```
import pandas as pd
import numpy as np # Import the numpy library
url = "https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv"
df = pd.read_csv(url)
```

```
df.info()
df.describe()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
                          0
Pregnancies               0
Glucose                   0
BloodPressure             0
SkinThickness             0
Insulin                   0
BMI                       0
DiabetesPedigreeFunction  0
Age                       0
Outcome                   0

dtype: int64
```

```
import numpy as np  # Add this line
cols_to_replace = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
for col in cols_to_replace:
    df[col] = df[col].replace(0, np.nan)
    df[col].fillna(df[col].median(), inplace=True)
```

```
<ipython-input-10-7f0fc1778cfe>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
```

```
from sklearn.preprocessing import StandardScaler
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
X_scaled = StandardScaler().fit_transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, stratify=y, random_state=42)
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Use estimator instead of base_estimator
```

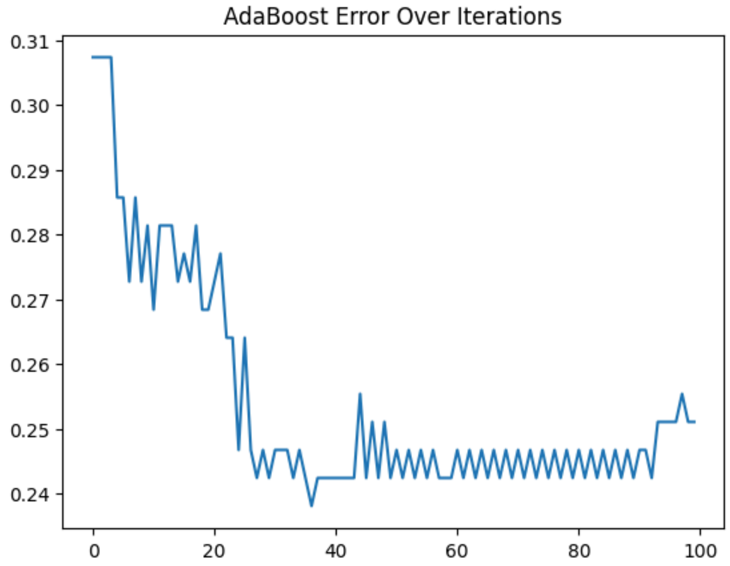
```
model_bag = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=100)
model_bag.fit(X_train, y_train)
y_pred_bag = model_bag.predict(X_test)
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Use estimator instead of base_estimator if using older version of scikit-learn
model_ada = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),  # Pass the base estimator directly
    n_estimators=100,
    learning_rate=1.0
)
model_ada.fit(X_train, y_train)
y_pred_ada = model_ada.predict(X_test)
```

```
import matplotlib.pyplot as plt
errors = [1 - score for score in model_ada.staged_score(X_test, y_test)]
plt.plot(errors)
plt.title("AdaBoost Error Over Iterations")
```

```
Text(0.5, 1.0, 'AdaBoost Error Over Iterations')
```



```
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(n_estimators=100, max_depth=5, oob_score=True)
model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)
```

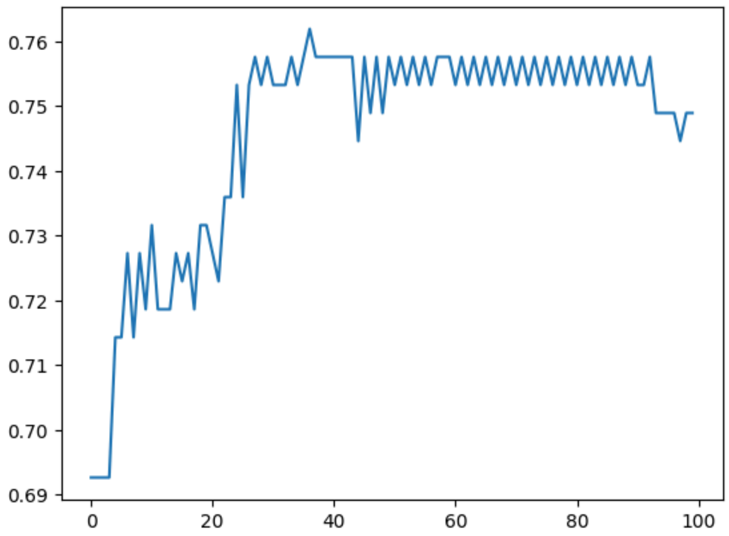
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

base_models = [('svm', SVC(probability=True)), ('knn', KNeighborsClassifier())]
meta_model = LogisticRegression()

stack_model = StackingClassifier(estimators=base_models, final_estimator=meta_model, cv=5)
stack_model.fit(X_train, y_train)
y_pred_stack = stack_model.predict(X_test)
```

```
ada = AdaBoostClassifier(n_estimators=100)
ada.fit(X_train, y_train)
plt.plot(list(ada.staged_score(X_test, y_test)))
```

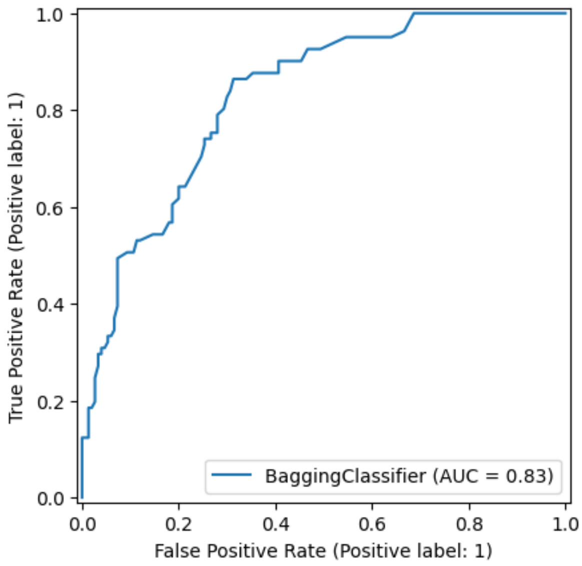
```
[<matplotlib.lines.Line2D at 0x79c4a70a7d90>]
```



```
feature_imp = pd.Series(model_rf.feature_importances_, index=X.columns).nlargest(5)
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_estimator(model_bag, X_test, y_test)
# Repeat for others
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x79c4a70d1d90>
```



```
import time
start = time.time()
# Replace 'model' with the actual model you want to use, for example:
model = model_rf  # or model_bag, model_ada, stack_model
model.fit(X_train, y_train)
print("Training Time:", time.time() - start)
```

```
Training Time: 0.40747618675231934
```