Machine Repair Predict

This project was done to identify or predict whether the working machine needs to be repaired in near future

Group No. : 01

Group Members : Rathnayaka R.M.K.D. EG/2021/4758 and Warnakulasuriya F.D.S. EG/2021/4850

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets pr
# You can also write temporary files to /kaggle/temp/, but they won't be saved outs
```

Load Dataset

In [2]:
```
data = pd.read_csv('predictive_maintenance.csv')
data #see whether dataset loaded
```

Out[2]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target
0	1	M14860	M	298.1	308.6	1551	42.8	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0
...	...	...	...	...	...	...	...	...	...
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0

10000 rows × 10 columns

Check whether there are null values

In [3]:
```
data.isnull().sum() #checking nulls
```

Out[3]:
```
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
```

Check whether there are duplicated values

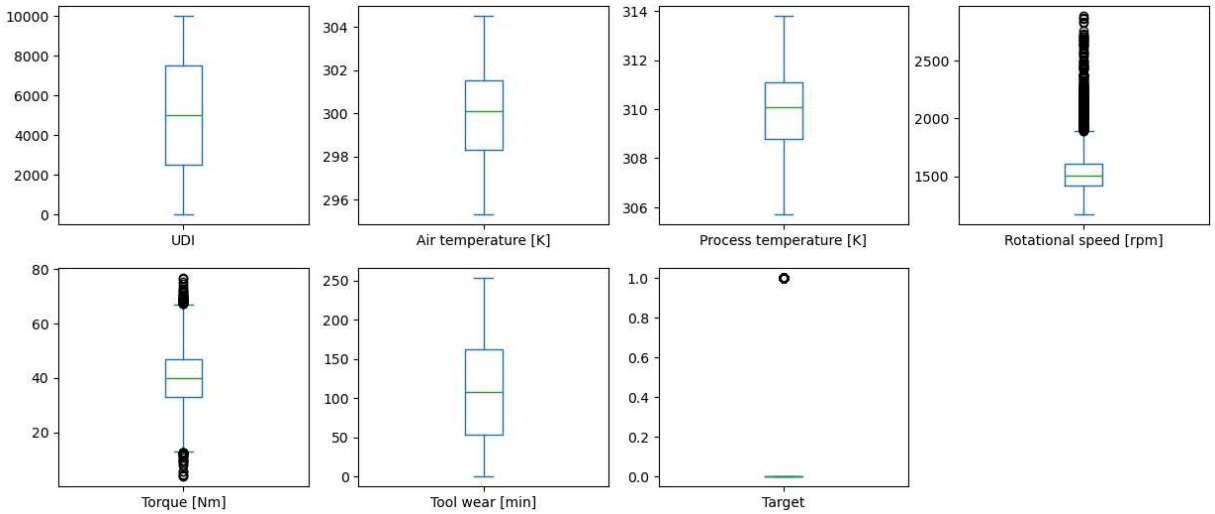
In [4]:
```
data.duplicated().sum() #checking duplicates
```

Out[4]: 0

Plot dataset

In [5]:
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (15, 20))
ax = fig.gca()
data.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
plt.show()
```

```
C:\Users\rmkav\AppData\Local\Temp\ipykernel_16672\2499133536.py:4: UserWarning: To o
utput multiple subplots, the figure containing the passed axes is being cleared.
  data.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
```



Calculate the outliers

In [6]:
```
# Counting outliers of Rotational speed column
# Calculate Q1, Q3, and IQR
Q1 = data["Rotational speed [rpm]"].quantile(0.25)
Q3 = data["Rotational speed [rpm]"].quantile(0.75)
IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Define outliers based on the bounds
outliers = data[(data["Rotational speed [rpm]"] < lower_bound) | (data["Rotational

# Count the number of outliers
number_of_outliers = outliers.shape[0]

# Output the number of outliers
number_of_outliers
```

Out[6]: 418

In [7]:
```
# Counting outliers of Torque [Nm] column
# Calculate Q1, Q3, and IQR
Q1 = data["Torque [Nm]"].quantile(0.25)
Q3 = data["Torque [Nm]"].quantile(0.75)
IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Define outliers based on the bounds
outliers = data[(data["Torque [Nm]"] < lower_bound) | (data["Torque [Nm]"] > upper_

# Count the number of outliers
number_of_outliers = outliers.shape[0]

# Output the number of outliers
number_of_outliers
```

Out[7]: 69

Duplicate the dataset

In [8]:
```
copied_df = data.copy() #copying dataset to maipulate the data
copied_df
```

Out[8]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target
0	1	M14860	M	298.1	308.6	1551	42.8	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0
...	...	...	...	...	...	...	...	...	...
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0

10000 rows × 10 columns

Checking unique values

In [9]:
```
# Check unique values in the column
unique_values = copied_df['Failure Type'].unique()
print(unique_values)
```

```
['No Failure' 'Power Failure' 'Tool Wear Failure' 'Overstrain Failure'
 'Random Failures' 'Heat Dissipation Failure']
```

In [10]:
```
unique_values = copied_df['Target'].unique()
print(unique_values)
```

```
[0 1]
```

In [11]:
```
unique_values = copied_df['Type'].unique()
print(unique_values)
```

```
['M' 'L' 'H']
```

Train Test Split

In [12]:
```
# train test validate spliting data
from sklearn.model_selection import train_test_split

# feature column and target column
X = copied_df.drop(columns=['Failure Type', 'Target','UDI','Product ID'])
y = copied_df['Failure Type']

X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, ra

# Split train+validation into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_si

# Verify the splits
print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (6000, 6)
Validation set shape: (2000, 6)
Test set shape: (2000, 6)
```

Visualise splitted dataset

In [13]:
```
X_train
```

Out[13]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
8588	L	297.3	307.8	1385	44.4	169
3178	M	300.2	309.4	1329	53.6	186
5200	L	303.8	312.8	1416	58.7	156
8889	M	297.7	308.8	2089	19.1	111
5789	M	301.8	311.5	1416	49.0	136
...	...	...	...	...	...	...
8871	M	297.9	309.1	1558	39.2	59
9826	L	298.4	309.2	1345	52.7	196
5268	L	303.4	312.9	1389	54.1	109
9666	M	299.1	310.3	1838	24.0	235
6090	L	300.9	310.8	1708	29.1	14

6000 rows × 6 columns

In [14]:
```
X_test
```

Out[14]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
6252	L	300.8	310.3	1538	36.1	198
4684	M	303.6	311.8	1421	44.8	101
1731	M	298.3	307.9	1485	42.0	117
4742	L	303.3	311.3	1592	33.7	14
4521	L	302.4	310.4	1865	23.9	129
...	...	...	...	...	...	...
6412	L	300.4	310.0	1423	44.2	189
8285	L	298.9	310.6	1387	52.7	2
7853	L	300.3	311.7	1317	56.5	7
1095	L	296.9	307.5	2721	9.3	18
6929	M	301.0	311.6	1301	55.0	9

2000 rows × 6 columns

In [15]:
```
X_val
```

Out[15]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
3188	M	300.1	309.2	1438	51.0	211
8293	M	298.9	310.4	1533	44.4	24
1710	L	298.2	307.8	1530	35.9	68
7510	L	300.5	311.8	1524	38.9	214
1461	L	298.9	310.2	1491	45.1	101
...	...	...	...	...	...	...
2834	L	300.3	309.1	1368	46.0	171
449	L	297.6	308.7	1622	37.9	79
6686	L	301.7	311.0	1715	24.8	17
3561	L	301.7	310.6	1486	38.5	78
9595	H	299.0	310.1	1432	50.0	45

2000 rows × 6 columns

In [16]:
```
y_train
```

Out[16]:
```
8588    No Failure
3178    No Failure
5200    No Failure
8889    No Failure
5789    No Failure
           ...
8871    No Failure
9826    No Failure
5268    No Failure
9666    No Failure
6090    No Failure
Name: Failure Type, Length: 6000, dtype: object
```

In [17]:
```
y_test
```

Out[17]:
```
6252       No Failure
4684       No Failure
1731       No Failure
4742       No Failure
4521       No Failure
            ...
6412       No Failure
8285       No Failure
7853       No Failure
1095    Power Failure
6929       No Failure
Name: Failure Type, Length: 2000, dtype: object
```

In [18]:
```
y_val
```

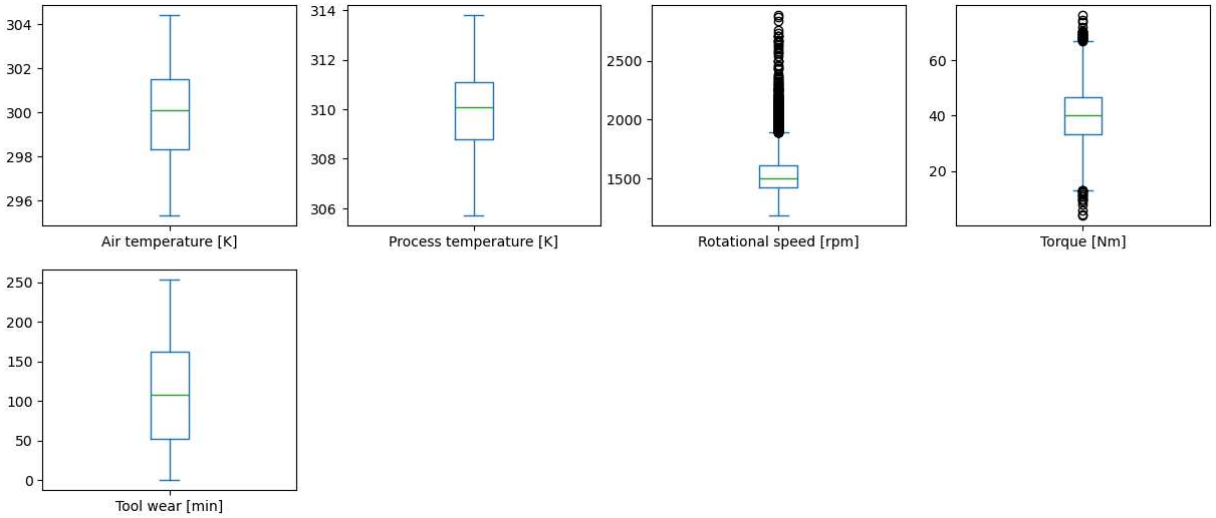
Out[18]:
```
3188           No Failure
8293           No Failure
1710           No Failure
7510    Tool Wear Failure
1461           No Failure
              ...
2834           No Failure
449            No Failure
6686           No Failure
3561           No Failure
9595           No Failure
Name: Failure Type, Length: 2000, dtype: object
```

Need to check on outliers

X_train

In [19]:
```
fig = plt.figure(figsize = (15, 20))
ax = fig.gca()
X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
plt.show()
```

```
C:\Users\rmkav\AppData\Local\Temp\ipykernel_16672\2695874865.py:3: UserWarning: To o
utput multiple subplots, the figure containing the passed axes is being cleared.
  X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
```



In [20]:
```python
# Counting outliers of Rotational speed column
# Calculate Q1, Q3, and IQR
Q1 = X_train["Rotational speed [rpm]"].quantile(0.25)
Q3 = X_train["Rotational speed [rpm]"].quantile(0.75)
IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Define outliers based on the bounds
outliers = X_train[(X_train["Rotational speed [rpm]"] < lower_bound) | (X_train["Ro

# Count the number of outliers
number_of_outliers = outliers.shape[0]

# Output the number of outliers
number_of_outliers
```

Out[20]: 284

In [21]:
```python
# Counting outliers of Torque [Nm] column
# Calculate Q1, Q3, and IQR
Q1 = X_train["Torque [Nm]"].quantile(0.25)
Q3 = X_train["Torque [Nm]"].quantile(0.75)
IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Define outliers based on the bounds
outliers = X_train[(X_train["Torque [Nm]"] < lower_bound) | (X_train["Torque [Nm]"]

# Count the number of outliers
number_of_outliers = outliers.shape[0]
```

```
# Output the number of outliers
number_of_outliers
```

Out[21]: 43

In [22]:
```
unique_values = X_train['Rotational speed [rpm]'].unique()
unique_values
```

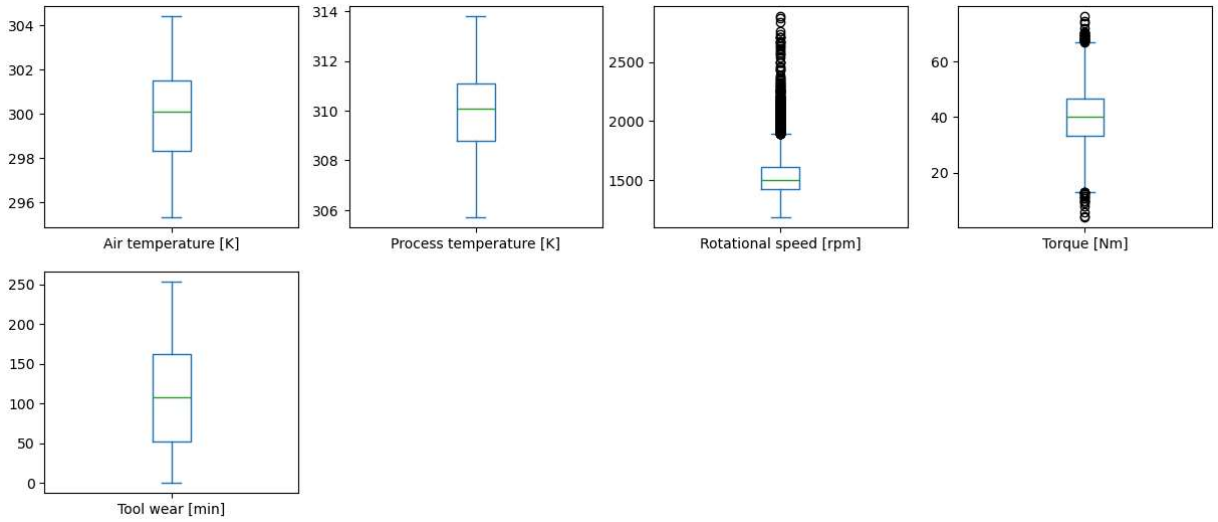
Out[22]:
```
array([1385, 1329, 1416, 2089, 1633, 1498, 1438, 1411, 1691, 1373, 1369,
       2016, 1430, 1410, 1706, 1304, 1446, 1488, 1561, 1523, 1499, 1454,
       1623, 1347, 1413, 1400, 1818, 1490, 1431, 1664, 1393, 1541, 1420,
       1518, 1622, 1642, 1447, 2051, 1482, 1335, 1481, 1437, 1572, 1644,
       1669, 1336, 1320, 1301, 1550, 1585, 1506, 1412, 1457, 1421, 1699,
       1877, 1602, 1640, 1466, 1374, 1439, 1892, 1881, 1382, 1574, 1233,
       1610, 1270, 1419, 1366, 1496, 1651, 1609, 1403, 1546, 1465, 1584,
       1469, 1375, 1535, 1485, 1619, 1476, 1537, 1645, 1360, 1749, 1621,
       1754, 1395, 1356, 1315, 1484, 1493, 1545, 1593, 1378, 1589, 1528,
       1514, 1460, 1637, 1422, 1614, 1324, 1433, 1740, 1817, 1470, 1486,
       1627, 1709, 1735, 1464, 1641, 1844, 1417, 1582, 1492, 1666, 1429,
       1407, 1281, 1548, 1389, 1692, 1453, 1330, 1442, 1379, 1504, 1638,
       1370, 1569, 1916, 2245, 1570, 1387, 1463, 2329, 1531, 1501, 1665,
       1715, 1405, 1898, 1565, 1615, 1830, 1834, 1823, 1440, 1571, 1508,
       1491, 1543, 1367, 1332, 1616, 1352, 1598, 1478, 1359, 1542, 1822,
       1444, 2009, 1354, 1620, 1296, 1309, 1436, 1402, 1677, 1409, 1592,
       1612, 1441, 1613, 1435, 1683, 1406, 1392, 1371, 1568, 1443, 1954,
       1384, 1328, 1552, 1340, 1357, 2579, 1842, 1618, 1268, 1383, 1727,
       1657, 1364, 1448, 1511, 1288, 1398, 1868, 1580, 1802, 1705, 1540,
       1479, 1764, 1794, 1449, 1475, 1344, 1694, 1668, 1451, 1628, 1811,
       1995, 1515, 1353, 1538, 1507, 1559, 1590, 1629, 1874, 1477, 1445,
       1558, 1517, 1500, 1520, 1591, 1289, 1282, 2032, 1458, 1553, 1519,
       1424, 1452, 1605, 1524, 1670, 1401, 1737, 1522, 1450, 1530, 1611,
       1634, 1798, 1337, 1386, 2833, 1312, 1462, 1269, 1775, 1744, 1434,
       1771, 1471, 1532, 1729, 1349, 1624, 1824, 1650, 2204, 1697, 1286,
       1710, 2267, 1363, 1836, 1314, 1534, 1390, 1595, 1880, 1494, 1290,
       1391, 1526, 1544, 1467, 1604, 1924, 1662, 1497, 1596, 1345, 1346,
       1361, 1704, 1394, 1547, 1675, 1607, 1732, 1527, 2024, 1397, 1674,
       1603, 1480, 1555, 1418, 1953, 1502, 1576, 1840, 1889, 1510, 1341,
       1468, 1513, 1297, 1680, 1399, 1516, 1456, 1689, 1525, 1949, 1427,
       2179, 2044, 1796, 1564, 1686, 1305, 1563, 1847, 1600, 1989, 1562,
       2709, 1743, 1414, 1455, 1529, 1937, 1473, 1635, 1426, 1887, 1631,
       1461, 2113, 1536, 1365, 1322, 1920, 1459, 1483, 1594, 1654, 1311,
       1331, 1948, 1755, 1556, 1859, 2022, 1820, 1649, 1338, 1408, 1728,
       1509, 1752, 1968, 1988, 1588, 1639, 1806, 1396, 1432, 1423, 1316,
       1852, 1425, 1751, 1873, 1896, 1388, 1586, 2384, 1310, 1512, 1716,
       1404, 1415, 1805, 1681, 1652, 2012, 1960, 1495, 1377, 1819, 2305,
       1698, 1656, 2049, 1348, 1578, 1696, 1472, 1910, 1816, 1489, 1786,
       1958, 1575, 1428, 1981, 1355, 1793, 1762, 1809, 1587, 1687, 1672,
       1551, 1533, 1785, 1333, 1708, 1678, 1864, 1741, 1487, 2372, 1917,
       1326, 1722, 1583, 1539, 1560, 1738, 1505, 1734, 1295, 1358, 1746,
       1760, 1884, 1567, 1279, 1810, 2000, 1321, 1855, 2102, 1318, 1808,
       1313, 1707, 1577, 1597, 1293, 1759, 1278, 1376, 1381, 1323, 1658,
       1334, 1566, 1969, 1863, 1720, 2218, 1807, 1254, 1726, 1646, 1812,
       2250, 1653, 1781, 1606, 1632, 1731, 1714, 1827, 1763, 1758, 1685,
       1617, 2011, 2174, 1362, 1753, 1725, 1905, 1927, 1814, 1549, 1688,
       1521, 2289, 1380, 1342, 1701, 1659, 1804, 1700, 1262, 1962, 1867,
       1350, 1503, 2092, 1263, 1647, 1985, 2271, 1787, 1317, 1351, 2074,
       2636, 1573, 1581, 1202, 1684, 1255, 2354, 2643, 1843, 1903, 2186,
       1302, 1679, 1799, 1713, 1368, 1474, 2144, 1319, 1878, 2496, 1579,
       1636, 1928, 1267, 1661, 1831, 1931, 1557, 1608, 1853, 1630, 1339,
       1372, 1774, 1766, 2028, 1258, 1625, 1343, 1946, 1673, 1306, 1298,
       1702, 2194, 2231, 1554, 1800, 1234, 1280, 1967, 1821, 1907, 1261,
       2065, 2248, 1951, 1256, 1871, 1791, 1922, 1965, 1667, 1923, 1682,
       1294, 1745, 1761, 1695, 1222, 1750, 1790, 1626, 1724, 1601, 1883,
       1721, 1736, 2119, 2083, 2706, 2437, 1227, 1742, 2158, 2197, 1711,
```

```
2084, 2161, 1801, 2297, 1325, 2270, 2137, 1911, 1893, 2093, 2312,
1307, 1266, 1829, 1703, 1676, 1599, 2659, 2678, 1671, 1308, 1991,
1828, 1655, 1908, 1274, 1643, 1693, 2564, 1243, 1769, 1327, 2098,
1972, 1292, 1770, 1926, 2034, 1776, 1284, 2262, 1648, 1858, 1780,
2001, 2116, 1747, 1832, 1841, 1719, 1271, 2097, 2323, 1249, 2090,
2183, 1663, 2293, 1803, 1964, 1918, 1717, 2595, 2007, 1950, 2886,
2465, 1959, 2567, 2182, 1778, 1815, 2057, 1876, 1300, 1181, 2153,
2127, 1895, 1235, 1748, 1942, 1915, 1909, 1888, 2029, 1690, 1772,
1983, 2141, 2344, 1718, 1869, 1941, 1784, 2256, 1275, 2008, 1739,
2129, 2006, 1906, 1913, 2424, 1283, 2109, 1226, 1825, 1925, 1975,
1303, 1788, 2737, 2151, 2261, 1767, 1845, 1813, 1712, 2206, 1940,
1789, 2288, 1850, 1238, 2052, 1980, 1299, 1733, 1660, 2676, 1260,
2071, 1792, 2456, 1276, 1236, 1977, 2056, 2240, 1902, 1894, 2117,
2355, 1272, 1207, 1229, 1285, 1782, 2235, 1779, 1890, 2033, 1891,
1773, 1783, 1854, 2130, 1897, 2266, 2440, 2157, 1851, 1838, 2068,
1921, 1882, 1192, 2047, 1984, 1861, 1899, 1961, 2663, 1826, 2069,
1757, 2010, 2211, 2101, 2710, 1730, 2497, 2077, 1856, 2760, 1875,
2617, 1978, 1839, 1872, 1932, 1287, 1765, 2243, 2140, 2142, 2203,
1862, 2188, 2003, 1865, 2165, 2874, 1944, 2540, 1837, 1723, 2013,
2021, 1866, 1976, 2073, 1797, 2103, 2370], dtype=int64)
```

Plotting train data

In [23]:
```
fig = plt.figure(figsize = (15, 20))
ax = fig.gca()
X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
plt.show()
```

```
C:\Users\rmkav\AppData\Local\Temp\ipykernel_16672\2695874865.py:3: UserWarning: To o
utput multiple subplots, the figure containing the passed axes is being cleared.
  X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
```



Encoding Failure types

In [24]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_train
```

Out[24]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [25]:
```
#cheking what were assigned
mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.cl
print("Mapping of original values to encoded values:", mapping)
```

```
Mapping of original values to encoded values: {'Heat Dissipation Failure': 0, 'No Fa
ilure': 1, 'Overstrain Failure': 2, 'Power Failure': 3, 'Random Failures': 4, 'Tool
Wear Failure': 5}
```

In [26]:
```
label_encoder = LabelEncoder()
y_test = label_encoder.fit_transform(y_test)
y_test
```

Out[26]:
```
array([1, 1, 1, ..., 1, 3, 1])
```

In [27]:
```
mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.cl
print("Mapping of original values to encoded values:", mapping)
```

```
Mapping of original values to encoded values: {'Heat Dissipation Failure': 0, 'No Fa
ilure': 1, 'Overstrain Failure': 2, 'Power Failure': 3, 'Random Failures': 4, 'Tool
Wear Failure': 5}
```

In [28]:
```
label_encoder = LabelEncoder()
y_val = label_encoder.fit_transform(y_val)
y_val
```

Out[28]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [29]:
```
mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.cl
print("Mapping of original values to encoded values:", mapping)
```

```
Mapping of original values to encoded values: {'Heat Dissipation Failure': 0, 'No Fa
ilure': 1, 'Overstrain Failure': 2, 'Power Failure': 3, 'Random Failures': 4, 'Tool
Wear Failure': 5}
```

Encoding Type

In [30]:
```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder(categories=[['L', 'M', 'H']])
X_train['Type'] = ordinal_encoder.fit_transform(X=X_train[['Type']])
X_train
```

Out[30]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
8588	0.0	297.3	307.8	1385	44.4	169
3178	1.0	300.2	309.4	1329	53.6	186
5200	0.0	303.8	312.8	1416	58.7	156
8889	1.0	297.7	308.8	2089	19.1	111
5789	1.0	301.8	311.5	1416	49.0	136
...	...	...	...	...	...	...
8871	1.0	297.9	309.1	1558	39.2	59
9826	0.0	298.4	309.2	1345	52.7	196
5268	0.0	303.4	312.9	1389	54.1	109
9666	1.0	299.1	310.3	1838	24.0	235
6090	0.0	300.9	310.8	1708	29.1	14

6000 rows × 6 columns

In [31]:
```
mapping = dict(zip(ordinal_encoder.categories_[0], [x for x in range(0,len(ordinal_
print("Mapping of original values to encoded values:", mapping)
```

```
Mapping of original values to encoded values: {'L': 0, 'M': 1, 'H': 2}
```

In [32]:
```
ordinal_encoder = OrdinalEncoder(categories=[['L', 'M', 'H']])
X_test['Type'] = ordinal_encoder.fit_transform(X=X_test[['Type']])
X_test
```

Out[32]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
6252	0.0	300.8	310.3	1538	36.1	198
4684	1.0	303.6	311.8	1421	44.8	101
1731	1.0	298.3	307.9	1485	42.0	117
4742	0.0	303.3	311.3	1592	33.7	14
4521	0.0	302.4	310.4	1865	23.9	129
...	...	...	...	...	...	...
6412	0.0	300.4	310.0	1423	44.2	189
8285	0.0	298.9	310.6	1387	52.7	2
7853	0.0	300.3	311.7	1317	56.5	7
1095	0.0	296.9	307.5	2721	9.3	18
6929	1.0	301.0	311.6	1301	55.0	9

2000 rows × 6 columns

In [33]:
```
mapping = dict(zip(ordinal_encoder.categories_[0], [x for x in range(0,len(ordinal_
print("Mapping of original values to encoded values:", mapping)
```

```
Mapping of original values to encoded values: {'L': 0, 'M': 1, 'H': 2}
```

In [34]:
```
ordinal_encoder = OrdinalEncoder(categories=[['L', 'M', 'H']])
X_val['Type'] = ordinal_encoder.fit_transform(X=X_val[['Type']])
X_val
```

Out[34]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
3188	1.0	300.1	309.2	1438	51.0	211
8293	1.0	298.9	310.4	1533	44.4	24
1710	0.0	298.2	307.8	1530	35.9	68
7510	0.0	300.5	311.8	1524	38.9	214
1461	0.0	298.9	310.2	1491	45.1	101
...	...	...	...	...	...	...
2834	0.0	300.3	309.1	1368	46.0	171
449	0.0	297.6	308.7	1622	37.9	79
6686	0.0	301.7	311.0	1715	24.8	17
3561	0.0	301.7	310.6	1486	38.5	78
9595	2.0	299.0	310.1	1432	50.0	45

2000 rows × 6 columns

In [35]:
```
mapping = dict(zip(ordinal_encoder.categories_[0], [x for x in range(0,len(ordinal_
print("Mapping of original values to encoded values:", mapping)
```

```
Mapping of original values to encoded values: {'L': 0, 'M': 1, 'H': 2}
```

Importing models and metrices

In [36]:
```
#importing SVM from sklearn
# Sklearn modules & classes
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
from sklearn import metrics

# Instantiate the Support Vector Classifier (SVC)
```

In [37]:
```
# handle outliers of the Torque column in X_train
# Calculate Q1, Q3, and IQR
Q1 = X_train["Rotational speed [rpm]"].quantile(0.25)
Q3 = X_train["Rotational speed [rpm]"].quantile(0.75)

IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

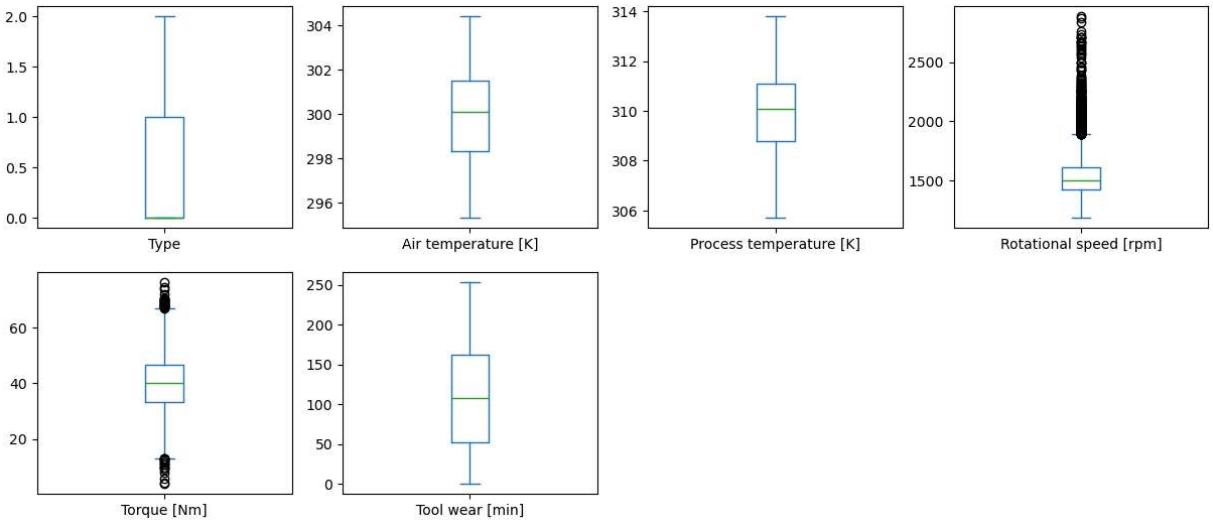
```
# Define outliers based on the bounds
outliers = X_train[(X_train["Rotational speed [rpm]"] < lower_bound) | (X_train["Ro

# Count the number of outliers
number_of_outliers = outliers.shape[0]

#Output the number of outliers
# number_of_outliers

# Display the values of the outliers
outliers_values = outliers["Rotational speed [rpm]"].values

# Output the outliers' values
outliers_values
# upper_bound


# Cap the values exceeding the upper bound
# X_train.loc[X_train["Rotational speed [rpm]"] > upper_bound, "Rotational speed [r

# # Cap the values below the lower bound
# X_train.loc[X_train["Rotational speed [rpm]"] < lower_bound, "Rotational speed [r

# X_train
```

Out[37]:
```
array([2089, 2016, 2051, 1892, 1916, 2245, 2329, 1898, 2009, 1954, 2579,
       1995, 2032, 2833, 2204, 2267, 1924, 1924, 2024, 1953, 1889, 1949,
       2179, 2044, 1989, 2709, 1889, 1937, 2113, 1920, 1948, 2022, 1968,
       1988, 1896, 2384, 2012, 1960, 2305, 2049, 1910, 1958, 1981, 2372,
       1917, 2000, 2102, 1981, 1969, 2218, 2250, 2329, 2011, 2174, 1905,
       1988, 1927, 1916, 2289, 1962, 1953, 2092, 1985, 2271, 2074, 2636,
       2354, 2643, 1903, 2186, 1910, 2144, 2496, 1928, 1931, 2028, 1981,
       1946, 1953, 2194, 2231, 1967, 2051, 1907, 2065, 2248, 1951, 1922,
       1965, 1923, 2119, 2083, 2706, 2437, 1892, 2158, 2197, 2084, 2161,
       1989, 2297, 1922, 2270, 2137, 1911, 1893, 2093, 2312, 1958, 1981,
       1924, 2659, 1924, 2678, 1967, 1927, 1991, 1908, 1924, 2564, 2098,
       1972, 1926, 2034, 1995, 2262, 1903, 2001, 2116, 1951, 1911, 2097,
       1922, 2323, 2090, 2183, 2293, 2174, 1964, 1918, 1989, 1981, 2595,
       2007, 1950, 2886, 2465, 2102, 1959, 2567, 2182, 2057, 2153, 2127,
       1895, 1942, 1915, 1909, 1888, 2029, 1989, 1889, 1983, 1910, 2141,
       2007, 2344, 1928, 2141, 1941, 2256, 1917, 2008, 1948, 2129, 2006,
       1906, 2098, 1892, 1889, 1913, 2194, 2424, 2141, 2161, 2008, 2109,
       1925, 1975, 2008, 1991, 2737, 2151, 1927, 2261, 1903, 2496, 2206,
       1940, 2288, 2052, 1980, 1969, 2676, 1962, 2071, 2456, 2183, 2113,
       1977, 2056, 2240, 1902, 1894, 2117, 2355, 1907, 1896, 2235, 1951,
       1918, 2056, 1895, 1890, 1969, 2033, 1891, 1960, 2130, 1897, 2266,
       2440, 2157, 1942, 2068, 1921, 1910, 2047, 1984, 1915, 2090, 1899,
       1961, 2663, 2069, 2010, 2211, 2101, 2710, 2497, 2077, 2129, 2760,
       2617, 1968, 1924, 2000, 1978, 1958, 1932, 2243, 2140, 2142, 2203,
       2188, 2003, 2165, 2874, 1944, 2540, 1925, 1946, 1909, 2013, 2021,
       1976, 2073, 2103, 2370, 1983, 2157, 1931, 1972, 1889], dtype=int64)
```

In [38]:
```
# X_test = X_test.drop('Rotational speed [rpm]', axis=1)
# X_train = X_train.drop('Rotational speed [rpm]', axis=1)
# X_val = X_val.drop('Rotational speed [rpm]', axis=1)
```

In [39]:
```
fig = plt.figure(figsize = (15, 20))
ax = fig.gca()
X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
plt.show()
```

```
C:\Users\rmkav\AppData\Local\Temp\ipykernel_16672\2695874865.py:3: UserWarning: To o
utput multiple subplots, the figure containing the passed axes is being cleared.
  X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
```



Handling outliers

In [40]:
```
# handle outliers of the Torque column in X_train
# Calculate Q1, Q3, and IQR
Q1 = X_train["Rotational speed [rpm]"].quantile(0.25)
Q3 = X_train["Rotational speed [rpm]"].quantile(0.75)

IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Define outliers based on the bounds
outliers = X_train[(X_train["Rotational speed [rpm]"] < lower_bound) | (X_train["Ro

# Count the number of outliers
number_of_outliers = outliers.shape[0]

#Output the number of outliers
# number_of_outliers

# Display the values of the outliers
outliers_values = outliers["Rotational speed [rpm]"].values

# Output the outliers' values
outliers_values
# upper_bound
```

```
# Cap the values exceeding the upper bound
X_train.loc[X_train["Rotational speed [rpm]"] > upper_bound, "Rotational speed [rpm

# Cap the values below the lower bound
X_train.loc[X_train["Rotational speed [rpm]"] < lower_bound, "Rotational speed [rpm

X_train
```

```
C:\Users\rmkav\AppData\Local\Temp\ipykernel_16672\1116835082.py:30: FutureWarning: S
etting an item of incompatible dtype is deprecated and will raise an error in a futu
re version of pandas. Value '1887.125' has dtype incompatible with int64, please exp
licitly cast to a compatible dtype first.
  X_train.loc[X_train["Rotational speed [rpm]"] > upper_bound, "Rotational speed [rp
m]"] = upper_bound
```

Out[40]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
8588	0.0	297.3	307.8	1385.000	44.4	169
3178	1.0	300.2	309.4	1329.000	53.6	186
5200	0.0	303.8	312.8	1416.000	58.7	156
8889	1.0	297.7	308.8	1887.125	19.1	111
5789	1.0	301.8	311.5	1416.000	49.0	136
...	...	...	...	...	...	...
8871	1.0	297.9	309.1	1558.000	39.2	59
9826	0.0	298.4	309.2	1345.000	52.7	196
5268	0.0	303.4	312.9	1389.000	54.1	109
9666	1.0	299.1	310.3	1838.000	24.0	235
6090	0.0	300.9	310.8	1708.000	29.1	14

6000 rows × 6 columns

In [41]:

```
# handle outliers of the Torque column in X_train
# Calculate Q1, Q3, and IQR
Q1 = X_train["Torque [Nm]"].quantile(0.25)
Q3 = X_train["Torque [Nm]"].quantile(0.75)

IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Define outliers based on the bounds
outliers = X_train[(X_train["Torque [Nm]"] < lower_bound) | (X_train["Torque [Nm]"]

# Count the number of outliers
```

```
number_of_outliers = outliers.shape[0]

# Output the number of outliers
# number_of_outliers

# Display the values of the outliers
outliers_values = outliers["Torque [Nm]"].values

# Output the outliers' values
# outliers_values


# Cap the values exceeding the upper bound
X_train.loc[X_train["Torque [Nm]"] > upper_bound, "Torque [Nm]"] = upper_bound

# Cap the values below the lower bound
X_train.loc[X_train["Torque [Nm]"] < lower_bound, "Torque [Nm]"] = lower_bound

X_train
```

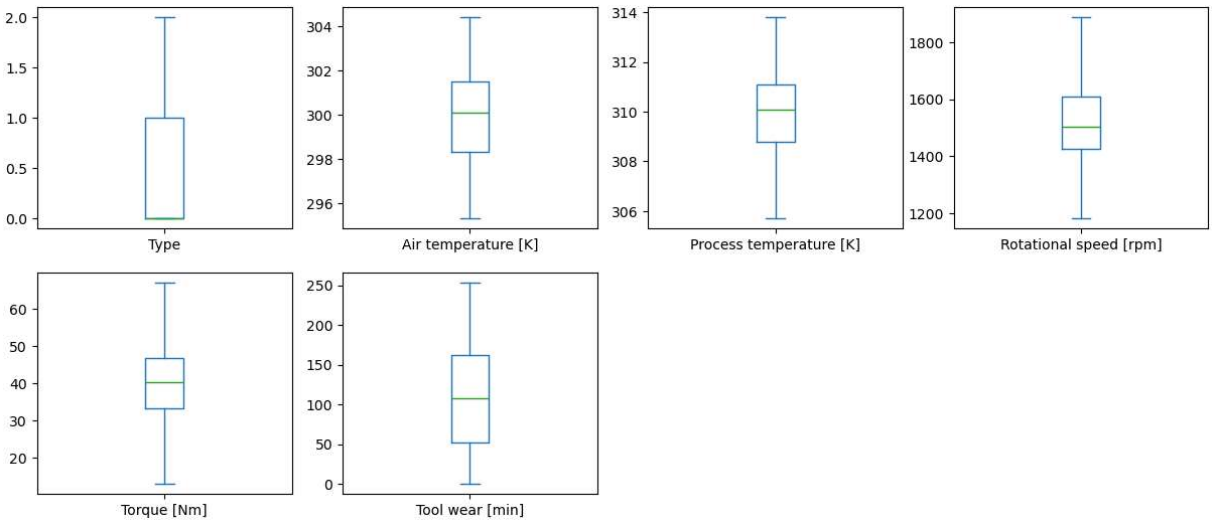
Out[41]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
8588	0.0	297.3	307.8	1385.000	44.4	169
3178	1.0	300.2	309.4	1329.000	53.6	186
5200	0.0	303.8	312.8	1416.000	58.7	156
8889	1.0	297.7	308.8	1887.125	19.1	111
5789	1.0	301.8	311.5	1416.000	49.0	136
...	...	...	...	...	...	...
8871	1.0	297.9	309.1	1558.000	39.2	59
9826	0.0	298.4	309.2	1345.000	52.7	196
5268	0.0	303.4	312.9	1389.000	54.1	109
9666	1.0	299.1	310.3	1838.000	24.0	235
6090	0.0	300.9	310.8	1708.000	29.1	14

6000 rows × 6 columns

In [42]:
```
fig = plt.figure(figsize = (15, 20))
ax = fig.gca()
X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
plt.show()
```

```
C:\Users\rmkav\AppData\Local\Temp\ipykernel_16672\2695874865.py:3: UserWarning: To o
utput multiple subplots, the figure containing the passed axes is being cleared.
  X_train.plot(ax=ax, kind = 'box', subplots=True, layout=(6,4), sharex=False)
```



Creating model and predicting with SVM

In [43]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Train the model on the training set
svc = SVC(C=1.0, random_state=1, kernel='linear')
svc.fit(X_train, y_train)

# Evaluate the model on the validation set
y_val_pred = svc.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)

print("Validation Accuracy: %.3f" % val_accuracy)
print("Validation Classification Report:\n", classification_report(y_val, y_val_pre

# If satisfied with validation performance, evaluate the model on the test set
y_test_pred = svc.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Test Accuracy: %.3f" % test_accuracy)
print("Test Classification Report:\n", classification_report(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
```

```
Validation Accuracy: 0.975
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.40      0.48        20
           1       0.98      1.00      0.99      1938
           2       0.69      0.92      0.79        12
           3       0.00      0.00      0.00        18
           4       0.00      0.00      0.00         3
           5       0.00      0.00      0.00         9

    accuracy                           0.97      2000
   macro avg       0.38      0.39      0.38      2000
weighted avg       0.96      0.97      0.97      2000

Test Accuracy: 0.974
Test Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.33      0.42        15
           1       0.98      1.00      0.99      1935
           2       0.67      0.92      0.77        13
           3       0.00      0.00      0.00        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.97      2000
   macro avg       0.37      0.38      0.36      2000
weighted avg       0.95      0.97      0.96      2000

Confusion Matrix:
 [[   5   10    0    0    0    0]
 [   2 1930    3    0    0    0]
 [   0    1   12    0    0    0]
 [   2   16    2    0    0    0]
 [   0    6    0    0    0    0]
 [   0   10    1    0    0    0]]
```

```
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Saving Machine State

In [44]:
```python
import pickle

# Save the scaler
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

# Save the model
with open('svc_model.pkl', 'wb') as f:
    pickle.dump(svc, f)
```

From Random Forest

In [45]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Train the Random Forest model on the training set
rf = RandomForestClassifier(n_estimators=100, random_state=42, max_depth=None)  # Y
rf.fit(X_train, y_train)

# Evaluate the model on the validation set
y_val_pred = rf.predict(X_val)
```

```
val_accuracy = accuracy_score(y_val, y_val_pred)

print("Validation Accuracy: %.3f" % val_accuracy)
print("Validation Classification Report:\n", classification_report(y_val, y_val_pre

# If satisfied with validation performance, evaluate the model on the test set
y_test_pred = rf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Test Accuracy: %.3f" % test_accuracy)
print("Test Classification Report:\n", classification_report(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
```

```
Validation Accuracy: 0.983
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.70      0.78        20
           1       0.99      1.00      0.99      1938
           2       0.75      0.50      0.60        12
           3       0.81      0.72      0.76        18
           4       0.00      0.00      0.00         3
           5       0.00      0.00      0.00         9

    accuracy                           0.98      2000
   macro avg       0.57      0.49      0.52      2000
weighted avg       0.98      0.98      0.98      2000

Test Accuracy: 0.979
Test Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.60      0.69        15
           1       0.98      1.00      0.99      1935
           2       0.86      0.46      0.60        13
           3       0.71      0.75      0.73        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.56      0.47      0.50      2000
weighted avg       0.97      0.98      0.97      2000

Confusion Matrix:
 [[   9    6    0    0    0    0]
 [   1 1928    0    6    0    0]
 [   1    6    6    0    0    0]
 [   0    5    0   15    0    0]
 [   0    6    0    0    0    0]
 [   0   10    1    0    0    0]]
```

```
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetr
icWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```