```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import PolynomialFeatures
import numpy as  np
from sklearn.feature_selection import VarianceThreshold
```

```
from google.colab import drive
drive.mount('/content/drive')
file_path="/content/drive/My Drive/ai4i2020.csv"
data = pd.read_csv(file_path)

# data=pd.read_csv("Downloads/ai4i2020.csv")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```
data.head()
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0	0

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
print("\nDataset Info:\n")
data.info()
```

```
Dataset Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 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   Machine failure         10000 non-null  int64
 9   TWF                     10000 non-null  int64
 10  HDF                     10000 non-null  int64
 11  PWF                     10000 non-null  int64
 12  OSF                     10000 non-null  int64
 13  RNF                     10000 non-null  int64
dtypes: float64(3), int64(9), object(2)
memory usage: 1.1+ MB
```

```
data.describe()
```

	UDI	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	100
mean	5000.50000	300.004930	310.005560	1538.776100	39.986910	107.951000	0.033900	0.004600	0.011500	
std	2886.89568	2.000259	1.483734	179.284096	9.968934	63.654147	0.180981	0.067671	0.106625	
min	1.00000	295.300000	305.700000	1168.000000	3.800000	0.000000	0.000000	0.000000	0.000000	
25%	2500.75000	298.300000	308.800000	1423.000000	33.200000	53.000000	0.000000	0.000000	0.000000	
50%	5000.50000	300.100000	310.100000	1503.000000	40.100000	108.000000	0.000000	0.000000	0.000000	

```
data['Machine failure'].value_counts()
```

Machine failure	count
0	9661
1	339

dtype: int64

```
Start coding or generate with AI.
```

```
data1=data.copy()
X = data1[['Type', 'Air temperature [K]', 'Process temperature [K]', 'Rotational speed [rpm]', 'Torque [Nm]',
          'Tool wear [min]']]
y = data1['Machine failure']
```

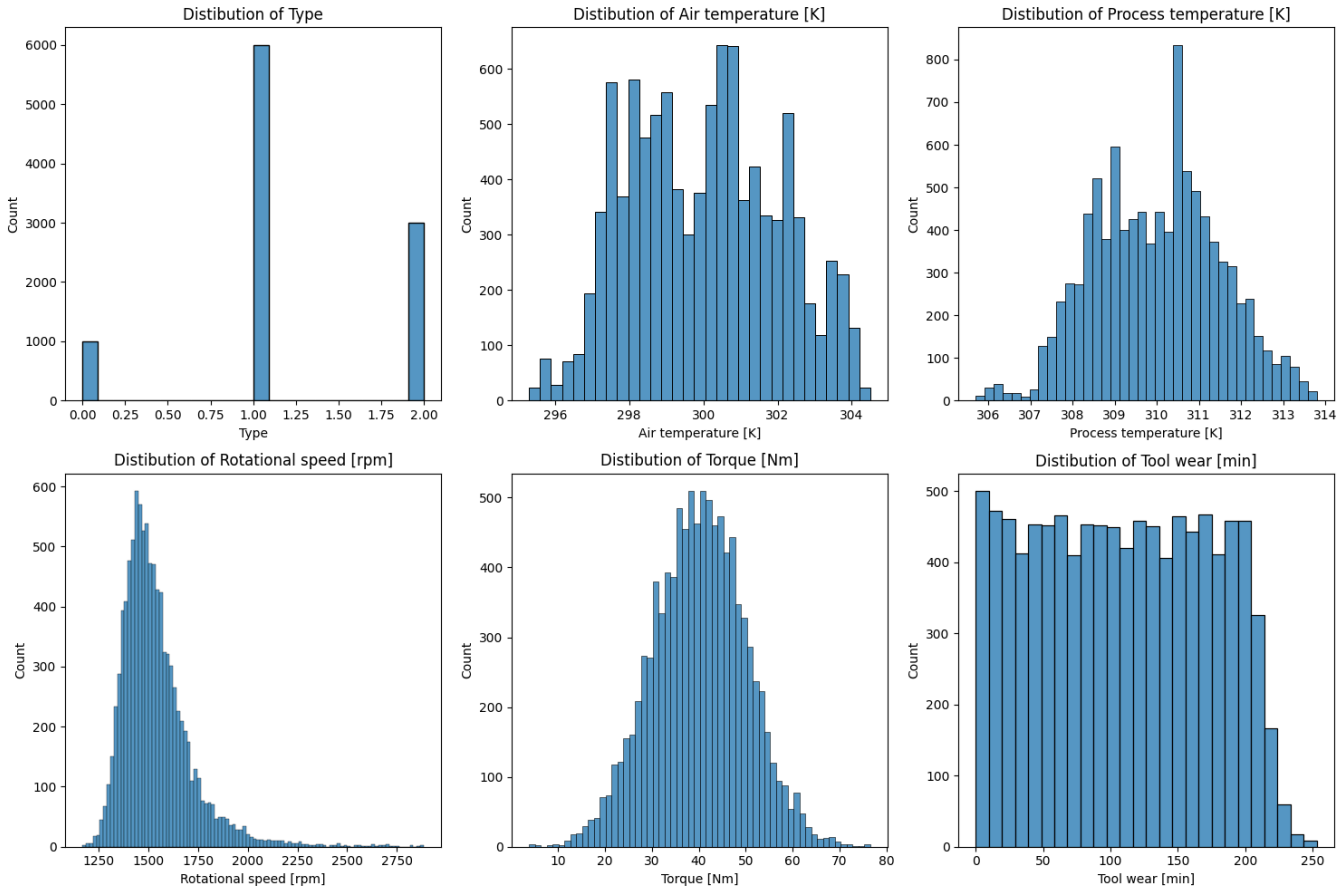
```
Start coding or generate with AI.
```

Feature Engineering

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X['Type'] = le.fit_transform(X['Type'])
```

```
<ipython-input-9-2982d9b4d4cf>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  X['Type'] = le.fit_transform(X['Type'])
```

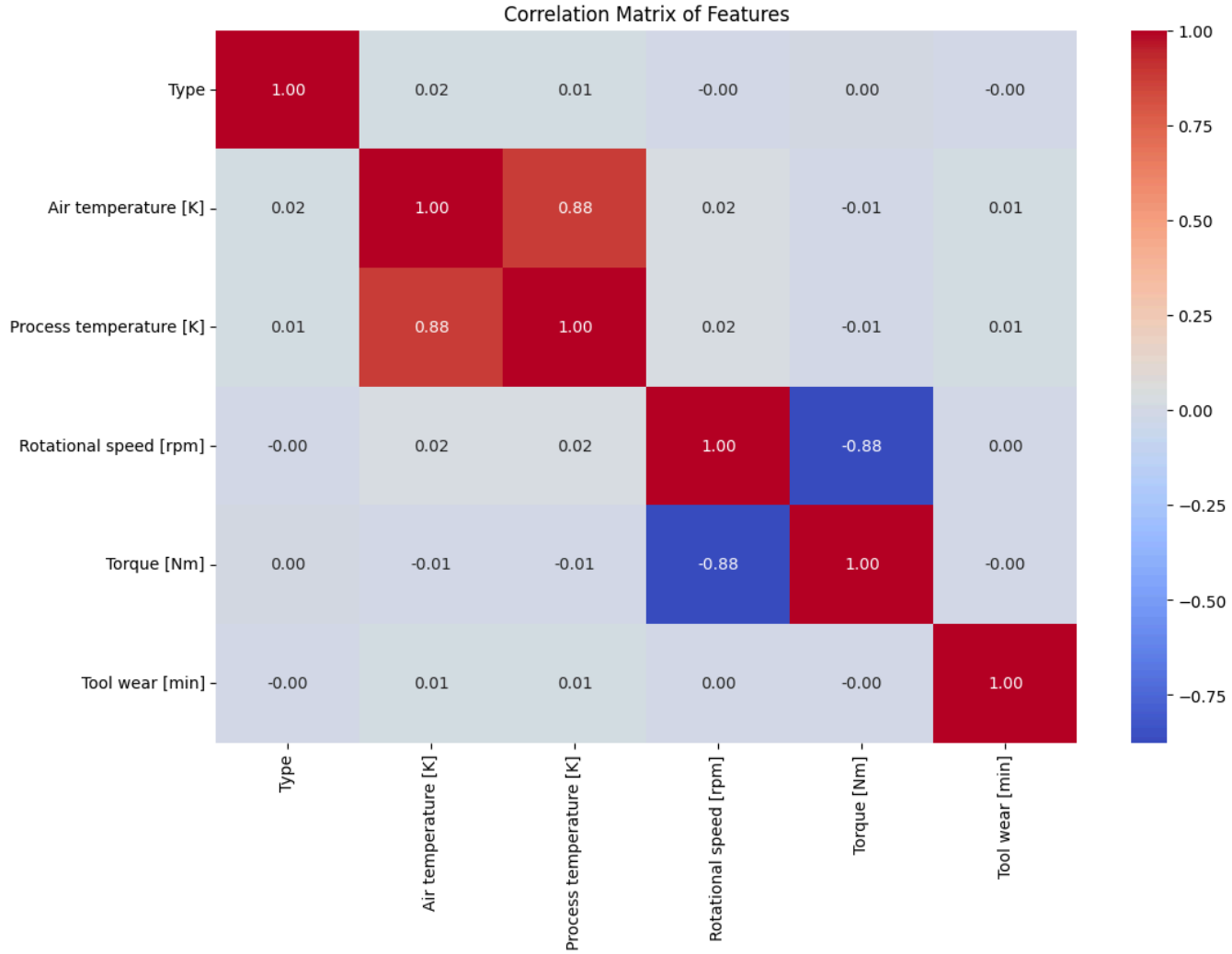
```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
for i, column in enumerate(X.columns):
    row = i // 3
    col = i % 3
    sns.histplot(X[column], ax=axes[row, col])
    axes[row, col].set_title(f'Distibution of {column}')
plt.tight_layout()
plt.show()
```




```
Start coding or generate with AI.
```

```
correlation_matrix = X.corr()
print("\nCorrelation Matrix:\n", correlation_matrix)

# Correlation Heatmap (Graph)
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Features')
plt.show()
```

```
Correlation Matrix:
                             Type  Air temperature [K]  \
Type                     1.000000             0.017599
Air temperature [K]      0.017599             1.000000
Process temperature [K]  0.013444             0.876107
Rotational speed [rpm]  -0.002693             0.022670
Torque [Nm]              0.004011            -0.013778
Tool wear [min]         -0.003930             0.013853

                         Process temperature [K]  Rotational speed [rpm]  \
Type                                    0.013444                -0.002693
Air temperature [K]                     0.876107                 0.022670
Process temperature [K]                 1.000000                 0.019277
Rotational speed [rpm]                  0.019277                 1.000000
Torque [Nm]                            -0.014061                -0.875027
Tool wear [min]                         0.013488                 0.000223

                         Torque [Nm]  Tool wear [min]
Type                        0.004011        -0.003930
Air temperature [K]        -0.013778         0.013853
Process temperature [K]    -0.014061         0.013488
Rotational speed [rpm]     -0.875027         0.000223
Torque [Nm]                 1.000000        -0.003093
Tool wear [min]            -0.003093         1.000000
```



Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Polynomial Features

```
from sklearn.preprocessing import PolynomialFeatures
import numpy as  np
# Apply Polynomial Features
poly = PolynomialFeatures(degree=2, include_bias=True)
X_poly = poly.fit_transform(X.select_dtypes(include=[np.number]))

poly_feature_names = poly.get_feature_names_out(X.select_dtypes(include=[np.number]).columns)
X_poly_df = pd.DataFrame(X_poly, columns=poly_feature_names)
```

Feature Selection

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.feature_selection import mutual_info_classif


info_gain = mutual_info_classif(X_poly_df,y)


top_features = X_poly_df.columns[info_gain.argsort()[::-1][:20]]
top_features
```

```
Index(['Air temperature [K] Torque [Nm]',
       'Process temperature [K] Torque [Nm]', 'Torque [Nm]^2', 'Torque [Nm]',
       'Rotational speed [rpm] Torque [Nm]', 'Rotational speed [rpm]^2',
       'Process temperature [K] Rotational speed [rpm]',
       'Rotational speed [rpm]', 'Air temperature [K] Rotational speed [rpm]',
       'Torque [Nm] Tool wear [min]', 'Type Rotational speed [rpm]',
       'Type Torque [Nm]', 'Tool wear [min]^2', 'Tool wear [min]',
       'Air temperature [K] Process temperature [K]',
       'Process temperature [K] Tool wear [min]',
       'Air temperature [K] Tool wear [min]', 'Type Tool wear [min]',
       'Type Air temperature [K]', 'Air temperature [K]'],
      dtype='object')
```

```
X_selceted=X_poly_df[top_features]
```

Scaling

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Standardize the numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X_selceted)
```

Train Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Decision Tree classifier

```
clf = DecisionTreeClassifier( max_depth=8,random_state=42)
clf.fit(X_train, y_train)

# Predict and evaluate the model
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2907
```

```
           1       0.91      0.76      0.83        93

    accuracy                           0.99      3000
   macro avg       0.95      0.88      0.91      3000
weighted avg       0.99      0.99      0.99      3000

0.9903333333333333
```

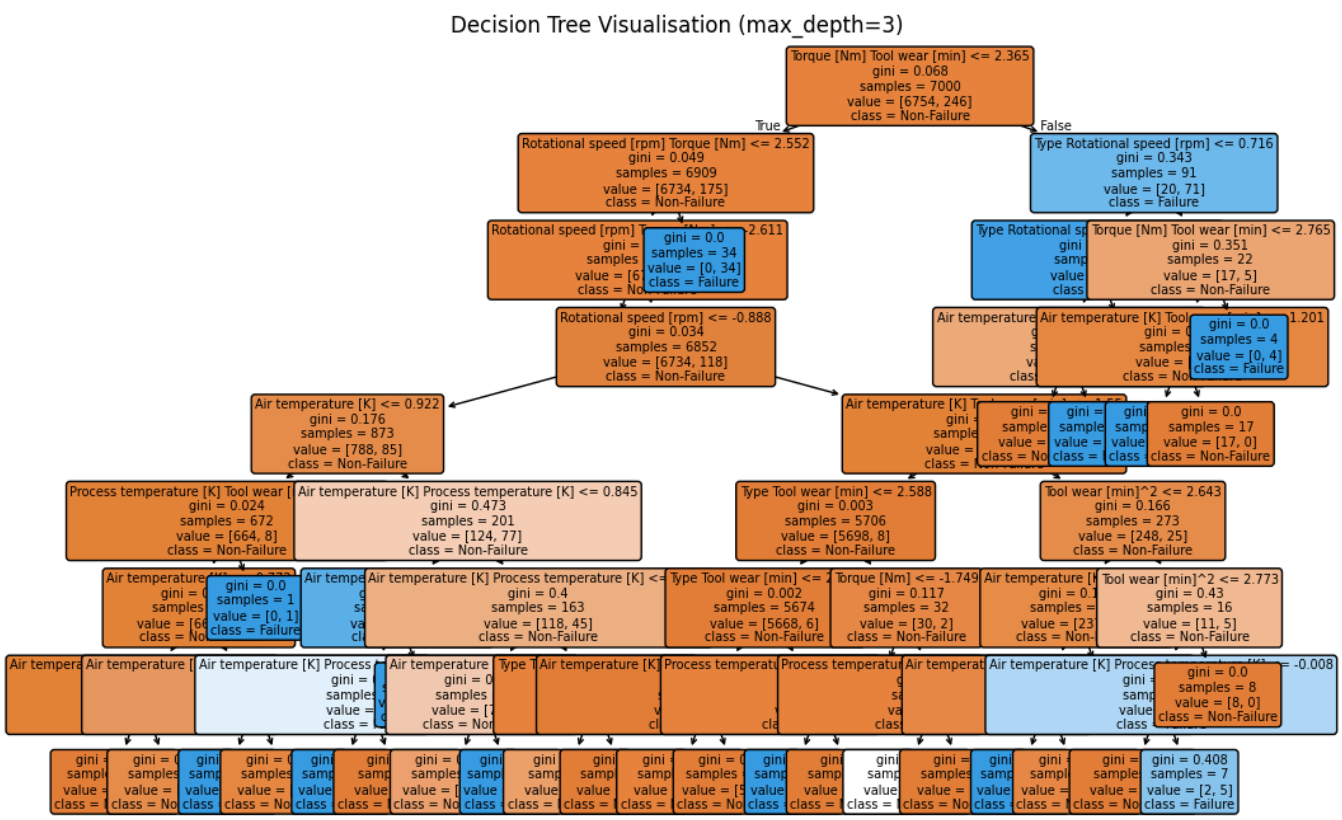
```
confusion_matrix(y_test, y_pred)
```

```
array([[2900,    7],
       [  22,   71]])
```

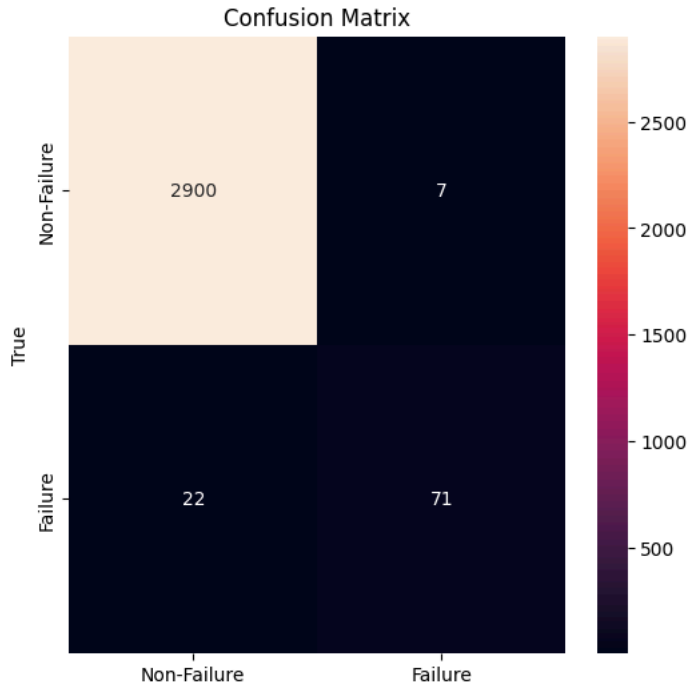
```
clf.get_depth()
```

```
8
```

```
from sklearn.tree import plot_tree
clf = DecisionTreeClassifier( max_depth=8, random_state=42)
clf.fit(X_train, y_train)
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=top_features, class_names=['Non-Failure', 'Failure'], rounded=True, fontsize=7)
plt.title('Decision Tree Visualisation (max_depth=3)')
plt.show()
```

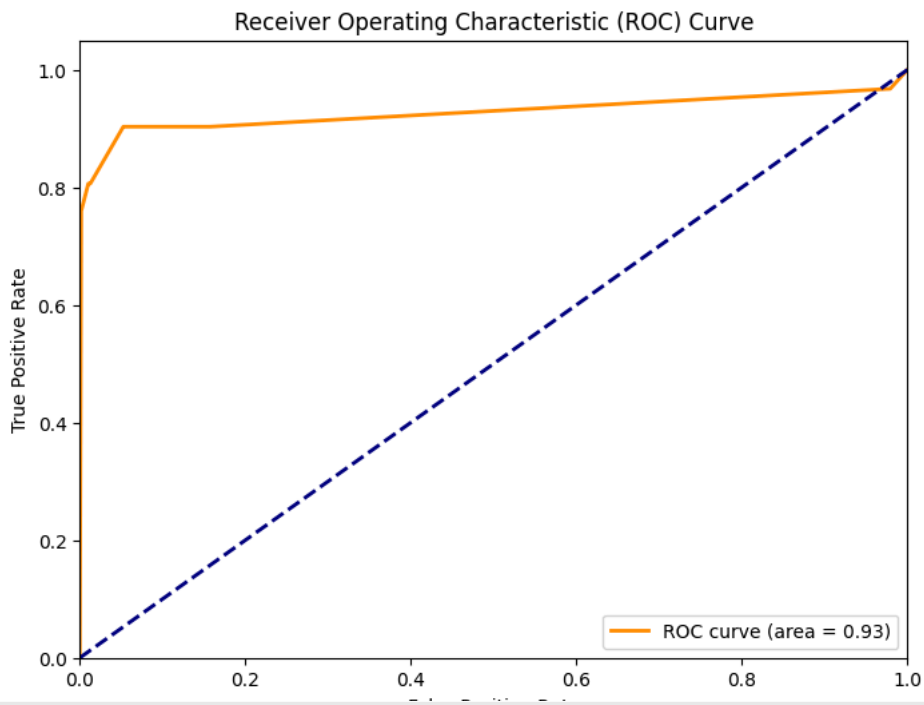



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, roc_curve, auc, precision_recall_curve

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=['Non-Failure', 'Failure'], yticklabels=['Non-Failure', 'Failure'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



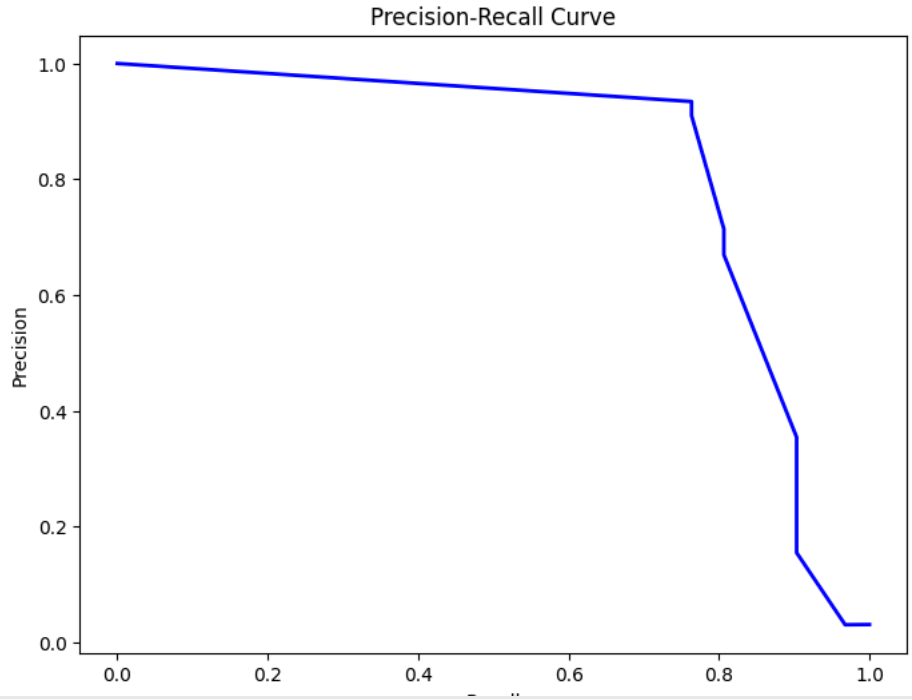
Start coding or generate with AI.

```
# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



```
Start coding or generate with AI.
```

```
# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, clf.predict_proba(X_test)[:, 1])

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='b', lw=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```




```
# Feature Importance Plot (Decision Tree)
feature_importances = clf.feature_importances_

plt.figure(figsize=(10, 6))
plt.barh(top_features, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance from Decision Tree')
plt.show()
```