```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Import support vector regressor algorithm
from sklearn.svm import SVR
from sklearn.linear_model import Ridge, Lasso
from sklearn.preprocessing import StandardScaler, LabelEncoder
# Import modelling methods
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold, cross_val_score
# Import the model performance evaluation metrics
from sklearn import metrics
# Import Adaboost, Gradient Boost, Random Forest and Stacking algorithm
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingRegressor, RandomForestRegressor, StackingRegressor
import warnings
warnings.filterwarnings('ignore')
from sklearn.tree import DecisionTreeClassifier, plot_tree
# Instead of load_boston, use fetch_california_housing:
from sklearn.datasets import fetch_california_housing   # to import california housing dataset
# to visualize decision boundaries
import graphviz
import xgboost as xgb
from xgboost import XGBRegressor
# Load the California housing dataset
housing = fetch_california_housing()
# Access the data and target variables
```

```
X = housing.data

y = housing.target

df = pd.read_csv('/content/indian_liver_patient.csv')

df.head()

df.isnull().sum()

# Drop missing values

df1 = df.dropna()

df1.isnull().any()

# Visualize correlation matrix

fig, ax = plt.subplots(figsize=(7,7))

# Select only numerical features for correlation calculation

numerical_df = df1.select_dtypes(include=['number'])

sns.heatmap(abs(numerical_df.corr()), annot=True, square=True, cbar=False, ax=ax,
linewidths=0.25)

# Drop correlated features

df2 = df1.drop(columns= ['Direct_Bilirubin', 'Alamine_Aminotransferase',
'Total_Protiens'])

# Drop correlated features

# Corrected column name from 'Total_Protiens' to 'Total_Proteins'

df2 = df1.drop(columns=['Direct_Bilirubin', 'Alamine_Aminotransferase',
'Total_Protiens'])

# Drop correlated features

# Corrected column name from 'Total_Protiens' to 'Total_Proteins'

# The original column name was likely misspelled as 'Total_Protiens'

# This line now uses the correct column name 'Total_Proteins' if it exists,

# otherwise it uses 'Total_Protiens'

df2 = df1.drop(columns=['Direct_Bilirubin', 'Alamine_Aminotransferase',

df1.columns[df1.columns.str.contains('Total_Pro')]

.tolist()[0]])
```

```
# Drop correlated features
# Corrected column name from 'Total_Protiens' to 'Total_Proteins'
# Check for correct column names in df1.columns
print(df1.columns)
# Adjust column names in drop method if needed.
# Example: If 'Total_Proteins' is actually 'Total_Protiens', use the original name
df2 = df1.drop(columns=['Direct_Bilirubin', 'Alamine_Aminotransferase', 'Total_Protiens']) **********
df2['Dataset'] = df2['Dataset'].replace(1,0)
df2['Dataset'] = df2['Dataset'].replace(2,1)
print('How many people have disease:', '\n', df2.groupby('Gender')[['Dataset']].sum(), '\n')
print('How many people participated in the study:', '\n', df2.groupby('Gender')[['Dataset']].count())
print('Percentage of people with the disease depending on gender:')
df2.groupby('Gender')[['Dataset']].sum()/ df2.groupby('Gender')[['Dataset']].count()
# defining the X and y variables
X = df2[['Gender', 'Total_Bilirubin','Alkaline_Phosphotase','Aspartate_Aminotransferase','Albumin','Albumin_and_Globulin_Ratio']]
y = pd.Series(df2['Dataset'])
labelencoder = LabelEncoder()
X['Gender'] = labelencoder.fit_transform(X['Gender'])
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.transform(x_test)
 ADB = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),
                         n_estimators=125,
                         learning_rate = 0.6,
```

```
                    random_state=42)
ADB.fit(X_train, y_train)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# calculating model evaluation metrics using cross_val_score like accuracy, R2 score, etc.
n_scores = cross_val_score(ADB, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
('Accuracy: %.3f' % (np.mean(n_scores)*100))
labels = ADB.predict(X_test)
matrix = metrics.confusion_matrix(y_test, labels)
# creating a heat map to visualize confusion matrix
sns.heatmap(matrix.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label');
 logit_roc_auc = metrics.roc_auc_score(y_test, labels)
fpr, tpr, thresholds = metrics.roc_curve(y_test, ADB.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='(area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
 from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
print(housing.keys())
print("shape of dataset",housing.data.shape)
 from sklearn.datasets import fetch_california_housing
```

```
# Load the California housing dataset
housing = fetch_california_housing()
# Access the data and target variables
data = housing.data
target = housing.target
# Print the keys of the dataset
print(housing.keys())
# Print the shape of the dataset
print("shape of dataset", data.shape)
 print(boston.feature_names)
df = pd.DataFrame(boston.data)
df.columns = boston.feature_names
 df.head()
df['PRICE'] = boston.target
df.info()
 X, y = df.iloc[:,:-1],df.iloc[:,-1]
xtrain, xtest, ytrain, ytest=train_test_split(X, y, random_state=12, test_size=0.15)
# with new parameters
gbr1 = GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse',
n_estimators=600,
max_depth=5,
learning_rate=0.01,
min_samples_split=4)
# with default parameters
gbr = GradientBoostingRegressor()
# fit with default parameters
gbr.fit(xtrain, ytrain)
ypred = gbr.predict(xtest)
```

```
# calculating Mean Squared Error
mse = metrics.mean_squared_error(ytest,ypred)
# mse for default model
print("MSE: %.2f" % mse)
# fit by passing hyperparameters
gbr1.fit(xtrain, ytrain)
ypred1 = gbr1.predict(xtest)
# calculating Mean Squared Error
mse1 = metrics.mean_squared_error(ytest, ypred1)
# mse for regularized model
print("MSE: %.2f" % mse1)
 x_ax = range(len(ytest))
plt.scatter(x_ax, ytest, s=5, color="blue", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
 xgb_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3,
learning_rate = 0.1,
max_depth = 5, alpha = 10, n_estimators = 10)
 xgb_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3,
learning_rate = 0.1,
max_depth = 5, alpha = 10, n_estimators = 10)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
xgb_reg.fit(X_train,y_train)
y_pred = xgb_reg.predict(X_test)
mse2 = metrics.mean_squared_error(y_test, y_pred)
print("MSE: %f" % (mse))
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
xgb_reg.fit(X_train,y_train)
```

```
y_pred = xgb_reg.predict(X_test)
mse2 = metrics.mean_squared_error(y_test, y_pred)
print("MSE: %f" % (mse))
 xgb = XGBRegressor()
rf = RandomForestRegressor(n_estimators=400, max_depth=5, max_features=6
ridge = Ridge()
 lasso = Lasso()
# Instead of assigning new column names, ensure X_test has the correct columns
# Assuming original_column_names is a list containing the 8 original column names
original_column_names = X_train.columns  # Get column names from training data
X_test = X_test[[c for c in original_column_names if c in X_test.columns]] # Select only the columns present in both X_train and X_te
# OR if you know the original column names:
# original_column_names = ['original_col1', 'original_col2', ..., 'original_col8']
# X_test = X_test[original_column_names]
pred = reg.predict(X_test)
score = metrics.r2_score(y_test, pred)
print(score)
```