Kelp Global Junior Data Scientist - Practice Papers

Prepared by Perplexity AI

Contents

1 KELP GLOBAL - JUNIOR DATA SCIENTIST PRACTICE PAPERS

1.1 Understanding the Format (Based on Senior's Feedback)

Your seniors mentioned that the interview was **tough** and included:
- Dataset with tasks like bug finding, providing solutions, and explaining approach
- Embedding demonstration
- RAG workflow design
- SQL query debugging and normalization
- JSON structural flaw detection and fixing
- Python script error identification and correction
- Scenario-based model selection with rules definition
- **No DSA questions**

2 PRACTICE PAPER SET 1

2.1 Section A: JSON Data Debugging (20 marks)

Question 1: You are given the following JSON data representing customer orders. Identify ALL structural flaws, explain what errors they can cause, and provide the corrected JSON.

```
{
    'customer_id': 1001,
    "name": "Rahul Sharma"
    "orders": [
        {"order_id": "ORD001", "amount": 2500.00, "status": "delivered",},
        {"order_id": "ORD002", "amount": undefined, "status": "pending"},
        {"order_id": "ORD002", "amount": 1800, "status": "shipped"}
    ],
    "email": 'rahul@example.com',
    "phone": +919876543210,
    "address": {
        "city": "Mumbai"
        "pincode": 400001
    }
}
```

Tasks:
(a) List all structural flaws and what errors they will cause (8 marks)
(b) Provide the corrected JSON (6 marks)
(c) How would you validate this JSON programmatically in Python? (6 marks)

2.2 Section B: SQL Debugging and Normalization (25 marks)

Question 2: Given the following table with data:

EmpID	EmpName	Dept	DeptHead	Projects	Salary	JoinDate
101	Amit Kumar	IT	Priya Singh	Proj1,Proj2	50000	2023-15-01
102	Sneha Patel	HR	Rahul Joshi	Proj3	NULL	01-Mar-2022
101	Amit Kumar	IT	Priya Singh	Proj1,Proj2	55000	2023-15-01

Tasks:
(a) Identify incorrect data and structural issues in this table (5 marks)
(b) Explain the anomalies present (insertion, update, deletion) (5 marks)
(c) Normalize this table to 3NF with proper schema design (8 marks)
(d) Write SQL query to create normalized tables with appropriate constraints (7 marks)

2.3 Section C: Python Script Debugging (20 marks)

Question 3: The following Python script has multiple errors. Identify the line numbers where errors will occur, explain why, and provide the corrected script.

```
1.  import pandas as pd
2.  import numpy as np
3.
4.  def analyze_sales(filepath)
5.      df = pd.read_csv(filepath)
6.
7.      # Clean the data
8.      df['revenue'] = df['price'] * df[quantity]
9.
10.     # Handle missing values
11.     df.fillna(method='ffill' inplace=True)
12.
13.     # Calculate statistics
14.     avg_revenue = df['revenue'].mean
15.     max_sale = df.loc[df['revenue'] == df['revenue'].max()]
16.
17.     # Group by category
18.     category_sales = df.groupby('category').agg({
19.         'revenue': ['sum', 'mean']
20.         'quantity': 'count'
21.     })
22.
23.     return avg_revenue max_sale, category_sales
24.
25. if __name__ = "__main__":
26.     result = analyze_sales("sales_data.csv")
27.     Print(result)
```

Tasks:
(a) Identify all errors with line numbers and reasons (12 marks)
(b) Rewrite the corrected script (8 marks)

2.4 Section D: RAG Workflow Design (15 marks)

Question 4: You are building a customer support chatbot for Kelp Global that answers questions about private equity investments using company documents.

Tasks:
(a) Draw a complete RAG workflow diagram showing all components (8 marks)
(b) Explain the role of each component (embedding model, vector database, retriever, LLM) (7 marks)

2.5 Section E: Model Selection Scenarios (20 marks)

Question 5: For each scenario below, specify which model/technique you would use, provide 3 rules/reasons for your choice, and mention any preprocessing required.

Scenario	Your Answer
S1: Predicting customer churn (yes/no) based on transaction history	
S2: Grouping customers based on purchasing behavior (no labels available)	
S3: Predicting house prices based on features like area, location, bedrooms	
S4: Detecting anomalies in credit card transactions	
S5: Classifying customer reviews into 5 sentiment categories	
S6: Building a recommendation system for products	

Format for each scenario:
- Model/Technique chosen
- 3 Rules/Reasons for selection
- Key preprocessing steps

3 PRACTICE PAPER SET 1 - ANSWERS

3.1 Section A: JSON Data Debugging - ANSWERS

(a) Structural Flaws Identified:

1. **Line 2: Single quotes used for key** - `'customer_id'` uses single quotes instead of double quotes
- Error: JSON parsing will fail with "Unexpected token"

2. **Line 3: Missing comma** - No comma after `"Rahul Sharma"`
- Error: Syntax error, unexpected string

3. **Line 5: Trailing comma** - Comma after `"delivered",`
- Error: JSON doesn't allow trailing commas

4. **Line 6: Undefined value** - `"amount": undefined`
- Error: `undefined` is not a valid JSON data type

5. **Line 7: Duplicate key** - `"order_id": "ORD002"` appears twice
- Error: Data ambiguity, unpredictable parsing behavior

6. **Line 9: Single quotes for value** - `'rahul@example.com'`
- Error: JSON requires double quotes for strings

7. **Line 10: Unquoted number with special character** - `+919876543210`
- Error: Phone number should be string, '+' makes it invalid number

8. **Line 12: Missing comma** - No comma between "Mumbai" and "pincode"
- Error: Syntax error

(b) Corrected JSON:

```json
{
    "customer_id": 1001,
    "name": "Rahul Sharma",
    "orders": [
        {"order_id": "ORD001", "amount": 2500.00, "status": "delivered"},
        {"order_id": "ORD002", "amount": null, "status": "pending"},
        {"order_id": "ORD003", "amount": 1800, "status": "shipped"}
    ],
    "email": "rahul@example.com",
    "phone": "+919876543210",
    "address": {
        "city": "Mumbai",
        "pincode": 400001
    }
}
```

(c) Python Validation Code:

```python
import json

def validate_json(json_string):
    try:
        data = json.loads(json_string)
        print("Valid JSON")
        return True, data
    except json.JSONDecodeError as e:
        print(f"Invalid JSON: {e.msg} at line {e.lineno}, column {e.colno}")
        return False, None

# Using jsonschema for structure validation
from jsonschema import validate, ValidationError

schema = {
    "type": "object",
    "required": ["customer_id", "name", "orders"],
    "properties": {
        "customer_id": {"type": "integer"},
        "name": {"type": "string"},
        "orders": {
```

```
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "order_id": {"type": "string"},
                    "amount": {"type": ["number", "null"]},
                    "status": {"type": "string"}
                }
            }
        }
    }
}
```

3.2 Section B: SQL Debugging and Normalization - ANSWERS

(a) Incorrect Data and Structural Issues:

1. **Duplicate EmpID**: EmpID 101 appears twice with different salaries
2. **Multi-valued attribute**: Projects column contains "Proj1,Proj2" - violates 1NF
3. **Invalid date format**: "2023-15-01" has invalid month (15)
4. **Inconsistent date formats**: Different formats used (YYYY-MM-DD vs DD-Mon-YYYY)
5. **NULL value**: Salary is NULL for EmpID 102
6. **Transitive dependency**: DeptHead depends on Dept, not directly on EmpID

(b) Anomalies Present:

- **Insertion Anomaly**: Cannot add new department without adding employee
- **Update Anomaly**: If DeptHead changes, must update multiple rows
- **Deletion Anomaly**: Deleting last employee in dept loses dept info

(c) Normalized Schema (3NF):

Employees Table:

```
Employees(EmpID PK, EmpName, DeptID FK, Salary, JoinDate)
```

Departments Table:

```
Departments(DeptID PK, DeptName, DeptHead)
```

Projects Table:

```
Projects(ProjectID PK, ProjectName)
```

Employee_Projects Table:

```
Employee_Projects(EmpID FK, ProjectID FK) - Composite PK
```

(d) SQL to Create Normalized Tables:

```sql
-- Departments Table
CREATE TABLE Departments (
    DeptID INT PRIMARY KEY AUTO_INCREMENT,
    DeptName VARCHAR(50) NOT NULL UNIQUE,
    DeptHead VARCHAR(100)
);

-- Employees Table
CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    DeptID INT,
    Salary DECIMAL(10,2) NOT NULL DEFAULT 0,
    JoinDate DATE NOT NULL,
    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID),
    CONSTRAINT chk_salary CHECK (Salary >= 0)
);

-- Projects Table
CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY AUTO_INCREMENT,
    ProjectName VARCHAR(100) NOT NULL
);

-- Employee_Projects Junction Table
CREATE TABLE Employee_Projects (
    EmpID INT,
    ProjectID INT,
    PRIMARY KEY (EmpID, ProjectID),
    FOREIGN KEY (EmpID) REFERENCES Employees(EmpID),
    FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID)
);
```

3.3 Section C: Python Script Debugging - ANSWERS

(a) Errors Identified:

Line	Error	Reason
4	Missing colon	Function definition needs ‘:‘ at end
8	Missing quotes	‘df[quantity]‘ should be ‘df[’quantity’]‘
11	Missing comma	Between ‘’ffill’‘ and ‘inplace=True‘
14	Missing parentheses	‘mean‘ is a method, needs ‘()‘
19-20	Missing comma	Between dictionaries in ‘agg()‘
23	Missing comma	Between return values
25	Single ‘=‘ used	Should be ‘==‘ for comparison
27	Capital P	‘Print‘ should be ‘print‘

(b) Corrected Script:

```python
import pandas as pd
import numpy as np
```

```python

def analyze_sales(filepath):
    df = pd.read_csv(filepath)

    # Clean the data
    df['revenue'] = df['price'] * df['quantity']

    # Handle missing values
    df.fillna(method='ffill', inplace=True)

    # Calculate statistics
    avg_revenue = df['revenue'].mean()
    max_sale = df.loc[df['revenue'] == df['revenue'].max()]

    # Group by category
    category_sales = df.groupby('category').agg({
        'revenue': ['sum', 'mean'],
        'quantity': 'count'
    })

    return avg_revenue, max_sale, category_sales

if __name__ == "__main__":
    result = analyze_sales("sales_data.csv")
    print(result)
```

3.4 Section D: RAG Workflow Design - ANSWERS

(a) RAG Workflow Diagram:

```
[Company Documents]
        |
        v
[Document Loader/Chunker]
        |
        v
[Embedding Model (OpenAI/BERT)]
        |
        v
[Vector Database (Pinecone/Chroma/FAISS)]
        ^
        |
[User Query] --> [Query Embedding] --> [Similarity Search]
        |                                      |
        v                                      v
[Retrieved Context + User Query] ------> [LLM (GPT/LLaMA)]
                                               |
                                               v
                                      [Generated Response]
```

(b) Component Roles:

1. **Embedding Model**: Converts text chunks and queries into dense vector representations (embeddings) that capture semantic meaning. Similar concepts have vectors close in space.

2. **Vector Database**: Stores document embeddings and enables efficient similarity search.

Uses approximate nearest neighbor algorithms for fast retrieval.

3. **Retriever**: Fetches the most relevant document chunks based on query embedding similarity. Determines the context window for the LLM.

4. **LLM**: Takes the retrieved context and user query to generate coherent, contextually appropriate responses. Grounds responses in retrieved information.

3.5 Section E: Model Selection Scenarios - ANSWERS

S1: Customer Churn Prediction (Binary Classification)
- **Model**: Logistic Regression, Random Forest, or XGBoost
- **Rules**:
1. Binary outcome (churn/no churn) suits classification algorithms
2. Random Forest handles non-linear relationships and feature importance
3. XGBoost provides high accuracy with imbalanced datasets
- **Preprocessing**: Handle class imbalance (SMOTE), feature scaling, encode categoricals

S2: Customer Grouping (Clustering)
- **Model**: K-Means Clustering or DBSCAN
- **Rules**:
1. No labels available - unsupervised learning required
2. K-Means works well with spherical clusters of similar sizes
3. Elbow method helps determine optimal number of clusters
- **Preprocessing**: Feature scaling (StandardScaler), dimensionality reduction (PCA)

S3: House Price Prediction (Regression)
- **Model**: Linear Regression, Ridge/Lasso, or Gradient Boosting Regressor
- **Rules**:
1. Continuous target variable (price) requires regression
2. Ridge/Lasso handles multicollinearity and feature selection
3. Gradient Boosting captures non-linear patterns
- **Preprocessing**: Handle missing values, encode location, feature engineering

S4: Anomaly Detection in Transactions
- **Model**: Isolation Forest or One-Class SVM
- **Rules**:
1. Anomalies are rare events - one-class classification needed
2. Isolation Forest efficiently isolates outliers
3. Doesn't require labeled fraud examples for training
- **Preprocessing**: Feature engineering on transaction patterns, scaling

S5: Sentiment Classification (Multi-class)
- **Model**: BERT or Random Forest with TF-IDF
- **Rules**:
1. Multi-class text classification - 5 sentiment categories
2. BERT captures contextual word embeddings for better understanding
3. Pre-trained models reduce training data requirements
- **Preprocessing**: Text cleaning, tokenization, handling class imbalance

S6: Product Recommendation System
- **Model**: Collaborative Filtering or Content-Based + Matrix Factorization
- **Rules**:
1. Collaborative filtering uses user-item interaction patterns
2. Matrix factorization handles sparse user-item matrices
3. Hybrid approach combines strengths of multiple techniques
- **Preprocessing**: Create user-item matrix, handle cold start problem

4 PRACTICE PAPER SET 2

4.1 Section A: JSON Data Debugging (20 marks)

Question 1: Analyze the following JSON representing API response for employee data. Find all flaws and fix them.

```
{
    "api_version": 2.1,
    "timestamp": 2024-12-02T10:30:00,
    "status_code": "200",
    "employees": [
        {
            "id": 001,
            "name": "Priya Mehta",
            "department": "Data Science",
            "skills": ["Python", "SQL", "ML",]
            "salary": 75000,
            "manager_id": null
        },
        {
            "id": 002,
            "name": 'Vikram Singh'
            "department": "Data Science",
            "skills": ["Python", "Tableau"],
            "salary": NaN,
            "manager_id": 001
        }
    ]
    "metadata": {
        "total_count": 2,
        "page": 1,
    }
}
```

Tasks:
(a) List all structural flaws with potential error messages (8 marks)
(b) Provide corrected JSON (6 marks)
(c) Write a Python function to detect and report common JSON issues (6 marks)

4.2 Section B: SQL Debugging and Normalization (25 marks)

Question 2: Analyze the following product inventory table:

ProductID	ProductName	Category	SupplierName	SupplierContact	OrderHistc
P001	Laptop Dell	Electronics	TechMart	9876543210, tech@mart.com	ORD1:10,(
P002	iPhone 15	Electronics,Mobile	AppleStore	NULL	ORD3:2
P001	Laptop Dell	Electronics	TechMart	9876543210, tech@mart.com	ORD1:10,(

Tasks:
(a) Identify data integrity and structural issues (6 marks)
(b) Explain why this violates normal forms (5 marks)
(c) Design normalized schema up to 3NF with relationships (7 marks)
(d) Write SQL queries for creating tables and a query to find products with negative stock (7 marks)

4.3 Section C: Python Script Debugging (20 marks)

Question 3: Debug the following data preprocessing script:

```
1.  import pandas as pd
2.  from sklearn.preprocessing import StandardScaler
3.  from sklearn.model_selection import train_test_split
4.
5.  class DataProcessor
6.      def __init__(self, data_path):
7.          self.df = pd.read_csv(data_path)
8.          self.scaler = StandardScaler[]
9.
10.     def clean_data(self):
11.         # Remove duplicates
12.         self.df.drop_duplicates(inplace=true)
13.
14.         # Handle missing values
15.         for col in self.df.columns
16.             if self.df[col].dtype == 'object':
17.                 self.df[col].fillna('Unknown', inplace=True)
18.             else:
19.                 self.df[col].fillna(self.df[col].median, inplace=True)
20.
21.     def encode_features(self, categorical_cols)
22.         for col in categorical_cols:
23.             self.df[col] = pd.get_dummies(self.df[col])
24.
25.     def split_data(self, target_col, test_size=0.2):
26.         X = self.df.drop(target_col)
27.         y = self.df[target_col]
28.         return train_test_split(X, y test_size=test_size, random_state=42)
29.
```

```
30. if __name__ == '__main__':
31.     processor = DataProcessor('data.csv')
32.     processor.clean_data()
33.     X_train, X_test, y_train, y_test = processor.split_data['target']
```

Tasks:
(a) Identify all errors with line numbers and explanations (12 marks)
(b) Provide the corrected script (8 marks)

4.4 Section D: RAG Workflow Design (15 marks)

Question 4: Design a RAG system for a financial document Q&A application that handles PDF reports, Excel data, and regulatory documents.

Tasks:
(a) Draw a detailed RAG architecture showing document processing pipeline (8 marks)
(b) Explain chunking strategies and their trade-offs for financial documents (4 marks)
(c) How would you handle numerical data and tables in this RAG system? (3 marks)

4.5 Section E: Model Selection Scenarios (20 marks)

Question 5: For each scenario, choose appropriate technique and justify with 3 rules.

Scenario	Description
S1	Predicting loan default amount (continuous value) for defaulting customers
S2	Identifying topics from a large collection of news articles (no predefined topics)
S3	Real-time fraud detection where model needs to update with new patterns
S4	Predicting next month's sales from historical time-series data
S5	Building a spam classifier with very limited labeled data (100 examples)
S6	Image classification to identify product defects in manufacturing

---

5 PRACTICE PAPER SET 2 - ANSWERS

5.1 Section A: JSON Data Debugging - ANSWERS

(a) Structural Flaws:

1. **Line 3**: Unquoted timestamp - `2024-12-02T10:30:00` should be string `"2024-12-02T10:30:00"`
2. **Line 5**: Leading zeros in number - `001` is invalid; use `1` or `"001"`
3. **Line 8**: Trailing comma in array - `["Python", "SQL", "ML",]`
4. **Line 8**: Missing comma - After skills array, before salary

5. **Line 13**: Single quotes - `'Vikram Singh'` should use double quotes
6. **Line 13**: Missing comma - After name value
7. **Line 16**: `NaN` is not valid - Use `null` instead
8. **Line 19**: Missing comma - After employees array, before metadata
9. **Line 22**: Trailing comma - After `"page": 1,`

(b) Corrected JSON:

```
{
    "api_version": 2.1,
    "timestamp": "2024-12-02T10:30:00",
    "status_code": "200",
    "employees": [
        {
            "id": 1,
            "name": "Priya Mehta",
            "department": "Data Science",
            "skills": ["Python", "SQL", "ML"],
            "salary": 75000,
            "manager_id": null
        },
        {
            "id": 2,
            "name": "Vikram Singh",
            "department": "Data Science",
            "skills": ["Python", "Tableau"],
            "salary": null,
            "manager_id": 1
        }
    ],
    "metadata": {
        "total_count": 2,
        "page": 1
    }
}
```

(c) Python Function to Detect JSON Issues:

```
import json
import re

def detect_json_issues(json_string):
    issues = []

    # Check for single quotes
    if re.search(r"(?<!\)'", json_string):
        issues.append("Single quotes detected - use double quotes")

    # Check for trailing commas
    if re.search(r',\s*[}\]]', json_string):
        issues.append("Trailing comma detected")

    # Check for undefined/NaN
    if 'undefined' in json_string:
        issues.append("'undefined' is not valid JSON - use null")
    if re.search(r':\s*NaN', json_string):
        issues.append("NaN is not valid JSON - use null")

```

```
    # Check for unquoted strings/dates
    if re.search(r':\s*\d{4}-\d{2}-\d{2}T', json_string):
        issues.append("Possible unquoted datetime detected")

    # Try parsing
    try:
        json.loads(json_string)
        issues.append("JSON parsed successfully")
    except json.JSONDecodeError as e:
        issues.append(f"Parse error: {e.msg} at line {e.lineno}")

    return issues
```

5.2 Section B: SQL Normalization - ANSWERS

(a) Data Integrity Issues:

1. **Duplicate ProductID**: P001 appears twice with different prices
2. **Multi-valued attributes**: Category has "Electronics,Mobile", OrderHistory has multiple orders
3. **Composite data**: SupplierContact contains phone AND email together
4. **Negative stock**: StockQty = -5 (invalid business rule)
5. **NULL supplier contact**: Data incompleteness

(b) Normal Form Violations:

- **1NF Violation**: Multi-valued attributes (Category, OrderHistory, SupplierContact)
- **2NF Violation**: Supplier info depends on SupplierName, not ProductID
- **3NF Violation**: SupplierContact transitively depends on SupplierName

(c) Normalized Schema (3NF):

```
Products(ProductID PK, ProductName, Price, StockQty)
Categories(CategoryID PK, CategoryName)
Product_Categories(ProductID FK, CategoryID FK) - Composite PK
Suppliers(SupplierID PK, SupplierName, Phone, Email)
Product_Suppliers(ProductID FK, SupplierID FK)
Orders(OrderID PK, ProductID FK, Quantity, OrderDate)
```

(d) SQL Implementation:

```
CREATE TABLE Products (
    ProductID VARCHAR(10) PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL,
    Price DECIMAL(10,2) NOT NULL CHECK (Price > 0),
    StockQty INT DEFAULT 0 CHECK (StockQty >= 0)
);

CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY AUTO_INCREMENT,
    CategoryName VARCHAR(50) UNIQUE NOT NULL
```

```sql
);

CREATE TABLE Product_Categories (
    ProductID VARCHAR(10),
    CategoryID INT,
    PRIMARY KEY (ProductID, CategoryID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

CREATE TABLE Suppliers (
    SupplierID INT PRIMARY KEY AUTO_INCREMENT,
    SupplierName VARCHAR(100) NOT NULL,
    Phone VARCHAR(15),
    Email VARCHAR(100)
);

-- Query for negative stock (would be prevented by CHECK constraint)
SELECT ProductID, ProductName, StockQty
FROM Products
WHERE StockQty < 0;
```

5.3 Section C: Python Script Debugging - ANSWERS

(a) Errors:

Line	Error	Explanation
5	Missing colon	‘class DataProcessor‘ needs ‘:‘
8	Square brackets	‘StandardScaler[]‘ should be ‘StandardScaler()‘
12	Lowercase true	Python boolean is ‘True‘, not ‘true‘
15	Missing colon	‘for‘ loop needs ‘:‘ at end
19	Missing parentheses	‘median‘ is method, needs ‘()‘
21	Missing colon	Function definition needs ‘:‘
23	Wrong usage	‘pd.get_dummies‘ creates DataFrame, not fit for column assignment
26	Missing axis	‘drop()‘ needs ‘axis=1‘ for columns
28	Missing comma	Between ‘y‘ and ‘test_size‘
33	Square brackets	Function call should use ‘()‘ not ‘[]‘

(b) Corrected Script:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

class DataProcessor:
    def __init__(self, data_path):
        self.df = pd.read_csv(data_path)
        self.scaler = StandardScaler()

    def clean_data(self):
        # Remove duplicates
```

```
        self.df.drop_duplicates(inplace=True)

        # Handle missing values
        for col in self.df.columns:
            if self.df[col].dtype == 'object':
                self.df[col].fillna('Unknown', inplace=True)
            else:
                self.df[col].fillna(self.df[col].median(), inplace=True)

    def encode_features(self, categorical_cols):
        le = LabelEncoder()
        for col in categorical_cols:
            self.df[col] = le.fit_transform(self.df[col])

    def split_data(self, target_col, test_size=0.2):
        X = self.df.drop(target_col, axis=1)
        y = self.df[target_col]
        return train_test_split(X, y, test_size=test_size, random_state=42)

if __name__ == '__main__':
    processor = DataProcessor('data.csv')
    processor.clean_data()
    X_train, X_test, y_train, y_test = processor.split_data('target')
```

5.4 Section D: RAG Workflow - ANSWERS

(a) Financial RAG Architecture:

```
                    Document Sources
                          |
    +--------+--------+--------+--------+
    |        |        |        |        |
   PDF     Excel    Word    Database
    |        |        |        |
    +--------+--------+--------+
                |
         Document Loader
         (PDFPlumber, openpyxl, docx)
                |
         Text Extraction
                |
         Chunking Strategy
         (Semantic/Recursive)
                |
    +--------+--------+
    |                 |
 Text Chunks      Table Data
    |                 |
    +--------+--------+
                |
         Embedding Model
         (text-embedding-ada-002)
                |
         Vector DB (Pinecone)
                |
    User Query --> Query Processing
                          |
```

```
                    Retrieval (Top-K)
                          |
                    Context Assembly
                          |
                    LLM (GPT-4)
                          |
                    Response + Citations
```

(b) Chunking Strategies:

1. **Fixed-size chunking**: Simple but may break semantic units
2. **Semantic chunking**: Respects paragraph/section boundaries, better context
3. **Recursive splitting**: Tries multiple separators (paragraphs → sentences → words)
4. **Document-aware chunking**: Preserves table structures and financial metrics together

Trade-offs: Smaller chunks = more precise retrieval but less context; Larger chunks = more context but may include irrelevant info

(c) Handling Numerical Data:

- Extract tables separately using specialized parsers
- Store tables as structured JSON in metadata
- Use hybrid search (keyword + semantic) for numerical queries
- Consider using SQL interface for complex numerical queries

5.5 Section E: Model Selection - ANSWERS

S1: Loan Default Amount Prediction
- **Model**: Gradient Boosting Regressor / XGBoost Regressor
- **Rules**:
1. Continuous target (amount) requires regression
2. XGBoost handles non-linear relationships in financial data
3. Built-in handling for missing values common in loan data

S2: Topic Identification (Topic Modeling)
- **Model**: LDA (Latent Dirichlet Allocation) or BERTopic
- **Rules**:
1. No predefined labels - unsupervised approach needed
2. LDA identifies latent topics from word distributions
3. BERTopic uses transformers for better semantic grouping

S3: Real-time Fraud Detection
- **Model**: Online Learning (SGD Classifier) or Streaming ML
- **Rules**:
1. Model must update incrementally with new data
2. SGD supports partial_fit for online learning
3. Low latency requirements suit linear models

S4: Time-Series Sales Prediction
- **Model**: ARIMA, Prophet, or LSTM

- **Rules**:
1. Temporal dependencies require time-series specific models
2. Prophet handles seasonality and holidays well
3. LSTM captures long-term patterns in sequential data

S5: Spam Classification (Limited Data)
- **Model**: Transfer Learning (Fine-tuned BERT) or Few-shot Learning
- **Rules**:
1. Limited data (100 samples) - pre-trained models essential
2. BERT's language understanding transfers to spam detection
3. Data augmentation can expand training set

S6: Manufacturing Defect Detection
- **Model**: CNN (Convolutional Neural Network) or Transfer Learning (ResNet)
- **Rules**:
1. Image data requires CNN architecture
2. Pre-trained models (ResNet, VGG) work well with limited manufacturing images
3. Can detect visual patterns humans might miss

6 PRACTICE PAPER SET 3

6.1 Section A: JSON Data Debugging (20 marks)

Question 1: Given the following JSON configuration for a machine learning pipeline. Find and fix all issues.

```
{
    "pipeline_config": {
        "model_type": "RandomForest",
        "hyperparameters": {
            "n_estimators": 100,
            "max_depth": None,
            "min_samples_split": "2",
            "random_state": 42.0
        },
        "features": [
            "age"
            "income"
            "credit_score",
        ],
        "target": 'churn',
        "preprocessing": {
            "scaling": true,
            "encoding": "onehot",
            "handle_missing": {
                "strategy": "mean"
                "columns": ["age", "income"]
            }
        }
    },
```

```
    "training_params": {
        "epochs": 50,
        "batch_size": 32,
        "learning_rate": .001,
        "callbacks": [EarlyStopping, ModelCheckpoint]
    }
}
```

Tasks:
(a) Identify all JSON flaws and their potential impacts (10 marks)
(b) Provide the corrected JSON (6 marks)
(c) How would you implement JSON schema validation for this config? (4 marks)

6.2 Section B: SQL Debugging and Normalization (25 marks)

Question 2: Consider the following analytics data table:

LogID	UserID	UserEmail	EventType	EventData	SessionID	DeviceT
1	U001	user1@test.com	click	{"page":"home","button":"signup"}	S001	Mobile
2	U001	user1@test.com	view	home	S001	Mobile
3	NULL	guest@test.com	click	{}	S002	Desktop

Tasks:
(a) Identify all data quality issues and inconsistencies (6 marks)
(b) What anomalies exist in this design? (5 marks)
(c) Design a star schema or normalized schema for analytics (7 marks)
(d) Write SQL to find users with multiple devices and sessions (7 marks)

6.3 Section C: Python Script Debugging (20 marks)

Question 3: Fix the following ML embedding and prediction script:

```
1.  import numpy as np
2.  import pandas as pd
3.  from sklearn.ensemble import RandomForestClassifier
4.  from gensim.models import Word2Vec
5.
6.  def create_embeddings(texts):
7.      # Tokenize texts
8.      tokenized = [text.split() for text in texts]
9.
10.     # Train Word2Vec
11.     model = Word2Vec(tokenized, vector_size=100, window=5)
12.
13.     # Get document embeddings
14.     embeddings = []
```

```
15.      for tokens in tokenized
16.          doc_vec = np.mean([model.wv[token] for token in tokens])
17.          embeddings.append(doc_vec)
18.
19.      return np.array(embeddings)
20.
21. def train_model(X, y):
22.      clf = RandomForestClassifier(n_estimators=100)
23.      clf.fit(X, y)
24.
25.      # Get feature importance
26.      importance = clf.feature_importances
27.      print(f"Top features: {importance.argsort()[-5:]}")
28.
29.      return clf
30.
31. if __name__ == '__main__':
32.      df = pd.read_csv('data.csv')
33.
34.      # Create embeddings
35.      X = create_embeddings(df['text'])
36.      y = df['label']
37.
38.      # Train
39.      model = train_model(X y)
40.
41.      # Predict
42.      predictions = model.predict[X[:10]]
43.      print(f"Predictions: (predictions)")
```

Tasks:
(a) Identify all errors with explanations (12 marks)
(b) Rewrite the corrected script with proper error handling (8 marks)

6.4 Section D: Embedding Demonstration (15 marks)

Question 4: Explain and demonstrate word embeddings.

Tasks:
(a) Explain the difference between Word2Vec (CBOW vs Skip-gram), GloVe, and BERT embeddings (5 marks)
(b) Write Python code to generate embeddings for a set of product descriptions using a pre-trained model (6 marks)
(c) How would you use embeddings for a product similarity search? (4 marks)

6.5 Section E: Model Selection Scenarios (20 marks)

Question 5: Choose appropriate techniques for each scenario with 3 rules each.

Scenario	Description
S1	Predicting employee attrition with highly imbalanced data (5% attrition rate)
S2	Detecting fake reviews among millions of product reviews
S3	Building a chatbot that needs to understand user intent
S4	Predicting stock price direction (up/down) for next day
S5	Identifying objects in satellite imagery for urban planning
S6	Matching job candidates to job postings based on skills

---

7 PRACTICE PAPER SET 3 - ANSWERS

7.1 Section A: JSON Data Debugging - ANSWERS

(a) JSON Flaws Identified:

Issue	Line	Impact
‘None‘ (Python keyword)	hyperparameters	Invalid - use ‘null‘
"2" as string	min_samples_split	Should be integer ‘2‘
‘42.0‘ float for seed	random_state	Works but typically integer
Missing commas	features array	Syntax error
Trailing comma	features array	Invalid JSON
Single quotes	target	Must use double quotes
‘true‘ lowercase	scaling	Valid in JSON, but note case
Missing comma	handle_missing	Between strategy and columns
‘.001‘ without leading zero	learning_rate	Valid but inconsistent
Unquoted values	callbacks array	Variable names not valid

(b) Corrected JSON:

```
{
    "pipeline_config": {
        "model_type": "RandomForest",
        "hyperparameters": {
            "n_estimators": 100,
            "max_depth": null,
            "min_samples_split": 2,
            "random_state": 42
        },
        "features": [
            "age",
            "income",
            "credit_score"
        ],
        "target": "churn",
        "preprocessing": {
            "scaling": true,
            "encoding": "onehot",
            "handle_missing": {
                "strategy": "mean",
```

```
                "columns": ["age", "income"]
            }
        }
    },
    "training_params": {
        "epochs": 50,
        "batch_size": 32,
        "learning_rate": 0.001,
        "callbacks": ["EarlyStopping", "ModelCheckpoint"]
    }
}
```

(c) JSON Schema Validation:

```
from jsonschema import validate

schema = {
    "type": "object",
    "required": ["pipeline_config", "training_params"],
    "properties": {
        "pipeline_config": {
            "type": "object",
            "required": ["model_type", "hyperparameters", "features", "target"],
            "properties": {
                "model_type": {"type": "string", "enum": ["RandomForest", "XGBoost", "LogisticRegression"]},
                "hyperparameters": {
                    "type": "object",
                    "properties": {
                        "n_estimators": {"type": "integer", "minimum": 1},
                        "max_depth": {"type": ["integer", "null"]},
                        "min_samples_split": {"type": "integer", "minimum": 2}
                    }
                },
                "features": {"type": "array", "items": {"type": "string"}},
                "target": {"type": "string"}
            }
        }
    }
}

validate(instance=config_data, schema=schema)
```

7.2 Section B: SQL Normalization - ANSWERS

(a) Data Quality Issues:

1. **Multi-valued column**: DeviceOS has "iOS,Android"
2. **Inconsistent EventData**: JSON object vs plain string
3. **NULL UserID**: Row 3 has NULL with email present
4. **Invalid timestamp**: "Invalid Timestamp" in Row 3
5. **Empty JSON**: EventData "{}" in Row 3
6. **Redundant data**: UserEmail repeated with UserID

(b) Anomalies:

1. **Update Anomaly**: Changing user email requires multiple row updates
2. **Insertion Anomaly**: Can't add user without event
3. **Deletion Anomaly**: Deleting events loses user info
4. **Data Inconsistency**: DeviceOS tied to session but device might change

(c) Star Schema Design:

```
DIMENSION TABLES:
- dim_users(UserID PK, UserEmail, RegistrationDate)
- dim_devices(DeviceID PK, DeviceType, DeviceOS)
- dim_events(EventTypeID PK, EventTypeName)
- dim_time(TimeID PK, Date, Hour, DayOfWeek)

FACT TABLE:
- fact_events(LogID PK, UserID FK, DeviceID FK, EventTypeID FK,
              TimeID FK, SessionID, EventData, Timestamp)
```

(d) SQL Queries:

```sql
-- Find users with multiple devices
SELECT u.UserID, u.UserEmail, COUNT(DISTINCT d.DeviceID) as device_count
FROM fact_events f
JOIN dim_users u ON f.UserID = u.UserID
JOIN dim_devices d ON f.DeviceID = d.DeviceID
GROUP BY u.UserID, u.UserEmail
HAVING COUNT(DISTINCT d.DeviceID) > 1;

-- Find users with multiple sessions
SELECT UserID, COUNT(DISTINCT SessionID) as session_count
FROM fact_events
WHERE UserID IS NOT NULL
GROUP BY UserID
HAVING COUNT(DISTINCT SessionID) > 1;
```

7.3 Section C: Python Script Debugging - ANSWERS

(a) Errors:

Line	Error	Explanation
15	Missing colon	‘for tokens in tokenized‘ needs ‘:‘
16	Missing axis in mean	‘np.mean()‘ on list of arrays needs ‘axis=0‘
16	KeyError possible	Token might not exist in vocabulary
26	Missing parentheses	‘feature_importances‘ is attribute, but needs ‘_‘
39	Missing comma	Between ‘X‘ and ‘y‘
42	Square brackets	‘predict[]‘ should be ‘predict()‘
43	Wrong brackets	‘(predictions)‘ should be ‘{predictions}‘

(b) Corrected Script:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from gensim.models import Word2Vec

def create_embeddings(texts):
    # Tokenize texts
    tokenized = [text.split() for text in texts]

    # Train Word2Vec
    model = Word2Vec(tokenized, vector_size=100, window=5, min_count=1)

    # Get document embeddings
    embeddings = []
    for tokens in tokenized:
        # Filter tokens that exist in vocabulary
        valid_tokens = [token for token in tokens if token in model.wv]
        if valid_tokens:
            doc_vec = np.mean([model.wv[token] for token in valid_tokens], axis
    =0)
        else:
            doc_vec = np.zeros(100)  # Zero vector if no valid tokens
        embeddings.append(doc_vec)

    return np.array(embeddings)

def train_model(X, y):
    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X, y)

    # Get feature importance
    importance = clf.feature_importances_
    print(f"Top features: {importance.argsort()[-5:]}")

    return clf

if __name__ == '__main__':
    df = pd.read_csv('data.csv')

    # Create embeddings
    X = create_embeddings(df['text'])
    y = df['label']

    # Train
    model = train_model(X, y)

    # Predict
    predictions = model.predict(X[:10])
    print(f"Predictions: {predictions}")
```

7.4 Section D: Embedding Demonstration - ANSWERS

(a) Embedding Comparison:

Method	CBOW	Skip-gram	GloVe	BERT
Approach	Predict target from context	Predict context from target	Global co-occurrence matrix	Bidire
Training	Local context window	Local context window	Global statistics	Large
Context	Fixed window	Fixed window	Document-level	Full s
Speed	Faster	Slower	Fast after matrix	Slow,
Best for	Frequent words	Rare words	Analogies	Conte

(b) Product Embedding Code:

```python
from sentence_transformers import SentenceTransformer
import numpy as np

def generate_product_embeddings(descriptions):
    # Load pre-trained model
    model = SentenceTransformer('all-MiniLM-L6-v2')

    # Generate embeddings
    embeddings = model.encode(descriptions, show_progress_bar=True)

    return embeddings

# Example usage
product_descriptions = [
    "Wireless bluetooth headphones with noise cancellation",
    "Premium over-ear headset for gaming",
    "Running shoes with cushioned sole",
    "Athletic sneakers for marathon training"
]

embeddings = generate_product_embeddings(product_descriptions)
print(f"Embedding shape: {embeddings.shape}")  # (4, 384)
```

(c) Product Similarity Search:

```python
from sklearn.metrics.pairwise import cosine_similarity
import faiss

# Build FAISS index for fast similarity search
def build_similarity_index(embeddings):
    dimension = embeddings.shape[1]
    index = faiss.IndexFlatIP(dimension)  # Inner product (cosine after
    normalization)

    # Normalize for cosine similarity
    faiss.normalize_L2(embeddings)
    index.add(embeddings)

    return index

def find_similar_products(query_embedding, index, k=5):
```

```
    faiss.normalize_L2(query_embedding.reshape(1, -1))
    distances, indices = index.search(query_embedding.reshape(1, -1), k)
    return indices[0], distances[0]
```

7.5 Section E: Model Selection - ANSWERS

S1: Imbalanced Employee Attrition
- **Model**: XGBoost with class weights or SMOTE + Random Forest
- **Rules**:
1. Class imbalance (5%) requires balancing techniques
2. `scale_pos_weight` parameter handles imbalance in XGBoost
3. Use F1-score/AUC-ROC instead of accuracy for evaluation

S2: Fake Review Detection
- **Model**: BERT fine-tuned for text classification
- **Rules**:
1. NLP task requires understanding semantic patterns
2. BERT captures contextual language nuances
3. Transfer learning reduces need for large labeled dataset

S3: Intent Classification Chatbot
- **Model**: BERT/RoBERTa or Rasa NLU
- **Rules**:
1. Intent classification is multi-class text problem
2. Pre-trained language models understand diverse phrasings
3. Can handle out-of-vocabulary words through subword tokenization

S4: Stock Direction Prediction
- **Model**: LSTM with technical indicators or Gradient Boosting
- **Rules**:
1. Time-series data with temporal dependencies
2. Binary classification (up/down) suits classification approach
3. Feature engineering crucial (moving averages, RSI, etc.)

S5: Satellite Imagery Object Detection
- **Model**: YOLO or Faster R-CNN
- **Rules**:
1. Object detection requires localization + classification
2. YOLO provides real-time detection capability
3. Transfer learning from ImageNet reduces data requirements

S6: Job-Candidate Matching
- **Model**: Siamese Network or Sentence-BERT
- **Rules**:
1. Similarity matching problem between two text sources
2. Siamese networks learn to compare document pairs
3. Can rank candidates by similarity score

8 PRACTICE PAPER SET 4

8.1 Section A: JSON Data Debugging (20 marks)

Question 1: The following JSON represents a dataset schema for a data science project. Find and correct all issues.

```
{
    "dataset_info": {
        "name": "Customer_Analytics",
        "version": 1.2.0,
        "created_at": new Date(),
        "author": "DS Team"
    },
    "schema": {
        "columns": [
            {
                "name": "customer_id",
                "dtype": "int64",
                "nullable": False,
                "constraints": {unique: true, primary_key: true}
            },
            {
                "name": "revenue",
                "dtype": "float64",
                "nullable": True,
                "default": 0.0,
                "validation": {
                    "min_value": 0,
                    "max_value": Infinity
                }
            },
            {
                "name": "segment"
                "dtype": "category",
                "categories": ["Bronze", "Silver", "Gold", "Platinum",]
            }
        ],
    },
    "transformations": [
        {"step": 1, "operation": "fillna", "params": {"value": None}},
        {"step": 2, "operation": "encode", "params": {method: "label"}}
    ]
}
```

Tasks:
(a) List all JSON errors with their line references and impacts (10 marks)
(b) Provide the fully corrected JSON (6 marks)
(c) Write code to validate JSON against expected schema programmatically (4 marks)

8.2 Section B: SQL Debugging and Normalization (25 marks)

Question 2: Given the following e-commerce orders table:

OrderID	CustomerID	CustomerName	CustomerPhone	ProductID	ProductName	ProductCategory
1001	C1	Amit	9999999999	P1	Laptop	Electronics
1001	C1	Amit	9999999999	P2	Mouse	Electronics
1002	C1	Amit K	8888888888	P1	Laptop	Electronics

Tasks:
(a) Identify ALL data quality and structural issues (7 marks)
(b) Normalize to 3NF with entity-relationship diagram description (8 marks)
(c) Write SQL to create normalized tables with proper constraints (5 marks)
(d) Write SQL query to find customers with multiple shipping addresses (5 marks)

8.3 Section C: Python Script Debugging (20 marks)

Question 3: Debug this data pipeline script:

```
1.  import pandas as pd
2.  import numpy as np
3.  from sklearn.pipeline import Pipeline
4.  from sklearn.impute import SimpleImputer
5.  from sklearn.preprocessing import StandardScaler, OneHotEncoder
6.  from sklearn.compose import ColumnTransformer
7.
8.  def build_preprocessor(numeric_cols, categorical_cols):
9.      numeric_transformer = Pipeline([
10.         ('imputer', SimpleImputer(strategy='median'))
11.         ('scaler', StandardScaler())
12.     ])
13.
14.     categorical_transformer = Pipeline([
15.         ('imputer', SimpleImputer(strategy='constant' fill_value='missing')
    ),
16.         ('encoder', OneHotEncoder(handle_unknown='ignore'))
17.     ])
18.
19.     preprocessor = ColumnTransformer([
20.         transformers(
21.             ('num', numeric_transformer, numeric_cols),
22.             ('cat', categorical_transformer, categorical_cols)
23.         )
24.     ])
25.
26.     return preprocessor
27.
28. def prepare_data(df, target_col):
29.     # Identify column types
30.     numeric_cols = df.select_dtypes(include=[np.number]).columns.toList()
31.     categorical_cols = df.select_dtypes(include=['object']).columns.tolist
    ()
```

```
32.
33.     # Remove target from features
34.     if target_col in numeric_cols
35.         numeric_cols.remove(target_col)
36.
37.     # Build and apply preprocessor
38.     preprocessor = build_preprocessor(numeric_cols, categorical_cols)
39.     X = df.drop(target_col, axis=1)
40.     y = df[target_col]
41.
42.     X_processed = preprocessor.fit_transform(X
43.
44.     return X_processed, y, preprocessor
45.
46. if __name__ == "__main__"
47.     data = pd.read_csv("train.csv")
48.     X, y, prep = prepare_data(data, 'target')
49.     print(f"Processed shape: {X.shape}")
```

Tasks:
(a) Find all errors with line numbers and explanations (12 marks)
(b) Provide the corrected script (8 marks)

8.4 Section D: RAG Workflow Design (15 marks)

Question 4: Design a RAG system for Kelp Global's internal knowledge base that handles financial reports, meeting notes, and investor presentations.

Tasks:
(a) Design the complete RAG pipeline with component specifications (6 marks)
(b) How would you handle document versioning and updates in the RAG system? (4 marks)
(c) Explain retrieval strategies: sparse vs dense retrieval and when to use hybrid (5 marks)

8.5 Section E: Model Selection Scenarios (20 marks)

Question 5: Select the best approach for each scenario with 3 justified rules.

Scenario	Description
S1	Forecasting inventory demand for next 30 days across 1000 products
S2	Identifying which features most impact customer lifetime value
S3	Detecting unusual server behavior from system logs
S4	Translating internal documents from English to Hindi
S5	Automatically extracting key information from contracts
S6	Personalized content ranking on a news feed

9 PRACTICE PAPER SET 4 - ANSWERS

9.1 Section A: JSON Data Debugging - ANSWERS

(a) JSON Errors:

Line	Error	Impact
4	‘1.2.0‘ not quoted	Version should be string ‘"1.2.0"‘
5	‘new Date()‘	JavaScript code not valid in JSON
11	‘False‘ Python keyword	Should be ‘false‘ (lowercase)
12	Unquoted keys	‘{unique: true, primary_key: true}‘ needs quoted keys
16	‘True‘ Python keyword	Should be ‘true‘ (lowercase)
21	‘Infinity‘	Not valid JSON, use very large number or null
24	Missing comma	Between ‘"name": "segment"‘ and next line
25	Trailing comma	In categories array
27	Trailing comma	After columns array
29	‘None‘ Python keyword	Should be ‘null‘
30	Unquoted key	‘method: "label"‘ needs ‘"method"‘

(b) Corrected JSON:

```
{
    "dataset_info": {
        "name": "Customer_Analytics",
        "version": "1.2.0",
        "created_at": "2024-12-02T10:00:00Z",
        "author": "DS Team"
    },
    "schema": {
        "columns": [
            {
                "name": "customer_id",
                "dtype": "int64",
                "nullable": false,
                "constraints": {"unique": true, "primary_key": true}
            },
            {
                "name": "revenue",
                "dtype": "float64",
                "nullable": true,
                "default": 0.0,
                "validation": {
                    "min_value": 0,
                    "max_value": 999999999
                }
            },
            {
                "name": "segment",
                "dtype": "category",
                "categories": ["Bronze", "Silver", "Gold", "Platinum"]
            }
        ]
    },
    "transformations": [
        {"step": 1, "operation": "fillna", "params": {"value": null}},
```

```
        {"step": 2, "operation": "encode", "params": {"method": "label"}}
    ]
}
```

(c) Schema Validation Code:

```
from jsonschema import validate, Draft7Validator

schema = {
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "required": ["dataset_info", "schema"],
    "properties": {
        "dataset_info": {
            "type": "object",
            "required": ["name", "version"],
            "properties": {
                "name": {"type": "string", "pattern": "^[A-Za-z_]+$"},
                "version": {"type": "string", "pattern": "^\d+\.\d+\.\d+$"}
            }
        },
        "schema": {
            "type": "object",
            "properties": {
                "columns": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "required": ["name", "dtype"],
                        "properties": {
                            "name": {"type": "string"},
                            "dtype": {"type": "string", "enum": ["int64", "
    float64", "category", "string"]},
                            "nullable": {"type": "boolean"}
                        }
                    }
                }
            }
        }
    }
}

validator = Draft7Validator(schema)
errors = list(validator.iter_errors(config_data))
for error in errors:
    print(f"Error: {error.message} at {error.path}")
```

9.2 Section B: SQL Normalization - ANSWERS

(a) Data Quality Issues:

1. **Calculated column mismatch**: TotalPrice 99999 2×50000 (should be 100000)
2. **Multi-valued column**: PaymentMethod has "COD, Card"
3. **Inconsistent customer names**: "Amit" vs "Amit K" for same customer
4. **Different phone numbers**: Same customer C1 has different phones
5. **Redundant data**: Customer info repeated in each order

6. **Same OrderID**: Orders 1001 appear twice (multi-line order vs duplication)
7. **No order line number**: Cannot distinguish products in same order

(b) Normalized Schema (3NF):

```
CUSTOMERS (CustomerID PK, CustomerName, CustomerPhone)
    |
    |-- 1:N
    v
ADDRESSES (AddressID PK, CustomerID FK, Address, AddressType)
    |
    |-- N:1
    v
ORDERS (OrderID PK, CustomerID FK, OrderDate, ShippingAddressID FK)
    |
    |-- 1:N
    v
ORDER_ITEMS (OrderID FK, ProductID FK, Quantity, UnitPrice, LineTotal)
              [Composite PK: OrderID, ProductID]
    |
    |-- N:1
    v
PRODUCTS (ProductID PK, ProductName, CategoryID FK)
    |
    |-- N:1
    v
CATEGORIES (CategoryID PK, CategoryName)

PAYMENTS (PaymentID PK, OrderID FK, PaymentMethod, Amount)
```

(c) SQL Creation:

```sql
CREATE TABLE Customers (
    CustomerID VARCHAR(10) PRIMARY KEY,
    CustomerName VARCHAR(100) NOT NULL,
    CustomerPhone VARCHAR(15)
);

CREATE TABLE Addresses (
    AddressID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID VARCHAR(10) NOT NULL,
    Address VARCHAR(200) NOT NULL,
    AddressType VARCHAR(20) DEFAULT 'Shipping',
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY AUTO_INCREMENT,
    CategoryName VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE Products (
    ProductID VARCHAR(10) PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL,
    CategoryID INT,
    UnitPrice DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
```

```sql

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID VARCHAR(10) NOT NULL,
    OrderDate DATE NOT NULL,
    ShippingAddressID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (ShippingAddressID) REFERENCES Addresses(AddressID)
);

CREATE TABLE Order_Items (
    OrderID INT,
    ProductID VARCHAR(10),
    Quantity INT NOT NULL CHECK (Quantity > 0),
    UnitPrice DECIMAL(10,2) NOT NULL,
    LineTotal DECIMAL(12,2) GENERATED ALWAYS AS (Quantity * UnitPrice),
    PRIMARY KEY (OrderID, ProductID),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

(d) Query for Multiple Addresses:

```sql
SELECT c.CustomerID, c.CustomerName, COUNT(DISTINCT a.Address) as address_count
FROM Customers c
JOIN Addresses a ON c.CustomerID = a.CustomerID
GROUP BY c.CustomerID, c.CustomerName
HAVING COUNT(DISTINCT a.Address) > 1;
```

9.3 Section C: Python Script Debugging - ANSWERS

(a) Errors:

Line	Error	Explanation
10-11	Missing comma	Between imputer and scaler in Pipeline list
15	Missing comma	Between ‘’constant’‘ and ‘fill_value‘
19-24	Wrong syntax	‘transformers()‘ is not valid; should be ‘transformers=[]‘
30	Wrong method	‘.toList()‘ should be ‘.tolist()‘ (lowercase)
34	Missing colon	‘if‘ statement needs ‘:‘
42	Missing parenthesis	Unclosed ‘fit_transform(X‘
46	Missing colon	‘if __name__‘ needs ‘:‘

(b) Corrected Script:

```python
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

```python
from sklearn.compose import ColumnTransformer

def build_preprocessor(numeric_cols, categorical_cols):
    numeric_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ])

    categorical_transformer = Pipeline([
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, numeric_cols),
            ('cat', categorical_transformer, categorical_cols)
        ]
    )

    return preprocessor

def prepare_data(df, target_col):
    # Identify column types
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

    # Remove target from features
    if target_col in numeric_cols:
        numeric_cols.remove(target_col)

    # Build and apply preprocessor
    preprocessor = build_preprocessor(numeric_cols, categorical_cols)
    X = df.drop(target_col, axis=1)
    y = df[target_col]

    X_processed = preprocessor.fit_transform(X)

    return X_processed, y, preprocessor

if __name__ == "__main__":
    data = pd.read_csv("train.csv")
    X, y, prep = prepare_data(data, 'target')
    print(f"Processed shape: {X.shape}")
```

9.4 Section D: RAG Workflow - ANSWERS

(a) RAG Pipeline Design:

```
Component Specifications:

1. Document Ingestion Layer:
   - PDF Parser: PyPDF2/PDFPlumber
   - DOCX Parser: python-docx
   - PPT Parser: python-pptx
   - Metadata extraction for versioning

```

```
2. Chunking Strategy:
   - Chunk size: 512 tokens with 50 overlap
   - Respect document sections/headers
   - Keep tables intact

3. Embedding Layer:
   - Model: text-embedding-3-small (OpenAI) or BGE
   - Dimension: 1536/768

4. Vector Store:
   - Primary: Pinecone/Chroma
   - Metadata: document_id, version, date, type

5. Retrieval:
   - Top-K: 5-10 chunks
   - Hybrid: BM25 + Dense retrieval

6. LLM:
   - GPT-4 or LLaMA-2-70B
   - Context window: 8K tokens
```

(b) Document Versioning:

1. **Version metadata**: Store version_id, timestamp with each chunk
2. **Incremental updates**: Re-embed only changed documents
3. **Document lineage**: Maintain parent-child relationships
4. **Time-based retrieval**: Allow queries specific to document versions
5. **Soft delete**: Mark old versions inactive rather than deleting

(c) Retrieval Strategies:

Strategy	How it Works	When to Use
Sparse (BM25)	Keyword matching based on TF-IDF	Exact term matches, specific terminology
Dense	Semantic similarity via embeddings	Conceptual queries, paraphrased questions
Hybrid	Combines both with weighted scoring	Production systems needing both precision and

Use hybrid when: queries mix specific terms with conceptual questions (common in enterprise Q&A).

9.5 Section E: Model Selection - ANSWERS

S1: Multi-Product Inventory Forecasting
- **Model**: LightGBM or Global Time-Series Model (like N-BEATS)
- **Rules**:
1. 1000 products need scalable approach
2. LightGBM handles categorical features (product types) well
3. Can leverage cross-product patterns for new products

S2: Feature Importance for CLV
- **Model**: SHAP with XGBoost or Permutation Importance

- **Rules**:
1. Need interpretable feature rankings
2. SHAP provides local and global explanations
3. Business stakeholders need actionable insights

S3: Server Log Anomaly Detection
- **Model**: Isolation Forest or LSTM Autoencoder
- **Rules**:
1. Unsupervised - normal behavior not fully defined
2. Isolation Forest works with high-dimensional log features
3. LSTM captures temporal patterns in log sequences

S4: English to Hindi Translation
- **Model**: Fine-tuned mBART or IndicTrans2
- **Rules**:
1. Neural machine translation for quality
2. IndicTrans2 specifically trained for Indian languages
3. Can handle domain-specific terminology with fine-tuning

S5: Contract Information Extraction
- **Model**: Named Entity Recognition (NER) + Document AI
- **Rules**:
1. Structured extraction from unstructured text
2. Pre-trained legal NER models available (spaCy, Hugging Face)
3. Layout-aware models handle document structure

S6: Personalized News Ranking
- **Model**: Learning to Rank (LambdaMART) or Two-Tower Model
- **Rules**:
1. User-item interaction requires collaborative approach
2. Two-tower encodes user preferences and content separately
3. Real-time scoring with pre-computed embeddings

10 PRACTICE PAPER SET 5

10.1 Section A: JSON Data Debugging (20 marks)

Question 1: Analyze this RAG system configuration JSON and fix all issues:

```
{
    "rag_config": {
        "embedding_model": {
            "name": "text-embedding-ada-002",
            "dimension": 1536,
            "max_tokens": 8191,
            "batch_size": 100.0
        },
```

```
        "vector_store": {
            "provider": 'pinecone',
            "index_name": "kelp-knowledge-base",
            "metric": "cosine",
            "replicas": 1
            "pods": 1
        },
        "chunking": {
            "strategy": "recursive",
            "chunk_size": 512,
            "chunk_overlap": 50,
            "separators": ["\\n\\n", "\\n", " "],
        },
        "retrieval": {
            "top_k": 5,
            "score_threshold": 0.75,
            "reranker": {
                "enabled": TRUE,
                "model": "cross-encoder/ms-marco-MiniLM-L-12-v2"
            },
            "filters": {
                "doc_type": ["pdf", "docx"],
                "max_age_days": Null
            }
        },
        "llm_config": {
            "model": "gpt-4-turbo",
            "temperature": .7,
            "max_tokens": 2000,
            "system_prompt": "You are a helpful assistant for Kelp Global...",
            "response_format": {"type": json}
        }
    },
    "metadata": {
        "version": "1.0",
        "created_by": "DS Team",
        "last_updated": 2024-12-01
    }
}
```

Tasks:
(a) Find all structural and syntactical errors with explanations (10 marks)
(b) Provide corrected JSON (6 marks)
(c) How would you test this configuration before deployment? (4 marks)

10.2 Section B: SQL Debugging and Normalization (25 marks)

Question 2: Given a data science project tracking table:

ProjectID	ProjectName	TeamMembers	DataSources	ModelTypes	Metrics	Sta
DS001	Churn_Model	Priya,Amit,Raj	CRM,Analytics	LogReg,XGB,RF	AUC:0.85,F1:0.78	Ac
DS001	Churn Model	Priya,Amit	CRM,Analytics	LogReg,XGB	AUC:0.85	Co
DS002	Sales_Forecast	NULL	Sales DB	Prophet	MAPE:12%	Ac

Tasks:
(a) Identify data quality issues, duplicates, and anomalies (6 marks)
(b) Why does this violate normal forms? (5 marks)
(c) Design normalized schema with proper relationships (7 marks)
(d) Write queries for: finding projects with no team, all active projects with their team count (7 marks)

10.3 Section C: Python Script Debugging (20 marks)

Question 3: Fix the following RAG implementation script:

```
1.  from langchain.embeddings import OpenAIEmbeddings
2.  from langchain.vectorstores import Chroma
3.  from langchain.chains import RetrievalQA
4.  from langchain.llms import ChatOpenAI
5.  from langchain.text_splitter import RecursiveCharacterTextSplitter
6.  import os
7.
8.  class RAGPipeline
9.      def __init__(self, api_key):
10.         os.environ['OPENAI_API_KEY'] = api_key
11.         self.embeddings = OpenAIEmbeddings[]
12.         self.vector_store = None
13.         self.qa_chain = none
14.
15.     def ingest_documents(self, documents):
16.         # Split documents
17.         splitter = RecursiveCharacterTextSplitter(
18.             chunk_size=500,
19.             chunk_overlap=50
20.             separators=["\\n\\n", "\\n", " "]
21.         )
22.         chunks = splitter.split_documents(documents)
23.
24.         # Create vector store
25.         self.vector_store = Chroma.from_documents(
26.             documents=chunks
27.             embedding=self.embeddings,
28.             persist_directory="./chroma_db"
29.         )
30.
31.     def setup_qa_chain(self, model_name='gpt-3.5-turbo'):
32.         llm = ChatOpenAI(model_name=model_name, temperature=0)
33.
34.         self.qa_chain = RetrievalQA.from_chain_type(
35.             llm=llm,
36.             chain_type="stuff"
37.             retriever=self.vector_store.as_retriever(search_kwargs={"k":
    5})
38.         )
39.
40.     def query(self, question)
41.         if self.qa_chain == None:
```

```
42.             raise ValueError("QA chain not initialized")
43.
44.         response = self.qa_chain.run[question]
45.         return response
46.
47. if __name__ == "__main__":
48.     rag = RAGPipeline("sk-xxx")
49.     # Load and ingest documents
50.     from langchain.document_loaders import DirectoryLoader
51.     loader = DirectoryLoader("./docs", glob="**/*.pdf")
52.     docs = loader.load()
53.     rag.ingest_documents(docs)
54.     rag.setup_qa_chain
55.     answer = rag.query("What is our revenue target?")
56.     print(answer)
```

Tasks:
(a) Identify all errors with line numbers and reasons (12 marks)
(b) Provide the corrected and improved script (8 marks)

10.4 Section D: Complete RAG + Embedding Task (15 marks)

Question 4: Design an end-to-end RAG system for private equity document analysis at Kelp Global.

Tasks:
(a) Design the complete system architecture diagram (5 marks)
(b) Explain embedding choices: why would you choose one embedding model over another? (5 marks)
(c) Write pseudocode/code for the retrieval and response generation pipeline (5 marks)

10.5 Section E: Model Selection Scenarios (20 marks)

Question 5: Comprehensive scenario-based model selection.

Scenario	Description
S1	Predicting customer segment (Premium, Standard, Basic) based on behavior
S2	Real-time recommendation of next best action for sales team
S3	Extracting and linking entities from financial news articles
S4	Predicting probability of deal closure for private equity investments
S5	Generating automated summaries of lengthy legal documents
S6	Building a semantic search system for internal company wiki

For each scenario: Model selection + 3 Rules + Key preprocessing steps

11 PRACTICE PAPER SET 5 - ANSWERS

11.1 Section A: JSON Data Debugging - ANSWERS

(a) JSON Errors:

Line	Error	Explanation
6	100.0	Should be integer 100 for batch_size
10	Single quotes	'pinecone' should use double quotes
13	Missing comma	After "replicas": 1
20	Trailing comma	After separators array
26	TRUE	JavaScript convention; should be true
31	Null	Wrong case; should be null
34	Leading zero missing	.7 should be 0.7
37	Unquoted value	{"type": json} should be {"type": "json"}
42	Unquoted date	2024-12-01 should be "2024-12-01"

(b) Corrected JSON:

```
{
    "rag_config": {
        "embedding_model": {
            "name": "text-embedding-ada-002",
            "dimension": 1536,
            "max_tokens": 8191,
            "batch_size": 100
        },
        "vector_store": {
            "provider": "pinecone",
            "index_name": "kelp-knowledge-base",
            "metric": "cosine",
            "replicas": 1,
            "pods": 1
        },
        "chunking": {
            "strategy": "recursive",
            "chunk_size": 512,
            "chunk_overlap": 50,
            "separators": ["\\n\\n", "\\n", " "]
        },
        "retrieval": {
            "top_k": 5,
            "score_threshold": 0.75,
            "reranker": {
                "enabled": true,
                "model": "cross-encoder/ms-marco-MiniLM-L-12-v2"
            },
            "filters": {
                "doc_type": ["pdf", "docx"],
                "max_age_days": null
            }
        },
        "llm_config": {
            "model": "gpt-4-turbo",
            "temperature": 0.7,
```

```
            "max_tokens": 2000,
            "system_prompt": "You are a helpful assistant for Kelp Global...",
            "response_format": {"type": "json"}
        }
    },
    "metadata": {
        "version": "1.0",
        "created_by": "DS Team",
        "last_updated": "2024-12-01"
    }
}
```

(c) Testing Strategy:

```
import json
from jsonschema import validate

# 1. Syntax validation
def test_json_syntax(config_path):
    try:
        with open(config_path) as f:
            config = json.load(f)
        return True, config
    except json.JSONDecodeError as e:
        return False, str(e)

# 2. Schema validation
def test_schema(config, schema):
    validate(instance=config, schema=schema)

# 3. Configuration testing
def test_config_values(config):
    assert config['rag_config']['chunking']['chunk_size'] > config['rag_config']['chunking']['chunk_overlap']
    assert config['rag_config']['retrieval']['top_k'] > 0
    assert 0 <= config['rag_config']['llm_config']['temperature'] <= 1

# 4. Integration testing - test with sample query before production
```

11.2 Section B: SQL Normalization - ANSWERS

(a) Data Quality Issues:

1. **Duplicate project**: DS001 appears twice with different data
2. **Multi-valued columns**: TeamMembers, DataSources, ModelTypes, Metrics
3. **Inconsistent naming**: "Churn_Model" vs "Churn Model"
4. **NULL team**: DS002 has no team members
5. **Invalid date**: "TBD" is not a proper date value
6. **Negative budget**: -50000 is invalid
7. **Inconsistent status**: Same project has Active and Completed

(b) Normal Form Violations:

- **1NF**: Multi-valued attributes (TeamMembers, DataSources, etc.)

- **2NF**: Metrics depend on ModelType, not just ProjectID
- **3NF**: Team member details might have transitive dependencies

(c) Normalized Schema:

```
PROJECTS (ProjectID PK, ProjectName, Status, StartDate, EndDate, Budget)
    - CHECK constraint: Budget > 0
    - CHECK constraint: Status IN ('Active', 'Completed', 'On Hold')

TEAM_MEMBERS (MemberID PK, MemberName, Role, Email)

PROJECT_TEAM (ProjectID FK, MemberID FK) - Composite PK

DATA_SOURCES (SourceID PK, SourceName, SourceType)

PROJECT_DATASOURCES (ProjectID FK, SourceID FK)

MODELS (ModelID PK, ProjectID FK, ModelType, ModelVersion)

MODEL_METRICS (MetricID PK, ModelID FK, MetricName, MetricValue, EvaluationDate
    )
```

(d) SQL Queries:

```sql
-- Projects with no team members
SELECT p.ProjectID, p.ProjectName
FROM Projects p
LEFT JOIN Project_Team pt ON p.ProjectID = pt.ProjectID
WHERE pt.MemberID IS NULL;

-- Active projects with team count
SELECT
    p.ProjectID,
    p.ProjectName,
    COUNT(pt.MemberID) as team_count
FROM Projects p
LEFT JOIN Project_Team pt ON p.ProjectID = pt.ProjectID
WHERE p.Status = 'Active'
GROUP BY p.ProjectID, p.ProjectName
ORDER BY team_count DESC;
```

11.3 Section C: Python Script Debugging - ANSWERS

(a) Errors:

Line	Error	Explanation
4	Wrong import	Should be from langchain.chat_models import ChatOpenAI
8	Missing colon	Class definition needs :
11	Square brackets	OpenAIEmbeddings[] should be OpenAIEmbeddings()
13	Wrong keyword	none should be None

Line	Error	Explanation
19-20	Missing comma	Between chunk_overlap and separators
26	Missing comma	Between chunks and embedding
36	Missing comma	After "stuff"
40	Missing colon	Function definition needs :
41	Best practice	Use is None instead of == None
44	Square brackets	run[] should be run()
54	Missing parentheses	setup_qa_chain needs () to call

(b) Corrected Script:

```python
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import Chroma
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI
from langchain.text_splitter import RecursiveCharacterTextSplitter
import os

class RAGPipeline:
    def __init__(self, api_key):
        os.environ['OPENAI_API_KEY'] = api_key
        self.embeddings = OpenAIEmbeddings()
        self.vector_store = None
        self.qa_chain = None

    def ingest_documents(self, documents):
        splitter = RecursiveCharacterTextSplitter(
            chunk_size=500,
            chunk_overlap=50,
            separators=["\n\n", "\n", " "]
        )
        chunks = splitter.split_documents(documents)

        self.vector_store = Chroma.from_documents(
            documents=chunks,
            embedding=self.embeddings,
            persist_directory="./chroma_db"
        )
        self.vector_store.persist()

    def setup_qa_chain(self, model_name='gpt-3.5-turbo'):
        llm = ChatOpenAI(model_name=model_name, temperature=0)

        self.qa_chain = RetrievalQA.from_chain_type(
            llm=llm,
            chain_type="stuff",
            retriever=self.vector_store.as_retriever(search_kwargs={"k": 5})
        )

    def query(self, question):
        if self.qa_chain is None:
            raise ValueError("QA chain not initialized")

        response = self.qa_chain.run(question)
        return response

if __name__ == "__main__":
    rag = RAGPipeline("sk-xxx")
```

```
    from langchain.document_loaders import DirectoryLoader
    loader = DirectoryLoader("./docs", glob="**/*.pdf")
    docs = loader.load()
    rag.ingest_documents(docs)
    rag.setup_qa_chain()
    answer = rag.query("What is our revenue target?")
    print(answer)
```

11.4 Section D: RAG + Embedding System - ANSWERS

(a) System Architecture:

```
KELP GLOBAL RAG SYSTEM ARCHITECTURE

Document Sources --> Document Processor --> Chunking
     |                      |                    |
   PDF/Excel/DOCX    LangChain Loaders    512 tokens
                                                 |
                                                 v
                                      Embedding Layer
                                  (text-embedding-3-small)
                                                 |
                                                 v
                                      Vector Database
                                     (Pinecone/Chroma)
                                                 |
           User Query ------------------------->|
                                                 |
                                                 v
                                Hybrid Retrieval (BM25 + Dense)
                                                 |
                                                 v
                                      Reranker Layer
                                                 |
                                                 v
                                      LLM (GPT-4/Claude)
                                                 |
                                                 v
                                   Response + Citations
```

(b) Embedding Model Selection:

Model	Dimension	Use Case	Trade-off
text-embedding-ada-002	1536	General purpose	Higher cost, larger storage
text-embedding-3-small	1536	Cost-effective	Good balance
BGE-base	768	Open source	No API costs, needs hosting
all-MiniLM-L6-v2	384	Fast, lightweight	Lower quality, faster

Selection Criteria:
1. Domain specificity - Financial domain may benefit from fine-tuned embeddings
2. Latency requirements - Smaller models for real-time applications

3. Cost constraints - Open-source vs commercial
4. Data privacy - Self-hosted for sensitive documents

(c) Pipeline Pseudocode:

```
def rag_query_pipeline(user_query, config):
    # 1. Embed the query
    query_embedding = embedding_model.encode(user_query)

    # 2. Retrieve relevant chunks
    sparse_results = bm25_index.search(user_query, top_k=20)
    dense_results = vector_db.similarity_search(query_embedding, top_k=20)

    # 3. Hybrid fusion
    combined_results = reciprocal_rank_fusion(sparse_results, dense_results)

    # 4. Rerank
    reranked = reranker_model.rerank(query=user_query, documents=
    combined_results[:20])
    top_contexts = reranked[:5]

    # 5. Build prompt with context
    context_text = "\n---\n".join([doc.page_content for doc in top_contexts])

    # 6. Generate response
    response = llm.generate(prompt_with_context, temperature=0.3)

    # 7. Add citations
    return add_source_citations(response, top_contexts)
```

11.5 Section E: Model Selection - ANSWERS

S1: Customer Segment Classification
- **Model**: XGBoost / LightGBM Multi-class Classifier
- **Rules**: Multi-class classification; Tree-based models handle mixed features; Feature importance explains segments
- **Preprocessing**: Handle class imbalance, encode categoricals, scale numerics

S2: Real-time Next Best Action
- **Model**: Multi-Armed Bandit or Reinforcement Learning
- **Rules**: Balance exploration vs exploitation; Real-time with continuous learning; Thompson Sampling handles uncertainty
- **Preprocessing**: Feature engineering for user context, action encoding

S3: Financial Entity Extraction
- **Model**: SpaCy NER + Entity Linking (fine-tuned on financial data)
- **Rules**: NER for extraction; Entity linking to knowledge base; Pre-trained financial NER available
- **Preprocessing**: Text normalization, sentence segmentation

S4: Deal Closure Probability
- **Model**: Logistic Regression or Gradient Boosting with calibration

- **Rules**: Binary classification with probability; Calibrated probabilities for decisions; Feature importance for insights
- **Preprocessing**: Handle missing data, engineer timeline features

S5: Legal Document Summarization
- **Model**: Fine-tuned T5 or LongT5 / GPT-4 with RAG
- **Rules**: Long document handling; Abstractive summarization; Legal terminology preservation
- **Preprocessing**: Document chunking, section identification

S6: Semantic Search for Wiki
- **Model**: Dense Retrieval (Sentence-BERT + FAISS)
- **Rules**: Semantic similarity over keywords; Fast retrieval with ANN; Easy to update
- **Preprocessing**: Document chunking, metadata extraction