Importing the liabraries

In [37]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

from imblearn.over_sampling import SMOTE

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score

from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

import time

import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [2]:
```
df=pd.read_csv('machine_failure_dataset.csv')
df.head()
```

Out[2]:

	Temperature	Vibration	Power_Usage	Humidity	Machine_Type	Failure_Risk
0	74.967142	56.996777	8.649643	20.460962	Mill	1
1	68.617357	54.623168	9.710963	25.698075	Lathe	0
2	76.476885	50.298152	8.415160	27.931972	Drill	1
3	85.230299	46.765316	9.384077	39.438438	Lathe	1
4	67.658466	53.491117	6.212771	32.782766	Drill	1

In [3]:
```
# Basic info about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Temperature   1000 non-null   float64
 1   Vibration     1000 non-null   float64
 2   Power_Usage   1000 non-null   float64
 3   Humidity      1000 non-null   float64
 4   Machine_Type  1000 non-null   object 
 5   Failure_Risk  1000 non-null   int64  
dtypes: float64(4), int64(1), object(1)
memory usage: 47.0+ KB
```

Exploratory Data Analysis

In [4]:
```
df.describe()
```

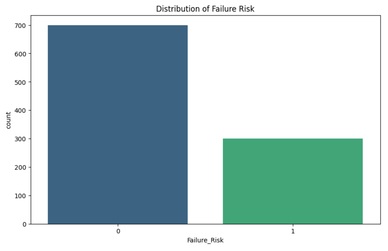
Out[4]:

	Temperature	Vibration	Power_Usage	Humidity	Failure_Risk
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	70.193321	50.354181	10.011668	29.906404	0.300000
std	9.792159	4.987272	1.966909	5.135663	0.458487
min	37.587327	35.298057	3.960976	15.352757	0.000000
25%	63.524097	46.968792	8.704001	26.312898	0.000000
50%	70.253006	50.315386	9.999498	30.000923	0.000000
75%	76.479439	53.644411	11.321831	33.334727	1.000000
max	108.527315	65.965538	17.852475	46.215465	1.000000

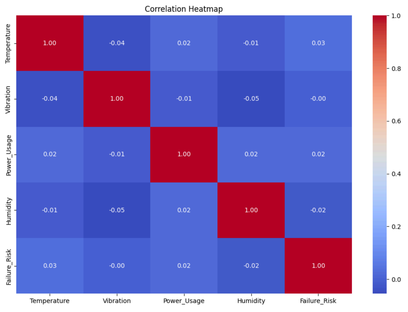
In [5]:
```
# checking for missing values
df.isnull().sum()
```

Out[5]:
```
Temperature     0
Vibration       0
Power_Usage     0
Humidity        0
Machine_Type    0
Failure_Risk    0
dtype: int64
```

In [6]:
```
# Visualize the distribution of Failure_Risk
plt.figure(figsize=(10,6))
sns.countplot(x='Failure_Risk', data=df, palette='viridis')
plt.title('Distribution of Failure Risk')
plt.show()
```



In [7]:
```
# Correlation heatmap
numeric_df=df.select_dtypes(include=[np.number])
plt.figure(figsize=(12,8))
sns.heatmap(numeric_df.corr(),  annot=True,  cmap='coolwarm',  fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



Feature Engineering

In [8]:
```
# Convert categorical variable 'Machine_Type' to dummy variables
df=pd.get_dummies(df,columns=['Machine_Type'], drop_first=True)


df.head()
```

Out[8]:

	Temperature	Vibration	Power_Usage	Humidity	Failure_Risk	Machine_Type_Lathe	Machine_Type_Mill
0	74.967142	56.996777	8.649643	20.460962	1	False	True
1	68.617357	54.623168	9.710963	25.698075	0	True	False
2	76.476885	50.298152	8.415160	27.931972	1	False	False
3	85.230299	46.765316	9.384077	39.438438	1	True	False
4	67.658466	53.491117	6.212771	32.782766	1	False	False

Model Building and Prediction

In [48]:
```
# Define the preprocessor with StandardScaler for numeric features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['Temperature', 'Vibration', 'Power_Usage', 'Humidity'])
    ])

# Load your dataset (assuming df is already defined)
X = df.drop(columns=['Failure_Risk'])
y = df['Failure_Risk']

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

# Function to evaluate model
def evaluate_model(model, X_train, X_test, y_train, y_test):
    start_train = time.time()
    model.fit(X_train, y_train)
    end_train = time.time()
    train_time = end_train - start_train

    start_pred = time.time()
    y_pred = model.predict(X_test)
    end_pred = time.time()
    pred_time = end_pred - start_pred

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Return runtime and accuracy
    return train_time + pred_time, accuracy

# List of classifiers to evaluate
classifiers = {
    'Logistic Regression': LogisticRegression(),
    'Support Vector Classifier': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(random_state=42),
    'XGBoost': xgb.XGBClassifier(random_state=42),
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(random_state=42)
}

results = []

# Iterate through models, creating pipelines, and evaluating
for name, model in classifiers.items():
    pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values
        ('scaler', StandardScaler()),  # Standardize numerical features
        ('classifier', model)  # Apply the model
    ])

    # Evaluate model
    run_time, accuracy = evaluate_model(pipeline, X_train, X_test, y_train, y_test)

    # Append results to the results list
    results.append({
        'Model': name,
        'Run Time (seconds)': run_time,
        'Accuracy': accuracy
    })

# Create a DataFrame with the results
results_df = pd.DataFrame(results)

# Display the results
print(results_df)
```

```
                          Model  Run Time (seconds)  Accuracy
0           Logistic Regression            0.024936     0.675
1  Support Vector Classifier              0.083510     0.675
2        K-Nearest Neighbors              0.022938     0.615
3              Random Forest              0.329296     0.620
4                    XGBoost              0.128754     0.595
5                Naive Bayes              0.010098     0.675
6              Decision Tree              0.028295     0.545
```

In [47]:
```
# Accessing best model and training

# Initialize and train the Navie Bayes model
# Train the model
model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

y_prob = model.predict_proba(X_test)[:, 1]

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, y_prob)

# results
print('Accuracy:', accuracy)
print('ROC AUC:', roc_auc)
print('\nClassification Report:\n', report)
```

```
Accuracy: 0.675
ROC AUC: 0.41455669455868946

Classification Report:
               precision    recall  f1-score   support

           0       0.68      1.00      0.81       135
           1       0.00      0.00      0.00        65
```

```
              precision    recall  f1-score   support

    accuracy                         0.68       200
   macro avg       0.34      0.50      0.40       200
weighted avg       0.46      0.68      0.54       200
```