**From Moodle**

Outline:  
  
Use Colab (unless you have demonstrated network blocking prevents the use of their source IPs for data harvesting - in which case use jupyter with frequent GitHub commits)  
  
Select some data from the internet, that you need to extract from an API or hidden API or using web scraping such as BeautifulSoup.  
Transform some elements of the data, create new features, clean and prepare the data.  
Load the data into an appropriate database.  
  
Present your work in the last class

**Also refer Assessment Brief**

**Step-by-Step Guide to Start**

1. **Review Your Assignment Requirements**:
   * Ensure you have a clear understanding of the project deliverables, including creating a data acquisition and preprocessing pipeline. This involves acquiring data (via API, web scraping, etc.), transforming features, and loading data into a database.
   * Keep in mind the assessment criteria and the weightage of various components such as implementation (40%), documentation (20%), and presentation (40%).
2. **Outline Your Plan**:
   * **Data Acquisition**: Determine the source of your data (API, web scraping, etc.) and decide on the tools you will use.
   * **Data Transformation**: Plan what preprocessing, transformations, and cleaning steps are required based on the data format and intended use.
   * **Data Loading**: Identify the database where the data will be stored.
3. **Set Up Your Google Colab Notebook**:
   * Since you are already using a shared Colab notebook, structure it to include sections for your project, such as **Introduction**, **Data Acquisition**, **Data Transformation**, **Data Loading**, **Testing**, and **Conclusion/Documentation**.
   * Use text cells to document your process clearly and provide explanations alongside your code.
4. **Select Appropriate Libraries/Tools**:
   * Use popular libraries like requests for API calls, BeautifulSoup for web scraping, pandas for data manipulation, and a database connector library like sqlite3 or SQLAlchemy.
   * Make sure you correctly attribute any resources used.
5. **Start Coding**:
   * Begin with a simple data acquisition example, such as fetching data from an open API or scraping a sample webpage.
   * Gradually move on to data cleaning and transformation. For instance, handling missing values, encoding categorical variables, and scaling numerical features.
   * Load the transformed data into a database.
6. **Testing**:
   * Create unit tests for key transformation functions.
   * Perform at least one integration test to verify that all components of your pipeline (data acquisition, transformation, and loading) work together.
7. **Document Your Work**:
   * Add detailed comments in your code and use text cells to explain each step.
   * Mention any challenges encountered and how you overcame them.
8. **Prepare for Interim Progress Check**:
   * Since this is the 9th week, your interim progress and presentation count for 10%. Share your progress with your professor and seek feedback for improvements.

**Example Outline in Colab:**

markdown

Copy code

# Data Acquisition and Preprocessing Pipeline Project

### Programming for Data Analysis - Module B9DA108

## 1. Introduction

- Brief overview of the project and goals.

## 2. Data Acquisition

- Explain source of data and method used (e.g., API, web scraping).

- Code and results.

## 3. Data Transformation

- Describe data cleaning, feature extraction, and transformations.

- Code and results.

## 4. Data Loading

- Specify database used and structure.

- Code and results.

## 5. Testing

- Unit tests and integration tests.

## 6. Conclusion

- Summary of results and findings.

## 7. Attribution

- List of all external resources, libraries, etc., with proper attribution.

This structure will help you to effectively tackle each component of the project. Let me know if you need more help with a specific part!

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

**Let's Break Down Your Assignment and Plan Your First Steps**

**Understanding the Task**

From the prompt, it seems you're expected to:

1. **Design and Implement a Data Pipeline:** This involves acquiring data from a source (like an API or web scraping), cleaning and transforming it, and storing it in a database.
2. **Test the Pipeline:** Ensure it works as expected by testing individual components and the overall system.
3. **Document the Process:** Clearly explain your design choices, implementation details, and testing results.
4. **Present Your Work:** Deliver a presentation to your class, showcasing your pipeline and addressing technical questions.

**Getting Started: Week 9**

**1. Choose a Data Source and Target Database:**

* **Data Source:** Consider a publicly available API (e.g., from a weather service, stock market, or social media platform) or a website to scrape.
* **Target Database:** A simple SQL database like SQLite or a cloud-based solution like Google Cloud Platform's BigQuery can be suitable.

**2. Set Up Your Google Colab Notebook:**

* **Import Necessary Libraries:** Start by importing libraries like requests for API calls, BeautifulSoup4 for web scraping, pandas for data manipulation, and sqlalchemy for database interactions.
* **Basic Structure:** Outline your notebook with sections for data acquisition, transformation, loading, and testing.

**3. Initial Data Acquisition and Exploration:**

* **API Calls or Web Scraping:** Write code to fetch data from your chosen source.
* **Data Cleaning and Exploration:** Use pandas to clean, filter, and explore the data.

**Example Code Snippet (Assuming a Weather API):**

Python

import requests

import pandas as pd

# Fetch weather data for a specific city

url = "https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR\_API\_KEY"

response = requests.get(url)

data = response.json()

# Convert to pandas DataFrame

df = pd.DataFrame.from\_dict(data, orient='index')

print(df.head())

Use code [with caution.](/faq#coding)

**Tips for Week 9:**

* **Break Down the Task:** Don't try to do everything at once. Focus on one step at a time.
* **Regularly Commit and Push:** Use version control to track your progress and collaborate with your professor.
* **Ask Questions:** Don't hesitate to seek help from your professor or classmates.
* **Document Your Process:** Write clear comments in your code to explain what it does.
* **Visualize Your Data:** Use libraries like Matplotlib or Seaborn to create insightful visualizations.

By following these steps and leveraging the provided resources, you can make significant progress on your assignment in Week 9.

**Would you like to delve deeper into a specific aspect of the project, such as data acquisition, transformation, or database loading?**

To develop your **Data Acquisition and Preprocessing Pipeline**, here's how you can approach each component:

**1. Data Acquisition**

* **Choose a Data Source**: Decide whether you want to use an API, web scraping, or extract data from a database.
  + **API**: If you go for an API, choose an open data API relevant to your project (e.g., weather data, financial data).
  + **Web Scraping**: If you prefer web scraping, select a website and make sure it complies with its terms of service.
  + **DB Extract**: You can also extract data from an existing database using SQL queries.
* **Tools to Use**:
  + **requests** library for API data fetching.
  + **BeautifulSoup** or **Scrapy** for web scraping.
  + **Database connectors** (e.g., sqlite3 or connectors for PostgreSQL, MySQL).
* **Example**:
  + Fetching weather data from an API using requests.
  + Scraping product details from an e-commerce website.

**2. Feature Extraction**

* **Identify Relevant Features**: Determine which columns or data points are important for your analysis.
* **Transform Data**: You might need to:
  + Extract parts of text or numerical data (e.g., split dates into day/month/year).
  + Convert formats (e.g., string to numerical encoding for categories).
  + Handle missing data (impute or remove missing values).

**3. Data Transformation**

* **Cleaning**: Address missing values, handle outliers, remove duplicates, etc.
* **Encoding Categorical Data**: Convert categorical data into a machine-readable format (e.g., one-hot encoding or label encoding).
* **Scaling and Normalization**: Standardize or normalize continuous variables to prepare for analysis.
* **Feature Engineering**: Create new features based on existing data if necessary (e.g., creating an average or sum of certain columns).

**4. Data Loading**

* **Choose a Database**: Select where you will store the cleaned and transformed data (e.g., SQLite, PostgreSQL, etc.).
* **Create and Load Tables**:
  + Establish a connection using a database connector.
  + Create tables as necessary and load data using INSERT statements or bulk loading operations.

**5. Testing and Documentation**

* **Unit Testing**: Create tests for key transformation functions to ensure they work as expected.
* **Integration Testing**: Verify that your entire pipeline works smoothly from data acquisition to loading.
* **Documenting the Pipeline**:
  + Describe each step, the tools used, and why certain transformations were applied.
  + Make sure your Colab notebook contains text cells explaining the logic, code, and results.

**Example Workflow in Colab:**

**python**

**Copy code**

**# Data Acquisition - Fetch data from an API (example)**

**import requests**

**url = 'https://api.example.com/data'**

**response = requests.get(url)**

**data = response.json()**

# Feature Extraction - Select relevant data points

import pandas as pd

df = pd.DataFrame(data)

selected\_features = df[['feature1', 'feature2', 'feature3']]

# Data Transformation - Handle missing values and encode categories

df = selected\_features.fillna(value={'feature1': 0})

df['feature2'] = pd.Categorical(df['feature2']).codes

# Load Data to a Database

import sqlite3

conn = sqlite3.connect('my\_database.db')

df.to\_sql('my\_table', conn, if\_exists='replace', index=False)

conn.close()

By following these steps, you will build a comprehensive data pipeline. Let me know if you need help with any specific part!

~~Transformation~~

*~~Based on the given dataset, here are several opportunities for~~* ***~~data transformations~~*** *~~and feature engineering that can enhance the dataset's usability for analysis or modeling:~~*

### ****~~1. Handling Missing Values~~****

* **~~Impute Missing Prices:~~** ~~Use either the mean, median, or mode to fill missing values in price.previous.value and price.lowestPriceInLast30Days.value.~~

~~python~~

~~Copy code~~

~~df['price.previous.value'] = df['price.previous.value'].fillna(df['price.previous.value'].median())~~

~~df['price.lowestPriceInLast30Days.value'] = df['price.lowestPriceInLast30Days.value'].fillna(df['price.current.value'])~~

* **~~Flag Missing Values:~~** ~~Create a binary column indicating whether a value was imputed.~~

~~python~~

~~Copy code~~

~~df['is\_price\_previous\_missing'] = df['price.previous.value'].isna().astype(int)~~

### ****~~2. Transforming Categorical Data~~****

* **~~Encode brandName as Numeric Codes:~~** ~~Assign each brand a numeric value.~~

~~python~~

~~Copy code~~

~~df['brandName\_encoded'] = pd.Categorical(df['brandName']).codes~~

* **~~One-Hot Encoding for price.currency:~~** ~~Create separate binary columns for each unique currency (useful if there are multiple).~~

~~python~~

~~Copy code~~

~~df = pd.get\_dummies(df, columns=['price.currency'], drop\_first=True)~~

* **~~Label Encoding for isSellingFast and price.isMarkedDown:~~** ~~Convert these boolean features into binary integers.~~

~~python~~

~~Copy code~~

~~df['isSellingFast'] = df['isSellingFast'].astype(int)~~

~~df['price.isMarkedDown'] = df['price.isMarkedDown'].astype(int)~~

### ****~~3. Feature Engineering~~****

#### ****~~Price-Based Features~~****

1. **~~Discount Percentage:~~** ~~Calculate the discount percentage where previous prices are available.~~

~~python~~

~~Copy code~~

~~df['discount\_percent'] = ((df['price.previous.value'] - df['price.current.value']) / df['price.previous.value']) \* 100~~

~~df['discount\_percent'] = df['discount\_percent'].fillna(0)~~

1. **~~Price Drop Flag:~~** ~~Identify products with a lower price compared to the last 30 days.~~

~~python~~

~~Copy code~~

~~df['price\_dropped'] = (df['price.current.value'] < df['price.lowestPriceInLast30Days.value']).astype(int)~~

1. **~~Price Range Indicator:~~** ~~Categorize products into bins based on price.current.value.~~

~~python~~

~~Copy code~~

~~df['price\_range'] = pd.cut(df['price.current.value'], bins=[0, 20, 50, 100], labels=['low', 'medium', 'high'])~~

#### ****~~Text-Based Features~~****

1. **~~Extract Product Type or Keywords from name:~~** ~~Identify and extract product types like "clogs," "heels," or "shoes."~~

~~python~~

~~Copy code~~

~~df['product\_type'] = df['name'].str.extract(r'(clogs|heels|shoes|slingback)', expand=False).fillna('other')~~

1. **~~Text Length or Word Count:~~** ~~Calculate the length of the product names as a potential indicator of descriptive detail.~~

~~python~~

~~Copy code~~

~~df['name\_length'] = df['name'].apply(len)~~

~~df['name\_word\_count'] = df['name'].apply(lambda x: len(x.split()))~~

#### ****~~Brand Popularity~~****

1. **~~Brand Frequency Count:~~** ~~Determine how often each brand appears in the dataset.~~

~~python~~

~~Copy code~~

~~brand\_counts = df['brandName'].value\_counts()~~

~~df['brand\_popularity'] = df['brandName'].map(brand\_counts)~~

### ****~~4. Aggregated Features~~****

* **~~Aggregate by Brand:~~** ~~Calculate average price and discount percentage for each brand.~~

~~python~~

~~Copy code~~

~~brand\_stats = df.groupby('brandName').agg(~~

~~avg\_price=('price.current.value', 'mean'),~~

~~avg\_discount=('discount\_percent', 'mean')~~

~~).reset\_index()~~

~~df = df.merge(brand\_stats, on='brandName', how='left')~~

### ****~~5. Normalization and Scaling~~****

* ~~Normalize or scale the numerical features to make them suitable for machine learning algorithms.~~

~~python~~

~~Copy code~~

~~from sklearn.preprocessing import MinMaxScaler~~

~~scaler = MinMaxScaler()~~

~~df[['price.current.value', 'price.previous.value']] = scaler.fit\_transform(df[['price.current.value', 'price.previous.value']])~~

### ****~~6. Additional Flags~~****

* **~~Luxury Brand Flag:~~** ~~Create a flag for luxury or premium brands based on a predefined list or price thresholds.~~

~~python~~

~~Copy code~~

~~luxury\_brands = ['Crocs', 'Stradivarius']~~

~~df['is\_luxury\_brand'] = df['brandName'].isin(luxury\_brands).astype(int)~~

* **~~New Arrival Indicator:~~** ~~Assume products with isSellingFast = True and no markdown as new arrivals.~~

~~python~~

~~Copy code~~

~~df['is\_new\_arrival'] = ((df['isSellingFast'] == 1) & (df['price.isMarkedDown'] == 0)).astype(int)~~

### ~~Summary of Transformations:~~

* ~~Handle missing values in price.previous.value and price.lowestPriceInLast30Days.value.~~
* ~~Encode brandName, isSellingFast, and price.currency.~~
* ~~Engineer features such as discount percentage, price range, and product type.~~
* ~~Create aggregated brand-level features and popularity indicators.~~
* ~~Scale numerical columns if needed.~~

~~Would you like to implement or explore any specific transformations?~~

4o

##### ~~You said:~~

~~what after encode~~

###### ~~ChatGPT said:~~

~~ChatGPT~~

After encoding categorical data, you can take several next steps depending on the goals of your analysis or machine learning project. Here are the key areas to focus on:

### ****1. Exploratory Data Analysis (EDA)****

Use the encoded data to perform deeper analysis and visualizations:

* **Correlation Analysis:**  
  Check correlations between encoded categorical variables and numerical features.

python

Copy code

corr = df.corr()

sns.heatmap(corr, annot=True, cmap='coolwarm')

plt.show()

* **Compare Categories:**  
  Visualize how encoded features impact other numerical variables.

python

Copy code

sns.boxplot(x='brandName\_encoded', y='price.current.value', data=df)

plt.title('Price Distribution by Brand')

plt.show()

* **Distribution Analysis:**  
  Check the distribution of encoded values.

python

Copy code

sns.countplot(x='brandName\_encoded', data=df)

plt.title('Frequency of Encoded Brands')

plt.show()

### ****2. Feature Engineering on Encoded Data****

* **Interaction Features:**  
  Combine encoded columns to capture relationships between categorical features.

python

Copy code

df['brand\_selling\_fast'] = df['brandName\_encoded'] \* df['isSellingFast']

* **Cluster Encoded Features:**  
  Apply clustering techniques like KMeans to group similar encoded categories based on other attributes.

python

Copy code

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=3)

df['brand\_cluster'] = kmeans.fit\_predict(df[['brandName\_encoded', 'price.current.value']])

* **Feature Importance:**  
  Use feature selection methods to determine the importance of encoded variables.

python

Copy code

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()

model.fit(df[['brandName\_encoded', 'isSellingFast']], df['price.current.value'])

importance = model.feature\_importances\_

print(importance)

### ****3. Prepare for Machine Learning****

* **Splitting the Data:**  
  Split into training and testing sets for supervised learning.

python

Copy code

from sklearn.model\_selection import train\_test\_split

X = df[['brandName\_encoded', 'price.current.value']]

y = df['isSellingFast']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

* **Standardize or Scale Features:**  
  If encoded data is being used alongside numerical features, standardize or scale them.

python

Copy code

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

* **Modeling:**  
  Use machine learning algorithms with encoded data. For example:
  + **Classification (Predict if isSellingFast):**

python

Copy code

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

* + **Regression (Predict price.current.value):**

python

Copy code

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X\_train, y\_train)

### ****4. Clustering or Grouping****

* Use the encoded features to group similar items or customers.
* Example: Grouping brands based on pricing and performance:

python

Copy code

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=5)

df['group'] = kmeans.fit\_predict(df[['brandName\_encoded', 'price.current.value']])

### ****5. Dimensionality Reduction****

If the dataset contains many encoded features or combinations, apply dimensionality reduction to simplify:

* **PCA (Principal Component Analysis):**

python

Copy code

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

reduced\_data = pca.fit\_transform(df[['brandName\_encoded', 'isSellingFast']])

### ****6. Insights and Recommendations****

Use encoded data to draw actionable business insights:

* **Top Performing Brands:** Analyze which brands are most frequently marked as isSellingFast.
* **Price Sensitivity:** Check if products with a higher discount percentage are selling faster.
* **Customer Preferences:** Identify common attributes of products frequently purchased.

Normalization

To normalize the given dataset, consider the type of data in each column and apply the appropriate normalization technique. Here's a breakdown of what can be normalized and how:

### ****Categorical Columns****

1. **id**, **name**, **brandName**, and **colourWayId**:
   * These are **identifiers** or **categorical variables** and typically don’t require normalization for numeric scaling. Instead, you can:
     + Leave them as-is if no further computation is needed.
     + Use **One-Hot Encoding** or **Label Encoding** if these are to be used in machine learning.

### ****Numeric Columns****

The numeric columns can be normalized using appropriate techniques based on their purpose:

#### ****1. Price Columns:****

* **price.current.value**, **price.previous.value**, and **price.lowestPriceInLast30Days.value**:
  + Use **Min-Max Normalization** or **Z-Score Normalization** to bring the values to a uniform scale.  
    Example:

python

Copy code

from sklearn.preprocessing import MinMaxScaler

# Example with Min-Max Scaling

scaler = MinMaxScaler()

df[['price.current.value', 'price.previous.value', 'price.lowestPriceInLast30Days.value']] = scaler.fit\_transform(

df[['price.current.value', 'price.previous.value', 'price.lowestPriceInLast30Days.value']]

)

This rescales prices to a range of [0,1][0, 1][0,1].

#### ****2. Boolean Columns:****

* **isSellingFast**, **price.isMarkedDown**:  
  These are boolean flags (True/False) and can be encoded as binary values (0 for False, 1 for True).

python

Copy code

df['isSellingFast'] = df['isSellingFast'].astype(int)

df['price.isMarkedDown'] = df['price.isMarkedDown'].astype(int)

#### ****3. Mixed or Derived Columns:****

* **productCode**:
  + If numeric, it could be treated as a categorical variable (e.g., encoding it).
  + If it's ordinal or sequential, normalization isn't necessary unless specified.

### ****Non-Normalizable Columns****

1. **name**:
   * This is a text field and doesn't require normalization. It can be processed for text analysis (e.g., tokenization, embedding) if needed.
2. **price.currency**:
   * Currencies don't require normalization but should be verified for consistency (all values are in "EUR").

### ****Normalization Example****

Here’s how you might normalize the dataset in Python using **Pandas** and **Scikit-learn**:

python

Copy code

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

# Sample dataset (replace with actual data)

data = {

'id': [206535097, 206554797, 207319481],

'price.current.value': [44.99, 30.00, 25.99],

'price.previous.value': [44.99, 34.99, 25.99],

'price.lowestPriceInLast30Days.value': [44.99, 26.24, 25.99],

'isSellingFast': [False, True, True],

'price.isMarkedDown': [False, True, False]

}

df = pd.DataFrame(data)

# Boolean to binary

df['isSellingFast'] = df['isSellingFast'].astype(int)

df['price.isMarkedDown'] = df['price.isMarkedDown'].astype(int)

# Min-Max Normalization for price columns

scaler = MinMaxScaler()

df[['price.current.value', 'price.previous.value', 'price.lowestPriceInLast30Days.value']] = scaler.fit\_transform(

df[['price.current.value', 'price.previous.value', 'price.lowestPriceInLast30Days.value']]

)

print(df)

### ****Expected Output After Normalization****

If your original price values were:

* **price.current.value**: [44.99, 30.00, 25.99]
* **price.previous.value**: [44.99, 34.99, 25.99]

They could be normalized to:

* **price.current.value**: [1.0, 0.28, 0.0]
* **price.previous.value**: [1.0, 0.59, 0.0]

### ****Considerations****

* Always check for outliers before applying normalization, as they can distort results.
* Min-Max scaling is sensitive to outliers; use **RobustScaler** or **Log Scaling** if data has extreme values.
* Ensure consistent handling of currencies if you plan to include datasets with different currencies.

4o

Top of Form



Bottom of Form

Second Para

The second paragraph outlines the expectations for the **implementation and testing** of your data acquisition and preprocessing pipeline. Here's a detailed breakdown:

**Implementation and Testing Details**

1. **Pipeline Development and Implementation**
   * **Design and Build:**  
     You are required to create a functioning data pipeline that handles:
     + **Data Acquisition:** This can be done using any method you choose, such as:
       - API calls (e.g., REST APIs for live data fetching).
       - Web scraping (for gathering data from websites).
       - Extracting data from databases (using SQL queries or similar).
     + **Feature Extraction and Transformation:** Once data is acquired, you must:
       - Identify key features from the dataset that are relevant to your analysis or purpose.
       - Apply transformations such as normalization, encoding categorical variables, cleaning missing data, etc., as needed.
     + **Loading:** Store the processed data into a database (e.g., SQL, NoSQL, or any other appropriate storage format).
   * **Choice of Complexity:**  
     You have the freedom to decide how complex the pipeline should be. For instance:
     + A simple pipeline might fetch data from an API, clean it, and load it into a database.
     + A more complex pipeline could include multiple stages like scheduling, parallel processing, or machine learning feature engineering.
2. **Testing Requirements**
   * **Unit Tests:**
     + Each transformation function in your pipeline should have unit tests. This ensures that every individual function works as expected.
     + Example: If you have a function that normalizes data, write tests to validate its correctness for various input cases (e.g., edge cases like null values, extremely large numbers, etc.).
   * **Integration Test:**
     + Test the interaction between the **frontend** and **backend** of your pipeline.
     + Example: Validate whether the data fetched from the API (frontend) is correctly processed by the backend and stored into the database.
     + This test ensures that the components of your pipeline work seamlessly when integrated.
3. **Documentation**
   * Your implementation needs to be **thoroughly documented.** This includes:
     + Explaining each part of the pipeline, including design decisions.
     + Clearly describing how the data flows through acquisition, transformation, and loading stages.
     + Instructions for testing, such as how to run the unit and integration tests.
4. **Testing Scenarios**
   * Consider edge cases and error handling in your tests. For example:
     + How does the pipeline handle an API returning incomplete data?
     + What happens if the database connection is interrupted?

Next

I want create like front end backend using this api data and using github and azure

###### ChatGPT said:

ChatGPT

Creating a front-end and back-end system using the database API data and deploying it using GitHub and Azure is a fantastic project! Here's a step-by-step guide:

## ****1. Plan Your Project****

* **Backend**: Use a web framework like Flask (Python), FastAPI, or Django to serve data from your SQLite database as an API.
* **Frontend**: Use a modern front-end framework like React, Vue.js, or Angular for a dynamic user interface.
* **Deployment**: Deploy your backend and frontend to Azure, with version control via GitHub.

## ****2. Backend: Create API to Serve Data****

You’ll expose your database data via a RESTful API.

### ****Setup a Flask Backend****

1. **Install Flask**:

bash

Copy code

pip install flask flask-cors

1. **Create a Backend Project Structure**:

bash

Copy code

backend/

├── app.py # Flask app

├── database.db # SQLite database (e.g., asos.db)

├── requirements.txt # Dependencies for deployment

1. **Code the Flask App (app.py)**:

python

Copy code

from flask import Flask, jsonify

import sqlite3

app = Flask(\_\_name\_\_)

# API endpoint to fetch all data

@app.route('/data', methods=['GET'])

def get\_data():

conn = sqlite3.connect('asos.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM asos\_data")

rows = cursor.fetchall()

conn.close()

# Transform rows to JSON-like list

columns = ["col1", "col2", "col3"] # Replace with actual column names

data = [dict(zip(columns, row)) for row in rows]

return jsonify(data)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

1. **Test the API**:
   * Run the backend:

bash

Copy code

python app.py

* + Open your browser at http://127.0.0.1:5000/data to see the data served as JSON.

1. **Save Dependencies**:

bash

Copy code

pip freeze > requirements.txt

## ****3. Frontend: Build the UI****

### ****Setup a React Frontend****

1. **Install React**:

bash

Copy code

npx create-react-app frontend

cd frontend

npm start

1. **Fetch and Display Data**: Update App.js to fetch data from your Flask API:

javascript

Copy code

import React, { useEffect, useState } from "react";

function App() {

const [data, setData] = useState([]);

useEffect(() => {

fetch("http://127.0.0.1:5000/data")

.then((response) => response.json())

.then((data) => setData(data));

}, []);

return (

<div>

<h1>ASOS Data</h1>

<ul>

{data.map((item, index) => (

<li key={index}>

{item.col1} - {item.col2}

</li>

))}

</ul>

</div>

);

}

export default App;

1. **Test the Frontend**:
   * Run the frontend:

bash

Copy code

npm start

* + Ensure it fetches data from your backend.

## ****4. Version Control with GitHub****

1. **Initialize Git**:

bash

Copy code

git init

git add .

git commit -m "Initial commit"

1. **Create a GitHub Repository**:
   * Go to [GitHub](https://github.com/), create a new repository, and push your code:

bash

Copy code

git remote add origin https://github.com/your-username/your-repo.git

git branch -M main

git push -u origin main

## ****5. Deploy to Azure****

Azure supports deployment of both backend and frontend. Here's how:

### ****Backend: Deploy Flask App to Azure****

1. **Install Azure CLI**:
   * Follow the instructions [here](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli).
2. **Login to Azure**:

bash

Copy code

az login

1. **Create a Python Web App**:

bash

Copy code

az webapp up --name your-app-name --resource-group your-resource-group --runtime "PYTHON:3.9"

1. **Push Your Flask Backend**: Azure will automatically deploy the backend from your repository when connected.

### ****Frontend: Deploy React App to Azure****

1. **Build the Frontend**:

bash

Copy code

npm run build

1. **Deploy React with Azure Static Web Apps**:
   * Create an Azure Static Web App:

bash

Copy code

az staticwebapp create \

--resource-group your-resource-group \

--name your-frontend-name \

--source ./build \

--branch main \

--token your-github-token

## ****6. Connect Frontend and Backend****

* Update your React app's API endpoint to point to the deployed backend's URL.
* For example:

javascript

Copy code

fetch("https://your-app-name.azurewebsites.net/data")

## ****7. Monitor and Maintain****

* Use Azure's monitoring tools to track application performance.
* Keep your GitHub repository updated with new commits for continuous deployment.

### Let me know if you'd like help with a specific part! 😊