Weather Prediction

Timeseries forecasting using a LSTM model.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
```

In [5]:

```
#Climate Data

from zipfile import ZipFile
import os

uri = "https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_201
zip_path = keras.utils.get_file(origin=uri, fname="jena_climate_2009_2016.csv.zip")
zip_file = ZipFile(zip_path)
zip_file.extractall()
csv_path = "jena_climate_2009_2016.csv"

df = pd.read_csv(csv_path)
```

In [6]:

```
df.head()
```

Out[6]:

	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2((mmol/m
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.3	3.33	3.11	0.22	1.94	3.
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.4	3.23	3.02	0.21	1.89	3.
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.9	3.21	3.01	0.20	1.88	3.
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.2	3.26	3.07	0.19	1.92	3.
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.1	3.27	3.08	0.19	1.92	3.

- The dataset consists of 14 features such as temperature, pressure, humidity etc, recorded once per 10 minutes.
- Time-frame Considered: Jan 10, 2009 - December 31, 2016

Index	Features	Description
1	Date Time	Date-time reference
2	p (mbar)	atmospheric pressure in millibars.
3	T (degC)	Temperature in Celsius
4	Tpot (K)	Temperature in Kelvin
5	Tdew (degC)	Temperature in Celsius relative to humidity.
6	rh (%)	the %RH determines the amount of water contained within collection objects.
7	VPmax (mbar)	Saturation vapor pressure
8	VPact (mbar)	Vapor pressure
9	VPdef (mbar)	Vapor pressure deficit
10	sh (g/kg)	Specific humidity
11	H2OC (mmol/mol)	Water vapor concentration
12	rho (g/m ** 3)	Airtight
13	wv (m/s)	Wind speed
14	max. wv (m/s)	Maximum wind speed
15	wd (deg)	Wind direction in degrees

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 420551 entries, 0 to 420550
Data columns (total 15 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   Date Time        420551 non-null  object 
 1   p (mbar)         420551 non-null  float64
 2   T (degC)         420551 non-null  float64
 3   Tpot (K)         420551 non-null  float64
 4   Tdew (degC)      420551 non-null  float64
 5   rh (%)           420551 non-null  float64
 6   VPmax (mbar)     420551 non-null  float64
 7   VPact (mbar)     420551 non-null  float64
 8   VPdef (mbar)     420551 non-null  float64
 9   sh (g/kg)        420551 non-null  float64
 10  H2OC (mmol/mol)  420551 non-null  float64
 11  rho (g/m**3)     420551 non-null  float64
 12  wv (m/s)         420551 non-null  float64
 13  max. wv (m/s)    420551 non-null  float64
 14  wd (deg)         420551 non-null  float64
dtypes: float64(14), object(1)
memory usage: 48.1+ MB
```

In [18]:

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

Out[18]:

```
Date Time          0
p (mbar)           0
T (degC)           0
Tpot (K)           0
Tdew (degC)        0
rh (%)             0
VPmax (mbar)       0
VPact (mbar)       0
VPdef (mbar)       0
sh (g/kg)          0
H2OC (mmol/mol)    0
rho (g/m**3)       0
wv (m/s)           0
max. wv (m/s)      0
wd (deg)           0
dtype: int64
```

Raw Data Visualization

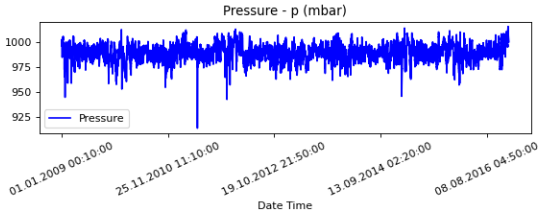
- Each feature has been displayed below to give us a better understanding of the data we are working with. This displays each feature's distinctive pattern from 2009 to 2016 in comparison to one another. Additionally, it identifies any irregularities so that they can be corrected during normalisation.

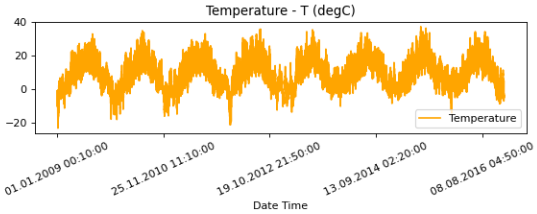
In [11]:

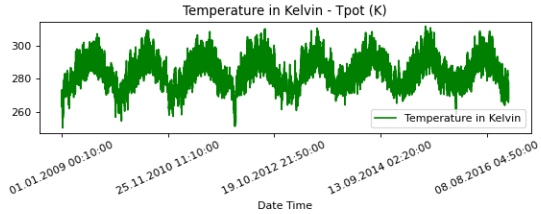
```
titles = [
    "Pressure",
    "Temperature",
    "Temperature in Kelvin",
    "Temperature (dew point)",
    "Relative Humidity",
    "Saturation vapor pressure",
    "Vapor pressure",
    "Vapor pressure deficit",
    "Specific humidity",
    "Water vapor concentration",
    "Airtight",
    "Wind speed",
    "Maximum wind speed",
    "Wind direction in degrees",]

feature_keys = [
    "p (mbar)",
    "T (degC)",
    "Tpot (K)",
    "Tdew (degC)",
    "rh (%)",
    "VPmax (mbar)",
    "VPact (mbar)",
    "VPdef (mbar)",
    "sh (g/kg)",
    "H2OC (mmol/mol)",
    "rho (g/m**3)",
    "wv (m/s)",
    "max. wv (m/s)",
    "wd (deg)",]

colors = [
    "blue",
    "orange",
    "green",
    "red",
    "purple",
    "brown",
    "pink",
    "gray",
    "olive",
    "cyan",]

date_time_key = "Date Time"
```

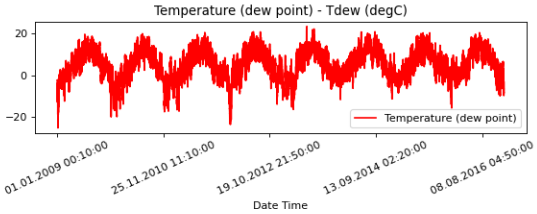
In [14]:

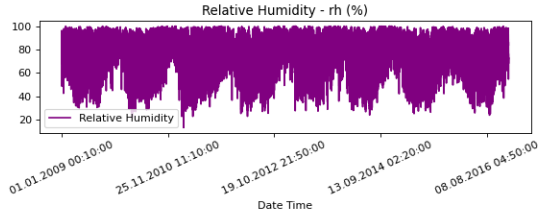
```
def show_raw_visualization(data):
    time_data = data[date_time_key]
    fig, axes = plt.subplots(
        nrows=7, ncols=2, figsize=(15, 20), dpi=80, facecolor="w", edgecolor="k"
    )
    for i in range(len(feature_keys)):
        key = feature_keys[i]
        c = colors[i % (len(colors))]
        t_data = data[key]
        t_data.index = time_data
        t_data.head()
        ax = t_data.plot(
            ax=axes[i // 2, i % 2],
            color=c,
            title="{} - {}".format(titles[i], key),
            rot=25,
        )
        ax.legend([titles[i]])
    plt.tight_layout()


show_raw_visualization(df)
```

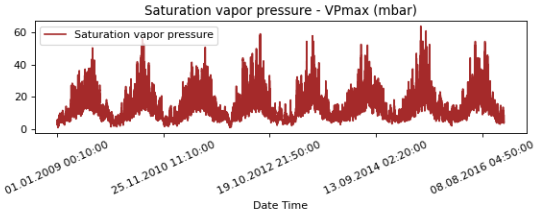

Pressure - p (mbar)
Pressure
Date Time

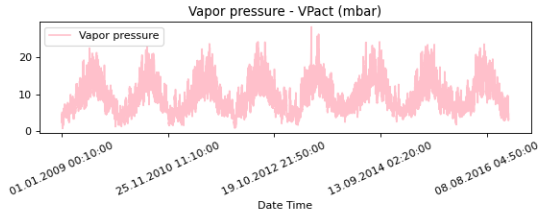

Temperature - T (degC)
Temperature
Date Time

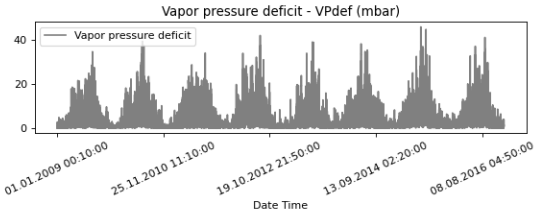

Temperature in Kelvin - Tpot (K)
Temperature in Kelvin
Date Time

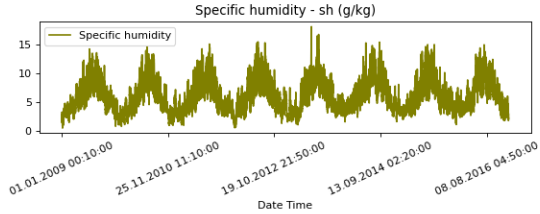

Temperature (dew point) - Tdew (degC)
Temperature (dew point)
Date Time

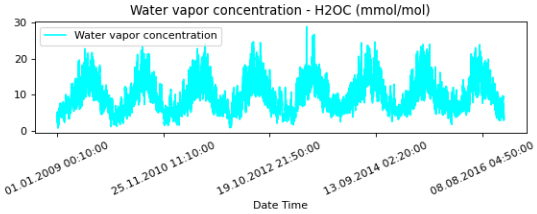

Relative Humidity - rh (%)
Relative Humidity
Date Time

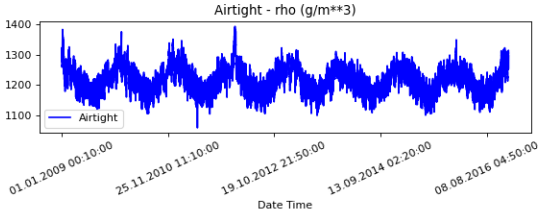

Saturation vapor pressure - VPmax (mbar)
Saturation vapor pressure
Date Time

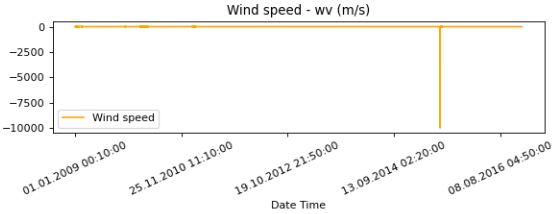

Vapor pressure - VPact (mbar)
Vapor pressure
Date Time

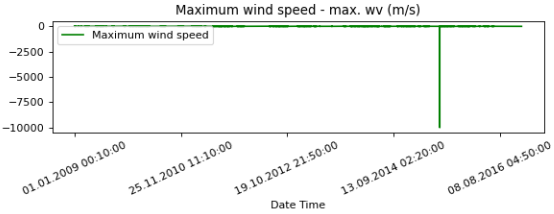

Vapor pressure deficit - VPdef (mbar)
Vapor pressure deficit
Date Time

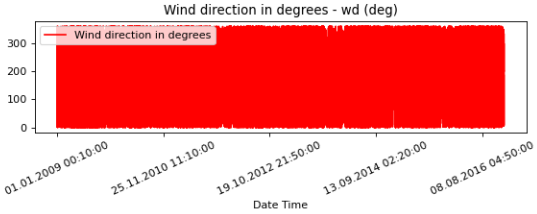

Specific humidity - sh (g/kg)
Specific humidity
Date Time


Water vapor concentration - H2OC (mmol/mol)
Water vapor concentration
Date Time


Airtight - rho (g/m**3)
Airtight
Date Time


Wind speed - wv (m/s)
Wind speed
Date Time

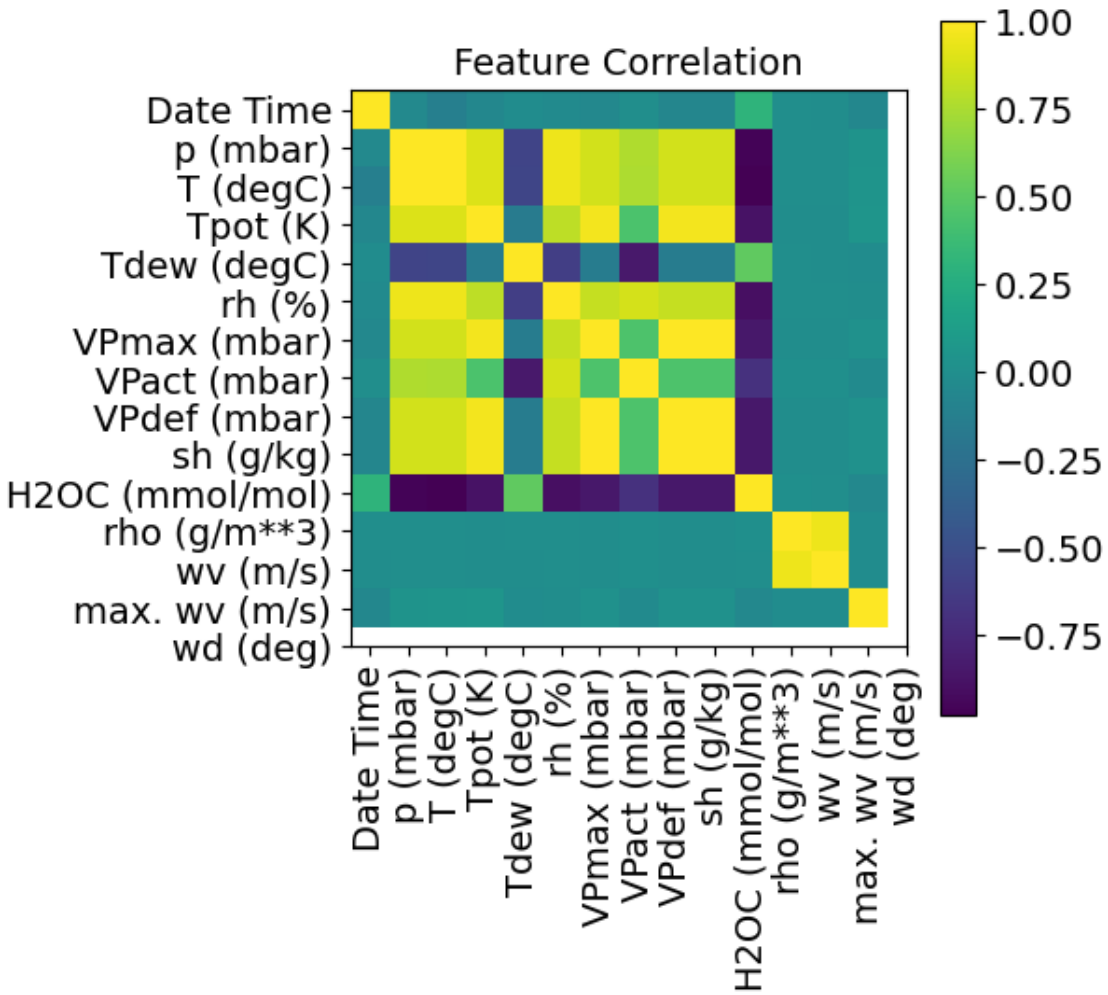

Maximum wind speed - max. wv (m/s)
Maximum wind speed
Date Time


Wind direction in degrees - wd (deg)
Wind direction in degrees
Date Time

In [15]:

```
#This heat map shows the correlation between different features.

def show_heatmap(data):
    plt.matshow(data.corr())
    plt.xticks(range(data.shape[1]), data.columns, fontsize=14,rotation = 90)
    plt.gca().xaxis.tick_bottom()
    plt.yticks(range(data.shape[1]), data.columns, fontsize = 14)

    cb = plt.colorbar()
    cb.ax.tick_params(labelsize = 14)
    plt.title("Feature Correlation", fontsize = 14)
    plt.show()

show_heatmap(df)
```

```
C:\Users\SAGAR\AppData\Local\Temp\ipykernel_13224\1101833162.py:4: Future
Warning: The default value of numeric_only in DataFrame.corr is deprecate
d. In a future version, it will default to False. Select only valid colum
ns or specify the value of numeric_only to silence this warning.
  plt.matshow(data.corr())
```



Data Preprocessing

- About 300,000 data points are selected in this case for training. A recording of an observation is made every 10 minutes, or six times an hour. Since no significant change is anticipated within 60 minutes,

we will resample once every hour. We accomplish this using the timeseries_dataset_from_array utility's sampling_rate option.

- We are tracking data from past 720 timestamps (720/6=120 hours). This data will be used to predict the temperature after 72 timestamps (72/6=12 hours).
- The model is shown data for first 5 days i.e. 720 observations, that are sampled every hour. The temperature after 72 (12 hours * 6 observation per hour) observation will be used as a label.

In [22]:

```
split_fraction = 0.715
train_split = int(split_fraction * int(df.shape[0]))
step = 6

past = 720
future = 72
learning_rate = 0.001
batch_size = 256
epochs = 10

def normalize(data, train_split):
    data_mean = data[:train_split].mean(axis = 0)
    data_std = data[:train_split].std(axis = 0)
    return(data - data_mean)/data_std
```

- A few factors, such as Relative Humidity and Specific Humidity, are redundant, as can be seen from the correlation heatmap. As a result, we won't be using all of the features.

In [24]:

```
print("The selected parameters are:",", ".join([titles[i] for i in [0, 1, 5, 7, 8, 10, 1

selected_features = [feature_keys[i] for i in [0, 1, 5, 7, 8, 10, 11]]
features = df[selected_features]
features.index = df[date_time_key]
features.head()

features = normalize(features.values, train_split)
features = pd.DataFrame(features)
features.head()
```

```
The selected parameters are: Pressure, Temperature, Saturation vapor pres
sure, Vapor pressure deficit, Specific humidity, Airtight, Wind speed
```

Out[24]:

	0	1	2	3	4	5	6
0	0.955451	-2.000020	-1.319782	-0.788479	-1.500927	2.237658	-0.732997
1	0.961528	-2.045185	-1.332806	-0.790561	-1.519521	2.287838	-0.936002
2	0.956666	-2.056766	-1.335410	-0.792642	-1.523239	2.298608	-1.283076
3	0.954236	-2.033604	-1.328898	-0.794724	-1.508364	2.272906	-1.184847
4	0.954236	-2.028972	-1.327596	-0.794724	-1.508364	2.268256	-1.197944

Training dataset

- The training dataset labels starts from the 792nd observation (720 + 72).

In [25]:

```
train_data = features.loc[0 : train_split - 1]
val_data = features.loc[train_split:]
```

In [29]:

```
# Training dataset

start = past + future
end = start + train_split

x_train = train_data[[i for i in range(7)]].values
y_train = features.iloc[start:end][[1]]

sequence_length = int(past/step)
```

In [30]:

```
dataset_train = keras.preprocessing.timeseries_dataset_from_array(
x_train,y_train,
sequence_length=sequence_length,
sampling_rate=step,
batch_size=batch_size)
```

Validation dataset

- The validation label dataset must start from 792 after train_split, hence we must add past + future (792) to label_start.

In [31]:

```
x_end = len(val_data) - past - future
label_start = train_split + past + future

x_val = val_data.iloc[:x_end][[i for i in range(7)]].values
y_val = features.iloc[label_start:][[1]]

dataset_val = keras.preprocessing.timeseries_dataset_from_array(
    x_val,
    y_val,
    sequence_length=sequence_length,
    sampling_rate=step,
    batch_size=batch_size,
)

for batch in dataset_train.take(1):
    inputs, targets = batch

print(f"Input shape: {inputs.numpy().shape}")
print(f"Target shape: {targets.numpy().shape}")
```

```
Input shape: (256, 120, 7)
Target shape: (256, 1)
```

Training

In [33]:

```
inputs = keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
lstm_out = keras.layers.LSTM(32)(inputs)
outputs = keras.layers.Dense(1)(lstm_out)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=keras.optimizers.Adam(learning_rate=learning_rate), loss="mse")
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 120, 7)]          0

 lstm (LSTM)                 (None, 32)                5120

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 5,153
Trainable params: 5,153
Non-trainable params: 0
_________________________________________________________________
```

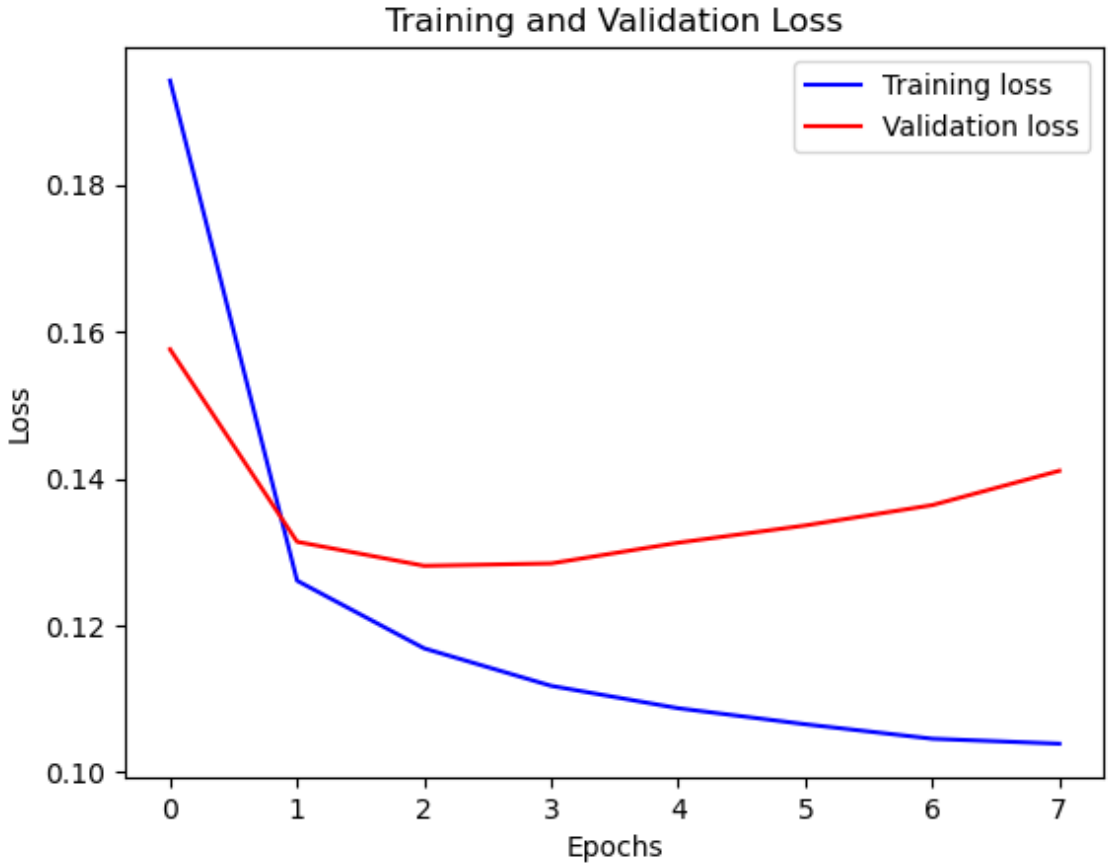
In [34]:

```
path_checkpoint = "model_checkpoint.h5"
es_callback = keras.callbacks.EarlyStopping(monitor="val_loss", min_delta=0, patience=5)

modelckpt_callback = keras.callbacks.ModelCheckpoint(
    monitor="val_loss",
    filepath=path_checkpoint,
    verbose=1,
    save_weights_only=True,
    save_best_only=True,
)

history = model.fit(
    dataset_train,
    epochs=epochs,
    validation_data=dataset_val,
    callbacks=[es_callback, modelckpt_callback],
)
```

```
Epoch 1/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1942
Epoch 1: val_loss improved from inf to 0.15761, saving model to model_che
ckpoint.h5
1172/1172 [==============================] - 406s 341ms/step - loss: 0.19
42 - val_loss: 0.1576
Epoch 2/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1261
Epoch 2: val_loss improved from 0.15761 to 0.13140, saving model to model
_checkpoint.h5
1172/1172 [==============================] - 412s 351ms/step - loss: 0.12
61 - val_loss: 0.1314
Epoch 3/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1169
Epoch 3: val_loss improved from 0.13140 to 0.12812, saving model to model
_checkpoint.h5
1172/1172 [==============================] - 388s 331ms/step - loss: 0.11
69 - val_loss: 0.1281
Epoch 4/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1118
Epoch 4: val_loss did not improve from 0.12812
1172/1172 [==============================] - 488s 416ms/step - loss: 0.11
18 - val_loss: 0.1284
Epoch 5/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1087
Epoch 5: val_loss did not improve from 0.12812
1172/1172 [==============================] - 506s 432ms/step - loss: 0.10
87 - val_loss: 0.1313
Epoch 6/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1065
Epoch 6: val_loss did not improve from 0.12812
1172/1172 [==============================] - 478s 408ms/step - loss: 0.10
65 - val_loss: 0.1336
Epoch 7/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1046
Epoch 7: val_loss did not improve from 0.12812
1172/1172 [==============================] - 527s 450ms/step - loss: 0.10
46 - val_loss: 0.1364
Epoch 8/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1039
Epoch 8: val_loss did not improve from 0.12812
1172/1172 [==============================] - 539s 460ms/step - loss: 0.10
39 - val_loss: 0.1411
```

In [35]:

```
def visualize_loss(history, title):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="Training loss")
    plt.plot(epochs, val_loss, "r", label="Validation loss")
    plt.title(title)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()


visualize_loss(history, "Training and Validation Loss")
```



Prediction

In [36]:

```
def show_plot(plot_data, delta, title):
    labels = ["History", "True Future", "Model Prediction"]
    marker = [".-", "rx", "go"]
    time_steps = list(range(-(plot_data[0].shape[0]), 0))
    if delta:
        future = delta
    else:
        future = 0

    plt.title(title)
    for i, val in enumerate(plot_data):
        if i:
            plt.plot(future, plot_data[i], marker[i], markersize=10, label=labels[i])
        else:
            plt.plot(time_steps, plot_data[i].flatten(), marker[i], label=labels[i])
    plt.legend()
    plt.xlim([time_steps[0], (future + 5) * 2])
    plt.xlabel("Time-Step")
    plt.show()
    return


for x, y in dataset_val.take(5):
    show_plot(
        [x[0][:, 1].numpy(), y[0].numpy(), model.predict(x)[0]],
        12,
        "Single Step Prediction",
    )
```

```
8/8 [==============================] - 1s 37ms/step
```