Maintenance Prediction Model

A Maintenance Prediction Model employs machine learning algorithms on a dataset to forecast the optimal timing for maintenance requirements in machinery or equipment.

Project Overview :

Objective :

Our project endeavors to apply supervised machine learning methods to anticipate the probability of machinery maintenance based on pertinent operational indicators.

Significance :

Predictive maintenance model is essential for anticipating potential issues in machinery before they become critical, using machine learning algorithms. This proactive approach helps minimize downtime, reduce operational disruptions, optimize resource allocation, and enhance overall efficiency. By predicting maintenance needs, the model enables cost savings and ensures machinery operates at peak performance, contributing to the success and sustainability of your operations.

Dataset :

- **Predictive Maintenance Dataset CSV File :**

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure

- **The dataset "predictive_maintenance.csv" obtained from Kaggle includes features such as UDI, Product ID, Type, Air temperature, Process temperature, Rotational speed, Torque, Tool wear, and Target, with the additional label "Failure Type." This dataset likely captures information related to machinery or equipment, aiming to predict maintenance needs based on various operational parameters.**

Dataset Handling

```
data.isnull().sum()

UDI                          0
Product ID                   0
Type                         0
Air temperature [K]          0
Process temperature [K]      0
Rotational speed [rpm]       0
Torque [Nm]                  0
Tool wear [min]              0
Target                       0
Failure Type                 0
dtype: int64
```

Dataset didn’t have any null entries

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   UDI                      10000 non-null  int64
 1   Product ID               10000 non-null  object
 2   Type                     10000 non-null  object
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64
 8   Target                   10000 non-null  int64
 9   Failure Type             10000 non-null  object
dtypes: float64(3), int64(4), object(3)
memory usage: 781.4+ KB
```

Some parameters of the dataset are in object

Dataset Handling :

- In order to train our data using Machine learning algorithms , its necessary for the data to be in numeric forms, so the given features of data set are converted to number format.

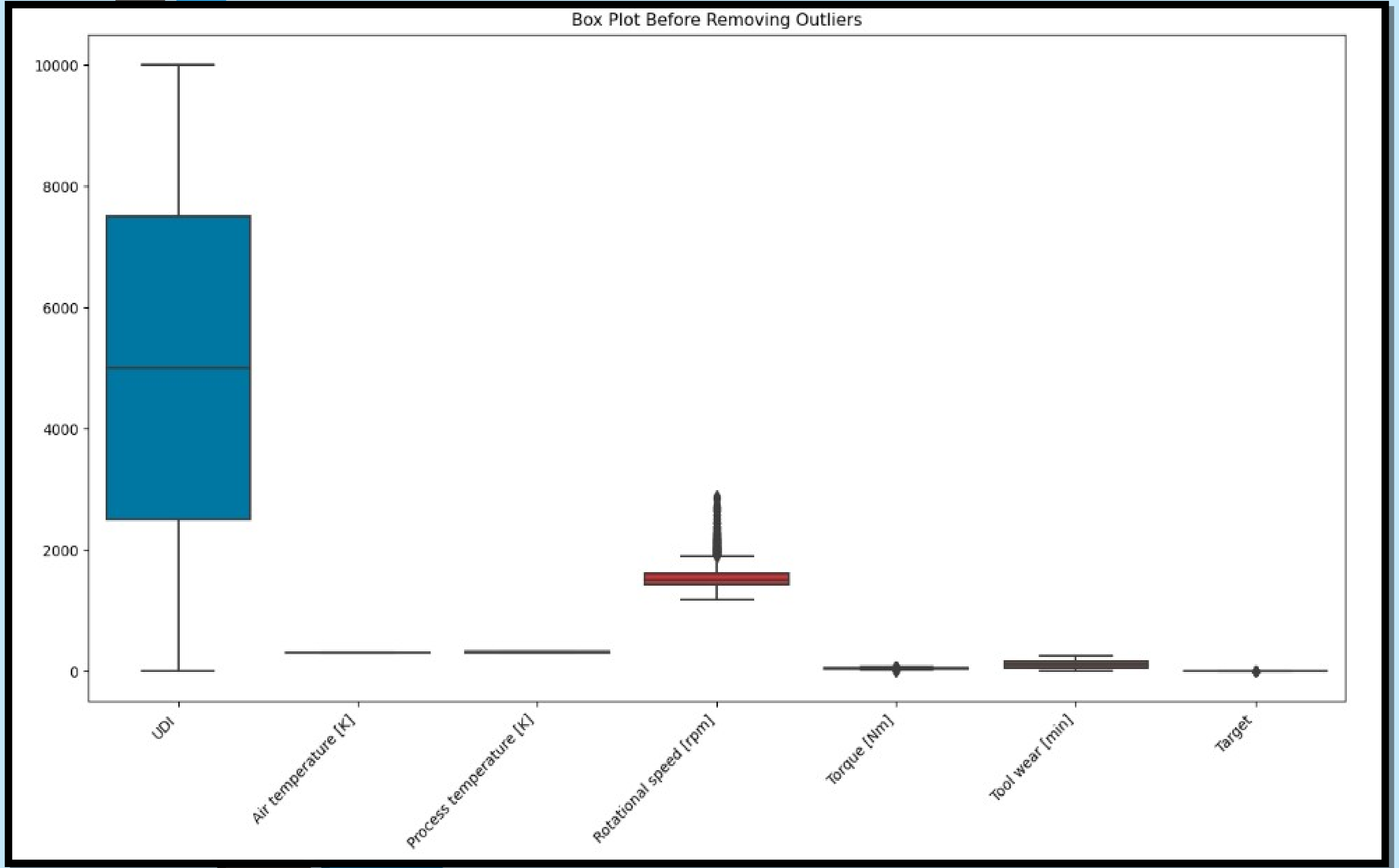
```python
# Load the CSV file
data = pd.read_csv('predictive_maintenance.csv')

# Identify object columns
object_columns = data.select_dtypes(include=['object']).columns

# Use LabelEncoder to convert object columns to numerical input
label_encoder = LabelEncoder()

for column in object_columns:
    data[column] = label_encoder.fit_transform(data[column])

# Now, 'data' DataFrame has numerical values for object columns
# You can save the modified DataFrame back to a CSV file if needed
data.to_csv('predictive_maintenancef.csv', index=False)
```

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   UDI                      10000 non-null  int64  
 1   Product ID               10000 non-null  int32  
 2   Type                     10000 non-null  int32  
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64  
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64  
 8   Target                   10000 non-null  int64  
 9   Failure Type             10000 non-null  int32  
dtypes: float64(3), int32(3), int64(4)
memory usage: 664.2 KB
```

Dataset is now in

Outliers :

Boxplot of outliers before removing them :



Number of outliers found in each column before removing outliers:

UDI: 0

Product ID: 0

Type: 0

Air temperature [K]: 0

Process temperature [K]: 0

Rotational speed [rpm]: 418

Torque [Nm]: 69

Tool wear [min]: 0

Target: 339

Failure Type: 0

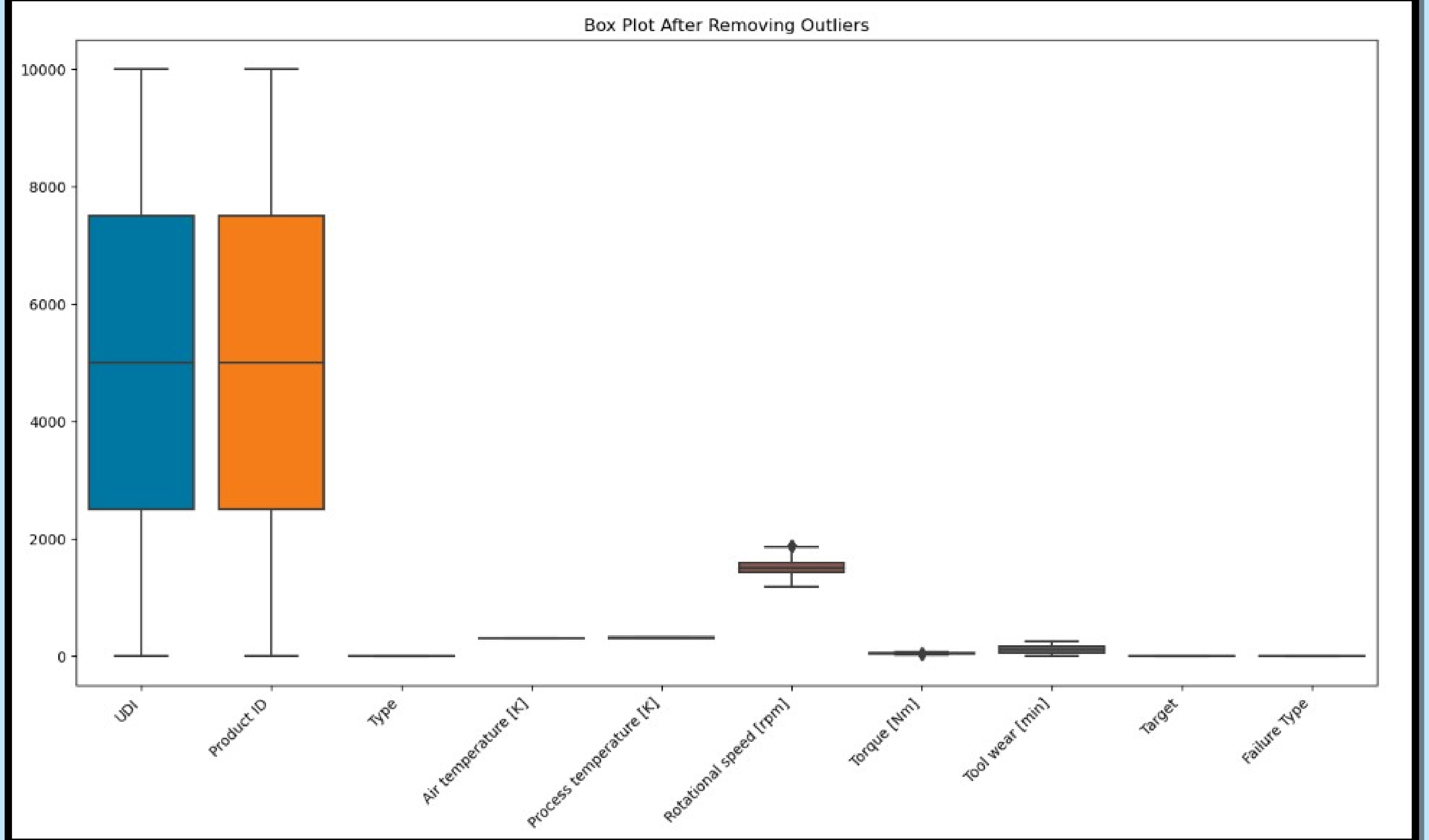
Method to Remove Outliers :

```
# Function to remove outliers using the IQR method and count outliers
def remove_outliers_iqr(column):
    # Convert column values to numeric, ignoring errors for non-convertible values
    column_numeric = pd.to_numeric(column, errors='coerce')

    # Drop NaN values after conversion
    column_numeric = column_numeric.dropna()

    Q1 = column_numeric.quantile(0.25)
    Q3 = column_numeric.quantile(0.75)
    IQR = Q3 - Q1

    # Count outliers
    outliers = ((column_numeric < (Q1 - 1.5 * IQR)) | (column_numeric > (Q3 + 1.5 * IQR)))
    num_outliers = outliers.sum()

    return column_numeric[~outliers], num_outliers
```

the

Outliers :

Boxplot of outliers after removing them :



Number of outliers found in each column after removing outliers:

UDI: 0

Product ID: 0

Type: 0

Air temperature [K]: 0

Process temperature [K]: 0

Rotational speed [rpm]: 108

Torque [Nm]: 3

Tool wear [min]: 0

Target: 0

Failure Type: 0

Libraries Used :

- **Pandas**
- **Numpy**
- **Sklearn (train_test_split,RandomForestClassifier,GradientBoostingClassifier,DecisionTreeClassifier,accuracy_score,precision_score,recall_score,f1_score,confusion_matrix)**

Exploratory Data Analysis (EDA)

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
count	10000.00000	10000.00000	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	4999.50000	1.19940	300.004930	310.005560	1538.776100	39.986910	107.951000	0.033900	1.039000
std	2886.89568	2886.89568	0.60023	2.000259	1.483734	179.284096	9.968934	63.654147	0.180981	0.379069
min	1.00000	0.00000	0.00000	295.300000	305.700000	1168.000000	3.800000	0.000000	0.000000	0.000000
25%	2500.75000	2499.75000	1.00000	298.300000	308.800000	1423.000000	33.200000	53.000000	0.000000	1.000000
50%	5000.50000	4999.50000	1.00000	300.100000	310.100000	1503.000000	40.100000	108.000000	0.000000	1.000000
75%	7500.25000	7499.25000	2.00000	301.500000	311.100000	1612.000000	46.800000	162.000000	0.000000	1.000000
max	10000.00000	9999.00000	2.00000	304.500000	313.800000	2886.000000	76.600000	253.000000	1.000000	5.000000

```python
# Assuming 'data' is your DataFrame and 'column_name' is the column you want to check
column_name = 'Target'  # Replace with your actual column name
value_counts = data[column_name].value_counts()
print(f"Value counts for {column_name}:\n{value_counts}")
```

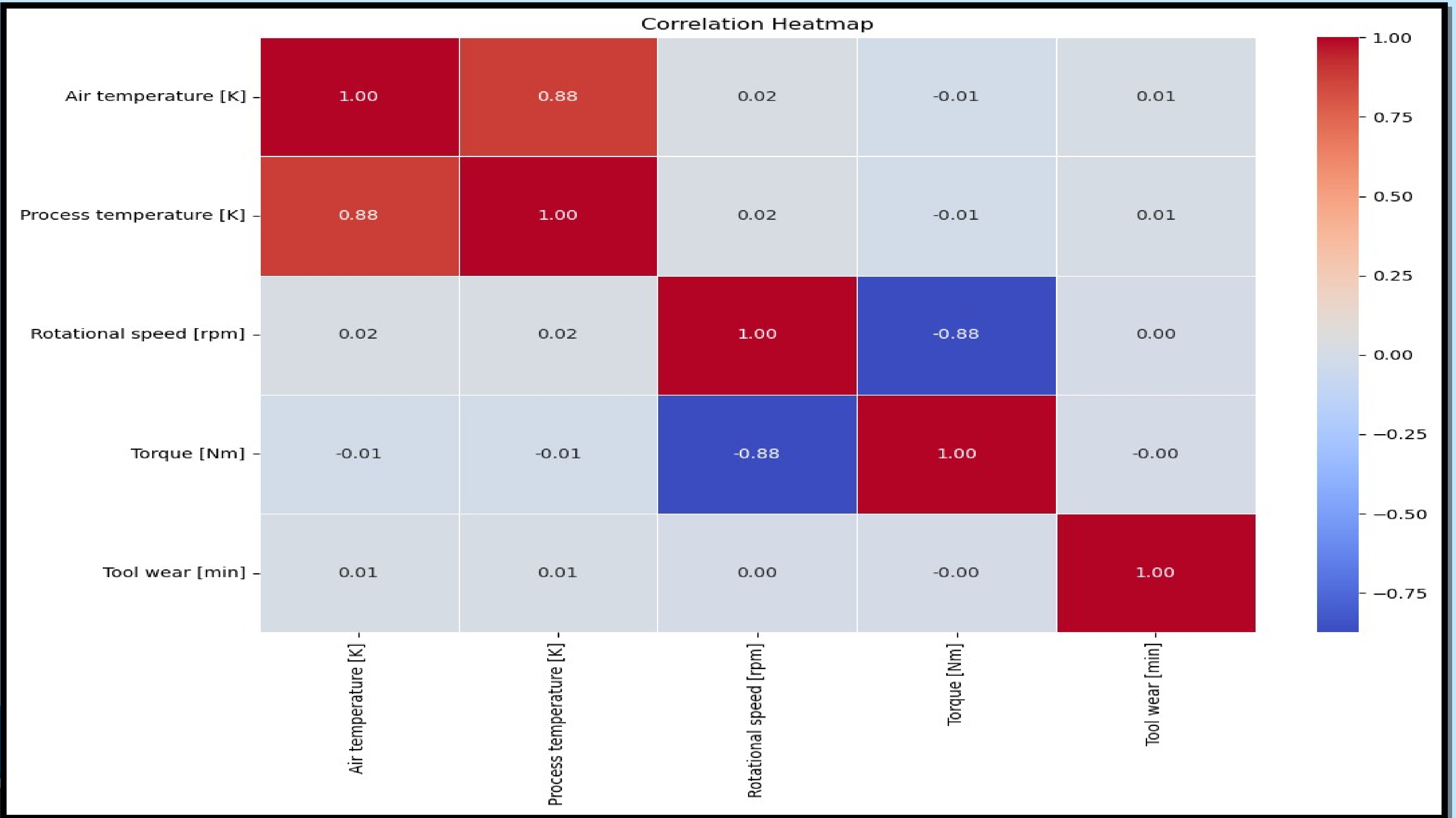
```
Value counts for Target:
Target
0    9661
1     339
Name: count, dtype: int64
```

Target Count

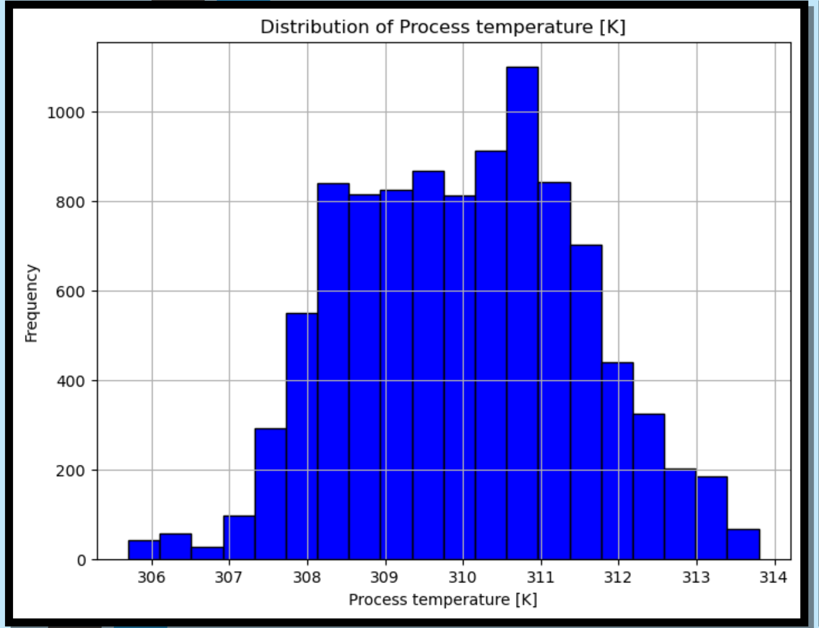
```
Correlation Table:
                          Air temperature [K] Process temperature [K]  \
Air temperature [K]                       1.0                0.876107
Process temperature [K]              0.876107                     1.0
Rotational speed [rpm]                0.02267                0.019277
Torque [Nm]                         -0.013778               -0.014061
Tool wear [min]                      0.013853                0.013488

                          Rotational speed [rpm] Torque [Nm] Tool wear [min]
Air temperature [K]                      0.02267   -0.013778        0.013853
Process temperature [K]                 0.019277   -0.014061        0.013488
Rotational speed [rpm]                       1.0   -0.875027        0.000223
Torque [Nm]                            -0.875027         1.0       -0.003093
Tool wear [min]                         0.000223   -0.003093             1.0
```

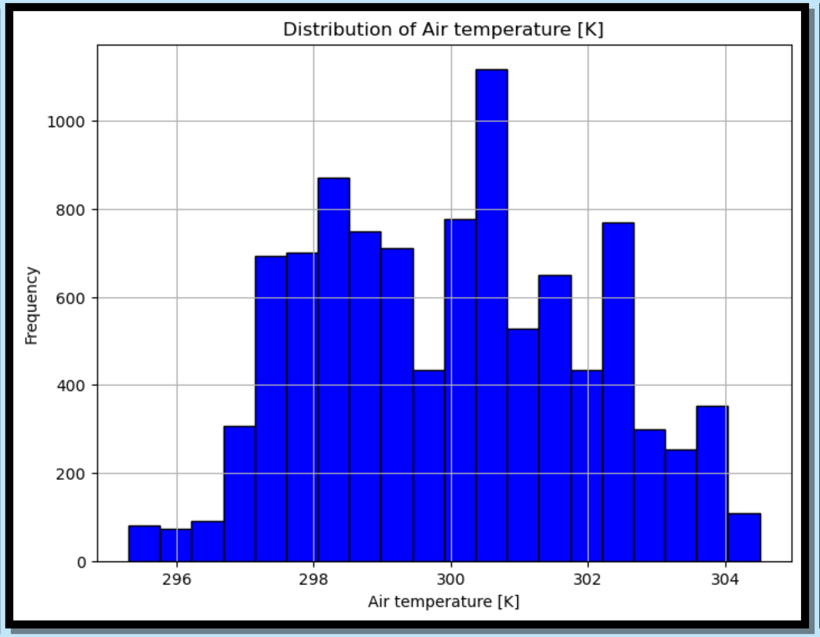
Correlation Table

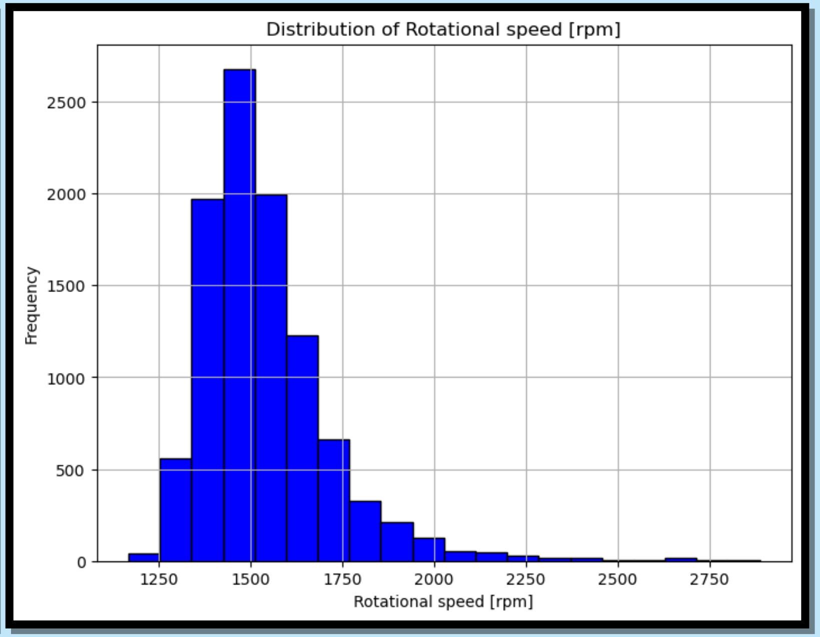
Graphical Analysis (Correlation) :

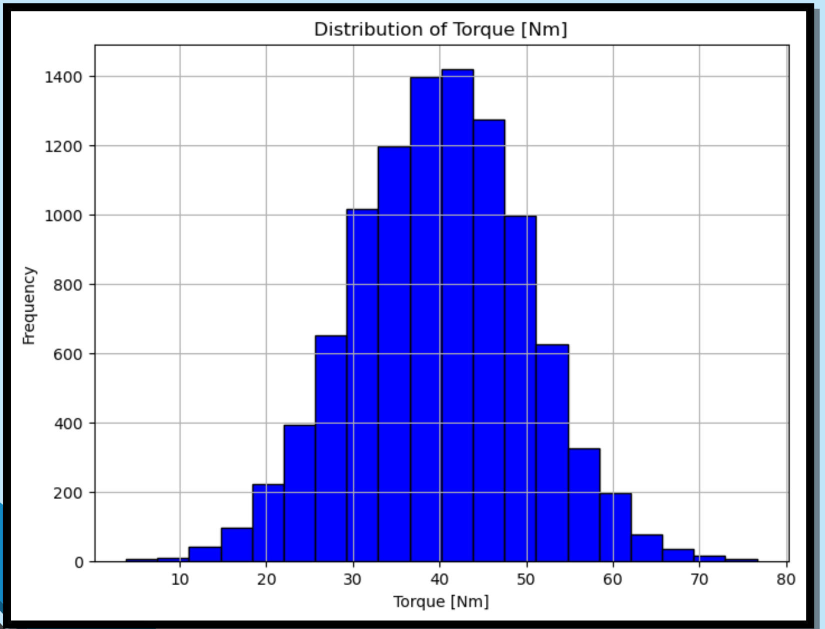


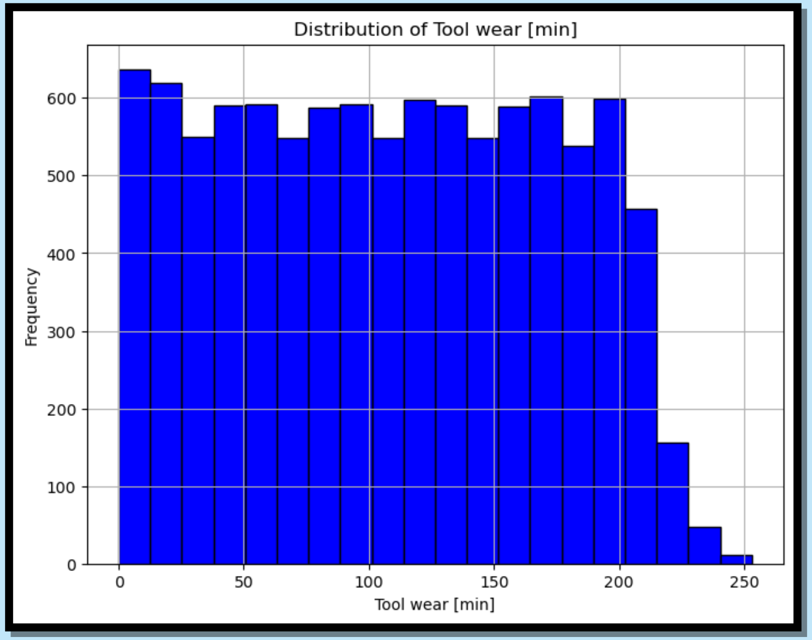
Distribution with Bar Graphs.


Distribution of Process temperature [K]
Frequency
0
200
400
600
800
1000
306
307
308
309
310
311
312
313
314
Process temperature [K]


Distribution of Air temperature [K]
Frequency
0
200
400
600
800
1000
296
298
300
302
304
Air temperature [K]


Distribution of Rotational speed [rpm]
Frequency
0
500
1000
1500
2000
2500
1250
1500
1750
2000
2250
2500
2750
Rotational speed [rpm]


Distribution of Torque [Nm]
Frequency
0
200
400
600
800
1000
1200
1400
10
20
30
40
50
60
70
80
Torque [Nm]

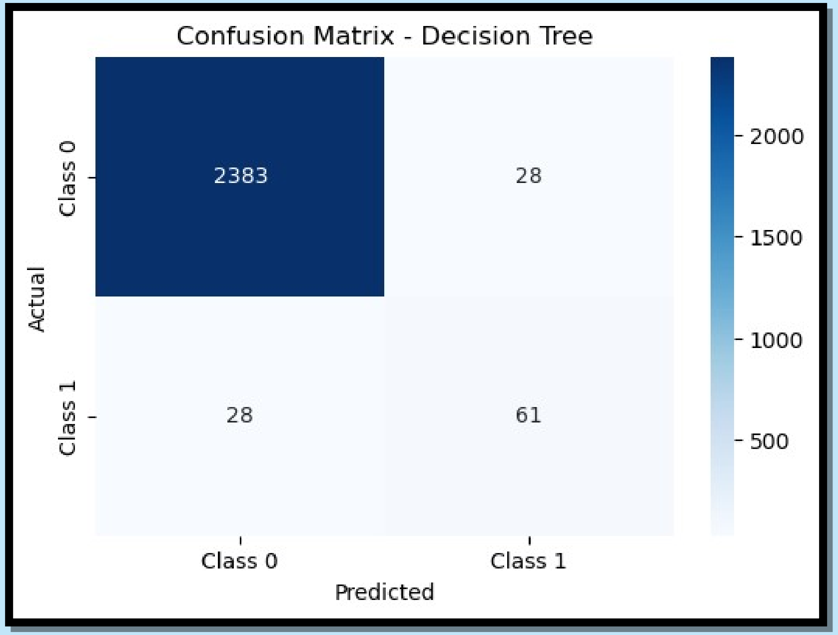

Distribution of Tool wear [min]
Frequency
0
100
200
300
400
500
600
0
50
100
150
200
250
Tool wear [min]

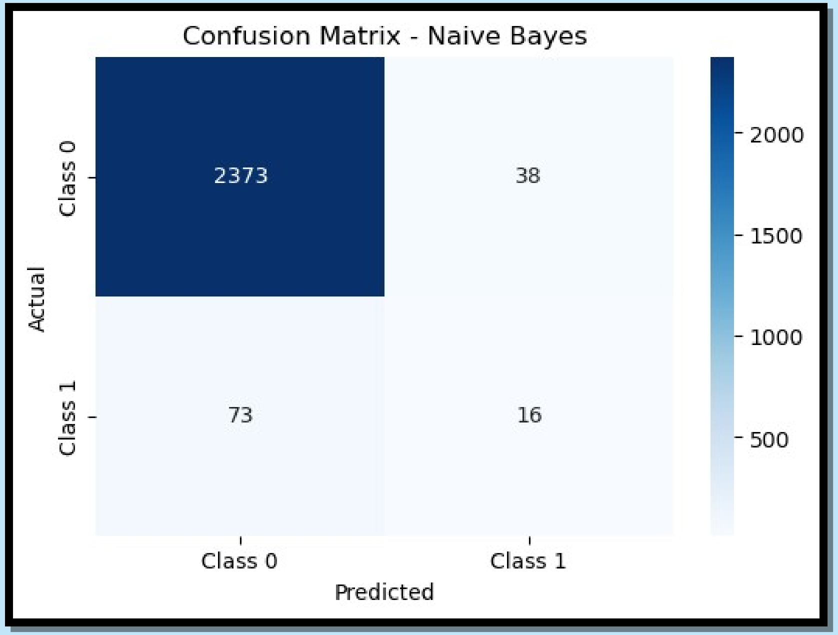
Dataset Training & Testing

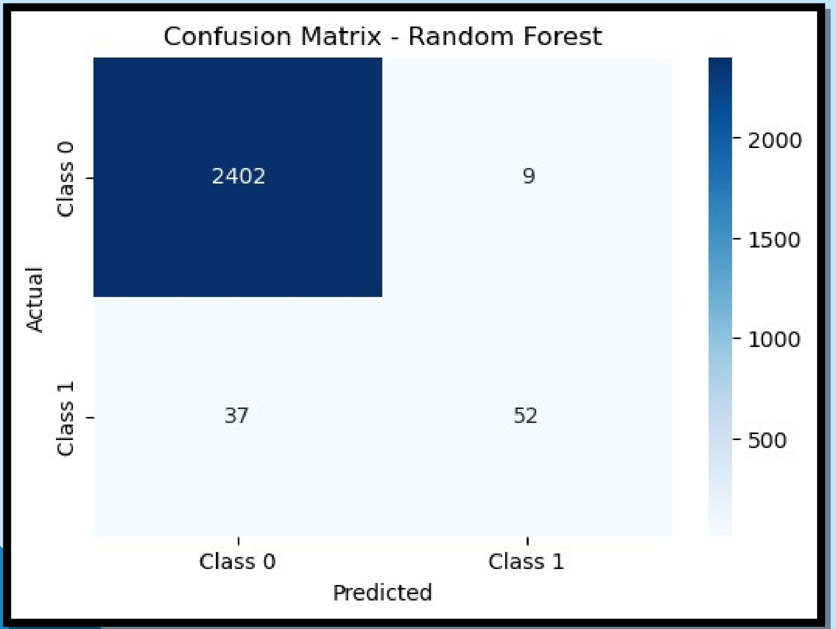
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(data_no_outliers[features], data_no_outliers[target], test_size=0.25, random_state=30)
```

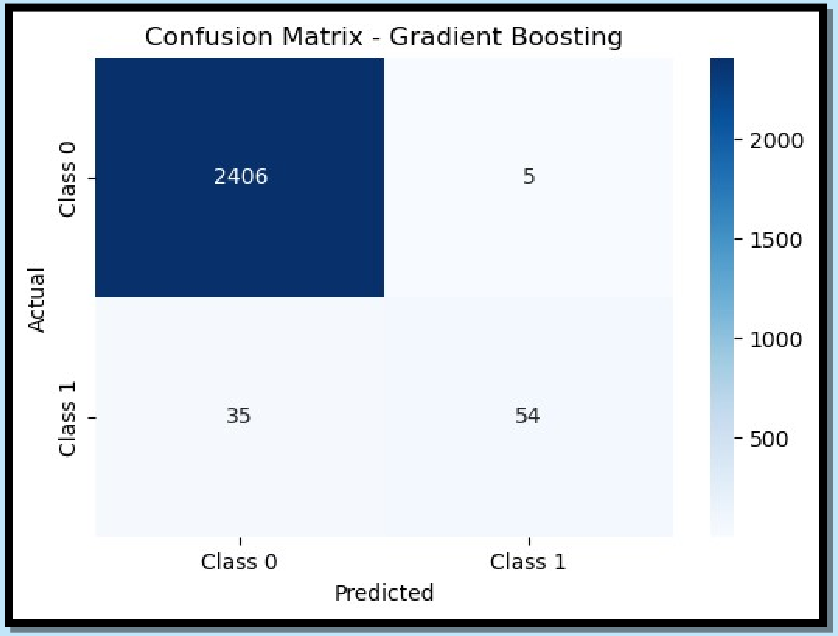
Training Size 75%
Test Size = 25%
Random State = 30

Confusion Matrix


Confusion Matrix - Decision Tree
Actual
Class 0
Class 1
Predicted
Class 0
Class 1
2383
28
28
61
2000
1500
1000
500


Confusion Matrix - Naive Bayes
Actual
Class 0
Class 1
Predicted
Class 0
Class 1
2373
38
73
16
2000
1500
1000
500

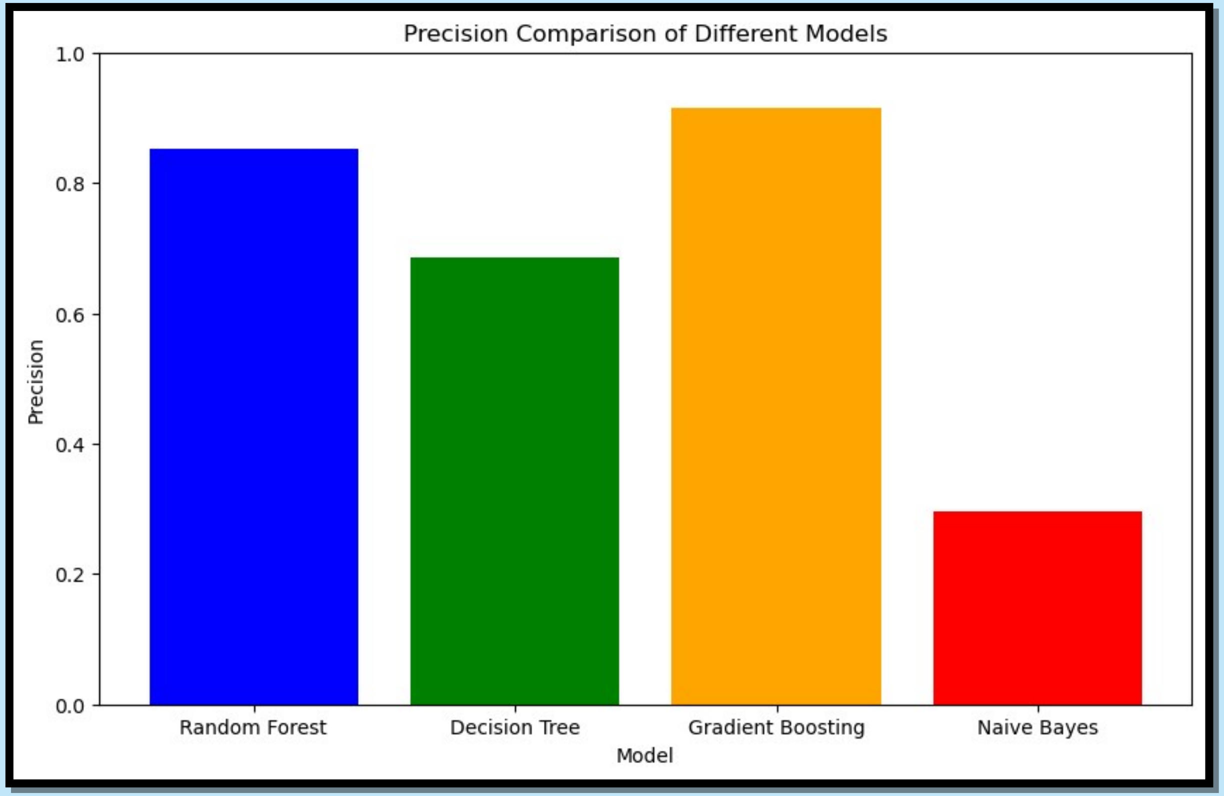

Confusion Matrix - Random Forest
Actual
Class 0
Class 1
Predicted
Class 0
Class 1
2402
9
37
52
2000
1500
1000
500


Confusion Matrix - Gradient Boosting
Actual
Class 0
Class 1
Predicted
Class 0
Class 1
2406
5
35
54
2000
1500
1000
500

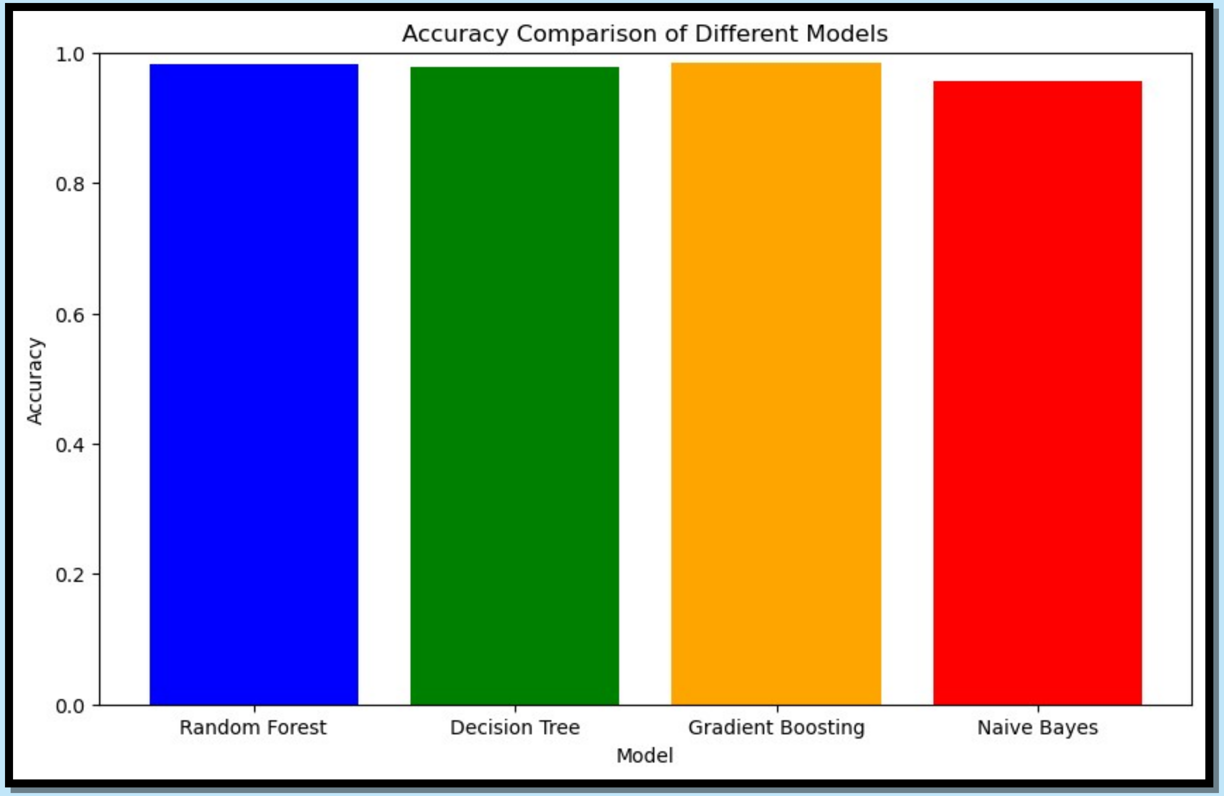
Performance of Models.

S No	Model	Accuracy	Precision	Recall	F1 Score	Specificity
0	Random Forest	0.9816	0.852459	0.584270	0.693333	0.996267
1	Decision Tree	0.9776	0.685393	0.685393	0.685393	0.988387
2	Gradient Boosting	0.9840	0.915254	0.606742	0.729730	0.997926
3	Naive Bayes	0.9556	0.296296	0.179775	0.223776	0.984239
4	KNN	0.9656	0.565217	0.146067	0.232143	0.995852

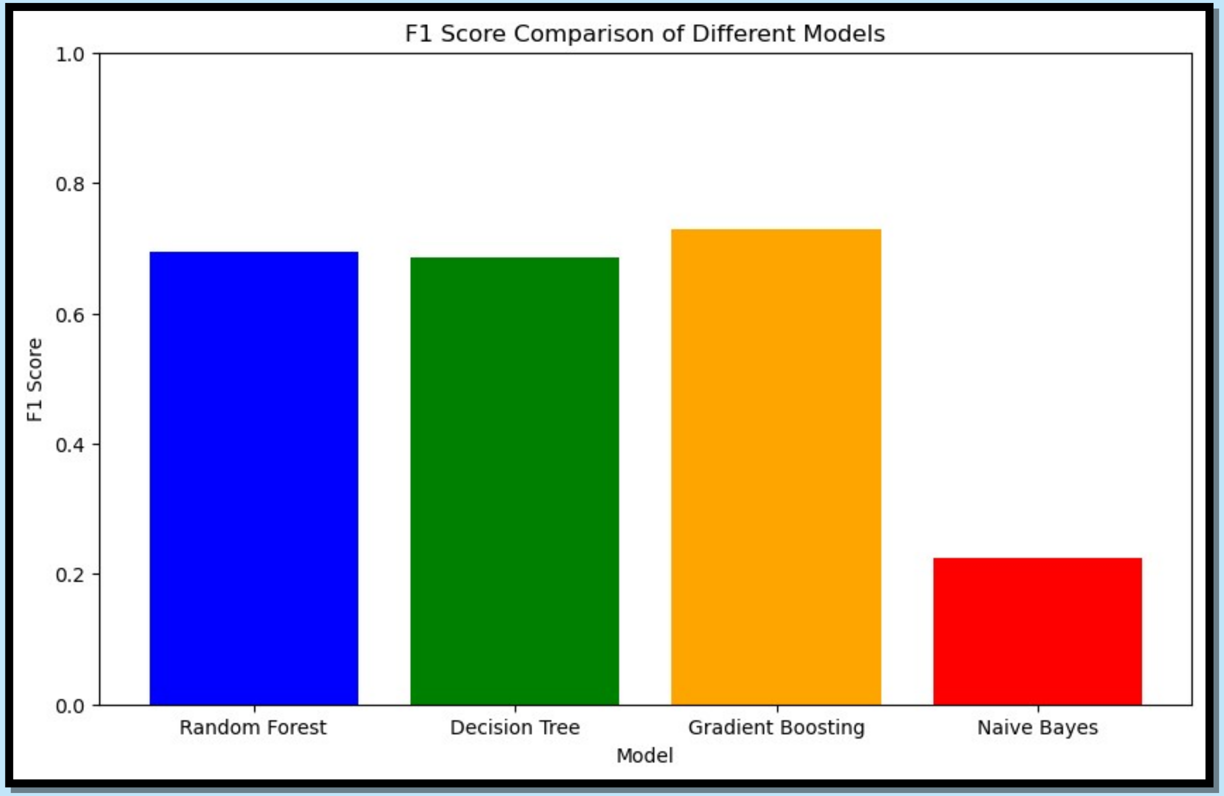
Comparison of Precision



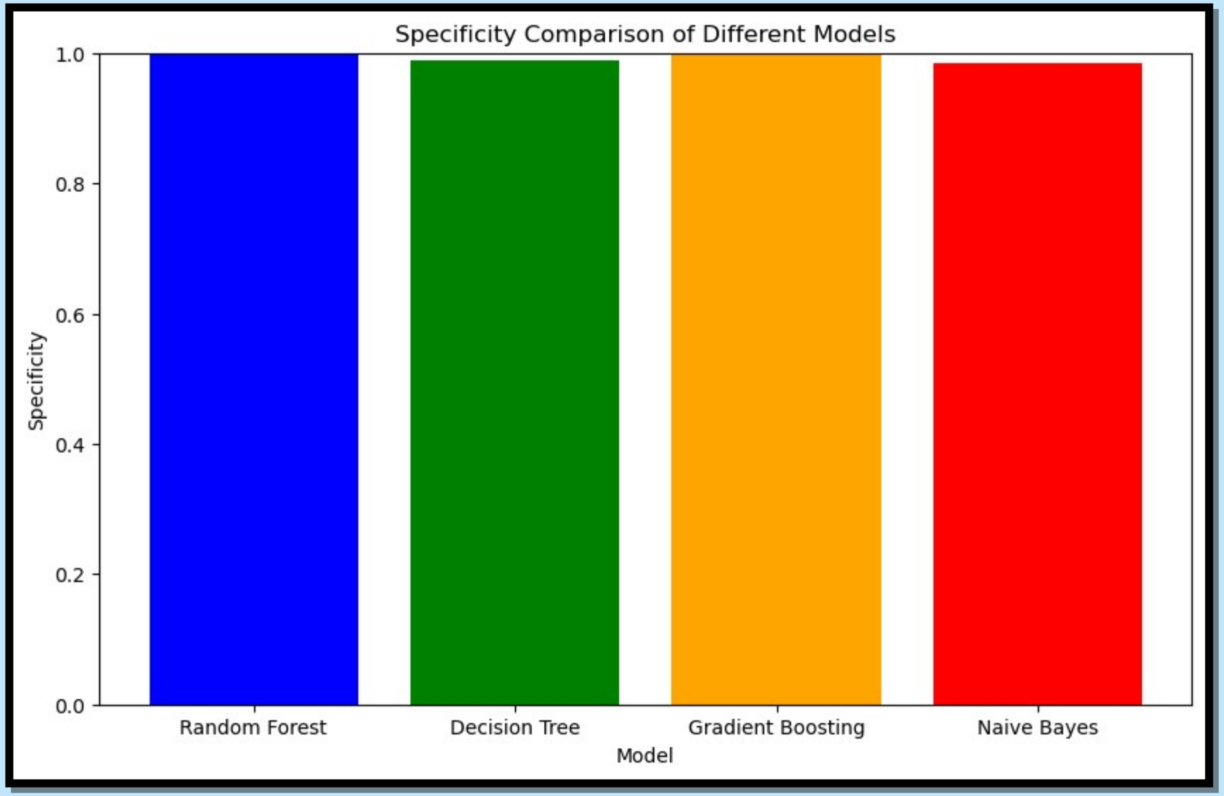
Comparison of Accuracy



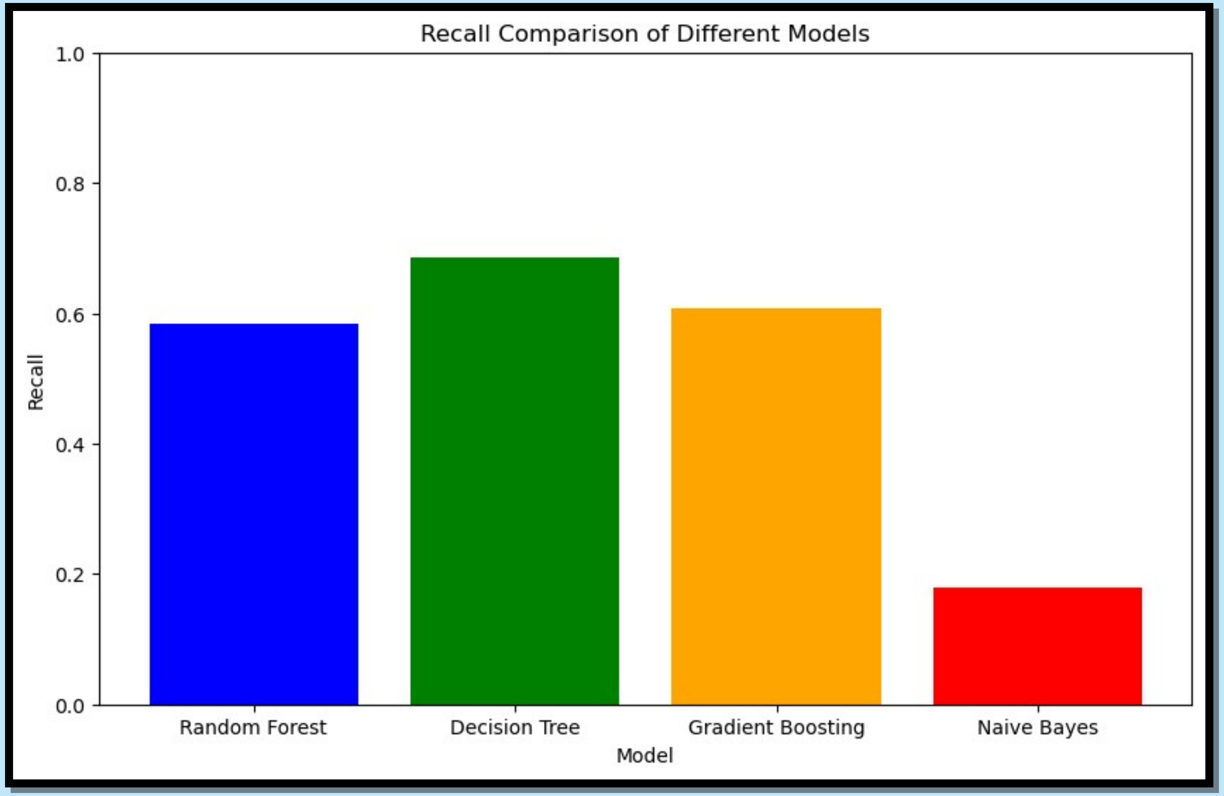
Comparison of F1 Score



Comparison of Specificity



Comparison of Recall



Results & Findings

1. **Random Forest** achieved the highest accuracy (98.16%) and specificity (99.63%) among the models, making it a strong performer overall.
2. **Gradient Boosting** demonstrated good accuracy (98.40%) and precision (91.53%), making it effective in correctly predicting positive cases.
3. **Decision Tree** performed well in terms of accuracy (97.76%) but had lower precision compared to Random Forest and Gradient Boosting.
4. **KNN** showed reasonable accuracy (96.56%) but had lower recall, suggesting it might miss some positive cases.
5. **Naive Bayes** had the lowest accuracy (95.56%) and precision (29.63%), indicating limitations in correctly identifying positive cases.

Conclusion.

- In conclusion, Random Forest and Gradient Boosting are the top performers, with Random Forest being particularly strong in terms of overall accuracy and specificity. Decision Tree and KNN are reasonable but may benefit from further tuning, while Naive Bayes might not be the best choice for this specific classification task.

Thank You