Datalentech.

`

DATA SCIENTIST CASE STUDY TEST

Focus Area: Time Series Forecasting & Real-Time Deployment

Estimated Duration: 6–10 hours (core + optional extension)

BUSINESS CONTEXT

You have been brought on as a data scientist at a national retail chain with hundreds of locations across the country. Your manager has asked you to build a model that can forecast daily sales per store to support smarter decisions around inventory, staffing, and promotions. Additionally, the business wants to integrate the model into a real-time API that receives daily inputs and returns sales predictions.

OBJECTIVE

1. Build an accurate forecasting model for daily sales across multiple stores.
2. Produce a forecast for the next six weeks.
3. (Optional Extension) Build and deploy a real-time API that serves predictions based on new inputs.

DATASET

Source: Rossmann Store Sales (Kaggle) **Link**:
https://www.kaggle.com/competitions/rossmann-store-sales/data?select=store.csv

Key Features:

- Date: Date of transaction
- **Id**: an Id that represents a (Store, Date) duple within the test set
- **Store**: a unique Id for each store
- **Sales**: the turnover for any given day (this is what you are predicting)
- **Customers**: the number of customers on a given day
- **Open**: an indicator for whether the store was open: 0 = closed, 1 = open
- **StateHoliday**: indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
- **SchoolHoliday**: indicates if the (Store, Date) was affected by the closure of public schools
- **StoreType**: differentiates between 4 different store models: a, b, c, d
- **Assortment**: describes an assortment level: a = basic, b = extra, c = extended
- **CompetitionDistance**: distance in meters to the nearest competitor store
- **CompetitionOpenSince[Month/Year]**: gives the approximate year and month of the time the nearest competitor was opened
- **Promo**: indicates whether a store is running a promo on that day
- **Promo2**: Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating

`

- **Promo2Since[Year/Week]**: describes the year and calendar week when the store started participating in Promo2
- PromoInterval: describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

You are free to engineer additional features or filter the dataset for efficient modeling.

CORE DELIVERABLES

1. **Modeling Notebook / Python Script**
    - Load and process the data
    - Explore and analyze key time series patterns
    - Build and evaluate forecasting model(s)
    - Forecast future sales for each store for the next 6 weeks
2. **Analytical Report (1–2 pages, PDF or Markdown)**
    - Summary of modeling approach and rationale
    - Key findings and insights
    - Evaluation metrics (e.g., RMSE, MAE, MAPE)
    - Forecast visualization examples

OPTIONAL EXTENSION: REAL-TIME DEPLOYMENT CHALLENGE

Goal: Build and expose your forecasting model via an API that supports real-time predictions.

Requirements:

- Create a REST API using FastAPI or Flask
- Endpoint should accept POST requests with JSON input like:

```
{
  "store": 215,
  "date": "2015-09-01",
  "promo": 1,
  "state_holiday": "0",
  "school_holiday": 0,
  "day_of_week": 2
}
```

- Return the forecasted sales in JSON format:

```
{
  "predicted_sales": 4586.42
`
        }
```

Bonus Points:

- Add Swagger/OpenAPI documentation
- Add a /retrain route to allow model updating with new data
- Dockerize the service
- Deploy the API using a free-tier cloud service (e.g., Render, Hugging Face Spaces, Heroku, etc.)

EVALUATION CRITERIA

Area	**Criteria**
Exploratory Analysis	Insightful visualizations, clear understanding of trends and anomalies
Forecasting Model	Accurate, well-justified, cleanly implemented model(s)
Validation Strategy	Time-aware validation and solid performance metrics
Code Quality	Clean, modular, well-documented code
Business Thinking	Clear recommendations and explanation of findings
(Bonus) Deployment	Functional, documented, and testable API for real-time inference

SUBMISSION INSTRUCTIONS

Please submit the following:

- Your Python notebook or scripts with forecasting code
- Your report (PDF or markdown)
- (Optional) A URL to your deployed API or GitHub repo with deployment instructions

Good luck, and we look forward to seeing your approach!