```
###! pip install ucimlrepo
```

Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests (see [Cortez et al., 2009], http://www3.dsi.uminho.pt/pcortez/wine/).

Dataset Characteristics - `Multivariate`
Subject Area - `Business`
Associated Tasks - `Classification, Regression`
Feature Type - `Real`

```
###! pip install scikit-learn
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

```
# Importing all datasets
wine_quality = pd.read_csv("wine_quality.csv")
wine_quality.head(4)
```

	Unnamed: 0	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxic
0	0	7.4	0.70	0.00	1.9	0.076	11
1	1	7.8	0.88	0.00	2.6	0.098	25
2	2	7.8	0.76	0.04	2.3	0.092	15
3	3	11.2	0.28	0.56	1.9	0.075	17

```
wine_quality.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Unnamed: 0            6497 non-null   int64  
 1   fixed_acidity         6497 non-null   float64
 2   volatile_acidity      6497 non-null   float64
 3   citric_acid           6497 non-null   float64
 4   residual_sugar        6497 non-null   float64
 5   chlorides             6497 non-null   float64
 6   free_sulfur_dioxide   6497 non-null   float64
 7   total_sulfur_dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6497 non-null   float64
 10  sulphates             6497 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 660.0 KB
```

```
wine_quality.isnull().sum()
```

```
Unnamed: 0              0
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```
wine_quality['quality'].value_counts()
```

```
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```

```
# This dataset contains features

wine_quality.describe()
```

	Unnamed: 0	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfu
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6
mean	3248.000000	7.215307	0.339666	0.318633	5.443235	0.056034	
std	1875.666681	1.296434	0.164636	0.145318	4.757804	0.035034	
min	0.000000	3.800000	0.080000	0.000000	0.600000	0.009000	
25%	1624.000000	6.400000	0.230000	0.250000	1.800000	0.038000	

```
# Classes to predict from.

wine_quality['quality'].value_counts()
```

```
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```

```
# Finding more from the data

wine_quality2 = wine_quality.groupby('quality')
wine_quality2['quality'].value_counts()
```

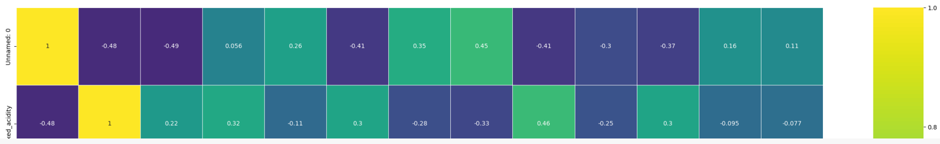
```
quality  quality
3        3            30
4        4           216
5        5          2138
6        6          2836
7        7          1079
8        8           193
9        9             5
Name: quality, dtype: int64
```

```
###! pip install seaborn
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(30, 30))
sns.heatmap(wine_quality.corr(),annot=True,cmap='viridis',linewidths=.5)
```

<AxesSubplot:>



```
wine_quality.columns
```

```
Index(['Unnamed: 0', 'fixed_acidity', 'volatile_acidity', 'citric_acid',
       'residual_sugar', 'chlorides', 'free_sulfur_dioxide',
       'total_sulfur_dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'],
      dtype='object')
```

```
from sklearn.model_selection import train_test_split
```

```
X = wine_quality.drop(columns="quality")
```

```
Y = wine_quality["quality"]
```

```
X_train, X_test, y_train, y_test = train_test_split(
  X,Y , random_state=104,test_size=0.25, shuffle=True)
```

```
from sklearn.linear_model import Lasso

from sklearn.feature_selection import SelectFromModel
```

```
model = SelectFromModel(Lasso(alpha=0.005,random_state=0))
```

```
model.fit(X_train,y_train)
```

▸ SelectFromModel
▸ estimator: Lasso
▸ Lasso

```
model.get_support()
```

```
array([ True,  True,  True, False,  True, False,  True,  True, False,
       False,  True,  True])
```

```
selected_features = X_train.columns[(model.get_support())]
```

```
selected_features
```

```
Index(['Unnamed: 0', 'fixed_acidity', 'volatile_acidity', 'residual_sugar',
       'free_sulfur_dioxide', 'total_sulfur_dioxide', 'sulphates', 'alcohol'],
      dtype='object')
```

```
X_train = X_train[selected_features]
```

```
X_train.head(2)
```

	Unnamed: 0	fixed_acidity	volatile_acidity	residual_sugar	free_sulfur_dioxide	total_sulfur_diox
6053	6053	6.6	0.18	17.3	17.0	14

```
X_test = X_test[selected_features]
```

```
X_test.head(2)
```

	Unnamed: 0	fixed_acidity	volatile_acidity	residual_sugar	free_sulfur_dioxide	total_sulfur_diox
5325	5325	4.8	0.17	2.9	22.0	11

```
X_train.drop(columns = "Unnamed: 0", inplace=True)
```

```
X_test.drop(columns = "Unnamed: 0", inplace=True)
```

```
from keras.models import Sequential
from keras.layers import InputLayer
from keras.layers import Dense
from keras.layers import Dropout
from keras.constraints import maxnorm

from  IPython import display
```

```
2023-11-17 16:26:41.821853: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to
```

```
import tensorflow as tf
```

```
X_train.shape
```

```
(4872, 7)
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
###! pip install optuna
```

```
import optuna
import sklearn
```

```
param_grid = {
    "learning_rate": [0.1, 0.2, 0.3, 0.4, 0.5],
    "max_depth": [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
    "subsample": [0.0, 1.0],
    "min_samples_leaf": [1, 2, 4],
    "min_samples_split": [2, 5, 10, 15],
    "n_estimators": [200, 400, 600, 800, 1000, 1200, 1300]
}
```

```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
def objective(trial):
    learning_rate = trial.suggest_float("learning_rate", 0.1, 0.5)
    max_depth = trial.suggest_int('max_depth', 10, 300)
    subsample = trial.suggest_float("subsample", 0.0, 1.0)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 4)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 15)
    n_estimators =  trial.suggest_int('n_estimators', 200, 1800)

    classf = GradientBoostingClassifier( learning_rate = learning_rate,
                                 max_depth = max_depth, subsample = subsample, min_samples_leaf = min_samples_leaf,
                                 min_samples_split = min_samples_split,n_estimators = n_estimators)


    #regr.fit(X_train, y_train)
    #y_pred = regr.predict(X_val)
    #return r2_score(y_val, y_pred)

    score = cross_val_score(classf, X_train, y_train, cv=5, scoring="accuracy")
    meanvalues = score.mean()

    return meanvalues
```

```
#Execute optuna and set hyperparameters
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=2)
```

```
[I 2023-11-17 16:26:43,374] A new study created in memory with name: no-name-aabe3d36-4d88-486a-872a-1e8f06aaaec2
[I 2023-11-17 16:51:35,730] Trial 0 finished with value: 0.4252789975254041 and parameters: {'learning_rate': 0.4204863554517012, 'max_depth': 71, 'subsa
[I 2023-11-17 17:07:34,175] Trial 1 finished with value: 0.37561459485073445 and parameters: {'learning_rate': 0.32684905577855206, 'max_depth': 68, 'sub
```

```
#Create an instance with tuned hyperparameters
optimised_gb = GradientBoostingClassifier(learning_rate = study.best_params['learning_rate'], max_depth = study.best_params['max_depth'], subsample = study.b
#learn
optimised_gb.fit(X_train,y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.4204863554517012, max_depth=71,
                           min_samples_split=5, n_estimators=1759,
                           subsample=0.6821624074620101)
```

```
optimised_gb.score(X_train,y_train)
```

```
0.6908866995073891
```

```
optimised_gb.score(X_test,y_test)
```

```
0.4307692307692308
```

```
y_pred_optuna = optimised_gb.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_absolute_error
print(classification_report(y_test,y_pred_optuna))
print(confusion_matrix(y_test,y_pred_optuna))
```

```
              precision    recall  f1-score   support

           3       0.02      0.20      0.04         5
           4       0.09      0.20      0.12        49
           5       0.55      0.44      0.49       546
           6       0.55      0.46      0.50       710
           7       0.33      0.39      0.36       268
           8       0.17      0.41      0.24        44
           9       0.00      0.00      0.00         3

    accuracy                           0.43      1625
   macro avg       0.24      0.30      0.25      1625
weighted avg       0.48      0.43      0.45      1625

[[  1   1   3   0   0   0   0]
 [  3  10  13  15   6   2   0]
 [ 20  53 242 163  57   9   2]
 [ 17  40 142 324 137  47   3]
 [  5   8  34  83 105  32   1]
 [  0   0   5   9  12  18   0]
 [  0   0   1   0   2   0   0]]
```

```
trial = study.best_trial
print('Accuracy: {}'.format(trial.value))
```

```
Accuracy: 0.4252789975254041
```

```
study.best_params
```

```
{'learning_rate': 0.4204863554517012,
 'max_depth': 71,
 'subsample': 0.6821624074620101,
 'min_samples_leaf': 1,
```

```
 'min_samples_split': 5,
 'n_estimators': 1759}
```

```
print("Best hyperparameters: {}".format(trial.params))
```

```
Best hyperparameters: {'learning_rate': 0.4204863554517012, 'max_depth': 71, 'subsample': 0.6821624074620101, 'min_samples_leaf': 1, 'min_samples_split':
```

```
y_pred_optuna = pd.DataFrame(y_pred_optuna)
```

```
y_pred_optuna = y_pred_optuna.rename(columns ={0: "Predict"} )
```

```
y_pred_optuna.value_counts()
```

```
Predict
6          594
5          440
7          319
4          112
8          108
3           46
9            6
dtype: int64
```

```
import pickle
```

```
with open("optimised_gb.pkl", "wb") as f:
    pickle.dump(optimised_gb, f)
```

Explainable AI

```
###!pip install shap
```

```
import shap
import matplotlib.pyplot as plt
```

```
explainer = shap.KernelExplainer(optimised_gb.predict_proba, X_train)
```

```
Using 4872 background data samples could cause slower run times. Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the background
```

```
shap_values = explainer.shap_values(X_test.iloc[0,:])
```

```
X_test.iloc[0,:]
```

```
fixed_acidity             4.80
volatile_acidity          0.17
residual_sugar            2.90
free_sulfur_dioxide      22.00
total_sulfur_dioxide    111.00
sulphates                 0.34
alcohol                  11.30
Name: 5325, dtype: float64
```

```
wine_quality.columns
```

```
Index(['Unnamed: 0', 'fixed_acidity', 'volatile_acidity', 'citric_acid',
       'residual_sugar', 'chlorides', 'free_sulfur_dioxide',
       'total_sulfur_dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'],
      dtype='object')
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((4872, 7), (1625, 7), (4872,), (1625,))
```

```
clf = DecisionTreeClassifier(random_state=0)
```

```
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
y_dt = clf.predict(X_test)
```

```
y_dt = pd.DataFrame(y_dt)
```

```
# compute SHAP values
explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(X_train)
```

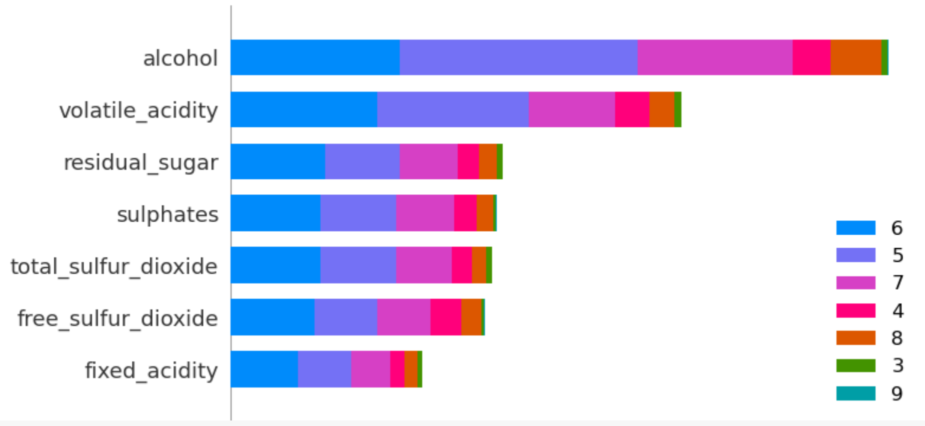
```
class_names = ['3', '4', '5', '6', '7', '8', '9']
```

SHAP SUMMARY PLOT

```
X_train.columns
```

```
Index(['fixed_acidity', 'volatile_acidity', 'residual_sugar',
       'free_sulfur_dioxide', 'total_sulfur_dioxide', 'sulphates', 'alcohol'],
      dtype='object')
```

```
shap.summary_plot(shap_values, X_train.values, plot_type="bar", class_names= class_names, feature_names = X_train.columns)
```




```
###! pip install xgboost==0.90
```

```
import xgboost
```

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(4872, 7) (1625, 7) (4872,) (1625,)
```

```
model_xgb = xgboost.XGBClassifier().fit(X = X_train,y = y_train)
```

```
# compute SHAP values
explainer = shap.Explainer(model_xgb, X_train)
```

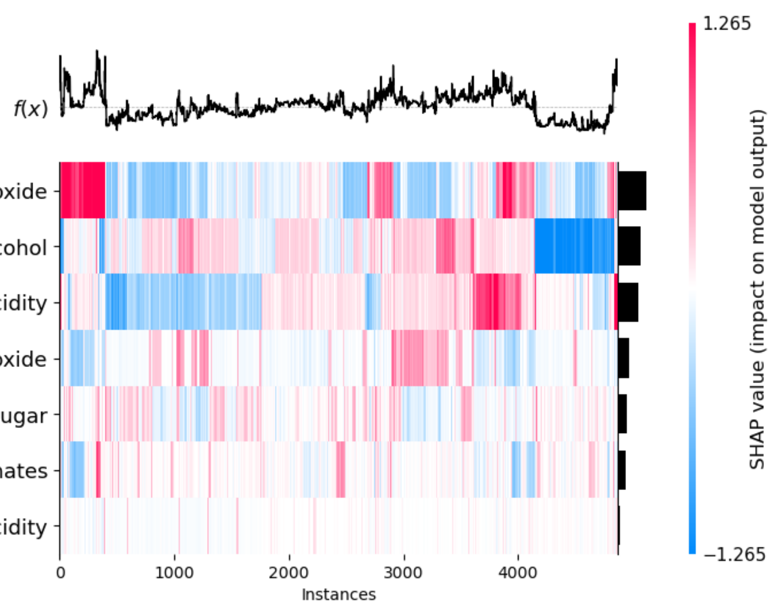
```
shap_values = explainer(X_train)
```

```
100%|===================| 33991/34104 [03:38<00:00]
```

HeatMap plot

This notebook is designed to demonstrate (and so document) how to use the shap.plots.heatmap function. It uses an XGBoost model trained on the classic UCI adult income dataset (which is a classification task to predict if people made over $50k annually in the 1990s).

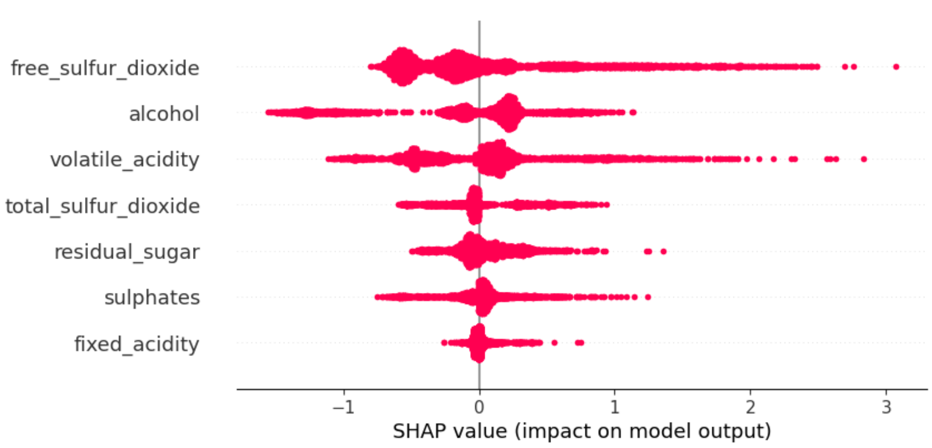
```
shap.plots.heatmap(shap_values[:,:,1])
```



beeswarm plot

This notebook is designed to demonstrate (and so document) how to use the shap.plots.beeswarm function. It uses an XGBoost model trained on the classic UCI adult income dataset (which is a classification task to predict if people made over $50k in the 1990s).

```
shap.plots.beeswarm(shap_values[:,:,1], color="shap_red")
```



The `violin summary plot` offers a compact representation of the distribution and variability of SHAP values for each feature. Individual violin plots are stacked by importance of the particular feature on model output (sum of the absolute values of the SHAP values per feature).

```
shap.plots.violin(shap_values[:,:,1], color="red")
```