VU VRIJE UNIVERSITEIT AMSTERDAM

WINTER SCHOOL ON THE USE OF GENERATIVE AI IN ACADEMIA

NETWORK INSTITUTE

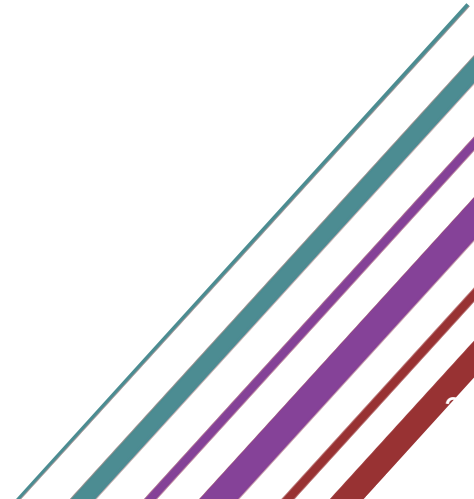
GenAI for Data Cleaning, Generation, and Analysis

Dr. Jiahuan (Joanne) Pei
Computer Science Department
J.PEI2@VU.NL

Agenda

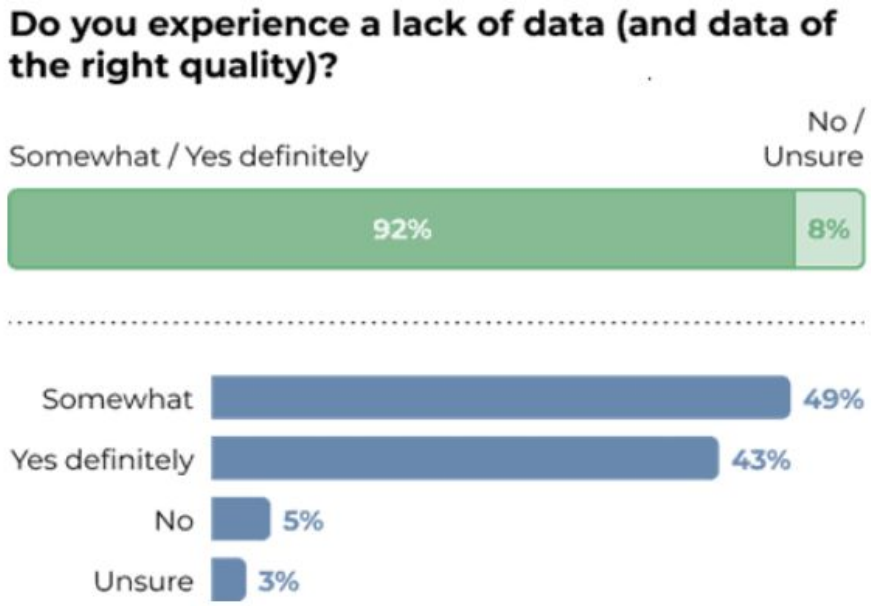
- GenAI in the Data Lifecycle – Foundations and Prompting
- Data Cleaning and Generation with GenAI
- Data Analysis and Interpretation with GenAI
- GenAI-based Workflow: Integrating GenAI Into Data Pipeline
- The State-Of-The-Art Research

GenAI in the Data Lifecycle
Foundations and Prompting

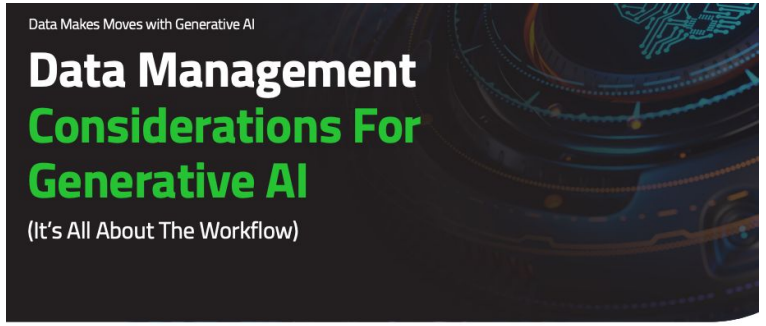


Why Data Management Matters in Generative AI?

- **Data quantity: two examples with statistics**



Source: [1]



It's the next big thing, newly embedded in every business model – from high-tech to home-grown. With the ability to generate human-like text, images, and even code, generative AI opens up a wide range of applications from content generation to natural language processing.

69% **Businesses using AI/ML to generate revenue streams.***

Generative AI became a household name thanks to widely available cutting-edge AI models like GPT-3 and its successors. Now every business must consider the upside of deploying GAI for insights, optimized efficiency, and new revenue streams. The success of any generative AI hinges not only on the quality and quantity of available data, but even more so on how easily that data can move into, and out of the model. For many projects, the agility with which a proprietary model handles data will be the make-or-break factor.

PROBLEM:

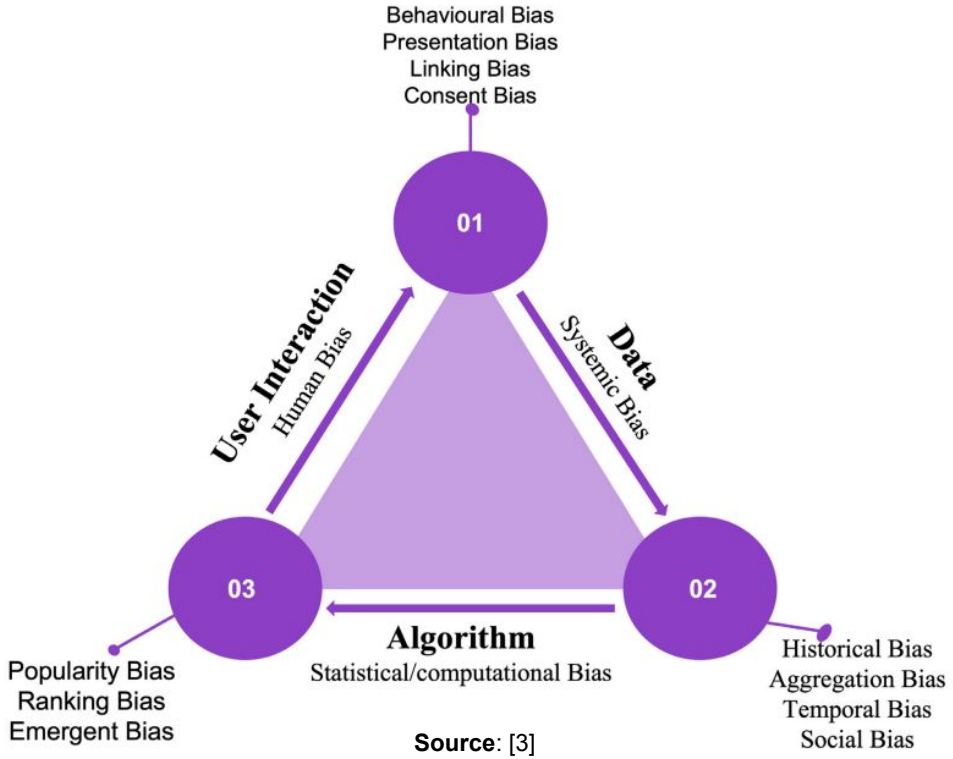
Most Data Architectures Are Not Prepared for Generative AI

The data that serves as the backbone of model training takes many forms. A given company's training data can consist of millions or billions of differently sized files, from structured or unstructured text to images and audio. Files may also be spread across a wide range of locations, including cloud, on-prem, or object storage, not to mention held in different data centers around the world.

Source: [2]

Why Data Management Matters in Generative AI?

Data quality: It provides the **foundation** of *high-quality*, *reliable*, and *secure* data needed for models to function effectively and avoid generating incorrect or biased information (hallucinations).

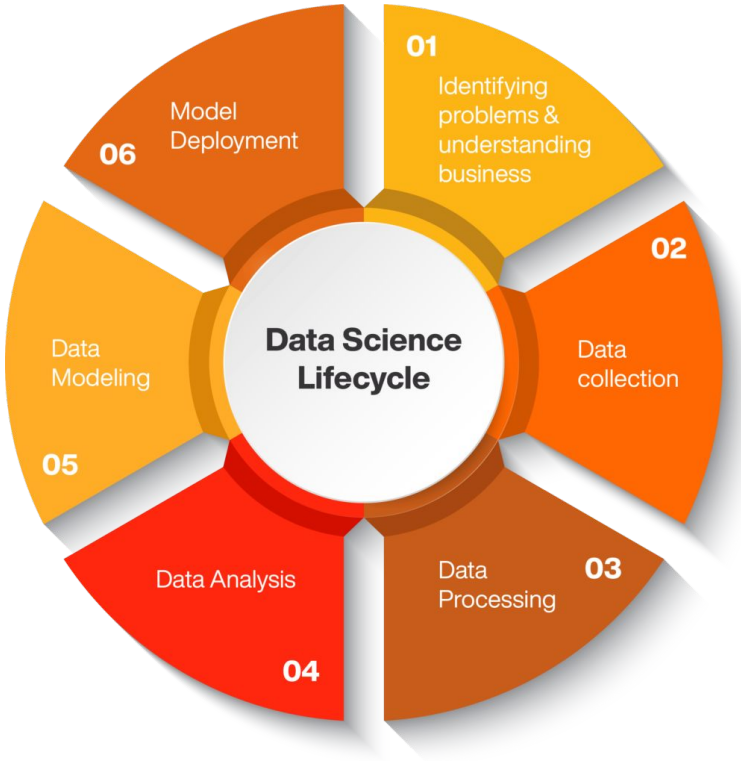


Source: [3]

- **User Interaction (01):** Human biases (behavioral, presentation, linking, consent) influence how users interact with systems, generating biased data.
- **Data (02):** Systemic biases, such as historical, aggregation, temporal, and social biases, become embedded in the collected data.
- **Algorithm (03):** Statistical and computational biases manifest in the algorithms, leading to issues like popularity, ranking, and emergent biases.
- **Interconnections:** The arrows show a continuous feedback loop: user interactions feed into data collection, data trains algorithms, and algorithms shape further user interactions.

Data Science Lifecycle

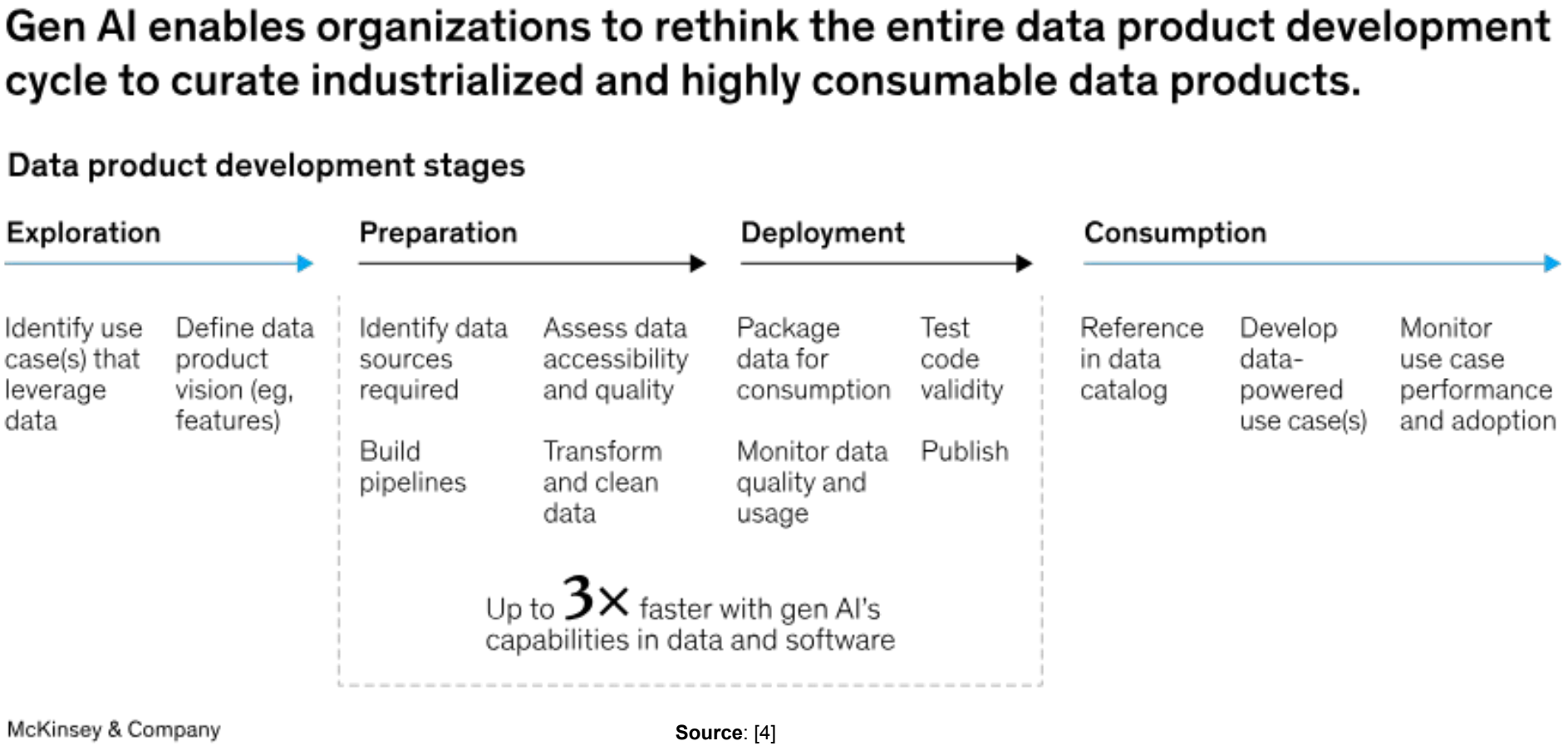
Iterative frameworks guiding projects from idea to deployment.



Source: [4]

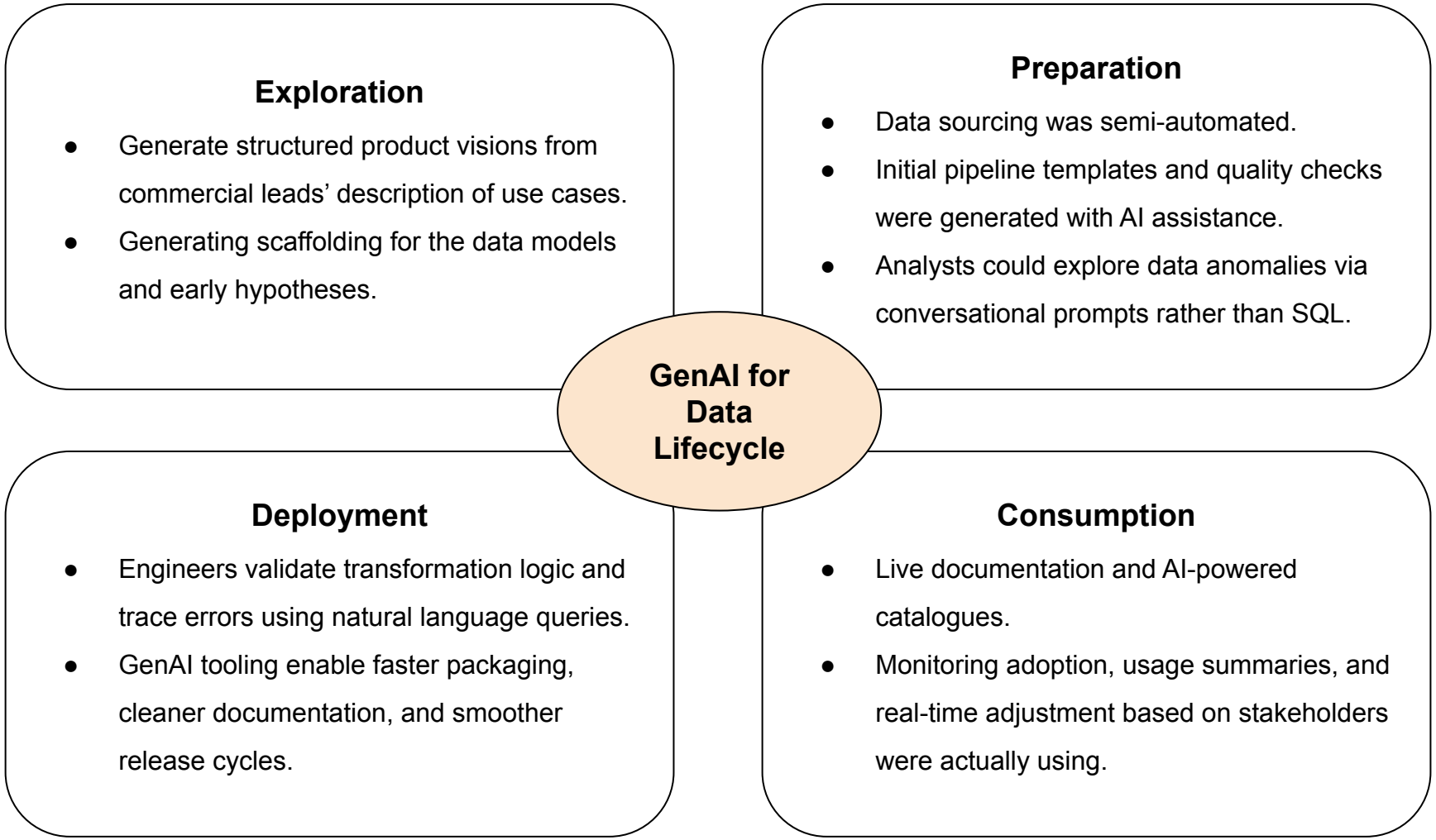
Phase	Description
01	Discovering the answers for basic questions including requirements, priorities and budget of the project.
02	Collecting data from relevant sources either in structured or unstructured form.
03	Processing and fine-tuning the raw data, critical for the goodness of the overall project.
04	Capturing ideas about solutions and factors that influence the data life cycle.
05	Preparing the appropriate model to achieve desired performance.
06	Executing the analysed model in desired format and channel.

How GenAI Change Data Lifecycle?



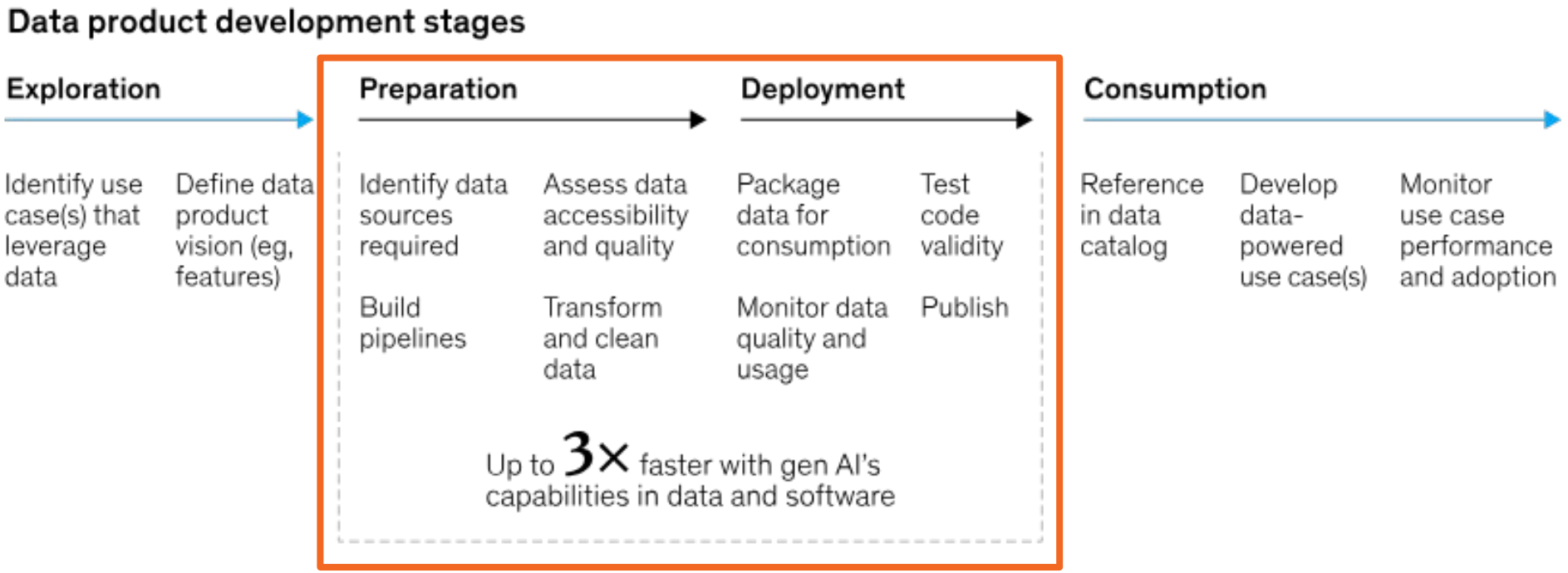
Source: [4]

How GenAI Change Data Lifecycle?



Focus of this lecture

Gen AI enables organizations to rethink the entire data product development cycle to curate industrialized and highly consumable data products.



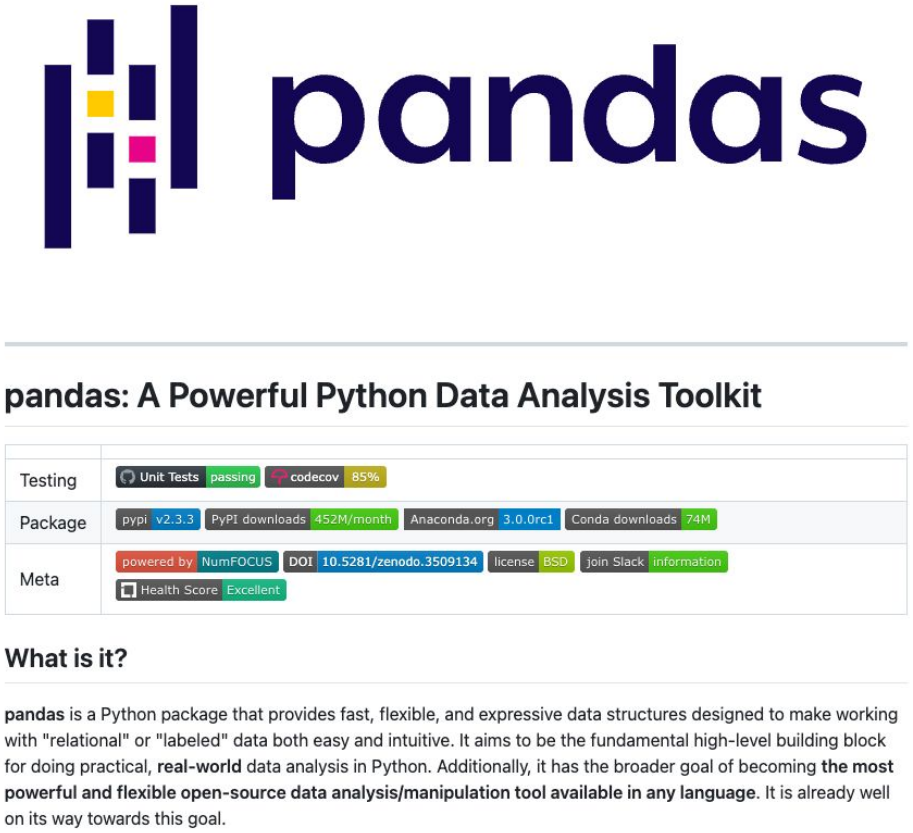
Data Cleaning and Generation

Data Cleaning & Preprocessing

- **Data cleaning**: the processing of identifying and correcting errors or inconsistencies in datasets.
  - Handling missing values
  - Removing duplicates
  - Correcting inaccuracies
  - Ensuring data consistency
  - Dealing with outliers
  - …
- **Data preprocessing**: activities that prepare raw data ready for analysis or model input.
  - It includes data clearing but more importantly
    - Standardizing data formatting
    - Feature scaling (standardize the range of independent variables, ensuring no single feature dominates the others)
    - Data transformation (e.g, log transformation or feature engineer → informative features)
    - Handling categorical variables (encode categorical variables using e.g., one-hot or label encoding)
    - Splitting data into training and testing sets
    - …

Traditional Paradigm — Pandas as An Example

- **What is Pandas?**
  - Pandas is a powerful, fast, and open-source library built on **NumPy**.
  - It is used for data manipulation and real-world data analysis in **Python**.
  - Easy handling of missing data, Flexible reshaping and pivoting of data sets, and size mutability make pandas a great tool for performing data manipulation and handling the data efficiently.



Traditional Paradigm — Pandas and Data Basics

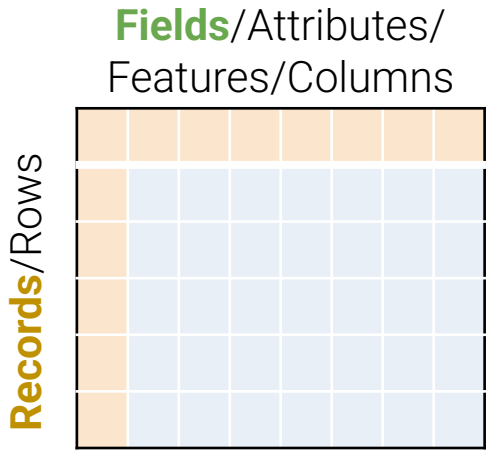
- **Why Pandas?**
  - **DataFrame and Series** allow data to be viewed like spreadsheets (rows/columns), making it easy to load from CSV, Excel, databases, and manipulate.
  - **Data Cleaning & Preparation**: Excellent tools for handling missing data (NaNs), filtering, selecting, merging, and reshaping datasets.
  - **Performance**: Built on NumPy, Pandas is fast for in-memory operations, even with large datasets, using optimized C code under the hood.
  - **Rich Functionality**: Offers built-in descriptive statistics (.describe()), powerful groupby() for aggregations, and flexible data alignment.
  - **Seamless Integration**: Works perfectly with other Python libraries like NumPy (numerical), Matplotlib/Seaborn (visualization), and Scikit-Learn (modeling).
  - **I/O Versatility**: Robust functions (read_csv, read_excel, etc.) make importing and exporting data straightforward.
  - **Accessibility**: Free, open-source, with extensive documentation and community support, making data analysis accessible.

Pandas: Exploring Tabular Data

We prefer rectangular data for data analysis (why?)

- Regular structures are easy manipulate and analyze
- A big part of data cleaning is about transforming data to be more rectangular

Two kinds of rectangular data: **Tables** and **Matrices**.



Tables (a.k.a. dataframes in R/Python and relations in SQL)

- Named columns with different types
- Manipulated using data transformation languages (map, filter, group by, join, …)

Matrices

- Numeric data of the same type (float, int, etc.)
- Manipulated using linear algebra

What are the differences?
Why would you use one over the other?

14

Pandas: DataFrame → Programmatic Spreadsheet

- A **DataFrame** is composed of one or more **Series**. The names of the Series form the column names, and the row labels form the **Index**.

In [1]:
```
import pandas as pd

meteorites = pd.read_csv('../data/Meteorite_Landings.csv', nrows=5)
meteorites
```

Out[1]:

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21	Fell	01/01/1880 12:00:00 AM	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720	Fell	01/01/1951 12:00:00 AM	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000	Fell	01/01/1952 12:00:00 AM	54.21667	-113.00000	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914	Fell	01/01/1976 12:00:00 AM	16.88333	-99.90000	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780	Fell	01/01/1902 12:00:00 AM	-33.16667	-64.95000	(-33.16667, -64.95)

Source: NASA's Open Data Portal

Load and display the DataFrame

Series:

[2]:
```
meteorites.name
```

[2]:
```
0      Aachen
1      Aarhus
2        Abee
3    Acapulco
4     Achiras
Name: name, dtype: object
```

Columns:

[3]:
```
meteorites.columns
```

[3]:
```
Index(['name', 'id', 'nametype', 'recclass', 'mass (g)', 'fall', 'year',
       'reclat', 'reclong', 'GeoLocation'],
      dtype='object')
```

Index:

[4]:
```
meteorites.index
```

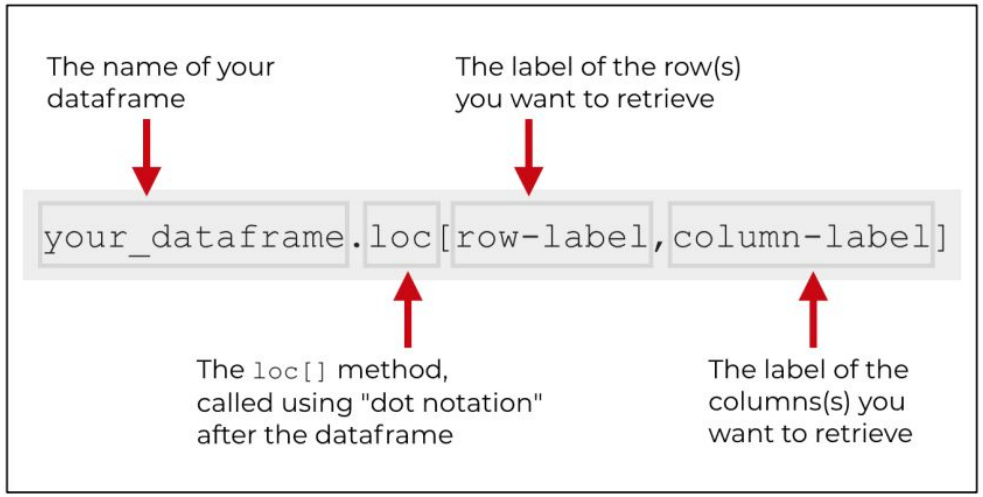
[4]:
```
RangeIndex(start=0, stop=5, step=1)
```

Pandas: Data Types

Data Types	Classes	Description
Numeric	int, float, complex	holds numeric values
String	str	holds sequence of characters
Sequence	list, tuple, range	holds collection of items
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either True or False
Set	set, frozenset	hold collection of unique items

Pandas: Locators → Obtain Data as A Matrix

- **.loc[]** is an important method used for accessing a group of rows and columns
- Written as .loc[rows, columns]
  - rows is usually a logical statement
  - columns is either a string (one column) or a list of columns
- .loc[] is label-based, meaning you specify rows and columns based on their row and column labels



Pandas: Data Cleaning

- **Load the Data**

```
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv('../Messy-dataset/messy_HR_data.csv')

df.head()
```

	Name	Age	Salary	Gender	Department	Position	Joining Date	Performance Score	Email	Phone Number
0	grace	25	50000	Male	HR	Manager	April 5, 2018	D	email@example.com	NaN
1	david	NaN	65000	Female	Finance	Director	2020/02/20	F	user@domain.com	123-456-7890
2	hannah	35	SIXTY THOUSAND	Female	Sales	Director	01/15/2020	C	email@example.com	098-765-4321
3	eve	NaN	50000	Female	IT	Manager	April 5, 2018	A	name@company.org	
4	grace	NaN	NAN	Female	Finance	Manager	01/15/2020	F	name@company.org	098-765-4321

Source: Messy-dataset. https://github.com/eyowhite/Messy-dataset

Pandas: Data Cleaning

- **Inspect the Data**

```
df.shape
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               1000 non-null   object
 1   Age                841 non-null    object
 2   Salary             1000 non-null   object
 3   Gender             1000 non-null   object
 4   Department         1000 non-null   object
 5   Position           1000 non-null   object
 6   Joining Date       1000 non-null   object
 7   Performance Score  1000 non-null   object
 8   Email              610 non-null    object
 9   Phone Number       815 non-null    object
dtypes: object(10)
memory usage: 78.3+ KB
```

	Name	Age	Salary	Gender	Department	Position	Joining Date	Performance Score	Email	Phone Number
count	1000	841	1000	1000	1000	1000	1000	1000	610	815
unique	10	5	6	3	5	5	5	5	3	4
top	alice	thirty	65000	Male	Finance	Assistant	2020/02/20	B	user@domain.com	123-456-7890
freq	118	176	184	355	218	214	232	225	213	236

What to look for?

- Missing values
- Wrong data types (dates as strings, numbers as objects)
- Unexpected zeros or negatives
- Outliers

Pandas: Data Cleaning

- **Fixed Data Types**

```
df = df.copy()

df = df.applymap(
    lambda x: x.strip() if isinstance(x, str) else x
)

df.replace(
    ['N/A', 'NA', 'na', 'None', '', '?', 'unknown'],
    pd.NA,
    inplace=True
)
```

```
df['Age'] = pd.to_numeric(df['Age'], errors='coerce').astype('Int64')
```

```
df['Salary'] = (
    df['Salary']
    .str.replace(r'[\$,]', '', regex=True)
)

df['Salary'] = pd.to_numeric(df['Salary'], errors='coerce')
```

```
df['Joining Date'] = pd.to_datetime(
    df['Joining Date'],
    errors='coerce'
)
```

```
categorical_cols = [
    'Gender',
    'Department',
    'Position',
    'Performance Score'
]

df[categorical_cols] = df[categorical_cols].astype('category')
```

```
string_cols = [
    'Name',
    'Email',
    'Phone Number'
]

df[string_cols] = df[string_cols].astype('string')
```

```
df.loc[~df['Email'].str.contains('@', na=False), 'Email'] = pd.NA
df['Phone Number'] = df['Phone Number'].str.replace(r'\D', '', regex=True)
```

Pandas: Data Cleaning

- **Fixed Data Types**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               1000 non-null   object
 1   Age                841 non-null    object
 2   Salary             1000 non-null   object
 3   Gender             1000 non-null   object
 4   Department         1000 non-null   object
 5   Position           1000 non-null   object
 6   Joining Date       1000 non-null   object
 7   Performance Score  1000 non-null   object
 8   Email              610 non-null    object
 9   Phone Number       815 non-null    object
dtypes: object(10)
memory usage: 78.3+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               1000 non-null   string
 1   Age                665 non-null    Int64
 2   Salary             690 non-null    float64
 3   Gender             1000 non-null   category
 4   Department         1000 non-null   category
 5   Position           1000 non-null   category
 6   Joining Date       186 non-null    datetime64[ns]
 7   Performance Score  1000 non-null   category
 8   Email              610 non-null    string
 9   Phone Number       624 non-null    string
dtypes: Int64(1), category(4), datetime64[ns](1), float64(1), string(3)
memory usage: 52.6 KB
```

Pandas: Data Cleaning

- **Check and Handle Missing Values**

```
df.isna().sum().sort_values(ascending=False)
```

```
Performance Score    1000
Joining Date          814
Email                 390
Phone Number          376
Age                   335
Salary                310
Name                    0
Gender                  0
Department              0
Position                0
dtype: int64
```

```
df = df.dropna(subset=['Name'])
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Salary'] = df['Salary'].fillna(df['Salary'].median())
df['Gender'] = df['Gender'].fillna(
    df['Gender'].mode()[0]
)
df['Department'] = df['Department'].fillna(
    df['Department'].mode()[0]
)
df['Position'] = df['Position'].fillna(
    df['Position'].mode()[0]
)

# Keep the same as original
df['Performance Score'] = df['Performance Score']
df['Joining Date'] = df['Joining Date']
df[['Email', 'Phone Number']] = df[['Email', 'Phone Number']]

df.isna().sum()
df.info()
```

Column	Type	Strategy
Name	string	Drop row if missing
Age	Int64	Median imputation
Salary	float	Median imputation
Gender	category	Mode imputation
Department	category	Mode imputation
Position	category	Mode imputation
Joining Date	datetime	Keep or drop if critical
Performance Score	float	Keep
Email	string	Leave missing
Phone Number	string	Leave missing

Pandas: Data Cleaning

- **And more…**

4. Remove Duplicates

```
[83]: df.duplicated().sum()
```

```
[83]: np.int64(0)
```

5. Validate Data Ranges & Logic

```
[86]: df = df[(df['Age'] >= 16) & (df['Age'] <= 75)]
      df = df[df['Salary'] > 0]
```

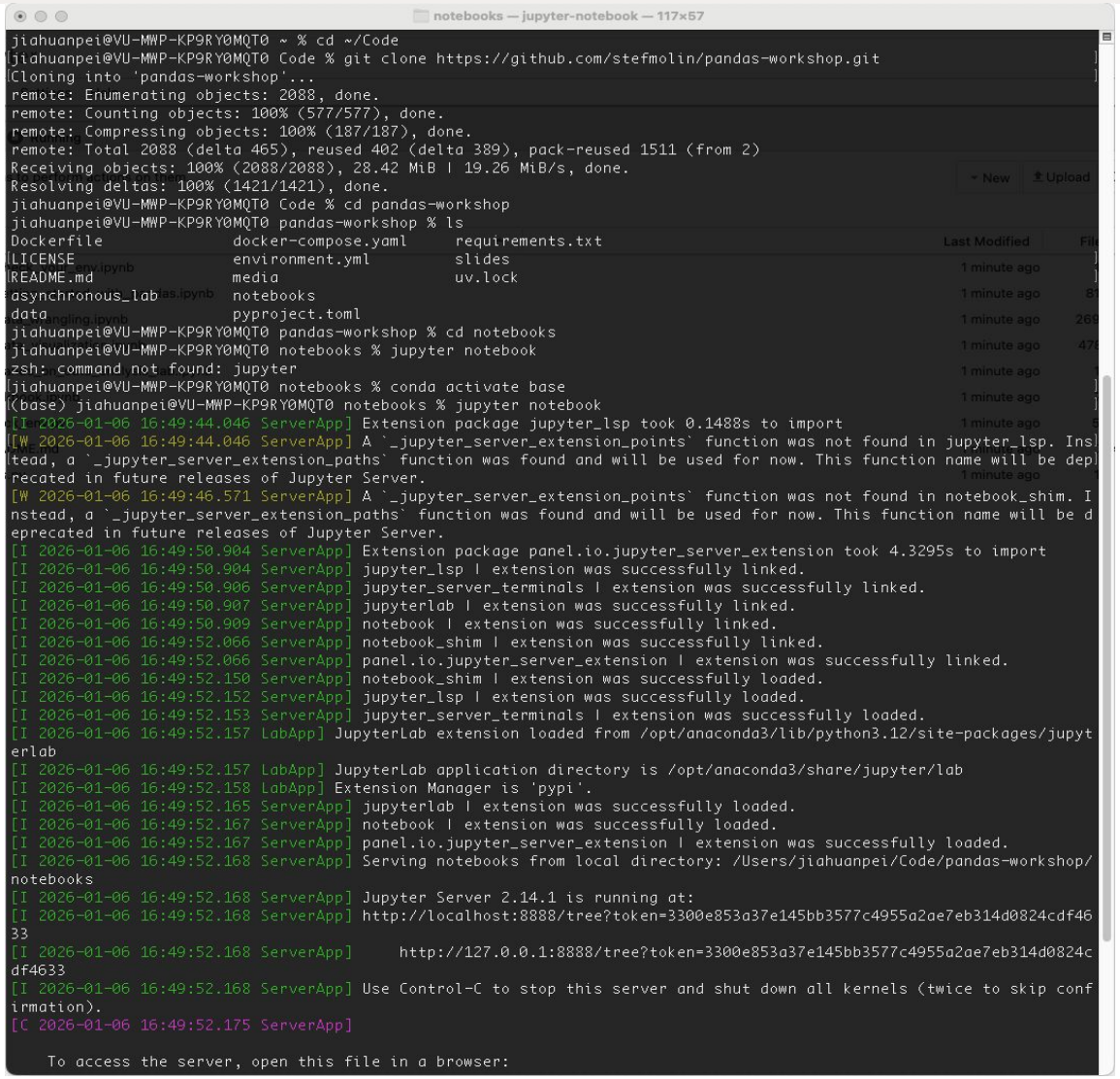
6. Detect & Handle Outliers

```
[87]: Q1 = df['Salary'].quantile(0.25)
      Q3 = df['Salary'].quantile(0.75)
      IQR = Q3 - Q1

      df = df[
          (df['Salary'] >= Q1 - 1.5 * IQR) &
          (df['Salary'] <= Q3 + 1.5 * IQR)
      ]
```

7. Standardize Text Data

```
[88]: text_cols = ['Gender', 'Department', 'Position']

      df[text_cols] = (
          df[text_cols]
          .apply(lambda col: col.str.lower().str.strip())
      )
```

Run the Pandas Jupyter Notebooks



https://github.com/stefmolin/pandas-workshop

The Challenges of Traditional Data Cleaning

- **Manual & Time-Consuming** (human inspection; Ad-hoc scripts for each dataset)
  - Cleaning can take 60–80% of project time; Slow iteration and delayed insights
  - E.g., Manually checking 50+ columns for invalid values
- **Rule-Based → Fragile (**Fixed rules break when data changes)
  - Pipelines fail silently; Rules must be rewritten for new datasets
  - E.g., Fails if new values appear ('UNK', '?', 'N/A')
- **Poor Scalability** (Works for small datasets, not big data)
  - Memory bottlenecks; Long execution times
  - E.g., Pandas scripts that crash on 10M+ rows
- **Inconsistent Cleaning Logic** (Different analysts apply different rules)
  - Non-reproducible results; Conflicting analyses
  - E.g., One analyst drops outliers, another caps them
- **Limited Error Detection** (Only known issues are fixed)
  - Hidden anomalies remain; Bad data enters models unnoticed
  - E.g., Age = 999 → not caught by simple NA checks

The Challenges of Traditional Data Cleaning

- **Subjective Decisions** (Imputation choices vary by person)
  - Bias introduced; Hard to justify decisions later
  - E.g., Mean vs median vs drop rows?
- **Lack of Context Awareness** (Rules ignore domain logic)
  - Technically “clean” but logically wrong data
  - E.g., Salary cleaned, but lower than minimum wage
- **Difficult to Maintain** (Scripts grow messy over time)
  - Hard to debug; New team members struggle
  - E.g., 500-line cleaning notebook with no documentation
- **Poor Handling of Unstructured Data** (Traditional tools focus on tabular data)
  - Text, logs, images poorly cleaned
  - E.g., Inconsistent job titles or free-text feedback
- No Feedback Loop (Cleaning doesn’t learn from past errors)
  - Same issues repeat across datasets
  - E.g., Re-fixing the same email format issues every month

How Modern Approaches Address These?

Traditional Issue	Modern Solution
Manual cleaning	Automation
Fixed rules	ML-based anomaly detection
Inconsistent logic	Standard pipelines
Hidden errors	Data validation frameworks
No learning	Adaptive systems

AI-assisted data quality tools!

Data Cleaning and Generation With GenAI

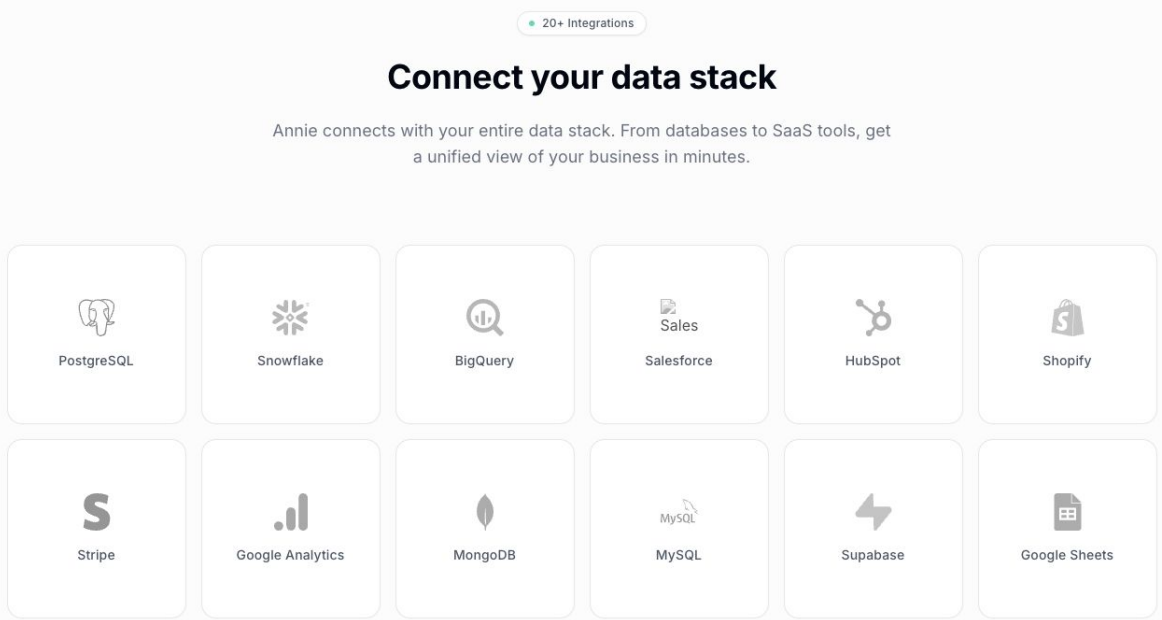
New GenAI Paradigm — PandasAI

Pandas AI: The Generative AI Python Library

PandasAI

Release v3.0.0 | ci-core passing | cd passing | codecov 91% | Discord | downloads 5M | License MIT | Open in Colab

PandasAI is a Python library that makes it easy to ask questions to your data in natural language. It helps non-technical users to interact with their data in a more natural way, and it helps technical users to save time, and effort when working with data.



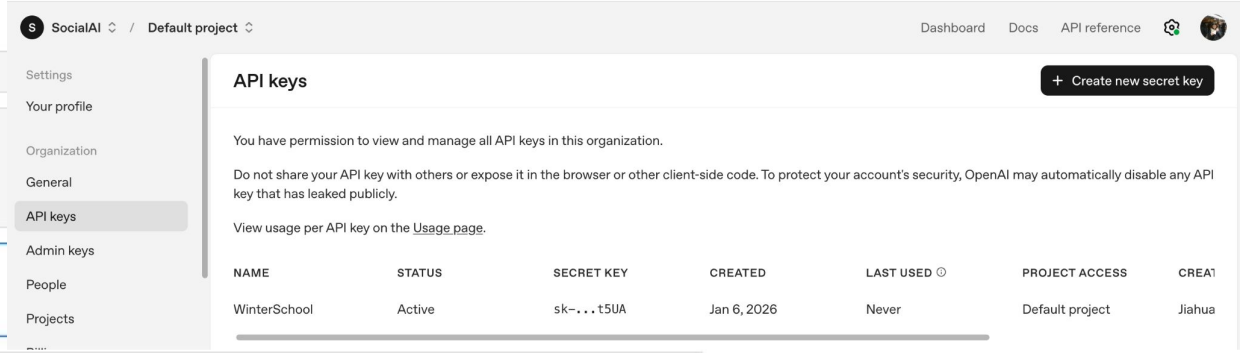
PandasAI with OpenAI Models via API

- **Link your LLMs to DataFrame**

```
# !pip install pandasai
```

```
import pandas as pd
# from pandasai import PandasAI
from pandasai import SmartDataframe
from pandasai.llm.openai import OpenAI
```

```
import os
openai_api_key = os.environ["OPENAI_API_KEY"]
llm = OpenAI(api_token=openai_api_key)
```




```
df = pd.read_csv("Datasets/ds_salaries.csv")
df.head()
```

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	cor
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	100	ES	
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	100	US	
2	2023	MI	CT	ML Engineer	25500	USD	25500	US	100	US	
3	2023	SE	FT	Data Scientist	175000	USD	175000	CA	100	CA	
4	2023	SE	FT	Data Scientist	120000	USD	120000	CA	100	CA	

```
# Initializing an instance of PandasAI with OpenAI environment
pandas_ai = SmartDataframe(df, config={"llm": llm})
```

PandasAI with OpenAI Models via API

- **Chat with DataFrame**

```
response=pandas_ai.chat('List the first 5 job titles by salary in usd')
print(response)
```

```
{'type': 'dataframe', 'value':                                job_title  salary_in_usd
3522                  Research Scientist         450000
2011                        Data Analyst         430967
528                         AI Scientist         423834
3747  Applied Machine Learning Scientist         423000
3675            Principal Data Scientist         416000}
                              job_title  salary_in_usd
3522                  Research Scientist         450000
2011                        Data Analyst         430967
528                         AI Scientist         423834
3747  Applied Machine Learning Scientist         423000
3675            Principal Data Scientist         416000
```

```
response=pandas_ai.chat('What is the average salary in usd by job titles? Make sure the output is sorted in descending order.')
print(response)
```

```
                       job_title  salary_in_usd
46         Data Science Tech Lead     375000.000
19           Cloud Data Architect     250000.000
35                      Data Lead     212500.000
28            Data Analytics Lead     211254.500
84       Principal Data Scientist     198171.125
..                            ...            ...
9   Autonomous Vehicle Technician      26277.500
0   3D Computer Vision Researcher      21352.250
91             Staff Data Analyst      15000.000
87         Product Data Scientist       8000.000
80             Power BI Developer       5409.000

[93 rows x 2 columns]
```

PandasAI with OpenAI Models via API

- **Chat with DataFrame**

```
response=pandas_ai.chat('List the first 5 job titles by salary in usd')
print(response)
```

```
{'type': 'dataframe', 'value':                                job_title  salary_in_usd
3522                  Research Scientist         450000
2011                        Data Analyst         430967
528                         AI Scientist         423834
3747  Applied Machine Learning Scientist         423000
3675            Principal Data Scientist         416000}
                                job_title  salary_in_usd
3522                  Research Scientist         450000
2011                        Data Analyst         430967
528                         AI Scientist         423834
3747  Applied Machine Learning Scientist         423000
3675            Principal Data Scientist         416000
```

```
response=pandas_ai.chat('What is the average salary in usd by job titles? Make sure the output is sorted in descending order.')
print(response)
```

```
                       job_title  salary_in_usd
46         Data Science Tech Lead     375000.000
19           Cloud Data Architect     250000.000
35                      Data Lead     212500.000
28            Data Analytics Lead     211254.500
84       Principal Data Scientist     198171.125
..                            ...            ...
9   Autonomous Vehicle Technician      26277.500
0   3D Computer Vision Researcher      21352.250
91             Staff Data Analyst      15000.000
87         Product Data Scientist       8000.000
80             Power BI Developer       5409.000

[93 rows x 2 columns]
```

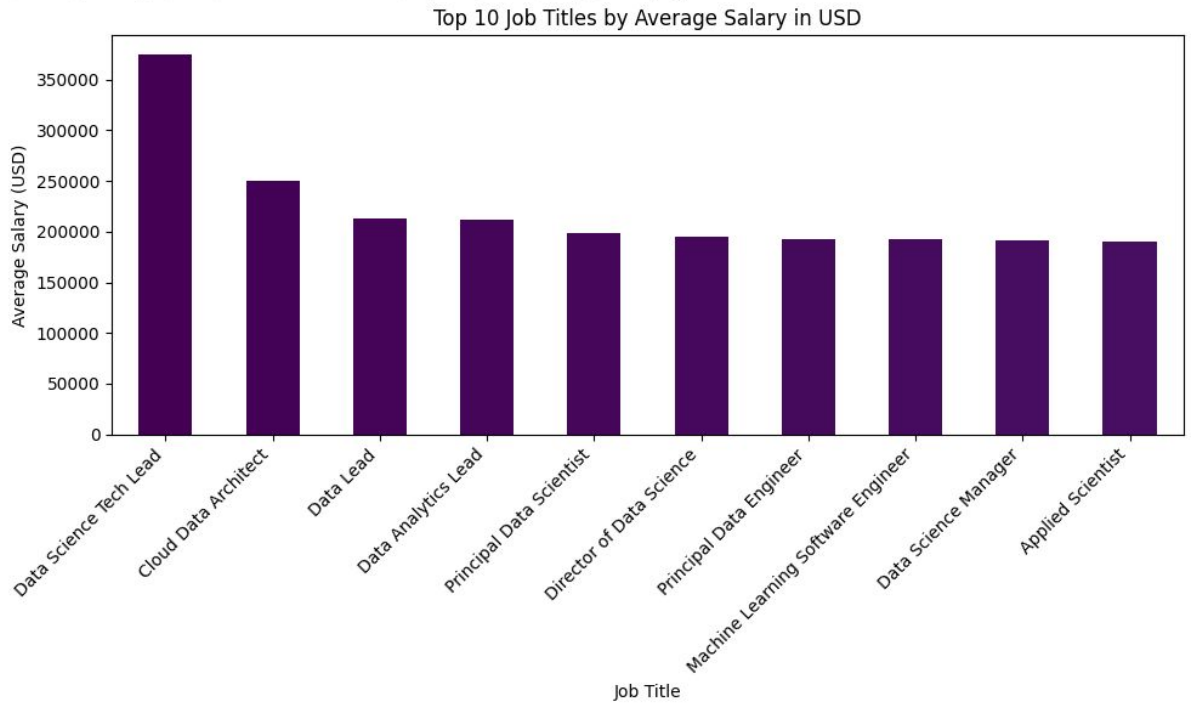
GenAI for Data Analysis and Interpretation

PandasAI with OpenAI Models via API

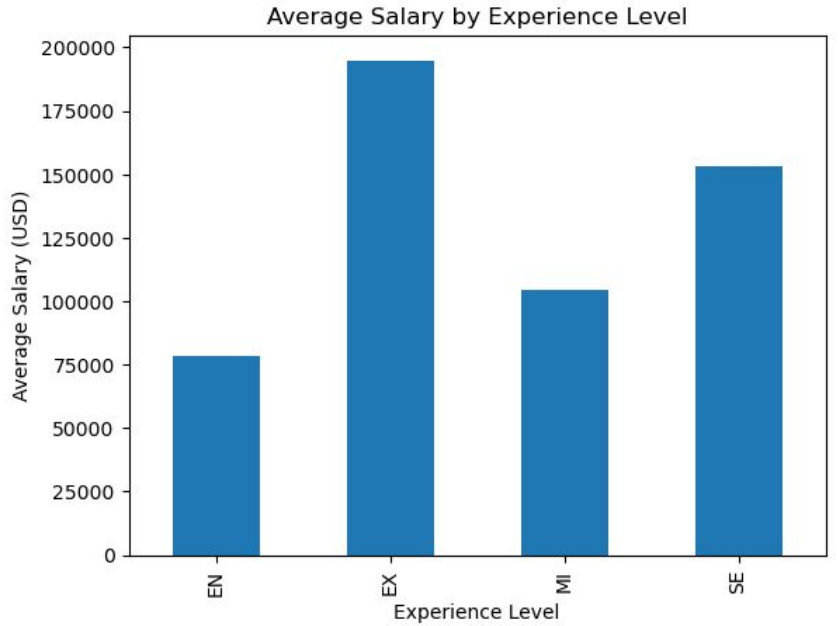
- **Chat with DataFrame**

```
response=pandas_ai.chat('Plot a bar chart showing top 10 job titles, using different colors for each bar')
print(response)
```

```
/Users/jiahuanpei/Code/PandasAI-Tutorials/exports/charts/temp_chart.png
```




```
response=pandas_ai.chat('Plot a bar chart showing average salary in usd by experience level')
print(response)
```




```
None
```

PandasAI with Open-Resource LLMs (Langchain-Ollama)

- **Link your LLMs and chat with DataFrame**

```
import pandas as pd
data = pd.read_csv("Datasets/population.csv")
data.head()
```

	Country	Population
0	United States	339996563
1	China	1425671352
2	Germany	83294633
3	Turkey	85816199
4	Japan	123294513

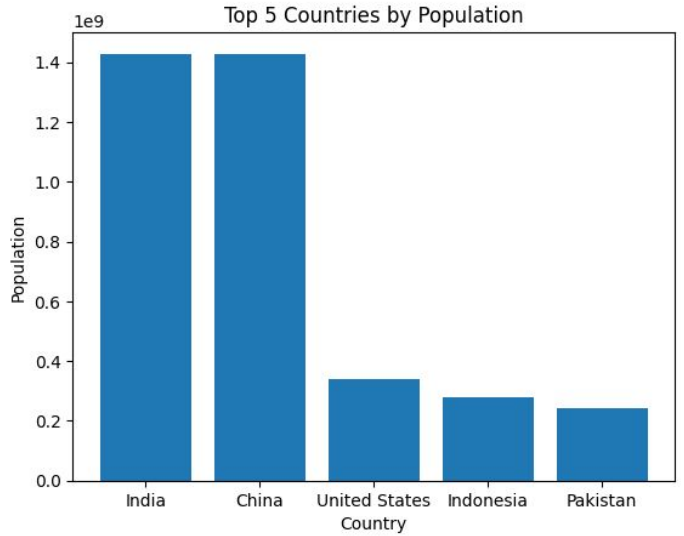
```
! pip install -U langchain-ollama
```

```
# from langchain_community.llms import Ollama
from langchain_ollama import OllamaLLM
llm = OllamaLLM(model="mistral")
```

```
from pandasai import SmartDataframe
df = SmartDataframe(data, config={"llm": llm})
```

```
df.chat('Which are top 5 countries by population?')
```

	Country	Population
8	India	1428627663
1	China	1425671352
0	United States	339996563
5	Indonesia	277534122
6	Pakistan	240485658




```
print(df.chat("What is the total populations of the top 5 countries by population?"))
```

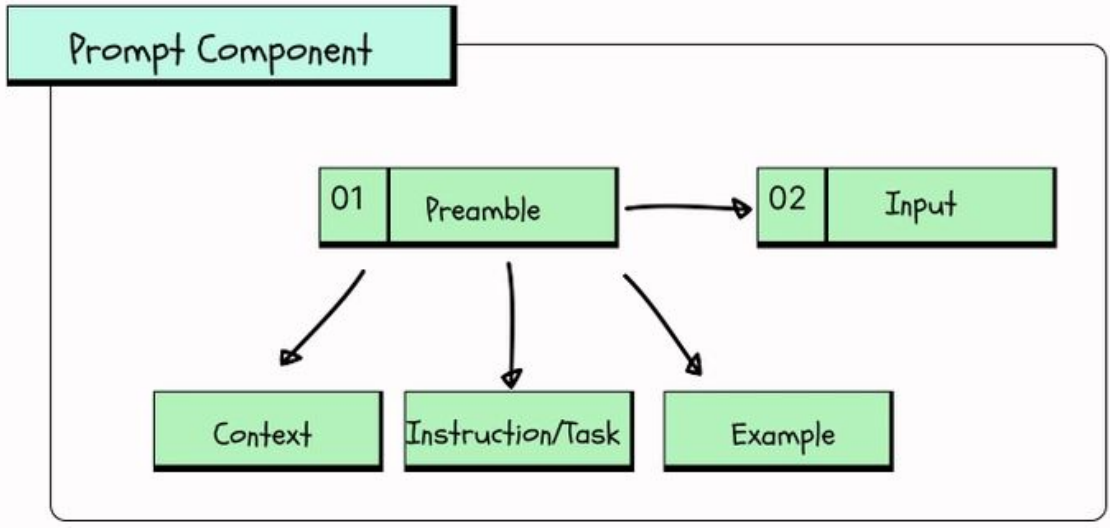
Key Components of the New GenAI Paradigm

- Prompts are the **interface**, but the **data schema**, **validation**, and **execution** determine the real quality and safety of GenAI-driven cleaning.

Component	Role in GenAI Data Cleaning
Prompts	Tell AI what to do
Data Context / Schema	Column names, types, domain info
LLM / Model	Generates code or analysis
Execution Layer	Runs the code safely
Validation	Ensures output is correct
Iterative Refinement	Improve outputs with follow-ups
Automation / Pipeline	Make the workflow repeatable
Explainability	Document AI decisions

Recall: What is a prompt?

- A prompt is the instruction(s) that we provide to a GenAI model to guide its response.
- Most well-designed prompts can be divided into two main components:
  - Preamble (defines the context and expectations of the model)
    - Context: background or relevant information so that the response is aligned with the situation or specific domain.
    - Task: the specific action we want the model to perform accurately.
  - Input (the content that the model must process, transform, or generate).



Prompts for GenAI-supported Data Transformations

Use case	Prompt	Why this works
Basic data cleaning	"Remove special characters from the Product Name column."	The prompt is straightforward and clear.
Filtering values	"Return only values which contain 'Mobile' in the Device column."	The prompt specifies both the filter criterion and the relevant column.
Generating code	"What Python expression can I use in addfieldx to calculate the difference in days between the Start Date and End Date columns?"	The prompt is clear about the columns involved and the desired calculation, leaving no room for interpretation.
Formatting dates	"I would like to convert my Date column format from DD-MM-YYYY to YYYY-MM-DD."	The prompt provides precise formatting instructions.
Adding columns based on certain conditions	"Add a new column called 'Channel type' that contains 'online' if values in the Channel column are 'Social Media', and 'offline' if values in the Channel column are 'TV'."	The prompt includes the new column's name and the conditions for its values.
Advanced data cleaning	"I want to: 1. Split the values in the 'Ad name' column into 4 separate columns, using the _ as the delimiter. 2. Remove empty rows from those 4 newly created columns."	The prompt breaks down the task into steps to ensure clarity and accuracy.

Best Practices for Writing Effective AI Prompts

- **Be Clear and Specific**
  - AI tools work best with precise instructions. Instead of saying, "Clean up my data," specify precisely what needs cleaning.
  - Good Example:

    "Find and remove duplicate email addresses in Column C while keeping the first occurrence."
  - Bad Example:

    "Remove duplicates." (Too vague—what column? Which duplicate should be kept?)

Best Practices for Writing Effective AI Prompts

- **Use Action Words**
  - AI responds well to clear, direct actions like convert, clean, find, remove, standardize, categorize, or format.
  - Good Example:

    "Find and replace all instances of 'N/A' in Column B with 'Not Available'."
  - Bad Example:

    "Change data." (It is too vague. What data needs changing? How should it be changed?)

Best Practices for Writing Effective AI Prompts

- **Specify Columns or Data Ranges**
  - AI works on entire datasets, so specifying columns or ranges prevents mistakes.
  - Good Example:

    "Standardize capitalization in Column D so that each word starts with a capital letter."
  - Bad Example:

    "Fix capitalization." (Fix where? Capitalization in what way?)

Best Practices for Writing Effective AI Prompts

- **Provide Formatting Details**
  - If formatting is required, specify the desired format.
  - Good Example:

    "Convert all dates in Column A to YYYY-MM-DD format."
  - Bad Example:

    "Fix the date format." (What format should it be?)

Best Practices for Writing Effective AI Prompts

- **Define Conditions for Cleaning Tasks**
  - If conditional logic applies, include it in the prompt.
  - Good Example:

    "Remove duplicate rows only if the values in Column B and Column C are identical."
  - Bad Example:

    "Remove some duplicates." (How should AI determine which to remove?)

Best Practices for Writing Effective AI Prompts

- **Use Natural Language, But Avoid Unnecessary Words**
  - AI can process natural language, but overly conversational prompts may reduce accuracy.
  - Good Example:

    "Identify and highlight all cells in Column E that contain numbers lower than 10."
  - Bad Example:

    "Hey AI, can you please take a look at Column E and tell me which ones have numbers under 10? Thanks!" (Too wordy—keep it structured.)

Basic Structure of an AI Prompt

Action + Data Range + Condition (if needed) + Formatting (if required)

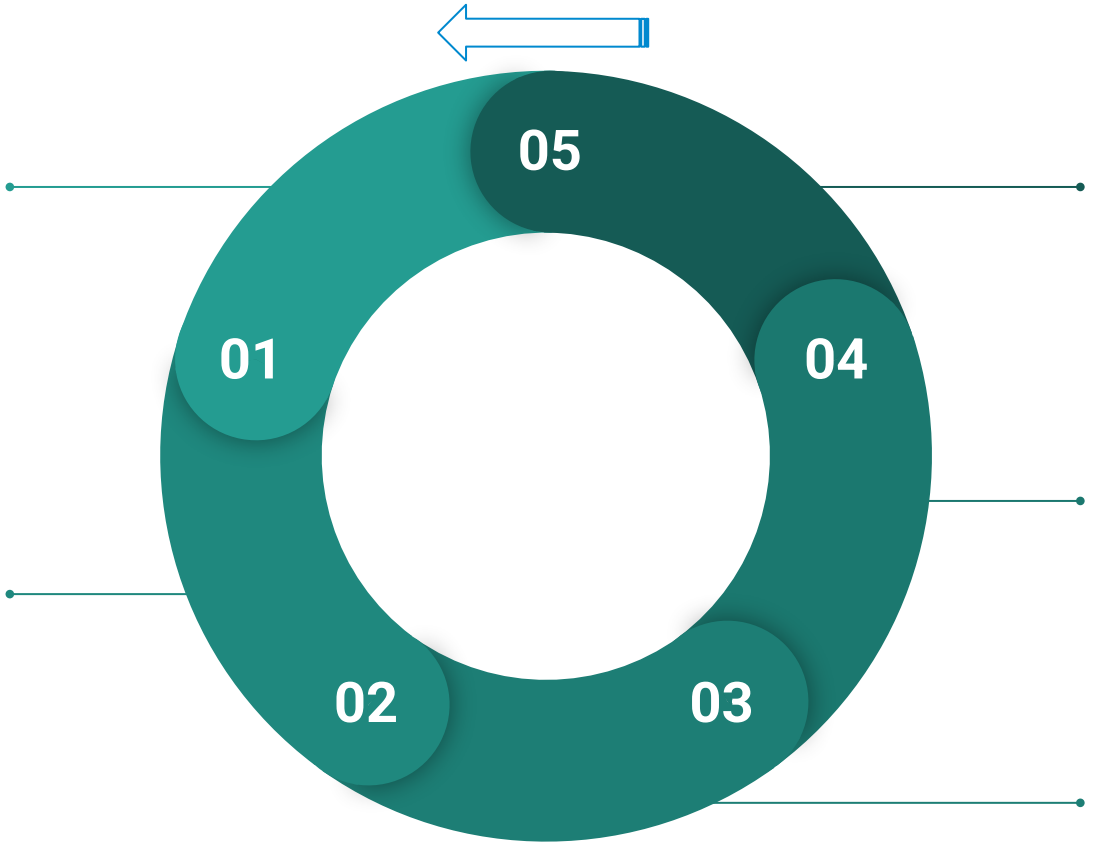
Examples of Well-Structured AI Prompts

- **Handling Missing Data**: "Fill all missing values in Column D using the column's median value." "Identify and highlight all empty cells in this dataset."
- **Detecting & Removing Duplicates**: "Find and remove duplicate rows in Column C while keeping the first occurrence." "Merge duplicate customer names in Column A, keeping the most recent record."
- **Standardizing Formatting**: "Ensure all email addresses in Column F are in lowercase." "Convert all text in Column B to title case (capitalize first letter of each word)."
- **Text Cleaning & Categorization**: "Replace all instances of 'N/A' with 'Not Available' in Column G." "Classify customer reviews in Column H as 'Positive,' 'Neutral,' or 'Negative' based on sentiment analysis."
- **Outlier & Error Detection**: "Flag all sales amounts in Column E that are above $10,000 as outliers." "Identify and correct any negative numbers in the 'Revenue' column."

GenAI-based Workflow: Integrating GenAI Into Data Pipeline

Workflow Summary

Iterate: refine prompts as needed



Prompts

natural language instructions

SmartDataframe / LLM

generates Pandas code & visualizations

EDA & Features

insights + engineered features

Validation

checks results for correctness

Execution Layer

applies transformations safely

Workflow

- Setup PandasAI with a LLM

```python
import os
import pandas as pd
from pandasai import SmartDataframe
from pandasai.llm.openai import OpenAI

# Set your OpenAI API key
# os.environ["OPENAI_API_KEY"] = "sk-your_api_key_here"  # replace with your key

# Load your HR dataset
# df = pd.read_csv("../Messy-dataset/messy_HR_data.csv")
df = pd.read_csv("../Messy-dataset/cleaned_messy_HR_data.csv")

# Initialize LLM and SmartDataframe
openai_api_key = os.environ["OPENAI_API_KEY"]
llm = OpenAI(api_token=openai_api_key)
sdf = SmartDataframe(df.sample(50), config={"llm": llm})
```

Workflow

- Prompt: Summary & Missing Data

```
import sys
sys.setrecursionlimit(5000)
# Ask AI to summarize the dataset
summary = sdf.chat("Summarize this HR dataset: column types, missing values, and basic stats") #
print(summary)
```

```
                      column_types  missing_values  \
Name                        string               0
Age                          Int64               0
Salary                     float64               0
Gender                      object               0
Department                  object               0
Position                    object               0
Joining Date          datetime64[ns]               0
Performance Score          float64              50
Email                       string              22
Phone Number                string              21
Tenure                       int64               0
Salary_Band               category              0
Performance_Category      category              50

                                                    basic_stats
Name                {'count': 50, 'unique': 9, 'top': 'alice', 'fr...
Age                 {'count': 50.0, 'unique': <NA>, 'top': <NA>, '...
```

Workflow

- Prompt: Detect Outliers

```
outliers = sdf.chat("Detect outliers in Age, Salary, and Performance Score columns and suggest handling")
print(outliers)
```

```
Age Outliers:
        Name  Age   Salary  Gender Department   Position Joining Date  \
10     david   25  70000.0    Male  Marketing    Analyst   2019-12-01
417     jack   50  50000.0    Male         IT    Analyst   2018-04-05
793  charlie   50  70000.0  Female    Finance    Manager   2019-03-25
181      eve   50  70000.0   Other  Marketing      Clerk   2018-04-05
704      bob   25  70000.0  Female  Marketing  Assistant   2019-12-01
635   hannah   25  55000.0    Male         HR   Director   2018-04-05
743  charlie   25  70000.0  Female         IT    Manager   2019-03-25
122  charlie   25  65000.0  Female      Sales    Manager   2020-01-15
235      eve   25  65000.0   Other         HR    Manager   2018-04-05
831    alice   50  65000.0    Male         IT    Manager   2019-12-01
502    alice   25  50000.0  Female      Sales    Analyst   2020-01-15
883   hannah   50  65000.0  Female  Marketing  Assistant   2020-02-20
647      ivy   25  70000.0   Other    Finance  Assistant   2020-01-15
193    alice   25  65000.0   Other    Finance    Analyst   2020-01-15
790     jack   25  65000.0    Male  Marketing   Director   2020-02-20
383     jack   50  65000.0   Other  Marketing    Analyst   2019-12-01

     Performance Score              Email  Phone Number  Tenure Salary_Band  \
10                 NaN              <NA>  123-456-7890      74   Very High
417                NaN              <NA>          <NA>      94         Low
793                NaN    user@domain.com          <NA>      82   Very High
181                NaN   name@company.org          <NA>      94   Very High
704                NaN              <NA>          <NA>      74   Very High
635                NaN              <NA>  555-555-5555      94         Low
743                NaN              <NA>          <NA>      82   Very High
122                NaN              <NA>  123-456-7890      72      Medium
235                NaN              <NA>  555-555-5555      94      Medium
831                NaN   name@company.org          <NA>      74      Medium
502                NaN  email@example.com  098-765-4321      72         Low
883                NaN    user@domain.com  098-765-4321      71      Medium
647                NaN   name@company.org  123-456-7890      72   Very High
193                NaN    user@domain.com          <NA>      72      Medium
790                NaN              <NA>  555-555-5555      71      Medium
383                NaN    user@domain.com          <NA>      74      Medium
```

```
    Performance_Category
10                   NaN
417                  NaN
793                  NaN
181                  NaN
704                  NaN
635                  NaN
743                  NaN
122                  NaN
235                  NaN
831                  NaN
502                  NaN
883                  NaN
647                  NaN
193                  NaN
790                  NaN
383                  NaN
Handling Suggestion: Consider capping at 16 and 75 or replacing with median.

Salary Outliers:
Empty DataFrame
Columns: [Name, Age, Salary, Gender, Department, Position, Joining Date, Performance Score, Email, Phone Number, Tenure, Salary_Band, Perfo
rmance_Category]
Index: []
Handling Suggestion: Consider capping at a reasonable maximum or replacing with median.

Performance Score Outliers:
Empty DataFrame
Columns: [Name, Age, Salary, Gender, Department, Position, Joining Date, Performance Score, Email, Phone Number, Tenure, Salary_Band, Perfo
rmance_Category]
Index: []
Handling Suggestion: Consider capping at a reasonable maximum or replacing with median.

Outliers detected and suggestions provided for handling.
```

Workflow

- Prompt: Automated Cleaning Suggestions

```
cleaned_df = sdf.chat("""
Clean the dataset:
- Impute missing Age with median
- Impute missing Salary with median
- Fill missing Gender/Department/Position with mode
- Leave Email and Phone Number missing
- Remove duplicates
Return the cleaned DataFrame
""")
```

```
cleaned_df
```

	Name	Age	Salary	Gender	Department	Position	Joining Date	Performance Score	Email	Phone Number
10	david	25	70000.0	Male	Marketing	Analyst	2019-12-01	B	<NA>	123-456-7890
576	alice	40	65000.0	Male	Finance	Clerk	2018-04-05	F	<NA>	<NA>
232	bob	40	65000.0	Female	Finance	Analyst	2020-01-15	D	email@example.com	098-765-4321
611	hannah	40	55000.0	Other	Sales	Manager	2019-12-01	A	email@example.com	123-456-7890
417	jack	50	50000.0	Male	IT	Analyst	2018-04-05	A	<NA>	<NA>
793	charlie	50	70000.0	Female	Finance	Manager	2019-03-25	B	user@domain.com	<NA>
965	david	35	65000.0	Female	Marketing	Director	2018-04-05	A	email@example.com	123-456-7890
181	eve	50	70000.0	Other	Marketing	Clerk	2018-04-05	B	name@company.org	<NA>
783	jack	40	50000.0	Male	IT	Assistant	2018-04-05	B	<NA>	<NA>

Workflow

- Exploratory Data Analysis (EDA)
  - Text summary (tables, means, medians)
  - Generates plots inline (histograms, boxplots, bar charts)

```
eda_summary = sdf.chat("""
Perform EDA:
- Show average Salary by Department and Position
- Show distribution of Age and Salary
- Show performance scores by Gender
- Visualize Salary distribution by Department
""")
print(eda_summary)
```

	Department	Position	Salary
0	Finance	Analyst	61666.666667
1	Finance	Assistant	70000.000000
2	Finance	Clerk	60000.000000
3	Finance	Manager	55000.000000
4	HR	Analyst	55000.000000
5	HR	Assistant	70000.000000
6	HR	Clerk	60000.000000
7	HR	Director	60000.000000
8	HR	Manager	67500.000000
9	IT	Analyst	52500.000000
10	IT	Assistant	60000.000000
11	IT	Director	70000.000000
12	IT	Manager	67500.000000
13	Marketing	Analyst	68333.333333
14	Marketing	Assistant	67500.000000
15	Marketing	Clerk	62500.000000
16	Marketing	Director	61000.000000
17	Marketing	Manager	50000.000000
18	Sales	Analyst	50000.000000
19	Sales	Assistant	60000.000000
20	Sales	Clerk	63333.333333
21	Sales	Director	65000.000000
22	Sales	Manager	60000.000000

<Figure size 1000x600 with 0 Axes>

Workflow

- Feature Engineering

```
features = sdf.chat("""
Suggest new features for predicting Performance Score:
- Derive Tenure from Joining Date
- Create Salary_Band based on quartiles
- Suggest any other useful columns
""")
print(features)
```

	Name	Age	Salary	Gender	Department	Position	Joining Date	\
10	david	25	70000.0	Male	Marketing	Analyst	2019-12-01	
576	alice	40	65000.0	Male	Finance	Clerk	2018-04-05	
232	bob	40	65000.0	Female	Finance	Analyst	2020-01-15	
611	hannah	40	55000.0	Other	Sales	Manager	2019-12-01	
417	jack	50	50000.0	Male	IT	Analyst	2018-04-05	
793	charlie	50	70000.0	Female	Finance	Manager	2019-03-25	
965	david	35	65000.0	Female	Marketing	Director	2018-04-05	
181	eve	50	70000.0	Other	Marketing	Clerk	2018-04-05	
783	jack	40	50000.0	Male	IT	Assistant	2018-04-05	

Workflow

- Validation / QA

```
validation = sdf.chat("""
Check the dataset for any remaining issues:
- Missing values
- Duplicates
- Logical inconsistencies (Age < 16 or > 75, Salary < 0)
""")
print(validation)
```

```
{'type': 'string', 'value': 'Missing Values:\nName                    0\nAge                     0\nSalary                  0\nGender
0\nDepartment              0\nPosition               0\nJoining Date            0\nPerformance Score       50\nEmail
22\nPhone Number           21\nTenure                  0\nSalary_Band             0\nPerformance_Category    50\nAge_Group
0\ndtype: int64\n\nDuplicates: 0\n\nAge Inconsistencies:\nEmpty DataFrame\nColumns: [Name, Age, Salary, Gender, Department, Position, Joini
ng Date, Performance Score, Email, Phone Number, Tenure, Salary_Band, Performance_Category, Age_Group]\nIndex: []\n\nSalary Inconsistencie
s:\nEmpty DataFrame\nColumns: [Name, Age, Salary, Gender, Department, Position, Joining Date, Performance Score, Email, Phone Number, Tenur
e, Salary_Band, Performance_Category, Age_Group]\nIndex: []'}
Missing Values:
Name                    0
Age                     0
Salary                  0
Gender                  0
Department              0
Position                0
Joining Date            0
Performance Score      50
Email                  22
Phone Number           21
Tenure                  0
Salary_Band             0
Performance_Category   50
Age_Group               0
dtype: int64

Duplicates: 0

Age Inconsistencies:
Empty DataFrame
Columns: [Name, Age, Salary, Gender, Department, Position, Joining Date, Performance Score, Email, Phone Number, Tenure, Salary_Band, Perfo
rmance_Category, Age_Group]
Index: []

Salary Inconsistencies:
Empty DataFrame
Columns: [Name, Age, Salary, Gender, Department, Position, Joining Date, Performance Score, Email, Phone Number, Tenure, Salary_Band, Perfo
rmance_Category, Age_Group]
Index: []
```

Advantages of This GenAI Workflow

- No need to write repetitive Pandas code manually
- AI suggests smart imputations and outlier handling
- Generates visualizations automatically
- Can create reusable, explainable cleaning steps
- Works interactively: you can refine prompts in real-time

Improvements for Real-World Datasets

- Safe Types: Convert all columns to safe types

```python
import os
import pandas as pd
from pandasai import SmartDataframe
from pandasai.llm.openai import OpenAI

# Set your OpenAI API key
# os.environ["OPENAI_API_KEY"] = "sk-your_api_key_here"

# Load full HR dataset
df = pd.read_csv("../Messy-dataset/cleaned_messy_HR_data.csv")

# 1️⃣ Ensure numeric columns are numeric
numeric_cols = ['Age', 'Salary', 'Performance Score']
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Fill missing numeric values
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Salary'] = df['Salary'].fillna(df['Salary'].median())
df['Performance Score'] = df['Performance Score'].fillna(df['Performance Score'].median())

# 2️⃣ Convert categorical columns to string
cat_cols = ['Gender', 'Department', 'Position']
for col in cat_cols:
    df[col] = df[col].astype(str)

# 3️⃣ Convert dates to string
df['Joining Date'] = df['Joining Date'].astype(str)
```

Improvements for Real-World Datasets

- Chunking for Safe AI Interaction
  - For large datasets, we don’t send the full DataFrame at once.
  - Split it into manageable chunks (e.g., 500 rows):

```
chunk_size = 500
chunks = [df[i:i+chunk_size] for i in range(0, len(df), chunk_size)]
```

Improvements for Real-World Datasets

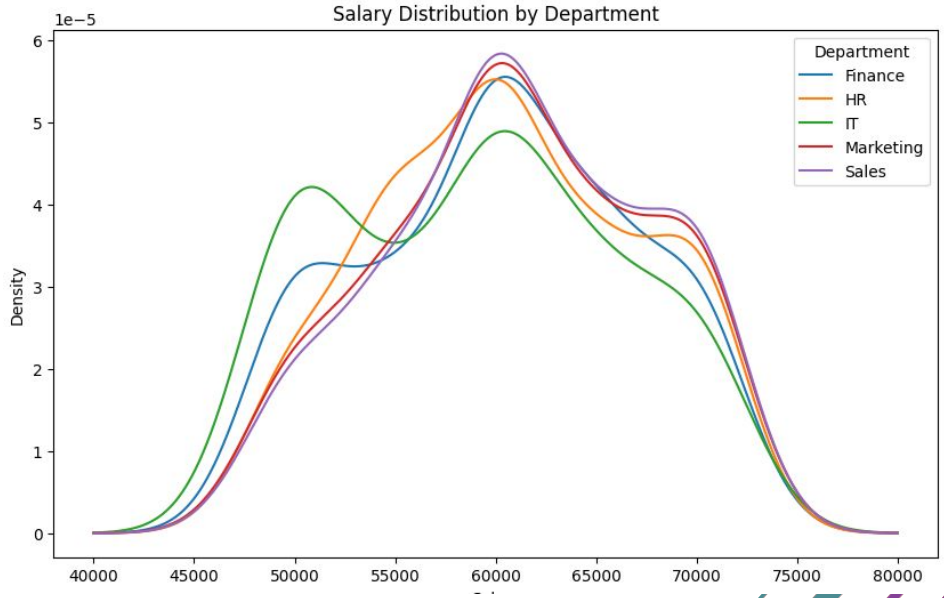
- Summaries, aggregates, plots → no full DataFrame return

```
def summarize_chunk(df_chunk):
    sdf = SmartDataframe(df_chunk, config={"llm": llm, "enable_cache": False})
    summary = sdf.chat("""
    Summarize the dataset:
    - Count missing values per column
    - Average Salary by Department
    - Average Performance Score by Gender
    """)
    return summary
```

Improvements for Real-World Datasets

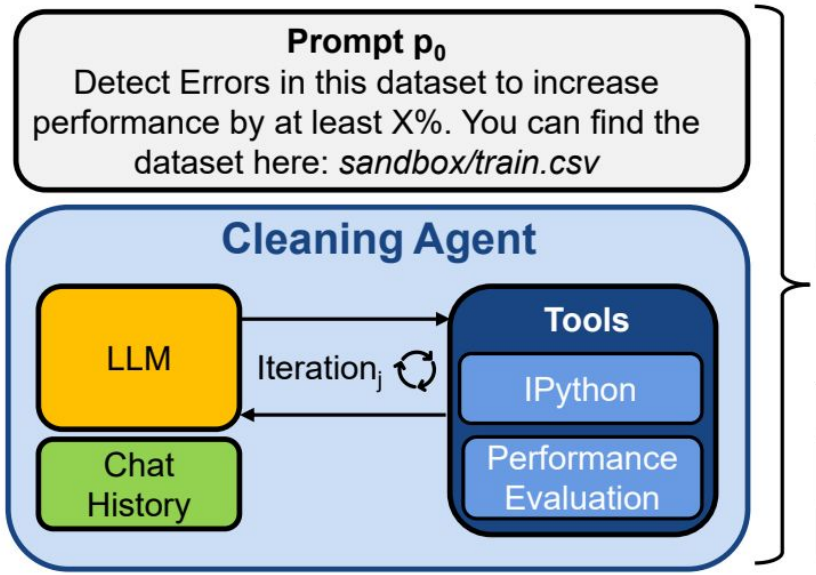
- Instead of returning full DataFrames, generate visualizations per chunk
- AI produces charts, but never serializes the full dataset → recursion safe.

```python
for i, chunk in enumerate(chunks):
    sdf = SmartDataframe(chunk, config={"llm": llm, "enable_cache": False})
    print(f"Plotting chunk {i+1}")
    sdf.chat("Plot Salary distribution by Department")
```



The State-Of-The-Art Research

Prompt-based method



Hint: No errors in *Name*

Dataset$_0$

Name	Sex	Age	Date of Marriage
Mr. Austen	female	35	2045.10.18
Mrs. Austen	female	4	2005.10.18

↓

Dataset$_{Best}$

Name	Sex	Age	Date of Marriage
Mr. Austen	**male**	35	**2005.10.18**
Mrs. Austen	female	**34**	2005.10.18

Figure 1: We provide the model with the path to the dataset along with a prompt instructing it to identify errors so that performance on a held-out set increases by a given threshold. At each iteration j, the LLM can send *code* to IPython to execute and get back the *sys.output* and/or send the *path* of the modified dataset $\mathcal{D}_i$ to get a *performance score*. The loop continues until the cumulative number of tokens used for the entire conversation reaches a pre-defined threshold. All the modified datasets $\mathcal{D}_{0...i}$ are stored and the dataset with the highest score is considered as $\mathcal{D}_{\text{Best}}$.

***Source**: Exploring LLM Agents for Cleaning Tabular Machine Learning Datasets [12]*

Single Agent

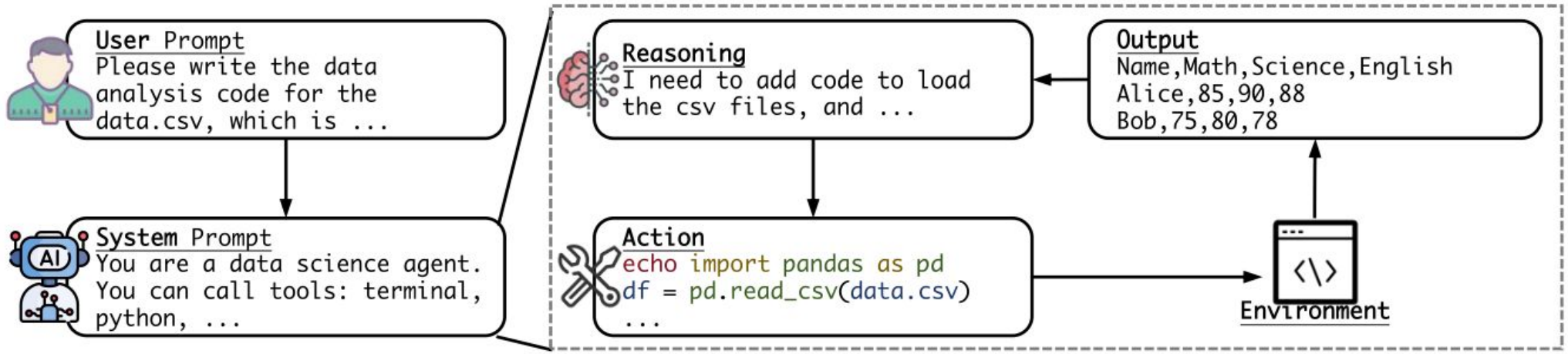


Figure 4: The basic structure for a single agent structure, with only the agent and execution environment.

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

Two Agents

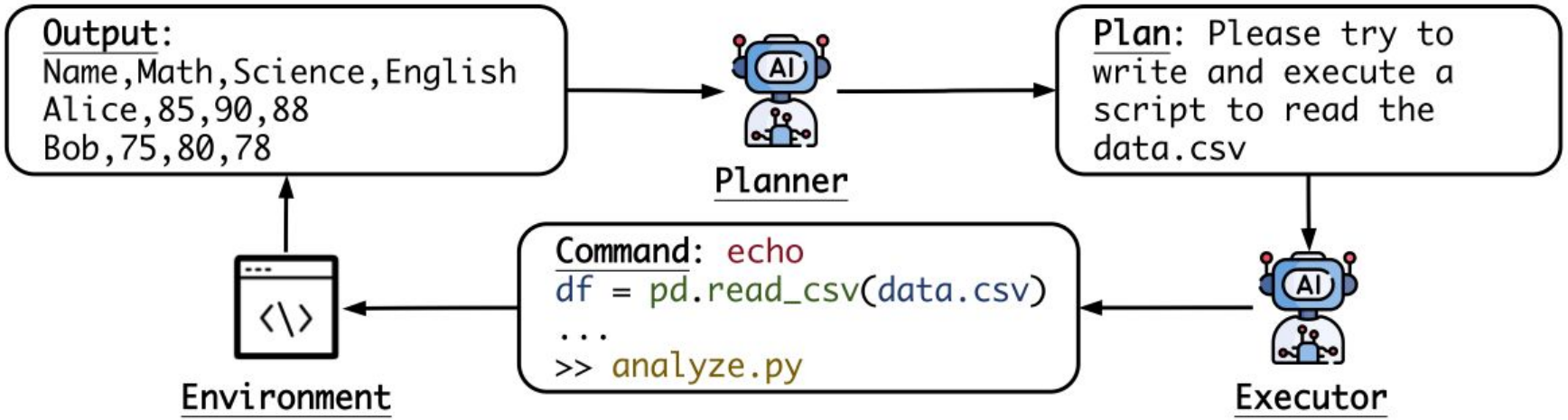


Figure 5: An example of planner and executor agent structure, where the planner generates a plan for the executor to execute in detail.

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

Multiple Agents with Minimal Functions

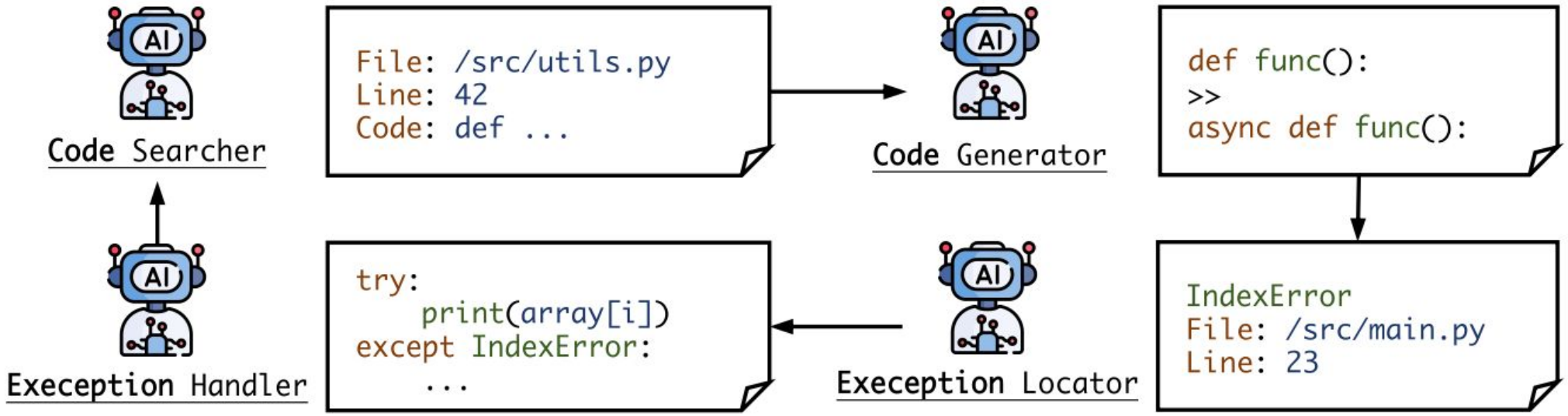


Figure 7: For agents with minimum functions, each agent is only responsible for a minimum function, such as search code, run code, etc.

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

Multiple Agents with Client-Server Architecture

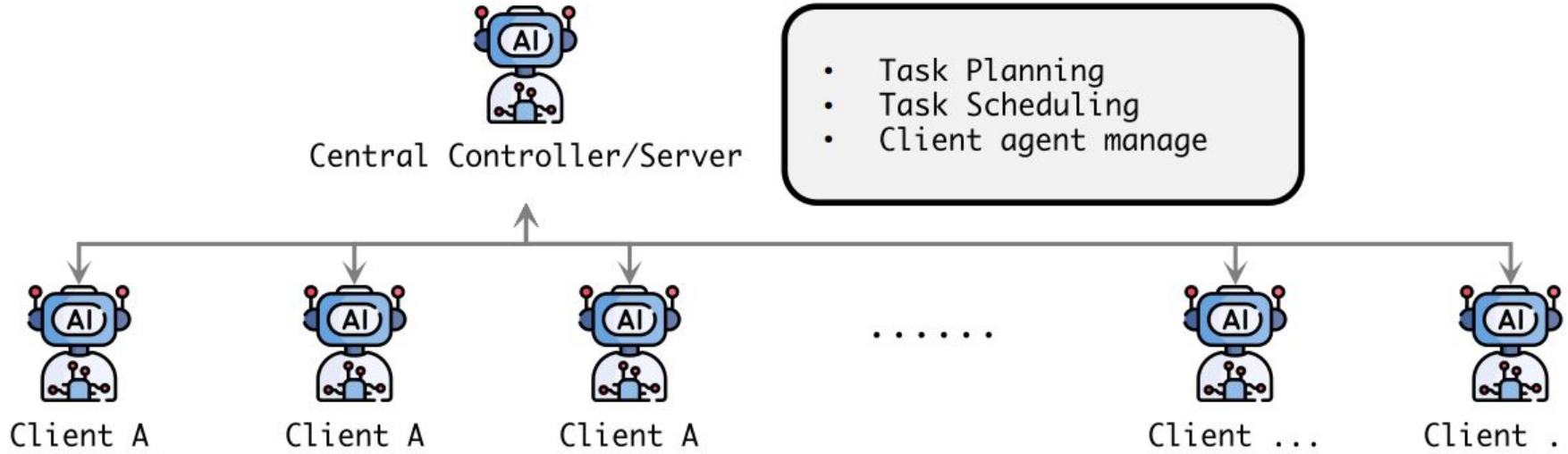


Figure 8: In the client-server agent structure, normally there will be a central server controls all the other clients.

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

Dynamic Agents – Hierarchical Agent Generation

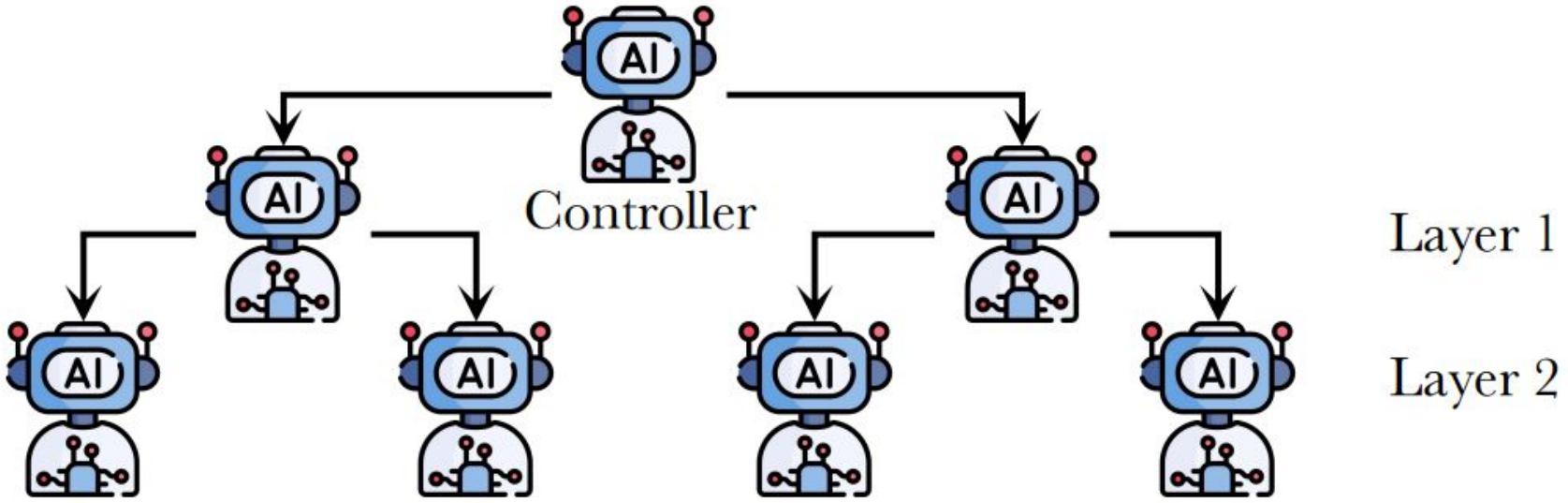


Figure 9: Hierarchical agent generation.

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

Dynamic Agents – Iterative Agent Generation via Feedback

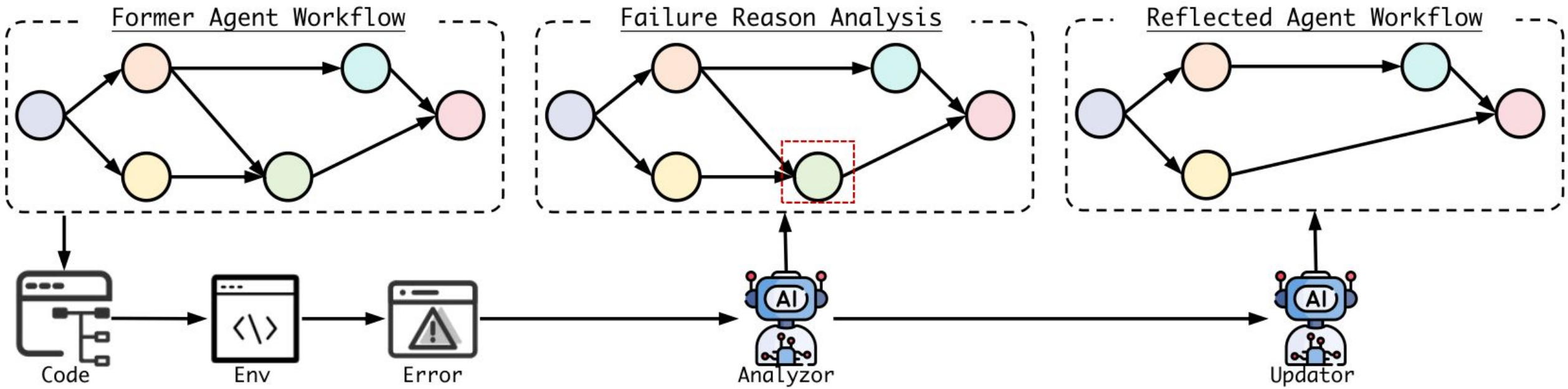


Figure 10: Iterative agent generation through feedback.

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

Summary of Agent Role

Framework	CS	FL	PG	VA	RI	ED	EH
AutoCodeRover	✓	✓	✓	✓			
CODES					✓		
HYPERAGENT	✓			✓			
MASAI	✓	✓	✓	✓			
Seeker						✓	✓

Table 1: This table summarizes the roles of Minimum Function Agents in different frameworks. The columns represent specific functions: Code Search (CS), Fault Localization (FL), Patch Generation (PG), Validation (VA), Repository Initialization (RI), Exception Detection (ED), and Exception Handling (EH). A checkmark (✓) indicates that the framework supports the corresponding function.

Role	PM	RA	AR	SM	TL	ML	FC	DE	SD	QA	TE
AgileCoder	✓			✓				✓	✓		✓
AutoML-Agent	✓									✓	
ChatDev		✓						✓			✓
FlowGen		✓	✓	✓				✓			✓
MetaGPT	✓		✓					✓		✓	✓
MAGIS	✓							✓		✓	
VisionCoder					✓	✓	✓	✓			✓

Table 2: Summary of SE Team Roles in Agent Designs: Product Manager (PM), Requirements Analyst (RA), Architect (AR), Scrum Master (SM), Team Leader (TL), Module Leader (ML), Function Coordinator (FC), Developer (DE), Senior Developer (SD), QA Engineer (QA), Tester (TE).

Trade-off Summary Across Agent Role Designs

Dimension	Single Agent	Two-Agent	Multi-Agent	Dynamic Agents
Reliability	○	◐	●	◐
Scalability	○	◐	●	●
Coordination Cost	●	◐	○	○
Predictability / Stability	◐	◐	●	○
Industrial Applicability	Quick analyses, ad-hoc queries, and low-risk automation	Medium-scale workflows requiring basic verification and predictable execution	Production pipelines with modular stages, quality checks, and stable data dependencies	Exploratory analytics, quick pipeline prototyping, or changeable environment

Table 3: Trade-off summary across agent role designs. ●= strong, ◐= moderate, ○= weak.

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

Typical Data Science Loop

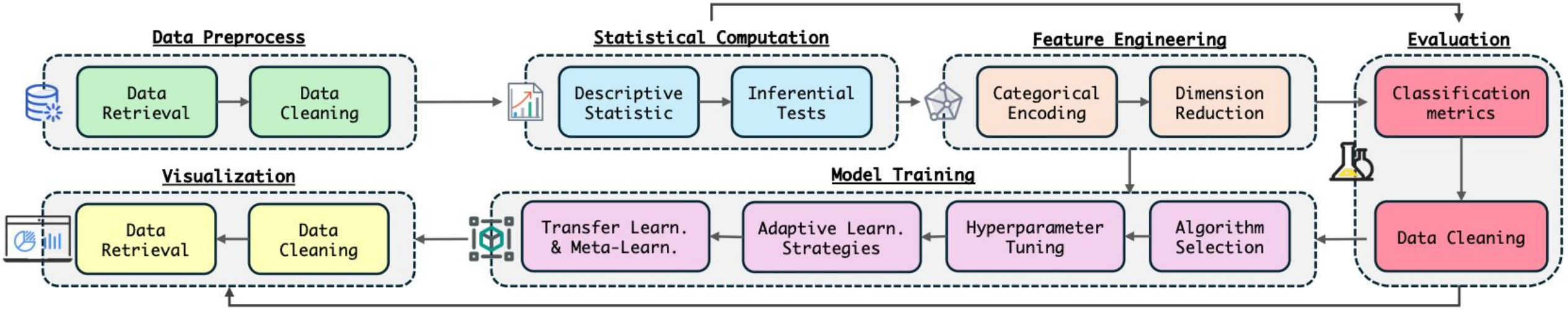
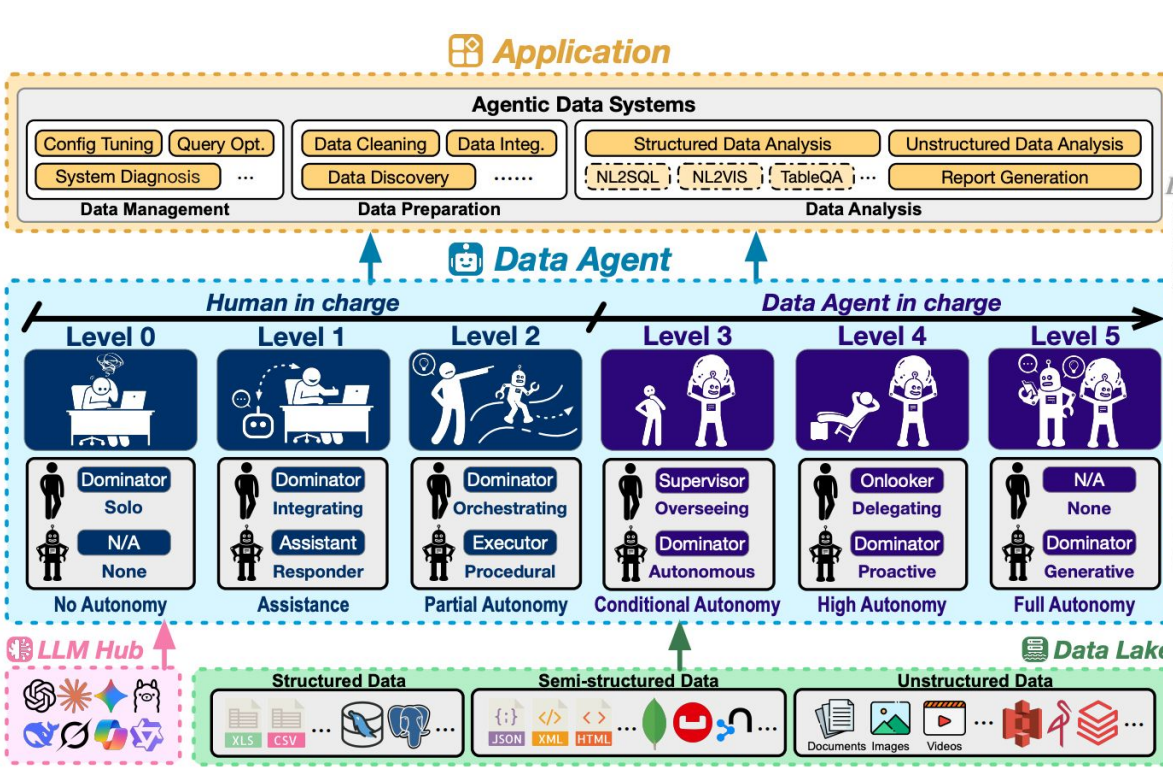
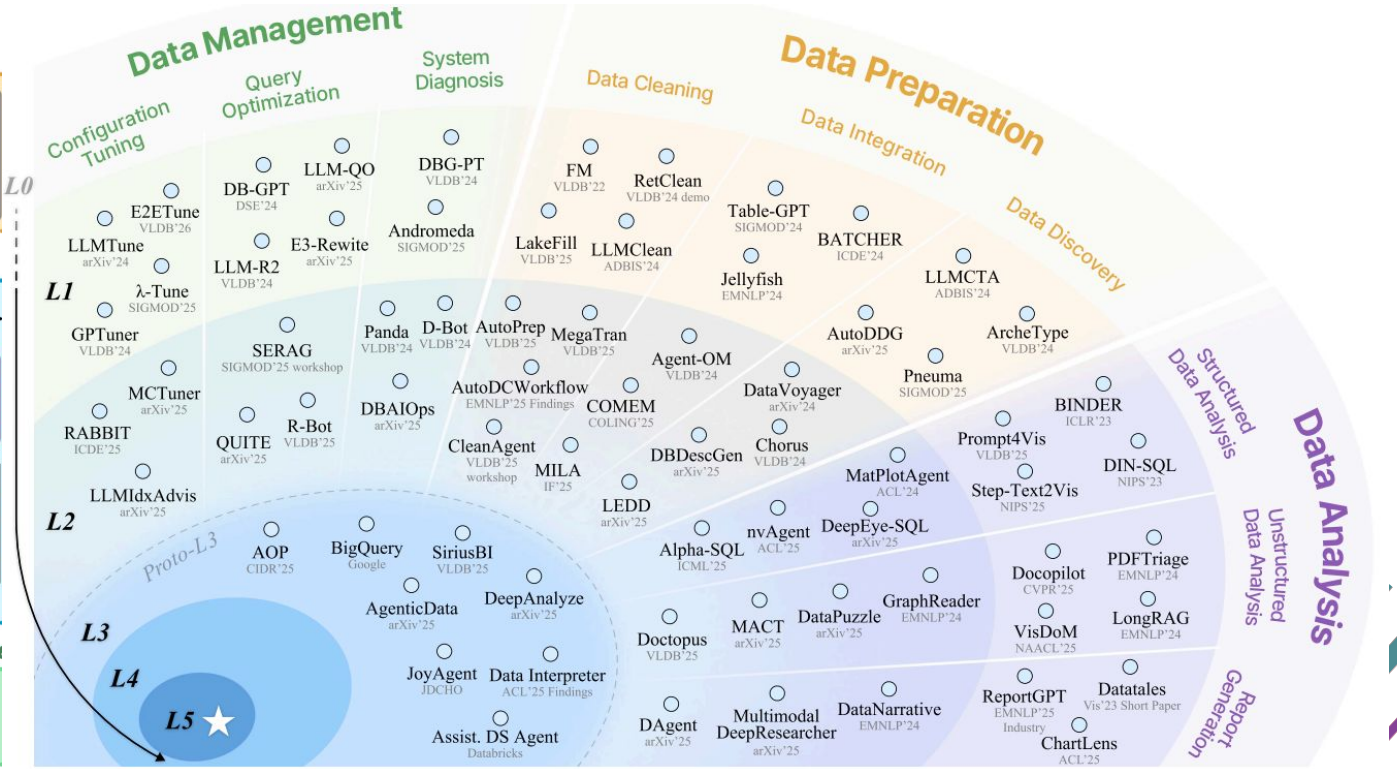


Figure 14: Typical data science loop

***Source**: Large Language Model-based Data Science Agent: A Survey [13]*

An Overview of Data Agents with Six Autonomy Levels





***Source**: A Survey of Data Agents: Emerging Paradigm or Overstated Hype [14]*

Summary

- GenAI in the Data Lifecycle – Foundations and Prompting
  - Why Data Management Matters in Generative AI?
  - Data Science Lifecycle
  - How GenAI Change Data Lifecycle?
  - Focus of this lecture
- From Traditional to Modern: Pandas → PandasAI
  - Data Cleaning and Generation with GenAI
  - Data Analysis, EDA, and Interpretation with GenAI
- GenAI-based Workflow: Integrating GenAI Into Data Pipeline
  - Basic workflow
  - Improvement concerns

Codes used in this lecture:
https://github.com/Jiahuan-Pei/GenAI-Winter-School-L6

Reference

1. Generative AI needs trustworthy Data
2. Data Management Considerations For Generative AI
3. Donald, A., Galanopoulos, A., Curry, E., Muñoz, E., Ullah, I., Waskow, M. A., ... & Kalra, M. (2023). Bias detection for customer interaction data: A survey on datasets, methods, and tools. IEEE Access, 11, 53703-53715.
4. What is the Data Science Lifecycle?
5. The missing data link: Five practical lessons to scale your data products
6. Data 100, Lecture 4 - Pandas III: EDA and Data Cleaning, Part 1
7. Pandas Workshop: [Slides][Code]
8. DATA CLEANING IN PYTHON FOR BEGINNERS
9. Messy-dataset
10. PandasAI Tutorials
11. 25 Best Secret AI Prompts for Data Cleaning You Need to Know Today!
12. Bendinelli, T., Dox, A., & Holz, C. Exploring LLM Agents for Cleaning Tabular Machine Learning Datasets. In ICLR 2025 Workshop on Foundation Models in the Wild.
13. Chen, K., Wang, P., Yu, Y., Zhan, X., & Wang, H. (2025). Large language model-based data science agent: A survey. arXiv preprint arXiv:2508.02744.
14. Zhu, Y., Wang, L., Yang, C., Lin, X., Li, B., Zhou, W., ... & Luo, Y. (2025). A Survey of Data Agents: Emerging Paradigm or Overstated Hype?. arXiv preprint arXiv:2510.23587.
15. Codes used in this lecture

Thank you for your attention!
Q & A

LECTURE TITLE

NAME AND SURNAME
DEPARTMENT
EMAIL ADDRESS

TITLE

TODO

TITLE OF SECTION