```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
d=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/mydata.csv")
pd.set_option('display.max_columns', None)
# to display all columns in the dataset
#pd.set_option('display.max_rows', None)
```

```
df=pd.DataFrame(d)
df
```

	Wind speed (m/s)	Wind speed, Standard deviation (m/s)	Wind speed, Minimum (m/s)	Wind speed, Maximum (m/s)	Long Term Wind (m/s)	Wind speed Sensor 1 (m/s)	Wind speed Sensor 1, Standard deviation (m/s)	Wind speed Sensor 1, Minimum (m/s)	Wind speed Sensor 1, Maximum (m/s)	Wind speed Sensor 2 (m/s)	Wind speed Sensor 2, Standard deviation (m/s)	Wind speed Sensor 2, Minimum (m/s)	
0	13.627191	1.407840	10.214900	16.075251	8.52	12.856080	1.007911	10.032020	14.631343	13.245921	1.371917	10.070001	1
1	13.495296	1.638197	10.133000	16.789400	8.52	12.936592	1.046722	10.186995	14.668793	13.265481	1.632925	9.932301	1
2	12.995932	1.472556	9.692451	15.579350	8.52	12.495875	1.017442	10.482636	14.219700	12.867713	1.196771	10.691900	1
3	12.508251	1.560829	9.031401	15.031251	8.52	12.312967	1.327846	9.358532	14.674883	12.518090	1.580093	8.487801	1
4	14.027196	1.530687	9.407600	16.705252	8.52	13.032389	1.049432	10.570018	14.883749	13.703166	1.256293	10.945251	1
...	...	...	...	...	...	...	...	...	...	...	...	...	
52555	10.503650	1.137052	8.504901	13.113800	8.31	10.761560	0.768969	9.411510	12.672993	10.479260	0.870995	8.754650	1
52556	10.074575	1.284305	7.686350	12.722301	8.31	10.238298	1.068604	7.735098	12.113985	10.177595	1.320105	6.632000	1
52557	10.264610	1.314398	7.501400	13.512501	8.31	10.062030	0.839748	7.843185	11.745881	10.029965	1.092780	7.652150	1
52558	9.741830	0.933406	8.120601	12.014901	8.31	10.122782	0.995698	8.190890	12.856588	10.110140	1.156416	7.751600	1
52559	10.683350	1.261210	8.372601	14.900300	8.31	11.135307	0.964449	9.069591	13.320295	10.731425	0.933305	8.365850	1

52560 rows × 138 columns

```
df=df.drop(['Unnamed: 53'], axis=1)
```

```
df.shape
```

```
(52560, 137)
```

```
df=df.apply(lambda x: x.fillna(x.mean()))
```

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

```
Wind speed (m/s)                          0
Wind speed, Standard deviation (m/s)      0
Wind speed, Minimum (m/s)                 0
Wind speed, Maximum (m/s)                 0
Long Term Wind (m/s)                      0
                                         ..
Temperature motor axis 2, Min (°C)        0
Temperature motor axis 2, StdDev (°C)     0
Temperature motor axis 3, Max (°C)        0
Temperature motor axis 3, Min (°C)        0
```

```
Temperature motor axis 3, StdDev (°C)    0
Length: 137, dtype: int64
```

```
temp=df[["Front bearing temperature (°C)"]]
```

```
temp
```

	Front bearing temperature (°C)
0	67.534999
1	70.333334
2	73.975864
3	69.183871
4	67.589656
...	...
52555	71.971668
52556	69.255001
52557	73.725001
52558	71.519999
52559	69.688334

52560 rows × 1 columns

```
x=temp.values.tolist()
```

```
x
```

```
       [67.95166601],
       [67.40500005],
       [72.67333298],
       [68.00166626],
       [67.19500046],
       [72.49999949],
       ...]
```

```
t = [j for sub in x for j in sub]
t
```

```
      66.65666656,
      65.94666672,
      65.31833344,
      65.95833308,
      67.48666636,
      68.83333333,
      69.90666606,
      71.04666697,
      72.53333333,
      73.22999929,
      67.68499934,
      69.03833516,
      69.48166809,
      68.71500066,
      67.9216657,
      67.53333333,
      68.14655041,
      69.65999985,
      69.75666733,
      69.72833379,
      71.27586154,
      71.46166662,
      65.37333374,
      67.04166718,
      67.97333374,
      69.00000025,
      68.96000163,
      70.03666662,
      71.01166662,
      69.82931124,
      69.74833298,
      71.00833359,
      70.42666626,
      70.90000051,
      67.38166682,
      66.16666667,
      70.99499995,
      72.97833354,
      72.08275841,
      67.21999969,
      66.81666768,
      67.69333394,
      68.72333298,
      68.61166687,
      68.2666659,
      68.17833277,
      68.70500005,
      71.04333344,
      68.38000031,
      68.33333435,
      72.81333338,
      67.95166601,
      67.40500005,
      72.67333298,
      68.00166626,
      67.19500046,
      72.49999949,
      ...]
```

```
w_size=30 # so window length is 30 and no of windows is 1752
j=0
i=0
l=0
M=0 # for collecting mean value
S=0 # for collecting standard deviation
mean=[]# collecting mean values of 1000 points so in total 5000 means
stand=[]
y=[] # for collecting those points for mean

for i in range(0,1752,1):
    M=np.mean(t[j:w_size])
    S=np.std(t[j:w_size])
    mean.append(M)
    stand.append(S)
    M=0
    S=0
```

```
    j+=30
    w_size+=30
```

```
print("length of mean ",len(mean))
```

```
length of mean  1752
```

```
alpha=3
w_size=30
k=0
j=0
i=0
label=[] # list for holding labels
for i in range(0,1752,1):
    for j in range(w_size):
        if j < len(mean) and j < len(stand):
            ut = mean[i] + alpha * stand[i]
            lt = mean[i] - alpha * stand[i]
            if lt < t[j] < ut:
                label.append(0)

            else:
                label.append(1)
```

```
label
```

```
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 ...]
```

```
len(label)
```

```
52560
```

```
df['class labels']=label
df
```

	Wind speed (m/s)	Wind speed, Standard deviation (m/s)	Wind speed, Minimum (m/s)	Wind speed, Maximum (m/s)	Long Term Wind (m/s)	Wind speed Sensor 1 (m/s)	Wind speed Sensor 1, Standard deviation (m/s)	Wind speed Sensor 1, Minimum (m/s)	Wind speed Sensor 1, Maximum (m/s)	Wind speed Sensor 2 (m/s)	Wind speed Sensor 2, Standard deviation (m/s)	Wind speed Sensor 2, Minimum (m/s)	
0	13.627191	1.407840	10.214900	16.075251	8.52	12.856080	1.007911	10.032020	14.631343	13.245921	1.371917	10.070001	1
1	13.495296	1.638197	10.133000	16.789400	8.52	12.936592	1.046722	10.186995	14.668793	13.265481	1.632925	9.932301	1
2	12.995932	1.472556	9.692451	15.579350	8.52	12.495875	1.017442	10.482636	14.219700	12.867713	1.196771	10.691900	1
3	12.508251	1.560829	9.031401	15.031251	8.52	12.312967	1.327846	9.358532	14.674883	12.518090	1.580093	8.487801	1
4	14.027196	1.530687	9.407600	16.705252	8.52	13.032389	1.049432	10.570018	14.883749	13.703166	1.256293	10.945251	1
...	...	...	...	...	...	...	...	...	...	...	...	...	
52555	10.503650	1.137052	8.504901	13.113800	8.31	10.761560	0.768969	9.411510	12.672993	10.479260	0.870995	8.754650	1
52556	10.074575	1.284305	7.686350	12.722301	8.31	10.238298	1.068604	7.735098	12.113985	10.177595	1.320105	6.632000	1
52557	10.264610	1.314398	7.501400	13.512501	8.31	10.062030	0.839748	7.843185	11.745881	10.029965	1.092780	7.652150	1
52558	9.741830	0.933406	8.120601	12.014901	8.31	10.122782	0.995698	8.190890	12.856588	10.110140	1.156416	7.751600	1
52559	10.683350	1.261210	8.372601	14.900300	8.31	11.135307	0.964449	9.069591	13.320295	10.731425	0.933305	8.365850	1

52560 rows × 138 columns

```
X=df[['Wind speed (m/s)','Power (kW)','Front bearing temperature (°C)','Rear bearing temperature (°C)','Nacelle temperature (°C)','Gear (
```

```
X
```

	Wind speed (m/s)	Power (kW)	Front bearing temperature (°C)	Rear bearing temperature (°C)	Nacelle temperature (°C)	Gear oil inlet temperature (°C)	Gear oi temperature (°C
0	13.627191	2037.502873	67.534999	60.026667	9.090000	32.038333	55.50000
1	13.495296	2010.745451	70.333334	61.261667	9.563333	49.581667	57.31500
2	12.995932	1974.779594	73.975864	63.718966	9.668965	45.624137	59.18793
3	12.508251	1958.147563	69.183871	61.275807	9.111667	32.253333	56.36129
4	14.027196	2016.503349	67.589656	59.813793	9.146667	36.763333	55.52068
...	...	...	...	...	...	...	.
52555	10.503650	1555.897095	71.971668	62.066667	14.275000	31.608333	56.86833
52556	10.074575	1348.674803	69.255001	60.156667	14.053333	36.563334	55.52500
52557	10.264610	1414.735852	73.725001	62.945000	14.336667	52.400000	58.52666
52558	9.741830	1373.458518	71.519999	61.678333	13.936666	30.828333	56.36833
52559	10.683350	1685.769108	69.688334	60.406666	13.898333	39.446667	55.80333

52560 rows × 9 columns

```
Y=df[['class labels']]
Y
```

	class labels
0	0
1	0
2	0
3	0
4	0
...	...
52555	0
52556	0
52557	0

```
import numpy as np
from sklearn.model_selection import train_test_split

# Assuming you have a dataset X (features) and y (labels/targets)

# Step 1: Split the data into training (60%), validation (20%), and testing (20%) sets
X_train, X_temp, y_train, y_temp = train_test_split(X, Y, test_size=0.2, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Step 2: Check the shapes of the resulting sets
print("Training set shapes:", X_train.shape, y_train.shape)
print("Validation set shapes:", X_valid.shape, y_valid.shape)
print("Testing set shapes:", X_test.shape, y_test.shape)
```

```
Training set shapes: (42048, 9) (42048, 1)
Validation set shapes: (5256, 9) (5256, 1)
Testing set shapes: (5256, 9) (5256, 1)
```

```
y_train = y_train.values.ravel()
y_valid = y_valid.values.ravel()
y_test = y_test.values.ravel()
```

```
import tensorflow as tf
from sklearn.metrics import confusion_matrix


# Step 2: Define and compile a neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 3: Train the model on the training data
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_valid, y_valid))

# Step 4: Make predictions on the test set
y_test_pred = model.predict(X_test)
y_test_pred = (y_test_pred > 0.5).astype(int)  # Convert probabilities to binary predictions

# Step 5: Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)

# Step 6: Display the confusion matrix
print("Confusion Matrix (Test Set):\n", conf_matrix)

# Step 7: Optionally, plot training and validation loss and accuracy over epochs
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')

plt.show()

plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
165/165 [==============================] - 0s 2ms/step - loss: 0.3460 - accuracy: 0.8248
```

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
Epoch 8/10
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)
report_test = classification_report(y_test, y_test_pred)
# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print(report_test )
```

```
Precision: 0.6893089190785587
Recall: 0.7485567671584349
F1 Score: 0.7177121771217712
              precision    recall  f1-score   support

           0       0.89      0.86      0.87      3697
           1       0.69      0.75      0.72      1559

    accuracy                           0.83      5256
   macro avg       0.79      0.80      0.80      5256
weighted avg       0.83      0.83      0.83      5256
```