Pre-processing Data Documentation

Project: UK Train Rides — Data Preparation (Power Query / Power BI)
Prepared for: UK Train Rides Analytics Project
Scope: Transform raw transaction-level data into a clean fact table and supporting dimension-ready fields for analysis, forecasting and mapping.

---

1. Raw data (source and sample)

Source file: transactional train ticket dataset (CSV/Excel) containing one row per ticket/transaction.
Primary columns (raw):

```
Transaction ID, Date of Purchase, Time of Purchase, Purchase Type, Payment Method, Railcard, Ticket Class, Ticket Type, Price, Departure Station, Arrival Destination, Date of Journey, Departure Time, Arrival Time, Actual Arrival Time, Journey Status, Reason for Delay, Refund Request
```

rows show:

- mixed purchase channels (`Online`, `Station`), payment methods (`Credit Card`, `Contactless`, `Debit Card`), railcard values including `None`, ticket classes and types, price values, station names, journey dates/times, actual arrival times, statuses (`On Time`, `Delayed`), reasons, and refund flags.

---

2. Initial data assessment — issues discovered

During inspection of the raw file we found:

1. **Separate date and time columns** for journey and purchase → need to be combined.
2. **Trips that cross midnight**: some scheduled/actual arrival times are earlier than departure times (e.g. departure 22:30 → arrival 00:30 next calendar day).
3. **Missing / blank values** in Reason for Delay; some Actual Arrival Time values present, some null.
4. **Mixed text formatting and stray spaces** in station names and categorical fields (upper/lower case differences).
5. **Yes/No fields** as text (`Refund Request`) need to be Boolean.
6. **No surrogate keys** — we must generate keys for Date, Ticket, Station, Status, Purchase to build a star schema.
7. **Potential duplicate and error rows** (deleted at early cleaning stage).

---

3. Objectives of preprocessing

- Produce correct DateTime stamps for scheduling and actual events (account for overnight trips).
- Compute durations, delay metrics and flags ready for DAX/analysis.
- Standardize textual fields and categorical values.
- Create compact keys (Ticket_Key, RouteKey) for joins and grouping.
- Prepare the fact table fields required by the model and downstream measures (Total Rides, Delay Rate, Forecast inputs).
- Keep provenance and reproducibility (document every step and formula).

---

4. Detailed step-by-step transformations (Power Query oriented)

All steps were performed inside Power Query. I show the business logic and (where useful) Power Query M style pseudocode or expressions.

4.1 Promote headers

- Action: `Use First Row as Headers` to ensure the first row contains proper column names.

4.2 Remove rows with errors and empty rows

- Action: `Remove Errors` then `Remove Blank Rows`.
- Rationale: get rid of corrupted or empty records that would break downstream calculations.

4.3 Trim and clean all text columns

- Action: For all text columns (station names, ticket fields, purchase type, payment method, reason, railcard) apply:
  - `Text.Trim` to remove leading/trailing spaces
  - `Text.Clean` to remove non-printable characters
  - `Text.Proper` (or equivalent) to capitalize each word consistently
- Example (M):

```
Table.TransformColumns(Source, {
  {"Departure Station", each Text.Proper(Text.Clean(Text.Trim(_))), type
text},
  {"Arrival Destination", each Text.Proper(Text.Clean(Text.Trim(_))), type
text},
  {"Ticket Class", each Text.Proper(Text.Trim(_)), type text}
})
```

- Rationale: ensures consistent joins and grouping.

4.4 Standardize boolean/text flags

- Convert `Refund Request` values (`Yes`/`No`) to boolean `true`/`false`:

```
if Text.Upper(Text.Trim([Refund Request])) = "YES" then true else false
```

4.5 Build scheduled datetime fields

- Merge `Date of Journey` + `Departure Time` → **JourneyDateTime_ScheduledDeparture**.
- Merge `Date of Journey` + `Arrival Time` → **JourneyDateTime_ScheduledArrival** (temporary).
- M pattern:

```
#datetime(Date.Year([Date of Journey]), Date.Month([Date of Journey]),
Date.Day([Date of Journey]),
         Time.Hour([Departure Time]), Time.Minute([Departure Time]), 0)
```

4.6 Adjust scheduled arrival for overnight trips (Final scheduled arrival)

- Business rule: if `Arrival Time` < `Departure Time` then scheduled arrival is the next calendar day.
- Implemented as:

```
if [Arrival Time] < [Departure Time]
then Date.AddDays(JourneyDateTime_ScheduledArrival, 1)
else JourneyDateTime_ScheduledArrival
```

- Final column name used: **JourneyDateTime_Arrival** (final scheduled arrival timestamp).

4.7 Build actual arrival datetime and adjust for overnight

- Merge `Date of Journey` (or `Arrival Date` where available) + `Actual Arrival Time` → **ActualArrivalDateTime**.
- Business rule for crossing midnight:

```
if [Actual Arrival Time] = null then null
else if [Actual Arrival Time] < [Departure Time]
     then Date.AddDays(ActualArrivalDateTime,1)
     else ActualArrivalDateTime
```

- Final column name used: **ActualArrivalDateTime** (or `null` if not available; preserves cancelled rows).

4.8 Remove intermediate time columns

- After final datetimes created, remove raw time columns to avoid confusion:
  - drop `Departure Time`, `Arrival Time`, `Actual Arrival Time`, etc.
- (Keep original raw file copy for provenance.)

4.9 Rename “Final” columns to final names

- If temporary columns used like `..._Final`, rename to descriptive names:
  - `JourneyDateTime_ScheduledDeparture`
  - `JourneyDateTime_Arrival`
  - `ActualArrivalDateTime`

4.10 Create PurchaseDateTime

- Merge `Date of Purchase` + `Time of Purchase` → **PurchaseDateTime**
- Drop original purchase date/time columns.

4.11 Add ActualArrivalExists flag

- Conditional column:

```
if [ActualArrivalDateTime] <> null then true else false
```

- Purpose: easy filter for "actual arrivals present".

4.12 Compute actual delay durations

- Create duration column: **ActualDelayDuration**:

```
[ActualArrivalDateTime] - [JourneyDateTime_Arrival]
```

- Then numeric minutes field: **ActualDelayInMinutes**

```
Duration.TotalMinutes([ActualDelayDuration])
```

- Note: rows with `ActualArrivalDateTime = null` remain null.

4.13 Clean and fill `Reason for Delay`

- New field `ReasonForDelay_Filled` with logic:
  1. If `Reason for Delay` is not null and not blank → keep value.
  2. Else if `Journey Status` <> `"Cancelled"` and `ActualDelayInMinutes` <= 0 → set `"No Delay"`.
  3. Else → null.
- M style:

```
if [Reason for Delay] <> null and Text.Trim([Reason for Delay]) <> "" then
[Reason for Delay]
else if [Journey Status] <> "Cancelled" and [ActualDelayInMinutes] <= 0
then "No Delay"
else null
```

- After verifying results, drop original `Reason for Delay`, and rename new field to `Reason for Delay`.

4.14 Compute scheduled and actual durations (minutes)

- **ScheduledDurationInMinutes**:

```
Duration.TotalMinutes([JourneyDateTime_Arrival] -
[JourneyDateTime_ScheduledDeparture])
```

- **ActualDurationInMinutes**:

```
Duration.TotalMinutes([ActualArrivalDateTime] -
[JourneyDateTime_ScheduledDeparture])
```

- Note: ActualDuration may be null if no ActualArrival.

4.15 Create `WasDelayed` and `IsCancelled` flags

- `IsCancelled`:

```
if Text.Upper([Journey Status]) = "CANCELLED" then true else false
```

- `WasDelayed`: logic

```
if Text.Upper([Journey Status]) = "CANCELLED" then false
else if [ActualDelayInMinutes] > 0 then true
else false
```

- Convert both to boolean.

4.16 Create Date and Time dimension fields (from scheduled departure)

- **Date** = `DateTime.Date([JourneyDateTime_ScheduledDeparture])`
- Insert Year, MonthName, WeekOfYear, DayName, DayOfWeek number, Hour.
- **IsWeekend**:

```
if [DayOfWeek] = 6 or [DayOfWeek] = 7 then true else false
```

(Use whatever DayOfWeek numbering convention your Date dimension uses; earlier you used 0/1 in prior logic — adjust to match.)

4.17 Create Ticket_Key (composite key)

- Combine `Railcard`, `Ticket Class`, `Ticket Type`:

```
[Ticket_Key] = Text.Combine({Text.From([Railcard]), Text.From([Ticket
Class]), Text.From([Ticket Type])}, "-")
```

- Purpose: later map to `Dim_Tickets` and replace text with surrogate `TicketKey`.

4.18 Create RouteKey

- Combine `Departure Station` and `Arrival Destination`:

```
[RouteKey] = [Departure Station] & " → " & [Arrival Destination]
```

- Purpose: route-level analysis and (optionally) `Dim_Routes`.

4.19 FareBucket creation

- Bucket prices into categories A..D based on price ranges (example mapping — adjust to business rules):

```
if [Price] <= 10 then "A" else if [Price] <= 30 then "B" else if [Price] <= 60 then "C" else "D"
```

- Create `FareBucket`.

4.20 LeadTimeDays

- Days between purchase and scheduled departure:

```
Duration.Days([JourneyDateTime_ScheduledDeparture] - [PurchaseDateTime])
```

4.21 Column reorder and cleanup

- Group columns into logical order:
  - identifiers → datetimes → durations/flags → ticket attributes → price/fare → route/station metadata → purchase info → original fields (kept in provenance copy).
- Remove intermediate helper columns.

4.22 Duplicate table and create Fact_Railway

- Duplicate cleaned query, then further tune it into `Fact_Railway` (the final fact table).
- Keep a master copy of the cleaned master table for traceability.

---

5. Resulting final columns (Fact_Railway expected schema)

After all preprocessing the fact table contains (representative list):

- `TransactionID`
- `PurchaseDateTime`
- `PurchaseType`
- `PaymentMethod`
- `Railcard`
- `TicketClass`
- `TicketType`
- `Price`
- `Ticket_Key`
- `RouteKey`
- `DepartureStation` / `DepartureStationKey` (after merging with Dim_Stations)

- `ArrivalDestination` / `ArrivalStationKey`
- `JourneyDateTime_ScheduledDeparture`
- `JourneyDateTime_Arrival` (scheduled final arrival)
- `ActualArrivalDateTime`
- `ActualArrivalExists` (boolean)
- `ActualDelayDuration` (Duration)
- `ActualDelayInMinutes` (numeric)
- `ScheduledDurationInMinutes`
- `ActualDurationInMinutes`
- `WasDelayed` (boolean)
- `IsCancelled` (boolean)
- `Reason for Delay` (cleaned)
- `RefundRequest` (boolean)
- `FareBucket`
- `LeadTimeDays`
- `Date` (derived from scheduled departure)
- `Year`, `MonthName`, `WeekOfYear`, `DayName`, `DayOfWeek`, `IsWeekend`
- `RideCount` (if aggregated or 1 per transaction)

---

6. Downstream model mapping and joins (how these columns were used)

- `Date` → `Dim_Date` (DateKey) for time intelligence.
- `Ticket_Key` → `Dim_Tickets[TicketKey]` for ticket attributes.
- `DepartureStation`/`ArrivalDestination` → `Dim_Stations[StationName]` (lookup to `StationKey`) — produce `DepartureStationKey` and `ArrivalStationKey` in fact table.
- `RouteKey` optionally → `Dim_Routes` if route dimension is created.
- `PurchaseDateTime` / aggregated purchase attributes → `Dim_Purchase` as needed.
- `IsCancelled`, `WasDelayed`, `ActualDelayInMinutes`, `Reason for Delay` → used directly for measures (DelayRate, Avg Delay, Cancellation Rate).
- `Price`, `RefundRequest`, `FareBucket` → revenue measures and segmentation.

---

7. Validation & QA checks performed

For each preprocessing step, these checks were executed:

1. **Row counts**: compare raw row count to cleaned row count (after Remove Errors/Blank) to detect unexpected losses.
2. **Datetime sanity**: check that `JourneyDateTime_Arrival - JourneyDateTime_ScheduledDeparture` yields positive scheduled durations, and `ActualArrivalDateTime` is >= scheduled departure (or null for cancelled).
3. **Overnight corrections**: validate sample rows where arrival time < departure time — confirm +1 day was applied.

4. **Reason for Delay logic**: sample check — rows with no reason & no delay → `No Delay`. Rows with delay >0 but blank reason remain null.
5. **Flag consistency**: `WasDelayed` true only when `ActualDelayInMinutes` > 0 and `IsCancelled` false. `IsCancelled` true only when Journey Status equals "Cancelled" (case-insensitive).
6. **Ticket_Key uniqueness**: ensure that Ticket_Key aggregated groups map correctly to distinct ticket categories.
7. **Price and revenue verification**: sum(Price) at fact level compared to raw source totals for sanity.
8. **Nulls**: check that required keys (Date, TicketClass, DepartureStation, ArrivalStation) have no nulls after cleaning; if nulls found, resolve via business rules or remove rows.

---

8. Reproducibility & versioning

- **Power Query steps**: each transformation created as an ordered query step (Applied Steps). Save and document the query names.
- **Backup**: keep an untouched raw file in repository (raw_data/). Create processed snapshot (processed_data/) before building model.
- **Versioning**: tag the process with a version (e.g., Preprocessing v1.0) and record date/time and author.

---

9. Notes, assumptions, and business rules

- **Overnight rule**: Any scheduled/actual arrival time that is strictly earlier than the departure time is assumed to belong to the next calendar day. This fits typical service patterns but must be validated for edge cases (multi-day journeys).
- **No Delay semantics**: If no reason provided and `ActualDelayInMinutes` ≤ 0 and journey not cancelled → we mark `"No Delay"`. This is a business convenience for reports; real reasons may be missing due to data capture issues.
- **Refund flag**: `Yes` → True (refund requested) — further business rules required to compute refund amounts.
- **Ticket_Key**: composite key created to uniquely identify combinations of railcard/class/type; later replaced by numeric surrogate in Dim_Tickets for performance.

---

10. Deliverables produced by preprocessing

- **Cleaned Fact table** (Fact_Railway) ready for modeling.
- **Derived columns**: scheduled & actual datetimes, durations, delay metrics, keys, flags, fare bucket, lead time.
- **Provenance data**: copies of raw input and intermediate snapshots.
- **List of power query steps** (documented in Applied Steps).

- **Validation report** summarizing checks and results.

---

11. Next recommended steps (after preprocessing)

1. Create `Dim_Date` covering full date range (extend into forecast horizon).
2. Build `Dim_Tickets`, `Dim_Stations`, `Dim_Status`, `Dim_Purchase` with numeric surrogate keys.
3. Merge surrogate keys back into Fact_Railway (Left Outer joins in Power Query) and remove text keys where appropriate.
4. Build Daily_Summary aggregation table (pre-aggregated daily metrics) for performance.
5. Implement DAX measures (Total Rides, Delay Rate, MOM growth, Forecast inputs).
6. Add `UK_Train_Stations_Coord_Saif` mapping to `Dim_Stations` (StationName) for map visuals.
7. Run a full model refresh and execute the validation queries/measures again to ensure consistency.

---

12. Appendix — Selected Power Query (M) expressions used

(Representative examples; adjust actual step names to your queries)

- **Trim & Proper**

```
= Table.TransformColumns(PrevStep, {
  {"Departure Station", each Text.Proper(Text.Clean(Text.Trim(_))), type
text},
  {"Arrival Destination", each Text.Proper(Text.Clean(Text.Trim(_))), type
text}
})
```

- **Create scheduled departure DateTime**

```
= Table.AddColumn(PrevStep, "JourneyDateTime_ScheduledDeparture", each
    #datetime(Date.Year([Date of Journey]), Date.Month([Date of Journey]),
Date.Day([Date of Journey]),
            Time.Hour([Departure Time]), Time.Minute([Departure Time]),
0), type datetime)
```

- **Adjust scheduled arrival for midnight crossing**

```
= Table.AddColumn(PrevStep, "JourneyDateTime_Arrival", each
    if [Arrival Time] < [Departure Time] then
Date.AddDays([JourneyDateTime_ScheduledArrival],1) else
[JourneyDateTime_ScheduledArrival], type datetime)
```

- **Actual delay minutes**

```
= Table.AddColumn(PrevStep,"ActualDelayInMinutes", each if
[ActualArrivalDateTime] = null then null else
Duration.TotalMinutes([ActualArrivalDateTime] - [JourneyDateTime_Arrival]),
type number)
```

- **Ticket key**

```
= Table.AddColumn(PrevStep, "Ticket_Key", each
Text.Combine({Text.From([Railcard]), Text.From([Ticket Class]),
Text.From([Ticket Type])}, "-"), type text)
```

Data Model Documentation

Railway Operations Analytics – Power BI Project

This document provides a full, professionally structured description of the final analytical data model used for railway journey analysis, forecasting, ticket demand, and operational performance evaluation.
It has been updated to include the **Stations Coordinates Table** and the **Daily Summary Table** added to the model.

---

1. Overview of the Data Model Architecture

The final model follows an optimized **Star Schema** consisting of:

Fact Table (1):

- Fact_Railway

Dimension Tables (6):

- Dim_Date
- Dim_Tickets
- Dim_Stations
- Dim_Purchase
- Dim_Status
- UK_Train_Stations_Coord_Saif ← **newly documented**

Supporting Aggregation Table (1):

- Daily_Summary ← **newly documented**

The model supports forecasting, delay analytics, geographic visualizations, revenue reporting, and customer behavior insights.

---

2. Fact Table

2.1 Fact_Railway

The core transactional table containing the cleaned and processed journey records.

Primary Key

- JourneyID

Foreign Keys

- DateKey
- StationDepartureKey
- StationArrivalKey
- TicketKey
- StatusKey
- PurchaseDateKey

Stored Measures

- Price
- RideCount
- ActualDelayInMinutes
- ScheduledDurationInMinutes
- ActualDurationInMinutes
- LeadTimeDays

Flags

- WasDelayed
- IsCancelled
- ActualArrivalExists

Date & Time Breakdown

- JourneyDateTime_ScheduledDeparture
- JourneyDateTime_Arrival
- ActualArrivalDateTime
- PurchaseDateTime

Calendar Columns

- Year, Month, Week of Year
- Day Name, Day of Week
- Hour
- IsWeekend

Purpose

This table drives:

- Ride forecasting
- Revenue forecasting
- Ticket class demand prediction
- Delay and cancellation insights
- Route and station performance analysis

---

3. Dimension Tables

3.1 Dim_Date

Primary Key

- DateKey

Attributes

- Full Date
- Year, Quarter, Month Name
- Week Number
- Day Name, Day of Week
- IsWeekend

Purpose

Used for time-series analysis, MoM/Yoy comparisons, forecasting alignment.

3.2 Dim_Tickets

Primary Key

- TicketKey

Attributes

- Ticket Class
- Ticket Type
- Railcard
- FareBucket (A–D)

Purpose

Supports revenue segmentation, demand forecasting, and price analysis.

3.3 Dim_Stations

Primary Key

- StationKey

Attributes

- Station Name
- Region (optional)

Purpose

Used for:

- Route-level analysis
- Performance per station
- Departure vs arrival patterns

3.4 Dim_Purchase

Primary Key

- PurchaseDateKey

Attributes

- Purchase Date
- Purchase Time
- Payment Method
- Purchase Channel

Purpose

Helps analyze ticket purchasing behaviors and lead-time patterns.

3.5 Dim_Status

Primary Key

- StatusKey

Attributes

- Journey Status
- Reason for Delay
- WasDelayed flag
- IsCancelled flag
- ActualArrivalExists flag

Purpose

Enables performance analysis, operational reporting, and delay-category segmentation.

3.6 UK_Train_Stations_Coord_Saif (NEW – Geographic Dimension)

This dimension enriches the model with geospatial data, enabling map-based reporting.

Primary Key

- Station Name (linked to Dim_Stations)

Attributes

- Latitude
- Longitude
- Station Name

Relationship

- Connected to **Dim_Stations[Station Name]** using a one-to-one relationship.

Purpose

Enables:

- Interactive map visuals
- Geographic clustering
- Route distance and regional performance analysis
- Heatmaps of passenger movement or delays

4. Supporting Table

4.1 Daily_Summary (NEW – Aggregated Daily Table)

This table stores preaggregated daily metrics that complement the detailed fact table.

Primary Key

- DateKey

Relationship

- Connected to **Fact_Railway** via DateKey (Date dimension)

Attributes

- Journey Date
- Average Price
- Day of Week Number
- IsWeekend Flag
- Ride Count (daily aggregated)
- Total Revenue

Purpose

Improves performance by:

- Accelerating daily-level visuals
- Supporting forecasting visuals
- Simplifying revenue and demand trend charts
- Enabling daily-level anomaly detection

Daily_Summary is used for dashboards where daily aggregation is sufficient without querying the full fact table.

---

5. Relationships Overview

Dim_Date → Fact_Railway

- 1-to-many (DateKey)

Dim_Tickets → Fact_Railway

- 1-to-many (TicketKey)

Dim_Stations → Fact_Railway

- 1-to-many (DepartureKey)
- 1-to-many (ArrivalKey)

Dim_Stations → UK_Train_Stations_Coord_Saif

- 1-to-1 based on Station Name

Dim_Purchase → Fact_Railway

- 1-to-many (PurchaseDateKey)

Dim_Status → Fact_Railway

- 1-to-many (StatusKey)

Dim_Date → Daily_Summary

- 1-to-many (DateKey)

---

6. Design Principles

✔ **Star Schema**

✔ **Surrogate Keys**

✔ **Optimized relationships**

✔ **Clean dimension attributes**

✔ **Forecasting-ready**

✔ **GIS-ready via the coordinates table**

✔ **Daily aggregated performance layer via Daily_Summary**

---

7. Analytical Capabilities Enabled

Forecasting

- Total Rides on the next month
- Revenue by Day on the next month
- Ticket class demand on the next month

Geospatial Analytics

- Delay clusters by location
- Traffic heatmaps
- Route performance on maps

Customer Behavior

- Ticket class trends
- Railcard influence
- Purchase patterns

Operational KPIs

- Delay root-cause analysis
- Station-level performance
- Route-level performance
- Cancellations, punctuality

Financial Insights

- Revenue by ticket class
- Fare bucket segmentation
- Daily, monthly, and seasonal patterns

DAX Measures Documentation

(For UK Railway Passenger Analytics Model)

This section documents all DAX measures created in the model, including their purpose, calculation logic, and usage context. Measures are grouped by analytical theme for clarity.

---

1 Ride Volume & Demand Measures

Total Rides

Purpose: Counts all completed train rides.
Logic: SUM of rides recorded in the fact table.

Actual Trips

Purpose: Counts trips that actually occurred (excludes cancelled journeys).
Usage: KPI cards, operations dashboards.

Total Rides Demand

Purpose: Represents total demand including:
– Completed rides
– Cancelled rides
– Or unmet demand (if modelled separately)

Forecasted Rides

Purpose: Predicts ride volume for future periods.
Usage: Forecasting visuals, MOM forecasting.

Total_Forecasted_Rides_May2024

Purpose: A fixed-time forecast measure created for May 2024.
Usage: Historical forecast accuracy, comparison with actuals.

Historical % of Total Rides

Purpose: Shows the contribution of a selected period to total historical rides.
Logic:

```
(Rides for selected period) / (Total historical rides)
```

---

2 Revenue & Financial Measures

Net Revenue

Purpose: Calculates revenue excluding refunds or discounts.
Logic: Gross revenue – Refunds.

Total Revenue (Gross)

Purpose: Sum of all revenue before deductions.

Total Refund Amount

Purpose: Total refunded money based on refund records or refund flags.

Lost_Revenue_From_Delays

Purpose: Estimates revenue lost due to delays (refund rules, compensation, etc.).

Net Revenue average per Railcard

Purpose: Average net revenue per railcard type.
Usage: Customer segmentation insights.

Net Revenue average per Railcard 2

Purpose: Variant of the previous measure (likely using alternative grouping logic or different card type field).

Avg Ticket Price

Purpose: Average price paid per ticket.
Logic: Total Revenue ÷ Total Tickets
Usage: Price sensitivity, fare bucket analysis.

Total Tickets

Purpose: Count of all tickets sold.

Total_Forecasted_Revenue_May2024

Purpose: Revenue forecast for May 2024.

---

3 Delay & Punctuality Measures

Total Delayed Rides

Purpose: Counts all rides where ActualDelayInMinutes > 0.

Total Cancelled Rides

Purpose: Count of all rides with status “Cancelled”.

Delay_Rate

Purpose: Percentage of rides delayed.
Formula:

```
Total Delayed Rides / Total Rides
```

Cancellation_Rate

Purpose: Percentage of cancelled rides.
Formula:
```
Total Cancelled Rides / Total Rides
```

Average Delay Time (Min)

Purpose: Mean of ActualDelayInMinutes.

Average Arrival Delay

Purpose: Average difference between scheduled and actual arrival times.

% of Delayed Trips

Purpose: Percent of trips experiencing delay.
Same logic as Delay_Rate (depending on your definition).

% of On-Time Rides

Purpose: Share of trips with zero delay.
Formula:
```
OnTime_Rides / Total Rides
```

OnTime_Rides

Purpose: Count of rides where ActualDelayInMinutes ≤ 0.

---

4 Trend & Growth Measures

Rides MOM Growth %

Purpose: Month-over-Month percentage change in rides.
Logic:
```
(Current Month Rides – Previous Month Rides) ÷ Previous Month Rides
```

Usage:
Forecasting visuals, KPIs, dashboards.