Project on Specialized Models: Time Series and Survival Analysis

Main objective of the analysis

This project is on deep learning for time series. Its main objective development of a model for prediction daily average temperature.

My goal in this project was to use different architectures of deep learning models and see which one would give a higher accuracy.

In this project, I used 10 models:

1) simple DNN,

2) CNN,

3) simple RNN,

4) GRU,

5) LSTM,

6) bidirectional GRU,

7) bidirectional LSTM,

8) CNN + simple RNN,

9) CNN + GRU,

10) CNN + LSTM.

Brief description of the data set

In this project, I used the weather data sets with data for Birmingham, Alabama. This data set is from Carnegie Mellon University.

It contains the following information:

dates – in format YYYY-MM-DD

tmax – daily maximum temperature in °F

tmin – daily minimum temperature in °F

prcp – daily precipitation amount

It provides daily historical data and contains 52230 rows/examples, beginning with 1879-01-01 until 2021-12-31.

Brief summary of data exploration and actions taken for data cleaning or feature engineering

```
In [4]: df.shape
Out[4]: (52230, 4)
```

```
In [7]: df.head()
```

Out[7]:

	Date	tmax	tmin	prcp
0	1879-01-01	48.0	37.0	0.61
1	1879-01-02	46.0	21.0	0.02
2	1879-01-03	21.0	11.0	0.00
3	1879-01-04	25.0	9.0	0.00
4	1879-01-05	29.0	19.0	0.00

```
In [15]: df.tail()
```

Out[15]:

	Date	tmax	tmin	prcp
52225	2021-12-27	70.0	59.0	0.00
52226	2021-12-28	75.0	62.1	0.00
52227	2021-12-29	73.9	64.9	0.28
52228	2021-12-30	70.0	61.0	2.85
52229	2021-12-31	71.1	57.0	0.01

Some data was missing, so I used interpolation because it is important to keep all the rows/examples for time series.

```
In [16]: df['tmax'] = df['tmax'].interpolate()
         df['tmin'] = df['tmin'].interpolate()
         df['prcp'] = df['prcp'].interpolate()
```

Then, this data set contains only daily minimum and daily maximum temperatures, but for predicting daily temperature, it is more interesting to use daily mean temperature. So, I did a simple calculation and created a new column:

In [20]:
```
df['tmean'] = round(((df['tmax'] + df['tmin']) / 2), 1)
df['tmean'] = df['tmean'].interpolate()
```

In [21]:
```
df.head()
```

Out[21]:

	Date	tmax	tmin	prcp	tmean
0	1879-01-01	48.0	37.0	0.61	42.5
1	1879-01-02	46.0	21.0	0.02	33.5
2	1879-01-03	21.0	11.0	0.00	16.0
3	1879-01-04	25.0	9.0	0.00	17.0
4	1879-01-05	29.0	19.0	0.00	24.0

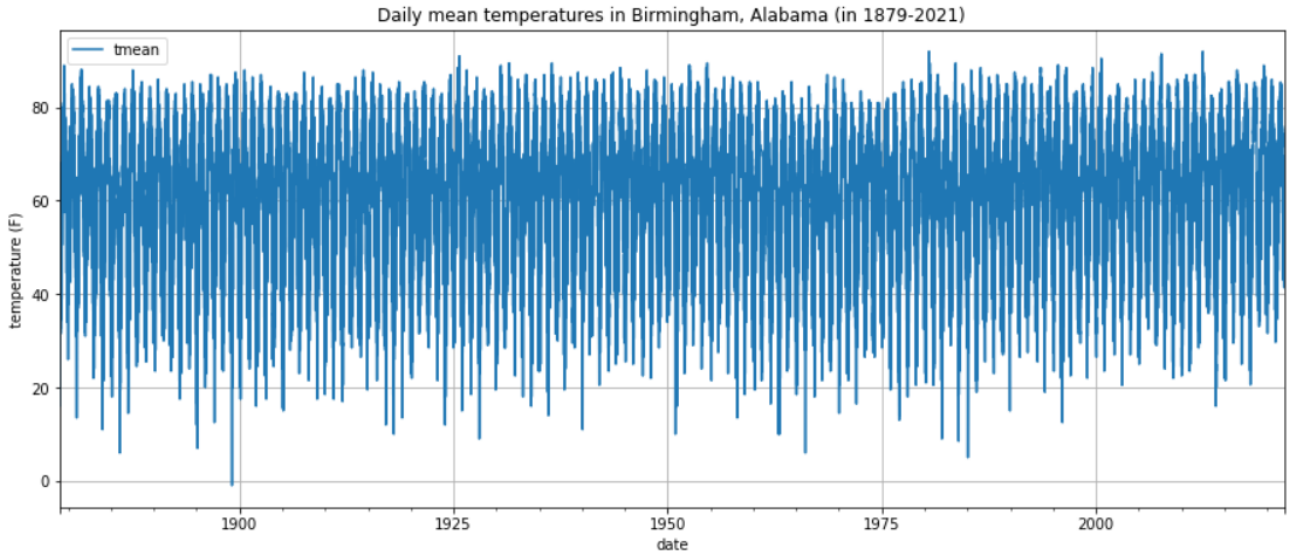
In [22]:
```
df.info()
```

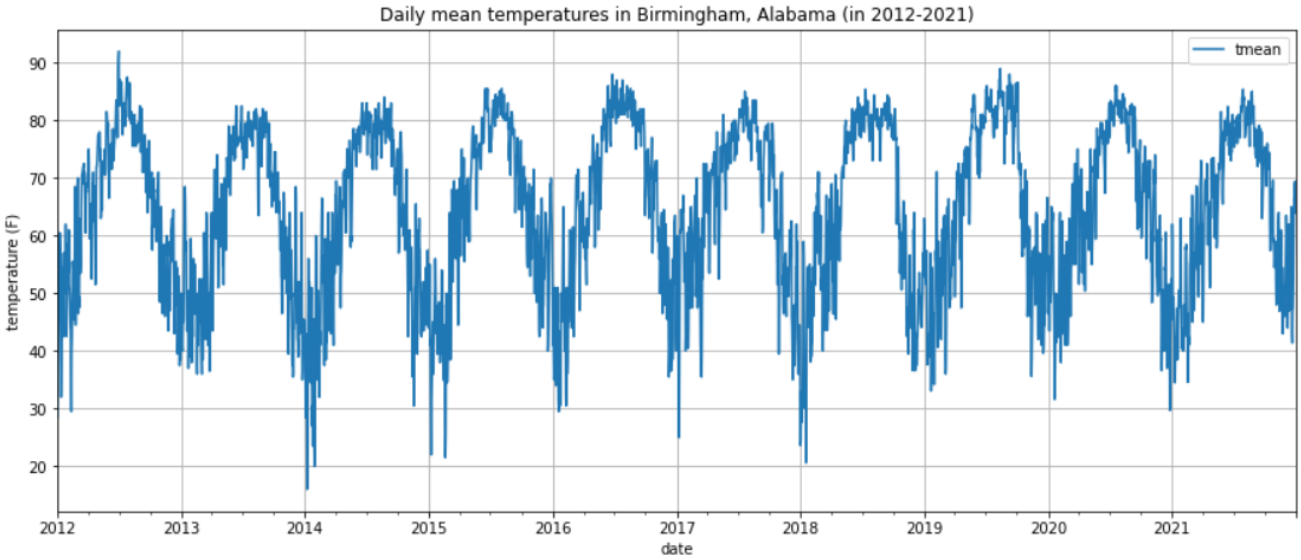
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52230 entries, 0 to 52229
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    52230 non-null  object 
 1   tmax    52230 non-null  float64
 2   tmin    52230 non-null  float64
 3   prcp    52230 non-null  float64
 4   tmean   52230 non-null  float64
dtypes: float64(4), object(1)
memory usage: 2.0+ MB
```

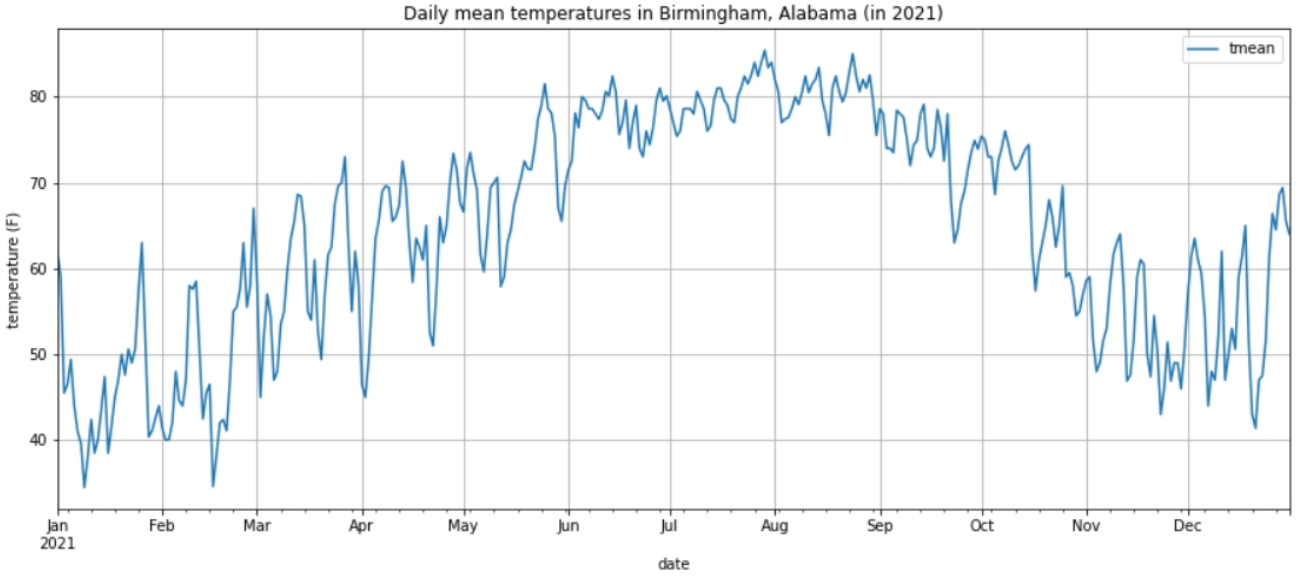
In [41]:
```
df.describe()
```

Out[41]:

	tmax	tmin	prcp	tmean
count	52230.000000	52230.000000	52230.000000	52230.000000
mean	71.292462	52.680492	0.134627	61.986506
std	15.521875	15.125850	0.373926	15.014825
min	7.000000	-9.000000	0.000000	-1.000000
25%	60.000000	40.000000	0.000000	50.000000
50%	73.000000	55.000000	0.000000	64.000000
75%	84.000000	67.000000	0.050000	75.500000
max	106.000000	82.000000	7.360000	92.000000




Daily mean temperatures in Birmingham, Alabama (in 2012-2021)
tmean
temperature (F)
90
80
70
60
50
40
30
20
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
date


Daily mean temperatures in Birmingham, Alabama (in 2021)
tmean
temperature (F)
80
70
60
50
40
Jan
2021
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
date

Summary of training of the Time Series & Deep Learning models

I built and trained 10 models:

1) simple DNN,

2) CNN,

3) simple RNN,

4) GRU,

5) LSTM,

6) bidirectional GRU,

7) bidirectional LSTM,

8) CNN + simple RNN,

9) CNN + GRU,

10) CNN + LSTM.

I used the same hyperparameters for training them in order to see which of them will perform best with the same hyperparameters.

In [8]:
```
window_size = 64
batch_size = 256
shuffle_buffer_size = 1000
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
print(train_set)
print(x_train.shape)
```

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=["mae"])
history = model.fit(train_set, epochs=100, verbose=1)
```

1) Simple DNN

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(20, input_shape=[None, 1], activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])
```

In [59]:
```
model_DNN.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, None, 20)          40
_________________________________________________________________
dense_4 (Dense)              (None, None, 10)          210
_________________________________________________________________
dense_5 (Dense)              (None, None, 1)           11
=================================================================
Total params: 261
Trainable params: 261
Non-trainable params: 0
_________________________________________________________________
```

2) CNN

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=128, kernel_size=3,
                           strides=1, padding="causal",
                           activation="relu",
                           input_shape=[None, 1]),
    tf.keras.layers.Dense(28, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1),
])
```

In [10]:
```
model_CNN.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, None, 128)         512
_________________________________________________________________
dense (Dense)                (None, None, 28)          3612
_________________________________________________________________
dense_1 (Dense)              (None, None, 10)          290
_________________________________________________________________
dense_2 (Dense)              (None, None, 1)           11
=================================================================
Total params: 4,425
Trainable params: 4,425
Non-trainable params: 0
_________________________________________________________________
```

3) Simple RNN

```
model = tf.keras.models.Sequential([
  tf.keras.layers.SimpleRNN(100, input_shape=[None, 1], return_sequences=True),
  tf.keras.layers.SimpleRNN(100, return_sequences=True, activation='relu'),
  tf.keras.layers.Dense(1)
])
```

```
In [16]: model_RNN.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, None, 100)         10200
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, None, 100)         20100
_________________________________________________________________
dense (Dense)                (None, None, 1)           101
=================================================================
Total params: 30,401
Trainable params: 30,401
Non-trainable params: 0
_________________________________________________________________
```

4) GRU

```
model = tf.keras.models.Sequential([
  tf.keras.layers.GRU(100, input_shape=[None, 1], return_sequences=True),
  tf.keras.layers.GRU(100),
  tf.keras.layers.Dense(1)
])
```

```
In [22]: model_GRU.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, None, 100)         30900
_________________________________________________________________
gru_1 (GRU)                  (None, 100)               60600
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 91,601
Trainable params: 91,601
Non-trainable params: 0
_________________________________________________________________
```

5) LSTM

```
model = tf.keras.models.Sequential([
  tf.keras.layers.LSTM(64, input_shape=[None, 1], return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1)
])
```

In [29]:
```
model_LSTM.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, None, 64)          16896
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 49,985
Trainable params: 49,985
Non-trainable params: 0
_________________________________________________________________
```

6) Bidirectional GRU

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Bidirectional(tf.keras.layers.GRU(100, input_shape=[None, 1], return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.GRU(100)),
  tf.keras.layers.Dense(1)
])
```

In [35]:
```
model_Bi_GRU.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, None, 200)         61800
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200)               181200
_________________________________________________________________
dense (Dense)                (None, 1)                 201
=================================================================
Total params: 243,201
Trainable params: 243,201
Non-trainable params: 0
```

7) Bidirectional LSTM

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, input_shape=[None, 1], return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
  tf.keras.layers.Dense(1)
])
```

```
In [41]: model_Bi_LSTM.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, None, 128)         33792
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               98816
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 132,737
Trainable params: 132,737
Non-trainable params: 0
_________________________________________________________________
```

8) CNN + simple RNN

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.SimpleRNN(100, input_shape=[None, 1], return_sequences=True),
  tf.keras.layers.SimpleRNN(100, return_sequences=True, activation='relu'),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1)
])
```

In [53]:
```
model_CNN_RNN.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, None, 32)          192
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, None, 100)         13300
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, None, 100)         20100
_________________________________________________________________
dense (Dense)                (None, None, 30)          3030
_________________________________________________________________
dense_1 (Dense)              (None, None, 10)          310
_________________________________________________________________
dense_2 (Dense)              (None, None, 1)           11
=================================================================
Total params: 36,943
Trainable params: 36,943
Non-trainable params: 0
_________________________________________________________________
```

9) CNN + GRU

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.GRU(100, input_shape=[None, 1], return_sequences=True),
  tf.keras.layers.GRU(100),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1)
])
```

In [101]:
```
model_CNN_GRU.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, None, 32)          192
_________________________________________________________________
gru (GRU)                    (None, None, 100)         40200
_________________________________________________________________
gru_1 (GRU)                  (None, 100)               60600
_________________________________________________________________
dense_3 (Dense)              (None, 30)                3030
_________________________________________________________________
dense_4 (Dense)              (None, 10)                310
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 11
=================================================================
Total params: 104,343
Trainable params: 104,343
Non-trainable params: 0
_________________________________________________________________
```

10) CNN + LSTM

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1)
])
```

```
In [79]: model_CNN_LSTM.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, None, 32)          192
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 64)          24832
_________________________________________________________________
lstm_3 (LSTM)                (None, None, 64)          33024
_________________________________________________________________
dense_3 (Dense)              (None, None, 30)          1950
_________________________________________________________________
dense_4 (Dense)              (None, None, 10)          310
_________________________________________________________________
dense_5 (Dense)              (None, None, 1)           11
=================================================================
Total params: 60,319
Trainable params: 60,319
Non-trainable params: 0
_________________________________________________________________
```

Evaluation

For evaluation of all the models I used mean absolute error.

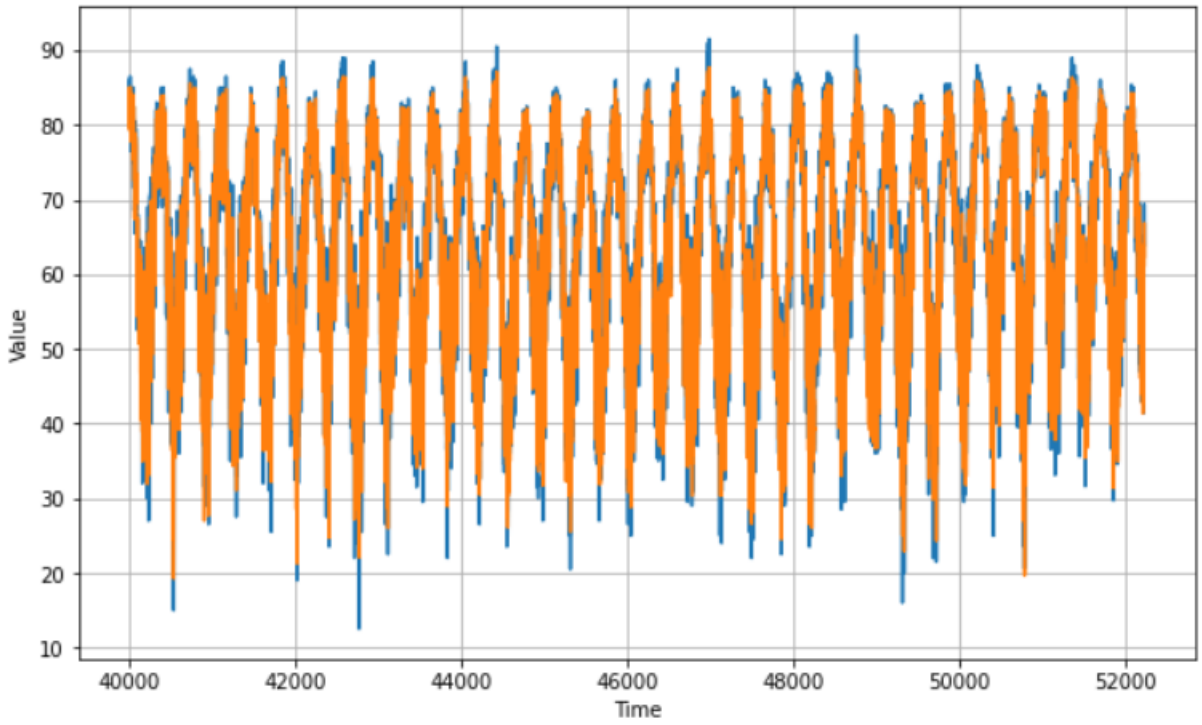
```
forecast = model_forecast(model, series[split_time - window_size: -1], window_size)[:,0]
```

```
tf.keras.metrics.mean_absolute_error(x_valid, forecast).numpy()
```

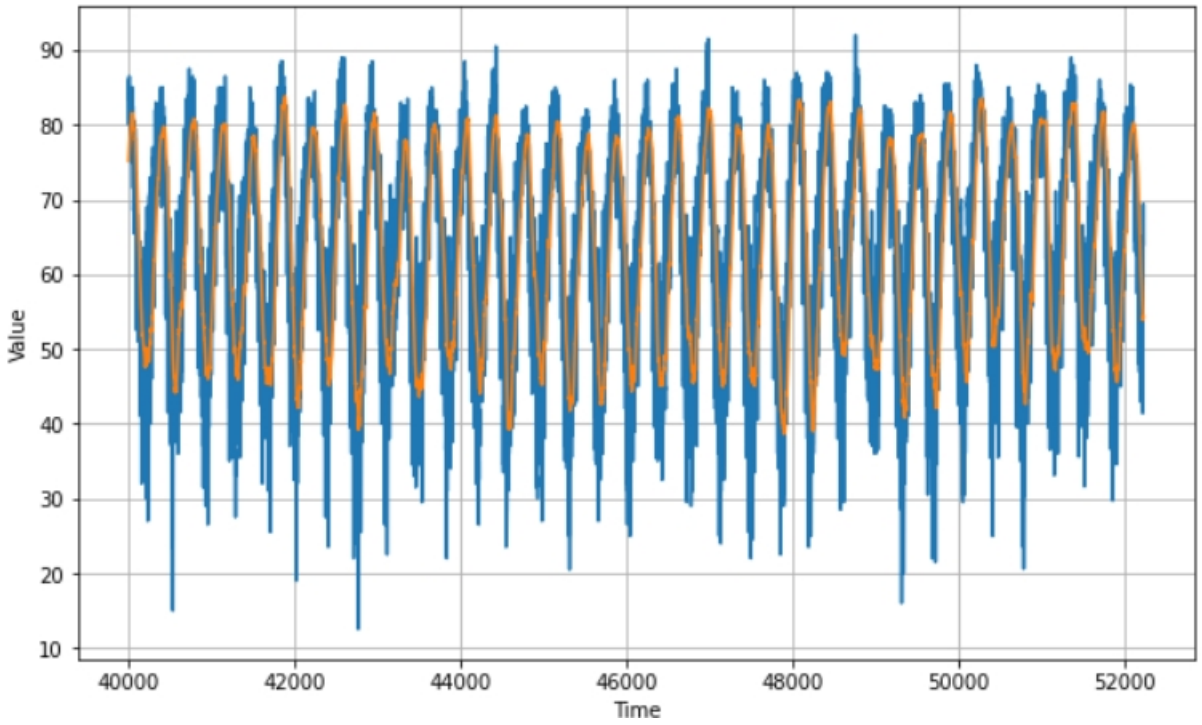
The screenshot of data set with the results of evaluation of the models is shown on the next page.

	Model	Mean Absolute Error
0	Simple DNN	3.716170
1	CNN	3.480151
2	Simple RNN	3.530616
3	GRU	7.725332
4	Bidirectional GRU	7.702015
5	LSTM	7.759338
6	Bidirectional LSTM	7.727791
7	CNN + Simple RNN	3.516199
8	CNN + GRU	7.746203
9	CNN + LSTM	3.293196

The plot for the best performing model (CNN + LSTM) looks like this:



For comparison, here is the plot for the worst performing model (LSTM):



In both plots, the actual values of the daily mean temperatures are blue and the predicted values are orange. These plots show that there is indeed a great difference in performance between these two models.

Recommended final model

The table of the mean absolute errors for all the models is shown again on the next page.

	Model	Mean Absolute Error
0	Simple DNN	3.716170
1	CNN	3.480151
2	Simple RNN	3.530616
3	GRU	7.725332
4	Bidirectional GRU	7.702015
5	LSTM	7.759338
6	Bidirectional LSTM	7.727791
7	CNN + Simple RNN	3.516199
8	CNN + GRU	7.746203
9	CNN + LSTM	3.293196

The CNN + LSTM model has the lowest mean absolute error. Its value is 3.29, that is, 3.29 degrees Fahrenheit, which is equivalent to 1.83 degrees Celsius. It is actually a good performance for such a simple model.

The second best model is the CNN model with mean absolute error = 3.48. This model is quite simple and is easier to train, but its performance is not much worse than the performance of the CNN + LSTM model.

However, since the CNN + LSTM model has the best performance and is still quite a simple model, it is the one that can be recommende.

Summary Key Findings and Insights

The goal of this project was to built and train a deep training model for time series which can be used for predicting daily average temperatures in Birmingham, Alabama.

I built and trained 10 models and then evaluated their performance. These 10 models are:

1) simple DNN,

2) CNN,

3) simple RNN,

4) GRU,

5) LSTM,

6) bidirectional GRU,

7) bidirectional LSTM,

8) CNN + simple RNN,

9) CNN + GRU,

10) CNN + LSTM.

I used the same hyperparameters for training all these models:

```
In [8]: window_size = 64
        batch_size = 256
        shuffle_buffer_size = 1000
        train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
        print(train_set)
        print(x_train.shape)
```

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=["mae"])
history = model.fit(train_set, epochs=100, verbose=1)
```

For evaluation, I used mean absolute error and got the following results:

	Model	Mean Absolute Error
0	Simple DNN	3.716170
1	CNN	3.480151
2	Simple RNN	3.530616
3	GRU	7.725332
4	Bidirectional GRU	7.702015
5	LSTM	7.759338
6	Bidirectional LSTM	7.727791
7	CNN + Simple RNN	3.516199
8	CNN + GRU	7.746203
9	CNN + LSTM	3.293196

The model that has the lowest mean absolute error is the CNN + LSTM model, which has mean absolute error = 3.29 degrees Fahrenheit or 1.83 degrees Celsius, which is quite a good result for such a simple model.

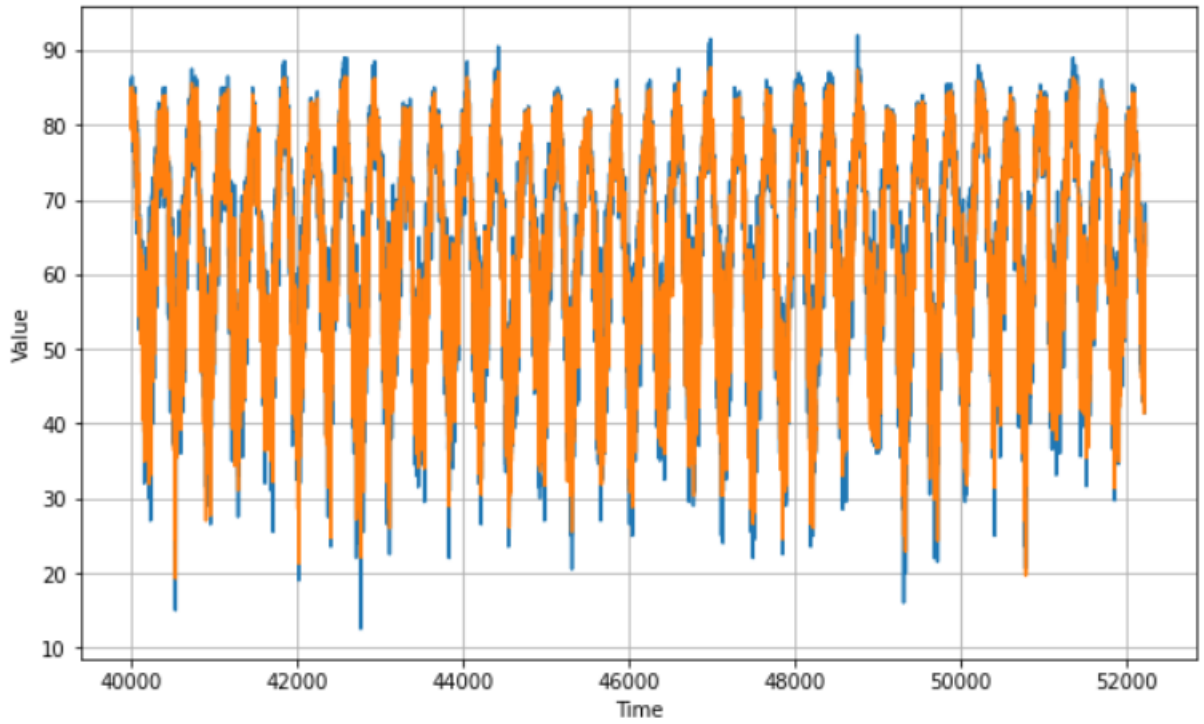
One interesting observation that can be made from the table of MAE of the models is that generally more simple models seem to have a higher accuracy than models that are more complicated. The table of the models sorted by increasing of their MAE looks like this:

Mean Absolute Error	Model
3.293196	CNN + LSTM
3.480151	CNN
3.516199	CNN + Simple RNN
3.530616	Simple RNN
3.716170	Simple DNN
7.702015	Bidirectional GRU
7.725332	GRU
7.727791	Bidirectional LSTM
7.746203	CNN + GRU
7.725332	GRU

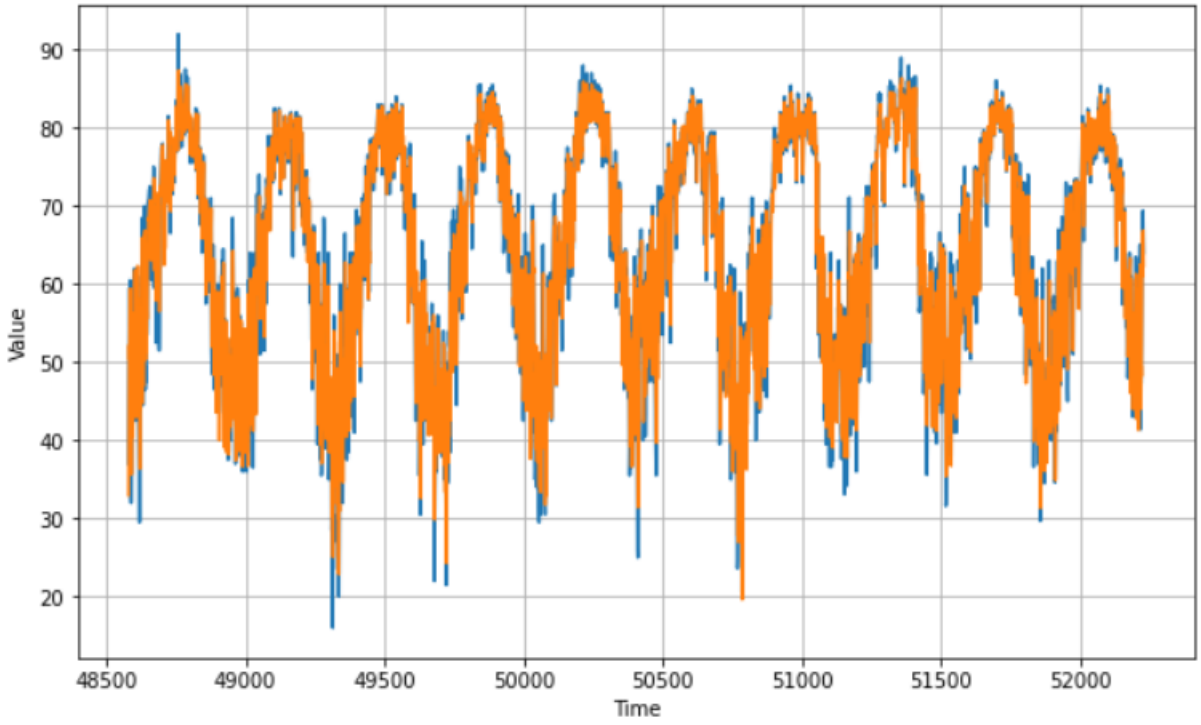
This is quite counter-intuitive. But it is possible that the result would be different if the models are trained with different hyperparameters or their architecture is changed.

The performance of the best model (CNN + LSTM) can be illustrated by the following diagrams where the actual values of the daily mean temperatures are shown with blue color and the predicted values are shown with orange color.

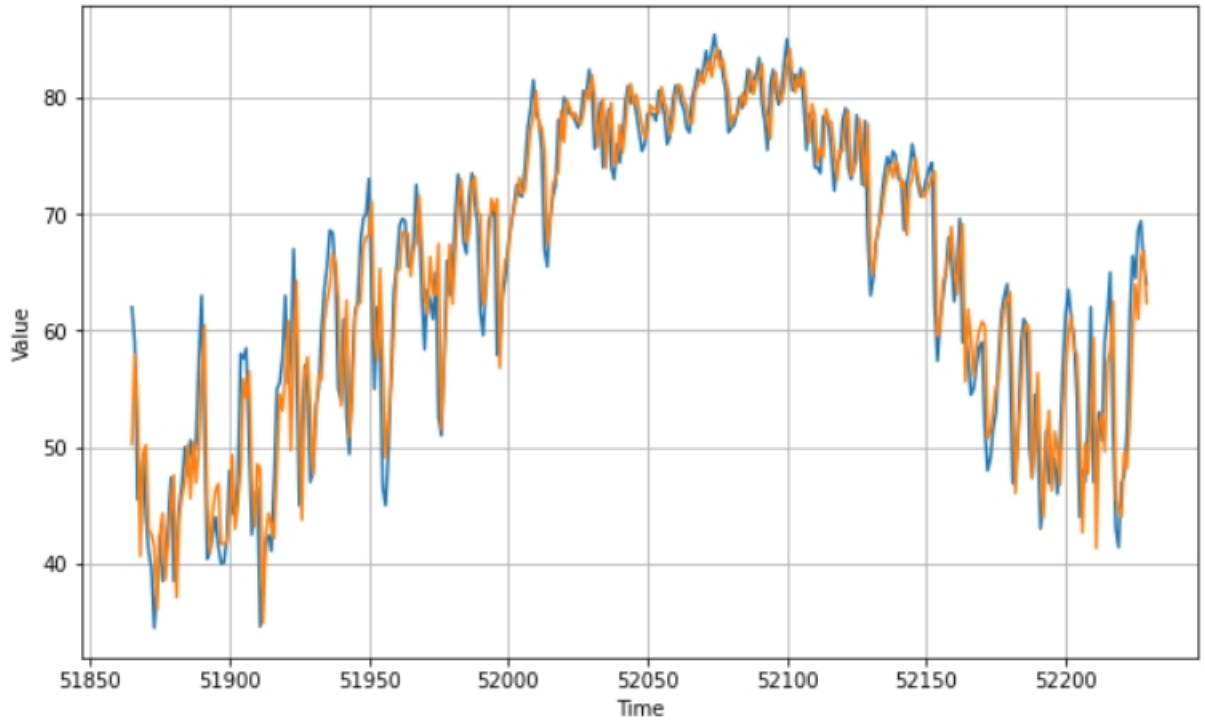
1) The plot for all the data from the test set shown on the next page.



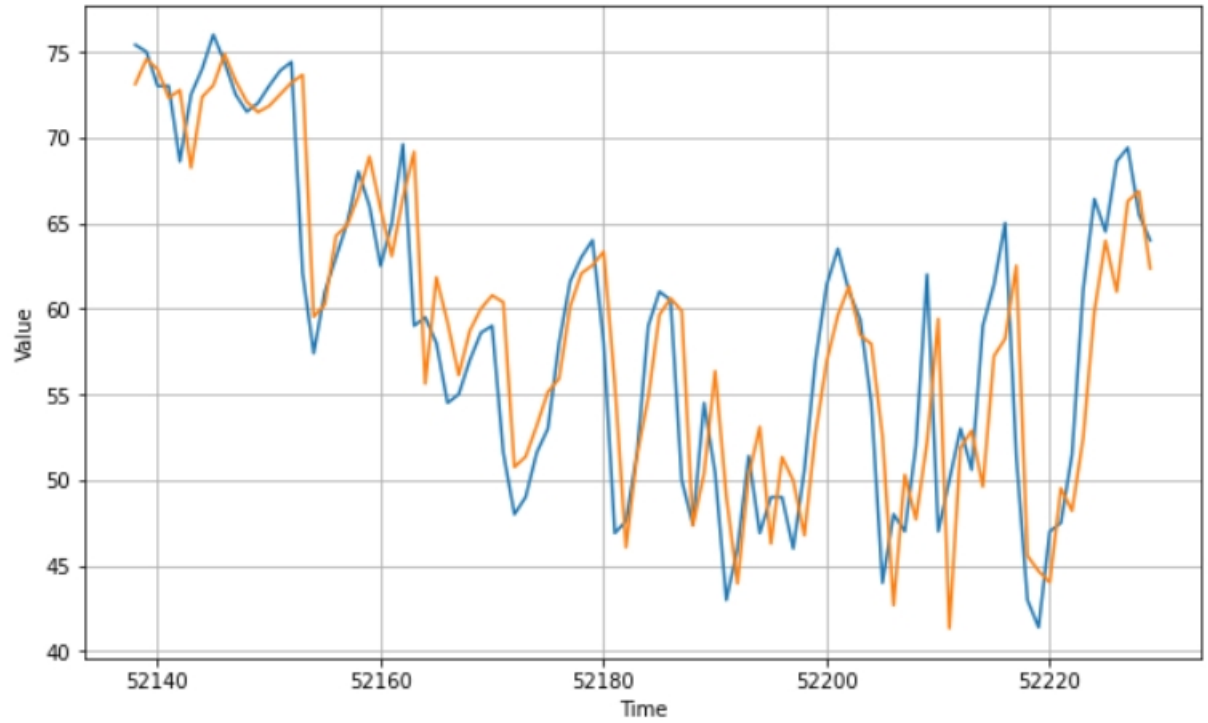
2) For the last 10 years (2012-2021):



3) For the year of 2021:



4) For October – December 2021:



The last diagram more clearly shows the difference between the actual values and the predicted ones. But the predicted values are quite close to the actual values.

Suggestions for next steps in analyzing this data

Although the CNN + LSTM model has quite a high accuracy, probably it is still possible to increase accuracy of this model. It can be achieved through tuning of hyperparameters and/or changing architecture of the model by adding more layers.

In this project, I did not tune hyperparameters. But probably better performance can be reached by increasing the number of epochs, increasing the window size, changing the batch size, changing the optimizer which is used, adding dropout, adding more layers to the neural network.

Since it looks quite counter-intuitive that more simple models tend to have better performance than more complex models, it would be good to analyze this phenomenon more. It is possible that this phenomenon occurs only for the hyperparameters that were chosen in this project, but with other hyperparameters performance of different models will change, and more simple models will have worse performance than more complex ones (which seems to be more intuitive).

The models that were built and trained in this project can predict daily mean temperatures only for one city, Birmingham, Alabama. For other cities, they need to be retrained with data sets of actual historical temperatures for those cities. However, probably, a model trained for one city can be used as a pre-trained model for transfer learning for other cities. Since the models built and trained in this project are quite simple and can easily and quickly be trained again with new data, there is not much sense to use them for transfer learning. But in case of creating more complex models with many layers, it may make sense.

In this project, I used daily mean temperatures for training the models. I calculated the daily mean temperatures before training the models because the original data set contained only daily minimum and maximum temperatures. But it might be good to try to use the original daily minimum and maximum temperatures together to train the models. Then, the daily mean temperatures can be used for evaluation of the model. And then the accuracy of the models trained this way can be compared to the accuracy of the models trained with the daily mean temperatures to see whether it can improve the model performance because it seems that this way might indeed improve their performance.

Also, since the data set which was used in this project ended with data for last year, it might be good in future to revisit the models and retrained them with updated data.