Lesson 2: **Data Collection and Management**

**Understanding data sources, data structures, and file formats is crucial for effective data analytics.**

Each data source and structure has its own characteristics, strengths, and weaknesses, influencing how you interact with and analyze the data. Here’s a comprehensive overview:

Data Wrangling: Techniques for collecting, cleaning, and preparing data for analysis.

* Data wrangling, also known as data munging, is the process of transforming raw data into a more usable format. It involves a series of steps aimed at cleaning, restructuring, and enriching the data to make it suitable for analysis or operational use.
  + The goal of data wrangling *is to prepare data so that it can be effectively analyzed and used to make informed decisions.*

**1. Data Sources and Structures**

**Relational Databases**

* **Description**: Store data in tables with rows and columns. Relationships are defined between tables using primary and foreign keys.
* **Example**: MySQL, PostgreSQL
  + **Table Example**:

sql

Copy code

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(50),

Salary DECIMAL(10, 2)

);

* + **Query Example**:

sql

Copy code

SELECT Name, Salary FROM Employees WHERE Department = 'Sales';

**NoSQL Databases**

* **Description**: Designed for unstructured or semi-structured data. Can be document-based, key-value stores, wide-column stores, or graph databases.
* **Examples**:
  + **Document-Based**: MongoDB
    - **Document Example**:

json

Copy code

{

"EmployeeID": 1,

"Name": "Alice",

"Department": "Sales",

"Salary": 75000

}

* + **Key-Value Store**: Redis ()
  + **REmote DIctionary Server***. It is an open-source, in-memory data structure store that can be used as a database, cache, and message broker. Redis supports various data structures such as strings, hashes, lists, sets, and more.*
    - **Example**:

python

Copy code

redis\_client.set('employee:1', '{"Name": "Alice", "Department": "Sales", "Salary": 75000}')

* + **Wide-Column Store**: Cassandra
    - **Example**:

cql

Copy code

CREATE TABLE Employees (

EmployeeID UUID PRIMARY KEY,

Name TEXT,

Department TEXT,

Salary DECIMAL

);

* + **Graph Database**: Neo4j
    - **Example**:

cypher

Copy code

CREATE (e:Employee {EmployeeID: 1, Name: 'Alice', Department: 'Sales', Salary: 75000})

**Data Warehouses**

* **Description**: Centralized repositories that store and analyze large volumes of historical data from multiple sources.
* **Examples**: Amazon Redshift, Google BigQuery
  + **Query Example (SQL)**:

sql

Copy code

SELECT Department, AVG(Salary) AS AverageSalary

FROM Employees

GROUP BY Department;

**APIs**

* **Description**: Allow interaction with external systems or services to fetch or send data.
* **Examples**:
  + **Twitter API**: Retrieve tweets or user data.
    - **Python Example**:

python

Copy code

import tweepy

auth = tweepy.OAuth1UserHandler(consumer\_key, consumer\_secret, access\_token, access\_token\_secret)

api = tweepy.API(auth)

tweets = api.user\_timeline(screen\_name='twitter\_user', count=10)

for tweet in tweets:

print(tweet.text)

* + **Google Maps API**: Retrieve location data.
    - **Python Example**:

python

Copy code

import requests

response = requests.get(f'https://maps.googleapis.com/maps/api/geocode/json?address=New+York&key=YOUR\_API\_KEY')

data = response.json()

print(data)

**Web Scraping**

* **Description**: Extract data from websites using libraries.
* **Examples**:
  + **BeautifulSoup Example**:

python

Copy code

from bs4 import BeautifulSoup

import requests

response = requests.get('https://example.com')

soup = BeautifulSoup(response.text, 'html.parser')

titles = soup.find\_all('h1')

for title in titles:

print(title.text)

**Spreadsheets**

* **Description**: Used for small to moderate amounts of data, easily manipulated in tools like Excel or Google Sheets.
* **Examples**:
  + **Excel File**: .xlsx
    - **Python Example**:

python

Copy code

import pandas as pd

df = pd.read\_excel('data.xlsx')

print(df.head())

**2. File Formats**

**CSV (Comma-Separated Values)**

* **Description**: Plain text file where each line represents a row of data, and columns are separated by commas.
* **Example**:
  + **File Content**:

Copy code

EmployeeID,Name,Department,Salary

1,Alice,Sales,75000

2,Bob,Marketing,65000

* + **Python Example**:

python

Copy code

import pandas as pd

df = pd.read\_csv('data.csv')

print(df.head())

**JSON (JavaScript Object Notation)**

* **Description**: Text format using key-value pairs, supports nested data structures.
* **Example**:
  + **File Content**:

json

Copy code

{

"employees": [

{"EmployeeID": 1, "Name": "Alice", "Department": "Sales", "Salary": 75000},

{"EmployeeID": 2, "Name": "Bob", "Department": "Marketing", "Salary": 65000}

]

}

* + **Python Example**:

python

Copy code

import json

with open('data.json') as f:

data = json.load(f)

print(data)

**XML (eXtensible Markup Language)**

* **Description**: Text format using tags to define data elements. Supports hierarchical data.
* **Example**:
  + **File Content**:

xml

Copy code

<employees>

<employee>

<EmployeeID>1</EmployeeID>

<Name>Alice</Name>

<Department>Sales</Department>

<Salary>75000</Salary>

</employee>

<employee>

<EmployeeID>2</EmployeeID>

<Name>Bob</Name>

<Department>Marketing</Department>

<Salary>65000</Salary>

</employee>

</employees>

* + **Python Example**:

python

Copy code

import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')

root = tree.getroot()

for employee in root.findall('employee'):

name = employee.find('Name').text

print(name)

**Parquet**

* **Description**: Columnar storage file format designed for efficient data processing.
* **Example**:
  + **Python Example**:

python

Copy code

import pandas as pd

df = pd.read\_parquet('data.parquet')

print(df.head())

**Avro**

* **Description**: Row-based storage format with schema support.
* **Example**:
  + **Python Example**:

python

Copy code

import fastavro

with open('data.avro', 'r') as f:

reader = fastavro.reader(f)

for record in reader:

print(record)

**HDF5 (Hierarchical Data Format version 5)**

* **Description**: File format designed to store and organize large amounts of data with complex relationships.
* **Example**:
  + **Python Example**:

python

Copy code

import h5py

with h5py.File('data.h5', 'r') as f:

print(list(f.keys())) # Print all top-level keys

data = f['dataset\_name'][:]

print(data)

**Choosing the Right Format and Structure**

* **Data Complexity**: Use JSON or XML for hierarchical data, CSV for flat data.
* **Data Size**: For large datasets, consider Parquet or HDF5.
* **Data Use Case**: Use databases for transactional data, APIs for real-time data, and spreadsheets for manual data management.

Data wrangling typically involves:

1. **Data Collection**

* Definition: Gathering raw data from various sources.
* Examples: Reading from files (CSV, Excel), databases, web scraping, APIs, and more.

2. **Data Cleaning**

* Definition: Identifying and correcting errors or inconsistencies in the data.
* Tasks:
  + Handling Missing Values: Filling in, interpolating, or removing missing data.
  + Removing Duplicates: Identifying and removing duplicate records.
  + Correcting Errors: Fixing typos or inconsistencies in the data.
  + Standardizing Formats: Ensuring consistency in data formats, such as date and time formats.

3. **Data Transformation**

* Definition: Converting data into a format that is more appropriate for analysis.
* Tasks:
  + Normalization: Scaling data to fit within a specific range.
  + Encoding: Converting categorical variables into numerical format (e.g., one-hot encoding).
  + Aggregation: Summarizing data, such as calculating averages or totals.
  + Merging/Joining: Combining data from different sources or tables.

4. **Data Enrichment**

* Definition: Enhancing the dataset by adding new information or features.
* Tasks:
  + Feature Engineering: Creating new features from existing data to improve analysis.
  + Data Integration: Combining data from multiple sources to provide a more comprehensive view.

5. **Data Structuring**

* Definition: Organizing data into a format that makes it easier to analyze.
* Tasks:
  + **Reshaping Data:** Pivoting, melting, or transposing data tables.
  + **Filtering:** Selecting relevant rows or columns based on certain criteria.

**Why Data Wrangling is Important**

1. Accuracy: Ensures that the data used for analysis is accurate and reliable.
2. Efficiency: Streamlines the data to make it easier and faster to analyze.
3. Consistency: Standardizes data formats and structures to prevent errors in analysis.
4. Insightful Analysis: Prepares data in a way that facilitates meaningful analysis and decision-making.

Data Wrangling: Techniques for Collecting, Cleaning, and Preparing Data for Analysis with Python

Case Study Example: Analyzing a Sales Dataset

Step 1: Collecting Data

Let's assume we have a CSV file named sales\_data.csv with the following content: use excel and save using CSV format.

OrderID,Product,Quantity,Price,Date

1,Widget A,4,20.00,2023-01-01

2,Widget B,2,15.00,2023-01-01

3,Widget A,1,20.00,2023-01-02

4,Widget C,5,30.00,2023-01-03

5,Widget B,,15.00,2023-01-04

6,Widget A,3,20.00,2023-01-05

7,,1,15.00,2023-01-06

8,Widget C,2,30.00,2023-01-07

Step 2: Loading the Data

import pandas as pd

# Load the data

data = pd.read\_csv('sales\_data.csv')

print("Initial Data:\n", data)

Output

Initial Data:

OrderID Product Quantity Price Date

0 1 Widget A 4.0 20.0 2023-01-01

1 2 Widget B 2.0 15.0 2023-01-01

2 3 Widget A 1.0 20.0 2023-01-02

3 4 Widget C 5.0 30.0 2023-01-03

4 5 Widget B NaN 15.0 2023-01-04

5 6 Widget A 3.0 20.0 2023-01-05

6 7 NaN 1.0 15.0 2023-01-06

7 8 Widget C 2.0 30.0 2023-01-07

Step 3: Cleaning the Data

1. Handling Missing Values:

# Check for missing values

print("Missing Values:\n", data.isnull().sum())

# Fill missing Quantity with 0 and drop rows with missing Product

data['Quantity'].fillna(0, inplace=True)

data.dropna(subset=['Product'], inplace=True)

print("Data after Handling Missing Values:\n", data)

Output

Missing Values:

OrderID 0

Product 1

Quantity 1

Price 0

Date 0

dtype: int64

Data after Handling Missing Values:

OrderID Product Quantity Price Date

0 1 Widget A 4.0 20.0 2023-01-01

1 2 Widget B 2.0 15.0 2023-01-01

2 3 Widget A 1.0 20.0 2023-01-02

3 4 Widget C 5.0 30.0 2023-01-03

5 6 Widget A 3.0 20.0 2023-01-05

7 8 Widget C 2.0 30.0 2023-01-07

1. Correcting Data Types:

# Convert 'Date' to datetime format

data['Date'] = pd.to\_datetime(data['Date'])#*use date to represent dates without time, and datetime to represent a specific point in time inclusive of both date and time.*

# Convert 'Price' to numeric

data['Price'] = pd.to\_numeric(data['Price'])

print("Data Types After Conversion:\n", data.dtypes)

Output

Data Types After Conversion:

OrderID int64

Product object

Quantity float64

Price float64

Date datetime64[ns]

dtype: object

1. Removing Duplicates:

# Remove duplicates

data.drop\_duplicates(inplace=True)# *we want the output of our modified data frame to essentially override the original data frame.*

print("Data after Removing Duplicates:\n", data)

Output:

Data after Removing Duplicates:

OrderID Product Quantity Price Date

0 1 Widget A 4.0 20.0 2023-01-01

1 2 Widget B 2.0 15.0 2023-01-01

2 3 Widget A 1.0 20.0 2023-01-02

3 4 Widget C 5.0 30.0 2023-01-03

5 6 Widget A 3.0 20.0 2023-01-05

7 8 Widget C 2.0 30.0 2023-01-07

Step 4: Preparing the Data

1. Creating New Columns:

# Calculate total sales

data['TotalSales'] = data['Quantity'] \* data['Price']

print("Data with Total Sales:\n", data)

Output

Data with Total Sales:

OrderID Product Quantity Price Date TotalSales

0 1 Widget A 4.0 20.0 2023-01-01 80.0

1 2 Widget B 2.0 15.0 2023-01-01 30.0

2 3 Widget A 1.0 20.0 2023-01-02 20.0

3 4 Widget C 5.0 30.0 2023-01-03 150.0

5 6 Widget A 3.0 20.0 2023-01-05 60.0

7 8 Widget C 2.0 30.0 2023-01-07 60.0

1. Filtering Data:

# Filter sales for Widget A

widget\_a\_sales = data[data['Product'] == 'Widget A']

print("Sales for Widget A:\n", widget\_a\_sales)

Output

Sales for Widget A:

OrderID Product Quantity Price Date TotalSales

0 1 Widget A 4.0 20.0 2023-01-01 80.0

2 3 Widget A 1.0 20.0 2023-01-02 20.0

5 6 Widget A 3.0 20.0 2023-01-05 60.0

1. Grouping Data:

# Group by Product and sum TotalSales

summary = data.groupby('Product')['TotalSales'].sum().reset\_index()#*Generate a new DataFrame or Series with the index reset.*

print("Sales Summary by Product:\n", summary)

Output

Sales Summary by Product:

Product TotalSales

0 Widget A 160.0

1 Widget B 30.0

2 Widget C 210.0

Step 5: Visualizing the Data

import matplotlib.pyplot as plt

import seaborn as sns

# Bar plot for total sales by product

plt.figure(figsize=(10, 6))

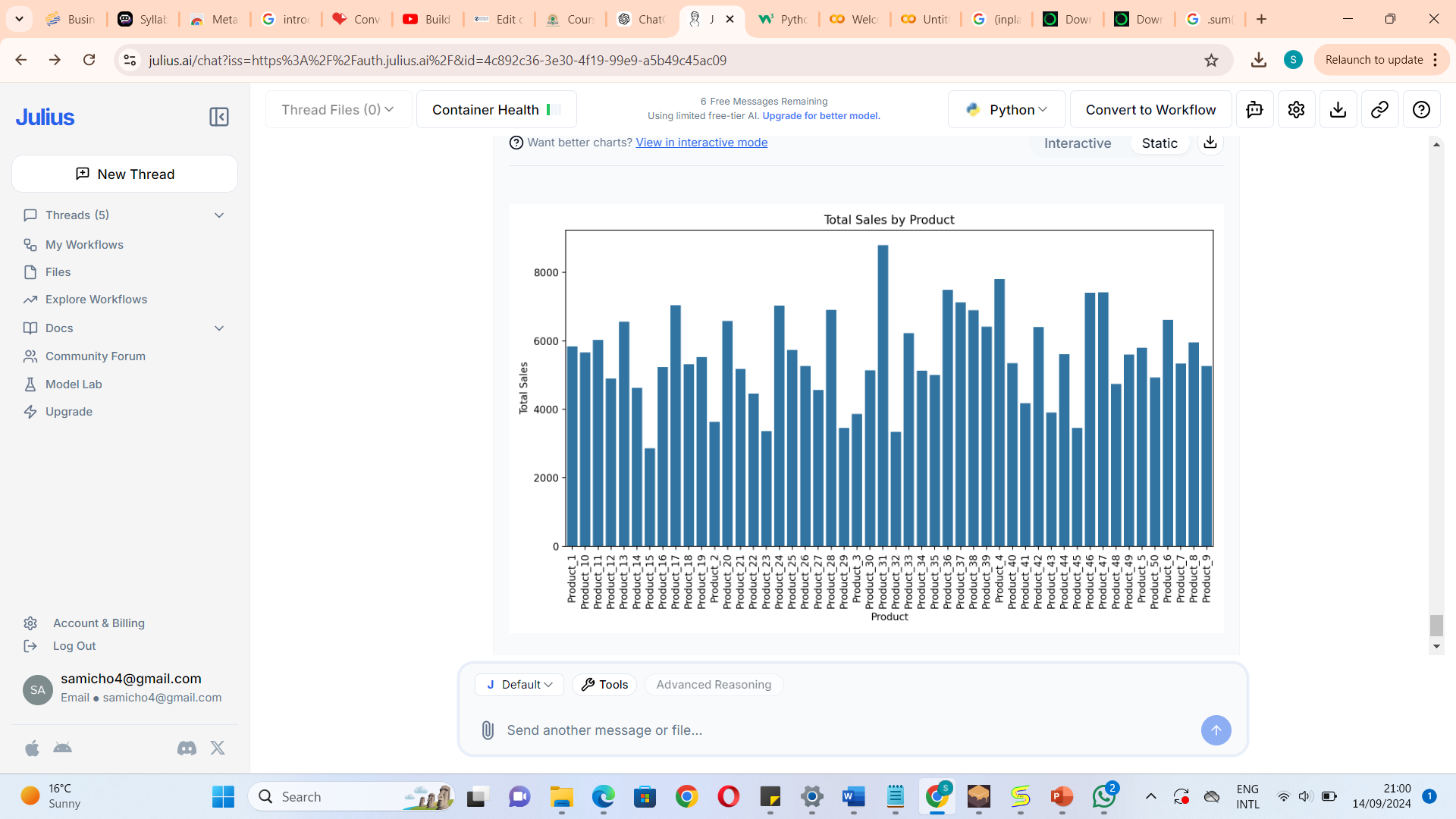
sns.barplot(data=summary, x='Product', y='TotalSales')

plt.title('Total Sales by Product')

plt.xlabel('Product')

plt.ylabel('Total Sales')

plt.show()



**Let's analyse the plot further to draw insights.**

The bar plot displays the total sales for each product, with the x-axis representing the product names and the y-axis representing the total sales amount. Here are some insights from the plot:

1. Top Performers: A few products stand out with significantly higher sales compared to others. These products are likely the most popular or have higher price points contributing to their total sales.
2. Sales Distribution: The sales distribution is uneven, with a small number of products generating a large portion of the total sales. This indicates a potential focus on these high-performing products for marketing and inventory management.
3. Opportunities for Growth: Products with lower sales might represent opportunities for growth through targeted promotions or product improvements.

**Conclusion**

In this case study, we demonstrated data wrangling techniques using Python to collect, clean, and prepare a sales dataset for analysis. The steps included:

1. **Data Collection:** Loading data from a CSV file.
2. **Data Cleaning:** Handling missing values, correcting data types, and removing duplicates.
3. **Data Preparation:** Creating new columns, filtering data, and aggregating results.
4. **Data Visualization:** Creating plots to visualize the results.

**Revision questions in data wrangling**

1. **How do you load a CSV file into a Pandas DataFrame?**

**Answer**:

import pandas as pd

df = pd.read\_csv('file.csv')

2.**How can you display the first 5 rows of a DataFrame?**

**Answer**:

print(df.head())

1. **How can you get the summary statistics of a DataFrame?**

print(df.describe())

**5.How do you check for missing values in a DataFrame?**

print(df.isnull().sum())

**Data Cleaning**

1. **How do you fill missing values in a DataFrame column with the mean of that column?**

df['column\_name'].fillna(df['column\_name'].mean(), inplace=True)

1. **How can you drop rows with any missing values from a DataFrame?**

df.dropna(inplace=True)

df.rename(columns={'old\_name': 'new\_name'}, inplace=True)

1. **Data Transformation**

How do you create a new column in a DataFrame that is a result of an operation on existing columns?......**solution?**

1. How can you apply multiple aggregation functions to a DataFrame grouped by a column?....solution?