In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Load the dataset
df = pd.read_csv('TRAFFIC DATA.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	LOCATION	TIME	DAYS OF THE WEEK	TRAFFIC CONDITION
0	FROM AGBARA CUSTOM TO IYANA ERA	9AM	MONDAY	NOT CONGESTED
1	FROM TRADE FAIR TO ABULER ADO	9.10AM	MONDAY	NOT CONGESTED
2	FROM BARRACKS TO VOLKS	9.15AM	MONDAY	NOT CONGESTED
3	FROM IYANA ISASHI TO AGBARA	9.04AM	MONDAY	NOT CONGESTED
4	FROM MOBOLAJI JOHNSON TO 7UP	9.04AM	MONDAY	NOT CONGESTED

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
LOCATION             0
TIME                 0
DAYS OF THE WEEK     0
TRAFFIC CONDITION    0
dtype: int64
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   LOCATION           299 non-null    object
 1   TIME               299 non-null    object
 2   DAYS OF THE WEEK   299 non-null    object
 3   TRAFFIC CONDITION  299 non-null    object
dtypes: object(4)
memory usage: 9.5+ KB
```

In [6]:
```
df.shape
```

Out[6]:
```
(299, 4)
```

In [7]:
```
df['TIME'].unique()
```

Out[7]:
```
array(['9AM', '9.10AM', '9.15AM', '9.04AM', '9.05AM', '9.20AM', '9.25AM',
       '9.00AM', '8.00AM', '8.02AM', '8.03AM', '8.15AM', '8.30AM',
       '9.08AM', '9.06AM', '9.07AM', '8.08AM', '8.09AM', '8.16AM',
       '8.10AM', '8.11AM', '8.01AM', '9.03AM', '8.18AM', '8.20AM',
       '8.45AM', '5.00PM', '5.04PM', '5.06PM', '5.30PM', '5.45PM',
       '6.00PM', '5.02PM', '5.20PM', '5.21PM', '5.10PM', '5.15PM',
       '5.25PM', '5.01PM', '5.05PM', '5.31PM', '5.22PM', '5.23PM',
       '5.35PM', '5.03PM', '5.24PM', '5.17PM', '5.18PM', '6.01PM',
       '6.15PM', '6.18PM', '6.20PM', '6.25PM', '6.06PM', '6.30PM',
       '6.40PM', '6.42PM', '6.45PM', '7.00PM', '5.32PM', '5.33PM',
       '5.07PM', '5.50PM', '5.51PM', '6.04PM', '6.03PM', '6.48PM',
       '6.50PM', '6.52PM', '10.00AM', '9.02AM', '9.03.AM', '10.00AM',
       '9.32AM', '8.25AM', '8.21AM', '8.22AM', '9.00PM', '8.30PM',
       '8.00PM', '7.30PM', '4.00PM', '3.00PM', '2.00PM', '1.00PM',
       '12.00PM', '11.00AM', '10.30AM', '9.30AM', '7.30AM', '7.00AM',
       '6.30AM', '6.00AM', '8.50AM'], dtype=object)
```

In [8]:
```python
print(df['TIME'][0])
print(df['TIME'][153])
print(df['TIME'][158])
print(df['TIME'][242])
```

```
9AM
9.03.AM
10.00AM
9.03.AM
```

In [9]:
```python
# Split the 'TimeColumn' into two new columns: 'Time' and 'Period'
df[['TIME', 'PERIOD']] = df['TIME'].str.extract(r'(\d+\.\d+)([APMapm]+)')
```

In [10]:
```python
df.head()
```

Out[10]:

	LOCATION	TIME	DAYS OF THE WEEK	TRAFFIC CONDITION	PERIOD
0	FROM AGBARA CUSTOM TO IYANA ERA	NaN	MONDAY	NOT CONGESTED	NaN
1	FROM TRADE FAIR TO ABULER ADO	9.10	MONDAY	NOT CONGESTED	AM
2	FROM BARRACKS TO VOLKS	9.15	MONDAY	NOT CONGESTED	AM
3	FROM IYANA ISASHI TO AGBARA	9.04	MONDAY	NOT CONGESTED	AM
4	FROM MOBOLAJI JOHNSON TO 7UP	9.04	MONDAY	NOT CONGESTED	AM

In [11]:
```python
df['TIME'][0] = '9.00'
df['PERIOD'][0] = 'AM'
df['TIME'][153] = '9.03'
df['PERIOD'][153] = 'AM'
df['TIME'][158] = '10.00'
df['PERIOD'][158] = 'AM'
df['TIME'][242] = '9.03'
df['PERIOD'][242] = 'AM'
```

In [12]:
```python
df.head()
```

Out[12]:

	LOCATION	TIME	DAYS OF THE WEEK	TRAFFIC CONDITION	PERIOD
0	FROM AGBARA CUSTOM TO IYANA ERA	9.00	MONDAY	NOT CONGESTED	AM
1	FROM TRADE FAIR TO ABULER ADO	9.10	MONDAY	NOT CONGESTED	AM
2	FROM BARRACKS TO VOLKS	9.15	MONDAY	NOT CONGESTED	AM
3	FROM IYANA ISASHI TO AGBARA	9.04	MONDAY	NOT CONGESTED	AM
4	FROM MOBOLAJI JOHNSON TO 7UP	9.04	MONDAY	NOT CONGESTED	AM

In [13]:

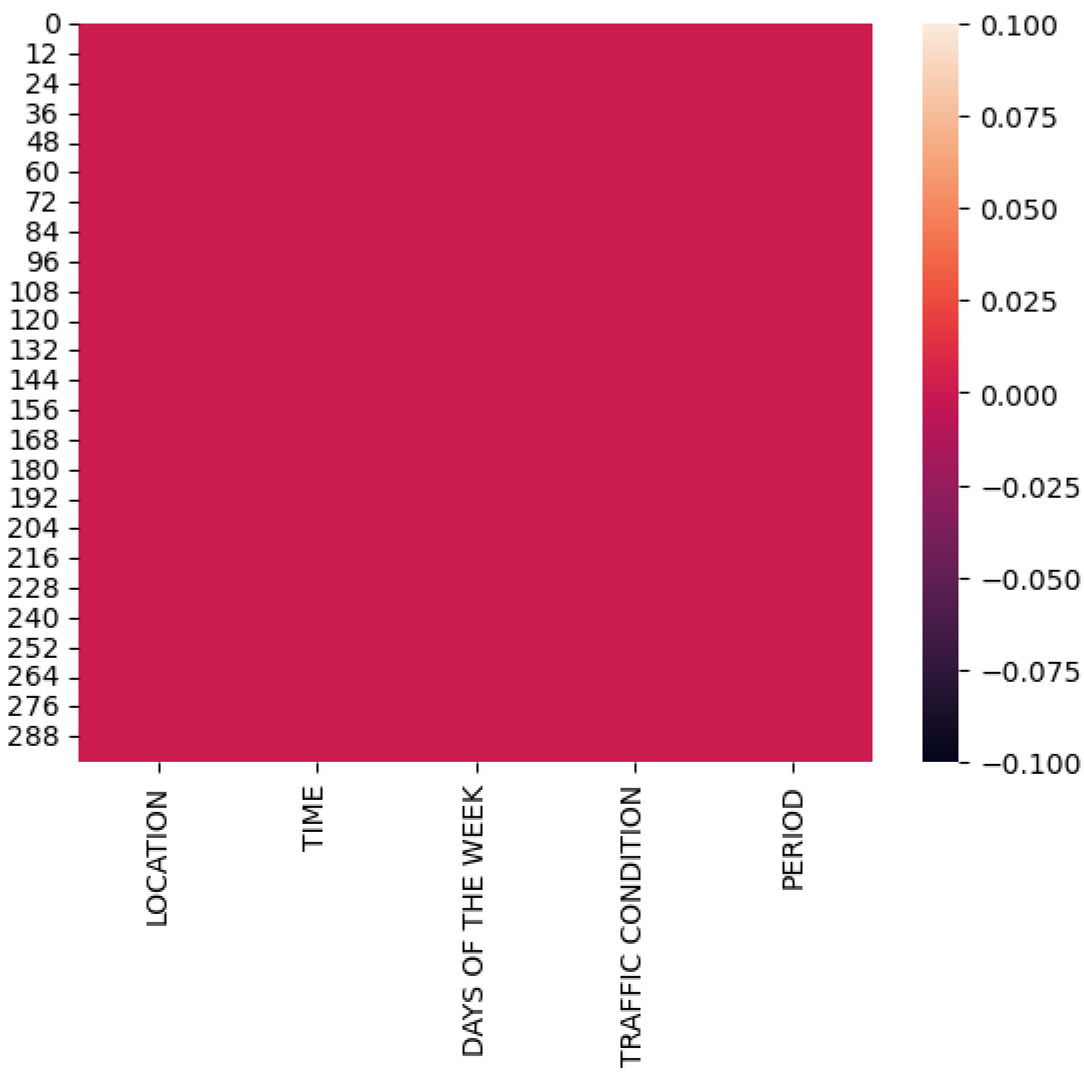
```
df.isnull().sum()
```

Out[13]:

```
LOCATION             0
TIME                 0
DAYS OF THE WEEK     0
TRAFFIC CONDITION    0
PERIOD               0
dtype: int64
```

In [14]:

```
sns.heatmap(df.isnull());
```



In [15]:

```
df.head()
```

Out[15]:

	LOCATION	TIME	DAYS OF THE WEEK	TRAFFIC CONDITION	PERIOD
0	FROM AGBARA CUSTOM TO IYANA ERA	9.00	MONDAY	NOT CONGESTED	AM
1	FROM TRADE FAIR TO ABULER ADO	9.10	MONDAY	NOT CONGESTED	AM
2	FROM BARRACKS TO VOLKS	9.15	MONDAY	NOT CONGESTED	AM
3	FROM IYANA ISASHI TO AGBARA	9.04	MONDAY	NOT CONGESTED	AM
4	FROM MOBOLAJI JOHNSON TO 7UP	9.04	MONDAY	NOT CONGESTED	AM

In [16]:
```
df['DAYS OF THE WEEK'].unique()
```

Out[16]:
```
array(['MONDAY', 'TUESDAY', 'WEDNESDAY', 'THURSDAY', 'FRIDAY', 'SATURDAY',
       'SUNDAY', 'STAURDAY'], dtype=object)
```

In [17]:
```
df[df['DAYS OF THE WEEK'] == 'STAURDAY']
```

Out[17]:

	LOCATION	TIME	DAYS OF THE WEEK	TRAFFIC CONDITION	PERIOD
79	IKEJA BRIDGE TO OBA AKRAN TO DANGOTE	5.23	STAURDAY	NOT CONGESTED	PM
80	LAGOS-IBADAN EXPRESSWAY FROM KARA TO OTETOLA	6.00	STAURDAY	NOT CONGESTED	PM

In [18]:
```
df['DAYS OF THE WEEK'][79:81] = 'SATURDAY'
```

In [19]:
```
df['DAYS OF THE WEEK'].unique()
```

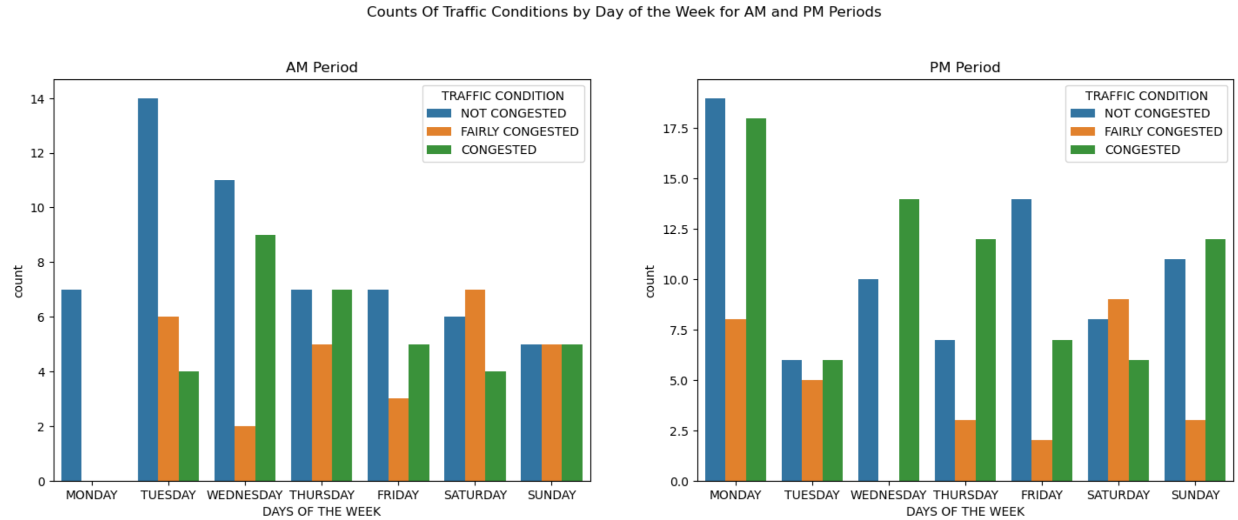
Out[19]:
```
array(['MONDAY', 'TUESDAY', 'WEDNESDAY', 'THURSDAY', 'FRIDAY', 'SATURDAY',
       'SUNDAY'], dtype=object)
```

In [20]:
```
df['TRAFFIC CONDITION'].unique()
```

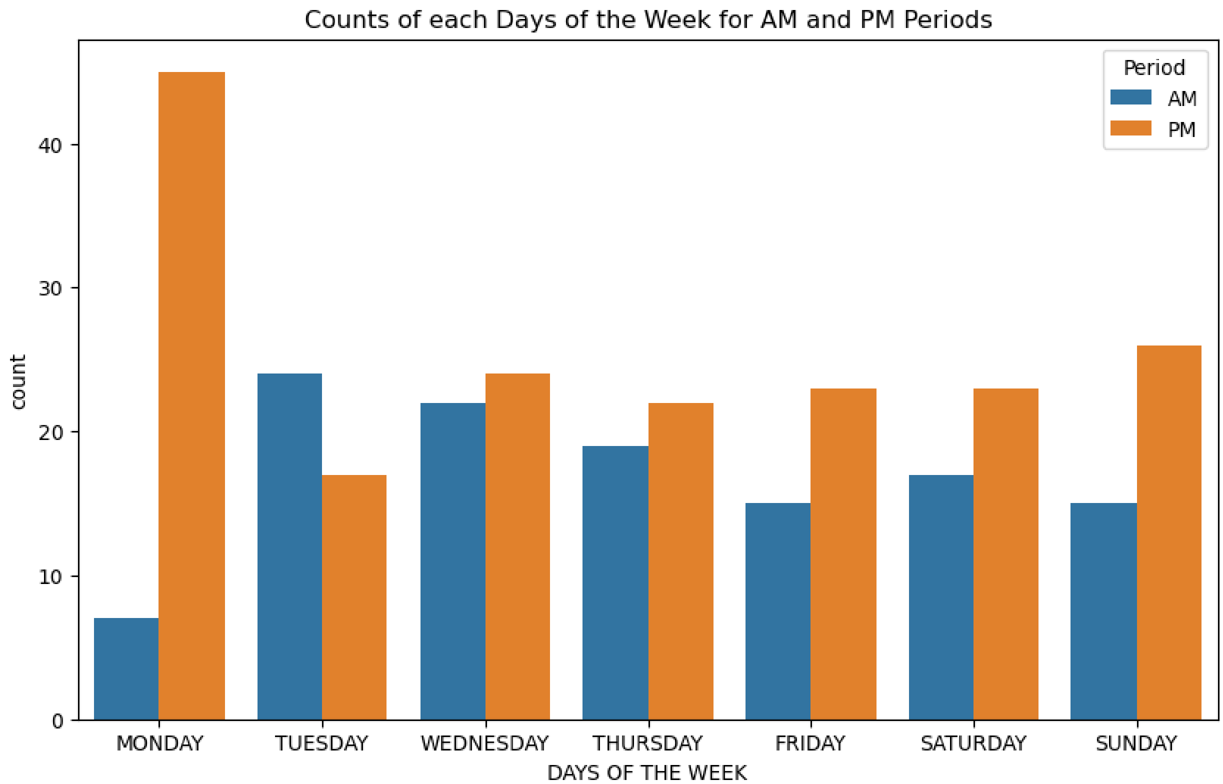
Out[20]:
```
array(['NOT CONGESTED', 'FAIRLY CONGESTED', 'CONGESTED'], dtype=object)
```

In [21]:
```
df_am = df[df['PERIOD'] == 'AM']
df_pm = df[df['PERIOD'] == 'PM']
```

In [22]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(17.5, 6))

# Plot for 'AM' period
sns.countplot(data=df_am, x='DAYS OF THE WEEK', hue='TRAFFIC CONDITION', ax=axes[0])
axes[0].set_title('AM Period')

# Plot for 'PM' period
sns.countplot(data=df_pm, x='DAYS OF THE WEEK', hue='TRAFFIC CONDITION', ax=axes[1])
axes[1].set_title('PM Period')

# Add an overall title
plt.suptitle('Counts Of Traffic Conditions by Day of the Week for AM and PM Periods',

# Show the plots
plt.show()
```



In [23]:
```python
plt.figure(figsize = (10,6))
sns.countplot(data =df, x = 'DAYS OF THE WEEK', hue='PERIOD')
plt.title('Counts of each Days of the Week for AM and PM Periods')
plt.legend(title='Period', loc='upper right');
```



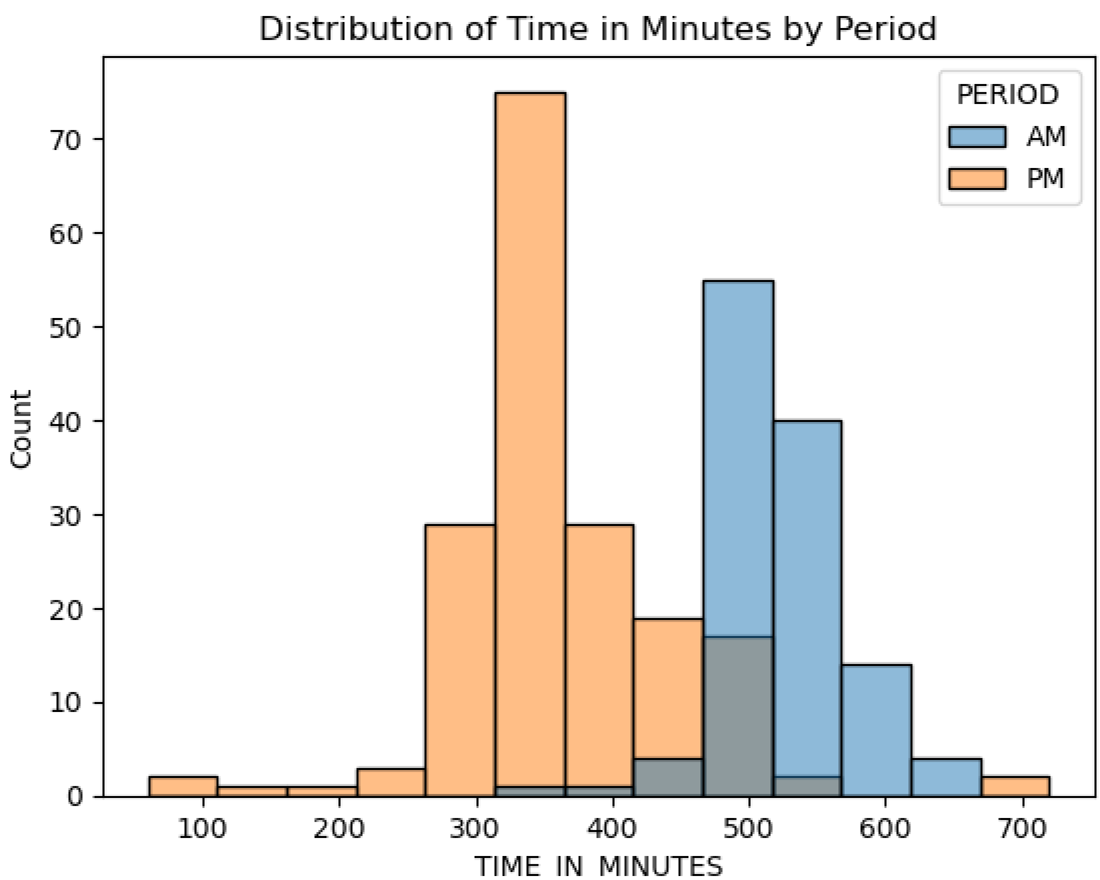
In [24]:
```python
# Extract hour and minute components as integers
df[['HOUR', 'MINUTE']] = df['TIME'].str.split('.', expand=True).astype(int)

# Convert time to minutes
df['TIME_IN_MINUTES'] = df['HOUR'] * 60 + df['MINUTE']
```

In [25]:
```python
df.head()
```

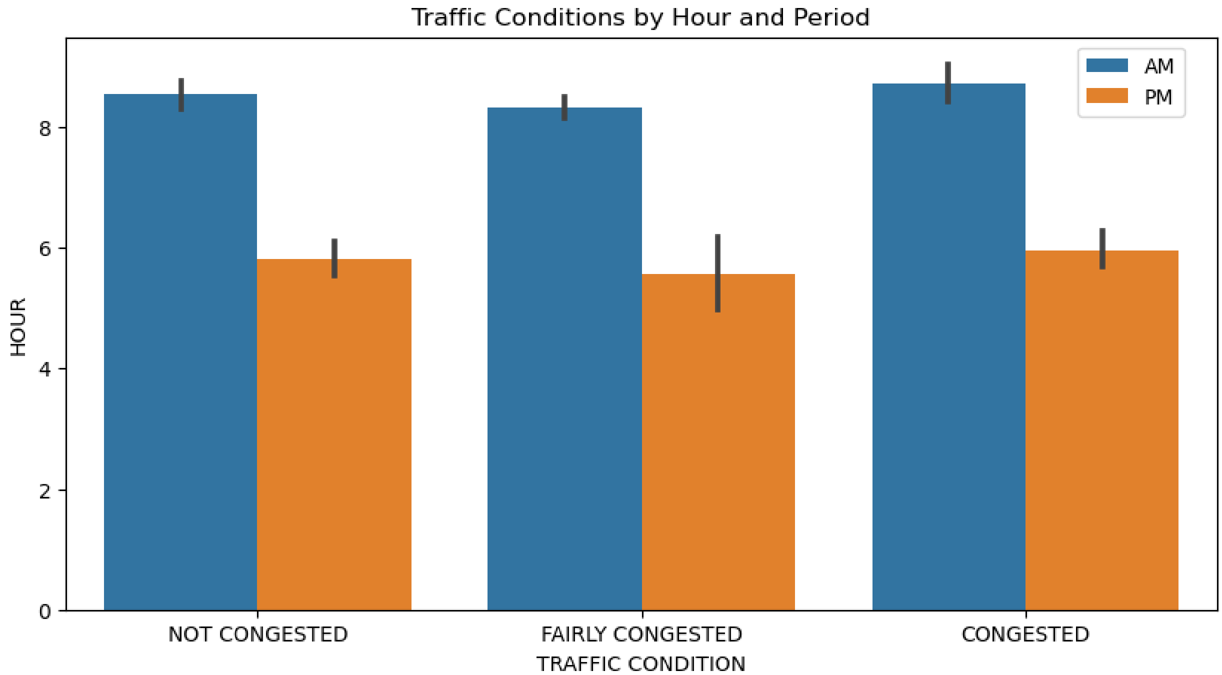
Out[25]:

	LOCATION	TIME	DAYS OF THE WEEK	TRAFFIC CONDITION	PERIOD	HOUR	MINUTE	TIME_IN_MINUTES
0	FROM AGBARA CUSTOM TO IYANA ERA	9.00	MONDAY	NOT CONGESTED	AM	9	0	540
1	FROM TRADE FAIR TO ABULER ADO	9.10	MONDAY	NOT CONGESTED	AM	9	10	550
2	FROM BARRACKS TO VOLKS	9.15	MONDAY	NOT CONGESTED	AM	9	15	555
3	FROM IYANA ISASHI TO AGBARA	9.04	MONDAY	NOT CONGESTED	AM	9	4	544
4	FROM MOBOLAJI JOHNSON TO 7UP	9.04	MONDAY	NOT CONGESTED	AM	9	4	544

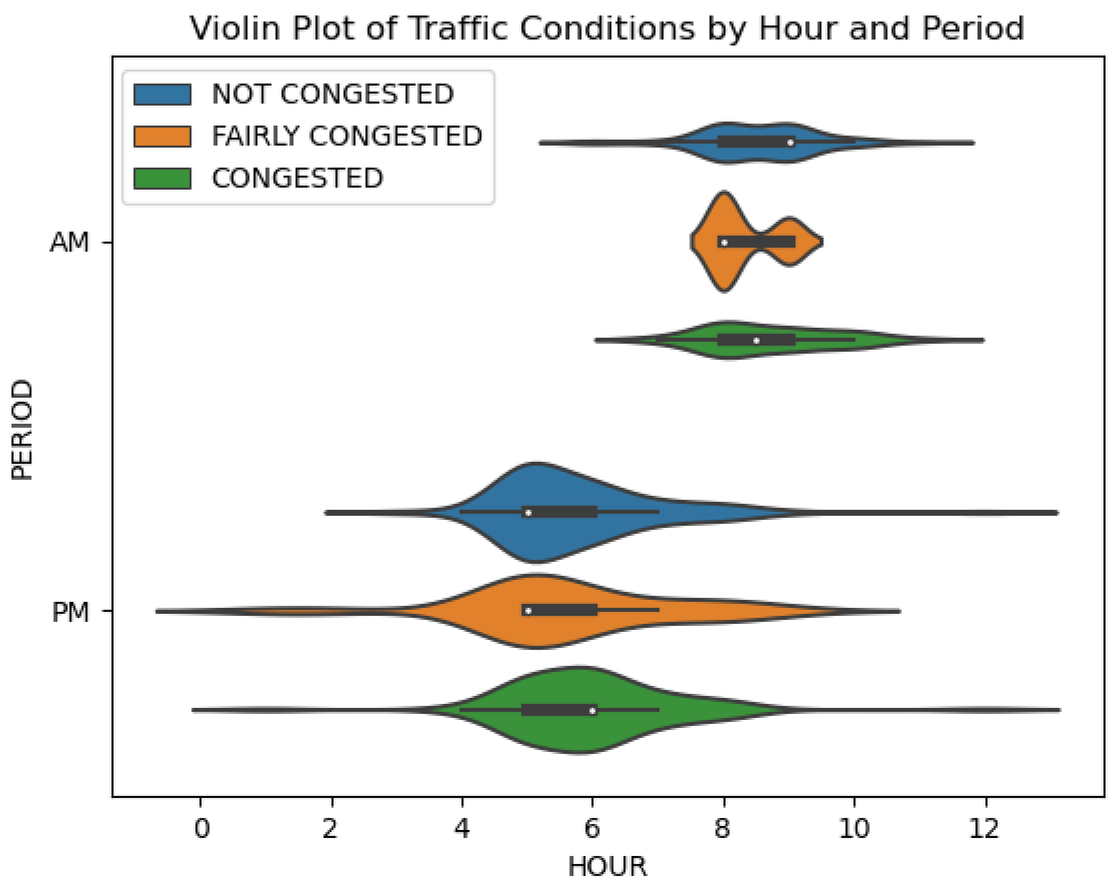
In [26]:
```
sns.histplot(data=df, x='TIME_IN_MINUTES', hue='PERIOD')
plt.title('Distribution of Time in Minutes by Period');
```



In [27]:
```
plt.figure(figsize=(10,5), dpi = 100)
sns.barplot(data=df, y='HOUR', x= 'TRAFFIC CONDITION', hue = 'PERIOD')
plt.legend(bbox_to_anchor= [0.87,1,0,0])
plt.title('Traffic Conditions by Hour and Period');
```

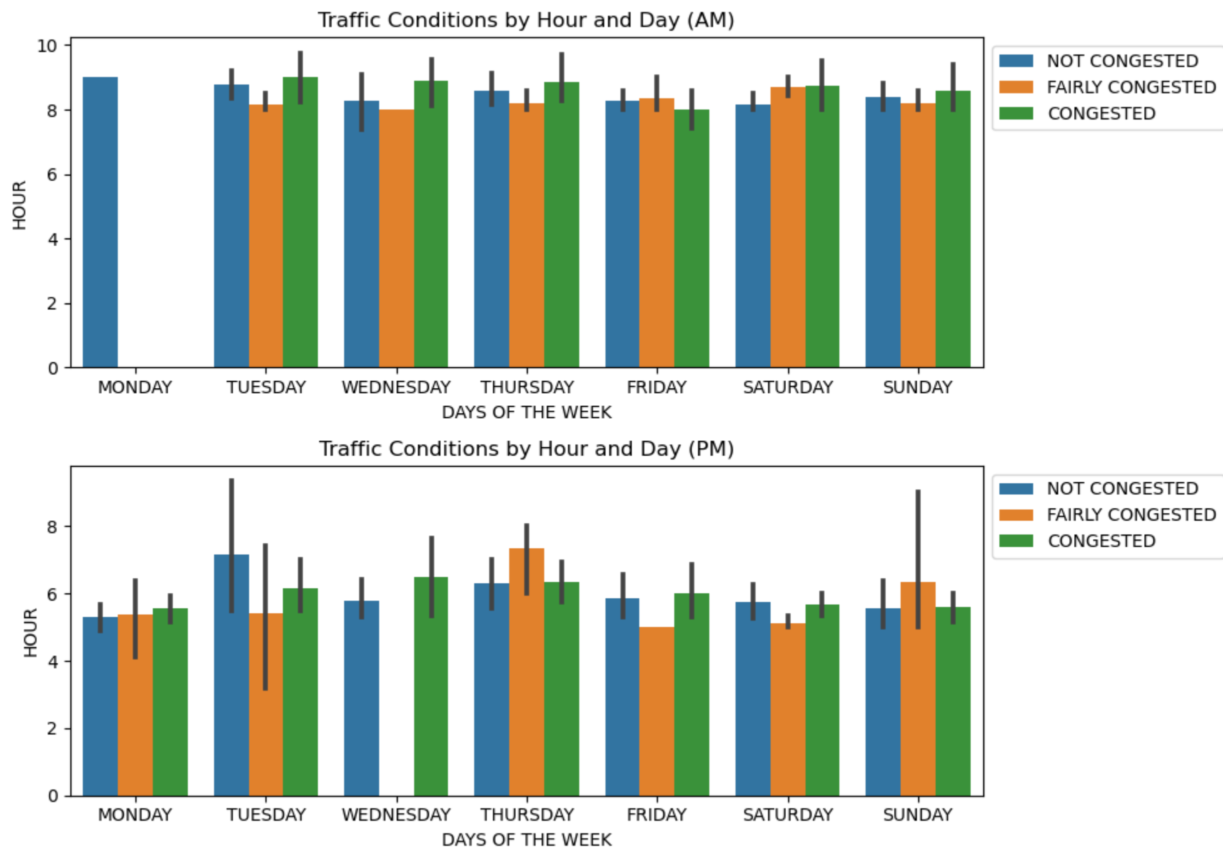


In [28]:
```
sns.violinplot(data=df, y='PERIOD', x= 'HOUR', hue='TRAFFIC CONDITION')
plt.legend(bbox_to_anchor= [0.4,1,0,0])
plt.title('Violin Plot of Traffic Conditions by Hour and Period');
```

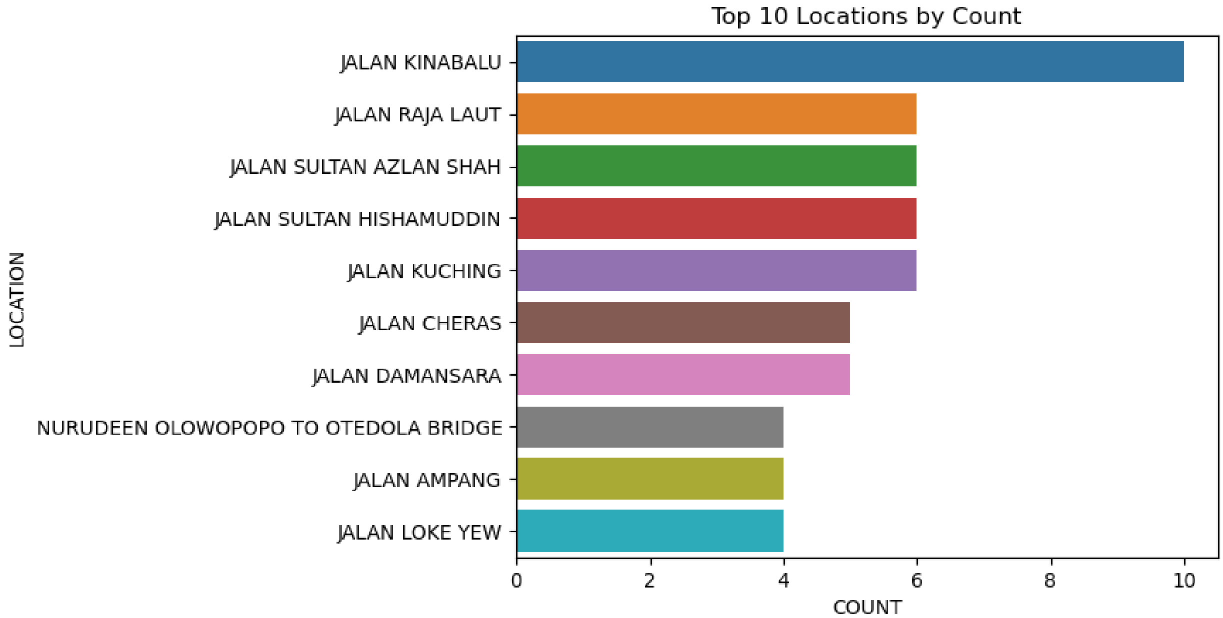


In [29]:
```
plt.figure(figsize=(10, 7))

# Subplot for 'AM' period
plt.subplot(2, 1, 1)
```

```
sns.barplot(data=df[df['PERIOD'] == 'AM'], y='HOUR', x='DAYS OF THE WEEK', hue='TRAFFI
plt.title('Traffic Conditions by Hour and Day (AM)')
plt.legend(bbox_to_anchor=[1, 1, 0, 0])

# Subplot for 'PM' period
plt.subplot(2, 1, 2)
sns.barplot(data=df[df['PERIOD'] == 'PM'], y='HOUR', x='DAYS OF THE WEEK', hue='TRAFFI
plt.title('Traffic Conditions by Hour and Day (PM)')
plt.legend(bbox_to_anchor=[1, 1, 0, 0])

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

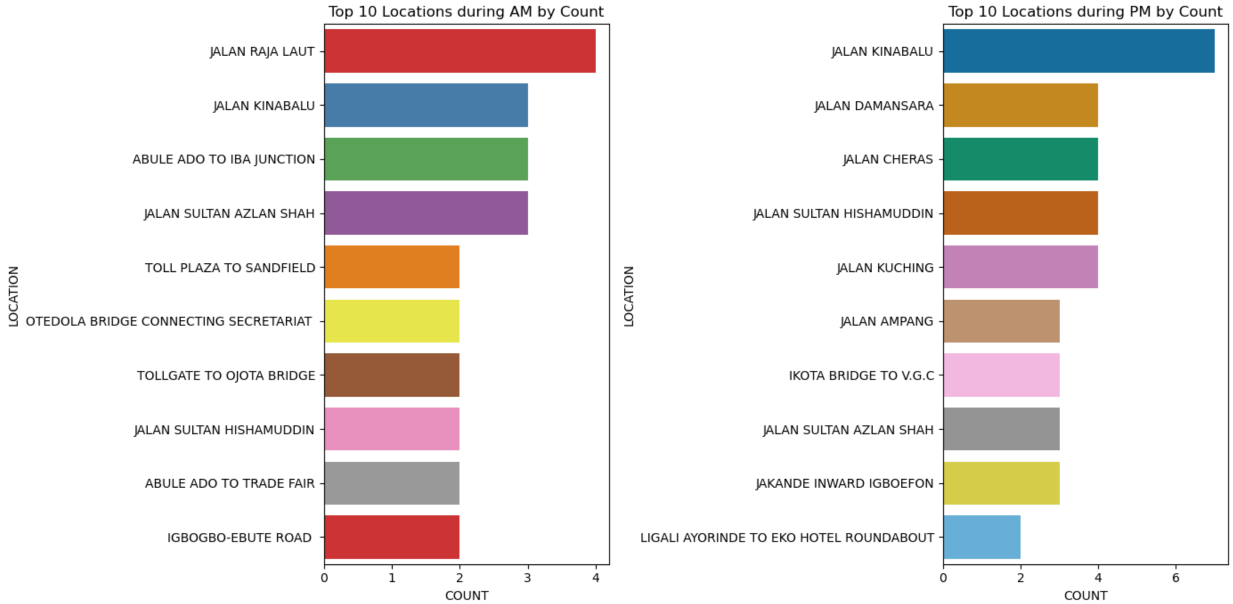


In [30]:
```
location_counts = df['LOCATION'].value_counts().reset_index()
location_counts.columns = ['LOCATION', 'COUNT']

top_n_locations = 10
sns.barplot(x='COUNT', y='LOCATION', data=location_counts.head(top_n_locations))
plt.title(f'Top {top_n_locations} Locations by Count');
```



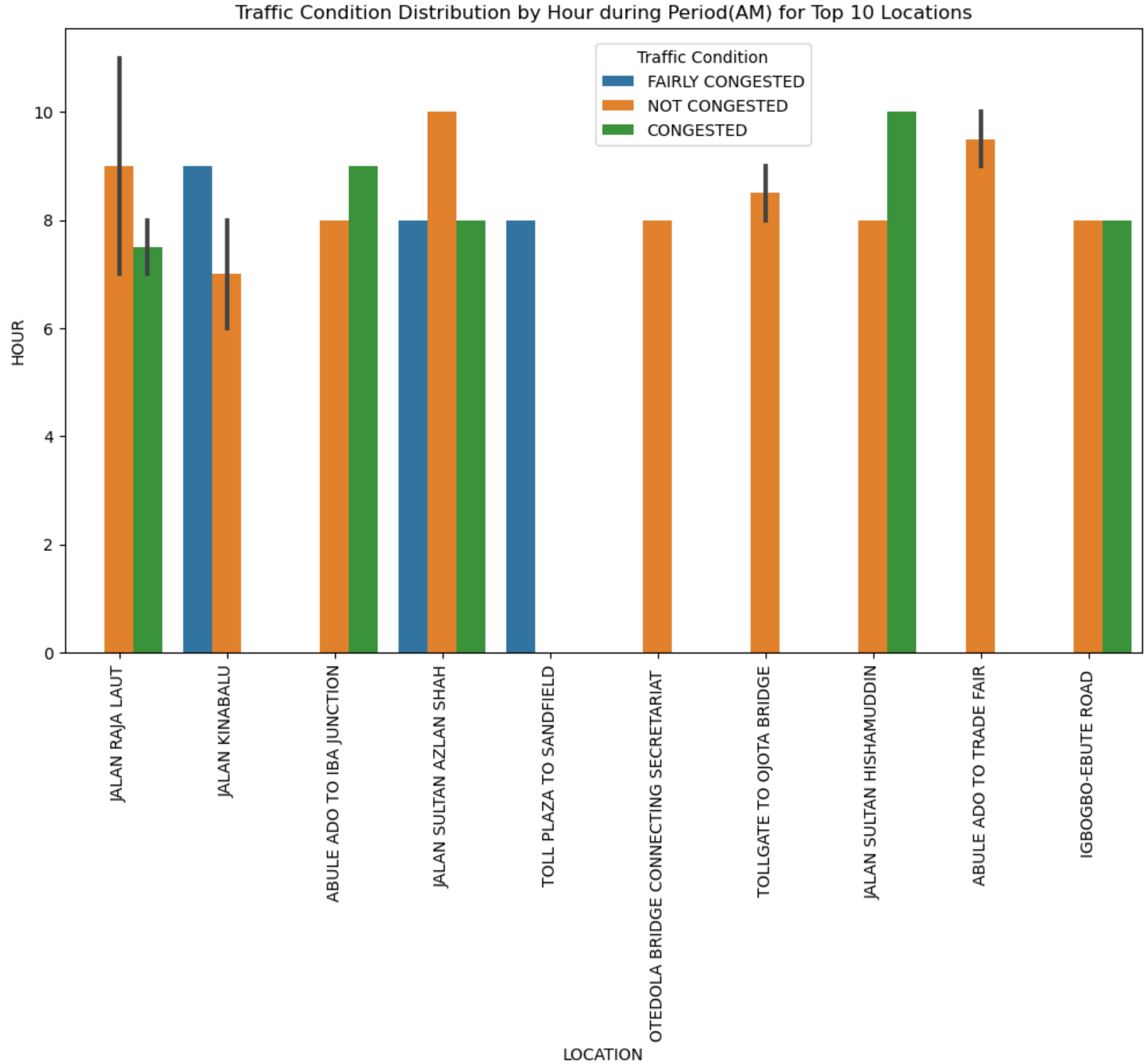
In [31]:

```
df_am = df[df['PERIOD'] == 'AM']
df_pm = df[df['PERIOD'] == 'PM']

# Set the number of top locations to display
top_n_locations = 10

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 7))

# Subplot for 'AM'
location_counts_am = df_am['LOCATION'].value_counts().reset_index()
location_counts_am.columns = ['LOCATION', 'COUNT']
sns.barplot(x='COUNT', y='LOCATION', data=location_counts_am.head(top_n_locations), ax
axes[0].set_title(f'Top {top_n_locations} Locations during AM by Count')

# Subplot for 'PM'
location_counts_pm = df_pm['LOCATION'].value_counts().reset_index()
location_counts_pm.columns = ['LOCATION', 'COUNT']
sns.barplot(x='COUNT', y='LOCATION', data=location_counts_pm.head(top_n_locations), ax
axes[1].set_title(f'Top {top_n_locations} Locations during PM by Count')

# Adjust layout
plt.tight_layout()
```

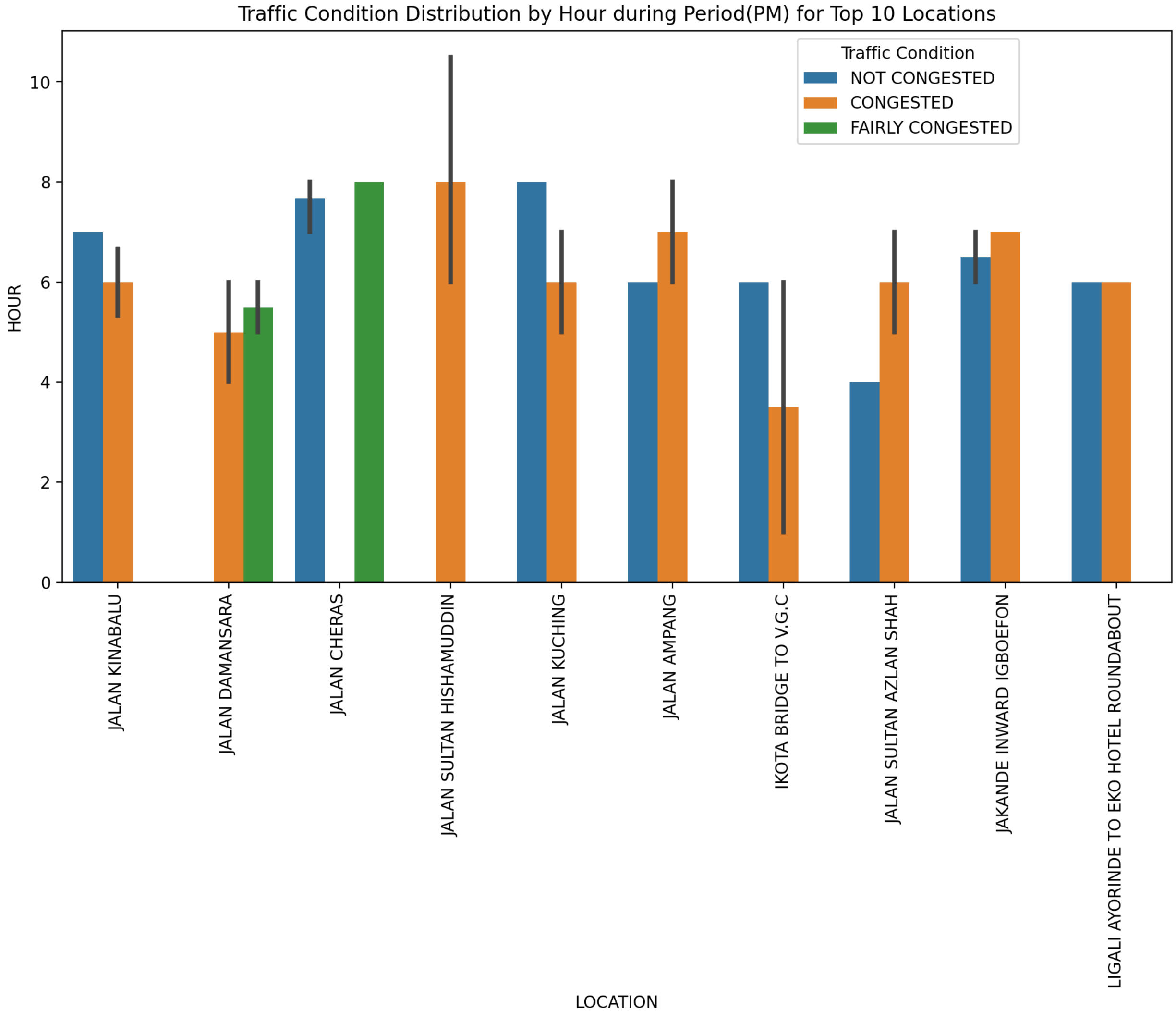


In [32]:

```
df_am = df[df['PERIOD'] == 'AM']
df_pm = df[df['PERIOD'] == 'PM']

# Set the number of top locations to display
top_n_locations = 10

# Create DataFrames for 'AM' and 'PM' top locations
top_locations_am = df_am['LOCATION'].value_counts().head(top_n_locations).index
df_am_top_locations = df_am[df_am['LOCATION'].isin(top_locations_am)]

top_locations_pm = df_pm['LOCATION'].value_counts().head(top_n_locations).index
df_pm_top_locations = df_pm[df_pm['LOCATION'].isin(top_locations_pm)]

# Plot for 'AM'
plt.figure(figsize=(12, 7), dpi=100)
sns.barplot(x='LOCATION', y='HOUR', hue='TRAFFIC CONDITION', data=df_am_top_locations,
plt.title(f'Traffic Condition Distribution by Hour during Period(AM) for Top {top_n_lc
plt.legend(title='Traffic Condition', bbox_to_anchor=[0.7, 0.8, 0, 0])
plt.xticks(rotation=90)
plt.show()
```

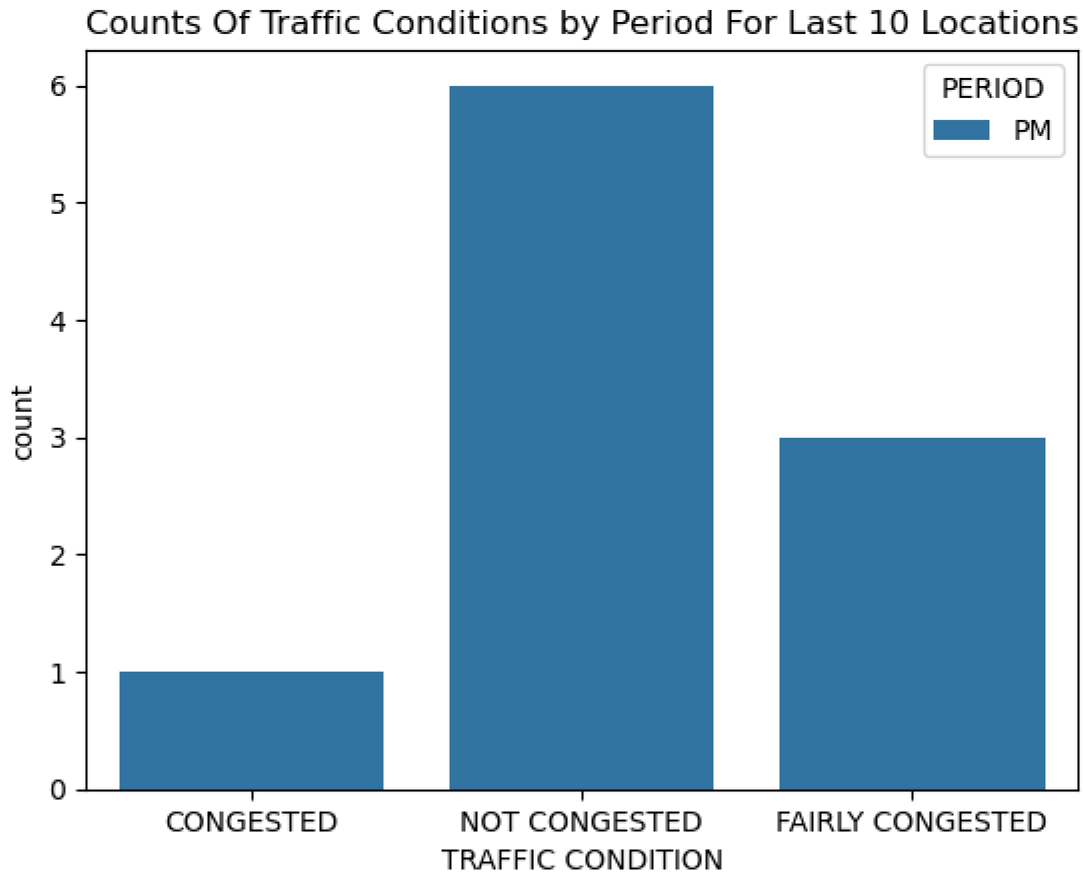


In [33]:
```
# Plot for 'PM'
plt.figure(figsize=(12, 6), dpi=200)
sns.barplot(x='LOCATION', y='HOUR', hue='TRAFFIC CONDITION', data=df_pm_top_locations,
plt.title(f'Traffic Condition Distribution by Hour during Period(PM) for Top {top_n_lo
plt.legend(title='Traffic Condition', bbox_to_anchor=[0.87, 1, 0, 0])
plt.xticks(rotation=90)
plt.show()
```



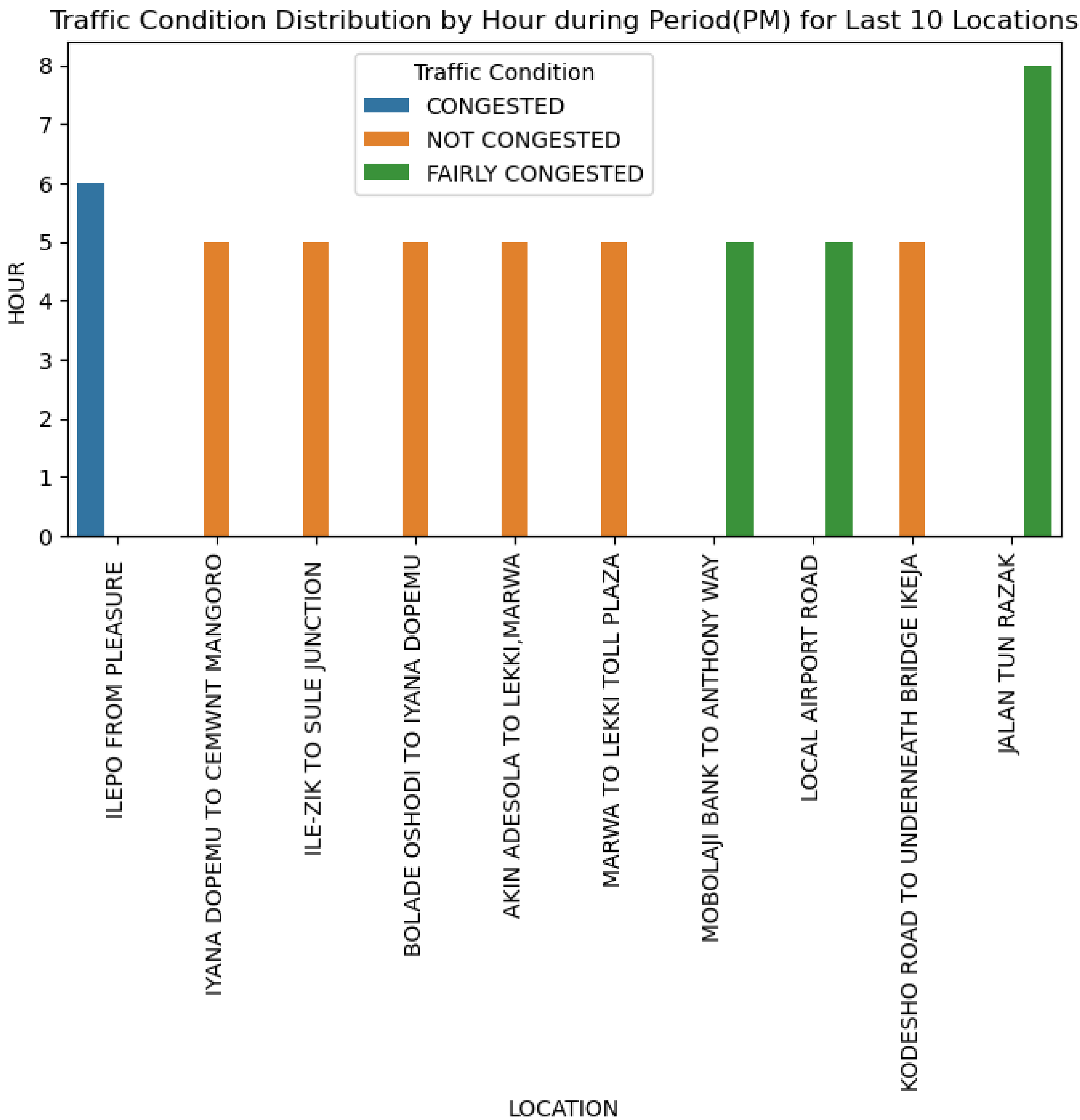
In [34]:

```
last_n_locations = 10

# Filter the DataFrame for the last locations
last_locations = location_counts.tail(last_n_locations)['LOCATION']
df_last_locations = df[df['LOCATION'].isin(last_locations)]

sns.countplot(data = df_last_locations, x='TRAFFIC CONDITION',hue='PERIOD')
plt.title('Counts Of Traffic Conditions by Period For Last 10 Locations');
```



In [48]:

```
# Plot for 'PM'
plt.figure(figsize=(8,4))
df_last_locations_pm =df_last_locations[df_last_locations['PERIOD']=='PM']
sns.barplot(x='LOCATION', y='HOUR', hue='TRAFFIC CONDITION', data=df_last_locations_pm
plt.title(f'Traffic Condition Distribution by Hour during Period(PM) for Last {last_n_
plt.legend(title='Traffic Condition', bbox_to_anchor=[0.6, 1, 0, 0])
plt.xticks(rotation=90);
```



In [134…

```
data = df
```

In [135…

```
data.head()
```

Out[135]:

	LOCATION	TIME	DAYS OF THE WEEK	TRAFFIC CONDITION	PERIOD	HOUR	MINUTE	TIME_IN_MINUTES
0	FROM AGBARA CUSTOM TO IYANA ERA	9.00	MONDAY	NOT CONGESTED	AM	9	0	540
1	FROM TRADE FAIR TO ABULER ADO	9.10	MONDAY	NOT CONGESTED	AM	9	10	550
2	FROM BARRACKS TO VOLKS	9.15	MONDAY	NOT CONGESTED	AM	9	15	555
3	FROM IYANA ISASHI TO AGBARA	9.04	MONDAY	NOT CONGESTED	AM	9	4	544
4	FROM MOBOLAJI JOHNSON TO 7UP	9.04	MONDAY	NOT CONGESTED	AM	9	4	544

In [136…

```
data = data.drop(['TIME', 'MINUTE'], axis=1)
```

In [137…

```
data
```

Out[137]:

	LOCATION	DAYS OF THE WEEK	TRAFFIC CONDITION	PERIOD	HOUR	TIME_IN_MINUTES
0	FROM AGBARA CUSTOM TO IYANA ERA	MONDAY	NOT CONGESTED	AM	9	540
1	FROM TRADE FAIR TO ABULER ADO	MONDAY	NOT CONGESTED	AM	9	550
2	FROM BARRACKS TO VOLKS	MONDAY	NOT CONGESTED	AM	9	555
3	FROM IYANA ISASHI TO AGBARA	MONDAY	NOT CONGESTED	AM	9	544
4	FROM MOBOLAJI JOHNSON TO 7UP	MONDAY	NOT CONGESTED	AM	9	544
...	...	...	...	...	...	...
294	JALAN KINABALU	SATURDAY	CONGESTED	PM	6	412
295	NURUDEEN OLOWOPOPO TO OTEDOLA BRIDGE	SATURDAY	FAIRLY CONGESTED	PM	6	412
296	JALAN TRAVERS	SUNDAY	CONGESTED	AM	10	600
297	JALAN RAJA LAUT	SUNDAY	CONGESTED	AM	8	480
298	JALAN RAJA	SUNDAY	CONGESTED	AM	8	481

299 rows × 6 columns

In [138…

```
X = data.drop('TRAFFIC CONDITION', axis=1)
```

In [141…

```
label_encoder = LabelEncoder()
data['TRAFFIC CONDITION'] = label_encoder.fit_transform(data['TRAFFIC CONDITION'])
```

In [148…

```
# Access the mapping
label_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder

# Print the mapping
print("Label Mapping:")
print(label_mapping)
```

```
Label Mapping:
{'CONGESTED': 0, 'FAIRLY CONGESTED': 1, 'NOT CONGESTED': 2}
```

In [149…

```
y = data['TRAFFIC CONDITION']
```

In [189…

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping
```

In [155…

```
y.unique()
```

Out[155]:

```
array([2, 1, 0])
```

In [156…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [358…

```
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.1))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.add(Dropout(0.1))  # Adjust the dropout rate as needed
optimizer = Adam(learning_rate=0.0001)  # Adjust the learning rate as needed
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['a
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)


early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=Tr
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.1, callbacks
```

```
Epoch 1/10
7/7 [==============================] - 1s 40ms/step - loss: 2.0143 - accuracy: 0.3163
- val_loss: 1.1656 - val_accuracy: 0.3333
Epoch 2/10
7/7 [==============================] - 0s 10ms/step - loss: 2.3884 - accuracy: 0.3023
- val_loss: 1.1431 - val_accuracy: 0.3750
Epoch 3/10
7/7 [==============================] - 0s 12ms/step - loss: 2.3332 - accuracy: 0.3023
- val_loss: 1.1257 - val_accuracy: 0.3750
Epoch 4/10
7/7 [==============================] - 0s 13ms/step - loss: 2.7143 - accuracy: 0.3256
- val_loss: 1.1124 - val_accuracy: 0.3750
Epoch 5/10
7/7 [==============================] - 0s 13ms/step - loss: 2.4196 - accuracy: 0.3256
- val_loss: 1.0987 - val_accuracy: 0.4167
Epoch 6/10
7/7 [==============================] - 0s 12ms/step - loss: 2.3962 - accuracy: 0.3907
- val_loss: 1.0865 - val_accuracy: 0.4167
Epoch 7/10
7/7 [==============================] - 0s 13ms/step - loss: 2.6453 - accuracy: 0.3814
- val_loss: 1.0753 - val_accuracy: 0.3750
Epoch 8/10
7/7 [==============================] - 0s 11ms/step - loss: 2.4761 - accuracy: 0.4279
- val_loss: 1.0671 - val_accuracy: 0.4583
Epoch 9/10
7/7 [==============================] - 0s 12ms/step - loss: 2.2431 - accuracy: 0.4093
- val_loss: 1.0601 - val_accuracy: 0.4583
Epoch 10/10
7/7 [==============================] - 0s 14ms/step - loss: 2.2691 - accuracy: 0.4605
- val_loss: 1.0538 - val_accuracy: 0.4583
Epoch 1/100
7/7 [==============================] - 0s 18ms/step - loss: 2.6342 - accuracy: 0.4186
- val_loss: 1.0473 - val_accuracy: 0.5000
Epoch 2/100
7/7 [==============================] - 0s 13ms/step - loss: 2.6876 - accuracy: 0.4140
- val_loss: 1.0418 - val_accuracy: 0.5000
Epoch 3/100
7/7 [==============================] - 0s 10ms/step - loss: 2.5570 - accuracy: 0.4419
- val_loss: 1.0356 - val_accuracy: 0.5000
Epoch 4/100
7/7 [==============================] - 0s 11ms/step - loss: 2.6614 - accuracy: 0.4279
- val_loss: 1.0294 - val_accuracy: 0.5000
Epoch 5/100
7/7 [==============================] - 0s 11ms/step - loss: 2.9037 - accuracy: 0.4419
- val_loss: 1.0245 - val_accuracy: 0.5000
Epoch 6/100
7/7 [==============================] - 0s 11ms/step - loss: 2.6852 - accuracy: 0.4698
- val_loss: 1.0200 - val_accuracy: 0.5000
Epoch 7/100
7/7 [==============================] - 0s 10ms/step - loss: 2.2267 - accuracy: 0.5023
- val_loss: 1.0172 - val_accuracy: 0.5417
Epoch 8/100
7/7 [==============================] - 0s 11ms/step - loss: 2.3075 - accuracy: 0.4884
- val_loss: 1.0153 - val_accuracy: 0.5000
Epoch 9/100
7/7 [==============================] - 0s 12ms/step - loss: 2.1496 - accuracy: 0.4884
- val_loss: 1.0139 - val_accuracy: 0.5000
Epoch 10/100
7/7 [==============================] - 0s 13ms/step - loss: 2.1100 - accuracy: 0.5256
- val_loss: 1.0131 - val_accuracy: 0.4583
```

```
Epoch 11/100
7/7 [==============================] - 0s 11ms/step - loss: 2.2281 - accuracy: 0.4744
- val_loss: 1.0113 - val_accuracy: 0.4583
Epoch 12/100
7/7 [==============================] - 0s 11ms/step - loss: 2.4076 - accuracy: 0.4465
- val_loss: 1.0109 - val_accuracy: 0.4583
Epoch 13/100
7/7 [==============================] - 0s 12ms/step - loss: 2.6680 - accuracy: 0.4930
- val_loss: 1.0095 - val_accuracy: 0.4583
Epoch 14/100
7/7 [==============================] - 0s 10ms/step - loss: 2.4070 - accuracy: 0.4930
- val_loss: 1.0087 - val_accuracy: 0.5000
Epoch 15/100
7/7 [==============================] - 0s 10ms/step - loss: 2.6329 - accuracy: 0.4837
- val_loss: 1.0058 - val_accuracy: 0.5000
Epoch 16/100
7/7 [==============================] - 0s 10ms/step - loss: 2.1595 - accuracy: 0.5442
- val_loss: 1.0048 - val_accuracy: 0.5000
Epoch 17/100
7/7 [==============================] - 0s 10ms/step - loss: 2.2861 - accuracy: 0.5256
- val_loss: 1.0037 - val_accuracy: 0.5000
Epoch 18/100
7/7 [==============================] - 0s 10ms/step - loss: 2.5021 - accuracy: 0.5442
- val_loss: 1.0027 - val_accuracy: 0.5000
Epoch 19/100
7/7 [==============================] - 0s 10ms/step - loss: 2.2018 - accuracy: 0.5721
- val_loss: 1.0011 - val_accuracy: 0.5417
Epoch 20/100
7/7 [==============================] - 0s 10ms/step - loss: 2.2621 - accuracy: 0.5535
- val_loss: 0.9994 - val_accuracy: 0.5417
Epoch 21/100
7/7 [==============================] - 0s 10ms/step - loss: 1.9867 - accuracy: 0.5721
- val_loss: 0.9985 - val_accuracy: 0.5417
Epoch 22/100
7/7 [==============================] - 0s 10ms/step - loss: 2.3526 - accuracy: 0.5535
- val_loss: 0.9968 - val_accuracy: 0.5417
Epoch 23/100
7/7 [==============================] - 0s 11ms/step - loss: 1.9081 - accuracy: 0.5349
- val_loss: 0.9972 - val_accuracy: 0.5417
Epoch 24/100
7/7 [==============================] - 0s 9ms/step - loss: 2.3362 - accuracy: 0.5814
- val_loss: 0.9980 - val_accuracy: 0.5417
Epoch 25/100
7/7 [==============================] - 0s 9ms/step - loss: 2.4168 - accuracy: 0.5814
- val_loss: 0.9982 - val_accuracy: 0.5417
Epoch 26/100
7/7 [==============================] - 0s 10ms/step - loss: 2.6101 - accuracy: 0.5302
- val_loss: 0.9998 - val_accuracy: 0.5417
Epoch 27/100
7/7 [==============================] - 0s 10ms/step - loss: 2.2659 - accuracy: 0.5860
- val_loss: 1.0003 - val_accuracy: 0.5417
```

Out[358]:

```
<keras.callbacks.History at 0x201a31da310>
```

In [359…

```
# Evaluate the model
y_probs = model.predict(X_test)
y_pred = np.argmax(y_probs, axis=1)
```

```
2/2 [==============================] - 0s 4ms/step
```

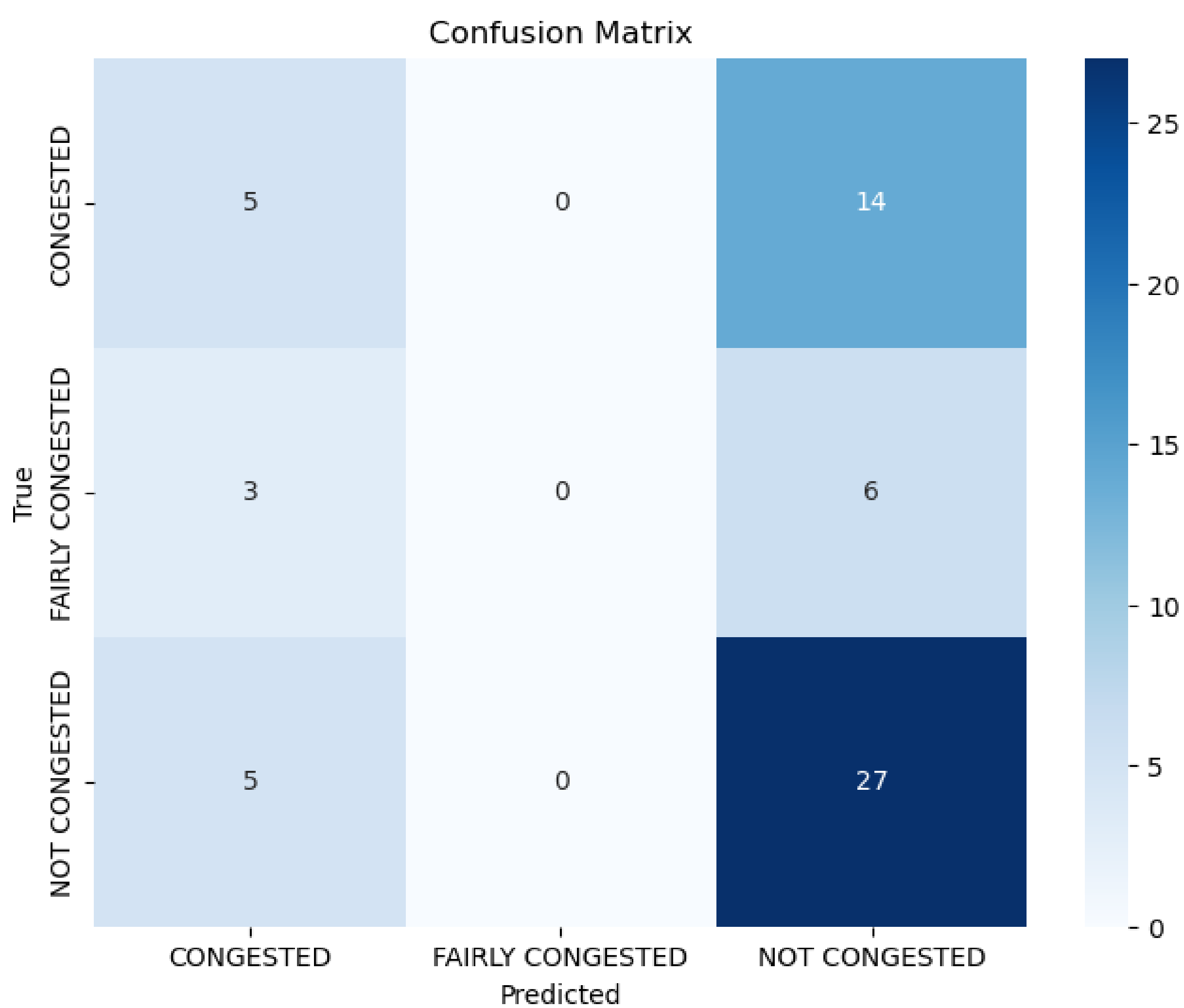
In [360…

```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.38      0.26      0.31        19
           1       0.00      0.00      0.00         9
           2       0.57      0.84      0.68        32

    accuracy                           0.53        60
   macro avg       0.32      0.37      0.33        60
weighted avg       0.43      0.53      0.46        60
```

In [365…

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_mapping,
            yticklabels=label_mapping)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [366…

```
# Calculate and print the total accuracy
total_accuracy = accuracy_score(y_test, y_pred)
print(f'Total Accuracy: {total_accuracy:.4f}')
```

```
Total Accuracy: 0.5333
```

In [367…

```
model.save("traffic_condition_model.h5")
```

In [ ]: