```
import pandas as pd
import numpy as np

np.random.seed(42)
data = {
    'Temperature': np.random.normal(75, 5, 200),
    'Vibration': np.random.normal(0.7, 0.1, 200),
    'Pressure': np.random.normal(32, 3, 200),
    'Failure': np.random.choice([0, 1], size=200, p=[0.8, 0.2])
}
df = pd.DataFrame(data)
```

```
# Step 2: Introduce issues artificially
# a) Missing values (5%)
for col in ['Temperature', 'Vibration', 'Pressure']:
    df.loc[df.sample(frac=0.05).index, col] = np.nan
```

DataFrame — meaning you are directly accessing and modifying those specific rows by their index

```
# b) Wrong formatting – introduce strings
df.loc[5, 'Temperature'] = '75°C'
df.loc[10, 'Pressure'] = 'thirty'
df.loc[20, 'Vibration'] = '0.8mm/s'
```

```
and will raise an error in a future version of pandas. Value '75°C' has dtype incompatible with float64, please explicitly cast to a

and will raise an error in a future version of pandas. Value 'thirty' has dtype incompatible with float64, please explicitly cast to

and will raise an error in a future version of pandas. Value '0.8mm/s' has dtype incompatible with float64, please explicitly cast to
```

column-swapping operation applied to a random 3% subset of your DataFrame's

```
# c) Interchange some columns (simulate sensor wiring error)
temp_indices = df.sample(frac=0.03, random_state=1).index
df.loc[temp_indices, ['Temperature', 'Pressure']] = df.loc[temp_indices, ['Pressure', 'Temperature']].values
```

extends your DataFrame by duplicating the first five rows and appending them to the end, effectively increasing the dataset size by five new entries

```
# d) Add duplicate rows
df = pd.concat([df, df.iloc[0:5]], ignore_index=True)
```

```
df['Sensor_Status'] = np.random.choice(['OK', 'FAULTY'], size=len(df))
```

```
df
```

	Temperature	Vibration	Pressure	Failure	Sensor_Status
0	NaN	0.735779	27.216717	0	OK
1	74.308678	0.756078	30.201875	0	OK
2	78.238443	0.808305	32.015731	0	OK
3	82.615149	0.80538	32.140942	0	FAULTY
4	73.829233	0.562233	30.649804	0	FAULTY
...	...	...	...	...	...
200	NaN	0.735779	27.216717	0	OK
201	74.308678	0.756078	30.201875	0	OK
202	78.238443	0.808305	32.015731	0	OK
203	82.615149	0.80538	32.140942	0	OK
204	73.829233	0.562233	30.649804	0	FAULTY

205 rows × 5 columns

Next steps: Generate code with df New interactive sheet

```
print(df.isnull().sum())
```

```
Temperature      11
Vibration        10
Pressure         10
Failure           0
Sensor_Status     0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Temperature    194 non-null    object
 1   Vibration      195 non-null    object
 2   Pressure       195 non-null    object
 3   Failure        205 non-null    int64
 4   Sensor_Status  205 non-null    object
dtypes: int64(1), object(4)
memory usage: 8.1+ KB
```

```
for col in df.columns:
    print(col, df[col].unique())
```

```
 27.18066103023738  33.61030909760167  29.73904776414709  27.72323887130608
 27.18066103922728 32.61039090760167 29.73094776414709 27.73323887120698
 30.0602813472722422 37.06142490521769 34.64491927084835 31.976082076050147
 36.43983241667008 32.23210492294285 29.41614739601521 36.56937223180897
 33.61673013105398 28.888261537020632 31.428983965749175
 29.373145239845726 27.85160080710699 34.778532642594925
 37.728249921410395 73.28642741736616 33.68890771007171 30.04807229263452
 30.53862384870591 30.222818227283394 29.40802769096055 32.14556488383448
 29.507149650766888 32.811370477339516 31.849285671652588
 31.28315585940077 29.277309013875207 30.269686008295 34.26617367747727
 33.502751562873144 29.067334265604348 32.29799691628768 34.25416137011537
 26.991784156635884 33.63008057713981 30.01212872316246 33.711796005779476
 29.710222530372448 26.585353698006443 27.117372686350514
 32.14425483998414 32.779167505164445 29.287050124686775 33.91577737633212
 27.01543981319312 31.80176060405805 28.36695140071263 30.044491676593523
 29.418759904148143 30.846333367310525 35.01887842764332
 30.269324391430555 34.507076336195425 28.610879436027147
 33.58941253374585 36.3247058619737 24.585066499618133 29.60931423358857
 33.73121638154162 31.39086384187102 33.11343762011393 30.18804443985254
 32.2597693624187 31.53296829382376 35.50334618497942 32.76326252990364
 30.764369101632596 30.53718132778252 30.702325436541138 33.18335642713489
 30.737046557539212 32.86932457068924 38.22620239593632 34.61337411029508
 31.021929403496475 35.60364176649183 30.775773880935347 25.88562639446644
 28.97574106724779 26.387624236922434 30.945459547876073 32.05525513756866
 37.02931193682585 32.980782121292485 31.34269841357341 34.48821674355047
 32.7068436743257 34.3125955816609 27.564241262660474 35.43126212962079
 33.01548922248324 33.898345598318855 73.88268607337075 32.74466175890101
 30.621917301379266 29.450466891605625 34.49100744963274 29.4317485222734
 32.21469871165818 30.56702765970465 33.43693947723918 33.00098631586084
 35.112619832773696 31.190375194119888 29.06370885265308 75.29104359223
 33.131901479134555]
Failure [0 1]
Sensor_Status ['OK' 'FAULTY']
```

Removal Of duplicated data

```
df.duplicated().sum()
```

```
np.int64(4)
```

```
df = df.drop_duplicates().reset_index(drop=True)
print(f"✅ After removal, dataset shape: {df.shape}")
```

```
✅ After removal, dataset shape: (201, 5)
```

```
print("Duplicates remaining:", df.duplicated().sum())
```

```
Duplicates remaining: 0
```

```
# List of columns expected to be numeric
num_cols = ['Temperature', 'Vibration', 'Pressure']

for col in num_cols:
    # Create a mask for entries that **cannot be converted to numeric**
    invalid_mask = pd.to_numeric(df[col], errors='coerce').isna() & df[col].notna()

    if invalid_mask.any():
        print(f"\n⚠️ Non-numeric entries found in column '{col}':")
        print(df.loc[invalid_mask, col].unique())
    else:
        print(f"\n✅ Column '{col}' has all numeric values.")
```

```

⚠️ Non-numeric entries found in column 'Temperature':
['75°C']

⚠️ Non-numeric entries found in column 'Vibration':
['0.8mm/s']

⚠️ Non-numeric entries found in column 'Pressure':
['thirty']
```

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

```
import numpy as np
import re
```

```
# Mapping words to numbers for simple cases
word_to_num = {'thirty':30, }

def clean_numeric(x):
    if pd.isnull(x):
        return np.nan
    if isinstance(x, (int, float)):
        return float(x)
    if isinstance(x, str):
        s = x.lower().strip()
        # Remove common units
        s = re.sub(r'°c|°f|c\b|f\b|mm/s|mm_s|mm s|psi\b|bar\b', '', s)
        s = s.replace(',', '').strip()
        # Convert words to numbers if present
        if s in word_to_num:
            return float(word_to_num[s])
        # Keep only digits, dot, minus
        s_clean = re.sub(r'[^0-9.\-]', '', s)
        if s_clean == '':
            return np.nan
        try:
            return float(s_clean)
        except:
            return np.nan
    return np.nan

# Apply cleaning to all numeric columns
for col in ['Temperature','Vibration','Pressure']:
    df[col] = df[col].apply(clean_numeric)

# Check result
print("\n✅ Columns after cleaning:")
print(df[['Temperature','Vibration','Pressure']].head(10))
```

```
✅ Columns after cleaning:
   Temperature  Vibration   Pressure
0          NaN   0.735779  27.216717
1    74.308678   0.756078  30.201875
2    78.238443   0.808305  32.015731
3    82.615149   0.805380  32.140942
4    73.829233   0.562233  30.649804
5    75.000000   0.606217  33.868550
6    82.896064   0.751504  28.797139
7    78.837174   0.751379  31.572862
8    72.652628   0.751505  32.360887
9    77.712800   1.085273  33.543317
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Temperature    201 non-null    float64
 1   Vibration      201 non-null    float64
 2   Pressure       201 non-null    float64
 3   Failure        201 non-null    int64
 4   Sensor_Status  201 non-null    object
dtypes: float64(3), int64(1), object(1)
memory usage: 8.0+ KB
```

```
#  Option 1: Fill missing with mean
for col in ['Temperature','Vibration','Pressure']:
    df[col].fillna(df[col].mean(), inplace=True)
```

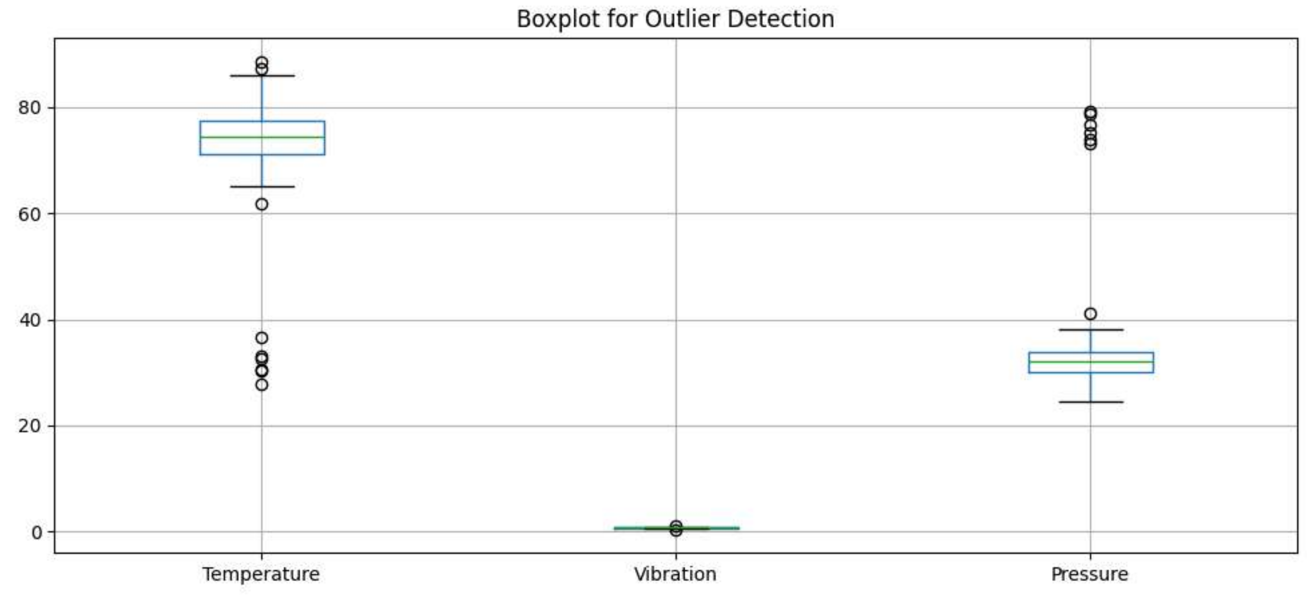
```
/tmp/ipython-input-718874585.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting 

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df[col].fillna(df[col].mean(), inplace=True)
```

```
df = df.drop(columns=['Sensor_Status'])
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  201 non-null    float64
 1   Vibration    201 non-null    float64
 2   Pressure     201 non-null    float64
 3   Failure      201 non-null    int64  
dtypes: float64(3), int64(1)
memory usage: 6.4 KB
None
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))
df[['Temperature', 'Vibration', 'Pressure']].boxplot()
plt.title("Boxplot for Outlier Detection")
plt.show()
```



```
from scipy.stats import zscore

# Compute Z-scores for numeric columns
z_scores = df[['Temperature','Vibration','Pressure']].apply(zscore)

# Find rows where any feature has Z-score > 3 (commonly considered an outlier)
outliers = df[(abs(z_scores) > 3).any(axis=1)]
print("Outlier rows:\n", outliers)
```

```
Outlier rows:
       Temperature  Vibration   Pressure  Failure
9       77.712800   1.085273  33.543317        1
34      36.507071   0.914394  79.112725        1
40      33.152196   0.620748  78.692333        0
58      30.339052   0.744382  76.656317        0
62      69.468325   0.375873  29.900823        0
102     27.804297   0.708111  73.286427        0
184     32.545599   0.624087  73.882686        0
198     30.667120   0.688546  75.291044        0
```

```
df_clean = df[(abs(z_scores) <= 3).all(axis=1)].reset_index(drop=True)
print("✅ Dataset shape after removing outliers:", df_clean.shape)
```

```
✅ Dataset shape after removing outliers: (193, 4)
```

here we do z score estimation but for non normal data set we will go with 1.Statical Method : IQR,Modified Z-Score 2.Transformation Techniques: Log Transformation,Box-Cox Transformation,Yeo-Johnson Transformation:

Scaling Techniques

Standardization (Z-score scaling)

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = df.copy()
df_scaled[['Temperature','Vibration','Pressure']] = scaler.fit_transform(
    df[['Temperature','Vibration','Pressure']]
)

print(df_scaled.head())
```

```
   Temperature  Vibration  Pressure  Failure
0     0.000000   0.288412 -0.736268        0
1     0.097805   0.500022 -0.365898        0
2     0.553524   1.044447 -0.140852        0
3     1.061073   1.013957 -0.125317        0
4     0.042205  -1.520676 -0.310323        0
```

Z-score scaling makes mean = 0 and std = 1 for all features Now all features have similar magnitude Effect on Gradient Descent: Gradients are balanced across features Each feature contributes proportionally to weight updates Faster convergence GD moves more smoothly towards minimu Stable learning rate You can use a normal learning rate without overshooting

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

Created Synthetic Dataset (200) Added artificial anomalies to simulate real-world data:

1. Non-numeric entries
2. Missing Value
3. Duplicates
4. Misformatted values

Initial Data Inspection: Column data types,Non-null counts,Presence of anomalies

Detected and Removed Duplicates Handled Non-Numeric Entries Handled Missing Values Detected and Removed Outliers |Z| > 3

Feature Scaling: Applied Z-score standardization using StandardScaler Mean ≈ 0, Standard Deviation ≈ 1 for all numeric features Explained why Z-score is better for regression, Min-Max normalization better for neural networks

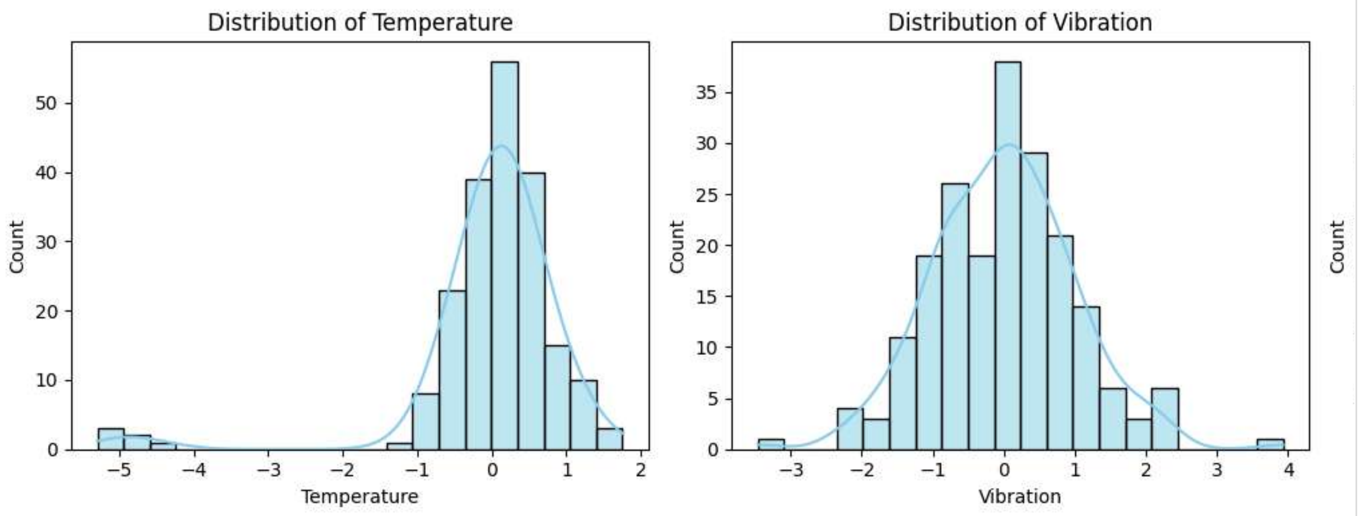
EDA : Explotry Data Analysis

```
# Basic statistics
print(df_scaled.describe())
```

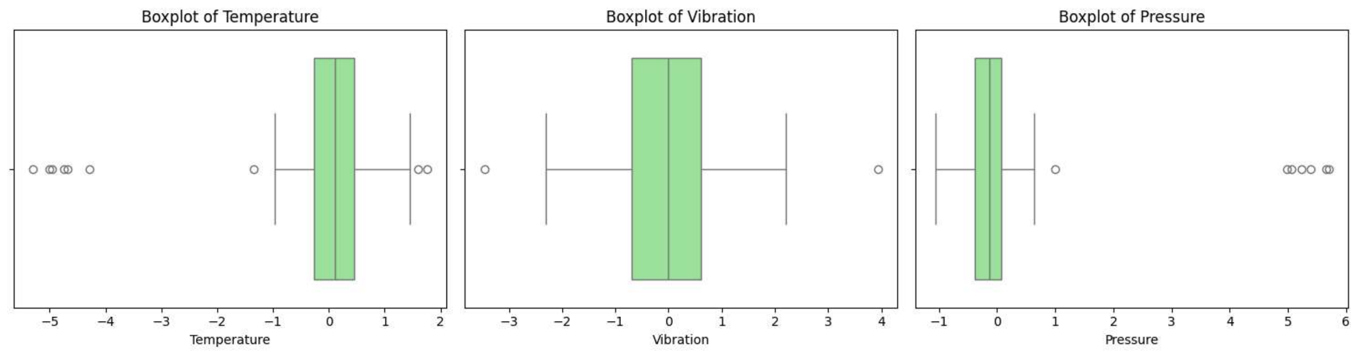
```
        Temperature     Vibration      Pressure     Failure
count  2.010000e+02  2.010000e+02  2.010000e+02  201.000000
mean   4.949054e-16 -7.732897e-16 -1.944271e-16    0.174129
std    1.002497e+00  1.002497e+00  1.002497e+00    0.380168
min   -5.295123e+00 -3.463342e+00 -1.062778e+00    0.000000
25%   -2.590643e-01 -7.012256e-01 -3.909448e-01    0.000000
50%    1.109183e-01 -2.314654e-15 -1.296784e-01    0.000000
75%    4.525778e-01  6.056681e-01  7.198883e-02    0.000000
max    1.755211e+00  3.931639e+00  5.702492e+00    1.000000
```

```
import matplotlib.pyplot as plt
import seaborn as sns

features = ['Temperature','Vibration','Pressure']
```

```
plt.figure(figsize=(15,4))
for i, col in enumerate(features):
    plt.subplot(1,3,i+1)
    sns.histplot(df_scaled[col], kde=True, bins=20, color='skyblue')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

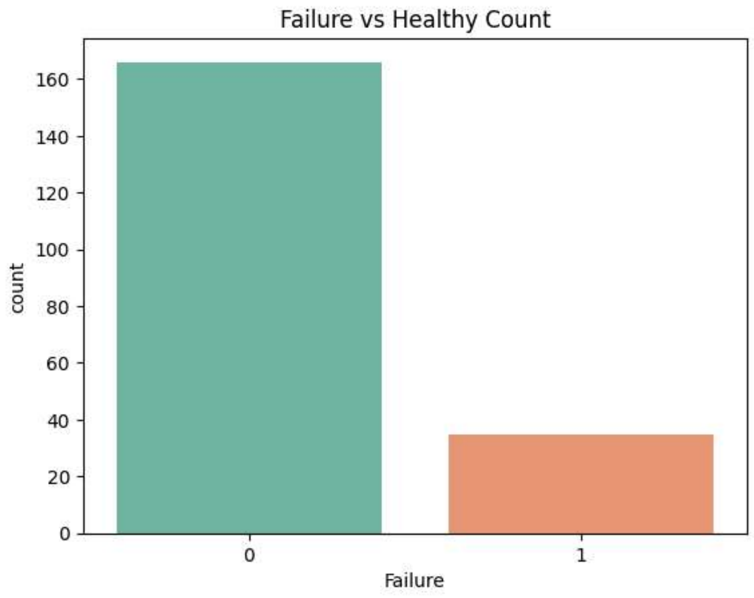


```
plt.figure(figsize=(15,4))
for i, col in enumerate(features):
    plt.subplot(1,3,i+1)
    sns.boxplot(x=df_scaled[col], color='lightgreen')
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```



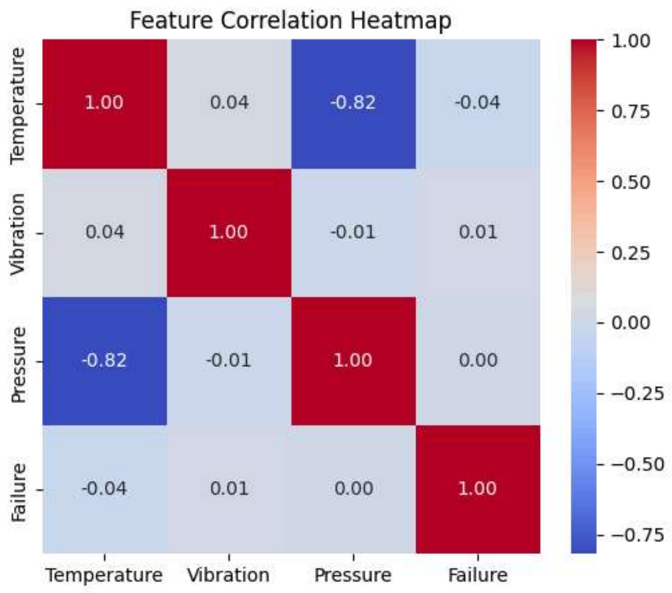
```
sns.countplot(x='Failure', data=df_scaled, palette='Set2')
plt.title('Failure vs Healthy Count')
plt.show()

print(df_scaled['Failure'].value_counts())
```

```
/tmp/ipython-input-1207308416.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.countplot(x='Failure', data=df_scaled, palette='Set2')
```

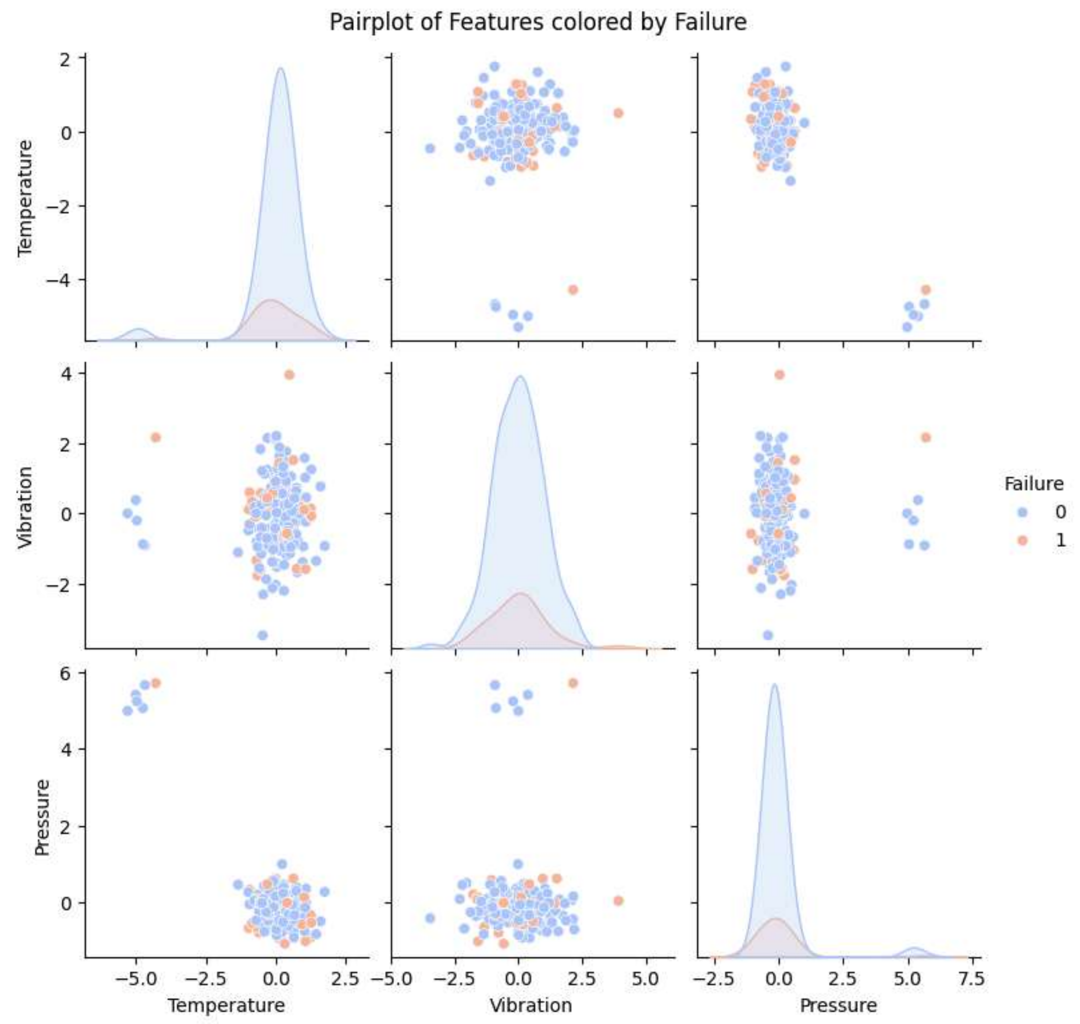


```
Failure
0    166
1     35
Name: count, dtype: int64
```

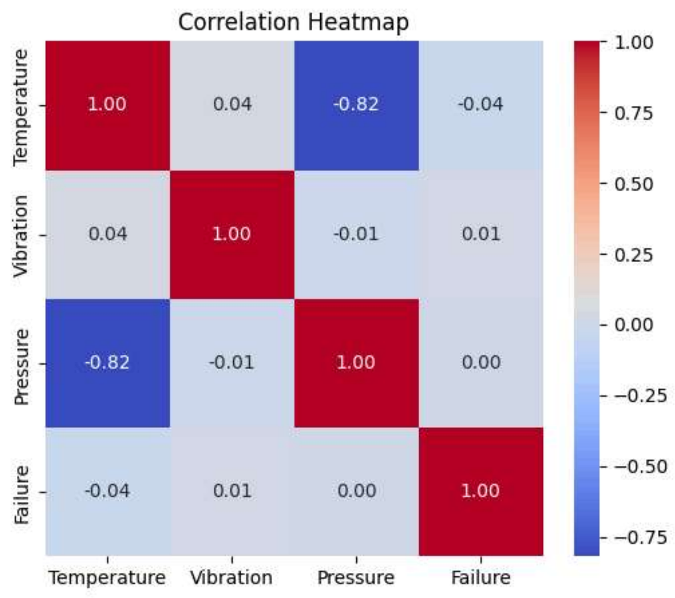
```
plt.figure(figsize=(6,5))
corr = df_scaled.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Feature Correlation Heatmap')
plt.show()
```



```
sns.pairplot(df_scaled, hue='Failure', vars=features, palette='coolwarm', diag_kind='kde')
plt.suptitle('Pairplot of Features colored by Failure', y=1.02)
plt.show()
```



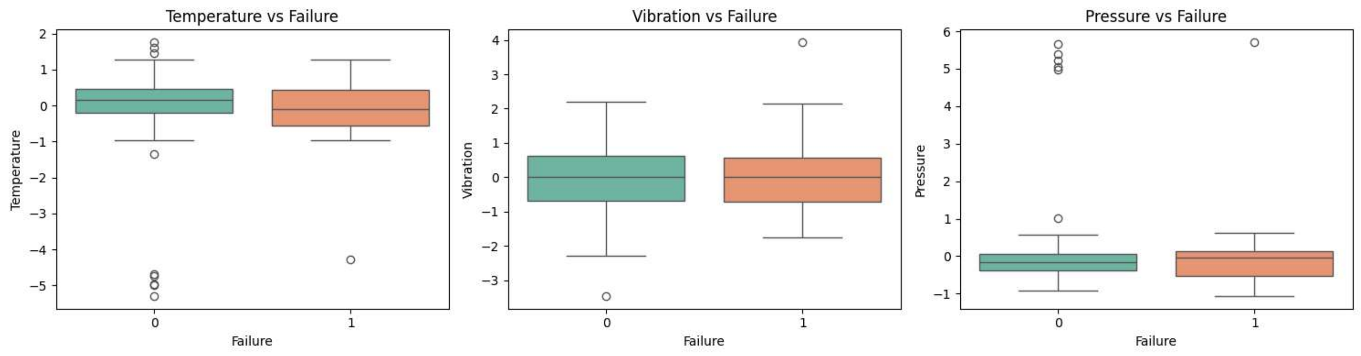
```
import seaborn as sns
import matplotlib.pyplot as plt

# Compute correlation
corr = df_scaled.corr()

# Heatmap
plt.figure(figsize=(6,5))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()

# Correlation of each feature with Failure
print("Feature correlation with Failure:")
print(corr['Failure'].sort_values(ascending=False))
```



```
Feature correlation with Failure:
Failure        1.000000
Vibration      0.013548
Pressure       0.001736
Temperature   -0.036679
Name: Failure, dtype: float64
```

```
features = ['Temperature', 'Vibration', 'Pressure']

plt.figure(figsize=(15,4))
for i, col in enumerate(features):
    plt.subplot(1,3,i+1)
    sns.boxplot(x='Failure', y=col, data=df_scaled, palette='Set2')
    plt.title(f'{col} vs Failure')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-2842312164.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.boxplot(x='Failure', y=col, data=df_scaled, palette='Set2')
/tmp/ipython-input-2842312164.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.boxplot(x='Failure', y=col, data=df_scaled, palette='Set2')
/tmp/ipython-input-2842312164.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.boxplot(x='Failure', y=col, data=df_scaled, palette='Set2')
```



```
from sklearn.linear_model import LogisticRegression

X = df_scaled[features]
y = df_scaled['Failure']

model = LogisticRegression()
model.fit(X, y)

importance = pd.DataFrame({'Feature': features, 'Coefficient': model.coef_[0]})
importance['Absolute'] = importance['Coefficient'].abs()
importance = importance.sort_values(by='Absolute', ascending=False)
print(importance)
```

```
       Feature  Coefficient  Absolute
0  Temperature    -0.247559  0.247559
2     Pressure    -0.191668  0.191668
1    Vibration     0.042553  0.042553
```

```
df=df_scaled.copy()
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

```
X = df[['Temperature', 'Vibration', 'Pressure']]
y = df['Failure']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
# Step 4: Model Evaluation
# -----------------------------
print("\n✅ Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\n✅ Classification Report:\n", classification_report(y_test, y_pred))
```

```
✅ Confusion Matrix:
 [[52  0]
 [ 9  0]]

✅ Classification Report:
               precision    recall  f1-score   support

           0       0.85      1.00      0.92        52
           1       0.00      0.00      0.00         9

    accuracy                           0.85        61
   macro avg       0.43      0.50      0.46        61
weighted avg       0.73      0.85      0.78        61

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined an
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined an
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined an
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
print(y.value_counts())
```