Revolutionizing Lead Generation

In lead generation, speed and accuracy are the keys to success. If the data is messy, the company can miss out on revenue. If the system can't scale, the company will leave money on the table. This script is designed to ensure that TLA will never miss an opportunity. It cleans, processes, and organizes the data so the team gets the best leads in real-time. With Azure's unparalleled scalability, this solution grows TLA business, ensuring that no lead goes uncaptured, no matter how many are generated.

How it works:

This script is designed to clean and process vehicle-related data from a CSV file. It performs several tasks like correcting swapped fields, cleaning up date formats, removing duplicates, and optimizing memory usage. It logs its progress and monitors memory consumption during execution. It was created running in a “venv” virtual environment using a configuration file (config.json) to store the parameters like file paths, brand names, and more.

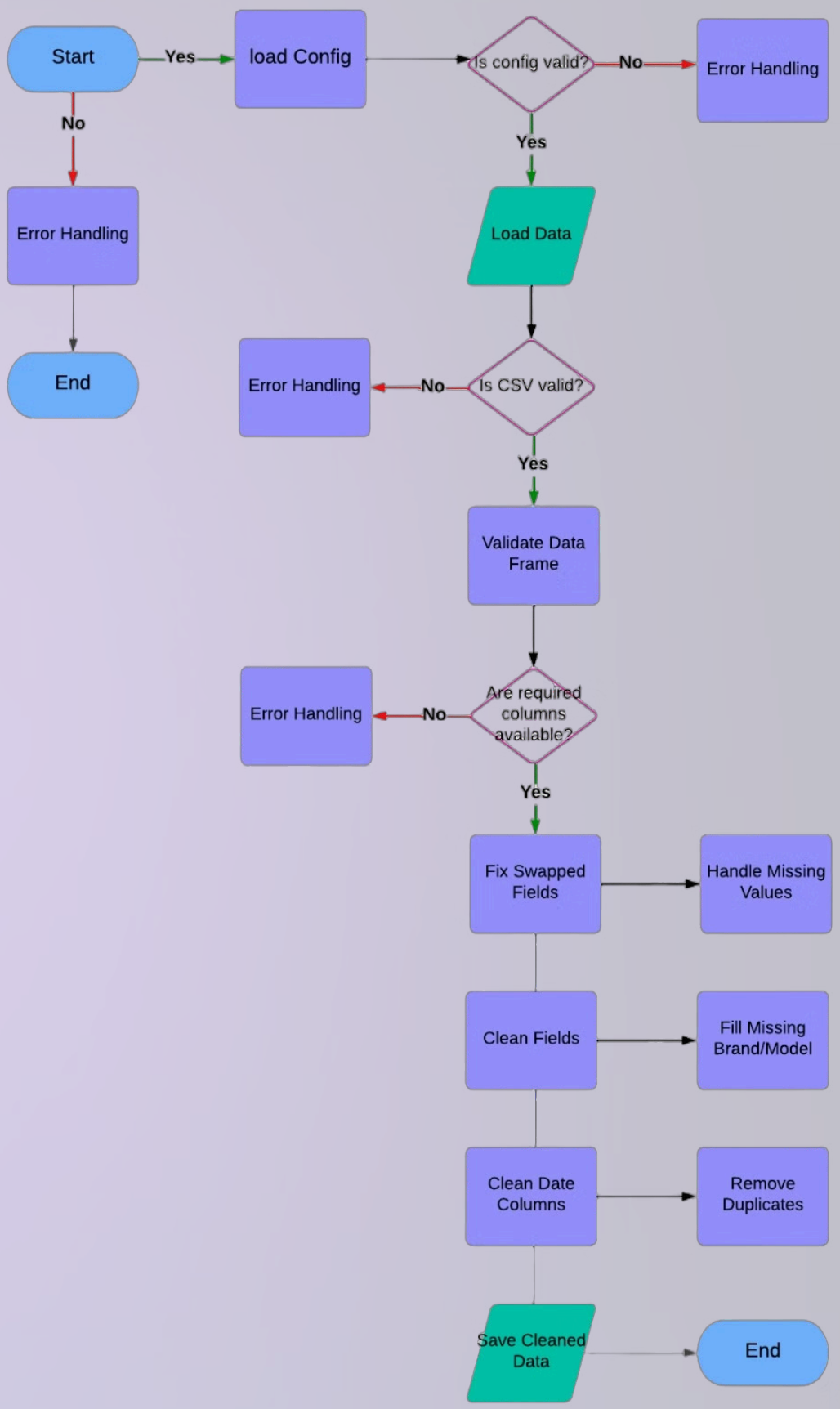
How It Works

The image provided in the input shows a flowchart of how the script works in conjunction with Azure services. It illustrates the data flow from ingestion through processing to final insights and actions.

At the end, the script generates a cleaned version of the CSV file, optimized for memory usage and free from duplicates, missing values, and incorrect data formats.

The script is set up with logging to track its progress. If an error occurs (such as a missing file or invalid data), the script logs the error and continues with its tasks or terminates gracefully.

Please download the script from the Github repository



Key Steps:

1. **Loading Configuration:**
   - The script reads a configuration file (`config.json`) to load settings like the file path for input and output data.
   - If the configuration file is not found or invalid, an error is logged.
2. **Logging Memory Usage:**
   - The script keeps track of the amount of memory used by logging memory consumption at various points, which helps identify potential performance bottlenecks.
3. **Loading Data:**
   - The script reads the data from a CSV file (path specified in the configuration) into a pandas DataFrame.
   - If the file doesn't exist or can't be parsed, the script logs an error.
4. **Fixing Swapped Fields:**
   - The script checks if "Brand", "Model", or "Derivative" fields have been swapped, based on a list of known car brands.
   - If they have been swapped, it corrects them by placing the values in their correct columns.
5. **Handling Missing Values:**
   - The script fills missing "Brand" values using forward and backward filling (using the closest available values).
6. **Cleaning Fields (Package and Derivative):**
   - The script cleans the "Derivative" field by removing unwanted brackets or text inside square brackets and extracting that information into a new "Package" column.
7. **Cleaning the Introduced Field (Date):**
   - The script ensures the "Introduced" field only contains valid dates by cleaning up invalid entries or unwanted prefixes (like "o", "to").
   - It strips away unwanted characters and extracts valid dates from the strings.
8. **Cleaning the Discontinued Field (Date):**
   - Similar to the **Introduced** field, the script cleans up and standardizes the **Discontinued** field by ensuring it contains valid dates.
9. **Removing Duplicates:**
   - It checks for and removes any duplicate rows from the DataFrame to ensure the data is unique.
10. **Memory Optimization:**
    - The script converts some of the columns (like "Brand", "Model", "Derivative") to a more memory-efficient type ("category"), which helps reduce memory consumption.
    - It also converts date columns ("Introduced" and "Discontinued") to the `datetime` format for better memory efficiency.
11. **Saving the Cleaned Data:**
    - Once the data is cleaned and processed, the script saves the updated DataFrame to a new CSV file (specified in the configuration).
12. **Timing the Script:**
    - The script tracks how long it takes to run the entire process and logs the total time once the process is completed.

Why It's Perfect for Lead Generation

1 Elegance and Efficiency

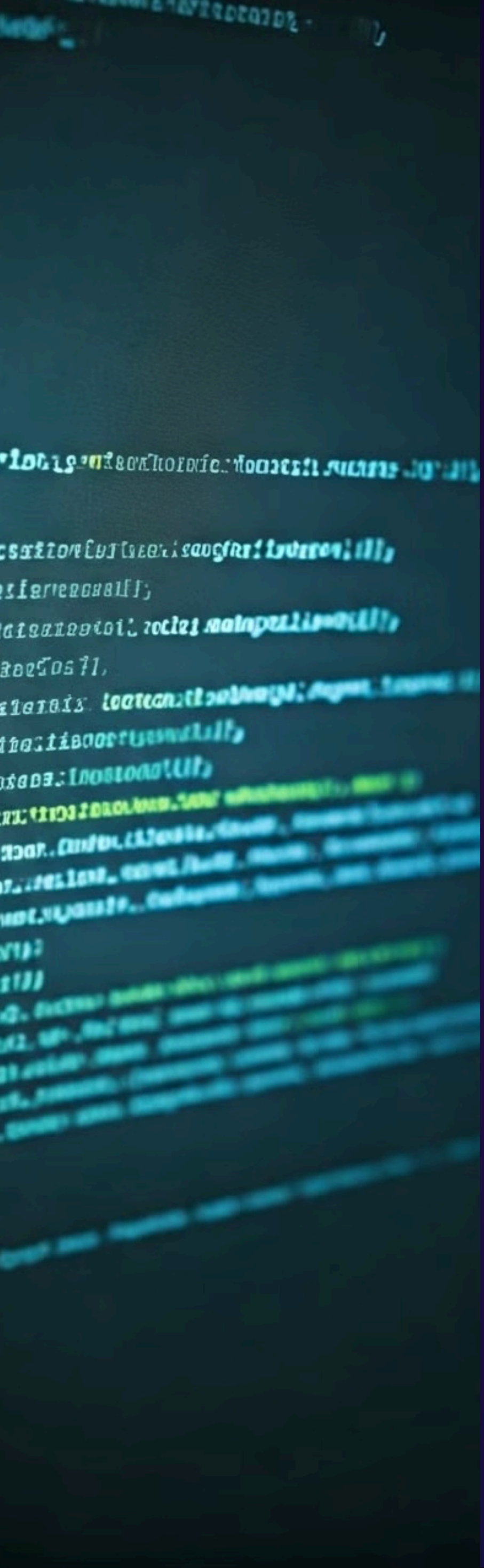
With elegance, efficiency, and the ability to scale effortlessly. This script is designed to give valuable insights from their data, while being completely optimized for Azure cloud scalability.

2 Azure-Ready

Right now, this script is ready to run on Azure, providing immediate benefits for lead generation processes.

3 Transformative Potential

This script with Azure has the potential to transform how businesses generate leads, making it more efficient, scalable, and automated than ever before. This is not a problem to solve, it's an opportunity to excel.



Key Features of the Script

1

1. Streamlined Logging and Error Handling

This script integrates robust logging and error handling. Every single step is monitored and captured, ensuring transparency and accountability. If something goes wrong, it can tell me where, why, and how to fix it.

2

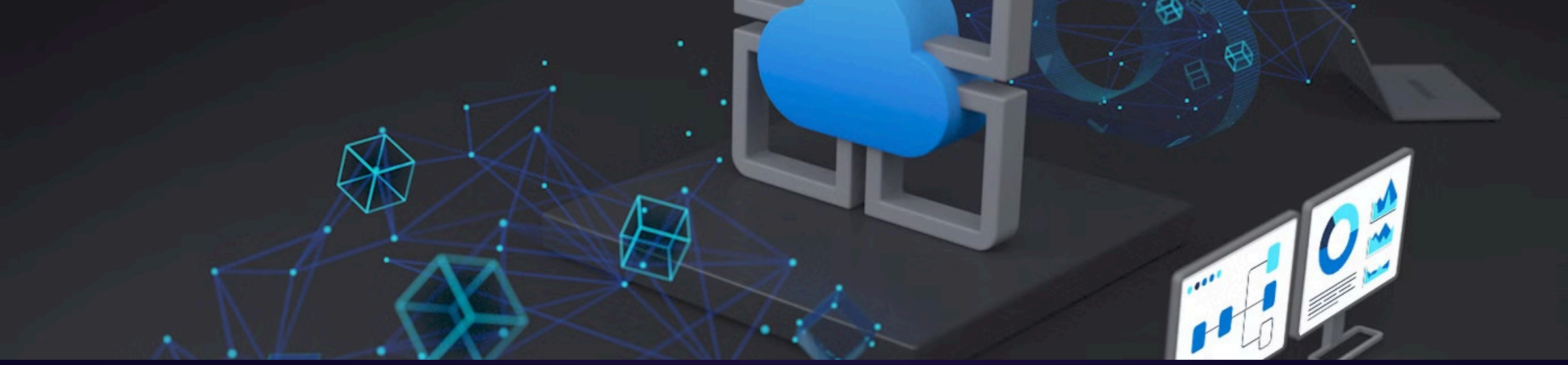
2. Efficiency in Data Cleaning

The script runs through 7,000 lines of data effortlessly in 0.38 seconds, optimizing memory usage by converting fields into the most efficient types. It's efficient. It's light. It can be automate the entire data pipeline by using **Azure Functions** or **Logic Apps** to trigger the data cleaning script whenever new data arrives in Blob Storage or Data Lake.

3

3. Parallel Processing

This script uses parallel processing with Swifter, giving it the ability to handle larger datasets with ease. It's the multitasking engine that makes this code more powerful and faster as the data grows, preparing it for Databricks by using PySpark it can distribute the computation across multiple nodes.



Azure Integration Benefits

Distributed Processing in Azure Databricks

By transitioning this code to Azure Databricks, it can leverage the full power of distributed computing. Every single node in Azure's cloud work together, handling enormous datasets effortlessly.

Automation with Azure Data Factory

With Azure Data Factory, this script can be scheduled and run automatically, triggered by real-time data changes. The moment new leads come in, they are cleaned, processed, and ready for action. It's automation at its best.

Autoscaling with Azure Functions

Think about this: the company grows — 10x, 20x, or 100x — the script doesn't even blink. It scales. Azure Functions can dynamically allocate resources to this code based on the incoming data.

i. Relational Database Schema

In this schema, it is proposed to split the vehicle data into multiple relational tables to normalize the structure, reduce redundancy, and establish clear relationships between entities like brands, models, and derivatives.

In the proposed relational database schema, normalization helps reduce data redundancy by storing unique entries, such as brands, models, and derivatives, only once. These entries are linked through foreign keys, ensuring that the data is well-organized and efficient. This structure is highly scalable, allowing new brands, models, or derivatives to be added without needing major changes, which supports future growth.

Data integrity is ensured by using foreign keys to link related tables, maintaining consistency across the dataset and reducing the likelihood of errors. Additionally, the relational setup provides flexibility, allowing new data to be added or modified without disrupting the existing structure, making updates straightforward and manageable.

Relationships:

1. Brands → Models:
   A one-to-many relationship between Brands and Models. Each brand can have multiple models.

2. Models → Derivatives:
   A one-to-many relationship between Models and Derivatives. Each model can have multiple derivatives.

3. Derivatives → Vehicle Info:
   A one-to-many relationship between Derivatives and Vehicle Info. Each derivative can have multiple vehicle entries with different introduction and discontinuation dates.

1. Brands Table

This table contains unique car brands and assigns each brand a unique BrandID to link it to other tables.

Column Name	Data Type	Description
BrandID	INT (Primary Key, Auto Increment)	Unique identifier for each brand.
Brand	VARCHAR	Name of the car brand.

2. Models Table

This table stores the car models, with each model linked to a BrandID as a foreign key to establish a relationship between the model and the brand.

Column Name	Data Type	Description
ModelID	INT (Primary Key, Auto Increment)	Unique identifier for each model.
Model	VARCHAR	Name of the car model.
BrandID	INT (Foreign Key)	Foreign key linking to BrandID in the Brands table.

3. Derivatives Table

Each derivative of a model is stored here. It has a foreign key ModelID that links it to the Models table.

Column Name	Data Type	Description
DerivativeID	INT (Primary Key, Auto Increment)	Unique identifier for each derivative.
Derivative	VARCHAR	Name of the derivative.
ModelID	INT (Foreign Key)	Foreign key linking to ModelID in the Models table.

4. Vehicle Info Table

This table stores the introduction and discontinuation dates of vehicles, with a foreign key DerivativeID linking it to the Derivatives table.

Column Name	Data Type	Description
VehicleID	INT (Primary Key, Auto Increment)	Unique identifier for each vehicle entry.
Introduced	DATE	The introduction date of the vehicle.
Discontinued	DATE	The discontinuation date of the vehicle, if applicable.
DerivativeID	INT (Foreign Key)	Foreign key linking to DerivativeID in the Derivatives table.

ii. An unstructured or semi-structured format

Instead of scattering the data across multiple tables, each with complex relationships, it’s possible to hold the entire brand’s story, its models, derivatives, and history, in one clean, flexible structure. Each brand is its own complete entity, neatly organized into models, and each model holds its own derivatives and vehicle history right where you need it. This structure is ideal for representing car hierarchies (brands → models → derivatives) in a flexible format.

Each vehicle's lifecycle (introduced and discontinued dates) is captured, which is useful for tracking availability over time

Please download the JSON script from the Github repository


```
els": [
{
    "Derivatives": [
        {
            "Derivative": "320i",
            "VehicleInfo": {
                "Introduced": "2019-05-01",
                "Discontinued": "2023-01-01"
            }
        },
        {
            "Derivative": "330i",
            "VehicleInfo": {
                "Introduced": "2020-04-01",
                "Discontinued": null
            }
        }
    ]
},
{
    "Model": "Series 5",
    "Derivatives": [
        {
            "Derivative": "520d",
            "VehicleInfo": {
                "Introduced": "2018-06-15",
                "Discontinued": "2022-12-31"
            }
        }
    ]
}


nd": "Mercedes-Benz",
els": [
{
    "Model": "C-Class",
    "Derivatives": [
        {
            "Derivative": "C200",
            "VehicleInfo": {
                "Introduced": "2017-03-10",
                "Discontinued": "2021-09-30"
            }
```

Azure Integration Benefits

The benefits of integrating with Azure go beyond just storing data. It unlocks a powerful suite of tools that significantly enhance the efficiency and scalability of this project.

iii. Technological Recommendations

1

Data Ingestion with Azure Event Hub

At the start of any data pipeline, it's important to have something that can handle a massive influx of real-time data. For this Azure Event Hub can ingest millions of events per second. Whether the data is coming from websites, IoT devices, API'S or forms, Event Hub is built to scale with the business needs. Event Hub passes the real-time data seamlessly to Azure Stream Analytics or Azure Data Lake for storage and immediate analysis. From 1,000 or 1 million events per second, Event Hub scales automatically to meet demand.

Cost Estimate: For a website with 45,000 views, assuming a small portion converts into leads, we can estimate a medium throughput in Event Hubs. The estimated cost for this service can range from $50 - $100 per month, depending on the number of events per lead form submission.

2

Data Storage with Azure Data Lake Gen 2

Once that data is ingested, it needs to be stored in a way that allows for easy access, analysis, and processing. By using Azure Data Lake Gen 2, which combines scalability with enterprise-level security. Data Lake Gen 2 integrates with Azure Databricks and Azure Synapse Analytics for efficient processing, and Azure Data Factory for orchestration. Store petabytes of structured and unstructured data, from transactional logs to multimedia with no limitations. With hot, cool, and archive tiers, it can reduce costs by storing less frequently accessed data at lower prices.

Storing up to 10 GB of leads data monthly could cost around $2 - $5 per month.

Data Processing and Orchestration

1

Data Processing

After the storage all that data needs to be processed, cleaned, analyzed, and turned into insights. For that: Azure Databricks and Azure Synapse Analytics. Whether for running a simple script or complex machine learning models, Databricks can be scalable, leveraging the power of Apache Spark for python scripts. Databricks spins up resources when it's needed and shuts them down when it's don't, saving costs on compute power. Synapse can handle petabyte-scale data analytics, combining structured and unstructured data in one place. It's possible to run SQL queries, Spark jobs, and machine learning models all from a single unified workspace. It can be integrated: Azure Data Factory orchestrates the movement between Databricks (for real-time data processing) and Synapse (for high-level analytics and reporting).

For a small-to-medium business database with monthly querying and around 4,000 leads, expect costs between $100 - $150 per month.

2

Data Orchestration

To tie all of this together, can be used Azure Data Factory. It's the component that automates and orchestrates the entire data pipeline. It can coordinates all the services: from data ingestion to storage, processing, and analytics, ensuring they work in harmony. Handle thousands of data pipelines simultaneously. With trigger-based workflows, it can be paid only for the data movement and necessary processing. Data Factory links Event Hub, Data Lake, Databricks, and Synapse Analytics together, ensuring a seamless data pipeline.

For scheduled data processing and orchestration tasks, it could cost around $100 - $150 per month. Plus Azure Logic Apps for Lead Distribution, that can be $20 - $50 per month depending on the number of actions triggered per lead distribution.

Cost Estimates and Sources

Total Estimated Monthly Cost: $300 - $500 per month for the full Azure infrastructure, handling 45,000 views, generating 4,000 leads, and converting 400 car sales. This estimate accounts for event ingestion, data processing, storage, and lead distribution.

Service	Estimated Monthly Cost
Event Hub	$50 - $100
Data Lake Storage	$2 - $5
Azure Databricks and Synapse Analytics	$100 - $150
Data Factory	$100 - $150
Azure Logic Apps	$20 - $50
Total Estimated Monthly Cost	$300 - $500

Cost Data:

- **Event Hub**: ~$0.028 per million events.
- **Data Lake Gen 2**: Hot tier costs ~$0.20 per GB, cool tier ~$0.01 per GB, archive tier ~$0.002 per GB.
- **Azure Databricks**: ~$0.07/hour per cluster.
- **Azure Synapse Analytics**: ~$5 per hour for reserved compute.
- **Data Factory**: ~$0.001 per pipeline run.

For a pipeline processing **10 million events daily**:

- **Event Hub**: ~$0.28/day
- **Data Lake Storage** (hot tier, 100 GB/day): ~$20/day
- **Databricks Clusters** (10 hours/day): ~$0.70/day
- **Synapse Analytics**: ~$50/day for heavy analytics.

Sources:

- https://techcommunity.microsoft.com/t5/finops-blog/optimize-your-azure-costs-with-our-expert-guidance-pricing-tools/ba-p/4244981
- Azure Pricing Calculator