In [1]:
```python
import pandas as pd

# Load the dataset into a Pandas DataFrame
df = pd.read_table("HT_Sensor_dataset.dat")

df.head()
```

Out[1]:

	id time R1 R2 R3 R4 R5 R6 R7 R8 Temp. Humidity
0	0 -0.999750 12.862100 10.368300 10.438300 ... NaN
1	0 -0.999472 12.861700 10.368200 10.437500 ... NaN
2	0 -0.999194 12.860700 10.368600 10.437000 ... NaN
3	0 -0.998916 12.860200 10.368600 10.437000 ... NaN
4	0 -0.998627 12.859500 10.368800 10.437400 ... NaN

In [2]:
```python
df['id time']
```

Out[2]:
```
0          0  -0.999750  12.862100  10.368300  10.438300 ...
1          0  -0.999472  12.861700  10.368200  10.437500 ...
2          0  -0.999194  12.860700  10.368600  10.437000 ...
3          0  -0.998916  12.860200  10.368600  10.437000 ...
4          0  -0.998627  12.859500  10.368800  10.437400 ...
                                    ...
928986    99  1.675182  12.622400  10.580500  10.743200 ...
928987    99  1.675460  12.623600  10.579600  10.743600 ...
928988    99  1.675738  12.624400  10.579500  10.743700 ...
928989    99  1.676016  12.624300  10.579700  10.744000 ...
928990    99  1.676304  12.624800  10.579100  10.744000 ...
Name: id time, Length: 928991, dtype: object
```

In [3]:
```python
x = []
for i in df['id time']:
    x.append(i.split())
```

In [4]:
```python
df2 = pd.DataFrame(x,columns=['id','time', 'R1' ,'R2','R3','R4','R5','R6','R7','R8','Tem
df2.head()
```

Out[4]:

	id	time	R1	R2	R3	R4	R5	R6	R7	R8	Temp
0	0	-0.999750	12.862100	10.368300	10.438300	11.669900	13.493100	13.342300	8.041690	8.739010	26.225700
1	0	-0.999472	12.861700	10.368200	10.437500	11.669700	13.492700	13.341200	8.041330	8.739080	26.230800
2	0	-0.999194	12.860700	10.368600	10.437000	11.669600	13.492400	13.340500	8.041010	8.739150	26.236500
3	0	-0.998916	12.860200	10.368600	10.437000	11.669700	13.492100	13.339800	8.040860	8.739360	26.241600
4	0	-0.998627	12.859500	10.368800	10.437400	11.669900	13.491900	13.339000	8.040870	8.739860	26.246200

In [5]:
```python
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 928991 entries, 0 to 928990
Data columns (total 12 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   id        928991 non-null  object
 1   time      928991 non-null  object
 2   R1        928991 non-null  object
 3   R2        928991 non-null  object
```

```
 4   R3         928991 non-null  object
 5   R4         928991 non-null  object
 6   R5         928991 non-null  object
 7   R6         928991 non-null  object
 8   R7         928991 non-null  object
 9   R8         928991 non-null  object
 10  Temp       928991 non-null  object
 11  Humidity   928991 non-null  object
dtypes: object(12)
memory usage: 85.1+ MB
```

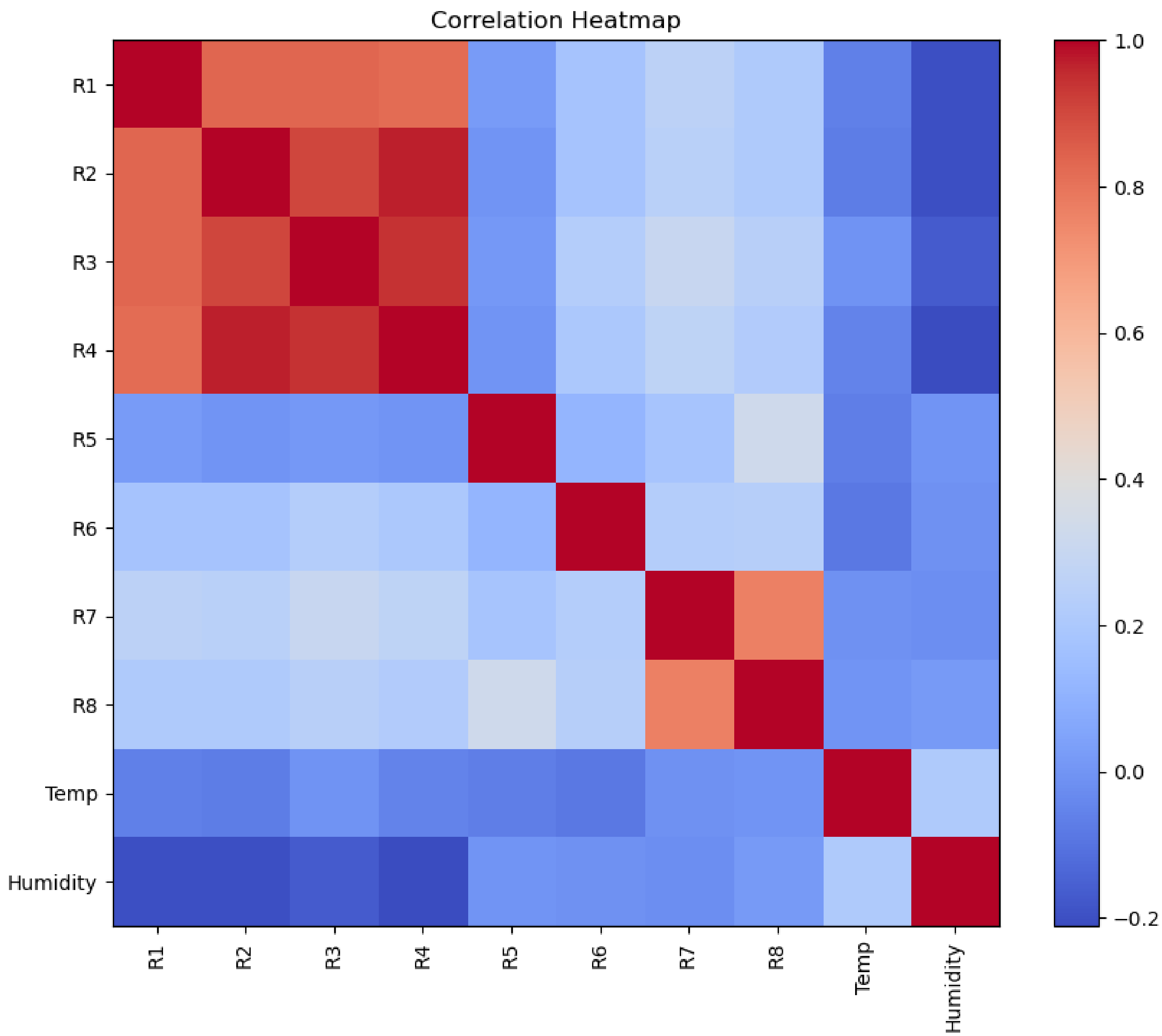
In [6]:
```
df2['id'] = df2['id'].astype(int)
df2['time'] = df2['time'].astype(float)
df2['R1'] = df2['R1'].astype(float)
df2['R2'] = df2['R2'].astype(float)
df2['R3'] = df2['R3'].astype(float)
df2['R4'] = df2['R4'].astype(float)
df2['R5'] = df2['R5'].astype(float)
df2['R6'] = df2['R6'].astype(float)
df2['R7'] = df2['R7'].astype(float)
df2['R8'] = df2['R8'].astype(float)
df2['Temp'] = df2['Temp'].astype(float)
df2['Humidity'] = df2['Humidity'].astype(float)
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 928991 entries, 0 to 928990
Data columns (total 12 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   id         928991 non-null  int32  
 1   time       928991 non-null  float64
 2   R1         928991 non-null  float64
 3   R2         928991 non-null  float64
 4   R3         928991 non-null  float64
 5   R4         928991 non-null  float64
 6   R5         928991 non-null  float64
 7   R6         928991 non-null  float64
 8   R7         928991 non-null  float64
 9   R8         928991 non-null  float64
 10  Temp       928991 non-null  float64
 11  Humidity   928991 non-null  float64
dtypes: float64(11), int32(1)
memory usage: 81.5 MB
```

In [7]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Correlation matrix of numerical columns
correlation_matrix = df2[['R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'Temp', 'Humid
print(correlation_matrix)

# Heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns, rotation=90)
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.title('Correlation Heatmap')
plt.show()
```

```
          R1        R2        R3        R4        R5        R6  \
R1  1.000000  0.829782  0.830017  0.815452  0.023163  0.179294   
R2  0.829782  1.000000  0.904101  0.970727  0.000083  0.179965   
R3  0.830017  0.904101  1.000000  0.938719  0.011904  0.230779   
R4  0.815452  0.970727  0.938719  1.000000 -0.001982  0.201893   
```

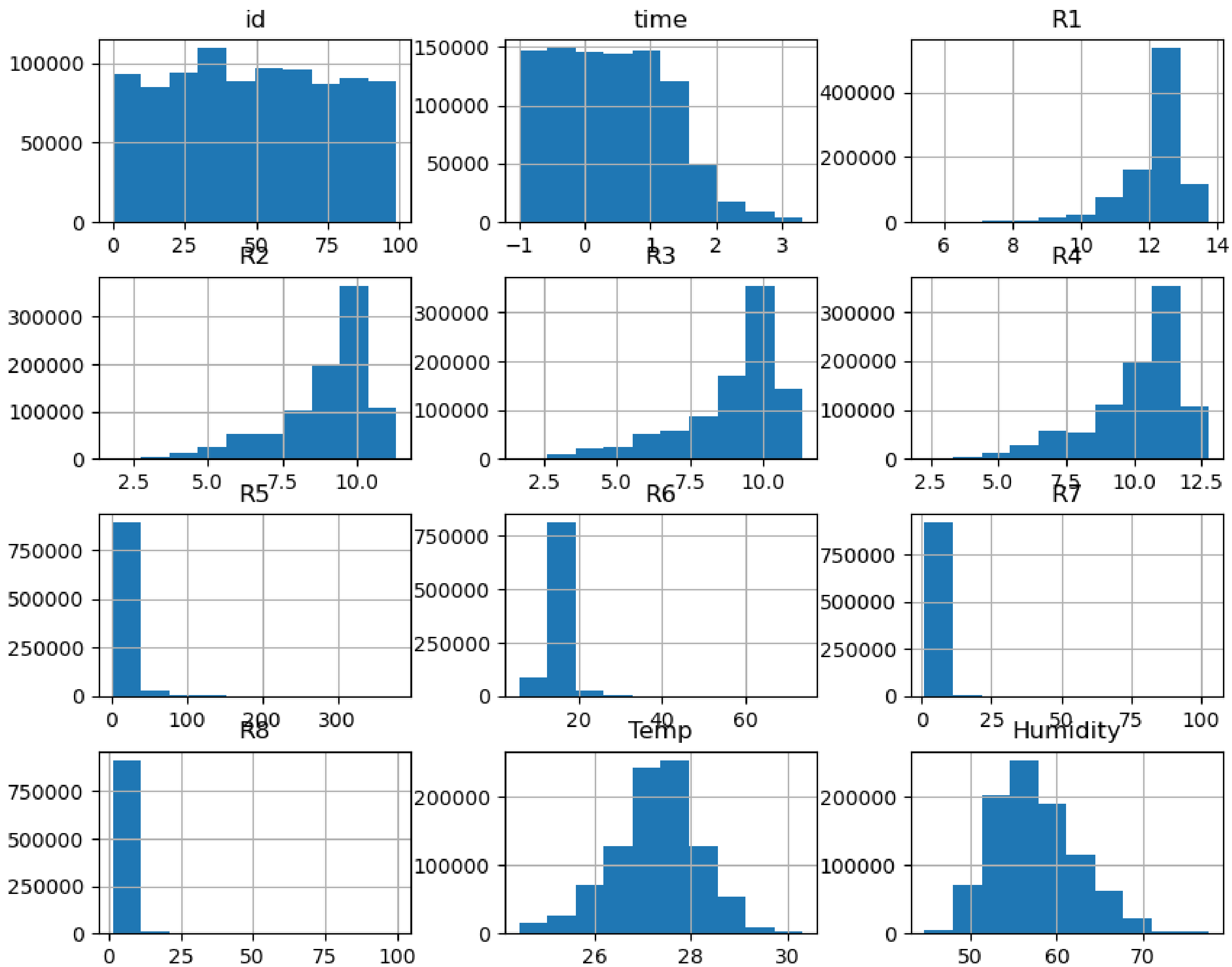
```
R5        0.023163  0.000083  0.011904 -0.001982  1.000000  0.117357
R6        0.179294  0.179965  0.230779  0.201893  0.117357  1.000000
R7        0.261193  0.251888  0.300779  0.269640  0.184570  0.232124
R8        0.210106  0.212323  0.247322  0.222034  0.329771  0.237968
Temp     -0.061570 -0.071798 -0.005071 -0.054750 -0.067408 -0.085537
Humidity -0.197800 -0.197358 -0.164143 -0.211453  0.001222 -0.009581

                R7        R8      Temp  Humidity
R1        0.261193  0.210106 -0.061570 -0.197800
R2        0.251888  0.212323 -0.071798 -0.197358
R3        0.300779  0.247322 -0.005071 -0.164143
R4        0.269640  0.222034 -0.054750 -0.211453
R5        0.184570  0.329771 -0.067408  0.001222
R6        0.232124  0.237968 -0.085537 -0.009581
R7        1.000000  0.763631 -0.009134 -0.020658
R8        0.763631  1.000000  0.000779  0.016815
Temp     -0.009134  0.000779  1.000000  0.213209
Humidity -0.020658  0.016815  0.213209  1.000000
```



In [8]:

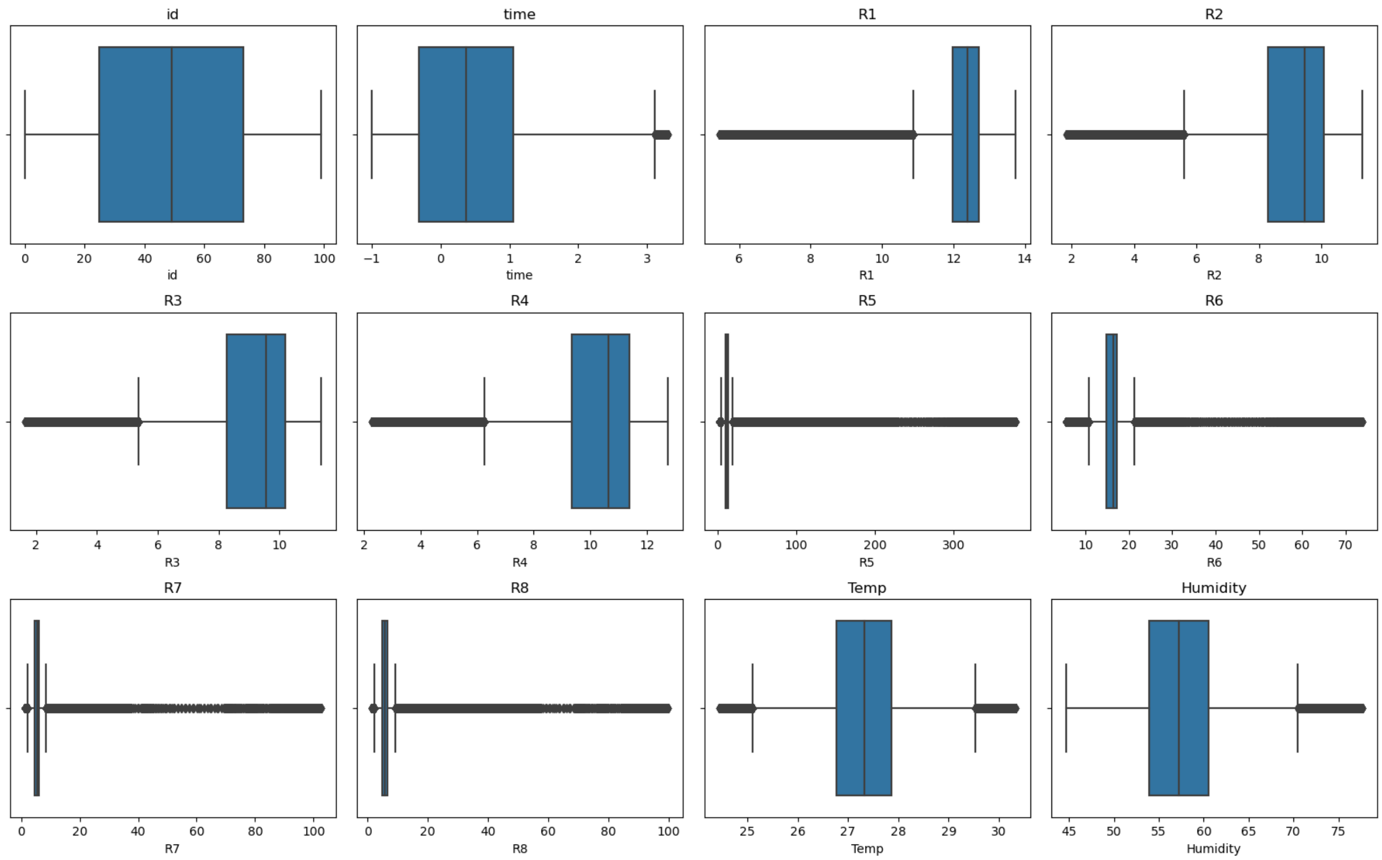
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Explore the distribution of numerical variables using histograms
```

```
df2.hist(figsize=(10, 8))
plt.show()
```



In [31]:

```
# Identify outliers in numerical variables using box plots or violin plots
# Generate box plots for all columns
# Set the figure size
plt.figure(figsize=(16, 10))

# Iterate over each column in df2
for i, column in enumerate(df2.columns):
    # Create subplots for each column
    plt.subplot(3, 4, i+1)

    # Generate the boxplot for the column
    sns.boxplot(x=df2[column])

    # Set the title of the subplot
    plt.title(column)

# Adjust the layout
plt.tight_layout()

# Display the plot
plt.show()
```



In [15]:
```python
from tsfresh.feature_extraction import extract_features
from tsfresh.feature_extraction import settings

settings_minimal = settings.MinimalFCParameters()
settings_minimal
X_tsfresh = extract_features(df2, column_id="id", default_fc_parameters=settings_minimal
```

```
Feature Extraction: 100%|██████████| 30/30 [00:06<00:00,  4.33it/s]
```

In [16]:
```python
Meta_data = pd.read_table("HT_Sensor_metadata.dat")
Meta_data.head()
```

Out[16]:

	id	date	Unnamed: 2	class	t0	dt
0	0	07-04-15	banana	13.49	1.64	NaN
1	1	07-05-15	wine	19.61	0.54	NaN
2	2	07-06-15	wine	19.99	0.66	NaN
3	3	07-09-15	banana	6.49	0.72	NaN
4	4	07-09-15	wine	20.07	0.53	NaN

In [17]:
```python
Meta_data.columns
```

Out[17]:
```
Index(['id', 'date', 'Unnamed: 2', 'class', 't0', 'dt'], dtype='object')
```

In [18]:
```python
categories = []
for filename in Meta_data["Unnamed: 2"]:
    if filename == 'banana':
        categories.append(1)
    elif filename == 'wine':
         categories.append(2)

    else:
        categories.append(0)
```

In [19]:
```
categories = pd.DataFrame(categories)
categories.columns = ["Target"]
Meta_data = Meta_data.drop(["id","date","dt","Unnamed: 2"],axis=1)
data = pd.concat([X_tsfresh,Meta_data,categories],axis=1)
data.head()
```

Out[19]:

	time__sum_values	time__median	time__mean	time__length	time__standard_deviation	time__variance	time__roc
0	10717.195523	0.837083	0.836301	12815.0	1.038550	1.078586	
1	2233.847747	0.248874	0.250600	8914.0	0.724202	0.524469	
2	3135.339255	0.330208	0.330105	9498.0	0.768101	0.589979	
3	3125.952756	0.341032	0.335799	9309.0	0.784538	0.615500	
4	4139.218981	0.774904	0.767090	5396.0	0.446518	0.199378	

5 rows × 113 columns

In [20]:
```
data = data.dropna()
```

In [21]:
```
from sklearn.model_selection import train_test_split
X = data.drop('Target',1)
y = data.Target
```

```
C:\Users\vinay\AppData\Local\Temp\ipykernel_15772\3195628193.py:2: FutureWarning: In a f
uture version of pandas all arguments of DataFrame.drop except for the argument 'labels'
will be keyword-only.
  X = data.drop('Target',1)
```

In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=3
```

In [23]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
classifier =RandomForestClassifier(n_estimators=100)
```

In [24]:
```
classifier.fit(X_train, y_train)
```

Out[24]:
```
RandomForestClassifier()
```

In [25]:
```
y_pred = classifier.predict(X_test)
```

In [26]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.8787878787878788
```

In [27]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[14  0  0]
 [ 0  4  4]
 [ 0  0 11]]
```

In [28]:
```
predicted_df=pd.DataFrame(y_pred*100,columns=['RandomForest_Predicted_value'])
predicted_df=pd.concat([df2,predicted_df],axis=1)
```

In [29]:
```
predicted_df.to_csv('Predictions_data.csv',index=False)
```

In [33]:
```
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
```

```python
from keras.optimizers import Adam
from keras.utils import Sequence

# Define the DataGenerator class
class DataGenerator(Sequence):
    def __init__(self, data, target, batch_size=32, shuffle=True):
        self.data = data
        self.target = target
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        return len(self.data) // self.batch_size

    def __getitem__(self, index):
        indexes = self.indexes[index * self.batch_size:(index + 1) * self.batch_size]
        X = self.data[indexes]
        y = self.target[indexes]
        return X, y

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.data))
        if self.shuffle:
            np.random.shuffle(self.indexes)

# Define the LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(128, input_shape=(10, 1)))
model_lstm.add(Dense(1))

# Compile the model
model_lstm.compile(loss='mse', optimizer=Adam())

# Generate example data
X_train = np.random.rand(1000, 10, 1)
y_train = np.random.rand(1000, 1)
X_val = np.random.rand(200, 10, 1)
y_val = np.random.rand(200, 1)

# Define the batch size
batch_size = 32

# Calculate the steps per epoch and validation steps
train_steps = len(X_train) // batch_size
val_steps = len(X_val) // batch_size

# Create the data generators
train_generator = DataGenerator(X_train, y_train, batch_size=batch_size)
val_generator = DataGenerator(X_val, y_val, batch_size=batch_size)

# Train the LSTM model using the data generator
model_lstm.fit(train_generator, epochs=10, steps_per_epoch=train_steps,
               validation_data=val_generator, validation_steps=val_steps, verbose=1)
```

```
Epoch 1/10
31/31 [==============================] - 4s 42ms/step - loss: 0.1184 - val_loss: 0.0838
Epoch 2/10
31/31 [==============================] - 0s 12ms/step - loss: 0.0885 - val_loss: 0.0825
Epoch 3/10
31/31 [==============================] - 0s 15ms/step - loss: 0.0871 - val_loss: 0.0809
Epoch 4/10
31/31 [==============================] - 1s 20ms/step - loss: 0.0861 - val_loss: 0.0825
Epoch 5/10
31/31 [==============================] - 1s 22ms/step - loss: 0.0862 - val_loss: 0.0886
Epoch 6/10
```

```
31/31 [==============================] - 1s 22ms/step - loss: 0.0876 - val_loss: 0.0812
Epoch 7/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0850 - val_loss: 0.0812
Epoch 8/10
31/31 [==============================] - 1s 20ms/step - loss: 0.0850 - val_loss: 0.0810
Epoch 9/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0837 - val_loss: 0.0789
Epoch 10/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0845 - val_loss: 0.0836
```

Out[33]:

```
<keras.callbacks.History at 0x20746fd8a60>
```

In [34]:

```
# Train the LSTM model using the data generator
history = model_lstm.fit(train_generator, epochs=10, steps_per_epoch=train_steps,
                         validation_data=val_generator, validation_steps=val_steps, verb

# Calculate the MSE loss on the validation set
val_loss = model_lstm.evaluate(val_generator, steps=val_steps)
print("Validation MSE Loss:", val_loss)
```

```
Epoch 1/10
31/31 [==============================] - 1s 22ms/step - loss: 0.0845 - val_loss: 0.0786
Epoch 2/10
31/31 [==============================] - 1s 20ms/step - loss: 0.0834 - val_loss: 0.0850
Epoch 3/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0835 - val_loss: 0.0792
Epoch 4/10
31/31 [==============================] - 1s 22ms/step - loss: 0.0844 - val_loss: 0.0796
Epoch 5/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0838 - val_loss: 0.0794
Epoch 6/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0830 - val_loss: 0.0799
Epoch 7/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0858 - val_loss: 0.0795
Epoch 8/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0829 - val_loss: 0.0821
Epoch 9/10
31/31 [==============================] - 1s 20ms/step - loss: 0.0830 - val_loss: 0.0802
Epoch 10/10
31/31 [==============================] - 1s 21ms/step - loss: 0.0832 - val_loss: 0.0819
6/6 [==============================] - 0s 10ms/step - loss: 0.0810
Validation MSE Loss: 0.08099149167537689
```

In [35]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Generate predictions on the validation set
y_pred = model_lstm.predict(val_generator, steps=val_steps)

# Trim the predictions to match the number of samples in y_val
y_pred = y_pred[:y_val.shape[0]]

# Calculate evaluation metrics
mse = mean_squared_error(y_val[:y_pred.shape[0]], y_pred)
mae = mean_absolute_error(y_val[:y_pred.shape[0]], y_pred)
r2 = r2_score(y_val[:y_pred.shape[0]], y_pred)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2):", r2)
```

```
6/6 [==============================] - 1s 8ms/step
Mean Squared Error (MSE): 0.07947535102153672
Mean Absolute Error (MAE): 0.24379008555863757
R-squared (R2): -0.0175377301521209
```

In [36]:

```
print(y_pred)
```

```
[[0.48013404]
 [0.4989518 ]
 [0.48109484]
 [0.48133805]
 [0.49007168]
 [0.49347866]
 [0.47061417]
 [0.49358183]
 [0.473361  ]
 [0.49662113]
 [0.48918846]
 [0.47718498]
 [0.48452854]
 [0.48006788]
 [0.4772067 ]
 [0.48033985]
 [0.49179146]
 [0.4838914 ]
 [0.48658326]
 [0.4875547 ]
 [0.4916631 ]
 [0.47706583]
 [0.49481574]
 [0.48379296]
 [0.48521593]
 [0.47807845]
 [0.4747829 ]
 [0.48635665]
 [0.47701195]
 [0.503272  ]
 [0.47208512]
 [0.49737903]
 [0.47621092]
 [0.47044957]
 [0.49358785]
 [0.48701987]
 [0.49937394]
 [0.4787477 ]
 [0.49545604]
 [0.4901449 ]
 [0.48962763]
 [0.4745546 ]
 [0.4838346 ]
 [0.487299  ]
 [0.47670105]
 [0.47202614]
 [0.4833661 ]
 [0.47236517]
 [0.49145418]
 [0.47438243]
 [0.49300864]
 [0.5030656 ]
 [0.48662704]
 [0.49667087]
 [0.48318174]
 [0.4615281 ]
 [0.4904661 ]
 [0.47171393]
 [0.48528633]
 [0.4954711 ]
 [0.47517604]
 [0.48304006]
 [0.45959732]
 [0.49135658]
 [0.5001926 ]
 [0.48409674]
```

```
[0.4787447 ]
[0.47149187]
[0.49745038]
[0.49143428]
[0.48048076]
[0.48846412]
[0.47916266]
[0.48970824]
[0.4738265 ]
[0.48879588]
[0.497323  ]
[0.48738483]
[0.47641975]
[0.4626567 ]
[0.48852316]
[0.46422184]
[0.48442724]
[0.47641113]
[0.48493943]
[0.47261015]
[0.47864467]
[0.49042216]
[0.49322203]
[0.48449472]
[0.48121527]
[0.4958615 ]
[0.47368062]
[0.48309106]
[0.47562215]
[0.47339413]
[0.47682998]
[0.49724343]
[0.47203055]
[0.4937745 ]
[0.4843877 ]
[0.49169925]
[0.48100945]
[0.4959895 ]
[0.4864793 ]
[0.47873315]
[0.4829289 ]
[0.48144504]
[0.49022883]
[0.4969212 ]
[0.47612676]
[0.48334026]
[0.48509726]
[0.48367542]
[0.48798135]
[0.47904256]
[0.4916648 ]
[0.48615974]
[0.4853431 ]
[0.47557732]
[0.4846228 ]
[0.47158876]
[0.47811815]
[0.4877626 ]
[0.48644653]
[0.48416206]
[0.47961956]
[0.5033321 ]
[0.49229458]
[0.48490414]
[0.4855862 ]
[0.47280195]
```

```
 [0.49430504]
 [0.48051223]
 [0.49692053]
 [0.48446402]
 [0.498577  ]
 [0.49879032]
 [0.4940765 ]
 [0.47651008]
 [0.4791855 ]
 [0.46294624]
 [0.49922794]
 [0.48461783]
 [0.48174074]
 [0.48508206]
 [0.4863552 ]
 [0.4905496 ]
 [0.47651628]
 [0.48244175]
 [0.4810529 ]
 [0.48380187]
 [0.4926874 ]
 [0.5004214 ]
 [0.4824202 ]
 [0.48506442]
 [0.48775572]
 [0.46866176]
 [0.48565042]
 [0.48431426]
 [0.46475068]
 [0.48805937]
 [0.47221246]
 [0.49140283]
 [0.49908444]
 [0.46597084]
 [0.4931399 ]
 [0.48310724]
 [0.4810311 ]
 [0.48046458]
 [0.4734306 ]
 [0.48957548]
 [0.49030393]
 [0.4723889 ]
 [0.4895257 ]
 [0.49057347]
 [0.4927932 ]
 [0.47011986]
 [0.4704981 ]
 [0.4756073 ]
 [0.49841204]
 [0.49034086]
 [0.4690941 ]
 [0.49670056]
 [0.47714588]
 [0.48301628]
 [0.48434535]
 [0.49886665]
 [0.4681844 ]
 [0.48870885]
 [0.4855081 ]
 [0.48154995]]
```

In [37]:
```
predicted_df2=pd.DataFrame(y_pred*100,columns=['LSTM_Predicted_value'])
predicted_df2=pd.concat([predicted_df,predicted_df2],axis=1)
```

In [38]:
```
predicted_df.to_csv('Predictions_data.csv',index=False)
```

In [ ]:

```

```

In [ ]:

```

```