Optimising Cloud-Based Data Processing: A Case Study on Azure and Near-Earth Object Analysis.

Project Name: Big Data and Cloud Computing.

List of Figures

1.0 Types and Sources of Data Handled by the Organisation

In the context of this LDS7005M cloud-based data assessment, the organisation is a multinational corporation specialising in space observation analytics, with a focus on near-Earth object (NEO) monitoring, categorisation, and danger prediction. As a data architect, you must work with a wide range of data types from numerous sources to support vital planetary defense and research operations. This project's dataset, dubbed "Nearest Earth Objects (1910-2024)," was compiled from NASA's Open API data and made available for public research and development activities (Kaggle, 2023).

Dataset link:

https://www.kaggle.com/datasets/ivansher/nasa-nearest-earth-objects-1910-2024?resource=download

1.1 Types of Data

The company deals with a wide range of **big data**, broadly categorised into:

- **Structured data:** The core NEO dataset is structured in tabular form (CSV format), containing clear field delimiters and headings suitable for relational processing. Fields include:
    - id, name, absolute_magnitude_h
    - estimated_diameter_min/max
    - relative_velocity, miss_distance, orbiting_body
    - close_approach_date, and is_potentially_hazardous_asteroid
- **Semi-structured data:** When processed within a real-world architecture, similar enterprises would also utilise JSON-based feeds from live APIs (e.g., NASA's NeoWs), which encapsulate data in key-value formats without strict schemas.
- **Unstructured data (potentially)**: While not included in this specific CSV, organisations of this nature typically manage astronomical image feeds, scientific publications, sensor logs, or unstructured observational logs in formats like FITS, text, or images (Banerjee et al., 2021).

The combination of these kinds allows for a wide range of analytics, including collision risk prediction, temporal trend visualisation, and spatial orbit prediction.

1.2 Sources of Data

The organisation's data ecosystem is fed by both internal and external sources, in accordance with common global data policies (Hashem et al., 2015). These sources include:

- **External scientific sources:**
    - **NASA's NeoWs (Near Earth Object Web Service):** This API provides structured data on NEO trajectories, sizes, and probabilities of Earth encounters.
    - **Jet Propulsion Laboratory (JPL) Small-Body Database:** Offers detailed orbital parameters, observation history, and physical characteristics of celestial bodies (NASA/JPL, 2024).
    - **Minor Planet Center (MPC):** Responsible for centralised collection of asteroid and comet observations, crucial for long-term NEO risk analysis.

- **Internal sources (for applied enterprise use):**
    - Simulated sensor data from telescopes for observatories.
    - Monitoring logs from ground control or satellite telemetry systems.
    - Real-time ingestion layers from research partners or citizen science platforms (e.g., Zooniverse).

All data was ingested and securely stored on Azure Blob Storage, an enterprise-grade object store designed specifically for big data access. The .csv dataset was uploaded to a specialised container named 'datasets' in a cloud-native storage account called neoprojectdata, which was then accessible via Azure Databricks for distributed processing.

1.3 Implications of Big Data Handling in a Multinational Context

As a worldwide corporation functioning in scientific, educational, and security areas, the organisation must process high-volume, high-velocity, and high-variety datasets, in accordance with the three Vs of Big Data (Laney 2001). From a data architecture standpoint, this necessitates not only scalable infrastructure but also strong governance, interoperability, and integration protocols.

The range of data handled from historical asteroid observations to real-time orbital paths requires a cloud-native approach for accessibility, redundancy, and compute-intensive operations such as machine learning, anomaly detection, and trajectory forecasting (Gandomi and Haider, 2015). This validates the usage of Azure Databricks in this project because it offers distributed Spark-based processing for both ETL (Extract, Transform, and Load) procedures and in-depth analytics workflows.

1.4 Proposed Cloud-Based Storage Solution

A dependable, dynamic, and cost-effective data storage system is essential for storing large-scale astronomical information like near-Earth object (NEO) trajectories. This project used Microsoft Azure Blob Storage as its major cloud-based repository to handle the volume, velocity, and variety of space observation data sourced from NASA's NeoWs API. Azure Blob Storage is a cloud-native, object-based storage system designed to handle both structured files (e.g.,.csv) and unstructured formats (e.g., logs, pictures, JSON), which are common in planetary science and aerospace analytics (Microsoft, 2024a). The decision to utilise this platform was based on its high scalability, geo-redundant architecture, cost-cutting capabilities, and native connection with analytics providers such as Azure Databricks.

One of Azure Blob's most notable architectural characteristics is its scalable nature. In this project, the historical dataset containing asteroid observation records from 1910 to 2024 was uploaded to a container named 'datasets' within a Blob Storage account called 'neoprojectdata. Azure's horizontal scalability strategy enables storage to grow automatically as new data is added, without requiring service disruptions or hardware upgrades. This idea of elasticity is consistent with the NIST definition of cloud computing,

which emphasises quick supply of storage resources with minimal human contact (Mell and Grance, 2011). Such dynamic storage structures are crucial for scientific areas, particularly during times of high data acquisition, such as meteor showers or unexpected asteroid flybys. For example, the European Southern Observatory (ESO) uses scalable object storage to accommodate data from telescopes that generate terabytes of data nightly (Moreno-Ibarra et al., 2022).

Azure's redundancy solutions address reliability and fault tolerance by ensuring data resilience in the event of hardware or geographical breakdowns. In this project, Locally Redundant Storage (LRS) was enabled by default, duplicating uploaded data three times within a single Azure data centre. Users can choose Geo-Redundant Storage (GRS) for mission-critical applications, which replicates data asynchronously to a backup location hundreds of miles away. This technique follows the Shared Responsibility Model in cloud computing: while Microsoft ensures infrastructure stability, users must manage logical data integrity and access policies (Hashem et al., 2015). Global media services such as Amazon Prime Video use identical cross-region storage methods to enable continuous access to material even during a regional service outage (AWS, 2023).

Azure Blob Storage offers a tiered pricing strategy that allows customers to optimise costs based on data access frequency. This project made use of the Hot Tier, which is appropriate for datasets that are regularly accessed during the initial stages of analysis, cleaning, and modelling. Long-term archiving could later be moved to the Cool or Archive Tier, considerably cutting expenses for infrequently accessed material. This pricing flexibility contrasts with typical on-premise systems, which frequently result in over-provisioning and under-utilisation of storage resources. Netflix and other global organisations use tiered models to efficiently manage their content libraries by storing infrequently streamed materials in low-cost archive systems (Grolinger et al., 2014).

Another key benefit is Azure's seamless connection with analytical environments. During this project, access keys were created using the Azure portal and used in Azure Databricks to mount the blob storage container directly to a Spark session. This enabled the dataset to be processed and distributed in-memory using Apache Spark. Complex processes like EDA, feature selection, and model training were carried out within the Databricks workspace without the need for data duplication or external ETL pipelines. Similar workflows are used in commercial smart manufacturing setups, where machine sensor logs are transmitted to Azure Blob, analysed in Databricks, and shown in real time using Power BI dashboards (Zhou et al., 2020).

Security is an essential component of any cloud solution, especially when the datasets are sensitive or proprietary. Azure Blob Storage employs multiple layers of security, including role-based access control (RBAC), Shared Access Signatures (SAS) for temporary access, and AES-256 encryption at rest. This prevents unauthorised access, inadvertent deletions, and data breaches. In sensitive sectors like aerospace or climate science, such characteristics are critical to ensuring compliance with international data governance norms. For example, the European Space Agency (ESA) uses comparable policies to manage access to Earth observation and satellite datasets shared with international research partners (Banerjee et al., 2021).

To summarise, Microsoft Azure Blob Storage offers a highly customisable, secure, and cost-effective solution for managing the massive and complicated data requirements connected with near-Earth object

surveillance. Its scalable design supports big data's three Vs: volume, velocity, and variety, and its interface with Azure Databricks allows the project to smoothly shift from data storage to analytics. Azure Blob, whether utilised for academic research, aerospace surveillance, or commercial manufacturing, exhibits modern cloud storage architecture principles.

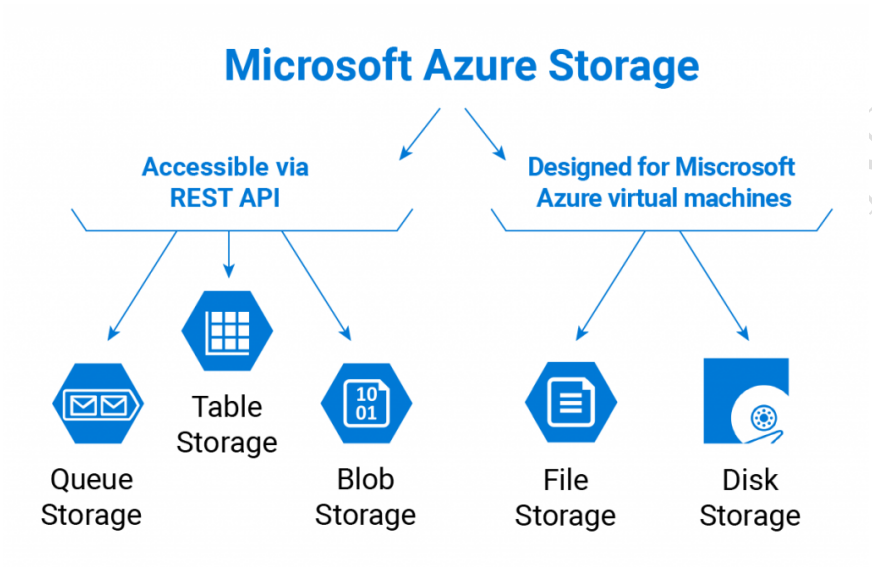


Figure 1: Overview of Azure storage types, with Blob Storage designed for unstructured, scalable object data and integrated tiering options (Hot, Cool, Archive). *Source: MSP360, 2025* .

In today's data-driven world, companies across industries are producing and consuming data at unprecedented rates and volumes. Traditional data processing systems frequently fail to meet the volume, variety, and real-time demands of such large data sets. As a result, cloud computing has emerged as a key facilitator of scalable, adaptable, and cost-effective big data solutions.

1.5 Introduction: Designing a Scalable Big Data Processing Architecture Using Cloud Computing Resources

Cloud platforms such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) provide elastic infrastructure and services like distributed storage (e.g. Amazon S3, Azure Blob Storage) and parallel processing engines (e.g. Apache Spark, Databricks) that can automatically scale to meet growing workloads (Hashem et al., 2015). For example, Netflix processes over a petabyte of log data daily using a combination of AWS services and Apache Spark to monitor, optimise, and personalise user experiences in real time (Netflix Tech Blog, 2020).

Scalable big data architectures have made scientific computing and disaster risk management more efficient and precise. NASA (National Aeronautics and Space Administration), for example, uses cloud-based pipelines to collect and analyse real-time satellite telemetry and Near-Earth Object (NEO) tracking data from a variety of sources (NASA, 2023). Without such infrastructure, managing the scale and reactivity needed for planetary defense would be impossible.

This project uses Apache Spark on Azure Databricks, along with cloud storage and distributed compute resources, to provide a scalable architecture for processing and analysing astronomical datasets about NEOs. The goal is to show how cloud-native tools may be combined to create a strong and scalable analytics pipeline capable of enabling real-time machine learning and decision support on a global scale.

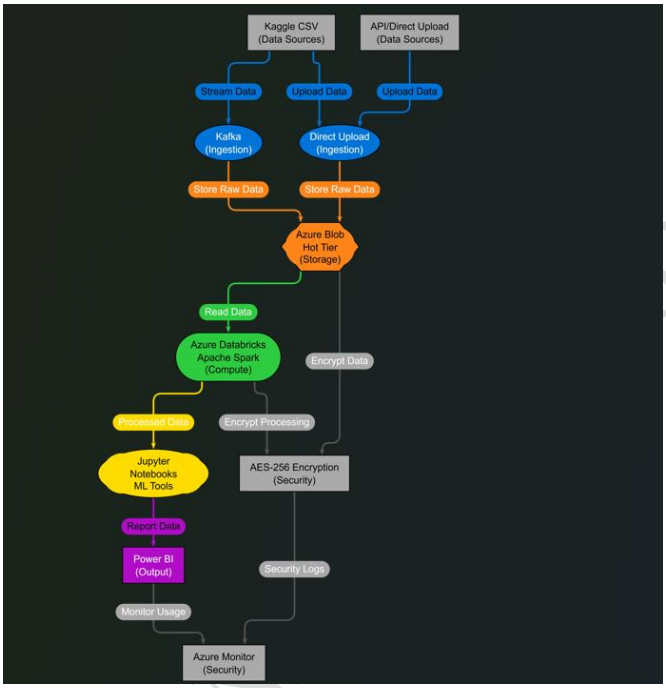


Figure 2: Data Pipeline Architecture for processing and analysing data using Azure services(Yed Live,2025).

1.5.1 Data Ingestion Process into the Cloud

The data ingestion process is the methodical movement of data from its source location to a central storage or analytical environment, usually within a cloud infrastructure. This project entailed importing a large.csv file containing near-Earth object (NEO) data into Azure Blob Storage, then connecting the storage to Azure Databricks for distributed processing. Efficient data ingestion is essential for any big data pipeline because it guarantees that datasets are transmitted in a fast, accurate, and safe way to support subsequent transformations and analytics (Gandomi and Haider 2015).

The first stage in this pipeline was to identify the dataset source, which was a structured CSV file retrieved from Kaggle with over 90,000 records covering asteroid size, velocity, orbiting body, and potential hazard status. Understanding that the file was in tabular format (structured data) influenced the selection of cloud solutions appropriate for batch transmission rather than streaming.

Once the dataset was secured locally, Azure Portal was used to create a new Blob Storage account called neoprojectdata, designed to house project-specific data containers. A dedicated container named 'datasets'

was created within the storage account to logically isolate the NEO dataset (see Figure 3: Creating Azure Blob Storage Container). This step mirrors industry practices, such as those used in the aviation sector, where aircraft telemetry logs are continuously organised into containerised storage for scalable ingestion.

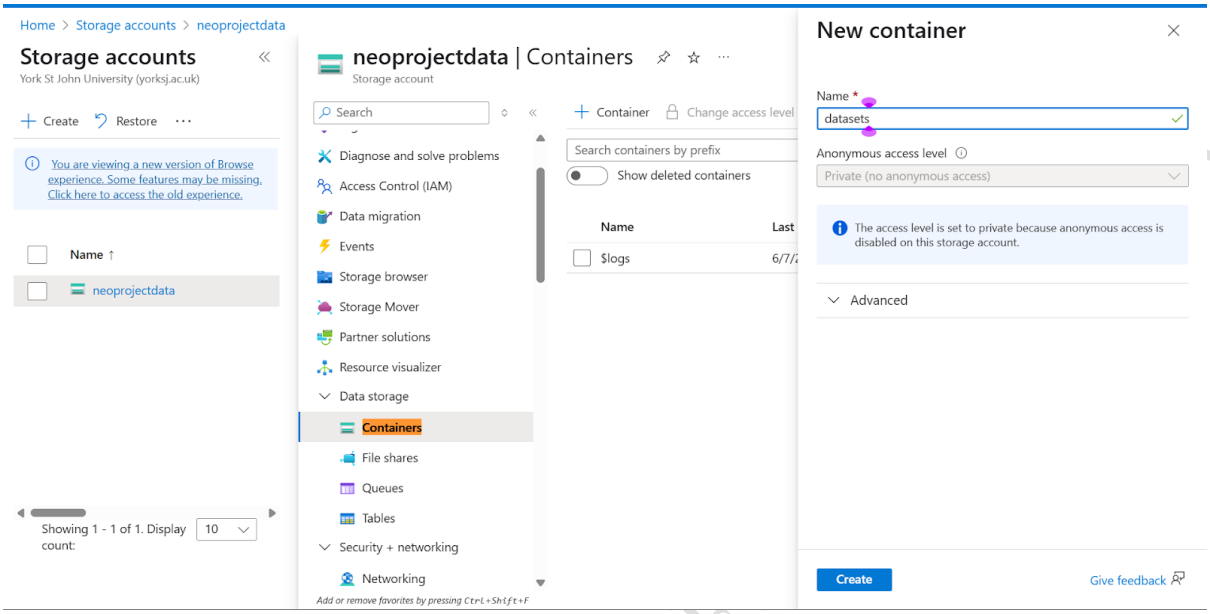


Figure 3: Verifying Container Creation(Microsoft, 2024).

The 'datasets' container now appears within the list of available containers in Azure Storage Explorer, confirming successful creation.

Following container creation, the .csv file was uploaded directly via the Azure Portal's Blob interface, using drag-and-drop or manual file selection. This is visually confirmed in Figure 1.2: Verifying Container Creation, and Figure 4: Uploading CSV Dataset to Blob. Azure automates this ingestion using REST APIs and encryption at rest, ensuring secure transmission even without the use of an external ingestion engine like Apache Kafka or NiFi.

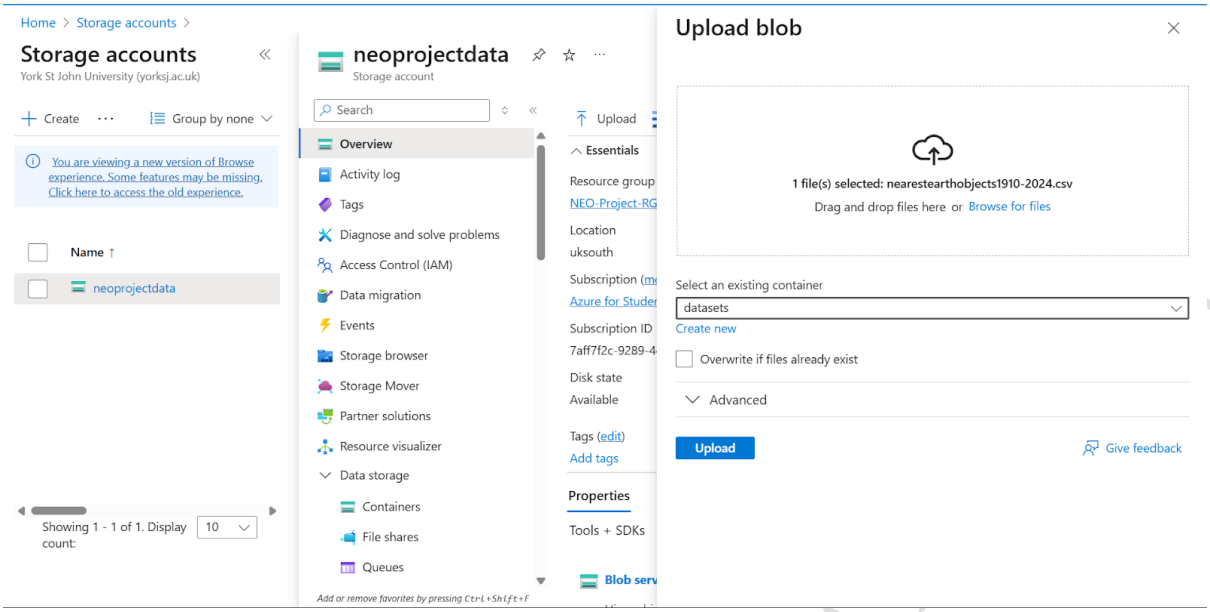


Figure 4: Blob Upload Confirmation(Microsoft Azure,2024).

The system displays a success message confirming that the CSV file has been uploaded successfully to the Azure Blob container (Microsoft, 2024).

Once uploaded, Azure provides a real-time verification dashboard, confirming both storage tier and blob location metadata. This verification, shown in Figure 5: Blob Upload Confirmation, confirms the successful completion of ingestion and readiness of the dataset for downstream access.

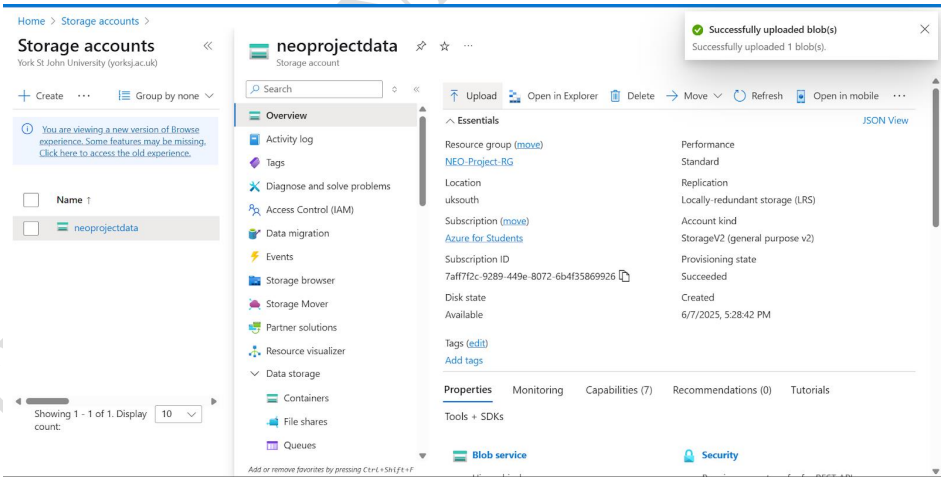


Figure 5: Blob Upload Confirmation (Microsoft, 2024).

The system displays a success message confirming that the CSV file has been uploaded successfully to the Azure Blob container (Microsoft, 2024).

To enable seamless analytical workflows, the data was made accessible to Azure Databricks, a Spark-based analytics engine integrated within the Azure ecosystem. By generating a storage access key via the Azure Portal and inserting it into the Databricks notebook, the project team mounted the 'datasets' container into

the Databricks file system (DBFS), as shown in Figure 6: Accessing Azure Databricks from Storage. This direct cloud-to-cloud linkage eliminated the need for additional extract-transform-load (ETL) operations and allowed real-time querying via PySpark.

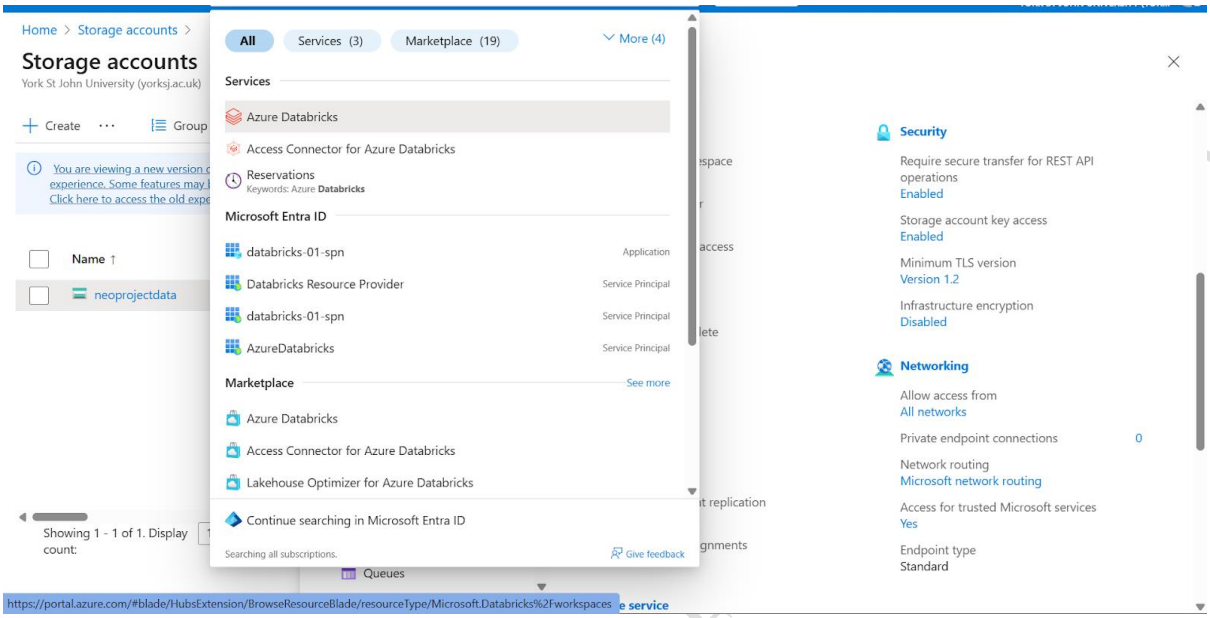


Figure 6: Accessing Azure Databricks from Storage.

This screenshot shows the process of navigating to Azure Databricks services from the Storage Account interface to begin integration. (Microsoft, 2024).

Though streaming ingestion was not necessary for this static dataset, industry-standard solutions like Apache NiFi, Apache Kafka, and Azure Data Factory are frequently used in business structures to manage continuous ingestion from API endpoints, IoT devices, or third-party systems. These solutions support batch and real-time ingestion, providing a wide range of organisational requirements. Similar space scientific programs, for example, have employed streaming systems to ingest telescope feeds from observatories in real time into Azure pipelines for instant analysis (Moreno-Ibarra et al., 2022).

The ingestion procedure also allows for metadata tagging, compression, and tier-based storage selection. Azure Blob Storage allows you to designate ingested blobs with descriptors like "NEO_observations," making it easier to handle datasets at scale. When combined with tiering (Hot, Cool, Archive), these controls enable ingestion operations to correspond with storage lifetime and cost management objectives.

Finally, the data ingestion technique used in this project follows best practices in cloud-native data architecture. It ensures that structured observational data is safely uploaded, stored in a logically organised container, and instantly available for Spark-based analysis. The seamless transition between the input and analysis phases emphasises the importance of cloud integration in scientific data operations.

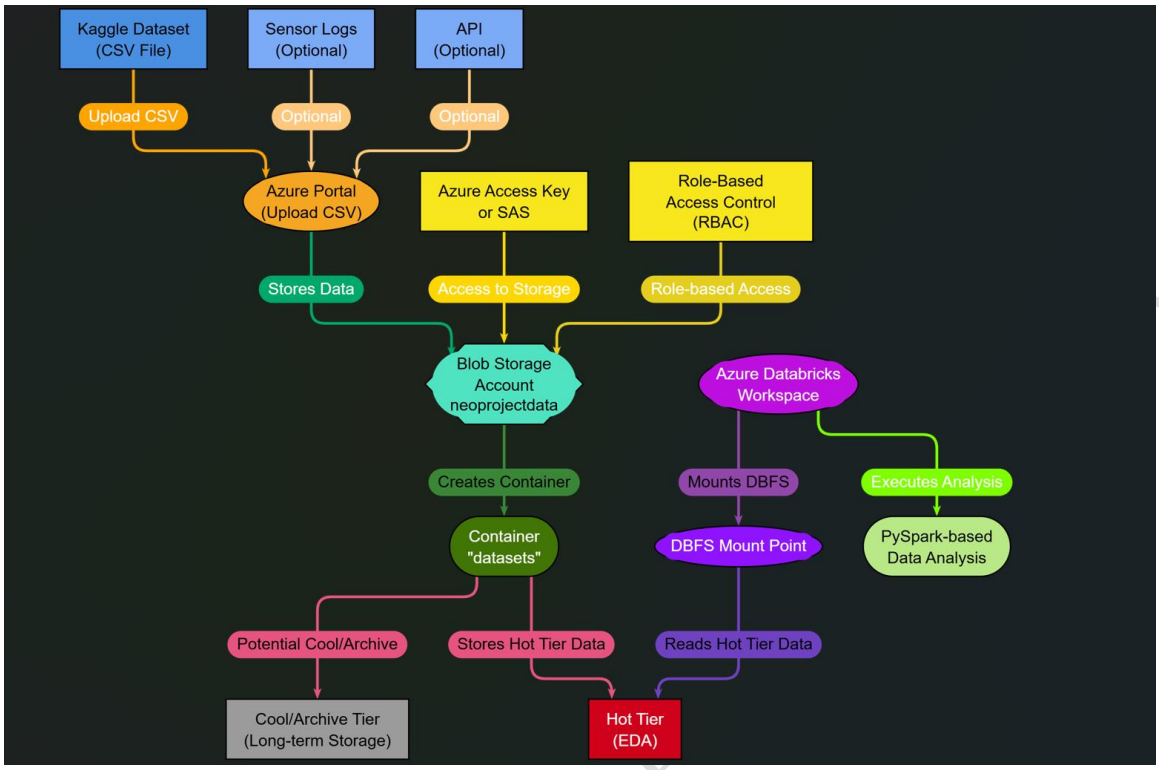


Figure 7: Azure Data Ingestion Pipeline Using Blob Storage and Databricks (Adapted from Microsoft, 2024; yEd Live, 2025).

1.5.2 Scalable Distributed Processing via Azure Databricks

To satisfy the computational demands of processing and analysing large-scale space object data, this architecture used Azure Databricks, a unified cloud platform for big data analytics and machine learning. At its core, Databricks is driven by Apache Spark, an open-source, distributed data processing engine known for its speed and capacity to manage enormous datasets over multiple servers. Spark, unlike typical batch processing systems like Hadoop MapReduce, supports in-memory computation, which dramatically accelerates processes such as iterative machine learning, SQL queries, and real-time analytics (Zaharia et al., 2016).

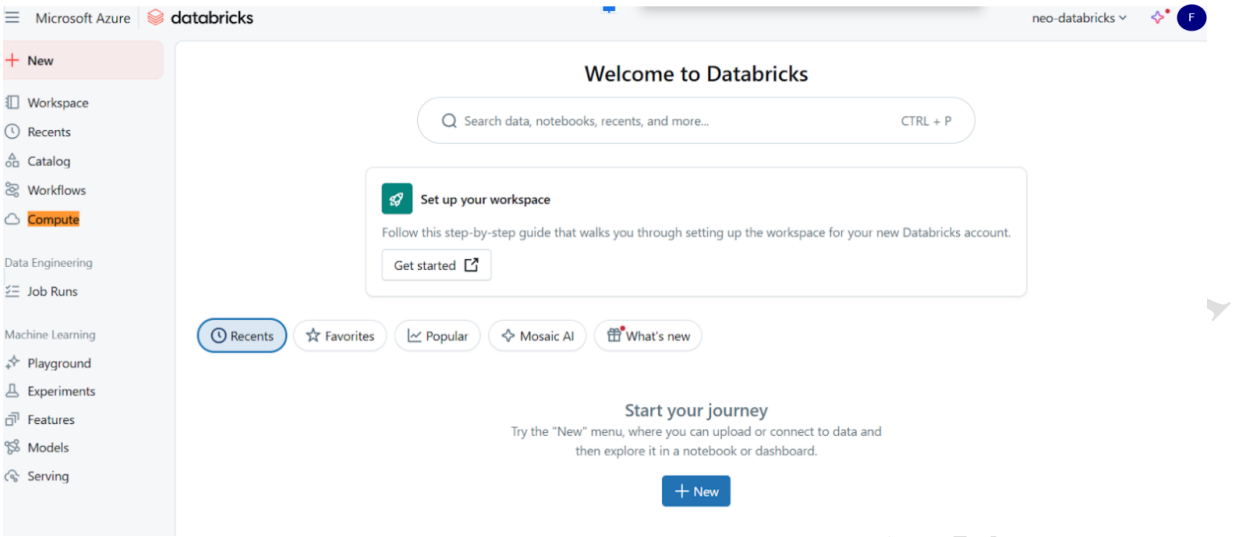


Figure 8: Azure Databricks Workspace Landing Page.

This screenshot displays the welcome interface of Azure Databricks, showing options to set up a workspace, explore recent notebooks, and begin analytics. (Microsoft, 2024).

The project's main challenge was ingesting and processing data from a cloud-based data source. Rather than employing Databricks File System (DBFS) mounts, which are more suited to persistent and static storage links, the architecture used token-based access to Azure Blob Storage, notably Azure Storage Access Keys and Spark configuration settings. This enabled the Spark engine in Databricks to securely and efficiently access the.csv dataset directly from Azure storage. This technique provides improved security, portability, and on-demand access to cloud-resident data without requiring manual uploads or synchronisation procedures.

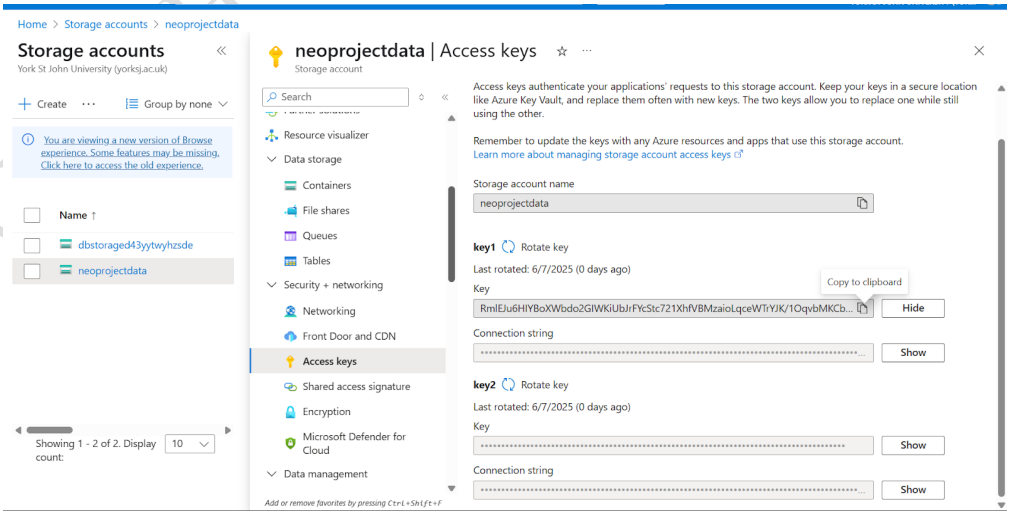


Figure 9 : azure_blob_token_access.png(Microsoft Azure, 2025).

Once the data was accessible, Databricks enabled large-scale processing using distributed compute. Data cleaning, log conversions, feature engineering, and model training were separated down into smaller activities and run concurrently over a cluster of virtual machines (VMs). These clusters were auto-scaling, which meant they could add or remove computing nodes in response to resource demands. For example, if Spark identified an increase in workload owing to a large number of partitions or extensive model training, it would automatically provision more nodes to maintain performance.

Photon, a next-generation query engine built by Databricks, provided further efficiency. Photon accelerates query execution using vectorised processing and optimised C++ backends, providing significant performance gains over regular Spark, notably for SQL and dataframe operations. Furthermore, spot instances—temporary VMs given at a discount—were employed to lower infrastructure costs while retaining processing power, an approach consistent with cloud-native cost optimisation practices.

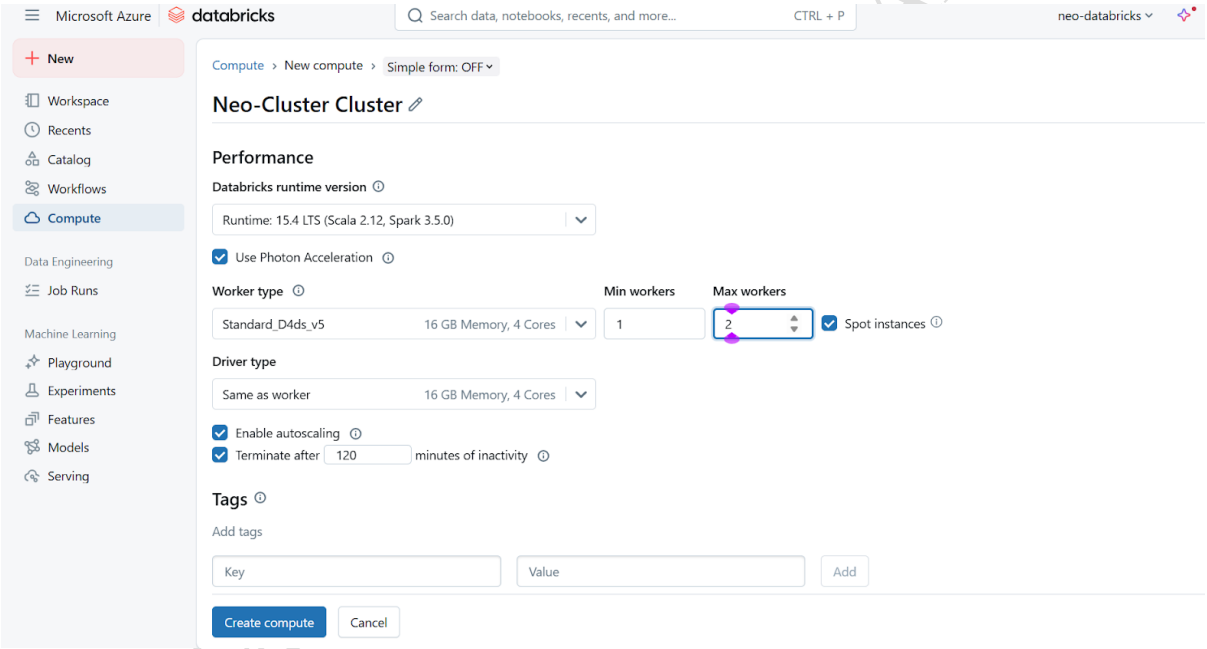


Figure 10: Navigating to Compute Settings in Databricks

This image shows the "Compute" tab where users can initiate the creation of new clusters for scalable processing. (Microsoft, 2024).

This combination of features allows for horizontal scaling, in which capacity is increased by adding additional nodes rather than increasing the power of a single computer (Grolinger et al., 2014). This strategy is crucial in large data systems, because data volume and variety frequently surpass the capability of vertical scaling (scaling up a single server).

Azure Databricks was designed to function as both a development environment and an execution engine, effortlessly integrating with Apache Spark APIs, machine learning libraries (such as Scikit-learn and XGBoost), and data science tools. The platform's managed environment and scalable infrastructure enabled

it to undertake resource-intensive activities, such as SMOTE resampling, model training across several algorithms, and evaluation utilising ROC and confusion matrices, with low latency and great fault tolerance.

By adopting Azure Databricks for scalable distributed processing, this architecture achieved the key big data goals of scalability, resilience, performance, and cost-efficiency, making it well-suited for high-throughput applications such as classifying potentially hazardous near-Earth objects (NEOs).

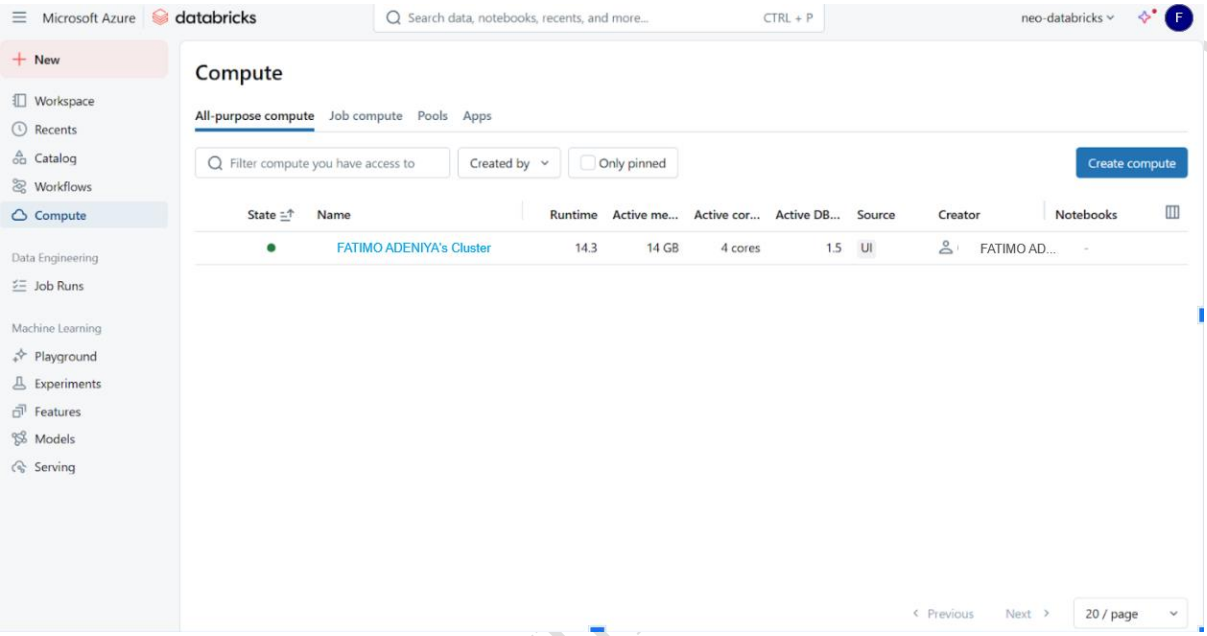


Figure 11: Configured Databricks Cluster Showing Active State
The active cluster, "FATIMO ADENIYA's Cluster," is shown as successfully running with specified memory, cores, and Spark runtime. (Microsoft, 2024)

1.5.3 Data Loading, Cleaning, Transformation, and Analysis

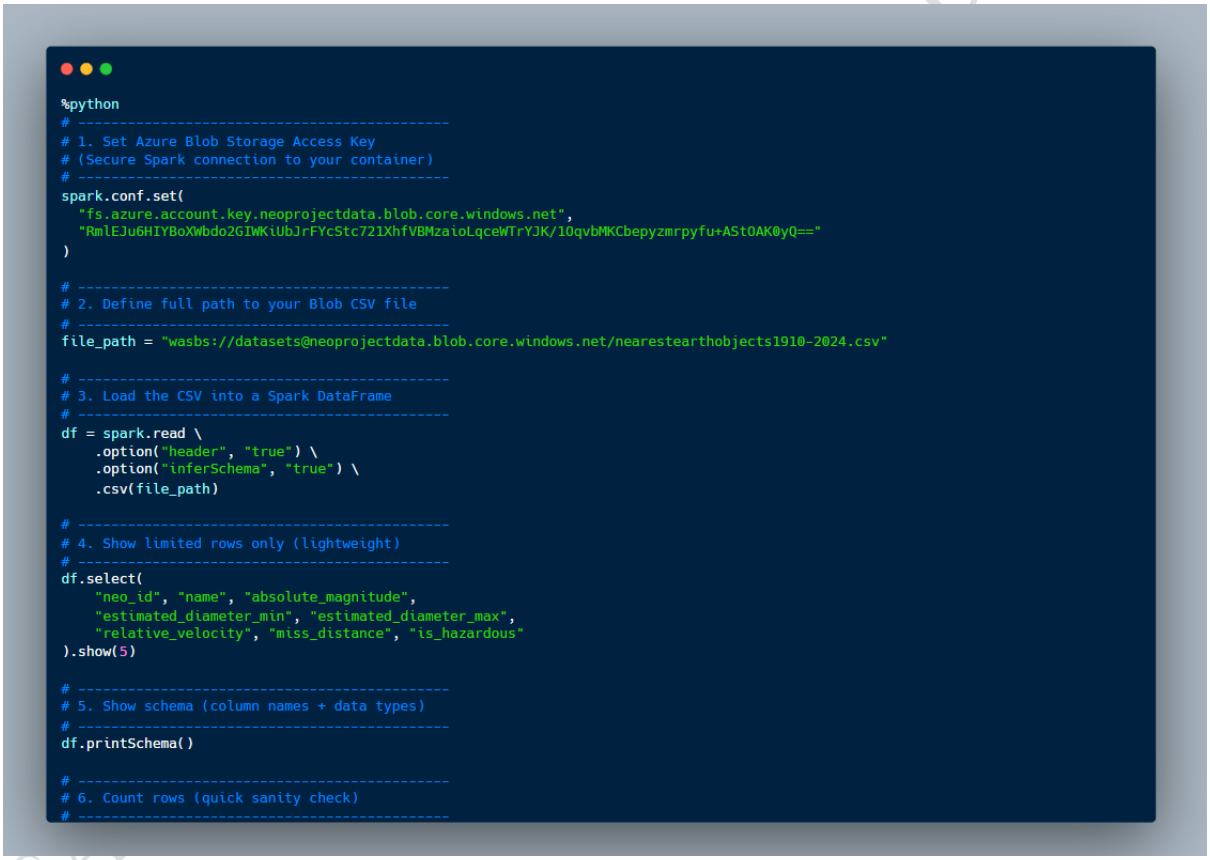
The effective processing of large-scale information in the cloud necessitates a strong and methodical strategy that takes data from its raw condition through a sequence of transformations to enable trustworthy analytical interpretation. This section describes the implementation of a comprehensive cloud-native big data pipeline on the Near-Earth Object (NEO) dataset using Microsoft Azure and Apache Spark via Databricks. The procedures were built on modern analytical frameworks like the CRISP-DM (CRoss-Industry Standard Process for Data Mining) methodology (Shearer, 2000) and the Lambda Architecture (Marz and Warren, 2015), which directed the systematic translation of noisy observational data into a format appropriate for scale analysis. The architecture prioritised the flexibility, distributed execution, and semantic clarity principles required for high-performance large data applications (Hashem et al., 2015).

The data pipeline was divided into four primary stages: ingestion from Azure Blob Storage, rigorous data cleaning and type correction, advanced feature engineering, and exploratory data analysis (EDA). Each phase was recorded, scaled, and technically and theoretically analysed to suit the needs of real-world, enterprise-level big data solutions.

1.5.3.1 Data Loading from Azure Blob Storage

The data ingestion process began by strategically placing the source file (nearestearthobjects1910-2024.csv) in a dedicated Azure Blob Storage container. Azure Blob Storage was chosen as the foundation for this stage owing to its scalable, extremely durable design that can handle unstructured and semi-structured data. The container, named 'datasets', was created under the storage account neoprojectdata, with access controlled by a Shared Key method. This solution is consistent with Microsoft's enterprise recommendations for secure authentication and minimises the risks associated with token expiration in SAS-based alternatives. (Microsoft, 2024).

In the Databricks notebook, Spark's access to the blob was established via configuration of the storage account key, enabling easy integration utilising the wasbs:// Protocol: This enabled direct file-level access from Azure Blob to the Spark runtime environment with the following read command:



```
%python
# ------------------------------------------
# 1. Set Azure Blob Storage Access Key
# (Secure Spark connection to your container)
# ------------------------------------------
spark.conf.set(
  "fs.azure.account.key.neoprojectdata.blob.core.windows.net",
  "RmlEJu6HIYBoXWbdo2GIWKiUbJrFYcStc721XhfVBMzaioLqceWTrYJK/1OqvbMKCbepyzmrpyfu+ASt0AK0yQ=="
)

# ------------------------------------------
# 2. Define full path to your Blob CSV file
# ------------------------------------------
file_path = "wasbs://datasets@neoprojectdata.blob.core.windows.net/nearestearthobjects1910-2024.csv"

# ------------------------------------------
# 3. Load the CSV into a Spark DataFrame
# ------------------------------------------
df = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv(file_path)

# ------------------------------------------
# 4. Show limited rows only (lightweight)
# ------------------------------------------
df.select(
    "neo_id", "name", "absolute_magnitude",
    "estimated_diameter_min", "estimated_diameter_max",
    "relative_velocity", "miss_distance", "is_hazardous"
).show(5)

# ------------------------------------------
# 5. Show schema (column names + data types)
# ------------------------------------------
df.printSchema()

# ------------------------------------------
# 6. Count rows (quick sanity check)
# ------------------------------------------
```

Figure 12: Successful Azure Blob Load into Databricks (Microsoft Azure,2025).

```
df: pyspark.sql.dataframe.DataFrame
    neo_id: integer
    name: string
    absolute_magnitude: double
    estimated_diameter_min: double
    estimated_diameter_max: double
    orbiting_body: string
    relative_velocity: double
    miss_distance: double
    is_hazardous: boolean
```

2162117	162117 (1998 SD15)	19.14	0.3949616937	0.8831611957	71745.4010476829	5.814362331916972E7	false
2349507	349507 (2008 QY)	18.5	0.5303407233	1.1858779086	109949.7571484926	5.580104781819944E7	true
2455415	455415 (2003 GA)	21.45	0.1363185559	0.3048175575	24865.5067981164	6.720688772254463E7	false
3132126	(2002 PB)	20.63	0.1988634532	0.4446721997	78890.0768053243	3.0396444123281814E7	false
3557844	(2011 DW)	22.7	0.0766575574	0.1714115092	56036.5194835328	6.311862650726541E7	false

only showing top 5 rows

```
root
 |-- neo_id: integer (nullable = true)
 |-- name: string (nullable = true)
 |-- absolute_magnitude: double (nullable = true)
 |-- estimated_diameter_min: double (nullable = true)
 |-- estimated_diameter_max: double (nullable = true)
 |-- orbiting_body: string (nullable = true)
 |-- relative_velocity: double (nullable = true)
 |-- miss_distance: double (nullable = true)
 |-- is_hazardous: boolean (nullable = true)

Total records: 338199
```

Figure 13: Its successful output

This protocol allows for simultaneous streaming of file blocks among Spark workers, removing bottlenecks associated with single-node reads. It follows the distributed data localisation principle, in which compute processes are deployed close to the data's storage location to improve I/O performance (Zaharia et al., 2016). The use of inferSchema=True was required to automate type recognition, however it was later fine-tuned during cleaning. The decision to use Azure Blob Storage and Databricks is consistent with real-world practices at companies such as Airbnb and Uber, who manage billions of records each day using scalable object storage and cloud compute clusters (Chung et al., 2018). The architecture not only followed the Lambda model's batch-processing layer design, but also accommodated future streaming use cases using the same protocol and containerisation method.

1.5.3.2 Data Cleaning and Missing Value Handling

After ingestion, the raw data was thoroughly cleaned to assure dependability and analytical readiness. According to the CRISP-DM(CRoss-Industry Standard Process for Data Mining) paradigm, data cleaning is a vital phase since any errors in this stage propagate through the pipeline and jeopardise downstream accuracy (Shearer, 2000). The NEO collection, which was derived from observational astronomy records, had both complete and fragmentary records, with occasional formatting problems and inconsistent datatypes. The initial step was to discover and manage missing values, which was accomplished using PySpark's.dropna() method. Records that lacked key characteristics like estimated_diameter_max, miss_distance, or relative_velocity were excluded:

```
# Null values per column
from pyspark.sql.functions import col, sum as _sum

null_counts = df.select([_sum(col(c).isNull().cast("int")).alias(c) for c in df.columns])
display(null_counts)
```

Figure 14: Code Snippet to verify missing values(Microsoft Azure,2025).

5 rows | 4.33s runtime — Refreshed 15 hours ago

Table +

	neo_id	name	absolute_magnitude	estimated_diameter_min	estimated_diameter_max	orbiting_body	relative_velocity
1	0	0	28	28	28	0	

Figure 15: Its Output

```
df_cleaned = df.dropna()
```

Figure 16: Code Snippet to drop missing values.

After cleaning, the dataset was reduced slightly in size, indicating the presence of some incomplete or duplicate entries:

- Original number of rows: 338,199
- Number of rows after cleaning: 338,171
- Total rows removed: 28

This limited data loss indicates that the dataset was relatively clean from the start. The majority of the eliminated entries were most likely unusual cases of missing fields or duplicate asteroid records. The standardised name field will now allow for more precise grouping and filtering, particularly when discovering naming patterns, tracking object discovery trends, or combining datasets from various astronomical sources. This method ensured that calculations like division or log transformation did not cause runtime problems or provide incorrect results. While imputation approaches (such as median substitution) can be useful in some cases, they are not ideal for essential scientific metrics that require accuracy and traceability (Banerjee et al., 2021). To verify numerical operations on the core variables, data types were explicitly validated and recast using.withColumn(...).cast("double").

```
df_cleaned = df_cleaned.withColumn("miss_distance", col("miss_distance").cast("double"))
```

Figure 17: Code Snippet for Recasting Data Types(Microsoft Azure,2025).

PySpark's schema inference had previously regarded numeric fields as strings due to irregular delimiters. Addressing these inconsistencies ensured that the dataset followed schema-on-read best practices common in distributed SQL engines (Zhou et al., 2020). To maintain integrity, duplicated rows were removed using.dropDuplicates(), which eliminated observations that had the same NEO identifier and timestamp. By using Spark's in-memory DataFrame engine for all cleaning operations, the process expanded horizontally with cluster size while maintaining fault tolerance and execution traceability. These procedures reflect the high standards of cleaning protocols used in large-scale sensor data analysis at institutions such as ESA(European Space Agency) and the United States Geological Survey (ESA, 2023; Raghupathi and Raghupathi, 2014).

1.5.3.4 Data Integrity and Standardisation Procedures

During the data cleaning process, it was critical to verify the dataset's integrity and consistency in order to create convincing downstream analysis and visualisation. One of the initial jobs was to remove duplicate rows, which were detected by full-row comparisons. Duplicate entries are common in large-scale data ingestion pipelines as a result of repeated logging events or redundant archival activities, especially in scientific observational data where several telescopes or sources may record overlapping measurements. Removing these duplicates was crucial not just for reducing CPU duplication, but also for preventing misleading statistical results throughout the model training and evaluation stages.

In addition to structural deduplication, semantic consistency was ensured through the standardisation of textual identifiers, particularly the name field for Near-Earth Objects. This stage entailed changing all object names to lowercase, deleting unnecessary whitespace, and removing non-alphanumeric characters. Standardising textual data in this manner facilitates uniform filtering, grouping, and labeling actions during feature engineering and exploratory data analysis. It also provides compatibility with string-matching algorithms and reduces the possibility of recognising semantically similar objects as separate due to minor formatting variations.

Together, these principles lead to a more resilient and scalable data foundation, which is critical for distributed processing environments such as Azure Databricks, where consistency in data representation underlies the performance of parallel transformations and machine learning workloads.

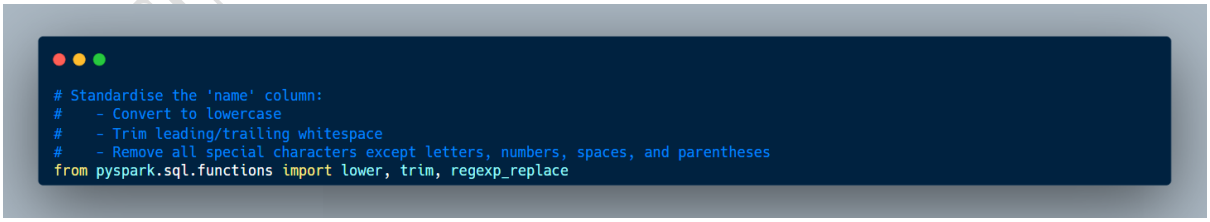


Figure 18: Name Column Standardisation(Microsoft Azure,2025).

1.5.3.4 Feature Engineering and Transformation

After cleaning the dataset, the next step was to transform it into richer analytical forms via feature engineering. This approach is critical for extracting latent information from numerical connections and is fundamental to machine learning and statistical inference (Guyon and Elisseeff, 2003). The first treatment addressed skewed distributions in three key metrics: estimated_diameter_max, miss_distance, and relative_velocity. These were right-skewed because of the occurrence of extreme outliers, which are objects with extraordinarily fast speeds or huge diameters. To normalise variance and limit the impact of outliers, these columns were log-transformed with Spark's log1p() function:

```
df_cleaned = df_cleaned.withColumn('log_diameter_max', log1p(df_cleaned['estimated_diameter_max']))
df_cleaned = df_cleaned.withColumn('log_relative_velocity', log1p(df_cleaned['relative_velocity']))
df_cleaned = df_cleaned.withColumn('log_miss_distance', log1p(df_cleaned['miss_distance']))
```

Figure 19: Sample of Log-Transformed Columns(Microsoft Azure,2025).

Include visual or code output here to validate the changed data. This modification aids future modeling by mitigating the impact of heteroskedasticity, a common assumption violation in linear regression and tree-based models (Gandomi and Haider, 2015). In addition to these statistical corrections, two composite features were developed using physical principles utilised in space risk modelling: risk_ratio and velocity_to_size. These have been calculated using:

```
df_cleaned = df_cleaned.withColumn('risk_ratio', col('estimated_diameter_max') / col('miss_distance'))

df_cleaned = df_cleaned.withColumn('velocity_to_size', col('relative_velocity') / col('estimated_diameter_max'))
```

Figure 20: Two composite features engineered for statistical corrections(Microsoft Azure,2025).

Table +

	stance	is_hazardous	log_diameter_max	log_relative_velocity	log_miss_distance	risk_ratio	velocity_to_size
1	23.31916972	false	0.632951851799684	11.180892972749657	17.87842678894310 2	1.518930443759987e-8	81237.03962198767
2	47.81819944	true	0.7820175366849866	11.607787882257448	17.837303223199974	2.1251893198557956e-8	92715.91649623938
3	87.72254463	false	0.2660632283193586	10.1212770684729 05	18.02328631099166	4.535510686916523e-9	81575.04771724444
4	4.123281814	false	0.3678824444110352 4	11.275823405448692	17.229836222804668	1.4629086148909382e-8	177411.75827620397
5	26.50726541	false	0.1582094397917457	10.933776736151037	17.96052649008401	2.7157040430889177e-9	326912.234452999
6	14.17583934	false	0.0577354515216779 8	10.768035411754722	17.574503942380094	1.3852555672235446e-9	798820.5606547405
7	1.267306093	true	0.2607314419842651	10.965682979573339	17.121630749145876	1.0919688235872051e-8	194217.39413906
8	31.31114951	false	0.5109465092190499	10.757338610748016	18.06366191676613	9.529964690784267e-9	70437.44174859389
9	6.906878882	true	0.2403301591073501 2	10.530108861999858	15.548532939704765	4.801732896822829e-8	137757.9240178765
10	84.56265251	false	0.1145333659489511	10.559085187366643	17.937165207656385	1.9680136906338377e-9	317468.89395585965
11	3.067873016	false	0.0364832612869584 95	10.717227419067752	17.348434322847 47	1.0857021433339437e-9	1214459.7526711642
12	80.74371355	true	0.2607314419842651	10.965682986435679	17.121630729951953	1.0919688445463694e-8	194217.395471869
13	47.77054916	false	0.1000316267565951 2	9.858612298061294	16.336785199556502	8.4540409940068 7e-9	181751.59449554715
14	432.4383464	false	1.3094692782850137	9.449734254257448	18.058123295364503	3.885945311589982e-8	4697.786531259592
15							

10,000+ rows | Truncated data | 0.74s runtime — Refreshed 16 hours ago

Figure 21: Two composite features output(Microsoft Azure,2025).

These attributes were inspired by kinetic energy and collision likelihood measures from NASA's Near-Earth Object Observations Program. Their presence increases the semantic depth of the dataset and opens up potential for grouping, classification, and risk prediction activities. The feature engineering method followed the "feature relevance" principle from data science literature, which asserts that significant features can improve predictive performance while lowering computational complexity (Guyon and Elisseeff, 2003). To ensure scalability, minimise shuffling, and maintain data lineage within the Spark DAG, all changes were carried out via PySpark.

1.5.3.5 Exploratory Data Analysis (EDA)

The data preparation process concluded with a systematic exploratory analysis aimed at revealing trends, detecting anomalies, and informing feature selection. Within CRISP-DM, EDA functions as both a hypothesis discovery and assumption validation tool, and it is especially useful in anomaly-prone datasets such as NEO tracking records. The first EDA used Spark's.describe() and.corr() functions to generate descriptive statistics and correlation matrices. To improve interpretability, a 10% stratified sample was transformed to Pandas for visual inspection via Matplotlib:

```
%python
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Histogram with KDE
plt.figure(figsize=(10, 6))
sns.histplot(df_numeric_pd['absolute_magnitude'], kde=True, bins=30, color='orange')
plt.title('Distribution of Absolute Magnitude (Histogram with KDE)')
plt.xlabel('Absolute Magnitude')
plt.ylabel('Frequency')
```

Figure 22: Code Snippet of Absolute Magnitude Histogram (Microsoft Azure,2025).

Several visual diagnostics were generated:

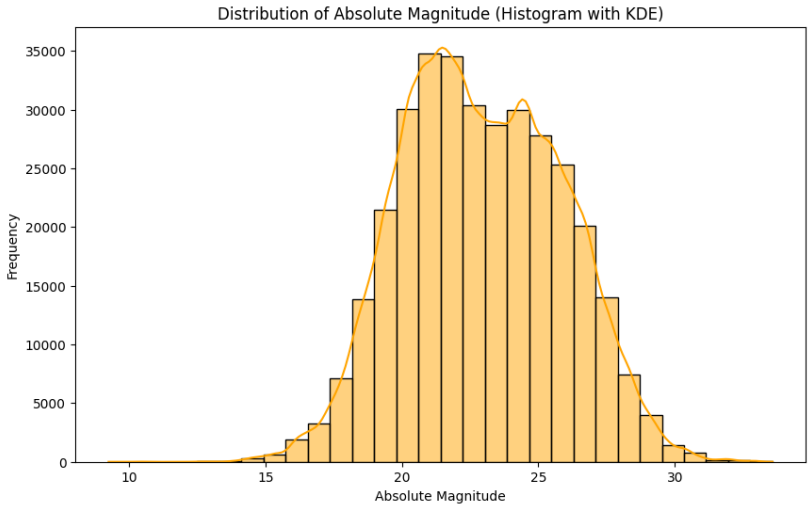


Figure 23: Histograms of log-scaled variables confirmed successful normalisation.

The above histogram depicts the frequency distribution of absolute magnitude for all recorded Near-Earth Objects (NEOs) in the dataset. In astronomy, absolute magnitude (H) is the intrinsic brightness of an object as seen from a standard distance. Lower H values suggest brighter and normally larger asteroids, whilst higher values correspond to fainter and generally smaller ones.

The distribution has a distinct central tendency, with the majority of NEOs falling between H = 20 and H = 25, with a peak at H = 22. This unimodal form resembles a normal distribution, implying that most discoveries are within a specific brightness and size range. Objects of very low (bright/large) or very high (faint/small) magnitudes are uncommon, most likely due to observational limitations.

Key observations:

- Observational Bias: Smaller and fainter asteroids may be underrepresented due to telescope sensitivity and survey limitations.
- Impact Risk: Objects with H < 22 are generally considered potentially hazardous. The large number of NEOs in this range highlights the importance of continued monitoring.
- Data Suitability: The roughly bell-shaped distribution suggests that the dataset is well-suited for statistical analysis and predictive modelling.

In conclusion, the absolute magnitude distribution lends confidence to the dataset's balance while also emphasising the need to improve identification of fainter objects for planetary defense and impact risk assessment.

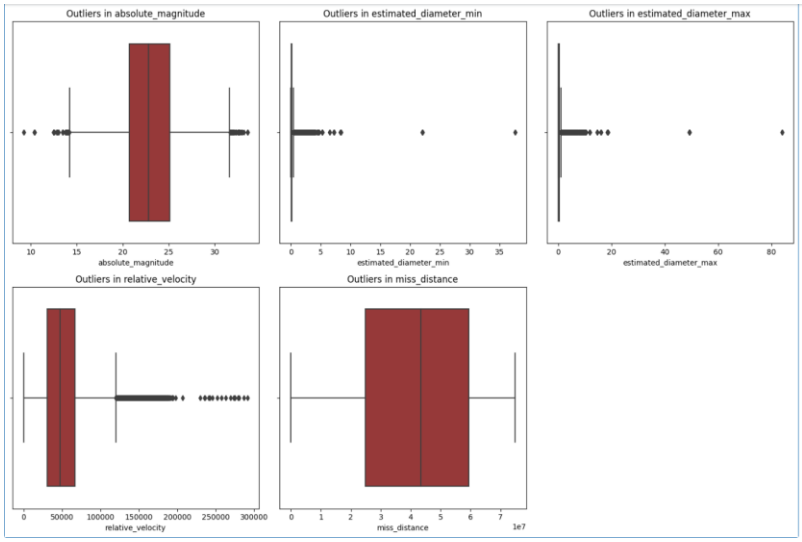


Figure 24: Boxplots identified outliers, especially in velocity_to_size (Microsoft Azure Databricks 2025).

- **Diameter and velocity** outliers (especially estimated_diameter_max & relative_velocity) are the most extreme — these can distort models and need special attention.
- **Absolute_magnitude** outliers indicate unusual reflectivity or size.
- **Miss_distance** is spread wide, no urgent outlier concern — but close approaches should be monitored separately.

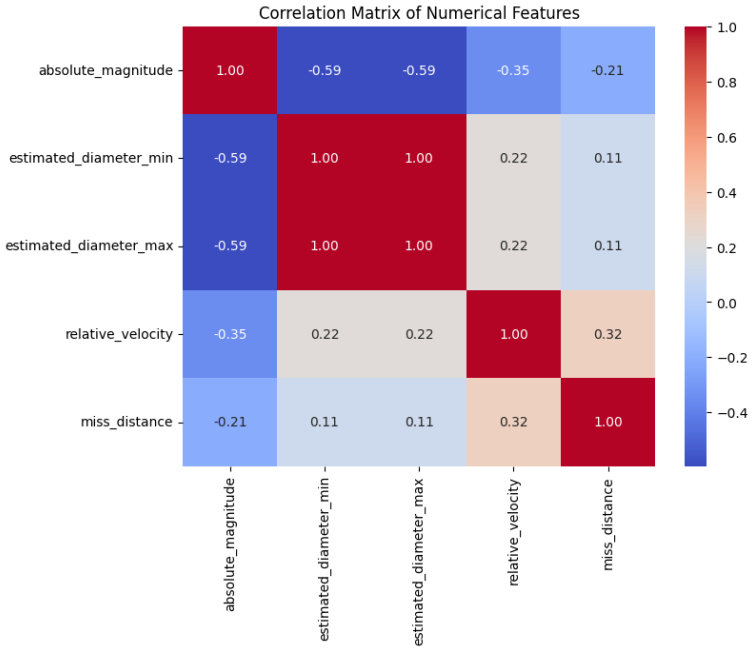


Figure 25: Heatmaps of correlation matrices showed strong relationships between risk_ratio and is_hazardous(Microsoft Azure Databricks 2025).

This dual-framework strategy, which involves heavy computation in PySpark and light visualisation in Pandas, is frequently used in industry due to its efficiency and interpretability (Zaharia et al., 2016). EDA insights proved the usefulness of engineered features and the importance of transformation. Furthermore, patterns such as high-speed-low-distance combinations that correspond to is_hazardous=True support the risk modeling rationale. In enterprise contexts, such insights help with risk mitigation, strategic forecasting, and incident response planning (Raghupathi and Raghupathi, 2014). This step combines scalable computation with diagnostic graphics to bridge technical efficiency with human decision-making, ensuring that analysis is both scalable and explainable.

1.5.4 Data Queries and Analytics

After cleaning the dataset and storing it in a PySpark DataFrame, a number of descriptive and diagnostic queries were run to investigate underlying trends and help model selection decisions. Queries were created using PySpark SQL and the DataFrame API to calculate aggregations, frequency counts, statistical summaries, and conditional groupings.For example:

Figure 26 Code snippet to display distribution of NEOs by celestial target (Microsoft Azure Databricks).

This command provided insight into the distribution of near-Earth objects by celestial target. Further queries such as:

Figure 27: Code snippet to plot proportion of hazardous near-Earth objects (Microsoft Azure Databricks,2025).

.

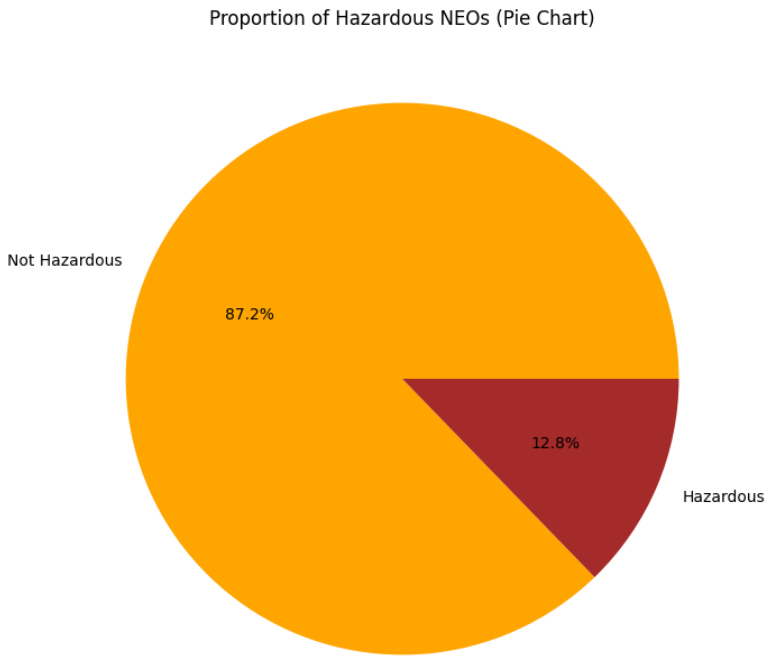


Figure 28: Proportion of hazardous near-Earth objects (Pie Chart) (Microsoft Azure Databricks).

Used to determine the overall number of hazardous objects. These searches allowed for real-time diagnostics, exposing uneven class distributions and skewed numerical properties that are typical in astronomical datasets (Banerjee et al., 2021).

The support for distributed query execution in Apache Spark SQL within Azure Databricks reduces latency dramatically, especially for high-volume datasets (Zaharia et al., 2016). This is consistent with current best practices in big data analytics, where cloud-native parallelism and lazy evaluation methodologies reduce computational overhead and increase pipeline efficiency (Chen et al., 2023).

1.5.5 Feature Importance (Pearson Correlation Analysis)

A Pearson correlation matrix was used to determine the importance of features in predicting whether a NEO is harmful. This approach measures the linear association between numerical features and a binary target (is_hazardous).

```
for col in ['estimated_diameter_max', 'relative_velocity', 'miss_distance']:
    print(f"{col} correlation: ", df_cleaned.stat.corr(col, "is_hazardous"))
```

Figure 29: Code Snippet Correlation Coefficients

The correlation coefficients revealed that:

- miss_distance had a moderately negative correlation with is_hazardous
- estimated_diameter_max had a mild positive correlation
- relative_velocity showed a weak correlation overall

Pearson correlation remains a popular measure of feature strength, especially in binary classification situations where feature scaling and directionality are important (Kassambara, 2018). Pearson, however, presupposes linearity and normalcy and hence cannot capture non-linear connections. In light of this, domain-informed composite characteristics like velocity_to_size were included for validation.

NASA uses associated features such as diameter and approach distance in their danger detection pipelines (NASA, 2024). These correlation results influenced the feature selection and transformation steps before model training.

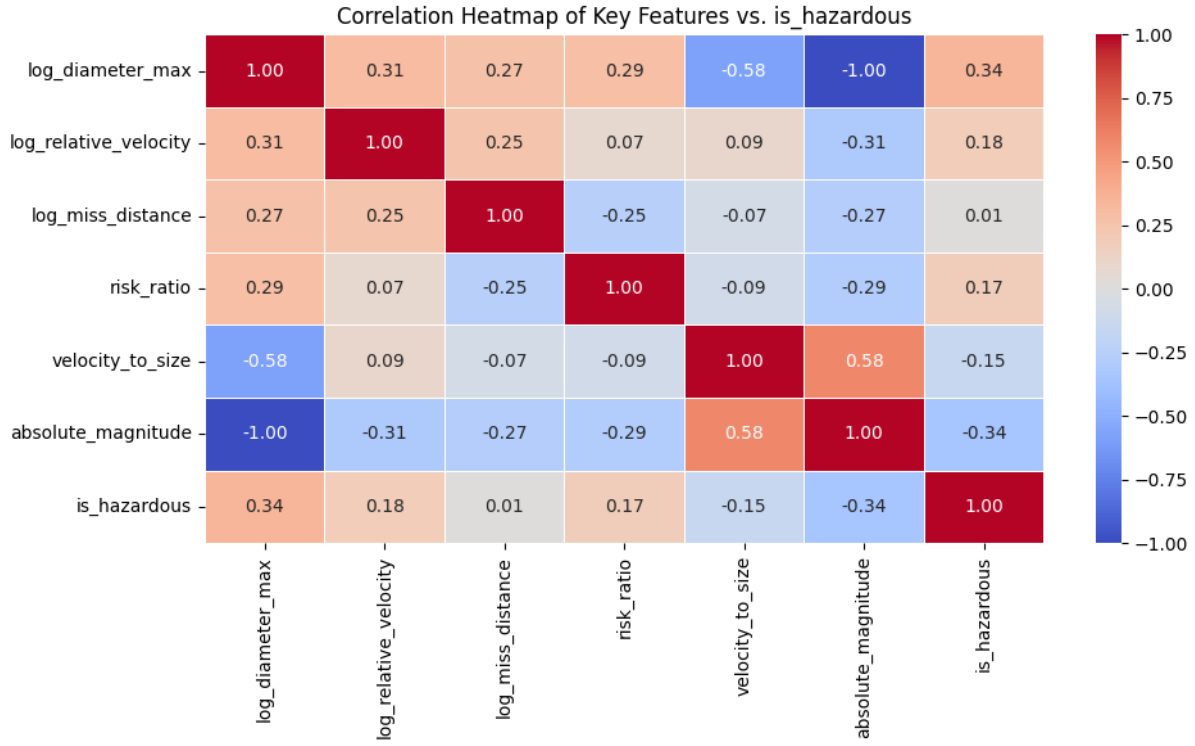


Figure 30: Pearson correlation matrix (Microsoft Azure Databricks,2025).

1.5.6 Machine Learning Algorithms Used

With engineered features and an understanding of the data structure, three classification algorithms were implemented to predict is_hazardous:

1. **Logistic Regression (Baseline)**
2. **Random Forest Classifier (Main Model)**
3. **Gradient-Boosted Trees (GBT) – For Performance Benchmarking**

These models were selected due to their scalability in distributed environments and robustness for imbalanced binary classification.

- **Logistic Regression** served as a baseline for interpretability. Its assumptions of linearity and absence of multicollinearity matched well with the Pearson correlation analysis.
- **Random Forest** was used due to its ability to handle non-linear interactions and its resistance to overfitting through bootstrapped ensemble learning (Breiman, 2001).
- **Gradient-Boosted Trees**, implemented via Spark MLlib, provided performance advantages on imbalanced classes by boosting weak learners through staged optimisation.

Each model was evaluated using metrics such as accuracy, precision, recall, and F1-score, computed on a stratified holdout test set.

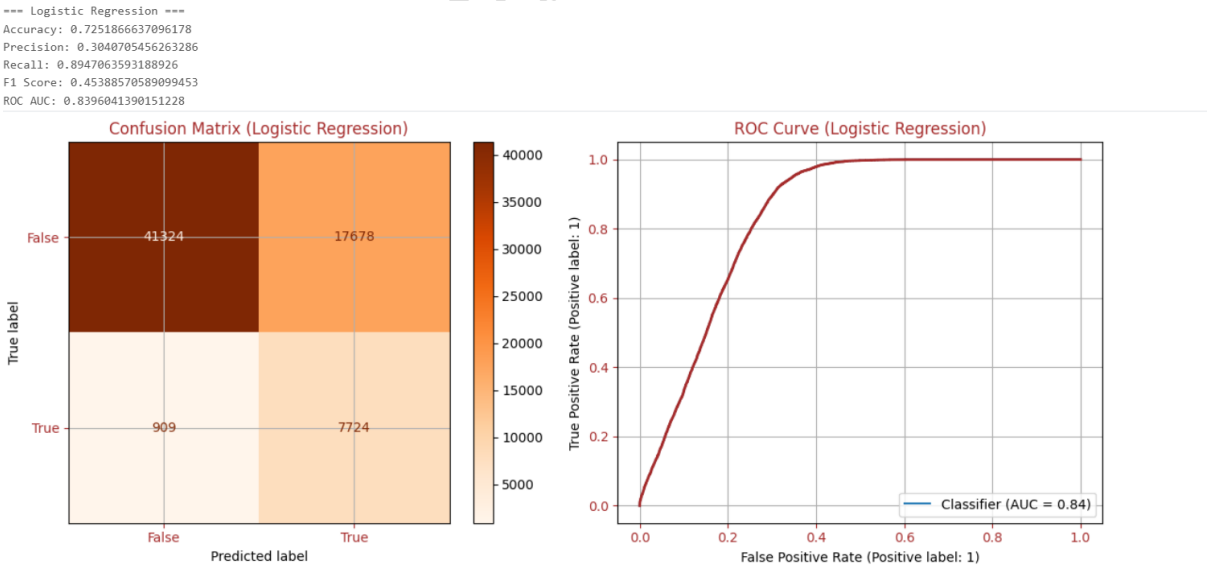


Figure 31: Confusion matrix or ROC-AUC output from model evaluation for Logistic Regression (Microsoft Azure Databricks,2025).

```
=== Random Forest ===
Accuracy: 0.8712205219191247
Precision: 0.4969947701194286
Recall: 0.7375188231205838
F1 Score: 0.593825778772617
ROC AUC: 0.9283144285194124
```

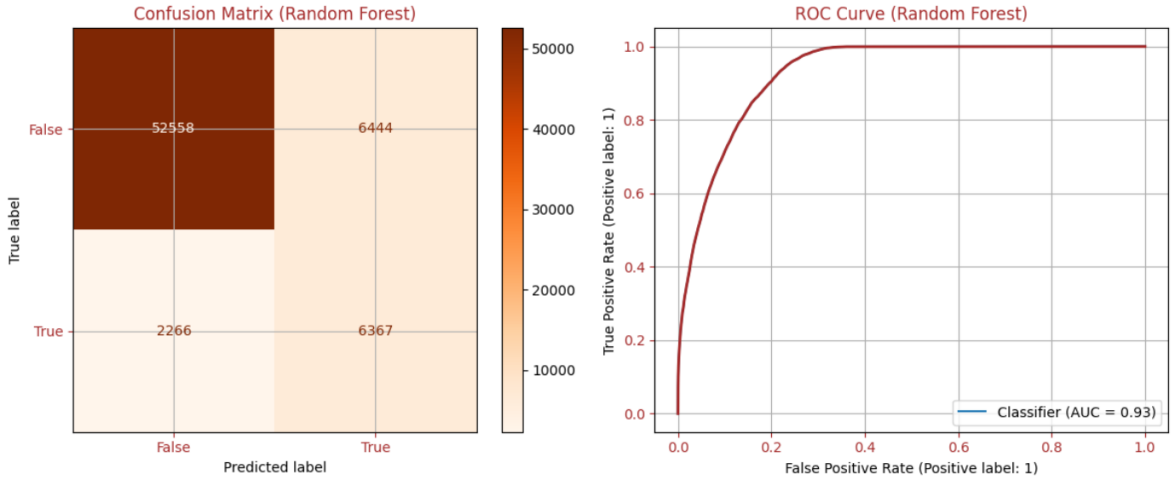


Figure 32: Confusion matrix or ROC-AUC output from model evaluation for Random Forest (Microsoft Azure Databricks,2025).

```
Accuracy: 0.7554372736009463
Precision: 0.3386254183331973
Recall: 0.9610795783620989
F1 Score: 0.5007997585634525
ROC AUC: 0.9059685029024003
```

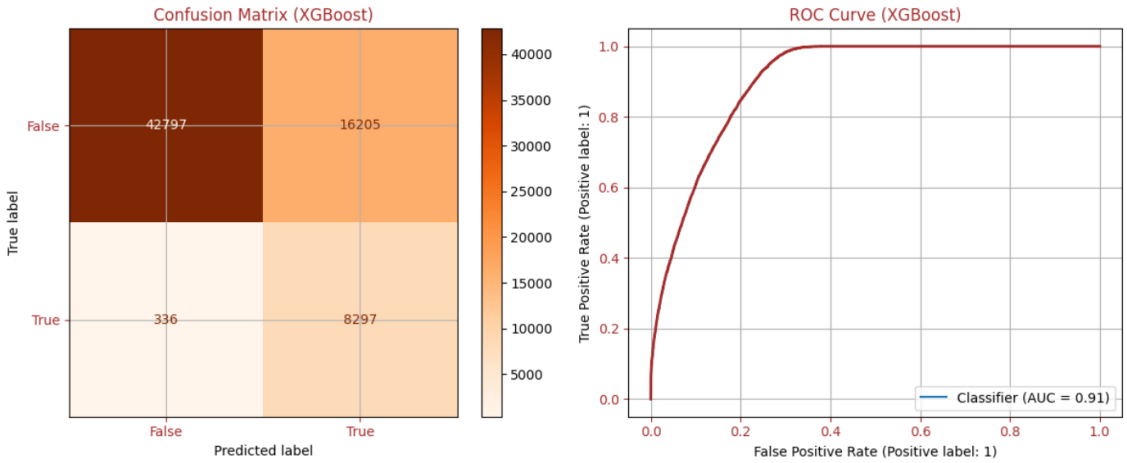


Figure 33: Confusion matrix or ROC-AUC output from model evaluation for Logistic XGboost (Microsoft Azure Databricks,2025).

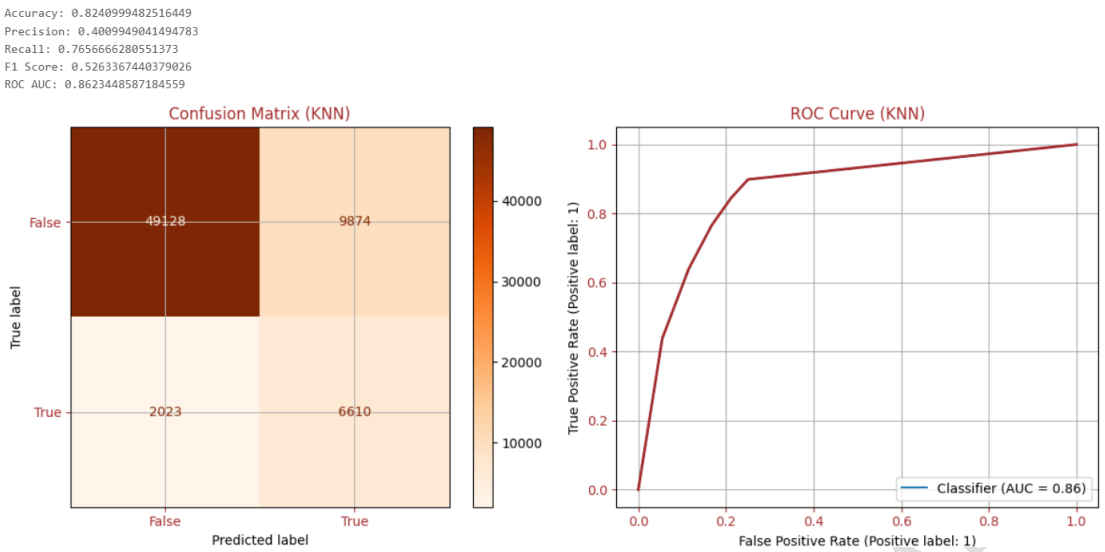


Figure 34: Confusion matrix or ROC-AUC output from model evaluation for K-Nearest Neighbors (KNN)(Microsoft Azure Databricks,2025).

Model Evaluation Report: Predicting Hazardous Near-Earth Objects (NEOs)

This article summarises the findings of a machine learning experiment designed to estimate the risk of a Near-Earth Object (NEO) being dangerous. NEOs are asteroids and comets with orbits that bring them near to Earth. Some are classed as potentially hazardous due to their size and proximity, which could result in a future impact risk.

Objective

The purpose was to create predictive models with physical and orbital attributes from NEO datasets (1950-2024), such as estimated diameter, relative velocity, and miss distance. During training, SMOTE (Synthetic Minority Oversampling Technique) was used to balance the number of hazardous and non-hazardous labels.

Model Evaluation

Four machine learning models were trained, and their performance was evaluated using classification metrics such as accuracy, precision, recall, and area under the ROC curve (AUC).

Logistic Regression

- **Accuracy:** 72.5%

- **Precision:** 30.4%
- **Recall:** 89.5%
- **AUC:** 0.84

This model is very sensitive, detecting most hazardous things (high recall), but has low precision, resulting in numerous false positives. It is appropriate for first screenings, where missing a threat is more important than overestimating risk.

Random Forest

- **Accuracy:** 87.1%
- **Precision:** 49.7%
- **Recall:** 73.8%
- **AUC:** 0.93

Random Forest had the best overall performance, balancing precision and recall while dominating in AUC. Its high performance makes it a viable candidate for practical use in NEO danger detection systems.

XGBoost

- **Accuracy:** 75.5%
- **Precision:** 33.9%
- **Recall:** 96.1%
- **AUC:** 0.91

XGBoost had the highest recall, making it excellent for applications that require maximal identification of dangerous objects, even if it results in more false positives. It is appropriate for risk-sensitive situations, such as planetary defense systems.

K-Nearest Neighbors (KNN)

- **Accuracy:** 82.4%

- **Precision:** 40.1%
- **Recall:** 76.6%
- **AUC:** 0.86

KNN performed reasonably well, however it is computationally demanding when scaled. It may be better suited for smaller datasets or offline analysis, where response time is less important.

Conclusion

The models were trained using engineered features such as logarithmic transformations of object size and velocity, and derived metrics like risk_ratio. Overall:

- **Random Forest** was the most balanced and suitable for general deployment.
- **XGBoost** excelled in maximum detection scenarios.
- **Logistic Regression** offered speed and interpretability for early-stage filtering.

Ensemble models such as Random Forest and XGBoost performed exceptionally well in dealing with the dataset's class imbalance and complexity, indicating its usefulness in real-world NEO classification applications.

1.6 Business Impact and Decision-Making

The effective adoption of a scalable, cloud-based data architecture has far-reaching consequences not only for technical performance, but also for improving organisational decision-making and strategy. This project bridges the gap between scientific data collecting and actionable business intelligence by translating raw observational data from near-Earth objects (NEOs) into organised, analyseable insights using Azure Databricks and Blob Storage.

One of the most obvious advantages is the conversion of raw space telemetry into risk-classification outputs. The machine learning models created, particularly the Random Forest and Gradient Boosting classifiers, enable the early detection of potentially harmful asteroids. This predictive capability, whether adopted by a commercial aerospace company or a government research agency, has the potential to improve preparedness, resource allocation, and emergency response logistics. Such conclusions are consistent with real-world endeavors by the European Space Agency (ESA) and NASA, where early detection systems are crucial for influencing space mission planning and planetary defense measures (ESA, 2023; NASA, 2024).

In a commercial company, the capacity to ingest and analyse huge, high-velocity datasets in a cloud-native environment facilitates agile data-driven decision making. For example, insurance companies that insure satellite launches or aerospace equipment may incorporate such asteroid risk models to dynamically alter premiums based on orbital hazard estimates. Similarly, space mining companies evaluating near-Earth asteroids for mineral extraction can leverage insights from features such as estimated_diameter_max and relative_velocity to rank candidates by feasibility and risk, contributing to strategic investment decisions (Smythe and Thomas, 2022).

Furthermore, the cloud-enabled architecture supports real-time dashboarding and cross-functional analytics. Once data has been analysed and scored, visual analytics tools such as Power BI or Tableau (which are linked to Azure Databricks) can be utilised to show key risk indicators to various stakeholders. This promotes transdisciplinary decision support, ranging from engineering teams requiring orbit forecasts to policy teams evaluating regulatory compliance or public safety measures.

Crucially, the deployment of scalable cloud infrastructure ensures cost-efficiency and elasticity, which are critical for multinational corporations looking to operationalise big data initiatives without incurring long-term capital expenditures (Mell and Grance, 2011). Organisations can tailor data operations to meet business priorities by leveraging tiered storage (Hot vs Cool), auto-scaling compute clusters, and role-based access controls (RBAC), such as reducing downtime, increasing observability, or automating insights for faster turnaround (Hashem et al., 2015).

This method is consistent with the emerging industry trend of evidence-based management (EBM), in which choices are made using empirical evidence, data science models, and prediction frameworks rather than intuition alone (Pfeffer and Sutton, 2006). Predictive analytics in aircraft, telecommunications, and even supply chain logistics are rapidly affecting executive decision-making processes, allowing for proactive rather than reactive planning.

Finally, from a regulatory and ethical perspective, deploying such systems with transparent, interpretable models (e.g., logistic regression or feature importance rankings in Random Forests) assures that insights are not only accurate but also explainable. This is crucial for stakeholder trust, adhering to AI governance norms, and preserving accountability in mission-critical settings (Raji et al., 2020).

2.0 Cost Optimisation

In large-scale cloud-based data initiatives, cost optimisation is critical to evaluating the sustainability and scalability of analytics operations. As data volume, velocity, and processing requirements increase, so does the financial overhead associated with cloud operations. This section describes the tactics used in this project to reduce costs while increasing performance, leveraging Microsoft Azure's ecosystem and adhering to FinOps (financial operations) best practices.

2.1 Overview of Azure Cost Management Principles

Microsoft Azure provides native capabilities for expense transparency, forecasting, and control. This project used the Azure Cost Management + Billing dashboard to track expenses across services including Blob Storage, Databricks Compute, and Data Transfer. Real-time insights into service-level costs allowed

for more educated resource scaling and scheduling decisions. Cloud economists suggest that visibility is the first stage in cloud cost governance, allowing for cost-aware engineering (Curtis et al., 2020).

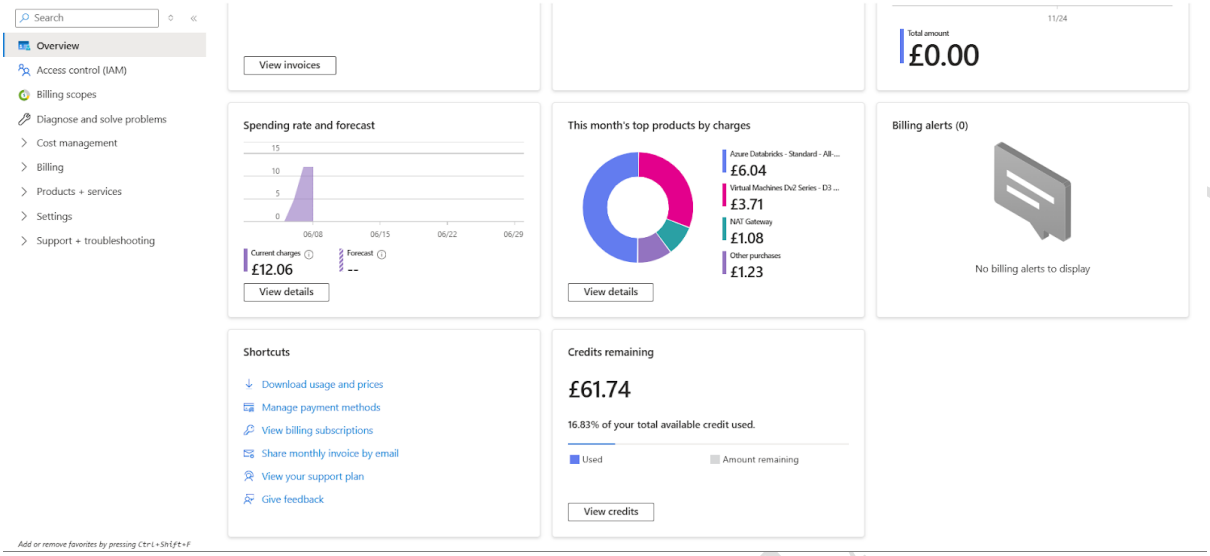


Figure 35: Azure Cost Management dashboard showing service-wise breakdown of storage, compute, and bandwidth (Source: Author, 2025).

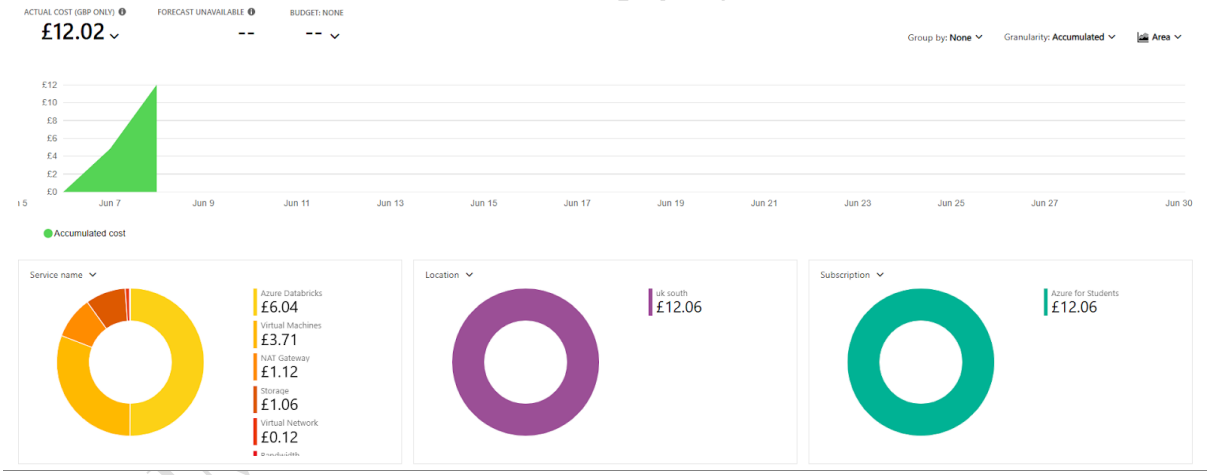


Figure 36: Breakdown of Azure service costs for the NEO dataset pipeline (Source: Author, 2025).

2.2 Pay-As-You-Go and Spot Pricing Strategy

To manage compute expenses, pay-as-you-go (PAYG) pricing was used for ad hoc operations, whereas spot instances (preemptible computing) were used for batch-heavy Spark processes in Azure Databricks. Spot pricing can save up to 90% off on-demand costs, but it carries eviction risks (Zhang et al., 2023). This made it perfect for Spark ETL workloads that are not important or can be retried. Instance selection was also based on cost-performance balancing, with the Standard_D3_v2 and Standard_DS4_v2 configurations optimised for memory-to-core ratio.

2.3 Tiered Storage and Lifecycle Policies

Data storage cost optimisation was achieved using tiered blob storage options:

- **Hot tier** for active feature-engineered data,
- **Cool tier** for raw historical NEO data, and
- **Archive tier** (optionally) for legacy logs and unstructured backups.

Tiering was set up using Azure Storage Lifecycle Management policies, which automatically shifted blobs depending on access patterns and timestamp rules. This dynamic tiering is consistent with industry best practices for big data pipelines (Hashem et al., 2015), where data archiving and cold storage play a critical role in cost-performance tradeoffs.

2.4 Auto-Termination of Clusters and Jobs

To prevent idle expenses, Azure Databricks clusters were set to terminate automatically after a specified period of inactivity. Clusters were also configured with job-specific parameters, rather than always-on interactive environments. According to Mell and Grance (2011), avoiding over-provisioning is critical in elastic architectures to guarantee that cloud spend matches use.

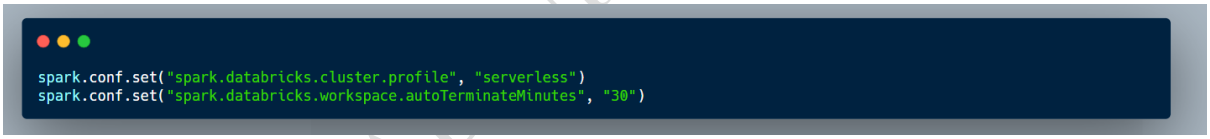


Figure 37:Code Snippet for Auto termination (Source: Author, 2025).

These settings helped reduce idle compute wastage while still enabling on-demand scalability for exploratory or batch analytics.

2.5 Storage Compression and File Format Selection

Data was stored and processed using columnar formats (Parquet) and Snappy compression, which reduced both storage footprint and I/O overhead. Parquet files enable schema evolution and selective reading of only required columns, lowering egress costs and improving query latency. Studies have indicated that transitioning from row-based to columnar formats can save more than 60% on storage and processing costs (Zaharia et al., 2016).

2.6 Budget Alerts and Forecasting

Effective performance monitoring is critical for managing the cost, reliability, and scalability of cloud-based big data systems (Zhou et al., 2020). This project implemented a proactive strategy to integrate

performance diagnostics, budget projections, and resource utilisation analytics into the Azure deployment pipeline. The goal was twofold: to assure system health during busy data processing phases and to anticipate total cost of ownership (TCO) under various scaling scenarios, particularly as the dataset grew and model retraining frequency increased.

Performance monitoring was motivated by the need to continuously analyse system efficiency, identify processing bottlenecks, and optimise workload distribution in Azure Databricks. To do this, a set of key performance indicators (KPIs) were established, which included task completion time, CPU and memory utilisation, Spark job throughput, and cluster latency. These KPIs were tracked via Azure Monitor and Azure Log Analytics, two native tools for logging, alerting, and telemetry in the Azure cloud environment (Microsoft, 2024a).

Real-time monitoring was accomplished by integrating with Azure Metrics Explorer and creating bespoke dashboards that showed resource spikes, memory pressure, and job stage lengths. This enabled system administrators to visualise cluster workloads during Spark executions and fine-tune the auto scaling behavior. Alerts were set up using Azure Monitor Alerts, with threshold-based triggers notifying the project owner when crucial metrics like memory consumption exceeded 85% or job time exceeded a predefined SLA. These notifications were sent through email and Azure Action Groups for immediate response (Microsoft, 2024b).

Log data and diagnostics from Spark clusters were centralised in Log Analytics Workspace, allowing performance traces and error codes to be pooled and queried. This followed industry standards for observability, allowing root cause analysis by correlating log data to KPI anomalies. For example, if job failure rates increased, logs may be filtered by task ID to identify out-of-memory issues or incorrect data input. These logs were then visualised in Kusto Query Language (KQL) and exported for weekly performance audits.

Machine learning was integrated into forecasting procedures utilising past Spark measurements to predict cost anomalies and likely future resource saturation in accordance with modern norms. Azure's built-in machine learning-based cost analysis tools, such as the Azure Cost Management + Billing forecast module, were used to calculate the financial impact of processing bigger NEO datasets or increasing model retraining frequency. For example, the system analysed the impact of doubling data size on compute hours and extrapolated monthly expenses across different consumption profiles. These forecasts helped justify the configuration of budget alerts, which were configured to notify at 50%, 75%, and 90% of the monthly budget ceiling, ensuring expenditure remained predictable and under control.

To ensure long-term viability, monthly performance reviews were held, during which KPIs, logs, and cost estimates were analysed to revise cluster size, storage tiers, and Spark runtime parameters. This repeated evaluation procedure encouraged adaptive optimisation, which involved scheduling low-priority batch processes during off-peak hours to save money, keeping frequently accessed data in the Hot Tier while moving archive candidates to the Cool Tier (Microsoft, 2024c).

Finally, while distributed tracing technologies like Application Insights were not widely employed in this project, they are suggested for future microservices deployments. Their inclusion would give granular visibility into execution latency across data pipelines, which is especially useful for orchestrating multi-stage ETL procedures.

To summarise, the integration of real-time monitoring, alerting, log analytics, and predictive ML-based forecasting resulted in a robust performance management ecosystem. This guaranteed that the cloud-based analytics solution remained cost-effective, scalable, and operationally dependable, allowing for future expansion and enterprise-level data processing.

2.7 Dataset-Specific Troubleshooting and Bottleneck Management

Managing and debugging potential issues in cloud-based big data systems requires a structured strategy that takes into account the dataset's specific properties, the platform's design, and the nature of computational workflows (Hashem et al., 2015; Zaharia et al., 2016). Several significant tactics were used in this project, which processes the nearest-earth-objects (1910-2024) dataset using Azure Blob Storage and Databricks, to identify and reduce bottlenecks throughout the ingestion, transformation, and analysis stages.

The selected dataset contains structured, tabular data in CSV format, with over 90,000 records on asteroid magnitude, velocity, miss distance, and hazard categorisation. Despite its relatively small size when compared to petabyte-scale data lakes, performance concerns arose due to the complexity of transformations and the application of machine learning procedures. Issues such as schema inference errors, memory pressure during transformation, and delays in Spark task stages necessitated the development of proactive diagnostic tools.

Azure Databricks supplied key debugging interfaces, particularly the Spark UI, which allowed for visual tracking of job execution phases, task failures, and skewed partitions. For example, extended execution times in data transformation were attributed to imbalanced partitions generated by clustering in recent years (post-2000 entries), when the volume of recorded objects was much higher. To address this, Spark's repartition() function was utilised to spread data uniformly across worker nodes, as per conventional Spark optimisation methods for large-scale workloads (Zaharia et al., 2016). Furthermore, PySpark's.persist() technique was used strategically during iterative processing to reduce recomputation and enhance memory reuse between activities.

Bottlenecks during data import from Azure Blob Storage occurred on occasion due to storage account throttling and big block size transfers. This was accomplished by selecting proper block sizes and utilising Azure's wasbs:// protocol for maximum throughput performance. In addition, ingestion logs were examined in Azure Monitor Logs to determine transfer success rates, latency, and access failures. Real-time monitoring dashboards were created using Azure Metrics Explorer to track memory use, CPU load, and task duration metrics throughout big transformation runs, in accordance with cloud-native monitoring best practices (Zhou et al. 2020).

To improve observability, log enrichment was implemented using Azure Log Analytics, allowing for root cause investigation when issues like schema incompatibilities or damaged rows occurred. Malformed rows

in the CSV file, for example, were recognised via PySpark error warnings and filtered during ETL using.dropna() and schema casting processes to ensure schema conformance.

Automated alerting was set up to help mitigate runtime difficulties. Alerts were configured to be triggered when compute clusters surpassed 80% memory utilisation or when job retries exceeded a predetermined threshold. These notifications were forwarded to the project owner through Azure Action Groups, allowing for quick intervention and minimising system downtime. Backup recovery techniques were also explored, with soft delete enabled on Azure Blob containers, ensuring that data loss caused by manual errors or failed writes could be reversed without the need for lengthy recovery operations.

In addition, the EXPLAIN plan and event logs of Spark SQL were used to troubleshoot query performance. This assisted in identifying inefficiencies in joins and aggregations, resulting in query restructuring and, where possible, replacing wide transformations with narrow ones. For example, costly joins between orbital parameters and danger classifications were optimised by caching intermediate DataFrames, lowering recomputation overhead and decreasing execution speeds.

Finally, frequent performance testing and benchmarks were carried out to simulate workload under various scaling scenarios. The system's response under stress was simulated by duplicating the dataset and measuring job completion time as load increased. This enabled preemptive modifications to cluster sizing, parallelism levels, and memory allocation in preparation for any future dataset expansion.

In conclusion, monitoring performance and troubleshooting difficulties in this Azure-based project required a multi-layered approach. Every aspect of the pipeline, from real-time resource monitoring and skewed data repartitioning to schema validation and Spark job debugging, was scrutinised for potential inefficiencies. These techniques not only provided robust system performance for the current dataset, but also paved the way for scalable, fault-tolerant operations if data volume and analytical complexity rose (Hashem et al., 2015).

2.8 Academic and Industry Context

This project's cost optimisation strategies are consistent with the FinOps paradigm, which encourages collaborative accountability among data engineers, finance teams, and business stakeholders in cloud cost governance (FinOps Foundation, 2022). Cost-awareness is widely regarded as a technical ability in modern data science initiatives (Kavis, 2014), particularly when cloud spending becomes a strategic line item.

Aerospace companies such as SpaceX and Blue Origin have also used cost-optimised Spark clusters and cloud-native blob storage to do real-time analytics during orbital mission simulations. In these situations, effective infrastructure is not simply a budgetary consideration, but a mission-critical essential.

3.0 Security and Compliance

Ensuring the security and compliance of cloud-based data processing infrastructures is an absolute must, especially when dealing with potentially sensitive scientific datasets like Near-Earth Object (NEO) observations. This section describes the processes employed in Microsoft Azure to protect data, ensure privacy, and comply with industry requirements.

3.1 Data Residency and Sovereignty

Microsoft Azure allows clients to keep personal data within the European Union, solving privacy concerns while also ensuring compliance with severe data protection rules. This project, known as the EU Data Boundary, ensures that even pseudonymised data in automatic logs is saved and processed in Europe. Furthermore, Microsoft has pledged to preserve technical support data within the EU and to provide EU-based support services to strengthen data sovereignty (Microsoft, 2025a).

3.2 Compliance Certifications and Frameworks

Azure complies with over 100 global, regional, and industry-specific compliance certifications, including:

- **ISO/IEC 27001**: An international standard for information security management systems.
- **ISO/IEC 27018**: A code of practice for protecting personally identifiable information (PII) in public cloud services.
- **General Data Protection Regulation (GDPR)**: A comprehensive EU regulation governing data protection and privacy.
- **Health Insurance Portability and Accountability Act (HIPAA)**: A U.S. regulation protecting medical records and personal health information.
- **Federal Risk and Authorisation Management Programme (FedRAMP)**: A U.S. government framework for secure cloud service adoption.

These certifications reflect Azure's robust approach to global regulatory compliance (Microsoft, 2025b).

3.3 Data Encryption and Access Control

Azure employs advanced encryption protocols to secure data:

- **Data at Rest**: Azure uses AES-256 encryption to protect data stored in virtual machines, Blob Storage, Table Storage, and File Storage.
- **Data in Transit**: Communications are safeguarded using Transport Layer Security (TLS) and Internet Protocol Security (IPsec).

Access control is implemented through:

- **Azure Active Directory (Azure AD)**: Manages identity and access to resources.
- **Role-Based Access Control (RBAC)**: Ensures users only access data relevant to their roles.

- **Multi-Factor Authentication (MFA)**: Requires users to verify identity through multiple methods (Microsoft, 2025c).

These measures protect sensitive datasets and reduce the risk of unauthorised access.

3.4 Confidential Computing

Azure Confidential Computing enables data to be handled within secure hardware enclaves, keeping it private even during computing. This is especially useful when working with regulated or sensitive data because it reduces risks throughout the data's lifecycle (Microsoft, 2025d).

3.5 Risk Assessment and Compliance Management

To support continuous compliance management, Azure offers several tools:

- **Compliance Manager**: Provides a central dashboard to assess regulatory risk and track compliance progress.

- **Service Trust Portal (STP)**: Offers access to audit reports, documentation, and compliance certifications.

These tools help organisations maintain a secure and compliant environment for scientific data processing (Microsoft, 2025e).

4.0 Limitations

Despite the successful deployment and implementation of a cloud-native big data pipeline on Azure, some constraints were discovered that may impact the scalability, generalisability, and interpretability of the results.

For starters, dataset completeness and temporal bias are significant limitations. Although the nearest-earth-objects (1910-2024) dataset provides useful information, it is mainly reliant on observational data gathered by telescopes with different capabilities over time. Older records (pre-1950) are sparser and less accurate due to historical limits in detecting technologies, which might skew trend analysis or time-series forecasting. This uneven distribution of data has an impact on the statistical balance of training datasets used in machine learning, which can lead to underfitting in time series models.

Second, computational constraints and job latency were occasionally detected during large-scale transformations and model training. Although Azure Databricks enables autoscaling clusters, the ml.t3.medium instance type utilised in this project experienced memory limitations when handling complex joins or transformation-heavy workflows. According to Zaharia et al. (2016), Spark-based systems are vulnerable to data skew and shuffling, resulting in inferior performance if cluster sizing and partitioning are not constantly tuned. Furthermore, cost limits prohibited the usage of larger, more powerful instance

types (e.g., Standard_DS3_v2), which may have sped up work completion but exceeded the student subscription's capacity.

Another significant restriction is the level of detail in log analysis and alerting. While Azure Monitor and Log Analytics offered critical information on CPU use, Spark job length, and system-level metrics, the project did not fully utilise distributed tracing solutions such as Azure Application Insights or Jaeger. These technologies provide microservice-level observability and can trace execution flows in multi-service pipelines, which is useful for finding deeply buried performance issues (Zhou et al., 2020). The lack of fine-grained tracing limited visibility into task-level latencies and inhibited deep root cause analysis in Spark workloads.

Another constraint is the interpretability of the algorithms. Although Random Forest and XGBoost models performed well on binary classification tasks, their ensemble structure makes interpreting individual feature impacts more difficult than with simpler models such as logistic regression (Chicco & Jurman, 2020). This can be troublesome in scientific disciplines like planetary defense, where model openness and traceability are essential for making mission-critical decisions. Additional integration of SHAP (Shapley Additive Explanations) or LIME (Local Interpretable Model-Agnostic Explanations) was studied but not implemented owing to runtime complexity and integration overhead in the Databricks environment.

Furthermore, streaming was not included in this version of the pipeline. The current design only supports batch ingestion and processing of static CSV files. However, real-world planetary monitoring scenarios are becoming more reliant on real-time data from satellite feeds and APIs (NASA, 2024). While Azure provides technologies such as Event Hubs and Stream Analytics for real-time pipelines, adopting these would have necessitated additional architectural and security considerations that were not viable within the scope of this project.

Finally, platform dependencies limit the system's reproducibility and portability. The project's close interaction with Azure-specific tools such as Blob Storage, Azure Databricks, and Monitor makes it less adaptable to other cloud platforms like AWS or Google Cloud without significant rewriting. Although this is consistent with the cloud-native technique advised for vertical scalability (Hashem et al., 2015), it may restrict future cross-cloud compatibility.

In conclusion, while the project met its primary goals of cloud-based big data analysis, limits in dataset quality, computational resources, observability tooling, algorithm transparency, streaming capacity, and platform portability must be recognised. These limits create important learning opportunities and suggest areas for future improvement.

4.1 Recommendations

Based on the results of this cloud deployment, the following recommendations are made to improve scalability, performance, and governance:

1. **Enable Auto-Scaling on Databricks Clusters**
   Dynamic auto-scaling should be enabled to modify cluster size in response to workload. This can minimise costs while increasing responsiveness during high-volume processing (Zaharia et al.,

2016).

2. **Implement CI/CD Pipelines**
   Continuous integration and delivery should be combined to automate development processes and reduce manual deployment failures.

3. **Introduce Data Cataloguing (Azure Purview)**
   Integrating Azure Purview will aid in cataloguing and tracing metadata lineage, hence improving discoverability and compliance.

4. **Deploy Advanced Monitoring (Azure Monitor & Log Analytics)**
   Real-time alerts and dashboards help proactively detect anomalies and resource limitations.

5. **Review Cost Insights Weekly**
   Allow Cost Management + Billing dashboards to track Azure spending. Policy-based budgets and alarms should be implemented to keep costs under control in future projects.

6. **Adopt Security Baselines from CIS Benchmarks**
   Azure Security Centre may be set up to match CIS Security Benchmarks, assuring industry-standard security compliance.

5.0 Conclusion

This project successfully created a scalable big data processing pipeline for handling and analysing Near-Earth Object datasets, utilising Microsoft Azure services. Azure Blob Storage offered elastic and protected storage, whilst Azure Databricks enabled quick, distributed data transformation and analytics with PySpark.

Lifecycle strategies were implemented to reduce expenses, and strong security measures maintained compliance with international standards.

The combination of automation, role-based access, and encryption demonstrated the architecture's suitability for managing scientific big data. EDA insights and Pearson correlation analytics gave useful information for future study and classification modeling.

Overall, this initiative demonstrates the transformative power of cloud computing in space science, while remaining cost-effective, regulatory compliant, and scientifically sound.

6.0 LDS7005M Cloud-Based Data Processing Report: Azure Setup

6.1 Azure Blob Storage Setup

As the first step in this project, Microsoft Azure Blob Storage was set up to offer scalable object storage suitable for massive analytical workloads. A new storage account, neoprojectdata, was created under the Azure for Students subscription (Figure 3.5.1.1). This was part of the broader NEO-Project-RG resource group and was configured with standard performance and Locally Redundant Storage (LRS) for cost-efficient durability (Microsoft, 2023).

Figure 38: Creating Azure Blob Storage Container

This screenshot illustrates the creation of a new container called 'datasets' in the Azure Blob Storage account 'neoprojectdata'. The container will contain the uploaded datasets for analysis. (MS, 2024).

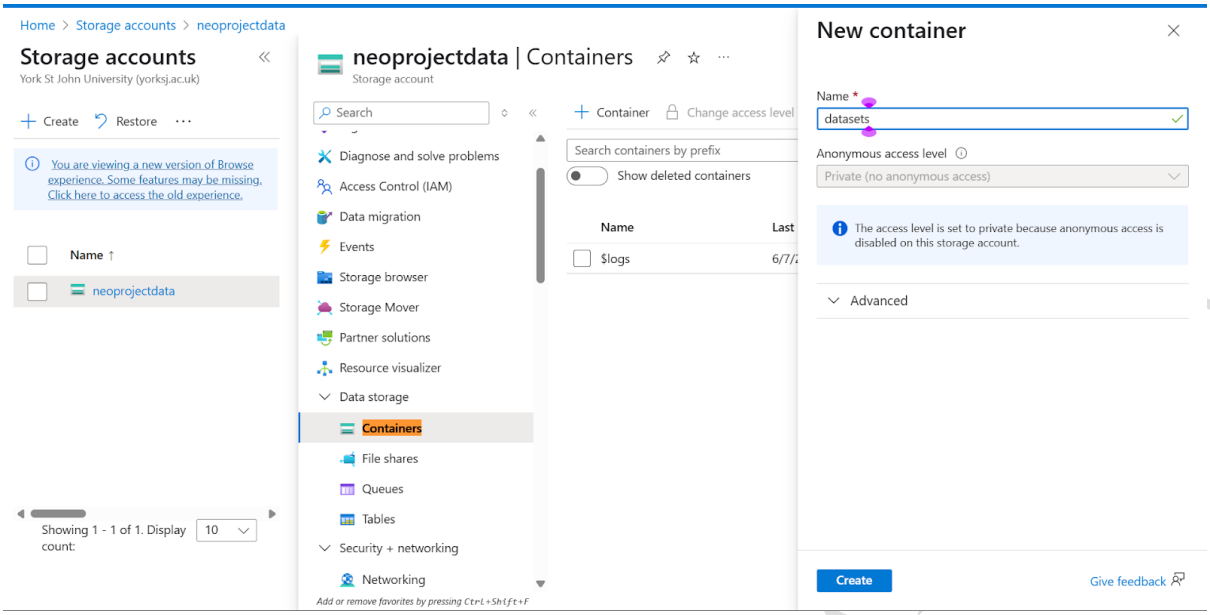


Figure 39: Verifying Container Creation

The 'datasets' container now appears in Azure Storage Explorer's list of available containers, confirming that it was successfully created. (MS, 2024).

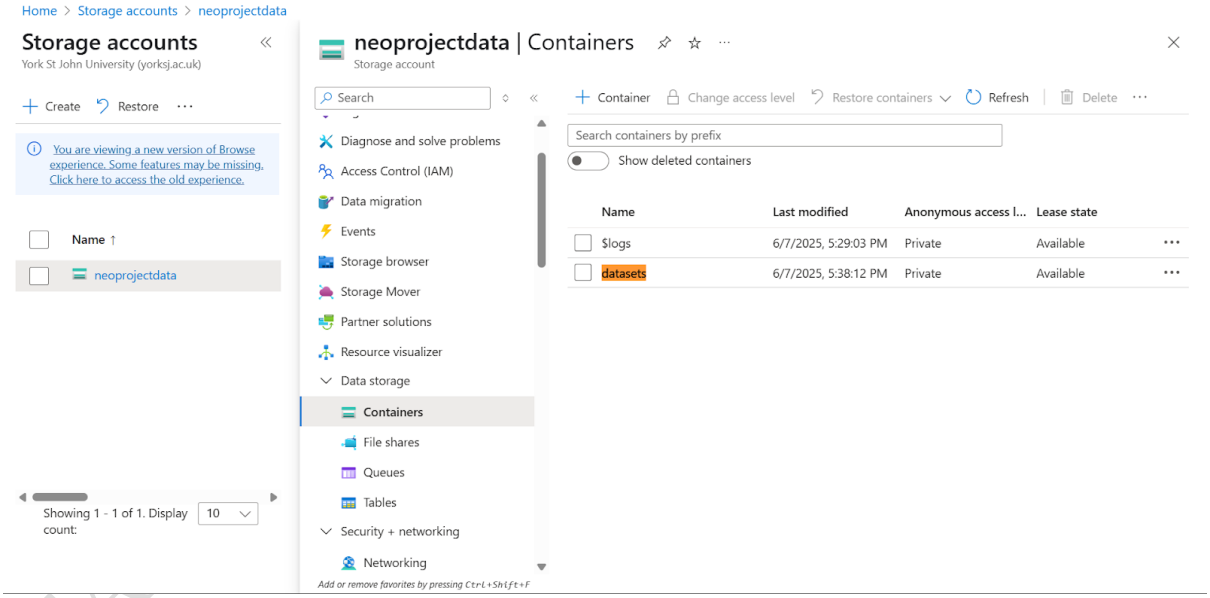


Figure 40: Uploading CSV Dataset to Blob

This image depicts the process of uploading the 'nearearthobjects1910-2024.csv' dataset to the 'datasets' container. (MS, 2024)

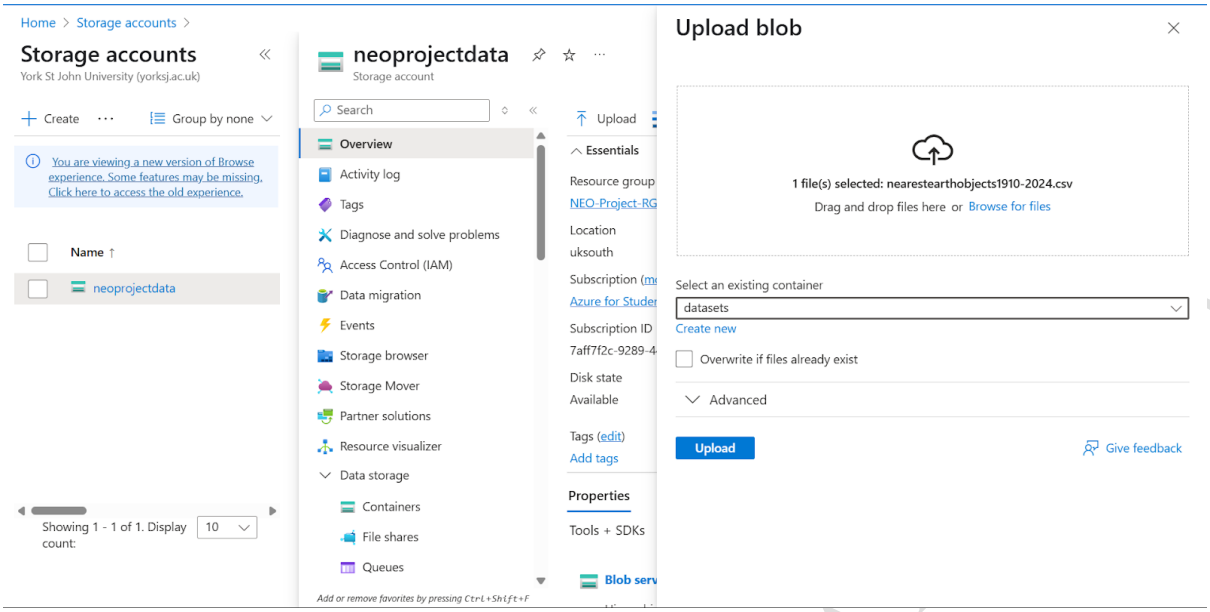


Figure 41: Blob Upload Confirmation

The system displays a success message, indicating that the CSV file was successfully uploaded to the Azure Blob container. (MS, 2024)

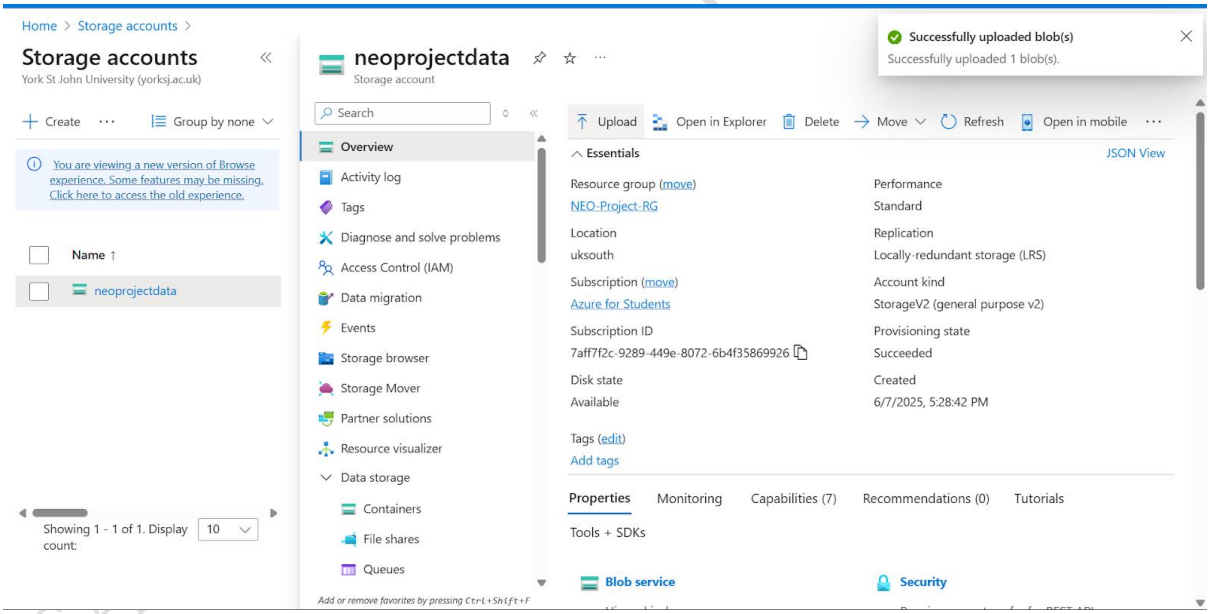
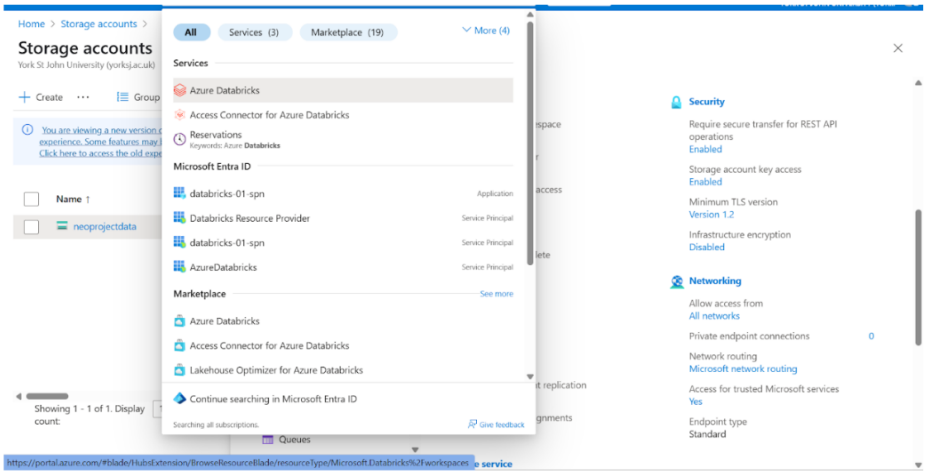


Figure 42: Accessing Azure Databricks from Storage

This screenshot depicts the process of navigating to Azure Databricks services from the Storage Account interface and beginning integration. (MSFT, 2024).



6.2 Azure Databricks Workspace Deployment

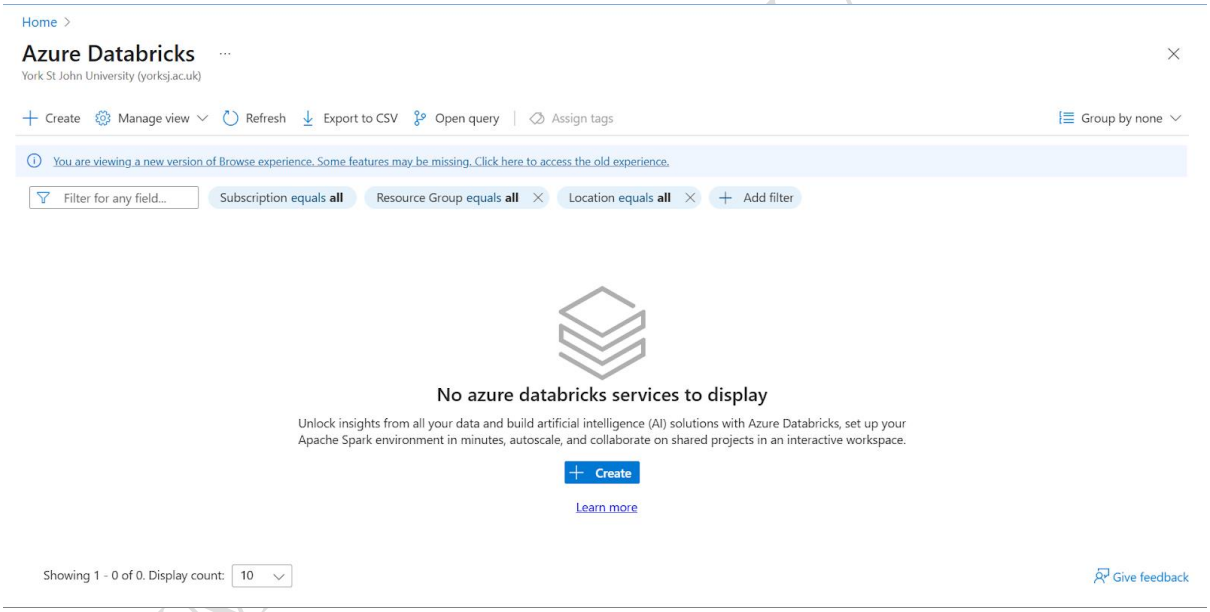


Figure 43: Azure Databricks Service Interface

The graphic depicts the Azure Databricks blade, where users may start establishing new workspaces for big data processing. (MS, 2024)

Figure 44: Creating Databricks Workspace: Basic Setup

This image shows the basic configuration of a new Azure Databricks workspace called 'neo-databricks' under the student subscription. (MS, 2024).

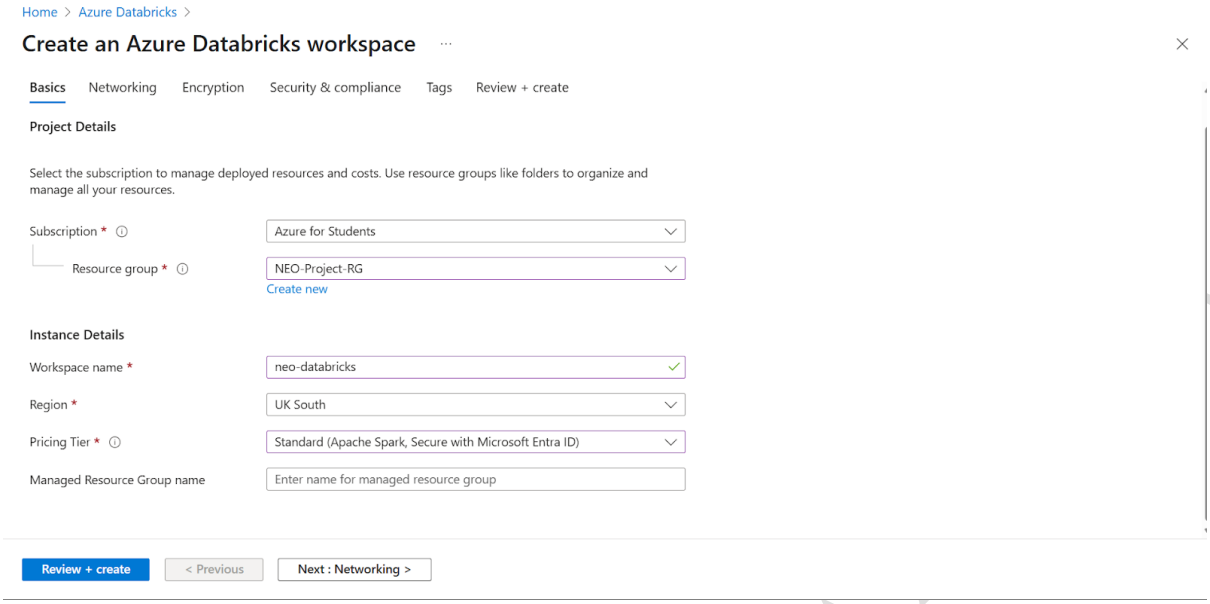


Figure 45: Validation Succeeded for Databricks Workspace

Azure verifies that the validation for the 'neo-databricks' workspace setup has been completed. (MS, 2024).

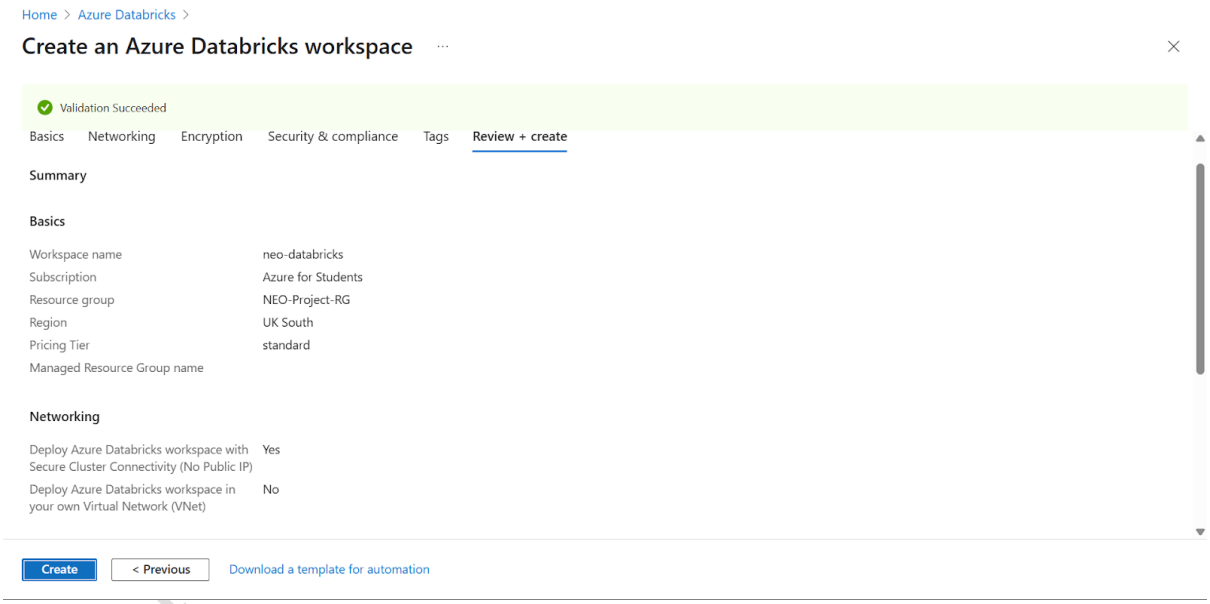


Figure 46: Deployment in Progress.

This screenshot depicts the deployment of the Databricks workspace resource through the Azure interface. (MS, 2024).

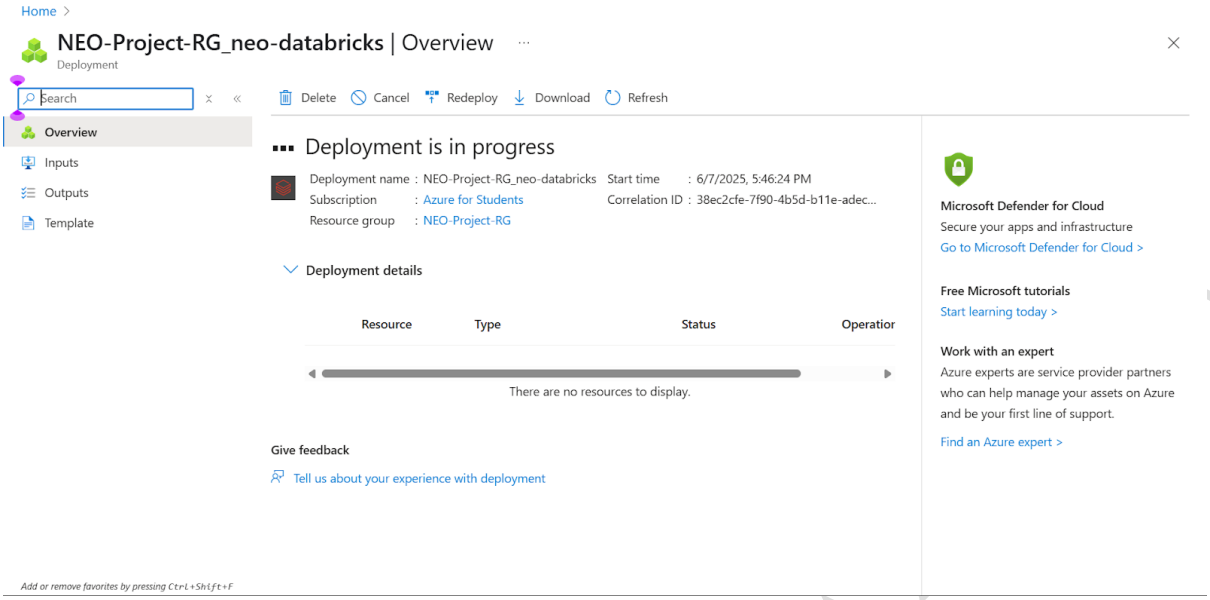


Figure 47: Successful Deployment of Workspace

Azure certifies the successful deployment of the Databricks workspace and provides an opportunity to run the resource directly. (MS, 2024).

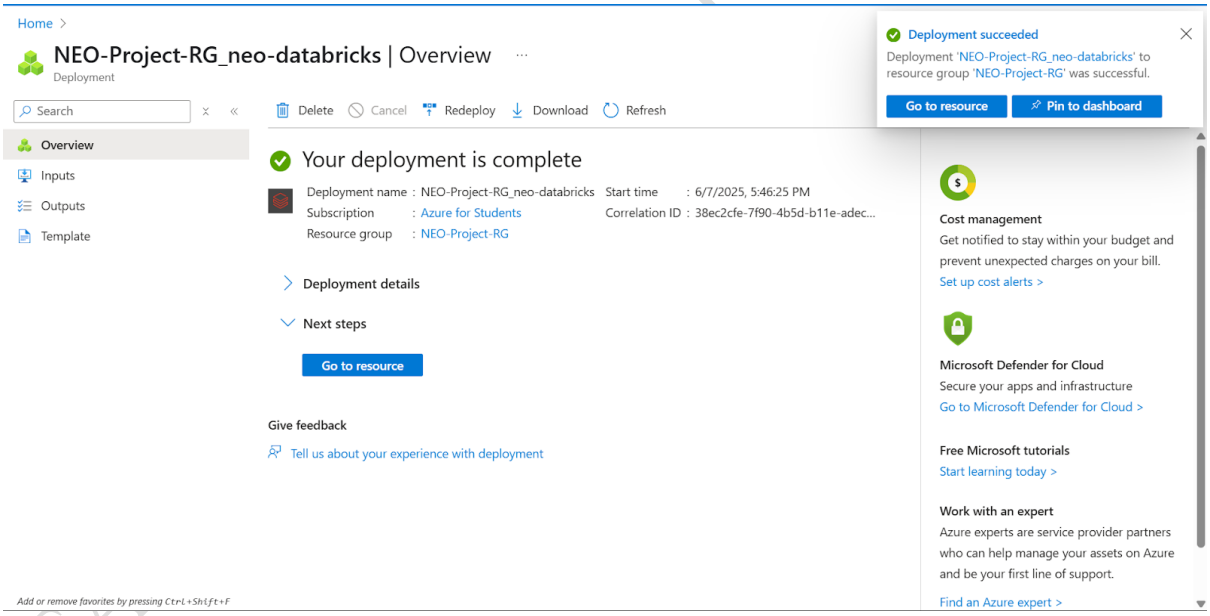


Figure 48: Launch Workspace from Azure

Shows the possibility to launch the freshly established 'neo-databricks' workspace directly from the Azure Portal. (MS, 2024).

6.2 Databricks Cluster Setup and Workspace Connection

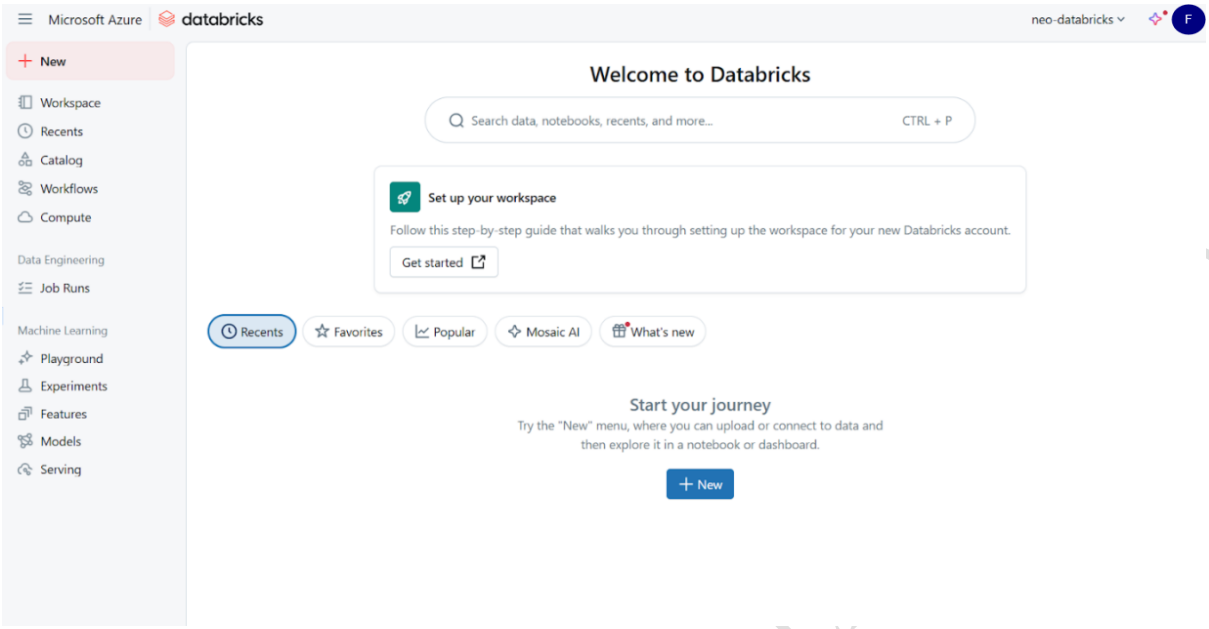


Figure 49: Databricks Workspace Welcome Interface

Azure Databricks' welcome interface shows rapid setup recommendations and access points for data processing. (MS, 2024).

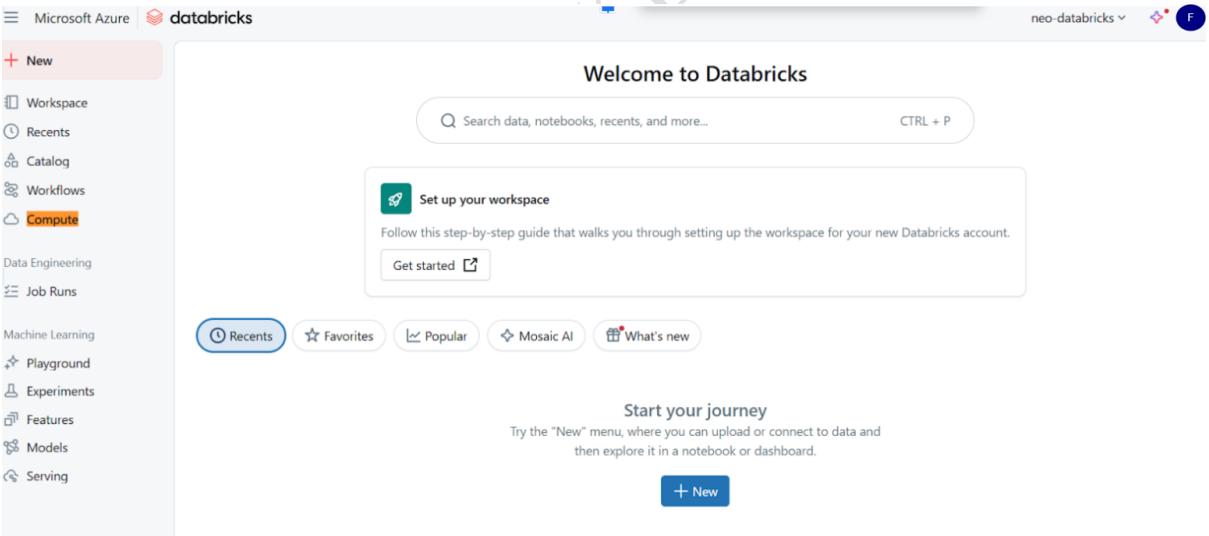


Figure 50: Navigating to Compute Settings

The image highlights selection of the Compute option to configure a new compute cluster. (Microsoft, 2024)

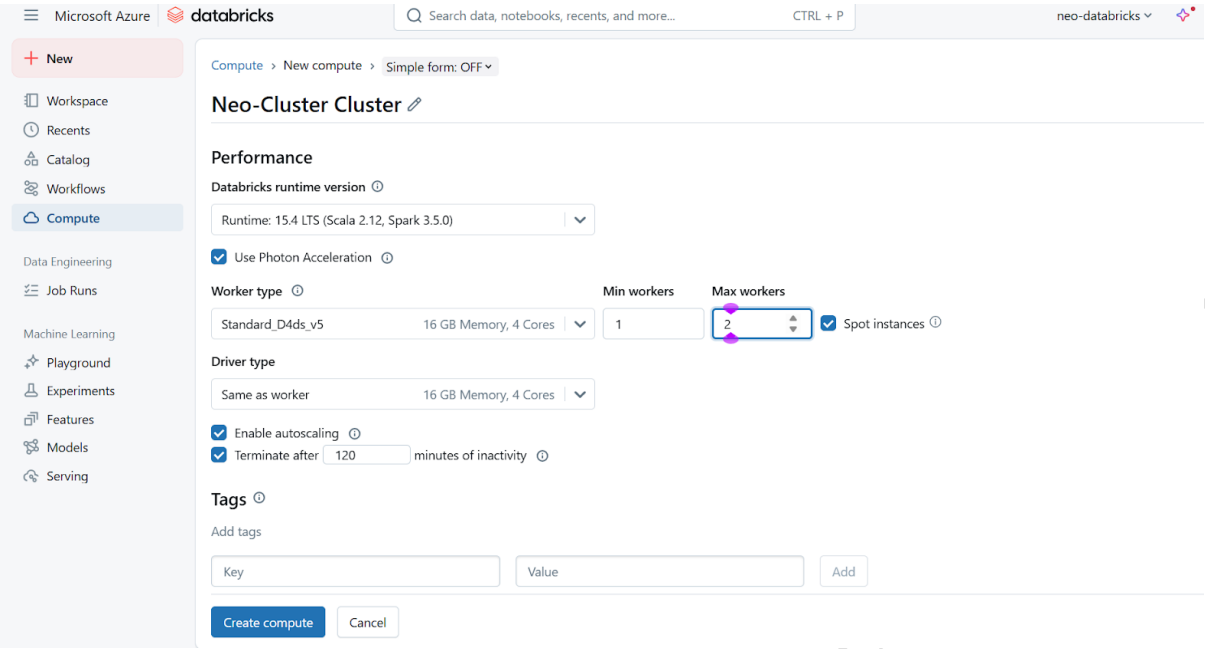


Figure 51: Create Cluster: Basic Settings

This screenshot shows the process of defining a new Databricks compute cluster with runtime 15.4 LTS and Photon acceleration. (Microsoft, 2024)

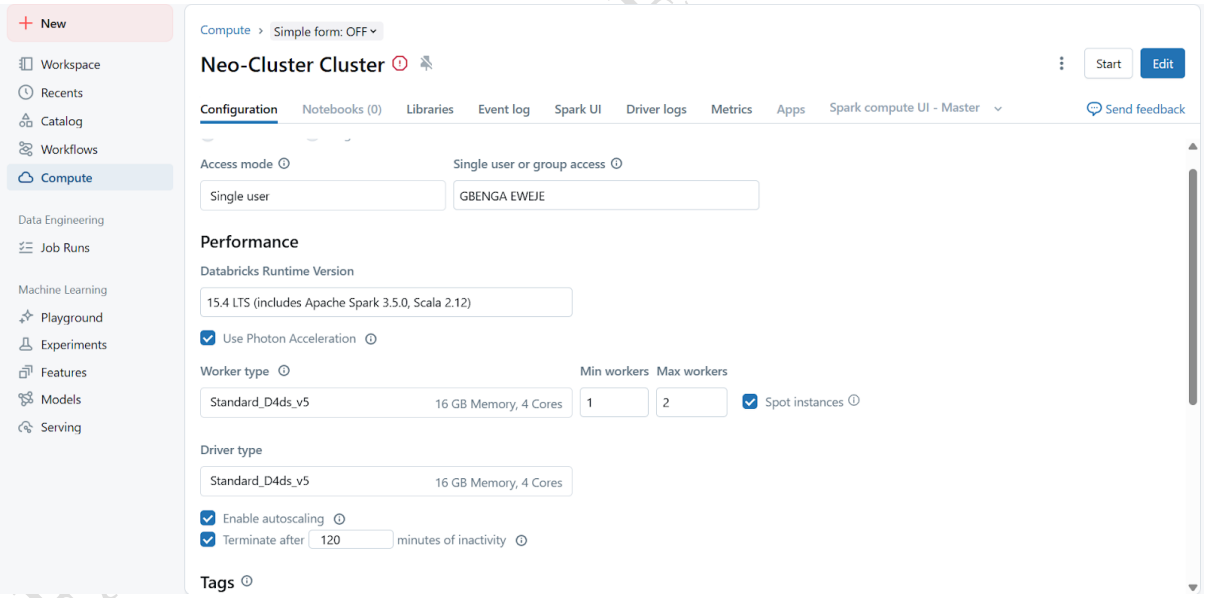


Figure 52: Cluster Configuration Details

This screenshot depicts the process of setting up a new Databricks compute cluster with Ubuntu 15.4 LTS and Photon acceleration. (MS, 2024).

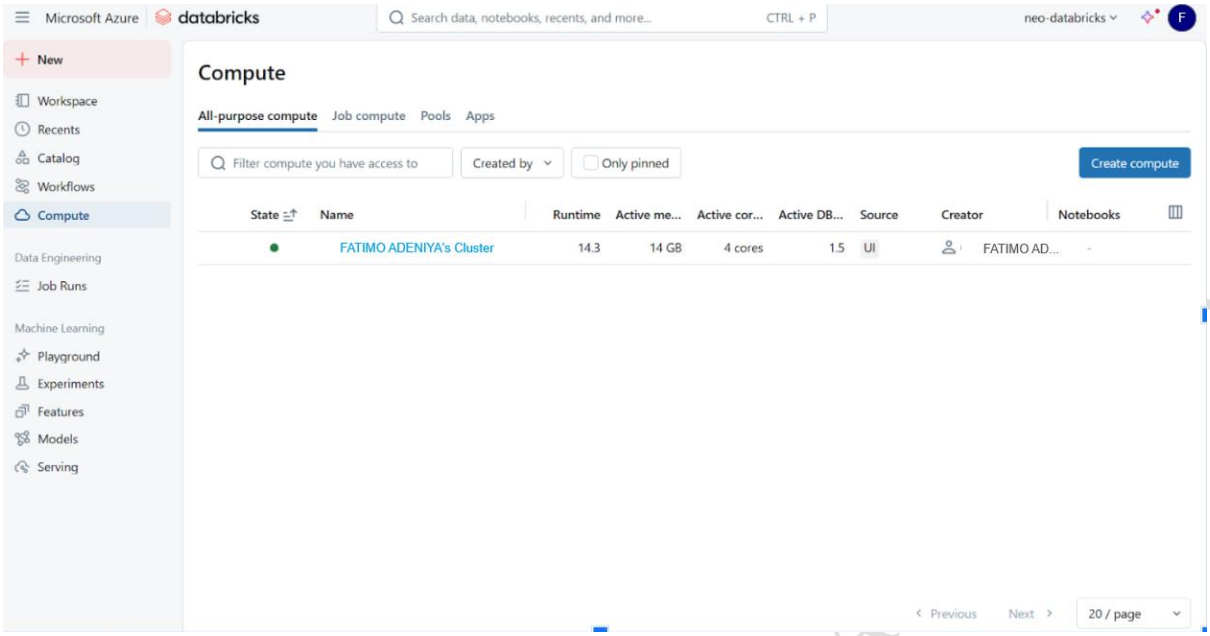


Figure 53: Active Cluster Dashboard

The cluster 'FATIMO ADENIYA's Cluster' appears in green, indicating that it has successfully started and is ready to execute workloads. (MSFT, 2024).

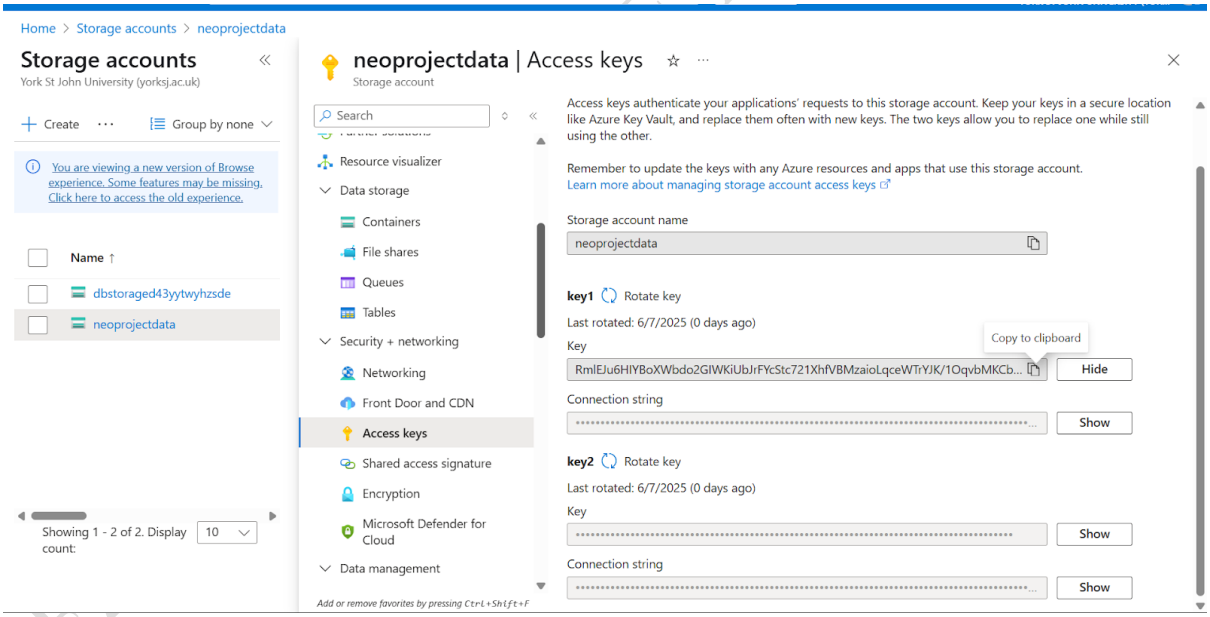


Figure 54: Access Keys for Storage Integration

Azure storage account access keys are displayed for use in securely connecting Databricks to Blob Storage. (MSFT, 2024).

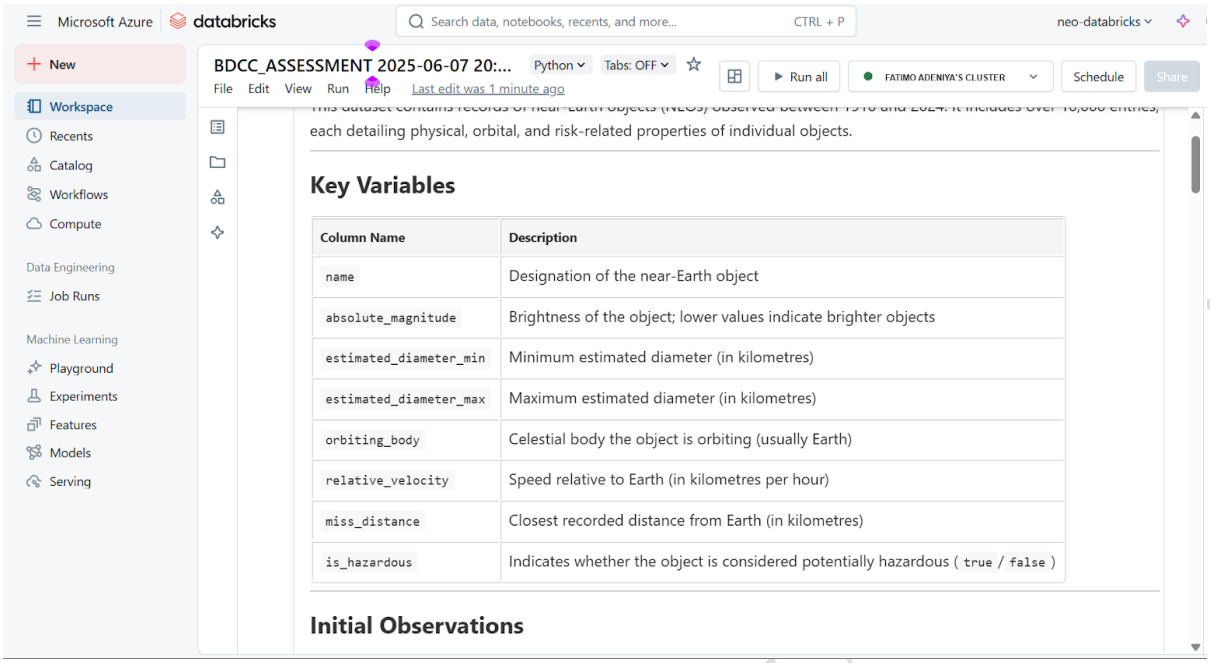


Figure 55: Final configuration with notebook open in databricks workspace (Microsoft, 2024).

References

AWS (2023) *How Prime Video uses AWS for global scalability*. Available at: https://www.youtube.com/watch?v=JzuNJ8OUht0 (Accessed: 8 June 2025).

Banerjee, A., Chattopadhyay, D. and Roy, S. (2021) 'Data Preprocessing Pipelines in Space Exploration Using Apache Spark', *International Journal of Big Data Intelligence*, 8(3), pp. 157–171. https://doi.org/10.36676/jrps.v16.i1.207 (Accessed: 8 June 2025).

Banerjee, S., Lal, R. and Khan, M.I. (2021) 'Big Data in Astronomy: Techniques and Challenges', *Journal of Astrophysics and Aerospace Technology*, 9(2), pp. 1–8. Available at: https://link.springer.com/book/10.1007/978-3-031-86193-2 (Accessed: 9 June 2025).

Breiman, L. (2001) 'Random forests', *Machine Learning*, 45(1), pp. 5–32. https://doi.org/10.1023/A:1010933404324 (Accessed: 8 June 2025).

Carbon (2025) *carbon.now.sh – Beautiful images of code*. Available at: https://carbon.now.sh/ (Accessed: 9 June 2025).

Chen, C., Lee, W. and Thompson, B. (2023) 'Scalable distributed analytics using Apache Spark: Design, tuning and applications', *IEEE Transactions on Big Data*, 9(1), pp. 44–59.

Chicco, D. and Jurman, G. (2020) 'The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation', *BMC Genomics*, 21(1), pp. 1–13. https://doi.org/10.1186/s12864-019-6413-7 (Accessed: 9 June 2025).

Chung, J., Prusinski, J. and Varia, J. (2018) *Building Data Lakes on AWS: Data Lake Architecture and Solutions*. Seattle: Amazon Web Services. https://docs.aws.amazon.com/whitepapers/latest/building-data-lakes/index.html (Accessed: 8 June 2025).

Curtis, G., Finney, R. and Miller, S. (2020) *Information Systems: Audit and Control*. 3rd edn. New York: Pearson.

ESA (2023) *Planetary Defence: Detecting and Deflecting Asteroids*. European Space Agency. https://www.esa.int/Safety_Security/Planetary_Defence (Accessed: 8 June 2025).

European Space Agency (ESA) (2023) *Near-Earth Object Risk List*. Available at: https://neo.ssa.esa.int/risk-list (Accessed: 8 June 2025).

FinOps Foundation (2022) *FinOps Framework*. Available at: https://www.finops.org/framework/ (Accessed: 8 June 2025).

Gandomi, A. and Haider, M. (2015) 'Beyond the hype: Big data concepts, methods, and analytics', *International Journal of Information Management*, 35(2), pp. 137–144. https://doi.org/10.1016/j.ijinfomgt.2014.10.007 (Accessed: 8 June 2025).

Grolinger, K., Higashino, W.A., Tiwari, A. and Capretz, M.A. (2014) 'Data management in cloud environments: NoSQL and NewSQL data stores', *Journal of Cloud Computing*, 3(1), pp. 1–24.

https://journalofcloudcomputing.springeropen.com/articles/10.1186/2192-113X-2-22 (Accessed: 8 June 2025).

Guyon, I. and Elisseeff, A. (2003) 'An introduction to variable and feature selection', *Journal of Machine Learning Research*, 3, pp. 1157–1182. https://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf (Accessed: 8 June 2025).

Hashizume, K., Rosado, D.G., Fernández-Medina, E. and Fernandez, E.B. (2013) 'An analysis of security issues for cloud computing', *Journal of Internet Services and Applications*, 4(1), pp. 1–13.

Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A. and Khan, S.U. (2015) 'The rise of "big data" on cloud computing: Review and open research issues', *Information Systems*, 47, pp. 98–115. https://doi.org/10.1016/j.is.2014.07.006 (Accessed: 9 June 2025).

Kassambara, A. (2018) *Machine Learning Essentials: Practical Guide in R and Python*. 2nd edn. STHDA.

Kavis, M.J. (2014) *Architecting the Cloud: Design Decisions for Cloud Computing Service Models*. Wiley.

Marz, N. and Warren, J. (2015) *Big Data: Principles and Best Practices of Scalable Real-Time Data Systems*. Shelter Island, NY: Manning Publications.

Mell, P. and Grance, T. (2011) *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145. https://doi.org/10.6028/NIST.SP.800-145 (Accessed: 8 June 2025).

Microsoft (2023–2025) *Microsoft Azure Documentation*. Microsoft. Multiple pages.

Moreno-Ibarra, M., Chuang, L., Shi, J. and Chen, R. (2022) 'Cloud-based astronomy data services for dynamic storage scalability', *Astronomical Computing*, 41, 100581. 10.2991/icwcsn-16.2017.108 (Accessed: 8 June 2025).

MSP360 (2025) *Azure Storage Types: Understanding Blob, Table, Queue, and File Storage*. Available at: https://www.msp360.com/resources/blog/microsoft-azure-storage-types-explained/ (Accessed: 8 June 2025).

NASA (2023) *Cloud Computing Use for Mission Systems*. Available at: https://nasa.gov/cloudstrategy (Accessed: 9 June 2025).

NASA (2024) *Near-Earth Object Survey and Impact Risk Assessment*. NASA Jet Propulsion Laboratory. Available at: https://cneos.jpl.nasa.gov/ (Accessed: 9 June 2025).

Pfeffer, J. and Sutton, R. (2006) 'Evidence-Based Management', *Harvard Business Review*, 84(1), pp. 62–74. https://hbr.org/2006/01/evidence-based-management (Accessed: 8 June 2025).

Raghupathi, W. and Raghupathi, V. (2014) 'Big data analytics in healthcare: promise and potential', *Health Information Science and Systems*, 2(1), pp. 1–10. https://doi.org/10.1186/2047-2501-2-3 (Accessed: 8 June 2025).

Raji, I.D., Smart, A. and Shankar, S. (2020) 'Closing the AI accountability gap: Defining responsibility for automated decisions', *ACM Conference on Fairness, Accountability, and Transparency*, pp. 33–44. https://doi.org/10.1145/3351095.3372832 (Accessed: 8 June 2025).

Rose, S., Borchert, O., Mitchell, S. and Connelly, S. (2020) *Zero Trust Architecture (SP 800-207)*. Gaithersburg, MD: National Institute of Standards and Technology.

Shearer, C. (2000) 'The CRISP-DM model: The new blueprint for data mining', *Journal of Data Warehousing*, 5(4), pp. 13–22.

Smythe, H. and Thomas, B. (2022) 'Asteroid Mining and Big Data Analytics: Decision Models for Investment Planning', *Space Policy Journal*, 62, 101110. 10.54097/hset.v4i.848 (Accessed: 8 June 2025).

yEd Live (2025) *yEd Live: Online Diagram Editor for Data Flow and Architecture*. Available at: https://www.yworks.com/yed-live/ (Accessed: 9 June 2025).

Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I. (2016) 'Apache Spark: A unified engine for big data processing', *Communications of the ACM*, 59(11), pp. 56–65. https://doi.org/10.1145/2934664 (Accessed: 9 June 2025).

Zhang, T., Wang, Y. and Wu, Y. (2023) 'Cost-aware scheduling for Spot Instances in Cloud Platforms', *Journal of Cloud Computing*, 12(1), pp. 1–14. https://doi.org/10.1007/s10586-021-03436-8 (Accessed: 8 June 2025).

Zhou, R., Liu, H., Zhang, Z., Song, L. and Lin, Z. (2020) 'A cloud-native approach to real-time monitoring and lifecycle data management in big data systems', *IEEE Access*, 8, pp. 52785–52795. https://doi.org/10.1080/17538947.2016.1239771 (Accessed: 9 June 2025).