Delhivery

❖ Topic: Feature Engineering
❖ Duration: 1 week

Introduction:

Delhivery, India's leading and rapidly growing integrated player, has set its sights on creating the commerce operating system.
They achieve this by utilizing world-class infrastructure, ensuring the highest quality in logistics operations, and harnessing cutting-edge engineering and technology capabilities.

Why this case study?

From Delhivery's Perspective:

- Delhivery aims to establish itself as the premier player in the logistics industry. This case study is of paramount importance as it aligns with the company's core objectives and operational excellence.
- It provides a practical framework for understanding and processing data, which is integral to their operations. By leveraging data engineering pipelines and data analysis techniques, Delhivery can achieve several critical goals.
- First, it allows them to ensure data integrity and quality by addressing missing values and structuring the dataset appropriately.
- Second, it enables the extraction of valuable features from raw data, which can be utilized for building accurate forecasting models.
- Moreover, it facilitates the identification of patterns, insights, and actionable recommendations crucial for optimizing their logistics operations.
- By conducting hypothesis testing and outlier detection, Delhivery can refine their processes and further enhance the quality of service they provide.

From Learners' Perspective:

- Learners will gain hands-on experience in data preprocessing and cleaning, which is often the most time-consuming aspect of data analysis.

- Feature engineering is a critical step in building machine learning models. In this case study, learners will understand how to extract meaningful features from raw data, including datetime manipulation and column splitting.
- The case study introduces learners to the concept of grouping data based on specific keys and then aggregating it. This is a key aspect of data analysis, especially when dealing with time-series data or data with a hierarchical structure.
- Learners will perform hypothesis testing, to validate assumptions and draw insights from data.
- The case study goes beyond data analysis by focusing on deriving actionable insights for a business. Learners will understand how data analysis can drive informed decision-making and recommendations.

Dataset:

https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv?1642751181

Instructions to download the dataset:

1. Click on the dataset **link provided above**.
2. When the webpage opens, **Right Click** and **Save As**.
3. Save the file onto your local system with a **.csv** extension.
4. Now you can load the data file on a **Jupyter Notebook** or **Google Colab**.

Column Profiling:

1. **data** - tells whether the data is testing or training data
2. **trip_creation_time** – Timestamp of trip creation
3. **route_schedule_uuid** – Unique ID for a particular route schedule
4. **route_type** – Transportation type
    a. **FTL** – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way
    b. **Carting**: Handling system consisting of small vehicles (carts)
5. **trip_uuid** - Unique ID given to a particular trip (A trip may include different source and destination centers)
6. **source_center** - Source ID of trip origin
7. **source_name** - Source Name of trip origin
8. **destination_cente** – Destination ID
9. **destination_name** – Destination Name
10. **od_start_time** – Trip start time

11. **od_end_time** – Trip end time
12. **start_scan_to_end_scan** – Time taken to deliver from source to destination
13. **is_cutoff** – Unknown field
14. **cutoff_factor** – Unknown field
15. **cutoff_timestamp** – Unknown field
16. **actual_distance_to_destination** – Distance in kms between source and destination warehouse
17. **actual_time** – Actual time taken to complete the delivery **(Cumulative)**
18. **osrm_time** – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time **(Cumulative)**
19. **osrm_distance** – An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) **(Cumulative)**
20. **factor** – Unknown field
21. **segment_actual_time** – This is a segment time. Time taken by the subset of the package delivery
22. **segment_osrm_time** – This is the OSRM segment time. Time taken by the subset of the package delivery
23. **segment_osrm_distance** – This is the OSRM distance. Distance covered by subset of the package delivery
24. **segment_factor** – Unknown field

How to get started?

To complete the case study, begin by downloading the CSV files from the provided link. Afterward, proceed to upload them onto Google Colab / Jupyter Notebook for further analysis.

If you are using Google Colab, you can directly start working on the notebook on Colab.

Install Anaconda using the link. Once Anaconda has been installed on your system, open Jupyter Notebook. Refer link.

Now, the CSV file needs to be uploaded/imported in the Colab/Jupyter notebook respectively.

Refer to this link for ways to upload a CSV file into Colab.

Once the file have been successfully uploaded/imported, you can conveniently access them within the notebook using the read_csv( ) method.

What is expected?

The company wants to understand and process the data coming out of data engineering pipelines:

- Clean, sanitize and manipulate data to get useful features out of raw fields
- Make sense out of the raw data and help the data science team to build forecasting models on it.

Submission Process:

- Type your insights and recommendations in the text editor.
- Convert your jupyter notebook into PDF (Save as PDF using Chrome browser's Print command), upload it in your Google Drive (set the permission to allow public access), and paste that link in the text editor.
- Optionally, you may add images/graphs in the text editor by taking screenshots or saving matplotlib graphs using plt.savefig(...).
- After submitting, you will not be allowed to edit your submission.

General Guidelines:

- Evaluation will be kept lenient, so make sure you attempt this case study.
- It is understandable that you might struggle with getting started on this. Just brainstorm, discuss with peers, or get help from TAs.
- There is no right or wrong answer. We have to get used to dealing with uncertainty in business. This is exactly the skill we want to develop.

What does 'good' look like?

1. **Basic data cleaning and exploration:**
    1. Handle missing values in the data.
    2. Converting time columns into pandas datetime.

3. Analyze structure & characteristics of the dataset.

2. Try merging the rows using the hint mentioned below.

Since delivery details of one package is divided into several rows (think of it as connecting flights to reach a particular destination).

Now think about...

- How should we treat their fields if we combine these rows?
- What aggregation would make sense if we merge?
- What would happen to the numeric fields if we merge the rows?

1. Grouping by segment
    a. Create a unique identifier for different segments of a trip based on the combination of the **trip_uuid**, **source_center**, and **destination_center** and name it as **segment_key**.
    b. You can use inbuilt functions like **groupby** and aggregations like **cumsum()** to merge the rows in columns **segment_actual_time**, **segment_osrm_distance**, **segment_osrm_time** based on the **segment_key.**
    c. This way you'll get new columns named **segment_actual_time_sum**, **segment_osrm_distance_sum**, **segment_osrm_time_sum**.
2. Aggregating at segment level
    a. Create a dictionary named **create_segment_dict**, that defines how to aggregate and select values.
        i. You can keep the first and last values for some numeric/categorical fields if aggregating them won't make sense.
    b. Further group the data by **segment_key** because you want to perform aggregation operations for different segments of each trip based on the **segment_key** value.
    c. The aggregation functions specified in the **create_segment_dict** are applied to each group of rows with the same **segment_key**.
    d. Sort the resulting DataFrame **segment**, by two criteria:
        i. First, it sorts by **segment_key** to ensure that segments are ordered consistently.
        ii. Second, it sorts by **od_end_time** in ascending order, ensuring that segments within the same trip are ordered by their end times from earliest to latest.

3. Feature Engineering:

Extract features from the below fields:

1. Calculate time taken between **od_start_time** and **od_end_time** and keep it as a feature named **od_time_diff_hour**. Drop the original columns, if required.
2. Destination Name: Split and extract features out of destination. City-place-code (State)
3. Source Name: Split and extract features out of destination. City-place-code (State)
4. Trip_creation_time: Extract features like month, year, day, etc.

4. In-depth analysis:

1. Grouping and Aggregating at Trip-level
    a. Groups the **segment** data by the **trip_uuid** column to focus on aggregating data at the trip level.
    b. Apply suitable aggregation functions like **first**, **last**, and **sum** specified in the **create_trip_dict** dictionary to calculate summary statistics for each trip.
2. Outlier Detection & Treatment
    a. Find any existing outliers in numerical features.
    b. Visualize the outlier values using **Boxplot**.
    c. Handle the outliers using the **IQR** method.
3. Perform one-hot encoding on categorical features.
4. Normalize/ Standardize the numerical features using **MinMaxScaler** or **StandardScaler**.

5. Hypothesis Testing:

1. Perform hypothesis testing / visual analysis between :
    a. actual_time aggregated value and OSRM time aggregated value.
    b. actual_time aggregated value and segment actual time aggregated value.
    c. OSRM distance aggregated value and segment OSRM distance aggregated value.
    d. OSRM time aggregated value and segment OSRM time aggregated value.
2. **Note:** Aggregated values are the values you'll get after merging the rows on the basis of **trip_uuid**.

6. **Business Insights & Recommendations**
    - Patterns observed in the data along with what you can infer from them.
        - Check from where most orders are coming from (State, Corridor, etc.)
        - Busiest corridor, avg distance between them, avg time taken, etc.
    - Actionable items for the business.

FAQs

Q. Which platform am I supposed to use?
You may use either Google Colab or Jupyter notebook.

Q. I am having issues setting up Jupyter notebook
Install Anaconda using the link. Once Anaconda has been installed on your system, open Jupyter Notebook. Refer link.