Replication & Learning Guide

(README + User Guide + Data Dictionary for `Dataset_v3`)

Prepared for researchers in DTA/VDF modeling

October 22, 2025

Abstract

This document is the authoritative README/User Guide for the I-405 bottleneck replication package (`Dataset_v3`). It provides (i) a project overview, (ii) directory structure, (iii) environment setup, (iv) step-by-step replication procedures mapping scripts to figures/analyses, (v) input–output checks and quality control (QC), and (vi) a machine-readable data dictionary. The package compares classical empirical VDFs (e.g., BPR), theory-driven models (FD/queue-based), and AI models (LSTM/Transformer/PINN), using loop-detector style data on I-405.

Contents

1 Project Overview

Goal. Reproduce analyses and figures for a VDF (Volume–Delay Function) study that bridges classical LPFs/VDFs, fluid-queue and FD-based approaches, and AI/PINN methods. The dataset centers on an I-405 bottleneck corridor with day-level Excel files (flows, speeds, densities, observed travel times) and a spatio-temporal speed heatmap.

Audience. Researchers familiar with traffic assignment, FD/queue theory, and ML baselines; students can use the learning checklists to verify inputs/outputs.

What you can do with this package:

- Generate bottleneck heatmaps and identify congestion windows.
- Calibrate/compare VDFs: BPR, FD-based (Greenshields), queue-based (congestion duration), and ML baselines (LSTM, Transformer).
- Train a PINN with physics constraints (conservation, queue balance, FD consistency) and compare against baselines.
- Run QC checks for input schema, unit consistency, and reproducibility.

2 Repository Layout

```
Dataset_v3/
 1. Input data/
  CA_I405_bottleneck_13.74_0403.xlsx
  CA_I405_bottleneck_13.74_0404.xlsx
  (other days .xlsx)

 2. Bottleneck heatmap/
  Speed.csv
  Speed profile.png
  Speed_Bottleneck Identification.py

 3. Different prediction models/
    1. BPR function_v3.py
    2. k_kc_v3.py
    3. Greenshields model_v3.py
    4. Queue based VDF_v6_mean.py
    4. Queue based VDF_v6_median.py
    5. LSTM.py
    6. Transformer.py
    7. PINN/
       7. PINN_v1.py
       7. PINN_v2.py
       1. observed/
          CA_I405_bottleneck_13.74_0403.xlsx
          CA_I405_bottleneck_13.74_0404.xlsx
          (other days .xlsx)
```

Notes:

- `1. Input data/` contains core daily Excel files with variables such as `Flow`, `Speed`, `Density`, `Queue`, `tt_obs_min`.
- `2. Bottleneck heatmap/Speed.csv` is a time × space speed matrix: first column `DateTime`, subsequent columns are milepost/detector IDs (e.g., `14.94`, `14.59`, ...).

- Scripts under `3. Different prediction models/` implement classical, theory-based, and AI/PINN models used for comparison.

3 Software Environment & Setup

3.1 Python Environment

Recommended: Python 3.10–3.11 with common scientific packages.

```
# Option A: conda
conda create -n vdf python=3.11 -y
conda activate vdf
pip install numpy pandas scipy matplotlib scikit-learn
pip install openpyxl xlsxwriter
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
# If you have a GPU, install the matching torch build for CUDA.

# Optional (deep learning baselines):
pip install tensorboard tqdm einops

# Optional (PINN helpers if used):
pip install pytorch-lightning
```

3.2 Folder Placement

Place this LaTeX guide at repo root (optional) and run scripts from the repository root so that relative paths in scripts resolve correctly.

4 Replication Map (Scripts → Outputs)

This section maps each script to its purpose, primary inputs, and expected outputs.

4.1 Heatmap and Bottleneck Identification

`Speed_Bottleneck Identification.py` Uses `Speed.csv` to produce a speed heatmap (time vs. milepost) and identify congestion windows (start, end).
Inputs: `Speed.csv`.
Outputs: `Speed profile.png`, and (optionally) derived CSVs with congestion intervals.

4.2 Classical/FD/Queue-Based Models

1. `BPR function_v3.py` Implements BPR calibration and evaluation against observed travel times.

2. `k_kc_v3.py` Capacity-ratio or density-critical modeling (e.g., parameters k, k_c).

3. `Greenshields model_v3.py` FD-based model linking $q(k)$, $v(k)$, and $t(k)$; often used to generate modeled speeds/travel times.

4. `Queue based VDF_v6_mean.py` Queue-based VDF using mean statistics (e.g., congestion duration, discharge rate).

4. `Queue based VDF_v6_median.py` Same as above using medians for robustness.
Inputs (for all above): daily Excel files under `1. Input data/`.
Outputs: model parameters, fitted curves, error metrics (RMSE/MAE), and plots.

4.3 AI Baselines

5. `LSTM.py` Sequence model for speed/flow/tt forecasting; typically trained per link or aggregated segment.

6. `Transformer.py` Attention-based baseline for sequence forecasting; compare vs. LSTM.
Inputs: curated sequences from daily Excel files or preprocessed tensors.
Outputs: predictions, training curves, error tables.

4.4 Physics-Informed Models

7. `PINN_v1.py`, `7. PINN_v2.py` Physics-informed NN with residual losses: conservation, queue balance, FD consistency.
Inputs: observed daily Excel files under `7. PINN/1. observed/`.
Outputs: PINN predictions, loss traces (data vs. physics residuals), comparative plots.

5 How to Run (Reproduction Steps)

Run from repo root (`Dataset_v3/`). Replace filenames as needed.

5.1 A. Heatmap

```
python "2. Bottleneck heatmap/Speed_Bottleneck Identification.py" \
  --input "2. Bottleneck heatmap/Speed.csv" \
  --outdir "2. Bottleneck heatmap/"
```

(If no CLI args are supported, open the script and set paths at the top.)

5.2 B. Classical / FD / Queue-Based

```
python "3. Different prediction models/1. BPR function_v3.py" \
  --data_dir "1. Input data/"

python "3. Different prediction models/3. Greenshields model_v3.py" \
  --data_dir "1. Input data/"

python "3. Different prediction models/4. Queue based VDF_v6_mean.py" \
  --data_dir "1. Input data/"
```

5.3 C. AI Baselines

```
python "3. Different prediction models/5. LSTM.py" \
  --data_dir "1. Input data/" --epochs 50 --batch_size 64

python "3. Different prediction models/6. Transformer.py" \
  --data_dir "1. Input data/" --epochs 50 --batch_size 64
```

5.4 D. PINN

```
python "3. Different prediction models/7. PINN/7. PINN_v1.py" \
  --obs_dir "3. Different prediction models/7. PINN/1. observed/" \
  --epochs 3000

python "3. Different prediction models/7. PINN/7. PINN_v2.py" \
  --obs_dir "3. Different prediction models/7. PINN/1. observed/" \
  --epochs 3000
```

Tip: If a script does not accept CLI arguments, edit the top-level constants to point to the correct folders.

6 Learning Guide & QC

6.1 1) Input Sanity Checks

C1. Schema. For daily Excel files, confirm columns exist: `DateTime`, `Flow`, `Speed`, `Density`, `Queue`, `tt_obs_min`. For `Speed.csv`, verify first column is `DateTime` and remaining columns are numeric station IDs.

C2. Units. Ensure consistent speed (mph or km/h) and density (veh/mi or veh/km). If needed, convert to a standard.

C3. Missing/zeroes. Inspect missingness patterns; document imputation or filtering choices.

6.2 2) Intermediate Checks (by model family)

M1. BPR/FD fits. Plot observed vs. fitted travel time (or speed). Confirm monotonicity near capacity and realistic asymptotes.

M2. Queue-based. Extract congestion duration and discharge rate; verify FIFO and cumulative arrivals–departures alignment.

M3. AI baselines. Hold-out at least one day for testing; verify no leakage. Report RMSE/MAE and compare to naive baselines.

M4. PINN. Inspect data-loss vs. physics-loss curves; ensure conservation residuals decrease. Compare predictions in oversaturated regimes against queue-based curves.

6.3 3) Output Consistency

O1. Replicability. Fix random seeds for ML runs. Save parameter JSONs and figures to a dated subfolder.

O2. Key KPIs. Report: RMSE/MAE for `tt_obs_min` (or speed), inferred D/C, congestion duration, discharge rate.

O3. Narrative check. Explain discrepancies (e.g., model underestimates during extended oversaturation).

7 Appendix A: Data Dictionary

The table below consolidates the key variables discovered in the daily Excel inputs and the speed heatmap CSV.

Variable	Description	Type	Unit / Example
`DateTime`	Timestamp of observation (local time)	datetime	e.g., 2017-04-03 07:15
`date_id`	Numeric date identifier	integer	e.g., 20170403
`Time` / `time_index` / `time`	Alternative time-of-day encodings used by scripts	mixed	e.g., 07:15, 435, 7.25
`Flow`	Total link flow per interval	float	veh / (aggregation interval)
`Flow per lane`	Lane-normalized flow	float	veh / interval / lane
`Flow per hour`	Flow rate normalized to per-hour	float	veh/h; e.g., 1800
`Speed`	Mean link speed	float	mph or km/h
`Density`	Mean density (from $q = k\,v$)	float	veh/mi or veh/km
`Queue`	Estimated queue length / state	float	veh or m
`tt_obs_min`	Observed travel time	float	minutes; e.g., 6.5
`Speed.csv: DateTime`	Time index for speed heatmap	datetime	
`Speed.csv: 14.94, 14.59, ...`	Column names denote detector/milepost IDs	float	speed at station; mph or km/h

Notes on Units. Ensure a single, consistent unit system prior to calibration (e.g., mph + veh/mi or km/h + veh/km).

8 Appendix B: Core Algorithm Pseudocode

8.1 B.1 BPR (flow-based) daily calibration + 80:20 OOS

Reference implementation available in the package.

```
Inputs:
  - Daily Excel files under 1. Input data/
  - Columns: Flow per hour, Speed (if no observed TT), tt_obs_min (optional)
Constants:
  - vf (free-flow speed), ca (capacity), L (segment length)
  - T_free_min = (L / vf) * 60

For each day file:
  1) Read df; tt_obs := tt_obs_min if present else (L / max(Speed, eps)) * 60
  2) q_vph := Flow per hour; x := clip(q_vph / ca, 0, +inf)
  3) Two-stage search for (alpha, beta) minimizing MAE(tt_obs, T_free_min * (1 + alpha *
      x^beta)):
     3.1) Coarse grid over alpha, beta; keep best (alpha0, beta0)
     3.2) Refined grid centered at (alpha0, beta0) -> (alpha*, beta*)
  4) tt_bpr := T_free_min * (1 + alpha* * x^(beta*))
```

```
  5) Record metrics: MAE, RMSE, MAPE, R2; save daily plot and time-series
After all days:
  6) Aggregate (alpha, beta) on training days (median or trimmed mean)
  7) OOS: apply aggregated params to last 20% days; export summary CSV/plots
```

8.2 B.2 BPR (density-based: k/k_c) daily calibration + OOS

Reference implementation available in the package.

```
Inputs:
  - Daily Excel files; columns: Density (or compute q/v), Speed, tt_obs_min (optional)
Constants:
  - k_c (critical density), L, vf
  - T_free_min = (L / vf) * 60
For each day:
  1) tt_obs := tt_obs_min if present else (L / max(Speed, eps)) * 60
  2) k := Density if present else (Flow per hour / max(Speed, eps))
  3) x := clip(k / k_c, 0, +inf)
  4) Two-stage search -> (alpha*, beta*) by MAE
  5) tt_hat := T_free_min * (1 + alpha* * x^(beta*)); compute metrics/plots
Aggregate over train days; evaluate on test days as in B.1
```

8.3 B.3 Greenshields FD calibration (robust) + OOS

Reference implementation available in the package.

```
Inputs:
  - Daily Excel; required columns: Speed, Flow per hour, Density (+ optional timestamp)
Pre-clean:
  - Keep finite (v,k), v > VMIN; trim k and v by (q_low, q_high) quantiles
Fit per-day FD:
  1) Solve least squares (robust loss) for v(k) = vf * (1 - k / k_jam)
  2) Predict v_all; map to TT_cal = clip((L / v_all) * 60, 0, TT_cap)
  3) Compare to TT_obs derived from Speed; compute MAE/RMSE/MAPE
OOS:
  4) Aggregate (vf, k_jam) on first 80% days (median or trimmed mean)
  5) Apply to last 20% days; export per-day and summary outputs
```

8.4 B.4 Queue-based VDF (γ from TT) with congestion window detection

Reference implementation available in the package.

```
Inputs:
  - Daily Excel; columns: Flow per hour, Queue, Speed (if no TT)
Constants:
  - STEP_MIN (e.g., 5), TT_CO_MIN, TT_FF_MIN
Prelim:
  1) tt_obs := tt_obs_min if present else (L / max(Speed, eps)) * 60
  2) Detect [t0, t3]:
     - Find first index with q > eps -> t0
     - Find last sustained exit run (q <= eps for >= MIN_RUN_EXIT) -> t3
  3) mu := median(Flow per hour[t0:t3]) over finite entries
```

```
Fit gamma (per day):
  4) On [t0, t3], regress (tt_obs - TT_CO_MIN) approx (gamma / (3*mu)) * (t - t0)^2 * (t3
       - t) * 60
     - Solve by closed-form alpha = (Z^T Y) / (Z^T Z), Z = ((t - t0)^2 (t3 - t)) * 60;
         gamma = 3 mu alpha
  5) Build tt_cal with fitted mu, gamma; compute MAE/RMSE/MAPE
80:20 protocol:
  6) Keep only valid days (successful fits)
  7) TRAIN_DAYS = floor(0.8 * n_valid); aggregate mu, gamma on train set (MEAN or MEDIAN)
  8) Test days: predict using aggregated mu, gamma; save per-day plots, CSV, and OOS
      summary
```

8.5 B.5 LSTM sequence model (day-wise 80:20)

Reference implementation available in the package.

```
Inputs:
  - Daily Excel -> target series y_t = tt_obs_min per timestamp
Protocol:
  1) Load all days; assert equal length per day
  2) Split by day: first 80% train, last 20% test
  3) Standardize using train mean/sd
  4) Build SeqDataset with sliding windows of length seq_len -> (X, y)
Model:
  - LSTM(input=1, hidden=H, layers=L) -> FC -> scalar
Train:
  - Optimize MSE over flattened train series windows
Test (per day in last 20%):
  - Context = [prev_day, current_day] standardized; slide windows -> preds
  - Keep last N points (current day); invert standardization
  - Compute MAE, RMSE, MAPE, R2; save plots and per-day time series
```

8.6 B.6 Transformer time-series model (day-wise 80:20)

Reference implementation available in the package.

```
Inputs & split:
  - Same as LSTM: daily tt_obs_min; 80% train, 20% test; standardize using train
Dataset:
  - Sliding windows (seq_len) -> (X_seq, y_next)
Model:
  - Linear input projection -> learned positional emb. -> TransformerEncoder(N layers, H
      heads) -> FC
Train:
  - MSE on train windows; log loss periodically
Test (per day):
  - Concatenate prev_day + current_day; form windows; predict all; keep last-day horizon
  - Invert standardization; compute MAE, RMSE, MAPE, R2; export plots/CSVs
```

8.7 B.7 PINN (physics-informed NN) for TT/State prediction

Concept consistent with the PINN folder.

```
Data:
  - Observations (tt_obs_min, speed, density) and/or boundary/initial conditions
Physics:
  - Residuals: conservation d*rho/dt + d q(rho)/dx = 0; queue balance; FD consistency q
      approx FD(rho)
Model:
  - NN_theta(x,t) -> predicted fields (rho, q, v, tt)
Loss:
  - L = w_data * MSE(pred, obs) + w_cons * ||conservation||^2
       + w_queue * ||queue_balance||^2 + w_FD * ||q - FD(rho)||^2
Train:
  - Sample collocation points; backprop to minimize L
Eval:
  - Compare predicted tt vs. tt_obs_min; inspect tradeoff between data and physics losses
```