# nd-forecasting-aiml-code-section

July 10, 2024

[2]:
```
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Dense, LSTM,␣
↪RepeatVector, TimeDistributed, Flatten
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff

# Initialize plotly notebook mode
py.init_notebook_mode(connected=True)
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)

# Ignore warnings
warnings.filterwarnings("ignore")

# Set seeds to make the experiment more reproducible
from tensorflow.compat.v1 import set_random_seed
from numpy.random import seed
set_random_seed(1)
seed(1)

# Enable inline plotting for matplotlib
%matplotlib inline
```

[3]:
```
#Loading data

train = pd.read_csv('train1.csv', parse_dates=['date'])
test = pd.read_csv('test1.csv', parse_dates=['date'])
```

[4]:
```
#Train set

train.describe()
```

[4]:
```
                                date          store           item  \
count                         913000  913000.000000  913000.000000
mean   2015-07-02 11:59:59.999999744       5.500000      25.500000
min              2013-01-01 00:00:00       1.000000       1.000000
25%              2014-04-02 00:00:00       3.000000      13.000000
50%              2015-07-02 12:00:00       5.500000      25.500000
75%              2016-10-01 00:00:00       8.000000      38.000000
max              2017-12-31 00:00:00      10.000000      50.000000
std                              NaN       2.872283      14.430878

               sales
count  913000.000000
mean       52.250287
min         0.000000
25%        30.000000
50%        47.000000
75%        70.000000
max       231.000000
std        28.801144
```

[5]:
```
train.head()
```

[5]:
```
        date  store  item  sales
0 2013-01-01      1     1     13
1 2013-01-02      1     1     11
2 2013-01-03      1     1     14
3 2013-01-04      1     1     13
4 2013-01-05      1     1     10
```

[6]:
```
#Time period of the train dataset
print('Min date from train set: %s' % train['date'].min().date())
print('Max date from train set: %s' % train['date'].max().date())
```

```
Min date from train set: 2013-01-01
Max date from train set: 2017-12-31
```

[7]:

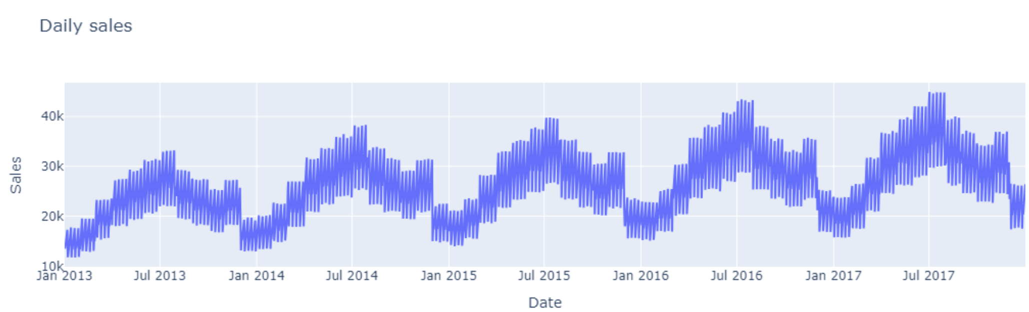
```
#Let's find out what's the time gap between the last day from training set from␣
↪the last day of the test set, this will be out lag (the amount of day that␣
↪need to be forecast)
lag_size = (test['date'].max().date() - train['date'].max().date()).days
print('Max date from train set: %s' % train['date'].max().date())
print('Max date from test set: %s' % test['date'].max().date())
print('Forecast lag size', lag_size)
```

```
Max date from train set: 2017-12-31
Max date from test set: 2018-03-31
Forecast lag size 90
```

[ ]:
```
#Basic EDA
To explore the time series data first we need to aggregate the sales by day
```

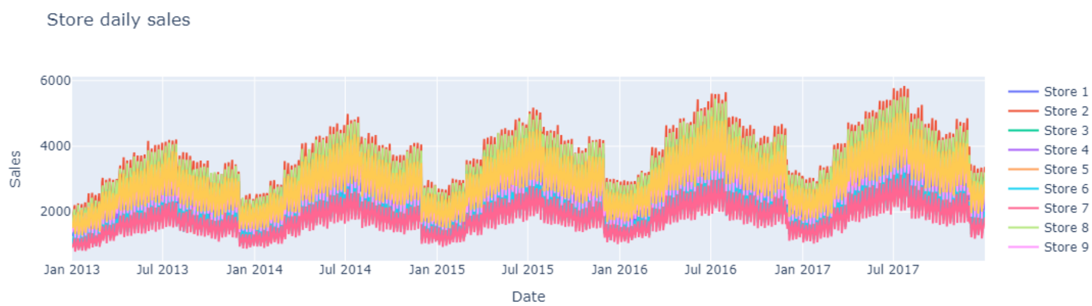
[8]:
```
daily_sales = train.groupby('date', as_index=False)['sales'].sum()
store_daily_sales = train.groupby(['store', 'date'], as_index=False)['sales'].
↪sum()
item_daily_sales = train.groupby(['item', 'date'], as_index=False)['sales'].
↪sum()
```

[9]:
```
#Overall daily sales

daily_sales_sc = go.Scatter(x=daily_sales['date'], y=daily_sales['sales'])
layout = go.Layout(title='Daily sales', xaxis=dict(title='Date'),␣
↪yaxis=dict(title='Sales'))
fig = go.Figure(data=[daily_sales_sc], layout=layout)
iplot(fig)
```

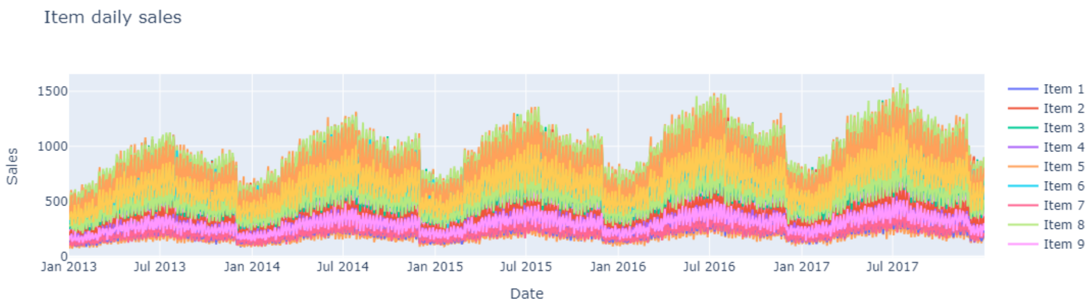



[10]:
```
#Daily sales by store

store_daily_sales_sc = []
for store in store_daily_sales['store'].unique():
```

```
    current_store_daily_sales = store_daily_sales[(store_daily_sales['store'] == store)]
    store_daily_sales_sc.append(go.Scatter(x=current_store_daily_sales['date'], y=current_store_daily_sales['sales'], name=('Store %s' % store)))

layout = go.Layout(title='Store daily sales', xaxis=dict(title='Date'), yaxis=dict(title='Sales'))
fig = go.Figure(data=store_daily_sales_sc, layout=layout)
iplot(fig)
```




[11]:
```
#Daily sales by item

item_daily_sales_sc = []
for item in item_daily_sales['item'].unique():
    current_item_daily_sales = item_daily_sales[(item_daily_sales['item'] == item)]
    item_daily_sales_sc.append(go.Scatter(x=current_item_daily_sales['date'], y=current_item_daily_sales['sales'], name=('Item %s' % item)))

layout = go.Layout(title='Item daily sales', xaxis=dict(title='Date'), yaxis=dict(title='Sales'))
fig = go.Figure(data=item_daily_sales_sc, layout=layout)
iplot(fig)
```




[12]:
```
#Sub-sample train set to get only the last year of data and reduce training time
train = train[(train['date'] >= '2017-01-01')]
```

[13]:
```
#Rearrange dataset so we can apply shift methods
train_gp = train.sort_values('date').groupby(['item', 'store', 'date'],␣
↪as_index=False)
train_gp = train_gp.agg({'sales':['mean']})
train_gp.columns = ['item', 'store', 'date', 'sales']
train_gp.head()
```

[13]:

| | item | store | date | sales |
|---|---|---|---|---|
| 0 | 1 | 1 | 2017-01-01 | 19.0 |
| 1 | 1 | 1 | 2017-01-02 | 15.0 |
| 2 | 1 | 1 | 2017-01-03 | 10.0 |
| 3 | 1 | 1 | 2017-01-04 | 16.0 |
| 4 | 1 | 1 | 2017-01-05 | 14.0 |

[14]:
```
#Transform the data into a time series problem
def series_to_supervised(data, window=1, lag=1, dropnan=True):
    cols, names = list(), list()
    # Input sequence (t-n, ... t-1)
    for i in range(window, 0, -1):
        cols.append(data.shift(i))
        names += [('%s(t-%d)' % (col, i)) for col in data.columns]
    # Current timestep (t=0)
    cols.append(data)
    names += [('%s(t)' % (col)) for col in data.columns]

    # Target timestep (t=lag)
    cols.append(data.shift(-lag))
    names += [('%s(t+%d)' % (col, lag)) for col in data.columns]

    # Put it all together
    agg = pd.concat(cols, axis=1)
```

```
    agg.columns = names

    # Drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

[15]:
```
#We will use the current timestep and the last 29 to forecast 90 days ahead
window = 29
lag = lag_size
series = series_to_supervised(train_gp.drop('date', axis=1), window=window,␣
 ↪lag=lag)
series.head()
```

[15]:
```
    item(t-29)  store(t-29)  sales(t-29)  item(t-28)  store(t-28)  \
29         1.0          1.0         19.0         1.0          1.0
30         1.0          1.0         15.0         1.0          1.0
31         1.0          1.0         10.0         1.0          1.0
32         1.0          1.0         16.0         1.0          1.0
33         1.0          1.0         14.0         1.0          1.0

    sales(t-28)  item(t-27)  store(t-27)  sales(t-27)  item(t-26)  …  \
29         15.0         1.0          1.0         10.0         1.0  …
30         10.0         1.0          1.0         16.0         1.0  …
31         16.0         1.0          1.0         14.0         1.0  …
32         14.0         1.0          1.0         24.0         1.0  …
33         24.0         1.0          1.0         14.0         1.0  …

    sales(t-2)  item(t-1)  store(t-1)  sales(t-1)  item(t)  store(t)  \
29        16.0        1.0         1.0        24.0        1         1
30        24.0        1.0         1.0         9.0        1         1
31         9.0        1.0         1.0        17.0        1         1
32        17.0        1.0         1.0        15.0        1         1
33        15.0        1.0         1.0        17.0        1         1

    sales(t)  item(t+90)  store(t+90)  sales(t+90)
29       9.0         1.0          1.0         33.0
30      17.0         1.0          1.0         15.0
31      15.0         1.0          1.0         21.0
32      17.0         1.0          1.0         29.0
33      24.0         1.0          1.0         19.0

[5 rows x 93 columns]
```

[16]:
```
#Drop rows with different item or store values than the shifted columns
last_item = 'item(t-%d)' % window
last_store = 'store(t-%d)' % window
```

```
series = series[(series['store(t)'] == series[last_store])]
series = series[(series['item(t)'] == series[last_item])]
```

[17]:
```
#Remove unwanted columns
columns_to_drop = [('%s(t+%d)' % (col, lag)) for col in ['item', 'store']]
for i in range(window, 0, -1):
    columns_to_drop += [('%s(t-%d)' % (col, i)) for col in ['item', 'store']]
series.drop(columns_to_drop, axis=1, inplace=True)
series.drop(['item(t)', 'store(t)'], axis=1, inplace=True)
```

[18]:
```
#Train/validation split
# Label
labels_col = 'sales(t+%d)' % lag_size
labels = series[labels_col]
series = series.drop(labels_col, axis=1)

X_train, X_valid, Y_train, Y_valid = train_test_split(series, labels.values,␣
 ↪test_size=0.4, random_state=0)
print('Train set shape', X_train.shape)
print('Validation set shape', X_valid.shape)
X_train.head()
```

```
Train set shape (100746, 30)
Validation set shape (67164, 30)
```

[18]:
```
        sales(t-29)  sales(t-28)  sales(t-27)  sales(t-26)  sales(t-25)  \
18801          97.0        111.0         90.0        115.0        123.0
160385         38.0         43.0         43.0         55.0         47.0
73123          55.0         45.0         41.0         46.0         47.0
90428         139.0        157.0         85.0         99.0        136.0
167151         86.0         58.0         88.0         87.0        114.0

        sales(t-24)  sales(t-23)  sales(t-22)  sales(t-21)  sales(t-20)  …  \
18801          70.0         99.0         74.0        107.0        108.0  …
160385         51.0         38.0         41.0         37.0         59.0  …
73123          36.0         30.0         46.0         41.0         42.0  …
90428         110.0        121.0        123.0        147.0         91.0  …
167151        113.0         64.0         76.0         87.0         81.0  …

        sales(t-9)  sales(t-8)  sales(t-7)  sales(t-6)  sales(t-5)  \
18801         85.0        95.0       123.0       109.0       127.0
160385        41.0        38.0        38.0        53.0        53.0
73123         38.0        36.0        40.0        50.0        44.0
90428        130.0       128.0       128.0        95.0       116.0
167151        55.0        66.0        59.0        53.0        63.0

        sales(t-4)  sales(t-3)  sales(t-2)  sales(t-1)  sales(t)
```

```
18801          132.0        87.0       101.0       102.0      114.0
160385          45.0        44.0        24.0        30.0       37.0
73123           44.0        40.0        38.0        50.0       49.0
90428          110.0       117.0       118.0       129.0      132.0
167151          59.0        77.0        39.0        56.0       62.0

[5 rows x 30 columns]
```

[ ]:
```
MLP for Time Series Forecasting
First we will use a Multilayer Perceptron model or MLP model, here our model␣
 ↪will have input features equal to the window size.
The thing with MLP models is that the model don't take the input as sequenced␣
 ↪data, so for the model, it is just receiving inputs and don't treat them as␣
 ↪sequenced data, that may be a problem since the model won't see the data␣
 ↪with the sequence patter that it has.
Input shape [samples, timesteps].
```

[19]:
```
epochs = 40
batch = 256
lr = 0.0003
adam = optimizers.Adam(lr)
```

[20]:
```
model_mlp = Sequential()
model_mlp.add(Dense(100, activation='relu', input_dim=X_train.shape[1]))
model_mlp.add(Dense(1))
model_mlp.compile(loss='mse', optimizer=adam)
```

[21]:
```
mlp_history = model_mlp.fit(X_train.values, Y_train, validation_data=(X_valid.
 ↪values, Y_valid), epochs=epochs, verbose=2)
```

```
Epoch 1/40
3149/3149 - 4s - 1ms/step - loss: 398.5072 - val_loss: 372.4298
Epoch 2/40
3149/3149 - 3s - 908us/step - loss: 366.0605 - val_loss: 359.1979
Epoch 3/40
3149/3149 - 3s - 847us/step - loss: 357.9803 - val_loss: 355.4276
Epoch 4/40
3149/3149 - 3s - 876us/step - loss: 355.0424 - val_loss: 354.5063
Epoch 5/40
3149/3149 - 3s - 863us/step - loss: 353.4018 - val_loss: 352.0721
Epoch 6/40
3149/3149 - 5s - 2ms/step - loss: 352.3002 - val_loss: 351.3567
Epoch 7/40
3149/3149 - 3s - 872us/step - loss: 351.3085 - val_loss: 350.8668
Epoch 8/40
3149/3149 - 3s - 876us/step - loss: 350.6004 - val_loss: 349.9607
Epoch 9/40
```

```
3149/3149 - 3s - 872us/step - loss: 349.8129 - val_loss: 349.8660
Epoch 10/40
3149/3149 - 3s - 872us/step - loss: 349.1013 - val_loss: 349.0649
Epoch 11/40
3149/3149 - 3s - 907us/step - loss: 348.5551 - val_loss: 348.6874
Epoch 12/40
3149/3149 - 3s - 872us/step - loss: 347.8871 - val_loss: 348.3574
Epoch 13/40
3149/3149 - 3s - 899us/step - loss: 347.3745 - val_loss: 347.7258
Epoch 14/40
3149/3149 - 3s - 870us/step - loss: 346.9056 - val_loss: 347.1882
Epoch 15/40
3149/3149 - 3s - 847us/step - loss: 346.3364 - val_loss: 347.2220
Epoch 16/40
3149/3149 - 3s - 853us/step - loss: 345.9473 - val_loss: 347.1063
Epoch 17/40
3149/3149 - 3s - 909us/step - loss: 345.4112 - val_loss: 346.5426
Epoch 18/40
3149/3149 - 3s - 901us/step - loss: 344.9576 - val_loss: 346.0747
Epoch 19/40
3149/3149 - 3s - 961us/step - loss: 344.5040 - val_loss: 345.5160
Epoch 20/40
3149/3149 - 3s - 1ms/step - loss: 344.1411 - val_loss: 345.5179
Epoch 21/40
3149/3149 - 3s - 1ms/step - loss: 343.7550 - val_loss: 345.2640
Epoch 22/40
3149/3149 - 3s - 957us/step - loss: 343.4117 - val_loss: 344.8588
Epoch 23/40
3149/3149 - 3s - 955us/step - loss: 343.1496 - val_loss: 344.9653
Epoch 24/40
3149/3149 - 3s - 1ms/step - loss: 342.7938 - val_loss: 344.7141
Epoch 25/40
3149/3149 - 3s - 927us/step - loss: 342.6217 - val_loss: 344.7766
Epoch 26/40
3149/3149 - 3s - 1ms/step - loss: 342.3199 - val_loss: 344.7184
Epoch 27/40
3149/3149 - 3s - 1ms/step - loss: 342.0515 - val_loss: 344.8938
Epoch 28/40
3149/3149 - 3s - 1ms/step - loss: 341.8879 - val_loss: 345.3483
Epoch 29/40
3149/3149 - 3s - 1ms/step - loss: 341.5847 - val_loss: 345.6905
Epoch 30/40
3149/3149 - 3s - 1ms/step - loss: 341.3428 - val_loss: 345.4123
Epoch 31/40
3149/3149 - 3s - 1ms/step - loss: 341.1367 - val_loss: 345.3517
Epoch 32/40
3149/3149 - 3s - 1ms/step - loss: 340.8497 - val_loss: 345.4098
Epoch 33/40
```

```
3149/3149 - 5s - 1ms/step - loss: 340.6091 - val_loss: 345.5343
Epoch 34/40
3149/3149 - 3s - 869us/step - loss: 340.4507 - val_loss: 345.7410
Epoch 35/40
3149/3149 - 3s - 877us/step - loss: 340.2572 - val_loss: 345.3567
Epoch 36/40
3149/3149 - 3s - 932us/step - loss: 340.0451 - val_loss: 344.6165
Epoch 37/40
3149/3149 - 3s - 985us/step - loss: 339.9380 - val_loss: 345.4362
Epoch 38/40
3149/3149 - 3s - 984us/step - loss: 339.6248 - val_loss: 345.4581
Epoch 39/40
3149/3149 - 3s - 1ms/step - loss: 339.4048 - val_loss: 345.9838
Epoch 40/40
3149/3149 - 3s - 1ms/step - loss: 339.2747 - val_loss: 345.7278
```

[ ]:
```
CNN for Time Series Forecasting
For the CNN model we will use one convolutional hidden layer followed by a max␣
 ↪pooling layer. The filter maps are then flattened before being interpreted␣
 ↪by a Dense layer and outputting a prediction.
The convolutional layer should be able to identify patterns between the␣
 ↪timesteps.
Input shape [samples, timesteps, features].
Data preprocess
Reshape from [samples, timesteps] into [samples, timesteps, features].
This same reshaped data will be used on the CNN and the LSTM model.
```

[22]:
```
X_train_series = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))
X_valid_series = X_valid.values.reshape((X_valid.shape[0], X_valid.shape[1], 1))
print('Train set shape', X_train_series.shape)
print('Validation set shape', X_valid_series.shape)
```

```
Train set shape (100746, 30, 1)
Validation set shape (67164, 30, 1)
```

[24]:
```
# Ensure warnings are ignored
import warnings
warnings.filterwarnings("ignore")

# Initialize plotly notebook mode
import plotly.offline as py
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)

# Set seeds for reproducibility
from tensorflow.compat.v1 import set_random_seed
from numpy.random import seed
```

```python
set_random_seed(1)
seed(1)

# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff

# Define your model
model_cnn = Sequential()
model_cnn.add(Conv1D(filters=64, kernel_size=2, activation='relu',␣
 ↪input_shape=(X_train_series.shape[1], X_train_series.shape[2])))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(50, activation='relu'))
model_cnn.add(Dense(1))

# Define the optimizer
adam = Adam()

# Compile the model
model_cnn.compile(loss='mse', optimizer=adam)

# Fit the model
cnn_history = model_cnn.fit(X_train_series, Y_train,␣
 ↪validation_data=(X_valid_series, Y_valid), epochs=epochs, verbose=2)
```

```
Epoch 1/40
3149/3149 - 8s - 2ms/step - loss: 414.0200 - val_loss: 419.8466
Epoch 2/40
3149/3149 - 7s - 2ms/step - loss: 398.1718 - val_loss: 409.6941
Epoch 3/40
3149/3149 - 6s - 2ms/step - loss: 391.0981 - val_loss: 396.3741
Epoch 4/40
3149/3149 - 6s - 2ms/step - loss: 373.6004 - val_loss: 373.4311
```

```
Epoch 5/40
3149/3149 - 6s - 2ms/step - loss: 364.5082 - val_loss: 369.3141
Epoch 6/40
3149/3149 - 6s - 2ms/step - loss: 361.6595 - val_loss: 366.7163
Epoch 7/40
3149/3149 - 6s - 2ms/step - loss: 360.1163 - val_loss: 360.3351
Epoch 8/40
3149/3149 - 8s - 3ms/step - loss: 358.9750 - val_loss: 358.8175
Epoch 9/40
3149/3149 - 6s - 2ms/step - loss: 357.7983 - val_loss: 357.8640
Epoch 10/40
3149/3149 - 6s - 2ms/step - loss: 357.7818 - val_loss: 356.1794
Epoch 11/40
3149/3149 - 7s - 2ms/step - loss: 356.2879 - val_loss: 355.1981
Epoch 12/40
3149/3149 - 8s - 3ms/step - loss: 355.3861 - val_loss: 354.4974
Epoch 13/40
3149/3149 - 7s - 2ms/step - loss: 355.0943 - val_loss: 354.7577
Epoch 14/40
3149/3149 - 9s - 3ms/step - loss: 354.2922 - val_loss: 352.7209
Epoch 15/40
3149/3149 - 12s - 4ms/step - loss: 353.6633 - val_loss: 353.7318
Epoch 16/40
3149/3149 - 10s - 3ms/step - loss: 353.1401 - val_loss: 352.8508
Epoch 17/40
3149/3149 - 9s - 3ms/step - loss: 352.7124 - val_loss: 353.1982
Epoch 18/40
3149/3149 - 9s - 3ms/step - loss: 352.4015 - val_loss: 354.2747
Epoch 19/40
3149/3149 - 8s - 2ms/step - loss: 351.8135 - val_loss: 353.9106
Epoch 20/40
3149/3149 - 7s - 2ms/step - loss: 351.5551 - val_loss: 352.2484
Epoch 21/40
3149/3149 - 7s - 2ms/step - loss: 351.3721 - val_loss: 350.6328
Epoch 22/40
3149/3149 - 8s - 2ms/step - loss: 350.8748 - val_loss: 352.9432
Epoch 23/40
3149/3149 - 7s - 2ms/step - loss: 350.7489 - val_loss: 351.1460
Epoch 24/40
3149/3149 - 7s - 2ms/step - loss: 350.2922 - val_loss: 352.3184
Epoch 25/40
3149/3149 - 7s - 2ms/step - loss: 350.3054 - val_loss: 352.3377
Epoch 26/40
3149/3149 - 7s - 2ms/step - loss: 349.8884 - val_loss: 350.8921
Epoch 27/40
3149/3149 - 8s - 2ms/step - loss: 349.7169 - val_loss: 352.8795
Epoch 28/40
3149/3149 - 9s - 3ms/step - loss: 349.1921 - val_loss: 351.9212
```

```
Epoch 29/40
3149/3149 - 9s - 3ms/step - loss: 349.0923 - val_loss: 351.6557
Epoch 30/40
3149/3149 - 7s - 2ms/step - loss: 348.9189 - val_loss: 350.1138
Epoch 31/40
3149/3149 - 8s - 2ms/step - loss: 348.5568 - val_loss: 351.7531
Epoch 32/40
3149/3149 - 7s - 2ms/step - loss: 348.3507 - val_loss: 351.5276
Epoch 33/40
3149/3149 - 7s - 2ms/step - loss: 348.2560 - val_loss: 349.7443
Epoch 34/40
3149/3149 - 7s - 2ms/step - loss: 347.8695 - val_loss: 349.0282
Epoch 35/40
3149/3149 - 7s - 2ms/step - loss: 347.8053 - val_loss: 350.4536
Epoch 36/40
3149/3149 - 8s - 3ms/step - loss: 347.7515 - val_loss: 349.6104
Epoch 37/40
3149/3149 - 7s - 2ms/step - loss: 347.4572 - val_loss: 348.1065
Epoch 38/40
3149/3149 - 7s - 2ms/step - loss: 347.2387 - val_loss: 349.3612
Epoch 39/40
3149/3149 - 7s - 2ms/step - loss: 347.1899 - val_loss: 348.8058
Epoch 40/40
3149/3149 - 7s - 2ms/step - loss: 346.8064 - val_loss: 348.5835
```

[ ]:
```
LSTM for Time Series Forecasting
Now the LSTM model actually sees the input data as a sequence, so it's able to learn patterns from sequenced data (assuming it exists) better than the other ones, especially patterns from long sequences.
Input shape [samples, timesteps, features].
```

[26]:
```
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, LSTM
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls
```

```python
import plotly.figure_factory as ff

# Ignore warnings
warnings.filterwarnings("ignore")

# Initialize plotly notebook mode
py.init_notebook_mode(connected=True)
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)

# Set seeds for reproducibility
from tensorflow.compat.v1 import set_random_seed
from numpy.random import seed
set_random_seed(1)
seed(1)

# Example data (replace with your actual data)
# X_train_series = ...
# Y_train = ...
# X_valid_series = ...
# Y_valid = ...
# epochs = ...

# Define and compile LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(50, activation='relu', input_shape=(X_train_series.
 ↪shape[1], X_train_series.shape[2])))
model_lstm.add(Dense(1))
adam_lstm = Adam()
model_lstm.compile(loss='mse', optimizer=adam_lstm)
lstm_history = model_lstm.fit(X_train_series, Y_train,␣
 ↪validation_data=(X_valid_series, Y_valid), epochs=epochs, verbose=2)
```

```
Epoch 1/40
3149/3149 - 25s - 8ms/step - loss: 909.9219 - val_loss: 586.7662
Epoch 2/40
3149/3149 - 27s - 9ms/step - loss: 483.8970 - val_loss: 434.0353
Epoch 3/40
3149/3149 - 31s - 10ms/step - loss: 443.2530 - val_loss: 479.1456
Epoch 4/40
3149/3149 - 30s - 10ms/step - loss: 418.9510 - val_loss: 380.8773
Epoch 5/40
3149/3149 - 32s - 10ms/step - loss: 389.4897 - val_loss: 373.5066
Epoch 6/40
3149/3149 - 31s - 10ms/step - loss: 413.4998 - val_loss: 699.3168
Epoch 7/40
3149/3149 - 30s - 9ms/step - loss: 411.3925 - val_loss: 386.6947
```

```
Epoch 8/40
3149/3149 - 30s - 10ms/step - loss: 386.3672 - val_loss: 386.5006
Epoch 9/40
3149/3149 - 30s - 9ms/step - loss: 273307.5000 - val_loss: 5248.6890
Epoch 10/40
3149/3149 - 30s - 10ms/step - loss: 2507.7153 - val_loss: 794.2533
Epoch 11/40
3149/3149 - 30s - 10ms/step - loss: 535.1879 - val_loss: 439.1197
Epoch 12/40
3149/3149 - 30s - 9ms/step - loss: 430.0691 - val_loss: 418.4175
Epoch 13/40
3149/3149 - 30s - 9ms/step - loss: 414.8895 - val_loss: 415.6528
Epoch 14/40
3149/3149 - 30s - 9ms/step - loss: 421.8434 - val_loss: 514.1371
Epoch 15/40
3149/3149 - 30s - 10ms/step - loss: 260749.5000 - val_loss: 551.3553
Epoch 16/40
3149/3149 - 30s - 10ms/step - loss: 522.8534 - val_loss: 488.4822
Epoch 17/40
3149/3149 - 29s - 9ms/step - loss: 7170.4995 - val_loss: 685.0638
Epoch 18/40
3149/3149 - 29s - 9ms/step - loss: 21784.2598 - val_loss: 1257.7423
Epoch 19/40
3149/3149 - 29s - 9ms/step - loss: 2816.6499 - val_loss: 1667.3778
Epoch 20/40
3149/3149 - 31s - 10ms/step - loss: 2651.1990 - val_loss: 526.4847
Epoch 21/40
3149/3149 - 30s - 9ms/step - loss: 485.6635 - val_loss: 458.8011
Epoch 22/40
3149/3149 - 27s - 8ms/step - loss: 1983.4930 - val_loss: 526.5444
Epoch 23/40
3149/3149 - 28s - 9ms/step - loss: 514.0348 - val_loss: 497.9482
Epoch 24/40
3149/3149 - 26s - 8ms/step - loss: 510.1153 - val_loss: 537.6179
Epoch 25/40
3149/3149 - 30s - 10ms/step - loss: 494.6291 - val_loss: 483.7385
Epoch 26/40
3149/3149 - 28s - 9ms/step - loss: 481.6730 - val_loss: 485.2057
Epoch 27/40
3149/3149 - 31s - 10ms/step - loss: 475.2910 - val_loss: 469.6256
Epoch 28/40
3149/3149 - 28s - 9ms/step - loss: 512.9554 - val_loss: 491.5516
Epoch 29/40
3149/3149 - 29s - 9ms/step - loss: 707.2020 - val_loss: 524.2556
Epoch 30/40
3149/3149 - 28s - 9ms/step - loss: 519.7422 - val_loss: 508.4399
Epoch 31/40
3149/3149 - 29s - 9ms/step - loss: 505.1311 - val_loss: 498.4631
```

```
Epoch 32/40
3149/3149 - 29s - 9ms/step - loss: 481.1415 - val_loss: 465.1991
Epoch 33/40
3149/3149 - 29s - 9ms/step - loss: 453.4642 - val_loss: 452.7737
Epoch 34/40
3149/3149 - 29s - 9ms/step - loss: 452.8546 - val_loss: 410.5185
Epoch 35/40
3149/3149 - 30s - 10ms/step - loss: 406.4594 - val_loss: 390.3579
Epoch 36/40
3149/3149 - 30s - 10ms/step - loss: 388.2645 - val_loss: 397.9205
Epoch 37/40
3149/3149 - 29s - 9ms/step - loss: 379.9992 - val_loss: 375.6209
Epoch 38/40
3149/3149 - 35s - 11ms/step - loss: 372.2477 - val_loss: 359.8785
Epoch 39/40
3149/3149 - 33s - 10ms/step - loss: 365.5259 - val_loss: 356.9262
Epoch 40/40
3149/3149 - 31s - 10ms/step - loss: 364.8278 - val_loss: 355.5641
```

[ ]:
```
CNN-LSTM for Time Series Forecasting
Input shape [samples, subsequences, timesteps, features].
Model explanation from the article
"The benefit of this model is that the model can support very long input␣
↪sequences that can be read as blocks or subsequences by the CNN model, then␣
↪pieced together by the LSTM model."

"When using a hybrid CNN-LSTM model, we will further divide each sample into␣
↪further subsequences. The CNN model will interpret each sub-sequence and the␣
↪LSTM will piece together the interpretations from the subsequences. As such,␣
↪we will split each sample into 2 subsequences of 2 times per subsequence."

"The CNN will be defined to expect 2 timesteps per subsequence with one feature.
↪ The entire CNN model is then wrapped in TimeDistributed wrapper layers so␣
↪that it can be applied to each subsequence in the sample. The results are␣
↪then interpreted by the LSTM layer before the model outputs a prediction."

Data preprocess
Reshape from [samples, timesteps, features] into [samples, subsequences,␣
↪timesteps, features].
```

[27]:
```
subsequences = 2
timesteps = X_train_series.shape[1]//subsequences
X_train_series_sub = X_train_series.reshape((X_train_series.shape[0],␣
↪subsequences, timesteps, 1))
X_valid_series_sub = X_valid_series.reshape((X_valid_series.shape[0],␣
↪subsequences, timesteps, 1))
print('Train set shape', X_train_series_sub.shape)
```

```
print('Validation set shape', X_valid_series_sub.shape)
```

```
Train set shape (100746, 2, 15, 1)
Validation set shape (67164, 2, 15, 1)
```

[29]:
```
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, LSTM,␣
↪TimeDistributed
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff

# Ignore warnings
warnings.filterwarnings("ignore")

# Initialize plotly notebook mode
py.init_notebook_mode(connected=True)
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)

# Set seeds for reproducibility
from tensorflow.compat.v1 import set_random_seed
from numpy.random import seed
set_random_seed(1)
seed(1)

# Example data (replace with your actual data)
# X_train_series_sub = ...
# Y_train = ...
# X_valid_series_sub = ...
# Y_valid = ...
# epochs = ...

# Define and compile CNN-LSTM model
model_cnn_lstm = Sequential()
```

```python
model_cnn_lstm.add(TimeDistributed(Conv1D(filters=64, kernel_size=1,␣
 ↪activation='relu'), input_shape=(None, X_train_series_sub.shape[2],␣
 ↪X_train_series_sub.shape[3])))
model_cnn_lstm.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model_cnn_lstm.add(TimeDistributed(Flatten()))
model_cnn_lstm.add(LSTM(50, activation='relu'))
model_cnn_lstm.add(Dense(1))

# Create a new instance of the Adam optimizer
adam_cnn_lstm = Adam()

# Compile the model
model_cnn_lstm.compile(loss='mse', optimizer=adam_cnn_lstm)

# Fit the model
cnn_lstm_history = model_cnn_lstm.fit(X_train_series_sub, Y_train,␣
 ↪validation_data=(X_valid_series_sub, Y_valid), epochs=epochs, verbose=2)
```

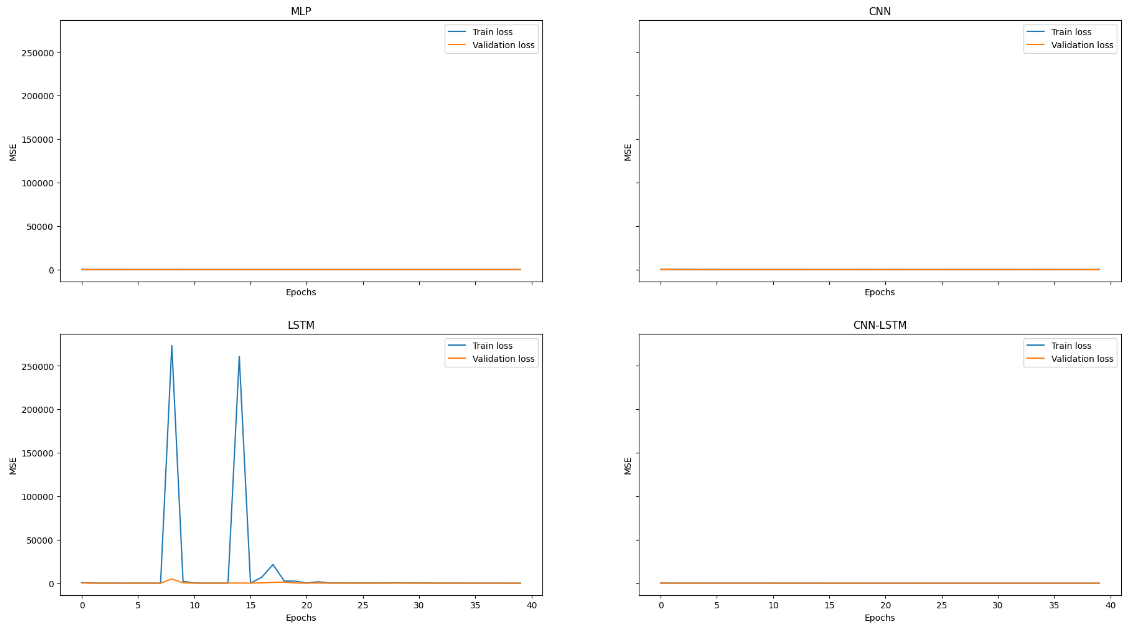
```
Epoch 1/40
3149/3149 - 20s - 6ms/step - loss: 437.0952 - val_loss: 420.5929
Epoch 2/40
3149/3149 - 17s - 5ms/step - loss: 401.1300 - val_loss: 394.3987
Epoch 3/40
3149/3149 - 17s - 5ms/step - loss: 390.0017 - val_loss: 388.5772
Epoch 4/40
3149/3149 - 17s - 5ms/step - loss: 387.3000 - val_loss: 385.2106
Epoch 5/40
3149/3149 - 18s - 6ms/step - loss: 385.1880 - val_loss: 380.1925
Epoch 6/40
3149/3149 - 17s - 5ms/step - loss: 382.8643 - val_loss: 376.2307
Epoch 7/40
3149/3149 - 17s - 5ms/step - loss: 379.2541 - val_loss: 373.4170
Epoch 8/40
3149/3149 - 18s - 6ms/step - loss: 375.3331 - val_loss: 369.3830
Epoch 9/40
3149/3149 - 18s - 6ms/step - loss: 371.7783 - val_loss: 364.8410
Epoch 10/40
3149/3149 - 16s - 5ms/step - loss: 368.9900 - val_loss: 362.2651
Epoch 11/40
3149/3149 - 17s - 6ms/step - loss: 366.5654 - val_loss: 359.7098
Epoch 12/40
3149/3149 - 18s - 6ms/step - loss: 364.4789 - val_loss: 359.4063
Epoch 13/40
3149/3149 - 18s - 6ms/step - loss: 362.7087 - val_loss: 357.6419
Epoch 14/40
3149/3149 - 17s - 5ms/step - loss: 361.3062 - val_loss: 356.2316
Epoch 15/40
```

```
3149/3149 - 18s - 6ms/step - loss: 359.9373 - val_loss: 355.9712
Epoch 16/40
3149/3149 - 18s - 6ms/step - loss: 359.0404 - val_loss: 354.9264
Epoch 17/40
3149/3149 - 17s - 5ms/step - loss: 358.1956 - val_loss: 353.8113
Epoch 18/40
3149/3149 - 18s - 6ms/step - loss: 357.4348 - val_loss: 355.0538
Epoch 19/40
3149/3149 - 19s - 6ms/step - loss: 356.8337 - val_loss: 352.8374
Epoch 20/40
3149/3149 - 17s - 5ms/step - loss: 355.8318 - val_loss: 352.6701
Epoch 21/40
3149/3149 - 17s - 5ms/step - loss: 355.3688 - val_loss: 353.1726
Epoch 22/40
3149/3149 - 18s - 6ms/step - loss: 354.8550 - val_loss: 353.2291
Epoch 23/40
3149/3149 - 18s - 6ms/step - loss: 354.2146 - val_loss: 352.5253
Epoch 24/40
3149/3149 - 17s - 5ms/step - loss: 353.7151 - val_loss: 352.6392
Epoch 25/40
3149/3149 - 22s - 7ms/step - loss: 353.3473 - val_loss: 351.6013
Epoch 26/40
3149/3149 - 20s - 6ms/step - loss: 352.9847 - val_loss: 350.9227
Epoch 27/40
3149/3149 - 18s - 6ms/step - loss: 352.4960 - val_loss: 350.5976
Epoch 28/40
3149/3149 - 19s - 6ms/step - loss: 352.1978 - val_loss: 350.4117
Epoch 29/40
3149/3149 - 18s - 6ms/step - loss: 351.8609 - val_loss: 350.0376
Epoch 30/40
3149/3149 - 18s - 6ms/step - loss: 351.7127 - val_loss: 349.9763
Epoch 31/40
3149/3149 - 19s - 6ms/step - loss: 351.3547 - val_loss: 349.3951
Epoch 32/40
3149/3149 - 19s - 6ms/step - loss: 351.1198 - val_loss: 349.6141
Epoch 33/40
3149/3149 - 18s - 6ms/step - loss: 351.1051 - val_loss: 349.0998
Epoch 34/40
3149/3149 - 18s - 6ms/step - loss: 350.7587 - val_loss: 349.2901
Epoch 35/40
3149/3149 - 18s - 6ms/step - loss: 350.7732 - val_loss: 349.3761
Epoch 36/40
3149/3149 - 19s - 6ms/step - loss: 350.5367 - val_loss: 349.5945
Epoch 37/40
3149/3149 - 18s - 6ms/step - loss: 350.3488 - val_loss: 349.9347
Epoch 38/40
3149/3149 - 19s - 6ms/step - loss: 350.2121 - val_loss: 348.7955
Epoch 39/40
```

```
3149/3149 - 21s - 7ms/step - loss: 349.9438 - val_loss: 349.4185
Epoch 40/40
3149/3149 - 19s - 6ms/step - loss: 350.0501 - val_loss: 349.7572
```

[30]:

```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True,figsize=(22,12))
ax1, ax2 = axes[0]
ax3, ax4 = axes[1]

ax1.plot(mlp_history.history['loss'], label='Train loss')
ax1.plot(mlp_history.history['val_loss'], label='Validation loss')
ax1.legend(loc='best')
ax1.set_title('MLP')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('MSE')

ax2.plot(cnn_history.history['loss'], label='Train loss')
ax2.plot(cnn_history.history['val_loss'], label='Validation loss')
ax2.legend(loc='best')
ax2.set_title('CNN')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('MSE')

ax3.plot(lstm_history.history['loss'], label='Train loss')
ax3.plot(lstm_history.history['val_loss'], label='Validation loss')
ax3.legend(loc='best')
ax3.set_title('LSTM')
ax3.set_xlabel('Epochs')
ax3.set_ylabel('MSE')

ax4.plot(cnn_lstm_history.history['loss'], label='Train loss')
ax4.plot(cnn_lstm_history.history['val_loss'], label='Validation loss')
ax4.legend(loc='best')
ax4.set_title('CNN-LSTM')
ax4.set_xlabel('Epochs')
ax4.set_ylabel('MSE')

plt.show()
```




[31]:
```
#MLP on train and validation
mlp_train_pred = model_mlp.predict(X_train.values)
mlp_valid_pred = model_mlp.predict(X_valid.values)
print('Train rmse:', np.sqrt(mean_squared_error(Y_train, mlp_train_pred)))
print('Validation rmse:', np.sqrt(mean_squared_error(Y_valid, mlp_valid_pred)))
```

```
3149/3149              2s 725us/step
2099/2099              1s 547us/step
Train rmse: 18.46580861442587
Validation rmse: 18.593740123172513
```

[32]:
```
#CNN on train and validation
cnn_train_pred = model_cnn.predict(X_train_series)
cnn_valid_pred = model_cnn.predict(X_valid_series)
print('Train rmse:', np.sqrt(mean_squared_error(Y_train, cnn_train_pred)))
print('Validation rmse:', np.sqrt(mean_squared_error(Y_valid, cnn_valid_pred)))
```

```
3149/3149              4s 1ms/step
2099/2099              2s 1ms/step
Train rmse: 18.572942897110625
Validation rmse: 18.67039267697171
```

[33]:
```
#LSTM on train and validation
lstm_train_pred = model_lstm.predict(X_train_series)
lstm_valid_pred = model_cnn.predict(X_valid_series)
print('Train rmse:', np.sqrt(mean_squared_error(Y_train, lstm_train_pred)))
print('Validation rmse:', np.sqrt(mean_squared_error(Y_valid, lstm_valid_pred)))
```

```
3149/3149              11s 4ms/step
2099/2099              2s 1ms/step
Train rmse: 18.803703086279103
Validation rmse: 18.67039267697171
```

[34]:

```
#CNN-LSTM on train and validation
cnn_lstm_train_pred = model_cnn_lstm.predict(X_train_series_sub)
cnn_lstm_valid_pred = model_cnn_lstm.predict(X_valid_series_sub)
print('Train rmse:', np.sqrt(mean_squared_error(Y_train, cnn_lstm_train_pred)))
print('Validation rmse:', np.sqrt(mean_squared_error(Y_valid,
 ↪cnn_lstm_valid_pred)))
```

```
3149/3149              5s 2ms/step
2099/2099              3s 2ms/step
Train rmse: 18.661503405810066
Validation rmse: 18.70179923571742
```

[ ]:

```
```