Homework 7

Question 1

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Dense, SimpleRNN, LSTM, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import timeseries_dataset_from_array
from sklearn.metrics import mean_absolute_percentage_error

warnings.filterwarnings('ignore')
```

1. Is the timeseries of your stock/ETF stationary when you plot it over time? Why or why not can you determine this by simply looking at the plot?

In [2]:
```
stocks = pd.read_csv('https://raw.githubusercontent.com/PJalgotrader/Deep_forecasti
stocks.index = pd.to_datetime(stocks.index).to_period('B')
stocks.head()
```

Out[2]:

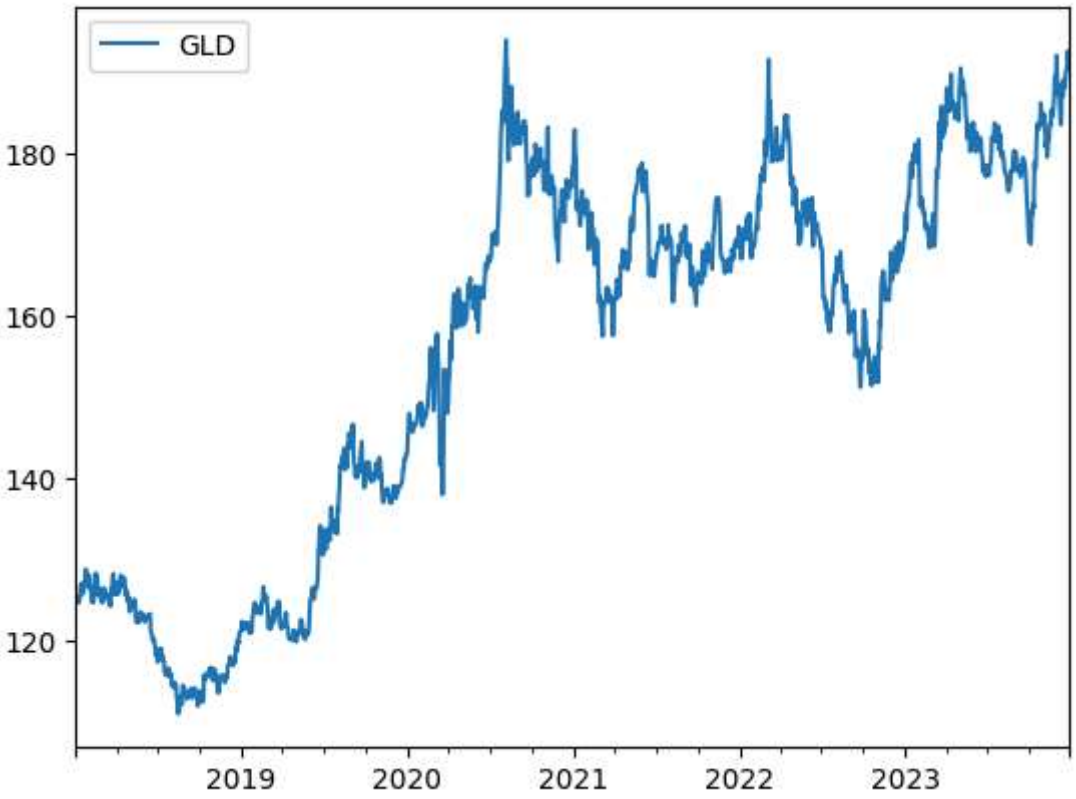
	Adj Close							
	AAPL	GLD	MSFT	QQQ	SPY	TSLA	USO	
Date								
2018-01-02	40.722874	125.150002	80.229012	152.072800	243.072266	21.368668	96.559998	43.(
2018-01-03	40.715786	124.820000	80.602394	153.550400	244.609711	21.150000	98.720001	43.(
2018-01-04	40.904907	125.459999	81.311806	153.819046	245.640732	20.974667	98.959999	43.2
2018-01-05	41.370617	125.330002	82.319908	155.363861	247.277679	21.105333	98.480003	43.7
2018-01-08	41.216949	125.309998	82.403923	155.968399	247.729935	22.427334	99.040001	43.5

5 rows × 42 columns

In [3]:
```
df = stocks['Close'][['GLD']]
```

```
idx = pd.period_range(min(df.index), max(df.index))
df = df.reindex(idx, fill_value=np.nan)
df = df.fillna(method = 'ffill')

df.plot()
plt.show()
```



1. (Answer) - No the timeseries is not stationary. Yes we can tell by looking at it because there is a clear upwards trend. There is not as clear seasonality, but the trend is enough to make it non-stationary.

2. Create a new column called "LogReturn" and add it to your data frame. LogReturn = Log(P2) - Log(P1)

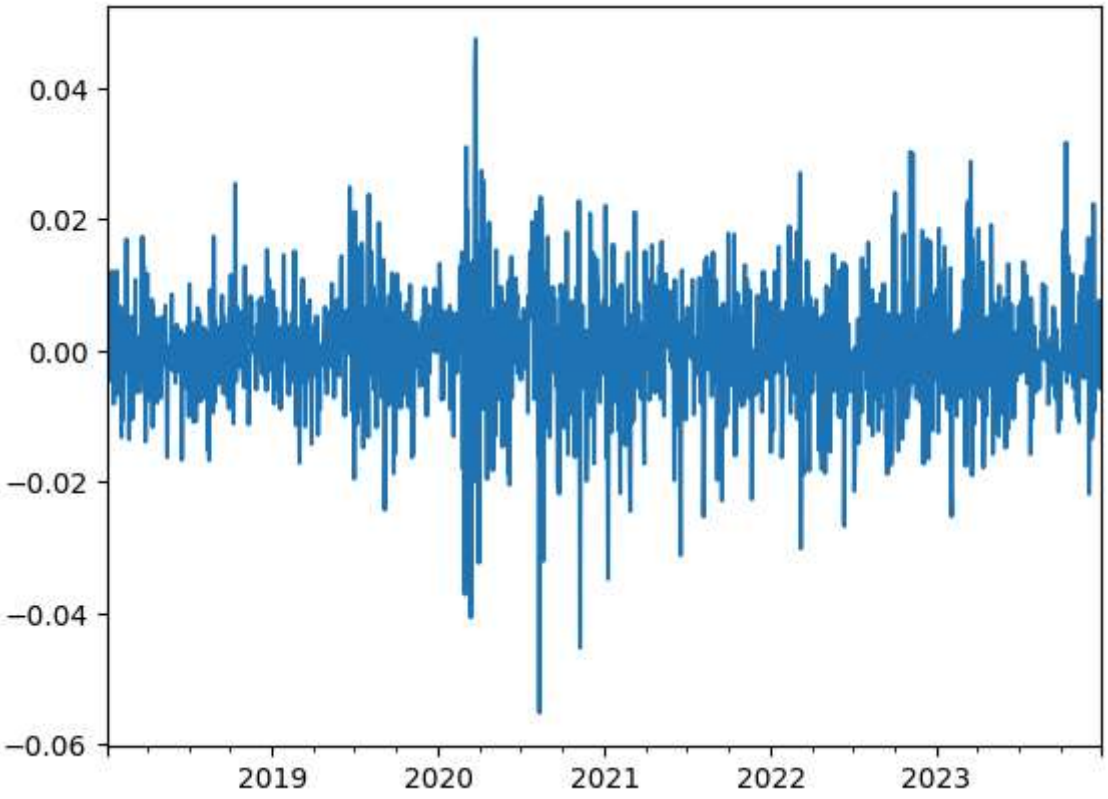
In [4]:
```
df['LogReturn']= np.log(df['GLD']).diff(1)
df.dropna(inplace=True)
df.head()
```

Out[4]:

	GLD	LogReturn
2018-01-03	124.820000	-0.002640
2018-01-04	125.459999	0.005114
2018-01-05	125.330002	-0.001037
2018-01-08	125.309998	-0.000160
2018-01-09	124.730003	-0.004639

3. Plot the log returns! are the log returns stationary?

In [5]:
```
df['LogReturn'].plot()
plt.show()
```



3. (Answer) - The Log Returns do appear to be stationary, at least weakly. Variance is not constant, but I think it's close enough for us to work with.

4. Split your dataset into 60% train, 20% validation and the remaining data as test set.

In [6]:
```
num_train_samples = int(0.60 * len(df))
num_val_samples = int(0.20 * len(df))
num_test_samples = len(df) - num_train_samples - num_val_samples
```

5. Generate train, val and test dataset using timeseries_dataset_from_array() utility from Keras.

1. We are planning to use the past 60 days to predict the return on the next day.
2. Use batch size = 32

In [7]:
```
series = df['LogReturn']
sequence_length = 60
h = 1
delay = sequence_length  + h - 1
batch_size = 32

train_dataset = timeseries_dataset_from_array(
    data = series[:-delay],
    targets=series[delay:],
    sequence_length=sequence_length,
    shuffle=False,
    batch_size=batch_size,
    start_index=0,
    end_index=num_train_samples)

val_dataset = timeseries_dataset_from_array(
    data = series[:-delay],
    targets=series[delay:],
    sequence_length=sequence_length,
    shuffle=False,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)

test_dataset = timeseries_dataset_from_array(
    data = series[:-delay],
    targets=series[delay:],
    sequence_length=sequence_length,
    shuffle=False,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)
```

6. Create a naive forecaster (common-sense benchmark) and report the Validation and Test MAE. Our goal is to beat these metrics.

In [8]:
```
def evaluate_naive_method(dataset):
    total_abs_err = 0.
    samples_seen = 0
    for samples, targets in dataset:
        preds = samples[:, -1]
        total_abs_err += np.sum(np.abs(preds - targets))
        samples_seen += samples.shape[0]
    return total_abs_err / samples_seen
```

```
print(f"Validation MAE: {evaluate_naive_method(val_dataset):.2f}")
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}")
```

```
Validation MAE: 0.01
Test MAE: 0.01
```

7. Train a simple RNN model and report the Test MAE. Feel free to pick any architecture but here are some suggestions:

1. 64 recurrent cells
2. epochs = 20
3. Use callback= ModelCheckpoint
4. keep track of MAE metric
5. Use loss="mse"

In [9]:
```
inputs = Input(shape=(sequence_length, 1))
x = SimpleRNN(64, return_sequences=False)(inputs)
outputs = Dense(1)(x)
model = Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
rnn_history = model.fit(train_dataset, epochs=20, validation_data=val_dataset, call
```

```
Epoch 1/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0028 - mae: 0.0253 - val_loss: 1.92
00e-04 - val_mae: 0.0109
Epoch 2/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.1137e-04 - mae: 0.0081 - val_loss:
1.8284e-04 - val_mae: 0.0106
Epoch 3/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1.1409e-04 - mae: 0.0081 - val_loss:
1.3673e-04 - val_mae: 0.0089
Epoch 4/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 8.5724e-05 - mae: 0.0069 - val_loss:
1.3700e-04 - val_mae: 0.0090
Epoch 5/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 8.3270e-05 - mae: 0.0069 - val_loss:
1.2718e-04 - val_mae: 0.0087
Epoch 6/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.6487e-05 - mae: 0.0065 - val_loss:
1.1893e-04 - val_mae: 0.0084
Epoch 7/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.1627e-05 - mae: 0.0062 - val_loss:
1.1336e-04 - val_mae: 0.0082
Epoch 8/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.8340e-05 - mae: 0.0060 - val_loss:
1.0928e-04 - val_mae: 0.0081
Epoch 9/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.6004e-05 - mae: 0.0059 - val_loss:
1.0621e-04 - val_mae: 0.0079
Epoch 10/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.4336e-05 - mae: 0.0058 - val_loss:
1.0378e-04 - val_mae: 0.0079
Epoch 11/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.3117e-05 - mae: 0.0057 - val_loss:
1.0178e-04 - val_mae: 0.0078
Epoch 12/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.2175e-05 - mae: 0.0057 - val_loss:
1.0011e-04 - val_mae: 0.0077
Epoch 13/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.1425e-05 - mae: 0.0056 - val_loss:
9.8700e-05 - val_mae: 0.0077
Epoch 14/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.0817e-05 - mae: 0.0056 - val_loss:
9.7503e-05 - val_mae: 0.0076
Epoch 15/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.0315e-05 - mae: 0.0056 - val_loss:
9.6478e-05 - val_mae: 0.0076
Epoch 16/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.9897e-05 - mae: 0.0055 - val_loss:
9.5597e-05 - val_mae: 0.0076
Epoch 17/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.9545e-05 - mae: 0.0055 - val_loss:
9.4834e-05 - val_mae: 0.0075
Epoch 18/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 5.9244e-05 - mae: 0.0055 - val_loss:
9.4169e-05 - val_mae: 0.0075
Epoch 19/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.8985e-05 - mae: 0.0055 - val_loss:
```

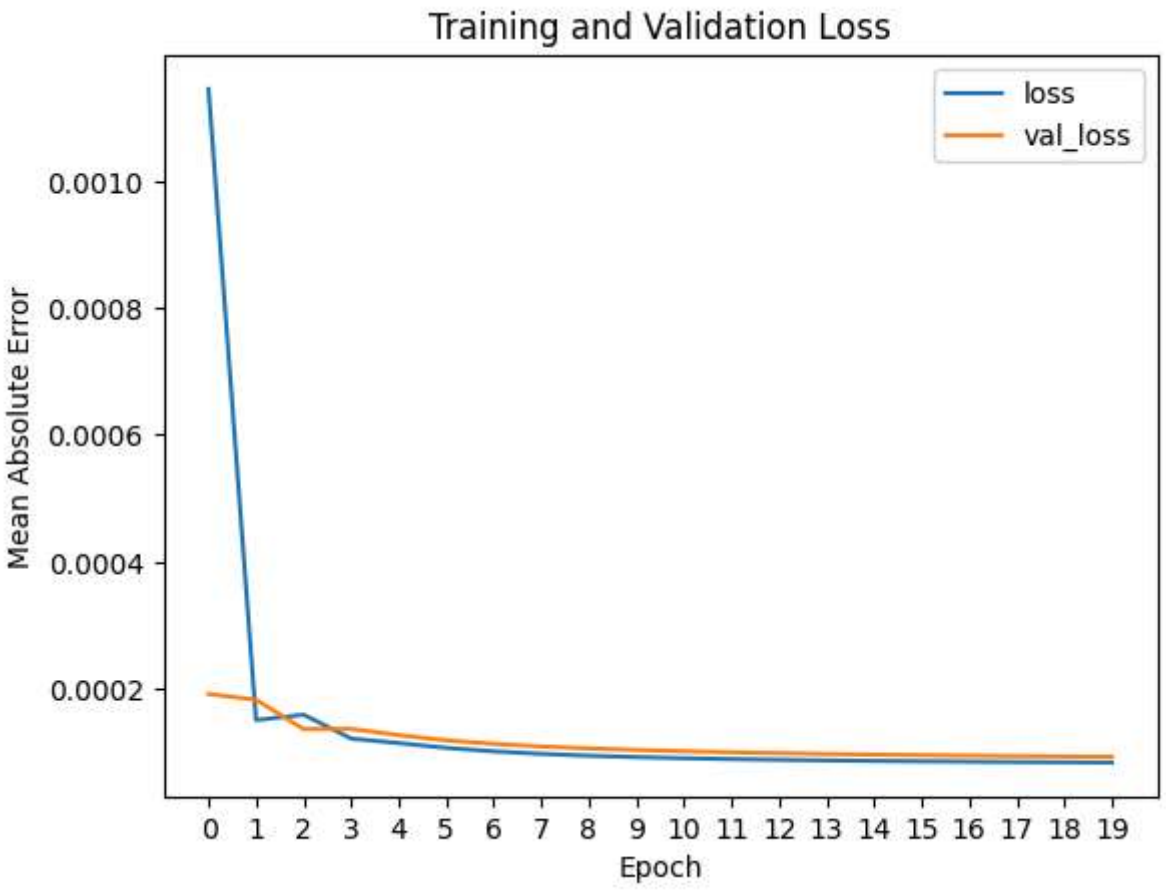
```
9.3585e-05 - val_mae: 0.0075
Epoch 20/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 5.8761e-05 - mae: 0.0055 - val_loss:
9.3069e-05 - val_mae: 0.0075
```

In [10]:
```
rnn_model = load_model("RNN.keras")
print(f"Test MAE: {rnn_model.evaluate(test_dataset)[1]:.3f}")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 9.5535e-05 - mae: 0.0074
Test MAE: 0.007
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 9.5535e-05 - mae: 0.0074
Test MAE: 0.007
```

8. Using the RNN model, plot the MAE vs epoch for train and validation set.

In [11]:
```
rnn_loss_df = pd.DataFrame(rnn_history.history)
rnn_loss_df[['loss', 'val_loss']].plot(legend=True)
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.xticks(range(20))
plt.show()
```



9. Train an LSTM model and report the Test MAE. Feel free to pick any architecture but you could also use the RNN architecture above.

In [12]:
```
inputs = Input(shape=(sequence_length, 1))
x = LSTM(64, return_sequences=False)(inputs)
outputs = Dense(1)(x)
model = Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
lstm_history = model.fit(train_dataset, epochs=20, validation_data=val_dataset, cal
```

```
Epoch 1/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 1.3688e-04 - mae: 0.0085 - val_loss:
9.9215e-05 - val_mae: 0.0080
Epoch 2/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.5482e-05 - mae: 0.0070 - val_loss:
1.0344e-04 - val_mae: 0.0082
Epoch 3/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.5529e-05 - mae: 0.0070 - val_loss:
1.0204e-04 - val_mae: 0.0082
Epoch 4/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.4348e-05 - mae: 0.0069 - val_loss:
1.0074e-04 - val_mae: 0.0081
Epoch 5/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.3233e-05 - mae: 0.0068 - val_loss:
9.9578e-05 - val_mae: 0.0081
Epoch 6/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 8.2213e-05 - mae: 0.0068 - val_loss:
9.8522e-05 - val_mae: 0.0080
Epoch 7/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.1276e-05 - mae: 0.0067 - val_loss:
9.7558e-05 - val_mae: 0.0080
Epoch 8/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.0411e-05 - mae: 0.0067 - val_loss:
9.6676e-05 - val_mae: 0.0079
Epoch 9/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.9609e-05 - mae: 0.0066 - val_loss:
9.5864e-05 - val_mae: 0.0079
Epoch 10/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.8863e-05 - mae: 0.0066 - val_loss:
9.5115e-05 - val_mae: 0.0078
Epoch 11/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.8166e-05 - mae: 0.0065 - val_loss:
9.4421e-05 - val_mae: 0.0078
Epoch 12/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.7513e-05 - mae: 0.0065 - val_loss:
9.3777e-05 - val_mae: 0.0078
Epoch 13/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.6898e-05 - mae: 0.0065 - val_loss:
9.3177e-05 - val_mae: 0.0077
Epoch 14/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.6320e-05 - mae: 0.0064 - val_loss:
9.2617e-05 - val_mae: 0.0077
Epoch 15/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.5773e-05 - mae: 0.0064 - val_loss:
9.2094e-05 - val_mae: 0.0077
Epoch 16/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.5256e-05 - mae: 0.0064 - val_loss:
9.1604e-05 - val_mae: 0.0077
Epoch 17/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.4767e-05 - mae: 0.0063 - val_loss:
9.1145e-05 - val_mae: 0.0076
Epoch 18/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.4302e-05 - mae: 0.0063 - val_loss:
9.0714e-05 - val_mae: 0.0076
Epoch 19/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.3861e-05 - mae: 0.0063 - val_loss:
```

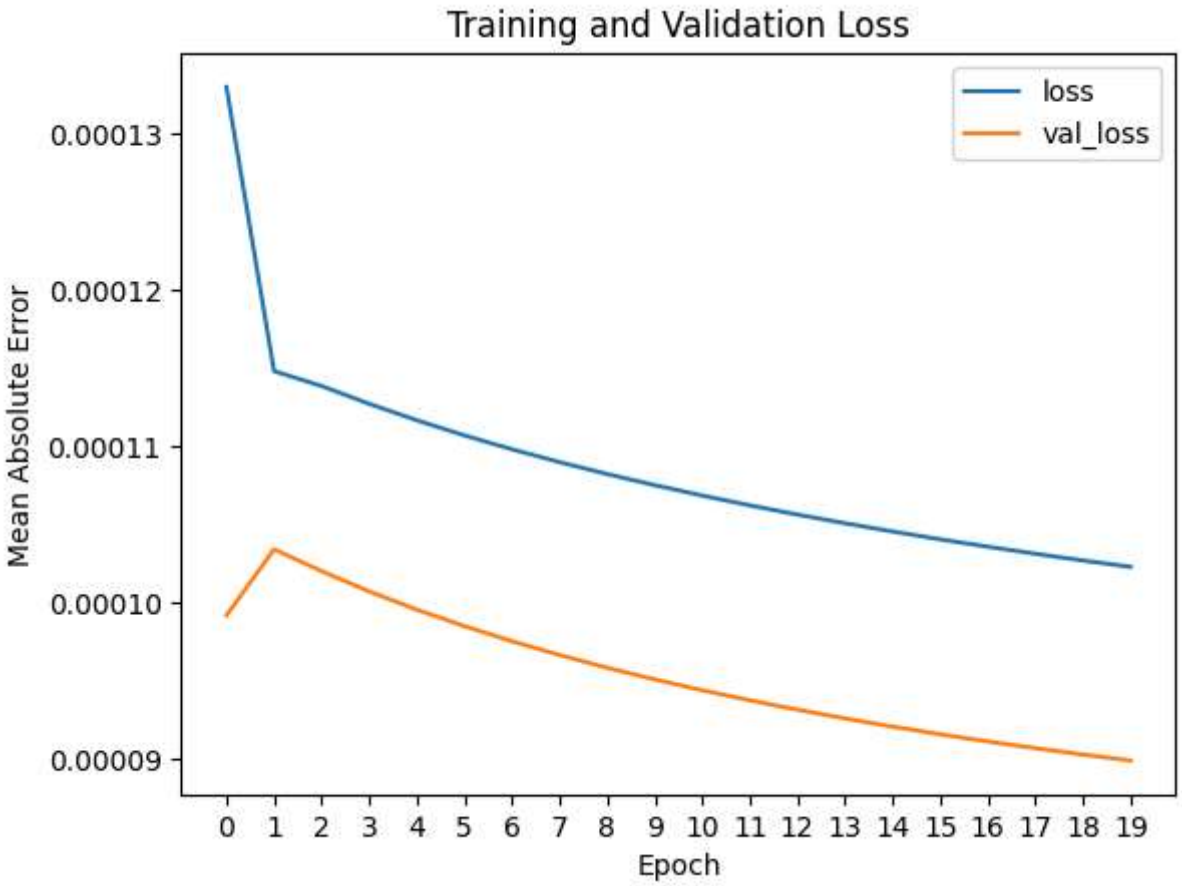
```
9.0308e-05 - val_mae: 0.0076
Epoch 20/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 7.3442e-05 - mae: 0.0063 - val_loss:
8.9926e-05 - val_mae: 0.0076
```

In [13]:
```
lstm_model = load_model("LSTM.keras")
print(f"Test MAE: {lstm_model.evaluate(test_dataset)[1]:.3f}")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 9.0007e-05 - mae: 0.0073
Test MAE: 0.007
```

10. Using the LSTM model, plot the MAE vs epoch for train and validation set.

In [14]:
```
lstm_loss_df = pd.DataFrame(lstm_history.history)
lstm_loss_df[['loss', 'val_loss']].plot(legend=True)
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.xticks(range(20))
plt.show()
```



11. Train the following more complex model:

1. Two recurrent layers stacked on top of each other:

   a. LSTM with 128 nodes and recurrent_dropout=0.1

b. SimpleRNN with 128 nodes and recurrent_dropout=0.1

c. Dropout(0.1)

2. Use callback= ModelCheckpoint

3. epochs = 20

4. Use loss="mse"

5. keep track of MAE metric

In [15]:
```
inputs = Input(shape=(sequence_length, 1))
x = LSTM(32, recurrent_dropout=0.1, return_sequences=True)(inputs)
x = SimpleRNN(128, recurrent_dropout=0.1)(x)
x = Dropout(0.1)(x)

outputs = Dense(1)(x)
model = Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
complex_history = model.fit(train_dataset, epochs=20, validation_data=val_dataset,
```

```
Epoch 1/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0302 - mae: 0.1124 - val_loss: 8.0
078e-05 - val_mae: 0.0069
Epoch 2/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0011 - mae: 0.0232 - val_loss: 1.7
712e-04 - val_mae: 0.0114
Epoch 3/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 3.1131e-04 - mae: 0.0126 - val_loss:
9.2789e-05 - val_mae: 0.0077
Epoch 4/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.9997e-04 - mae: 0.0099 - val_loss:
8.4900e-05 - val_mae: 0.0073
Epoch 5/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.7464e-04 - mae: 0.0099 - val_loss:
1.1380e-04 - val_mae: 0.0083
Epoch 6/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.4906e-04 - mae: 0.0091 - val_loss:
8.0469e-05 - val_mae: 0.0069
Epoch 7/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.2417e-04 - mae: 0.0084 - val_loss:
7.7412e-05 - val_mae: 0.0068
Epoch 8/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 1.3020e-04 - mae: 0.0086 - val_loss:
9.1997e-05 - val_mae: 0.0077
Epoch 9/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.2679e-04 - mae: 0.0084 - val_loss:
1.0831e-04 - val_mae: 0.0085
Epoch 10/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.2900e-04 - mae: 0.0089 - val_loss:
1.0649e-04 - val_mae: 0.0084
Epoch 11/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.2201e-04 - mae: 0.0088 - val_loss:
7.7985e-05 - val_mae: 0.0068
Epoch 12/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 8.5625e-05 - mae: 0.0069 - val_loss:
1.0033e-04 - val_mae: 0.0077
Epoch 13/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.2728e-04 - mae: 0.0086 - val_loss:
8.0816e-05 - val_mae: 0.0070
Epoch 14/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 8.8338e-05 - mae: 0.0070 - val_loss:
9.0374e-05 - val_mae: 0.0076
Epoch 15/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 9.4058e-05 - mae: 0.0071 - val_loss:
1.0222e-04 - val_mae: 0.0082
Epoch 16/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.1207e-04 - mae: 0.0081 - val_loss:
8.3926e-05 - val_mae: 0.0072
Epoch 17/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 9.1851e-05 - mae: 0.0071 - val_loss:
9.1115e-05 - val_mae: 0.0076
Epoch 18/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 9.5084e-05 - mae: 0.0073 - val_loss:
1.1781e-04 - val_mae: 0.0089
Epoch 19/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.1124e-04 - mae: 0.0081 - val_loss:
```

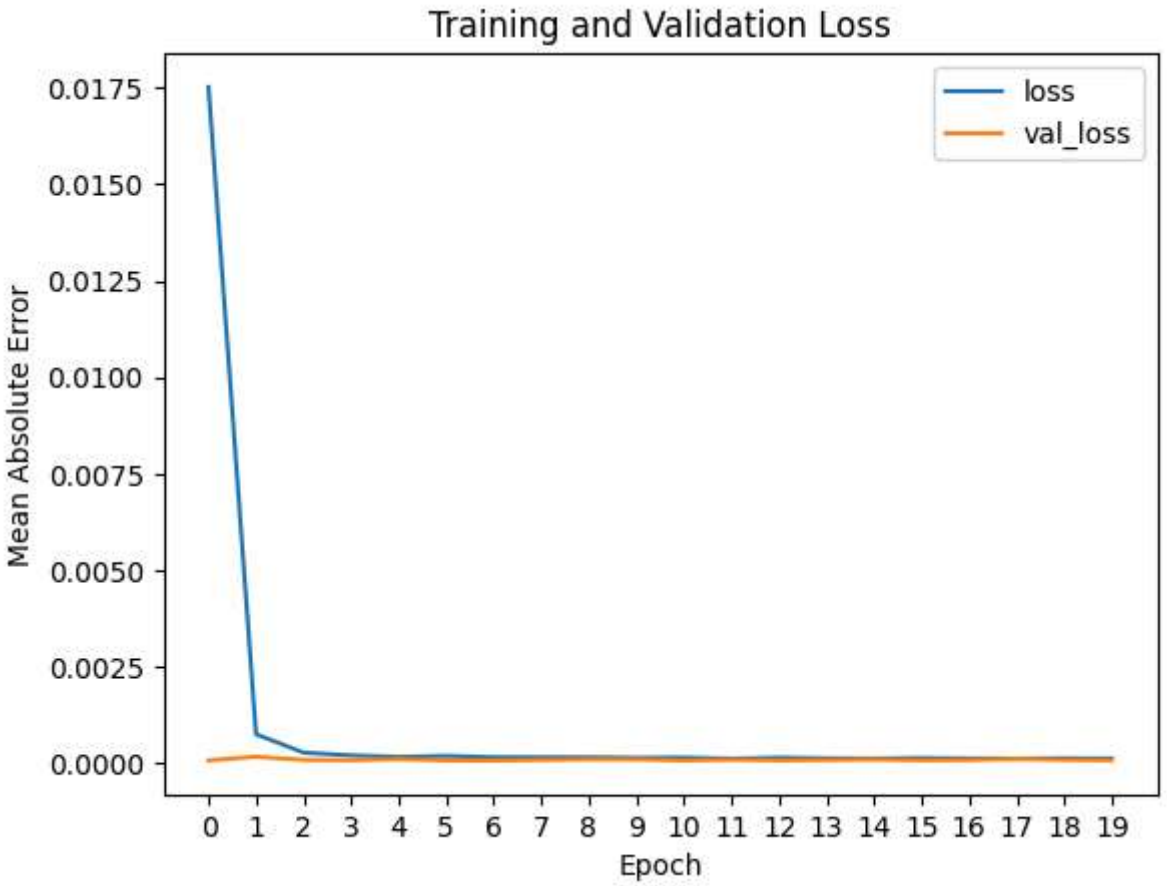
```
9.3806e-05 - val_mae: 0.0078
Epoch 20/20
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 9.0013e-05 - mae: 0.0070 - val_loss:
8.3500e-05 - val_mae: 0.0072
```

In [16]:
```
complex_model = load_model("Complex.keras")
print(f"Test MAE: {complex_model.evaluate(test_dataset)[1]:.3f}")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 7.5558e-05 - mae: 0.0064
Test MAE: 0.006
```

12. Using the stacked complex model, plot the MAE vs epoch for train and validation set.

In [17]:
```
complex_loss_df = pd.DataFrame(complex_history.history)
complex_loss_df[['loss', 'val_loss']].plot(legend=True)
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.xticks(range(20))
plt.show()
```



13. Which model is the winner? naive, RNN, LSTM or stacked?

In [18]:
```
print(f"Naive MAE: {evaluate_naive_method(test_dataset):.4f}")
print(f"RNN MAE: {rnn_model.evaluate(test_dataset)[1]:.4f}")
```

```
print(f"LSTM MAE: {lstm_model.evaluate(test_dataset)[1]:.4f}")
print(f"Complex MAE: {complex_model.evaluate(test_dataset)[1]:.4f}")
```

```
Naive MAE: 0.0083
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 9.5535e-05 - mae: 0.0074
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 9.5535e-05 - mae: 0.0074
RNN MAE: 0.0068
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 9.0007e-05 - mae: 0.0073
LSTM MAE: 0.0068
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 7.5558e-05 - mae: 0.0064
Complex MAE: 0.0059
```

13. (Answer) - The winner is the Complex Stacked model, with a MAE of `0.0059`, compared to the RNN and LSTM MAE of `0.0068` and the Naive MAE of `0.0083`.