DATA TEST SEPIDEH

Contents

Data collection

Data collection phase is either using collected data in CSV file format or collecting real-time data using API activities, or database.

CSV: Loading dataset by reading or loading the data from a csv file using Pandas.read_csv and saving it as a Dataframe to be able to handle the data.

API: Collecting data by connecting to the online source using a *get* request from *requests* library in Python with this line of code requests.get("web page address").text and saving it into a variable. Then we need to parse the data to make it usable for data manipulation which is in JSON format. It saves the data as key-value pairs which we can save into CSV after reading it. For *structured JSON* files it's easy,

```
with open("jsonfile.json", encoding = "utf-8") as inputfile:
    data = pandas.read_json(inputfile)
data.to_csv("csvfile.csv", encoding = "utf-8", index = False)
```

for *unstructured files* we need an extra line of code before converting the data to CSV format, after loading the JSON using json.loads(inputfile.read()) we normalize the data to be able to handle it. This line of code df = pandas.json_normalize(data) does do before converting the dataset to CSV.

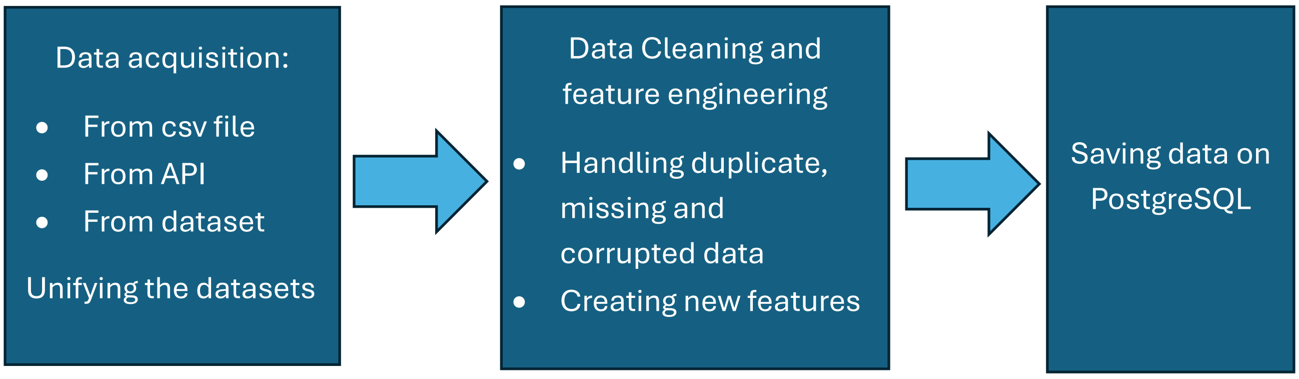
Database: to be able to access a database (for example PostgreSQL) using Python, we need to have the appropriate connector for the database system that we are using. **Pyodbc, psycopg2, sqlite3** are a few libraries that use .connect command for this purpose. Then with **read_sql()** commands we would be able to run queries.

consistency

- To have a consistence dataset we need to make a consistence one, even though the sources of collection are different, offline and online. We need to unify the dataset and the data format. By feature engineering we can handle missing and invalid values and have datasets merged. We need to convert them to the same format which here I have chosen CSV which is easy to load as dataframe and apply preprocessing and feature engineering techniques. Also, can be easily saved to CSV files and download.

- To have consistency in methods used and avoid errors, try_except commands are beneficial. Coding the operations needed, as functions can also help with consistency.

Overview of the steps



Data acquisition:
From csv file
From API
From dataset
Unifying the datasets
Data Cleaning and feature engineering
Handling duplicate, missing and corrupted data
Creating new features
Saving data on PostgreSQL

Data Cleaning and Preprocessing

Features that are currently present in our dataset are "employee_id, event_type, timestamp, and details".

- **employee_id** should not be null and it is necessary to identify the relationship between actions and person taking that action for analytical reasons like which group of people take the most specific actions (a simple example). In that case to handle missing values, we can drop the rows containing these null values.

  But if the action and the timestamp (for example) is the point of interest, it can be ignored.

- **event_type** is the type of action taken by the user. Considering the task and what is needed to be extracted, missing values can be filled with the mode (if the mode isn't null).
- **timestamp** is featuring the date and time. It is needed to be in *datetime* format so that we can extract detailed features from it, like day, month, and time of day. Its values must not be later than today and now. For example it can't be dated for an oncoming day. So, a limit should be set for the API data collection to check its validity.

  To handle missing values we can replace them with null or the neighbouring values which can be close to the missing time value.
- **details** is presenting key_value pairs as in a dictionary type. According to the task, we need to separate them into different features. The pairs represent the event itself which can be the button pushed or a click. The new features will be **key_pressed** and **task_name**. If the column contains a dictionary the *key* will fill the first feature and the *value* will fill the second one. Otherwise, we need to convert each datapoint to dict and then follow the steps mentioned.

Loading data into PostgreSQL database

To save dataframes and CSV files from Python to PostgreSQl, these libraries can be used **psycopg2** and **sqlalchemy.** We should again establish a connection and then convert the dataframe or CSV to SQL file and write the data into the server table.

Loading the data one row at a time is time consuming because each time a connection is established and closed which is very much inconvenient for large datasets. Also, we will face the same problem with loading a large dataset all at once into the PostgreSQL because it is processed and saved which takes time. **Small batches** however would be convenient to process, establish the connection and save the data.

While doing this step, it is crucial to have in mind that format in source and destination should be the same to avoid errors.

AI readiness and scaling

There are different data ingestion tools including Apache kafka, Spark Streaming (component of Apache Spark), and Spark Structured Screaming. Each having some advantages and disadvantages.

According to documentation and online sources, kafka has some disadvantages including complexity, scalability, and unreliable data collection. Spark Streaming is on the other hand favourable to kafka because of its scalability, ability to handle complexity, and fault tolerance. However, it is a legacy project and there are no upcoming updates for it. The recommended replace is Structured Streaming, a scalable and fault-tolerant stream processing engine built on the Spark SQL engine, according to Apache Spark documentation.

Now if we want to use this engine, according to documentation, spark.readStream will generate a reader object that can be used to have the streaming data saved (for example as a CSV file with the csv method). It provides access to online data.

With the correct functions we can check the data for cleanness and anomalies. A function that can apply all the steps necessary for data cleaning and feature engineering is beneficial for this part.

AI model dataset

Considering the activity details collected from user activities and feature dependencies, features in a pre-processed dataset could include

- extracted time details from the one feature (day, month, etc)
- extracting features like peak_hour and the opposite
- some new features related to each user such as number of clicks or pushed buttons
- finding the mode or counting values for features like event_type or details and finding the most pushed buttons or the event with the highest number

- some features based on comparison for example with the highest values
- Sometimes, based on the task, features created by *groupby* method can help extracting valuable information based on more than one feature.

To prepare the dataset for model training, we have to make sure it is ready, namely, handling duplicate, missing or invalid datapoints, normalizing or feature encoding. Feature importance can also help eliminate unnecessary features, along with correlation detection.

As for choosing operations on how to prepare dataset for model training (data should not contain categorical values),

- **get_dummies** is beneficial when we have two values with the same merit that we want to feature engineer, like true/false which is not ordered.
- By orderly values like scaling from very good to very bad, **label_encolding** is the right choice.
- When there is no order, **one_hot** encoding is a better choice, however, it can create a sparse dataset if there are a large number of different values in a feature.
- **Feature_scaling** is used when we need to normalize *numerical values*. For example, when the numbers are too high, it helps to normalize them to a more suitable range for feeding to the model.

The model

Considering the final task the model could be Supervised, unsupervised, or semi-supervised. Based on the task and what we want to achieve, we may need to define a target feature for the dataset before feeding to the model.