Peak Hour Traffic Analysis — Instructions, Methods & Report Template

Author: Traffic Analyst (template)

Purpose: This document explains, step-by-step, how to produce a predictive model and a comprehensive report that forecasts hourly traffic volumes at road junctions, identifies peak hours and patterns, and provides visualisations and actionable recommendations.

---

1. Executive Summary (what this deliverable contains)

- A repeatable pipeline to aggregate raw traffic sensor data into hourly summaries.
- Definition and computation of congestion metrics (vehicle counts, speed reduction, congestion index).
- Methods to identify peak hours (statistical & algorithmic).
- Temporal pattern analysis (weekday/weekend, monthly/seasonal, holidays & events).
- Analysis of external factors (weather, events) and how to include them as regressors.
- Predictive modelling approaches (baselines, classical time-series, ML, deep learning), evaluation measures and model selection guidance.
- Data visualisations (heatmaps, line plots, scatter plots, seasonal plots) and a final report template with recommendations.

---

2. Required Inputs / Data schema

Minimum fields (one row per sensor reading or aggregated measurement):

- `timestamp` — precise UTC or local datetime (ISO format). Example: `2024-06-01 07:23:00`.
- `junction_id` or `sensor_id` — unique identifier for the junction.
- `vehicle_count` — number of vehicles observed in the measurement period.
- `avg_speed` — average speed (km/h or mph) of vehicles in the period.

Highly recommended external fields (for better explanations & predictions):

- `lane_count` or `vehicle_capacity` (if available)
- `occupancy` or `headway`
- Weather: `temperature`, `precipitation`, `wind_speed`, `visibility`, `condition` (categorical: rain/snow/clear)
- `event_flag` or `event_type` (boolean/label for special events)
- `roadwork` or `incident_flag`

File format: CSV, Parquet or a database export. Timezone must be known. If timestamps are in different timezones, normalize to a single timezone (prefer local time).

---

3. Pipeline (high level)

1. **Ingest & store raw data** — keep raw files untouched for reproducibility.
2. **Preprocess** — parse timestamps, align timezones, handle duplicates, mark missing windows.
3. **Aggregate to hourly** — resample readings to hourly summaries per `junction_id`.
4. **Compute congestion metrics** — define and compute metrics (see §4).
5. **Exploratory Data Analysis (EDA)** — visualize hourly profiles, heatmaps and distributions.
6. **Feature engineering** — create lags, rolling stats, seasonality terms and external regressors.
7. **Modeling** — baseline, time-series and ML models.
8. **Evaluate & validate** — time-based CV and holdout tests.

9. **Visualize & report** — produce heatmaps, line charts and a final PDF report with insights.

---

4. Congestion metrics (definitions + how to compute)

Use a few complementary metrics:

1. **Average vehicle count per hour** — (sum or mean depending on sensor frequency). Aggregate per junction and hour-of-day.

2. **Average speed reduction** — defines how much slower traffic is compared to free-flow baseline.
- `speed_reduction = max(0, baseline_speed - avg_speed)`.
- `baseline_speed` can be estimated as the 85th percentile speed during off-peak hours or computed per junction.

3. **Congestion Index (CI)** — example formula (normalized combining count & speed):

```
CI = w1 * normalize(vehicle_count) + w2 * normalize(speed_reduction)
```

Choose weights `w1,w2` (default 0.6/0.4) or use PCA to derive combined index.

4. **Vehicle per lane / capacity ratio** — useful if lane counts are known.

5. **Delay estimate** — using speed and distance to estimate average delay per vehicle (if link length known).

Implementation notes:
- Normalize per-junction before combining across junctions (z-score or min-max by junction).
- Compute metrics for every hourly bin and store as a table: `(junction_id, timestamp_hour, vehicle_count, avg_speed, CI, ...)`.

---

5. Peak hour identification (algorithms & approaches)

5.1 Simple ranking

- For each junction compute mean vehicle_count (or CI) by `hour_of_day` across the dataset and pick the top `k` hours.

5.2 Statistical thresholds

- Compute moving average (e.g., 3-hour MA) and standard deviation.
- Mark an hour as 'peak' if `CI > mean(CI_hour) + 1.5 * std(CI_hour)` for that hour-of-day.

5.3 Consistency filtering

- A peak hour is *consistent* if it appears as a peak in > X% of weeks (e.g., 60%). This filters one-off spikes.

5.4 Clustering hourly profiles

- For each junction, build a 24-dimensional vector (average CI for each hour). Use KMeans or hierarchical clustering to group similar daily profiles (e.g., morning rush, evening rush, bi-modal, flat).

5.5 Change point detection

- Use change-point detection (ruptures library or Bayesian change point) on the hourly CI series to identify sudden regime shifts (new congestion patterns).

---

6. Temporal patterns to examine

- **Weekday vs Weekend**: compute separate hourly profiles for weekdays and weekends and compare.
- **Day-of-week differences**: e.g., Fridays may have later evening peaks.
- **Monthly/seasonal**: compare monthly averages; include school holidays and daylight savings checks.
- **Special days**: holidays, festival days, or stadium events.

Visualization tips:
- Heatmap with `hour_of_day` on x-axis and `date` or `day_of_week` on y-axis.
- Small multiples: one heatmap per month.

---

7. External factors (weather & events)

7.1 Correlation & initial checks

- Merge weather and event data by hour and junction (or nearest weather station).
- Compute correlation matrix; use partial correlations to control for time-of-day.

7.2 Regression & importance

- Build a simple linear regression or tree-based model (RandomForest/LightGBM) to estimate feature importances of weather and event flags.
- Use SHAP values to get local/global explanations.

7.3 Causality checks

- Granger causality for time series (weather variables causing vehicle count changes).
- Be cautious — correlation ≠ causation; events may be confounded with time-of-day.

---

8. Predictive modelling approaches (detailed steps)

8.1 Baselines

- **Naive persistence**: predict same hour value from prior week (t-168h) or previous day (t-24h).
- **Rolling mean**: average of same hour over last N weeks.

8.2 Time-series models (per junction)

- **SARIMA / SARIMAX** — seasonal ARIMA with exogenous regressors (weather/events).
- **Prophet** — easy to include holidays and regressors (if package available).

Pros: interpretable seasonality; Cons: needs one model per junction or hierarchical approach.

8.3 Machine learning regression

- **LightGBM / XGBoost / RandomForest** using tabular features (recommended for speed & accuracy):
- Features: `hour_of_day`, `day_of_week`, `is_weekend`, `month`, lag features (t-1, t-24, t-168), rolling means (3h, 24h, 168h), weather features and event flags, holiday flag, junction encoding (one-hot or embedding).
- Target: `vehicle_count` (or CI). If counts are highly skewed, consider log-transform.

8.4 Deep learning

- **LSTM / GRU**: sequences of hourly data for each junction.
- **Temporal Convolutional Networks (TCN)** or Transformer models for long-range dependencies.
- Consider multi-task networks with shared encoder + junction-specific decoder.

8.5 Multi-junction vs per-junction training

- **Per-junction models** if each junction has lots of data and different behaviours.
- **Global model** with junction_id encoded (categorical embedding) to learn shared temporal patterns across junctions — usually more data-efficient.

---

9. Evaluation strategy

- **Train/Validation/Test split** must be time based (no shuffling):
- e.g., train: 70% earliest data, validation: next 15%, test: final 15%.
- **Rolling-origin (Walk-forward) CV**: repeat retraining/predicting in time windows.

Metrics:
- **MAE** (primary for interpretable absolute errors)
- **RMSE**
- **MAPE / sMAPE** (careful with near-zero counts)
- **R^2**
- **Peak detection metrics**: treat peak hours as a classification task (precision/recall/F1 for correctly predicting the top-k hours per day).

Model selection: prefer models with lowest validation MAE and stable performance across junctions & time windows.

---

10. Visualization recommendations (examples)

- **Hourly profile line plot**: mean vehicle_count by `hour_of_day` with confidence interval shaded.
- **Heatmap**: `hour_of_day` vs `date` (or `day_of_week`) colored by CI.
- **Monthly small multiples**: separate heatmap per month.
- **Scatter**: vehicle_count vs avg_speed, color by weather condition.
- **Feature importance**: bar chart and SHAP summary plot for tree-based models.
- **Calendar or timeline**: annotate special events and incidents.

Tools: Python: `pandas`, `matplotlib`, `seaborn` (or plotly for interactive), `folium` if spatial maps are needed.

---

11. Actionable recommendations (example)

- Staggered work hours or school schedules around identified consistent peak windows.
- Dynamic traffic signal timing around morning/evening peaks.
- Real-time traveler information and route suggestions for commuters.
- Use temporary lane management or contraflow during predictable high-demand events.

---

12. Deliverables & Report structure (what to include in the PDF)

- Title & executive summary (1 page)

- Data description & preprocessing steps (1-2 pages)
- Congestion metrics & peak hour identification methods (2 pages)
- Visualizations (heatmaps, line plots, scatter plots) (3-6 pages)
- Modeling approach, evaluation and results (3-6 pages)
- Impact of external factors (1-2 pages)
- Recommendations & next steps (1-2 pages)
- Appendix: code snippets & data dictionary (2-4 pages)

---

13. Reproducible code snippets (Python/pandas + LightGBM example)

> **Note:** these are templates — paths, column names and parameter choices must be adapted to your dataset.

```
# basics
import pandas as pd
import numpy as np

# read data
df = pd.read_csv('traffic_raw.csv', parse_dates=['timestamp'])

# normalize timezone (example: Asia/Kolkata)
df['timestamp'] = df['timestamp'].dt.tz_localize(None)

# aggregate to hourly per junction
hourly = (
    df.set_index('timestamp')
      .groupby('junction_id')
      ['vehicle_count','avg_speed']
      .resample('H')
      .agg({'vehicle_count':'sum','avg_speed':'mean'})
      .reset_index()
)

# feature engineering: time features
hourly['hour'] = hourly['timestamp'].dt.hour
hourly['dow'] = hourly['timestamp'].dt.dayofweek
hourly['is_weekend'] = hourly['dow'].isin([5,6]).astype(int)

# create lag features
def add_lags(g, lags=[1,24,168]):
    g = g.sort_values('timestamp')
    for l in lags:
        g[f'lag_{l}'] = g['vehicle_count'].shift(l)
    return g

hourly = hourly.groupby('junction_id').apply(add_lags).reset_index(drop=True)

# rolling stats
hourly['roll_3h_mean'] = hourly.groupby('junction_id')['vehicle_count'].transform(lambda s: s.shift(1).rolling(

# drop rows with NaNs originating from lags
hourly = hourly.dropna(subset=['lag_168'])

# train-test split (time-based)
cutoff = hourly['timestamp'].quantile(0.85)
train = hourly[hourly['timestamp'] <= cutoff]
test = hourly[hourly['timestamp'] > cutoff]

# LightGBM training example
import lightgbm as lgb
from sklearn.metrics import mean_absolute_error

features = ['hour','dow','is_weekend','lag_1','lag_24','lag_168','roll_3h_mean']
train_set = lgb.Dataset(train[features], label=train['vehicle_count'])
valid_set = lgb.Dataset(test[features], label=test['vehicle_count'])
params = {'objective':'regression','metric':'mae','verbosity':-1}
```

```
model = lgb.train(params, train_set, valid_sets=[valid_set], early_stopping_rounds=50, num_boost_round=1000)

pred = model.predict(test[features])
print('MAE:', mean_absolute_error(test['vehicle_count'], pred))
```

---

14. Quick checklist for running on your data

1. Ensure timestamps are clean and timezone-correct.
2. Aggregate raw readings to hourly per junction.
3. Compute baseline speeds and CI per junction.
4. Visualize hourly profiles to confirm data quality.
5. Build lag and rolling features; add weather/events as regressors.
6. Start with simple baseline models; then try LightGBM and SARIMAX.
7. Evaluate with time-aware CV and report MAE/RMSE + peak detection accuracy.

---

15. How I can help next (options)

- **Option A (analysis only)**: I provide a filled PDF report (this template + examples) — already prepared and attached.
- **Option B (hands-on)**: You upload the dataset (CSV) and I run the full pipeline on your data, train models, produce figures and deliver an updated PDF with the results.
- **Option C (code notebook)**: I produce a runnable Jupyter notebook (`.ipynb`) that implements the pipeline and can be executed locally or on Colab.

Please tell me which option you prefer, or upload your data now and I will run Option B.

Appendix: file-naming & submission

- Use `Peak_Hour_Traffic_Analysis_<YourName>.pdf` as the final filename (max 10 MB).
- If you want me to run the pipeline on your data, upload a CSV (zipped if >10 MB) and include a short data dictionary describing column names.

End of template.