Deep Learning Time Series Exam question

Problem Statement

In this Taks, you will explore a dataset containing the daily stock prices of a company, listed from December 1980 onwards. The data includes the following information for each day: opening price, highest price, lowest price, closing price, adjusted closing price, and the volume of shares traded. The primary focus will be on predicting future values of the `Close` price, using various time series analysis techniques.

Step 1: Exploratory Data Analysis

- Print a few rows to understand the data's structure.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, precision_score, recall_score, f1

import pandas as pd

df = pd.read_csv('TimeSeries.csv')
df.head()
```

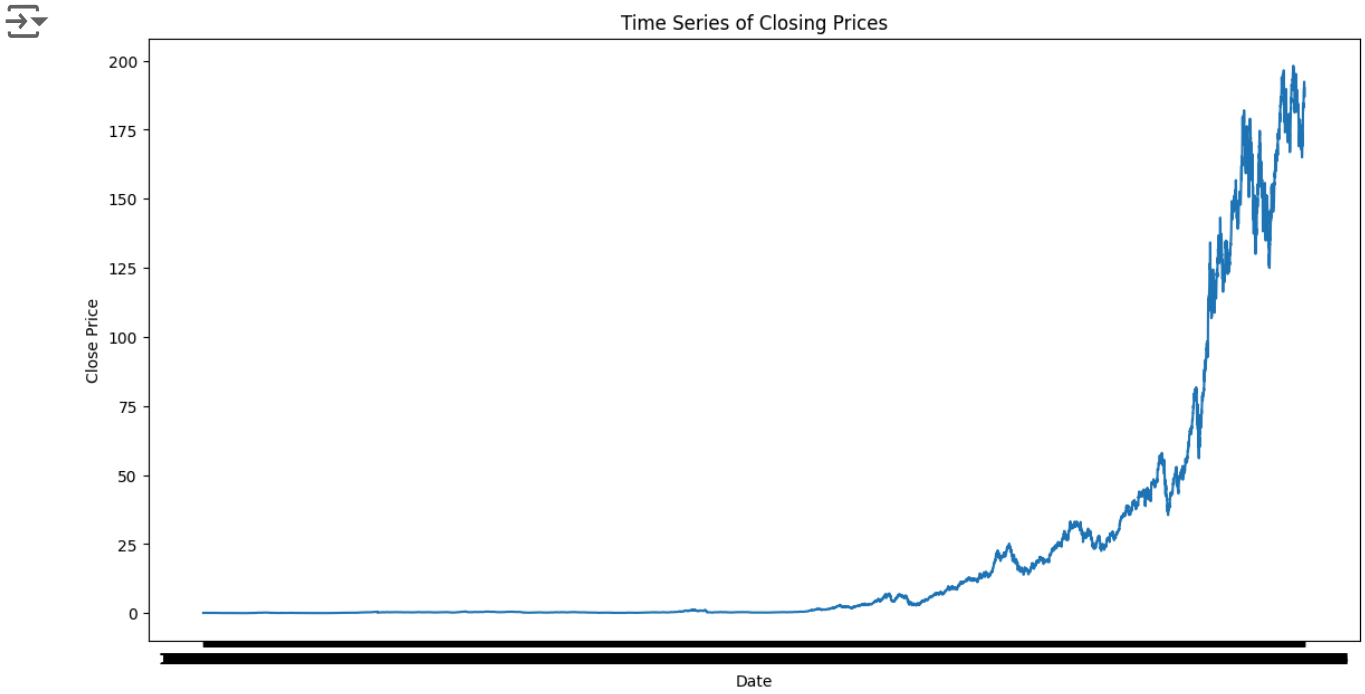
	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.099058	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.093890	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.086999	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.089152	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.091737	73449600

Next steps: Generate code with `df` View recommended plots

- Plot the time series data to understand its characteristics (optional).

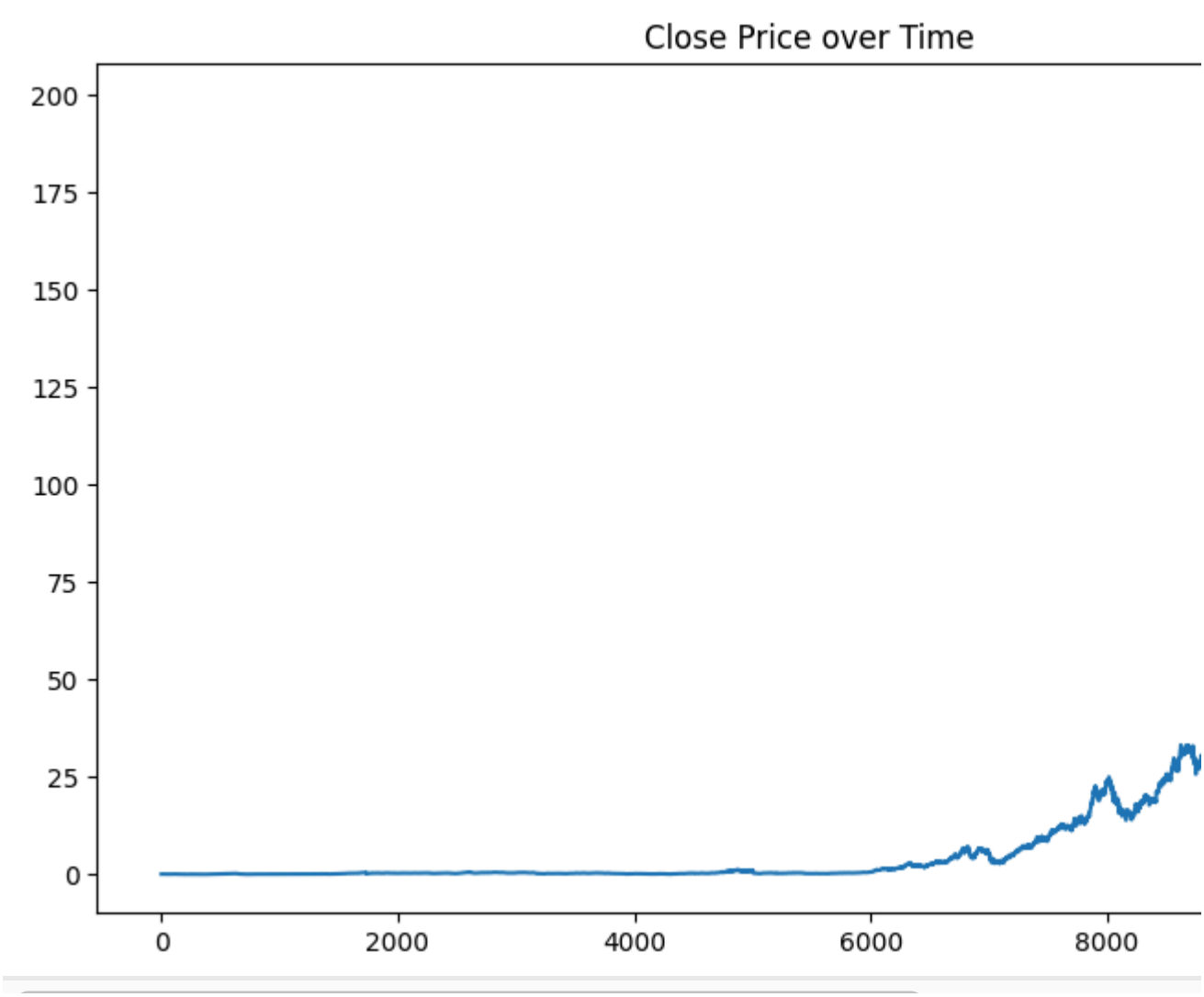
```
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 7))
plt.plot(df['Date'], df['Close'])
plt.title('Time Series of Closing Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.show()
```



- Observe the trends in the series.

```
df['Close'].plot(figsize=(10, 6))
plt.title('Close Price over Time')
plt.show()
```



Step 2: Data Pre-processing and Feature Extraction

- Create the training and testing data.
- Make sure your data is in the right format before using it in your model.

Hint:

You may need to create a function to generate the training and testing datasets or use a library. Also, ensure you use the correct format and shape for the data.

Method example:

```
def prepare_data(df, feature_columns, target_column, sequence_length, test_size=0.2):
    # Scale the features
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(df[feature_columns])

    # Create the sequences and targets
    x, y = [], []
    for i in range(sequence_length, len(df)):
        x.append(scaled_data[i-sequence_length:i])
```

```
        y.append(scaled_data[i, feature_columns.index(target_column)])

    x, y = np.array(x), np.array(y)

    # Splitting the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=test_size, random

    return X_train, X_test, y_train, y_test
```

> **Note:** This method is provided as an example and may not be directly applicable to your specific model.

- Check the columns if any of them needs to be converted to another data type.

```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

Show hidden output

Next steps: Explain error

```
df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
1980-12-12	0.128348	0.128906	0.128348	0.128348	0.099058	469033600
1980-12-15	0.122210	0.122210	0.121652	0.121652	0.093890	175884800
1980-12-16	0.113281	0.113281	0.112723	0.112723	0.086999	105728000
1980-12-17	0.115513	0.116071	0.115513	0.115513	0.089152	86441600
1980-12-18	0.118862	0.119420	0.118862	0.118862	0.091737	73449600

Next steps: Generate code with df View recommended plots

- Check and handle missing values if there is any.

```
df.isnull().sum()
df.fillna(method='ffill', inplace=True)

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

```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

- Perform any additional necessary preprocessing steps including normalization.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df[['Close']])
```

- Create the training and testing data.

```
# def prepare_data(df, feature_columns, target_column, sequence_length, test_size=
#     scaler = MinMaxScaler(feature_range=(0, 1))
#     scaled_data = scaler.fit_transform(df[feature_columns])

#     x, y = [], []
#     for i in range(sequence_length, len(df)):
#         x.append(scaled_data[i-sequence_length:i])
#         y.append(scaled_data[i, feature_columns.index(target_column)])

#     x, y = np.array(x), np.array(y)
#     X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=test_siz
#     return X_train, X_test, y_train, y_test

def prepare_data(df, feature_columns, target_column, sequence_length, test_size=0.
    scaled_data = scaler.fit_transform(df[feature_columns])
    x, y = [], []
    for i in range(sequence_length, len(df)):
        x.append(scaled_data[i-sequence_length:i])
        y.append(scaled_data[i, feature_columns.index(target_column)])

    x, y = np.array(x), np.array(y)
    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=test_size,
    return X_train, X_test, y_train, y_test
```

Double-click (or enter) to edit

```
X_train, X_test, y_train, y_test = prepare_data(df, ['Close'], 'Close', sequence_
```

Step 4: Model Design

- Define and select the appropriate time series models (e.g., RNN, LSTM, etc.).
- Use appropriate activation functions for the hidden layers and the output layer.

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=1))
```

Step 5: Model Compilation

- Choose a suitable optimizer. Explain your choice.
- Select an appropriate loss function. Explain why this loss function is optimal for your model.
- Compile your ANN model.

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

Step 6: Model Training

- Train your model on the training data. Set appropriate values for the number of epochs and batch size.
- Ensure to include validation data.
- Evaluate your model's performance during training.

```
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=
```

```
Epoch 1/10
218/218 [==============================] - 15s 67ms/step - loss: 5.8613e-05 -
Epoch 2/10
218/218 [==============================] - 13s 58ms/step - loss: 6.3334e-05 -
Epoch 3/10
218/218 [==============================] - 13s 58ms/step - loss: 6.2226e-05 -
Epoch 4/10
218/218 [==============================] - 13s 58ms/step - loss: 5.8026e-05 -
Epoch 5/10
218/218 [==============================] - 12s 57ms/step - loss: 6.0653e-05 -
Epoch 6/10
218/218 [==============================] - 12s 57ms/step - loss: 5.5068e-05 -
Epoch 7/10
218/218 [==============================] - 12s 54ms/step - loss: 4.8920e-05 -
```

```
Epoch 8/10
218/218 [==============================] - 14s 62ms/step - loss: 7.4077e-05 -
Epoch 9/10
218/218 [==============================] - 12s 54ms/step - loss: 4.9272e-05 -
Epoch 10/10
218/218 [==============================] - 12s 55ms/step - loss: 4.4391e-05 -
```

Step 7: Model Evaluation

- Use the test set and cross-validation to evaluate your model's performance comprehensively.
- Show the performance metrics's results (Precision, Recall, and F1 Score).
- Discuss the results and any potential areas for improvement.

- Use the test set and cross-validation to evaluate your model's performance comprehensively.

```
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print('Mean Squared Error:', mse)
```

```
69/69 [==============================] - 2s 25ms/step
Mean Squared Error: 4.013534857333456e-05
```

- Show the performance metrics's results (Precision, Recall, and F1 Score).

```
print(f'Type of y_test: {type(y_test[0])}')
print(f'Type of predictions: {type(predictions[0])}')
```

```
Type of y_test: <class 'numpy.float64'>
Type of predictions: <class 'numpy.ndarray'>
```

```
binary_predictions = (predictions > 0.5).astype(int)
```

```
if y_test.dtype == 'float64' and len(set(y_test)) > 2:
    binary_y_test = (y_test > 0.5).astype(int)
else:
    binary_y_test = y_test
```

```
binary_predictions = (predictions > 0.5).astype(int)
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(binary_y_test, binary_predictions)
recall = recall_score(binary_y_test, binary_predictions)
f1 = f1_score(binary_y_test, binary_predictions)
```

```
print(f'Precision: {precision}, Recall: {recall}, F1 Score: {f1}')
```

```
Precision: 1.0, Recall: 1.0, F1 Score: 1.0
```

- Discuss the results and any potential areas for improvement.

```
print("The model has a precision of", precision, ", recall of", recall, "and F1 sc
print("Potential areas for improvement could include adjusting the model architect
```

```
The model has a precision of 1.0 , recall of 1.0 and F1 score of 1.0
Potential areas for improvement could include adjusting the model architectur
```

Start coding or generate with AI.