Introduction:
In healthcare, events such as heart attacks and accidents present critical challenges. Preventing these incidents is always preferable to treating them after they occur. Timely intervention can often mean the difference between life and death. This report focuses on developing predictive models to assist healthcare providers in identifying patients who are at high risk for heart attacks and accidents based on their medical history and behavioral patterns. By predicting the risk of a patient, healthcare professionals can take timely actions that could potentially save more lives.

The dataset used for this analysis contains hourly records of 500 patients over a 30-day period. Each record includes various health and behavioral metrics, such as hypertension score, alertness levels, intoxication levels, and physical activity. In addition, the dataset provides information on risk factors, including family medical history, smoking habits, obesity, and past heart attacks. Together, this data creates a comprehensive profile of each patient's health and lifestyle.

Objective:
Designing a Rule-Based Decision Support System: Using predefined criteria, the system will calculate a risk score to predict the likelihood of heart attacks or accidents. The rules will be based on evidence based metrics from medical knowledge such as hypertension score and previous heart attack history.

Implementing a Machine Learning Model: In addition to the rule-based approach, we will implement a machine learning solution trained on similar datasets. This model will learn from patterns in patient data to predict the likelihood of heart attacks or accidents.

Data Analysis:

Statistics for alertness: Mean: 0.76 Median: 0.88 Mode: -26.98 Standard Deviation: 0.91	Statistics for hypertension: Mean: 0.30 Median: 0.22 Mode: 0.00 Standard Deviation: 0.30	Statistics for intoxication: Mean: 0.02 Median: 0.01 Mode: 0.00 Standard Deviation: 0.03
Statistics for smoker: Mean: 0.49 Median: 0.48 Mode: 0.00 Standard Deviation: 0.28	Statistics for overweight: Mean: 0.51 Median: 0.50 Mode: 0.00 Standard Deviation: 0.29	Statistics for family_history: Mean: 0.50 Median: 0.50 Mode: 0.01 Standard Deviation: 0.29

Statistics for goof_ball: Mean: 0.49 Median: 0.49 Mode: 0.00 Standard Deviation: 0.30	Action Breakdown: sleep: 33.37% work: 25.70% alcohol: 16.49% nothing: 14.19% coffee: 10.23% patient died: 0.01%	Occurrence Counts: Heart Attacks: 596 Accidents: 2606 Deaths: 50 Heart Attack Rate: 0.17% Accident Rate: 0.76% Death Rate: 0.01%

Table 1: Descriptive Statistics for 500 patients for Set 2

Rule Based Solution Approach:

To predict the risk of heart attacks, accidents, and death for each patient, this approach uses a rule based solution. The system evaluates and determines the probability of a patient incurring a heart attack, accident, and death. The goal of this solution is to accurately predict these events for each patient using a set of predefined rules based on medical knowledge. An example of this is for predicting heart attacks, the decision variables which are weighted include hypertension score, family history, smoking, overweight, history of heart attacks, and etc. The system also assesses the risk of accidents based on Intoxication and Alertness, with Overweight serving as an additional factor influencing both outcomes.

```
def calculate_heart_attack_risk(patient_data):
    """Calculate heart attack risk score for a patient."""
    score = 0
    if patient_data['hypertension'].any():
        score += 3
    if patient_data['family_history'].any():
        score += 2
    if patient_data['smoker'].any():
        score += 2
    if patient_data['overweight'].any():
        score += 1
    score += 4 * patient_data['heart_attack'].sum()
    return score
```

Figure 1: Weights for Heart Attack Risk Prediction

```python
def calculate_accident_risk(patient_data):
    """Calculate accident risk score for a patient."""
    score = 0
    score += 3 * (patient_data['intoxication'].sum() / 720)
    score += 2 * ((patient_data['alertness'] < 0.5).sum() / 720)
    if patient_data['overweight'].any():
        score += 1
    return score
```

Figure 2: Weights for Accidents Risk Prediction

```python
def calculate_death_risk(heart_attack_risk, accident_risk):
    """Calculate death risk based on heart attack and accident risks."""
    # Assuming death risk is a weighted combination of both risks
    return 0.6 * heart_attack_risk + 0.4 * accident_risk
```

Figure 3: Death Risk Prediction

	Set 2	**Set 3**
Predicted Heart Attack Risk Accuracy	62.20%	64.80%
Predicted Accident Risk Accuracy	71.60%	69.20%
Predicted Death Risk Accuracy	88.80%	88.60%

Table 2: Prediction Accuracy Table for Rule Based Solution
Set 2(Training Set), Set 3(Testing Set)

Machine Learning Based Solution Approach:

This solution employs a supervised machine learning based solution approach to predict the health risks for patients. This solution uses random forest classifiers, which are learning methods that construct multiple decision trees and merge their predictions for improved accuracy. The process involves data preprocessing, feature engineering, model train, and evaluation.

This solution employs a supervised machine learning approach to predict health risks for patients. It uses Random Forest classifiers, which are ensemble learning methods that construct multiple decision trees and merge their predictions for improved accuracy and reduced overfitting. The process involves data preprocessing, feature engineering, model training, and evaluation. The approach separates the prediction task into two main components: heart attack risk and accident risk. For each risk type, relevant features are selected and a dedicated Random Forest model is trained. The final death risk is calculated as an average of the heart attack and accident probabilities, providing a comprehensive risk assessment for each patient. This modular approach allows for targeted feature selection and model optimization for each risk type, potentially leading to more accurate and interpretable results.

```python
def train_and_predict(X, y, X_test):
    """Train a Random Forest model and make predictions."""
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(X_test)

    model = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
    model.fit(X_train_scaled, y_train)

    val_probas = model.predict_proba(X_val_scaled)[:, 1]
    val_preds = (val_probas > 0.5).astype(int)  # Convert probabilities to binary predictions
    roc_auc = roc_auc_score(y_val, val_probas)
    accuracy = calculate_accuracy(y_val, val_preds)
    print(f"Validation ROC AUC Score: {roc_auc:.4f}")
    print(f"Validation Accuracy: {accuracy:.2%}")

    return model.predict_proba(X_test_scaled)[:, 1], roc_auc, accuracy
```

Figure 4: Random Forest Classifier

```python
def engineer_features(df):

    df['hypertension_risk'] = df['hypertension'] * 2
    df['heart_attack_risk_score'] = df['hypertension_risk'] + df['family_history'] + df['smoker'] + df['overweight'] + df['heart_attack_history'] * 2


    df['accident_risk_score'] = df['intoxication_hours'] / 24 + df['low_alertness_hours'] / 24 + df['overweight']

    return df
```

Figure 5: Engineer Features

	Set 2	**Set 3**
Predicted Heart Attack Risk Accuracy	100.00%	100.00%

Predicted Accident Risk Accuracy	80.00%	76.80%
Predicted Death Risk Accuracy	54.60%	100.00%

Table 3: Prediction Accuracy Table for Machine Learning Based Solution Set 2(Training Set), Set 3(Testing Set)

Performance:

Rule Based Solution:

Data processing (per patient per hour):

5 condition checks (hypertension, family history, smoker, overweight, heart attack)

2 comparisons (intoxication, alertness)

3 additions (for risk scores)

Total: 10 operations per hour

Risk calculation (per patient):

3 additions (combining risk factors)

1 multiplication (for heart attack count)

2 divisions (normalizing risks)

1 weighted addition (for death risk)

Total: 7 operations per patient

Total operations: (500 patients *720 hours* 10) + (500 *7) = 3,603,500*

Estimating 5 instructions per operation:

Total instructions: 3,603,500 *5 = 18,017,500*

16.0247 second execution:

MIPS = 18,017,500 / 16,024,700 ≈ 1.124 MIPS

Machine Learning Based Solution:

Feature extraction (per patient):

8 boolean checks (hypertension, family history, etc.)

2 sum operations (intoxication, low alertness hours)

2 divisions (for hours calculation)

Total: 12 operations per patient

Random Forest prediction (assuming 100 trees, 10 levels deep, 9 features):

100 *10* 9 = 9,000 comparisons per patient

3 models (heart attack, accident, death)

Total operations: (500 *12) + (500* 9,000 *3) = 13,506,000*

Estimating 3 instructions per operation (simpler operations on average):

Total instructions: 13,506,000 *3 = 40,518,000*
34.7078 second execution:
MIPS = 40,518,000 / 34,707,800 ≈ 1.167 MIPS

Rule Based Solution	Machine Learning Solution
1.124 MIPS	1.167 MIPS

Table 4: Computation Complexity

The machine learning solution has more MIPS than the rule based solution even though the machine learning solution takes more time to execute. This shows that the machine learning solution was using the CPU more efficiently compared to the rule based solution. This is probably because the machine learning solution is doing more complex operations than the rule based solution.

Conclusion:
In conclusion, the rule based solution did an adequate job at predicting heart attacks and accidents, but the performance of the machine learning model is more accurate and although has a slower execution time in comparison but the machine learning solution is more efficient. The machine learning approach was able to capture more complex patterns and connections between the variables, which likely the weights of the rule based solution could not do. There is a possibility that the weights were not ideal and can be further tweaked to get higher accuracy for the rule based solution. This suggests that machine learning models, with their ability to learn from large amounts of data, are better equipped to handle the nuanced relationships between risk factors like hypertension, intoxication, and smoking.

Appendix A:
Final Code Rule Based Solution:

```python
import pandas as pd
import numpy as np

def load_data(file_path):
    """Load data from the Excel file."""
    try:
        df = pd.read_excel(file_path)
        return df
    except FileNotFoundError:
        print(f"Error: The file at {file_path} was not found.")
        return None
    except Exception as e:
        print(f"An unexpected error occurred while loading the file: {e}")
        return None

def calculate_heart_attack_risk(patient_data):
    """Calculate heart attack risk score for a patient."""
    score = 0
    if patient_data['hypertension'].any():
        score += 3
    if patient_data['family_history'].any():
        score += 2
    if patient_data['smoker'].any():
        score += 2
    if patient_data['overweight'].any():
        score += 1
    score += 4 * patient_data['heart_attack'].sum()
    return score
```

```python
def calculate_accident_risk(patient_data):
    """Calculate accident risk score for a patient."""
    score = 0
    score += 3 * (patient_data['intoxication'].sum() / 720)
    score += 2 * ((patient_data['alertness'] < 0.5).sum() / 720)
    if patient_data['overweight'].any():
        score += 1
    return score

def calculate_death_risk(heart_attack_risk, accident_risk):
    """Calculate death risk based on heart attack and accident risks."""
    # Assuming death risk is a weighted combination of both risks
    return 0.6 * heart_attack_risk + 0.4 * accident_risk
```

```python
def process_patient_data(df):
    """Process data for all 500 patients, considering 24 hours per day for 30 days."""
    results = []
    max_heart_attack_score = 0
    max_accident_score = 0

    for i in range(500):
        patient_data = df.iloc[i*720:(i+1)*720]
        heart_attack_risk = calculate_heart_attack_risk(patient_data)
        accident_risk = calculate_accident_risk(patient_data)

        max_heart_attack_score = max(max_heart_attack_score, heart_attack_risk)
        max_accident_score = max(max_accident_score, accident_risk)

        results.append({
            'patient_id': i,
            'heart_attack_risk': heart_attack_risk,
            'accident_risk': accident_risk,
            'hypertension_history': patient_data['hypertension'].any(),
            'family_history': patient_data['family_history'].any(),
            'smoker': patient_data['smoker'].any(),
            'overweight': patient_data['overweight'].any(),
            'heart_attack_count': patient_data['heart_attack'].sum(),
            'accident_count': patient_data['accident'].sum(),
            'intoxication_hours': patient_data['intoxication'].sum(),
            'low_alertness_hours': (patient_data['alertness'] < 0.5).sum(),
            'actual_heart_attack': patient_data['heart_attack'].any(),
            'actual_accident': patient_data['accident'].any(),
            'actual_death': (patient_data['action'] == 'patient died').any()  # Add this line
        })

    for result in results:
        result['heart_attack_risk'] = (result['heart_attack_risk'] / max_heart_attack_score) * 100
        result['accident_risk'] = (result['accident_risk'] / max_accident_score) * 100
        result['death_risk'] = calculate_death_risk(result['heart_attack_risk'], result['accident_risk'])

    return pd.DataFrame(results)
```

```python
def calculate_accuracy(processed_data):
    """Calculate accuracy of heart attack, accident, and death predictions."""
    heart_attack_threshold = 50  # Adjust this threshold as needed
    accident_threshold = 50  # Adjust this threshold as needed
    death_threshold = 50  # Adjust this threshold as needed

    heart_attack_correct = sum((processed_data['heart_attack_risk'] > heart_attack_threshold) == processed_data['actual_heart_attack'])
    accident_correct = sum((processed_data['accident_risk'] > accident_threshold) == processed_data['actual_accident'])
    death_correct = sum((processed_data['death_risk'] > death_threshold) == processed_data['actual_death'])

    heart_attack_accuracy = (heart_attack_correct / len(processed_data)) * 100
    accident_accuracy = (accident_correct / len(processed_data)) * 100
    death_accuracy = (death_correct / len(processed_data)) * 100

    return heart_attack_accuracy, accident_accuracy, death_accuracy

def print_patient_predictions(processed_data, output_file):
    """Write predictions for each patient to a file."""
    with open(output_file, 'w') as f:
        for _, patient in processed_data.iterrows():
            f.write(f"Patient {patient['patient_id']}:\n")
            f.write(f"  Heart Attack Risk: {patient['heart_attack_risk']:.2f}%\n")
            f.write(f"  Accident Risk: {patient['accident_risk']:.2f}%\n")
            f.write(f"  Death Risk: {patient['death_risk']:.2f}%\n")
            f.write(f"  Actual Heart Attacks: {patient['heart_attack_count']}\n")
            f.write(f"  Actual Accidents: {patient['accident_count']}\n")
            f.write("  Risk Factors:\n")
            f.write(f"    Hypertension History: {'Yes' if patient['hypertension_history'] else 'No'}\n")
            f.write(f"    Family History: {'Yes' if patient['family_history'] else 'No'}\n")
            f.write(f"    Smoker: {'Yes' if patient['smoker'] else 'No'}\n")
            f.write(f"    Overweight: {'Yes' if patient['overweight'] else 'No'}\n")
            f.write(f"    Hours with Intoxication: {patient['intoxication_hours']}\n")
            f.write(f"    Hours with Low Alertness: {patient['low_alertness_hours']}\n")
            f.write("\n")
```

```python
def main():
    file_path = "set3_500_patients.xlsx"
    output_file = "patient_predictions.txt"
    data = load_data(file_path)
    if data is not None:
        processed_data = process_patient_data(data)
        print_patient_predictions(processed_data, output_file)
        print(f"Patient predictions have been written to {output_file}")

        heart_attack_accuracy, accident_accuracy, death_accuracy = calculate_accuracy(processed_data)
        print(f"Heart Attack Prediction Accuracy: {heart_attack_accuracy:.2f}%")
        print(f"Accident Prediction Accuracy: {accident_accuracy:.2f}%")
        print(f"Death Prediction Accuracy: {death_accuracy:.2f}%")

if __name__ == "__main__":
    main()
```

Appendix B:
Final Code Machine Based Solution:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.preprocessing import StandardScaler

def load_data(file_path):
    """Load data from the Excel file."""
    try:
        df = pd.read_excel(file_path)
        return df
    except FileNotFoundError:
        print(f"Error: The file at {file_path} was not found.")
        return None
    except Exception as e:
        print(f"An unexpected error occurred while loading the file: {e}")
        return None

```

```python
def process_patient_data(df):
    """Process data for all 500 patients, considering 24 hours per day for 30 days."""
    results = []

    for i in range(500):
        patient_data = df.iloc[i*720:(i+1)*720]
        results.append({
            'patient_id': i,
            'hypertension': patient_data['hypertension'].any(),
            'family_history': patient_data['family_history'].any(),
            'smoker': patient_data['smoker'].any(),
            'overweight': patient_data['overweight'].any(),
            'intoxication_hours': patient_data['intoxication'].sum(),
            'low_alertness_hours': (patient_data['alertness'] < 0.5).sum(),
            'heart_attack_count': patient_data['heart_attack'].sum(),
            'accident_count': patient_data['accident'].sum(),
            'heart_attack_history': patient_data['heart_attack'].sum() > 0,
            'risk_factor_sum': patient_data['hypertension'].any() +
                               patient_data['family_history'].any() +
                               patient_data['smoker'].any() +
                               patient_data['overweight'].any(),
            'died': (patient_data['action'] == 'died').any(),
        })

    return pd.DataFrame(results)

def engineer_features(df):

    df['hypertension_risk'] = df['hypertension'] * 2
    df['heart_attack_risk_score'] = df['hypertension_risk'] + df['family_history'] + df['smoker'] + df['overweight'] + df['heart_attack_history'] * 2


    df['accident_risk_score'] = df['intoxication_hours'] / 24 + df['low_alertness_hours'] / 24 + df['overweight']

    return df
```

```python
def train_model(X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)

    model = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
    model.fit(X_train_scaled, y_train)

    val_probas = model.predict_proba(X_val_scaled)[:, 1]
    val_preds = (val_probas > 0.5).astype(int)
    roc_auc = roc_auc_score(y_val, val_probas)
    accuracy = calculate_accuracy(y_val, val_preds)
    print(f"Validation ROC AUC Score: {roc_auc:.4f}")
    print(f"Validation Accuracy: {accuracy:.2%}")

    return model, scaler

def predict(model, scaler, X):
    X_scaled = scaler.transform(X)
    return model.predict_proba(X_scaled)[:, 1]

def calculate_accuracy(y_true, y_pred):
    """Calculate accuracy of predictions."""
    return accuracy_score(y_true, y_pred)

def predict_death(heart_attack_probs, accident_probs):
    """Calculate death probability based on heart attack and accident probabilities."""
    return (heart_attack_probs + accident_probs) / 2
```

```python
def print_patient_predictions(processed_data, heart_attack_probs, accident_probs, death_probs, death_preds, output_file):
    """Write predictions for each patient to a file."""
    with open(output_file, 'w') as f:
        for i, patient in processed_data.iterrows():
            f.write(f"Patient {patient['patient_id']}:\n")
            f.write(f"  Heart Attack Risk: {heart_attack_probs[i]:.2f}%\n")
            f.write(f"  Accident Risk: {accident_probs[i]:.2f}%\n")
            f.write(f"  Death Risk: {death_probs[i]:.2f}%\n")
            f.write(f"  Predicted Death: {'Yes' if death_preds[i] else 'No'}\n")
            f.write(f"  Actual Death: {'Yes' if patient['died'] else 'No'}\n")
            f.write(f"  Actual Heart Attacks: {int(patient['heart_attack_count'])}\n")
            f.write(f"  Actual Accidents: {int(patient['accident_count'])}\n")
            f.write("  Risk Factors:\n")
            f.write(f"    Hypertension History: {'Yes' if patient['hypertension'] else 'No'}\n")
            f.write(f"    Family History: {'Yes' if patient['family_history'] else 'No'}\n")
            f.write(f"    Smoker: {'Yes' if patient['smoker'] else 'No'}\n")
            f.write(f"    Overweight: {'Yes' if patient['overweight'] else 'No'}\n")
            f.write(f"    Hours with Intoxication: {patient['intoxication_hours']:.2f}\n")
            f.write(f"    Hours with Low Alertness: {int(patient['low_alertness_hours'])}\n")
            f.write("\n")
```

```python
def main():

    train_file_path = "set2_500_patients.xlsx"

    test_file_path = "set3_500_patients.xlsx"
    output_file = "patient_predictions_set3.txt"


    train_data = load_data(train_file_path)
    if train_data is None:
        return

    processed_train_data = process_patient_data(train_data)
    processed_train_data = engineer_features(processed_train_data)


    heart_attack_features = ['hypertension', 'family_history', 'smoker', 'overweight', 'heart_attack_history', 'heart_attack_risk_score']
    accident_features = ['intoxication_hours', 'low_alertness_hours', 'overweight', 'accident_risk_score']


    X_heart_attack_train = processed_train_data[heart_attack_features]
    X_accident_train = processed_train_data[accident_features]
    y_heart_attack_train = processed_train_data['heart_attack_count'] > 0
    y_accident_train = processed_train_data['accident_count'] > 0


    print("Training Heart Attack Model:")
    heart_attack_model, heart_attack_scaler = train_model(X_heart_attack_train, y_heart_attack_train)

    print("\nTraining Accident Model:")
    accident_model, accident_scaler = train_model(X_accident_train, y_accident_train)


    test_data = load_data(test_file_path)
    if test_data is None:
        return

    processed_test_data = process_patient_data(test_data)
    processed_test_data = engineer_features(processed_test_data)


    X_heart_attack_test = processed_test_data[heart_attack_features]
    X_accident_test = processed_test_data[accident_features]


    heart_attack_probs = predict(heart_attack_model, heart_attack_scaler, X_heart_attack_test)
    accident_probs = predict(accident_model, accident_scaler, X_accident_test)


    death_probs = predict_death(heart_attack_probs, accident_probs)
    death_preds = (death_probs > 50).astype(int)


    heart_attack_accuracy = calculate_accuracy(processed_test_data['heart_attack_count'] > 0, heart_attack_probs > 0.5)
    accident_accuracy = calculate_accuracy(processed_test_data['accident_count'] > 0, accident_probs > 0.5)
    death_accuracy = calculate_accuracy(processed_test_data['died'], death_preds)


    print_patient_predictions(processed_test_data, heart_attack_probs * 100, accident_probs * 100, death_probs, death_preds, output_file)
    print(f"Patient predictions have been written to {output_file}")

    print(f"\nHeart Attack Model Accuracy on Test Set: {heart_attack_accuracy:.2%}")
    print(f"Accident Model Accuracy on Test Set: {accident_accuracy:.2%}")
    print(f"Death Prediction Accuracy on Test Set: {death_accuracy:.2%}")
    print(f"Average Death Risk: {death_probs.mean():.2f}%")
    print(f"Patients with Death Risk > 50%: {(death_probs > 50).sum()} out of {len(death_probs)} patients")
    print(f"Actual Deaths: {processed_test_data['died'].sum()} out of {len(processed_test_data)} patients")

if __name__ == "__main__":
    main()
```