CSE 587 **Fall 2022**

Project Phase #2

Abhi Vijayakumar and **Harshitha Siddaramaiah**

Problem Statement: Prediction of Heart Disease using patient examination results.

This project focuses on analyzing the patient examination results and whether engaging in regular physical activity, maintaining a healthy diet and weight, managing stress, avoiding smoking and vaping, and getting quality sleep each night can all reduce the risk of heart disease and help people live longer, healthier lives.

Project Goal:

About half the population of Americans have at least 1 of the 3 key risk factors for heart disease: high blood pressure, high cholesterol, and smoking. Other key indicators include diabetic status, obesity (high BMI), not getting enough physical activity or drinking too much alcohol. Our project aims at detecting those factors that have the greatest impact on heart disease. The outcome of this project could serve as important information in healthcare in order to raise awareness and

the actions we can all take to prevent heart diseases. Our project can be considered as a starting point that could potentially help bigger research projects capable of investing billions of dollars in preventing, detecting, treating cardiovascular conditions, and to develop new programs to alleviate heart health disparities. Overall, in this project we are doing our bit to ensure a healthier future for humanity.

A brief about Phase #1:

In phase #1, the data was preprocessed and the following steps were completed.

1. Checking of Null values
2. Binary data encoding
3. Categorical encoding
4. Checking any Duplicate entries in the data
5. Object datatype to String datatype
6. Adding a BMI_Normalcy column
7. Checked for any whitespaces in string datatype column
8. Rounding float values to 2 decimal places
9. Removing data outliers
10. Remove irrelevant features

Post the completion of preprocessing, we performed the Exploratory Data Analysis (EDA) to know the most important columns and remove the irrelevant features from the data. From the EDA, the below inferences were derived.

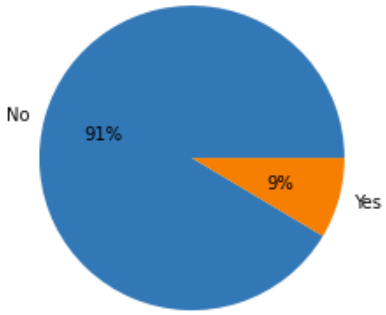
- The chosen data is unbalanced i.e. around 91% of the data had people who do not suffer from Heart disease.
- People who have higher BMI are more prone to have heart disease especially if the BMI is above 25.5.
- The older the people (above 65 years), the more they are susceptible to get a heart disease.
- From the analysis above, it came out as drinking alcohol does not have an impact on heart disease.
- Diabetic people and those above the age of 60, have more chances of suffering from a heart disease.
- People who smoke are more prone to get a heart disease.
- On comparing male to female ratio, the BMI of females is less than that of males. Also the Mental Health, Physical Activity of females are slightly higher than that of males. Since those contribute to Heart Disease, females are less susceptible towards getting a Heart disease when compared to males.
- People who suffer from kidney disease have a high risk of getting heart diseases.
- Those who are suffering from skin cancer have high chances of getting heart diseases.
- Stroke is highly associated with Heart Disease, and people who had a stroke in the past are prone to get a heart disease.

- From the correlation matrix, it is seen that Physical activity, Alcohol drinking, race and sleep time doesn't have an effect on Heart disease.

Phase #2:

The below pie chart indicates that the ratio of people who have and who do not have a heart disease is very imbalanced. In the 319,795 observations in our data set, only 9% of the people actually have a heart disease. Since the chosen data is unbalanced i.e. 91% of the people in the data do not suffer from heart disease and only 9% have a heart disease, we need to balance the data.

```
In [9]: plt.pie(y.value_counts(),autopct='%1.0f%%',labels=['No',"Yes"])

Out[9]: ([<matplotlib.patches.Wedge at 0x1cf60a49ac0>,
          <matplotlib.patches.Wedge at 0x1cf60a62100>],
         [Text(-1.0604682899620466, 0.2922447706717311, 'No'),
          Text(1.0604682626001118, -0.2922448699599368, 'Yes')],
         [Text(-0.5784372490702072, 0.15940623854821695, '91%'),
          Text(0.5784372341455155, -0.15940629270542006, '9%')])
```



If the algorithms are trained with the original data, it is highly unlikely to create models with high accuracy as the prediction would diagnose most of the results as false negatives since majority of the data has Heart Disease as "No."

Sampling:

In order to resolve the above issue, one of the approaches is to ensure that the percentage of the positive and negative cases is as balanced as possible. In this case, we can either delete the negative samples or generate new synthetic samples. We decided to do the latter.

SMOTE sampling technique was carried out for the below mentioned machine learning models:

1. Logistic Regression
2. KNN
3. Decision tree

4. Random Forest
5. ADABoost
6. Bagging

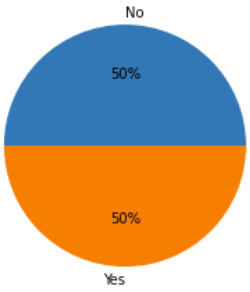
Synthetic Minority Over-sampling Technique (SMOTE):

In SMOTE, the minority class is over-sampled by introducing synthetic instances where each minority class sample is taken. The generated data are inserted along the line segments joining some of the k-nearest neighbors of the minority class. Neighbors are randomly chosen from k-nearest neighbors depending upon the amount of over-sampling that is required.

SMOTE

```
In [7]:  over = SMOTE(k_neighbors=2000)
         x_smote, y_smote = over.fit_resample(x, y)
         print(y_smote.value_counts())

         0    292422
         1    292422
         Name: HeartDisease, dtype: int64
```

```
In [12]: plt.pie(y_smote.value_counts(),autopct='%1.0f%%',labels=['No',"Yes"])

Out[12]: ([<matplotlib.patches.Wedge at 0x2b2a938a5b0>,
           <matplotlib.patches.Wedge at 0x2b2a938abb0>],
          [Text(6.735557395310444e-17, 1.1, 'No'),
           Text(-2.0206672185931328e-16, -1.1, 'Yes')],
          [Text(3.6739403974420595e-17, 0.6, '50%'),
           Text(-1.1021821192326178e-16, -0.6, '50%')])
```



SMOTE enhances the data to 50 - 50. When the results were carried out in random undersampling, the data was still unbalanced (67 - 33). Hence, we considered only SMOTE sampling and did not go ahead with undersampling technique.

Machine Learning Models:

Our problem statement is to predict if a person/patient is at the risk of getting a heart disease or not, based on the patient's examination results. This falls under **"Classification"** type of problems where the machine learning algorithms' task is to classify the patient's data in order to label whether the patient is **"at the risk of getting a heart disease"** or **"not"**. We also compared the models with SMOTE sampling and without sampling the data to know which yields the best result.

Training and Test Data Split:

The total dataset after sampling has 584,844 samples. The training and test data is divided into 75 - 25 which will be used in the below discussed algorithms to identify which model works the best for our problem statement.

```
In [7]:  x = df.drop('HeartDisease',axis = 1)
         y = df['HeartDisease']
```

```
[10]:  X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.25, random_state = 42)
```

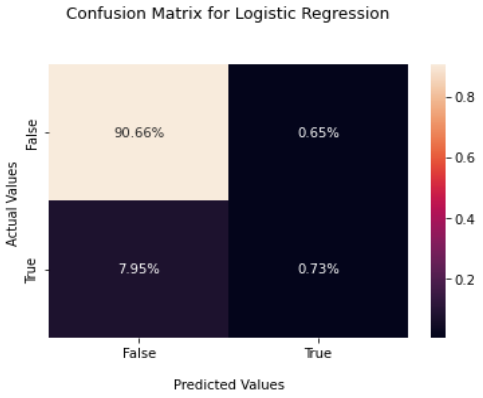
1. Logistic Regression:

We chose to explore Logistic regression as it is a great tool for analyzing binary and categorical data, allowing us to perform a contextual analysis to understand the relationships between the variables, test for differences, estimate effects, make predictions, and plan for future scenarios. It also finds an equation that predicts an outcome for a binary variable, Y, from one or more response variables, X. In our case Y would be whether a person is at the risk of getting a heart disease or not and X would be the patient's examination details. It is also used to describe the data and the relationship between one dependent variable and one or more independent variables. From our problem statement, we can identify whether a person's lifestyle or history of other diseases would have an impact on heart disease.

Without Sampling:

In [19]:
```
classifier = LogisticRegression(solver='lbfgs', max_iter=1000)
classifier.fit(X_train, Y_train)
Logistic_pred = classifier.predict(X_test)
Logistic_matrix = confusion_matrix(Y_test, Logistic_pred, labels=[0, 1])
```

In [20]:
```
ax = sns.heatmap(Logistic_matrix/np.sum(Logistic_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Logistic Regression\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



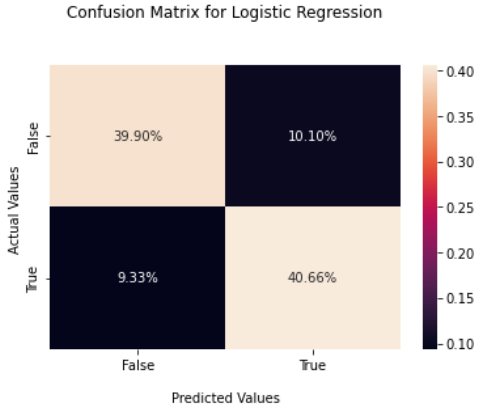
In [21]:
```
print(classification_report(Y_test, Logistic_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     73004
           1       0.53      0.08      0.15      6945

    accuracy                           0.91     79949
   macro avg       0.72      0.54      0.55     79949
weighted avg       0.89      0.91      0.88     79949
```

After SMOTE sampling:

In [22]:
```
classifier = LogisticRegression(solver='lbfgs', max_iter=1000)
classifier.fit(X_train, Y_train)
Logistic_pred = classifier.predict(X_test)
Logistic_matrix = confusion_matrix(Y_test, Logistic_pred, labels=[0, 1])
```

In [23]:
```
ax = sns.heatmap(Logistic_matrix/np.sum(Logistic_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Logistic Regression\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [24]:
```
print(classification_report(Y_test, Logistic_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.80      0.80     73112
           1       0.80      0.81      0.81     73099

    accuracy                           0.81    146211
   macro avg       0.81      0.81      0.81    146211
weighted avg       0.81      0.81      0.81    146211
```

By running the Logistic regression on sampled and unsampled data the accuracy is 81% for the former and 91% for the latter case. But the precision is higher in terms of detecting heart disease in a sampled data set than that of an unsampled data set. However the false positives and false negatives are higher (around 9% and 10%) when the data gets sampled.

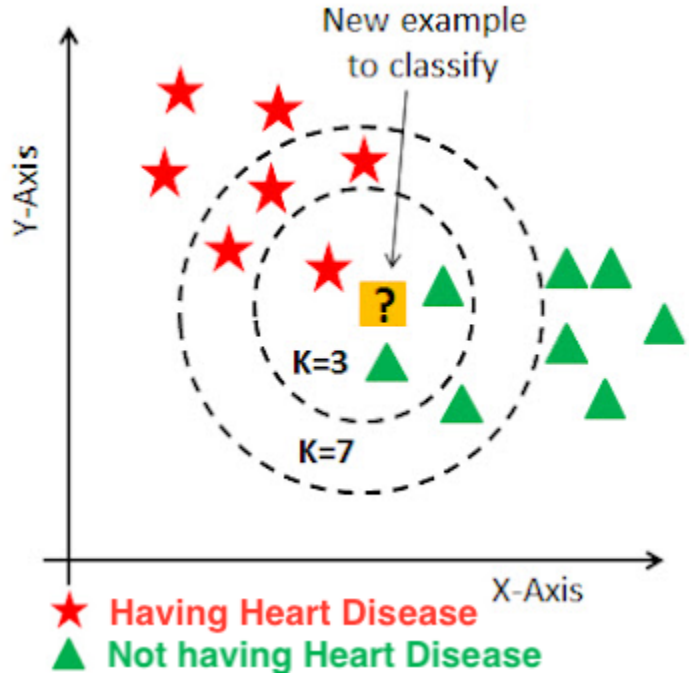
2. K-Nearest Neighbours (KNN):

KNN is a supervised machine learning algorithm that can be used for both classification and regression tasks. If it is used for classification purposes, the mode of the closest observations will serve for prediction. If it is used for regression tasks, the predictions will be based on the mean or median of the K closest observations.

KNN classifies the new data points based on the similarity measure of the earlier stored data points. Based on what group the data points nearest to it belong to it estimates the likelihood of the data point becoming a member of one group or another. In our use case, the two groups would be:

1. Group having a Heart Disease.

2. Group not having a Heart Disease.



The KNN algorithm is quite accessible and easy to understand. For an observation that's not in the dataset, the algorithm will simply look for the K number of instances defined as similar based on the closest perimeter to that observation. Any data point falls under a specific group if it's close enough to it.

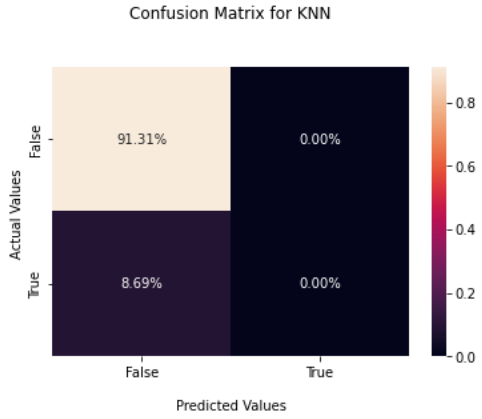
The main advantage of KNN over other algorithms is that KNN can be used for multiclass classification. Therefore if the data consists of more than two labels or in simple words if you are required to classify the data in more than two categories then KNN can be a suitable algorithm.

Before Sampling:

In [32]:
```
knn = KNeighborsClassifier(n_neighbors = 283)
knn.fit(X_train, Y_train)
print(knn.score(X_test, Y_test))
knn_pred = knn.predict(X_test)
knn_matrix = confusion_matrix(Y_test, knn_pred, labels=[0, 1])
```

```
0.9131321217276014
```

In [33]:
```
ax = sns.heatmap(knn_matrix/np.sum(knn_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for KNN\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [34]:
```
print(classification_report(Y_test, knn_pred))
```

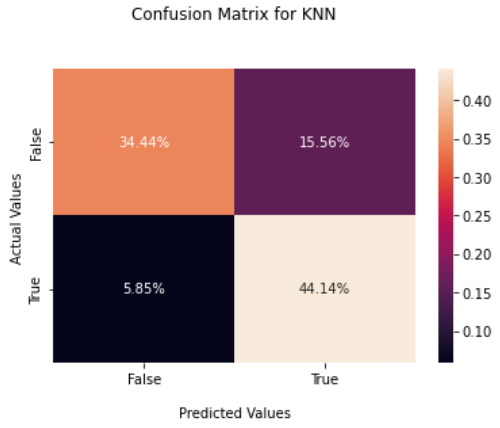
```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     73004
           1       0.00      0.00      0.00      6945

    accuracy                           0.91     79949
   macro avg       0.46      0.50      0.48     79949
weighted avg       0.83      0.91      0.87     79949
```

SMOTE:

In [16]:
```
knn = KNeighborsClassifier(n_neighbors = 381)
knn.fit(X_train, Y_train)
print(knn.score(X_test, Y_test))
knn_pred = knn.predict(X_test)
knn_matrix = confusion_matrix(Y_test, knn_pred, labels=[0, 1])
```

```
0.9131321217276014
```

In [20]:
```
ax = sns.heatmap(knn_matrix/np.sum(knn_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for KNN\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [21]:
```
print(classification_report(Y_test, knn_pred))
```

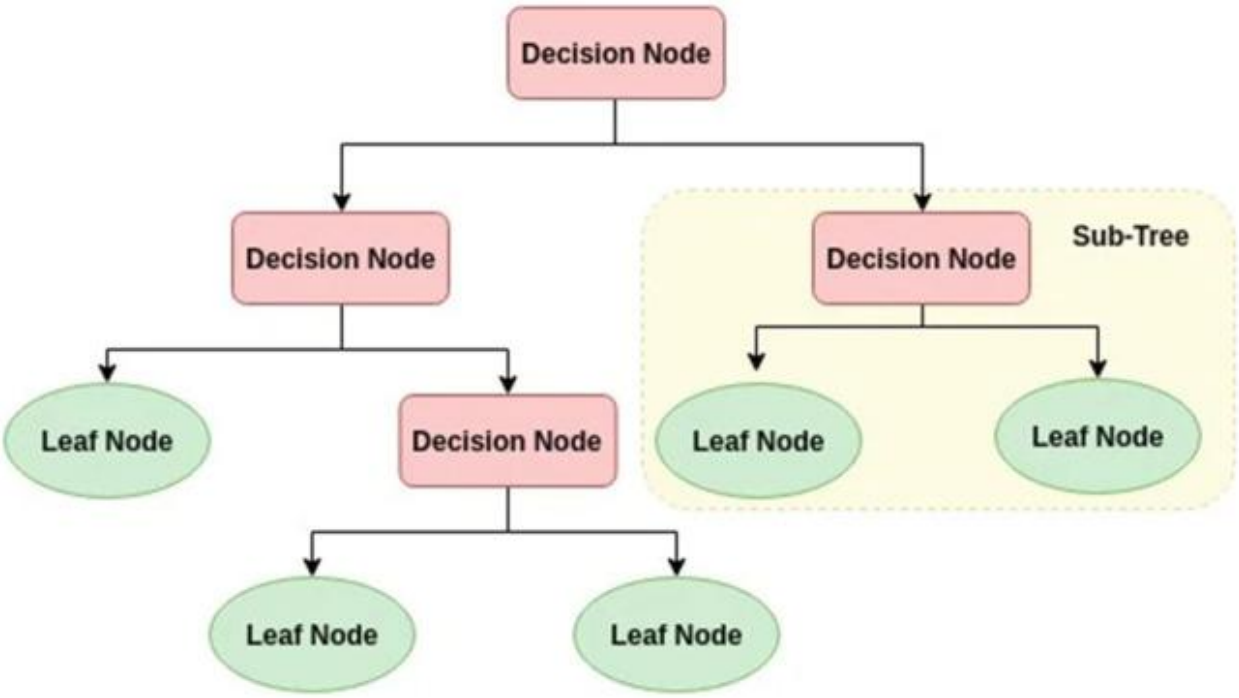
```
              precision    recall  f1-score   support

           0       0.85      0.69      0.76     73112
           1       0.74      0.88      0.80     73099

    accuracy                           0.79    146211
   macro avg       0.80      0.79      0.78    146211
weighted avg       0.80      0.79      0.78    146211
```

The accuracy for KNN is around 91% for unsampled data. But the algorithm fails in determining the false positives and checking if the person is suffering from heart disease or not as the f1 score is almost 0. Whereas when the algorithm is carried out on SMOTE data set then the accuracy is around 79%, the false positive is greatly reduced to around 6%. But the precision of the model i.e. if the person has a heart disease is only 74%.

3. Decision Tree Classification Algorithm:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision Tree, there are two nodes: Decision Node and Leaf Node, as shown in the below image. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.



The decisions or the tests are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. In order to build this decision tree, we use something called the CART algorithm, which stands for Classification and Regression Tree algorithm.

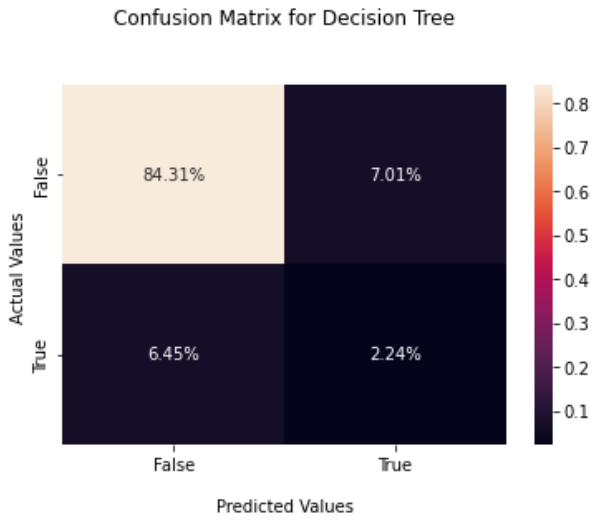
A decision tree simply asks a question, and based on the answer (Yes/No), it further splits the tree into subtrees. For our use case, we would be checking if a person smokes or drinks alcohol will impact him/her in getting a heart disease or if a person's age, sex, race or any other pre-existing medical condition can impact him/her in getting a heart disease.

Before Sampling:

In [22]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, Y_train)
dtc_pred = dtc.predict(X_test)
dtc_matrix = confusion_matrix(Y_test, dtc_pred, labels=[0, 1])
dtc.score(X_test, Y_test)
```

Out[22]: 0.865476741422657

In [23]:
```
ax = sns.heatmap(dtc_matrix/np.sum(dtc_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Decision Tree\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [24]:
```
print(classification_report(Y_test, dtc_pred))
```

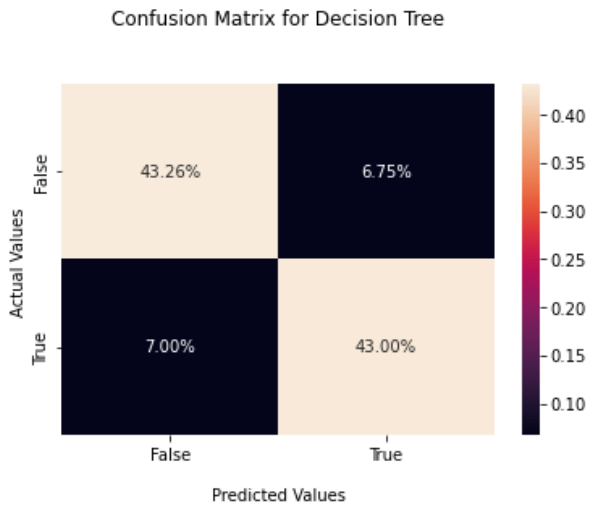
```
              precision    recall  f1-score   support

           0       0.93      0.92      0.93     73004
           1       0.24      0.26      0.25      6945

    accuracy                           0.87     79949
   macro avg       0.59      0.59      0.59     79949
weighted avg       0.87      0.87      0.87     79949
```

SMOTE:

In [25]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, Y_train)
dtc_pred = dtc.predict(X_test)
dtc_matrix = confusion_matrix(Y_test, dtc_pred, labels=[0, 1])
dtc.score(X_test, Y_test)
```

Out[25]: 0.8625479615076841

In [26]:
```
ax = sns.heatmap(dtc_matrix/np.sum(dtc_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Decision Tree\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [27]:
```
print(classification_report(Y_test, dtc_pred))
```

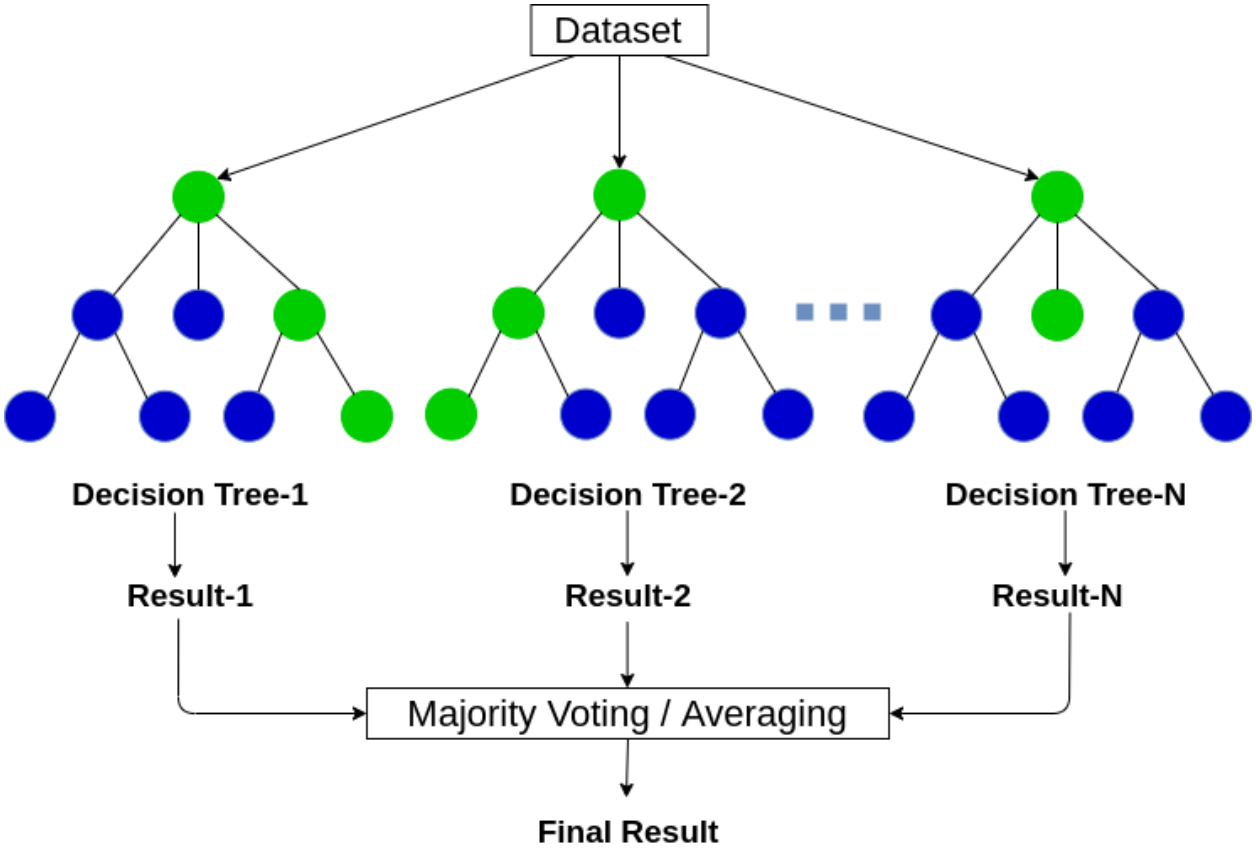
```
              precision    recall  f1-score   support

           0       0.86      0.87      0.86     73112
           1       0.86      0.86      0.86     73099

    accuracy                           0.86    146211
   macro avg       0.86      0.86      0.86    146211
weighted avg       0.86      0.86      0.86    146211
```

Through the Decision Tree, the accuracy achieved is around 86% when the data is sampled. Also, the false positives are reduced to 7% when the data is sampled and 6.45% when the algorithm was run on unsampled data in comparison with Logistic Regression. The decision tree algorithm is quite easy to understand and interpret. But often, a single tree is not sufficient for

producing effective results. Decision tree is also prone to overfitting. This is why we thought of using the Random Forest model next.

4. Random Forest:

Random Forest is a tree based algorithm that establishes the outcome based on the predictions of multiple decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome. It is a supervised learning algorithm.



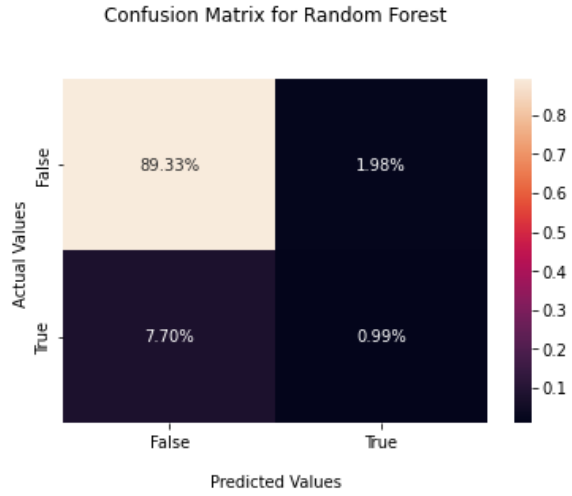
The Random Forest algorithm combines the output of multiple (randomly created) Decision Trees to generate the final output. This eliminates the risk of failure of a single decision tree to predict correctly. For our use case, we can have multiple decision trees for multiple features deciding if they impact the risk of getting a heart disease. For example, smoking and having an abnormal BMI can both impact on getting a heart disease can be derived from the output of multiple decision trees, aggregated by Random Forest classifier.

As discussed earlier, it also eradicates the limitations of a decision tree algorithm by reducing the overfitting of datasets and increases precision.

Before Sampling:

In [13]:
```
# Random Forest
Rf = RandomForestClassifier(n_estimators = 10, random_state=0)
Rf.fit(X_train, Y_train)
Rf_pred = Rf.predict(X_test)
Rf_matrix = confusion_matrix(Y_test, Rf_pred, labels=[0, 1])
```

In [14]:
```
ax = sns.heatmap(Rf_matrix/np.sum(Rf_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Random Forest\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



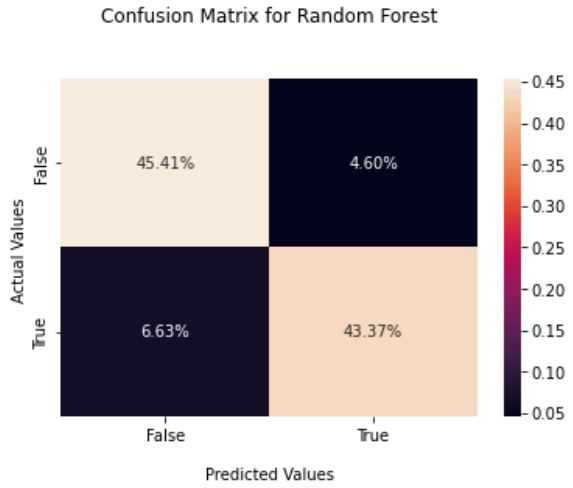
In [15]:
```
print(classification_report(Y_test, Rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     73004
           1       0.33      0.11      0.17      6945

    accuracy                           0.90     79949
   macro avg       0.63      0.55      0.56     79949
weighted avg       0.87      0.90      0.88     79949
```

SMOTE:

In [14]:
```
# Random Forest
Rf = RandomForestClassifier(n_estimators = 10, random_state=0)
Rf.fit(X_train, Y_train)
Rf_pred = Rf.predict(X_test)
Rf_matrix = confusion_matrix(Y_test, Rf_pred, labels=[0, 1])
```

In [15]:
```
ax = sns.heatmap(Rf_matrix/np.sum(Rf_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Random Forest\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

plt.show()
```



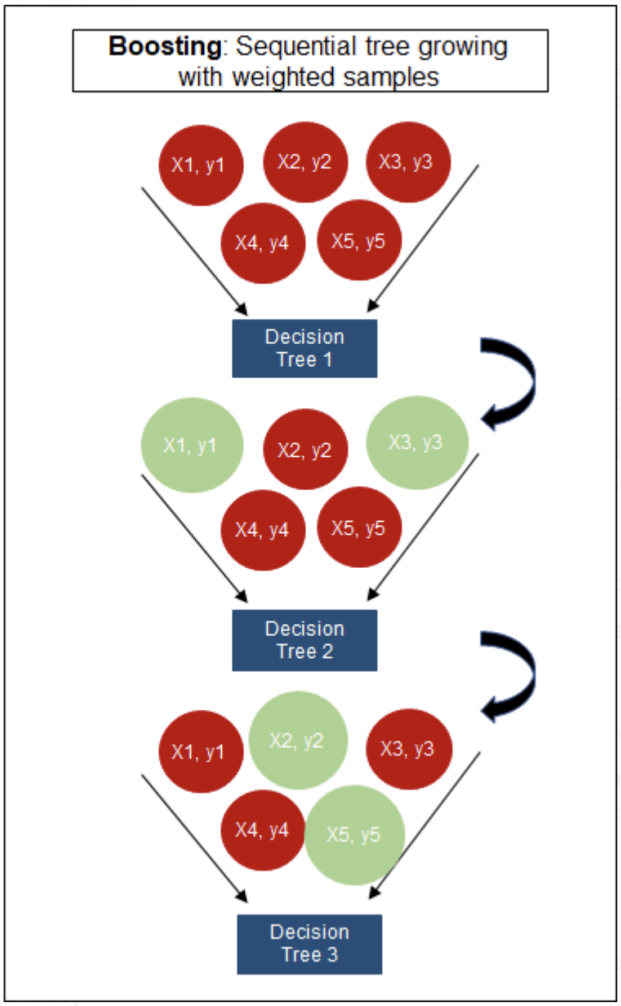
In [17]:
```
print(classification_report(Y_test, Rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.91      0.89     73112
           1       0.90      0.87      0.89     73099

    accuracy                           0.89    146211
   macro avg       0.89      0.89      0.89    146211
weighted avg       0.89      0.89      0.89    146211
```

As mentioned above, Random Forest eradicates the limitations of Decision tree and the accuracy is higher than the previously discussed algorithms. Around 90% of accuracy is achieved when the data is unsampled and around 89% on sampled data. The False positives and False negatives are also slightly better. This algorithm can handle noise relatively well, but more knowledge from the user is required to adequately tune the algorithm when compared to AdaBoost.

5. ADA Boost:

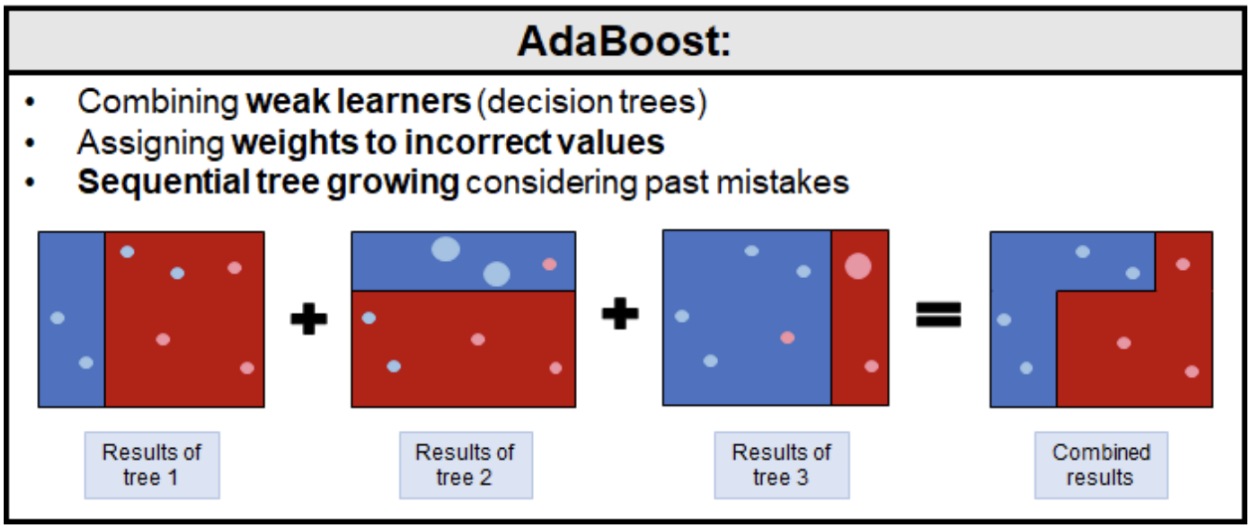
Boosting combines the outcome of many weak learners into one very accurate prediction. A weak learner refers to a learning algorithm that only predicts slightly better than randomly. The boosting approach is a sequential algorithm that makes predictions for *T* rounds on the entire training sample and iteratively improves the performance of the boosting algorithm with the information from the prior round's prediction accuracy.



AdaBoost is an ensemble learning algorithm based on boosting. The algorithm first builds a weak learner based on the training data and then increases the weight of the samples that were misclassified by weak learning in the previous round. Then, it reduces the weight of the correctly classified samples, loops this process until the weak learner reaches the specified value, and then linearly combines all weak learners to obtain the final strong classifier by weighted majority voting. The decision tree algorithm selects variables by evaluating the characteristics and depth of dividing nodes, reducing the dimension of variables.

The integrated model has better generalization error and can effectively reduce the overfitting combination phenomenon. Eventually, the decision is a result of summing up all the base

models. It is one of the most efficient techniques in ML. Moreover, this algorithm is easy to understand and to visualize. AdaBoost is best used in a dataset with low noise, when computational complexity or timeliness of results is not a main concern.



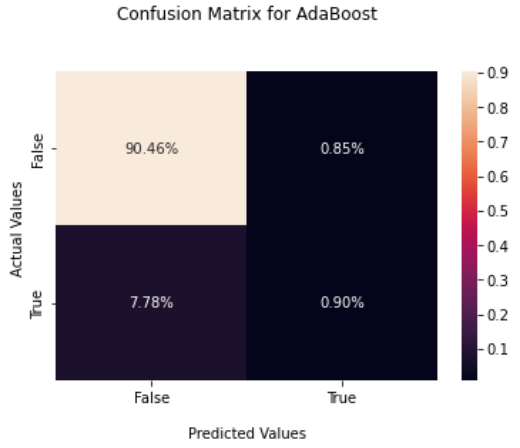
Summary and visualization of AdaBoost algorithm for classification problems. Larger points indicate that these samples were previously misclassified and a higher weight was assigned to them. Source: Julia Nikulski.

Before Sampling:

In [29]:
```
# AdaBoost
AdaBoost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=3),  n_estimators = 100, random_s
AdaBoost.fit(X_train, Y_train)
AdaBoost_pred = AdaBoost.predict(X_test)
AdaBoost_matrix = confusion_matrix(Y_test, AdaBoost_pred, labels=[0, 1])
AdaBoost.score(X_test, Y_test)
```

Out[29]: 0.9136324406809341

In [30]:
```
ax = sns.heatmap(AdaBoost_matrix/np.sum(AdaBoost_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for AdaBoost\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [31]:
```
print(classification_report(Y_test, AdaBoost_pred))
```

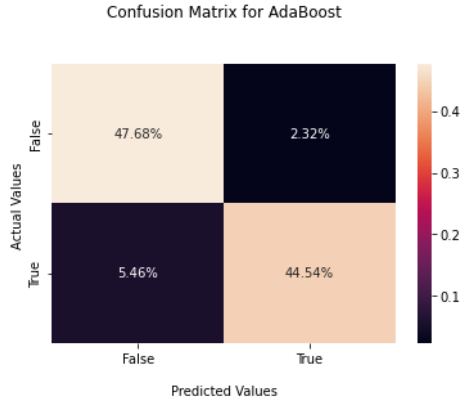
```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     73004
           1       0.51      0.10      0.17      6945

    accuracy                           0.91     79949
   macro avg       0.72      0.55      0.56     79949
weighted avg       0.89      0.91      0.89     79949
```

SMOTE:

In [17]:
```
# AdaBoost
AdaBoost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=3),  n_estimators = 100, random_state=0)
AdaBoost.fit(X_train, Y_train)
AdaBoost_pred = AdaBoost.predict(X_test)
AdaBoost_matrix = confusion_matrix(Y_test, AdaBoost_pred, labels=[0, 1])
AdaBoost.score(X_test, Y_test)
```

Out[17]: 0.9222151548105135

In [18]:
```
ax = sns.heatmap(AdaBoost_matrix/np.sum(AdaBoost_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for AdaBoost\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [19]:
```
print(classification_report(Y_test, AdaBoost_pred))
```

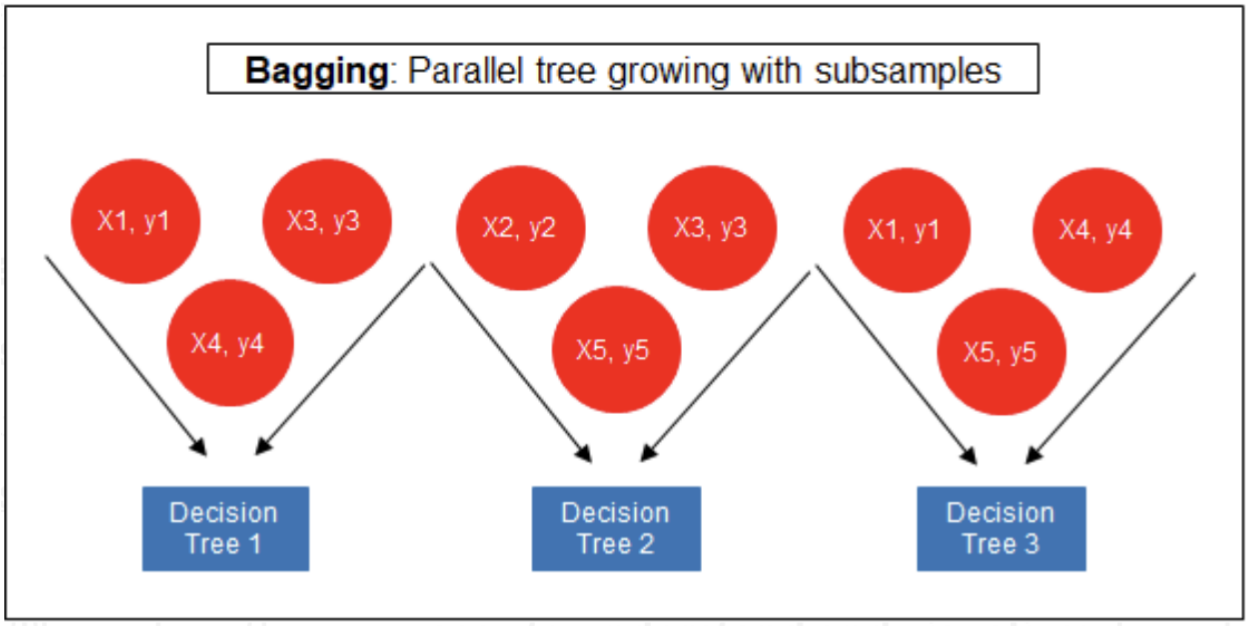
```
              precision    recall  f1-score   support

           0       0.90      0.95      0.92     73112
           1       0.95      0.89      0.92     73099

    accuracy                           0.92    146211
   macro avg       0.92      0.92      0.92    146211
weighted avg       0.92      0.92      0.92    146211
```

Compared to the above algorithms, ADABoost works well in terms of accuracy without sampling and after sampling of data. Without Sampling, the accuracy is around 91%, while 92% is achieved through SMOTE sampling. The recall and precision falls down when the algorithm is run on the unsampled data which affects the testing data if the person is more prone to heart disease. However, the precision and recall is higher on Sampled dataset and the false positives is only around 5% compared to other models.

6. Bagging:

Bagging refers to non-sequential learning. It is an ensemble learning method that is commonly used to reduce variance within a noisy dataset. The bagging approach is also called *bootstrapping*. In this study, random forest ADAboost and Bagging are ensemble learning algorithms based on decision trees.

In bagging, a random sample of data from the training set is selected or drawn with replacement—i.e. the individual data points can be chosen more than once. Each of these draws are independent of the previous round's draw but have the same distribution. These randomly selected samples are then used to grow a decision tree (weak learner). The most popular class is then chosen as the final prediction value. Voting procedure for each classification model is then performed. Consequently, the classification outcome is determined based on the majority of the average values.



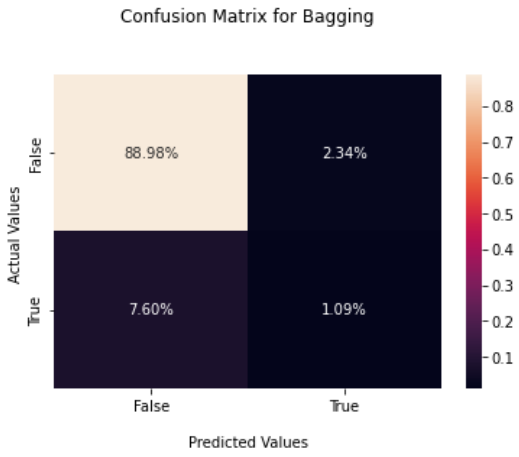
Before Sampling:

In [25]:
```
# Bagging
Bag = BaggingClassifier(n_estimators = 10, random_state=0)
Bag.fit(X_train, Y_train)
Bag_pred = Bag.predict(X_test)
Bag_matrix = confusion_matrix(Y_test, Bag_pred, labels=[0, 1])
```

In [26]:
```
Bag.score(X_test, Y_test)
```

Out[26]: 0.9006491638419493

In [27]:
```
ax = sns.heatmap(Bag_matrix/np.sum(Bag_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Bagging\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [28]:
```
print(classification_report(Y_test, Bag_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95     73004
           1       0.32      0.13      0.18      6945

    accuracy                           0.90     79949
   macro avg       0.62      0.55      0.56     79949
weighted avg       0.87      0.90      0.88     79949
```

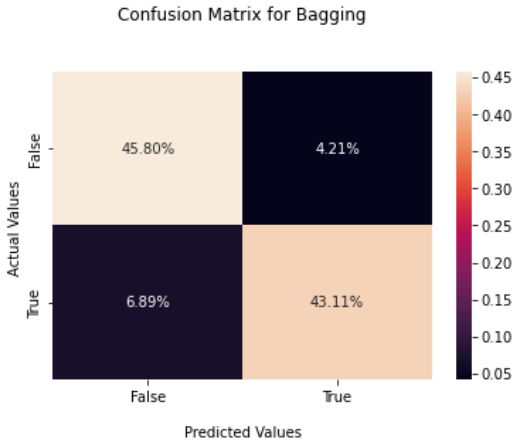
SMOTE:

In [11]:
```
# Bagging
Bag = BaggingClassifier(n_estimators = 10, random_state=0)
Bag.fit(X_train, Y_train)
Bag_pred = Bag.predict(X_test)
Bag_matrix = confusion_matrix(Y_test, Bag_pred, labels=[0, 1])
```

In [12]:
```
Bag.score(X_test, Y_test)
```

Out[12]: 0.8890712737071766

In [13]:
```
ax = sns.heatmap(Bag_matrix/np.sum(Bag_matrix), annot=True, fmt='.2%')

ax.set_title('Confusion Matrix for Bagging\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [16]:
```
print(classification_report(Y_test, Bag_pred))
```

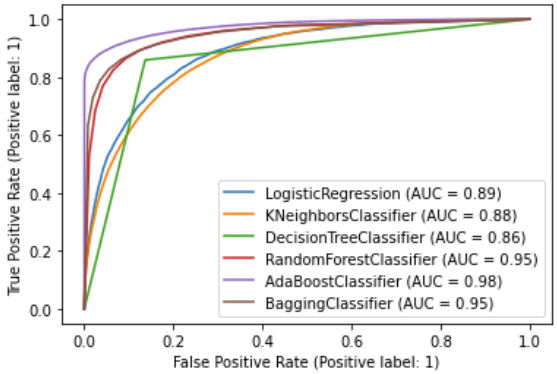
```
              precision    recall  f1-score   support

           0       0.87      0.92      0.89     73112
           1       0.91      0.86      0.89     73099

    accuracy                           0.89    146211
   macro avg       0.89      0.89      0.89    146211
weighted avg       0.89      0.89      0.89    146211
```

Bagging algorithm also works well on Sampled dataset and the accuracy achieved is around 89%. For unsampled data, the accuracy is 90% but the precision and recall for a person having heart disease is only 30% and 10%. But for the sampled dataset, the precision and the recall rate is around 90%. The false positive and false negative is around 7% and 4% which is slightly higher than ADABoost.

Evaluating using ROC and AUC:

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC refers to the probability curve and AUC represents the degree of separability. This helps us in identifying how much the model can differentiate the given classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

In [20]:
```
from sklearn.metrics import RocCurveDisplay
ax = plt.gca()
logistic_disp = RocCurveDisplay.from_estimator(classifier, X_test, Y_test, ax=ax)
knn_disp = RocCurveDisplay.from_estimator(knn, X_test, Y_test, ax=ax)
dt_disp = RocCurveDisplay.from_estimator(dtc, X_test, Y_test, ax=ax)
randomforest_disp = RocCurveDisplay.from_estimator(Rf, X_test, Y_test, ax=ax)
adaboost_disp = RocCurveDisplay.from_estimator(AdaBoost, X_test, Y_test, ax=ax)
bag_disp = RocCurveDisplay.from_estimator(Bag, X_test, Y_test, ax=ax)
plt.show()
```



From the above graph, ADABoost works the best as the AUC is 0.98. As mentioned above, higher the AUC, the better the models predict the classification results.

Model Comparison:

The models are now compared based on the precision, recall, f1-measure and accuracy. For our problem statement, reducing false negatives is paramount as it is a matter of life and death. On the other hand, false positives are not as important, as further medical tests would reveal the misdiagnosis. Therefore, we are looking for a **high recall** for the heart disease class. Below is a side by side comparison of the models using various metrics.

In [50]:
```
createframe = {
            'Model':[],
            'Sampling Method':[],
            'Heart Disease Precision':[],
            'No Heart Disease Precision':[],
            'Heart Disease Recall':[],
            'No Heart Disease Recall':[],
            'Heart Disease F1':[],
            'No Heart Disease F1':[],
            'Accuracy':[]}
```

In [61]:
```
createframe['Model'].append("ADABoost")
createframe['Sampling Method'].append("SMOTE")
createframe['Heart Disease Precision'].append(0.95)
createframe['No Heart Disease Precision'].append(0.90)
createframe['Heart Disease Recall'].append(0.89)
createframe['No Heart Disease Recall'].append(0.95)
createframe['Heart Disease F1'].append(0.92)
createframe['No Heart Disease F1'].append(0.92)
createframe['Accuracy'].append(0.92)
```

In [63]:
```
comparison = pd.DataFrame(createframe)
comparison = comparison.set_index('Model')
comparison
```

Out[63]:

Model	Sampling Method	Heart Disease Precision	No Heart Disease Precision	Heart Disease Recall	No Heart Disease Recall	Heart Disease F1	No Heart Disease F1	Accuracy
Logistic Regression	Unsampled	0.53	0.92	0.08	0.99	0.15	0.95	0.91
KNN	Unsampled	0.00	0.91	0.00	1.00	0.00	0.95	0.91
Decision Tree	Unsampled	0.24	0.93	0.26	0.92	0.25	0.93	0.87
Random Forest	Unsampled	0.33	0.92	0.11	0.98	0.17	0.95	0.90
ADABoost	Unsampled	0.51	0.92	0.10	0.99	0.17	0.95	0.91
Bagging	Unsampled	0.32	0.92	0.13	0.97	0.18	0.95	0.90
Logistic Regression	SMOTE	0.80	0.81	0.81	0.80	0.81	0.80	0.81
KNN	SMOTE	0.74	0.85	0.88	0.69	0.80	0.76	0.79
Decision Tree	SMOTE	0.86	0.86	0.86	0.87	0.86	0.86	0.86
Random Forest	SMOTE	0.90	0.87	0.87	0.91	0.89	0.89	0.89
ADABoost	SMOTE	0.95	0.90	0.89	0.95	0.92	0.92	0.92
Bagging	SMOTE	0.91	0.87	0.86	0.92	0.89	0.89	0.89

- In the above table, we can see that all the models yield best results for accuracy for **Unsampled** data. However, the values for the "**Heart Disease"** (Heart Disease=Yes) class have low values for precision, recall and F1 scores when compared to that of the "**No Heart Disease"** (Heart Disease=No) class. This is because the data is unbalanced and only 9% of our dataset has people having a heart disease.
- **KNN** model for **Unsampled** dataset yielded 0% for precision, recall and F1 score for the Heart Disease class (i.e. percentage of heart disease classifications that were actually correct). Whereas, the same model when fitted with **SMOTE** yielded 74% precision, 88% recall and 80% for F1 score for the Heart Disease class.
- Amongst all the classifiers explored so far, we can see from the above table and confidently order the models from the lowest to the highest performant classifiers:

  KNN < Logistic Regression < Decision Tree < Random Forest < Bagging < ADABoost.

- The **ADABoost** model with **SMOTE** yielded the highest precision (**95%**) for the Heart Disease class (i.e. percentage of heart disease classifications that were actually correct). They also yielded the best overall accuracy (**92%**).

For our problem statement, we care the most about identifying the risk of getting a heart disease in order to take proactive measures in saving the patient from getting a heart disease. Therefore, using the **ADABoost** model to select patients for further tests would be the best course of action for medical experts.

References:

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

https://towardsdatascience.com/why-do-we-set-a-random-state-in-machine-learning-models-bb2dc68d8431#:~:text=The%20random%20state%20hyperparameter%20in,test%20sets%20across%20different%20executions.

https://towardsdatascience.com/the-ultimate-guide-to-adaboost-random-forests-and-xgboost-7f9327061c4f

https://aparnamishra144.medium.com/how-to-split-test-train-split-data-using-scikit-learn-library-86dc816d21a2

https://python.plainenglish.io/what-the-heck-is-random-state-24a7a8389f3d

https://en.wikipedia.org/wiki/Statistical_classification

https://monkeylearn.com/blog/classification-algorithms/

https://stackoverflow.com/questions/11568897/value-of-k-in-k-nearest-neighbor-algorithm

https://machinelearningmastery.com/a-gentle-introduction-to-model-selection-for-machine-learning/

https://stackoverflow.com/questions/62658215/convergencewarning-lbfgs-failed-to-converge-status-1-stop-total-no-of-iter

https://joss.theoj.org/papers/10.21105/joss.02173.pdf

http://www.kitainformatika.com/2019/10/hitung-manual-algoritma-k-nearest.html

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5