Big spark Ecommerce dataset example data use  
  
My proposal for this is to build a simple **fraud detection model**. This is a classic ML problem that directly addresses the suspected\_fraud column in the data.

**1. Data Exploration and Foundational Engineering**

The first step is still data exploration and cleaning, but with a focus on preparing the data for a model. I will use **Python** with **Pandas** and **Scikit-learn** in a **Jupyter Notebook**, which is excellent for iterative development.

**Identified Quality Issues to Address:**

* **Missing Values:** The suspected\_fraud column has missing values. I'll need to decide how to handle this. Since it's our target variable, the best practice is to remove these row. This is to have a clean target variable for supervised learning.
* **Inconsistent Formatting:** The country\_code column has inconsistent casing (gb, GB, UK). I will standardize these to a single format (GB) and handle the missing values by creating a new category like Unknown.
* **Data Types:** I will ensure the order\_date column is converted to a proper datetime format for potential time-based feature engineering.

**Engineering a Reproducible Environment:** To demonstrate good engineering practices, I have started by setting up a virtual environment and creating a requirements.txt file listing all the libraries used (pandas, scikit-learn, etc.).

**2. The "Something Interesting" Part: The ML Pipeline**

For this we will build an example ml model trained from the data set data with a test, train split of the data. The steps for this are listed below

* **Feature Engineering:**
  + I'll create new features from the order\_date, such as the **day of the week** or **month** of the order, as fraud patterns can be cyclical.
  + I'll also consider extracting the email **domain** from the email column to see if certain email providers or domains are more associated with fraud.
* **Model Building and Evaluation:**
  + **Data Split:** I'll split the data into training and testing sets to properly evaluate the model's performance on unseen data.
  + **Baseline Model:** I'll create a simple baseline model (e.g., a Logistic Regression classifier) to establish a minimum performance target (haven’t done this in this example).
  + **Candidate Models:** I'll choose a more sophisticated model, like a **Random Forest** or **Gradient Boosting** classifier, which are well-suited for tabular data and can handle the data's inherent class imbalance (fraud cases don’t happen that often).
  + **Evaluation Metrics:** Since fraud data is often imbalanced (far fewer fraudulent transactions than legitimate ones), I will not rely on accuracy alone. Instead, I will use more appropriate metrics like **Precision**, **Recall**, and the **F1-score**.

**3. The "ML Engineering" Aspect: From Model to Production**

Going ML

* **Code Structure:** The project is a single script model\_train\_test\_serve.py that houses loading the data, feature engineering and training the model. This also houses serializing the model using joblib (along with the one hot encoder and feature names) ready to be used for inference.
* **Deployment:** The model is deployed then loaded from the filesystem and with inference post data (that is one-hot encoded) a prediction is made. I have created a simple FLASK REST API endpoint (‘/predict\_fraud’) that takes new order data as input and returns a fraud prediction. This can then be served to a live application.

Final submission is a jupyter notebook that showcases the process of data cleaning and exploration to building a production-ready model. Complete with code, explanations and a clear rationale for every decision. This showcases an example of a full lifecycle of an ML project. An improvement for this would be splitting each part of the script into separate files for modularity.

# Deployment to AWS:

* The above notebook can be trained and saved to AWS model artifact registry where it can be deployed into a real-time endpoint (I would choose a real time endpoint as this is a fraud detection use case which requires low latency). We can use sage maker notebook instance for this. Data on Instances used for running the notebook can also be encrypted using AWS KMS. For scaling sagemaker supports autoscaling for models hosted on the real time inference endpoint.
* For security we would have this situated in a VPC-only with no internet access
* S3 buckets where the data is stored would also not have public access and data at rest can be encrypted using AWS KMS
* ECR for container images would be private and scanned
* Execution roles would have **least-privilege access**
* **Monitoring audits would use Cloudtrail to track sagemaker changes and Cloudwatch with sagemaker model monitoring to track training/deployment instance logs**
* **For infrastructure the workflow lifecycle can be automated (so ingesting new ecommerce orders > to train > new improved models) using sagemaker pipelines**
* **An example test file is giving to test the simple train test serve script**

**Any queries, questions, happy to go over in the final interview call**