Time series analysis for IT system using LSTM model

Thomas Selin

Aim of the project?

- **Predict usage of a IT service** (for example an mobile phone app)

Why?

- Important when choosing when to do maintenance (the IT service might not working for the users during this time) or risky changes, to affect as few users as possible

- Important when provisioning hardware resources/scaling the service to be able to handle the expected nr. of users. You don’t want to over-provision/over-scale as it is expensive so a good forecast can be very valuable.

2 version of the same dataset.

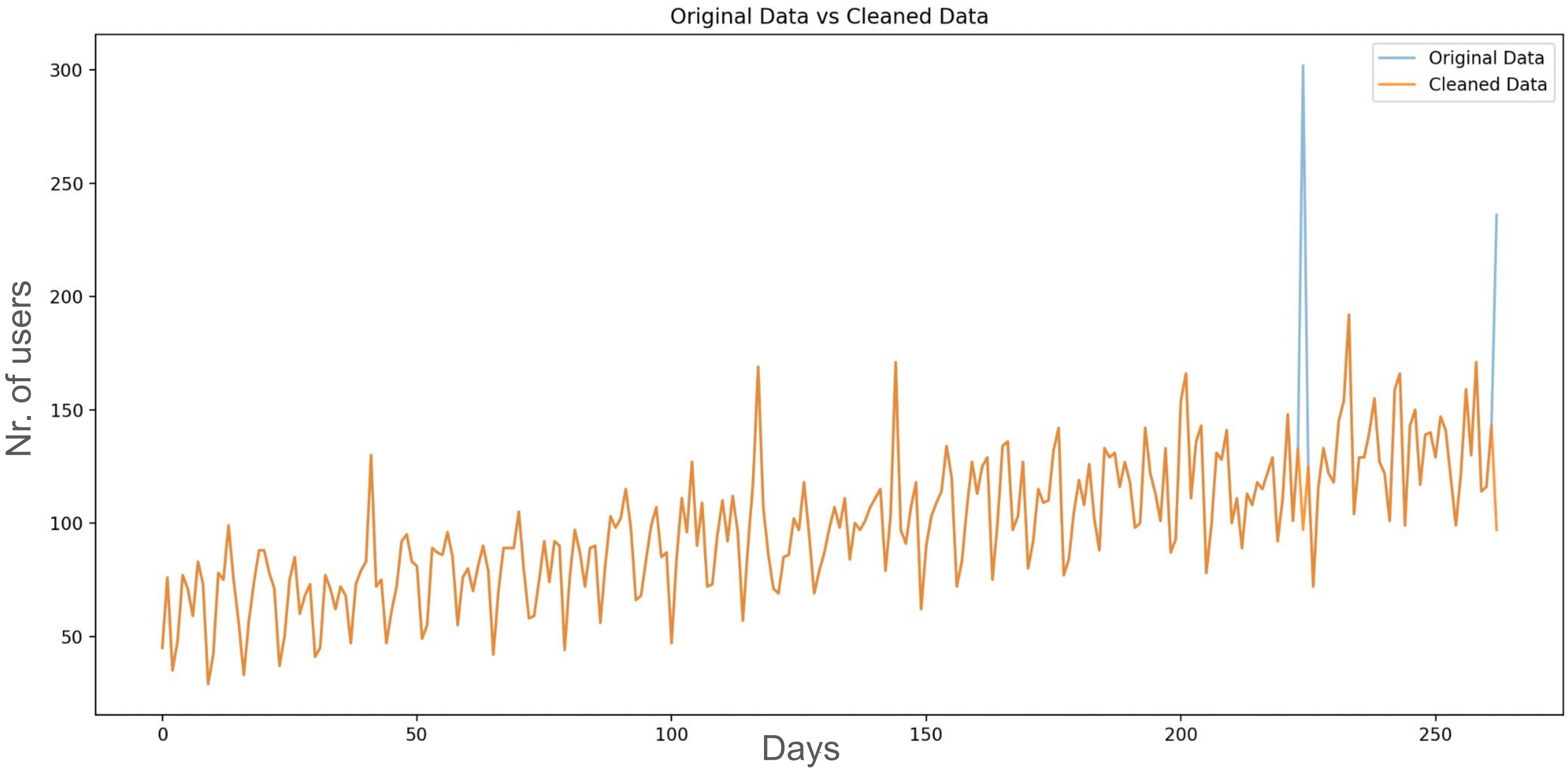
Daily:
hour with max nr. users for each day
263 values

users
45
76
35
48
77
71
59
83
73
29
42
78
and so on...

Hourly
6300 values

users
10
13
17
26
31
32
43
45
53
59
65
67
and so on...

Cleaning of data: Daily (263 values)



Mean absolute error (MAE):

- Gives equal weight to all errors regardless of size
- Keep error metric in same units as your target variable, more explainable

Mean Squared Error (MSE):

- Penalizes large errors more heavily
- Has higher sensitivity to outliers

Conclusion

Depending on your IT system, MAE or MSE could be more important. I will compare both.
For ex. MSE would be better if consistent high performance of the IT system is important, as it penalizes outliers heavily.

Daily data.

Version 1

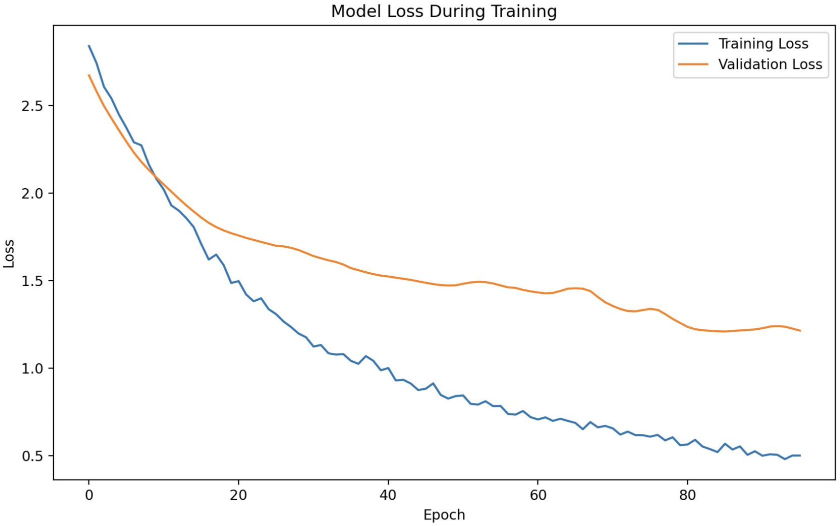
```
model = Sequential([
    LSTM(32, input_shape=(1, lookback), return_sequences=False,
         kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    Dropout(0.2),
    Dense(8, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    Dense(1)
])
```

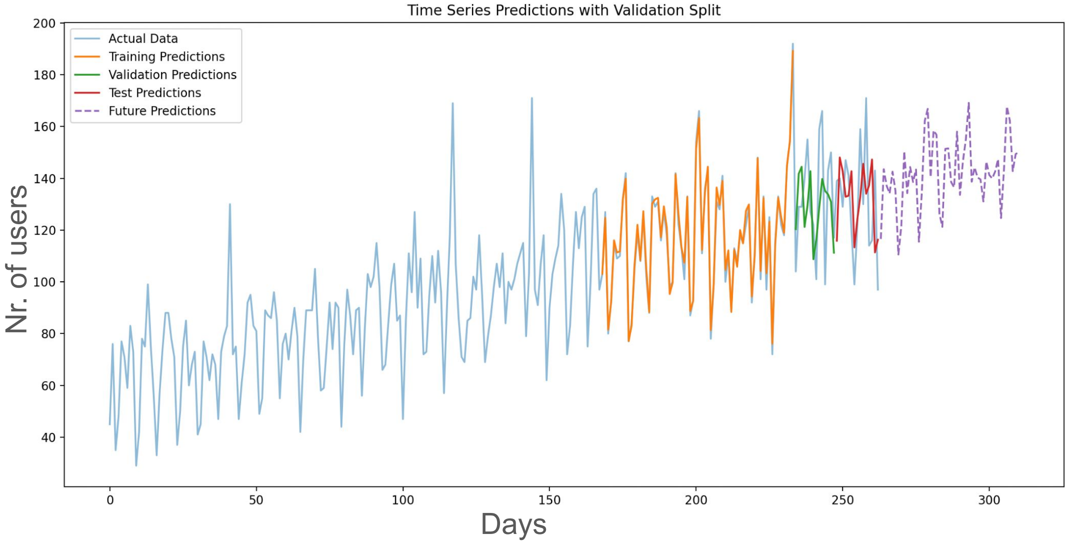
```
Epoch 95: val_loss did not improve from 1.24016
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.4855 - val_loss: 1.2871
Epoch 95: early stopping
Restoring model weights from the end of the best epoch: 85.
```

```
Training Metrics:
MSE: 28.82
MAE: 3.08

Validation Metrics:
MSE: 306.01
MAE: 15.26

Test Metrics:
MSE: 487.88
MAE: 18.56
```

Conclusion:

- Probably overfitted
- Try fewer nodes
- Try more dropout





Daily data.

Version 2

```
model = Sequential([
    LSTM(16, input_shape=(1, lookback), return_sequences=False,
         kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(8, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(1)
])
```

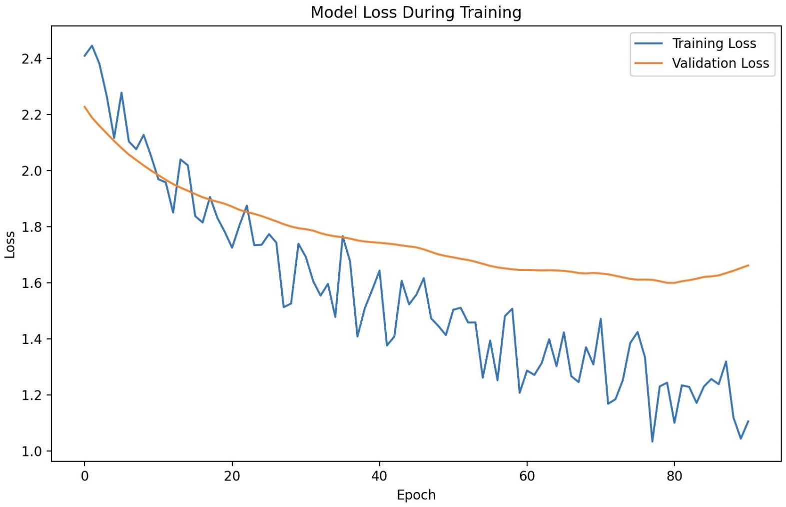
```
Training Metrics:
MSE: 160.40
MAE: 9.73

Validation Metrics:
MSE: 362.80
MAE: 15.25

Test Metrics:
MSE: 398.82
MAE: 17.06
```

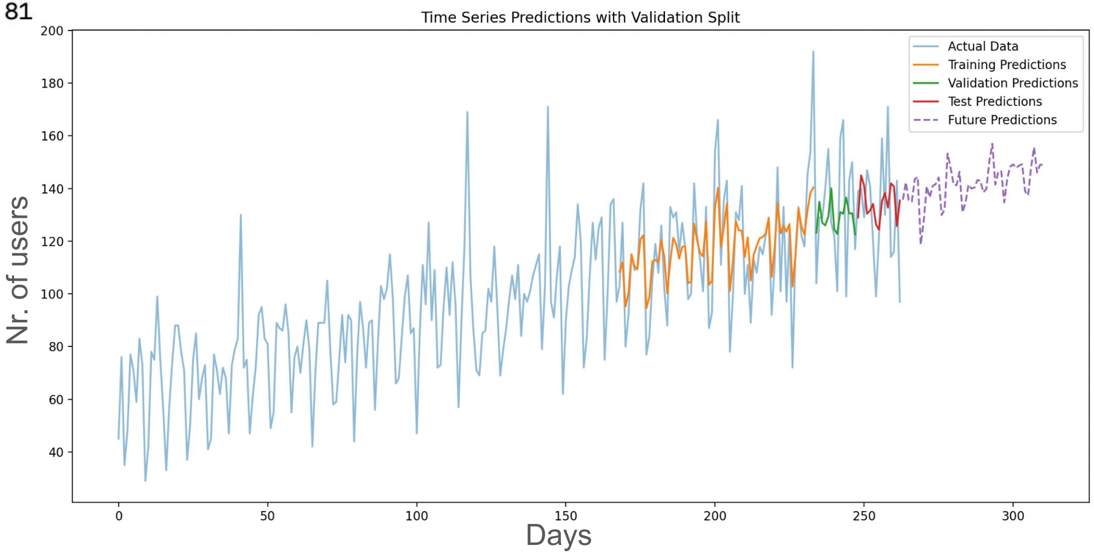
Conclusion:

- Less overfitted

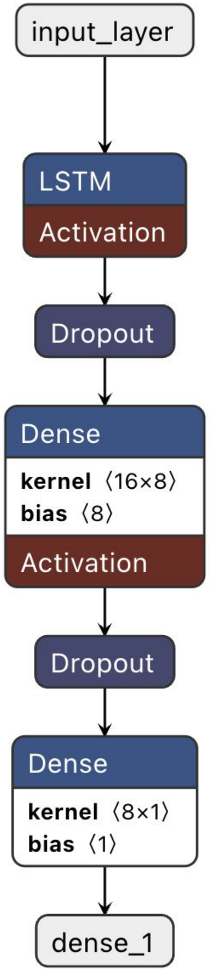
- Better than model on last slide

```
Epoch 91/150
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.1000
Epoch 91: val_loss did not improve from 1.59955
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.1071 - val_loss: 1.6617
Epoch 91: early stopping
Restoring model weights from the end of the best epoch: 81
```

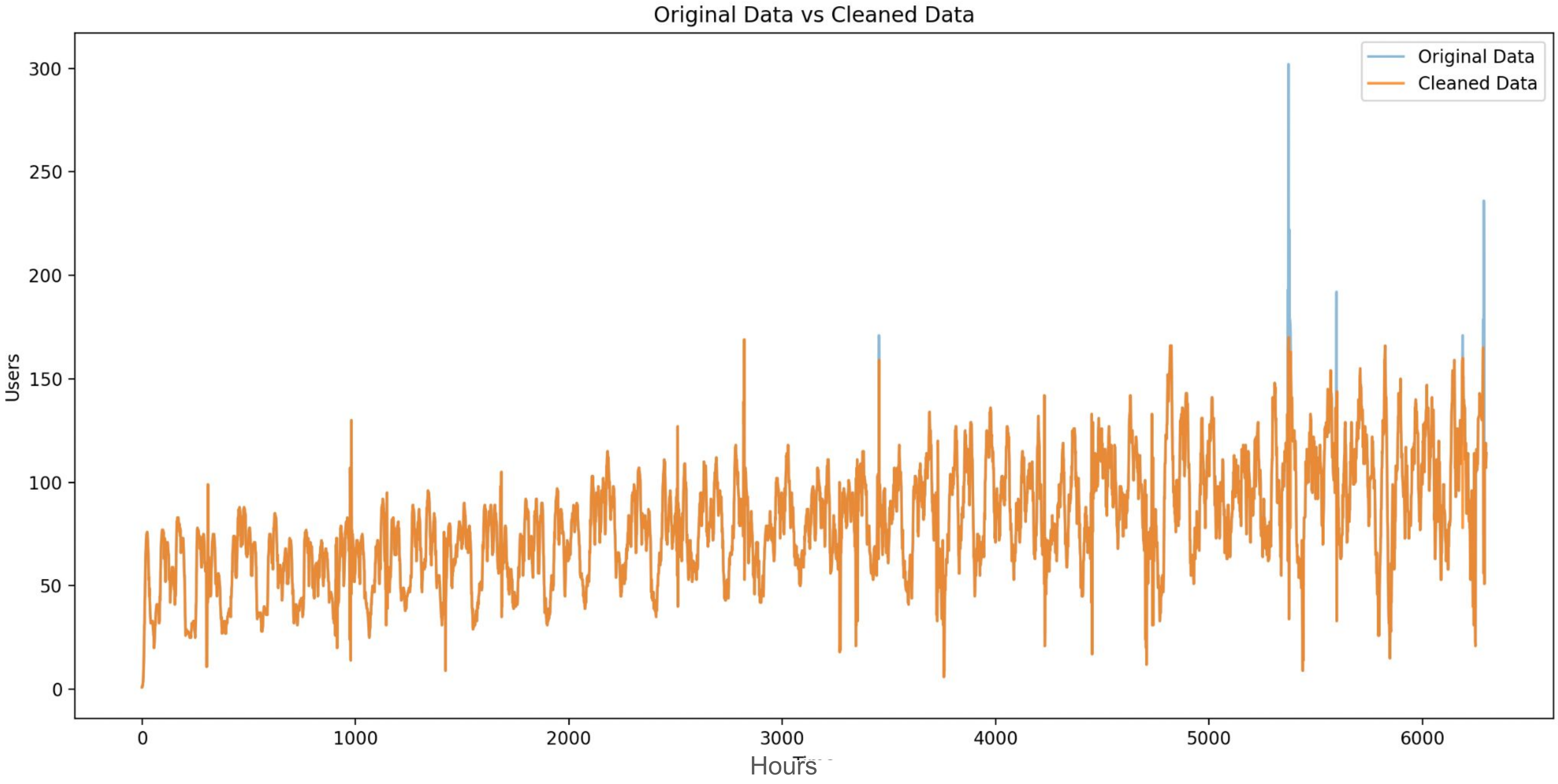




Current model architecture



Cleaning of data: Hourly (6300 values)



Hourly data.

Version 1

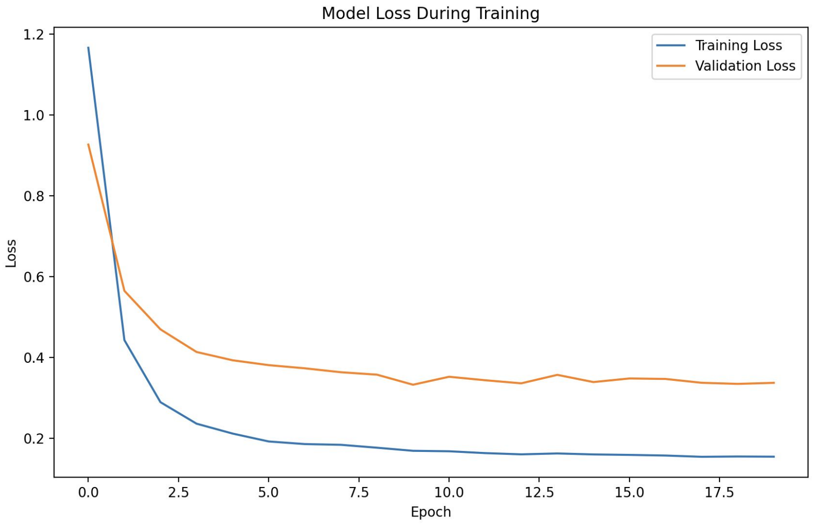
```
model = Sequential([
    LSTM(32, input_shape=(1, lookback), return_sequences=False,
         kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    Dropout(0.2),
    Dense(8, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    Dense(1)
])
```

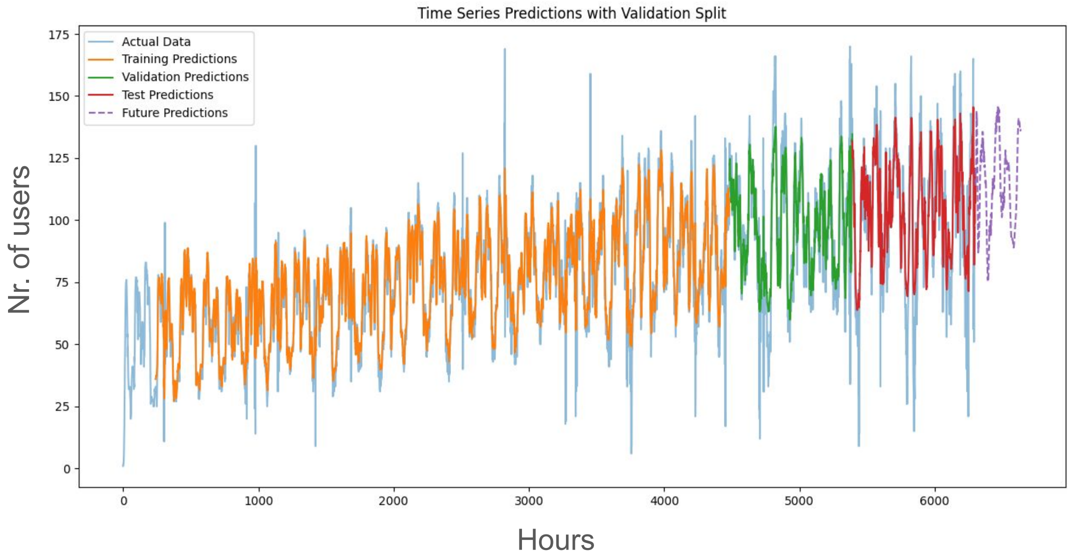
```
Training Metrics:
MSE: 56.99
MAE: 4.64

Validation Metrics:
MSE: 138.92
MAE: 7.75

Test Metrics:
MSE: 202.82
MAE: 10.23
```

Conclusion:

- Better than model on daily dataset
- Some overfitting
- Try fewer nodes

```
Epoch 40: val_loss did not improve from 0.32188
135/135 ━━━━━━━━━━━━━━━━━━━━ 0s 555us/step - loss: 0.1626 - val_loss: 0.3368
Epoch 40: early stopping
Restoring model weights from the end of the best epoch: 30.
```





Hourly data.

Version 2

```
model = Sequential([
    LSTM(16, input_shape=(1, lookback), return_sequences=False,
         kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(8, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(1)
])
```

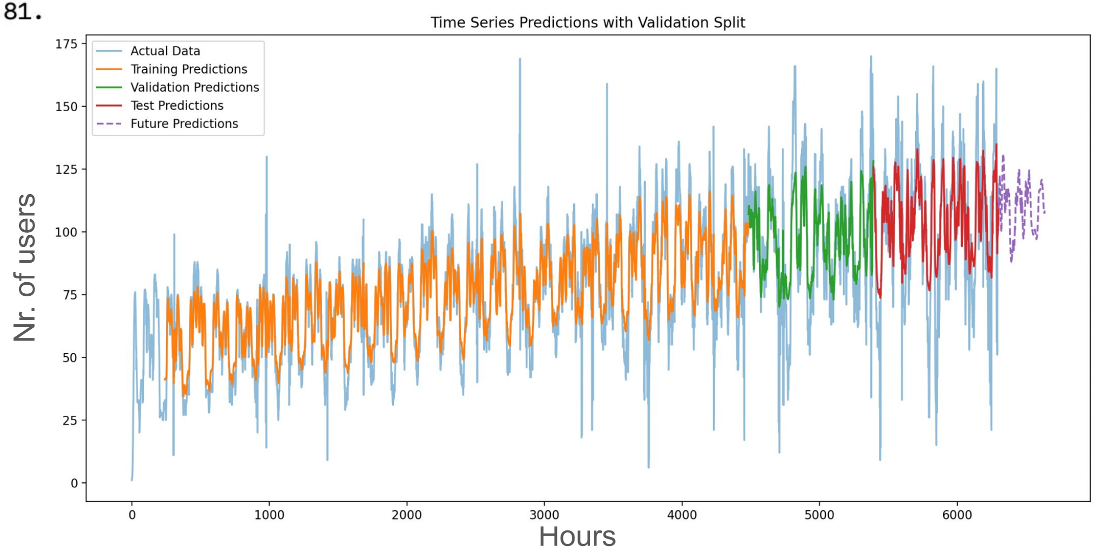
```
Training Metrics:
MSE: 77.67
MAE: 6.03

Validation Metrics:
MSE: 219.20
MAE: 10.44

Test Metrics:
MSE: 292.74
MAE: 12.76
```

Conclusion:

- More overfitted

- Worse than model on last slide

```
Epoch 91/150
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.1000
Epoch 91: val_loss did not improve from 1.59955
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.1071 - val_loss: 1.6617
Epoch 91: early stopping
Restoring model weights from the end of the best epoch: 81.
```



Main learnings

Complex data benefits from more nodes: The hourly data benefited from more nodes

Experimenting with lessening overfitting is important: It is worth trying several methods, e.g. dropout, nr. of nodes/layers

Early stoppage was helpful: Helps a lot when experimenting to find good models

Model checkpoint was helpful: Helps a lot when experimenting to find good models

More data (hourly), better model: More data contains more information about the trends

```
# Callbacks for early stopping and choosing best model from training history
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    ),
    ModelCheckpoint(
        'best_model.keras',
        monitor='val_loss',
        save_best_only=True,
        verbose=1
    )
]
```

(page 1/10): Code in python programming language.

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error, mean_absolute_error

def remove_outliers(data, threshold=1.2):
   """Remove outliers using IQR method"""
   Q1 = np.percentile(data, 25)
   Q3 = np.percentile(data, 75)
   IQR = Q3 - Q1

   lower_bound = Q1 - threshold * IQR
   upper_bound = Q3 + threshold * IQR

   mask = (data >= lower_bound) & (data <= upper_bound)
   data_cleaned = data.copy()
   median_value = np.median(data[mask])
   data_cleaned[~mask] = median_value

   return data_cleaned
```

(page 2/10): Code in python programming language.

```
def detrend_data(data):
    """Remove trend from data and return trend parameters"""
    X = np.arange(len(data)).reshape(-1, 1)
    from sklearn.linear_model import LinearRegression
    trend_model = LinearRegression()
    trend_model.fit(X, data)
    trend = trend_model.predict(X)
    detrended_data = data - trend
    return detrended_data, trend_model

def create_sequences(data, lookback, step=1):
    """Create sequences for time series prediction with overlapping"""
    sequences = []
    targets = []
    for i in range(0, len(data) - lookback, step):
        sequences.append(data[i:i+lookback])
        targets.append(data[i+lookback])
    return np.array(sequences), np.array(targets)

# Load and prepare data
# users = pd.read_csv('daily.csv', header=0) # Uncomment to use the daily version of the dataset. Make sure that the future_steps and lookback variables is in daily time unit
users = pd.read_csv('hourly.csv', header=0) # Uncomment to use the hourly version of the dataset. Make sure that the future_steps and lookback variables is in hourly time unit
data = users.values.reshape(-1, 1)
```

(page 3/10): Code in python programming language.

```
# Remove outliers
data_cleaned = remove_outliers(data, threshold=2)

# Plot original and cleaned data
plt.figure(figsize=(15, 7))
plt.plot(data, label='Original Data', alpha=0.5)
plt.plot(data_cleaned, label='Cleaned Data', alpha=0.8)
plt.title('Original Data vs Cleaned Data')
plt.xlabel('Time')
plt.ylabel('Users')
plt.legend()
plt.show()

detrended_data, trend_model = detrend_data(data_cleaned)

# Scale the detrended data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(detrended_data)

# Create all sequences first
lookback = 24 * 10  # nr of days/hours lookback
sequences, targets = create_sequences(scaled_data, lookback, step=1)

# Now split the sequences into train, validation, and test sets
train_size = int(len(sequences) * 0.7)
val_size = int(len(sequences) * 0.15)
```

(page 4/10): Code in python programming language.

```
# Split sequences
train_x = sequences[:train_size]
train_y = targets[:train_size]

val_x = sequences[train_size:train_size + val_size]
val_y = targets[train_size:train_size + val_size]

test_x = sequences[train_size + val_size:]
test_y = targets[train_size + val_size:]

# Reshape for LSTM (samples, timesteps, features)
train_x = np.reshape(train_x, (train_x.shape[0], 1, train_x.shape[1]))
val_x = np.reshape(val_x, (val_x.shape[0], 1, val_x.shape[1]))
test_x = np.reshape(test_x, (test_x.shape[0], 1, test_x.shape[1]))

# One of the used model architectures. It was the best arch for the hourly dataset. See ../project_report.pdf for more details
model = Sequential([
   LSTM(32, input_shape=(1, lookback), return_sequences=False,
        kernel_regularizer=l2(0.01)),
   Dropout(0.2),
   Dense(8, activation='relu', kernel_regularizer=l2(0.01)),
   Dense(1)
])
```

(page 5/10): Code in python programming language.

```python
# # One of the used model architectures. It was the best arch for the daily dataset. See ../project_report.pdf for more details.
# model = Sequential([
#     LSTM(16, input_shape=(1, lookback), return_sequences=False,
#          kernel_regularizer=l2(0.01)),
#     Dropout(0.5),
#     Dense(8, activation='relu', kernel_regularizer=l2(0.01)),
#     Dropout(0.5),
#     Dense(1)
# ])

model.compile(
   loss="mean_squared_error",
   optimizer=Adam(learning_rate=0.001)
)
```

(page 6/10): Code in python programming language.

```
# Callbacks for early stopping and choosing best model from training history
callbacks = [
   EarlyStopping(
       monitor='val_loss',
       patience=10,
       restore_best_weights=True,
       verbose=1
   ),
   ModelCheckpoint(
       'best_model.keras',
       monitor='val_loss',
       save_best_only=True,
       verbose=1
   )
]

# Train model with validation data
history = model.fit(
   train_x, train_y,
   epochs=150,
   batch_size=32,
   validation_data=(val_x, val_y),
   callbacks=callbacks,
   verbose=1
)
```

(page 7/10): Code in python programming language.

```
# Plot training history
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss During Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Make predictions
train_pred = model.predict(train_x)
val_pred = model.predict(val_x)
test_pred = model.predict(test_x)

# Inverse transform predictions
train_pred = scaler.inverse_transform(train_pred)
val_pred = scaler.inverse_transform(val_pred)
test_pred = scaler.inverse_transform(test_pred)

# Generate indices for plotting
train_indices = np.arange(lookback, lookback + len(train_pred))
val_indices = np.arange(lookback + len(train_pred), lookback + len(train_pred) + len(val_pred))
test_indices = np.arange(lookback + len(train_pred) + len(val_pred), lookback + len(train_pred) + len(val_pred) + len(test_pred))
```

(page 8/10): Code in python programming language.

```
# Add trend back
train_trend = trend_model.predict(train_indices.reshape(-1, 1))
val_trend = trend_model.predict(val_indices.reshape(-1, 1))
test_trend = trend_model.predict(test_indices.reshape(-1, 1))

train_pred_final = train_pred + train_trend
val_pred_final = val_pred + val_trend
test_pred_final = test_pred + test_trend

# Calculate and print metrics
def print_metrics(y_true, y_pred, dataset_name):
   mse = mean_squared_error(y_true, y_pred)
   mae = mean_absolute_error(y_true, y_pred)
   print(f"\n{dataset_name} Metrics:")
   print(f"MSE: {mse:.2f}")
   print(f"MAE: {mae:.2f}")

# Calculate metrics for each dataset
print_metrics(data_cleaned[train_indices], train_pred_final, "Training")
print_metrics(data_cleaned[val_indices], val_pred_final, "Validation")
print_metrics(data_cleaned[test_indices], test_pred_final, "Test")

# Make future predictions
last_sequence = test_x[-1]
future_predictions = []
future_steps = 24 * 14 # nr of days or hours future predictions
```

(page 9/10): Code in python programming language.

```
for _ in range(future_steps):
    # Predict next value
    next_pred = model.predict(last_sequence.reshape(1, 1, -1))
    future_predictions.append(next_pred[0, 0])

    # Update sequence
    last_sequence = np.roll(last_sequence, -1)
    last_sequence[0, -1] = next_pred

# Scale back future predictions and add trend
future_pred_array = np.array(future_predictions).reshape(-1, 1)
future_pred_unscaled = scaler.inverse_transform(future_pred_array)

future_indices = np.arange(
    test_indices[-1] + 1,
    test_indices[-1] + 1 + len(future_predictions)
).reshape(-1, 1)

future_trend = trend_model.predict(future_indices)
future_pred_with_trend = future_pred_unscaled + future_trend
```

(page 10/10): Code in python programming language.

```python
# Plot results
plt.figure(figsize=(15, 7))
plt.plot(data_cleaned, label='Actual Data', alpha=0.5)
plt.plot(train_indices, train_pred_final, label='Training Predictions')
plt.plot(val_indices, val_pred_final, label='Validation Predictions')
plt.plot(test_indices, test_pred_final, label='Test Predictions')
plt.title('Time Series Predictions with Validation Split')
# Plot future predictions
plt.plot(future_indices, future_pred_with_trend, label='Future Predictions', linestyle='--')
plt.legend()
plt.show()
```