Variables

CRIM : per capita crime rate by town

ZN : proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS : proportion of non-retail business acres per town

CHAS : Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

NOX : nitric oxides concentration (parts per 10 million)

RM : average number of rooms per dwelling

AGE : proportion of owner-occupied units built prior to 1940

AGE : proportion of owner-occupied units built prior to 1940

DIS : weighted distances to five Boston employment centres

RAD : index of accessibility to radial highways

TAX : full-value property-tax rate per $10,000

PTRATIO : pupil-teacher ratio by town

B : 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

LSTAT : % lower status of the population

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# from sklearn import datasets
# boston = datasets.load_boston()
# features = pd.DataFrame(boston.data,
# columns=boston.feature_names)
# targets = boston.target
```

```
data_url = "/content/boston.csv"
df = pd.read_csv(data_url)
df.head()
```

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
del df['Unnamed: 0']
```

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
#$Simple Ensemble Techniques
# Max Voting
# Averaging
```

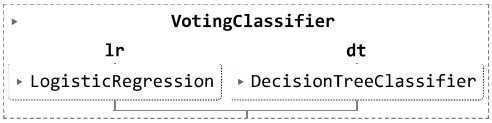
```
# Weighted Averaging

# Major Techniques
# Bagging
# Boosting
# Stacking


df.isnull().sum()
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
Price      0
dtype: int64
```

```
#We can also this model
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
X = df.drop(['Price'],axis=1)
y = df.Price
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=32)
model1 = LogisticRegression(random_state=1)
model2 = DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='hard')
# model.fit(X_train,y_train)
# model.score(X_test,y_test)
model
```




```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV


X = df.drop(['Price'],axis=1)
y = df.Price
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=32)
model = RandomForestRegressor()
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

```
0.8739505920595765
```

```
random_grid = {'bootstrap': [True, False],
               'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
               'max_features': ['auto', 'sqrt'],
               'min_samples_leaf': [1, 2, 4],
               'min_samples_split': [2, 5, 10],
               'n_estimators': [130, 180, 230],
               'criterion': ['squared_error', 'absolute_error', 'friedman_mse', 'poisson']}


# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(RandomForestRegressor(), param_distributions = random_grid,scoring='accuracy',cv=5,verbose=3)
# Fit the random search model
rf_random.fit(X_train,y_train)
```

```
rf_random.best_params_
```

```
{'n_estimators': 130,
 'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': 50,
 'criterion': 'absolute_error',
 'bootstrap': True}
```

```
rf_random.best_estimator_
```

```
RandomForestRegressor(bootstrap=False, max_depth=40, max_features='sqrt',
                      min_samples_split=5, n_estimators=130)
```

```
model = RandomForestRegressor(bootstrap=False,criterion = 'absolute_error', max_depth=40, max_features='sqrt',min_samples_split=5, n_estimato
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

```
0.8882944428271824
```

```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test.head(1))
y_pred
```

```
array([16.075])
```