In [1]:

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

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

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

# Jupyter Notebook Hack: This code ensures that the results of multiple commands within a given cell are all displayec
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
```

- 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"
targetDataName = 'None'

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

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 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 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)
```

```
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...
 ['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'}
```

load_model(): Load One Trained Model at a Time

In [5]:

```
# trained_models = [] # if user wants to check all trained models from all Cvs
# def load_model(abbrev):
#   # this method will load the pickled model that is chosen by user (hardcoded for time being)
#   # should return model object
# #     if algorithms in ['Naive Bayes','Logistics Regression','Decision Tree', 'Random Forest', "Extreme Gradient Boo
#     #FIXME: find a way to load one model with matching abbreviation at a time when called by instance variable in te
#     for count in range(0,cv_partitions):
#         for value in abbrev:
#             model_file = experiment_path + '/hcc-data_example/models/pickledModels/' + str(abbrev[value]) + '_' + st
#             file = open(model_file, 'rb')
#             trained_model = pickle.load(file)
#             print(trained_model)
#             file.close()



#         return trained_model
```

In [6]:

```
#test load_model() method
# load_model(abbrev)
# for value in abbrev:
#     print(abbrev[value])
```

dataPrep(): Loading target CV Training & Testing Sets

In [7]:

```
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 [8]:

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

SHAP: get_explainer()

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

Types of SHAP Explainers

.Explainer()

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

.TreeExplainer()

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

.LinearExplainer()

- Computes SHAP values for a linear model, optionally accounting for inter-feature correlations.
- This computes the SHAP values for a linear model and can account for the correlations among the input features.
- Assuming features are independent leads to interventional SHAP values which for a linear model are coef[i] * (x[i] - X.mean(0)[i]) for the ith feature.

- If instead we accountfor correlations then we prevent any problems arising from colinearity and share credit among correlated features.
- Accounting for correlations can be computationally challenging, but LinearExplainer uses sampling to estimate a transform that can then be applied to explain any prediction of the model.

In [9]:

```
def get_explainer(model, abbrev, trainX):

    explainer = None
    trained_model = model

#     print(model) # check if model is loaded into method
#     print(algorithms)
    if abbrev in ["NB"]:  # 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 abbrev in ["LR"]:
        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 abbrev in ['DT', 'RF', "XGB", "LGB","CGB"]:
        explainer = shap.TreeExplainer(trained_model)



    return explainer
```

SHAP: compute_shapValues()

In [10]:

```
def compute_shapValues(model, abbrev, 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 abbrev in ["NB"]:
        shap_values= explainer(testX)  # permutation object cannot use .expected_value function like LR
        print(shap_values)



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



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






    return shap_values
```

SHAP: shap_summary()

NOTES

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

FIXES

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

UPDATES 7/29/22

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

Plot Types for SHAP v0.41.0

Waterfall

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

Summary (type: violin & bar)

- Summary plots of SHAP values across a whole dataset

Dependence

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

Force

- Visualize cumulative SHAP values with an additive force layout.

Beeswarm

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

In [11]:

```
def shap_summary(abbrev, 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 abbrev in ["NB"]:
        print('Summary Plot for SHAP Values in Class 0 & 1 in Test Set: \n')
        shap.summary_plot(shap_values, testX, plot_type='violin')

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

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


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



    # #     # 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')  #FIXME: does not display actual feature names
        shap.summary_plot(shap_values, testX, plot_type='violin')

        print('SHAP Bar Plot for SHAP Values Test Set: \n') #FIXME: does not display actual feature names
        shap.summary_plot(shap_values, testX, plot_type="bar")

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

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


            # 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, testX, plot_type='bar')

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

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



#     return expected_value
```

In [12]:

```
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)
```

In [ ]:

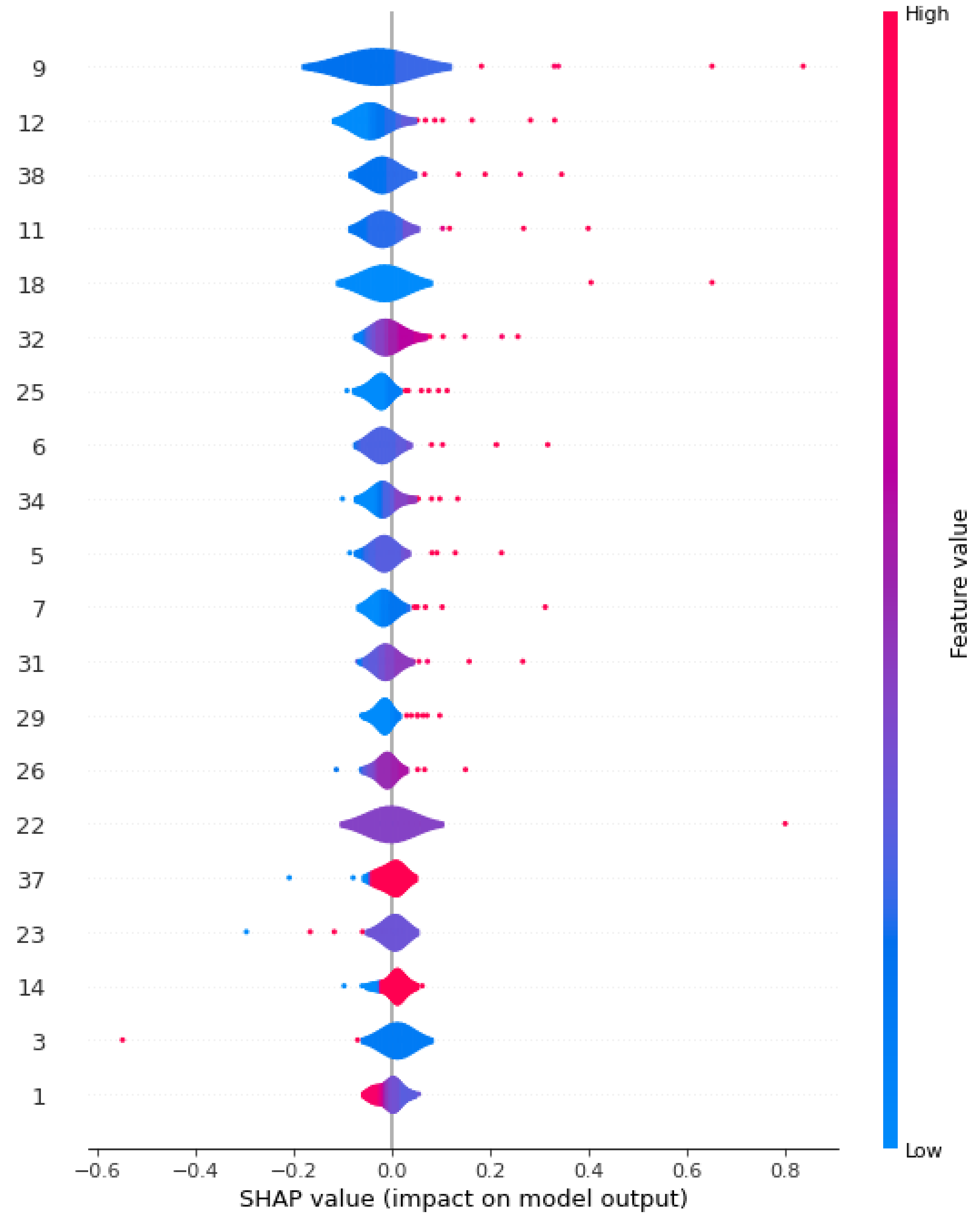
Testing All Functions

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

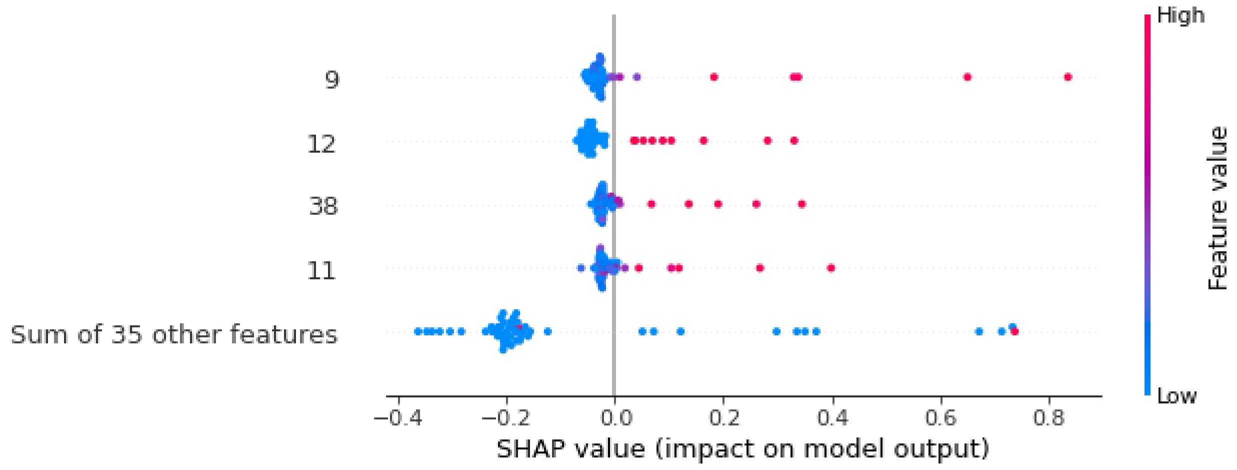
In [17]:

```
# testing all methods
run = False # parameter in run_force_plot(); set to True if user wants to display force plots for trained models

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

    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
            train_path = experiment_path + train_file_path + 'hcc-data_example_CV_' + str(cvCount) + '_Train.csv'
            test_path = experiment_path + train_file_path + 'hcc-data_example_CV_' + str(cvCount) + '_Test.csv'
            trainX, trainY,testX, testY= dataPrep(train_path,instance_label,class_label, test_path)


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

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

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



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


                elif abbrev[algorithm] in ['LR', 'XGB']:
                    print('\nSummary Plot for {} SHAP Values from Class 0 in Test Set: \n'.format(abbrev[algorithm]))
                    shap_summary(abbrev[algorithm], shap_values, explainer, trainX, testX)  # retrieve shap summary pl

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


                else:
                    print('\nForce Plot for {} SHAP Values from Class 0 in Test Set: \n'.format(abbrev[algorithm]))
                    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[algorithm]))
                    shap.force_plot(explainer.expected_value[1], shap_values[1], feature_names=feature_names)
                    break
```

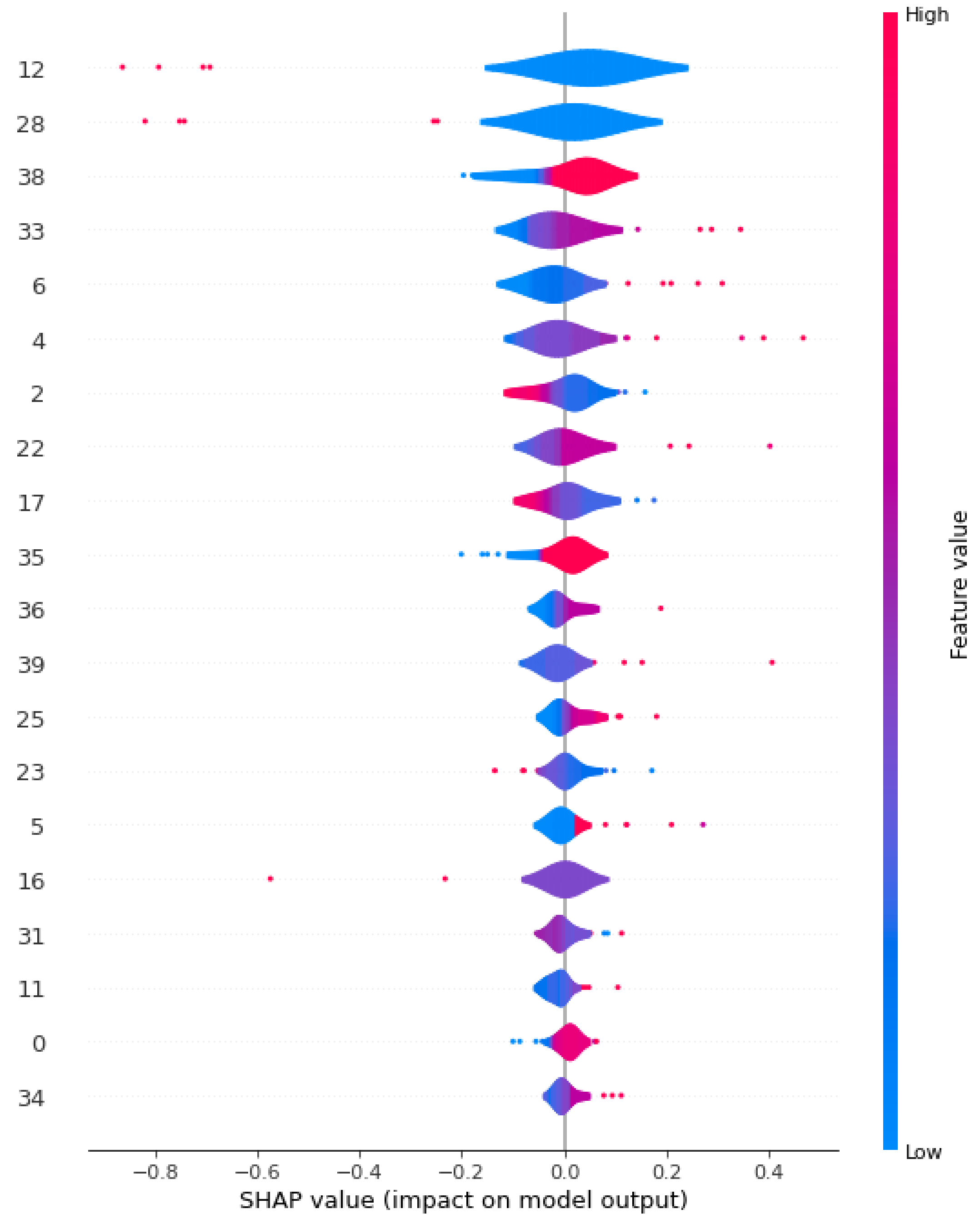
```
----------------------------------------
hcc-data_example
----------------------------------------
NB
NB0 In CV0...

Checking if correct model is loaded...
 GaussianNB()

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

Checking shap values for NB...

.values =
array([[ 8.33333333e-04, -4.41666667e-02, -2.00000000e-02, ...,
        -8.33333333e-04,  1.25000000e-02, -2.66666667e-02],
       [ 1.66666667e-03,  2.25000000e-02,  2.91666667e-02, ...,
         7.50000000e-03,  1.58333333e-02, -3.75000000e-02],
       [ 4.16666667e-03, -5.83333333e-03,  2.00000000e-02, ...,
         1.66666667e-03,  2.58333333e-02, -1.66666667e-03],
       ...,
       [-3.33333333e-03, -8.33333333e-03, -7.50000000e-03, ...,
         0.00000000e+00,  1.66666667e-03, -2.58333333e-02],
       [-2.31296463e-18, -3.33333333e-03, -8.33333333e-03, ...,
         0.00000000e+00, -1.91666667e-02, -2.33333333e-02],
       [ 6.66666667e-03,  3.41666667e-02, -2.08333333e-02, ...,
         0.00000000e+00, -4.50000000e-02,  3.45000000e-01]])

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

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

Checking shap plots for NB...

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



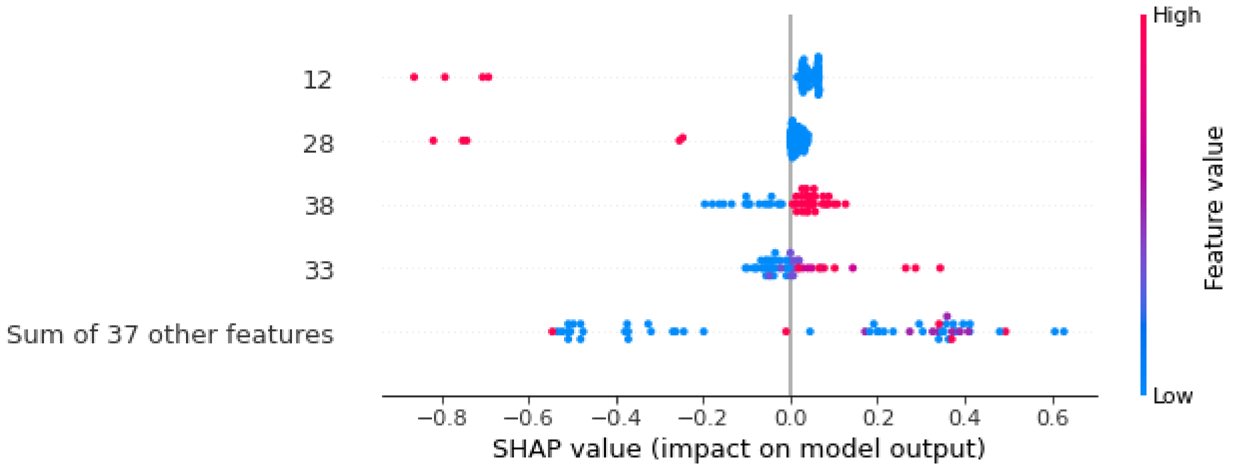
```
SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set:
```



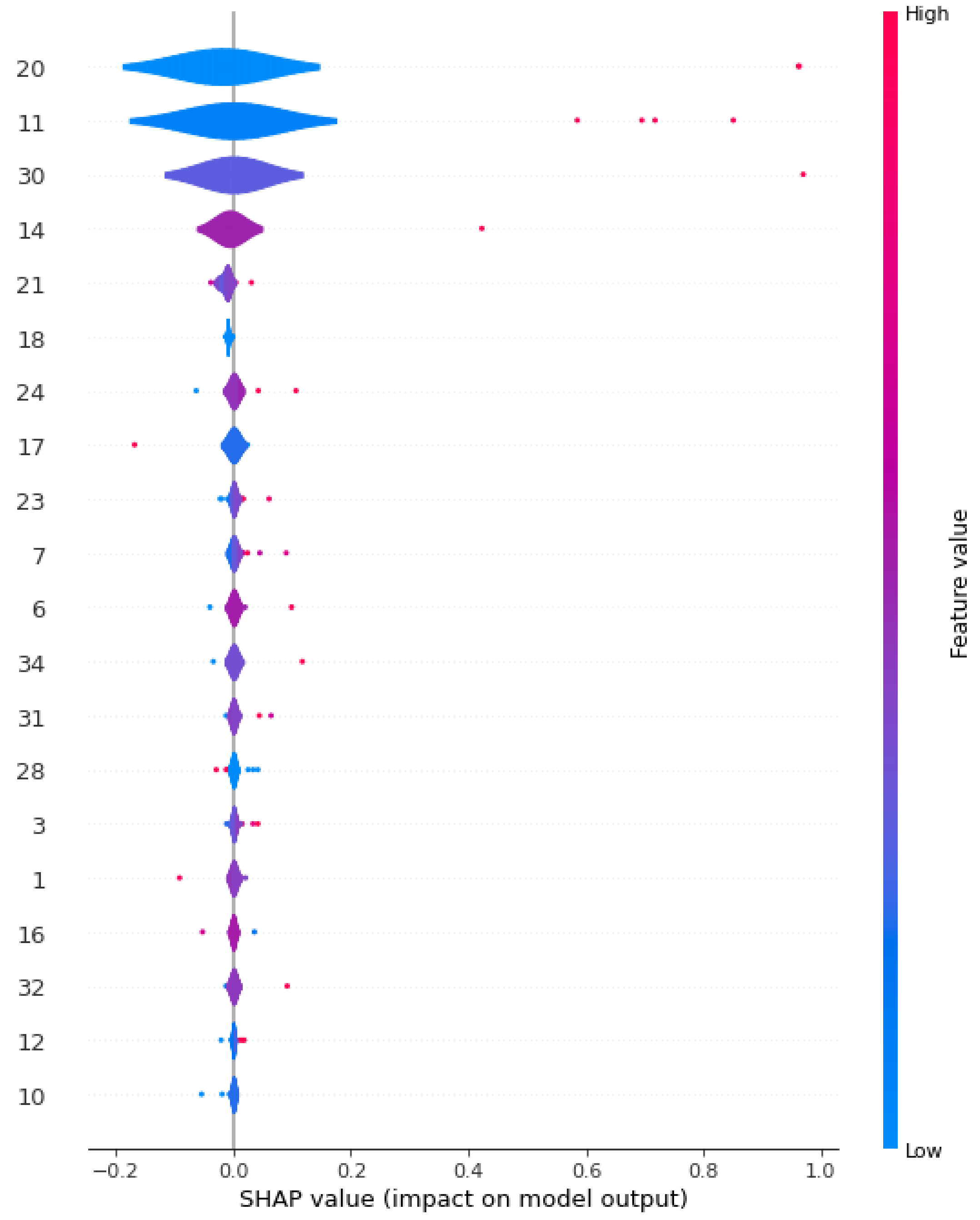
```
NB1 In CV1...

Checking if correct model is loaded...
 GaussianNB()

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

Checking shap values for NB...

.values =
array([[-0.01833333,  0.025     , -0.09666667, ...,  0.12666667,
        -0.00833333, -0.01      ],
       [ 0.03416667, -0.01416667, -0.01      , ..., -0.19666667,
        -0.01916667, -0.01916667],
       [ 0.01583333,  0.0125    ,  0.03      , ..., -0.05833333,
        -0.015     , -0.00916667],
       ...,
       [-0.01416667, -0.01333333,  0.015     , ..., -0.09833333,
        -0.03083333, -0.01416667],
       [ 0.        , -0.005     ,  0.00666667, ...,  0.005     ,
        -0.00083333,  0.        ],
       [ 0.03916667,  0.01666667, -0.03583333, ...,  0.1       ,
        -0.005     , -0.01166667]])

.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 NB...

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



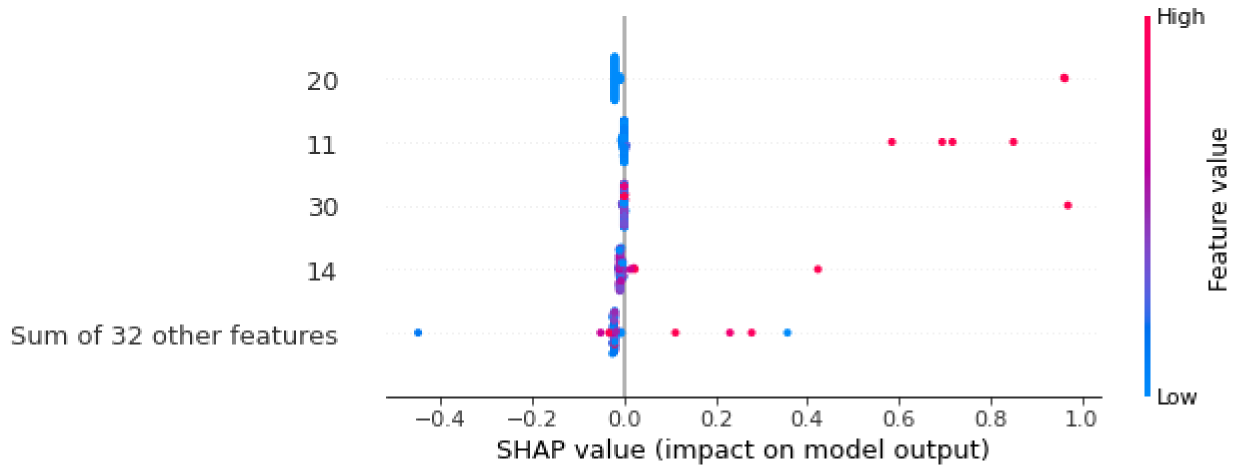
```
SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set:
```



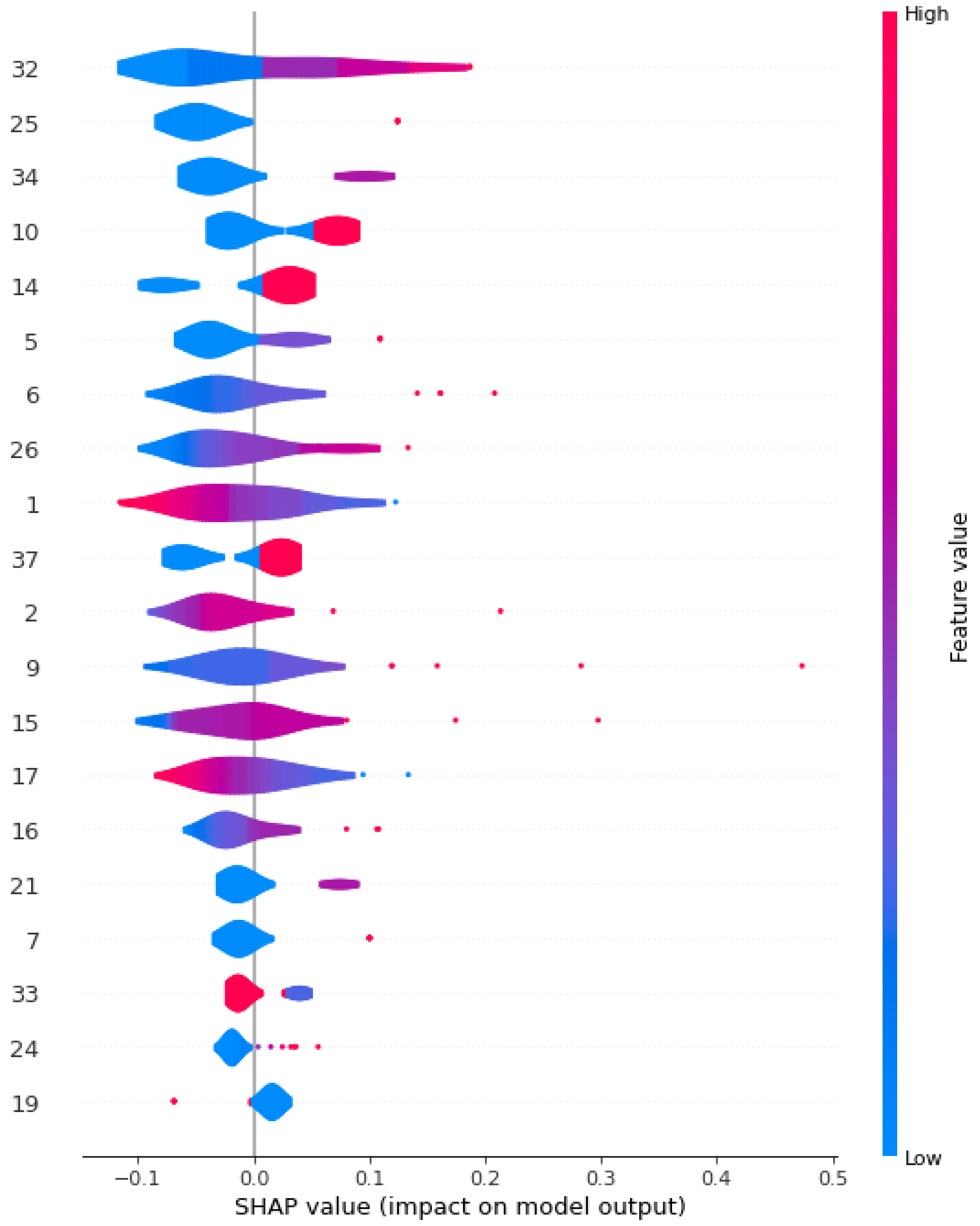
```
NB2 In CV2...

Checking if correct model is loaded...
 GaussianNB()

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

Checking shap values for NB...

.values =
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.00166667,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.00166667,  0.01416667, -0.01      , ..., -0.00333333,
        -0.005     ,  0.00416667],
       ...,
       [ 0.        ,  0.01083333,  0.        , ..., -0.00416667,
        -0.00916667,  0.00416667],
       [-0.00666667, -0.09166667,  0.01      , ..., -0.0225    ,
        -0.03416667,  0.01      ],
       [ 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 NB...

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



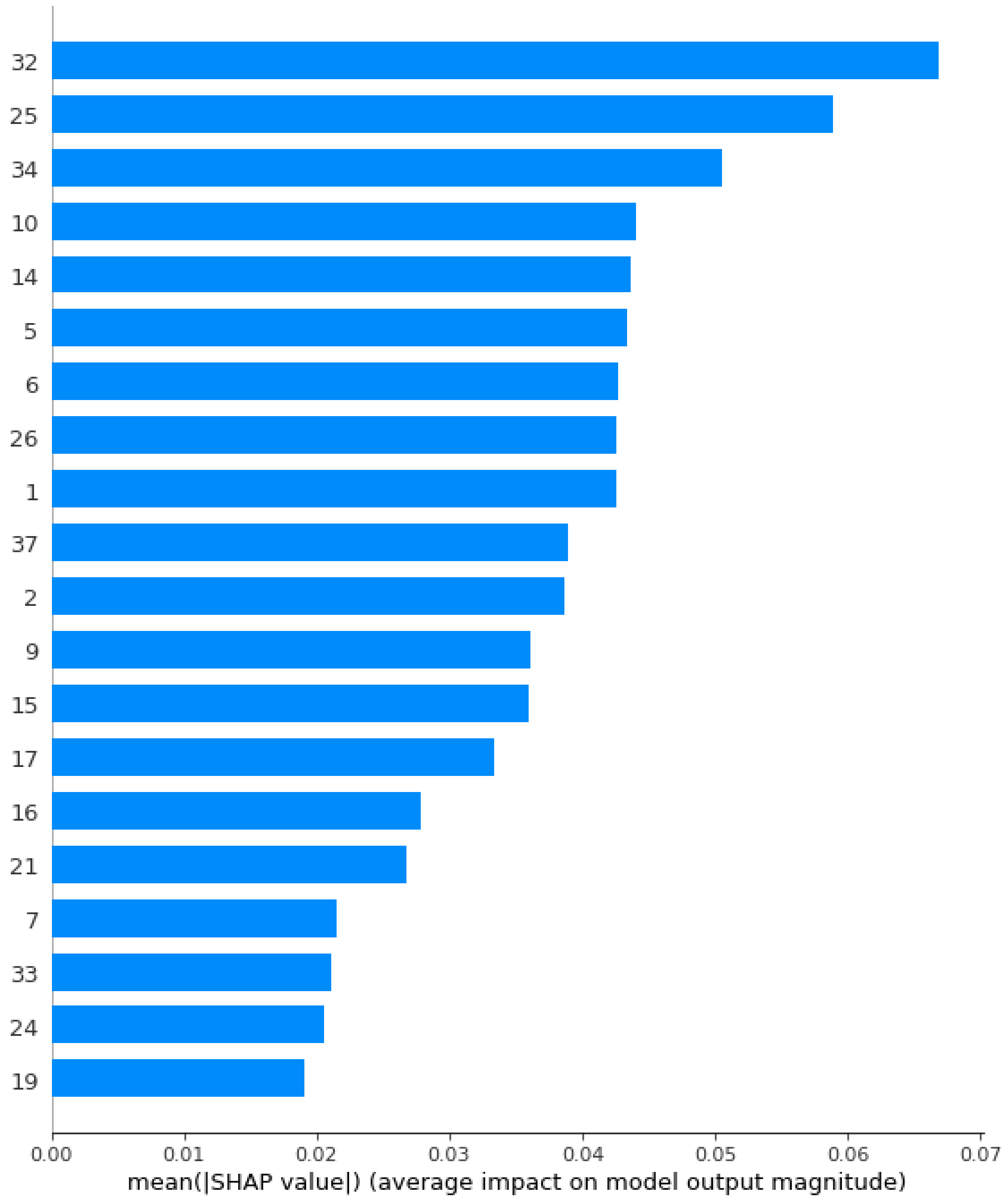
```
SHAP Beeswarm Plot for Top 5 SHAP Values in Class 0 & 1 in Test Set:
```



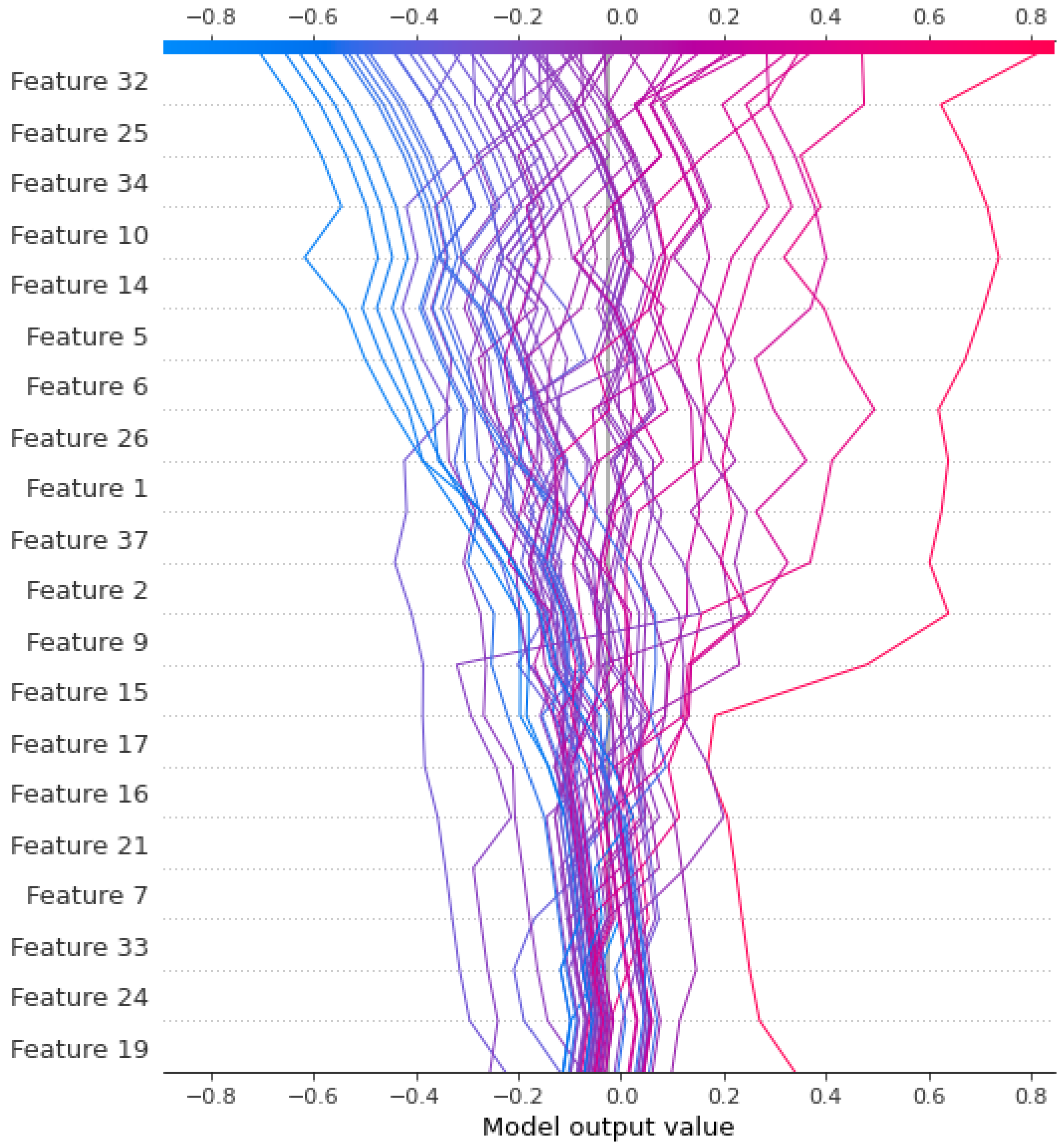
```
LR
LR0 In CV0...

Checking if correct model is loaded...
 LogisticRegression(C=0.006606805070193189, dual=True,
                   max_iter=193.8544995971634, random_state=42,
                   solver='liblinear')

Checking explainer for LR0...
<shap.explainers._linear.Linear object at 0x7fca0b1fcee0>

Checking shap values for LR...

[[-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 LR...

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



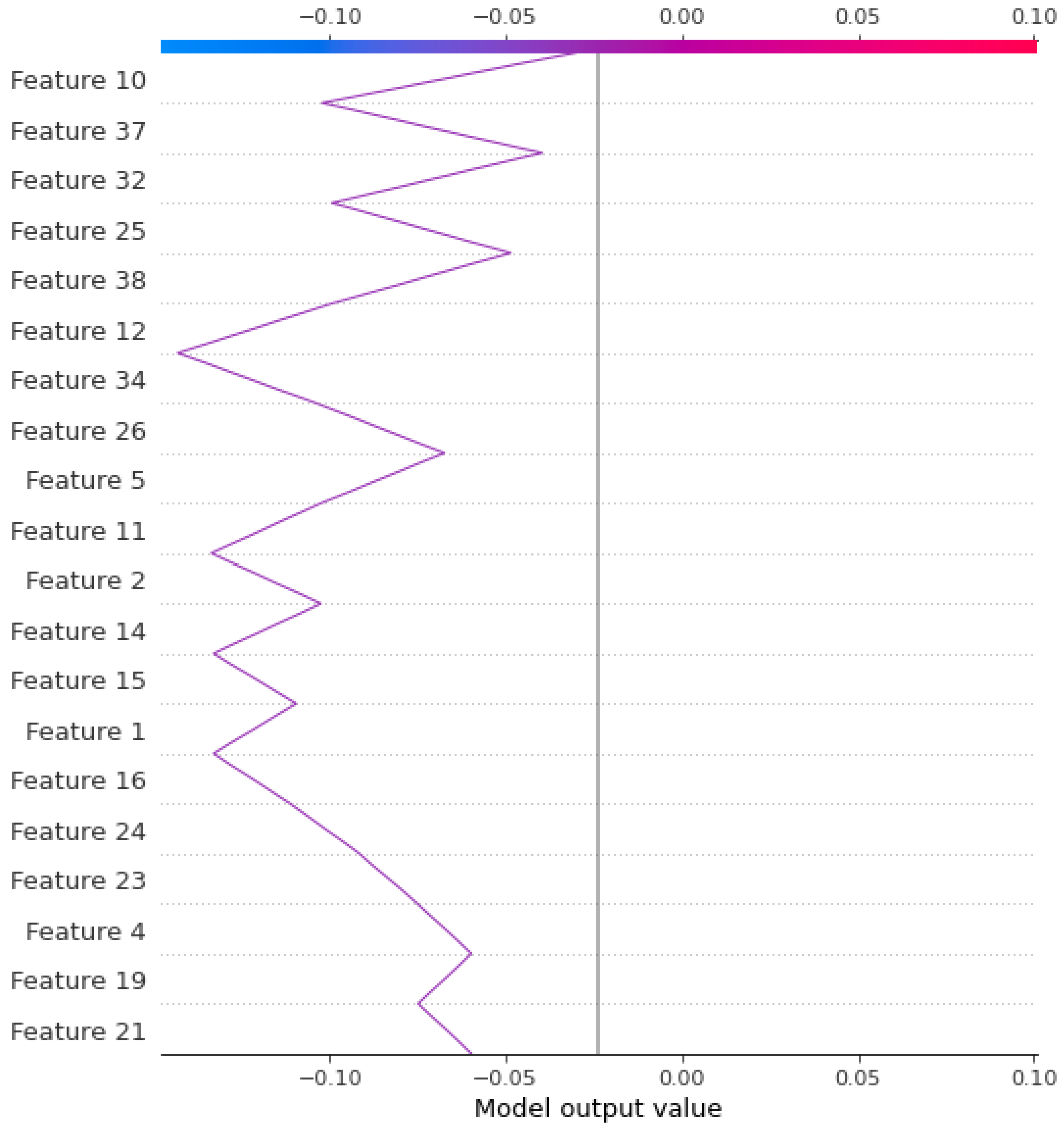
```
SHAP Bar Plot for SHAP Values Test Set:
```



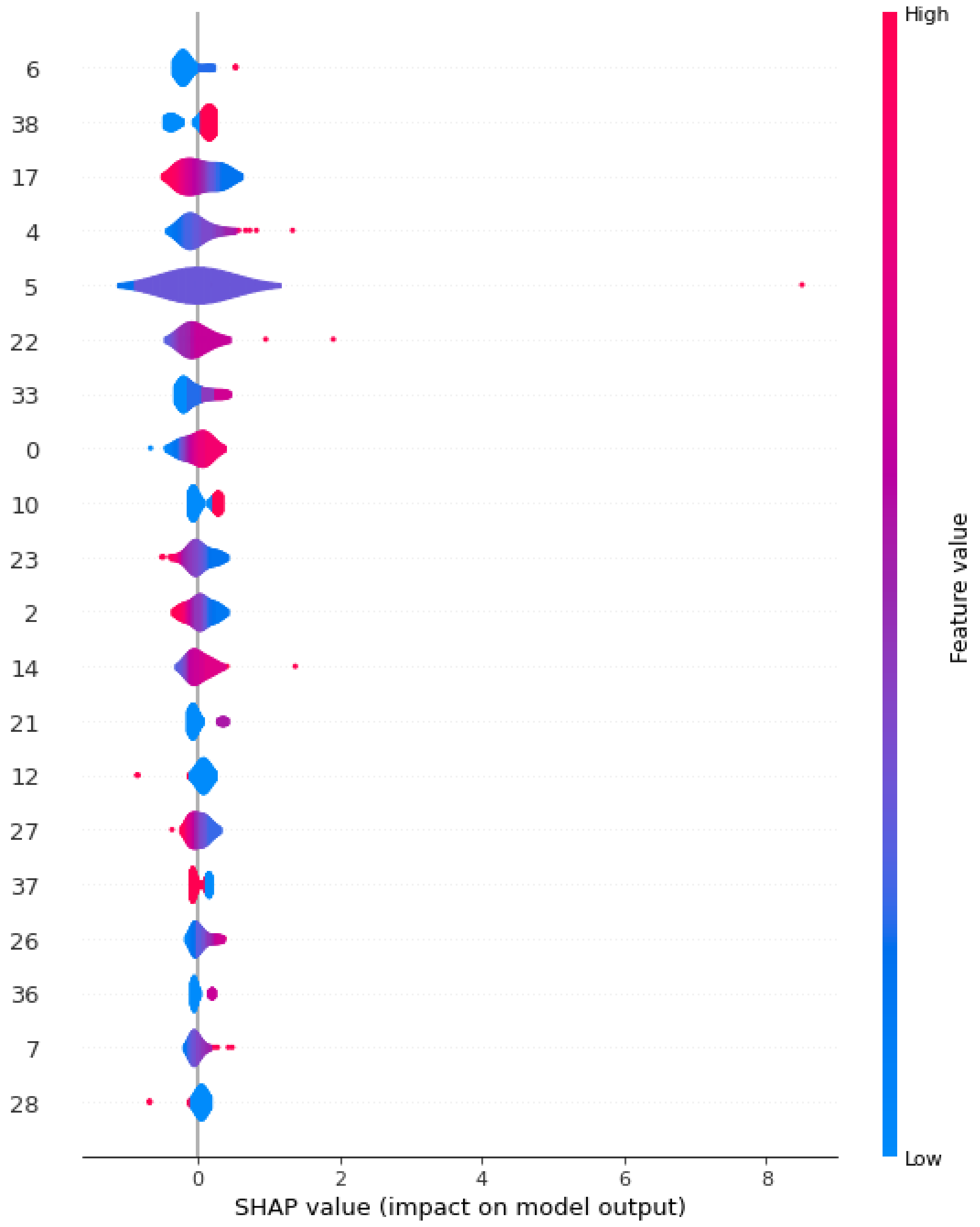
```
SHAP Decision Plot for SHAP Values in Test Set:
```



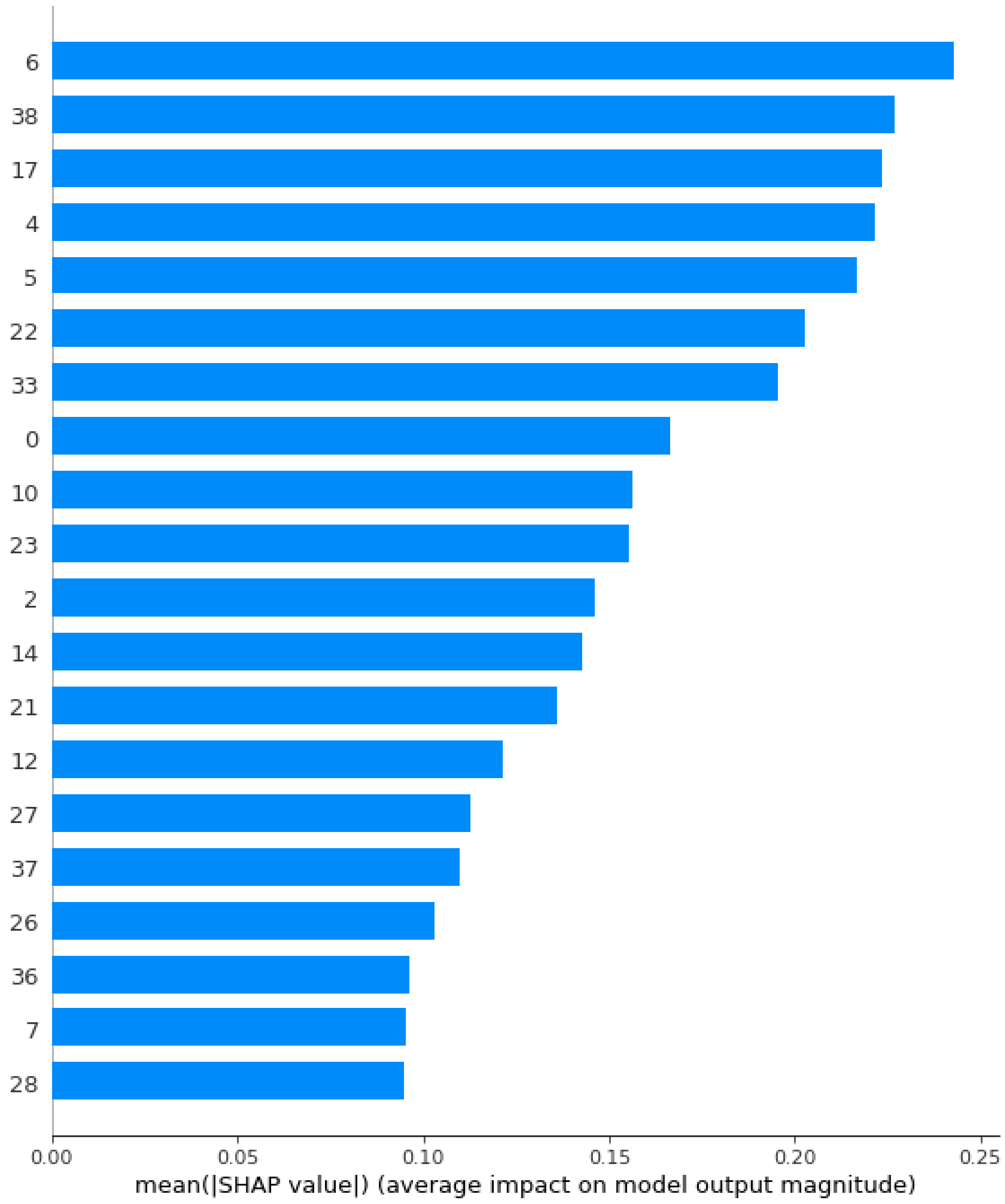
```
SHAP Decision Plot for Single-Prediction in Test Set:
```



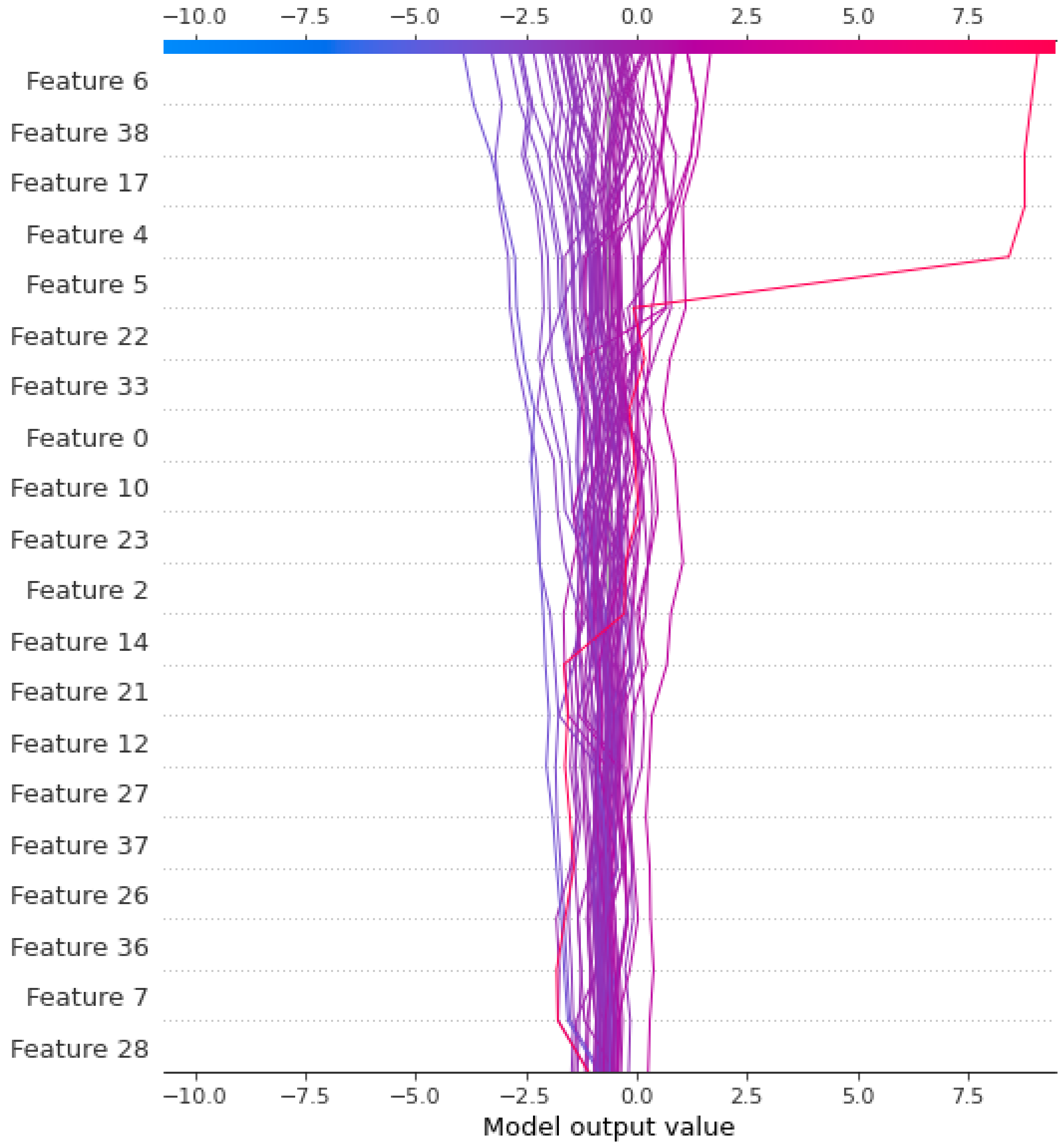
```
LR1 In CV1...

Checking if correct model is loaded...
 LogisticRegression(C=0.06359900885943309, max_iter=48.076782938152924,
                   random_state=42, solver='sag')

Checking explainer for LR1...
<shap.explainers._linear.Linear object at 0x7fca0bb92160>

Checking shap values for LR...

[[-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 LR...

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



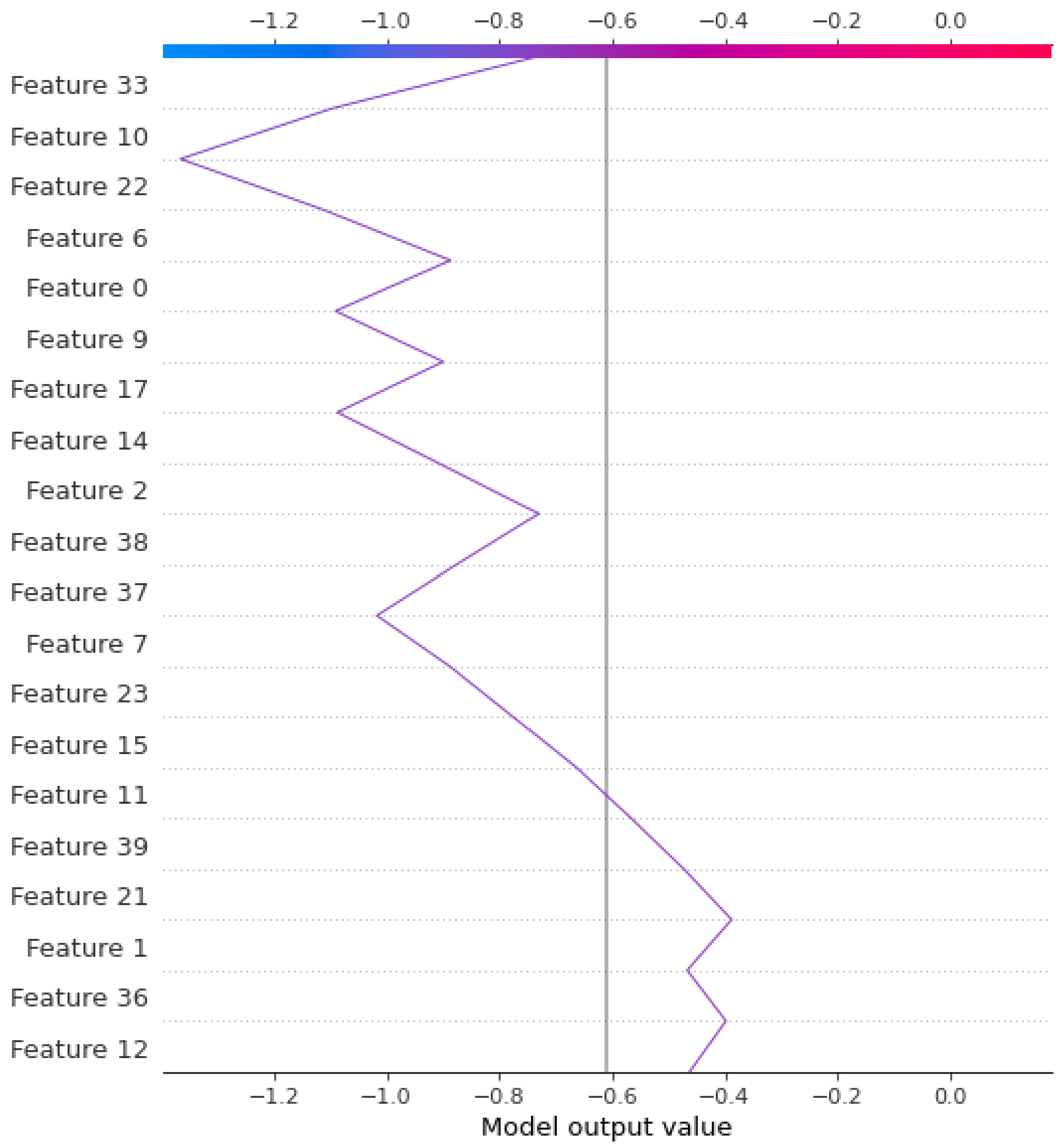
```
SHAP Bar Plot for SHAP Values Test Set:
```



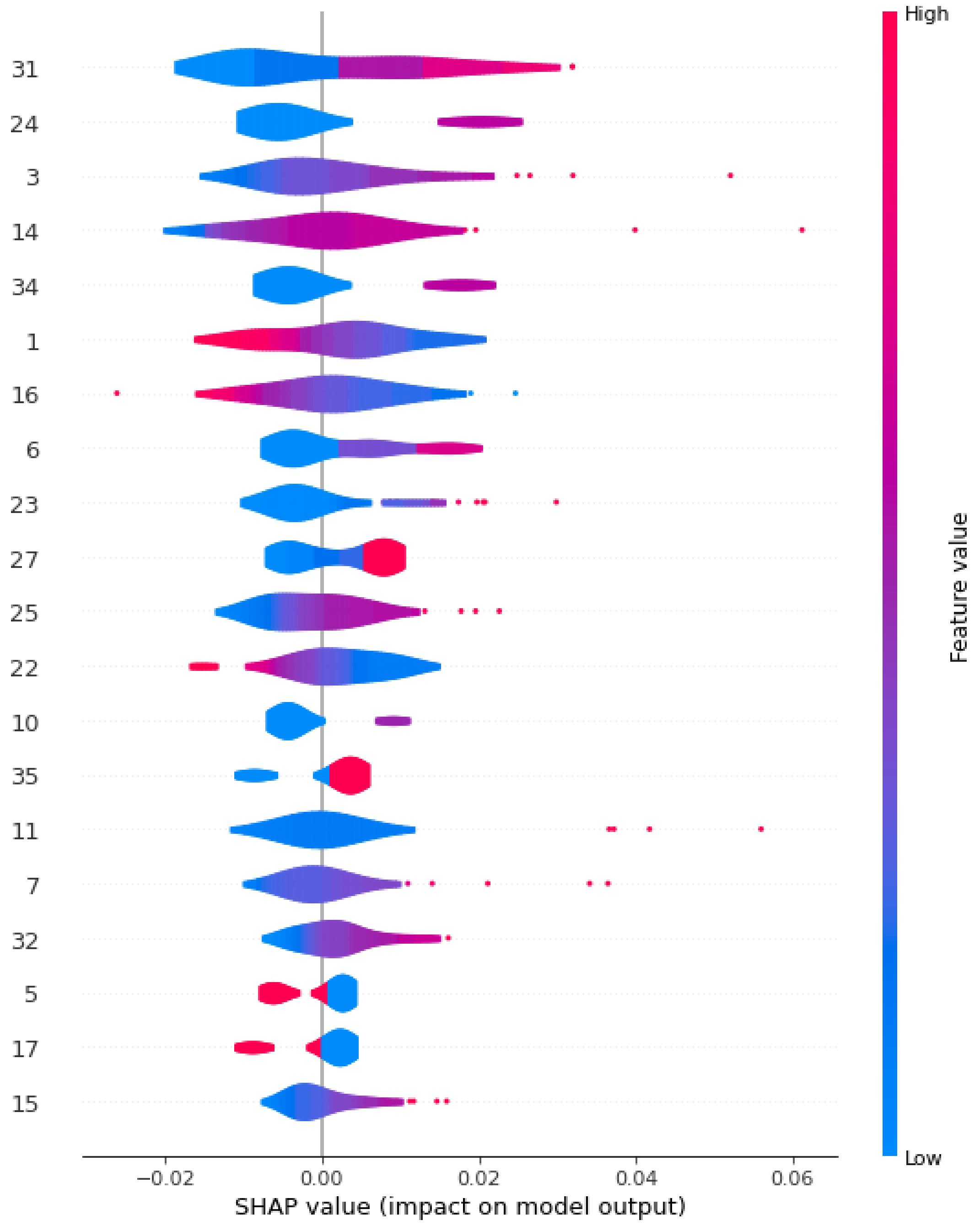
```
SHAP Decision Plot for SHAP Values in Test Set:
```



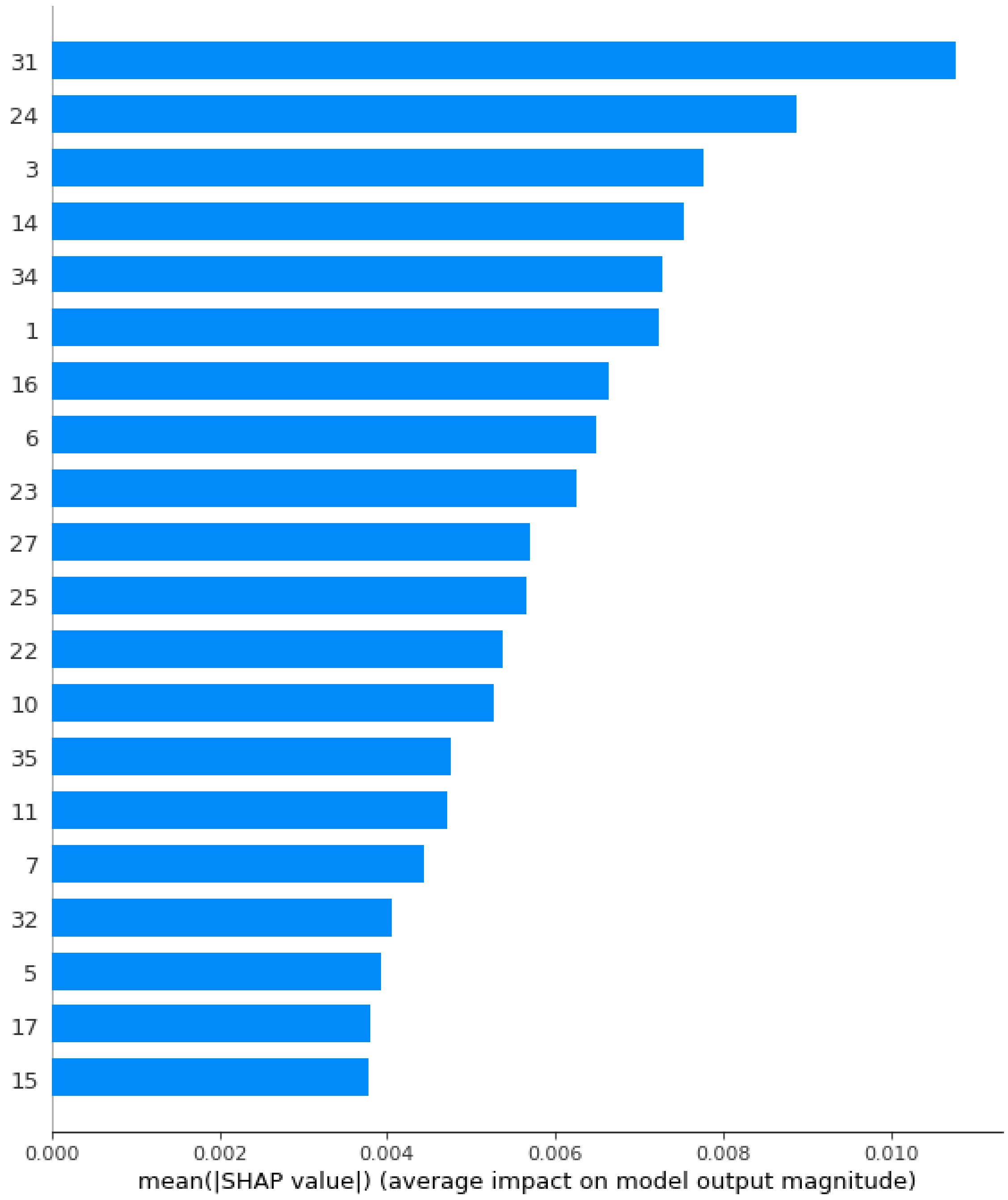
```
SHAP Decision Plot for Single-Prediction in Test Set:
```



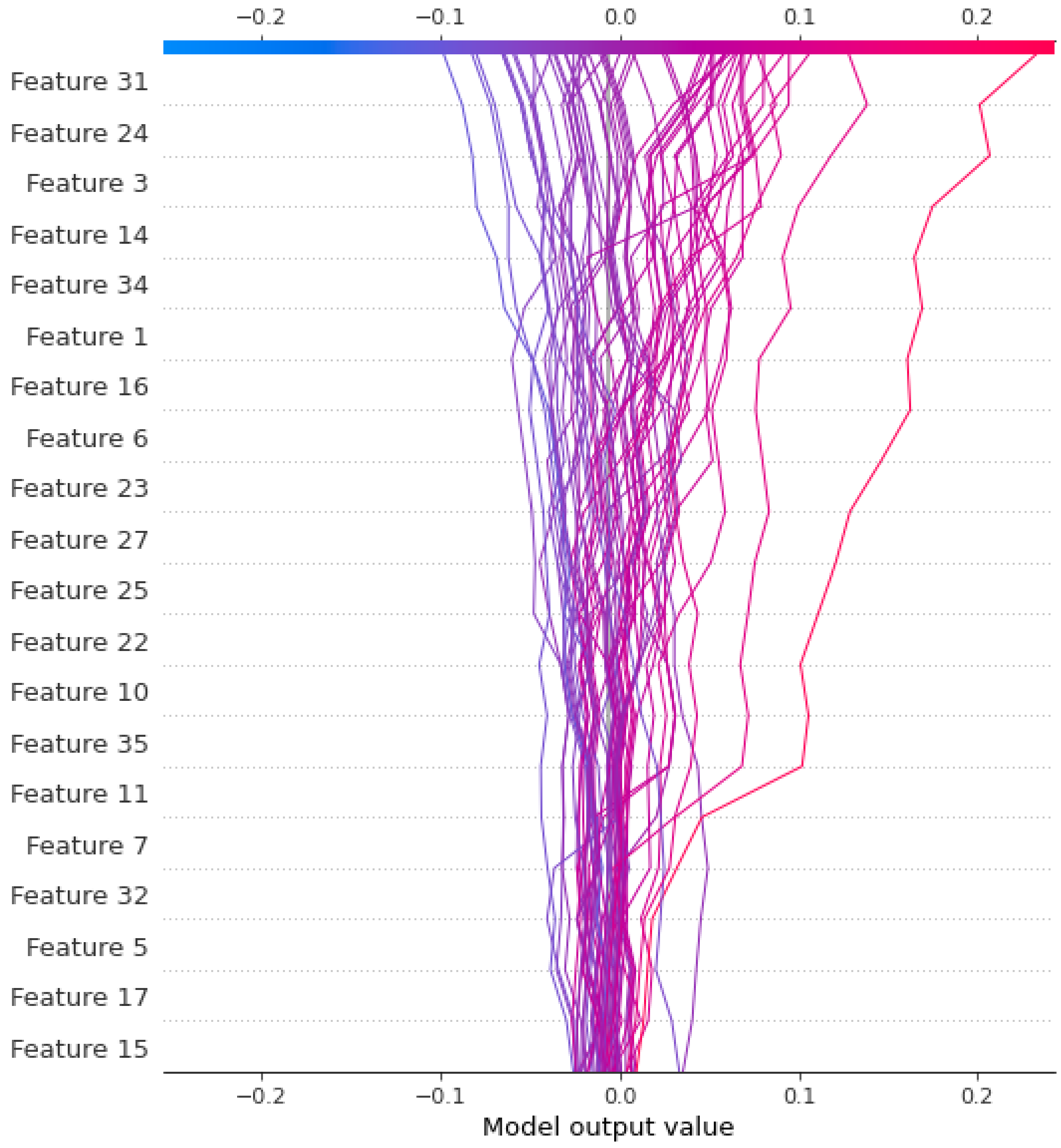
```
LR2 In CV2...

Checking if correct model is loaded...
 LogisticRegression(C=0.0006580360277501316, class_weight='balanced', dual=True,
                   max_iter=112.07606211860569, random_state=42,
                   solver='liblinear')

Checking explainer for LR2...
<shap.explainers._linear.Linear object at 0x7fca0b83b2e0>

Checking shap values for LR...

[[-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 LR...

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



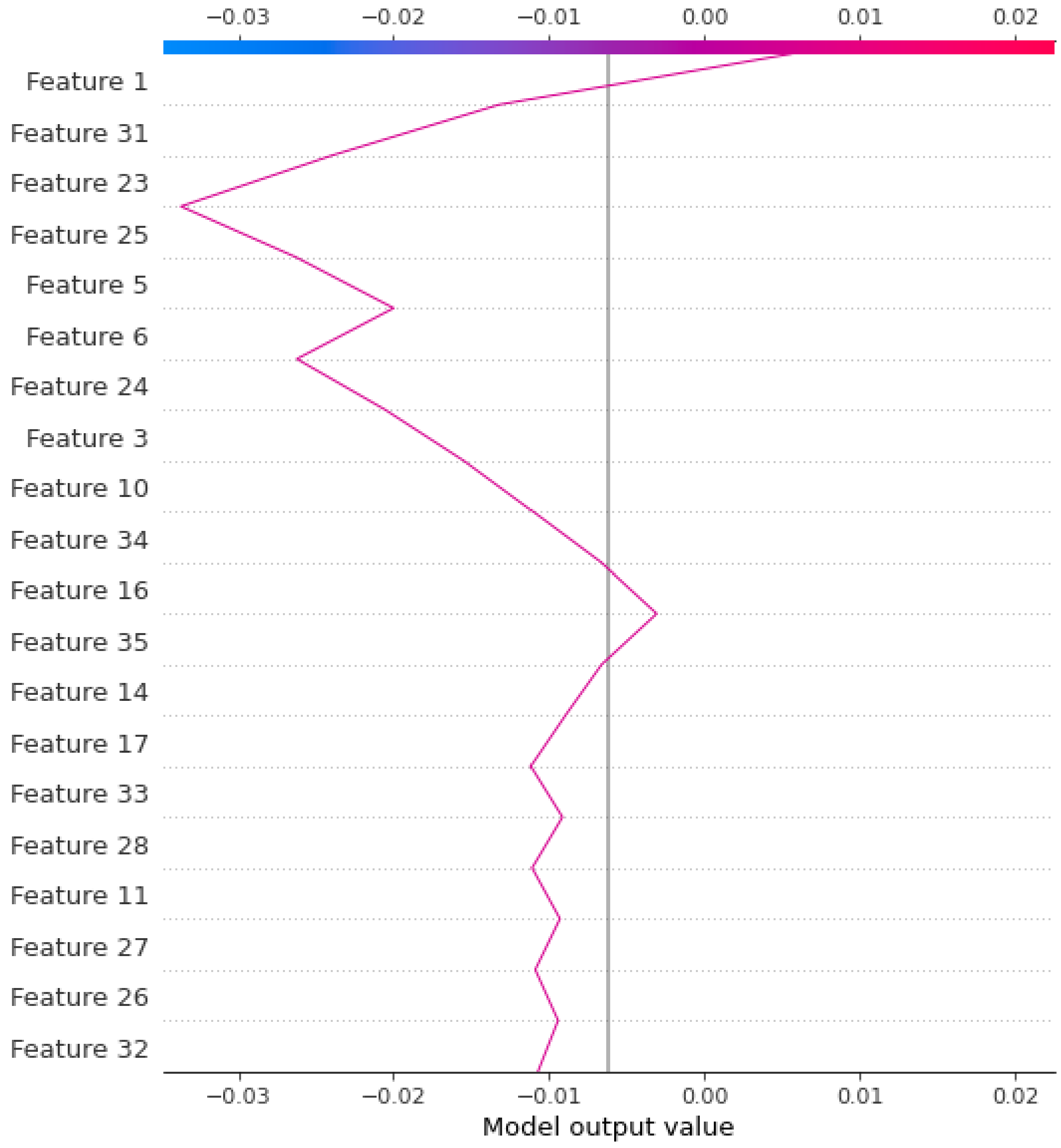
```
SHAP Bar Plot for SHAP Values Test Set:
```



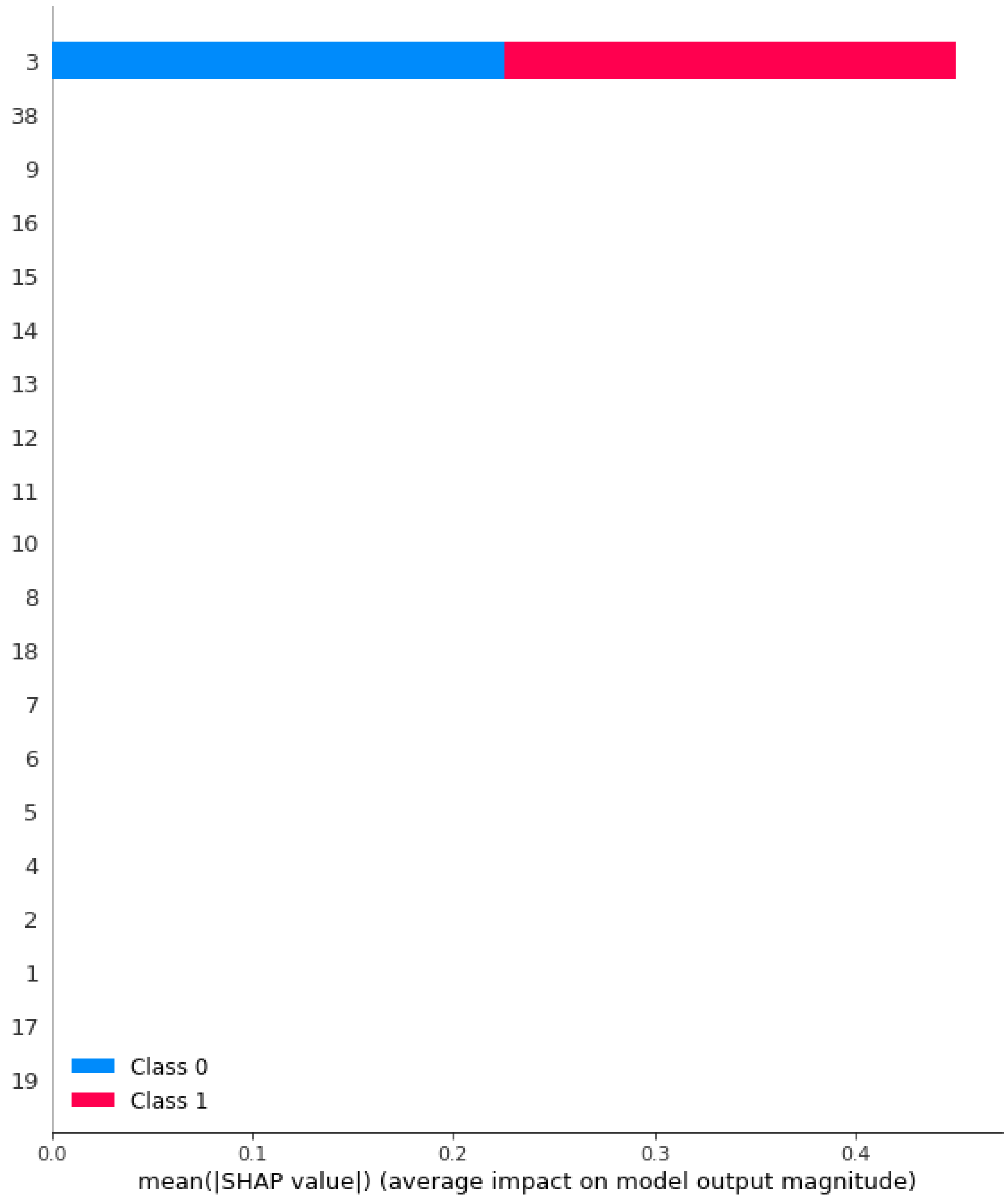
```
SHAP Decision Plot for SHAP Values in Test Set:
```



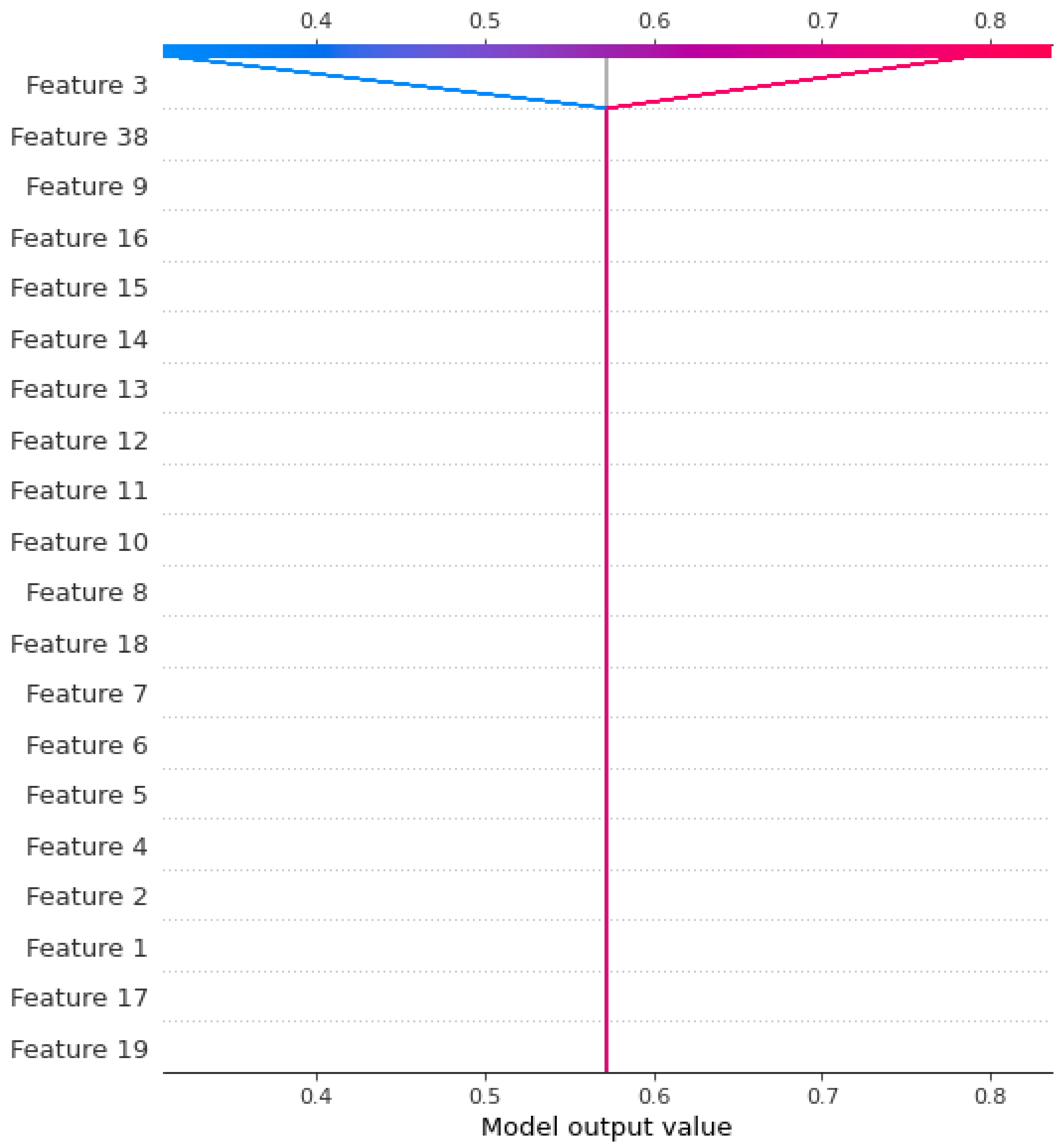
```
SHAP Decision Plot for Single-Prediction in Test Set:
```



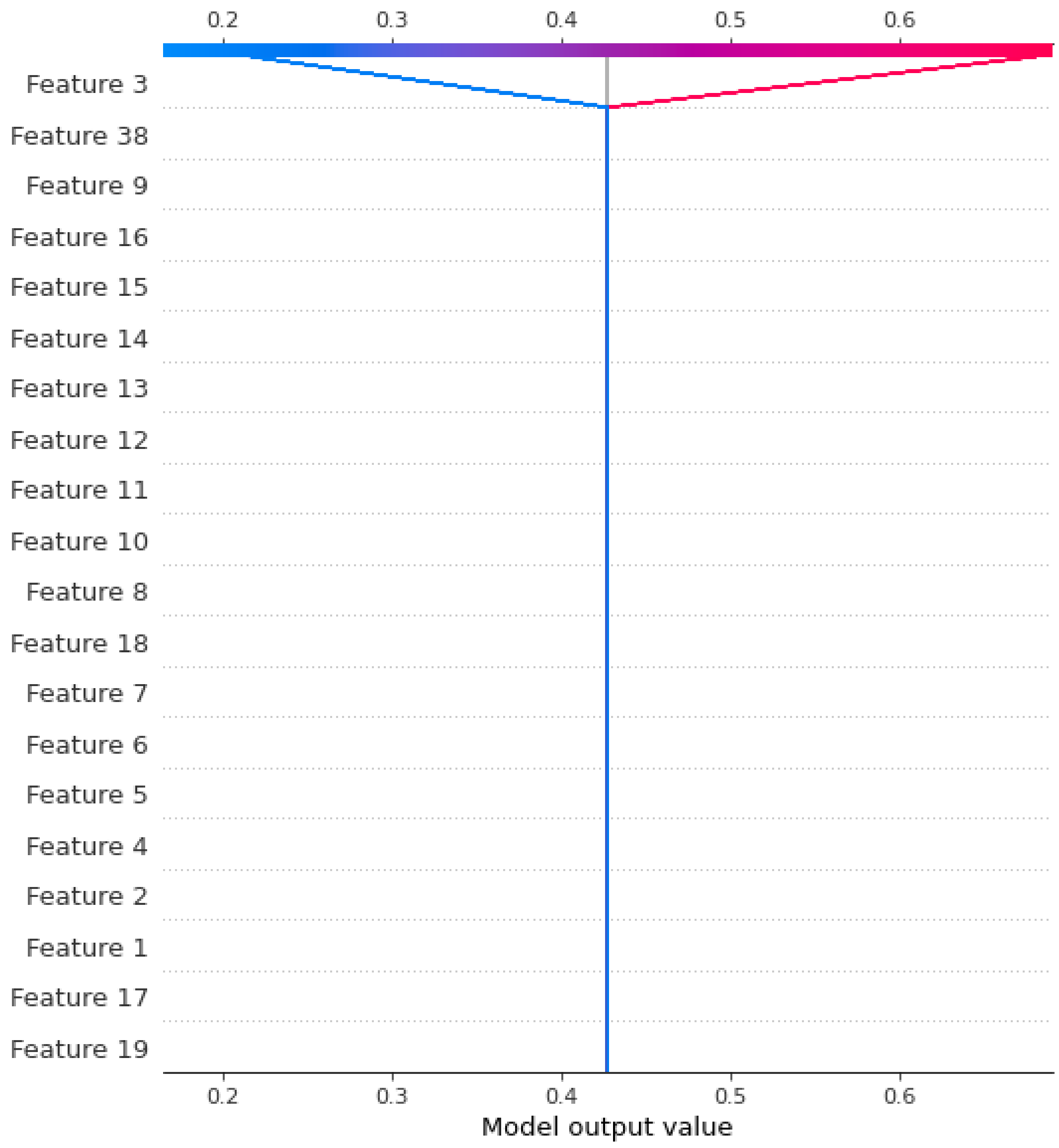
```
DT
DT0 In CV0...

Checking if correct model is loaded...
 DecisionTreeClassifier(max_depth=17, min_samples_leaf=35, min_samples_split=45,
                       random_state=42)

Checking explainer for DT0...
<shap.explainers._tree.Tree object at 0x7fca1c73aac0>

Checking shap values for DT...

[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 DT...

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



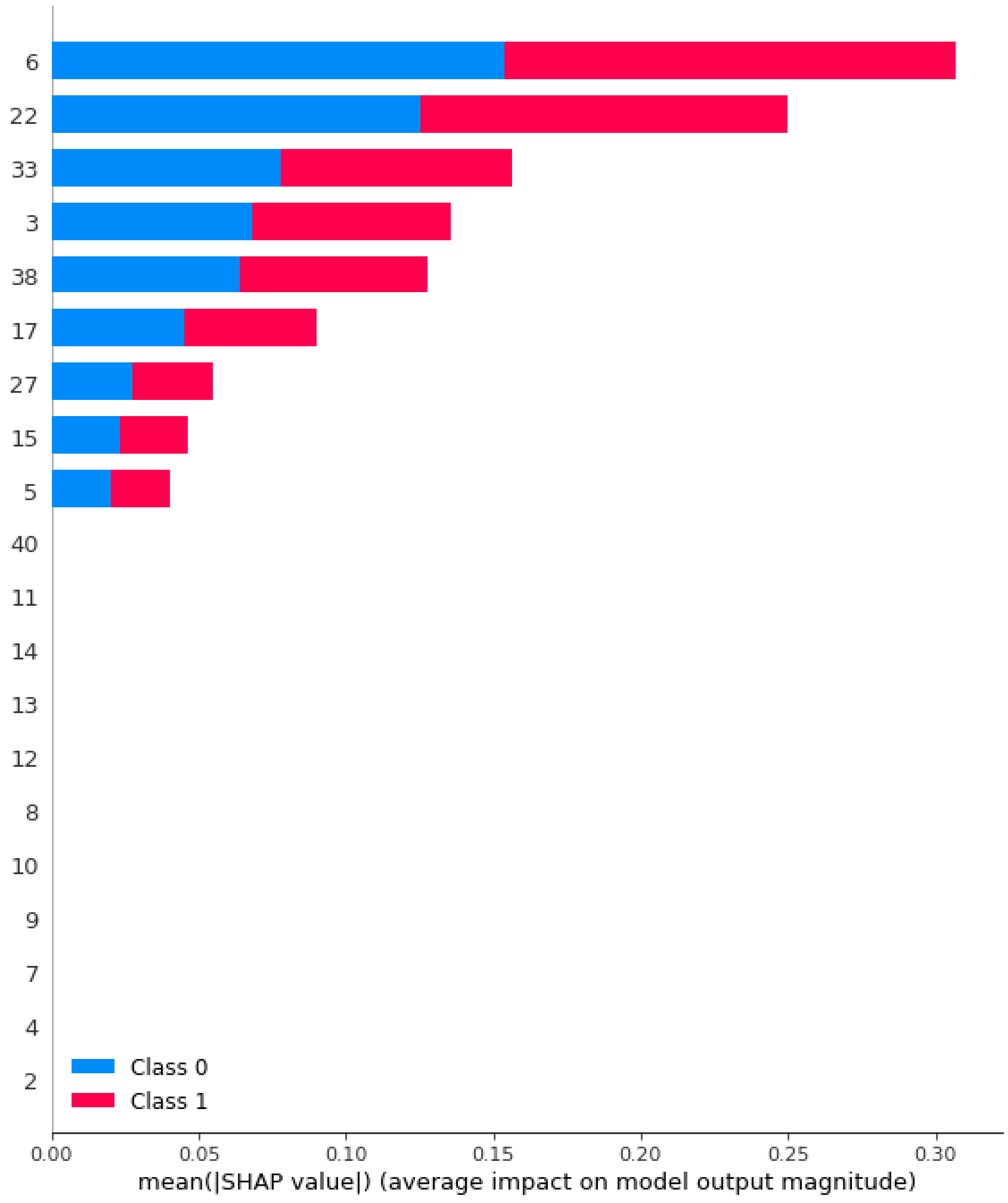
```
Decision Plot for SHAP Values from Class 0 in Test Set:
```



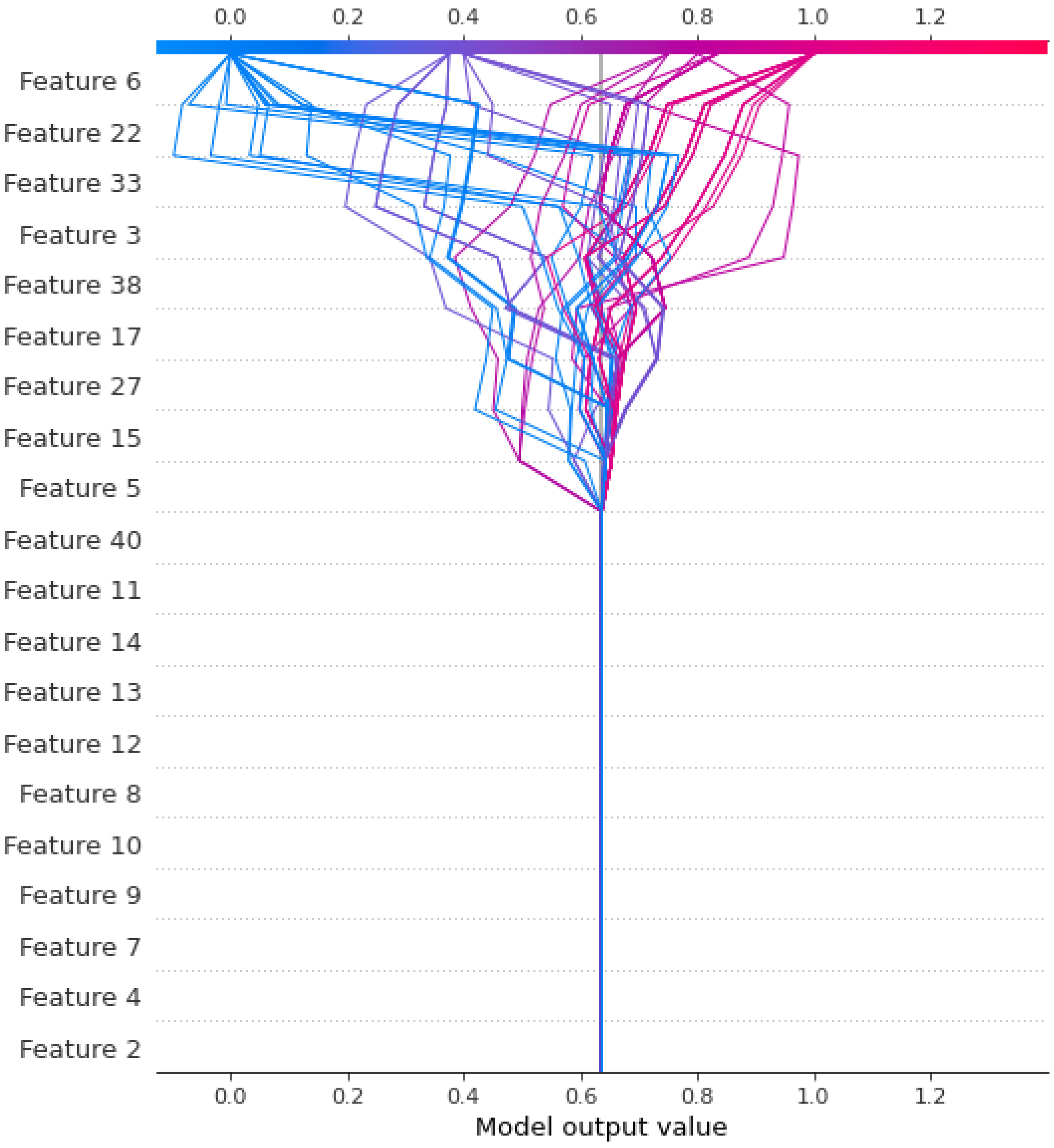
```
Decision Plot for SHAP Values from Class 1 in Test Set:
```



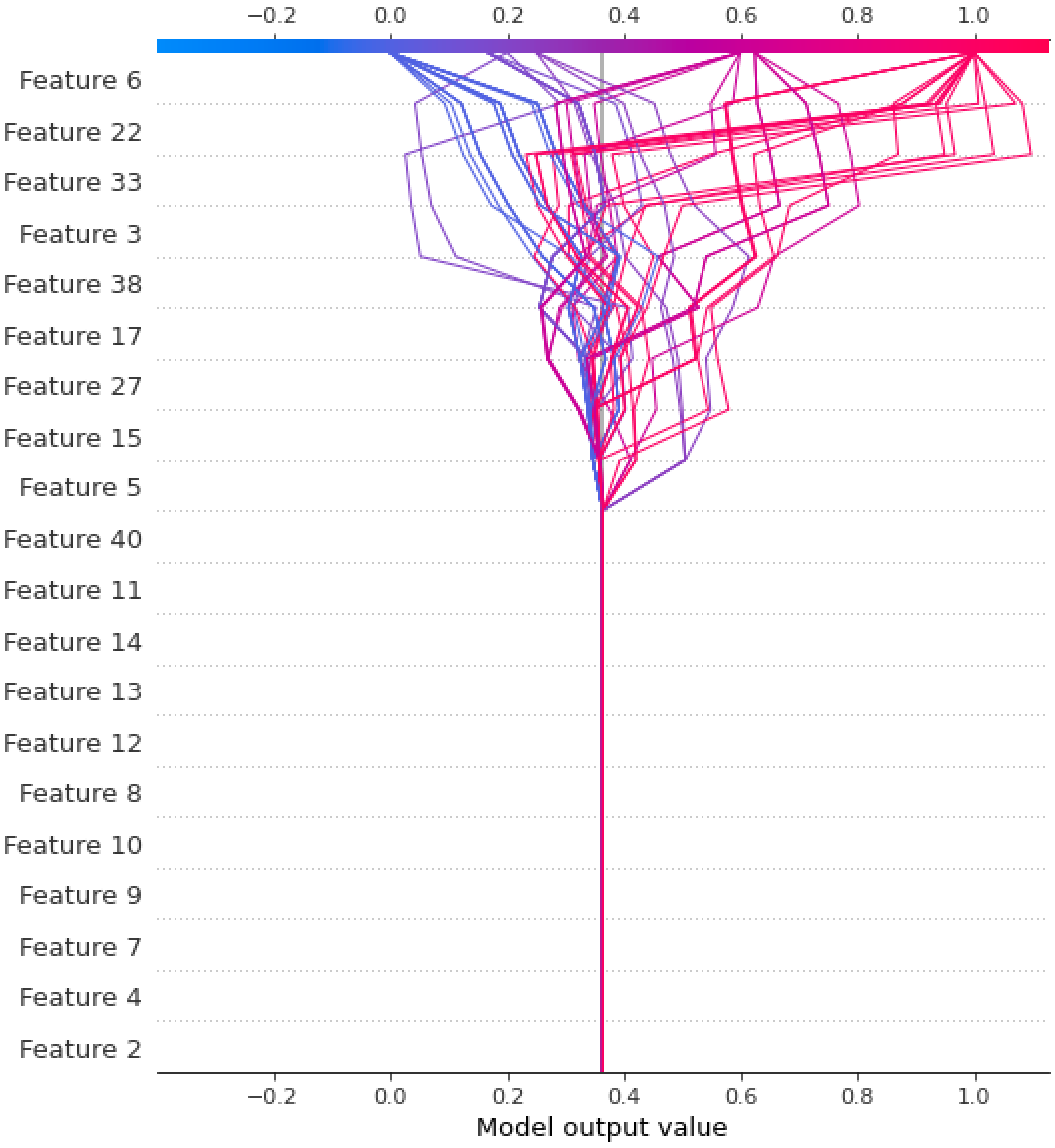
```
DT1 In CV1...

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')

Checking explainer for DT1...
<shap.explainers._tree.Tree object at 0x7fca1dd86d60>

Checking shap values for DT...

[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 DT...

Expected value for DT: [0.63636364 0.36363636]
Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set:
```



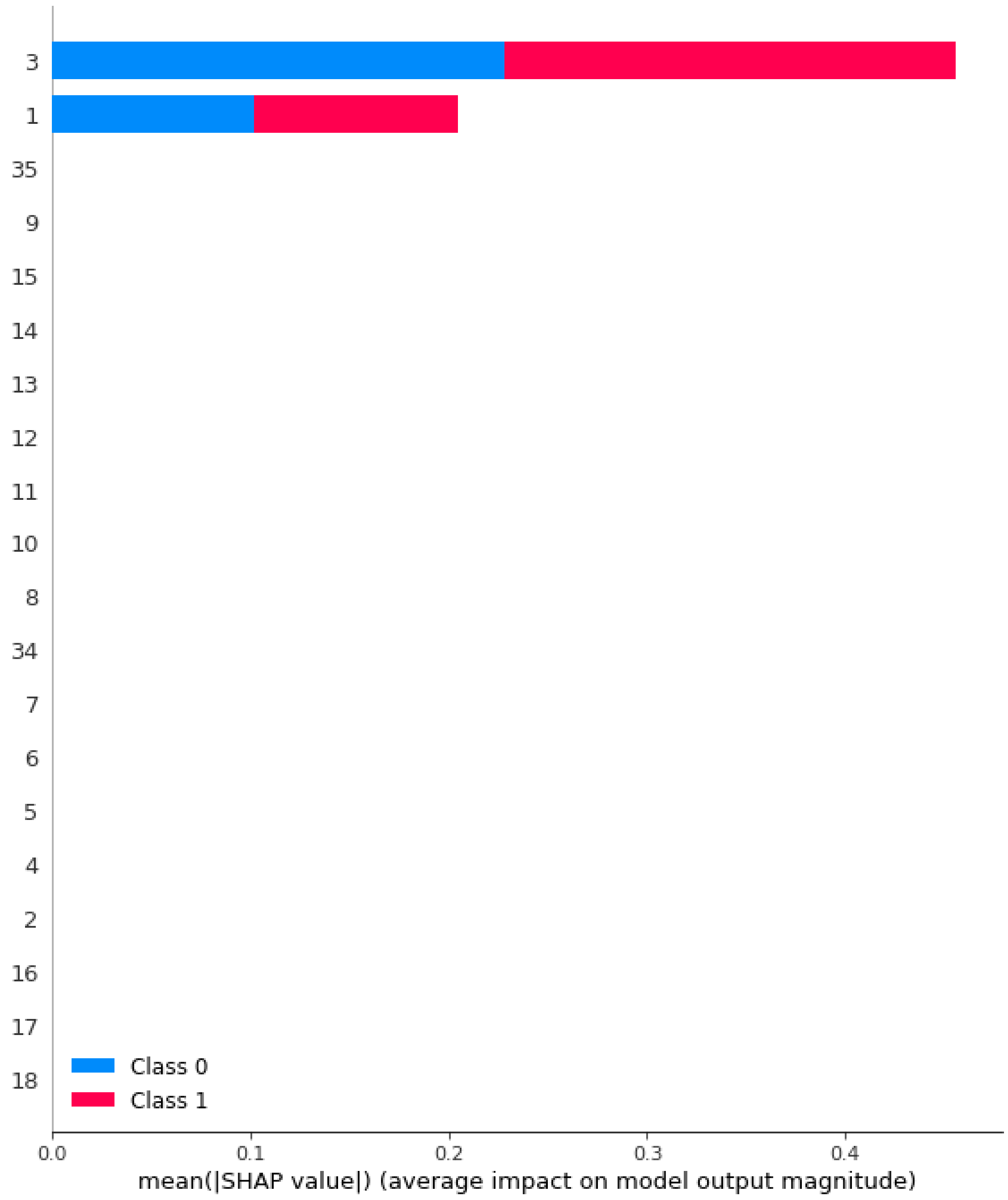
```
Decision Plot for SHAP Values from Class 0 in Test Set:
```



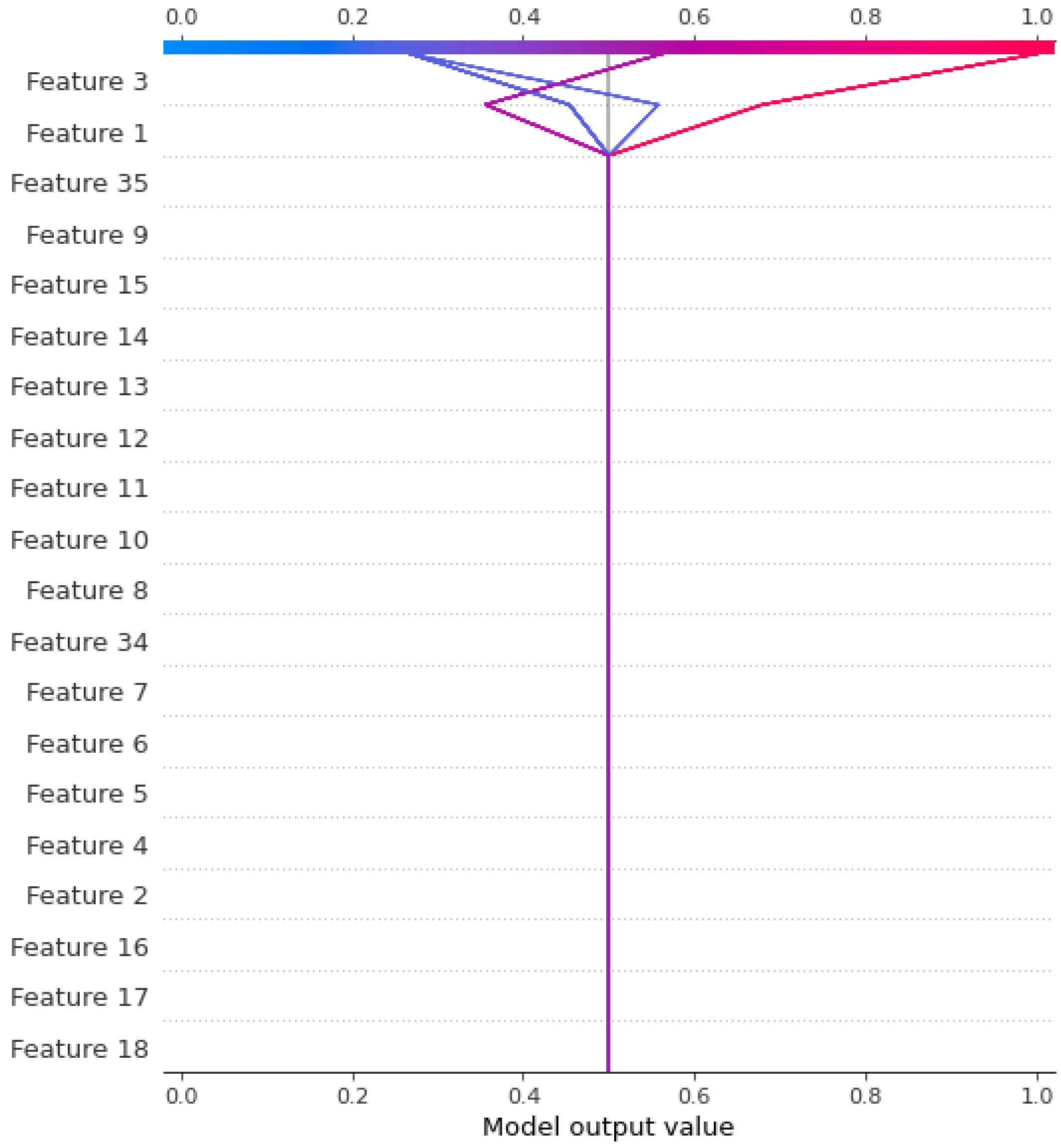
```
Decision Plot for SHAP Values from Class 1 in Test Set:
```



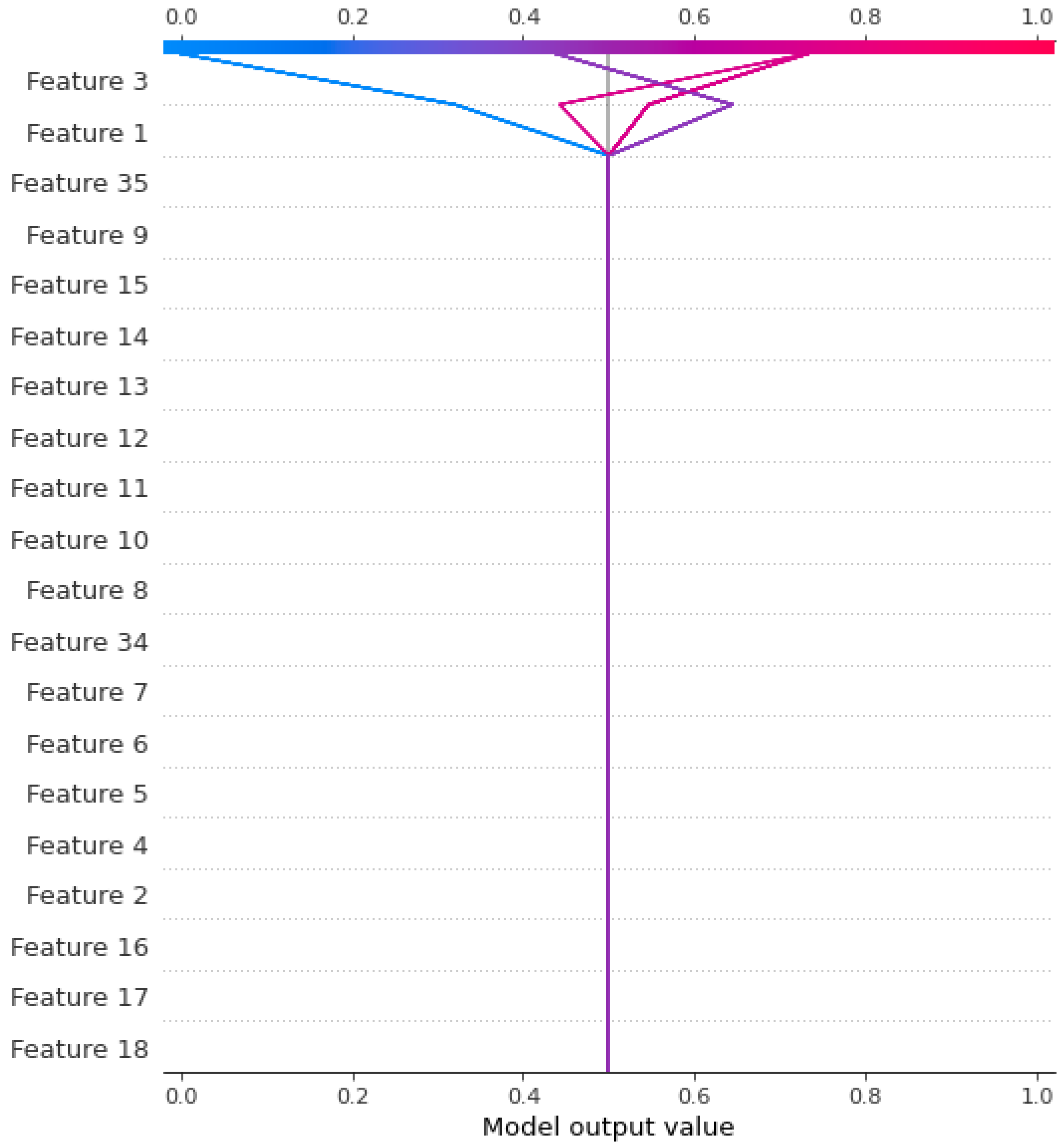
```
DT2 In CV2...

Checking if correct model is loaded...
 DecisionTreeClassifier(class_weight='balanced', max_depth=29,
                       min_samples_leaf=30, min_samples_split=45,
                       random_state=42)

Checking explainer for DT2...
<shap.explainers._tree.Tree object at 0x7fca0aff5310>

Checking shap values for DT...

[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 DT...

Expected value for DT: [0.5 0.5]
Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set:
```



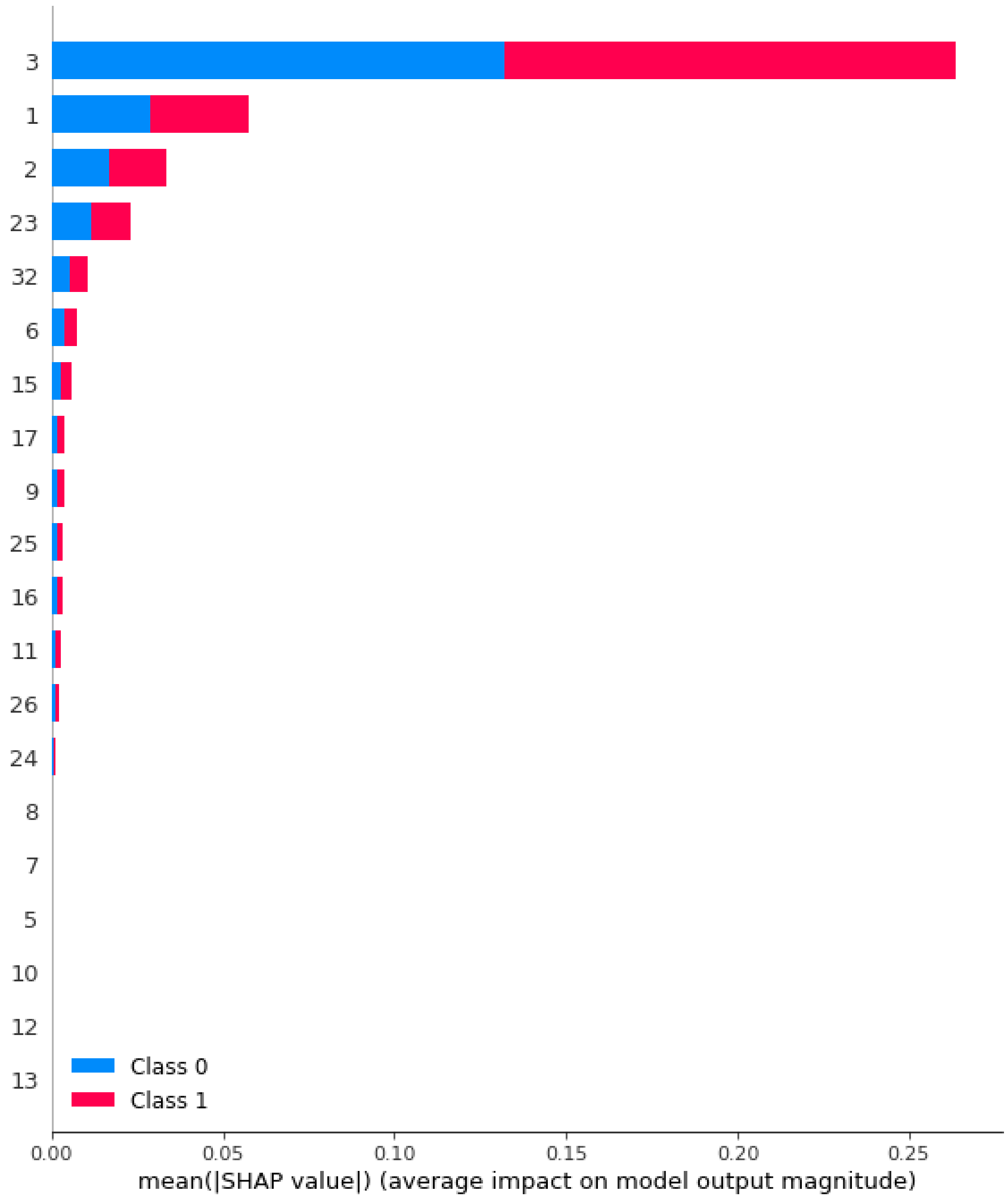
```
Decision Plot for SHAP Values from Class 0 in Test Set:
```



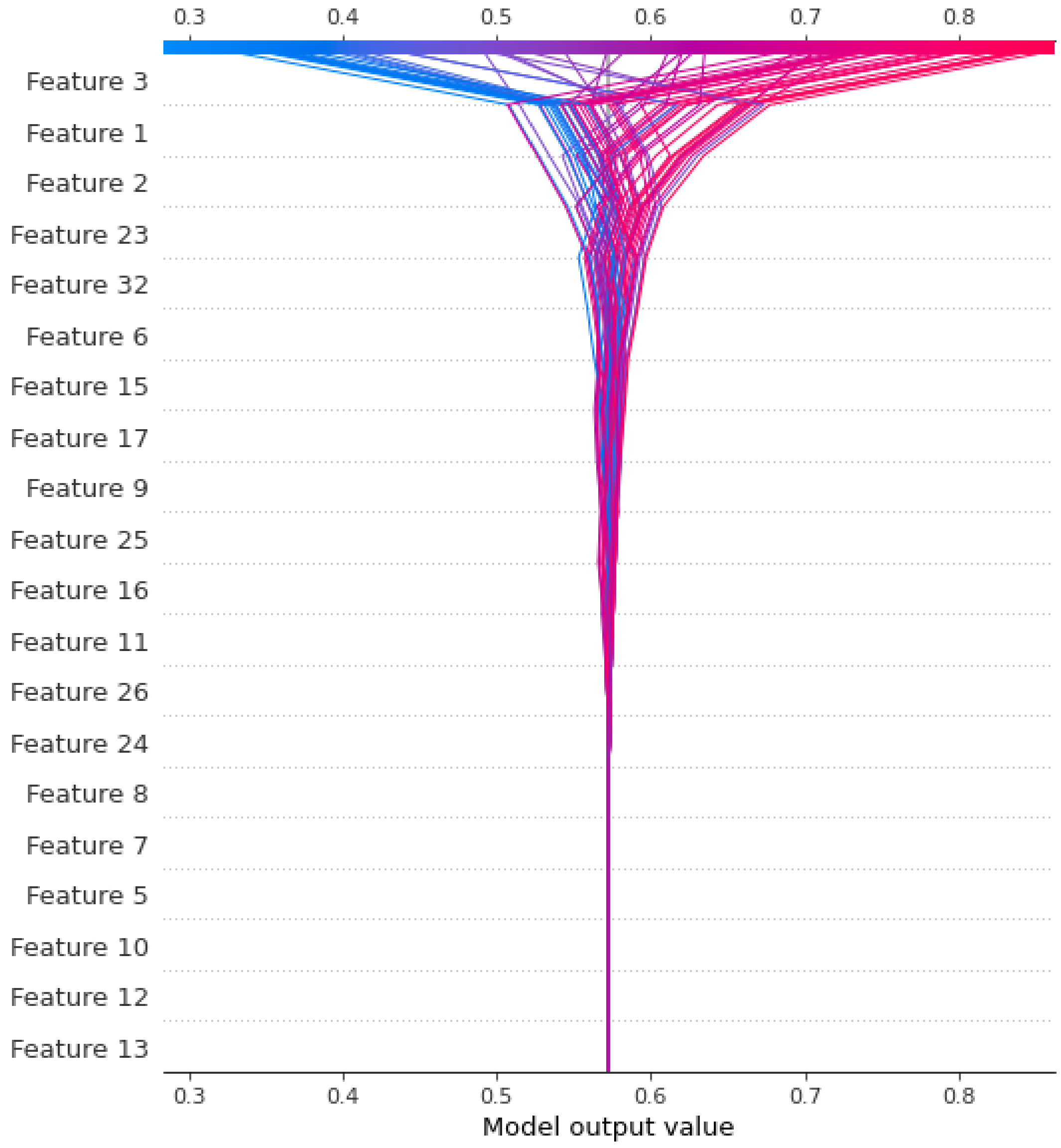
```
Decision Plot for SHAP Values from Class 1 in Test Set:
```



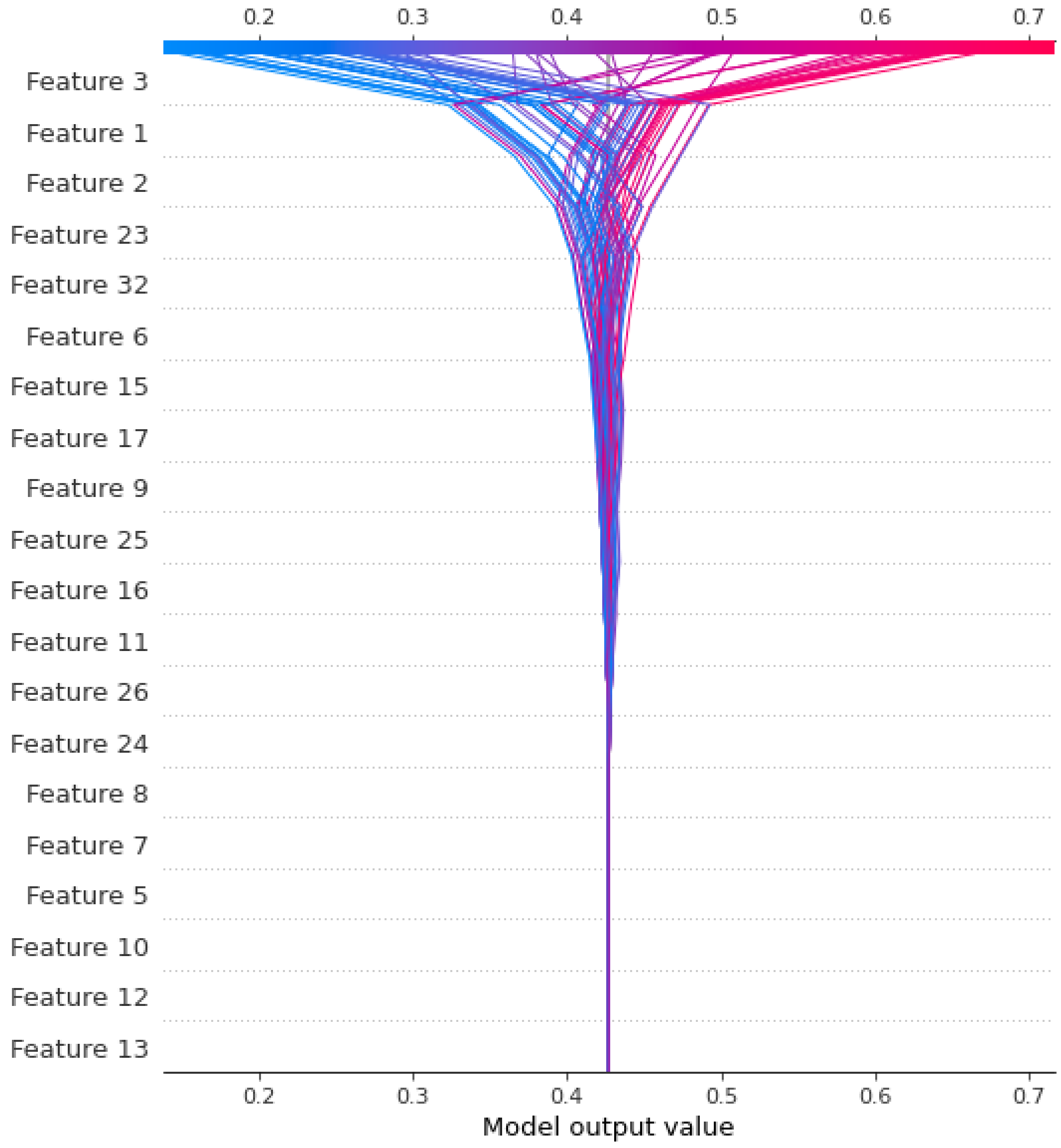
```
RF
RF0 In CV0...

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)

Checking explainer for RF0...
<shap.explainers._tree.Tree object at 0x7fca1d5e2e20>

Checking shap values for RF...

[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 RF...

Expected value for RF: [0.5728125 0.4271875]
Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set:
```



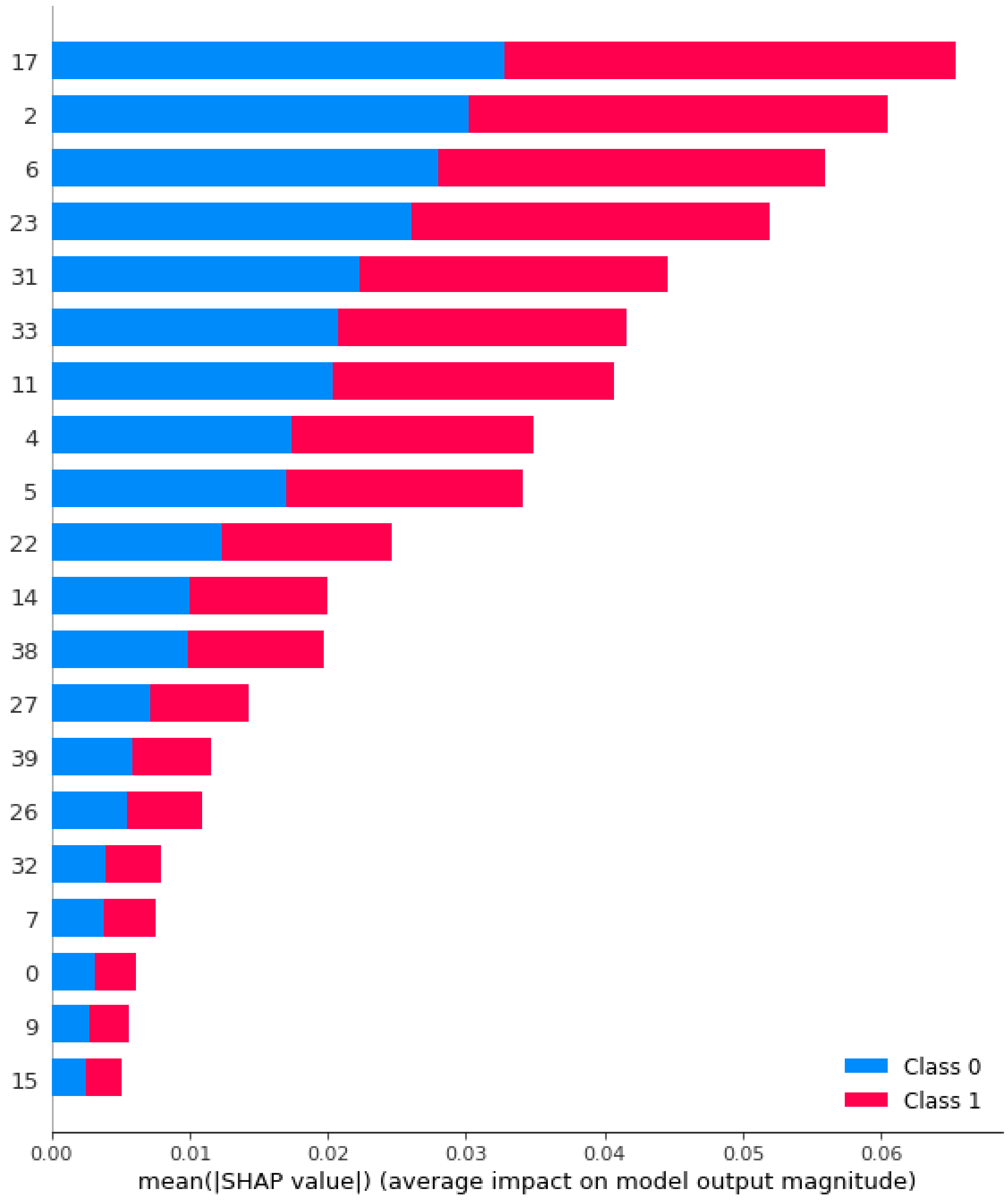
```
Decision Plot for SHAP Values from Class 0 in Test Set:
```



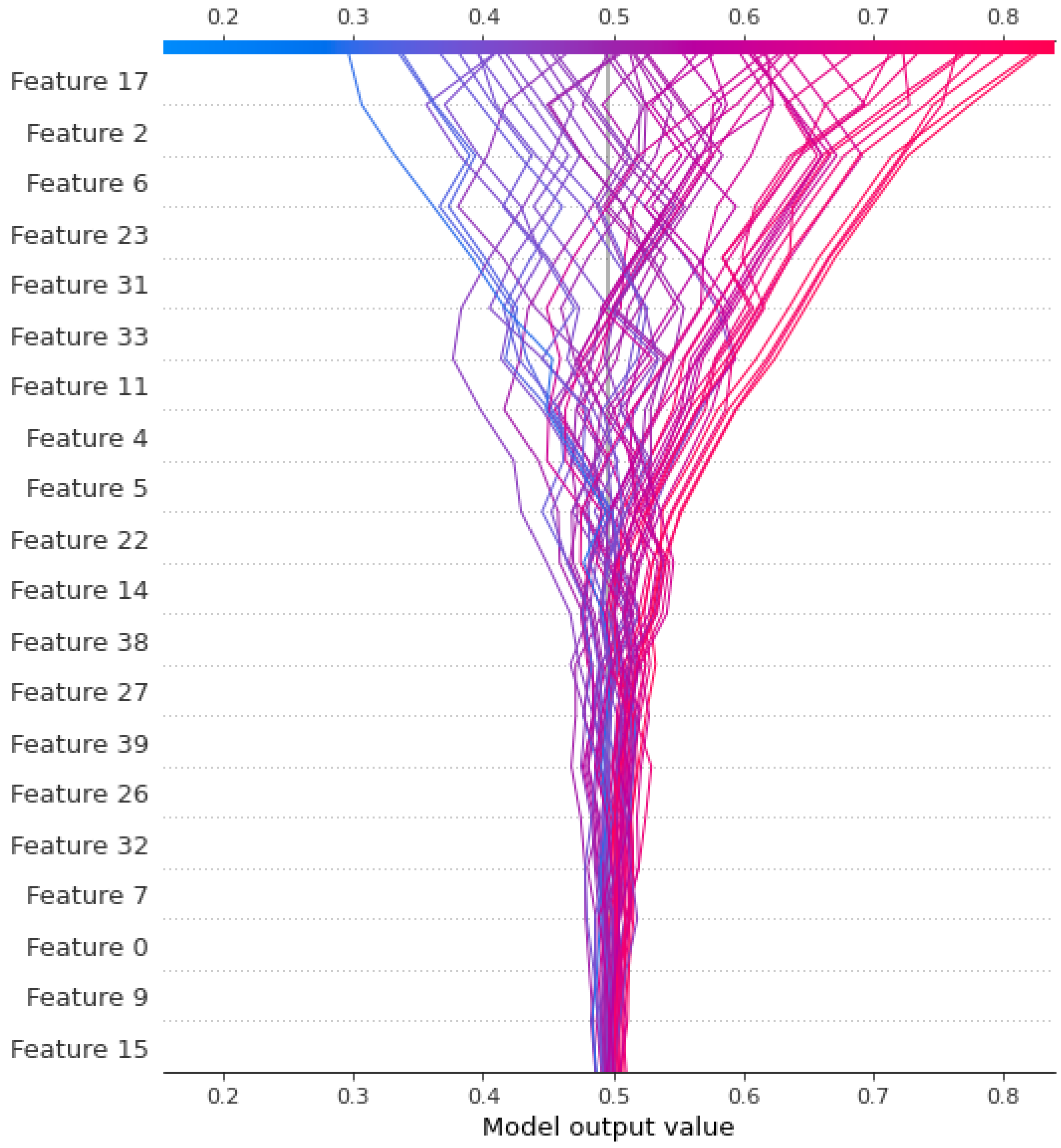
```
Decision Plot for SHAP Values from Class 1 in Test Set:
```



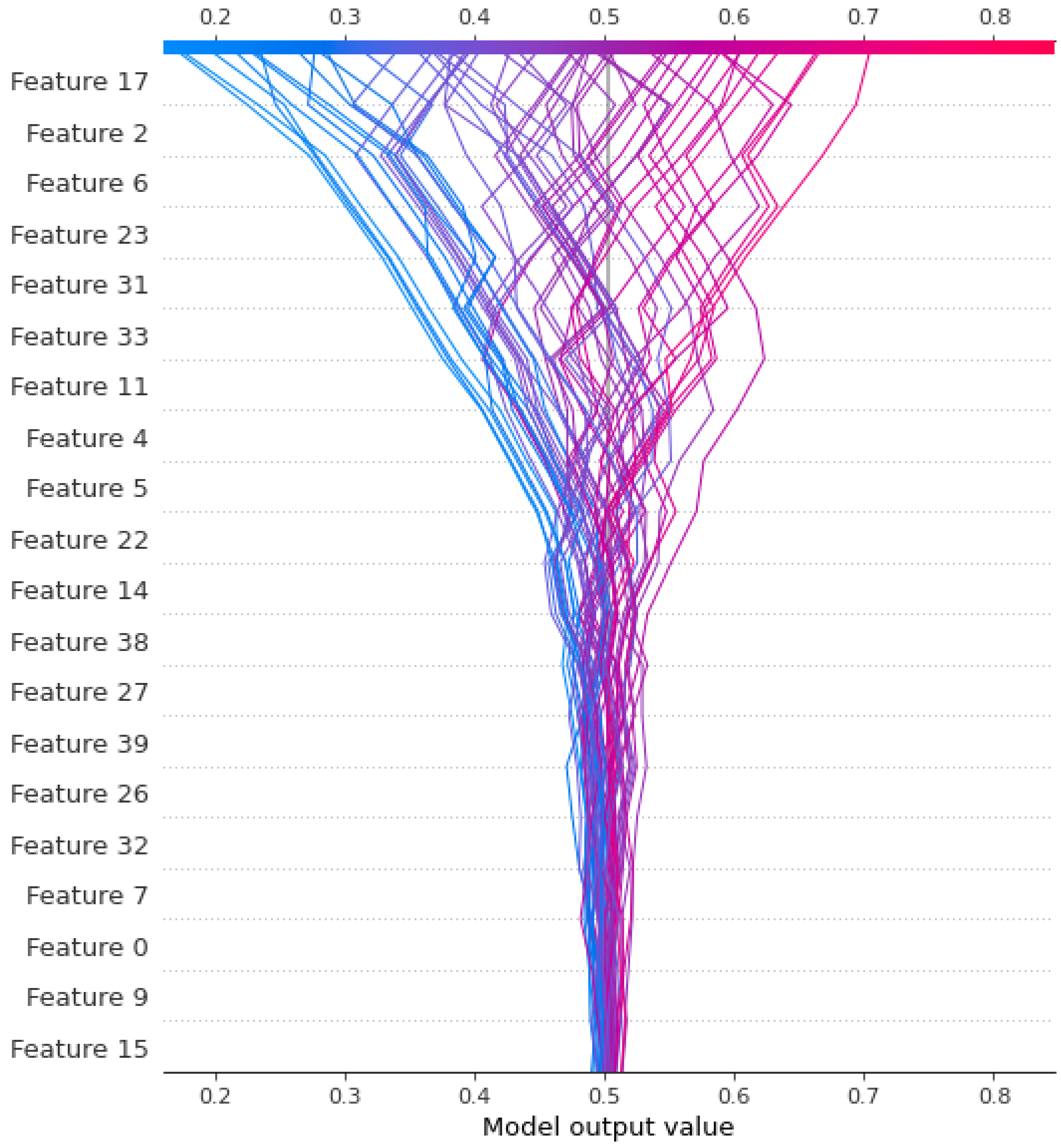
```
RF1 In CV1...

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)

Checking explainer for RF1...
<shap.explainers._tree.Tree object at 0x7fca0b91f8e0>

Checking shap values for RF...

[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 RF...

Expected value for RF: [0.49673858 0.50326142]
Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set:
```



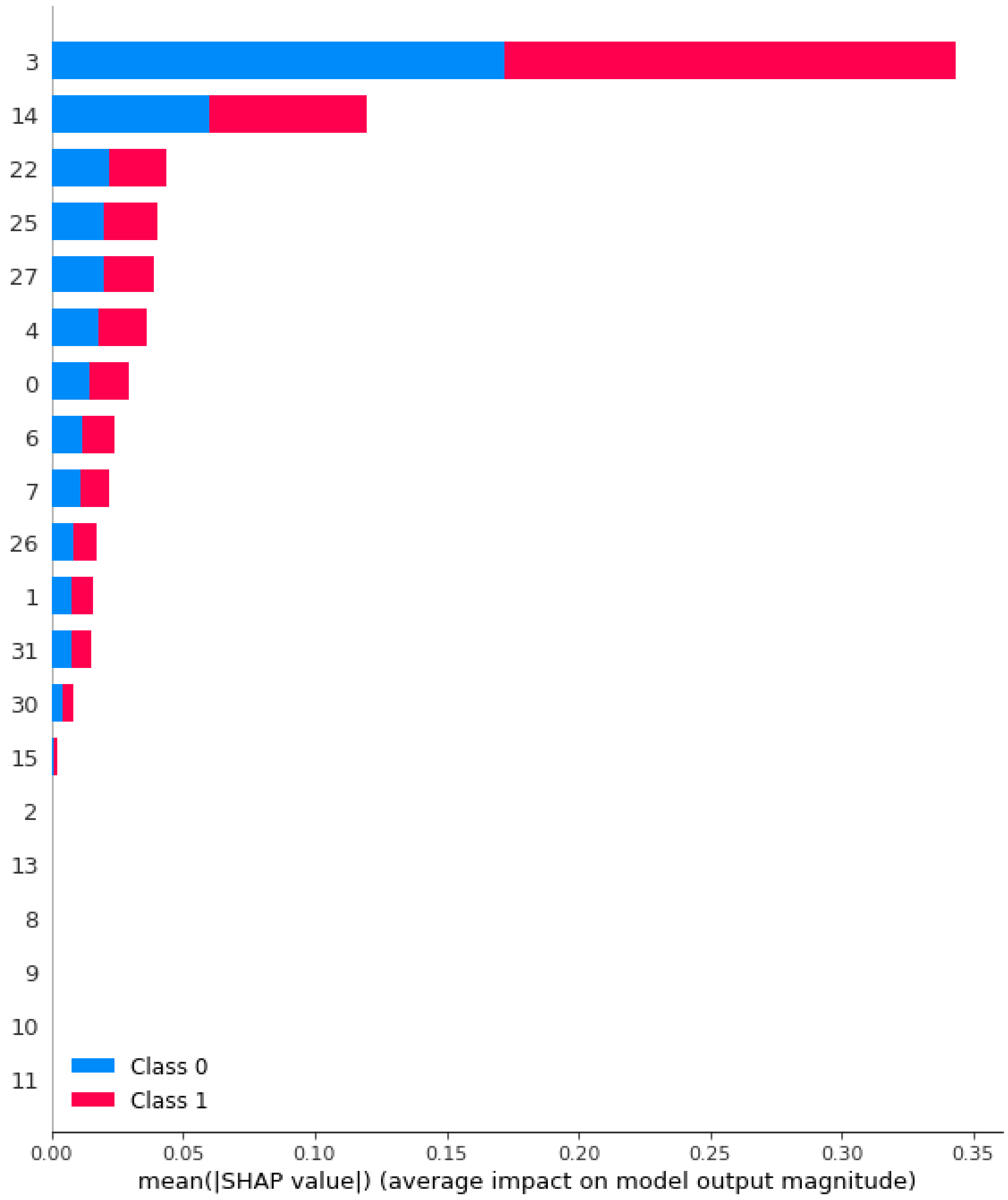
```
Decision Plot for SHAP Values from Class 0 in Test Set:
```



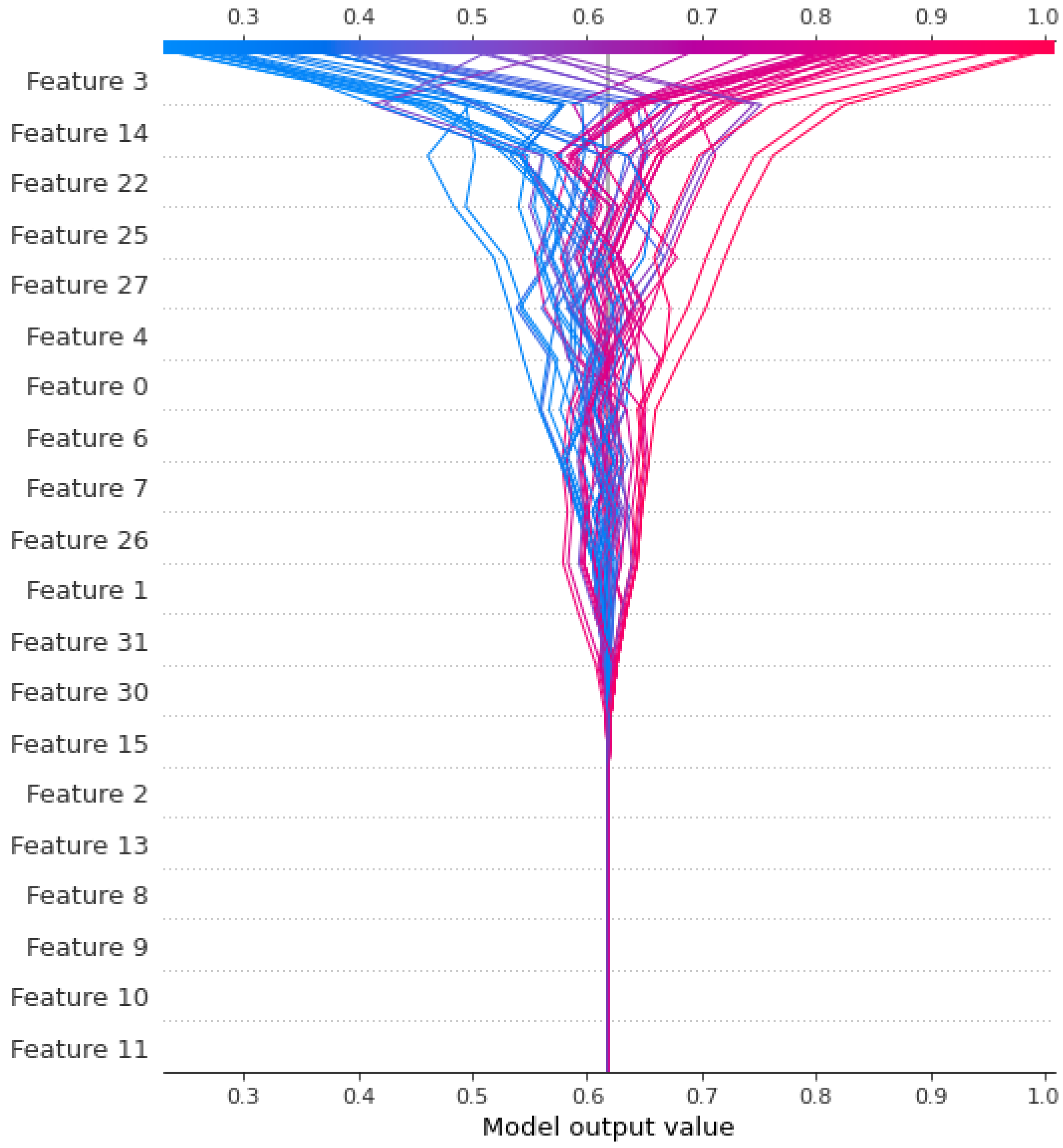
```
Decision Plot for SHAP Values from Class 1 in Test Set:
```



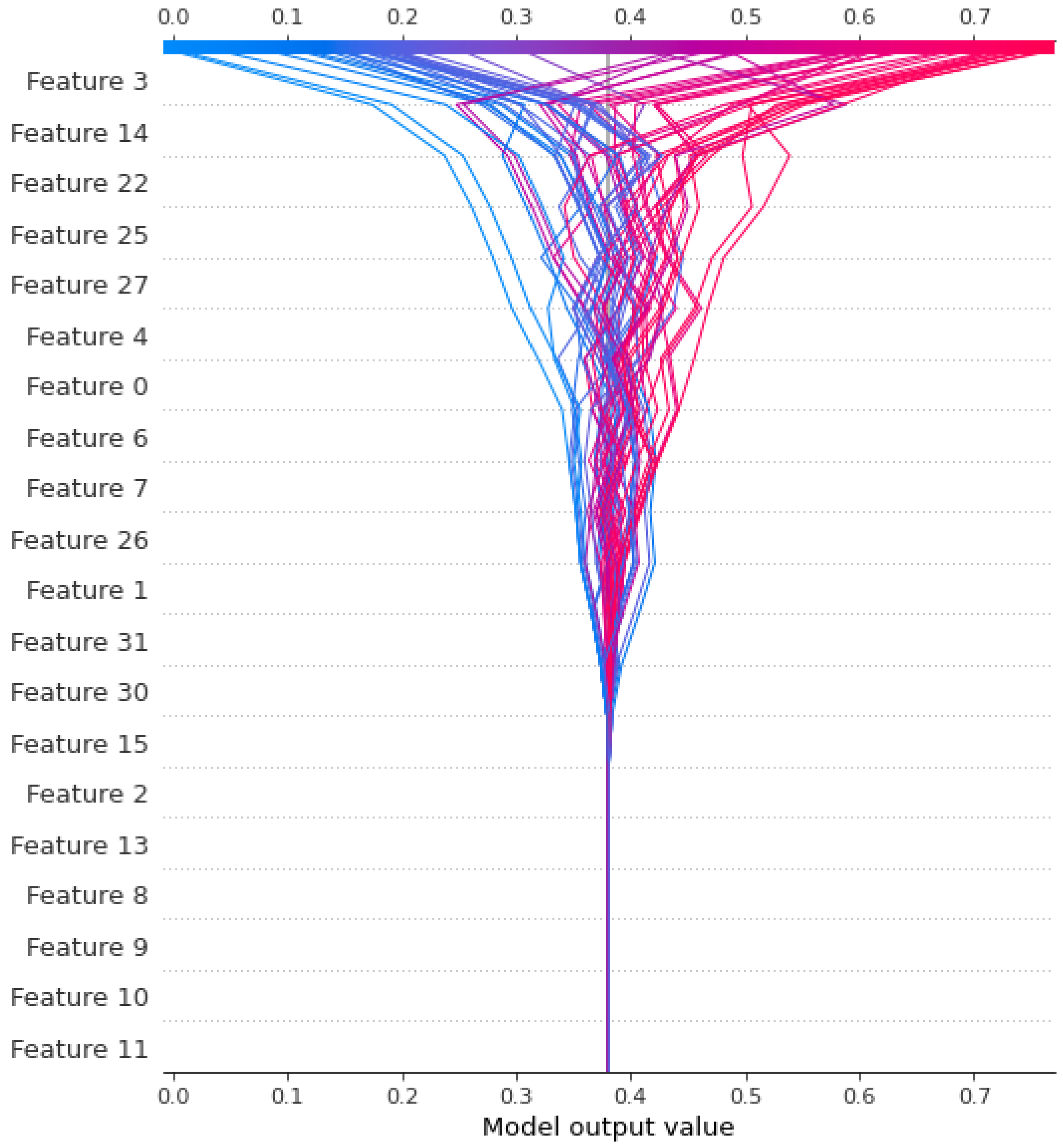
```
RF2 In CV2...

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)

Checking explainer for RF2...
<shap.explainers._tree.Tree object at 0x7fca0b849850>

Checking shap values for RF...

[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 RF...

Expected value for RF: [0.61909091 0.38090909]
Bar Summary Plot for SHAP Values in Class 0 & 1 in Test Set:
```



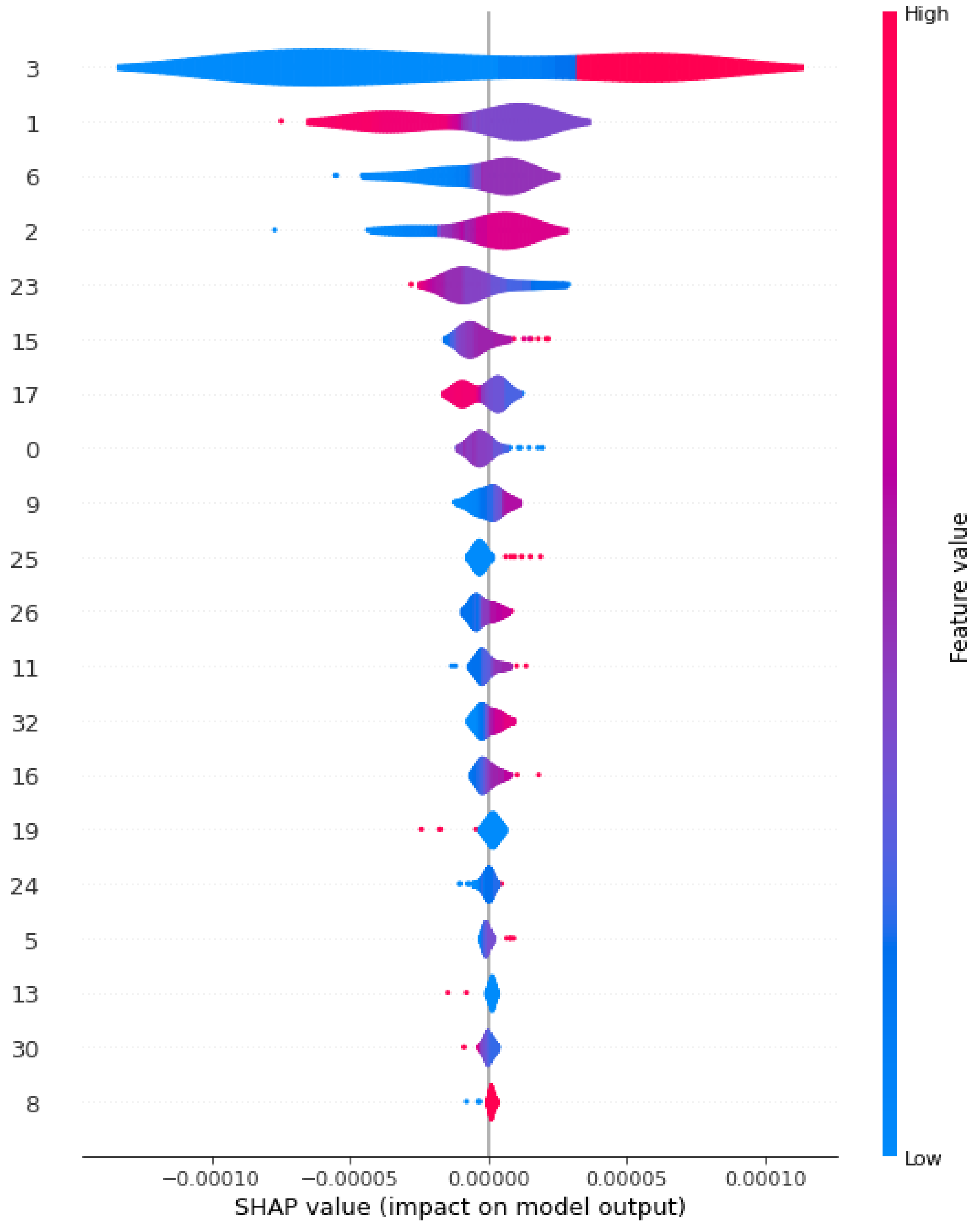
```
Decision Plot for SHAP Values from Class 0 in Test Set:
```



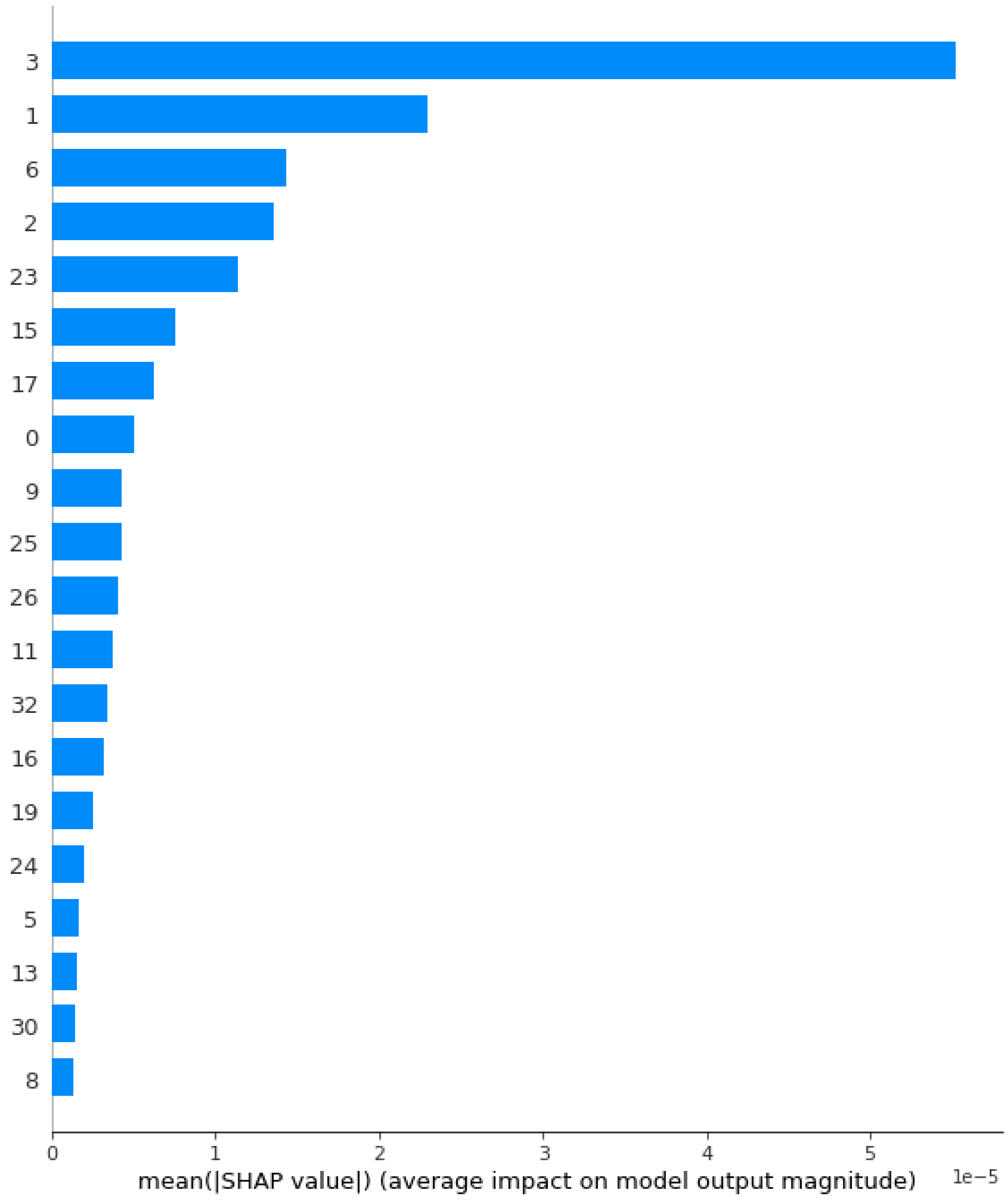
```
Decision Plot for SHAP Values from Class 1 in Test Set:
```



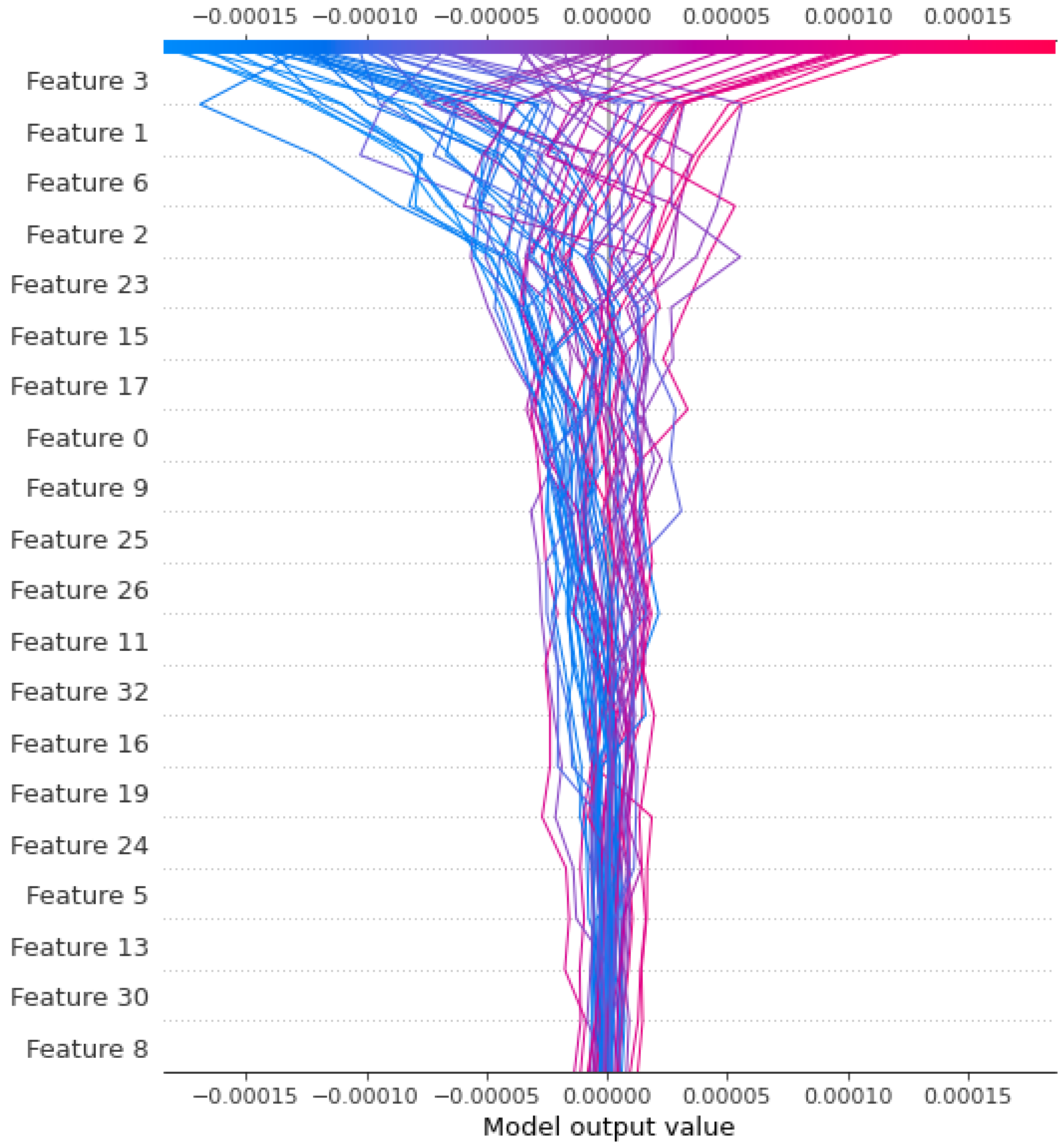
```
XGB
XGB0 In CV0...

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, ...)

Checking explainer for XGB0...
<shap.explainers._tree.Tree object at 0x7fca1c56cbe0>

Checking shap values for XGB...

[[-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 XGB...

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



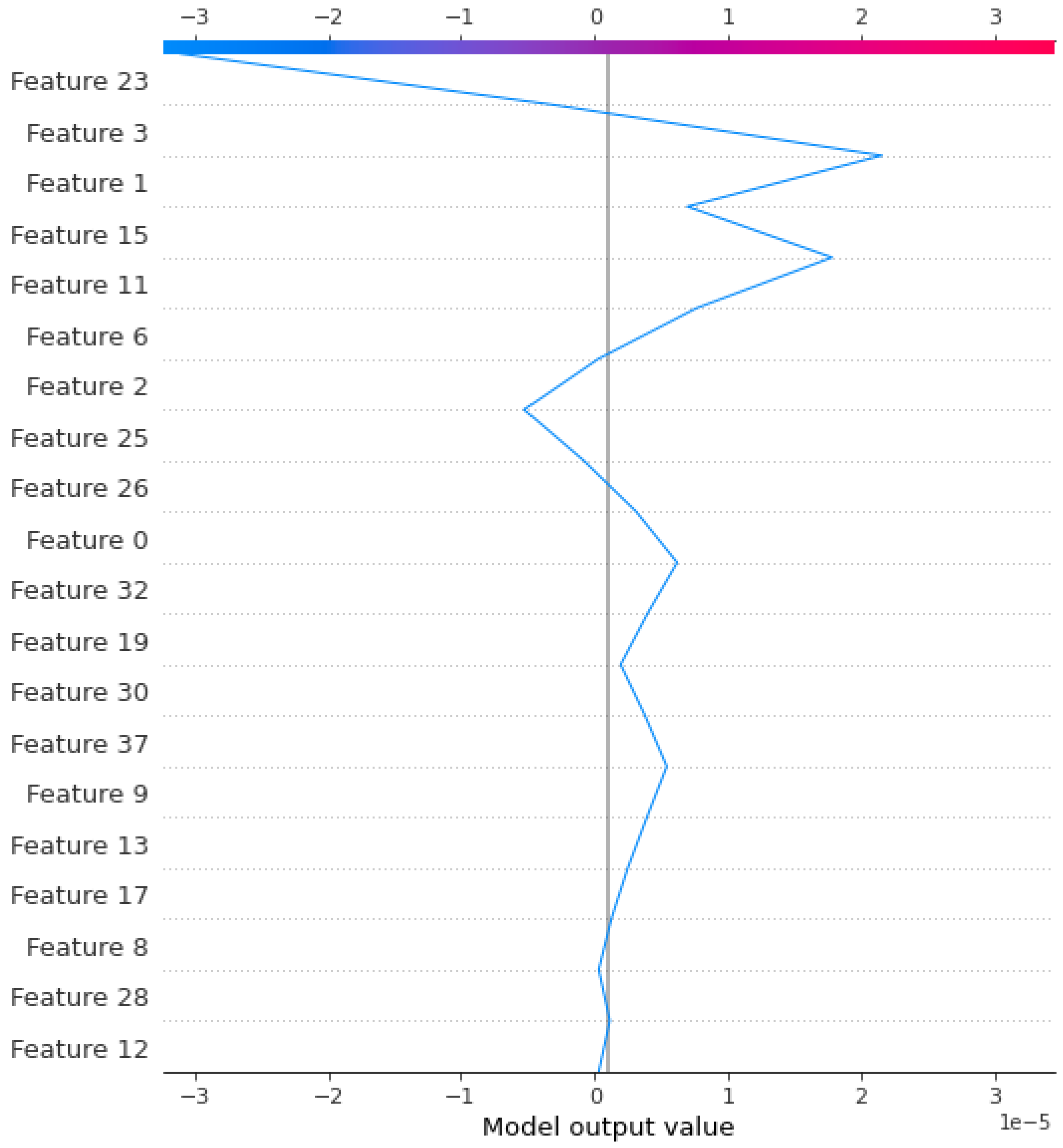
```
SHAP Bar Plot for SHAP Values Test Set:
```



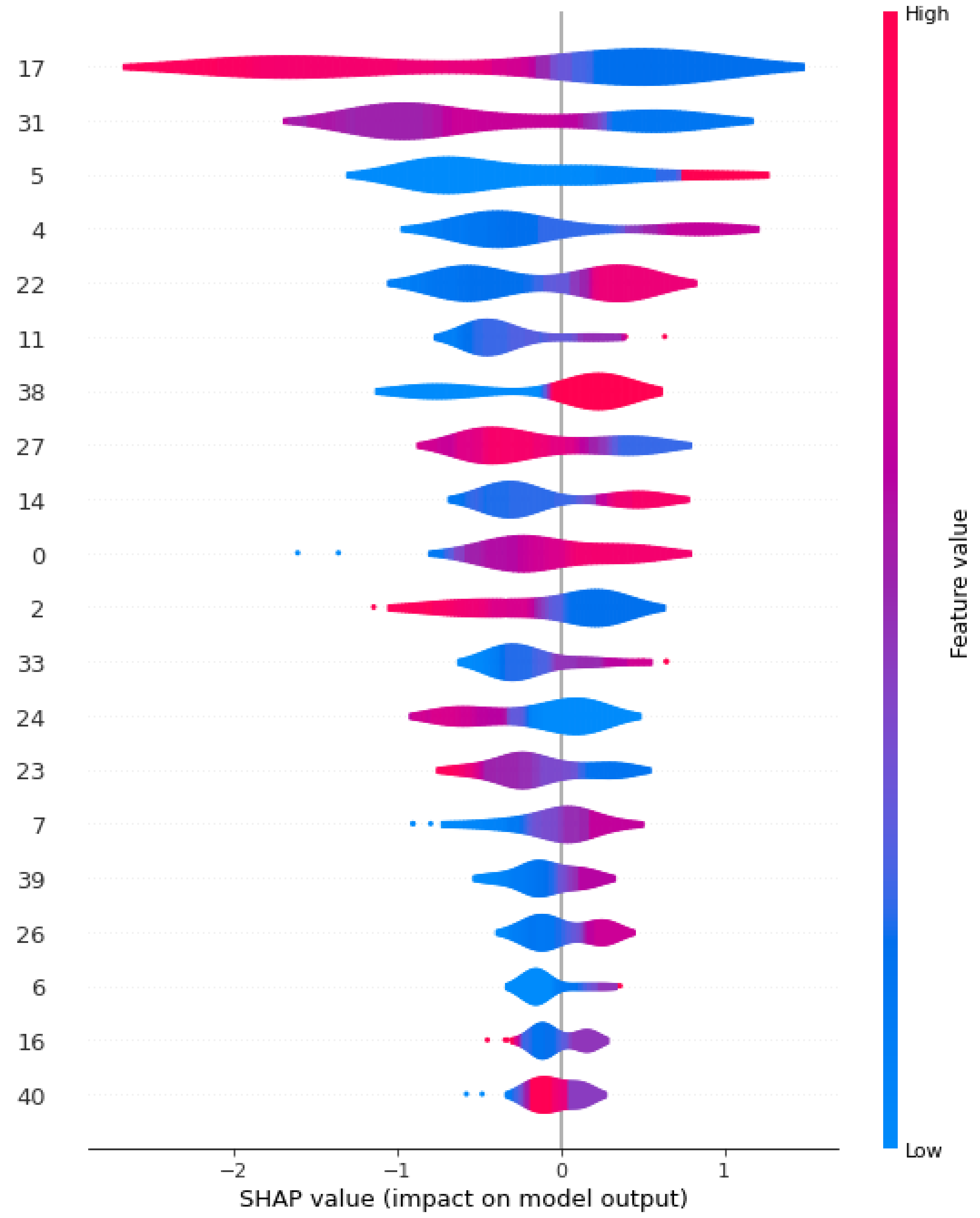
```
SHAP Decision Plot for SHAP Values in Test Set:
```



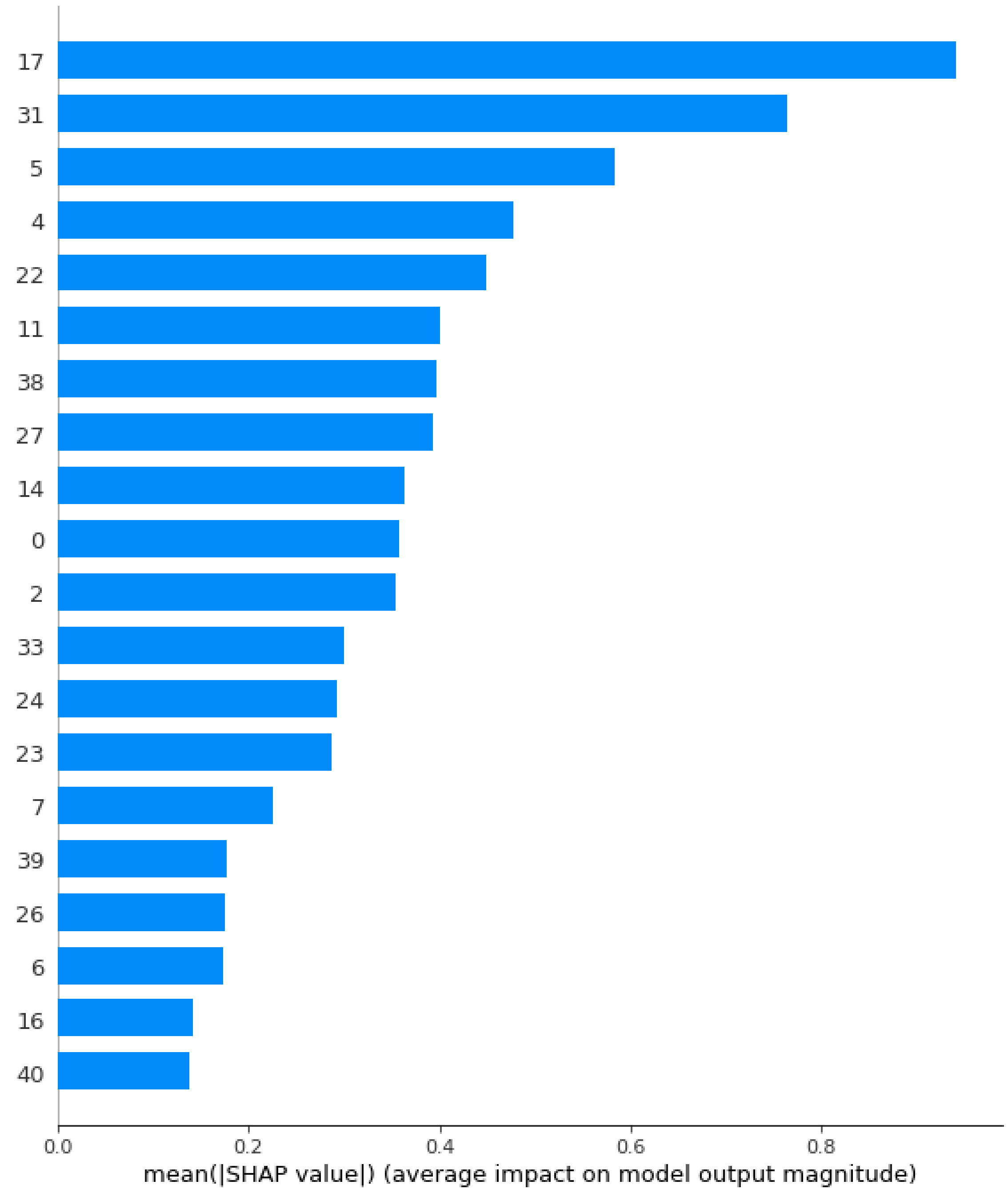
```
SHAP Decision Plot for Single-Prediction in Test Set:
```



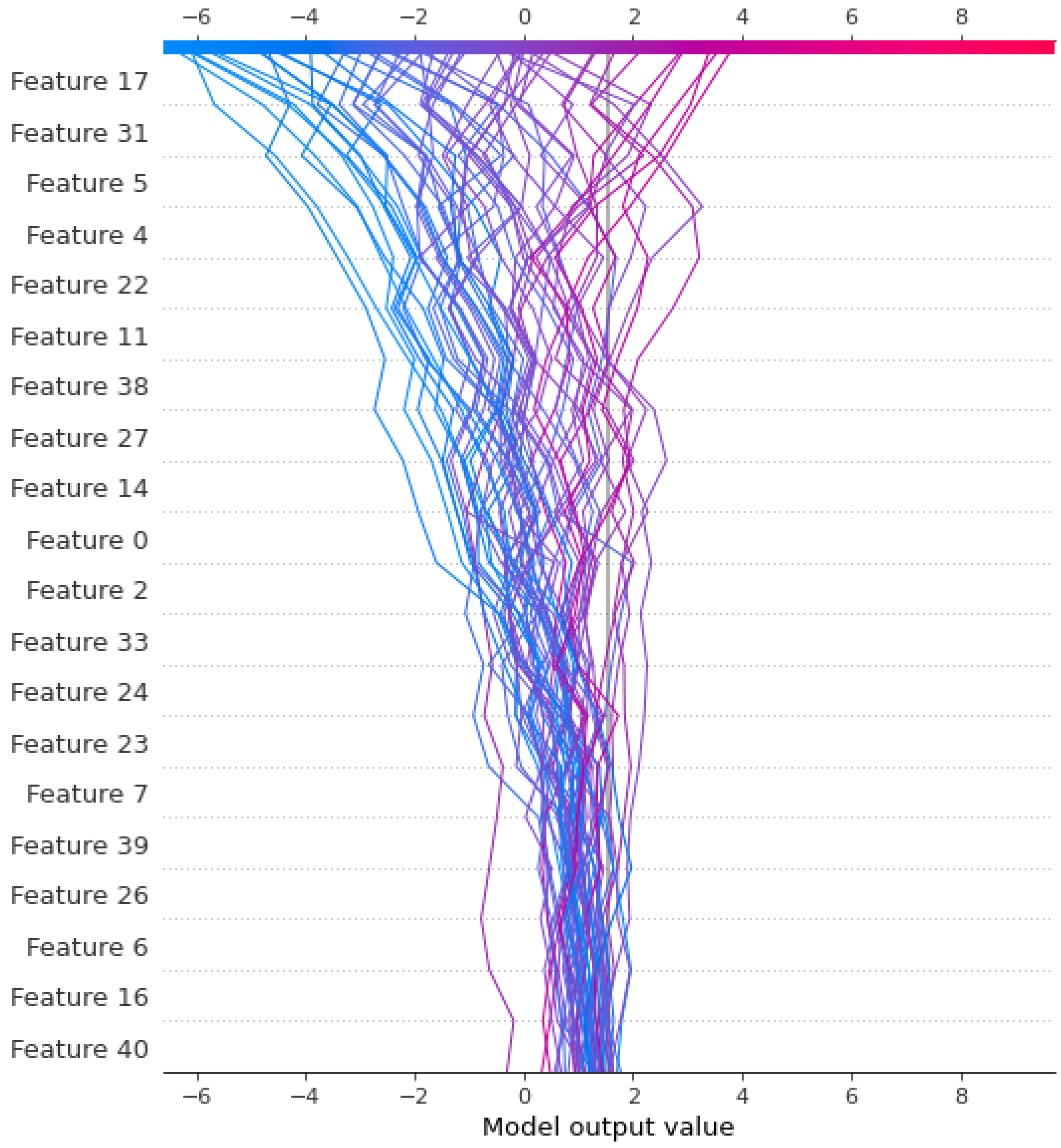
```
XGB1 In CV1...

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, ...)

Checking explainer for XGB1...
<shap.explainers._tree.Tree object at 0x7fca1c1f3490>

Checking shap values for XGB...

[[-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 XGB...

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



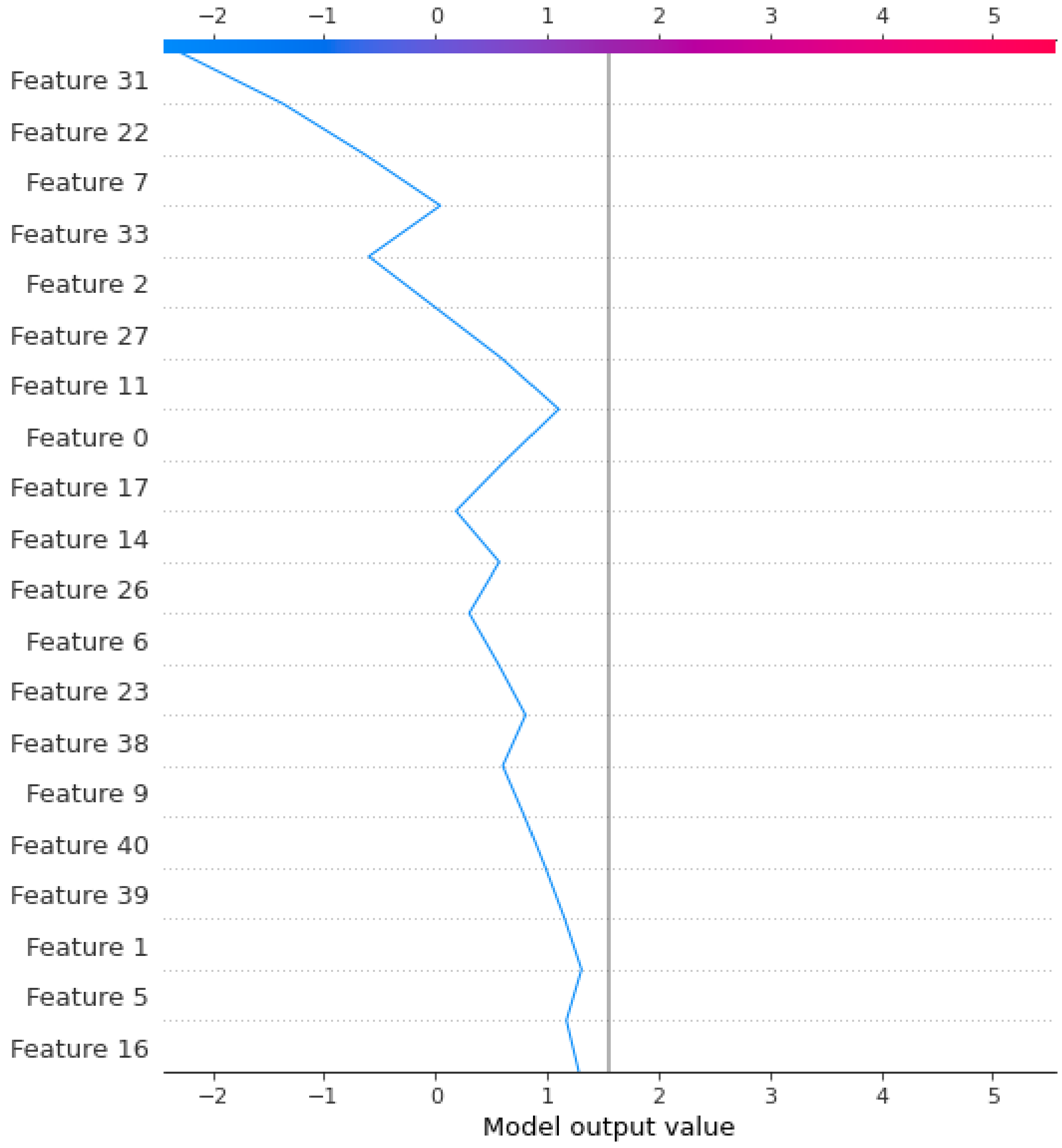
```
SHAP Bar Plot for SHAP Values Test Set:
```



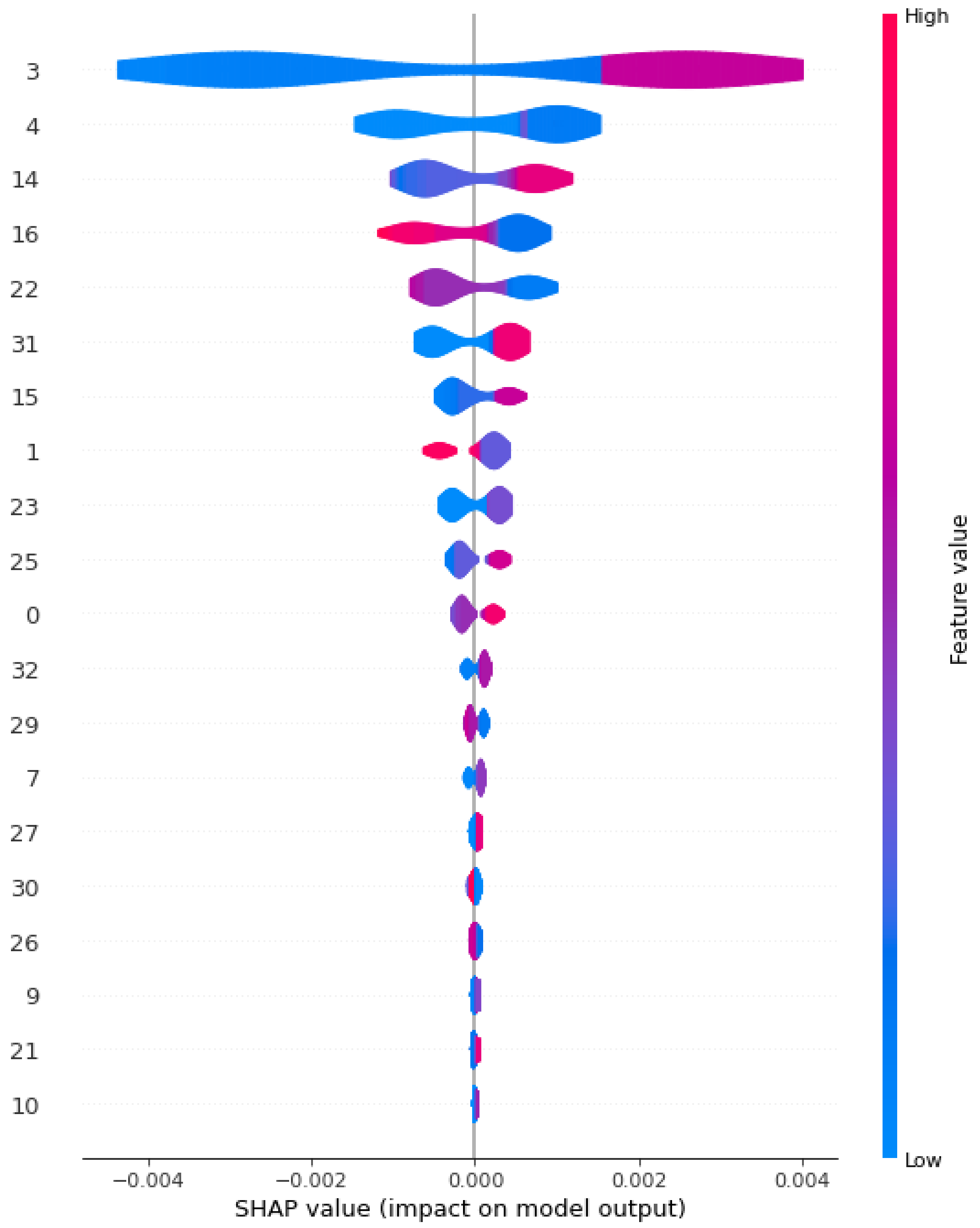
```
SHAP Decision Plot for SHAP Values in Test Set:
```



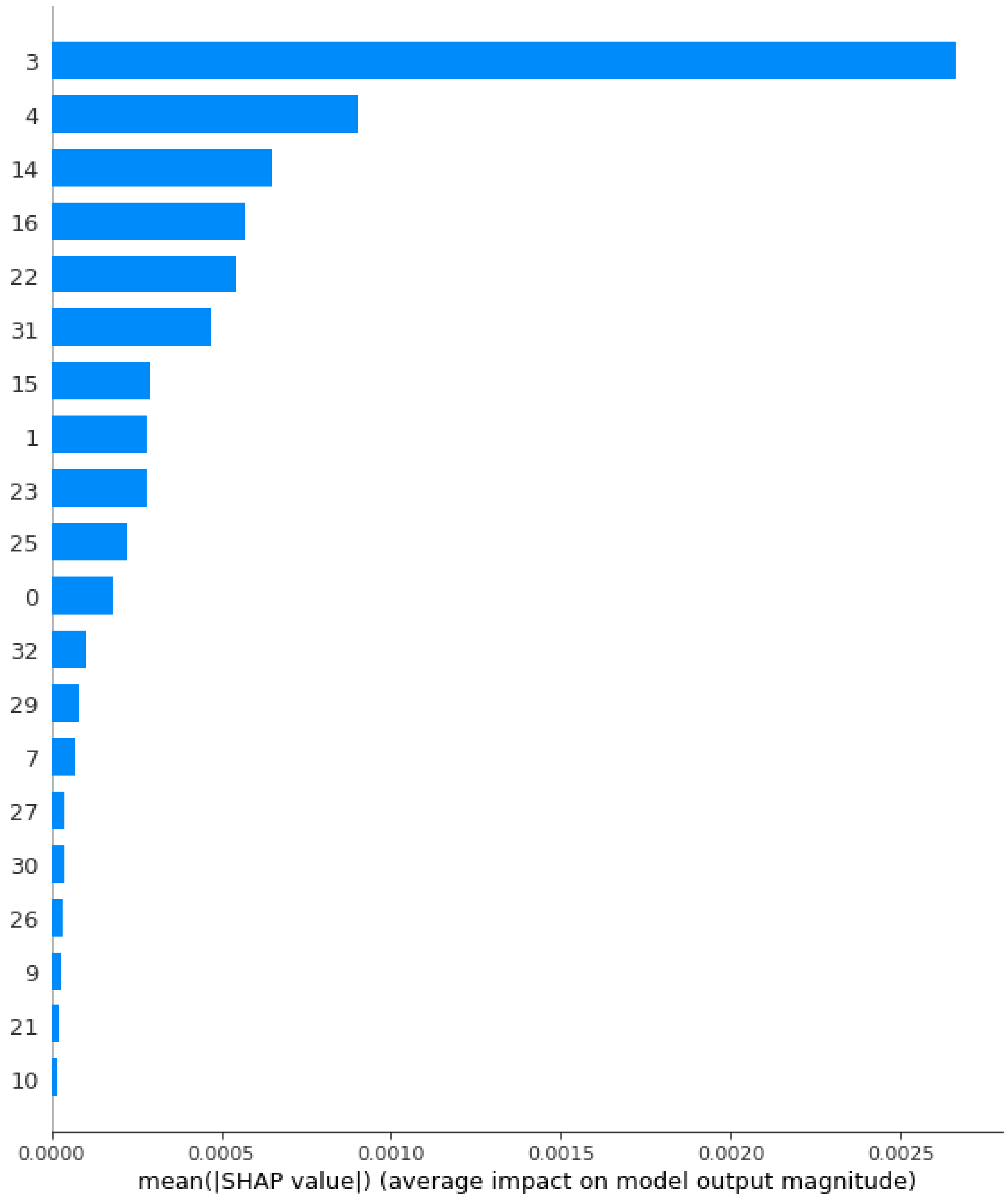
```
SHAP Decision Plot for Single-Prediction in Test Set:
```



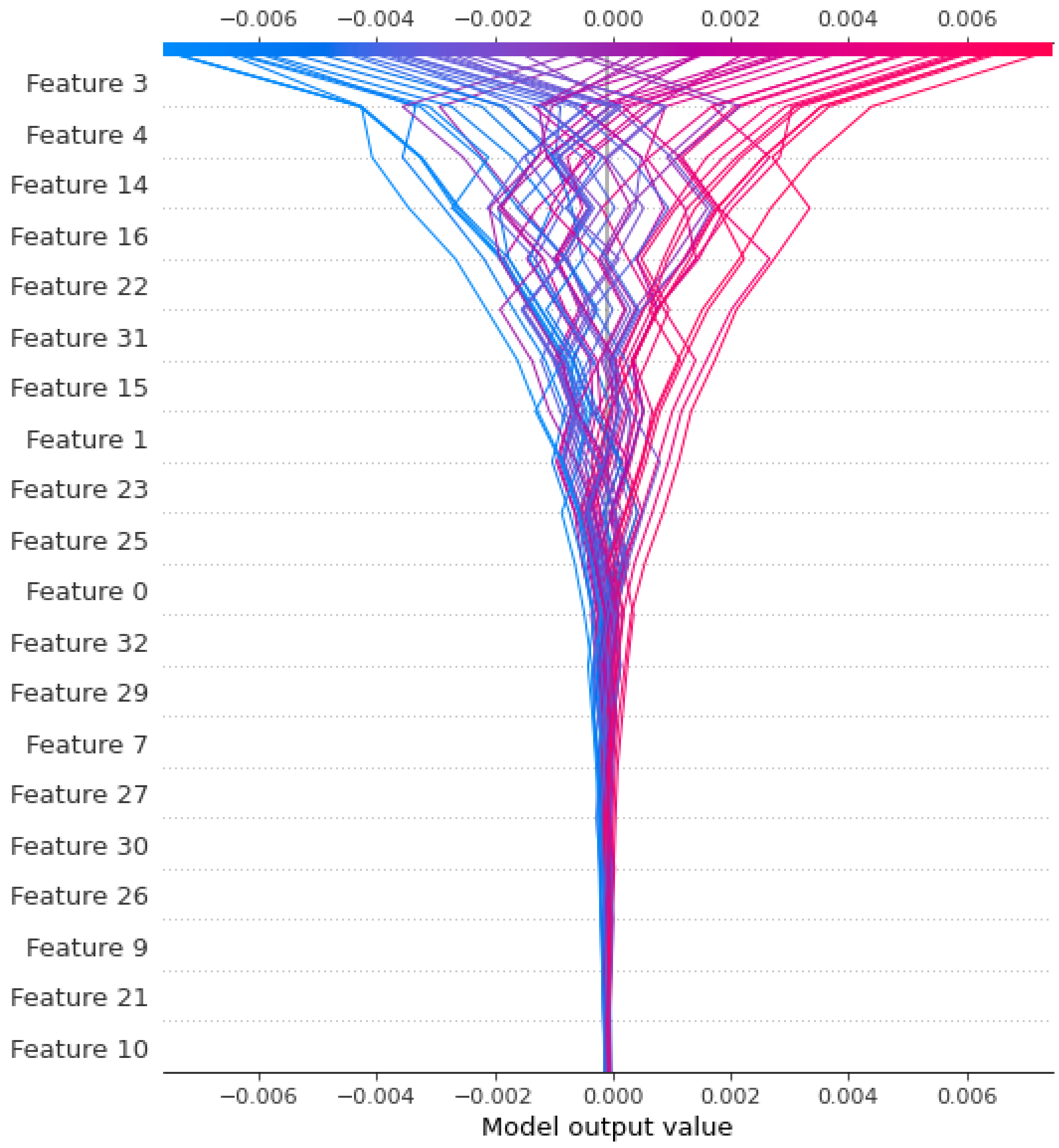
```
XGB2 In CV2...

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, ...)

Checking explainer for XGB2...
<shap.explainers._tree.Tree object at 0x7fca1c6b26a0>

Checking shap values for XGB...

[[-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 XGB...

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



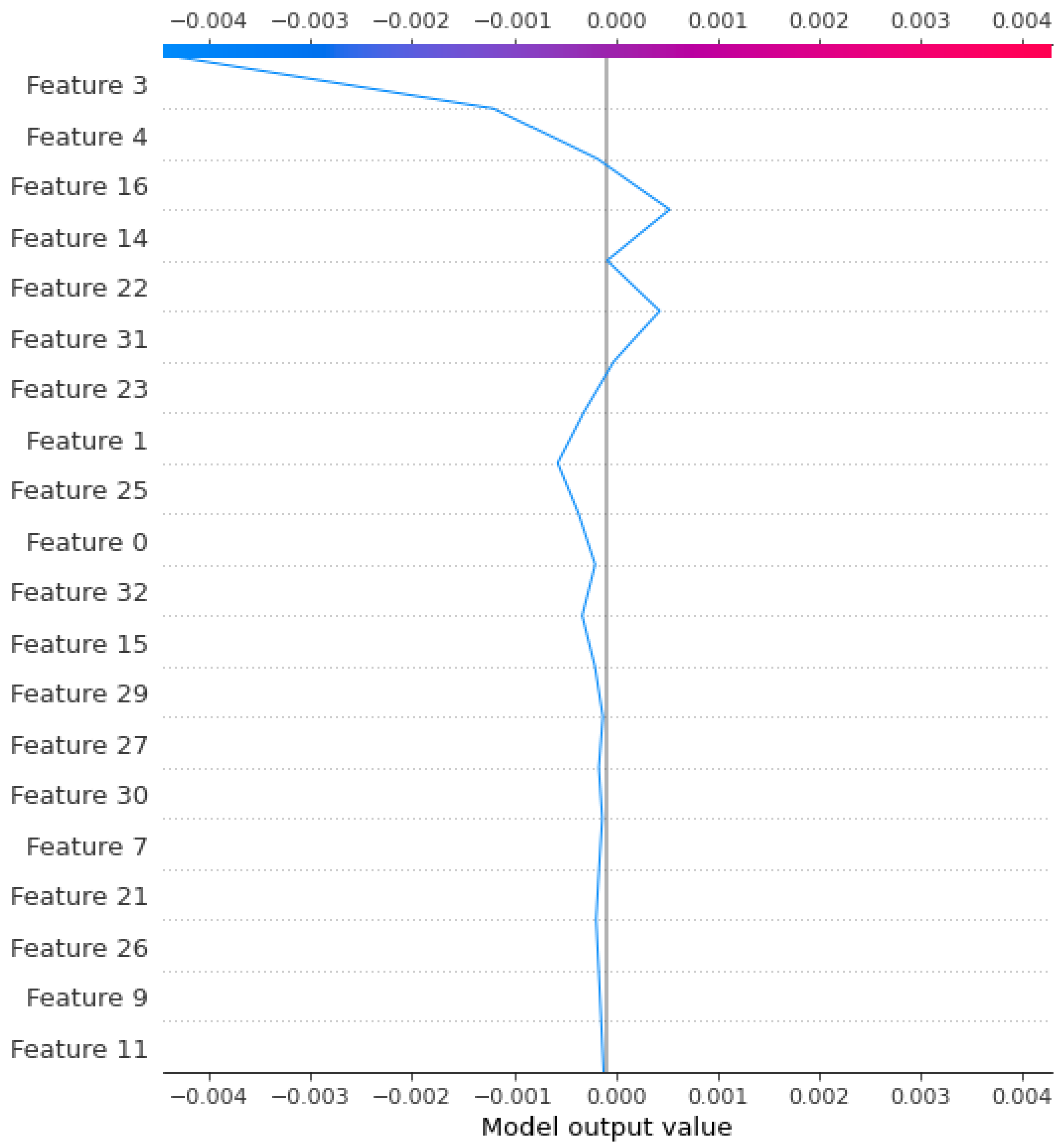
```
SHAP Bar Plot for SHAP Values Test Set:
```



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



```
SHAP Decision Plot for Single-Prediction in Test Set:
```



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

Checking if correct model is loaded...
 GaussianNB()

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

Checking shap values for NB...
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [17], in <cell line: 4>()
     32 print('\nChecking explainer for {}{}...\n{}'.format(abbrev[algorithm], cvCount, explainer))  # print explainer
to check if explainer exists
     34 print('\nChecking shap values for {}...\n'.format(abbrev[algorithm]))
---> 35 shap_values = compute_shapValues(model, abbrev[algorithm], explainer, trainX, trainY, testX, testY)
     37 print('\nChecking shap plots for {}...\n'.format(abbrev[algorithm]))
     38 shap_summary(abbrev[algorithm], shap_values, explainer, trainX, testX)

Input In [10], in compute_shapValues(model, abbrev, explainer, trainX, trainY, testX, testY)
      8 shap_values = None
     11 if abbrev in ["NB"]:
---> 12     shap_values= explainer(testX)  # permutation object cannot use .expected_value function like LR
     13     print(shap_values)
     17 if abbrev in ["LR"]:

File ~/opt/anaconda3/lib/python3.9/site-packages/shap/explainers/_permutation.py:82, in Permutation.__call__(self, max
_evals, main_effects, error_bounds, batch_size, outputs, silent, *args)
     78 def __call__(self, *args, max_evals=500, main_effects=False, error_bounds=False, batch_size="auto",
     79              outputs=None, silent=False):
     80     """ Explain the output of the model on the given arguments.
     81     """
---> 82     return super().__call__(
     83         *args, max_evals=max_evals, main_effects=main_effects, error_bounds=error_bounds, batch_size=batch_siz
e,
     84         outputs=outputs, silent=silent
     85     )

File ~/opt/anaconda3/lib/python3.9/site-packages/shap/explainers/_explainer.py:266, in Explainer.__call__(self, max_ev
als, main_effects, error_bounds, batch_size, outputs, silent, *args, **kwargs)
    264     feature_names = [[] for _ in range(len(args))]
    265 for row_args in show_progress(zip(*args), num_rows, self.__class__.__name__+" explainer", silent):
--> 266     row_result = self.explain_row(
    267         *row_args, max_evals=max_evals, main_effects=main_effects, error_bounds=error_bounds,
    268         batch_size=batch_size, outputs=outputs, silent=silent, **kwargs
    269     )
    270     values.append(row_result.get("values", None))
    271     output_indices.append(row_result.get("output_indices", None))

File ~/opt/anaconda3/lib/python3.9/site-packages/shap/explainers/_permutation.py:140, in Permutation.explain_row(self,
max_evals, main_effects, error_bounds, batch_size, outputs, silent, *row_args)
    137     i += 1
    139 # evaluate the masked model
--> 140 outputs = fm(masks, zero_index=0, batch_size=batch_size)
    142 if row_values is None:
    143     row_values = np.zeros((len(fm),) + outputs.shape[1:])

File ~/opt/anaconda3/lib/python3.9/site-packages/shap/utils/_masked_model.py:57, in MaskedModel.__call__(self, masks,
 zero_index, batch_size)
     55 if len(masks.shape) == 1:
     56     if getattr(self.masker, "supports_delta_masking", False):
---> 57         return self._delta_masking_call(masks, zero_index=zero_index, batch_size=batch_size)
     59     # we need to convert from delta masking to a full masking call because we were given a delta masking
     60     # input but the masker does not support delta masking
     61     else:
     62         full_masks = np.zeros((int(np.sum(masks >= 0)), self._masker_cols), dtype=np.bool)

File ~/opt/anaconda3/lib/python3.9/site-packages/shap/utils/_masked_model.py:203, in MaskedModel._delta_masking_call(s
elf, masks, zero_index, batch_size)
    200     batch_positions[i+1] = batch_positions[i] + num_varying_rows[i]
    202 # joined_masked_inputs = self._stack_inputs(all_masked_inputs)
--> 203 outputs = self.model(*subset_masked_inputs)
    204 _assert_output_input_match(subset_masked_inputs, outputs)
    206 if self.linearize_link and self.link != links.identity and self._linearizing_weights is None:

File ~/opt/anaconda3/lib/python3.9/site-packages/shap/models/_model.py:26, in Model.__call__(self, *args)
     25 def __call__(self, *args):
---> 26     out = self.inner_model(*args)
     27     is_tensor = safe_isinstance(out, "torch.Tensor")
     28     out = out.cpu().detach().numpy() if is_tensor else np.array(out)

File ~/opt/anaconda3/lib/python3.9/site-packages/sklearn/naive_bayes.py:82, in _BaseNB.predict(self, X)
     68 """
     69 Perform classification on an array of test vectors X.
     70 
   (...)
     79     Predicted target values for X.
     80 """
     81 check_is_fitted(self)
---> 82 X = self._check_X(X)
     83 jll = self._joint_log_likelihood(X)
     84 return self.classes_[np.argmax(jll, axis=1)]

File ~/opt/anaconda3/lib/python3.9/site-packages/sklearn/naive_bayes.py:251, in GaussianNB._check_X(self, X)
    249 def _check_X(self, X):
    250     """Validate X, used only in predict* methods."""
--> 251     return self._validate_data(X, reset=False)

File ~/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:585, in BaseEstimator._validate_data(self, X, y, rese
t, validate_separately, **check_params)
    582     out = X, y
    584 if not no val X and check params.get("ensure_2d", True):
--> 585     self._check_n_features(X, reset=reset)
    587 return out

File ~/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:400, in BaseEstimator._check_n_features(self, X, rese
t)
    397     return
    399 if n_features != self.n_features_in_:
--> 400     raise ValueError(
    401         f"X has {n_features} features, but {self.__class__.__name__} "
    402         f"is expecting {self.n_features_in_} features as input."
    403     )

ValueError: X has 39 features, but GaussianNB is expecting 37 features as input.
```

Next Steps

- 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

In [ ]:

In [ ]: