# Executive Summary: Logical Database Implementation for ADR Logistics

## 1. Project overview

ADR Logistics manages a large fleet of vehicles operating across different routes and conditions. Previously, their data was scattered across multiple spreadsheets and manual logs, making it hard to track performance or spot issues quickly. The goal of our project was to build a centralized database that brings everything together—vehicle details, maintenance records, sensor data, and delivery performance—in one place. This involved cleaning and integrating data from a sample dataset and then modelling it into a well-structured relational database.

To build this, we used a simulated logistics dataset from Kaggle that closely reflects real-world fleet operations (Kaggle, n.d.). It included 27 fields per vehicle, covering everything from vehicle details (model, make, year, etc.) to maintenance records, sensor readings (like engine temperature and tire pressure), and delivery metrics. Although synthetic, the dataset mirrored real-world fleet data that was detailed enough to guide a practical design.

We started by cleaning the data using Python and pandas—standardizing formats, fixing missing values, and making sure everything was consistent. This was essential because raw fleet and IoT data is often messy and hard to work with (Rahm and Do, 2000). Once cleaned, we imported the data into our structured relational database. The result is a single, reliable system that reduces duplication, supports real-time queries, and lays the groundwork for analysis and automation. In short, we transformed disconnected data into an organized, easy-to-use tool that can help ADR Logistics reduce downtime, spot trends, and make smarter decisions around vehicle maintenance.

## 2. Evaluation of Database Modelling and Normalisation

### 2.1. Database Modelling

We chose relational modelling because the structure of ADR's data was clearly defined and well-suited to tabular representation. Each vehicle in the fleet has multiple related records—maintenance logs, trip summaries, and sensor outputs—that fit naturally into a **one-to-many** schema. Relational databases are ideal for this kind of setup because they offer strong consistency, enforce data integrity through primary and foreign keys, and prevent duplication (Connolly and Begg, 2015).

The relational model allowed us to create four normalised tables—vehicle, maintenance, trip_info, and sensor_data—linked by a common identifier (vehicle_id). This structure supports referential integrity and ensures that every maintenance entry, trip record, or sensor reading is correctly tied to a valid vehicle in the system (Elmasri and Navathe, 2016). The use of separate but connected tables also improves clarity and scalability, as new attributes or data types can be added without breaking the schema (Rob and Coronel, 2007).

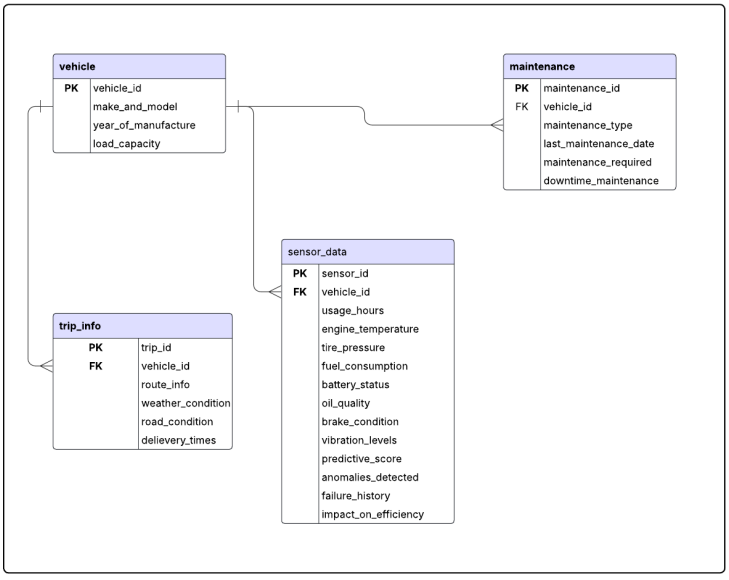



**Figure 1:** LucidChart Entity-Relationship Diagram

While other data models like NoSQL are useful for unstructured or high-velocity data, they often sacrifice schema rigidity and relational joins for flexibility (Atzeni and Torlone, 2010; Cattell, 2011). Since ADR's operational data is structured, predictable, and relational in nature, a traditional relational model provided the most reliable and maintainable foundation for the system (Hoffer, Ramesh and Topi, 2011).

### 2.2. Normalisation

We structured the database to reduce redundancy by aiming for a normalized design. Initially, we believed it followed Third Normal Form (3NF) due to the clear separation of entities. However, after closer analysis, we found that the schema fits better with **Second Normal Form (2NF)**.

According to Connolly and Begg (2015), Second Normal Form (2NF) requires that all non-key attributes rely on the whole primary key, not just part of it. Since each of our tables uses a single-column key—like vehicle_id or maintenance_id—and all other columns describe that key, the structure meets 2NF requirements. For instance, in the Maintenance table, maintenance_type, maintenance_cost, and last_maintenance_date all describe the specific record identified by maintenance_id.

I changed the normalization from 3NF because a few transitive dependencies remain. For example, maintenance_cost often relates to maintenance_type, and weather_conditions may correlate with route_info, meaning some non-key fields will depend on other non-key fields (Elmasri & Navathe, 2016).

Although further normalization (e.g., splitting out maintenance_type into a separate table) could resolve this, we decided it wasn't necessary at this scale. Maintaining 2NF keeps the schema simple, avoids unnecessary joins, and still ensures atomic fields with no partial dependencies—meeting both 1NF and 2NF (Rob & Coronel, 2007).

Minor repetition, like reoccurring maintenance_type values, is manageable and improves usability. This kind of pragmatic design is common in production systems, where full

normalization can sometimes complicate querying without offering real benefits (Hoffer, Ramesh & Topi, 2011).

For ADR Logistics, sticking with 2NF strikes a strong balance between clarity and efficiency. The database is clean, scalable, and easy to query—while still leaving room for further normalization if future needs demand it.

## 3. Evaluation of Database Management System (DBMS)

Choosing the right database was crucial. We compared SQL and NoSQL options, considering ADR's structured fleet data and the need for consistent, cross-table analysis. We ultimately chose **PostgreSQL**, a robust open-source SQL database that fit the current use case best.

Relational databases like PostgreSQL use structured tables and relationships, making them ideal for queries like "Which vehicles had the highest maintenance costs?" PostgreSQL enforces data consistency through ACID transactions and foreign keys (Connolly & Begg, 2015), which was essential given ADR's interconnected maintenance, trip, and sensor records.

Beyond structure, PostgreSQL offers strong performance and flexibility. It handles moderate workloads well, supports indexing and parallel queries, and is free to use—avoiding the licensing costs of commercial platforms (Cattell, 2011; Petrov, 2021). It also supports extensions like PostGIS, allowing for geospatial queries in future use cases such as GPS-based fleet tracking (Obe & Hsu, 2020).

PostgreSQL is widely adopted in logistics. Magaya migrated to PostgreSQL to improve speed and reduce costs (Magaya, 2021). Element Fleet Management uses it for real-time vehicle data replication (Treehouse Software, 2018), and MapTrack built their entire fleet tracking platform using PostgreSQL and PostGIS (ElifTech, 2023).

We also explored **NoSQL** options like MongoDB and Cassandra, which are great for scalability and schema flexibility—ideal for streaming large volumes of data such as real-time telemetry (Han et al., 2011; Pokorny, 2013). Built on the BASE model, NoSQL systems offer high availability

and horizontal scaling (Moniruzzaman & Hossain, 2013), but they don't enforce relationships or consistency as strictly as SQL. Denormalized data can lead to faster reads but risks inconsistencies.

In short, NoSQL could be useful for future high-volume tasks like real-time sensor ingestion. But for ADR's current structured needs—where data accuracy and relational integrity are vital—**PostgreSQL was the most reliable and scalable choice**.

## 4. Data Privacy and Legal Compliance (GDPR Considerations)

Although the dataset used didn't include personal identifiers, we designed the system with GDPR compliance in mind. In real-world use, driver names, route patterns, or even location history could be considered personal data under GDPR (Voigt & Von dem Bussche, 2017).

To stay ahead of that, we applied **privacy-by-design** from the beginning:

- **Lawfulness and consent**: ADR will need a clear legal reason to collect and process any personal data—usually either operational need or direct employee consent. Our schema allows for consent flags or anonymized identifiers if needed (Tankard, 2016).
- **Data minimization**: Only necessary data should be collected. For example, route IDs or region codes can be used instead of storing detailed per-second GPS logs—unless truly required.
- **User rights**: The relational structure makes it easy to retrieve, modify, or delete records tied to a particular driver. That supports GDPR rights like access, correction, or the right to be forgotten (Tikkinen-Piri, Rohunen and Markkula, 2018).
- **Security**: We recommended standard protections—like TLS encryption, hashed IDs, and role-based access—along with secure hosting. This aligns with ENISA's best practices (ENISA, 2021).
- **Retention policies**: The system can support data lifecycle rules, like deleting or anonymizing records after a set number of years. This is key for staying compliant with GDPR's storage limitation principle.

While our dataset didn't include personal identifiers, the design is built to handle them securely if needed. Features like **privacy mode**, already used in tools like Geotab, could be integrated to pause tracking during personal time (Geotab, 2025).

Ultimately, GDPR isn't just a legal requirement—it's about trust. With a clean, relational structure and data controls baked in, ADR will be able to scale safely and transparently.

## 5. Analysis and Key Findings

Alongside designing the database, I explored the dataset to show the kind of insights ADR Logistics could gain from a centralized system. Two key questions were focused on: **What types of maintenance are most common, and how much do they cost on average?** Also, **how many vehicles are currently flagged as needing maintenance**? These kinds of questions are important for planning both operational workload and budget.

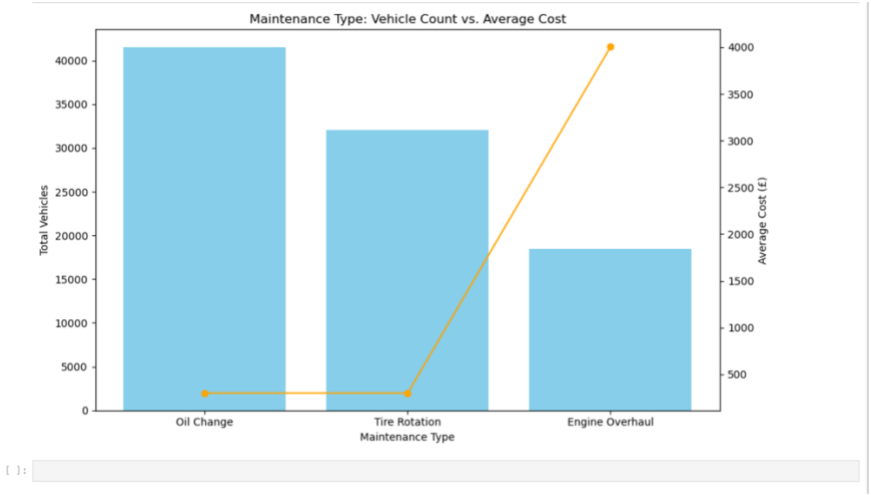



**Figure 2:** Frequency of maintenance types across the ADR fleet. *(Source: Author's own, 2025)*

Figure 2 shows that **oil changes and tire rotations dominate the fleet's service events**(refer to appendix A & B), while major repairs like engine overhauls are rare (approx.

20%). For ADR, this highlights a cost-saving opportunity: by streamlining routine maintenance (e.g., scheduling bulk oil changes), downtime can be reduced.

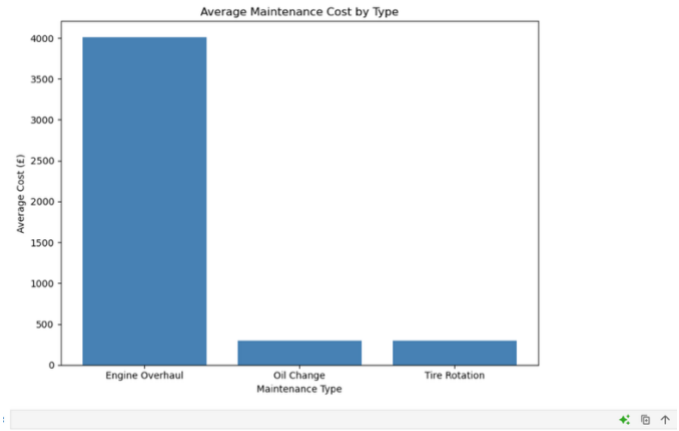



**Figure 3**: Average Maintenance Cost by Type. *(Source: Author's own, 2025)*

Figure 3 shows that oil changes and tire rotations both cost around £299 on average(refer to appendix C & D), while engine overhauls jump to £4,005(Author's own, 2025). Although overhauls are less frequent, their financial impact is significant. Proactively addressing sensor alerts could help ADR avoid these costly events.

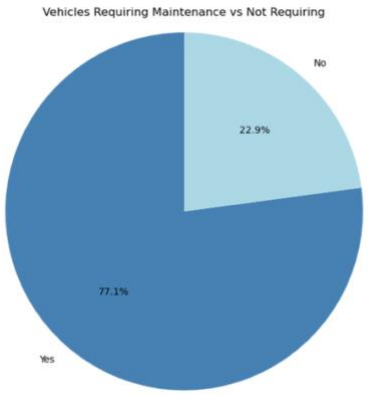



**Figure 4**: Distribution of vehicles flagged for maintenance. *(Source: Author's own, 2025)*

Figure 4 shows that roughly 77% of the vehicles in the dataset were flagged as needing maintenance(Author's own, 2025). None of the unflagged vehicles showed any sensor anomalies, whereas over half of the flagged ones did(refer to appendix E, F, G). This confirms the predictive model's logic and shows how sensor readings can help ADR move from reactive to preventive servicing.

For ADR, this means the system isn't just a log of what's already happened—it can also be used proactively. If a sensor reading spikes and the maintenance_required flag flips to true, the system can alert management early, potentially preventing costly repairs. With a few SQL queries, the team can list vehicles that need attention, join in sensor data, and act before downtime occurs.

Because the data is now normalized and centralized, ADR can perform more advanced analytics—like checking if certain vehicle models have higher maintenance costs or if specific routes lead to more breakdowns. The structure also allows easy export to Python for machine learning (McKinney, 2022). For instance, a model could be trained using sensor_data and maintenance_required as inputs, and its risk scores written back to the database in the predictive_score field.

Over time, this could support a dashboard showing which vehicles are likely to fail in the next month—enabling smarter resource allocation.

## 6. Conclusion and Recommendations

The ADR Logistics database project successfully delivered a well-structured, centralized system that replaces fragmented logs with a single source of truth. Our Second Normal Form (2NF) design balances simplicity and accuracy (Connolly & Begg, 2015), reducing redundancy while keeping the schema accessible. By using PostgreSQL—a robust, open-source SQL backend—we ensured support for complex queries and integration with future analytics, while remaining flexible for scaling or combining with NoSQL if needed (Obe & Hsu, 2020).

A key win is that ADR can now answer important operational questions quickly—whether that's identifying vehicles due for service, tracking costs by maintenance type, or monitoring sensor flags. The database is GDPR-conscious by design (Voigt & Von dem Bussche, 2017), and ready for predictive maintenance workflows using structured inputs like sensor data and service history.

To fully realize the system's value, we recommend the following next steps:

- **Integrate Real-Time Feeds** (High Priority)

Link the database with IoT devices and telematics tools to stream sensor data automatically. This enables timely alerts for overheating, brake failure, etc. (Zanella et al., 2014).

- **Build Dashboards** (High Priority)

Connect the system to tools like Power BI or Tableau for live KPI reporting. Include filters for fleet managers and alerts when thresholds are crossed (Chaudhuri et al., 2011).

- **Train Staff and assign Data Stewards** (medium Priority)

Ensure users understand how to use dashboards, interpret queries, and follow data protection best practices (Alashoor & Baskerville, 2015).

- **Plan for Growth** (Medium Priority)

Monitor query performance and consider moving to cloud-hosted PostgreSQL or splitting telemetry into a separate warehouse if volumes rise (Stonebraker & Çetintemel, 2005).

- **Pilot a Predictive Model** (Low to Medium Priority)

Use Python and scikit-learn to develop a prototype that predicts which vehicles are likely to fail. Feed outputs back into the main dashboard (Wang et al., 2020).

In short, this project provides more than just a database—it lays the groundwork for a smarter fleet. ADR now has a system that supports quick decision-making, compliance, and future innovation. By acting on these recommendations, the company can turn operational data into tangible savings and better fleet reliability.

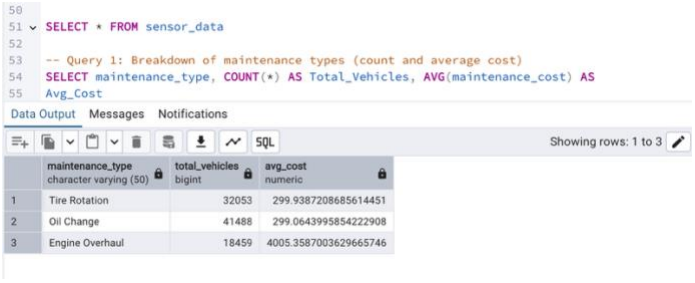
## References


1. Alashoor, T. and Baskerville, R. (2015) 'The privacy implications of digital identity: A research agenda', *AIS Transactions on Replication Research*, 1(1), pp. 1–12.
2. Atzeni, P. and Torlone, R. (2010) 'Data models for next-generation database systems', *ACM Computing Surveys*, 42(3), pp. 1–29.
3. Cattell, R. (2011) 'Scalable SQL and NoSQL data stores', *ACM SIGMOD Record*, 39(4), pp. 12–27.
4. Chaudhuri, S., Dayal, U. and Narasayya, V. (2011) 'An overview of business intelligence technology', *Communications of the ACM*, 54(8), pp. 88–98.
5. Connolly, T. and Begg, C. (2015) *Database Systems: A Practical Approach to Design, Implementation, and Management*. 6th edn. Harlow: Pearson.
6. ElifTech (2023) *MapTrack Fleet Management Solution Case Study*. Available at: https://www.eliftech.com/case-studies/fleet-management/ (Accessed: 14 July 2025).
7. Elmasri, R. and Navathe, S.B. (2016) *Fundamentals of Database Systems*. 7th edn. Boston: Pearson.
8. ENISA (2021) *Guidelines on pseudonymisation techniques and best practices*. European Union Agency for Cybersecurity.
9. Geotab (2025) 'GDPR compliance and privacy for fleets'. Available at: https://www.geotab.com/uk/fleet-management-solutions/privacy-gdpr/ (Accessed: 11 July 2025).

10. Han, J., Haihong, E., Le, G. and Du, J. (2011) 'Survey on NoSQL database systems and comparison with relational databases', *Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering*, 1, pp. 476–481.
11. Hoffer, J.A., Ramesh, V. and Topi, H. (2011) *Modern Database Management*. 10th edn. Boston: Pearson.
12. Kaggle (n.d.) *Logistics Vehicle Maintenance History Dataset*. Available at: https://www.kaggle.com/datasets/datasetengineer/logistics-vehicle-maintenance-history-dataset?resource=download/ (Accessed: 27 May 2025).
13. Magaya (2021) *Magaya Corporation re-architects platform using PostgreSQL*. Available at: https://www.magaya.com/news/postgresql-upgrade (Accessed: 14 July 2025).
14. McKinney, W. (2022) *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 3rd edn. Sebastopol: O'Reilly Media.
15. Moniruzzaman, A.B.M. and Hossain, S.A. (2013) 'NoSQL database: New era of databases for big data analytics—classification, characteristics and comparison', *International Journal of Database Theory and Application*, 6(4), pp. 1–14.
16. Obe, R.O. and Hsu, L.S. (2020) *PostGIS in Action*. 3rd edn. Shelter Island: Manning Publications.
17. Petrov, S. (2021) 'Scaling PostgreSQL: Indexing, Partitioning and Performance Optimization', *International Journal of Database Management Systems*, 13(3), pp. 31–43.
18. Pokorny, J. (2013) 'NoSQL databases: a step to database scalability in web environment', *International Journal of Web Information Systems*, 9(1), pp. 69–82.
19. Rahm, E. and Do, H.H. (2000) 'Data cleaning: Problems and current approaches', *IEEE Data Engineering Bulletin*, 23(4), pp. 3–13.
20. Rob, P. and Coronel, C. (2007) *Database Systems: Design, Implementation, and Management*. 8th edn. Boston: Cengage Learning.

21. Stonebraker, M. and Çetintemel, U. (2005) '"One Size Fits All": An Idea Whose Time Has Come and Gone', *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pp. 2–11.
22. Tankard, C. (2016) 'What the GDPR means for businesses', *Network Security*, 2016(6), pp. 5–8.
23. Tikkinen-Piri, C., Rohunen, A. and Markkula, J. (2018) 'EU General Data Protection Regulation: Changes and implications for personal data collecting companies', *Computer Law & Security Review*, 34(1), pp. 134–153.
24. Treehouse Software (2018) *Element Fleet Management Corporation - PostgreSQL Integration Case Study*. Available at: https://www.treehouse.com/case-studies/element-fleet (Accessed: 14 July 2025).
25. Voigt, P. and Von dem Bussche, A. (2017) *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Cham: Springer.
26. Wang, W., Ma, H., Ma, Y. and Li, J. (2020) 'A review on prognostics and health management of commercial vehicle systems', *Mechanical Systems and Signal Processing*, 142, p. 106760.
27. Zanella, A. et al. (2014) 'Internet of Things for Smart Cities', *IEEE Internet of Things Journal*, 1(1), pp. 22–32.

## Appendix



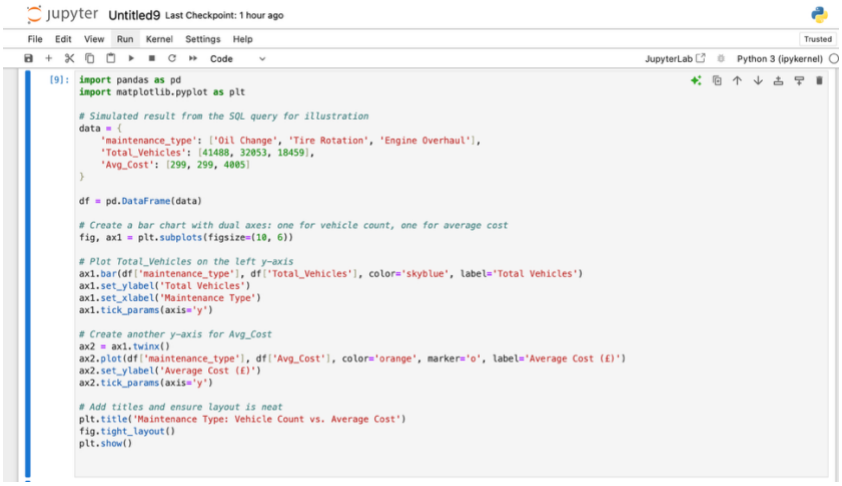
```
50
51  SELECT * FROM sensor_data
52
53  -- Query 1: Breakdown of maintenance types (count and average cost)
54  SELECT maintenance_type, COUNT(*) AS Total_Vehicles, AVG(maintenance_cost) AS
55  Avg_Cost
```

Data Output | Messages | Notifications

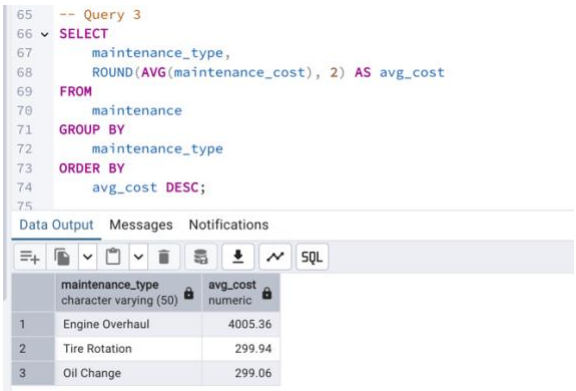
Showing rows: 1 to 3

| | maintenance_type character varying (50) | total_vehicles bigint | avg_cost numeric |
|---|---|---|---|
| 1 | Tire Rotation | 32053 | 299.9387208685614451 |
| 2 | Oil Change | 41488 | 299.0643995854222908 |
| 3 | Engine Overhaul | 18459 | 4005.3587003629665746 |

Appendix A: SQL Query Output – maintenance types (avg cost and count)

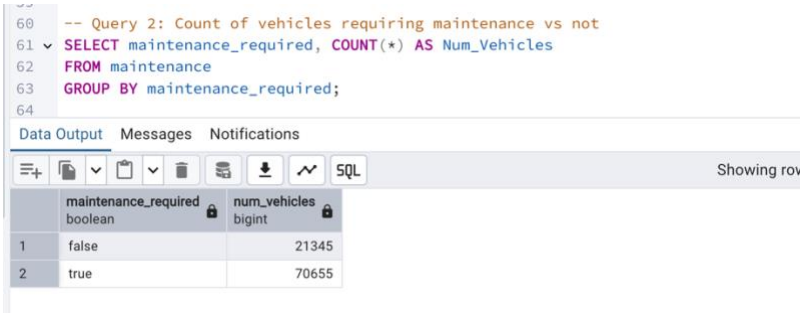



Appendix B: Bar Chart of maintenance types in Python




Appendix C: SQL Query Output – Average Maintenance Cost

Appendix D: Bar Chart Output of Maintenance Cost by Type in Python



```
60  -- Query 2: Count of vehicles requiring maintenance vs not
61  SELECT maintenance_required, COUNT(*) AS Num_Vehicles
62  FROM maintenance
63  GROUP BY maintenance_required;
64
```

Data Output | Messages | Notifications

Showing rov

| | maintenance_required boolean | num_vehicles bigint |
|---|---|---|
| 1 | false | 21345 |
| 2 | true | 70655 |

Appendix E: SQL Query Output for vehicles requiring maintenance or not

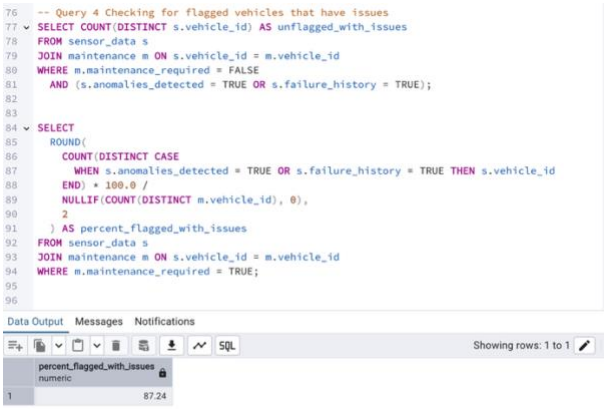
```
[19]: import pandas as pd
      import matplotlib.pyplot as plt

      # Simulated result from the SQL query for illustration
      data = {
          'maintenance_required': ['Yes', 'No'],
          'Num_Vehicles': [70751, 20994]
      }

      df = pd.DataFrame(data)

      # Create a pie chart for vehicles flagged vs not flagged
      plt.figure(figsize=(6, 6))
      plt.pie(
          df['Num_Vehicles'],
          labels=df['maintenance_required'],
          autopct='%1.1f%%',
          startangle=90,
          colors=['steelblue', 'lightblue']
      )
      plt.title('Vehicles Requiring Maintenance vs Not Requiring')
      plt.axis('equal')  # Ensures the pie chart is a circle
      plt.tight_layout()
      plt.show()
```

Appendix F: Python pie chart output for vehicles requiring maintenance or not



```
76  -- Query 4 Checking for flagged vehicles that have issues
77  SELECT COUNT(DISTINCT s.vehicle_id) AS unflagged_with_issues
78  FROM sensor_data s
79  JOIN maintenance m ON s.vehicle_id = m.vehicle_id
80  WHERE m.maintenance_required = FALSE
81    AND (s.anomalies_detected = TRUE OR s.failure_history = TRUE);
82
83
84  SELECT
85    ROUND(
86      COUNT(DISTINCT CASE
87        WHEN s.anomalies_detected = TRUE OR s.failure_history = TRUE THEN s.vehicle_id
88      END) * 100.0 /
89      NULLIF(COUNT(DISTINCT m.vehicle_id), 0),
90      2
91    ) AS percent_flagged_with_issues
92  FROM sensor_data s
93  JOIN maintenance m ON s.vehicle_id = m.vehicle_id
94  WHERE m.maintenance_required = TRUE;
95
96
```

Data Output | Messages | Notifications

Showing rows: 1 to 1

| | percent_flagged_with_issues numeric |
|---|---|
| 1 | 87.24 |

Appendix G: SQL Query for flagged vehicles