IoT_SmartParking_LSTM

February 25, 2025

0.0.1 AAI-530 IoT Based Smart Parking - Occupancy Status Prediction using LSTM

[3]:
```python
# import necessary python libraries
import pandas as pd
import numpy as np
```

[4]:
```python
# Load the original real-time Smart Parking Management dataset
df = pd.read_csv("IIoT_Smart_Parking_Management.csv")
```

[5]:
```python
df
```

[5]:
```
                         Timestamp  Parking_Spot_ID  Sensor_Reading_Proximity  \
0    2021-01-01 00:00:00.000000000               20                  1.023651
1    2021-01-02 06:39:16.756756756               49                  3.903349
2    2021-01-03 13:18:33.513513513               38                 10.315709
3    2021-01-04 19:57:50.270270270               31                  6.588039
4    2021-01-06 02:37:07.027027027                8                  8.213969
..                             …               …                       …
995  2024-06-24 21:22:52.972972960                5                  5.349471
996  2024-06-26 04:02:09.729729728               47                 15.688164
997  2024-06-27 10:41:26.486486480                7                  0.357255
998  2024-06-28 17:20:43.243243232               49                  0.293735
999  2024-06-30 00:00:00.000000000               23                  1.657731

     Sensor_Reading_Pressure  Vehicle_Type_Weight  Vehicle_Type_Height  \
0                   1.541461          1831.770127             4.392528
1                   1.621719          1330.815754             4.595638
2                   6.292374          1255.134827             4.313721
3                   1.659870          1523.442919             3.567329
4                   3.278467          1758.490837             5.145836
..                       …                   …                   …
995                10.515457          1267.050258             4.442869
996                 2.661805          1547.138376             4.413585
997                 1.411642          1552.856947             4.380228
998                12.630766          1299.945385             4.091230
999                 7.449078          1559.375000             6.684525

      User_Type  Weather_Temperature  Weather_Precipitation  \
```

```
0       Visitor           18.092553                       1
1    Registered           13.397533                       0
2    Registered           21.687410                       0
3       Visitor           18.683461                       0
4       Visitor           19.214876                       0
..          …                 …                    …
995     Visitor           19.430937                       0
996     Visitor           25.426111                       0
997  Registered           20.192776                       0
998  Registered           17.581707                       0
999  Registered           18.766378                       0

    Nearby_Traffic_Level  …  Occupancy_Status  Vehicle_Type  \
0                    Low  …          Occupied           Car
1                    Low  …          Occupied           Car
2                   High  …            Vacant           Car
3                 Medium  …            Vacant    Motorcycle
4                   High  …          Occupied           Car
..                   …  …               …            …
995                  Low  …            Vacant           Car
996                  Low  …            Vacant           Car
997                  Low  …            Vacant           Car
998               Medium  …          Occupied           Car
999               Medium  …          Occupied           Car

     Parking_Violation  Sensor_Reading_Ultrasonic  Parking_Duration  \
0                    0                 102.951052                 4
1                    0                  87.559131                 3
2                    1                 100.061854                 5
3                    1                 110.594598                 2
4                    0                  84.786963                 2
..                 …                        …                 …
995                  0                 105.332652                 2
996                  1                 124.841337                 2
997                  0                  93.011015                 1
998                  1                  89.972326                 2
999                  0                  97.877279                 2

     Environmental_Noise_Level Dynamic_Pricing_Factor  Spot_Size  \
0                    55.620740                    0.8   Standard
1                    56.682386                    1.2    Compact
2                    59.239322                    0.8   Standard
3                    44.545155                    0.8   Standard
4                    48.012604                    0.8   Standard
..                         …                    …        …
995                  69.507857                    0.8  Oversized
996                  49.958346                    1.5   Standard
```

```
997                60.676107                  1.0   Standard
998                56.465611                  1.2  Oversized
999                44.105778                  1.2   Standard

    Proximity_To_Exit User_Parking_History
0            6.610474             6.660310
1            8.678719             6.766187
2           13.795262            -0.910052
3            1.678721            10.415888
4           20.012252             4.355544
..                ...                  ...
995          3.686763             1.749779
996         11.989485             2.569270
997          4.265255            11.013160
998          5.713190             4.561407
999          2.691136             8.600266

[1000 rows x 28 columns]
```

0.0.2 LSTM for Parking Occupancy Status Prediction on Original Dataset (Without Feature Engineering)

[6]:

```
# import necessary python libraries
import pandas as pd
import numpy as np

# import sklearn for preprocessing
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split

# import tensorflow for using with LSTM model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# import matplotlib for visualization
import matplotlib.pyplot as plt

# Load the original real-time IoT Smart Parking dataset
df = pd.read_csv("IIoT_Smart_Parking_Management.csv")

# Convert the timestamp column to datetime format
df['Timestamp'] = pd.to_datetime(df['Timestamp'])

# Encode categorical columns with one-hot encoding for better performance with␣
↪LSTM
```

```python
df = pd.get_dummies(df, columns=['User_Type', 'Nearby_Traffic_Level',␣
↪'Parking_Lot_Section',
                                 'Payment_Status', 'Vehicle_Type', 'Spot_Size'])

# Encode the 'Occupancy_Status' column into numerical format
df['Occupancy_Status'] = df['Occupancy_Status'].apply(lambda x: 1 if x ==␣
↪'Occupied' else 0)

# Compute the correlation matrix, including 'Occupancy_Status'
correlation_matrix = df.corr()

# Find correlation with the target ('Occupancy_Status') separately
correlation_with_occupancy = correlation_matrix['Occupancy_Status'].abs().
↪sort_values(ascending=False)

# Check the correlation values of all features with 'Occupancy_Status'
print("Correlation with Occupancy_Status: \n", correlation_with_occupancy)

# Select features with correlation > 0.04 (you can adjust this threshold)
highly_correlated_features =␣
↪correlation_with_occupancy[correlation_with_occupancy > 0.04].index.tolist()

# If 'Occupancy_Status' itself is in the list, remove it
highly_correlated_features.remove('Occupancy_Status')

# Check what features were selected
print("Highly Correlated Features: ", highly_correlated_features)
```

```
Correlation with Occupancy_Status:
 Occupancy_Status                 1.000000
Occupancy_Rate                   0.063943
Parking_Lot_Section_Zone C       0.062415
Environmental_Noise_Level        0.060882
Parking_Spot_ID                  0.051377
Vehicle_Type_Electric Vehicle    0.050817
Nearby_Traffic_Level_High        0.043693
User_Parking_History             0.039441
Sensor_Reading_Proximity         0.037237
User_Type_Visitor                0.036741
Sensor_Reading_Ultrasonic        0.036265
User_Type_Staff                  0.035445
Spot_Size_Compact                0.033674
Parking_Lot_Section_Zone A       0.032682
Dynamic_Pricing_Factor           0.032043
Vehicle_Type_Car                 0.029943
Timestamp                        0.027086
Parking_Duration                 0.026814
```

```
Exit_Time                        0.024956
Nearby_Traffic_Level_Medium      0.023401
Parking_Lot_Section_Zone B       0.022443
Vehicle_Type_Weight              0.021822
Vehicle_Type_Height              0.020971
User_Type_Registered             0.020876
Proximity_To_Exit                0.020494
Spot_Size_Standard               0.020419
Weather_Temperature              0.019355
Weather_Precipitation            0.015424
Sensor_Reading_Pressure          0.013092
Entry_Time                       0.012984
Spot_Size_Oversized              0.012030
Payment_Amount                   0.009261
Payment_Status_Overdue           0.009067
Reserved_Status                  0.007459
Nearby_Traffic_Level_Low         0.004916
Vehicle_Type_Motorcycle          0.004858
Payment_Status_Unpaid            0.004506
Parking_Lot_Section_Zone D       0.004427
Parking_Violation                0.003669
Payment_Status_Paid              0.002173
Electric_Vehicle                 0.000504
Name: Occupancy_Status, dtype: float64
Highly Correlated Features:  ['Occupancy_Rate', 'Parking_Lot_Section_Zone C',
'Environmental_Noise_Level', 'Parking_Spot_ID', 'Vehicle_Type_Electric Vehicle',
'Nearby_Traffic_Level_High']
```

0.0.3 The above output shows the correlation of other fields w.r.t Occupancy Status.

0.0.4 The top highly correlated features could be considered as features to be used for prediction

[7]:

```python
# If there are no highly correlated features left, issue a warning
if len(highly_correlated_features) == 0:
    print("Warning: No features have high enough correlation. Consider lowering␣
 ↪the correlation threshold.")
else:
    # Subset the dataframe with the selected highly correlated features
    df = df[highly_correlated_features + ['Occupancy_Status']]  # Add␣
 ↪'Occupancy_Status' back for target

    # Now, prepare features (X) and target (y)
    X = df.drop('Occupancy_Status', axis=1).values
    y = df['Occupancy_Status'].values

    # Verify if the feature set (X) has any valid columns
    print(f"Shape of X before scaling: {X.shape}")
```

```
    # Proceed with scaling if X has features
    if X.shape[1] > 0:
        # Use MinMaxScaler for normalization
        scaler = MinMaxScaler()
        X_scaled = scaler.fit_transform(X)

        # Prepare the data for time series (sequence data for LSTM)
        sequence_length = 5  # Predict for the next 5 days

        # Function to create sequences
        def create_sequences(X, y, seq_length):
            X_seq, y_seq = [], []
            for i in range(len(X) - seq_length):
                X_seq.append(X[i:i + seq_length])
                y_seq.append(y[i + seq_length])
            return np.array(X_seq), np.array(y_seq)

        X_seq, y_seq = create_sequences(X_scaled, y, sequence_length)

        # Split data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq, test_size=0.2, shuffle=True)

        # Build and train the LSTM model
        from tensorflow.keras.layers import Input

        # Build and train the LSTM model
        model = Sequential()

        # Add multiple LSTM layers with different numbers of units
        model.add(Input(shape=(X_train.shape[1], X_train.shape[2])))  # Input shape for time-series data

        model.add(LSTM(units=128, return_sequences=True))
        model.add(Dropout(0.3))  # Dropout layer to prevent overfitting

        model.add(LSTM(units=128, return_sequences=True))
        model.add(Dropout(0.3))  # Dropout layer to prevent overfitting

        model.add(LSTM(units=64, return_sequences=True))
        model.add(Dropout(0.3))  # Dropout layer to prevent overfitting

        model.add(LSTM(units=64, return_sequences=False))  # The final LSTM layer
        # Add Dropout layer to prevent overfitting
        model.add(Dropout(0.3))
```

```
        model.add(Dense(units=1, activation='sigmoid'))

        # Compile the model
        model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
        history = model.fit(X_train, y_train, epochs=100, batch_size=16,␣
↪validation_data=(X_test, y_test))  # Increased epochs

        # Evaluate the model
        loss, accuracy = model.evaluate(X_test, y_test)
        print(f'Accuracy: {accuracy * 100:.2f}%')
    else:
        print("Error: The feature set has no valid features after correlation␣
↪filtering.")
```

```
Shape of X before scaling: (1000, 6)
Epoch 1/100
50/50              15s 51ms/step -
accuracy: 0.5086 - loss: 0.6926 - val_accuracy: 0.5980 - val_loss: 0.6879
Epoch 2/100
50/50              1s 25ms/step -
accuracy: 0.5473 - loss: 0.6907 - val_accuracy: 0.5980 - val_loss: 0.6832
Epoch 3/100
50/50              1s 27ms/step -
accuracy: 0.5605 - loss: 0.6876 - val_accuracy: 0.5980 - val_loss: 0.6851
Epoch 4/100
50/50              1s 26ms/step -
accuracy: 0.5407 - loss: 0.6912 - val_accuracy: 0.5980 - val_loss: 0.6842
Epoch 5/100
50/50              1s 26ms/step -
accuracy: 0.5548 - loss: 0.6888 - val_accuracy: 0.5980 - val_loss: 0.6830
Epoch 6/100
50/50              1s 26ms/step -
accuracy: 0.5498 - loss: 0.6890 - val_accuracy: 0.5980 - val_loss: 0.6852
Epoch 7/100
50/50              1s 26ms/step -
accuracy: 0.5334 - loss: 0.6911 - val_accuracy: 0.5980 - val_loss: 0.6840
Epoch 8/100
50/50              1s 26ms/step -
accuracy: 0.5370 - loss: 0.6899 - val_accuracy: 0.5980 - val_loss: 0.6805
Epoch 9/100
50/50              1s 26ms/step -
accuracy: 0.5244 - loss: 0.6938 - val_accuracy: 0.5930 - val_loss: 0.6840
Epoch 10/100
50/50              1s 25ms/step -
accuracy: 0.5406 - loss: 0.6899 - val_accuracy: 0.5779 - val_loss: 0.6878
```

```
Epoch 11/100
50/50            1s 25ms/step -
accuracy: 0.5099 - loss: 0.6937 - val_accuracy: 0.5980 - val_loss: 0.6805
Epoch 12/100
50/50            1s 26ms/step -
accuracy: 0.5106 - loss: 0.6954 - val_accuracy: 0.5980 - val_loss: 0.6806
Epoch 13/100
50/50            1s 27ms/step -
accuracy: 0.5141 - loss: 0.6943 - val_accuracy: 0.5829 - val_loss: 0.6846
Epoch 14/100
50/50            1s 25ms/step -
accuracy: 0.5144 - loss: 0.6937 - val_accuracy: 0.5980 - val_loss: 0.6824
Epoch 15/100
50/50            1s 26ms/step -
accuracy: 0.5442 - loss: 0.6889 - val_accuracy: 0.5879 - val_loss: 0.6861
Epoch 16/100
50/50            1s 26ms/step -
accuracy: 0.5502 - loss: 0.6889 - val_accuracy: 0.5528 - val_loss: 0.6872
Epoch 17/100
50/50            1s 26ms/step -
accuracy: 0.5463 - loss: 0.6883 - val_accuracy: 0.5075 - val_loss: 0.6889
Epoch 18/100
50/50            1s 25ms/step -
accuracy: 0.5264 - loss: 0.6913 - val_accuracy: 0.5327 - val_loss: 0.6875
Epoch 19/100
50/50            1s 26ms/step -
accuracy: 0.5345 - loss: 0.6901 - val_accuracy: 0.5729 - val_loss: 0.6838
Epoch 20/100
50/50            1s 25ms/step -
accuracy: 0.5585 - loss: 0.6857 - val_accuracy: 0.5678 - val_loss: 0.6846
Epoch 21/100
50/50            1s 25ms/step -
accuracy: 0.5479 - loss: 0.6864 - val_accuracy: 0.5930 - val_loss: 0.6838
Epoch 22/100
50/50            1s 27ms/step -
accuracy: 0.5357 - loss: 0.6891 - val_accuracy: 0.5628 - val_loss: 0.6876
Epoch 23/100
50/50            1s 26ms/step -
accuracy: 0.5424 - loss: 0.6891 - val_accuracy: 0.5930 - val_loss: 0.6857
Epoch 24/100
50/50            1s 26ms/step -
accuracy: 0.5610 - loss: 0.6885 - val_accuracy: 0.5829 - val_loss: 0.6851
Epoch 25/100
50/50            1s 26ms/step -
accuracy: 0.5600 - loss: 0.6862 - val_accuracy: 0.5226 - val_loss: 0.6864
Epoch 26/100
50/50            1s 25ms/step -
accuracy: 0.5281 - loss: 0.6912 - val_accuracy: 0.5276 - val_loss: 0.6871
```

```
Epoch 27/100
50/50           1s 23ms/step -
accuracy: 0.4940 - loss: 0.6928 - val_accuracy: 0.5930 - val_loss: 0.6831
Epoch 28/100
50/50           1s 26ms/step -
accuracy: 0.5284 - loss: 0.6910 - val_accuracy: 0.5427 - val_loss: 0.6861
Epoch 29/100
50/50           1s 25ms/step -
accuracy: 0.5133 - loss: 0.6919 - val_accuracy: 0.5678 - val_loss: 0.6841
Epoch 30/100
50/50           1s 26ms/step -
accuracy: 0.5538 - loss: 0.6861 - val_accuracy: 0.5980 - val_loss: 0.6803
Epoch 31/100
50/50           1s 26ms/step -
accuracy: 0.5338 - loss: 0.6900 - val_accuracy: 0.5779 - val_loss: 0.6840
Epoch 32/100
50/50           1s 26ms/step -
accuracy: 0.5345 - loss: 0.6901 - val_accuracy: 0.5477 - val_loss: 0.6837
Epoch 33/100
50/50           1s 26ms/step -
accuracy: 0.5560 - loss: 0.6839 - val_accuracy: 0.5025 - val_loss: 0.6902
Epoch 34/100
50/50           1s 26ms/step -
accuracy: 0.5143 - loss: 0.6902 - val_accuracy: 0.5477 - val_loss: 0.6872
Epoch 35/100
50/50           1s 25ms/step -
accuracy: 0.5470 - loss: 0.6864 - val_accuracy: 0.5980 - val_loss: 0.6809
Epoch 36/100
50/50           1s 26ms/step -
accuracy: 0.5499 - loss: 0.6859 - val_accuracy: 0.5678 - val_loss: 0.6839
Epoch 37/100
50/50           1s 24ms/step -
accuracy: 0.5354 - loss: 0.6839 - val_accuracy: 0.5528 - val_loss: 0.6916
Epoch 38/100
50/50           1s 25ms/step -
accuracy: 0.5474 - loss: 0.6853 - val_accuracy: 0.5578 - val_loss: 0.6857
Epoch 39/100
50/50           1s 25ms/step -
accuracy: 0.5686 - loss: 0.6817 - val_accuracy: 0.5980 - val_loss: 0.6780
Epoch 40/100
50/50           1s 25ms/step -
accuracy: 0.5527 - loss: 0.6870 - val_accuracy: 0.5427 - val_loss: 0.6939
Epoch 41/100
50/50           1s 26ms/step -
accuracy: 0.5438 - loss: 0.6830 - val_accuracy: 0.5427 - val_loss: 0.6841
Epoch 42/100
50/50           1s 26ms/step -
accuracy: 0.5599 - loss: 0.6780 - val_accuracy: 0.5779 - val_loss: 0.6884
```

```
Epoch 43/100
50/50            1s 26ms/step -
accuracy: 0.5654 - loss: 0.6762 - val_accuracy: 0.6080 - val_loss: 0.6714
Epoch 44/100
50/50            1s 26ms/step -
accuracy: 0.5673 - loss: 0.6738 - val_accuracy: 0.5829 - val_loss: 0.6873
Epoch 45/100
50/50            1s 26ms/step -
accuracy: 0.5893 - loss: 0.6747 - val_accuracy: 0.5678 - val_loss: 0.6995
Epoch 46/100
50/50            1s 26ms/step -
accuracy: 0.6153 - loss: 0.6687 - val_accuracy: 0.5678 - val_loss: 0.6814
Epoch 47/100
50/50            1s 25ms/step -
accuracy: 0.5842 - loss: 0.6643 - val_accuracy: 0.6181 - val_loss: 0.6746
Epoch 48/100
50/50            1s 25ms/step -
accuracy: 0.5736 - loss: 0.6659 - val_accuracy: 0.5678 - val_loss: 0.6885
Epoch 49/100
50/50            1s 25ms/step -
accuracy: 0.5541 - loss: 0.6723 - val_accuracy: 0.5729 - val_loss: 0.6969
Epoch 50/100
50/50            1s 25ms/step -
accuracy: 0.5831 - loss: 0.6522 - val_accuracy: 0.5779 - val_loss: 0.6742
Epoch 51/100
50/50            1s 25ms/step -
accuracy: 0.5949 - loss: 0.6572 - val_accuracy: 0.6080 - val_loss: 0.6814
Epoch 52/100
50/50            1s 25ms/step -
accuracy: 0.5870 - loss: 0.6518 - val_accuracy: 0.5377 - val_loss: 0.6853
Epoch 53/100
50/50            1s 25ms/step -
accuracy: 0.5690 - loss: 0.6726 - val_accuracy: 0.5879 - val_loss: 0.6901
Epoch 54/100
50/50            1s 25ms/step -
accuracy: 0.6053 - loss: 0.6519 - val_accuracy: 0.5779 - val_loss: 0.7015
Epoch 55/100
50/50            1s 25ms/step -
accuracy: 0.5935 - loss: 0.6635 - val_accuracy: 0.5729 - val_loss: 0.7047
Epoch 56/100
50/50            1s 26ms/step -
accuracy: 0.6251 - loss: 0.6263 - val_accuracy: 0.6231 - val_loss: 0.6865
Epoch 57/100
50/50            1s 24ms/step -
accuracy: 0.6318 - loss: 0.6372 - val_accuracy: 0.6131 - val_loss: 0.6848
Epoch 58/100
50/50            1s 25ms/step -
accuracy: 0.6057 - loss: 0.6239 - val_accuracy: 0.6030 - val_loss: 0.6922
```

```
Epoch 59/100
50/50            1s 25ms/step -
accuracy: 0.6262 - loss: 0.6439 - val_accuracy: 0.5327 - val_loss: 0.6864
Epoch 60/100
50/50            1s 24ms/step -
accuracy: 0.6064 - loss: 0.6430 - val_accuracy: 0.5729 - val_loss: 0.7133
Epoch 61/100
50/50            1s 25ms/step -
accuracy: 0.6413 - loss: 0.6211 - val_accuracy: 0.5980 - val_loss: 0.6977
Epoch 62/100
50/50            1s 26ms/step -
accuracy: 0.6072 - loss: 0.6377 - val_accuracy: 0.6030 - val_loss: 0.6997
Epoch 63/100
50/50            1s 25ms/step -
accuracy: 0.6240 - loss: 0.6406 - val_accuracy: 0.5879 - val_loss: 0.7075
Epoch 64/100
50/50            1s 21ms/step -
accuracy: 0.6578 - loss: 0.6218 - val_accuracy: 0.5779 - val_loss: 0.7098
Epoch 65/100
50/50            1s 26ms/step -
accuracy: 0.6592 - loss: 0.6160 - val_accuracy: 0.6030 - val_loss: 0.6939
Epoch 66/100
50/50            1s 25ms/step -
accuracy: 0.6682 - loss: 0.5935 - val_accuracy: 0.6181 - val_loss: 0.6872
Epoch 67/100
50/50            1s 26ms/step -
accuracy: 0.6619 - loss: 0.5967 - val_accuracy: 0.6231 - val_loss: 0.7237
Epoch 68/100
50/50            1s 26ms/step -
accuracy: 0.6497 - loss: 0.6125 - val_accuracy: 0.5930 - val_loss: 0.6968
Epoch 69/100
50/50            1s 24ms/step -
accuracy: 0.6691 - loss: 0.5770 - val_accuracy: 0.6181 - val_loss: 0.6950
Epoch 70/100
50/50            1s 25ms/step -
accuracy: 0.6898 - loss: 0.5792 - val_accuracy: 0.6131 - val_loss: 0.6760
Epoch 71/100
50/50            1s 23ms/step -
accuracy: 0.7190 - loss: 0.5693 - val_accuracy: 0.6080 - val_loss: 0.6745
Epoch 72/100
50/50            1s 23ms/step -
accuracy: 0.6743 - loss: 0.5653 - val_accuracy: 0.5930 - val_loss: 0.7240
Epoch 73/100
50/50            1s 18ms/step -
accuracy: 0.6979 - loss: 0.5523 - val_accuracy: 0.5980 - val_loss: 0.7141
Epoch 74/100
50/50            1s 25ms/step -
accuracy: 0.6679 - loss: 0.5860 - val_accuracy: 0.5879 - val_loss: 0.7887
```

```
Epoch 75/100
50/50            1s 25ms/step -
accuracy: 0.6504 - loss: 0.5732 - val_accuracy: 0.6030 - val_loss: 0.7161
Epoch 76/100
50/50            1s 24ms/step -
accuracy: 0.7135 - loss: 0.5325 - val_accuracy: 0.5980 - val_loss: 0.7677
Epoch 77/100
50/50            1s 24ms/step -
accuracy: 0.7173 - loss: 0.5119 - val_accuracy: 0.6080 - val_loss: 0.7090
Epoch 78/100
50/50            1s 27ms/step -
accuracy: 0.7195 - loss: 0.5262 - val_accuracy: 0.6231 - val_loss: 0.7831
Epoch 79/100
50/50            1s 23ms/step -
accuracy: 0.7076 - loss: 0.5296 - val_accuracy: 0.5829 - val_loss: 0.8576
Epoch 80/100
50/50            1s 23ms/step -
accuracy: 0.7478 - loss: 0.5136 - val_accuracy: 0.5729 - val_loss: 0.7777
Epoch 81/100
50/50            1s 22ms/step -
accuracy: 0.7321 - loss: 0.5065 - val_accuracy: 0.5930 - val_loss: 0.8418
Epoch 82/100
50/50            1s 24ms/step -
accuracy: 0.7161 - loss: 0.5079 - val_accuracy: 0.5930 - val_loss: 0.8268
Epoch 83/100
50/50            1s 24ms/step -
accuracy: 0.7565 - loss: 0.4857 - val_accuracy: 0.5678 - val_loss: 0.9248
Epoch 84/100
50/50            1s 19ms/step -
accuracy: 0.7493 - loss: 0.5000 - val_accuracy: 0.5729 - val_loss: 0.7858
Epoch 85/100
50/50            1s 25ms/step -
accuracy: 0.7400 - loss: 0.4855 - val_accuracy: 0.6030 - val_loss: 0.9039
Epoch 86/100
50/50            1s 24ms/step -
accuracy: 0.7456 - loss: 0.4992 - val_accuracy: 0.6030 - val_loss: 0.8656
Epoch 87/100
50/50            1s 24ms/step -
accuracy: 0.7632 - loss: 0.4582 - val_accuracy: 0.5829 - val_loss: 0.9056
Epoch 88/100
50/50            1s 23ms/step -
accuracy: 0.7737 - loss: 0.4143 - val_accuracy: 0.5980 - val_loss: 0.8556
Epoch 89/100
50/50            1s 24ms/step -
accuracy: 0.7429 - loss: 0.4723 - val_accuracy: 0.5930 - val_loss: 0.9928
Epoch 90/100
50/50            1s 22ms/step -
accuracy: 0.7714 - loss: 0.4347 - val_accuracy: 0.6181 - val_loss: 1.0318
```

```
Epoch 91/100
50/50              1s 24ms/step -
accuracy: 0.7770 - loss: 0.4161 - val_accuracy: 0.5879 - val_loss: 0.9246
Epoch 92/100
50/50              1s 22ms/step -
accuracy: 0.7821 - loss: 0.4275 - val_accuracy: 0.5829 - val_loss: 0.9134
Epoch 93/100
50/50              1s 25ms/step -
accuracy: 0.7907 - loss: 0.4105 - val_accuracy: 0.5779 - val_loss: 1.0431
Epoch 94/100
50/50              1s 20ms/step -
accuracy: 0.8022 - loss: 0.3894 - val_accuracy: 0.5678 - val_loss: 0.9302
Epoch 95/100
50/50              1s 26ms/step -
accuracy: 0.8048 - loss: 0.4010 - val_accuracy: 0.5528 - val_loss: 1.0985
Epoch 96/100
50/50              1s 26ms/step -
accuracy: 0.8118 - loss: 0.3746 - val_accuracy: 0.5377 - val_loss: 1.0374
Epoch 97/100
50/50              1s 24ms/step -
accuracy: 0.8222 - loss: 0.3748 - val_accuracy: 0.5427 - val_loss: 1.1300
Epoch 98/100
50/50              1s 23ms/step -
accuracy: 0.8156 - loss: 0.3824 - val_accuracy: 0.5729 - val_loss: 1.0261
Epoch 99/100
50/50              1s 25ms/step -
accuracy: 0.8037 - loss: 0.3807 - val_accuracy: 0.5729 - val_loss: 1.1109
Epoch 100/100
50/50              1s 24ms/step -
accuracy: 0.8182 - loss: 0.3664 - val_accuracy: 0.5879 - val_loss: 1.0845
7/7              0s 11ms/step -
accuracy: 0.6025 - loss: 1.0245
Accuracy: 58.79%
```

0.0.5 As can be seen above a training accuracy of about 81% is achieved but the validation accuracy is 58.79%

0.0.6 for sequence length 5 (5 Day) prediction

[17]:
```python
# Print model summary
model.summary()
```

```
Model: "sequential_3"


 Layer (type)                           Output Shape                        ␣
↪Param #
```

```
lstm_12 (LSTM)                    (None, 1, 128)                 ␣
↪89,088

dropout_12 (Dropout)              (None, 1, 128)                   ␣
↪  0

lstm_13 (LSTM)                    (None, 64)                     ␣
↪49,408

dropout_13 (Dropout)              (None, 64)                       ␣
↪  0

dense_3 (Dense)                   (None, 1)                        ␣
↪ 65
```

Total params: 415,685 (1.59 MB)

Trainable params: 138,561 (541.25 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 277,124 (1.06 MB)

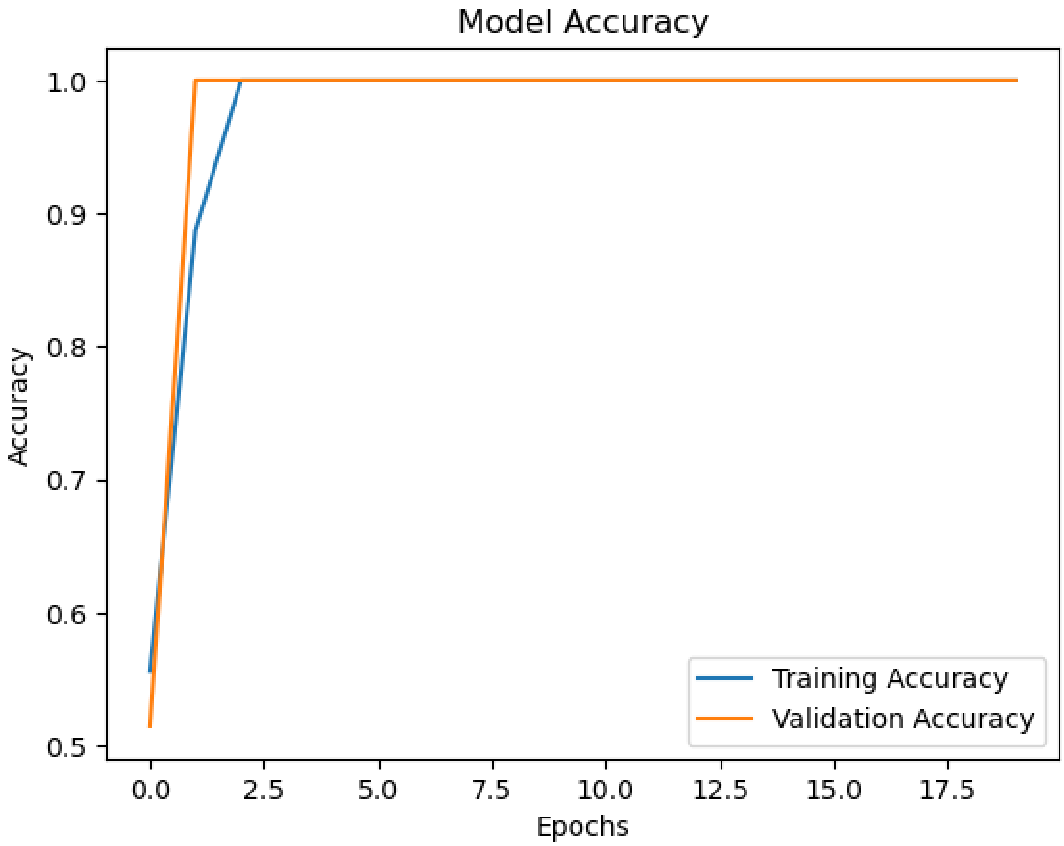
[20]:

```
# Predict the occupancy status for the next 5 days
predictions = model.predict(X_test)

# Convert predictions back to Occupancy_Status
predicted_occupancy = ['Occupied' if pred > 0.5 else 'Vacant' for pred in␣
 ↪predictions]

# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
7/7               0s 4ms/step
```



0.0.7 LSTM for predicting Parking Occupancy Status for 1 sequence step on Feature Engineered Dataset

[24]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler, LabelEncoder # Import␣
 ↪LabelEncoder here
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Load the feature engineering dataset
# Refer to IoT_SmartParking_Feature_Engg.ipynb for the feature engineering
```

```
df = pd.read_csv('IoT_SmartParking_Processed.csv')

# Inspect the data
print(df.head())

# Preprocessing
# Occupancy_Status is the target variable and other columns are features
target_column = 'Occupancy_Status'
features = [col for col in df.columns if col != target_column and col !=␣
↪'Timestamp' and df[col].dtype != object]

X = df[features].values
y = df[target_column].values

# Normalize features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Encode target variable to numeric using LabelEncoder
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Reshape input for LSTM (samples, timesteps, features)
timesteps = 1  # Can be adjusted based on sequence dependency
X_reshaped = X_scaled.reshape((X_scaled.shape[0], timesteps, X_scaled.shape[1]))

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y_encoded,␣
↪test_size=0.2, random_state=42)

from tensorflow.keras.layers import Input
# Build and train the LSTM model
model = Sequential()
model.add(LSTM(units=128, return_sequences=True, input_shape=(X_train.shape[1],␣
↪X_train.shape[2])))  # Increased units
model.add(Dropout(0.2))  # Increased dropout to reduce overfitting
model.add(LSTM(units=64, return_sequences=False))
model.add(Dropout(0.2))  # Increased dropout
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=16,␣
↪validation_data=(X_test, y_test))
```

```python
# Evaluate the model
y_pred = (model.predict(X_test) > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
                      Timestamp  Parking_Spot_ID  Sensor_Reading_Proximity  \
0 2021-01-01 00:00:00.000000000               20                  1.023651
1 2021-01-02 06:39:16.756756756               49                  3.903349
2 2021-01-03 13:18:33.513513513               38                 10.315709
3 2021-01-04 19:57:50.270270270               31                  6.588039
4 2021-01-06 02:37:07.027027027                8                  8.213969

   Sensor_Reading_Pressure  Vehicle_Type_Weight  Vehicle_Type_Height  \
0                 1.541461          1831.770127             4.392528
1                 1.621719          1330.815754             4.595638
2                 6.292374          1255.134827             4.313721
3                 1.659870          1523.442919             3.567329
4                 3.278467          1758.490837             5.145836

    User_Type  Weather_Temperature  Weather_Precipitation  \
0     Visitor            18.092553                      1
1  Registered            13.397533                      0
2  Registered            21.687410                      0
3     Visitor            18.683461                      0
4     Visitor            19.214876                      0

  Nearby_Traffic_Level  …  DayOfWeek_4  DayOfWeek_5  DayOfWeek_6  \
0                  Low  …          1.0          0.0          0.0
1                  Low  …          0.0          1.0          0.0
2                 High  …          0.0          0.0          1.0
3               Medium  …          0.0          0.0          0.0
4                 High  …          0.0          0.0          0.0

   DayOfWeek_0.1  DayOfWeek_1.1  DayOfWeek_2.1 DayOfWeek_3.1 DayOfWeek_4.1  \
0            0.0            0.0            0.0           0.0           1.0
1            0.0            0.0            0.0           0.0           0.0
2            0.0            0.0            0.0           0.0           0.0
3            1.0            0.0            0.0           0.0           0.0
4            0.0            0.0            1.0           0.0           0.0

  DayOfWeek_5.1 DayOfWeek_6.1
0           0.0           0.0
1           1.0           0.0
2           0.0           1.0
3           0.0           0.0
4           0.0           0.0
```

```
[5 rows x 53 columns]

C:\Users\mahesh\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)

Epoch 1/20
50/50            7s 22ms/step -
accuracy: 0.4973 - loss: 0.6840 - val_accuracy: 0.5150 - val_loss: 0.6330
Epoch 2/20
50/50            0s 6ms/step -
accuracy: 0.6727 - loss: 0.5257 - val_accuracy: 1.0000 - val_loss: 0.0717
Epoch 3/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 0.0340 - val_accuracy: 1.0000 - val_loss: 0.0035
Epoch 4/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 0.0033 - val_accuracy: 1.0000 - val_loss: 0.0013
Epoch 5/20
50/50            0s 8ms/step -
accuracy: 1.0000 - loss: 0.0016 - val_accuracy: 1.0000 - val_loss: 8.4834e-04
Epoch 6/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 1.0000 - val_loss: 5.9847e-04
Epoch 7/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 9.4841e-04 - val_accuracy: 1.0000 - val_loss:
4.4341e-04
Epoch 8/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 7.3739e-04 - val_accuracy: 1.0000 - val_loss:
3.7866e-04
Epoch 9/20
50/50            0s 8ms/step -
accuracy: 1.0000 - loss: 5.7706e-04 - val_accuracy: 1.0000 - val_loss:
2.6559e-04
Epoch 10/20
50/50            0s 8ms/step -
accuracy: 1.0000 - loss: 5.0073e-04 - val_accuracy: 1.0000 - val_loss:
2.1909e-04
Epoch 11/20
50/50            0s 6ms/step -
accuracy: 1.0000 - loss: 3.5442e-04 - val_accuracy: 1.0000 - val_loss:
1.8338e-04
Epoch 12/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 2.9279e-04 - val_accuracy: 1.0000 - val_loss:
```

```
1.5708e-04
Epoch 13/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 2.7121e-04 - val_accuracy: 1.0000 - val_loss:
1.3585e-04
Epoch 14/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 2.2568e-04 - val_accuracy: 1.0000 - val_loss:
1.1755e-04
Epoch 15/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 2.1943e-04 - val_accuracy: 1.0000 - val_loss:
1.0379e-04
Epoch 16/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 1.9607e-04 - val_accuracy: 1.0000 - val_loss:
9.1955e-05
Epoch 17/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 1.4722e-04 - val_accuracy: 1.0000 - val_loss:
8.2813e-05
Epoch 18/20
50/50            0s 8ms/step -
accuracy: 1.0000 - loss: 1.4460e-04 - val_accuracy: 1.0000 - val_loss:
7.3032e-05
Epoch 19/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 1.2194e-04 - val_accuracy: 1.0000 - val_loss:
6.7159e-05
Epoch 20/20
50/50            0s 7ms/step -
accuracy: 1.0000 - loss: 1.2943e-04 - val_accuracy: 1.0000 - val_loss:
6.0770e-05
7/7            1s 97ms/step
Accuracy: 100.00%
```

[26]:

```
model.summary()
```

```
Model: "sequential_5"


 Layer (type)                                Output Shape                        ␣
↪Param #

 lstm_16 (LSTM)                              (None, 1, 128)                      ␣
↪89,088
```

```
 dropout_16 (Dropout)                 (None, 1, 128)                            ␣
↪  0

 lstm_17 (LSTM)                       (None, 64)                             ␣
↪49,408

 dropout_17 (Dropout)                 (None, 64)                                ␣
↪  0

 dense_5 (Dense)                      (None, 1)                                 ␣
↪ 65
```

Total params: 415,685 (1.59 MB)

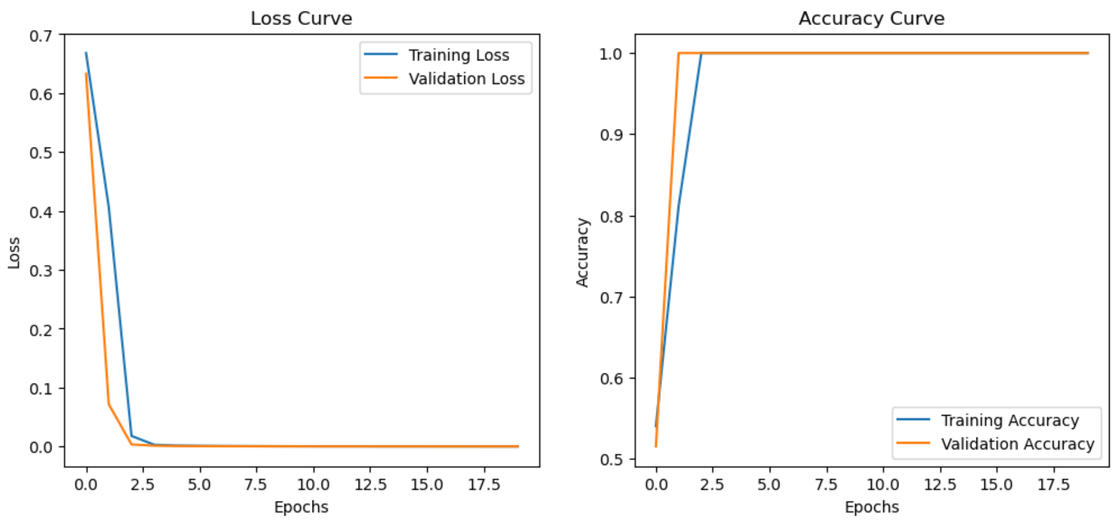
Trainable params: 138,561 (541.25 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 277,124 (1.06 MB)

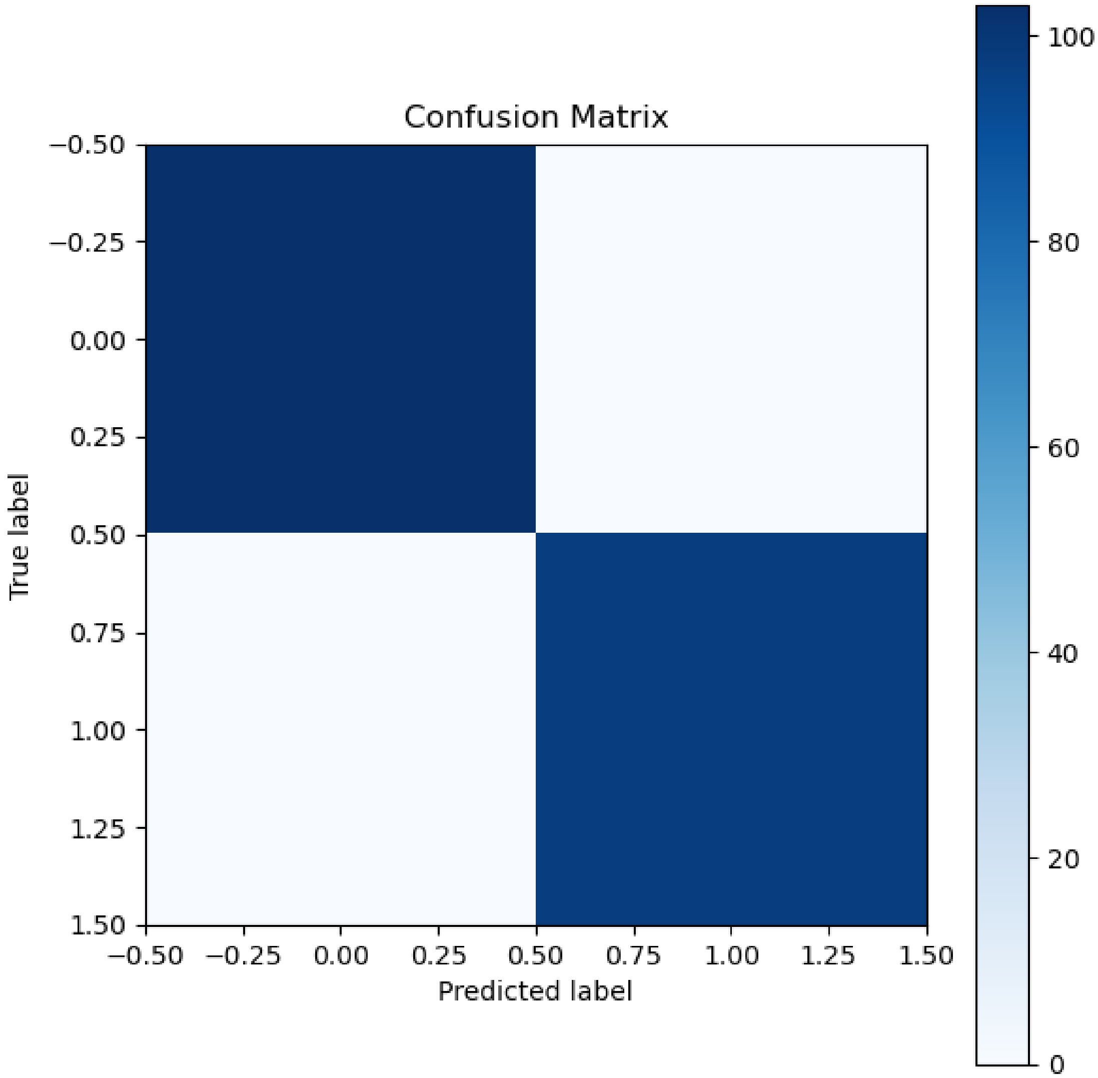
[27]:
```
# Plot training history
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Curve')

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



[28]:
```
# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



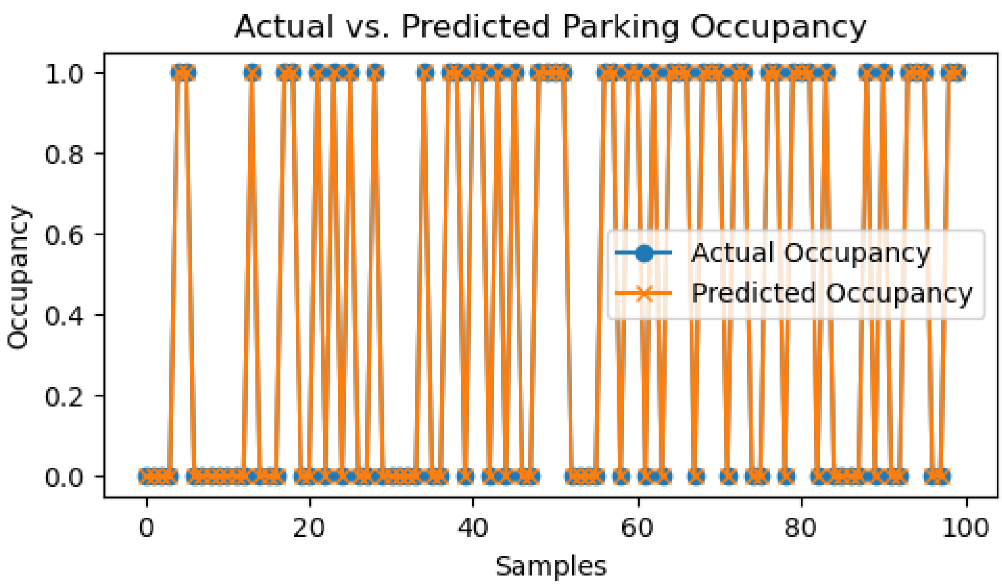

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       103
           1       1.00      1.00      1.00        97

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

[30]:
```
# Plot actual vs predicted values
plt.figure(figsize=(6, 3))
```

```python
plt.plot(y_test[:100], label='Actual Occupancy', marker='o')
plt.plot(y_pred[:100], label='Predicted Occupancy', marker='x')
plt.xlabel('Samples')
plt.ylabel('Occupancy')
plt.legend()
plt.title('Actual vs. Predicted Parking Occupancy')
plt.show()
```



0.0.8 As can be seen above F1-Score, Accuracy, Prediction and Recall of 100%

0.0.9 Achieved using LSTM for prediction of Occupancy Status using Feature Engineered Dataset