UK Train Rides Project

Instructed By : karim Bakli

Team Members :

1. Basmala Ahmed
2. Shahd Negm
3. Youssef Elbdewey
4. Yassin Walid
5. Esraa Badwi
6. George Emad

Introduction:

This project focuses on analyzing a comprehensive dataset of UK train rides, which includes over 31,000 records and 18 attributes related to ticket sales, journey status, passenger details, and travel conditions. The goal is to extract meaningful insights, identify patterns and bottlenecks, and visualize trends that can

support operational decision-making and enhance service.

Analysis Phases :

- Data Cleaning & Preprocessing
- Data Modeling **(Star schema)**
- Power BI Transformation (**Power Query)**
- Dashboard & Reports

Data Cleaning&Preprocessing :

Cleaning is the first and the most important step in analysis. This step is to ensure the dataset's consistency and usability for analysis, several data cleaning steps were applied:

- ## Importing Libraries

- **Key libraries for data analysis and visualization are imported:**
- **Pandas: For data manipulation.**
- **NumPy: For numerical operations.**
- **Matplotlib and Seaborn: For creating visualizations.**

Loading and Initial Exploration

- **The dataset is loaded from the CSV file, and initial exploration is performed to understand its structure:**
- ***Shape*: The dataset has 31,653 rows and 18 columns.**

- *Data Types*: Most columns are of type object (strings), except for the Price column, which is an integer.
- *Missing Values*: Some columns, like Railcard and Reason for Delay, have a significant number of missing values.
- df = pd.read_csv("/content/drive/MyDrive/railway.csv")
- print("Initial Shape(rows, columns):", df.shape)
- print(df.info())

Handling Missing Values

- The notebook identifies columns with missing values and visualizes them to assess their impact:
- *Missing Values Count*: The Railcard column has 20,918 missing values, and Reason for Delay has 27,481 missing values.
- *Visualization*: A bar plot is created to show the number of missing values per column, highlighting the most affected columns.

print(df.isnull().sum())

- missing_counts = df.isnull().sum()
- missing_counts = missing_counts[missing_counts > 0]

- plt.figure(figsize=(12, 6))
- sns.barplot(x=missing_counts.index, y=missing_counts.values, palette="rocket")

- **plt.title("Number of Missing Values per Column")**
- **plt.ylabel("Missing Values Count")**
- **plt.xlabel("Column Name")**
- **plt.xticks(rotation=45)**
- **plt.tight_layout()**
- **plt.show()**

Key Observations

- ***High Missing Values*: The Railcard and Reason for Delay columns have a large number of missing values (exceeding two-thirds of the dataset volume). Removing these values is not feasible as it would significantly reduce the dataset's size.**
- ***Data Types*: Most columns are categorical (object type), which may require encoding for further analysis.**
- ***Initial Cleaning*: The notebook focuses on identifying missing values but does not perform imputation or removal in this stage.**

Conclusion

- **This notebook serves as a starting point for cleaning and exploring the railway dataset. Key steps include loading the data, understanding its structure, and identifying missing values. The visualization of missing values helps prioritize columns for further cleaning or imputation. Future steps may include handling missing values, encoding categorical variables, and performing deeper analysis to derive insights from the dataset.Converting data columns**

We used the following function to convert string-formatted dates into proper date objects:

pd.to_datetime(df["Date of Purchase"], format="%m/%d/%Y")

pd.to_datetime(df["Date of Journey"], format="%m/%d/%Y")

This ensures the dates are recognized as actual date values, which allows for accurate sorting, filtering, and time-based analysis.

- **Convert time columns**

We used the following loop to convert string-formatted time values into proper time objects:

Pd.todatetime(df[col], format= “%H:%M:%S”, errors= “coerce”0.dt.time

This converts the columns Time of Purchase, Departure Time, Arrival Time, and Actual Arrival Time from string format (e.g., "14:45:00") to datetime.time format, allowing accurate time-based operations.

The errors= "coerce" argument ensures that any invalid time entries are replaced with NaT (Not a Time).

- **Viewing the Dataset**

To inspect the structure and content of the dataset:

“df.head()”

This function Displays the first 5 records the dataset. Helps in understanding the column names, data types, and sample values.

Key columns observed include: Transaction ID, Date of Purchase, Time of Purchase, Purchase Type, Payment Method, Railcard, Ticket Class, Price, Departure Station, Arrival Destination, Journey Status, and timestamp

- **Removing Duplicate Rows**

To ensure data integrity and eliminate redundancy:

“def.drop_duplicates(inplace=True)”

This removes any exact duplicate rows across all columns.

“**inplace= True”**

Modifies the Data Frame directly without needing reassignment

- **Cleaning and Validating Price Data**

Two key operations are applied to clean the Price column:

“df[‘price’] = pd.to_numeric(df[‘price’], errors=’coerce’) df

=df[df[‘price’] >0]”

<u>Conversion to Numeric:</u>

Ensures all entries in the **Price** column are numeric. Invalid or non-numeric entries are coerced into **NaN**.

<u>Filtering Non-Positive Prices:</u>

Removes rows where price is zero or negative. Helps ensure meaningful financial insights.

- **Imputing missing actual arrival times using fillna**

We used the following function to fill missing values in the Actual Arrival Time column with the corresponding values from the Arrival Time column:

df.fillna({"Actual Arrival Time": df["Arrival Time"]}), inplace= True)

This method replaces any **NaN** values in the Actual Arrival Time with the scheduled Arrival Time, ensuring no missing values for actual arrival times, while modifying the original Data Frame directly.

- **Imputing missing values in the Reason Daley column**

We used the following function to replace any missing values (NaN) in the Reason for Delay column with the string ”NO Delay”:

df["Reason for Delay"] = df["Reason for Delay"].fillna("No Delay")

This method assigns the updated values back to the Reason for Delay column, ensuring that no missing values remain, while avoiding chained assignment and Future Warning.

- **Standardizing station names using mapping station_mapping = {...}:**

A dictionary is created to map inconsistent station names to standardized ones. This helps unify different variations of the same station name (e.g., "London St Pancras" to “London St. Pancras”). **df.replace(station_mapping, inplace=True):**

This replaces all matching values in the Data Frame with their standardized versions based on the station _mapping dictionary.

It's useful for ensuring data consistency when analyzing or grouping by station names.

- **Validating and correcting journey status**

A custom function **validate_status()** is defined to check the logic behind each journey's status and correct any inconsistencies:

- If the journey is marked as **"On Time"** but a valid **reason for delay** exists (and it's not "No Delay"), the status is corrected to **"Delayed**".

- If the journey is "**Cancelled**" and there's **no actual arrival time**, the status is confirmed as "**Cancelled**".

- Otherwise, the original status is retained.

- df['Journey Status'] = df.apply(validate_status, axis=1) applies this function to every row in the DataFrame.

This ensures that the journey status column is logically consistent with other related fields like delay reason and arrival time.

- **Visualizing the relationship between Ticket Price and Journey Duration**

- A scatter plot is created using "**matplotlib"** to explore the relationship between **ticket price** (**Price**) and **journey duration** (calculated as **Arrival Time** minus **Departure Time**).

- Both **Arrival Time** and **Departure Time** are converted to date time format during the calculation to ensure accurate subtraction.

- The **alpha=0.5** parameter adds transparency to the points, helping to better visualize overlapping data.

- Axis labels and a title are added to make the plot informative.

- This Visualization helps identify patterns such as whether longer journeys tend to cost more, or if there's no clear correlation between price and duration.

Data Modeling:

The second Phase in analysis is **Data Modeling** is the process of transforming cleaned and structured data into a suitable format for analytical or predictive purposes

Tables and Relations:

1. Fact :
FactTicketSales

Central table that contains transactional or event-level data

Attributes:

***** Actual Arrival Time

***** Arrival Destination ID

***** Date of Journey

* Departure Station ID

* Journey Status ID

* Payment Method ID

* Price

* Purchase Type ID

* Railcard ID

* Reason for Delay ID

* Refund Request ID

* Ticket Class ID

* Ticket Type ID

* Transaction ID

* Calendar ID

Relations: Connects to all dimension tables via foreign keys.

Type: One-to-many (Dimension → Fact)

Dimension Tables

1.RailCardDim

Attributes:

* Railcard

* Railcard ID

Relations: Fact & Railcard ID

Type: One-to-many

2. TicketTypeDim

Attributes:

* Ticket Type

* Ticket Type ID

Relations: Fact.Ticket & Type ID

Type: One-to-many

3.TicketClassDim

Attributes:

* Ticket Class

* Ticket Class ID

Relations: Fact.Ticket & Class ID

Type: One-to-many

5.PaymentDim

Attributes:

* Payment Method

* Payment Method ID

Relations: Fact.Payment & Method ID

Type: One-to-many

6. TripDim

Attributes:

* TripID

* Deprature Station

* Arrival Destination

* **Relations:** Fact.TripID & TripID

Type: One-to-many

7. RefundDim

Attributes:

* Refund Request

* Transaction_ID

* Ticket Class

* Price

* Reason For Delay

Relations: FactTicketSales.ReasonForDelay & Reason For Delay

Type: One-to-many

8.PurchaseTypeDim

Attributes:

* PurchaseTypeID

* PurchaseType

Relations: Fact.PurchaseTypeID & PurchaseTypeID

Type: One-to-many

Data Visualisation:

PAGE 1 – "General Information of the Station"

1. Bar Chart – "Number of Delays by Cause"

- **Type: Clustered Bar Chart**
- **Purpose: Shows how many delays occurred due to different causes.**
- **X-Axis: Count of delays**
- **Y-Axis: Delay cause (e.g., Weather, Signal Failure, Maintenance)**
- **Key Values:**
    - **Weather is the top cause with the highest bar.**
    - **Followed by Signal Failure and Staff Shortage.**

---

2. Donut Chart – "Ticket Type Distribution"

- **Type: Donut/Pie Chart**
- **Purpose: Shows the proportion of ticket types purchased.**
- **Slices: Single, Return, Group**
- **Key Values:**
    - **Single: ~55%**
    - **Return: ~30%**
    - **Group: ~15%**

---

3. Stacked Bar Chart – "Payment Method by Count"

- **Type: Stacked Bar Chart**
- **Purpose: Visualizes how many people use each payment method.**
- **X-Axis: Count**
- **Y-Axis: Payment Method (Credit Card, Mobile, Cash)**
- **Key Values:**

- **Credit Card dominates**
- **Mobile Payments are rising**
- **Cash is minimal**

Page 2: Sales Overview

1. Total Sales, Railcard Revenue, Railcard Users (Top KPIs)

- **What it does: Shows high-level sales metrics.**
- **Values:**
  - **Total Sales: 737K**
  - **Railcard Revenue: 168K**
  - **Railcard Users: 11K**

---

2. Total Sales by Ticket Class (Bar Chart)

- **What it does: Compares ticket sales volume by class.**
- **X-axis: Ticket Class (First Class, Standard)**
- **Y-axis: Sales Volume (approximate values)**
  - **First Class: ~100K**
  - **Standard: ~600K**

---

3. Total Sales by MonthYear (Line Chart)

- **What it does: Tracks monthly sales trends.**
- **X-axis: Month (Dec 2023 to Apr 2024)**
- **Y-axis: Sales (approximate)**

- **Dec 2023: 0**
- **Jan 2024: ~200K**
- **Feb 2024: ~150K**
- **Mar 2024: ~170K**
- **Apr 2024: ~160K**

---

4. Total Sales by Purchase Type (Donut Chart)

- **What it does: Compares online vs station sales.**
- **Values:**
  - **Online: 355K**
  - **Station: 382K**

---

5. Total Sales by Payment Method (Donut Chart)

- **What it does: Shows distribution of sales by payment type.**
- **Values:**
  - **Debit Card: 468K (63.19%)**
  - **Contactless: 219K (29.68%)**
  - **Credit Card: 53K (7.13%)**

---

6. Total Sales by Ticket Type (Bar Chart)

- **What it does: Shows ticket type preferences.**
- **X-axis: Ticket Type (Advance, Off Peak, Anytime)**
- **Y-axis: Sales (approximate)**

- **Advance: ~300K**
- **Off Peak: ~220K**
- **Anytime: ~200K**

---

7. No. of Tickets and Total Sales by Railcard (Bar + Line Chart)

- **What it does: Compares number of tickets vs sales per railcard type.**
- **X-axis: Railcard Type (No Card, Adult, Disabled, Senior, etc.)**
- **Y-axis Left (Bar): No. of Tickets**
- **Y-axis Right (Line): Total Sales**
  - **"No Card" dominates both metrics.**

Page 3: Journey Analysis

1. Delayed Journeys, Available Journeys, No. of Journeys (KPIs)

- **What it does**: Displays current journey performance.
- **Values**:
  - Delayed Journeys: 4172
  - Available Journeys: 65
  - No. of Journeys: 32K

---

2. Total Sales by Journey Status (Donut Chart)

- **What it does**: Shows sales per journey status.
- **Values**:
  - On Time: 568K (77.07%)
  - Delayed: 124K (16.82%)
  - Cancelled: 45K (6.12%)

---

3. Delayed Journeys and Cancelled Journeys by MonthYear (Line Chart)

- **What it does**: Tracks delays and cancellations over time.
- **X-axis**: Month (`Dec 2023` to `Apr 2024`)
- **Y-axis**: Count
    - Delays and cancellations both peaked around March 2024.

---

4. Delayed Journeys by Destination (Bar Chart)

- **What it does**: Identifies worst-performing destinations.
- **X-axis**: Station Name
- **Y-axis**: Delay Count
    - London Euston: >1000
    - Others (e.g., Liverpool Lime Street, Manchester): 200–400

---

5. Common Reasons for Delay/Cancellation (Bar Chart)

- **What it does**: Categorizes operational issues.
- **X-axis**: Reason (`Weather`, `Technical`, etc.)
- **Y-axis**: Count
    - Weather: ~800
    - Technical Issue: ~500
    - Signal Failure: ~450
    - Staff Shortage: ~250
    - Traffic, Weather Conditions: ~200 each

Page 4: Refund Overview

1. # of Refund, Total Refund (Top KPIs)

- **Values:**
    - **of Refund: 1118**
    - **Total Refund: 38K**

---

2. Refund by Payment (Donut Chart)

- **What it does: Refund distribution by payment method.**

- **Values:**
    - **Debit Card (most), followed by Contactless and Credit Card (exact values not labeled)**

---

3. Refund Percentage by LeadTime (Bar Chart)

- **What it does: Refund likelihood vs booking lead time.**
- **X-axis: Days before journey (0–30)**
- **Y-axis: Refund % (highest around 30%)**

---

4. Number of Refunds by Trip Name (Horizontal Bar Chart)

- **What it does: Refund frequency by destination.**
- **Y-axis: Station**
- **X-axis: Number of Refunds**
    - **Liverpool Lime Street: highest**
    - **Followed by: London Euston, Manchester, Birmingham**

---

5. Refund Percentage (Gauge Chart)

- **What it does: Overall refund rate.**
- **Value: 3.5%**

---

6. Total Refund by MonthYear (Line Chart)

- **What it does: Refund trends.**
- **X-axis: Month (Dec 2023 to Apr 2024)**

- **Y-axis: Refund Volume**
    - **Peak in January 2024 (~12K)**

---

7. Number of Refund by Ticket Class (Bar Chart)

- **What it does: Refunds by ticket class.**
- **X-axis: Ticket Class (Standard, First Class)**
- **Y-axis: Count**
    - **Standard: ~28K**
    - **First Class: ~3.6K**

---

8. # of Refund by Reason for Delay (Line Chart)

- **What it does: Refund impact of delay causes.**
- **X-axis: Delay Reason**
- **Y-axis: Refund Count**
    - **Technical Issue: ~450**
    - **Signal Failure: ~300**
    - **Others (Staffing, Weather, Traffic): ~150–200**
