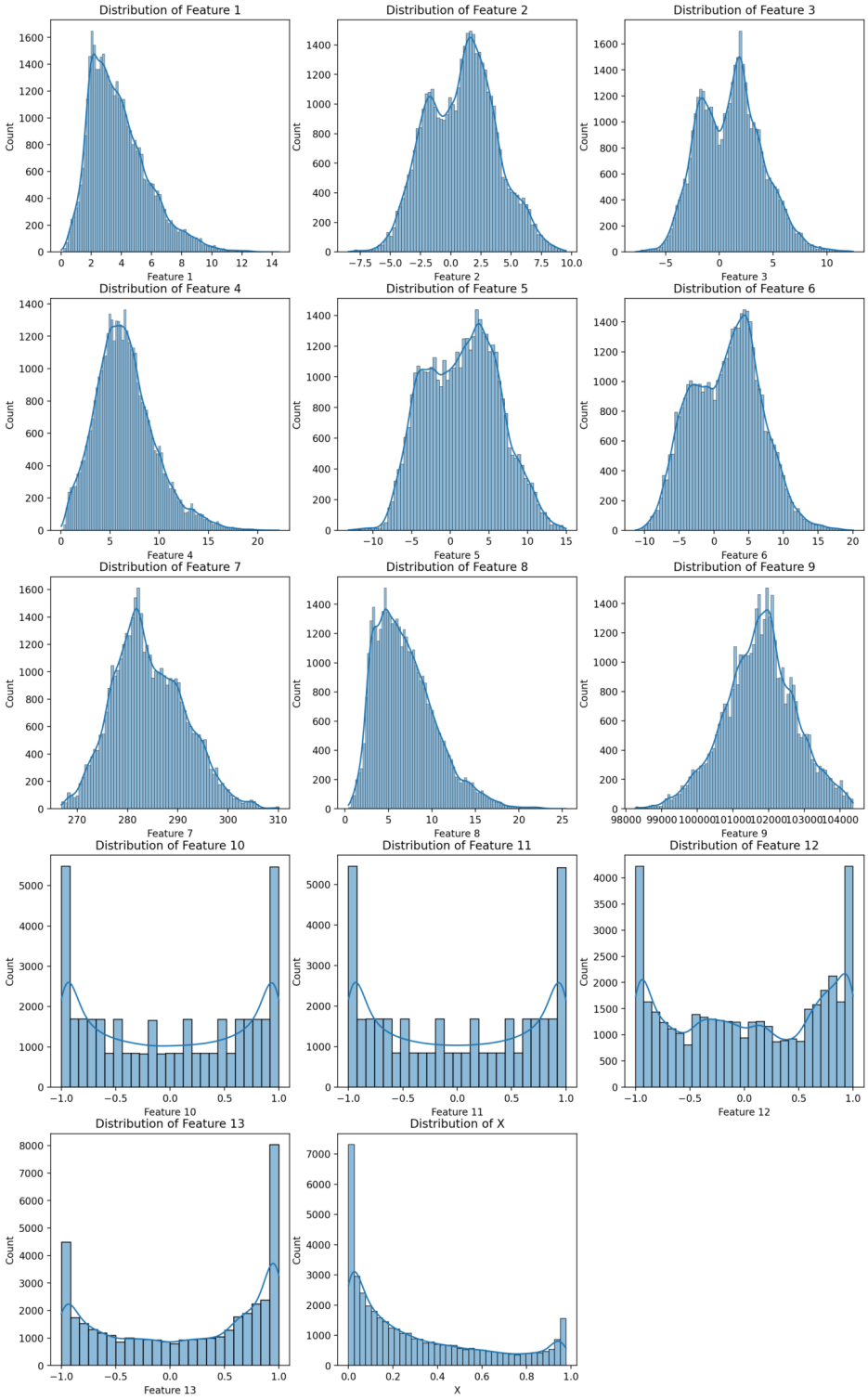
Inspecting the data

Date	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8
2021-12-02 00:00:00	4.1365	-1.6996	3.7711	7.7968	-4.0888	6.6387	277.105	7.1576
2021-12-02 00:15:00	4.1472	-1.7423	3.7634	7.7566	-4.1116	6.5772	277.024	7.1866
2021-12-02 00:30:00	4.1579	-1.785	3.7553	7.7164	-4.1338	6.5156	276.943	7.2157
2021-12-02 00:45:00	4.1686	-1.8277	3.7466	7.6761	-4.1554	6.4541	276.862	7.2447
2021-12-02 01:00:00	4.1798	-1.8706	3.7379	7.6354	-4.176	6.3923	276.78	7.2738
2021-12-02 01:15:00	4.1445	-1.8689	3.6992	7.6274	-4.1918	6.3723	276.779	7.171
2021-12-02 01:30:00	4.1092	-1.8669	3.6606	7.6193	-4.2076	6.3522	276.778	7.0681
2021-12-02 01:45:00	4.0739	-1.8646	3.6221	7.6113	-4.2233	6.3321	276.777	6.9653
2021-12-02 02:00:00	4.0395	-1.8626	3.5844	7.6037	-4.2393	6.3123	276.774	6.863
2021-12-02 02:15:00	4.1485	-1.9709	3.6505	7.7414	-4.4318	6.3473	276.731	7.0695

missing values: Feature 1 0 Feature 2 0 Feature 3 0 Feature 4 0 Feature 5 0 Feature 6 0 Feature 7 0 Feature 8 0 Feature 9 0 Feature 10 0 Feature 11 0 Feature 12 0 Feature 13 0 X 0 dtype: int64

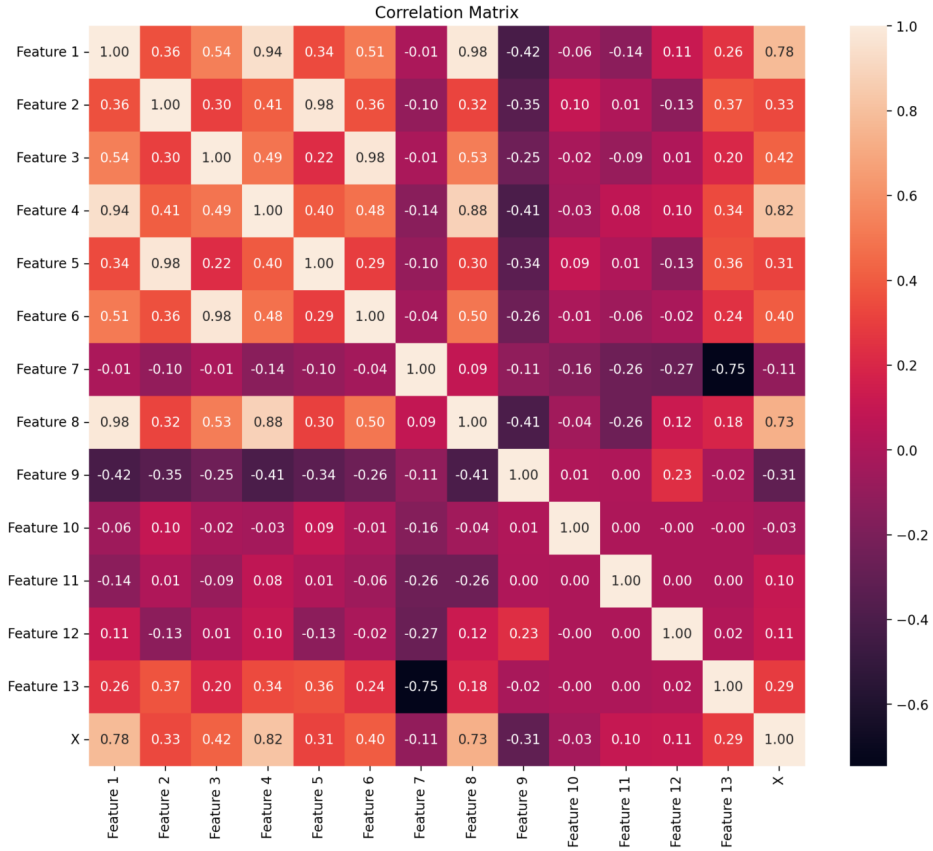
no missing values

Data Distribution of Features



Features 10-13 are already scaled. The rest will need to be scaled. The data is normally distributed in the other features. we can use a minmax scaler to get it on the same scale as features 10-13

Feature Correlation Matrix

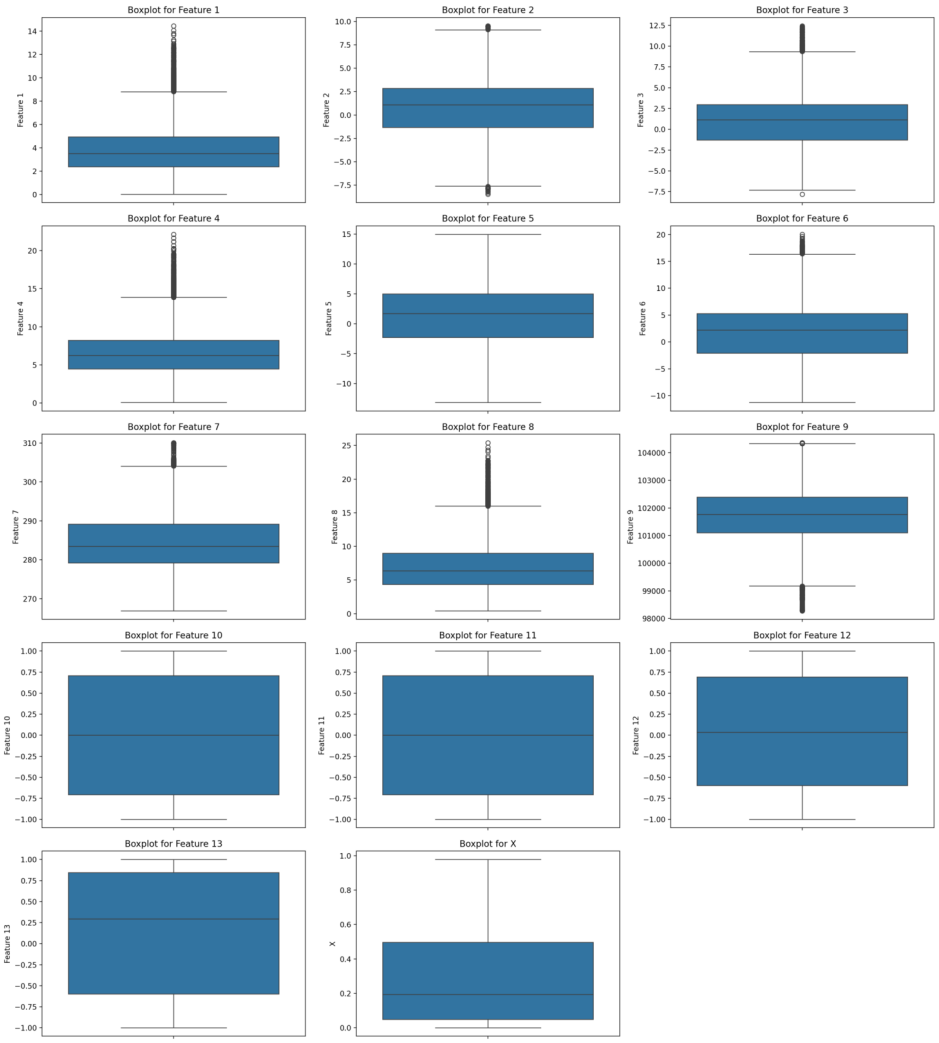


We can see that some features are highly correlated with the target, and some are not. We also see that some features are highly correlated with each other

Features, 1,4, and 8 are highly correlated with each other, and they all are highly correlated with the target

Features 2 and 5, and features 3 and 6 are highly correlated with each other as well. We may be able to remove or combine some of these redundant features to improve our model, but it is difficult to know what we can remove or manipulate without context of the data

Check for outliers with boxplots



We can see that there are outliers on a few of the features, but no major outliers. features 1,4,8 especially have some outliers. It is hard to tell without context whether these extreme values are important for predicting the target

for now we can go ahead without removing them, and come back to this later.

Pre-processing the data

Now that we've had a good look at the data, we can preprocess it and then train a base model

As mentioned, we will scale features 1-9

Train model

we are dealing with 15-minute time-series data. in order to predict the target at some time, we likely will have to look back a certain number of timesteps

We can use a LSTM model in order to train the model on this temporal information

however, without context of the data, we cannot know exactly what time window to use for training. We can test different time windows

After training the model several times with different hyperparameters, I found the following to be the most optimal:

	0
time_steps	12

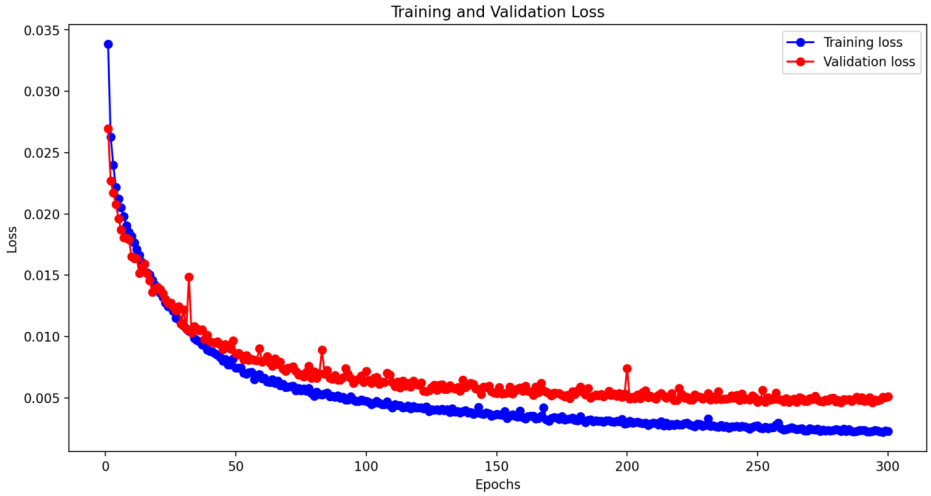
	0
epochs	300
learning rate	0.0005
LSTM layers	[100, 50, 50, 1]
featuers dropped	[2, 7, 10]

Split the model into Train, Val, test sets (70%, 15%, 15%)

Training...

```
None
```

```
0.004591948352754116
```



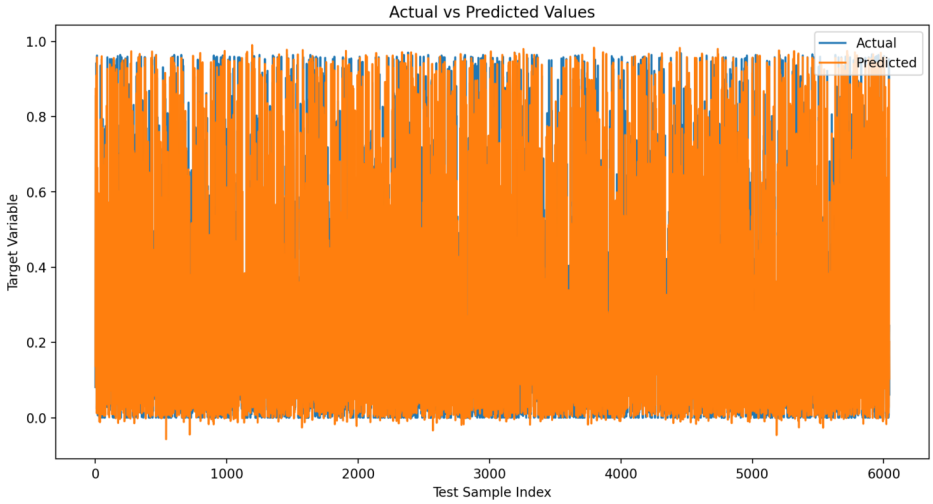
The training and validation loss are improving steadily overtime, with no signs of overfitting

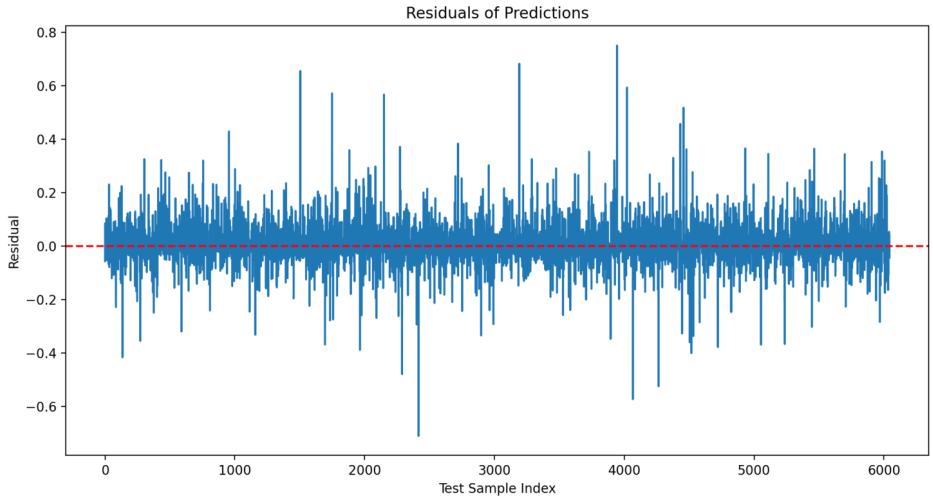
All of the metrics are good. The MSE is not much higher than the MAE, which means that the errors in the predictions are not very large

An R2 score of 0.945 is quite high, indicating that the model explains a large portion of the variance in the data

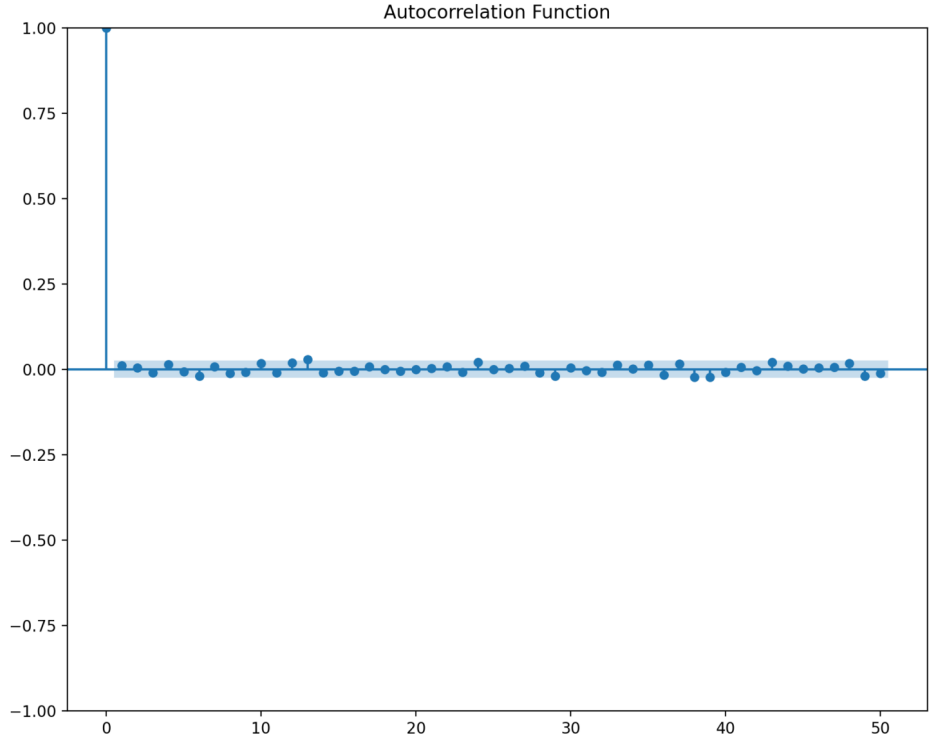
let's use the model generate predictions on the test data, and some analysis on the results

	0
MAE	0.0459
MSE	0.0055
RMSE	0.0741
R2	0.9383





Durbin-Watson Statistic: 1.9464880544126975



The residuals plot seems to be quite uniform. This means that the model captures the variance in the data quite well.

The autocorrelation plot, and the durban-watson statistic indicate that there is no autocorrelation in the residuals

This means that the residuals are mostly noise, and are not correlated to any time-frames in the time series

improvements

Overall, the results are solid, but the model can be improved, and the results can be further analyzed

In terms of the results, it would be beneficial to further analyze them to detect time-specific patterns. It's possible that my errors have a pattern of occuring at certain times of the day, month, or year.

This analysis would help guide improvements of the model. For example, an ensemble method could be used, where one model is trained on specific time periods, or time windows, and its results are combined with those of another model

My current results show that the model was learning well, and was continuing to improve at 300 epochs. With more time, I would have liked to experiment with more complex model architectures and longer training periods. It would be interesting to also test different models, such as a transformer.

I could have also performed further feature engineering: Adding features from rolling statistics in the data may help the model learn if there are patterns in the data. Additional combinations or removal of features may help learning as well.

Additional Question: 3D data

If I was given similar data, but for 100 points on a 10x10 grid, I would take a similar approach but with a convolutional neural net. This would help not only capture the temporal relationships in the data, but it would learn the spatial relationships as well

In order to achieve this, I would use tensorflow's ConvLSTM. This maintains the LSTM's core functionality and purpose, but uses convolutional layers in its gates, allowing the model to capture spatial data

The preprocessing of the data would be very similar, but with 3D data instead of 2D. This would result in data with the shape: (batch_size, timesteps, 10, 10), which can be fed into the LSTM

The implementation of the model would be as follows:

```
model=Sequential([
    #create a convolutional layer for each input (time_steps, 10x10 grid)
    #adjust the filter size, kernel size, and normalization / pooling depending on
    TimeDistributed(Conv2D(filter, kernel, activation), input_shape=(12,10,10))
    TimeDistributed(BatchNormalization())
    TimeDistributed(MaxPooling2D(pool_size))

    # Now, the LSTM layers will capture relationships in the data across time / sp
    ConvLSTM2D(filters, kernel)
    ConvLSTM2D(filters, kernel)
    # add more layers if needed

    # output a 10x10 grid of predictions, or whatever shape you want the predictio
    Dense(10,10)

])
```