Problem Statement

Anova Insurance, a global health insurance company, seeks to optimize its insurance policy premium pricing based on the health status of applicants. Understanding an applicant's health condition is crucial for two key decisions:
- Determining eligibility for health insurance coverage.
- Deciding on premium rates, particularly if the applicant's health indicates higher risks.

The objective is to Develop a predictive model that utilizes health data to classify individuals as 'healthy' or 'unhealthy'. This classification will assist in making informed decisions about insurance policy premium pricing.

Analysis

The analysis below is for the same dataset as seen earlier where we had done the classification using the KNN model, the Logistic regression model and a Decision Tree classifier. This time I will use some homogeneous ensemble models using Bagging and Random Forest classifier and compare that with the Decision Tree classifier.

The dataset contains 9549 rows and 20 columns (original data without preprocessing), the no. of columns becomes 23 post preprocessing because of encoding, the 23 columns includes both numerical and categorical variables. The different variables in the dataset have been explained in the data dictionary, while the Target variable is a binary outcome variable, with '1' indicating 'Unhealthy' and '0' indicating 'Healthy'.

```
In [4]: df_hc.dtypes
Out[4]:
Age                         float64
BMI                         float64
Blood_Pressure              float64
Cholesterol                 float64
Glucose_Level               float64
Heart_Rate                  float64
Sleep_Hours                 float64
Exercise_Hours              float64
Water_Intake                float64
Stress_Level                float64
Target                        int64
Smoking                       int64
Alcohol                       int64
Diet                          int64
MentalHealth                  int64
PhysicalActivity              int64
MedicalHistory                int64
Allergies                     int64
Diet_Type__Vegan               bool
Diet_Type__Vegetarian          bool
Blood_Group_AB                 bool
Blood_Group_B                  bool
Blood_Group_O                  bool
dtype: object
```

```
In [8]: print(missing_values)
Age                      0
BMI                      0
Blood_Pressure           0
Cholesterol              0
Glucose_Level            0
Heart_Rate               0
Sleep_Hours              0
Exercise_Hours           0
Water_Intake             0
Stress_Level             0
Target                   0
Smoking                  0
Alcohol                  0
Diet                     0
MentalHealth             0
PhysicalActivity         0
MedicalHistory           0
Allergies                0
Diet_Type__Vegan         0
Diet_Type__Vegetarian    0
Blood_Group_AB           0
Blood_Group_B            0
Blood_Group_O            0
dtype: int64
```

The boolean data types are converted to Int datatype before splitting the dataset into train and test. There are no missing values in any of the columns.

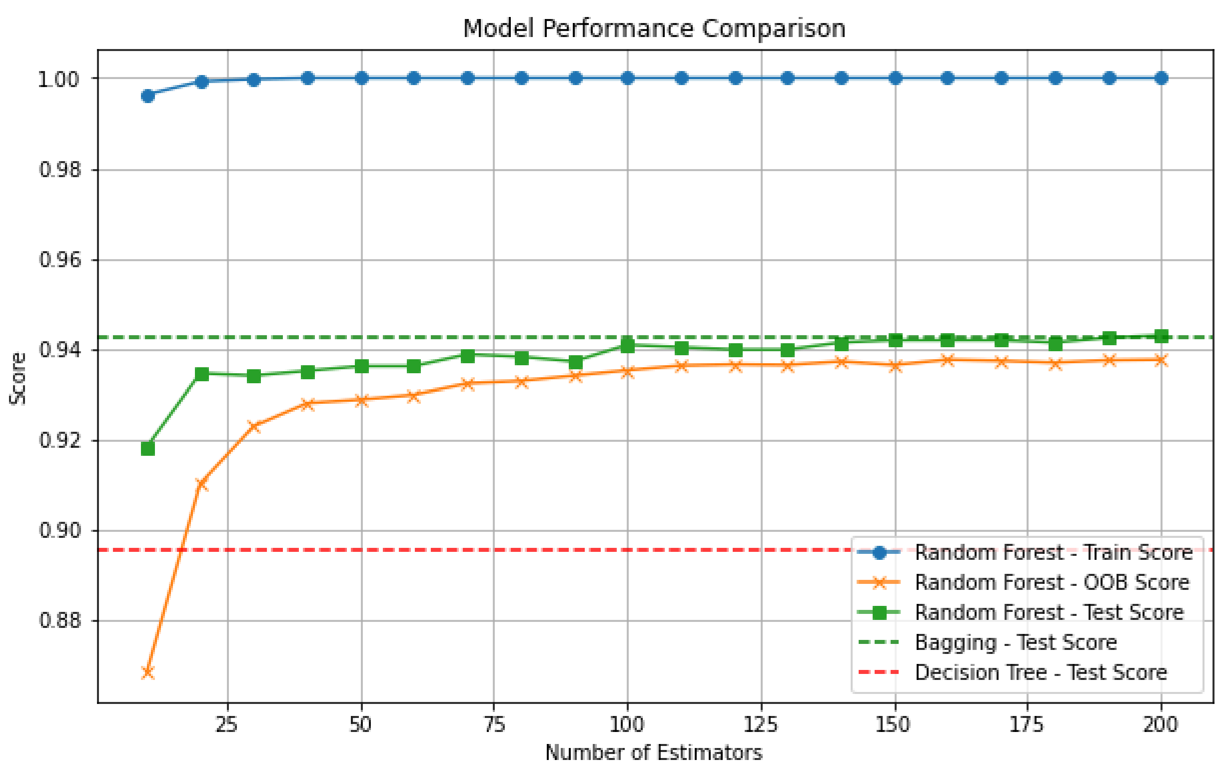
After splitting the data into 80% train and 20% test data, and using scaling, I have initialised the Random Forest, Classic Bagging and Decision Tree classifiers respectively.

The Random Forest classifier is trained with a range of n_estimators to explore how test accuracy and Out-Of-Bag scores evolve with the number of trees. Similarly, a classical bagging classifier with n_estimators = 100 was trained using DecisionTreeClassifier as the base model. And finally, a single DecisionTreeClassifier was trained to serve as a baseline comparison.

The training accuracy, OOB scores, and test accuracy were recorded for the Random Forest classifier, and Test and training scores were evaluated for Bagging classifier.

```
Performance Metrics:
Random Forest - OOB Score (Best): 0.9376
Random Forest - Test Accuracy (Best): 0.9429
Bagging - Train Accuracy: 1.0000, Test Accuracy: 0.9424
Decision Tree - Train Accuracy: 1.0000, Test Accuracy: 0.8953
```

A combined plot shows the performance metrics across the different models, highlighting the differences-



As we can see in the above plot, the Out-Of-Bag score stabilizes as the number of trees increases, indicating diminishing returns. The Out-Of-Bag (OOB) score indicates strong generalization and validates the Random Forest's ability to perform well without overfitting. Random Forest achieves the best test accuracy among the models, indicating its superior performance for this dataset. The train accuracy shows 100%, while for test data, the test accuracy stabilizes after around 50 estimators, meaning adding more trees does not significantly improve performance.

Similar to Random Forest, the train data accuracy shows 100% for both Bagging and baseline Decision Tree classifiers, indicating overfitting. While test accuracy for Bagging classifier is slightly lower than Random Forest, likely because Bagging does not perform feature randomization, limiting its ability to handle feature redundancy. The baseline DecisionTree classifier shows the lowest test accuracy among the models as expected, showing poor generalization compared to ensemble methods. Random Forest's OOB and test scores converge near the 50-100 estimator mark, suggesting that more estimators beyond this range provide minimal improvement. The Decision Tree's test score remains constant and significantly lower than Bagging or Random Forest.

Hyperparameter Tuning

I tried to check if hyperparameter tuning could further improve the performance across the different models. I used GridSearchCV for Random Forest by tuning n_estimators, max_depth, min_samples_split, and min_samples_leaf. For Bagging with Decision Trees, I tuned n_estimators and parameters of the base DecisionTreeClassifier, while for Standalone Decision Tree, I tuned max_depth, min_samples_split, and min_samples_leaf.

The performance summary after hyperparameter tuning is shown below:

Model	Best params	Train Score	Test score
Random Forest	{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}	1	0.9382
Bagging	{'base_estimator__max_depth': 20, 'base_estimator__min_samples_leaf': 1, 'base_estimator__min_samples_split': 2, 'n_estimators': 150}	1	0.9414
Decision Tree	{'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2}	0.9345	0.9000

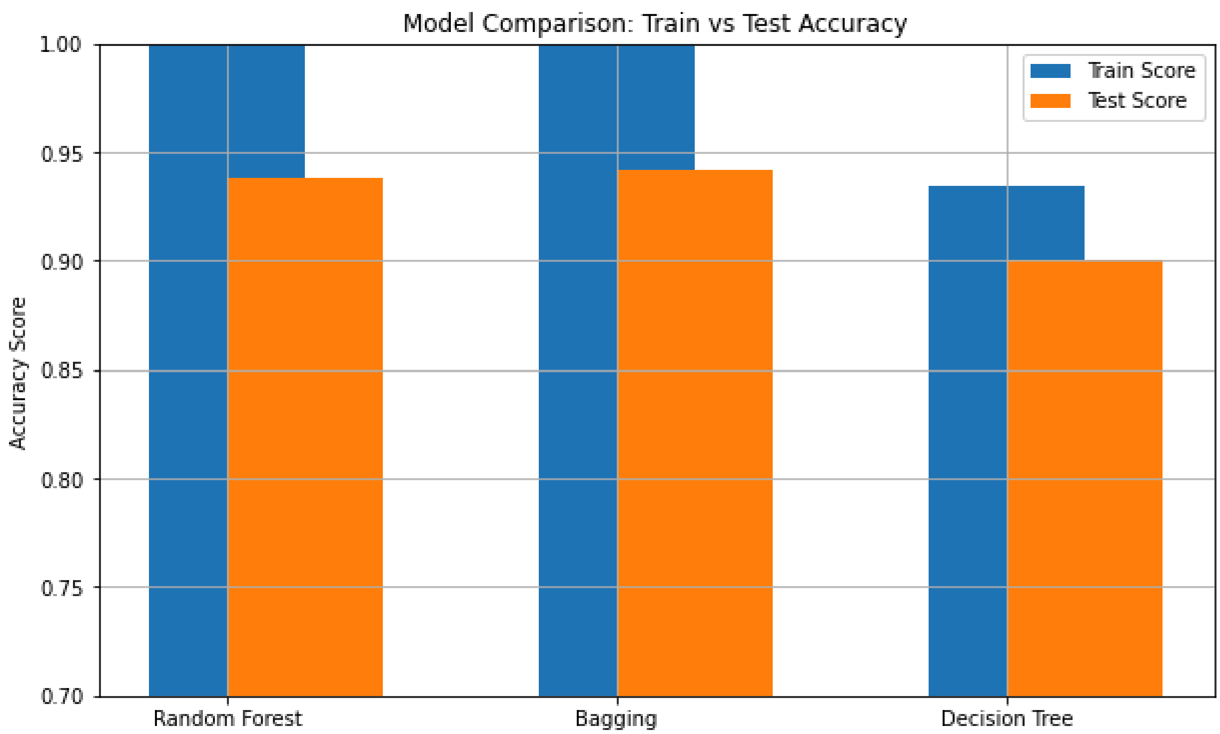
The tuned parameters for Random Forest (max_depth=20, min_samples_leaf=1, min_samples_split=2, n_estimators=100) result in a slightly lower test accuracy (**93.82%**)

compared to the previous best (**94.29%**) while the training score still remains at 100%. This marginal decrease in test score suggests the hyperparameter tuning might have slightly overfit to the training data or failed to generalize better than the default settings.

Similarly, for Bagging with Decision Tree classifier, the new parameters (max_depth=20, min_samples_leaf=1, min_samples_split=2, n_estimators=150) yield a similar test accuracy (**94.14%** vs. **94.24%**), albeit slightly lower. Also, train score remains 100%, showing no improvement in addressing overfitting with hyperparameter tuning.

For the baseline DecisionTree classifier, the new parameters (max_depth=10, min_samples_leaf=4, min_samples_split=2) lead to a slight improvement in test accuracy (from **89.53%** to **90%**). The training score decreases from 100% to 93.45%, indicating that the tree is now less overfit to the training data. The improvement suggests probably pruning (max_depth=10) and regularization (min_samples_leaf=4) successfully reduced the overfitting.

I have compared the train and test accuracy after hyperparameter tuning across all three models in a bar chart as shown below -



<u>Conclusion</u>

The drop in test accuracy for Random Forest classifier suggests the default hyperparameters might be sufficient for this dataset. The drop in performance highlights that min_samples_leaf=1 is probably too low, leading to overfitting. While for baseline DecisionTree classifier, the tuned tree is now more generalized, with improved test accuracy and reduced overfitting.