In [1]:
```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import pandas as pd
import numpy as np
```

In [2]:
```
df_t= pd.read_csv('temperature.csv', index_col=0)
df_h= pd.read_csv('humidity.csv', index_col=0)
df_p = pd.read_csv('pressure.csv', index_col=0)
df_w = pd.read_csv('wind_speed.csv', index_col=0)
df_h.columns
```

Out[2]:
```
Index(['Vancouver', 'Portland', 'San Francisco', 'Seattle', 'Los Angeles',
       'San Diego', 'Las Vegas', 'Phoenix', 'Albuquerque', 'Denver',
       'San Antonio', 'Dallas', 'Houston', 'Kansas City', 'Minneapolis',
       'Saint Louis', 'Chicago', 'Nashville', 'Indianapolis', 'Atlanta',
       'Detroit', 'Jacksonville', 'Charlotte', 'Miami', 'Pittsburgh',
       'Toronto', 'Philadelphia', 'New York', 'Montreal', 'Boston',
       'Beersheba', 'Tel Aviv District', 'Eilat', 'Haifa', 'Nahariyya',
       'Jerusalem'],
      dtype='object')
```

In [5]:
```
def normalize(data):
    data_mean = data.mean (axis=0)
    data_std = data.std (axis=0)
    return (data - data_mean) / data_std
```

In [7]:
```
city = 'San Diego'
tempog = df_t[city]
temp = df_t[city].rename("temperature").to_frame(name='temperature')
humid = df_h[city].rename("humidity").to_frame (name='humidity')
press = df_p[city].rename("presure").to_frame (name='presure')
wind = df_w[city].rename("wind").to_frame (name='wind')
features = pd.concat([temp, press, humid, wind], axis=1)
#features. index -time
features = features.dropna()
features
featuresog = features
featuresog
```

Out[7]:

datetime	temperature	presure	humidity	wind
2012-10-01 13:00:00	291.530000	1013.0	82.0	0.0
2012-10-01 14:00:00	291.533501	1013.0	81.0	0.0
2012-10-01 15:00:00	291.543355	1013.0	81.0	0.0
2012-10-01 16:00:00	291.553209	1013.0	81.0	0.0
2012-10-01 17:00:00	291.563063	1013.0	80.0	0.0
...	...	...	...	...
2017-11-29 20:00:00	292.150000	1017.0	72.0	2.0
2017-11-29 21:00:00	292.740000	1017.0	72.0	1.0
2017-11-29 22:00:00	292.580000	1016.0	68.0	2.0
2017-11-29 23:00:00	292.610000	1016.0	63.0	2.0
2017-11-30 00:00:00	291.400000	1017.0	72.0	2.0

44899 rows × 4 columns

In [9]:

```
features = normalize(features.values)
features= pd.DataFrame (features)
features
```

Out[9]:

	0	1	2	3
0	0.222888	-0.528665	0.731908	-1.181374
1	0.223482	-0.528665	0.680392	-1.181374
2	0.225155	-0.528665	0.680392	-1.181374
3	0.226829	-0.528665	0.680392	-1.181374
4	0.228502	-0.528665	0.628876	-1.181374
...	...	...	...	...
44894	0.328173	-0.061829	0.216750	0.162904
44895	0.428364	-0.061829	0.216750	-0.509235
44896	0.401194	-0.178538	0.010687	0.162904
44897	0.406288	-0.178538	-0.246892	0.162904
44898	0.200812	-0.061829	0.216750	0.162904

44899 rows × 4 columns

In [11]:

```
training_size = int(0.8 * features.shape[0])
train_data = features.loc[0: training_size - 1]
val_data = features.loc[training_size:]
```

In [13]:
```
start = 432 + 36
end = start + training_size
x_train = train_data.values
y_train = features.iloc[start:end][[0]]
sequence_length = int(432 / 6)
```

In [15]:
```
from tensorflow import keras
dataset_train = keras. preprocessing.timeseries_dataset_from_array(
    data=x_train,
    targets  =y_train,
    sequence_length=sequence_length,
    sampling_rate=6,
    batch_size=64,
)
```

In [17]:
```
x_val_end = len(val_data) - start
label_start = training_size + start
x_val = val_data.iloc[:x_val_end] [[i for i in range(4)]].values
y_val = features.iloc[label_start:][[0]]
dataset_val = keras.preprocessing.timeseries_dataset_from_array(
    x_val,
    y_val,
    sequence_length=sequence_length,
    sampling_rate=6,
    batch_size=64,
)
```

In [19]:
```
for batch in dataset_train. take (1):
    inputs, targets = batch
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 (name="Weather_forcaster", inputs=inputs, outputs = outputs)
model.compile(optimizer = keras.optimizers.Adam (learning_rate=0.001), loss="mse"
model. summary()
```

```
Model: "Weather_forcaster"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 72, 4)]           0

 lstm (LSTM)                 (None, 32)                4736

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 4769 (18.63 KB)
Trainable params: 4769 (18.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

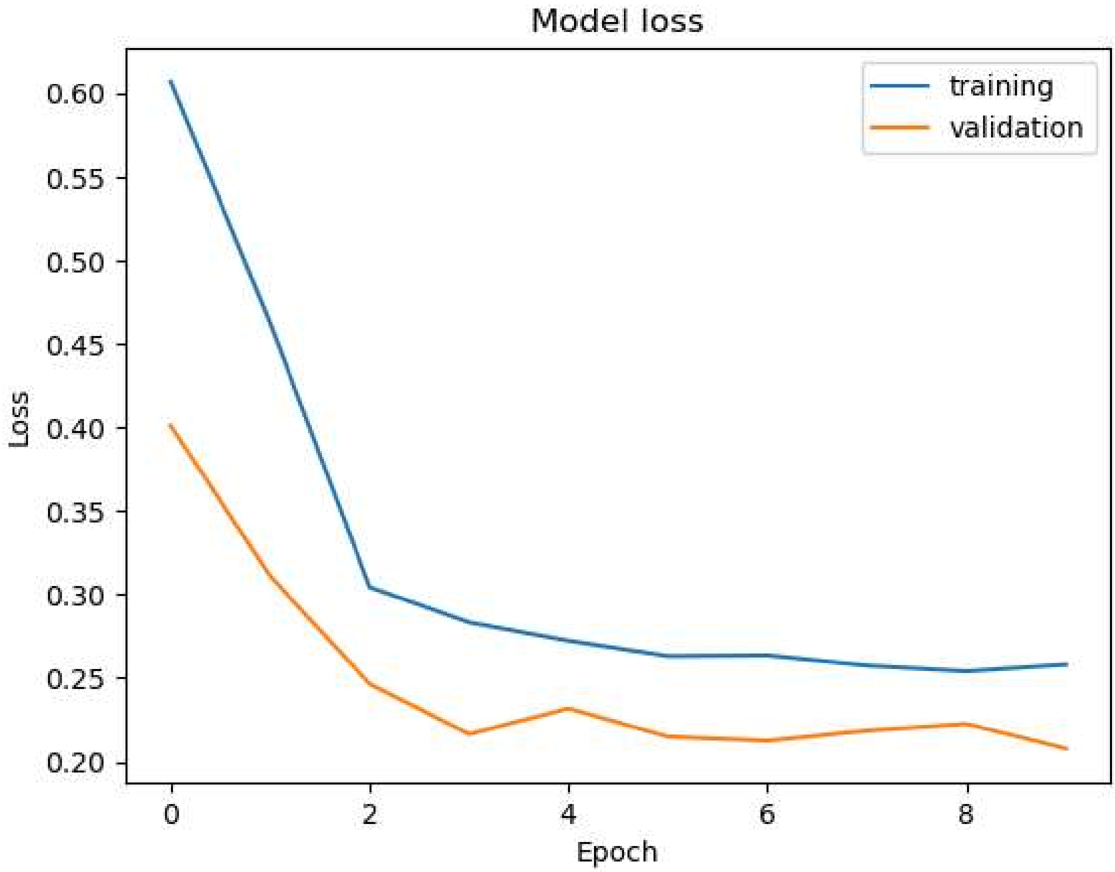
In [21]:
```
history = model.fit(
    dataset_train,
    epochs=10,
    validation_data=dataset_val
)
```

```
Epoch 1/10
555/555 [==============================] - 10s 13ms/step - loss: 0.6069 - val_los
s: 0.4010
Epoch 2/10
555/555 [==============================] - 7s 12ms/step - loss: 0.4623 - val_loss:
0.3110
Epoch 3/10
555/555 [==============================] - 7s 13ms/step - loss: 0.3044 - val_loss:
0.2465
Epoch 4/10
555/555 [==============================] - 7s 13ms/step - loss: 0.2833 - val_loss:
0.2165
Epoch 5/10
555/555 [==============================] - 7s 13ms/step - loss: 0.2723 - val_loss:
0.2318
Epoch 6/10
555/555 [==============================] - 7s 13ms/step - loss: 0.2632 - val_loss:
0.2149
Epoch 7/10
555/555 [==============================] - 7s 12ms/step - loss: 0.2636 - val_loss:
0.2125
Epoch 8/10
555/555 [==============================] - 7s 13ms/step - loss: 0.2575 - val_loss:
0.2185
Epoch 9/10
555/555 [==============================] - 7s 13ms/step - loss: 0.2541 - val_loss:
0.2225
Epoch 10/10
555/555 [==============================] - 7s 13ms/step - loss: 0.2581 - val_loss:
0.2079
```

In [27]:

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend (['training', 'validation'], loc='upper right')
plt.show()
```



In [31]:
```
dataset = featuresog.values
dataset = dataset.astype('float32')
scaler = MinMaxScaler (feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
#print('dataset. shape', dataset.shape)
num_of_features = len(features.columns)
print('Number of features', num_of_features)

look_back = sequence_length
```

```
Number of features 4
```

In [45]:
```
train_size_percent= 0.80
pred_col = featuresog.columns.get_loc("temperature")
print(pred_col)
#function to split the data
def create_dataset(dataset, pred_col, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset [i: (i+look_back), :]
        dataX.append(a)
        dataY.append(dataset [i + look_back, pred_col])
    return np.array(dataX), np.array(dataY)

train_size = int(len(dataset) * train_size_percent)
test_size = len(dataset) - train_size
train, test = dataset [0: train_size, :], dataset [train_size: len(dataset), :]
trainX, trainY = create_dataset (train, pred_col, look_back=look_back)
testX, testY = create_dataset (test, pred_col, look_back=look_back)
# reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], look_back, num_of_features))
testX = np.reshape(testX, (testX. shape[0], look_back, num_of_features))
print('Training dataset length', len(train))
```

```
print('Testing dataset length ', len(test))
print('look_back', look_back)
```

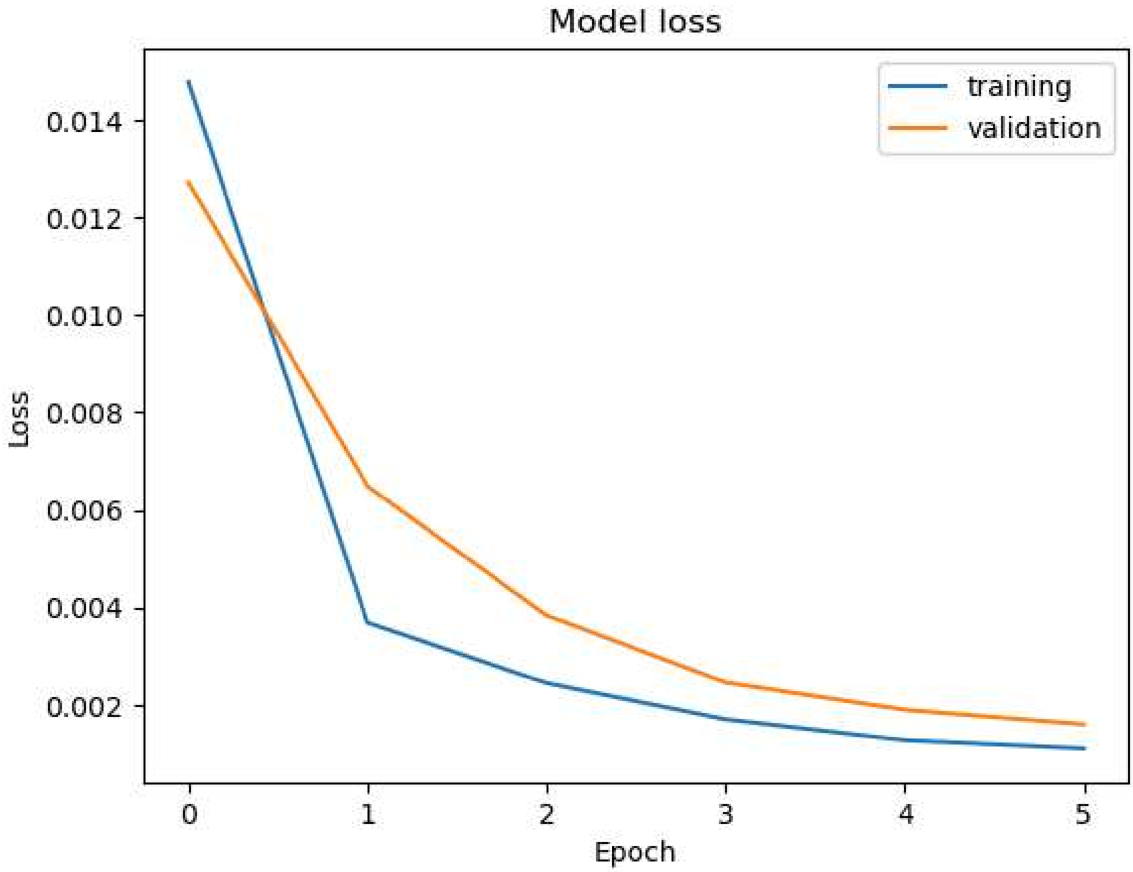
```
0
Training dataset length 35919
Testing dataset length  8980
look_back 72
```

In [49]:
```
expr_name = 'expr_4'
# Look_back = 24*120 # 60 days, as each entry is for 1 hour
lstm_layers = 64 # 64
epochs = 6 #6
batch_size = 128 #128

model = Sequential()
model.add(LSTM(lstm_layers, input_shape=(look_back,num_of_features)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
history= model.fit(trainX,trainY,validation_split=0.30, epochs=epochs, batch_size
```

```
Epoch 1/6
197/197 [==============================] - 10s 37ms/step - loss: 0.0148 - val_los
s: 0.0127
Epoch 2/6
197/197 [==============================] - 7s 34ms/step - loss: 0.0037 - val_loss:
0.0065
Epoch 3/6
197/197 [==============================] - 7s 35ms/step - loss: 0.0024 - val_loss:
0.0038
Epoch 4/6
197/197 [==============================] - 7s 33ms/step - loss: 0.0017 - val_loss:
0.0025
Epoch 5/6
197/197 [==============================] - 6s 33ms/step - loss: 0.0013 - val_loss:
0.0019
Epoch 6/6
197/197 [==============================] - 7s 34ms/step - loss: 0.0011 - val_loss:
0.0016
```

In [51]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend (['training', 'validation'], loc='upper right')
plt.show()
```



In [77]:

```
import math
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

#Get something which has as many features as dataset
trainPredict_extended = np.zeros((len(trainPredict), num_of_features))
#Put the predictions there
trainPredict_extended[:, pred_col] = trainPredict[:,0]
#Inverse transform it and select the 3rd column.
trainPredict = scaler.inverse_transform(trainPredict_extended) [:,pred_col]

#Get something which has as many features as dataset
testPredict_extended = np.zeros((len(testPredict), num_of_features))
#Put the predictions there
testPredict_extended[:,pred_col] = testPredict[:,0]
#Inverse transform it and select the pred_coL column.
testPredict = scaler.inverse_transform(testPredict_extended)[:,pred_col]

trainY_extended = np.zeros((len(trainY), num_of_features))
trainY_extended[:, pred_col]=trainY
trainY = scaler.inverse_transform(trainY_extended) [:, pred_col]

testY_extended = np.zeros((len(testY), num_of_features))
testY_extended[:, pred_col]-testY
testY = scaler.inverse_transform(testY_extended) [:, pred_col]

#calculate root mean squared error
trainscore_RMSE = math.sqrt(mean_squared_error(trainY, trainPredict))
testscore_RMSE = math.sqrt(mean_squared_error(testY, testPredict))

#calculate absolute mean error
trainScore_MAE= np.sum(np.absolute (trainY - trainPredict))/len(trainY)
testScore_MAE= np.sum(np.absolute (testY - testPredict))/len (testY)
```

```
#shift train predictions for plotting
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :]= np.nan
trainPredictPlot[look_back: len(trainPredict)+look_back, pred_col] = trainPredict

#shift test predictions for plotting
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :]= np.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, pred_col] = tes

#contruct pandas dataframe for plotting
time_df = pd.DataFrame(featuresog.index)
time_df['Actual'] = scaler.inverse_transform(dataset)[:, pred_col]
df1= pd.DataFrame (trainPredictPlot[:,pred_col], columns=['Train'])
df2= pd.DataFrame(testPredictPlot[:,pred_col], columns=['Test'])
time_df2= pd.concat([time_df, df1, df2], axis=1, sort=False)

# print(time_df2)

time_df2.set_index('datetime', inplace=True)
```

```
1121/1121 [==============================] - 6s 5ms/step
279/279 [==============================] - 1s 5ms/step
```

In [83]:

```
print(trainscore_RMSE)
print(testscore_RMSE)
```

```
3431172313701.46
26.11277553072241
```

In [87]:

```
fig, ax = plt.subplots(figsize=(15,7))
time_df2.plot(ax=ax, rot=90,alpha=0.7)
plt.xlabel('Timestamp')
plt.ylabel('Temperature Value')
plt.title('Temperature Prediction')
plt.savefig(expr_name + '.png', bbox_inches = "tight")
```