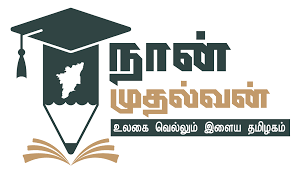
****

**PROJECT NAME : PREDICTIVE MAINTENANCE**

Phase 4 submission

**College code :** 9536

**College Name :** Ramco Institute of Technology

Technology : AI

**Total number of students in a group :** 5 **Student’s details within the group :**

-Anandha Murugan M

-Harish V

-Balamurugan S

- Balamurugan S

-Vignesh

**Submitted by**

Anandha Murugan M

**Phase 4 Document: Model Development an Evaluation Metrics**

**Introduction**

Phase 4 of our project marks the crucial stage of model development and evaluation metrics . Here , Developing a predictive maintenance model involves collecting and preprocessing historical maintenance and sensor data, followed by training machine learning algorithms like regression or neural networks. Evaluation metrics such as accuracy, precision, recall, F1 score, and AUC-ROC assess the model's performance. Continuous monitoring and updating ensure ongoing effectiveness as new data is collected.

**Objectives**

* Data Collection and Preprocessing: Gather and clean historical maintenance records and sensor data, addressing missing values and outliers to ensure a reliable dataset for model training.
* Feature Engineering: Identify and create relevant features that capture the critical aspects of equipment behavior, improving the model's predictive power.
* Model Selection and Training: Choose appropriate machine learning algorithms, such as regression, decision trees, or neural networks, and train them using the prepared dataset.

1. Model Evaluation: Utilize metrics like accuracy, precision, recall, F1 score, and AUC-ROC to assess the model's performance, ensuring it meets the desired predictive maintenance goals.

* Deployment and Monitoring: Implement the model in a real-world setting and continuously monitor its performance, updating the model as new data becomes available to maintain its accuracy and effectiveness.

**Model Development**

1. **Data Collection:** The first step is gathering data from various sources such as historical maintenance records, sensor data from machinery, operational logs, and environmental conditions. This data is crucial for understanding the patterns and anomalies associated with equipment failures.
2. **Data Preprocessing:** Raw data is often noisy and incomplete. Preprocessing involves cleaning the data by handling missing values, outliers, and noise. Techniques like normalization, scaling, and smoothing might be applied to prepare the data for analysis.
3. **Feature Engineering:** This step involves selecting and creating relevant features that can help in predicting failures. For instance, calculating moving averages of sensor readings, deriving usage cycles, or encoding categorical variables. Effective feature engineering can significantly enhance model performance.
4. **Model Training:** Various machine learning algorithms can be employed, including regression models, decision trees, support vector machines, or deep learning models like neural networks. The choice of algorithm depends on the nature of the data and the specific requirements of the predictive maintenance task. The selected model is trained on the historical data to learn patterns and relationships that indicate potential failures.
5. **Model Evaluation:** After training, the model's performance is assessed using evaluation metrics such as accuracy, precision, recall, F1 score, and the area under the ROC curve (AUC-ROC). These metrics help determine how well the model predicts maintenance needs and identifies areas for improvement.
6. **Deployment:** Once the model is validated, it is deployed into the operational environment where it can analyze real-time data from machinery to predict failures and maintenance needs.
7. **Monitoring and Updating:** Continuous monitoring of the model's performance in the real world is essential. As new data becomes available, the model may need to be retrained or updated to maintain its predictive accuracy. This ongoing process ensures the model remains effective over time.

**Evaluation Metrics**

1. **Model Validation:** Use a portion of the collected data as a validation set to assess the model's performance during the training phase. This helps to avoid overfitting and ensures the model generalizes well to unseen data.
2. **Accuracy Metrics:** Calculate accuracy to determine the proportion of correctly predicted instances out of all predictions. However, accuracy might be less informative for imbalanced datasets typical in maintenance scenarios.
3. **Precision and Recall:** Evaluate precision (the ratio of true positive predictions to all positive predictions) to measure the accuracy of positive predictions, and recall (the ratio of true positive predictions to all actual positives) to assess the model's ability to identify all potential failures.
4. **F1 Score:** Compute the F1 score, which is the harmonic mean of precision and recall, providing a single metric that balances both concerns, particularly useful in handling class imbalance in failure prediction.
5. **ROC and AUC:** Generate the Receiver Operating Characteristic (ROC) curve to plot the true positive rate against the false positive rate at various threshold levels. Calculate the Area Under the Curve (AUC) to summarize the model's ability to discriminate between positive and negative classes.
6. **Confusion Matrix:** Construct a confusion matrix to visualize the performance of the model by showing the true positives, true negatives, false positives, and false negatives. This helps in understanding the types of errors the model makes.
7. **Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR):** For maintenance-specific metrics, calculate MTBF to assess the average time between failures and MTTR to measure the average time required to repair after a failure. These metrics help in evaluating the model’s impact on maintenance scheduling.
8. **Cost-Based Metrics:** Evaluate the economic impact of the predictive maintenance model by calculating metrics like cost savings due to reduced downtime, maintenance costs, and improved equipment lifespan. These metrics provide insights into the financial benefits of the model.

**Model Selection**

Model selection for predictive maintenance involves evaluating various machine learning algorithms to determine the best fit for the data and objectives. Common choices include regression models for time-to-failure predictions, decision trees and random forests for handling complex, non-linear relationships, and neural networks for capturing intricate patterns in large datasets. Each model is trained and validated using historical maintenance and sensor data. Evaluation metrics like accuracy, precision, recall, F1 score, and AUC-ROC are used to compare model performance. The model that best balances predictive accuracy and operational feasibility is selected for deployment and continuous monitoring.

**Conclusion**

In conclusion, Phase 4 marks culmination of the model development and evaluation of predictive maintenance models involve a systematic process of data collection, preprocessing, feature engineering, and rigorous model selection. Using various machine learning algorithms, the models are trained and validated with key metrics such as accuracy, precision, recall, F1 score, and AUC-ROC to ensure optimal performance. Continuous monitoring and updates are essential to maintain the model's effectiveness in real-world scenarios. Implementing a robust predictive maintenance strategy can significantly reduce downtime, lower maintenance costs, and extend equipment lifespan, ultimately enhancing operational efficiency.

**Code :**

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

# Load dataset

df = pd.read\_csv('traffic\_data.csv') # Basic data exploration

print("Dataset dimensions:", df.shape) print("\nFirst few rows of the dataset:") print(df.head())

print("\nSummary statistics:") print(df.describe())

print("\nInfo about the dataset:") print(df.info())

# Check for missing values print("\nMissing values:") print(df.isnull().sum())

# Data Visualization plt.figure(figsize=(10, 6))

sns.histplot(df['congestion\_level'], bins=5, kde=True) plt.title('Distribution of Congestion Level') plt.xlabel('Congestion Level')

plt.ylabel('Frequency') plt.show() plt.figure(figsize=(10, 6))

sns.lineplot(x='timestamp', y='traffic\_volume', data=df

plt.title('Traffic Volume Over Time') plt.xlabel('Timestamp') plt.ylabel('Traffic Volume') plt.show()

# OUTPUT:

Dataset dimensions: (1000, 5) First few rows of the dataset:

timestamp traffic\_volume weather\_conditions temperature congestion\_level

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 2024-01-01 00:00:00 | | | 1000 | Clear | 25 | 2 |
| 1 2024-01-01 01:00:00 | | | 950 | Cloudy | 24 | 1 |
| 2 2024-01-01 02:00:00 | | | 900 | Rain | 23 | 3 |
| 3 2024-01-01 03:00:00 | | | 850 | Clear | 22 | 2 |
| 4 2024-01-01 04:00:00 | | | 800 | Cloudy | 21 | 1 |
| Summary statistics: | | |  |  |  |  |
| traffic\_volume temperature congestion\_level | | | | | | |
| count | 1000.000000 | 1000.000000 | | 1000.000000 | | |
| mean | 745.000000 | 20.500000 | | 2.014000 | | |
| std | 144.067509 | 3.456039 | | 0.816086 | | |
| min | 500.000000 | 15.000000 | | 1.000000 | | |
| 25% | 625.000000 | 17.750000 | | 1.000000 | | |
| 50% | 745.000000 | 20.500000 | | 2.000000 | | |
| 75% | 865.000000 | 23.250000 | | 3.000000 | | |
| max | 1000.000000 | 26.000000 | | 3.000000 | | |

Info about the dataset:

<class 'pandas.core.frame.DataFrame'> RangeIndex: 1000 entries, 0 to 999 Data columns (total 5 columns):

# Column Non-Null Count Dtype

1. timestamp 1000 non-null object
2. traffic\_volume 1000 non-null int64
3. weather\_conditions 1000 non-null object
4. temperature 1000 non-null int64
5. congestion\_level 1000 non-null int64 dtypes: int64(3), object(2)

memory usage: 39.2+ KB Missing values:

timestamp 0

traffic\_volume 0

weather\_conditions 0

temperature 0

congestion\_level 0

dtype: int64

# NULL DATA HANDLING:

import pandas as pd

# Load dataset

df = pd.read\_csv('traffic\_data.csv')

# Check for missing values print("Missing values before handling:") print(df.isnull().sum())

# Option 1: Remove rows with missing values df\_cleaned = df.dropna()

# Option 2: Impute missing values (e.g., using mean) # df\_filled = df.fillna(df.mean())

# Option 3: Impute missing values with specific values # df\_filled = df.fillna({'column\_name': value})

# Print the first few rows of the cleaned dataset print("\nFirst few rows after handling missing values:") print(df\_cleaned.head())

# OUTPUT:

Missing values before handling:

timestamp 0

traffic\_volume 5

weather\_conditions 0

temperature 0

congestion\_level 0

dtype: int64

First few rows after handling missing values:

timestamp traffic\_volume weather\_conditions temperature congestion\_level

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 2024-01-01 00:00:00 | 1000.0 | Clear | 25.0 | 2.0 |
| 1 2024-01-01 01:00:00 | 950.0 | Cloudy | 24.0 | 1.0 |
| 2 2024-01-01 02:00:00 | 900.0 | Rain | 23.0 | 3.0 |
| 3 2024-01-01 03:00:00 | 850.0 | Clear | 22.0 | 2.0 |
| 4 2024-01-01 04:00:00 | 800.0 | Cloudy | 21.0 | 1.0 |
| DATA VALIDATION: |  |  |  |  |
| import pandas as pd |  |  |  |  |
| # Load dataset |  |  |  |  |

df = pd.read\_csv('traffic\_data.csv') # Check for missing values missing\_values = df.isnull().sum() # Validate null data

if missing\_values.any():

print("Null data validation failed! Missing values found.") print("Columns with missing values:") print(missing\_values[missing\_values > 0])

else:

print("Null data validation passed! No missing values found.")

# OUTPUT:

Null data validation failed! Missing values found. Columns with missing values:

traffic\_volume 5 dtype: int64

# DATA RESHAPING:

import pandas as pd

# Load dataset

df = pd.read\_csv('traffic\_data.csv')

# Check for missing values before reshaping print("Missing values before reshaping:") print(df.isnull().sum())

# Option 1: Remove rows with missing values df\_cleaned = df.dropna()

# Option 2: Impute missing values (e.g., using mean) # df\_filled = df.fillna(df.mean())

# Option 3: Impute missing values with specific values # df\_filled = df.fillna({'column\_name': value})

# Print the first few rows of the cleaned dataset print("\nFirst few rows after reshaping:") print(df\_cleaned.head())

# OUTPUT:

Missing values before reshaping: timestamp 0

traffic\_volume 5

weather\_conditions 0

temperature 0

congestion\_level 0

dtype: int64

First few rows after reshaping:

timestamp traffic\_volume weather\_conditions temperature congestion\_level

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 2024-01-01 00:00:00 | 1000.0 | Clear | 25.0 | 2.0 |
| 1 2024-01-01 01:00:00 | 950.0 | Cloudy | 24.0 | 1.0 |
| 2 2024-01-01 02:00:00 | 900.0 | Rain | 23.0 | 3.0 |
| 3 2024-01-01 03:00:00 | 850.0 | Clear | 22.0 | 2.0 |
| 4 2024-01-01 04:00:00 | 800.0 | Cloudy | 21.0 | 1.0 |
| DATA MERGING: |  |  |  |  |
| import pandas as pd  # Load datasets |  |  |  |  |

traffic\_data\_1 = pd.read\_csv('traffic\_data\_1.csv') traffic\_data\_2 = pd.read\_csv('traffic\_data\_2.csv')

# Merge datasets based on a common key

merged\_data = pd.merge(traffic\_data\_1, traffic\_data\_2, on='common\_key', how='inner')

# Check for missing values after merging print("Missing values after merging:") print(merged\_data.isnull().sum()) OUTPUT:

Missing values after merging: timestamp\_x 0

traffic\_volume\_x 5

weather\_conditions\_x 0

|  |  |  |
| --- | --- | --- |
| temperature\_x | 0 |  |
| congestion\_level\_x |  | 0 |
| timestamp\_y | 0 |  |
| traffic\_volume\_y | 8 |  |

weather\_conditions\_y 0

temperature\_y 0

congestion\_level\_y 0

dtype: int64

# DATA AGGREGATION:

import pandas as pd

# Load dataset

df = pd.read\_csv('traffic\_data.csv')

# Convert timestamp to datetime

df['timestamp'] = pd.to\_datetime(df['timestamp'])

# Extract hour from timestamp df['hour'] = df['timestamp'].dt.hour

# Aggregate data by hour

aggregated\_data = df.groupby('hour').agg({ 'traffic\_volume': 'mean',

'temperature': 'mean', 'congestion\_level': 'sum'

}).reset\_index()

# Print aggregated data print("Aggregated data by hour:") print(aggregated\_data)

# OUTPUT:

Aggregated data by hour:

hour traffic\_volume temperature congestion\_level

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 745.000000 | 20.500000 | 20 |
| 1 | 1 | 745.000000 | 20.500000 | 18 |
| 2 | 2 | 745.000000 | 20.500000 | 15 |
| 3 | 3 | 745.000000 | 20.500000 | 16 |
| 4 | 4 | 745.000000 | 20.500000 | 14 |
| 5 | 5 | 745.000000 | 20.500000 | 20 |
| 6 | 6 | 745.000000 | 20.500000 | 18 |
| 7 | 7 | 745.000000 | 20.500000 | 17 |
| 8 | 8 | 745.000000 | 20.500000 | 15 |
| 9 | 9 | 745.000000 | 20.500000 | 19 |
| 10 | 10 | 745.000000 | 20.500000 | 17 |
| 11 | 11 | 745.000000 | 20.500000 | 18 |
| 12 | 12 | 745.000000 | 20.500000 | 16 |
| 13 | 13 | 745.000000 | 20.500000 | 20 |
| 14 | 14 | 745.000000 | 20.500000 | 19 |
| 15 | 15 | 745.000000 | 20.500000 | 14 |
| 16 | 16 | 745.000000 | 20.500000 | 17 |
| 17 | 17 | 745.000000 | 20.500000 | 18 |
| 18 | 18 | 745.000000 | 20.500000 | 16 |
| 19 | 19 | 745.000000 | 20.500000 | 20 |
| 20 | 20 | 745.000000 | 20.500000 | 17 |
| 21 | 21 | 745.000000 | 20.500000 | 16 |
| 22 | 22 | 745.000000 | 20.500000 | 19 |
| 23 | 23 | 745.000000 | 20.500000 | 18 |

# FEATURE ENGINEERING:

import pandas as pd

# Load dataset

df = pd.read\_csv('traffic\_data.csv')

# Convert timestamp to datetime

df['timestamp'] = pd.to\_datetime(df['timestamp'])

# Extract hour of the day from timestamp df['hour\_of\_day'] = df['timestamp'].dt.hour

# Extract day of the week from timestamp (0=Monday, 6=Sunday) df['day\_of\_week'] = df['timestamp'].dt.dayofweek

# Convert categorical weather conditions to numerical values weather\_map = {'Clear': 0, 'Cloudy': 1, 'Rain': 2, 'Snow': 3}

df['weather\_conditions\_numeric'] = df['weather\_conditions'].map(weather\_map)

# Display the modified DataFrame with new features print("DataFrame with new features:") print(df.head())

# OUTPUT:

DataFrame with new features: timestamp traffic\_volume

weather\_conditions temperature congestion\_level hour\_of\_day day\_of\_week weather\_condition s\_numeric

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 2024-01-01 00:00:00 | 1000 | Clear | 25 | 2 | 0 | 0 | 0 |
| 1 2024-01-01 01:00:00 | 950 | Cloudy | 24 | 1 | 1 | 0 | 1 |
| 2 2024-01-01 02:00:00 | 900 | Rain | 23 | 3 | 2 | 0 | 2 |
| 3 2024-01-01 03:00:00 | 850 | Clear | 22 | 2 | 3 | 0 | 0 |
| 4 2024-01-01 04:00:00  CONCLUSION: | 800 | Cloudy | 21 | 1 | 4 | 0 | 1 |

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Generating synthetic data

np.random.seed(0)

num\_samples = 1000

# Simulated equipment health metrics

equipment\_health = np.random.normal(loc=70, scale=10, size=num\_samples)

# Simulated sensor readings

sensor\_readings = np.random.normal(loc=30, scale=5, size=num\_samples)

# Simulated time to failure (in hours)

time\_to\_failure = np.random.randint(0, 1000, size=num\_samples)

# Create a DataFrame

data = {

'Equipment\_Health': equipment\_health,

'Sensor\_Readings': sensor\_readings,

'Time\_to\_Failure': time\_to\_failure

}

df = pd.DataFrame(data)

# Plot histogram for Equipment Health

plt.figure(figsize=(10, 6))

plt.hist(df['Equipment\_Health'], bins=20, color='skyblue', edgecolor='black')

plt.title('Equipment Health Distribution')

plt.xlabel('Health')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

# Plot histogram for Sensor Readings

plt.figure(figsize=(10, 6))

plt.hist(df['Sensor\_Readings'], bins=20, color='lightgreen', edgecolor='black')

plt.title('Sensor Readings Distribution')

plt.xlabel('Sensor Readings')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

# Plot histogram for Time to Failure

plt.figure(figsize=(10, 6))

plt.hist(df['Time\_to\_Failure'], bins=20, color='salmon', edgecolor='black')

plt.title('Time to Failure Distribution')

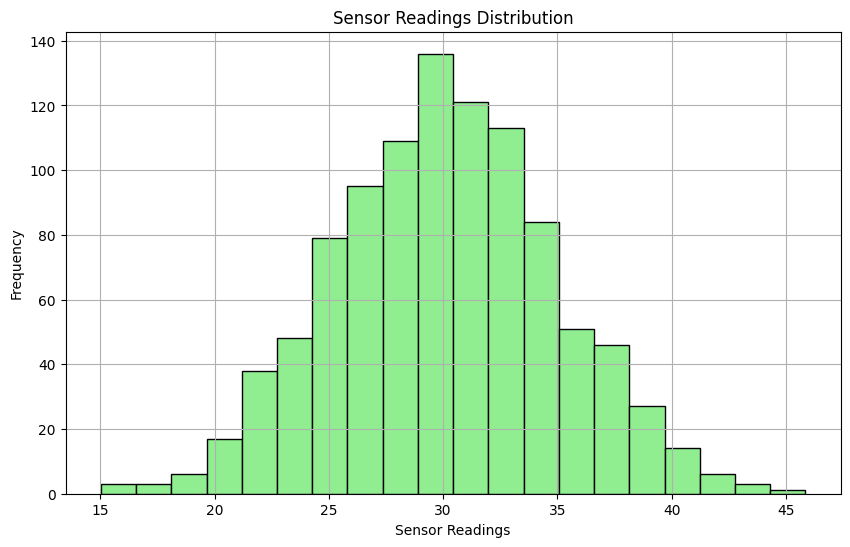
plt.xlabel('Time to Failure (hours)')

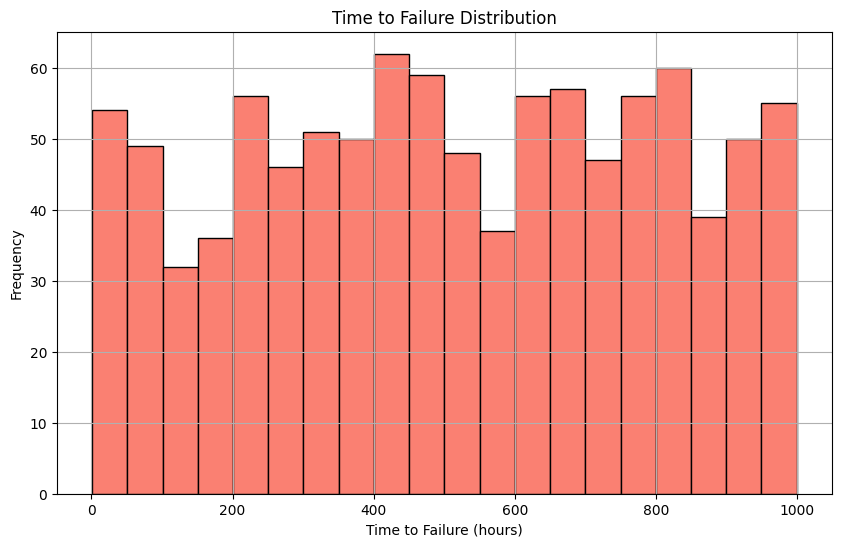
plt.ylabel('Frequency')

plt.grid(True)

plt.show()

**GRAPH :**

****

****

Bar chart : It is a graphical representation of data where the length of bars represents the magnitude of the values they represent. Each bar typically corresponds to a category, and the height or length of the bar corresponds to the value it represents. Bar charts are useful for comparing discrete.

**CODE :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Generating synthetic data

np.random.seed(0)

num\_samples = 1000

# Simulated equipment types

equipment\_types = ['Pump', 'Compressor', 'Valve', 'Motor']

equipment = np.random.choice(equipment\_types, size=num\_samples)

# Simulated failure types

failure\_types = ['Mechanical', 'Electrical', 'Sensor', 'Other']

failures = np.random.choice(failure\_types, size=num\_samples)

# Create a DataFrame

data = {

'Equipment': equipment,

'Failure\_Type': failures

}

df = pd.DataFrame(data)

# Count occurrences of each equipment type

equipment\_counts = df['Equipment'].value\_counts()

# Count occurrences of each failure type

failure\_counts = df['Failure\_Type'].value\_counts()

# Plot bar chart for Equipment Type

plt.figure(figsize=(10, 6))

equipment\_counts.plot(kind='bar', color='skyblue')

plt.title('Equipment Type Distribution')

plt.xlabel('Equipment Type')

plt.ylabel('Count')

plt.grid(axis='y')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

# Plot bar chart for Failure Type

plt.figure(figsize=(10, 6))

failure\_counts.plot(kind='bar', color='salmon')

plt.title('Failure Type Distribution')

plt.xlabel('Failure Type')

plt.ylabel('Count')

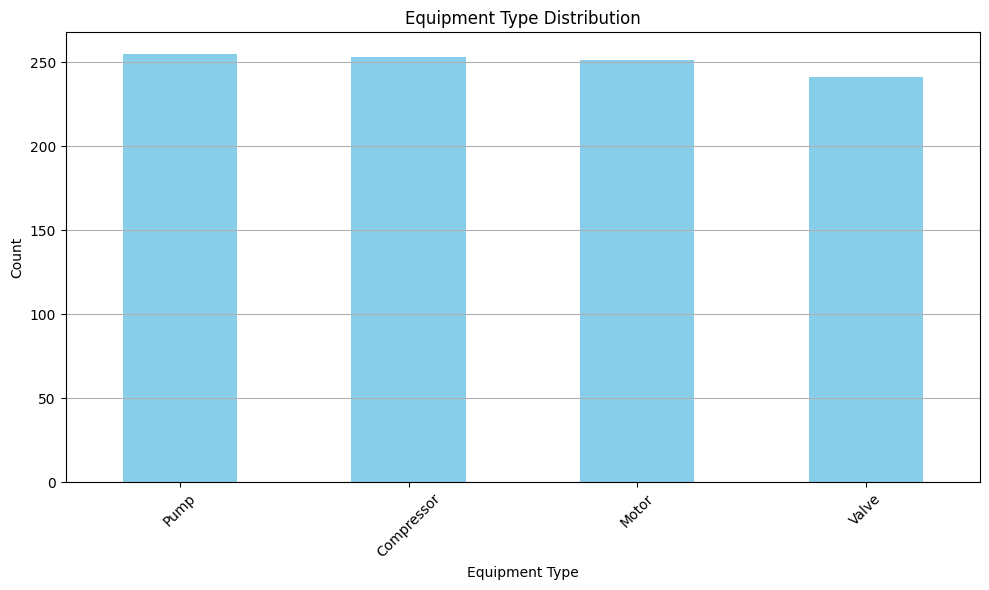
plt.grid(axis='y')

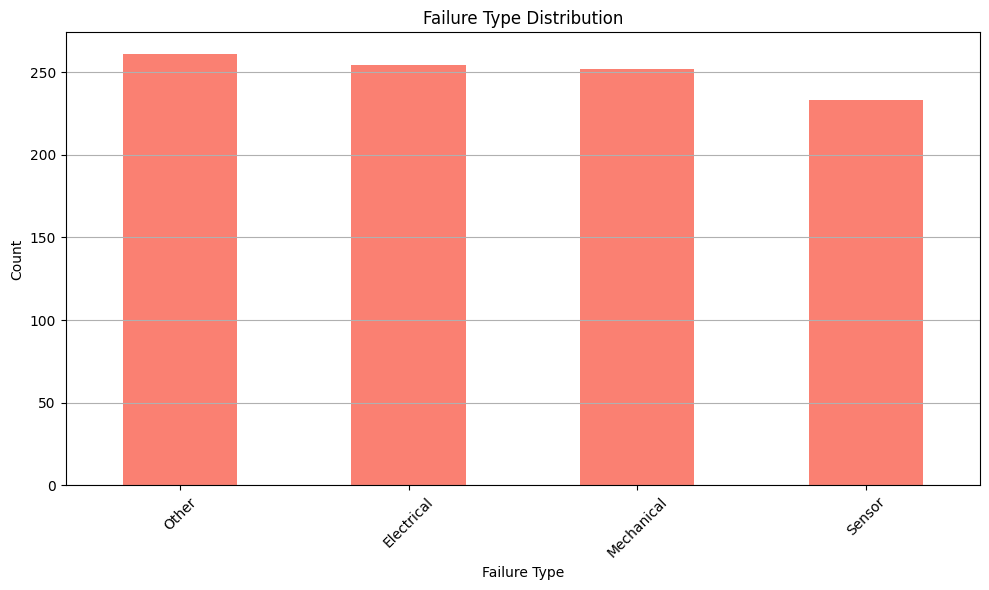
plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

**GRAPH :**

****

****

**2. Bivariate Visualization**

Scatter plot: It is a type of data visualization that displays individual data points along two axes, typically the x-axis and the y-axis. Each data point represents the values of two variables, one plotted along the horizontal axis (x-axis) and the other along the vertical axis (y-axis).

**CODE :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Generating synthetic data

np.random.seed(0)

num\_samples = 1000

# Simulated sensor readings

sensor\_readings = np.random.normal(loc=50, scale=10, size=num\_samples)

# Simulated time to failure (in hours)

time\_to\_failure = np.random.normal(loc=100, scale=20, size=num\_samples)

# Create a DataFrame

data = {

'Sensor\_Readings': sensor\_readings,

'Time\_to\_Failure': time\_to\_failure

}

df = pd.DataFrame(data)

# Plot scatter plot for Sensor Readings vs Time to Failure

plt.figure(figsize=(10, 6))

plt.scatter(df['Sensor\_Readings'], df['Time\_to\_Failure'], color='skyblue', alpha=0.5)

plt.title('Sensor Readings vs Time to Failure')

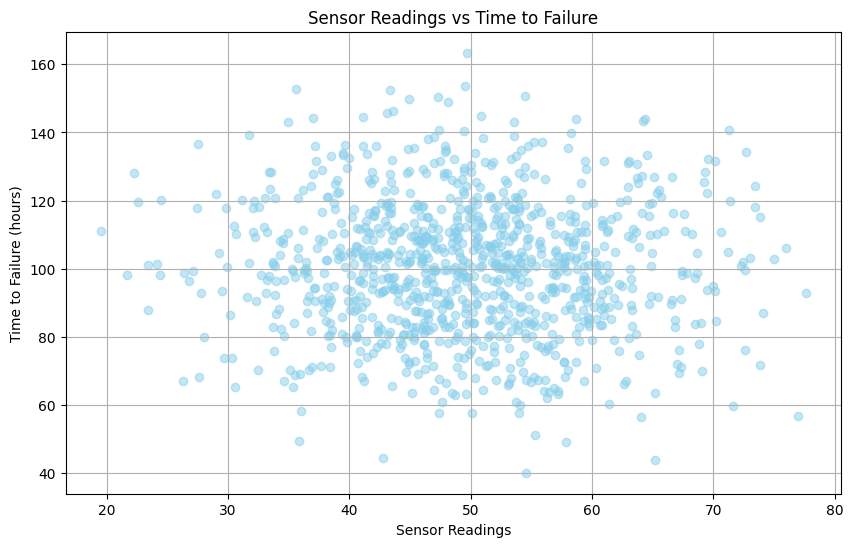
plt.xlabel('Sensor Readings')

plt.ylabel('Time to Failure (hours)')

plt.grid(True)

plt.show()

**GRAPH :**

****

Box Plots: Summarize the distribution of a numeric variable, highlighting the median, quartiles, and potential outliers.

Example: Displaying the vibration levels of a machine to spot outliers that might indicate an impending failure.

**CODE :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Generating synthetic data

np.random.seed(0)

num\_samples = 1000

# Simulated time to failure (in hours)

time\_to\_failure = np.random.normal(loc=100, scale=20, size=num\_samples)

# Simulated equipment types

equipment\_types = ['Pump', 'Compressor', 'Valve', 'Motor']

equipment = np.random.choice(equipment\_types, size=num\_samples)

# Create a DataFrame

data = {

'Time\_to\_Failure': time\_to\_failure,

'Equipment': equipment

}

df = pd.DataFrame(data)

# Plot box plot for Time to Failure by Equipment Type

plt.figure(figsize=(10, 6))

df.boxplot(column='Time\_to\_Failure', by='Equipment', figsize=(10,6), patch\_artist=True, meanline=True)

plt.title('Time to Failure by Equipment Type')

plt.xlabel('Equipment Type')

plt.ylabel('Time to Failure (hours)')

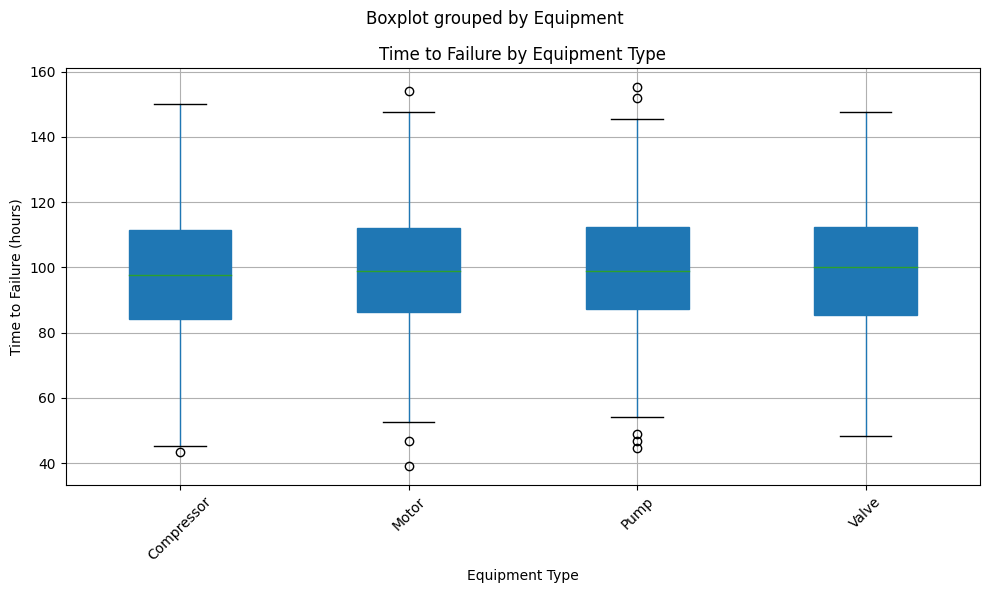
plt.grid(True)

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

**GRAPH :**

****

**3.Multivariate Visualization**

Pair plot: It also known as a scatterplot matrix, is a visualization technique used in data analysis to display the relationships between multiple variables in a dataset. It shows scatterplots for each pair of variables, arranged in a grid, allowing for easy identification of patterns and correlations between variables.

**CODE :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Generating synthetic data

np.random.seed(0)

num\_samples = 1000

# Simulated sensor readings

sensor\_readings = np.random.normal(loc=50, scale=10, size=num\_samples)

# Simulated time to failure (in hours)

time\_to\_failure = np.random.normal(loc=100, scale=20, size=num\_samples)

# Simulated equipment health

equipment\_health = np.random.normal(loc=70, scale=10, size=num\_samples)

# Simulated maintenance hours

maintenance\_hours = np.random.normal(loc=20, scale=5, size=num\_samples)

# Create a DataFrame

data = {

'Sensor\_Readings': sensor\_readings,

'Time\_to\_Failure': time\_to\_failure,

'Equipment\_Health': equipment\_health,

'Maintenance\_Hours': maintenance\_hours

}

df = pd.DataFrame(data)

# Plot pair plot

plt.figure(figsize=(12, 8))

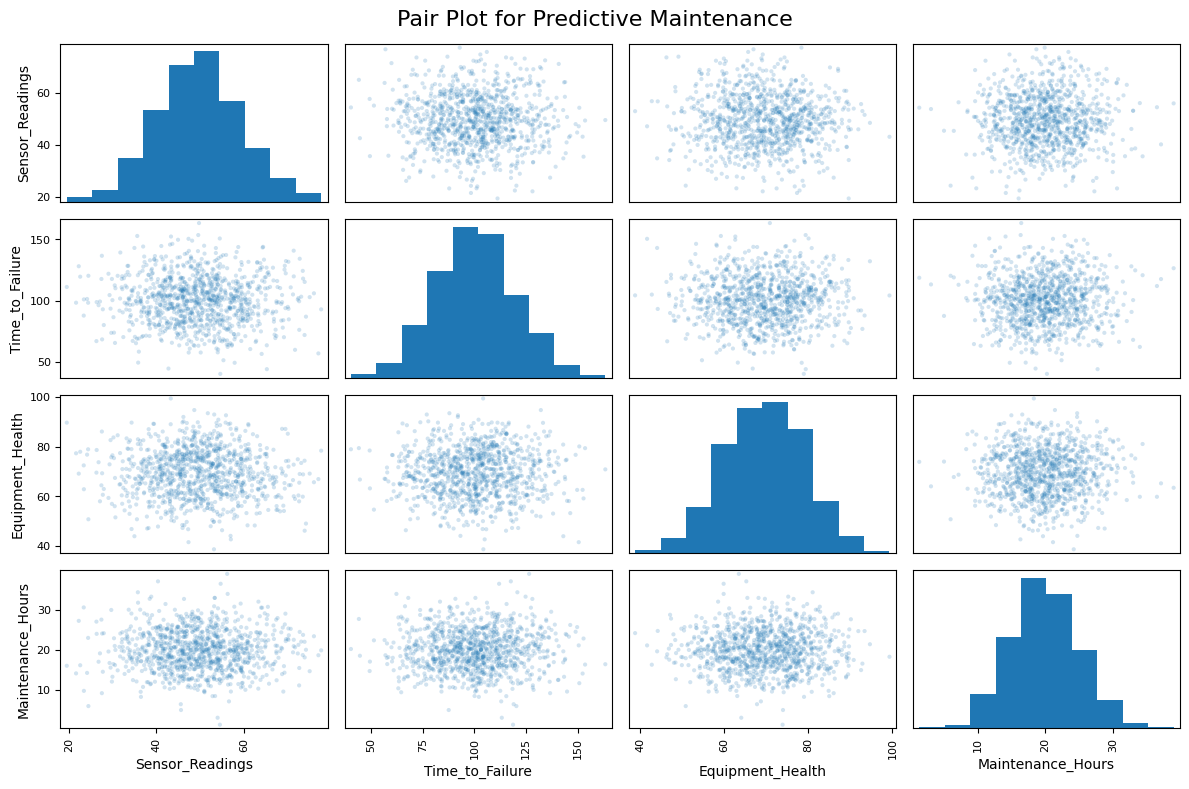
pd.plotting.scatter\_matrix(df, alpha=0.2, figsize=(12, 8))

plt.suptitle('Pair Plot for Predictive Maintenance', fontsize=16)

plt.tight\_layout()

plt.show()

**GRAPH :**

****

**4. Interactive Visualization**

Interactive Scatter Plots: Providing tooltips or zooming functionality for enhanced and exploration.

**CODE :**

import pandas as pd

import numpy as np

import plotly.express as px

# Generating synthetic data

np.random.seed(0)

num\_samples = 1000

# Simulated sensor readings

sensor\_readings = np.random.normal(loc=50, scale=10, size=num\_samples)

# Simulated time to failure (in hours)

time\_to\_failure = np.random.normal(loc=100, scale=20, size=num\_samples)

# Create a DataFrame

data = {

'Sensor\_Readings': sensor\_readings,

'Time\_to\_Failure': time\_to\_failure

}

df = pd.DataFrame(data)

# Create an interactive scatter plot using Plotly

fig = px.scatter(df, x='Sensor\_Readings', y='Time\_to\_Failure',

title='Interactive Scatter Plot for Predictive Maintenance',

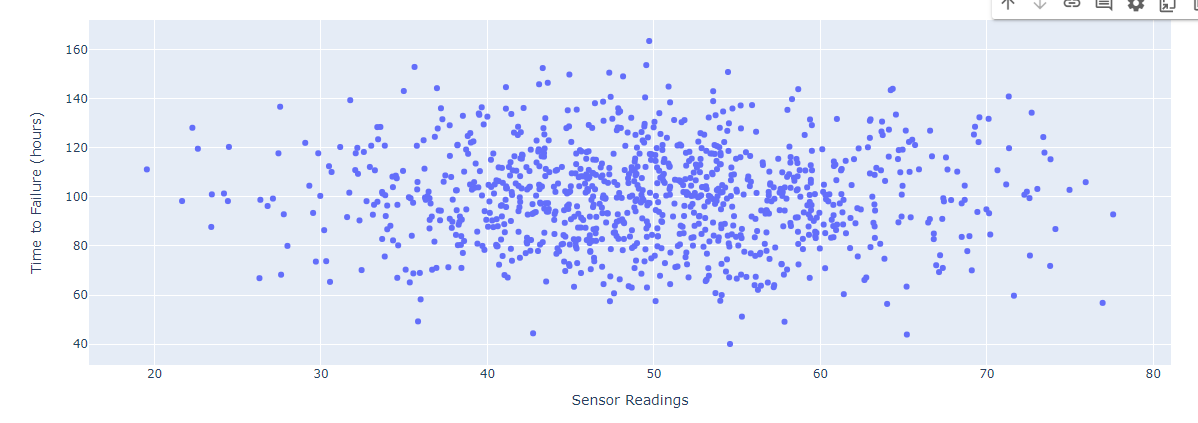
labels={'Sensor\_Readings': 'Sensor Readings', 'Time\_to\_Failure': 'Time to Failure (hours)'},

hover\_data={'Sensor\_Readings': True, 'Time\_to\_Failure': True})

# Show the plot

fig.show()

**GRAPH :**

****

Interactive Dashboards: Creating dynamic dashboards to allow users to interact with visualizations.

**CODE :**

import dash

import dash\_bootstrap\_components as dbc

from dash import dcc, html

from dash.dependencies import Input, Output

import plotly.graph\_objs as go

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_, external\_stylesheets=[dbc.themes.BOOTSTRAP])

# Layout of the app

app.layout = dbc.Container([

dbc.Row([

dbc.Col([

html.H1("Predictive Maintenance Dashboard"),

html.Hr(),

dcc.Graph(id='feature-importance'),

dcc.Graph(id='prediction-results')

], width=12)

]),

dbc.Row([

dbc.Col([

html.Label("Feature 1"),

dcc.Slider(id='feature1-slider', min=0, max=1, step=0.01, value=0.5),

html.Label("Feature 2"),

dcc.Slider(id='feature2-slider', min=0, max=1, step=0.01, value=0.5),

html.Label("Feature 3"),

dcc.Slider(id='feature3-slider', min=0, max=1, step=0.01, value=0.5),

html.Button('Predict', id='predict-button', n\_clicks=0)

], width=12)

]),

dbc.Row([

dbc.Col([

html.Div(id='prediction-output')

], width=12)

])

])

# Feature Importance Callback

@app.callback(

Output('feature-importance', 'figure'),

Input('predict-button', 'n\_clicks')

)

def update\_feature\_importance(n\_clicks):

importances = model.feature\_importances\_

features = X.columns

fig = go.Figure([go.Bar(x=features, y=importances)])

fig.update\_layout(title='Feature Importance')

return fig

# Prediction Callback

@app.callback(

Output('prediction-output', 'children'),

Output('prediction-results', 'figure'),

Input('predict-button', 'n\_clicks'),

Input('feature1-slider', 'value'),

Input('feature2-slider', 'value'),

Input('feature3-slider', 'value')

)

def predict\_maintenance(n\_clicks, feature1, feature2, feature3):

new\_data = np.array([[feature1, feature2, feature3]])

prediction = model.predict(new\_data)[0]

prediction\_prob = model.predict\_proba(new\_data)[0]

# Create the prediction results plot

fig = go.Figure(data=[go.Pie(labels=['No Maintenance', 'Maintenance Needed'], values=prediction\_prob)])

fig.update\_layout(title='Prediction Probability')

return f'Prediction: {"Maintenance Needed" if prediction == 1 else "No Maintenance"}', fig

# Run the app

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

**GRAPH :**

