Optimizing Data Engineering Through a Comprehensive Exploration of Modern Pipeline Tools

[1]Mukesh Kumar Saini, [2]Arun Saini

[1]Researcher and Solution Architect, [2]Senior Specialist Engineer
Computer Science and Software Engineering,
[1]Wipro Limited, [2]Tata Consultancy Services

ABSTRACT*:* There are currently many pipeline tools available for data engineering, which data scientists use to tackle data wrangling issues and perform tasks ranging from data ingestion and preparation to its use as input for machine learning (ML). These tools either come with essential built-in components or can be integrated with others to achieve the desired data engineering outcomes. While some of these tools are fully or partially commercial, there is also a range of open-source options capable of handling expert-level data engineering tasks. This survey explores various categories and examples of these pipeline tools based on their design and intended data engineering functions. The categories covered include Extract Transform Load/Extract Load Transform (ETL/ELT), Data Integration, Ingestion, and Transformation pipelines, Data Pipeline Orchestration and Workflow Management, and Machine Learning Pipelines. Additionally, the survey outlines how these tools are used within these categories, providing examples and discussing case studies that showcase real-world applications. These case studies illustrate user experiences with sample data, the complexities involved in the pipelines, and summarize approaches for preparing data for machine learning.

***Keywords* -** Artificial Intelligence (AI), Data Engineering, Structured Data, Unstructured Data ETL, Machine Learning, Data Integration, Data Pipeline etc.

I. INTRODUCTION

The growing volume and real-time nature of data have heightened the importance of data wrangling, a critical aspect of data engineering. This field encompasses obtaining, organizing, understanding, extracting, and formatting data to prepare it for analysis. The primary goal of data engineering is to convert raw data into a structured format that is suitable for downstream tasks, such as feeding machine learning models. This process is often tedious and time-consuming, consuming a significant portion of a data scientist's time up to 80% in some cases, as noted in data science and artificial intelligence (AI) projects [1].

The central challenge in data engineering is the lack of a universal solution to automatically transform raw data into the desired format. To address this challenge, various advancements have been developed by both corporate entities and academic institutions. The key is to implement data engineering procedures through a series of semi-automated or automated operations that can effectively convert raw data into structured data. These procedures are often organized into data engineering pipeline frameworks [8,12].

This survey aims to identify and review pipeline tools and frameworks designed to tackle data engineering challenges. These tools and frameworks vary widely, each targeting different aspects of data engineering. Some tools are specifically designed to address particular data issues, while others emphasize user engagement, extensibility, and integration with other tools. Although a one-size-fits-all framework would be ideal, it is impractical because different datasets present unique challenges. Therefore, a more realistic approach is to use frameworks that allow for customization and integration of custom solutions as new data challenges arise.

In this paper, we provide a comprehensive survey of pipeline tools and frameworks that offer solutions for broad data engineering issues. We categorize these tools based on their design and intended use cases:

- **Extract Transform Load (ETL) / Extract Load Transform (ELT) Pipelines:** These pipelines primarily focus on data integration. ETL pipelines extract data from various sources, transform it into a suitable format, and load it into a data warehouse or other storage system. ELT pipelines, on the other hand, extract data, load it into the target system first, and then perform transformations.
- **Pipelines for Data Integration, Ingestion, and Transformation:** These pipelines are designed to address data organization tasks, such as aggregating data from multiple sources, cleansing, and transforming it to create a unified dataset.
- **Orchestration and Workflow Management Pipelines:** These pipelines automate the organization and management of complex workflows, coordinating the execution of various data processing tasks and ensuring efficient data flow.
- **Machine Learning Pipelines:** Specifically designed for machine learning tasks, these pipelines streamline the processes involved in preparing data, training models, and deploying them into production.

Although no single pipeline solution fits all scenarios, categorizing these tools helps data engineers choose appropriate solutions for their specific needs. Additionally, this categorization provides a framework for researching and improving existing solutions.

The paper also includes a review of pipeline tools applied to a specific dataset, the Household Energy Dataset [10]. Through this review, we illustrate how various pipelines address issues related to data organization, quality, and feature engineering, demonstrating their effectiveness in preparing the dataset for machine learning.

The paper is organized as follows: **Section 2** introduces generalized data engineering pipelines and discusses their design principles, **Section 3** outlines the survey methodologies used to evaluate the pipeline tools, **Section 4** provides a detailed discussion of the different pipeline categories based on their functions, **Section 5** presents case studies of selected pipelines, showcasing their application to real-world datasets, **Section 6** explores trends and features of the selected pipelines, highlighting their advancements and areas of improvement, **Section 7** offers recommendations for selecting and using pipeline tools effectively, **Section 8** concludes the paper with a summary of findings and future directions for research.

II. Data Engineering Pipelines

A data pipeline is a series of steps and processes that moves data from its source to its final destination, where it is utilized. For instance, in a data science project, data must be formatted correctly before being fed into a machine learning model. Data pipelines are used to transform data into this required format.

To ensure data is ready for use, it must be captured, ingested, and integrated into the pipeline. This involves managing the flow of data through various stages, including capturing it from different sources, integrating it, and preparing it for its intended purpose. Effective data pipeline management requires well-coordinated data orchestration and workflow management to ensure that data flows seamlessly and is prepared appropriately for its final use.

When designing an effective data pipeline framework, several questions need to be addressed: What type of data is being handled and in what format? Can the data be read, and if so, how should it be organized? How can data quality issues be managed, including the handling of missing data? Is it possible to engineer features to enhance the data's usefulness?

Data pipelines can follow different logical processes depending on the use case. For instance, an ETL (Extract, Transform, Load) pipeline extracts raw data from various sources, transforms it into the necessary format, and then loads it into a destination system. In contrast, ELT (Extract, Load, Transform) is preferred for big data scenarios. In this approach, raw data is extracted and loaded into data warehouses or data lakes before being transformed.

Given that data often comes from multiple sources and in various formats, data integration—combining data from heterogeneous sources—is a crucial step [12]. The field of data integration has continually evolved with technological advancements, enhancing both artificial intelligence and data management. These improvements have significantly benefited data engineering pipeline frameworks, making them more effective at handling diverse and complex data environments [2].

This schematic illustrates an ideal data engineering pipeline, depicting the entire process from data sourcing to final output.

- **Data Sourcing:** The pipeline receives data from multiple sources, which can be either in batch mode (processed at specific intervals) or in real-time streaming data.
- **Data Processing:** This stage encompasses all activities that occur between the reception of data and its final output. The aim is to address various data wrangling issues, such as data cleaning, transformation, and integration. Although the pipeline is designed to handle all these issues, in practice, some challenges may be resolved while others might persist.
- **Output:** Once data processing is complete, the data is either stored for future use or immediately consumed for purposes such as visualization and machine learning.

This figure highlights the key components and stages of an ideal data pipeline, emphasizing the need for effective data handling and processing to ensure that data is appropriately prepared for its intended applications.

In an ideal design, a data pipeline should be highly adaptable and capable of handling various data scenarios efficiently. Ideally, it should seamlessly ingest raw data inputs, whether in batches—processed at specific times or triggered by certain conditions—or in real-time streams. The pipeline should effectively manage data through various stages of the system, as illustrated in Figure 1.

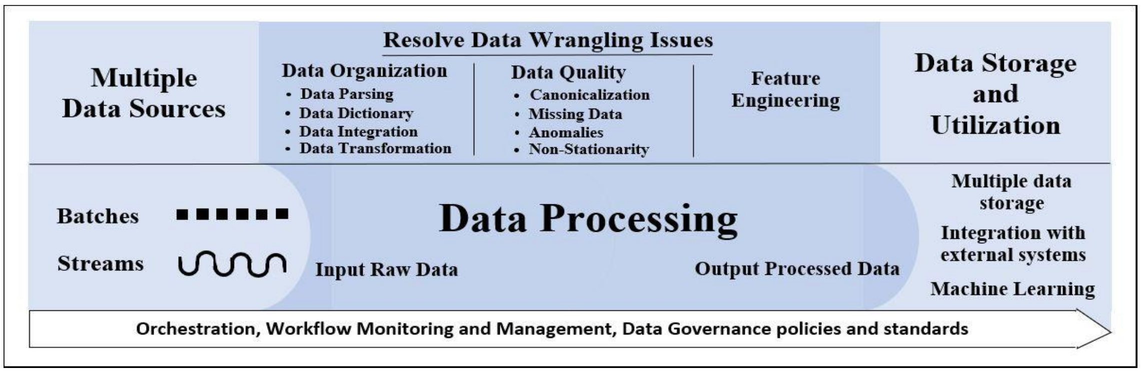

Figure 1: An Ideal Pipeline for Data Engineering [5])

A well-designed pipeline must be flexible enough to accommodate changes in data volume or requirements. For instance, if a pipeline originally designed for small data needs to handle large volumes of data, it should automatically scale up to manage the increased load. Conversely, it should also be able to scale down when necessary. This flexibility ensures that the pipeline remains operational and efficient despite changes in data demands.

The support levels for data engineering pipelines vary significantly. Some pipelines offer basic frameworks, while others provide plug-and-play components and features. For example, TFX (TensorFlow Extended) offers preloaded components for machine learning pipelines. Ideally, a data pipeline would include preloaded and scalable components at every stage to adapt to evolving data engineering needs.

The pipeline must handle data volumes smoothly, regardless of speed, volume changes, or data diversity. It should process both batch and streaming data, executing all necessary data processing operations to resolve data issues and ensure completeness. Once processing is complete, the pipeline should output the data to storage, enable integration with other systems, or make it available for machine learning applications.

An ideal data pipeline would maintain end-to-end automation and monitoring, adhering to data governance standards. While a perfect fit-for-all pipeline that addresses all data wrangling issues end-to-end would be ideal, practical challenges remain regarding pipeline operability, component compatibility, and handling diverse data types. Currently, pipelines are often designed to address specific data wrangling issues. Combining multiple pipelines into a cohesive framework can provide enhanced benefits and address a broader range of data engineering tasks.

III. Discussion

The survey methodology for evaluating tools, pipelines, and frameworks began with a preliminary scan of the web to grasp the general landscape of available resources and to understand the discourse from both technical and non-technical communities. This initial scan helped identify various pipeline resources, distinguishing between those that are open and freely available and those that are proprietary with a commercial focus.

Further insights were gained by examining the documentation available on dedicated websites, including commercial sites for closed-source tools. This review provided detailed information about the development of these pipelines, including introduction dates, supported languages, and general applicability.

Additional understanding came from analyzing developer feedback, sales pitches, and promotional materials from commercial entities, which highlighted the technical and commercial aspects of different pipelines for data engineering tasks. The information helped explain how these tools could be utilized by both technical and non-technical users for data organization, data quality, and feature engineering. Some tasks required prior technical knowledge and depended on the tools' built-in capabilities, customization options, or integration with other tools.

In-depth technical surveys were conducted by exploring GitHub repositories dedicated to these tools. User communities and specific projects provided insights into the pipeline ecosystem. Technical publications further enriched the understanding by offering formal applications and discussions within the technical community. The survey aimed to provide a balanced view that would be accessible to both technical and non-technical audiences. It sought to help users understand available pipelines and engage in hands-on experience with a tool using sample datasets. Over 40 sources were reviewed, including tool materials, websites, promotional content, documentation, and technical papers. Out of these, four pipelines were selected for practical testing to showcase their capabilities and functionalities.

The selection criteria included whether the tool was open-source (O) or closed-source (C), ease of use, and capabilities as described by technical, commercial, and public user experiences. The chosen pipelines were Apache Spark, Microsoft SQL Server Integration Services, Apache Airflow, and TFX (TensorFlow Extended).

These four tools represent different categories of data engineering pipelines: (a) ETL/ELT Pipelines, (b) Data Integration and Ingestion, (c) Pipeline Orchestration and Workflow Management, and (d) Machine Learning and Model Deployment. While some tools may fit into multiple categories, the survey selected one tool per category for practical engagement. The goal was to offer a general overview of how data engineering pipelines can be designed and to support users in performing data engineering tasks. The survey does not aim to provide a comparative analysis but rather an overview of the pipelines' availability, broad usability, and initial user experiences, as detailed in Section 5.1.

IV. Clusters of Data Pipelines

Data pipelines can be categorized based on several factors. One way to classify them is by the processing approach:

- **Batch Pipelines:** These pipelines process data at specific, predefined times or conditions, transferring data from source to destination in discrete chunks.
- **Streaming Data Pipelines:** An advancement over batch processing, these pipelines handle real-time data, processing it continuously as it arrives to meet the demand for immediate information and results.

Given the rapid pace of data generation and the need for scalability, pipelines can also be categorized by their operating environment:

- **Local or On-Premises Pipelines:** These are deployed and managed within a local infrastructure.
- **Cloud-Based Pipelines:** These pipelines are designed for deployment and operation in cloud environments, accommodating the need for scalable and flexible operations.

Additionally, pipelines can be grouped by their key functions:

- **ETL/ELT Data Pipelines:** These pipelines focus on Extracting, Transforming, and Loading (ETL) or Extracting, Loading, and Transforming (ELT) data.
- **Integration, Ingestion, and Transformation:** Pipelines in this category manage data integration, ingestion, and transformation processes.
- **Pipeline Orchestration & Workflow Management:** These pipelines handle the coordination and management of complex workflows.
- **Machine Learning and Model Deployment:** Pipelines designed for deploying and managing machine learning models.

These classifications help in understanding the diverse capabilities and use cases of different data pipelines.

4.1 ETL/ELT Data Pipeline

ETL pipelines are designed to integrate data by extracting it from various sources, transforming it, and then loading it into a destination. In contrast, ELT pipelines, commonly used for big data, first load the extracted data into the destination before performing any transformations. A range of ETL/ELT pipeline tools and frameworks is available, as summarized in Table 1. This table includes information on the introduction year of these tools/frameworks/platforms, whether they are open-source (O) or closed-source (C), the primary programming languages they support, and their capabilities, including built-in (B) features or the ability to integrate (I) with other tools [4,7.11].

Notable ETL/ELT Pipeline Tools:

- **Apache Spark**: An open-source platform supporting multiple languages including Python, Java, SQL, Scala, and R. It facilitates data engineering, data science, and machine learning tasks with distributed and scalable parallel data processing. Spark is well-suited for large-scale big data query and analysis, offering built-in features for data wrangling and graph processing through Directed Acyclic Graphs (DAG). It operates effectively in both local and cloud environments, supporting batch and real-time processing. Despite its advantages in parallel processing and flexibility, Spark can experience reliability issues and performance degradation when handling long DAGs, making it prone to failures [3,4].
- **AWS Glue**: A closed-source, serverless ETL and ELT service that simplifies the creation, monitoring, and management of data pipelines. It supports integration with various tools for machine learning, analytics, and application development, and operates with multiple backend languages such as Apache Spark, Ray, and Python. AWS Glue features a visually interactive interface, codeless functions, and connectivity with notebooks and integrated development environments (IDEs). It provides data quality metrics, monitoring, and management, addressing some data quality issues.
- **Apache NiFi**: An open-source tool that also focuses on data quality, monitoring, and management. It provides extensive data integration capabilities, akin to those of AWS Glue.
- **Striim**: This tool offers built-in artificial intelligence (AI) capabilities for real-time data integration and streaming. SAS Data Integration Studio: Known for its intuitive, visual, and collaborative interface for data integration, making it user-friendly for managing complex data integration tasks.

Each of these tools has unique features and capabilities tailored to different data engineering needs and environments.

4.2 Integration, Ingestion and Transformation

Data integration, ingestion, and transformation pipelines, as detailed in Table 2, address key data engineering challenges such as organizing data from diverse sources, handling heterogeneous data, and integrating various data aspects to meet specific requirements. These pipelines are essential for obtaining, integrating, and making data available, thereby addressing concerns related to data governance and management. With the influx of large, complex, and varied data from multiple sources, these pipelines play a crucial role in data consolidation and transformation [7,13,14].

Table 1: ETL / ELT Pipelines

Tools/Framework/Platform	Year of Introduction	Source	Primary Language	Data Organisation				Data Quality					Processing Format				Environment	
				Data Parsing	Data Dictionary	Data Integration	Data Transformation	Canonicalization	Missing Data	Anomalies	Non-Stationarity	Feature Engineering	ETL	Batch	Streaming	DAG / Graph	Local	Cloud
Apache Spark	2010	O	Scala, Java	B	I	B	I	I	I	I	I	B	I	B	B	B	B	B
Apache Flink	2010	O	Java	I	I	I	I	I	I	I	I	I	I	I	B	B	B	B
Apache Griffin	2016	O	Scala	I	I	I	I	B	B	B	B		I			I	B	B
Luigi	2012	O	Python	I	I	I	I	I	I	I	I	I	B	B		B	B	I
Dagster	2019	O	Python	I	I	I	I	I	I	I	I	I	I	I	I	B	B	B
Kedro	2019	O	Python	I	I	I	I	I	I	I	I	I	B	B	I	B	B	I
DBT (Data Build Tool)	2016	O	SQL(Jj)			I	B	I					B	I		B	B	I
Bonobo	2013	O	Python			I		I					B			B	B	
AWS Glue	2017	C	Python, Scala	B		B	B	I					B	B	I	B		B
Snowflake Data Pipelines	2014	C	SQL			I	B	I					B	I		I		B
Google Cloud Dataflow	2015	C	Java, Python			B		I					B	B	B	B		B
Databricks Delta Lake	2017	O	Scala, Python			I		I					B	I		B	B	B
StreamSets	2015	O	Java, Python			I		I					B	B	B	B		B
Dataiku	2013	C	Python, R		B	I		I	I				B	I		B	B	B
Alteryx	2010	C	Py, R, SQL, Ax		B	I		I					B			B		B
SAS Data Integration Studio	2005	C	SAS	B										B		B	B	
Striim	2012	C	SQL, XML, JSON, JS			B										B		B

a. Py: Python; Jj: with Jinja templating; JS: JavaScript; Ax: Alteryx Expression Language
Within the capabilities, B: Built-in capabilities; I: Integrates with other tools to achieve.

Key Tools for Data Integration, Ingestion, and Transformation [8.9]:

- **Apache Kafka**: This open-source platform supports multiple programming languages including Java, Python, Scala, and Go (https://kafka.apache.org/). It excels at ingesting data from various sources and transforming it during processing. Apache Kafka operates in both local and cloud environments, ensuring data security, fault tolerance, and scalability to handle real-time data processing demands. It processes data in batches and supports high-performance distributed event streaming, making it suitable for real-time event streaming with high speed and low latency. Despite its advantages—such as reliable, flexible, efficient, and scalable real-time processing—Apache Kafka has a steep learning curve and complexity in setup, operational monitoring, and support.

Table 2: Integration, Ingestion, and Transformation

Tools/Framework/Platform	Year of Introduction	Source	Primary Language	Data Organisation				Data Quality					Processing Format				Environment	
				Data Parsing	Data Dictionary	Data Integration	Data Transformation	Canonicalization	Missing Data	Anomalies	Non-Stationarity	Feature Engineering	ETL	Batch	Streaming	DAG / Graph	Local	Cloud
Apache Kafka	2011	O	Sc, Jv, Py, Go	I	I	B	I	I	I	I	I	I	I	B	B	I	B	B
Apache Gobblin	2017	O	Java	I	I	B	I	I	I	I	I	I	B	B	I	I	I	B
Singer	2017	O	Python			B							B	B		I	B	
Stitch	2016	C	SQL			B										B		B
Segment	2012	C	JS, Py, Jv, JSON			B										B		B
Fivetran	2012	C	SQL			B										B		B
Apache NiFi	2014	O	Java	B		B	B							I	B	B	B	I
Talend Data Integration	2006	O	Java	B		B	B	I					B	B		B	B	B
Informatica PowerCentre	1993	C	IPL			B	B							B		B	B	I
IBM InfoSphere DataStage	1996	C	IDL		B	B								B		B		B
Microsoft SSIS	2005	C	SQL, .NET			B								B		B	B	
SAS Data Integration Studio	2005	C	SAS	B										B		B	B	
Oracle Data Integrator (ODI)	1996	C	ODI Language			B								B	B	B	B	
Qlik Replicate (AR)		C				B									B	B	B	
SnapLogic	2006	C	Py, Jv, Ruby, JS			B										B		B
Xplenty	2011	C	SQL, Python, JS			B										B	B	
Matillion	2011	C	Python			B										B		B
Striim	2012	C	SQL, XML, JSON, JS			B										B		

a. Microsoft SSIS:Microsoft SQL Server Integration Services; (AR): (formerly Attunity Replicate)
b. Py: Python; IPL: Informatica PowerCenter language; JS: JavaScript; IDL: InfoSphere DataStage Language; Sc: Scala; Jv: Java
Within the capabilities, B: Built-in capabilities; I: Integrates with other tools to achieve.

- **Microsoft SQL Server Integration Services (SSIS)**: A closed-source platform (with some open-source components) designed for building ETL, data integration, and transformation workflows. It supports SQL and .NET languages (https://learn.microsoft.com/en-us/sql/integration-services). SSIS can ingest data from multiple files and relational sources, transforming and loading it into various destinations. It operates both on-premises and in the cloud. Featuring a graphical interface, SSIS offers customization options and ease of use, whether with or without coding, and can be extended to integrate with other tools for enhanced functionality.

These tools provide essential functionalities for managing complex data workflows, integrating various data sources, and addressing data governance and management issues.

.4.3 Orchestration and Workflow Management

Table 3 highlights tools, frameworks, or platforms used for Pipeline Orchestration and Workflow Management. These systems are designed to centralize the automation, execution, monitoring, and management of workflows across data pipelines, from data sourcing through to utilization. They ensure that interconnected pipelines or phases of a pipeline are executed in a coordinated and orderly manner, facilitating end-to-end data processing.

- **Apache Airflow**: An open-source tool designed for data pipeline orchestration and workflow management. It features a web interface for managing ETL and data integration workflows, supports Python, and handles automated workflow orchestration, scheduling, and monitoring. Apache Airflow allows for the parallel or sequential execution of tasks, visualizing workflows using Directed Acyclic Graphs (DAGs) to indicate the status of data processing (https://airflow.apache.org/). It can be used as a standalone tool or deployed to the cloud and primarily processes data in batches. It can also integrate with streaming frameworks like Apache Kafka to extend its capabilities. While Airflow is highly extensible and integrates with various tools and APIs, it presents a steep learning curve for initial setup and operation.

- **Apache Beam**: An open-source framework that supports multiple languages, including Python, Java, SQL, and Go. Apache Beam is designed for parallel and distributed processing of big data pipelines, enabling scalable and flexible data processing (https://beam.apache.org/).
- **Microsoft Azure Data Factory**: A closed-source, cloud-based service for data pipeline orchestration and workflow management, specifically tailored for the Azure ecosystem. It supports Azure-specific languages and provides robust tools for managing data workflows in a cloud environment (https://azure.microsoft.com).

These tools facilitate the automation and management of complex data workflows, ensuring efficient and reliable end-to-end data processing.

Table 3: Orchestration and Workflow Management Pipelines

Tools/Framework/Platform	Year of Introduction	Source	Primary Language	Data Organisation				Data Quality					Processing Format			Environment		
				Data Parsing	Data Dictionary	Data Integration	Data Transformation	Canonicalization	Missing Data	Anomalies	Non-Stationarity	Feature Engineering	ETL	Batch	Streaming	DAG / Graph	Local	Cloud
Apache Airflow	2014	O	Python		I	I	I	I	I	I	I	I	B	B	I	B	B	B
Apache Beam	2016	O	Py, Jv, SQL, GO	I	I	B	I	I	I	I	I	I	B	B	B	B	B	B
Microsoft Azure Data Factory	2015	C	AL	B										B	I	B		B
Zapier	2011	C	JavaScript													B		B

a. Py: Python; Jv: Java, AL: Azure-specific Languages
Within the capabilities, B: Built-in capabilities; I: Integrates with other tools to achieve.

4.4 Machine Learning and Model Deployment

The output from a data engineering pipeline serves as input for various processes, including machine learning pipelines. These pipelines, which are detailed in Table 4, focus on optimizing input data and efficiently managing the data through iterative phases of model development, training, deployment, and monitoring. Implementing automatic machine learning (AutoML) across the entire iterative process for a given dataset poses significant challenges, as it demands a high level of sophistication and expertise.

Table 4: Machine Learning and Model Deployment Pipelines

Tools/Framework/Platform	Year of Introduction	Source	Primary Language	Data Organisation				Data Quality					Processing Format				Environment	
				Data Parsing	Data Dictionary	Data Integration	Data Transformation	Canonicalization	Missing Data	Anomalies	Non-Stationarity	Feature Engineering	ETL	Batch	Streaming	DAG / Graph	Local	Cloud
TFX (TensorFlow Extended)	2017	O	Python							I	I	B			I	I		B
Kubeflow	2017	O	Py, C++, JS												I	I		B
Metaflow	2018	O	Python											B	I	B		B
ZenML	2020	O	Python													B		B
MLRun	2019	O	Python												I	I		B
CML		O	Python			B										I		B
Cortex Lab		O	Python													B		B
Seldon Core	2016	O	Python, Go												I	I		B
AutoKeras	2019	O	Python									I				I		B
H2O AutoML	2012	O	Python, R							I		I				I		B

a. Py: Python; JS: JavaScript;
Within the capabilities, B: Built-in capabilities; I: Integrates with other tools to achieve.

Just as orchestration is crucial for data engineering pipelines to ensure process continuity and prevent failures, it is equally important in machine learning pipelines. Effective orchestration helps maintain the flow of processes, while optimization ensures that the model functions as intended. Continuous monitoring is essential to detect any model drift early on. Ideally, managing the entire ML pipeline—from dataset handling to model deployment—should be automated whenever possible [6,12].

Two notable tools/frameworks for machine learning and model deployment are TFX (TensorFlow Extended) and AutoKeras.

TFX is an open-source machine learning pipeline platform developed by Google. Based on TensorFlow and supporting Python, TFX can interface with APIs in C++, JavaScript, and Java (https://www.tensorflow.org/tfx/guide). It provides a robust foundation for constructing end-to-end ML pipelines by combining standard components in a sequential manner, with the option to customize through additional components. TFX is extensible with built-in libraries, scalable for various ML tasks, and integrates with tools like Apache Airflow and Kubeflow for orchestration [6].

V. Case Study on household energy dataset

The Household Energy Dataset encompasses data from 255 homes, collected over a period of 23 months. It includes sensor readings for electricity and gas, as well as supplementary data and documentation that provide additional context [10]. The dataset is available online and is organized into .csv files that are compressed in .gz and then .zip formats.

Each sensor reading file is named with identifiers for the home, room, sensor, sensor type, and the type of reading. Additional information may be available through metadata records. The dataset includes over 16,000 .csv files. Analysis reveals several data wrangling challenges, such as missing data, anomalous readings, inconsistent feature spellings, and absent primary keys for table joins.

5.1 Evaluating Framework Tools

The Household Energy Dataset was assessed using one representative tool from each category of tools, frameworks, and platforms to illustrate how various solutions can address data engineering issues. Given the variety of challenges across different subsets of the dataset, specific samples of household sensor data and metadata were selected to showcase the application of these tools. The process involved downloading and unzipping the data files (.zip or .gz), examining the data structure, and performing data engineering tasks to improve data organization, quality, and feature extraction, preparing it for machine learning.

The following tools were reviewed in their respective categories:

- **ETL/ELT Data Pipelines:** Apache Spark with PySpark in Jupyter Notebook
- **Integration, Ingestion, and Transformation:** Microsoft SQL Server Integration Services (SSIS)
- **Orchestration and Workflow Management:** Apache Airflow
- **Machine Learning and Model Deployment:** TensorFlow Extended (TFX)

These tools were chosen based on their design purposes, effectiveness in resolving data wrangling issues, support for custom additions, integration capabilities, sophistication, usability, and popularity within the user community.

5.2 Pipeline with Apache Spark

The Apache Spark environment was configured in Jupyter Notebook using PySpark, the Python API for Spark. A Spark session was initialized with SparkSession.builder. The dataset metadata and sensor data were downloaded using the Python requests library. Given the large size of the household sensors data (over 15GB), a sample was used for demonstration.

Data parsing involved unzipping the metadata and survey.zip dataset with zipfile, and extracting household_sensors.csv.gz using gzip. The .csv files were then read into Spark DataFrames using spark.read.csv. Initial data exploration was conducted with the show() method.

Data organization, integration, and feature engineering tasks included:

- Assigning feature names and data types using StructType, StructField, TimestampType, and IntegerType based on the data dictionary.
- Creating a new column for filenames using withColumn() and regex operations.
- Integrating sensor readings for a specific home using union().
- Printing schema with .printSchema() and converting to Pandas DataFrame for statistical analysis with toPandas().
- Performing data deduplication with exceptAll and dropDuplicates.
- Renaming columns with withColumnRenamed() and transforming data with withColumn(), when(), and otherwise().
- Splitting column names, dropping undesired features, and joining DataFrames.
- Indexing categorical features with StringIndexer and applying transformations.
- Extracting datetime parts and saving the DataFrame to PostgreSQL using write.format(), mode, option, and save.
- Reading tables from PostgreSQL with read.format(), option, and load.
- Combining feature values into a single feature with VectorAssembler.

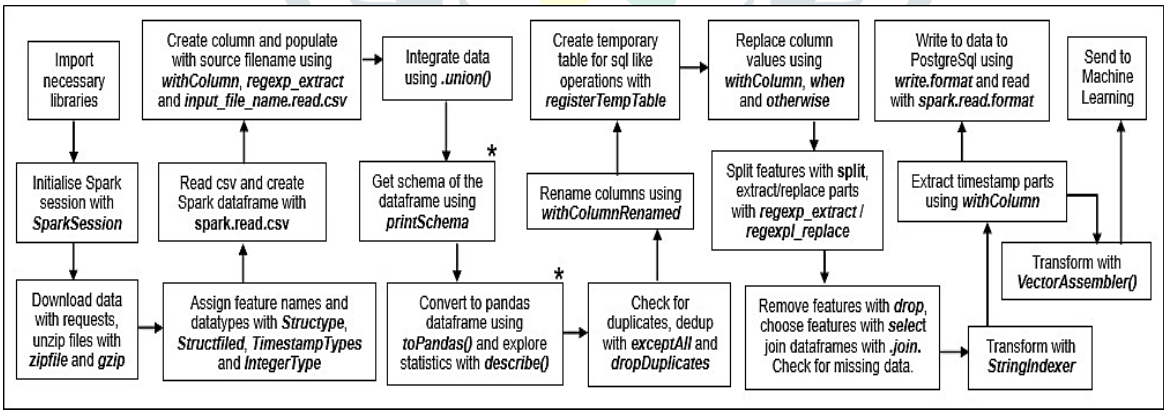
Import necessary libraries
Initialise Spark session with SparkSession
Download data with requests, unzip files with zipfile and gzip
Assign feature names and datatypes with Structype, Structfiled, TimestampTypes and IntegerType
Read csv and create Spark dataframe with spark.read.csv
Create column and populate with source filename using withColumn, regexp_extract and input_file_name.read.csv
Integrate data using .union()
Get schema of the dataframe using printSchema
*
Convert to pandas dataframe using toPandas() and explore statistics with describe()
*
Check for duplicates, dedup with exceptAll and dropDuplicates
Rename columns using withColumnRenamed
Create temporary table for sql like operations with registerTempTable
Replace column values using withColumn, when and otherwise
Split features with split, extract/replace parts with regexp_extract / regexpl_replace
Remove features with drop, choose features with select join dataframes with .join. Check for missing data.
Transform with StringIndexer
Extract timestamp parts using withColumn
Write to data to PostgreSql using write.format and read with spark.read.format
Transform with VectorAssembler()
Send to Machine Learning

Figure 2 provides a summary of the Apache Spark pipeline flow.

5.3 Pipeline with Microsoft SSIS

The SSIS pipeline was set up in Visual Studio and connected to a local SQL Server instance. SSIS, used for data integration and transformation, involved several built-in and custom tasks within the Control Flow component.

Key steps included:

- Using the Execute Process Task to run Python code for downloading and unzipping files.
- Linking the resulting .csv files to a Data Flow Task for configuring connections and previewing data.
- Using the Sort component to define order for data integration via the Merge Join component.

- Applying data transformations and standardizations with the Data Conversion component.
- Selecting columns with Copy Column and loading data into SQL Server with the SQL Server Destination component.

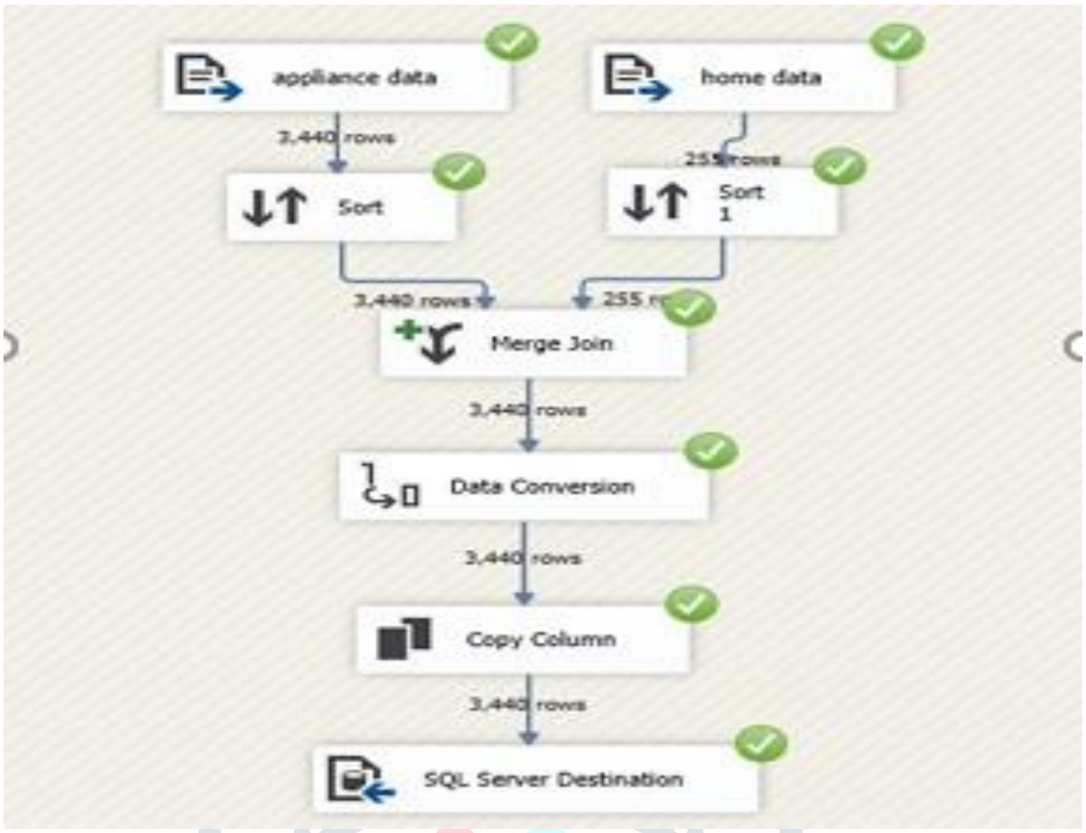

Figure 3 illustrates the SSIS pipeline processing the Household Energy dataset metadata.

5.4 Pipeline with Apache Airflow

The Apache Airflow environment was set up to interact with a PostgreSQL database managed via DBeaver. The airflow components were initialized by starting the scheduler and webserver.

An Airflow Directed Acyclic Graph (DAG) was created to manage the pipeline tasks. The tasks included:

- Using the PythonOperator to download and extract .csv files.
- Loading raw .csv data into PostgreSQL tables with PostgresHook.
- Performing feature extraction and data cleaning.
- Managing task execution and orchestration automatically, with error logging and recovery features.

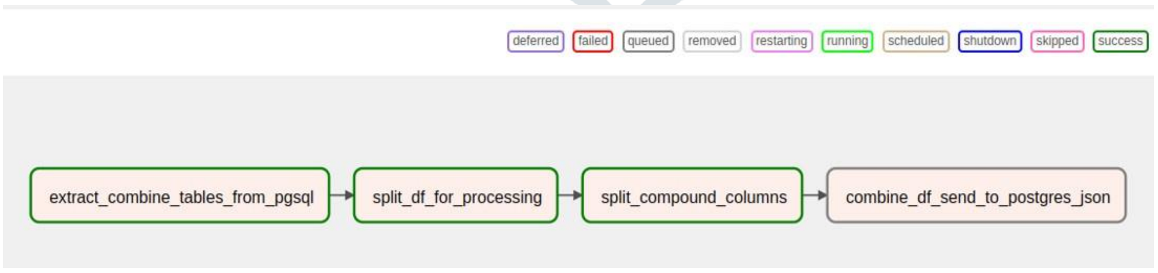

Figure 4 depicts the Airflow DAG for processing the IDEAL Household Energy Dataset.

5.5 Pipeline with TFX (TensorFlow Extended)

The TFX pipeline was configured in Google Colab to utilize the required packages. The pipeline included:

- Downloading and extracting the data using Python libraries.
- Parsing the data into TFX format with the CsvExampleGen component, splitting data into training and evaluation sets.
- Using StatisticsGen to obtain data statistics, SchemaGen to generate schemas, and ExampleValidator for data quality checks.
- Performing feature extraction with the Transform component, which required a custom Python script for data transformation.

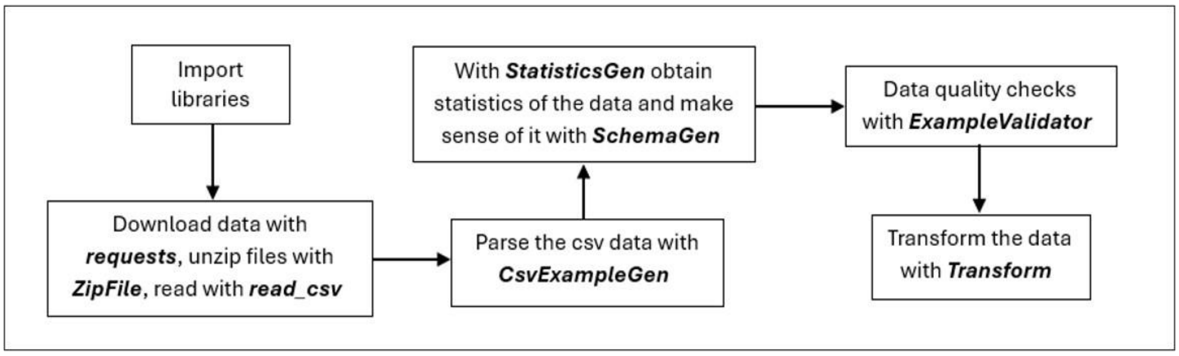
Import libraries
Download data with ***requests***, unzip files with ***ZipFile***, read with ***read_csv***
Parse the csv data with ***CsvExampleGen***
With ***StatisticsGen*** obtain statistics of the data and make sense of it with ***SchemaGen***
Data quality checks with ***ExampleValidator***
Transform the data with ***Transform***

Figure 5 An Example TensorFlow Extended pipeline

VI. Discussion of Trends in Features

Modern data engineering tools and pipelines offer a variety of built-in components to tackle challenges in data organization, quality, and feature engineering. These components are frequently updated to address evolving data management needs. Moreover, many tools provide customization options, allowing users to build tailored solutions. Despite exploring several tools, the case study highlights that many components within each tool remain underutilized.

Microsoft SQL Server Integration Services (SSIS): SSIS excels with its user-friendly interface, which includes descriptive task and component labels, making it easier to understand their functionality before use. However, setting up SSIS involves several steps, including installation, configuration, and managing processed data storage. SSIS's graphical user interface and troubleshooting logs simplify problem resolution and enhance usability.

Apache Spark with PySpark in Jupyter Notebook: Apache Spark's integration with Jupyter Notebook offers a straightforward setup and efficient data processing using Python. The ease of installation and use is offset by the need for precise coding. Unlike SSIS's drag-and-drop interface, Spark requires users to write code, but it offers flexibility and power for data manipulation and processing.

Apache Airflow: Apache Airflow provides a robust solution for managing data pipelines, though it has a steep learning curve. Initial setup and configuration can be challenging, especially for those unfamiliar with the tool. However, once configured, Airflow's Python-based interface, extensive integration capabilities, and graphical DAG interface offer powerful, customizable, and flexible workflow management. Its ability to integrate with multiple tools and provide isolated environments for development and production enhances its utility for complex workflows.

TensorFlow Extended (TFX): TFX offers a comprehensive pipeline for end-to-end machine learning, from data ingestion and validation to model training and deployment. Setting up TFX in Google Colab is relatively straightforward, and its integration with TensorFlow Data Validation (TFDV) provides scalable data analytics. TFDV's capabilities in detecting data anomalies and variations contribute to improved data quality for machine learning applications [3].

VII. Recommendation

Based on the evaluation, Apache Airflow is recommended as the preferred tool for implementing data engineering pipelines. Airflow's open-source nature and scalability make it suitable for long-term data management needs. Its customization capabilities, using Python code for defining tasks and workflows, allow for extensive flexibility. Airflow's support for integrating various tools—such as Apache Spark for data processing and TFX for machine learning orchestration—enhances its versatility. Despite the initial complexity of setting up Airflow, its rich ecosystem of plugins and operators facilitates effective workflow management and orchestration. Airflow's ability to run and test code in isolated environments further supports its use in both development and production settings.

VIII. Conclusion

Choosing the most suitable pipeline for data engineering depends on various factors, including the nature of the data, the tool's capabilities, the user's expertise, and the specific problem being addressed. Evaluating multiple pipelines during the exploratory phase can provide insights into their effectiveness and suitability. While Apache Airflow presents a steep learning curve initially, its comprehensive functionality, strong community support, and flexibility make it a preferred choice for managing complex workflows. Conversely, Apache Spark with PySpark in Jupyter Notebook offers a more straightforward setup for initial exploration and quick implementations. Ultimately, the choice of tool should align with the project's requirements and the team's familiarity with the technology.

IX. References

1. Dasu, T., & Johnson, T. (2003). *Exploratory data mining and data cleaning*. John Wiley & Sons.
2. Dong, X. L., & Rekatsinas, T. (2018). Data integration and machine learning: A natural synergy. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 1645–1650).
3. Caveness, E., et al. (2020). TensorFlow Data Validation: Data analysis and validation in continuous ML pipelines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (pp. 2793–2796).
4. Kweun, M., et al. (2020). Lineage checkpoint approach for long-lineage problem in Apache Spark. In *2020 IEEE International Conference on Big Data (Big Data)* (pp. 5733–5735). IEEE.
5. Nazabal, A., et al. (2020). Data engineering for data analytics: A classification of the issues, and case studies. *arXiv preprint arXiv:2004.12929*.
6. Ru-tao, Z., et al. (2020). A machine learning pipeline generation approach for data analysis. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)* (pp. 1488–1493). IEEE.
7. Van Dongen, G., & Van den Poel, D. (2020). Evaluation of stream processing frameworks. *IEEE Transactions on Parallel and Distributed Systems*, *31*(8), 1845–1858.
8. Widanage, C., et al. (2020). High performance data engineering everywhere. In *2020 IEEE International Conference on Smart Data Services (SMDS)* (pp. 122–132). IEEE.
9. Hlupić, T., & Puniš, J. (2021). An overview of current trends in data ingestion and integration. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)* (pp. 1265–1270). IEEE.
10. Pullinger, M., et al. (2021). The IDEAL household energy dataset, electricity, gas, contextual sensor data, and survey data for 255 UK homes. *Scientific Data*, *8*(1), 146.
11. Aseman-Manzar, M.-M., et al. (2022). Cost-aware resource recommendation for DAG-based big data workflows: An Apache Spark case study. *IEEE Transactions on Services Computing*.
12. De Bie, T., et al. (2022). Automating data science. *Communications of the ACM*, *65*(3), 76–87. https://doi.org/10.1145/3495256
13. Nadal, S., et al. (2022). Operationalizing and automating data governance. *Journal of Big Data*, *9*(1), 117.
14. Saini, Mukesh Kumar, and Yash Pal. "A STUDY OF ARTIFICIAL INTELLIGENCE, AND SOFTWARE ENGINEERING IN HEALTH CARE." (2017).
15. Vyas, S., et al. (2022). Performance evaluation of Apache Kafka—a modern platform for real-time data streaming. In *2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM)* (Vol. 2, pp. 465–470). IEEE.