assignment-2

September 2, 2024

[ ]:
```python
# Name: Arju Srivastava
# Section : 2CA
# Roll No: 12
```

[3]:
```python
# Load the dataset using Pandas.
import pandas as pd
file = "C:\\Users\\windows\\Downloads\\sensor_data.csv"
df = pd.read_csv(file)
# Display the first few rows of the dataset to understand its structure.
print(df.head())
```

```
             Timestamp  Temperature       Pressure   Humidity  Vibration
0  2024-01-01 00:00:00    27.483571            NaN        NaN   0.585502
1  2024-01-01 01:00:00    24.308678  101114.677339  55.607845   0.719909
2  2024-01-01 02:00:00    28.238443  101153.642742        NaN   1.373647
3  2024-01-01 03:00:00    32.615149  100923.861365        NaN   1.305185
4  2024-01-01 04:00:00    23.829233            NaN  36.223306  10.000000
```

[7]:
```python
# Identify and handle missing values:
# Replace missing values using the mean or median.

df['Temperature'].fillna(df['Temperature'].mean(), inplace=True)
df['Pressure'].fillna(df['Pressure'].mean(), inplace=True)
df['Humidity'].fillna(df['Humidity'].mean(), inplace=True)
df['Vibration'].fillna(df['Vibration'].mean(), inplace=True)
df
```

```
C:\Users\windows\AppData\Local\Temp\ipykernel_4664\2308645533.py:4:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  df['Temperature'].fillna(df['Temperature'].mean(), inplace=True)
C:\Users\windows\AppData\Local\Temp\ipykernel_4664\2308645533.py:5:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Pressure'].fillna(df['Pressure'].mean(), inplace=True)
C:\Users\windows\AppData\Local\Temp\ipykernel_4664\2308645533.py:6:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Humidity'].fillna(df['Humidity'].mean(), inplace=True)
C:\Users\windows\AppData\Local\Temp\ipykernel_4664\2308645533.py:7:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


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

[7]:

	Timestamp	Temperature	Pressure	Humidity	Vibration
0	2024-01-01 00:00:00	27.483571	101889.818616	49.961066	0.585502
1	2024-01-01 01:00:00	24.308678	101114.677339	55.607845	0.719909
2	2024-01-01 02:00:00	28.238443	101153.642742	49.961066	1.373647
3	2024-01-01 03:00:00	32.615149	100923.861365	49.961066	1.305185

```
4   2024-01-01 04:00:00    23.829233  101889.818616  36.223306  10.000000
..                  …          …             …          …          …
95  2024-01-04 23:00:00    17.682425  101517.658690  43.070904   0.765412
96  2024-01-05 00:00:00    26.480601  100883.071282  58.995999   0.143433
97  2024-01-05 01:00:00    26.305276  101401.862553  53.072995   1.676936
98  2024-01-05 02:00:00    25.334767  101354.104359  58.128621   0.942730
99  2024-01-05 03:00:00    23.827064  100753.514851  56.296288   1.618908

[100 rows x 5 columns]
```

[9]:

```python
# Identify and remove or cap outliers in the dataset using NumPy.

import numpy as np

def cap_outliers(series, lower_quantile=0.05, upper_quantile=0.95):
    lower_bound = series.quantile(lower_quantile)
    upper_bound = series.quantile(upper_quantile)
    return np.clip(series, lower_bound, upper_bound)

df['Temperature'] = cap_outliers(df['Temperature'])
df['Pressure'] = cap_outliers(df['Pressure'])
df['Humidity'] = cap_outliers(df['Humidity'])
df['Vibration'] = cap_outliers(df['Vibration'])
```

[11]:

```python
# Convert the Timestamp column to a proper datetime format.
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df
```

[11]:

```
             Timestamp  Temperature       Pressure   Humidity  Vibration
0  2024-01-01 00:00:00    27.483571  101889.818616  49.961066   0.585502
1  2024-01-01 01:00:00    24.308678  101114.677339  55.607845   0.719909
2  2024-01-01 02:00:00    28.238443  101153.642742  49.961066   1.373647
3  2024-01-01 03:00:00    32.390034  100923.861365  49.961066   1.305185
4  2024-01-01 04:00:00    23.829233  101889.818616  36.223306   1.915785
..                 …          …             …          …          …
95 2024-01-04 23:00:00    17.682425  101517.658690  43.070904   0.765412
96 2024-01-05 00:00:00    26.480601  100883.071282  58.995999   0.496599
97 2024-01-05 01:00:00    26.305276  101401.862553  53.072995   1.676936
98 2024-01-05 02:00:00    25.334767  101354.104359  58.128621   0.942730
99 2024-01-05 03:00:00    23.827064  100753.514851  56.296288   1.618908

[100 rows x 5 columns]
```

[14]:

```python
# Normalize the Temperature, Pressure, and Vibration columns to a range between
# 0 and 1 using Min-Max scaling.
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()

df[['Temperature', 'Pressure', 'Vibration']] = scaler.
 ↪fit_transform(df[['Temperature', 'Pressure', 'Vibration']])
df
```

[14]:

```
              Timestamp  Temperature  Pressure   Humidity  Vibration
0   2024-01-01 00:00:00     0.693635  0.764077  49.961066   0.062644
1   2024-01-01 01:00:00     0.495391  0.280652  55.607845   0.157351
2   2024-01-01 02:00:00     0.740770  0.304953  49.961066   0.617994
3   2024-01-01 03:00:00     1.000000  0.161647  49.961066   0.569753
4   2024-01-01 04:00:00     0.465454  0.764077  36.223306   1.000000
..                  …           …         …          …        …
95  2024-01-04 23:00:00     0.081641  0.531975  43.070904   0.189414
96  2024-01-05 00:00:00     0.631009  0.136208  58.995999   0.000000
97  2024-01-05 01:00:00     0.620061  0.459758  53.072995   0.831700
98  2024-01-05 02:00:00     0.559462  0.429973  58.128621   0.314357
99  2024-01-05 03:00:00     0.465319  0.055409  56.296288   0.790812

[100 rows x 5 columns]
```

[19]:

```
# Create a new column that calculates the moving average of Temperature over a␣
 ↪window of 10 readings.
df['Temperature_Moving_Avg'] = df['Temperature'].rolling(window=10).mean()
df.tail()
```

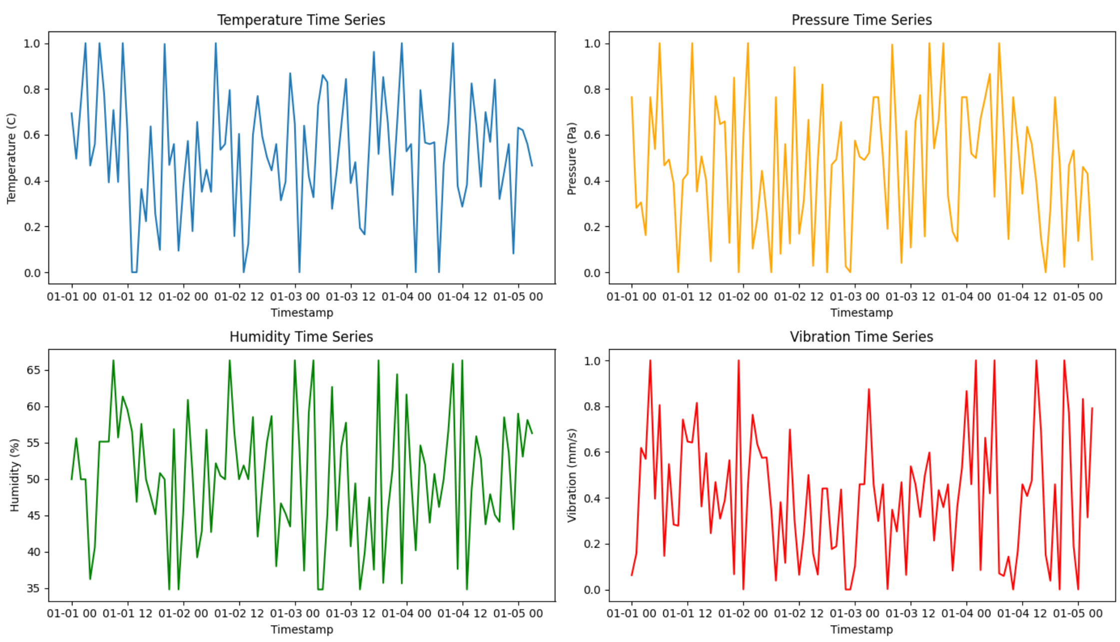
[19]:

```
              Timestamp  Temperature  Pressure   Humidity  Vibration  \
95  2024-01-04 23:00:00     0.081641  0.531975  43.070904   0.189414
96  2024-01-05 00:00:00     0.631009  0.136208  58.995999   0.000000
97  2024-01-05 01:00:00     0.620061  0.459758  53.072995   0.831700
98  2024-01-05 02:00:00     0.559462  0.429973  58.128621   0.314357
99  2024-01-05 03:00:00     0.465319  0.055409  56.296288   0.790812

    Temperature_Moving_Avg
95                0.534409
96                0.515075
97                0.512962
98                0.531591
99                0.508243
```

[16]:

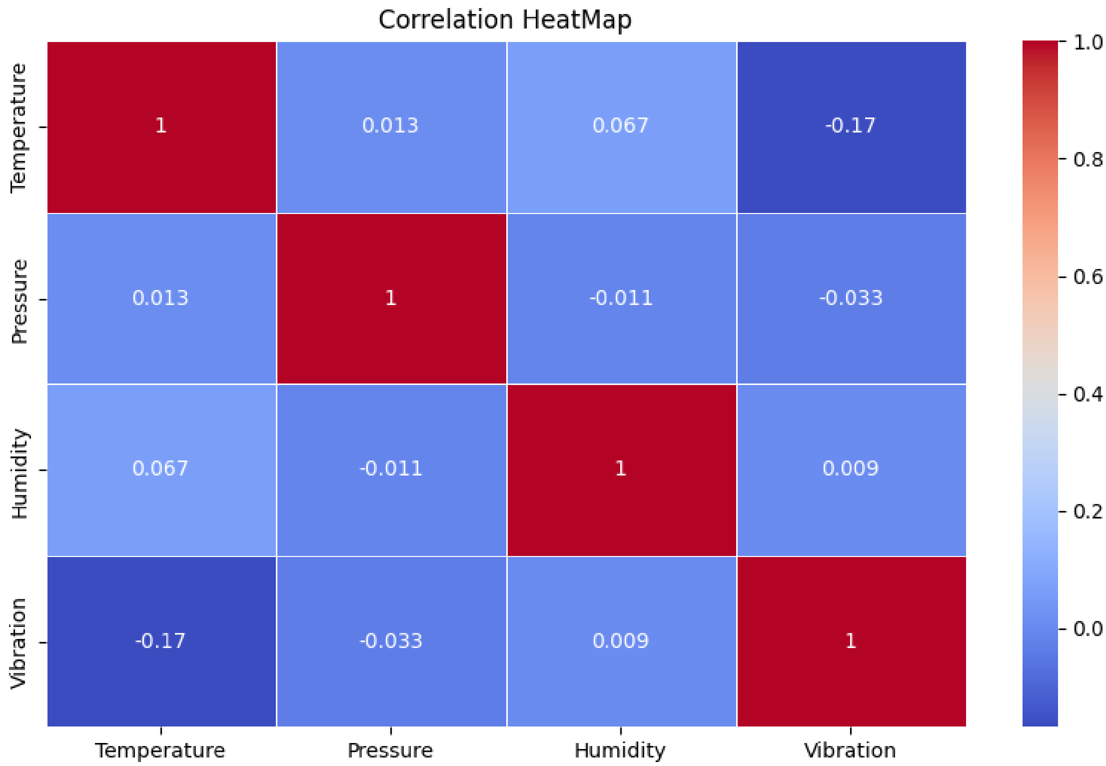
```
# Plot the time series data for Temperature, Pressure, Humidity, and Vibration.
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 8))

plt.subplot(2, 2, 1)
plt.plot(df['Timestamp'], df['Temperature'], label='Temperature')
```

```python
plt.xlabel('Timestamp')
plt.ylabel('Temperature (C)')
plt.title('Temperature Time Series')

plt.subplot(2, 2, 2)
plt.plot(df['Timestamp'], df['Pressure'], label='Pressure', color='orange')
plt.xlabel('Timestamp')
plt.ylabel('Pressure (Pa)')
plt.title('Pressure Time Series')

plt.subplot(2, 2, 3)
plt.plot(df['Timestamp'], df['Humidity'], label='Humidity', color='green')
plt.xlabel('Timestamp')
plt.ylabel('Humidity (%)')
plt.title('Humidity Time Series')

plt.subplot(2, 2, 4)
plt.plot(df['Timestamp'], df['Vibration'], label='Vibration', color='red')
plt.xlabel('Timestamp')
plt.ylabel('Vibration (mm/s)')
plt.title('Vibration Time Series')

plt.tight_layout()
plt.show()
```



[21]:
```
# Create histograms for each sensor to visualize the distribution of the␣
↪readings.
df[['Temperature','Pressure','Humidity','Vibration']].hist(bins=30,␣
↪figsize=(14,8))
```

[20]:
```
# Plot a heatmap to visualize the correlation between different sensor readings.
import seaborn as sns

plt.figure(figsize=(10, 6))
correlation_matrix = df[['Temperature', 'Pressure', 'Humidity', 'Vibration']].
↪corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation HeatMap")
plt.show()
```



[ ]: