25/08/2024

Predicting Traffic Volume

Time and Weather Impacts

Team Member:

Haya Almalki

Jehan Almutairi

Hanan Mohammed

INTRODUCTION

The "Traffic Volume Dataset" represents data collected from a highway in Minneapolis, Minnesota, USA, spanning the years 2012 to 2018. This dataset includes a variety of key variables designed to provide a comprehensive view of traffic patterns in the area.

Key variables include:

Traffic Volume: The number of cars passing a specific point on the road within a specific hour.

Weather Data: Includes information on temperature, rainfall, snowfall, and cloud cover percentage.

Date and Time: Includes details about the hour and day of the week to capture temporal patterns.

Weather Description: Provides text descriptions of weather conditions such as "Clear," "Cloudy," "Rain," and more.

THE OBJECTIVES

- **Objective:** To build a model that can predict future traffic volume based on historical data and time-related and weather-related features.
- **Importance:** This model aims to enhance traffic management, reduce congestion, and improve road safety.

DATASET

We uploaded the dataset from Kaggle called (**https://www.kaggle.com/datasets/rohith203/traffic-volume-dataset**), which contains.

- Data Loading and Preprocessing
- The data was downloaded from Kaggle using the following command:
- !kaggle datasets download -d rohith203/traffic-volume-dataset
- After downloading, the " 'Train.csv' " file was read into a DataFrame using pandas:
- df = pd.read_csv('Train.csv')

DATA PREPROCESSING

Exploratory Data Analysis (EDA) & Preprocessing:

- Dropped the "is_holiday" column due to a high percentage of missing values.
- Extracted features from the "date_time" column.
- Removed outliers in "rain_p_h" and "temperature."
- Visualized the distribution of categorical features like "weather_type" and "weather_description."

Data Splitting

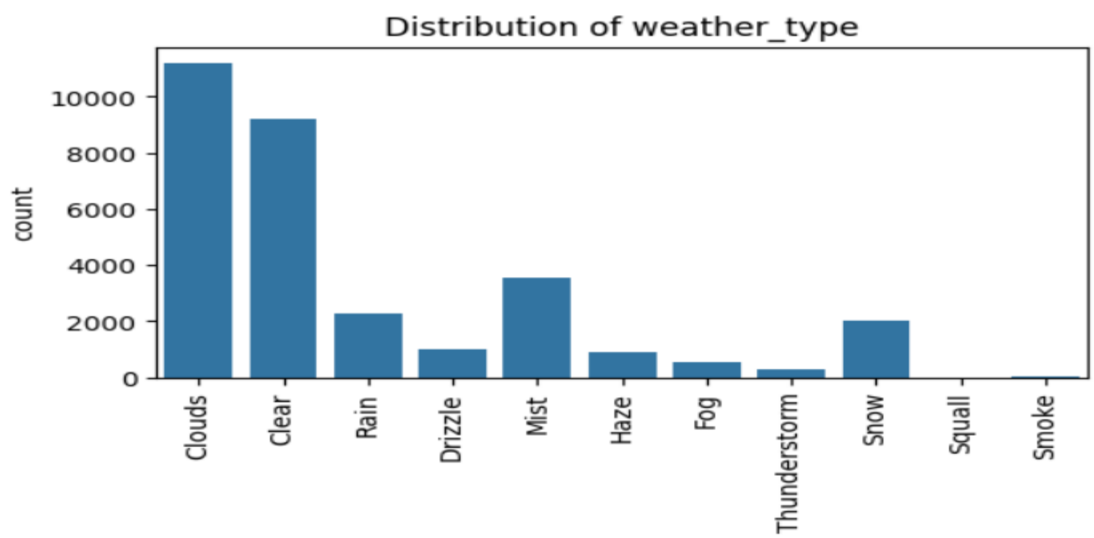
The time series data was split into two sets:

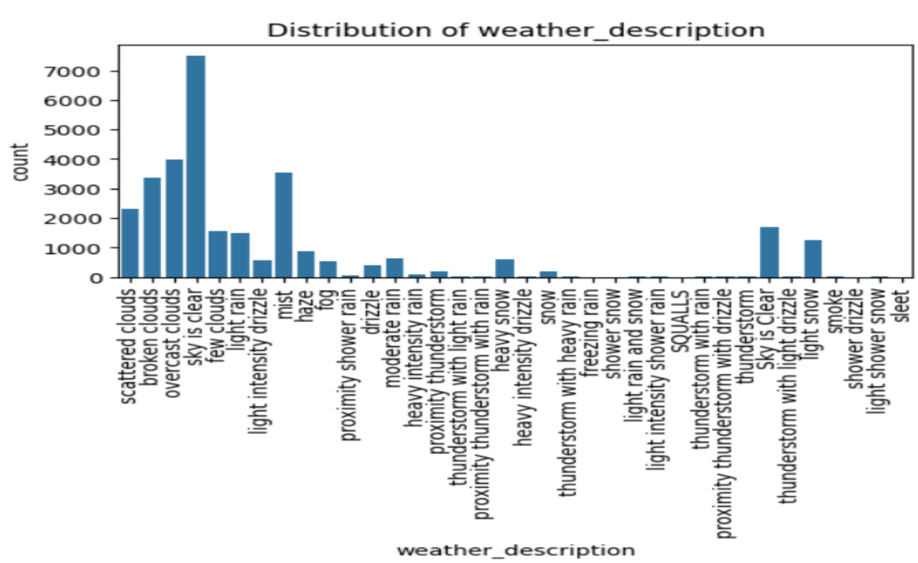
- **Training Set**: Represents 80% of the data, spanning from the start of the dataset up to 80% of the timeline.
- **Testing Set**: Represents the remaining 20% of the data, covering the final portion of the dataset timeline.

Features: Used weather data and time as inputs, with traffic volume as the target variable.

Here we have a PLOT to show distribution of categorical features

.





Feature Engineering

Data normalization ensures that all features are on a similar scale, improving model performance by preventing features with larger ranges from dominating the learning process. It also speeds up training and can lead to more accurate predictions.

```
# Normalize the features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_features = scaler.fit_transform(features)
```

LSTM Model Construction

Model: Implemented Long Short-Term Memory (LSTM) networks, given their ability to understand complex temporal patterns and predict future trends based on past data.

The LSTM model was constructed using TensorFlow, with two LSTM layers and a dropout layer to prevent overfitting.

Model Training

The model was trained over 50 epochs using the Adam optimizer and mean_squared_error as the loss function.

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
[25] # Train the model
history = model.fit(X_train, Y_train, epochs=50, batch_size=32, validation_data=(X_
775/775 ━━━━━━━━━━━━━━━━━━━━ 23s 28ms/step - loss: 0.0437 - val_loss: 0.0113
Epoch 2/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 19s 25ms/step - loss: 0.0133 - val_loss: 0.0105
Epoch 3/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 24s 29ms/step - loss: 0.0118 - val_loss: 0.0101
Epoch 4/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 19s 25ms/step - loss: 0.0114 - val_loss: 0.0098
Epoch 5/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 21s 27ms/step - loss: 0.0106 - val_loss: 0.0100
Epoch 6/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 19s 25ms/step - loss: 0.0103 - val_loss: 0.0088
Epoch 7/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 22s 27ms/step - loss: 0.0095 - val_loss: 0.0082
Epoch 8/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - loss: 0.0090 - val_loss: 0.0079
Epoch 9/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - loss: 0.0088 - val_loss: 0.0079
Epoch 10/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - loss: 0.0086 - val_loss: 0.0079
Epoch 11/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 40s 27ms/step - loss: 0.0084 - val_loss: 0.0075
Epoch 12/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 41s 28ms/step - loss: 0.0083 - val_loss: 0.0076
Epoch 13/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - loss: 0.0082 - val_loss: 0.0074
Epoch 14/50
775/775 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - loss: 0.0082 - val_loss: 0.0077
```

Model Results

The model performed well:

```
# Evaluate the model using test data
result=model.evaluate(X_test,Y_test)
result
```

```
194/194 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0070
0.006836108863353729
```

Model Predictions

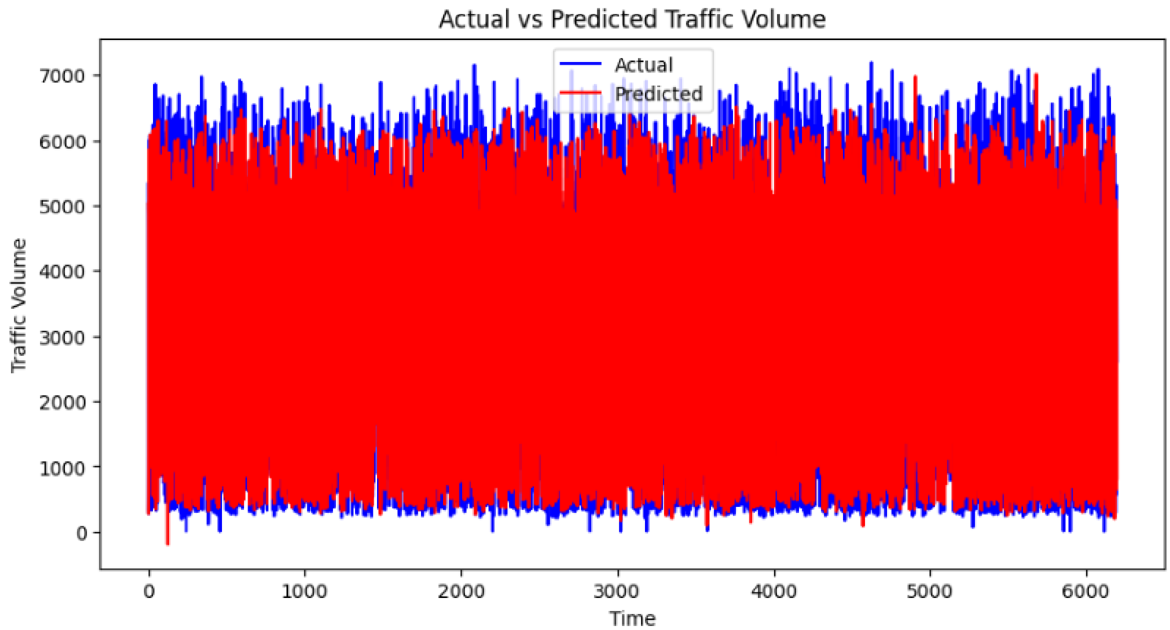
```
775/775 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step
194/194 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step
```

Predictions were generated using the model, and inverse_transform was applied to revert the predictions to their original scale:

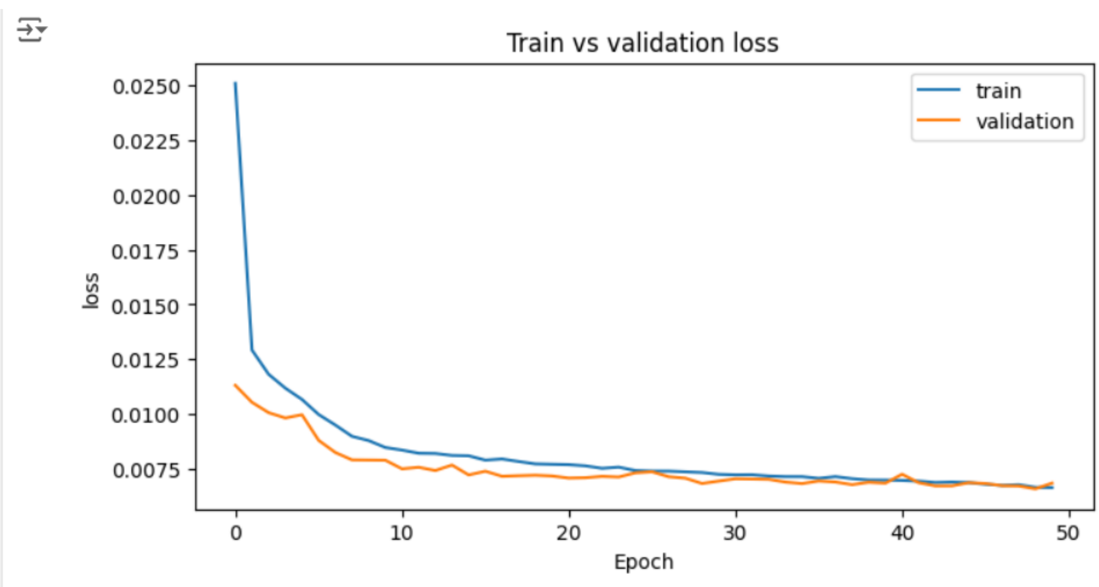
```
# Predicting
trainPredict = model.predict(X_train)
testPredict = model.predict(X_test)

# Ensure predictions have the correct number of features
trainPredict = trainPredict.reshape(-1, 1)
testPredict1 = testPredict.reshape(-1, 1)

# Inverse transform the predictions
trainPredict = scaler.inverse_transform(np.concatenate((trainPredict, np.zeros((trainPredict.shape[0], 1))), axis=1))[:,0]
testPredict1 = scaler.inverse_transform(np.concatenate((testPredict1, np.zeros((testPredict1.shape[0], 1))), axis=1))[:,0]

# Inverse transform the actual values
trainY = scaler.inverse_transform(np.concatenate((Y_train.reshape(-1, 1), np.zeros((Y_train.shape[0], 1))), axis=1))[:,0]
testY = scaler.inverse_transform(np.concatenate((Y_test.reshape(-1, 1), np.zeros((Y_test.shape[0], 1))), axis=1))[:,0]
```

Comparison between actual and predicted values:

When we compared the actual values with the predictions, we got the following PLOT:



We got the result of accuracy and loss for training and validation accuracy as shown in the next chart.



CONCLUSION

- The model achieved reasonable accuracy, and the predictions aligned closely with actual traffic volumes. The results could further be enhanced by exploring additional features and model tuning.
- we may various models to improve including Gated Recurrent Units (GRU) or Recurrent Neural Networks (RNN) for handling sequential data patterns. Additionally, ARIMA can be applied for time series forecasting where historical data shows linear trends and seasonality.

Group Task:

Member	Tasks
Haya Almalki	Coding, and layout.
Jehan Almutairi	Coding, and presentation and creating notebook.
Hanan Mohammed	Coding,,layout and report.