flaschenpost SE – IT Exercise Task #12 „Service Time Predicition" – Data Science MLOps

Ziel:
Erstelle einen funktionierenden, reproduzierbaren End-to-End-Prototypen, der auf Basis historischer Daten die *Service Time* (Dauer der Auslieferung) für Bestellungen vorhersagt.

Der Fokus liegt auf einem sauberen technischen Aufbau, einer funktionierenden API und Reproduzierbarkeit – nicht auf der Modellgüte.

Hintergrund
flaschenpost liefert in über 30 Städten in Deutschland Getränke und Supermarktartikel aus.

Jede Lieferung besteht aus mehreren Schritten (Vorbereitung, Fahrt, Service, Rückgabe). Ein zentraler Teil des Prozesses ist die **Service Time** – also die Zeit, die ein Fahrer für die tatsächliche Lieferung beim Kunden benötigt.

Diese Zeit hängt von verschiedenen Faktoren ab, z. B. Etage, Aufzug, oder Umfang der Bestellung.

Deine Aufgabe ist es, auf Basis eines bereitgestellten Datensatzes ein Modell zu entwickeln, das diese Servicezeit für einzelne Bestellungen vorhersagt.

Daten
Die bereitgestellten Datensätze enthalten Informationen zu Bestellungen, Artikeln, Servicezeiten und Fahrern:

1. **Orders**
   warehouse_id, order_time, has_elevator, floor, is_business, web_order_id, customer_id
2. **Articles**
   warehouse_id, box_id, article_id, article_weight_in_g, web_order_id
3. **ServiceTimes**
   service_time_start, service_time_end, service_time_in_minutes, order_datetime, web_order_id, driver_id, trip_id, customer_id
4. **DriverOrderMapping**
   driver_id, web_order_id

Label:
service_time_in_minutes = service_time_end - service_time_start

> Auf einem Trip *(trip_id)* werden üblicherweise mehrere Bestellungen *(web_order_id)* durch einen Fahrer *(driver_id)* ausgeliefert.

Mini-Challenge – Aufgaben
Erstelle ein System, das:

1. **Daten für das Training aufbereitet inkl. Feature Engineering**
2. **Ein Modell trainiert**, das die Servicezeit für Bestellungen vorhersagen kann
3. **Eine REST-API** bereitstellt, die POST-Anfragen mit *(driver_id, web_order_id)* entgegennimmt und Vorhersagen zurückgibt. Wobei *web_order_id* ein Wert aus **Orders** ist und *driver_id* beliebig sein kann
    - Die API soll **Batch-Requests** (mehrere Tuples) verarbeiten können.
4. **Feature- und Prediction-Logging** implementiert, sodass nachvollziehbar bleibt, welche Daten und Vorhersagen verarbeitet wurden.
5. **Reproduzierbare Ergebnisse** liefert
6. **(Optional)** mehrere Modellversionen verwalten und abrufbar machen kann
7. **(Optional)** gespeicherte Features und Predictions inspizierbar machen

Wichtig:
Der Fokus liegt auf einer lauffähigen, reproduzierbaren Lösung – nicht auf komplexen Modellen oder Tuning.

Anforderungen an das Ergebnis
Dein Projekt sollte Folgendes enthalten:

- **Feature Engineering & EDA** (z.B. EDA einem Notebook)
- **Trainingsskript**
- **API**: stellt Endpunkte für Predictions bereit.
- **Logging-Komponente**: speichert relevante Informationen zu Features, Predictions und Modellversionen.
- **README.md**: erklärt, wie dein Projekt ausgeführt werden kann.

README.md – Erwarteter Inhalt
Bitte beschreibe in deinem README klar und knapp:

1. **Setup** – wie das Projekt gestartet werden kann (Umgebung, Installation, Startbefehle).
2. **Feature Engineering** - wie die Daten für das Model Training bereitgestellt werden
3. **Training** – wie das Modell trainiert und gespeichert wird.
4. **API** – wie die Schnittstelle gestartet wird und wie Anfragen aussehen sollen (z. B. POST mit JSON-Body).
5. **Logging** – wo Logs gespeichert werden und welche Informationen sie enthalten.
6. **Reproduzierbarkeit** – Wie sichergestellt wird, dass das Modelltraining unter denselben Bedingungen
   (gleiche Daten, gleiche Parameter, ggf. gleiche Seeds) zu konsistenten Ergebnissen führt.
7. **(Optional)** – wie Modellversionen oder Feature-Lookups verwendet werden können.

Rahmenbedingungen

- Zeitrahmen: ca. **4 Stunden**
- Programmiersprache: **Python**

- Frei wählbare Bibliotheken für ML, API, Logging etc.
- Fokus: **Struktur, Sauberkeit, Reproduzierbarkeit, Funktionalität**
- Modellperformance ist **nicht** so wichtig

Hinweis
Die Lösung soll zeigen, wie du an ein End-to-End-Problem herangehst:
vom Aufbereiten der Daten, Training über die Inferenz bis zur API und zum Logging – mit Blick auf Skalierbarkeit, Nachvollziehbarkeit und Wartbarkeit.

Die zugehörigen Datensätze findest du in beigefügtem Sharepoint Link.
Stelle uns gerne ein public repository (Link) oder eine Zip Datei zur Verfügung.

Viel Erfolg!