In [23]:
```
# required packages & models
import os
import sys
import pickle
import warnings
warnings.filterwarnings('ignore')
import csv
import sklearn
import shap
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
from termcolor import colored as cl #text customization

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

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


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

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

Things to do:

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

Notes

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

Run Parameters

In [13]:
```
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 [14]:
```
jupyterRun = 'True'
# Loading necessary variables specified earlier in the pipeline from metadatafor dataPrep()
file = open(experiment_path + "metadata.pickle", 'rb')
metadata = pickle.load(file)
# file.close()
# print(metadata)

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

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



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

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

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

Checking for algorithms used in STREAMLINE...
 ['Logistic Regression']
```

load_model(): Load One Trained Model at a Time

In [15]:
```
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 [16]:
```
#test load_model() method
load_model()
```

Out[16]:
```
LogisticRegression(C=0.006606805070193189, dual=True,
                   max_iter=193.8544995971634, random_state=42,
                   solver='liblinear')
```

dataPrep(): Loading target CV Training & Testing Sets

In [17]:
```
def dataPrep(train_file_path,instance_label,class_label, test_file_path):
    # Loads target cv training dataset, separates class from features and removes instance labels

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


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



    return trainX, trainY, testX, testY
```

In [18]:
```
#test data_prep() method
trainX, trainY,testX, testY= dataPrep(train_file_path,instance_label,class_label, test_file_path)
print('\nChecking testX for CV0 values...\n', testX)
```

```
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()

- 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 [19]:
```
def get_explainer(model, algorithms, trainX):

    explainer = None
    trained_model = model

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


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

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

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

    return explainer
```

SHAP: compute_shapValues()

In [20]:
```
def compute_shapValues(model, algorithms, explainer, trainX, trainY, testX, testY):
  # this method will calculate shapley values
  # this includes creating expected_values and shap_values
  # returns shap_values (will be called by shap_summary)


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

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

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

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

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



    return shap_values
```

SHAP: shap_summary()

NOTES

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

FIXES

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

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 [24]:
```
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
    expected_value = explainer.expected_value
    print(expected_value)



   # checks algorithm in given list to execute shap summaries
    if algorithms[0] in ["Naive Bayes"]:
        print('Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
        shap.summary_plot(shap_values, testX, plot_type='violin')
        print('SHAP Bar Plot for Summary Plot for SHAP Values in Class 0 & 1 in Test Set:\n')
        shap.summary_plot(shap_values, testX, plot_type="bar")
        print('SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set: \n')
        shap.plots.beeswarm(shap_values, max_display=5)  #max_display allows user to choose # of features to display



    # force plot isnt working...might something that im doing wrong
    # waterfall plot also doesnt work...i get "AttributeError: 'numpy.ndarray' object has no attribute 'base_values'"
    if algorithms[0] in ["Logistic Regression"]:
        print('\nForce Plot for SHAP Values in Test Set:  \n')
        shap.force_plot(explainer.expected_value, shap_values, testX)
        print('Summary Plot for SHAP Values in Test Set: \n')
        shap.summary_plot(shap_values, testX, plot_type='violin')
        print('SHAP Bar Plot for SHAP Values Test Set: \n')
        shap.summary_plot(shap_values, testX, plot_type="bar")
        print('SHAP Decision Plot for SHAP Values in Test Set: \n')    # ideal for tree-based models but still works f
        shap.decision_plot(expected_value, shap_values)

#         sample_ind = 25
# #         shap.plots.waterfall(shap_values[sample_ind], max_display=14)
#         shap.waterfall_plot(shap_values[2, sample_ind:], max_display=14, show=True)


#     if algorithms[0] in ['Decision Tree']:



    # RF NOT APPLICABLE TO ANY OTHER SHAP PLOT THAN THE ONES LISTED
    # WILL CONSIDER USING MULTIOUTPUT SHAP PLOTS B/C RANDOMFOREST IS MULTIOUTPUT
```

```
    if algorithms[0] in ['Random Forest']:
        print('Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
        shap.summary_plot(shap_values, testX)

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

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

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

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




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


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

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

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

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


        print('\nForce Plot for SHAP Values from Class 0 in Test Set: \n')
        shap.force_plot(expected_value[0], shap_values[0], testX.columns.values,  matplotlib = True, show = False)
#
#         shap.force_plot(expected_value[0], shap_values[0])









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

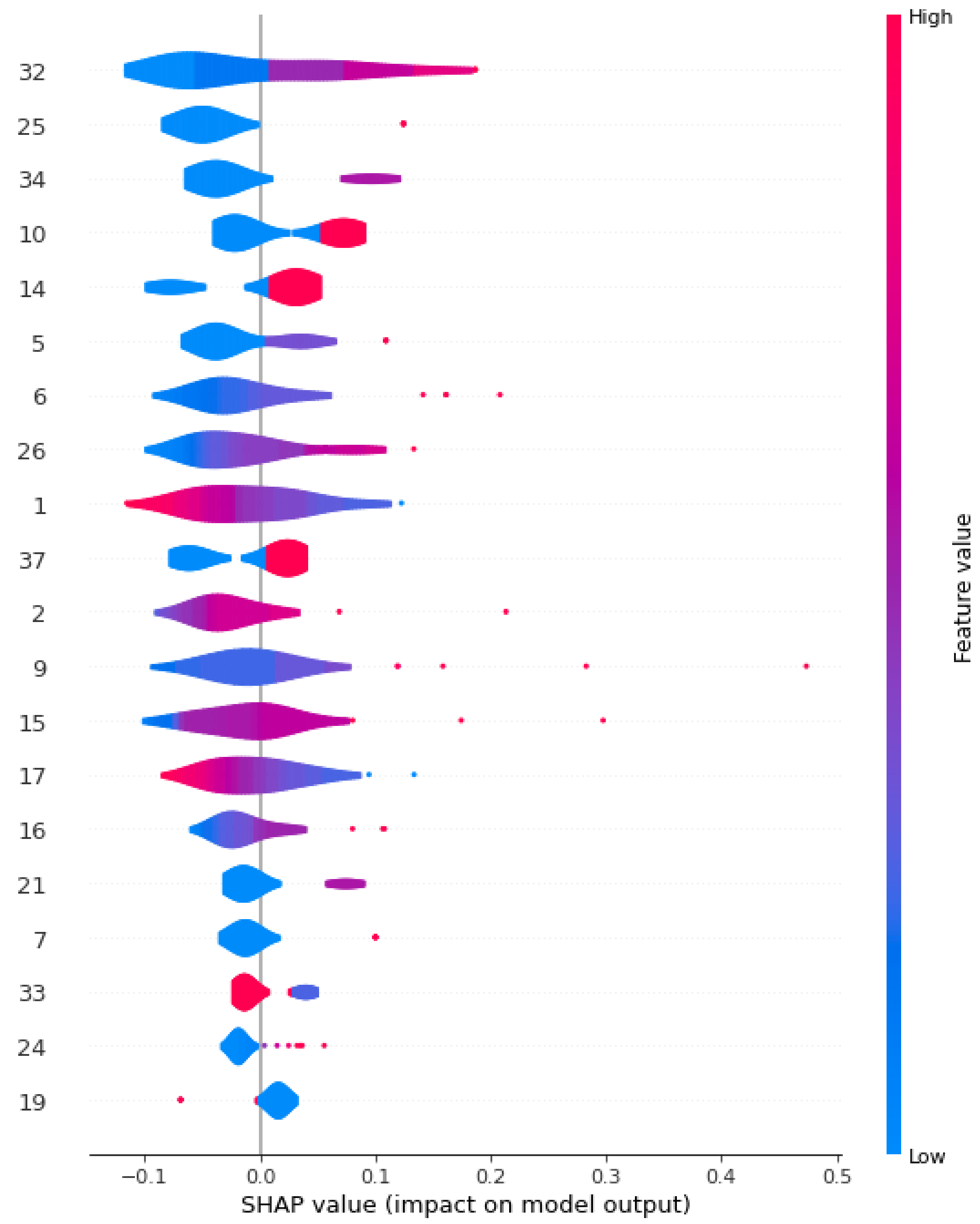
Testing All Functions

In [25]:

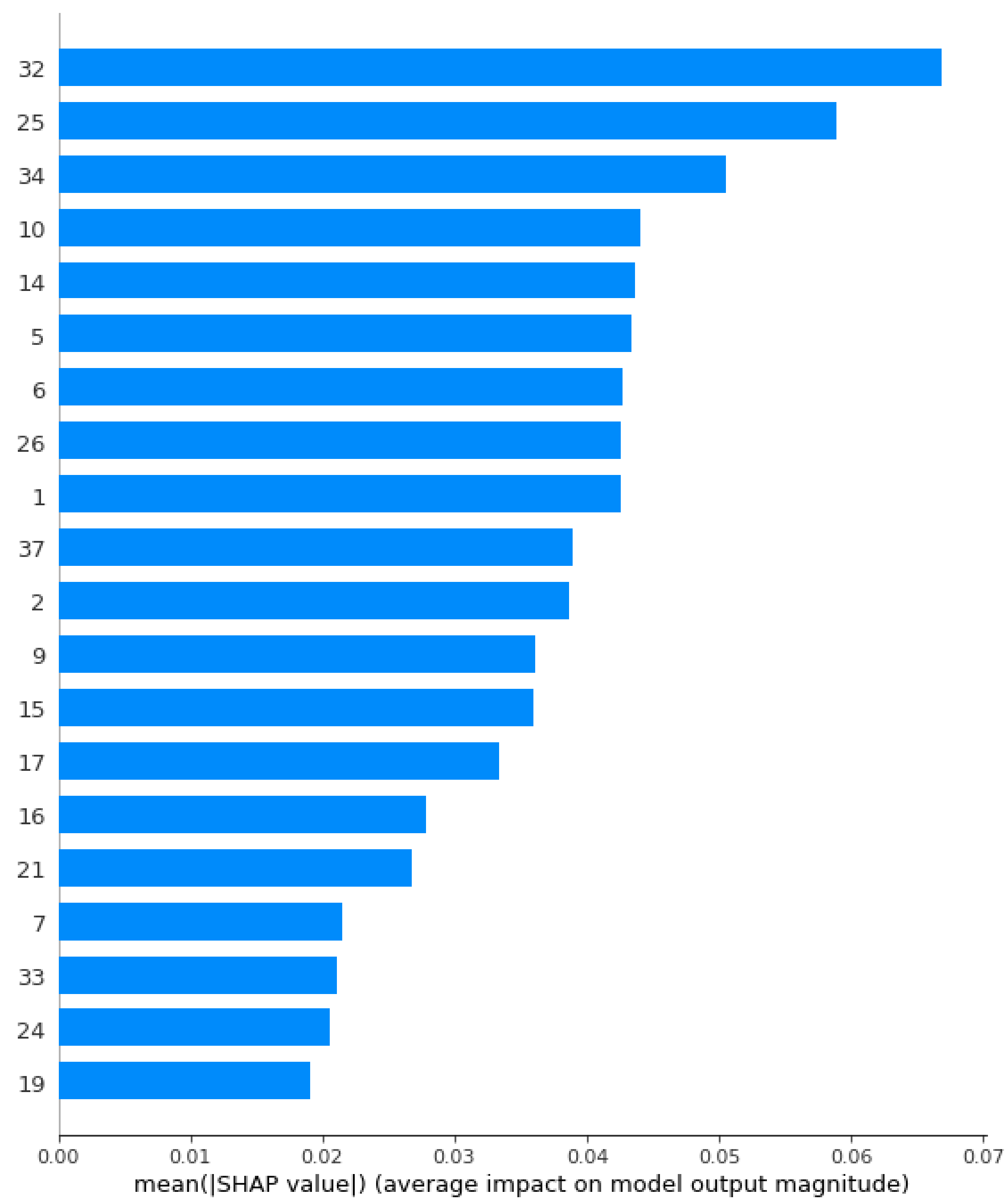
```
# testing all methods
model = load_model()  # load Logistic Regression model and algorithms list
print(model)
print(algorithms) # print to make sure variables are separated




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





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


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

```
LogisticRegression(C=0.006606805070193189, dual=True,
                   max_iter=193.8544995971634, random_state=42,
                   solver='liblinear')
['Logistic Regression']

Predict_proba_ values:
 [[0.54006568 0.45993432]
 [0.38437342 0.61562658]
 [0.4497301  0.5502699 ]
 [0.54018126 0.45981874]
 [0.55249932 0.44750068]
 [0.57777693 0.42222307]
 [0.52097236 0.47902764]
 [0.50791201 0.49208799]
 [0.59704543 0.40295457]
 [0.54508908 0.45491092]
 [0.54522498 0.45477502]
 [0.60863595 0.39136405]
 [0.66883779 0.33116221]
 [0.58060668 0.41939332]
 [0.30765335 0.69234665]
 [0.4395686  0.5604314 ]
 [0.56812058 0.43187942]
 [0.54279401 0.45720599]
 [0.49569352 0.50430648]
 [0.5226984  0.4773016 ]
 [0.59060598 0.40939402]
 [0.53870037 0.46129963]
 [0.54252028 0.45747972]
 [0.57223769 0.42776231]
 [0.60839303 0.39160697]
 [0.519376   0.480624  ]
 [0.52426963 0.47573037]
 [0.57136882 0.42863118]
 [0.50388422 0.49611578]
 [0.62960679 0.37039321]
 [0.52899117 0.47100883]
 [0.65153734 0.34846266]
 [0.50778276 0.49221724]
 [0.54725191 0.45274809]
 [0.61986803 0.38013197]
 [0.62219285 0.37780715]
 [0.60113814 0.39886186]
 [0.46274443 0.53725557]
 [0.44647415 0.55352585]
 [0.49122909 0.50877091]
 [0.42946541 0.57053459]
 [0.53611811 0.46388189]
 [0.55077081 0.44922919]
 [0.47597186 0.52402814]
 [0.65795631 0.34204369]
 [0.40933649 0.59066351]
 [0.4204977  0.5795023 ]
 [0.64469226 0.35530774]
 [0.54974999 0.45025001]
 [0.52523407 0.47476593]
 [0.41451564 0.58548436]
 [0.47032848 0.52967152]
 [0.6321777  0.3678223 ]
 [0.62623157 0.37376843]
 [0.50760677 0.49239323]]

.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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0.
 0. 0. 1. 1. 0. 0. 0.]

Checking if explainer for model exists...
 <shap.explainers._linear.Linear object at 0x7f7c0be3b550>

Checking if shap values for model is returned...

[[-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 if shap plots are returned and consistent...

-0.023696555525940875

Force Plot for SHAP Values in Test Set:
```

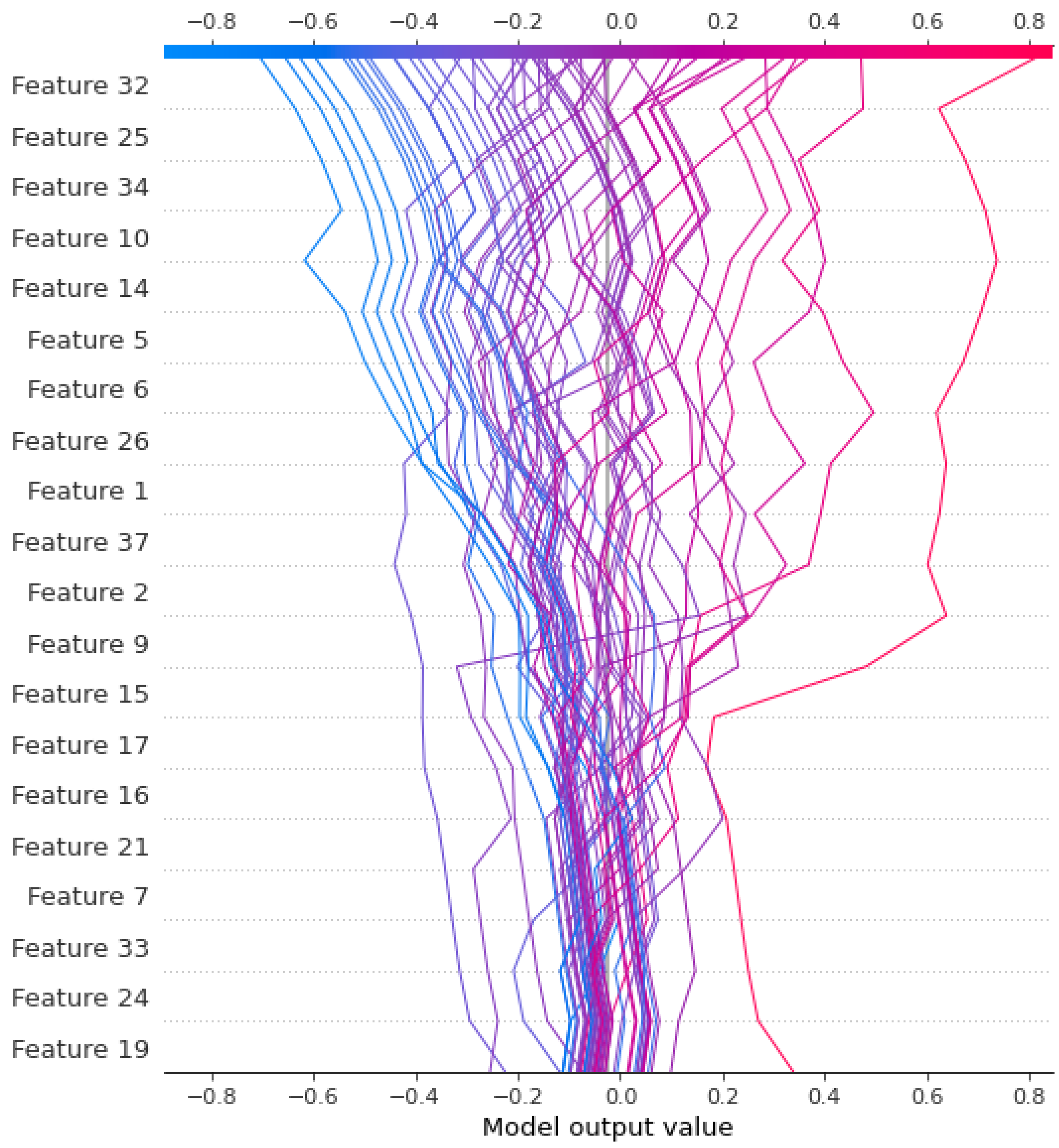
```
Summary Plot for SHAP Values in Test Set:
```




```
SHAP Bar Plot for SHAP Values Test Set:
```




```
SHAP Decision Plot for SHAP Values in Test Set:
```



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

# print(metrics)
```

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

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

In [ ]:

In [ ]: