UK Train Rides Project

Supervised By: eng/ karim Bakli

Team Work:

- Retaj Mohammed(Team Leader)
- Mostafa Hesham
- Mohammed Ehab
- Malak Hany
- Noran Sherif
- Nermin Ahmed

Introduction:

This project focuses on analyzing and managing data related to train rides across the United Kingdom, a country with one of the busiest and most complex rail networks in Europe. The UK rail system serves millions of passengers daily, connecting major cities, suburban areas, and rural regions through a network of operators, stations, and routes. Efficient handling of this data is crucial for optimizing scheduling, monitoring performance, improving passenger experience, and supporting infrastructure planning. The project aims to provide a technical foundation for processing, visualizing, and deriving insights from UK train ride datasets, supporting both operational decision-making and long-term strategy development.

We make our analysis from three steps:

- Data Cleaning
- Data Modeling
- Dashboard Design

Data Cleaning:

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

- **Importing Libraries**

We begin the analysis by importing the necessary Python library:

"Import pandas as pd"

'pandas' is used for data manipulation and analysis, particularly effective with tabular data.

df = pd.read_csv('C:\\Users\\dell\\data analysis\\railway.csv')

This loads the CSV file into a DataFrame named df.

- **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 step in our analysis is **Data Modeling** is the process of transforming cleaned and structured data into a suitable format for analytical or predictive purposes. It involves selecting appropriate algorithms, defining input features, and training models that can identify patterns or make predictions based on historical data.

Tables and Relations:

1. Fact

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
* Calendar ID

Relations: Connects to all dimension tables via foreign keys.

Type: One-to-many (Dimension → Fact)

Dimension Tables

2. Rail_card

Attributes:

***** Railcard

* Railcard ID

Relations: Fact & Railcard ID

Type: One-to-many

3. Ticket Type

Attributes:

* Ticket Type

* Ticket Type ID

Relations: Fact.Ticket & Type ID

Type: One-to-many

4.Ticket Class

Attributes:

* Ticket Class

* Ticket Class ID

Relations: Fact.Ticket & Class ID

Type: One-to-many

5.Payment Method

Attributes:

* Payment Method

* Payment Method ID

Relations: Fact.Payment & Method ID

Type: One-to-many

6.Journey Status

Attributes:

* Journey Status

* Journey Status ID

Relations: Fact.Journey & Status ID

> Type: One-to-many

8. Departure Station

Attributes:

* Departure Station

* Departure Station ID

Relations: Fact.Departure & Station ID

Type: One-to-many

9. Arrival Destination

Attributes:

* Arrival Destination

* Arrival Destination ID

Relations: Fact.Arrival & Destination ID

Type: One-to-many

10.Reason for Delay

Attributes:

* Reason for Delay

* Reason for Delay ID

Relations: Fact & Reason for Delay ID

Type: One-to-many

11.Refund Request

Attributes:

* Refund Request

* Refund Request ID

Relations: Fact.Refund & Request ID

Type: One-to-many

12.CalendarTable

Attributes:

* Date

* Day

* Day of Week

* Day of Week Number

* Month Name

Relations: Fact.Date of Journey

Type: One-to-many

Data Visualisation:

Page 1: Home – Navigation Hub

Description:
This is the landing page of the dashboard. It features a visually engaging background image of a train with the project title *"UK Train Station"*. It serves as a navigational hub, using clickable text boxes that link to the following four pages:

- **Overview:** Summary of train activity and KPIs.
- **Revenue:** Analysis of income generated by ticket sales.
- **Passenger Behavior:** Insights into travel habits and delays.
- **Summary:** A detailed table combining all relevant trip data.

Each section is briefly described to help users know what to expect in the linked dashboards.

Page 2: Overview – Train Ride Analysis

Description:
This dashboard provides a high-level summary of train ride data across the UK. It focuses on total rides, destinations, ticket classes, and Railcard usage to offer a quick yet comprehensive overview of key performance metrics.

Main Visual Elements:

- **KPI Cards:**
  - *Total Rides (32K)*
  - *Destinations (32)*
  - *Rides with Railcard (11K)*
- **Pie Chart – Total Rides by Ticket Class**
  Shows the percentage split between Standard (90.34%) and First Class (9.66%).
- **Matrix Table – Monthly Departures by Station**
  Displays ride volume per station per month with color shading for intensity.
- **Map – Total Rides by Destination**
  Visualizes geographic spread of arrival destinations.
- **Bar Chart – Arrival Destinations**
  Highlights most popular destinations, with Birmingham having the highest volume (~7.7K).
- **Tree Map – Railcard Usage by Passenger Type**
  Shows categories like Adult, Senior, and Disabled.
- **Slicer – Month Filter**
  Allows users to explore data for specific months (January to April).

Purpose:
To deliver a clear and interactive summary of train ride trends, popular destinations, and Railcard usage.

Page 3: Revenue – Ticket Sales and Income Analysis

Description:
This page provides a comprehensive view of revenue performance across different categories and timeframes.

Main Visual Elements:

- **KPI Cards:**
  - *Total Revenue*
  - *Revenue Per Week*
  - *Average Daily Revenue*
- **Area Chart – Revenue by Arrival Destination**
  Shows total income per city or station.
- **Line Chart – Revenue by Day of Week**
  Identifies daily revenue trends and peak days.
- **Donut Chart – Revenue by Payment Method**
  Displays distribution of revenue by cash, card, or online payment.
- **Clustered Column Chart – Revenue by Ticket Type**
  Compares revenue from ticket categories like Return, Advance, etc.
- **Gauge Charts:**
  - *Max Revenue*
  - *Min Revenue*
- **Slicer – Ticket Class (Standard / First Class)**

Purpose:
To assess financial performance, optimize ticket pricing strategies, and track high-performing routes and ticket types.

Page 4: Passenger Behavior – Travel Patterns and Delay Insights

Description:
This page analyzes passenger behavior, payment habits, and delay causes.

Main Visual Elements:

- **Clustered Bar Chart – Passenger Distribution**
  Analyzes how different types of tickets or classes are used.

- **Donut Chart – Revenue by Payment Method**
  Shows which payment channels are preferred.
- **Gauge Chart – Total Journeys**
  Displays the overall count of rides.
- **Tree Map – Delay Reasons**
  Categorizes the causes behind train delays.
- **Matrix – Total Rides by Day of Week and Hour**
  Visual heatmap showing peak travel times.
- **Slicers:**
  - *Ticket Type:* Advance, Anytime, Off-Peak
  - *Ticket Class:* Standard, First Class

Purpose:
To uncover behavior trends, inform service planning, and highlight operational weaknesses related to delays.

Page 5: Summary – Detailed Data Overview

Description:
The Summary page is a structured data table that consolidates the key details of all train journeys in the dataset.

Main Visual Elements:

- **Data Table Columns:**
  - Departure Station
  - Arrival Destination
  - Payment Method
  - Journey Status
  - Purchase Type
  - Railcard Type
  - Reason for Delay
  - Refund Request
  - Ticket Class
  - Ticket Type
- **Filters:**
  - *Month Slicer* – filters data by month
  - *Weekday Slicer* – filters data by day of the week
  - *Text Filter* – enables keyword search

Purpose:
To provide a raw, detailed view of all data records for in-depth analysis or export. Ideal for reviewers and data analysts.

Page 6: Tooltip – Summary Cards (Total Rides & Total Revenue)

- **Linked To:**
  - **Overview Page:**

- *Matrix Chart (***Monthly Departures by Station**
  Displays ride volume per station per month with color shading for intensity)

    - *Tree Map(***– Railcard Usage by Passenger Type)**
    - *Pie Chart(***Total Rides by Ticket Class)**
  - **Revenue Page:**
    - *Line Chart(***Revenue by Day of Week)**
- **Visuals:**
  - Card for **Total Rides**
  - Card for **Total Revenue**

Function:
Gives users a quick glance at core metrics while hovering over specific visuals.

Page 7: Tooltip – Revenue by Arrival Destination

- **Linked To:**
  - **Revenue Page:** *Area Chart (***Revenue by Arrival Destination)**
- **Visual:**
  - Clustered Column Chart showing **Total Revenue by Arrival Destination**

Function:
Adds clarity to which destinations generate the most revenue during user interaction.

Page 8: Tooltip – Total Rides by Month

- **Linked To:**
  - **Passenger Behavior Page:** *Matrix Chart (***Total Rides by Day of Week and Hour)**
- **Visual:**
  - Clustered Column Chart showing **Total Rides by Month**

Function:
Helps users detect seasonal patterns in ride volume directly from the heatmap.

Page 9: Tooltip – Delayed Rides

- **Linked To:**
  - **Passenger Behavior Page:** *Tree Map (***Delay Reasons)**
- **Visual:**
  - Clustered Column Chart showing **Delayed Rides**

Function:
Provides an immediate breakdown of delayed trips when hovering over delay reasons.