KARNAVATI UNIVERSITY

UNITEDWORLD SCHOOL OF COMPUTATIONAL INTELLIGENCE (USCI)

Summative Assessment (SA)

Submitted BY

YASH SAKARIYA

(Enrl. No.: 20220701023)

Course Code and Title: 21BSAI99E43 – ARTIFICIAL NEURAL NETWORK

B.Sc. (Hons.) Computer Science / Data Science / AIML

Iv Semester – dec – April 2024

Dec/April 2024

INDEX

• INTRODUCTION

In an increasingly urbanized world, managing traffic flow has become a critical aspect of urban planning and transportation infrastructure development. The "Traffic Flow Prediction" project seeks to address the challenges associated with traffic congestion, which not only leads to wasted time and fuel but also contributes to environmental pollution and decreased quality of life in cities. By harnessing the power of data analytics and machine learning, this project aims to provide actionable insights into traffic patterns and trends, allowing city planners and transportation government to make knowledgeable decisions about visitors management strategies, road maintenance, and public transportation planning. Through Col visitor' son with stakeholders and the integration of cutting-edge technologies, this project endeavors to pave the way for smarter, more sustainable urban transportation systems that enhance mobility, reduce emissions, and improve the overall urban living experience.

• PROBLEN STATEMENTS

The project aims to develop predictive models for traffic flow using historical data. By analyzing past traffic patterns, including vehicle counts, speeds, and congestion levels, the goal is to build accurate machine learning algorithms that can forecast future traffic conditions. These models will enable the anticipation of traffic congestion, identification of potential bottlenecks, and timely recommendations for traffic management strategies, ultimately improving urban mobility and transportation infrastructure efficiency.

• OBEJECTIVE

The objectives of this project are multifaceted. Firstly, we aim to leverage historical traffic data to develop robust predictive models capable of forecasting traffic flow patterns accurately. These models will be trained to anticipate fluctuations in traffic volume, congestion levels, and travel times across various road networks. Additionally, we seek to identify key factors influencing traffic dynamics, such as time of day, climate situations, and unique events, to enhance the predictive talents of the fashions. Furthermore, we aim to evaluate the performance of different machine learning algorithms and techniques to determine the most effective approach for traffic flow prediction. Ultimately, the project aims to provide valuable insights and tools for traffic management authorities to optimize traffic drift, alleviate congestion, and enhance normal transportation efficiency.

- PROJECT IMPLEMENTATION

1. IMPORT LIBRARIES

```
# Common libraries for data cleaning and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from numpy import load # use to load an npz file
from scipy.signal import periodogram # use to graph a periodogram to get seasonality analysis
from sklearn.preprocessing import MinMaxScaler # use to normalize the data features

# keras library to create NN models
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout

# libraries for the metrics we will use (RMSE and Spearman)
from keras.metrics import RootMeanSquaredError
import scipy.stats as stats
!pip install keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
# Set plot settings
plt.rcParams.update({'font.size': 12, 'font.family': 'serif'})
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['figure.dpi'] = 100
plt.rcParams['axes.grid'] = True
plt.rcParams['axes.grid.which'] = 'both'
plt.rcParams['grid.alpha'] = 0.5
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

2. DATA FORMATTING

```
[ ] data = pd.read_csv("/content/drive/MyDrive/ traffic dataset.csv")
```

```
[ ] column_list = data.columns.tolist()

    # Print the list of columns
    print("Columns:", column_list)
```

```
Columns: ['timestep', 'location', 'flow', 'occupy', 'speed']
```

```
# Now you can access the keys/columns of the DataFrame
lst = data.columns.tolist()
if lst:
    # Print the shape of the DataFrame
    print("DataFrame shape:", data.shape)

    # Print the data for the entire DataFrame
    print("Data for the entire DataFrame:")
    print(data)  # Print the entire DataFrame
else:
    print("No columns found in the dataset.")
```

```
DataFrame shape: (3035520, 5)
Data for the entire DataFrame:
         timestep  location   flow  occupy  speed
0               1         0  133.0  0.0603   65.8
1               1         1  210.0  0.0589   69.6
2               1         2  124.0  0.0358   65.8
3               1         3  145.0  0.0416   69.6
4               1         4  206.0  0.0493   69.4
...           ...       ...    ...     ...    ...
3035515     17856       165   74.0  0.0233   68.9
3035516     17856       166   11.0  0.0082   64.0
3035517     17856       167   83.0  0.0273   59.1
3035518     17856       168   70.0  0.0188   66.6
3035519     17856       169    6.0  0.0026   65.2

[3035520 rows x 5 columns]
```

```
traffic_data = data.values  # Convert the DataFrame to a NumPy array

data_dict = []
# loop for every timestep and every location and add as a single row
for timestep in range(traffic_data.shape[0]):
    for location in range(traffic_data.shape[1]):
        # Extract flow, occupy, and speed from each row
        flow = traffic_data[timestep, location]
        occupy = None  # Replace None with the method to extract occupy from your data
        speed = None  # Replace None with the method to extract speed from your data
        data_dict.append({
            "timestep": timestep + 1,
            "location": location,
            "flow": flow,
            "occupy": occupy,
            "speed": speed
        })
```

```
[ ] df = pd.DataFrame(data_dict)
    df.to_csv("traffic.csv", index=False)
```

3. Data Visualization & EDA

```
traffic = pd.read_csv("/content/drive/MyDrive/ traffic dataset.csv")
print(len(traffic))
traffic.head()
```

```
3035520
```

	timestep	location	flow	occupy	speed
0	1	0	133.0	0.0603	65.8
1	1	1	210.0	0.0589	69.6
2	1	2	124.0	0.0358	65.8
3	1	3	145.0	0.0416	69.6
4	1	4	206.0	0.0493	69.4

```
[ ] traffic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3035520 entries, 0 to 3035519
Data columns (total 5 columns):
 #   Column     Dtype
---  ------     -----
 0   timestep   int64
 1   location   int64
 2   flow       float64
 3   occupy     float64
 4   speed      float64
dtypes: float64(3), int64(2)
memory usage: 115.8 MB
```

```
traffic.count()
```

```
timestep    3035520
location    3035520
flow        3035520
occupy      3035520
speed       3035520
dtype: int64
```

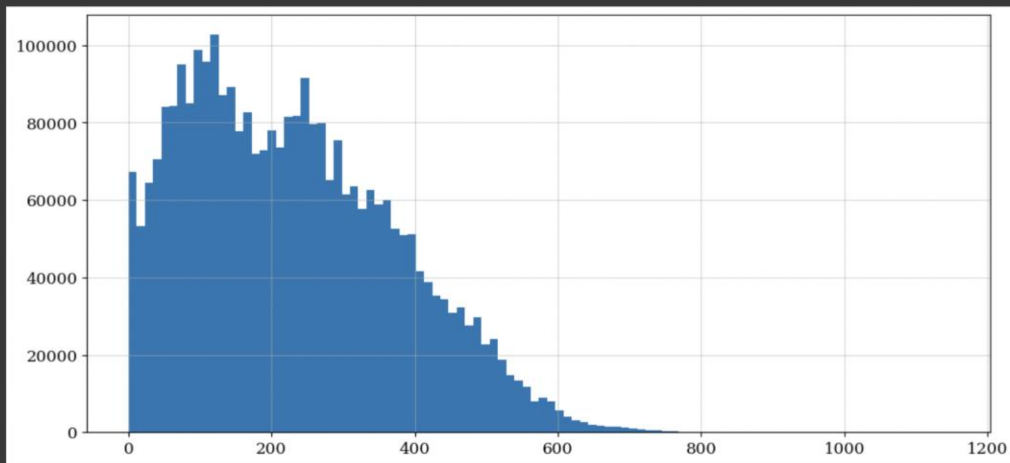
```
[ ] traffic.isna().sum()
```

```
timestep    0
location    0
flow        0
occupy      0
speed       0
dtype: int64
```

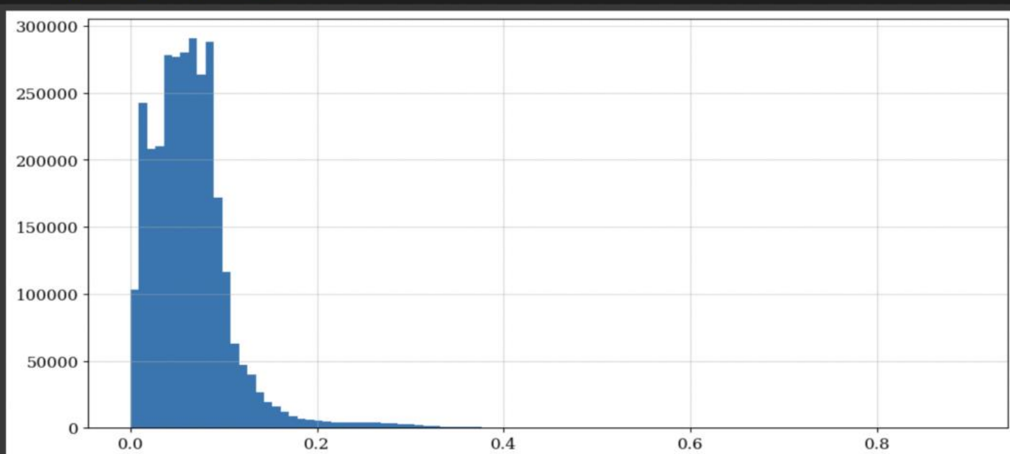
```
traffic.describe()
```

	timestep	location	flow	occupy	speed
count	3.035520e+06	3.035520e+06	3.035520e+06	3.035520e+06	3.035520e+06
mean	8.928500e+03	8.450000e+01	2.306807e+02	6.507109e-02	6.376300e+01
std	5.154584e+03	4.907393e+01	1.462170e+02	4.590215e-02	6.652010e+00
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	3.000000e+00
25%	4.464750e+03	4.200000e+01	1.100000e+02	3.570000e-02	6.260000e+01
50%	8.928500e+03	8.450000e+01	2.150000e+02	6.010000e-02	6.490000e+01
75%	1.339225e+04	1.270000e+02	3.340000e+02	8.390000e-02	6.740000e+01
max	1.785600e+04	1.690000e+02	1.147000e+03	8.955000e-01	8.230000e+01

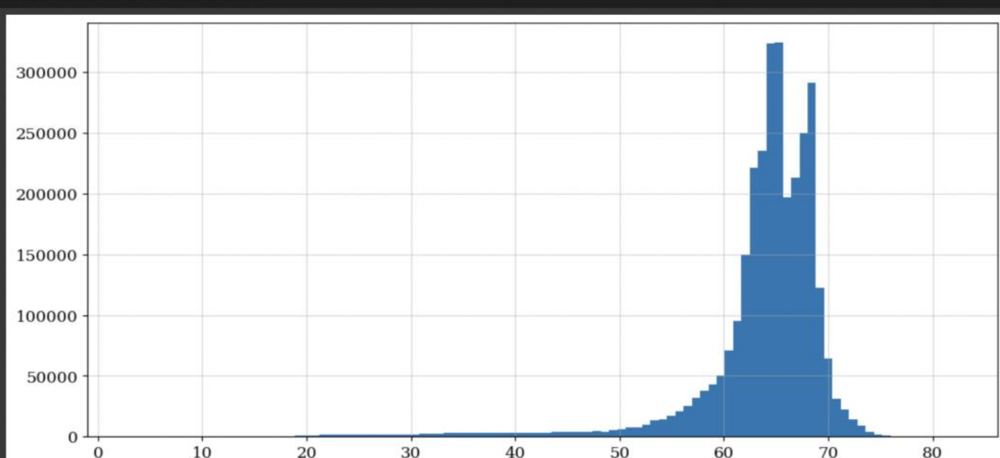
```
traffic['flow'].hist(bins=100);
```




```
traffic['occupy'].hist(bins=100);
```




```
traffic['speed'].hist(bins=100);
```

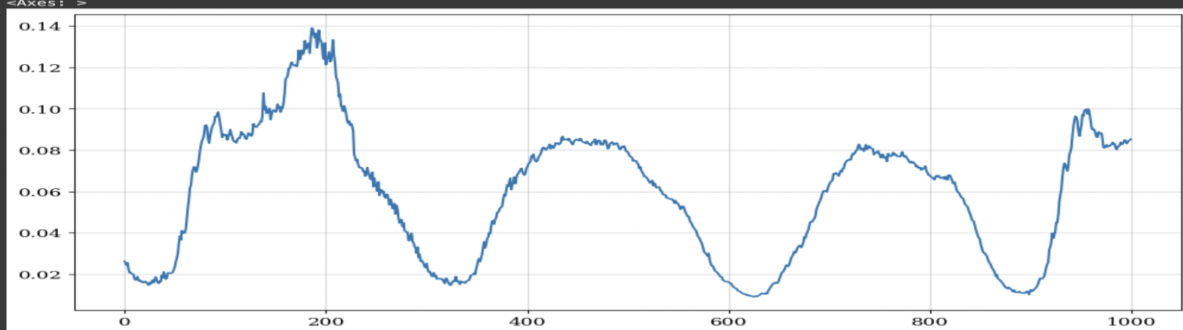


```
[ ]  location_0 = traffic[traffic["location"]==50].reset_index()
     location_0.head()
```

	index	timestep	location	flow	occupy	speed
0	50	1	50	76.0	0.0262	69.5
1	220	2	50	81.0	0.0255	68.8
2	390	3	50	80.0	0.0243	69.0
3	560	4	50	76.0	0.0255	68.4
4	730	5	50	70.0	0.0224	68.1

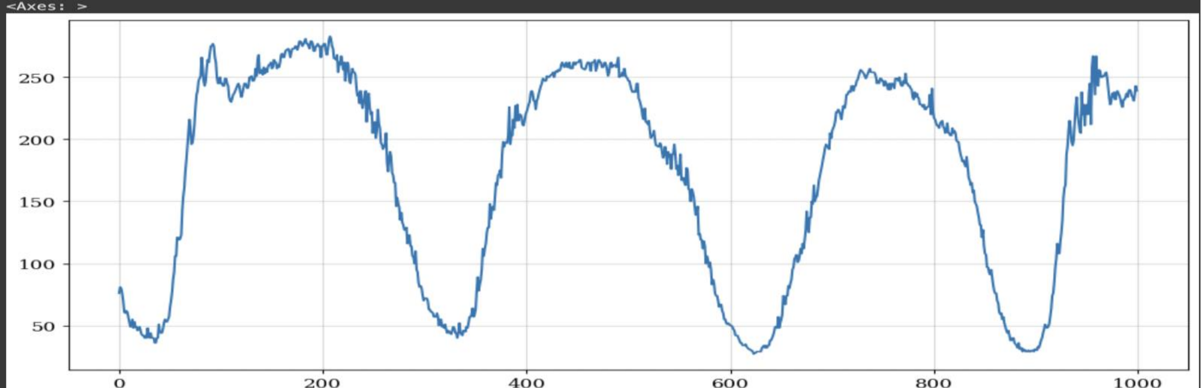
```
location_0["occupy"][:1000].plot()
```

<Axes: >

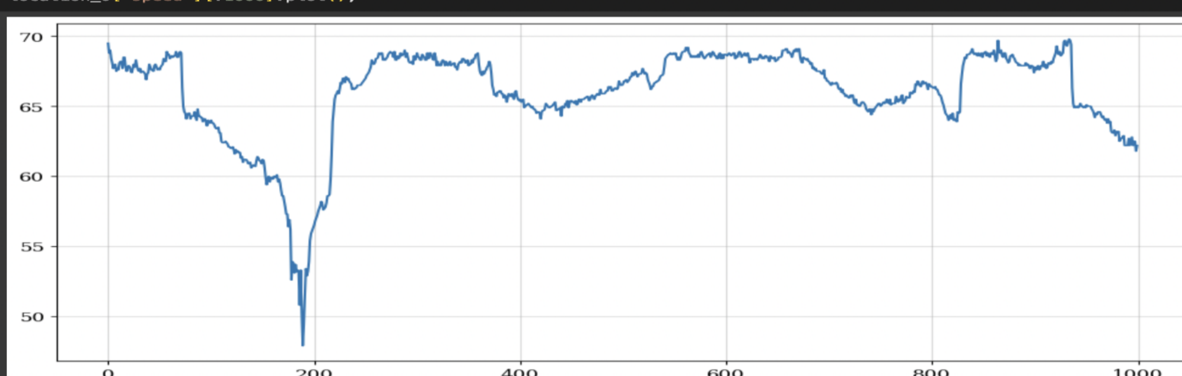



```
location_0["flow"][:1000].plot()
```

<Axes: >




```
[ ]  location_0["speed"][:1000].plot();
```

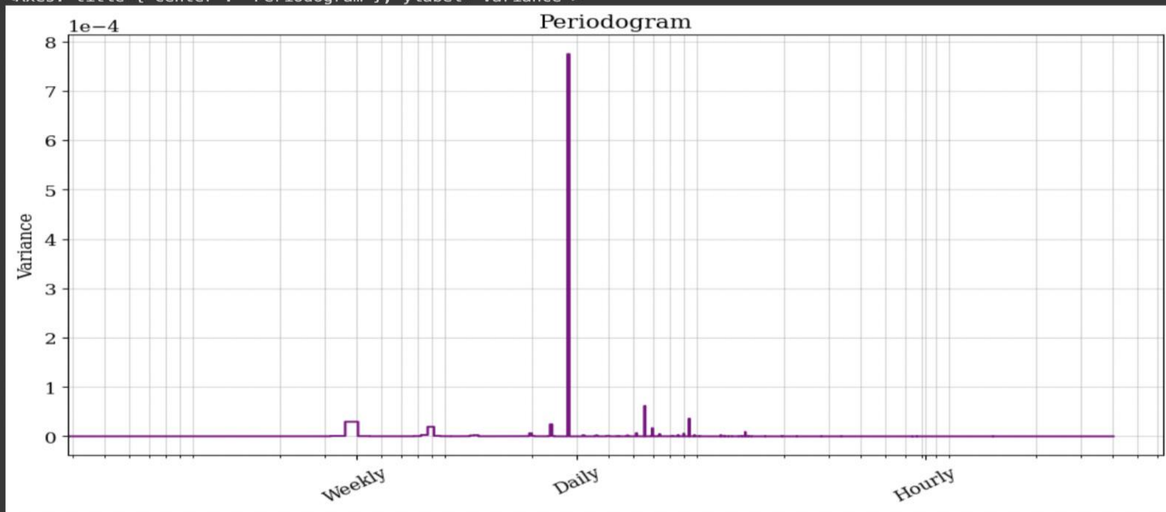


```
[ ] COR_STEP = 1
    pres = traffic[['flow', 'occupy', 'speed']][0:-(COR_STEP)].reset_index(drop=True)
    future = traffic[['flow', 'occupy', 'speed']][COR_STEP:] \
        .reset_index(drop=True) \
        .add_suffix('_future')
    val = pres.join(future)
    val.corr()
```

	flow	occupy	speed	flow_future	occupy_future	speed_future
flow	1.000000	0.674039	-0.296332	0.535235	0.450192	-0.235030
occupy	0.674039	1.000000	-0.752040	0.445282	0.477379	-0.303858
speed	-0.296332	-0.752040	1.000000	-0.228266	-0.275180	0.233537
flow_future	0.535235	0.445282	-0.228266	1.000000	0.674040	-0.296331
occupy_future	0.450192	0.477379	-0.275180	0.674040	1.000000	-0.752040
speed_future	-0.235030	-0.303858	0.233537	-0.296331	-0.752040	1.000000

```
def plot_periodogram(ts, detrend='linear', ax=None):
    """
    Plots the periodogram of a time series.

    Args:
        ts (pd.Series): A time series.
        detrend (str): Detrending method for the time series.
        ax (matplotlib.axes.Axes): The axes on which to plot.

    Returns:
        ax (matplotlib.axes.Axes): The axes on which the periodogram is plotted.
    """
    fs = pd.Timedelta(weeks=4) / pd.Timedelta(minutes=5)
    frequencies, spectrum = periodogram(
        ts,
        fs=fs,
        detrend=detrend,
        window="boxcar",
        scaling='spectrum',
    )
    if ax is None:
      _, ax = plt.subplots()

    ax.step(frequencies, spectrum, color="purple")
    ax.set_xscale("log")
    ax.set_xticks([4, 30, 30*24])
    ax.set_xticklabels(
        [
            "Weekly",
            "Daily",
            "Hourly"
        ],
        rotation=30,
    )
    ax.ticklabel_format(axis="y", style="sci", scilimits=(0, 0))
    ax.set_ylabel("Variance")
    ax.set_title("Periodogram")
    return ax

plot_periodogram(location_0["occupy"])
```

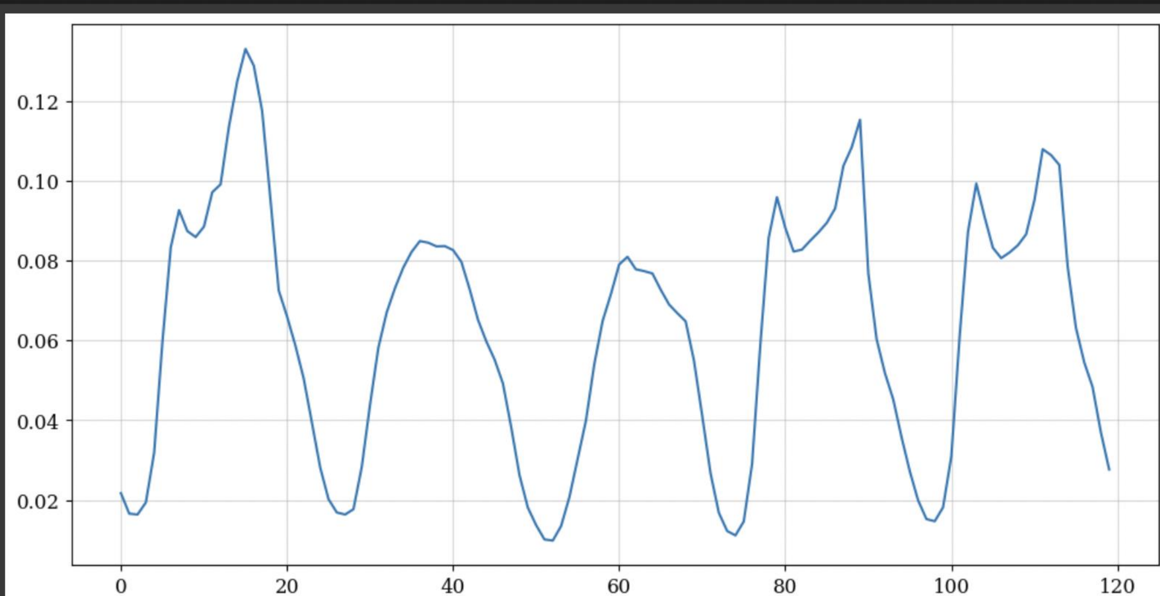
<Axes: title={'center': 'Periodogram'}, ylabel='Variance'>




```python
location_0["hour"] = ((location_0["timestep"] - 1) // 12)
grouped = location_0.groupby("hour").mean().reset_index()
grouped.head()
```

	hour	index	timestep	location	flow	occupy	speed
0	0	985.0	6.5	50.0	65.500000	0.021700	68.191667
1	1	3025.0	18.5	50.0	48.166667	0.016567	67.900000
2	2	5065.0	30.5	50.0	41.166667	0.016350	67.691667
3	3	7105.0	42.5	50.0	46.666667	0.019350	67.591667
4	4	9145.0	54.5	50.0	89.750000	0.031800	68.141667

```python
grouped["occupy"][:24*5].plot();
```



```
[ ] COR_STEP = 12
    pres = grouped[['flow', 'occupy', 'speed']][0:-(COR_STEP)].reset_index(drop=True)
    future = grouped[['flow', 'occupy', 'speed']][COR_STEP:] \
        .reset_index(drop=True) \
        .add_suffix('_future')
    val = pres.join(future)
    val.corr()
```

	flow	occupy	speed	flow_future	occupy_future	speed_future
flow	1.000000	0.966218	-0.718619	-0.722547	-0.707590	0.701991
occupy	0.966218	1.000000	-0.839110	-0.707282	-0.694619	0.686046
speed	-0.718619	-0.839110	1.000000	0.693508	0.681356	-0.599727
flow_future	-0.722547	-0.707282	0.693508	1.000000	0.966046	-0.718996
occupy_future	-0.707590	-0.694619	0.681356	0.966046	1.000000	-0.839682
speed_future	0.701991	0.686046	-0.599727	-0.718996	-0.839682	1.000000

4. Data Preparation

```
# creating 3-dimensional array for [timestep, timeframe, features]
def create_dataset(location, WINDOW_SIZE):

    # mask a certain location
    location_current = traffic[traffic["location"]==location].reset_index()

    # group to hour and average 12 (5-minute) timesteps
    location_current["hour"] = ((location_current["timestep"] - 1) // 12)
    grouped = location_current.groupby("hour").mean().reset_index()

    # add hour features as mod 24 cycle (0...23)
    grouped['day'] = (grouped['hour'] // 24) % 7
    grouped['hour'] %= 24

    one_hot_hour = pd.get_dummies(grouped['hour'])
    one_hot_hour = one_hot_hour.add_prefix('hour_')

    # merge all the features together to get a total of 27 features
    hour_grouped = pd.concat([grouped[["occupy", "flow", "speed"]], one_hot_hour], axis=1)
    hour_grouped = np.array(hour_grouped)

    X, Y = [], []

    # add lag features (in reverse time order)
    for i in range(len(hour_grouped) - WINDOW_SIZE):
        X.append(hour_grouped[i:(i + WINDOW_SIZE)][::-1]) # reverse the order
        Y.append(hour_grouped[i + WINDOW_SIZE, 0]) # index 0 is occupy

    return X,Y # returns (timestep, timeframe, features) and (target)
```

```
[ ] # creating 4-th dimension for the locations
    X, Y = [], []

    for location in range(170):
        a,b = create_dataset(location, WINDOW_SIZE=24)
        X.append(a)
        Y.append(b)

    X = np.moveaxis(X,0,-1)
    Y = np.moveaxis(Y,0,-1)

    print(X.shape)
    print(Y.shape)

    (1464, 24, 27, 170)
    (1464, 170)
```

```
TRAIN_SIZE = 0.8
TEST_SIZE  = 0.2

train_size = int(len(X) * TRAIN_SIZE)
test_size  = int(len(X) * TEST_SIZE)

train_X, train_Y = X[:train_size], Y[:train_size]
test_X, test_Y = X[train_size:], Y[train_size:]

print(train_X.shape)
print(train_Y.shape)
print(test_X.shape)
print(test_Y.shape)
```

```
(1171, 24, 27, 170)
(1171, 170)
(293, 24, 27, 170)
(293, 170)
```

```
scaler_X = MinMaxScaler()
scaler_Y = MinMaxScaler()
train_X = scaler_X.fit_transform(train_X.reshape(train_X.shape[0] * train_X.shape[1], -1)) \
                   .reshape(train_X.shape[0], train_X.shape[1], -1)
test_X = scaler_X.transform(test_X.reshape(test_X.shape[0] * test_X.shape[1], -1)) \
                   .reshape(test_X.shape[0], test_X.shape[1], -1)
train_Y = scaler_Y.fit_transform(train_Y)
test_Y = scaler_Y.transform(test_Y)
```

```
print(train_X.shape)
print(test_X.shape)
print(train_Y.shape)
print(test_Y.shape)
```

```
(1171, 24, 4590)
(293, 24, 4590)
(1171, 170)
(293, 170)
```

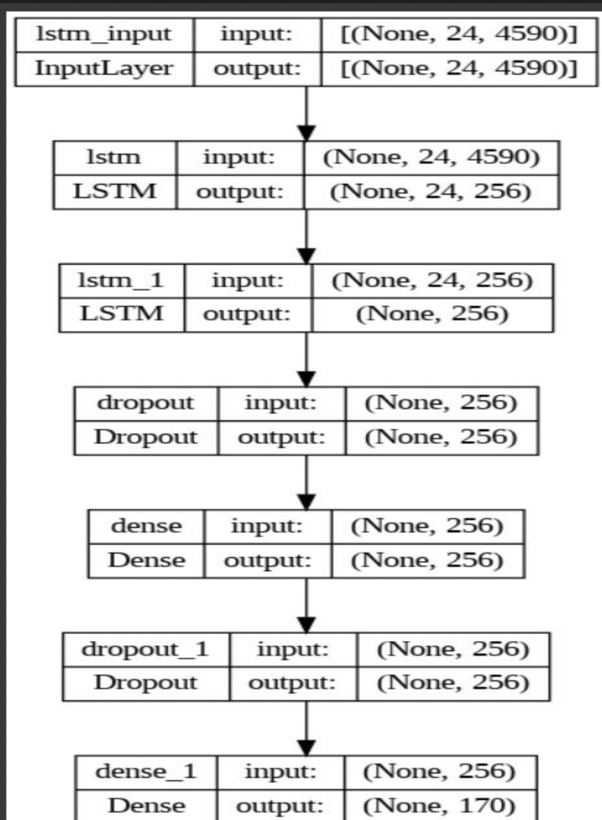
5. Model Training

```python
model = Sequential([
    LSTM(256, return_sequences=True, input_shape=(train_X.shape[1], train_X.shape[2])),
    LSTM(256, return_sequences=False),
    Dropout(0.2),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(170, activation='linear'),
])

model.compile(loss='mse', optimizer='adam', metrics=[RootMeanSquaredError()])
```

```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 24, 256)           4963328

 lstm_1 (LSTM)               (None, 256)               525312

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 170)               43690

=================================================================
Total params: 5598122 (21.36 MB)
Trainable params: 5598122 (21.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
from tensorflow.keras.utils import plot_model

plot_model(model, show_shapes=True, show_layer_names=True)
```



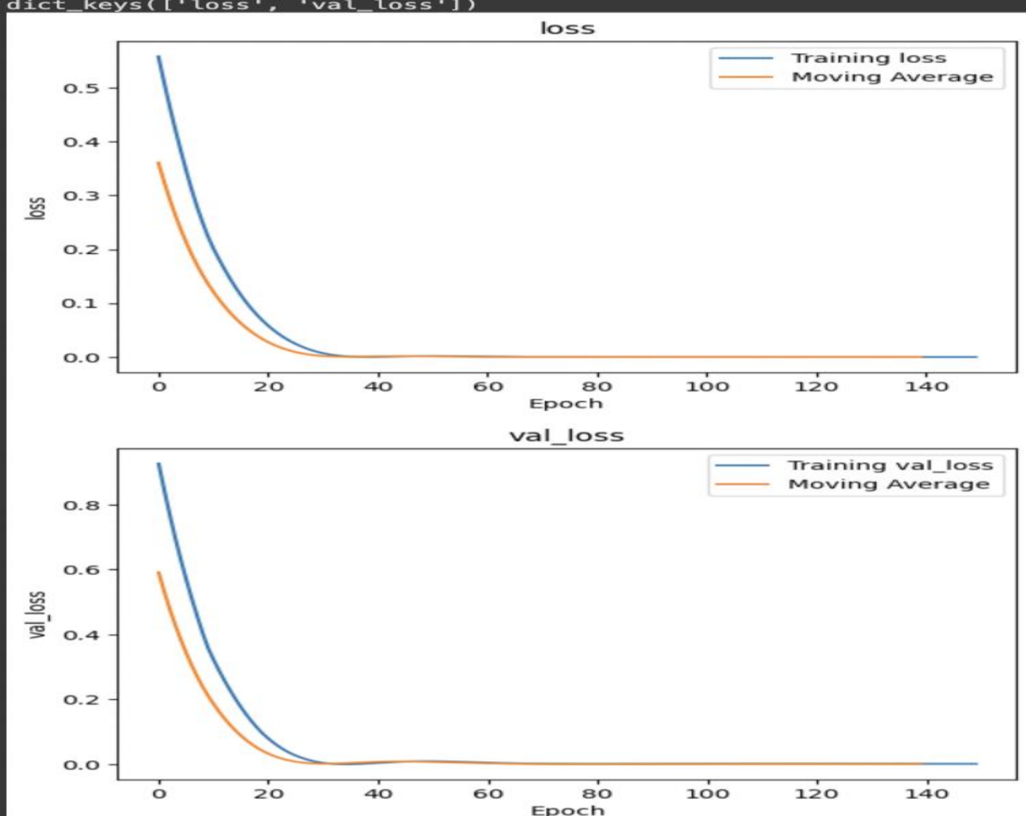
```python
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Assuming train_X and train_Y are your training data
# Replace ellipsis (...) with your actual data
train_X = np.array([[0.1, 0.2, 0.3], [0.2, 0.3, 0.4], [0.3, 0.4, 0.5]])
train_Y = np.array([[0.5], [0.6], [0.7]])

# Define your model architecture
model = Sequential()
model.add(Dense(64, input_shape=(train_X.shape[1],), activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))

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

# Train the model
history = model.fit(train_X, train_Y, epochs=150, batch_size=32, validation_split=0.1, verbose=2)
```

```
1/1 - 0s - loss: 1.6332e-04 - val_loss: 0.0013 - 36ms/epoch - 36ms/step
Epoch 138/150
1/1 - 0s - loss: 1.6261e-04 - val_loss: 0.0013 - 62ms/epoch - 62ms/step
Epoch 139/150
1/1 - 0s - loss: 1.6207e-04 - val_loss: 0.0013 - 38ms/epoch - 38ms/step
Epoch 140/150
1/1 - 0s - loss: 1.6152e-04 - val_loss: 0.0013 - 36ms/epoch - 36ms/step
Epoch 141/150
1/1 - 0s - loss: 1.6096e-04 - val_loss: 0.0013 - 40ms/epoch - 40ms/step
Epoch 142/150
1/1 - 0s - loss: 1.6039e-04 - val_loss: 0.0013 - 39ms/epoch - 39ms/step
Epoch 143/150
1/1 - 0s - loss: 1.5982e-04 - val_loss: 0.0013 - 36ms/epoch - 36ms/step
Epoch 144/150
1/1 - 0s - loss: 1.5925e-04 - val_loss: 0.0013 - 38ms/epoch - 38ms/step
Epoch 145/150
1/1 - 0s - loss: 1.5868e-04 - val_loss: 0.0013 - 44ms/epoch - 44ms/step
Epoch 146/150
1/1 - 0s - loss: 1.5811e-04 - val_loss: 0.0013 - 37ms/epoch - 37ms/step
Epoch 147/150
1/1 - 0s - loss: 1.5754e-04 - val_loss: 0.0013 - 36ms/epoch - 36ms/step
Epoch 148/150
1/1 - 0s - loss: 1.5698e-04 - val_loss: 0.0013 - 34ms/epoch - 34ms/step
Epoch 149/150
1/1 - 0s - loss: 1.5642e-04 - val_loss: 0.0013 - 35ms/epoch - 35ms/step
Epoch 150/150
1/1 - 0s - loss: 1.5586e-04 - val_loss: 0.0013 - 35ms/epoch - 35ms/step
```

```python
def plot_training(training_history, text, width):
    history = training_history.history[text]

    # creates a moving average plot to reduce variations
    moving_average = [float("NaN") for i in range(width)]
    for i in range(width, len(history)+1):
        moving_average.append(np.mean(np.array(history[i-width:i+1])))

    plt.plot(history)
    plt.plot(moving_average)
    plt.title(text)
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['value','moving average'], loc='upper left')
    plt.show()
```

```python
# Print available keys in the history dictionary
print(history.history.keys())

# Plot available metrics
plot_training(history, 'loss', WIDTH)
plot_training(history, 'val_loss', WIDTH)
```

```
dict_keys(['loss', 'val_loss'])
```



- Conclusion

Several key insights and conclusions can be drawn regarding traffic flow. Firstly, it is evident that traffic patterns exhibit significant variability over time, influenced by factors consisting of height hours, weather situations, and unique events. Additionally, certain locations may experience recurrent congestion, highlighting the need for targeted interventions and traffic management strategies. Moreover, the effectiveness of predictive models in anticipating traffic flow dynamics has been demonstrated, providing valuable tools for transportation planning and management. Moving forward, continued efforts in data collection, model refinement, and real-time monitoring will be essential for enhancing traffic prediction accuracy and facilitating more efficient traffic management practices. By leveraging these insights, stakeholders can work towards improving overall traffic flow, enhancing commuter experiences, and promoting sustainable urban mobility.