```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'daily-temperature-of-major-cities:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F694560%2F1215964%2Fbu

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading daily-temperature-of-major-cities, 13523007 bytes compressed
[==================================================] 13523007 bytes downloaded
Downloaded and uncompressed: daily-temperature-of-major-cities
Data source import complete.
```

Introduction

In this notebook, I am creating a tensorflow based timeseries forecasting model using CNN & LSTM.

Acknowledgments:

This notebook is inspired by the course 4 of TensorFlow in Practice Specialization which is Sequences, Time Series and Prediction by Laurence Moroney.

I used this course to prepare for the tensorflow speciality examination, and I am using the methods and codes that Mr. Moroney used in the course.

Sections:

If you find this notebook helpful for you, please upvote!

[ ] ↳ *2 cells hidden*

Importing and exploring the dataset

```
data = pd.read_csv("/kaggle/input/daily-temperature-of-major-cities/city_temperature.csv")
data.head()
```

	Region	Country	State	City	Month	Day	Year	AvgTemperature
0	Africa	Algeria	NaN	Algiers	1	1	1995	64.2
1	Africa	Algeria	NaN	Algiers	1	2	1995	49.4
2	Africa	Algeria	NaN	Algiers	1	3	1995	48.8
3	Africa	Algeria	NaN	Algiers	1	4	1995	46.4
4	Africa	Algeria	NaN	Algiers	1	5	1995	47.9

Checking if all the cities has the data for a full range

```
data['City'].value_counts()
```

```
     Springfield      18530
     Columbus         18530
     Portland         18530
     Washington DC    18530
     Washington       18530
                      ...
     Frankfurt         4136
     Flagstaff         3574
     Pristina          3427
     Yerevan           3226
     Bonn              3133
     Name: City, Length: 321, dtype: int64
```

I wanted to develop a timeseries model for a single city. For this purpouse, I am taking the city Chennai (previously known as Madras), from Tamil Nadu, India. The city where I reside.

Chennai generally has only two season. It is hot for almost throughout the year, and rains in November/December months.

```
chennai = data[data["City"] == "Chennai (Madras)"]
chennai.head()
```

	Region	Country	State	City	Month	Day	Year	AvgTemperature
331055	Asia	India	NaN	Chennai (Madras)	1	1	1995	72.4
331056	Asia	India	NaN	Chennai (Madras)	1	2	1995	73.5
331057	Asia	India	NaN	Chennai (Madras)	1	3	1995	72.6
				Chennai				

Checking if all the year has complete records

```
chennai["Year"].value_counts()

     2008    366
     2000    366
     2016    366
     2015    366
     2004    366
            ...
     2001    365
     1999    365
     1998    365
     1997    365
     2020    134
     Name: Year, Length: 26, dtype: int64
```

Imputing missing values

The dataset has recorded missing values with the number -99. The chennai dataset has missing values close to 29 records.

I will use forward fill method to impute the missing values for the dataset. That is, we will take the previously non missing value and fill it in the place of the missing value.

First replacing -99 with np.nan

```
"""-99 is put in place of missing values.
We will have to forward fill with the last non missing value before -99
"""
chennai["AvgTemperature"] = np.where(chennai["AvgTemperature"] == -99, np.nan, chennai["AvgTemperature"])
chennai.isnull().sum()

     Region               0
     Country              0
     State             9266
     City                 0
     Month                0
     Day                  0
     Year                 0
     AvgTemperature      29
     dtype: int64
```

Now using ffill() method to fill the np.nan that we created

```
chennai["AvgTemperature"] = chennai["AvgTemperature"].ffill()
chennai.isnull().sum()

     Region               0
     Country              0
     State             9266
     City                 0
     Month                0
     Day                  0
     Year                 0
     AvgTemperature       0
     dtype: int64
```

Since there is no single column that contains the date, creating a new column called Time_steps to combine the year month and date fields

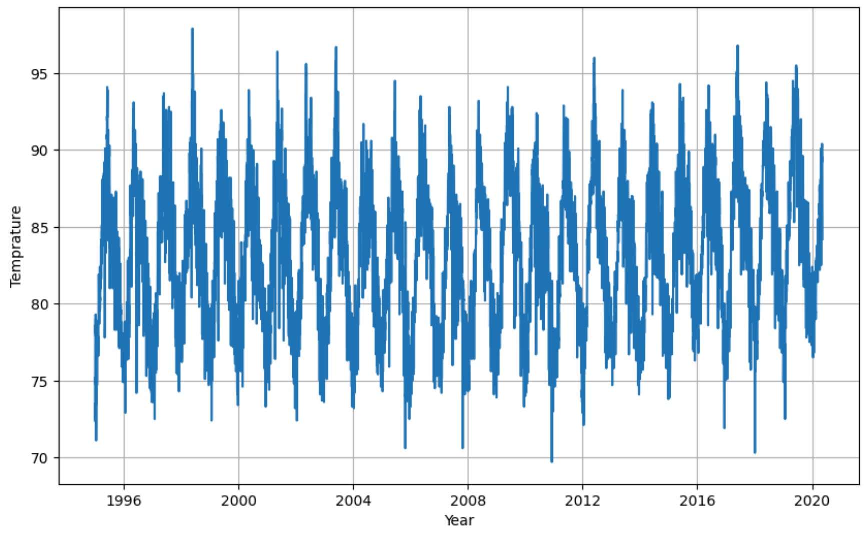
```
chennai.dtypes
chennai["Time_steps"] = pd.to_datetime((chennai.Year*10000 + chennai.Month*100 + chennai.Day).apply(str),format='%Y%m%d')
chennai.head()
```

	Region	Country	State	City	Month	Day	Year	AvgTemperature	Time_steps
331055	Asia	India	NaN	Chennai (Madras)	1	1	1995	72.4	1995-01-01
331056	Asia	India	NaN	Chennai (Madras)	1	2	1995	73.5	1995-01-02
331057	Asia	India	NaN	Chennai (Madras)	1	3	1995	72.6	1995-01-03

```
def plot_series(time, series, format="-", start=0, end=None):
    """to plot the series"""
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Year")
    plt.ylabel("Temprature")
    plt.grid(True)
```

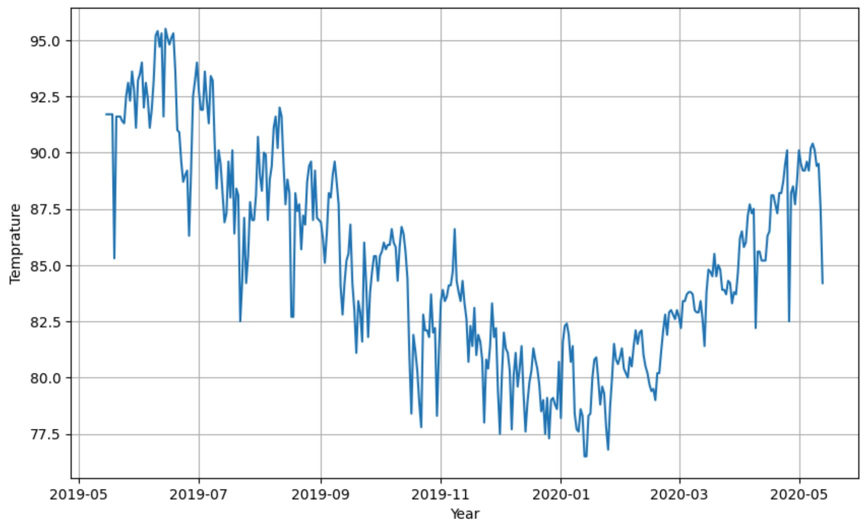
Plotting the timeseries for the entire duration

```
time_step = chennai["Time_steps"].tolist()
temprature = chennai["AvgTemperature"].tolist()

series = np.array(temprature)
time = np.array(time_step)
plt.figure(figsize=(10, 6))
plot_series(time, series)
```



Plotting for recent one year only

```
plt.figure(figsize=(10, 6))
plot_series(time[-365:], series[-365:])
```



There are totally 9,266 records on the dataset. We will keep 8000 records for training (85%) and keep remaining 15% for testing

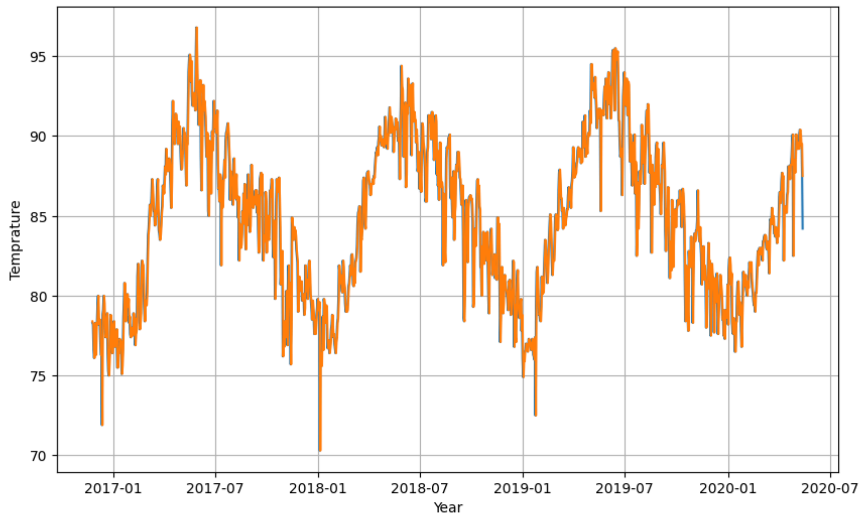
```
split_time = 8000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]
```

Naive forecast

In naive forecast, we will take the record in month - 1 (the month previously) and assume that it will be carried forward for the next observation also.

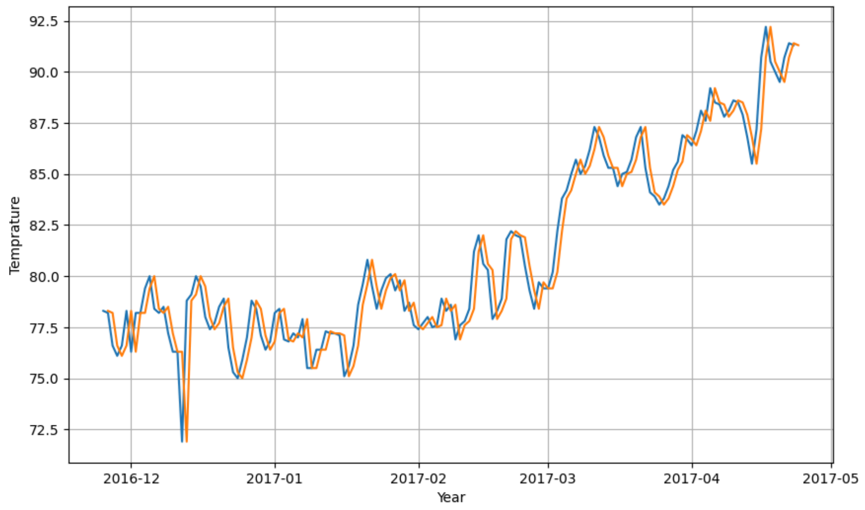
```
naive_forecast = series[split_time - 1:-1]
```

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, naive_forecast)
```



Since the plot above is so crowded, we will take for a small section of the dataset and visualize it.

```
#Zoom in and see only few points
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid, start=0, end=150)
plot_series(time_valid, naive_forecast, start=1, end=151)
```



```
print(tf.keras.metrics.mean_squared_error(x_valid, naive_forecast).numpy())
print(tf.keras.metrics.mean_absolute_error(x_valid, naive_forecast).numpy())
```

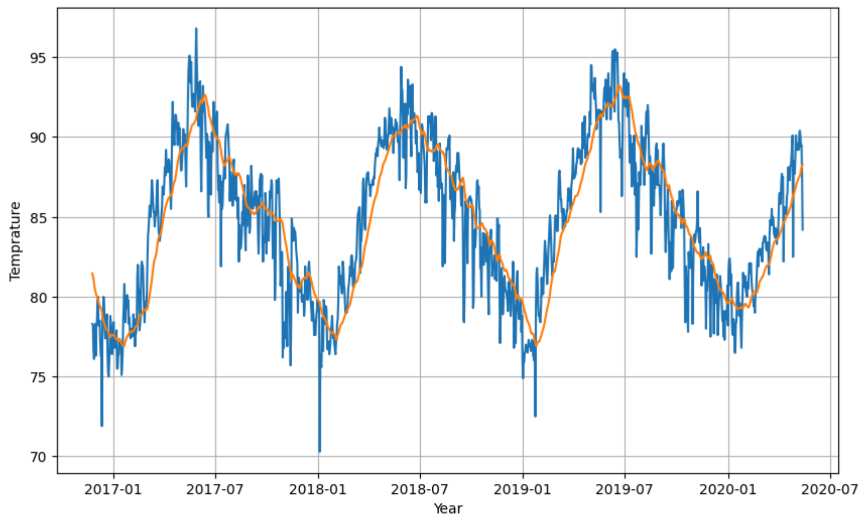
```
     2.447661927330174
     1.0900473933649284
```

Moving average forecast

In moving average forecast, we will take the value of average for the previous window period and take it as the prediction for the next period.

```
def moving_average_forecast(series, window_size):
    """Forecasts the mean of the last few values.
     If window_size=1, then this is equivalent to naive forecast"""
    forecast = []
    for time in range(len(series) - window_size):
        forecast.append(series[time:time + window_size].mean())
    return np.array(forecast)
```

```
moving_avg = moving_average_forecast(series, 30)[split_time - 30:]

plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, moving_avg)
```



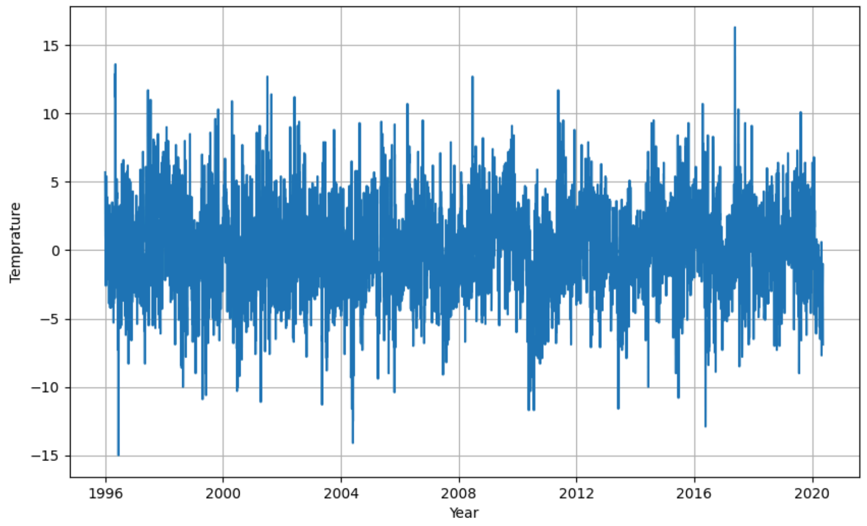
```
print(tf.keras.metrics.mean_squared_error(x_valid, moving_avg).numpy())
print(tf.keras.metrics.mean_absolute_error(x_valid, moving_avg).numpy())
```

```
5.45876674565561
1.841850974196946
```

Differencing

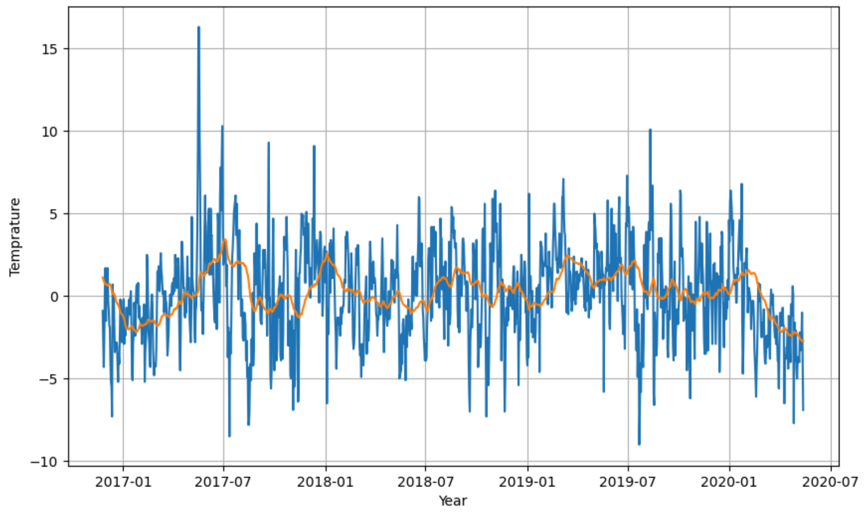
We will use a technique called differencing to remove the trend and seasonality from the data. Here we difference the data between what the value was 365 days (1 year back). The differencing should always follow the seasonal pattern.

```
diff_series = (series[365:] - series[:-365])
diff_time = time[365:]

plt.figure(figsize=(10, 6))
plot_series(diff_time, diff_series)
plt.show()
```



```
diff_moving_avg = moving_average_forecast(diff_series, 50)[split_time - 365 - 50:]

plt.figure(figsize=(10, 6))
```

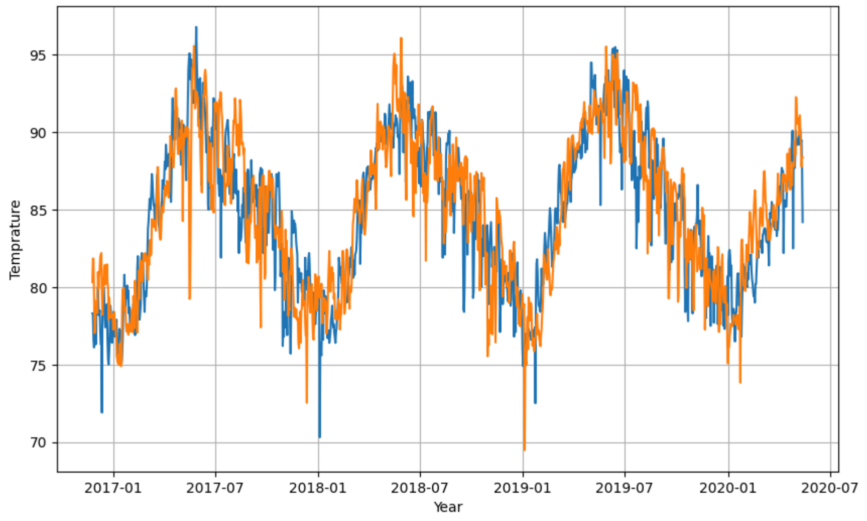
```
plot_series(time_valid, diff_series[split_time - 365:])
plot_series(time_valid, diff_moving_avg)
plt.show()
```



Restoring trend and seasonality

But these are just the forecast of the differenced timeseries. To get the value for the original timeseries, we have to add back the value of t-365

```
diff_moving_avg_plus_past = series[split_time - 365:-365] + diff_moving_avg

plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, diff_moving_avg_plus_past)
plt.show()
```

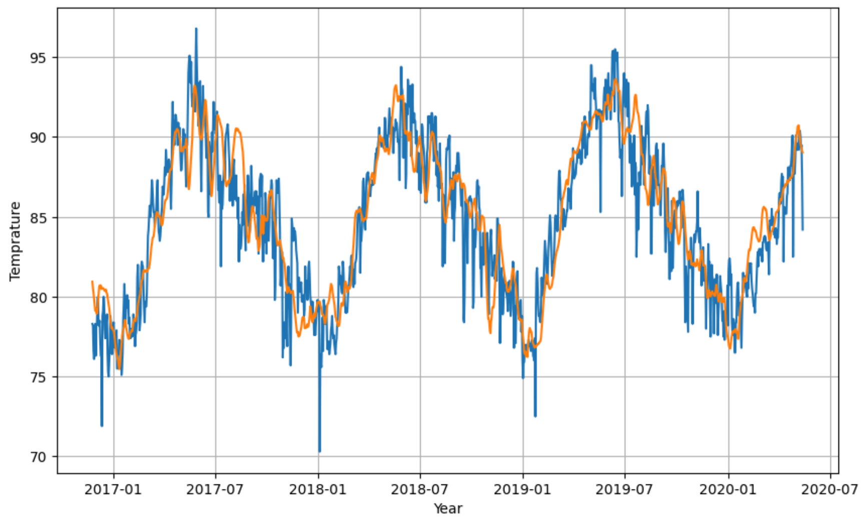


```
print(tf.keras.metrics.mean_squared_error(x_valid, diff_moving_avg_plus_past).numpy())
print(tf.keras.metrics.mean_absolute_error(x_valid, diff_moving_avg_plus_past).numpy())

     7.969171658767773
     2.1771153238546606
```

Smoothing with moving average again

The above plot has a lot of noise. To smooth it again, we do a moving average on that

```
diff_moving_avg_plus_smooth_past = moving_average_forecast(series[split_time - 370:-360], 10) + diff_moving_avg

plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, diff_moving_avg_plus_smooth_past)
plt.show()
```



```
print(tf.keras.metrics.mean_squared_error(x_valid, diff_moving_avg_plus_smooth_past).numpy())
print(tf.keras.metrics.mean_absolute_error(x_valid, diff_moving_avg_plus_smooth_past).numpy())
```

```
5.630983364928909
1.8089115323854659
```

How to prepare a window dataset?

A window dataset is used in the dataset prepration of the tensorflow. It yields a prefetched dataset with the x and y variables as tensors.

Step 1: Converting the numpy array into a tensor using tensor_slices

```
series1 = tf.expand_dims(series, axis=-1)
ds = tf.data.Dataset.from_tensor_slices(series1[:20])
for val in ds:
    print(val.numpy())
```

```
[72.4]
[73.5]
[72.6]
[75.2]
[74.8]
[76.4]
[78.4]
[78.6]
[78.1]
[79.3]
[77.9]
[79.]
[73.4]
[76.7]
[73.7]
[77.]
[71.1]
[72.6]
[76.1]
[75.7]
```

Step 2: tf window option groups 5 (window size) into a single line

But for the last observations for which there are no observations to group will be kept as remaining as in the outupt of this cell

```
dataset = ds.window(5, shift=1)
for window_dataset in dataset:
    for val in window_dataset:
        print(val.numpy(), end=" ")
    print()
```

```
[72.4] [73.5] [72.6] [75.2] [74.8]
[73.5] [72.6] [75.2] [74.8] [76.4]
[72.6] [75.2] [74.8] [76.4] [78.4]
[75.2] [74.8] [76.4] [78.4] [78.6]
[74.8] [76.4] [78.4] [78.6] [78.1]
[76.4] [78.4] [78.6] [78.1] [79.3]
[78.4] [78.6] [78.1] [79.3] [77.9]
[78.6] [78.1] [79.3] [77.9] [79.]
[78.1] [79.3] [77.9] [79.] [73.4]
[79.3] [77.9] [79.] [73.4] [76.7]
[77.9] [79.] [73.4] [76.7] [73.7]
[79.] [73.4] [76.7] [73.7] [77.]
[73.4] [76.7] [73.7] [77.] [71.1]
[76.7] [73.7] [77.] [71.1] [72.6]
[73.7] [77.] [71.1] [72.6] [76.1]
[77.] [71.1] [72.6] [76.1] [75.7]
[71.1] [72.6] [76.1] [75.7]
[72.6] [76.1] [75.7]
[76.1] [75.7]
[75.7]
```

Step 3: Drop reminder set to True will drop the variables which are not having the grouping

```
dataset = ds.window(5, shift=1, drop_remainder=True)
for window_dataset in dataset:
    for val in window_dataset:
        print(val.numpy(), end=" ")
    print()
```

```
[72.4] [73.5] [72.6] [75.2] [74.8]
[73.5] [72.6] [75.2] [74.8] [76.4]
[72.6] [75.2] [74.8] [76.4] [78.4]
[75.2] [74.8] [76.4] [78.4] [78.6]
[74.8] [76.4] [78.4] [78.6] [78.1]
[76.4] [78.4] [78.6] [78.1] [79.3]
[78.4] [78.6] [78.1] [79.3] [77.9]
[78.6] [78.1] [79.3] [77.9] [79.]
[78.1] [79.3] [77.9] [79.] [73.4]
[79.3] [77.9] [79.] [73.4] [76.7]
[77.9] [79.] [73.4] [76.7] [73.7]
[79.] [73.4] [76.7] [73.7] [77.]
[73.4] [76.7] [73.7] [77.] [71.1]
[76.7] [73.7] [77.] [71.1] [72.6]
[73.7] [77.] [71.1] [72.6] [76.1]
[77.] [71.1] [72.6] [76.1] [75.7]
```

Step 4: flat map option will group the 5 observation in a single tensor variable

```
dataset = ds.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
for window in dataset:
    print(window.numpy())
```

```
  [77.9]
  [79. ]]
 [[78.1]
  [79.3]
  [77.9]
  [79. ]
  [73.4]]
 [[79.3]
  [77.9]
  [79. ]
  [73.4]
  [76.7]]
 [[77.9]
  [79. ]
  [73.4]
  [76.7]
  [73.7]]
 [[79. ]
  [73.4]
  [76.7]
  [73.7]
  [77. ]]
 [[73.4]
  [76.7]
  [73.7]
  [77. ]
  [71.1]]
 [[76.7]
  [73.7]
  [77. ]
  [71.1]
  [72.6]]
 [[73.7]
  [77. ]
  [71.1]
  [72.6]
  [76.1]]
 [[77. ]
  [71.1]
  [72.6]
  [76.1]
```

Step 5: map option will split the variables into X and y variables

```
dataset = ds.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
for x,y in dataset:
    print(x.numpy(), y.numpy())
```

```
  [75.2]
  [74.8]] [[76.4]]
 [[72.6]
  [75.2]
  [74.8]
  [76.4]] [[78.4]]
 [[75.2]
  [74.8]
  [76.4]
  [78.4]] [[78.6]]
 [[74.8]
  [76.4]
  [78.4]
  [78.6]] [[78.1]]
 [[76.4]
  [78.4]
  [78.6]
  [78.1]] [[79.3]]
 [[78.4]
  [78.6]
  [78.1]
  [79.3]] [[77.9]]
 [[78.6]
  [78.1]
  [79.3]
  [77.9]] [[79.]]
 [[78.1]
  [79.3]
  [77.9]
  [79. ]] [[73.4]]
  [73.4]
  [76.7]] [[73.7]]
 [[79. ]
  [73.4]
  [76.7]
  [73.7]] [[77.]]
 [[73.4]
  [76.7]
  [73.7]
  [77. ]] [[71.1]]
 [[76.7]
  [73.7]
  [77. ]
  [71.1]] [[72.6]]
 [[73.7]
  [77. ]
  [71.1]
  [72.6]] [[76.1]]
 [[77. ]
  [71.1]
  [72.6]
  [76.1]] [[75.7]]
```

Step 6: shuffle option will shuffle the dataset into random order.

Till the previous step, the observation would have been in the correct order. the shuffle will ensure that the data are randomly mixed up

```
dataset = ds.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
dataset = dataset.shuffle(buffer_size=10)
for x,y in dataset:
    print(x.numpy(), y.numpy())
```

```
 [73.4]
 [76.7]] [[73.7]]
[[78.4]
 [78.6]
 [78.1]
 [79.3]] [[77.9]]
[[73.5]
 [72.6]
 [75.2]
 [74.8]] [[76.4]]
[[73.4]
 [76.7]
 [73.7]
 [77. ]] [[71.1]]
[[72.6]
 [75.2]
 [74.8]
 [76.4]] [[78.4]]
[[72.4]
 [73.5]
 [72.6]
 [75.2]] [[74.8]]
[[76.7]
 [73.7]
 [77. ]
 [71.1]] [[72.6]]
[[74.8]
 [76.4]
 [78.4]
 [78.6]] [[78.1]]
[[79.3]
 [77.9]
 [79. ]
 [73.4]] [[76.7]]
[[76.4]
 [78.4]
 [78.6]
 [78.1]] [[79.3]]
[[78.1]
 [79.3]
 [77.9]
 [79. ]] [[73.4]]
[[75.2]
 [74.8]
 [76.4]
 [78.4]] [[78.6]]
[[77. ]
```

```
  [73.7]] [[77.]]
 [[73.7]
  [77. ]
  [71.1]
  [72.6]] [[76.1]]
```

Step 7: Batch option will put the variables into mini-batches suitable for training. It will group both X and y into mini batches

```
dataset = ds.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
dataset = dataset.shuffle(buffer_size=10)
dataset = dataset.batch(2).prefetch(1)
for x,y in dataset:
    print("x = ", x.numpy())
    print("y = ", y.numpy())
    print("*"*25)
```

```
   [78.6]
   [78.1]]]
 y =  [[[78.1]]

  [[79.3]]]
 *************************
 x =  [[[77. ]
   [71.1]
   [72.6]
   [76.1]]

  [[78.1]
   [79.3]
   [77.9]
   [79. ]]]
 y =  [[[75.7]]

  [[73.4]]]
 *************************
 x =  [[[79. ]
   [73.4]
   [76.7]
   [73.7]]

  [[77.9]
   [79. ]
   [73.4]
   [76.7]]]
 y =  [[[77. ]]

  [[73.7]]]
 *************************
 x =  [[[73.7]
   [77. ]
   [71.1]
   [72.6]]

  [[73.4]
   [76.7]
   [73.7]
   [77. ]]]
 y =  [[[76.1]]

  [[71.1]]]
 *************************
 x =  [[[73.5]
   [72.6]
   [75.2]
   [74.8]]

  [[79.3]
   [77.9]
   [79. ]
   [73.4]]]
 y =  [[[76.4]]

  [[76.7]]]
 *************************
```

Window size is how many observations in the past do you want to see before making a prediction. Batch size is similar to mini-batches set while training the neural network

```
window_size = 60
batch_size = 32
shuffle_buffer_size = 1000
```

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    """
    To create a window dataset given a numpy as input

    Returns: A prefetched tensorflow dataset
    """
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[1:]))
    return ds.batch(batch_size).prefetch(1)
```

Finding the correct learning rate

Using a call back for LearningRateScheduler(). For every epoch this just changes the learning rate a little so that the learning rate varies from 1e-8 to 1e-6

Also a new loss function Huber() is introduced which is less sensitive to outliers.

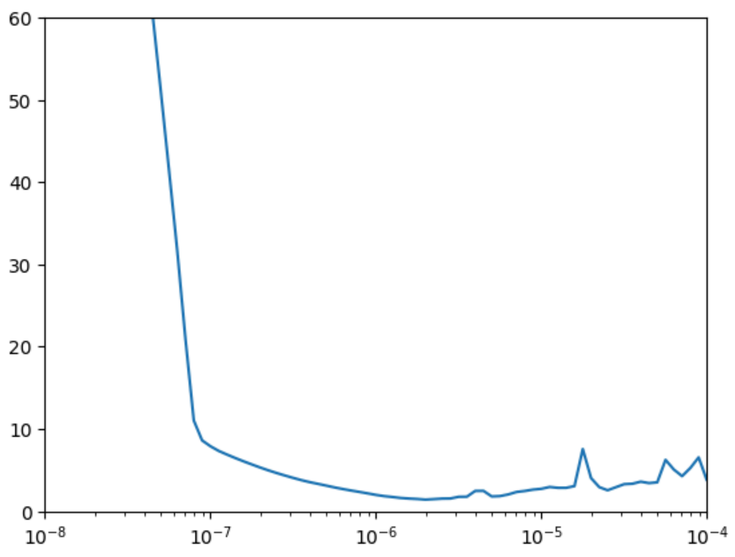
```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
window_size = 64
batch_size = 256
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
print(train_set)
print(x_train.shape)

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),
  tf.keras.layers.Lambda(lambda x: x * 400)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))

optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

```
    31/31 [==============================] - 12s 370ms/step - loss: 18.4383 - mae: 18.9371 - lr: 1.9953e-04
    Epoch 88/100
    31/31 [==============================] - 12s 382ms/step - loss: 32.0735 - mae: 32.5695 - lr: 2.2387e-04
    Epoch 89/100
    31/31 [==============================] - 12s 381ms/step - loss: 33.0535 - mae: 33.5506 - lr: 2.5119e-04
    Epoch 90/100
    31/31 [==============================] - 12s 385ms/step - loss: 43.5198 - mae: 44.0178 - lr: 2.8184e-04
    Epoch 91/100
    31/31 [==============================] - 12s 387ms/step - loss: 45.9228 - mae: 46.4222 - lr: 3.1623e-04
    Epoch 92/100
    31/31 [==============================] - 12s 383ms/step - loss: 40.3438 - mae: 40.8411 - lr: 3.5481e-04
    Epoch 93/100
    31/31 [==============================] - 12s 351ms/step - loss: 66.6575 - mae: 67.1538 - lr: 3.9811e-04
    Epoch 94/100
    31/31 [==============================] - 12s 368ms/step - loss: 58.6790 - mae: 59.1750 - lr: 4.4668e-04
    Epoch 95/100
    31/31 [==============================] - 12s 386ms/step - loss: 39.0622 - mae: 39.5581 - lr: 5.0119e-04
    Epoch 96/100
    31/31 [==============================] - 12s 376ms/step - loss: 59.9971 - mae: 60.4953 - lr: 5.6234e-04
    Epoch 97/100
    31/31 [==============================] - 12s 371ms/step - loss: 68.1158 - mae: 68.6146 - lr: 6.3096e-04
    Epoch 98/100
    31/31 [==============================] - 12s 377ms/step - loss: 129.6006 - mae: 130.1005 - lr: 7.0795e-04
    Epoch 99/100
    31/31 [==============================] - 12s 378ms/step - loss: 83.3589 - mae: 83.8573 - lr: 7.9433e-04
    Epoch 100/100
    31/31 [==============================] - 12s 378ms/step - loss: 199.4189 - mae: 199.9180 - lr: 8.9125e-04
```

We plot this on a semilog axis

```
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 60])
```

```
    (1e-08, 0.0001, 0.0, 60.0)
```



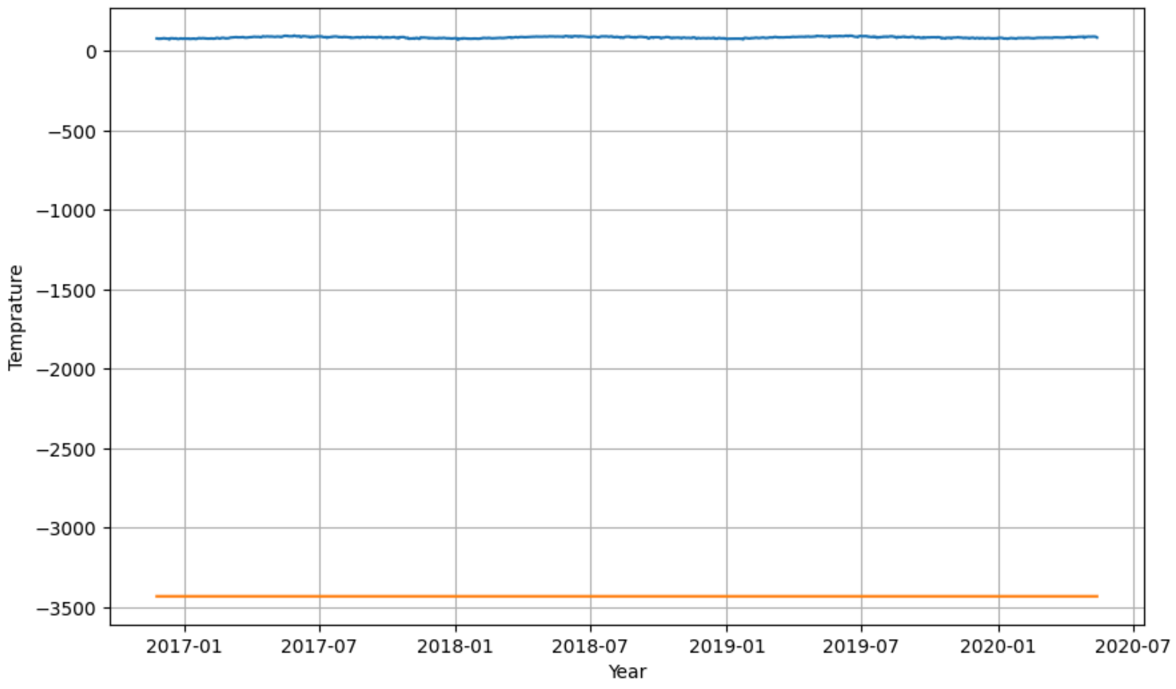
We take the step where the learning rate drops the steepest to train our neural network.

```
def model_forecast(model, series, window_size):
    """
    Given a model object and a series for it to predict, this function will return the prediction
    """
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast
```

```
rnn_forecast = model_forecast(model, series[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]
```

```
    288/288 [==============================] - 11s 36ms/step
```

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, rnn_forecast)
```

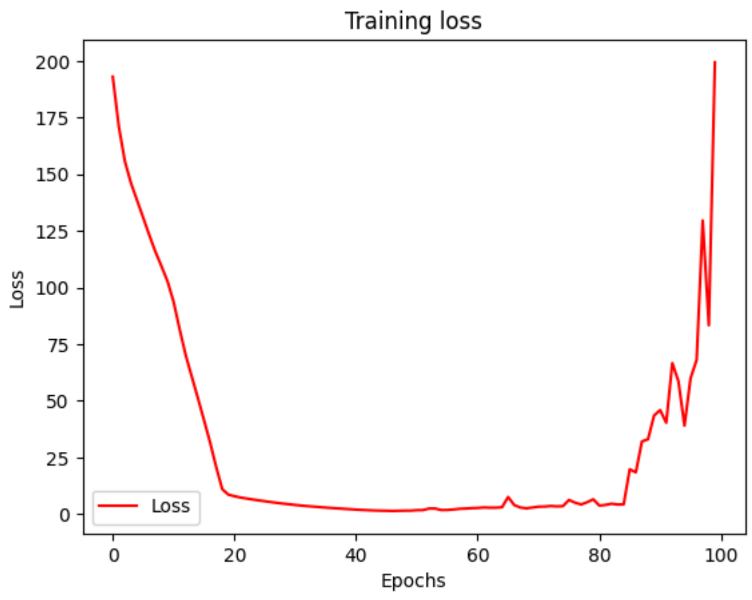


```
tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```

```
     3516.4124
```

```
loss=history.history['loss']

epochs=range(len(loss)) # Get number of epochs


#-----------------------------------------------------------
# Plot training and validation loss per epoch
#-----------------------------------------------------------
plt.plot(epochs, loss, 'r')
plt.title('Training loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Loss"])

plt.figure()

zoomed_loss = loss[200:]
zoomed_epochs = range(200,500)


#-----------------------------------------------------------
# Plot training and validation loss per epoch
#-----------------------------------------------------------
plt.plot(zoomed_epochs, zoomed_loss, 'r')
plt.title('Training loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Loss"])

plt.figure()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-44-e608038cec4e> in <cell line: 24>()
     22 # Plot training and validation loss per epoch
     23 #------------------------------------------------
---> 24 plt.plot(zoomed_epochs, zoomed_loss, 'r')
     25 plt.title('Training loss')
     26 plt.xlabel("Epochs")

3 frames

/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_base.py in _plot_args(self, tup, kwargs, return_kwargs,
ambiguous_fmt_datakey)
    502
    503         if x.shape[0] != y.shape[0]:
--> 504             raise ValueError(f"x and y must have same first dimension, but "
    505                              f"have shapes {x.shape} and {y.shape}")
    506         if x.ndim > 2 or y.ndim > 2:

ValueError: x and y must have same first dimension, but have shapes (300,) and (0,)
```


Training loss
Loss
Epochs
Loss