NANDHA ENGINEERING COLLEGE (AUTONOMOUS), ERODE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING

ASSIGNMENT I

ACADEMIC YEAR : 2024-2025 **CLASS : III - CSE**

MARKS : 20 marks **SEM : V**

TEAM 7:(22CS083 TO 22CS086)

S.No	QUESTION	Marks
1	Create a neural network to predict future values of a financial asset, such as stock prices, based on historical market data.	10
2	Implement an autoencoder to learn a set of features from input data that can be used to improve classification performance. Evaluate the effectiveness of these features in a downstream classification task.	10

Faculty signature **Student signature**

1.Create a neural network to predict future values of a financial asset, such as stock prices, based on historical market data.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

import yfinance as yf


# Step 1: Download the dataset

ticker = 'AAPL'

data = yf.download(ticker, start='2015-01-01', end='2023-01-01')

data = data[['Close']]


# Step 2: Preprocess the data

scaler = MinMaxScaler(feature_range=(0, 1))

scaled_data = scaler.fit_transform(data)


# Step 3: Create sequences

sequence_length = 60

X = []

y = []
```

```
for i in range(sequence_length, len(scaled_data)):
    X.append(scaled_data[i-sequence_length:i, 0])
    y.append(scaled_data[i, 0])

X, y = np.array(X), np.array(y)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))

# Step 4: Split the data
split_ratio = 0.8
train_size = int(len(X) * split_ratio)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Step 5: Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=25))
model.add(Dense(units=1))
```

```
# Step 6: Compile and train the model

model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=20, batch_size=32)


# Step 7: Evaluate the model

predictions = model.predict(X_test)

predictions = scaler.inverse_transform(predictions)


# Plotting the results

plt.figure(figsize=(16, 8))

plt.plot(data.index[train_size + sequence_length:], data['Close'][train_size + sequence_length:], color='blue', label='Actual Prices')

plt.plot(data.index[train_size + sequence_length:], predictions, color='red', label='Predicted Prices')

plt.title(f'{ticker} Stock Price Prediction')

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.legend()

plt.show()


# Step 8: Make future predictions (optional)

# Use the last sequence to predict the next price

last_sequence = scaled_data[-sequence_length:]
```

last_sequence = np.reshape(last_sequence, (1, sequence_length, 1))

next_price_scaled = model.predict(last_sequence)

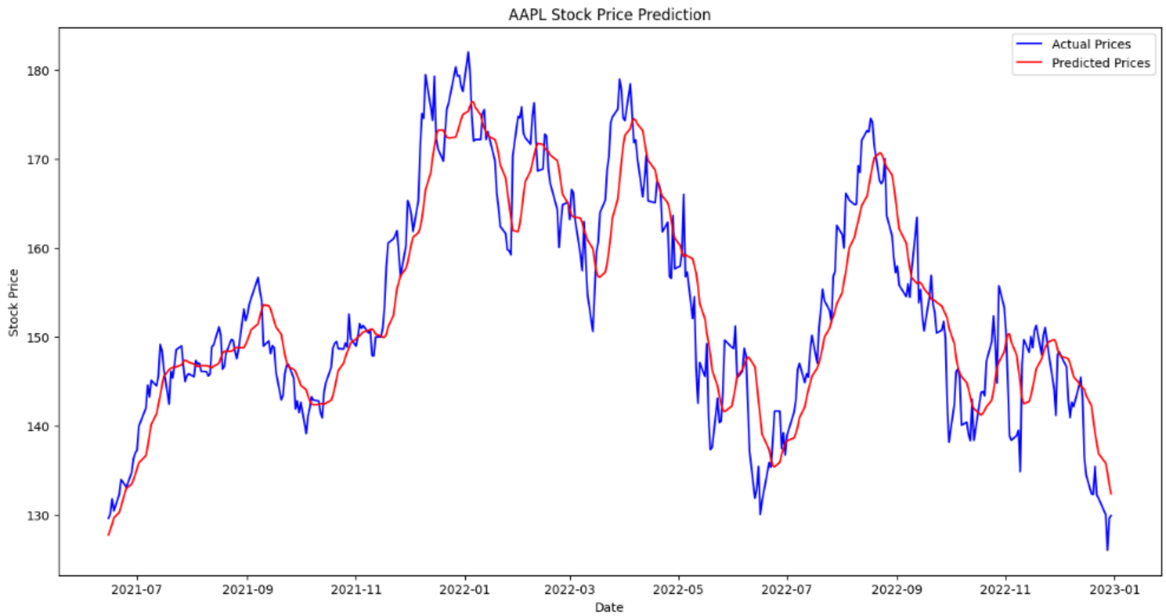
next_price = scaler.inverse_transform(next_price_scaled)

print(f'The predicted next price for {ticker} is: {next_price[0, 0]:.2f} USD')

Output:

```
Epoch 1/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 7s 48ms/step - loss: 0.0160
Epoch 2/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - loss: 0.0013
Epoch 3/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - loss: 0.0011
Epoch 4/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - loss: 0.0010
Epoch 5/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - loss: 8.1075e-04
Epoch 6/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 7.1657e-04
Epoch 7/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - loss: 6.6201e-04
Epoch 8/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - loss: 7.1641e-04
Epoch 9/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - loss: 6.1871e-04
Epoch 10/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - loss: 5.6393e-04
Epoch 11/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 5.7768e-04
Epoch 12/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - loss: 6.4627e-04
Epoch 13/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - loss: 6.3658e-04
Epoch 14/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 6.9670e-04
Epoch 15/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - loss: 5.2732e-04
Epoch 16/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - loss: 6.0051e-04
Epoch 17/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 5.2306e-04
Epoch 18/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 5.5646e-04
```

```
Epoch 19/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - loss: 5.7104e-04
Epoch 20/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 4.9139e-04
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
The predicted next price for AAPL is: 131.63 USD
```

2. Implement an autoencoder to learn a set of features from input data that can be used to improve classification performance. Evaluate the effectiveness of these features in a downstream classification task.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import tensorflow as tf
from tensorflow.keras import layers, models

# Step 1: Create and Prepare Your Data
# Generate a synthetic dataset
X, y = make_classification(n_samples=1000,    # Number of samples
                           n_features=20,     # Number of features
                           n_informative=15,  # Number of informative features
                           n_redundant=5,     # Number of redundant features
                           n_classes=3,       # Number of classes
                           random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize the data
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 2: Build the Autoencoder
# Define the autoencoder architecture
input_dim = X_train.shape[1]  # Number of features

# Encoder
input_layer = layers.Input(shape=(input_dim,))
encoded = layers.Dense(64, activation='relu')(input_layer)
encoded = layers.Dense(32, activation='relu')(encoded)
encoded = layers.Dense(16, activation='relu')(encoded)

# Latent space
latent_space = layers.Dense(8, activation='relu')(encoded)

# Decoder
decoded = layers.Dense(16, activation='relu')(latent_space)
decoded = layers.Dense(32, activation='relu')(decoded)
decoded = layers.Dense(64, activation='relu')(decoded)
output_layer = layers.Dense(input_dim, activation='sigmoid')(decoded)

# Autoencoder model
autoencoder = models.Model(input_layer, output_layer)

# Compile the model
```

```
autoencoder.compile(optimizer='adam', loss='mse')

# Summary of the model
autoencoder.summary()

# Step 3: Train the Autoencoder
history = autoencoder.fit(X_train, X_train, epochs=50, batch_size=32, validation_split=0.2)

# Step 4: Extract Features Using the Encoder
# Extract the encoder part of the autoencoder
encoder = models.Model(input_layer, latent_space)

# Encode the training and test data
X_train_encoded = encoder.predict(X_train)
X_test_encoded = encoder.predict(X_test)

# Step 5: Build and Train a Classifier
# Build a classifier using the encoded features
classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
classifier.fit(X_train_encoded, y_train)

# Predict on the test set
y_pred = classifier.predict(X_test_encoded)

# Step 6: Evaluate Performance
```

```
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{report}")

# Optional: Evaluate using original features for comparison
classifier_original = RandomForestClassifier(n_estimators=100, random_state=42)
classifier_original.fit(X_train, y_train)
y_pred_original = classifier_original.predict(X_test)

accuracy_original = accuracy_score(y_test, y_pred_original)
report_original = classification_report(y_test, y_pred_original)

print(f"Accuracy with Original Features: {accuracy_original}")
print(f"Classification Report with Original Features:\n{report_original}")
```

...

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 20)	0
dense (Dense)	(None, 64)	1,344
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 8)	136
dense_4 (Dense)	(None, 16)	144
dense_5 (Dense)	(None, 32)	544
dense_6 (Dense)	(None, 64)	2,112
dense_7 (Dense)	(None, 20)	1,300

...

Total params: 8,188 (31.98 KB)

...

Trainable params: 8,188 (31.98 KB)

...

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 1.2416 - val_loss: 1.1585
Epoch 2/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1.1575 - val_loss: 0.9884
Epoch 3/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.0100 - val_loss: 0.9567
Epoch 4/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1.0099 - val_loss: 0.9504
Epoch 5/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.9956 - val_loss: 0.9359
Epoch 6/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.9633 - val_loss: 0.9193
Epoch 7/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.9689 - val_loss: 0.9013
Epoch 8/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.9235 - val_loss: 0.8891
Epoch 9/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.9286 - val_loss: 0.8734
Epoch 10/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.9050 - val_loss: 0.8646
Epoch 11/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.9240 - val_loss: 0.8516
Epoch 12/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8866 - val_loss: 0.8382
Epoch 13/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.8893 - val_loss: 0.8281
Epoch 14/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8602 - val_loss: 0.8231
Epoch 15/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8598 - val_loss: 0.8223
```

```
Epoch 16/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8687 - val_loss: 0.8200
Epoch 17/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8760 - val_loss: 0.8176
Epoch 18/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8582 - val_loss: 0.8120
Epoch 19/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8588 - val_loss: 0.8114
Epoch 20/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.8748 - val_loss: 0.8082
Epoch 21/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.8313 - val_loss: 0.8084
Epoch 22/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.8420 - val_loss: 0.8069
Epoch 23/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.8570 - val_loss: 0.8076
Epoch 24/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.8312 - val_loss: 0.8074
Epoch 25/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.8594 - val_loss: 0.8043
Epoch 26/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 0.8454 - val_loss: 0.7977
Epoch 27/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.8295 - val_loss: 0.7927
Epoch 28/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.8368 - val_loss: 0.7893
Epoch 29/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.8239 - val_loss: 0.7843
Epoch 30/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.8016 - val_loss: 0.7831
```

```
Epoch 31/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8206 - val_loss: 0.7818
Epoch 32/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.7906 - val_loss: 0.7804
Epoch 33/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.7788 - val_loss: 0.7779
Epoch 34/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.8072 - val_loss: 0.7750
Epoch 35/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.8080 - val_loss: 0.7756
Epoch 36/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.8214 - val_loss: 0.7731
Epoch 37/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.8000 - val_loss: 0.7725
Epoch 38/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.7984 - val_loss: 0.7747
Epoch 39/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.7916 - val_loss: 0.7700
Epoch 40/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.8180 - val_loss: 0.7692
Epoch 41/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.8095 - val_loss: 0.7665
Epoch 42/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.7925 - val_loss: 0.7689
Epoch 43/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.7975 - val_loss: 0.7676
Epoch 44/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.7823 - val_loss: 0.7655
Epoch 45/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.8007 - val_loss: 0.7650
```

```
Epoch 46/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.7964 - val_loss: 0.7646
Epoch 47/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.8241 - val_loss: 0.7640
Epoch 48/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.7961 - val_loss: 0.7674
Epoch 49/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.7911 - val_loss: 0.7638
Epoch 50/50
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.8074 - val_loss: 0.7625
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

```
Accuracy: 0.625
Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.66      0.61        70
           1       0.72      0.67      0.70        73
           2       0.58      0.53      0.55        57

    accuracy                           0.62       200
   macro avg       0.62      0.62      0.62       200
weighted avg       0.63      0.62      0.63       200

Accuracy with Original Features: 0.765
Classification Report with Original Features:
              precision    recall  f1-score   support

           0       0.72      0.86      0.78        70
           1       0.84      0.71      0.77        73
           2       0.75      0.72      0.73        57

    accuracy                           0.77       200
   macro avg       0.77      0.76      0.76       200
weighted avg       0.77      0.77      0.76       200
```