Supplementary material (Python Code)

```
# This code is provided for understanding of the modeling example.
# As part of the research project, some contents were not shown in this material.

import pandas as pd
import numpy as np
import plotly
np.random.seed(0)
import matplotlib.pyplot as plt

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

plt.rcParams["figure.figsize"] = [2, 2]
plt.rcParams["font.size"] = 10

data = pd.read_csv('C:/scoliosis.csv') # ,sep=';')

data.shape
data.columns
data['Scoliosis'] = data['Scoliosis'].astype(int)
data['Scoliosis'].hist()

import numpy as np
import pandas as pd
import os
from sklearn import metrics

# Interpretable models
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import graphviz



X = data.drop(['Scoliosis'], axis=1)
y = data['Scoliosis']

lr = LogisticRegression(random_state=0)
##############   Performance metric
def plot_roc_curve(fprs, tprs):
    """Plot the Receiver Operating Characteristic from a list
    of true positive rates and false positive rates."""

    # Initialize useful lists + the plot axes.
    tprs_interp = []
    aucs = []
    mean_fpr = np.linspace(0, 1, 100)
    f, ax = plt.subplots(figsize=(14,10))

    # Plot ROC for each K-Fold + compute AUC scores.
    for i, (fpr, tpr) in enumerate(zip(fprs, tprs)):
        tprs_interp.append(np.interp(mean_fpr, fpr, tpr))
        tprs_interp[-1][0] = 0.0
        roc_auc = auc(fpr, tpr)
        aucs.append(roc_auc)
        ax.plot(fpr, tpr, lw=1, alpha=0.3,
                 label='ROC %d-fold (AUC = %0.2f)' % (i, roc_auc))

    # Plot the base line.
    plt.plot([0, 1], [0, 1], linestyle='--', lw=3, color='r',
             label='Base', alpha=.8)

    # Plot the mean ROC.
    mean_tpr = np.mean(tprs_interp, axis=0)
    mean_tpr[-1] = 1.0
    mean_auc = auc(mean_fpr, mean_tpr)
    std_auc = np.std(aucs)
    ax.plot(mean_fpr, mean_tpr, color='g',
             label=r'Mean ROC (AUC = %0.2f $₩pm$ %0.2f)' % (mean_auc, std_auc),
             lw=4, alpha=.8)

    # Plot the standard deviation around the mean ROC.
    std_tpr = np.std(tprs_interp, axis=0)
    tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
    tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
    ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                     label=r'$₩pm$ 1 std. dev.')

    # Fine tune and show the plot.
    ax.set_xlim([-0.05, 1.05])
    ax.set_ylim([-0.05, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic')
```

```
    ax.legend(loc="lower right")
    plt.show()
    return (f, ax)

def compute_roc_auc(index):
    y_predict = LR.predict_proba(X.iloc[index])[:,1]
    fpr, tpr, thresholds = roc_curve(y.iloc[index], y_predict)
    auc_score = auc(fpr, tpr)
    return fpr, tpr, auc_score

cv = StratifiedKFold(n_splits=5, random_state=123, shuffle=True)
results = pd.DataFrame(columns=['training_score', 'test_score'])
fprs, tprs, scores = [], [], []

for (train, test), i in zip(cv.split(X, y), range(5)):
    LR.fit(X.iloc[train], y.iloc[train])
    _, _, auc_score_train = compute_roc_auc(train)
    fpr, tpr, auc_score = compute_roc_auc(test)
    scores.append((auc_score_train, auc_score))
    fprs.append(fpr)
    tprs.append(tpr)

plot_roc_curve(fprs, tprs);
pd.DataFrame(scores, columns=['AUC Train', 'AUC Test'])
############## End of performance metric

# Use KFold
kf = KFold(n_splits=5, shuffle=True, random_state=1111)

# Create splits
```

```
splits = kf.split(X)

# Print the number of indices
for train_index, val_index in splits:
    print("Number of training indices: %s" % len(train_index))
    print("Number of validation indices: %s" % len(val_index))

from sklearn.model_selection import KFold
kf = KFold(n_splits=5, shuffle=False).split(range(25))

# print the contents of each training and testing set
print('{} {:^61} {}'.format('Iteration', 'Training set observations', 'Testing set observations'))
for iteration, data in enumerate(kf, start=1):
    print('{:^9} {} {:^25}'.format(iteration, data[0], str(data[1])))

from sklearn.model_selection import cross_val_score

# K-fold cross-validation with models
lr = LogisticRegression(random_state=0)
scores = cross_val_score(lr, X, y, cv=5, scoring='accuracy')
print(scores)

import matplotlib.pyplot as plt
%matplotlib inline

# plot the value of the cross-validated accuracy (y-axis)
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for lr')
plt.ylabel('Cross-Validated Accuracy')
```

```
# Convert categorical variables into dummy/indicator variables
train_processed = pd.get_dummies(train)
test_processed = pd.get_dummies(test)

# Filling Null Values
train_processed = train_processed.fillna(train_processed.mean())
test_processed = test_processed.fillna(test_processed.mean())

# Create X_train,Y_train,X_test for a specific set for the train and test sets split
X_train = train_processed.drop(['Scoliosis'], axis=1)
y_train = train_processed['Scoliosis']

X_test   = test_processed.drop(['Scoliosis'], axis=1)
y_test   = test_processed['Scoliosis']

# Display
print("Processed DataFrame for Training : Scoliosis is the Target, other columns are features.")
display(train_processed.head())

from sklearn.feature_selection import mutual_info_classif # Mutual information for a discrete target

#Set a random seed for the notebook so that individual runs of the notebook yield the same results
randSeed = 99 #changing this value will potentially change the models and results due to stochastic elements of the pipeline.
np.random.seed(randSeed)

mi_results = mutual_info_classif(X_train, y_train, random_state=randSeed)

#Present results
header = train.columns.tolist()
```

```
features = header[0:len(header)-1]
names_scores = {'Names':features, 'Scores':mi_results}
ns = pd.DataFrame(names_scores)
ns = ns.sort_values(by='Scores')
ns #Report sorted feature scores

#Visualize sorted feature scores
ns['Scores'].plot(kind='barh',figsize=(5,8))
plt.ylabel('Parameters')
plt.xlabel('Mutual Information Score')
plt.yticks(np.arange(len(features)), ns['Names'])
plt.title('Mutual Information of the parameters')

import lime
import lime.lime_tabular

lr.fit(X_train, y_train)
predict_fn_rf = lambda x: lr.predict_proba(x).astype(float)
X = X_train.values
choosen_instance = X_test.loc[[3]].values[0]
exp = explainer.explain_instance(choosen_instance, predict_fn_rf,num_features=20)
exp.show_in_notebook(show_table=True, show_all=False)
```