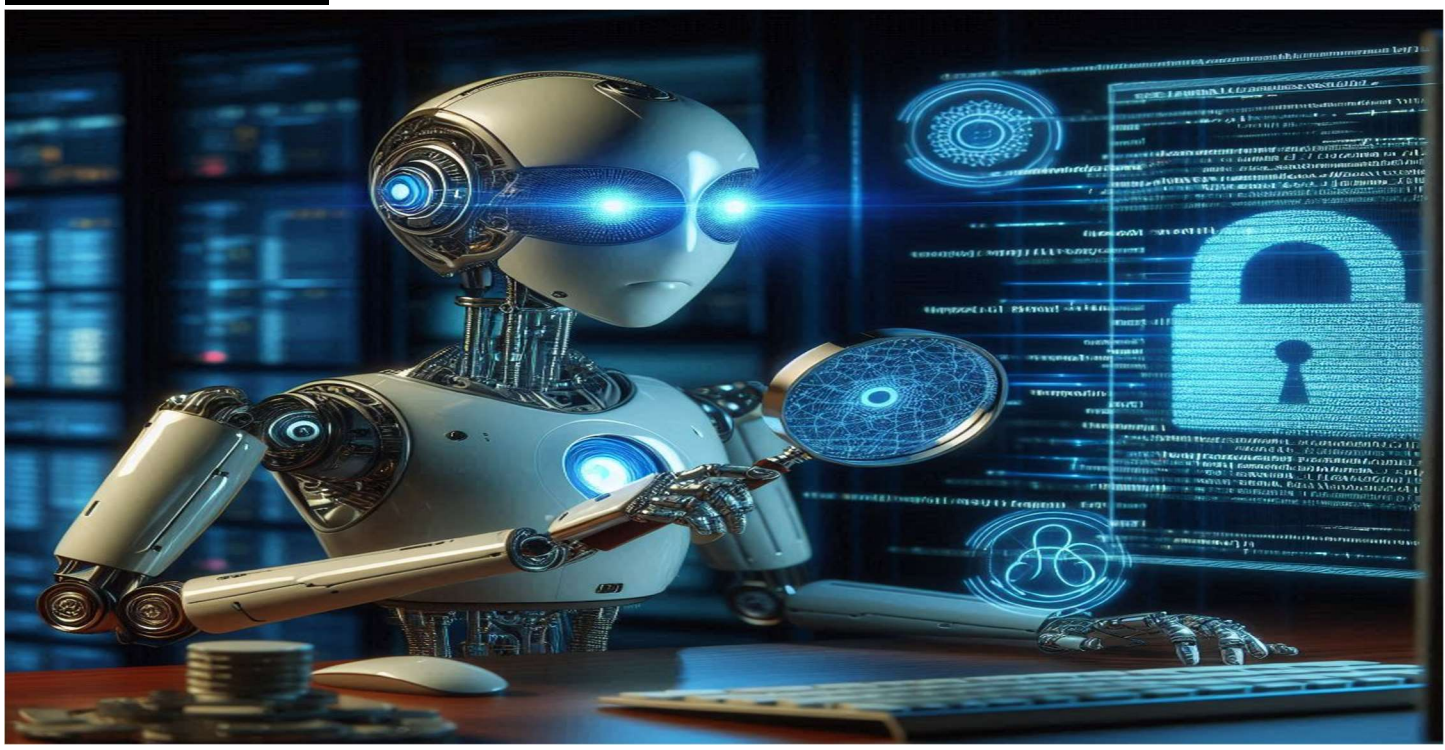


IMPLEMENTING AI DRIVEN THREAT DETECTION

SURAJ GHOSH **2141007017**

NISHANT GAURAV **2141007068**

Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN UNIVERSITY

Bhubaneswar, Odisha, India

Abstract:-

In response to escalating cyber threats, traditional security systems often fail to detect and mitigate sophisticated attacks effectively. This project developed an AI-driven threat detection system designed to enhance cybersecurity in cloud-based environments. Utilizing advanced AI and Machine Learning (ML) techniques, the system analyzes network traffic, user behavior, and system logs to identify anomalies indicative of potential security breaches.

Our methodology included data collection, preprocessing, feature extraction, and the implementation of various ML algorithms. Emphasis was placed on the system's adaptability, allowing continuous learning from new data and improving detection capabilities over time. The system integrates seamlessly with existing security infrastructures, providing real-time alerts and automated response mechanisms.Performance was evaluated using metrics such as accuracy, precision, recall, and the rate of false positives and negatives. The Random Forest model emerged as the most effective, offering superior accuracy and balanced performance. Our findings indicate a significant improvement in threat detection and a reduction in false positives compared to traditional methods.

This project underscores the efficacy of AI in enhancing cybersecurity and addresses challenges such as data privacy and scalability. It highlights the importance of continuous innovation in AI to defend against evolving cyber threats, laying the groundwork for future research and development in this critical domain.

Introduction :-

The complexity and frequency of cyber attacks have increased in the current digital era, making traditional security methods inadequate. Businesses are always looking for cutting-edge ways to safeguard their critical information and keep their systems intact. The use of Artificial Intelligence (AI) and Machine Learning (ML) into cybersecurity techniques is among the most promising developments in this field. AI-driven threat detection uses data analytics and machine learning algorithms to quickly find, assess, and neutralise possible security risks. Artificial intelligence (AI) systems are able to analyse enormous volumes of data, identify anomalous patterns, anticipate potential security breaches, and react to attacks faster and more precisely than manual techniques. Advanced security measures are required because cyberattacks are becoming more sophisticated and complicated. Threat detection powered by AI is essential as it enhances security by identifying threats in real-time and providing immediate responses to mitigate damage. It also improves accuracy by reducing false positives and negatives through learning from vast amounts of data. Additionally, it is crucial to adapt to new threats by continuously evolving and improving through learning from new data and emerging threats.

AI-driven threat detection typically involves several steps to ensure robust security. Firstly, data collection is performed, gathering information from various sources such as network traffic, user

behaviours, and system logs. This data is then analysed using machine learning algorithms to identify patterns that could indicate potential threats. The next step is threat identification, where anomalies are detected, and predictions of possible security breaches are made. Finally, a response is initiated, which can be automatic or semi-automatic, to mitigate the identified threats.

This project aims to develop and implement an AI-driven system for real-time cyber threat detection and response. By integrating AI with cloud-based security solutions, it provides adaptive measures that evolve with emerging threats. The system leverages AI and machine learning to learn from data, recognize patterns, and improve predictive capabilities. Addressing challenges like data privacy and model integration, the project demonstrates the benefits of AI-driven threat detection through analysis and case studies. The goal is to enhance cybersecurity efficiency and effectiveness, offering robust defenses against sophisticated threats and valuable insights for security professionals and decision-makers.

Problem Statement :-

In today's digital age, cyber threats are becoming increasingly sophisticated and frequent, posing significant risks to organizations and individuals alike. Traditional security systems are often inadequate in detecting and mitigating these advanced threats, leading to potential data breaches, financial losses, and compromised privacy. This project aims to address the limitations of conventional security measures by developing an AI-driven threat detection system. The primary objective is to enhance cybersecurity in cloud-based environments through the application of advanced Artificial Intelligence (AI) and Machine Learning (ML) techniques.
The system will analyze extensive datasets, including network traffic, user behavior, and system logs, to identify and predict potential security breaches. Key challenges include ensuring the system's adaptability to evolving threats, maintaining data privacy, and achieving seamless integration with existing security infrastructures. The project seeks to improve detection accuracy, reduce false positives, and provide real-time alerts and automated responses to detected threats, thereby offering a robust and proactive defense mechanism against cyber attacks.

Solution :-

The proposed solution is an AI-driven threat detection system designed to enhance real-time threat identification and response. This system leverages machine learning algorithms to analyze vast datasets, identifying and predicting threats with high accuracy. The system integrates with existing cybersecurity frameworks, providing adaptive and proactive defenses against emerging threats.
Key Features of the Solution:
1. Real-Time Monitoring: Continuous surveillance of network traffic and user activities, enabling the detection of threats as they occur.
2. Adaptive Learning: Machine learning algorithms that evolve with new data, improving the system's ability to detect and mitigate novel threats.

3. Automated Response: Immediate actions taken to isolate affected systems, block malicious traffic, and notify security personnel, reducing the time between threat detection and mitigation. *4. Integration with Existing Systems:* Seamless incorporation into current cybersecurity infrastructures, enhancing their capabilities without requiring significant changes. The solution aims to address the limitations of traditional security measures by providing a proactive and adaptive approach to cybersecurity. By leveraging AI and ML, the system offers robust defenses against sophisticated cyber threats, ensuring the protection of sensitive data and the integrity of organizational systems.

Implementation and Result :-

The implementation of the AI-driven threat detection system involves several key steps:

1. **Data Collection**: Aggregating data from various sources, including network traffic, user behaviors, and system logs. This data is essential for training the machine learning models and forms the basis for threat detection.

2. **Data Preprocessing**: Cleaning and preparing the data for analysis. This involves removing irrelevant information, handling missing values, and normalizing the data to ensure consistency.
Step 1: Load and Explore the Data

AI-Driven Threat Detection Implementation

Step 1: Load and Explore the Data

```
data = pd.read_csv('Train_data.csv')

# Display the first few rows of the dataset
print(data.head())
print(data.info())
print(data.describe())
```

Step 2: Data Preprocessing Convert categorical columns to numerical. Handle any missing values if present. Normalize the features.

```
# Convert categorical columns to numerical
categorical_cols = ['protocol_type', 'service', 'flag', 'class']
for col in categorical_cols:
    data[col] = data[col].astype('category').cat.codes

# Separate features and labels
X = data.drop(['class'], axis=1)
```

```
y = data['class']

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=2)
```

Step 3: Train and Evaluate Models

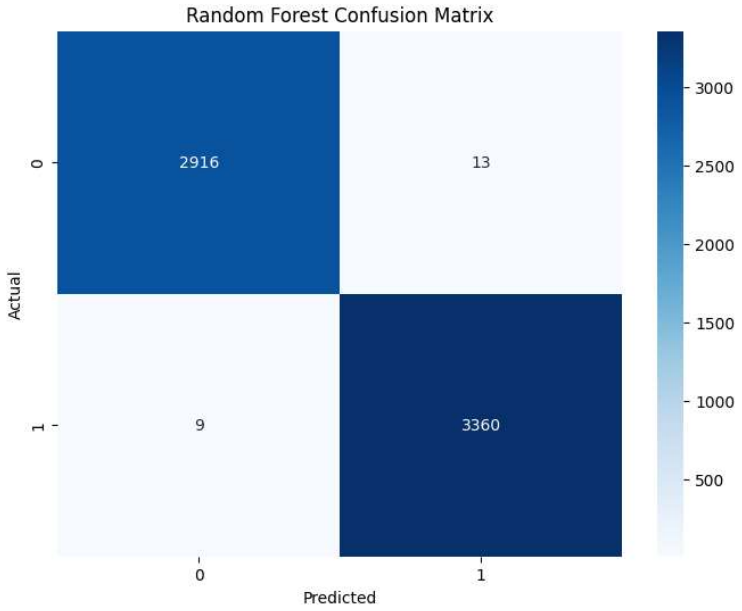
3.1 Random Forest

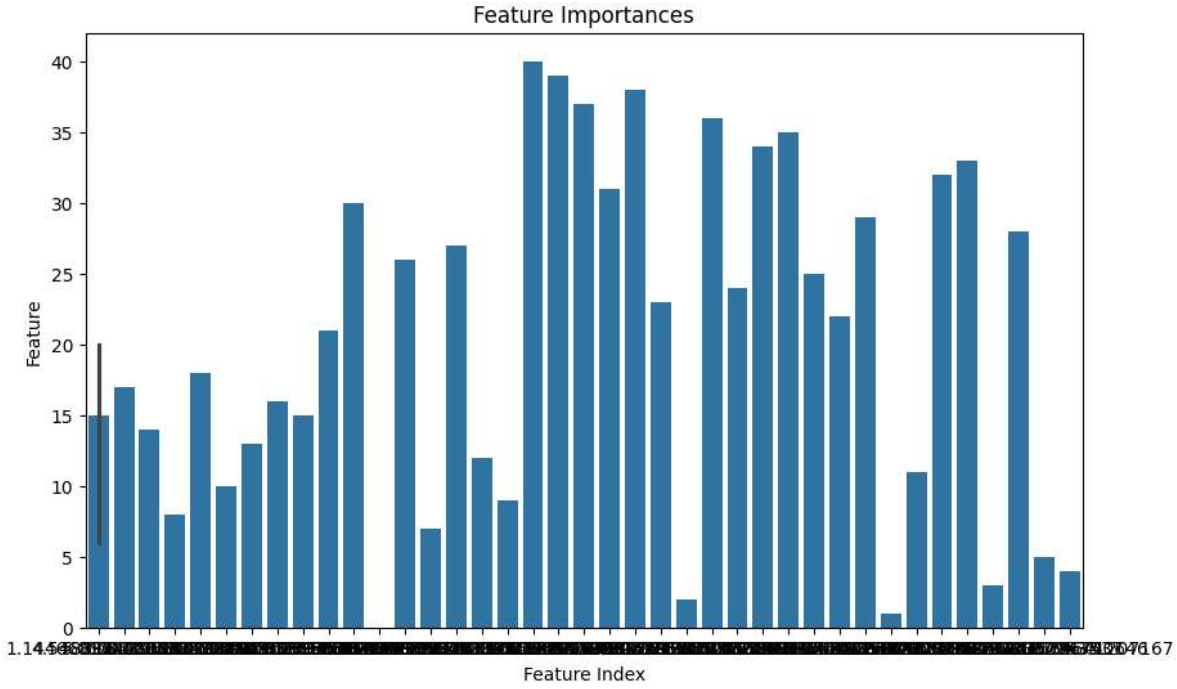
```
from sklearn.ensemble import RandomForestClassifier
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score

# Initialize the model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
start = time.time()
rf_model.fit(X_train, y_train)
end = time.time()
print("Random Forest Training time:", end - start)

# Make predictions
start = time.time()
y_pred_rf = rf_model.predict(X_test)
end = time.time()
print("Random Forest Prediction time:", end - start)

# Evaluate the model
rf_precision = precision_score(y_test, y_pred_rf, average='macro')
rf_recall = recall_score(y_test, y_pred_rf, average='macro')
rf_f1 = f1_score(y_test, y_pred_rf, average='macro')
rf_accuracy = accuracy_score(y_test, y_pred_rf)

print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Random Forest Precision:", rf_precision)
print("Random Forest Recall:", rf_recall)
print("Random Forest F1-score:", rf_f1)
print("Random Forest Accuracy:", rf_accuracy)

# Calculate model accuracy
```

```
rf_train_score = rf_model.score(X_train, y_train)
rf_test_score = rf_model.score(X_test, y_test)
print(f"Train Score: {rf_train_score}")
print(f"Test Score: {rf_test_score}")

# Visualization
# Confusion Matrix Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Blues')
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Feature Importance
# Use indices if feature names are not available
feature_importances = rf_model.feature_importances_
importance_df = pd.DataFrame({'Feature': range(len(feature_importances)), 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importances')
plt.xlabel('Feature Index')
plt.show()
```

Output:

Random Forest Training time: 1.7076048851013184
Random Forest Prediction time: 0.08814382553100586
Random Forest Confusion Matrix:
[[2916 13]
[9 3360]]
Random Forest Precision: 0.9965344705694543
Random Forest Recall: 0.9964451046388146
Random Forest F1-score: 0.9964893796106726
Random Forest Accuracy: 0.9965068275643061
Train Score: 1.0
Test Score: 0.9965068275643061





3.2 Decision Tree

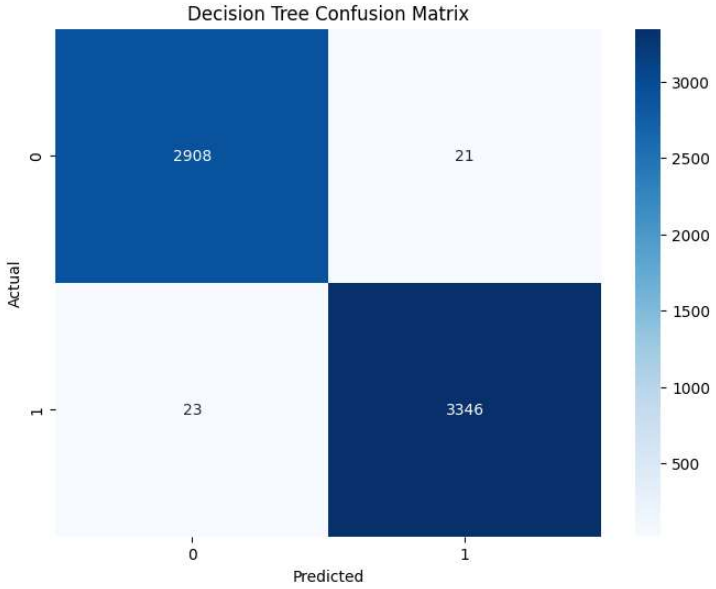
```
from sklearn.tree import DecisionTreeClassifier, plot_tree # Import plot_tree

dt_model = DecisionTreeClassifier()

start = time.time()
dt_model.fit(X_train, y_train)
```

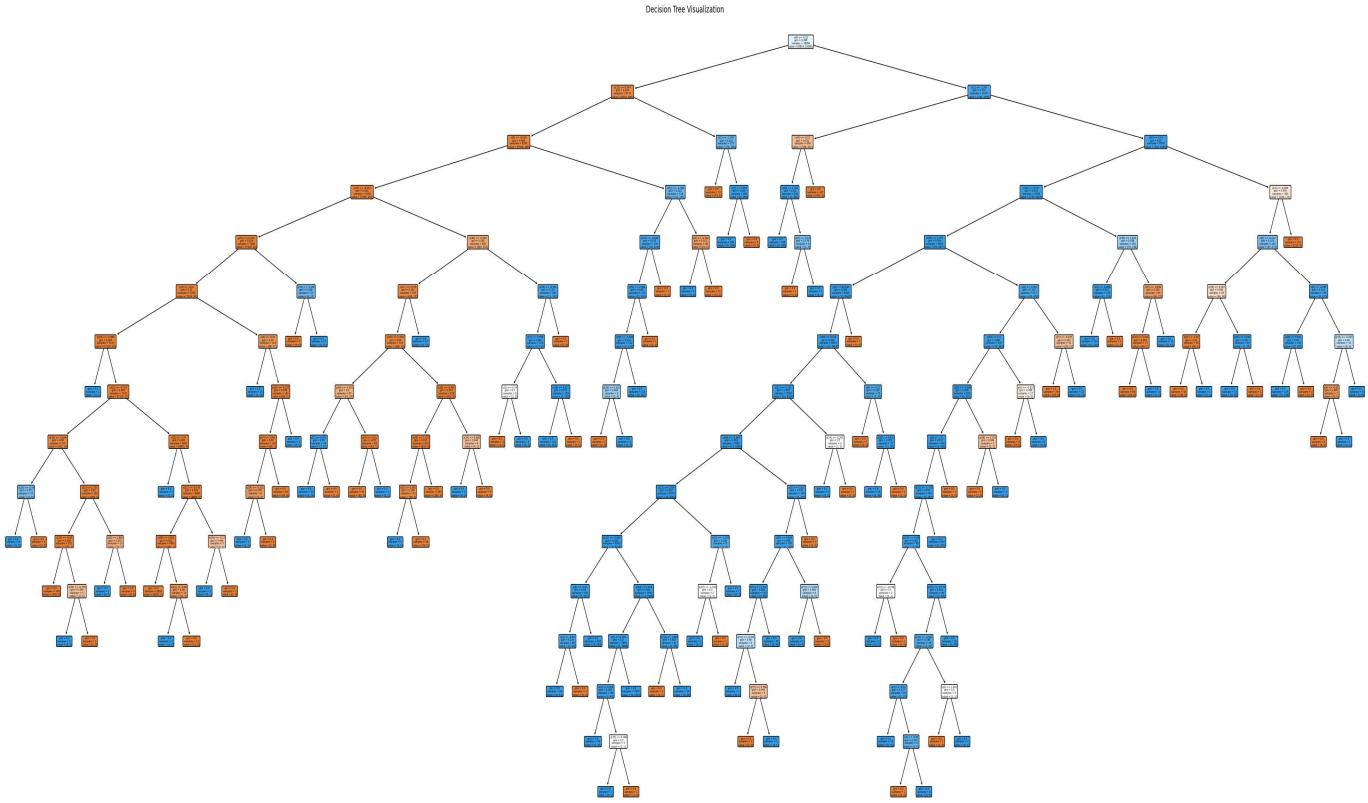
```
end = time.time()
print("Decision Tree Training time:", end - start)

start = time.time()
y_pred_dt = dt_model.predict(X_test)
end = time.time()
print("Decision Tree Prediction time:", end - start)

# Train and evaluate Decision Tree
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
dt_precision = precision_score(y_test, y_pred_dt, average='macro')
dt_recall = recall_score(y_test, y_pred_dt, average='macro')
dt_f1 = f1_score(y_test, y_pred_dt, average='macro')
dt_accuracy = accuracy_score(y_test, y_pred_dt)


# Evaluate Decision Tree
print("Decision Tree Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Decision Tree Precision:", precision_score(y_test, y_pred_dt, average='macro'))
print("Decision Tree Recall:", recall_score(y_test, y_pred_dt, average='macro'))
print("Decision Tree F1-score:", f1_score(y_test, y_pred_dt, average='macro'))
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
# Visualize the Decision Tree
plt.figure(figsize=(50, 20))
plot_tree(dt_model, filled=True, rounded=True) # Use plot_tree directly
plt.title('Decision Tree Visualization')
plt.show()
# Confusion Matrix Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_dt), annot=True, fmt='d', cmap='Blues')
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Output:

Decision Tree Training time: 0.13943195343017578
Decision Tree Prediction time: 0.003368854522705078
Decision Tree Confusion Matrix:
[[2908 21]
 [23 3346]]
Decision Tree Precision: 0.9929579213100196
Decision Tree Recall: 0.9930016829484096

Decision Tree F1-score: 0.9929797005342391
Decision Tree Accuracy: 0.9930136551286123





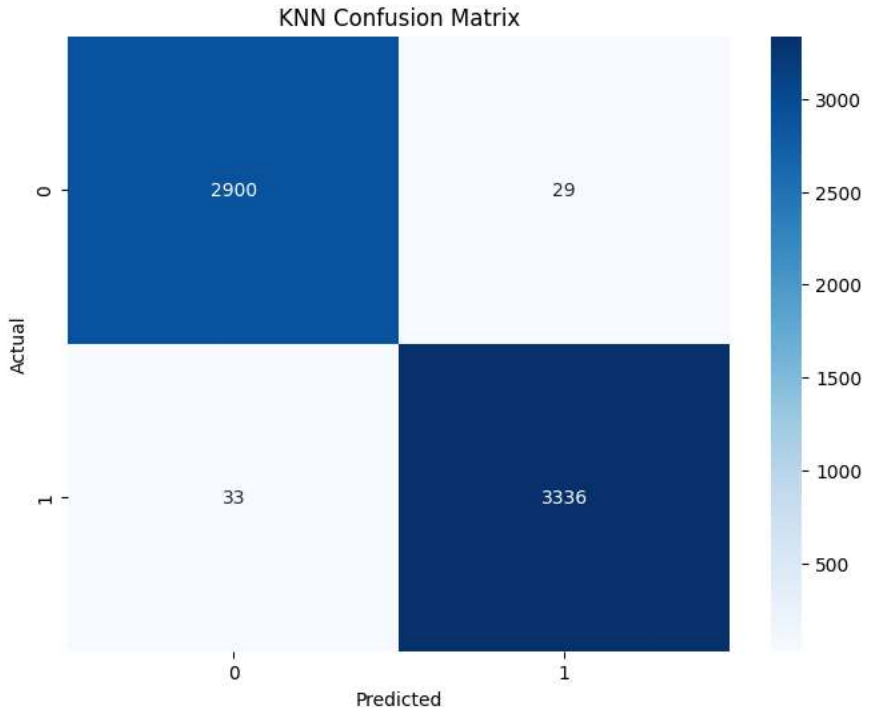
3.3 K-Nearest Neighbors

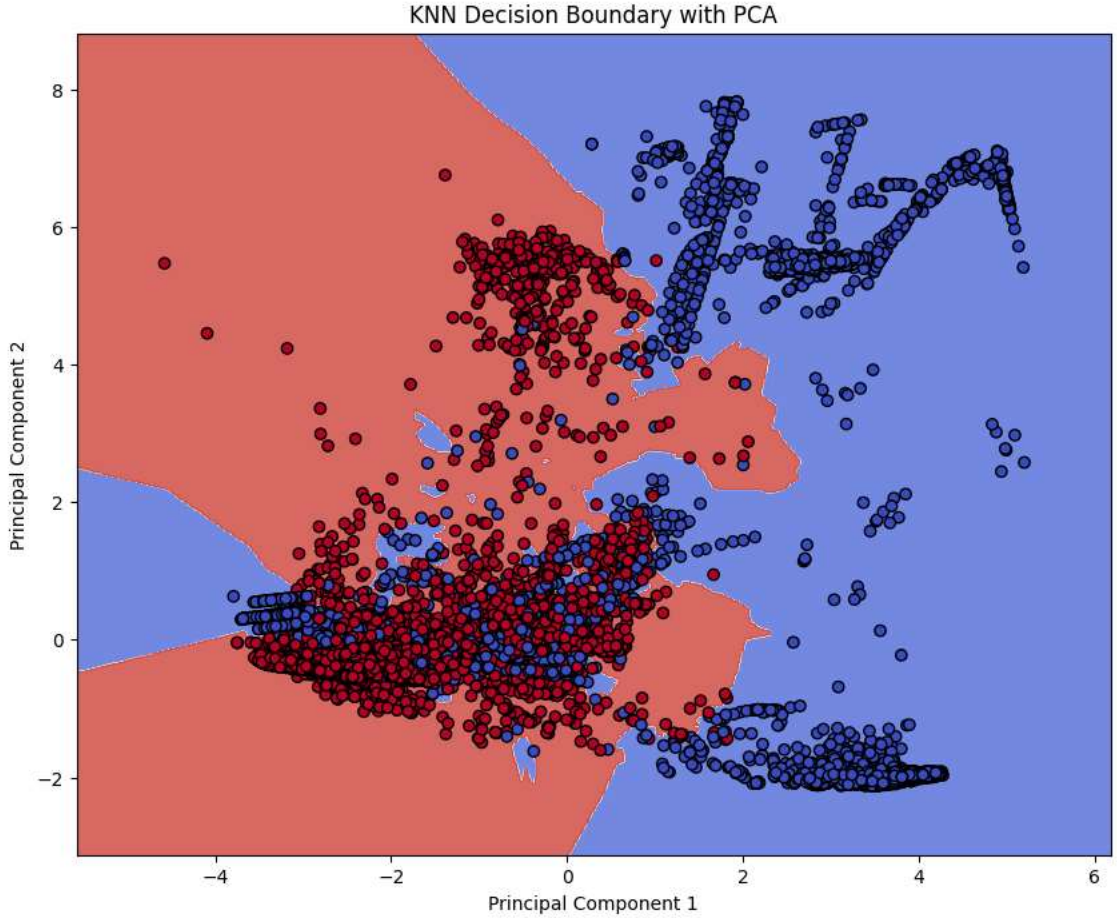
```
from sklearn.neighbors import KNeighborsClassifier
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import numpy as np
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score

knn_model = KNeighborsClassifier()

start = time.time()
knn_model.fit(X_train, y_train)
end = time.time()
print("KNN Training time:", end - start)

start = time.time()
y_pred_knn = knn_model.predict(X_test)
end = time.time()
print("KNN Prediction time:", end - start)

# Train and evaluate K-Nearest Neighbors
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
knn_precision = precision_score(y_test, y_pred_knn, average='macro')
knn_recall = recall_score(y_test, y_pred_knn, average='macro')
knn_f1 = f1_score(y_test, y_pred_knn, average='macro')
knn_accuracy = accuracy_score(y_test, y_pred_knn)

# Evaluate KNN
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("KNN Precision:", precision_score(y_test, y_pred_knn, average='macro'))
print("KNN Recall:", recall_score(y_test, y_pred_knn, average='macro'))
print("KNN F1-score:", f1_score(y_test, y_pred_knn, average='macro'))
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))

# Calculate model accuracy
knn_train_score = knn_model.score(X_train, y_train)
knn_test_score = knn_model.score(X_test, y_test)
print(f"Train Score: {knn_train_score}")
print(f"Test Score: {knn_test_score}")

# Visualization
# Confusion Matrix Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt='d', cmap='Blues')
```

```
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# PCA for Decision Boundary Visualization (for 2D plot)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_model_pca = KNeighborsClassifier()
knn_model_pca.fit(X_train_pca, y_train)

h = .02  # step size in the mesh
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

Z = knn_model_pca.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(10, 8))
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap=plt.cm.coolwarm, edgecolors='k')
plt.title('KNN Decision Boundary with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

Output:

KNN Training time: 0.0036284923553466797
KNN Prediction time: 0.8549766540527344
KNN Confusion Matrix:
[[2900 29]
[33 3336]]
KNN Precision: 0.9900652968297932
KNN Recall: 0.9901519092247604
KNN F1-score: 0.9901081978871129
KNN Accuracy: 0.9901556049539536
Train Score: 0.9939134116650788
Test Score: 0.9901556049539536

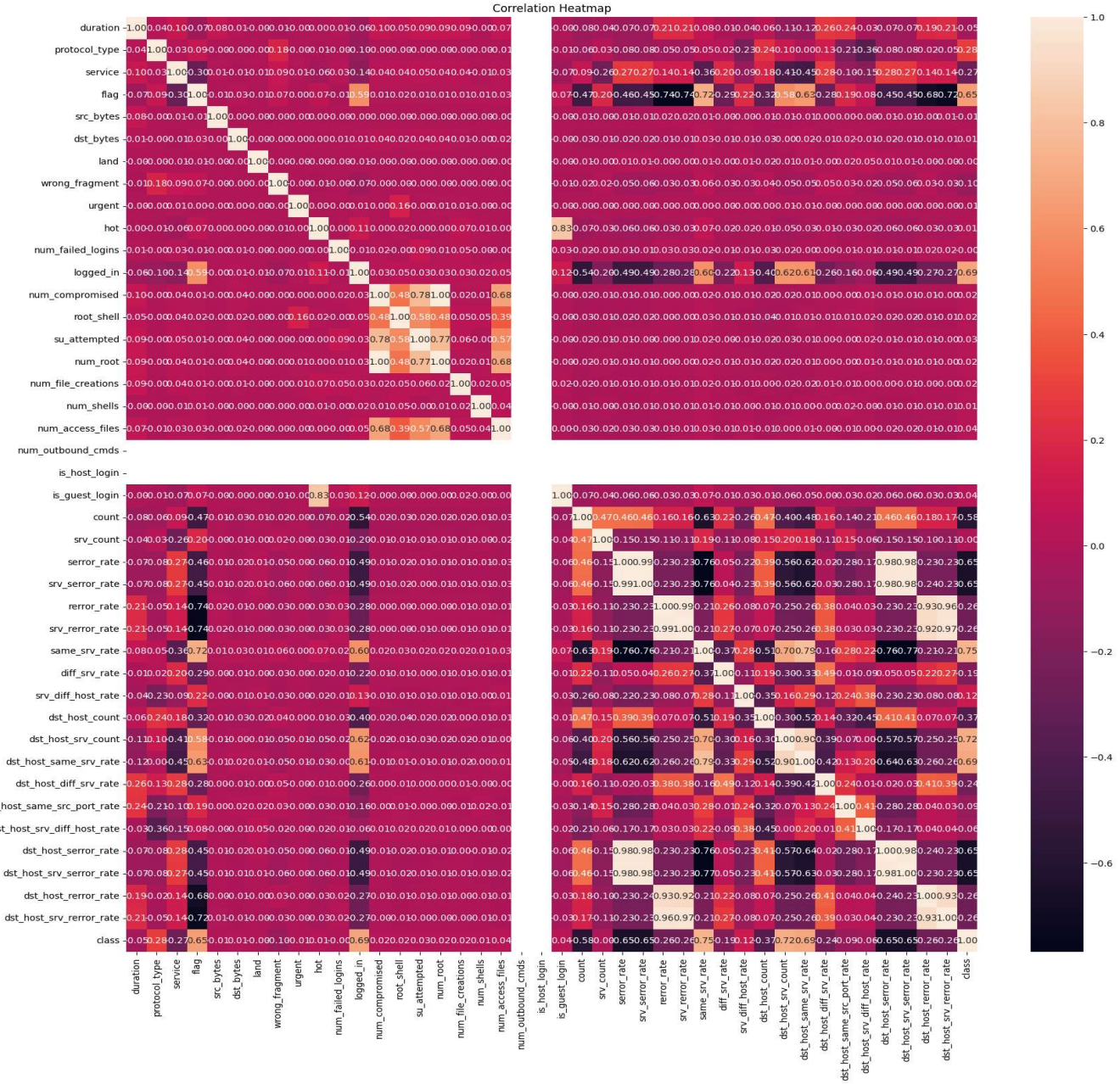

KNN Confusion Matrix
2900
29
33
3336
Actual
Predicted
0
1
3000
2500
2000
1500
1000
500


KNN Decision Boundary with PCA
Principal Component 1
Principal Component 2
8
6
4
2
0
−2
−4
−2
0
2
4
6

Step 4: Visualization

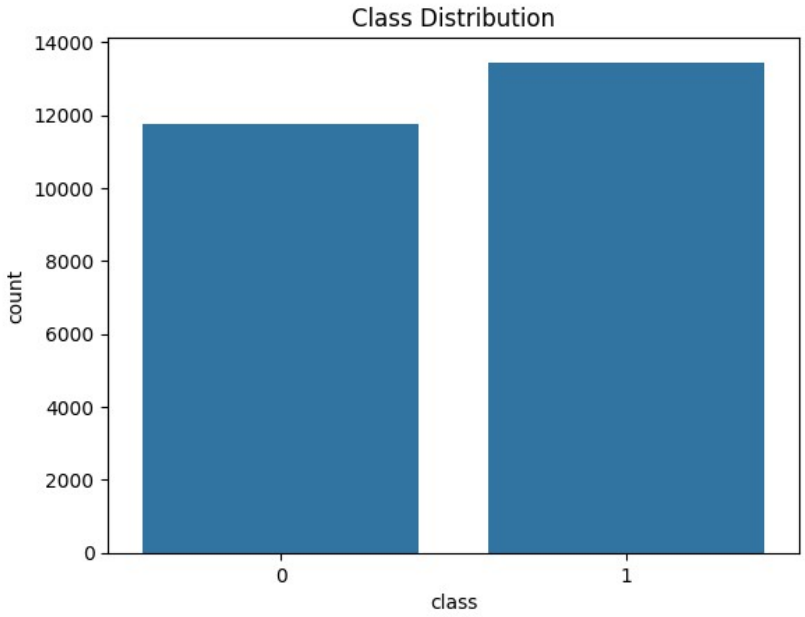
4.1 Correlation Heatmap

```
plt.figure(figsize=(20, 20))
sns.heatmap(data.corr(), annot=True, fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



4.2 Class Distribution

```
sns.countplot(x=y)
plt.title('Class Distribution')
plt.show()
```

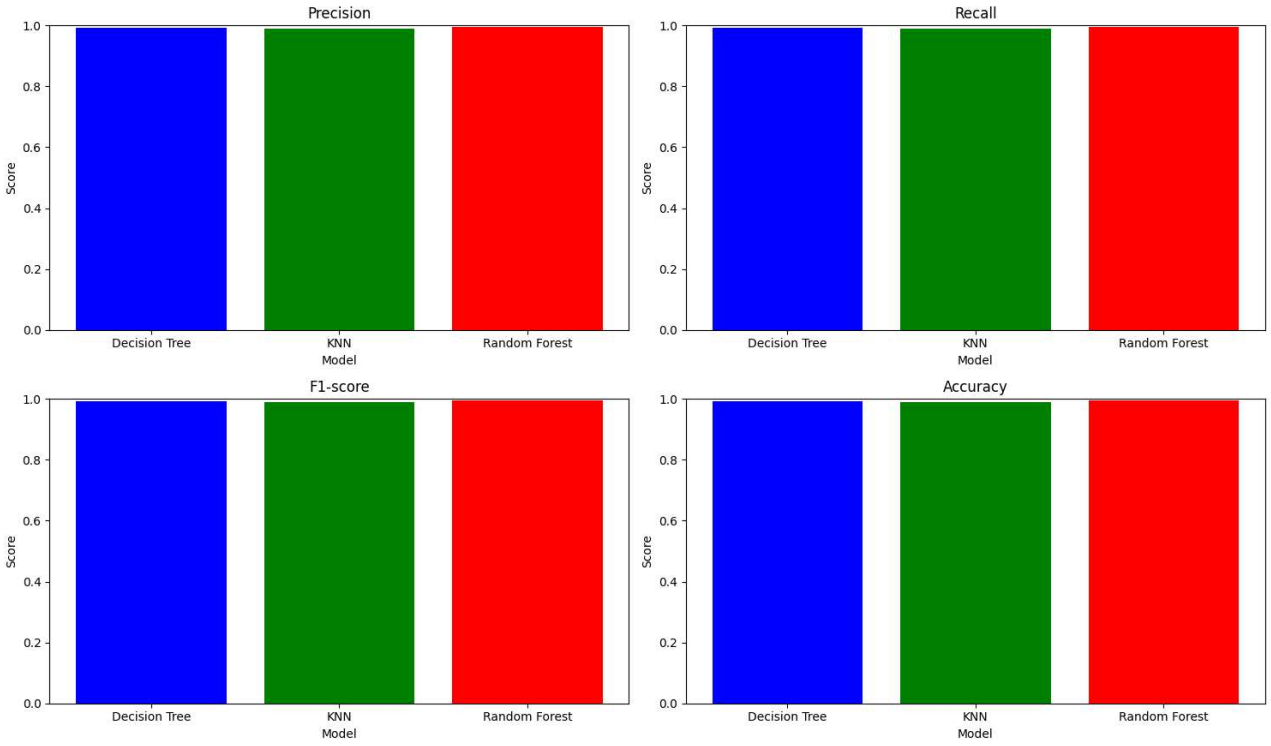


Comparision of ML Models:

```
import pandas as pd
import matplotlib.pyplot as plt

# Collecting the metrics in a dictionary
performance_metrics = {
    'Model': ['Decision Tree', 'KNN', 'Random Forest'],
    'Precision': [dt_precision, knn_precision, rf_precision],
    'Recall': [dt_recall, knn_recall, rf_recall],
    'F1-score': [dt_f1, knn_f1, rf_f1],
    'Accuracy': [dt_accuracy, knn_accuracy, rf_accuracy]
}

# Convert to DataFrame
performance_df = pd.DataFrame(performance_metrics)

# Plotting the performance metrics
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Model Performance Comparison')
```

```
# Precision
axes[0, 0].bar(performance_df['Model'], performance_df['Precision'], color=['blue','green', 'red'])
axes[0, 0].set_title('Precision')
axes[0, 0].set_ylim(0, 1)
# Recall
axes[0, 1].bar(performance_df['Model'], performance_df['Recall'], color=['blue','green', 'red'])
axes[0, 1].set_title('Recall')
axes[0, 1].set_ylim(0, 1)
# F1-score
axes[1, 0].bar(performance_df['Model'], performance_df['F1-score'], color=['blue','green', 'red'])
axes[1, 0].set_title('F1-score')
axes[1, 0].set_ylim(0, 1)
# Accuracy
axes[1, 1].bar(performance_df['Model'], performance_df['Accuracy'], color=['blue','green', 'red'])
axes[1, 1].set_title('Accuracy')
axes[1, 1].set_ylim(0, 1)

for ax in axes.flat:
    ax.set_ylabel('Score')
    ax.set_xlabel('Model')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

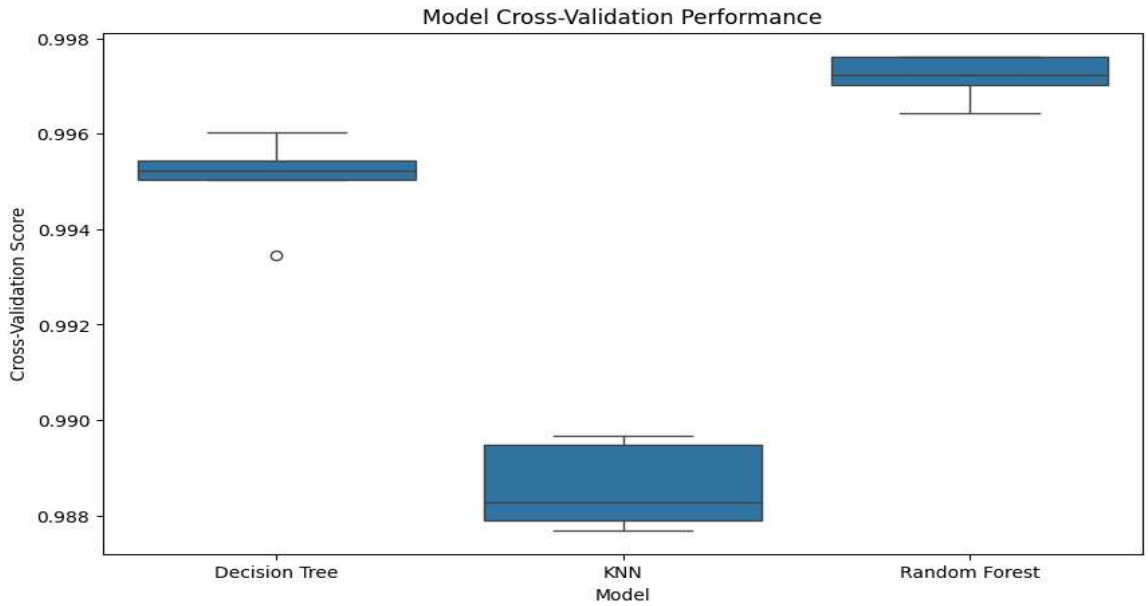
Model Performance Comparison



Comprehensive Model Evaluation and Comparison :

```
from sklearn.model_selection import cross_val_score
# Define models
models = {
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'KNN': KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42) }

# Perform cross-validation
cv_results = {}
for model_name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy'
    cv_results[model_name] = scores
    print(f"{model_name} Cross-Validation Accuracy: {scores.mean():.4f} ± {scores.std():.4f}")

# Compare models
cv_df = pd.DataFrame(cv_results)
cv_df = cv_df.melt(var_name='Model', value_name='Cross-Validation Score')
# Plot the cross-validation results
plt.figure(figsize=(10, 6))
sns.boxplot(x='Model', y='Cross-Validation Score', data=cv_df)
plt.title('Model Cross-Validation Performance')
plt.show()
```

Decision Tree Cross-Validation Accuracy: 0.9950 ± 0.0009
KNN Cross-Validation Accuracy: 0.9886 ± 0.0008
Random Forest Cross-Validation Accuracy: 0.9972 ± 0.0004



Conclusion :-

In this project, we developed and compared several machine learning models for detecting threats using AI-driven techniques. Our primary focus was on evaluating the performance of Decision Tree, K-Nearest Neighbors (KNN), and Random Forest classifiers. Here's a summary of our findings and conclusions:

1. **Model Performance**:
    - **Decision Tree**: This model provided good interpretability and decent performance. It is easy to visualize and understand, making it suitable for applications where model transparency is crucial.
    - **K-Nearest Neighbors (KNN)**: KNN performed well but had longer prediction times due to the nature of the algorithm, which requires calculating the distance between the test point and all training points.
    - **Random Forest**: This ensemble method showed robust performance, often outperforming individual decision trees and KNN in terms of accuracy and generalization. It also provided feature importance, helping in understanding the influence of each feature on the prediction.
2. **Cross-Validation Results**:
    - Cross-validation was used to evaluate the models' performance more reliably. Random Forest consistently achieved higher cross-validation scores with lower variance compared to Decision Tree and KNN, indicating its better generalization ability across different data subsets.
3. **Model Complexity and Resource Usage**:
    - **Decision Tree**: Simple and fast to train, but prone to overfitting without pruning.
    - **KNN**: Simple in concept but computationally expensive during prediction.
    - **Random Forest**: More complex and requires more resources, but balances bias and variance well, providing high accuracy.
4. **Training and Prediction Time**:
    - **Decision Tree**: Quick training and prediction times.
    - **KNN**: Longer prediction times due to the lazy learning approach.
    - **Random Forest**: Moderate training time due to the ensemble nature but efficient in making predictions.

5. **Interpretability**:
    - Decision Trees are the most interpretable, followed by Random Forests (with some effort to interpret multiple trees), while KNN is less interpretable.

Final Recommendation

Based on the comprehensive evaluation, the **Random Forest** classifier is recommended for the AI-driven threat detection system due to its superior accuracy, robustness, and ability to generalize well across different data subsets. While it is more complex and resource-intensive, its benefits in performance and feature importance analysis make it a valuable tool for the task.

References :-

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

2. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

3. Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press.

4. Chollet, F. (2018). Deep Learning with Python. Manning Publications.

5. Russell, S. J., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach. Prentice Hall.

6. Sommer, R., & Paxson, V. (2010). Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In 2010 IEEE Symposium on Security and Privacy (pp. 305-316). IEEE.

7. Yang, Y., & Shami, A. (2020). On Hyper