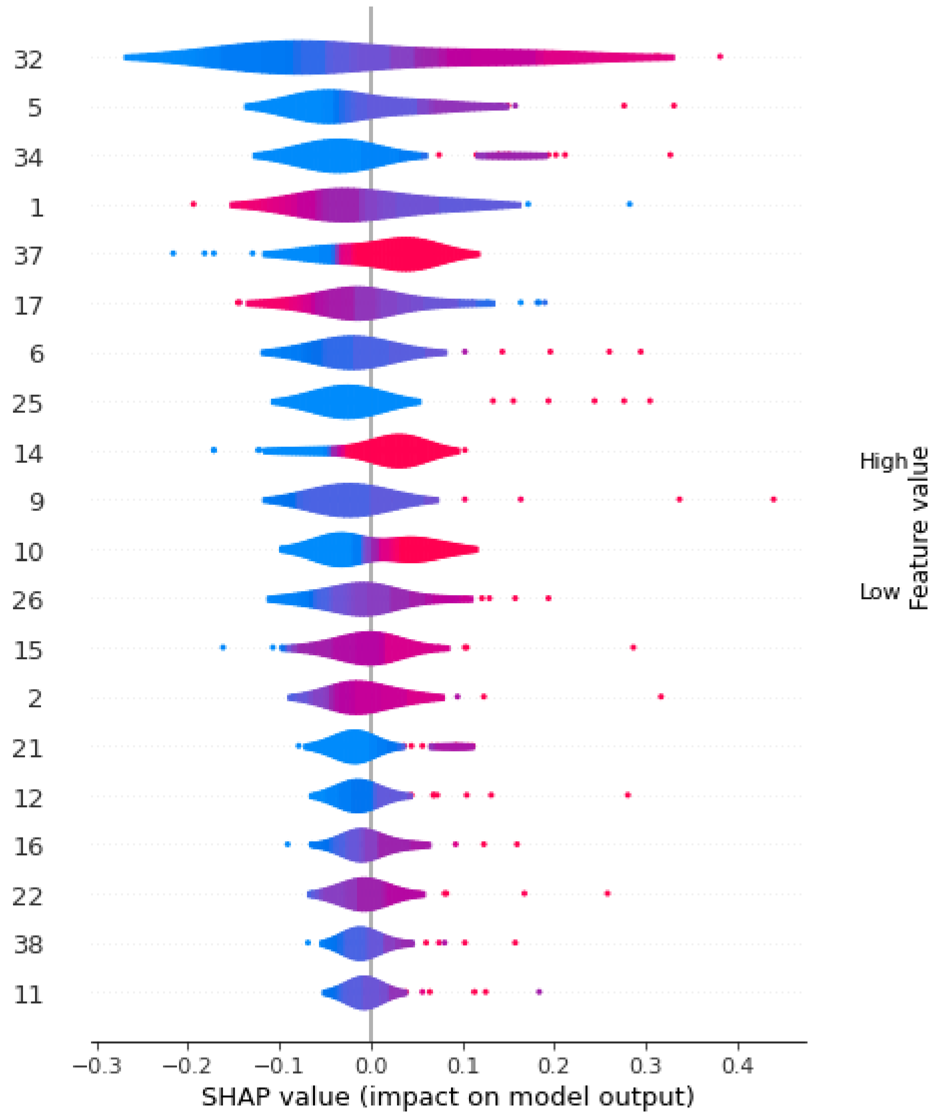
Testing All Functions

In [52]:
```
# testing all methods
model = load_model()  # load Logistic Regression model and algorithms list
print(model)
print(algorithms) # print to make sure variables are separated
```

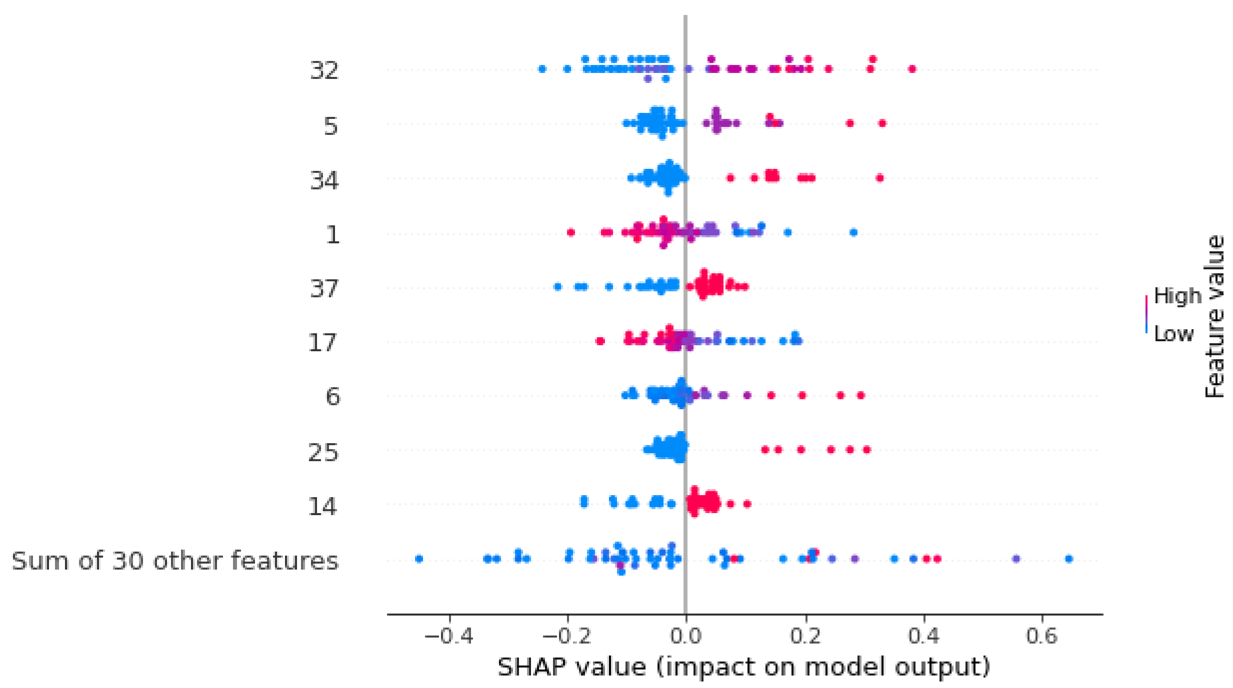
```
y_pred = model.predict(testX)  # calculate model prediction for trainX of CV0
probas_= model.predict_proba(testX)  # calculate model prediction probabilities for trainX of CV0
print('\nPredict_proba_ values: \n', probas_)
print('\n.Predict() values: \n',y_pred)  # print results to show model is being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




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


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

```
LogisticRegression(C=7.666887654082441e-05, class_weight='balanced',
                   max_iter=446.50817382752405, random_state=42, solver='sag')
['Logistic Regression']

Predict_proba_ values:
 [[0.50080677 0.49919323]
 [0.49795059 0.50204941]
 [0.49895507 0.50104493]
 [0.5005019  0.4994981 ]
 [0.50058415 0.49941585]
 [0.50091563 0.49908437]
 [0.500088   0.499912  ]
 [0.5003579  0.4996421 ]
 [0.50156036 0.49843964]
 [0.50048665 0.49951335]
 [0.50062842 0.49937158]
 [0.50200011 0.49799989]
 [0.50338211 0.49661789]
 [0.5011524  0.4988476 ]
 [0.49568778 0.50431222]
 [0.49858531 0.50141469]
 [0.50117175 0.49882825]
 [0.50038181 0.49961819]
 [0.49994903 0.50005097]
 [0.50044663 0.49955337]
 [0.50138927 0.49861073]
 [0.50029758 0.49970242]
 [0.50037199 0.49962801]
 [0.50146296 0.49853704]
 [0.50185595 0.49814405]
 [0.5001579  0.4998421 ]
 [0.50016198 0.49983802]
 [0.50106263 0.49893737]
 [0.49975696 0.50024304]
 [0.50236838 0.49763162]
 [0.50016145 0.49983855]
 [0.50266026 0.49733974]
 [0.49986703 0.50013297]
 [0.50085819 0.49914181]
 [0.50194072 0.49805928]
 [0.50215954 0.49784046]
 [0.50160764 0.49839236]
 [0.49903662 0.50096338]
 [0.49870913 0.50129087]
 [0.49948508 0.50051492]
 [0.49828659 0.50171341]
 [0.50038725 0.49961275]
 [0.50078312 0.49921688]
 [0.49925831 0.50074169]
 [0.50305413 0.49694587]
 [0.49830699 0.50169301]
 [0.49810239 0.50189761]
 [0.50277418 0.49722582]
 [0.50069838 0.49930162]
 [0.50045804 0.49954196]
 [0.49857858 0.50142142]
 [0.49883654 0.50116346]
 [0.5021134  0.4978866 ]
 [0.502136   0.497864  ]
 [0.49970779 0.50029221]]

.Predict() values:
 [0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0.
 0. 0. 1. 1. 0. 0. 1.]

Checking if explainer for model exists...
 <shap.explainers._permutation.Permutation object at 0x7f8978abf7c0>

Checking if shap values for model is returned...

.values =
array([[-3.08395285e-18, -9.09090909e-02, -3.83838384e-02, ...,
        -8.08080808e-03,  2.82828283e-02, -2.62626263e-02],
       [-8.08080808e-03,  4.44444444e-02,  3.17171717e-01, ...,
         0.00000000e+00,  4.84848485e-02, -2.02020202e-02],
       [-1.41414141e-02,  8.08080808e-03,  5.05050505e-02, ...,
         1.21212121e-02,  7.27272727e-02, -3.03030303e-02],
       ...,
       [-6.06060606e-03, -2.62626263e-02, -1.61616162e-02, ...,
         4.04040404e-03,  2.42424242e-02, -2.22222222e-02],
       [-1.01010101e-02, -8.08080808e-03, -3.43434343e-02, ...,
         0.00000000e+00, -1.81818182e-02, -4.04040404e-03],
       [-2.02020202e-03,  1.13131313e-01, -6.06060606e-03, ...,
         8.08080808e-03, -1.81818182e-01,  1.57575758e-01]])

.base_values =
array([0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
       0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
       0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
       0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
       0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
       0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
```

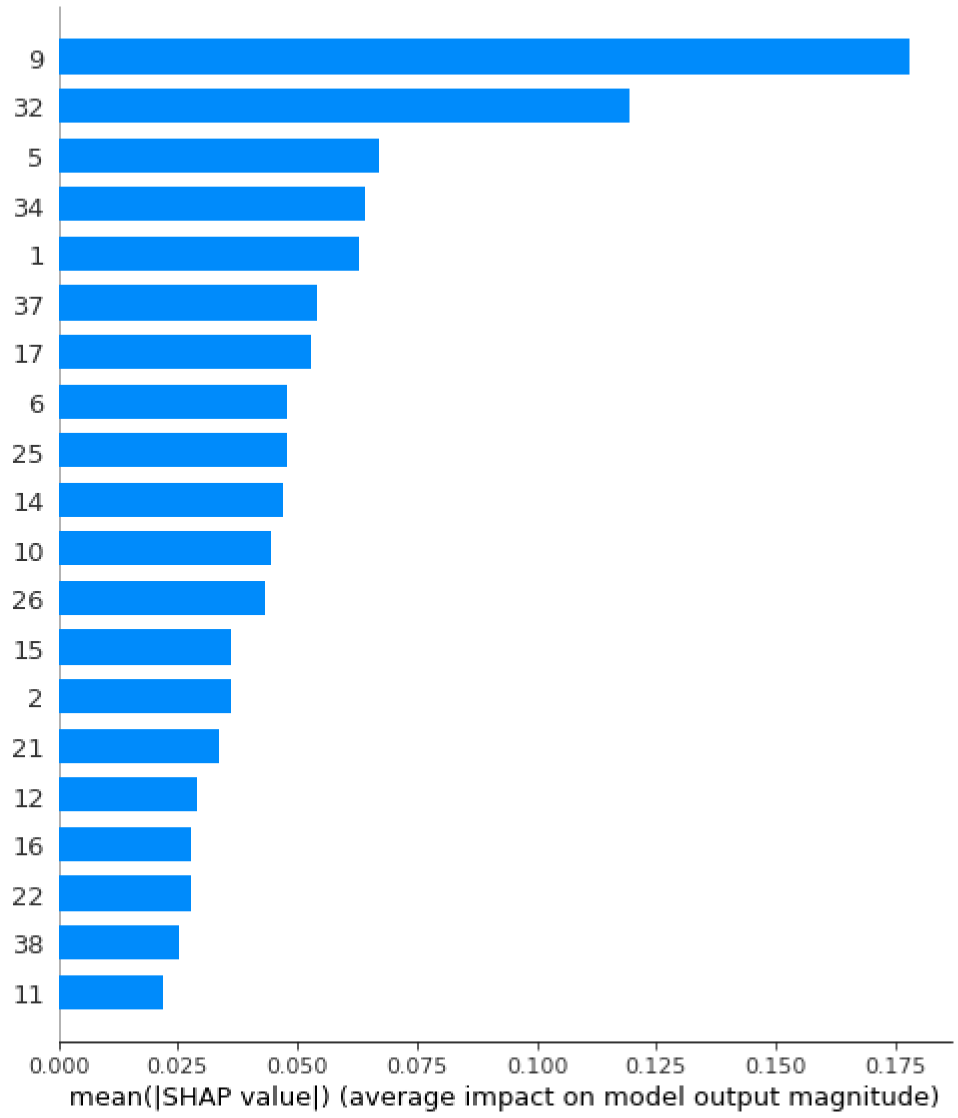
```
        0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
        0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
        0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
        0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091,
        0.30909091, 0.30909091, 0.30909091, 0.30909091, 0.30909091])

.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 if shap plots are returned and consistent...

SHAP summary for Test Set
```




```
SHAP Beeswarm Plot for Test Set
```




```
SHAP Bar Plot for Test Set
```



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

# print(metrics)
```