Annexure-I

Term Paper

"Post-Operative Survival Prediction in Lung Cancer Patients"

A Term Paper Report

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

(Computer Science Engineering)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

SUBMITTED BY

Name of student: Pragati Singh

Registration Number: 12115751

Faculty: Sajjad Manzoor Mir

Annexure-II

Student Declaration

To whom so ever it may concern

I, Pragati Singh, 12115751, hereby declare that the work done by me on **“Post-Operative Survival Prediction on Cancer Patients using Machine Learning”** from Feb 2024 to October 2024, is a record of original work for the partial fulfilment of the requirements for the award of the degree, Bachelor of Technology.

Name of the student: Pragati Singh

Registration Number: 12115751

Dated: 25TH OCTOBER 2024

ACKNOWLEDGEMENT

Primarily I would like to thank God for being able to learn a new technology. Then I would like to express my special thanks of gratitude to the teacher and instructor of the course Machine Learning who provided me the golden opportunity to learn a new technology.

I would like to also thank my own college Lovely Professional University for offering such a course which not only improve my programming skill but also taught me other new technology. Then I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance for choosing this course.

Finally, I would like to thank everyone who have helped me a lot.

Dated: 25TH OCTOBER 2024

Table of Contents

ABSTRACT

Lung cancer poses a significant public health challenge, necessitating early detection and accurate diagnosis for effective treatment. This report investigates a machine learning approach for classifying lung cancer using a dataset comprising 454 entries, represented in a CSV file with 17 features. The dataset includes clinical and demographic variables such as Forced Vital Capacity (FVC), Forced Expiratory Volume in one second (FEV1), tumour size, and patient history of conditions like diabetes and asthma, as well as symptoms like pain and cough. Each entry is labelled with a diagnosis indicating the presence or absence of lung cancer. We applied three machine learning algorithms—support vector machines (SVM), random forest, and logistic regression—to analyse the data and classify the entries. The models were evaluated based on their performance metrics, achieving an accuracy of x% on the training set and y% on the test set. The findings illustrate that these machine learning techniques can effectively differentiate between benign and malignant conditions, emphasizing their potential as valuable tools for the early detection and management of lung cancer.

OBJECTIVE

The primary objective of this project is to develop and evaluate machine learning models for the classification of lung cancer using a comprehensive dataset comprising 454 entries with 17 clinical and demographic features. This includes variables such as Forced Vital Capacity (FVC), Forced Expiratory Volume in one second (FEV1), tumour size, and patient histories related to various conditions. Through thorough data analysis, the project aims to identify key factors that significantly influence lung cancer diagnosis, thereby enhancing our understanding of the disease's clinical presentation.
To achieve accurate classification, the project will implement and train three distinct machine learning algorithms—support vector machines (SVM), random forest, and logistic regression. Each model will be rigorously evaluated using appropriate performance metrics, including accuracy, precision, recall, and F1-score, on both training and test datasets. The project seeks not only to compare the effectiveness of these algorithms in distinguishing between benign and malignant cases but also to provide insights and recommendations for improving early detection and treatment strategies in clinical settings

INTRODUCTION

Background: Lung cancer is the leading cause of cancer-related deaths globally, with significant mortality rates due to its aggressive nature. Major risk factors include tobacco use and environmental exposures. Surgical intervention, particularly thoracic surgery, is crucial for early-stage patients. The integration of machine learning in oncology offers opportunities to improve diagnosis, treatment planning, and post-operative outcome prediction for lung cancer patients.

1. **What is Thoracic or Lung Cancer**?
   Lung cancer, a malignancy originating in the lungs, is recognized as the leading cause of cancer-related deaths globally. It arises when cells in the lung tissue begin to grow uncontrollably, leading to the formation of tumours that can disrupt normal lung function. The condition is predominantly categorized into two main types: non-small cell lung cancer (NSCLC) and small cell lung cancer (SCLC). NSCLC is the most common form, accounting for approximately 85% of cases, while SCLC, known for its rapid growth and spread, constitutes the remaining percentage.

2. **What Are the Causes of Lung Cancer?**
   The primary cause of lung cancer is smoking, which is responsible for most cases; however, non-smokers can also develop the disease due to factors such as exposure to second-hand smoke, environmental pollutants, occupational hazards (like asbestos exposure), and genetic predispositions. Understanding these risk factors is crucial for developing prevention strategies and targeted treatments.

3. **What Are the Existing Cures and Remedies?**
   Existing treatment options for lung cancer vary based on the stage and type of cancer and may include surgery, chemotherapy, radiation therapy, targeted therapy, and immunotherapy. Surgical intervention, particularly thoracic surgery, is a critical option for patients with localized lung cancer and can significantly improve survival rates. Despite the availability of these treatments, the prognosis for lung cancer remains poor, especially for advanced stages, highlighting the need for ongoing research into more effective therapies and early detection methods.

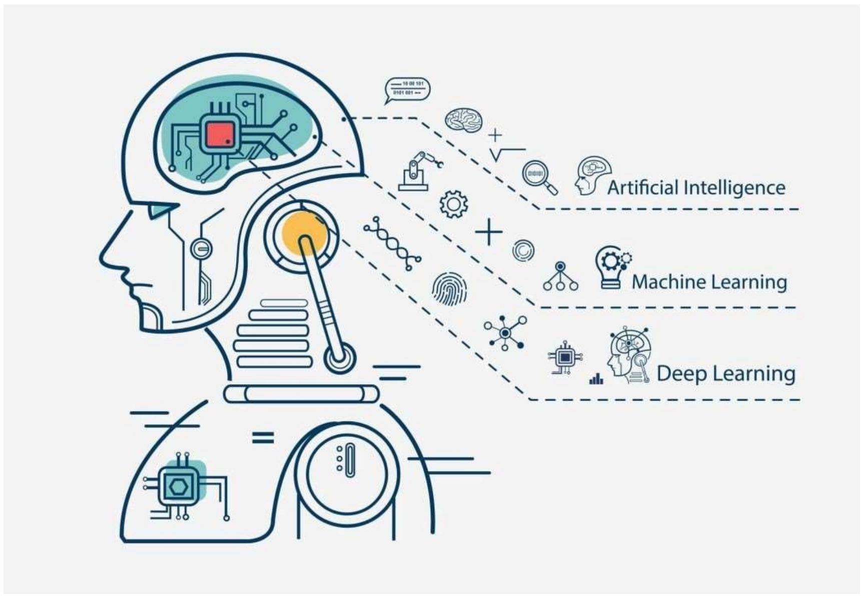
4. **Why is There a Need for Machine Learning in Lung Cancer?**
   Machine learning (ML) has emerged as a transformative tool in the field of healthcare, particularly in oncology. With its ability to analyse vast datasets and identify complex patterns, ML can enhance diagnostic accuracy, predict patient outcomes, and personalize treatment plans. In the context of lung cancer, ML models can be utilized to classify patients based on their post-operative life expectancy, aiding clinicians in making informed decisions regarding patient

management. As data continues to be collected and analysed, the integration of machine learning into thoracic surgery holds the potential to significantly improve patient outcomes and survival rates.

THEORETICAL BACKGROUND

1. What is Machine Learning?

Machine learning (ML) is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform tasks without explicit programming. Instead of following predetermined rules, machine learning systems learn from data, identifying patterns and making decisions based on the information they analyse.



Key Concepts of Machine Learning:

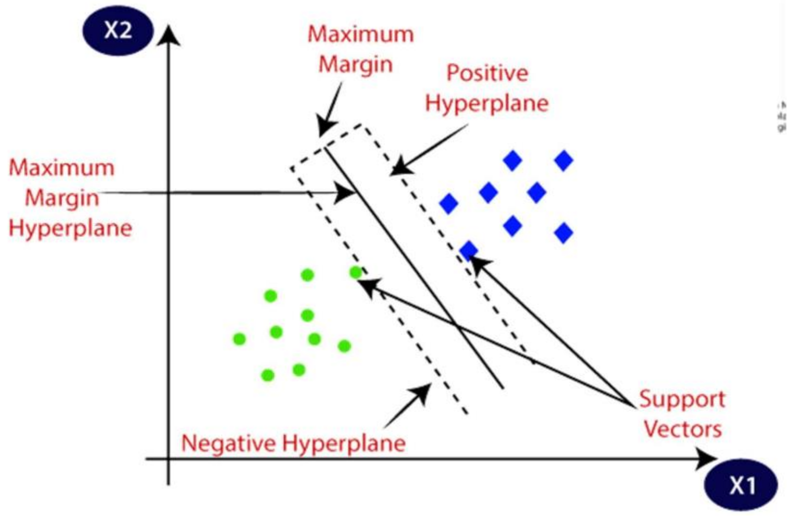
Data: ML relies on large datasets to train models. The quality and quantity of data significantly impact the performance of machine learning algorithms.
Algorithms: Various algorithms are used in ML, including supervised learning (where the model is trained on labelled data), unsupervised learning (where the model identifies patterns in unlabelled data), and reinforcement learning (where the model learns through trial and error to maximize a reward).

Training and Testing: Models are trained on a subset of the data and then tested on a separate dataset to evaluate their performance. This helps ensure that the model generalizes well to new, unseen data.
Applications: Machine learning is applied in numerous fields, including healthcare (for diagnostics and treatment predictions), finance (for fraud detection and credit scoring), marketing (for customer segmentation), and many more.

Machine learning (ML) can be broadly categorized into several types, each with distinct characteristics and applications. The primary types of ML include supervised learning, where algorithms are trained on labelled datasets to predict outcomes based on input features; unsupervised learning, which involves finding patterns or groupings in unlabelled data; semi-supervised learning, which combines a small amount of labelled data with a larger pool of unlabelled data to enhance learning accuracy; and reinforcement learning, where algorithms learn to make decisions through interactions with an environment, optimizing for cumulative rewards. Within these categories, various algorithms are employed: supervised learning utilizes algorithms such as linear regression, logistic regression, decision trees, random forest, and support vector machines (SVM); unsupervised learning features algorithms like K-means clustering, hierarchical clustering, and principal component analysis (PCA); reinforcement learning includes Q-learning and policy gradients; and deep learning leverages neural networks, particularly convolutional neural networks (CNN) for image data and recurrent neural networks (RNN) for sequence data. Each type and algorithm serve specific purposes and is selected based on the dataset's characteristics and the objectives of the analysis.

What is SUPPORT VECTOR MACHINE?

Support Vector Machines (SVM) are a powerful class of supervised learning algorithms used primarily for classification tasks, though they can also be employed for regression. The key idea behind SVM is to find the optimal hyperplane that separates data points from different classes in a high-dimensional feature space. This hyperplane maximizes the margin between the classes, defined as the distance between the closest points of each class, known as support vectors. SVM can handle both linear and non-linear classification tasks. In cases where data is not linearly separable, the kernel trick is employed to map the original data into a higher dimensional space where a hyperplane can be used for separation. Common kernels include polynomial and radial basis function (RBF) kernels, which allow SVM to adapt to complex data distributions.



One of the significant advantages of SVM is its effectiveness in high-dimensional spaces and its robustness against overfitting, especially in cases where the number of dimensions exceeds the number of samples. However, SVMs can be computationally intensive and may require significant memory, making them less efficient for very large datasets. The choice of kernel and the tuning of hyperparameters also require careful consideration, as they can greatly influence the model's performance.

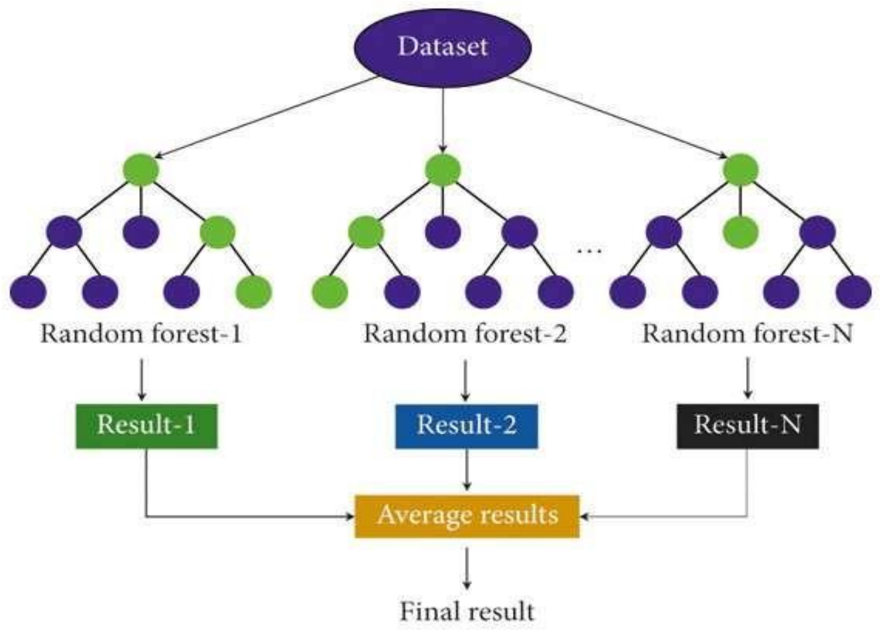
What is Random Forest Classification?

Random Forest is an ensemble learning technique that combines multiple decision trees to enhance predictive accuracy and control overfitting. It operates by constructing a multitude of decision trees during training and outputs the mode of their predictions for classification tasks or the average for regression tasks. The method introduces randomness into the model-building process by selecting a random subset of features and data points for each tree, which contributes to the diversity of the trees in the forest.

One of the main advantages of Random Forest is its robustness; it performs well across various types of datasets, even when they contain noise and missing values. Its inherent feature importance evaluation can also provide insights into the relevance of each predictor variable, making it a valuable tool for feature selection. Additionally, Random Forest is less sensitive to overfitting compared to individual decision trees, due to the averaging of predictions from multiple trees.

However, Random Forest can be computationally intensive, requiring more resources for training and prediction compared to simpler models. The interpretability of the model is also lower than that of a single decision tree, as it becomes challenging to understand the overall decision-making process of the ensemble. Despite these limitations, Random Forest remains a popular choice in various applications, including finance, healthcare, and marketing, due to its versatility and performance.

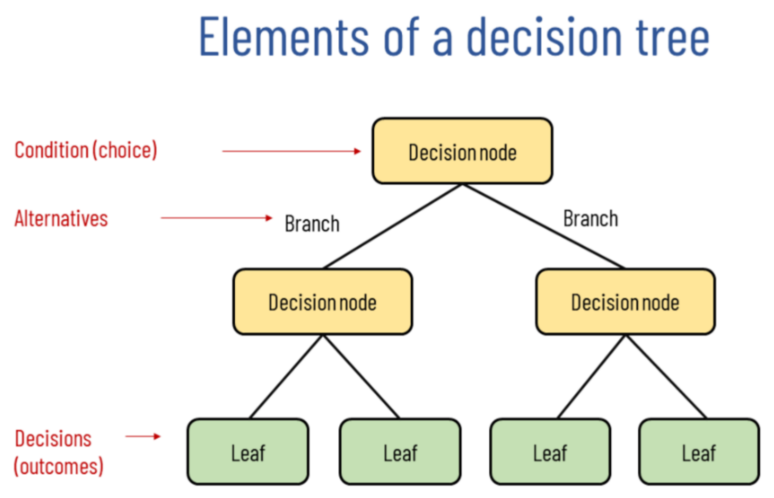
Flow-Chart:



What are Decision Trees?

A decision tree is a flowchart-like structure used to make decisions or predictions. It consists of nodes representing decisions or tests on attributes, branches representing the outcome of these decisions, and leaf nodes representing final outcomes or predictions. Each internal node corresponds to a test on an attribute, each branch corresponds to the result of the test, and each leaf node corresponds to a class label or a continuous value.

Structure of a Decision Tree



Root Node: Represents the entire dataset and the initial decision to be made.
Internal Nodes: Represent decisions or tests on attributes. Each internal node has one or more branches.
Branches: Represent the outcome of a decision or test, leading to another node.
Leaf Nodes: Represent the final decision or prediction. No further splits occur at these nodes.

How Decision Trees Work?

The process of creating a decision tree involves:
Selecting the Best Attribute: Using a metric like Gini impurity, entropy, or information gain, the best attribute to split the data is selected.
Splitting the Dataset: The dataset is split into subsets based on the selected attribute.
Repeating the Process: The process is repeated recursively for each subset, creating a new internal node or leaf node until a stopping criterion is met (e.g., all instances in a node belong to the same class or a predefined depth is reached).

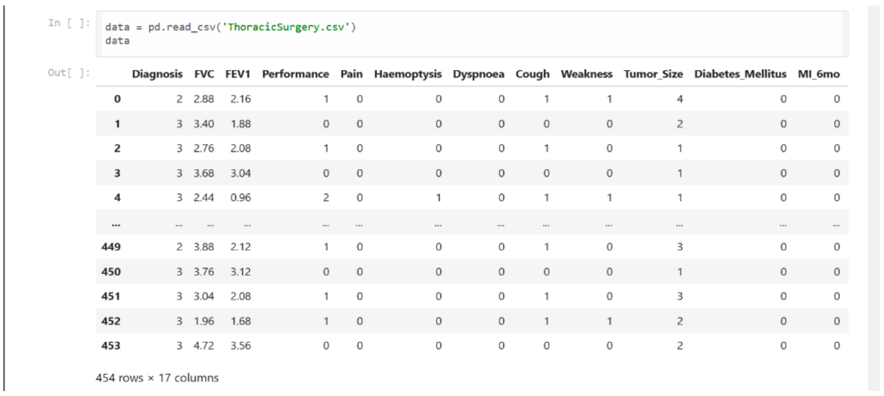
METHODOLOGY

- **Problem Definition**

Clearly define the problem you want to solve. Determine the type of machine learning task involved— whether it is a classification, regression, clustering, or recommendation problem. Set specific goals and metrics for success, such as accuracy, precision, recall, or mean squared error, depending on the task.

- **Data Collection**

Gather the necessary data relevant to your problem. This may involve obtaining datasets from public repositories, conducting surveys, or using existing company databases. Ensure that the data collected is representative of the problem domain and sufficient in size for training an effective model.



```
In [ ]: data = pd.read_csv('ThoracicSurgery.csv')
        data
```

Out[ ]:

	Diagnosis	FVC	FEV1	Performance	Pain	Haemoptysis	Dyspnoea	Cough	Weakness	Tumor_Size	Diabetes_Mellitus	MI_6mo
0	2	2.88	2.16	1	0	0	0	1	1	4	0	0
1	3	3.40	1.88	0	0	0	0	0	0	2	0	0
2	3	2.76	2.08	1	0	0	0	1	0	1	0	0
3	3	3.68	3.04	0	0	0	0	0	0	1	0	0
4	3	2.44	0.96	2	0	1	0	1	1	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
449	2	3.88	2.12	1	0	0	0	1	0	3	0	0
450	3	3.76	3.12	0	0	0	0	0	0	1	0	0
451	3	3.04	2.08	1	0	0	0	1	0	3	0	0
452	3	1.96	1.68	1	0	0	0	1	1	2	0	0
453	3	4.72	3.56	0	0	0	0	0	0	2	0	0

454 rows × 17 columns

- **Data Preprocessing**

Prepare the data for analysis by performing the following steps:

- **Data Cleaning:** Handle missing values, remove duplicates, and correct inconsistencies.
- **Data Transformation**: Normalize or standardize numerical features and encode categorical variables using techniques like one-hot encoding or label encoding.
- **Feature Selection/Extraction:** Identify and select relevant features that contribute to the predictive power of the model. Techniques like correlation analysis, recursive feature elimination, or PCA can be used.
- Data Splitting

Split the dataset into training, validation, and test sets. A common approach is to use 70% of the data for training, 15% for validation, and 15% for testing. The training set is used to train the model, the validation set helps in tuning hyperparameters, and the test set assesses the model's performance on unseen data.

- Model Selection

Choose appropriate machine learning algorithms based on the problem type and data characteristics. Common algorithms include logistic regression, decision trees, random forests, support vector machines, and neural networks. You may also consider using ensemble methods or deep learning models for complex tasks.

- Model Training

Train the selected model using the training dataset. This involves feeding the input features into the model and adjusting the model parameters through techniques such as gradient descent. Monitor the training process to ensure convergence and prevent overfitting.

- Hyperparameter Tuning

Optimize model performance by tuning hyperparameters using techniques like grid search or random search on the validation set. This step is crucial to finding the best combination of parameters for your model.

- Model Evaluation

Evaluate the trained model using the test dataset. Use appropriate metrics based on the task (e.g., accuracy, F1-score, ROC-AUC for classification; mean absolute error, R-squared for regression) to assess performance. Compare results against baseline models or existing solutions.

- Model Interpretation

Analyze the model results to gain insights into its behavior. Techniques like feature importance scores, SHAP values, or LIME can help explain model predictions, making it easier to interpret the outcomes and ensure trustworthiness.

- Deployment

Once satisfied with the model’s performance, deploy it into a production environment. This could involve integrating the model into an application, providing an API for real-time predictions, or generating reports for stakeholders.

- Monitoring and Maintenance

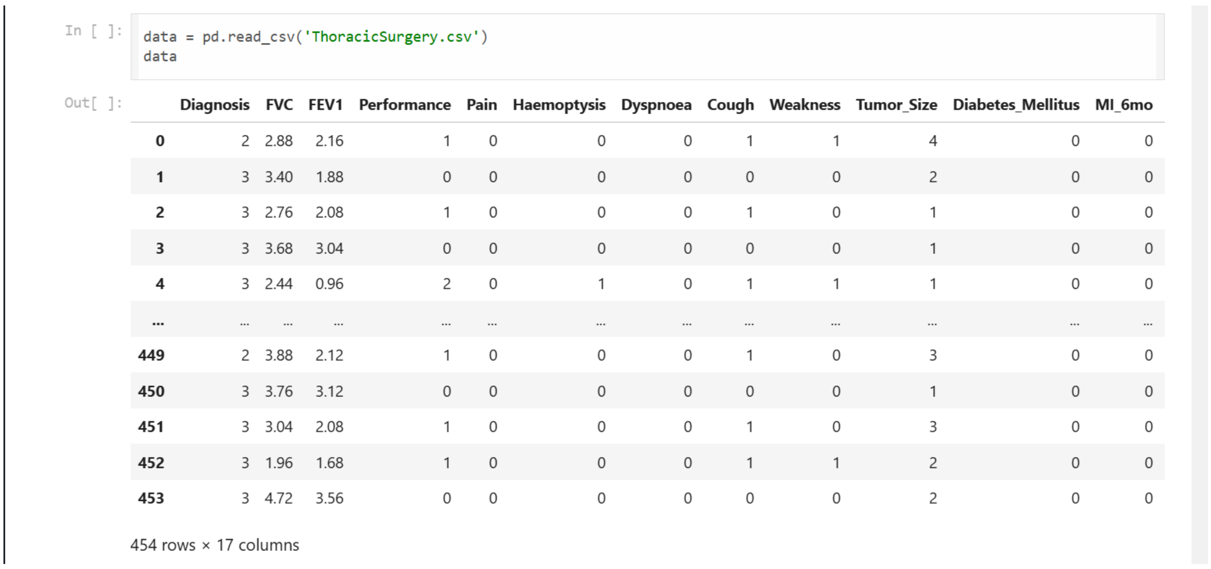
Continuously monitor the model’s performance in real-world conditions to ensure it remains accurate and reliable. Be prepared to update the model periodically with new data, retrain it if necessary, and address any drift in performance over time.

- Documentation and Reporting

Document the entire process, including decisions made, methodologies used, and outcomes achieved.

Code:

1. Importing Data Set:



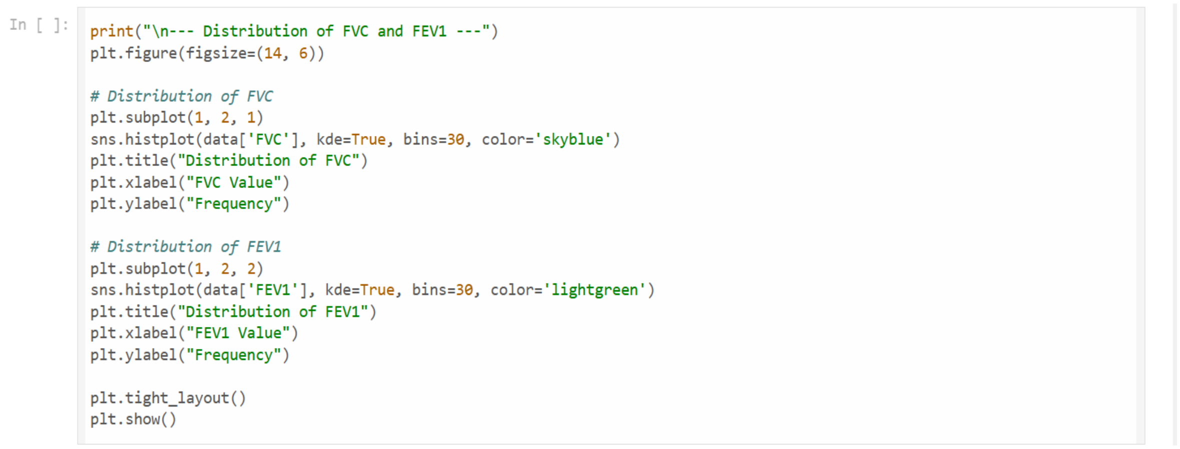
```
In [ ]: data = pd.read_csv('ThoracicSurgery.csv')
        data
```

Out[ ]:

	Diagnosis	FVC	FEV1	Performance	Pain	Haemoptysis	Dyspnoea	Cough	Weakness	Tumor_Size	Diabetes_Mellitus	MI_6mo
0	2	2.88	2.16	1	0	0	0	1	1	4	0	0
1	3	3.40	1.88	0	0	0	0	0	0	2	0	0
2	3	2.76	2.08	1	0	0	0	1	0	1	0	0
3	3	3.68	3.04	0	0	0	0	0	0	1	0	0
4	3	2.44	0.96	2	0	1	0	1	1	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
449	2	3.88	2.12	1	0	0	0	1	0	3	0	0
450	3	3.76	3.12	0	0	0	0	0	0	1	0	0
451	3	3.04	2.08	1	0	0	0	1	0	3	0	0
452	3	1.96	1.68	1	0	0	0	1	1	2	0	0
453	3	4.72	3.56	0	0	0	0	0	0	2	0	0

454 rows × 17 columns

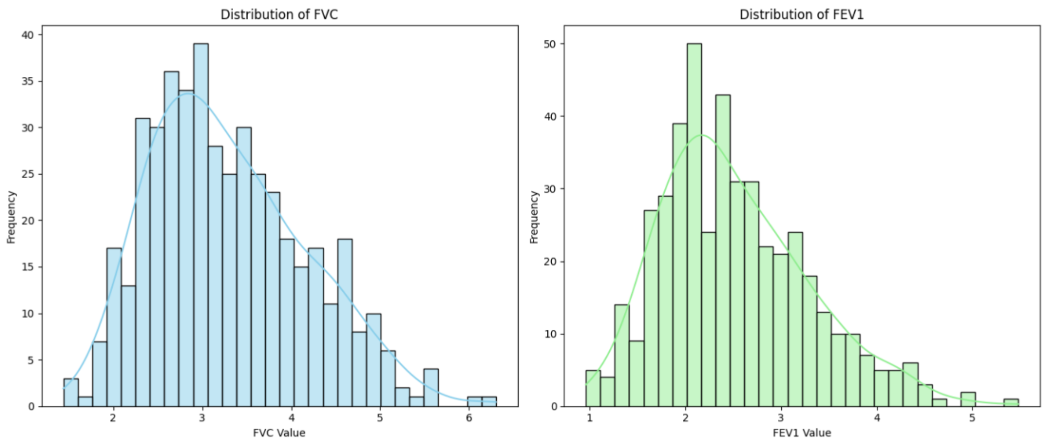
2. Exploratory Data Analysis:



```
In [ ]: print("\n--- Distribution of FVC and FEV1 ---")
        plt.figure(figsize=(14, 6))

        # Distribution of FVC
        plt.subplot(1, 2, 1)
        sns.histplot(data['FVC'], kde=True, bins=30, color='skyblue')
        plt.title("Distribution of FVC")
        plt.xlabel("FVC Value")
        plt.ylabel("Frequency")

        # Distribution of FEV1
        plt.subplot(1, 2, 2)
        sns.histplot(data['FEV1'], kde=True, bins=30, color='lightgreen')
        plt.title("Distribution of FEV1")
        plt.xlabel("FEV1 Value")
        plt.ylabel("Frequency")

        plt.tight_layout()
        plt.show()
```

Here, we understand the spread and shape of the FVC and FEV1 data and identify the Skewness in distributions (e.g., if values are mostly clustered on one side).
Also, the presence of outliers. Variations between FVC and FEV1, which are important in respiratory health evaluations.

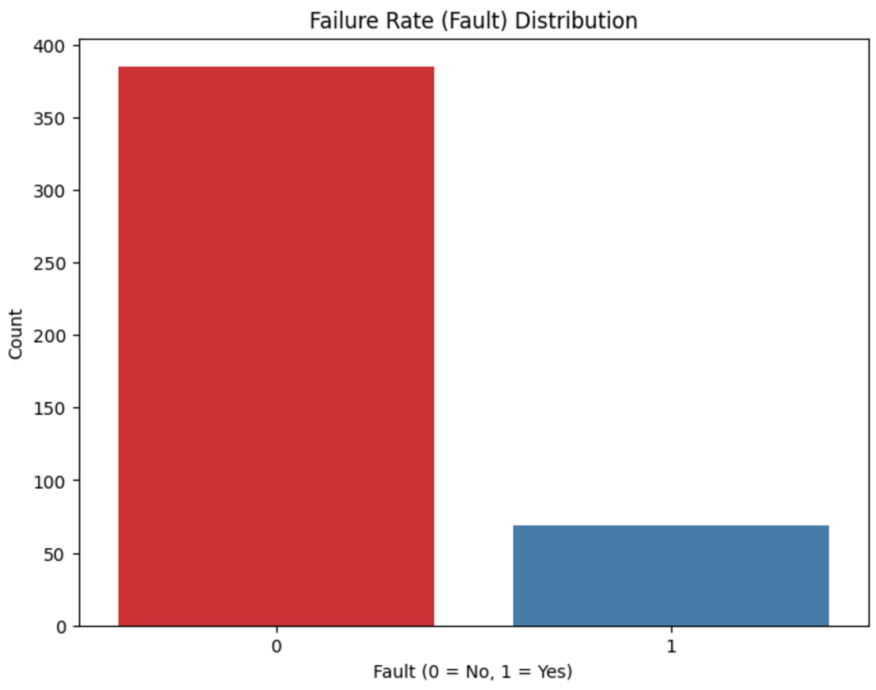
```
--- Distribution of FVC and FEV1 ---
```



Also,

```
In [ ]: # Failure Rate Analysis
        print("\n--- Failure Rate Distribution ---")
        plt.figure(figsize=(8, 6))
        sns.countplot(x='Fault', data=data, palette='Set1')
        plt.title("Failure Rate (Fault) Distribution")
        plt.xlabel("Fault (0 = No, 1 = Yes)")
        plt.ylabel("Count")
        plt.show()
```

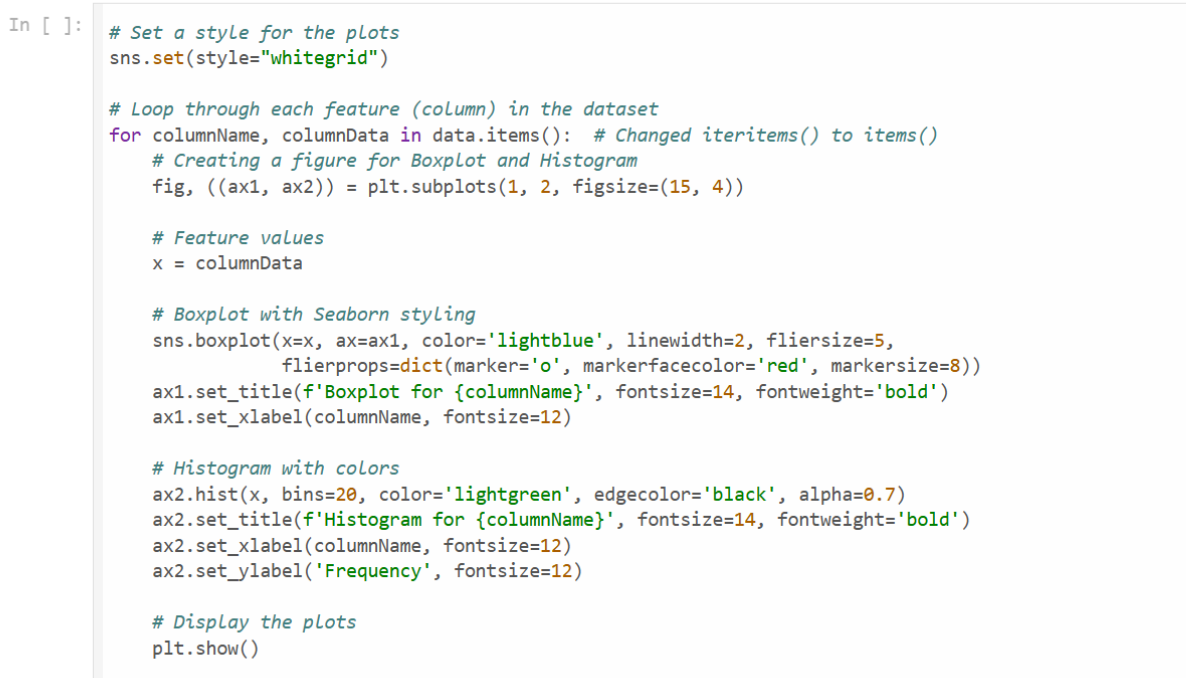
```
--- Failure Rate Distribution ---
```



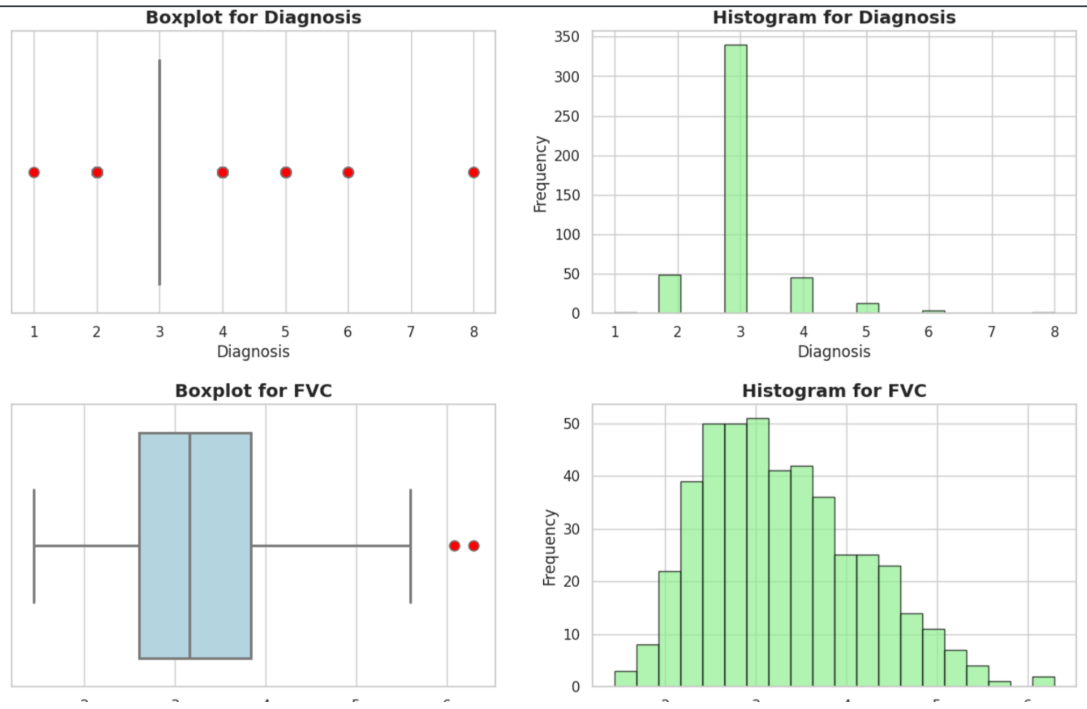
Understanding the distribution of failures in the dataset.
Identifying whether the dataset is imbalanced (e.g., significantly more 0s than 1s or vice versa).
Guiding subsequent analysis or modelling:
If Fault is highly imbalanced, strategies like resampling or using metrics suitable for imbalanced datasets (e.g., F1-score) might be needed.

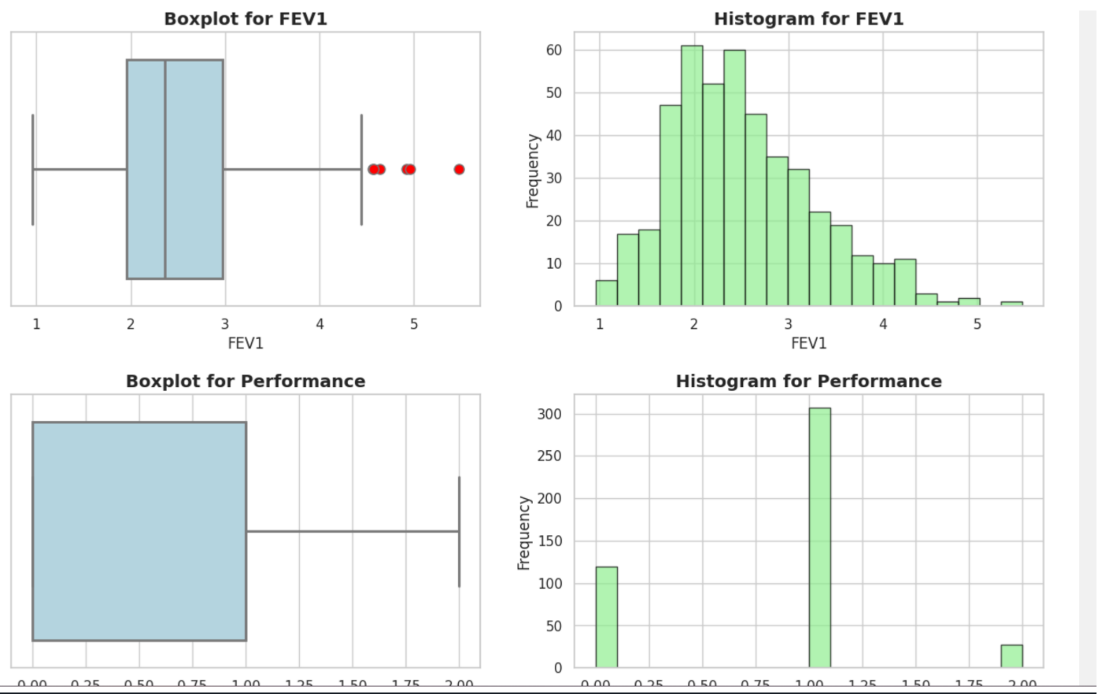
Performing visualization is part of Exploratory Data Analysis (EDA) and helps:

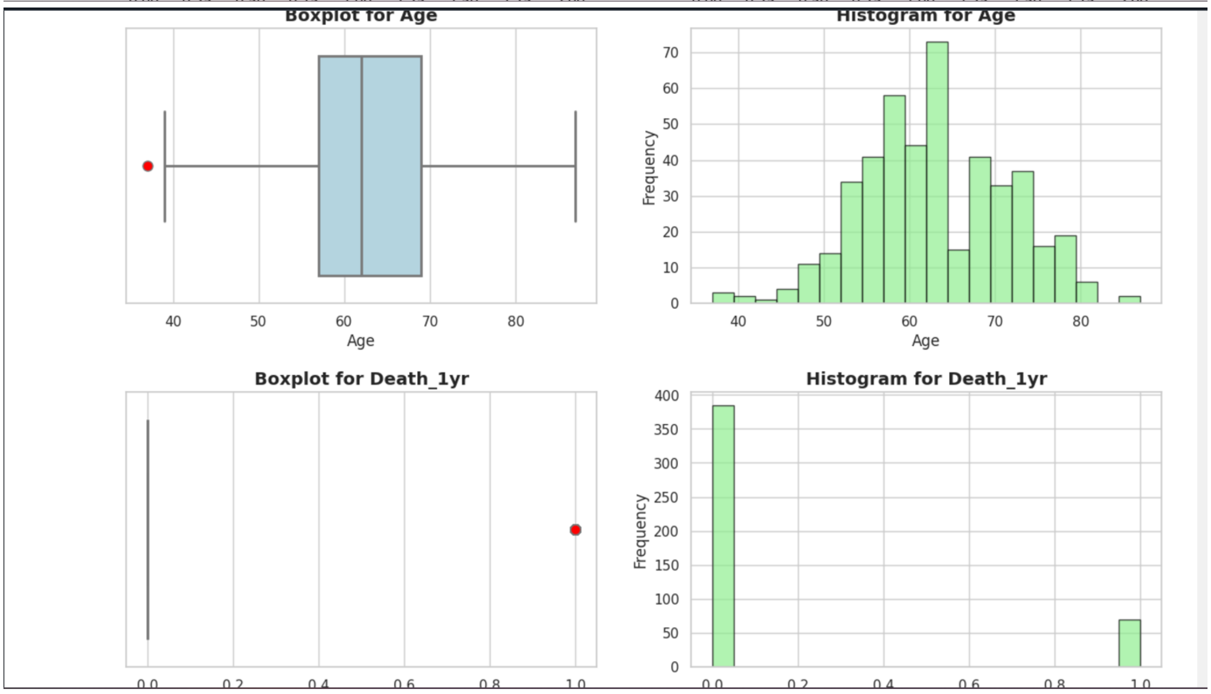
- Identify the range, median, and spread of each feature.
- Detect outliers using Boxplots.
- Understand the shape and distribution of the data using Histograms.



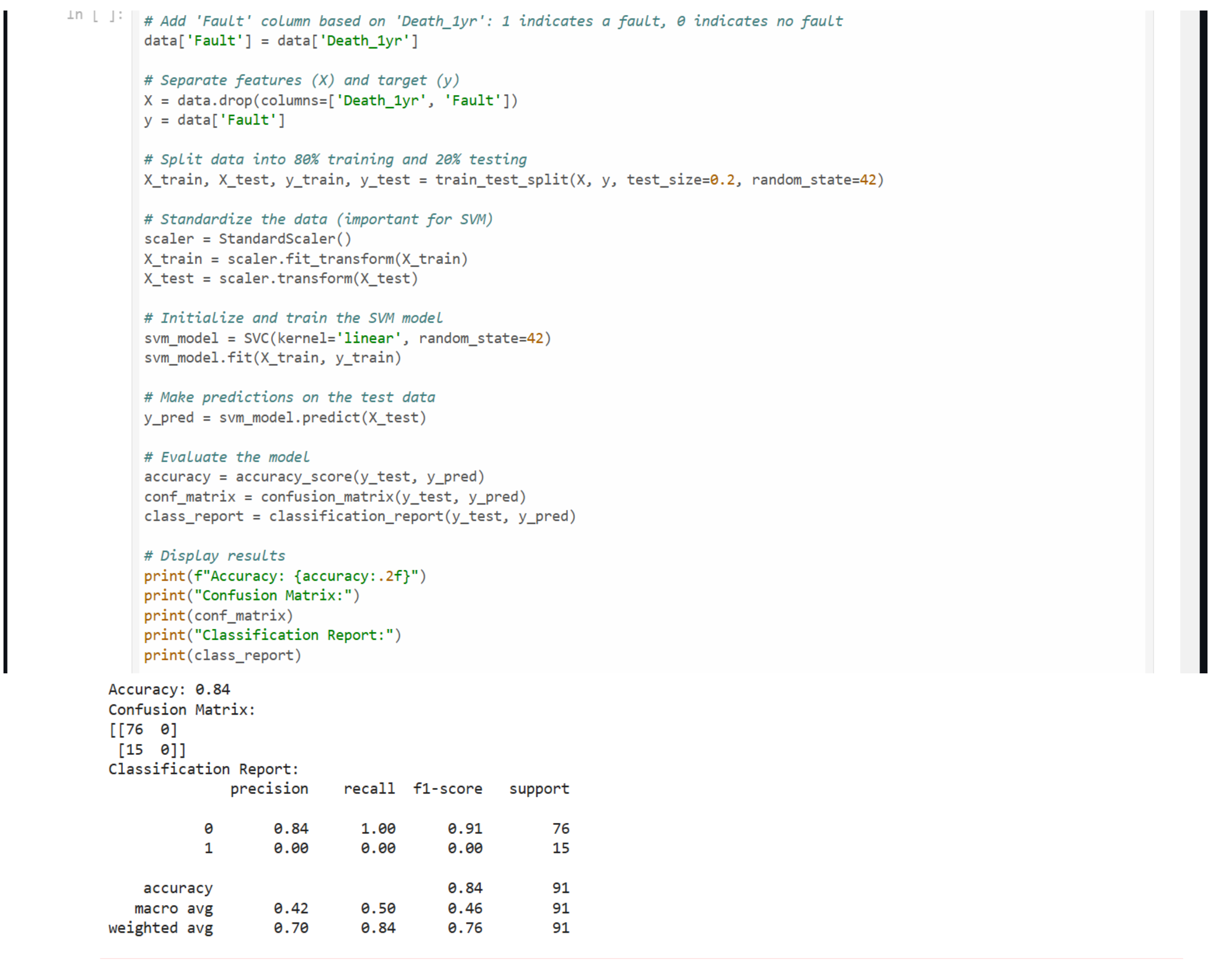
```
In [ ]:
# Set a style for the plots
sns.set(style="whitegrid")

# Loop through each feature (column) in the dataset
for columnName, columnData in data.items():  # Changed iteritems() to items()
    # Creating a figure for Boxplot and Histogram
    fig, ((ax1, ax2)) = plt.subplots(1, 2, figsize=(15, 4))

    # Feature values
    x = columnData

    # Boxplot with Seaborn styling
    sns.boxplot(x=x, ax=ax1, color='lightblue', linewidth=2, fliersize=5,
                flierprops=dict(marker='o', markerfacecolor='red', markersize=8))
    ax1.set_title(f'Boxplot for {columnName}', fontsize=14, fontweight='bold')
    ax1.set_xlabel(columnName, fontsize=12)

    # Histogram with colors
    ax2.hist(x, bins=20, color='lightgreen', edgecolor='black', alpha=0.7)
    ax2.set_title(f'Histogram for {columnName}', fontsize=14, fontweight='bold')
    ax2.set_xlabel(columnName, fontsize=12)
    ax2.set_ylabel('Frequency', fontsize=12)

    # Display the plots
    plt.show()
```




Boxplot for FEV1
FEV1
Histogram for FEV1
Frequency
FEV1
Boxplot for Performance
Histogram for Performance
Frequency


Boxplot for Age
Age
Histogram for Age
Frequency
Age
Boxplot for Death_1yr
Histogram for Death_1yr
Frequency

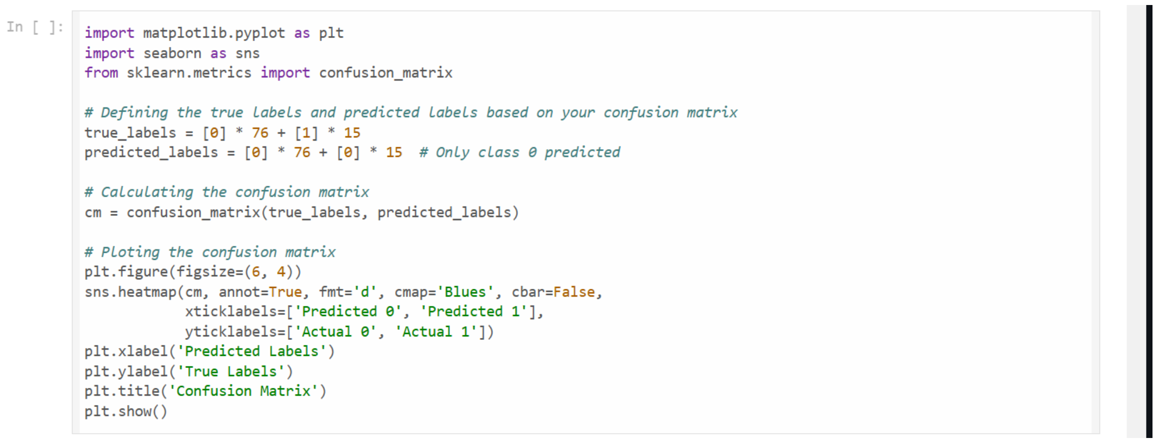
USING SVM:



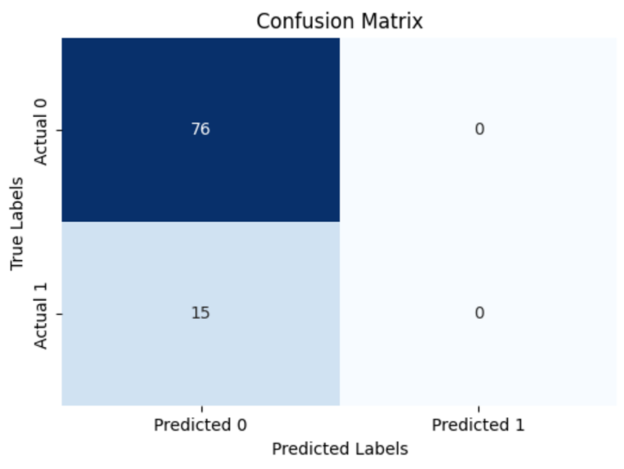
```
In [ ]: # Add 'Fault' column based on 'Death_1yr': 1 indicates a fault, 0 indicates no fault
        data['Fault'] = data['Death_1yr']

        # Separate features (X) and target (y)
        X = data.drop(columns=['Death_1yr', 'Fault'])
        y = data['Fault']

        # Split data into 80% training and 20% testing
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # Standardize the data (important for SVM)
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        # Initialize and train the SVM model
        svm_model = SVC(kernel='linear', random_state=42)
        svm_model.fit(X_train, y_train)

        # Make predictions on the test data
        y_pred = svm_model.predict(X_test)

        # Evaluate the model
        accuracy = accuracy_score(y_test, y_pred)
        conf_matrix = confusion_matrix(y_test, y_pred)
        class_report = classification_report(y_test, y_pred)

        # Display results
        print(f"Accuracy: {accuracy:.2f}")
        print("Confusion Matrix:")
        print(conf_matrix)
        print("Classification Report:")
        print(class_report)
```

```
Accuracy: 0.84
Confusion Matrix:
[[76  0]
 [15  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        76
           1       0.00      0.00      0.00        15

    accuracy                           0.84        91
   macro avg       0.42      0.50      0.46        91
weighted avg       0.70      0.84      0.76        91
```

Confusion Matrix:



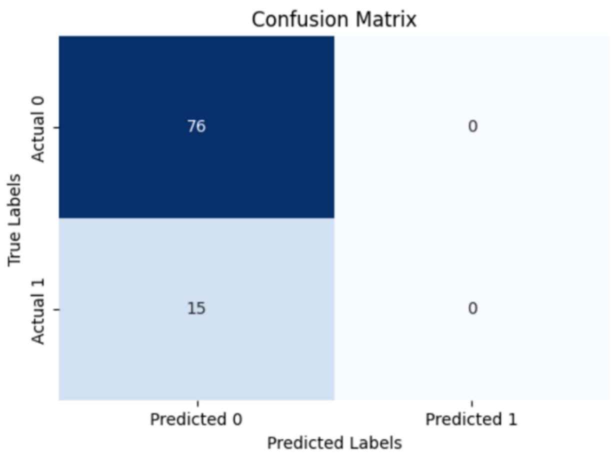
```
In [ ]: import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.metrics import confusion_matrix

        # Defining the true labels and predicted labels based on your confusion matrix
        true_labels = [0] * 76 + [1] * 15
        predicted_labels = [0] * 76 + [0] * 15  # Only class 0 predicted

        # Calculating the confusion matrix
        cm = confusion_matrix(true_labels, predicted_labels)

        # Ploting the confusion matrix
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                    xticklabels=['Predicted 0', 'Predicted 1'],
                    yticklabels=['Actual 0', 'Actual 1'])
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.title('Confusion Matrix')
        plt.show()
```



USING RANDOM FOREST:

```
In [ ]:
from sklearn.ensemble import RandomForestClassifier
# Standardize the data (important for models like SVM, but not critical for Random Forest)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],            # Number of trees
    'max_depth': [None, 10, 20, 30],             # Maximum depth of trees
    'min_samples_split': [2, 5, 10],             # Minimum samples to split an internal node
    'min_samples_leaf': [1, 2, 4],               # Minimum samples per leaf node
    'bootstrap': [True, False]                   # Whether to use bootstrap sampling
}

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)

# Get the best parameters from grid search
print("Best Parameters:", grid_search.best_params_)

# Train the model with the best parameters
best_rf_model = grid_search.best_estimator_

# Make predictions on the test data
y_pred = best_rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display results
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)

# Plot the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

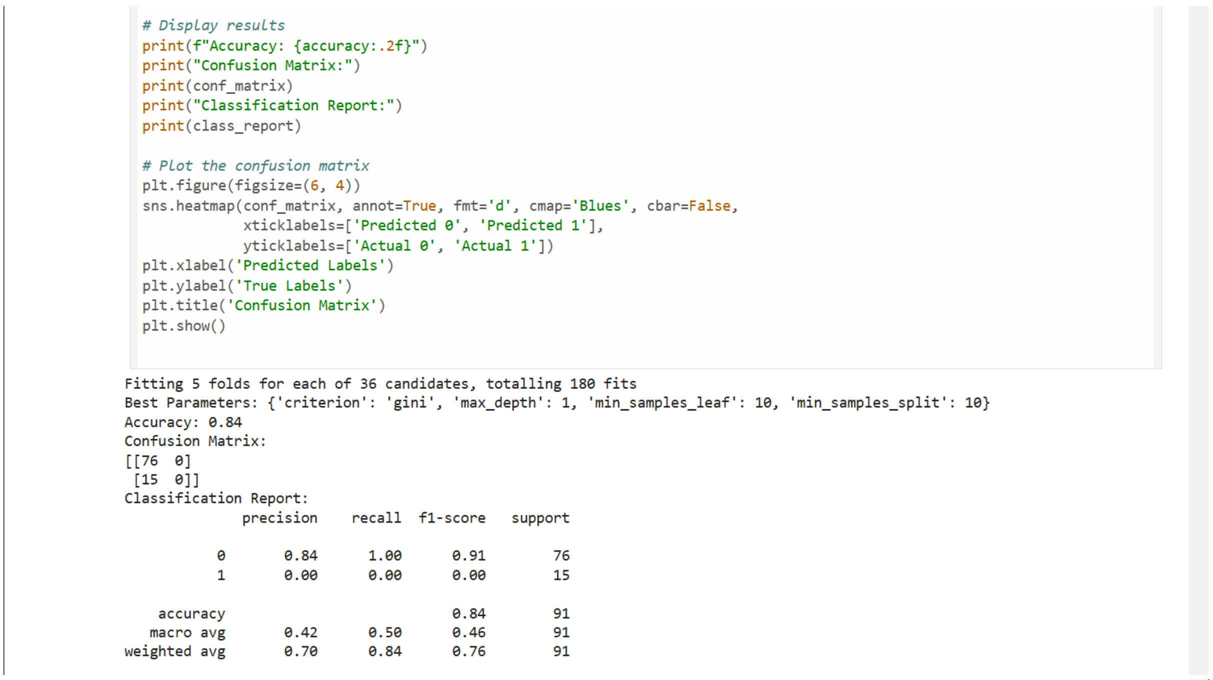
```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best Parameters: {'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 30
0}
Accuracy: 0.84
Confusion Matrix:
[[76  0]
 [15  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        76
           1       0.00      0.00      0.00        15

    accuracy                           0.84        91
   macro avg       0.42      0.50      0.46        91
weighted avg       0.70      0.84      0.76        91
```



USING DECISION TREES:

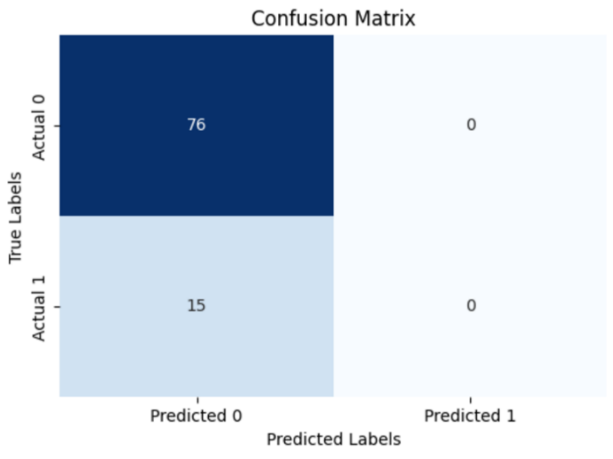
In [ ]:

```python
from sklearn.tree import DecisionTreeClassifier


# Initialize Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'max_depth': [1, 2, 3],                    # Very shallow trees to reduce complexity
    'min_samples_split': [10, 20, 50],          # Limit splits to prevent overfitting
    'min_samples_leaf': [10, 20],               # Limit leaf nodes to reduce complexity
    'criterion': ['gini', 'entropy']           # Two common criteria for splitting nodes
}

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid, cv=5, scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)

# Get the best parameters from grid search
print("Best Parameters:", grid_search.best_params_)

# Train the model with the best parameters
best_dt_model = grid_search.best_estimator_

# Make predictions on the test data
y_pred = best_dt_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```



```
# Display results
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)

# Plot the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

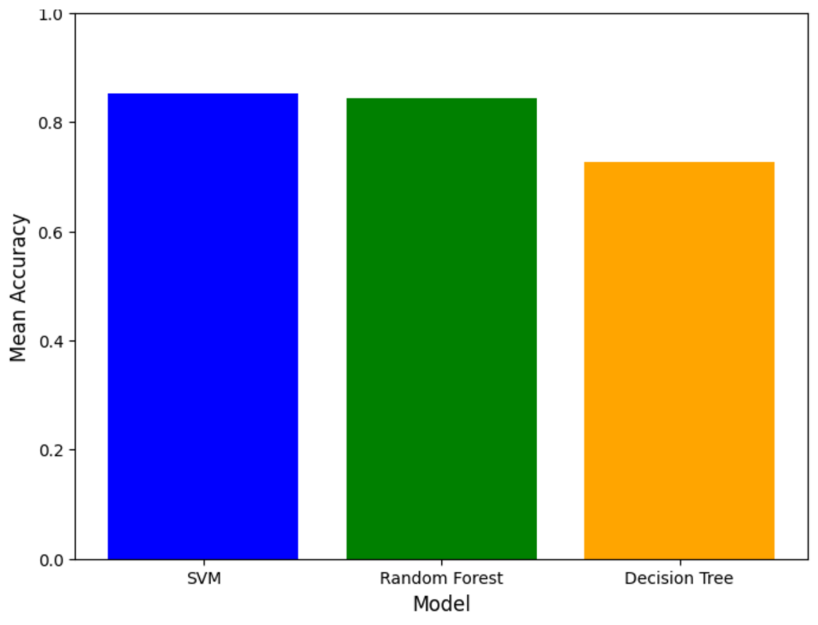
```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'criterion': 'gini', 'max_depth': 1, 'min_samples_leaf': 10, 'min_samples_split': 10}
Accuracy: 0.84
Confusion Matrix:
[[76  0]
 [15  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        76
           1       0.00      0.00      0.00        15

    accuracy                           0.84        91
   macro avg       0.42      0.50      0.46        91
weighted avg       0.70      0.84      0.76        91
```



Comparison:

Comparing the three models used and analysing their respective accuracies.
Clearly, from highest accuracy is of the Support Vector Model with the accuracy of 85%, and Random Forest, the second highest with 84% accuracy, and lastly Decision Tree with the least accuracy of 73%.

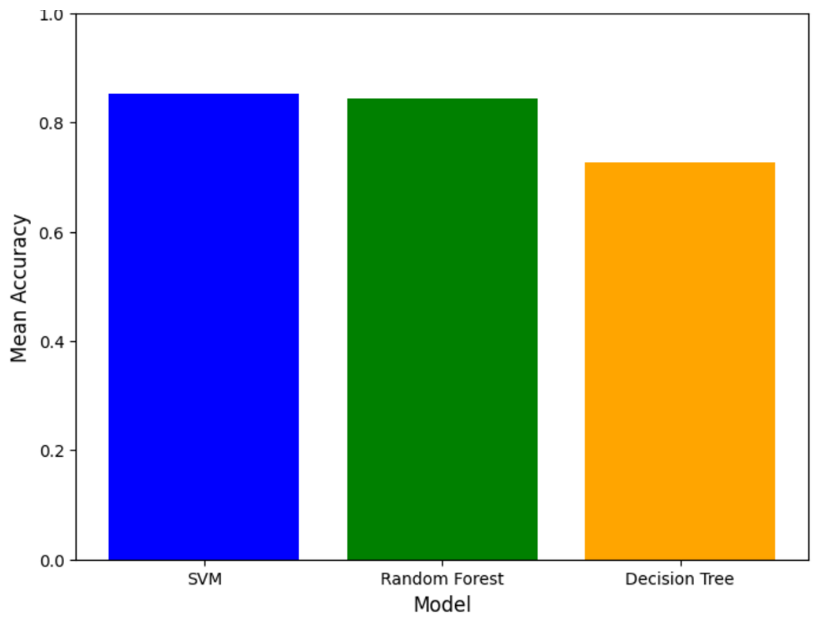
```
In [ ]:  # Standardize the data for SVM (not critical for Decision Trees or Random Forest)
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)

         # Initialize the models
         svm_model = SVC(random_state=42)
         rf_model = RandomForestClassifier(random_state=42)
         dt_model = DecisionTreeClassifier(random_state=42)

         # Create a dictionary of models
         models = {'SVM': svm_model, 'Random Forest': rf_model, 'Decision Tree': dt_model}

         # List to store mean accuracy scores
         mean_accuracies = []

         # Perform cross-validation to get mean accuracy for each model
         for model_name, model in models.items():
             if model_name == 'SVM':
                 # For SVM, use the scaled data
                 accuracy = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='accuracy')
             else:
                 # For Random Forest and Decision Tree, no scaling needed
                 accuracy = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')

             mean_accuracies.append(accuracy.mean())

         # Plot the bar graph for comparison
         plt.figure(figsize=(8, 6))
         plt.bar(models.keys(), mean_accuracies, color=['blue', 'green', 'orange'])
         plt.title('Comparison of Model Performance (Mean Accuracy)', fontsize=14)
         plt.xlabel('Model', fontsize=12)
         plt.ylabel('Mean Accuracy', fontsize=12)
         plt.ylim([0, 1])  # Set y-axis limit from 0 to 1 (accuracy range)
         plt.show()

         # Print the mean accuracies for reference
         for model_name, mean_acc in zip(models.keys(), mean_accuracies):
             print(f"{model_name} Mean Accuracy: {mean_acc:.2f}")
```



```
SVM Mean Accuracy: 0.85
Random Forest Mean Accuracy: 0.84
Decision Tree Mean Accuracy: 0.73
```

RESULT

In this machine learning project, three models—Support Vector Machine (SVM), Random Forest, and Decision Tree—were evaluated based on their mean accuracy scores to determine their performance on the given dataset. Here's a detailed analysis of the results:



```
SVM Mean Accuracy: 0.85
Random Forest Mean Accuracy: 0.84
Decision Tree Mean Accuracy: 0.73
```

Support Vector Machine (SVM):
The SVM model achieved a mean accuracy of 0.85, making it the best-performing model in this study. SVM's strength lies in its ability to find the optimal hyperplane for classification, particularly effective in high-dimensional spaces. Its superior accuracy suggests that the dataset was well-suited to SVM's ability to handle complex decision boundaries.

Random Forest:
The Random Forest model closely followed with a mean accuracy of 0.84. Random Forest combines multiple decision trees using an ensemble approach, which enhances robustness and generalization. Its performance, comparable to SVM, indicates that the dataset benefited from the diverse decision trees contributing to predictions. However, its slightly lower accuracy suggests that SVM might handle subtle complexities in the data better.

Decision Tree:
The Decision Tree model recorded a mean accuracy of 0.73, the lowest among the three. While Decision Trees are straightforward and interpretable, their tendency to overfit smaller datasets or fail to capture complex relationships likely contributed to the reduced accuracy. This result highlights the limitations of using a single tree compared to ensemble methods like Random Forest.

CONCLUSION

The results of this machine learning project highlight the strengths and limitations of the three models—Support Vector Machine (SVM), Random Forest, and Decision Tree—in terms of predictive accuracy. Among these, SVM demonstrated the highest performance with a mean accuracy of 85%, making it the most effective model for this dataset. SVM's capability to handle high-dimensional feature spaces and its robustness in identifying optimal decision boundaries likely contributed to its superior accuracy. This makes SVM a strong candidate for scenarios requiring precise and reliable classification.

The Random Forest model performed comparably, achieving a mean accuracy of 84%. Random Forest, with its ensemble of decision trees, provides a balance between performance and robustness by reducing the risk of overfitting. Its high accuracy suggests that the dataset benefited from the diversity and aggregation inherent in Random Forest. However, the slight difference in performance between SVM and Random Forest indicates that SVM might have an edge in capturing the dataset's more intricate relationships.

The Decision Tree model, on the other hand, lagged with a mean accuracy of 73%. While Decision Trees are simple and interpretable, their performance here was limited by a higher tendency to overfit and an inability to generalize as effectively as ensemble methods or SVM.

In conclusion, both SVM and Random Forest emerge as reliable models for this task, with SVM slightly outperforming Random Forest. Decision Trees, while easier to understand, may require additional techniques, such as pruning or assembling, to improve their predictive capabilities. These findings underscore the importance of selecting advanced algorithms for datasets requiring high accuracy and nuanced predictions.

BIBLIOGRAPHY

1. Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), 115- 118.
2. Tschandl, P., Rosendahl, C., & Kittler, H. (2018). The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Scientific Data, 5, 180161.
3. Brinker, T. J., Hekler, A., Enk, A. H., & von Kalle, C. (2019). Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task. European Journal of Cancer, 113, 47-54.
4. Haenssle, H. A., Fink, C., Schneiderbauer, R., Toberer, F., Buhl, T., Blum, A., ... & Hofmann- Wellenhof, R. (2018). Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 157 dermatologists. Annals of Oncology, 29(8), 1836-1842.
5. Codella, N., Gutman, D., Celebi, M. E., Helba, B., Marchetti, M. A., Dusza, S. W., ... & Halpern,
A. (2018). Skin lesion analysis toward melanoma detection: A challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), hosted by the International Skin Imaging Collaboration (ISIC). arXiv preprint arXiv:1803.10417.
6. Han, S. S., Kim, M. S., Lim, W., Park, G. H., Park, I., Chang, S. E., ... & Lee, H. (2018). Classification of the clinical images for benign and malignant cutaneous tumors using a deep learning algorithm. Journal of Investigative Dermatology, 138(7), 1529-1538.
7. Yu, L., Chen, H., Dou, Q., Qin, J., Heng, P. A., & Zheng, G. (2017). Automated melanoma recognition in dermoscopy images via very deep residual networks. IEEE Transactions on Medical Imaging, 36(4), 994-1004.

Project Git-Hub Link:
https://github.com/pragattee/ML_PROJECT

THANK YOU!