MLOps IMPLEMENTATION ON AZURE FOR REAL-WORLD DATA SCIENCE

Use case (Insurance Industry)

by

Patricia Inyang

Student number : 220356622

A thesis for the degree of

Masters in Data Analytics

Submitted to

Department of Applied Mathematics and Data Science

College of Engineering and Physical Sciences

Supervised by

Brynjar Hartmannsson and Tim Johnson - Kainos Software LTD

Dr. Roberto Alamino - Aston University

15 January, 2024

Abstract

Many machine learning models face limited utilisation in real-world organisational systems due to the intricate processes and cross-functional discipline necessary for preparing a model for deployment in a production environment. This dissertation addresses this challenge by implementing Machine Learning Operations (MLOps) to successfully operationalise an end-to-end machine learning cycle within Azure Machine Learning (AML). The specific use case involved the development of a Vehicle Insurance model to predict responses ('Interested' or 'Not interested') from existing health insurance policyholders regarding an offered vehicle insurance policy. The model will be used to identify cross-selling opportunities in the insurance sector. The methodology progressed beyond the experimentation phases which encompassed the data preprocessing, Exploratory data analysis to model training and evaluation in a Jupyter notebook. A logistic regression model with a performance Recall of 85.9% was approved for initial deployment to production. The implementation utilized Version 2 of AML Python SDK and AML Command line interface(CLI) to build and test modular, readable and reproducible scripts, which were then configured as AML components and linked through pipelines. These pipelines covered model training to deployment of model to an AML managed Batch-endpoint, RAI insight dashboard setup, and batch inference job submission. The MLOps pipelines were automated using GitHub action workflows, facilitating continuous integration and continuous delivery for ongoing model improvements in the production environment. The use of Azure service principal credential enabled secure authentication between GitHub and the AML workspace, facilitating non-interactive execution of production environment workloads.

Despite the success of this project, the MLOps infrastructure currently operates at a 3^{rd} maturity level, falling short of a fully matured MLOps. This leaves opportunity for further work to implement continuous monitoring and continuous operation of the model in production, serving real-life data, and detecting issues like data and model drifts. These improvements would enable the system to trigger retraining and redeployment of the optimal-performing model when necessary.

Acknowledgement

I am deeply grateful to God Almighty for sustaining me throughout my MSc study year. A special thanks to my program director, Paul Griffiths, for collaborating with Kainos Data science Leader, John Calin to provide an opportunity to undertake this industry-based project. The success of this project and its profound impact on my career progression were greatly influenced by the collaborative guidance and mentorship of my industry supervisors, Brynjar Hartmannsson and Tim Johnson. Many thanks to them for their patience and invaluable support.

I extend my gratitude to my dissertation supervisor, Roberto Alamino, for his thoughtful support and encouragement. I also appreciate the College of Engineering and Physical Sciences and its dedicated staff for their considerate guidance.

A heartfelt thank you to all my family members and friends who believed in me and supported me on this journey. I want to express particular appreciation to my loving husband and son, who sacrificed our time together to allow me to focus on my studies. Their unwavering love has been invaluable.

Contents

List of Figures

List of Tables

List of Key Abbreviations

This list describes abbreviations that are repeatedly used in the body of the dissertation.

AML	Azure Machine Learning
CD	Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
DevOps	Development Operations
Insurtech	Insurance Technology
ML	Machine Learning
MLOps	Machine Learning Operations
RAI	Responsible Artificial Intelligence
SDK	Software Development Kit

1 INTRODUCTION

In this era of data-driven innovations, the exponential growth in big data is expected to lead to a substantial increase in economic activities attributed to artificial Intelligence (AI), until 2030 [1, 2]. Machine Learning (ML) which encompasses the majority of advancements in artificial Intelligence (AI) [3], is the process of using statistical algorithms to analyze data and identify inherent patterns, with the aim of developing self-learning models that are capable of predicting outcomes and classifying information without human intervention. These ML models developed by Data scientists and Machine learning engineers provide useful insights for businesses across various industries to make informed decisions that make them more efficient and sustainable [4].

Despite the notable advancements in machine learning models, research shows a substantial gap between their development and successful deployment for real-life applications. Even when these models manage to transition to live production and deployment, a significant number ultimately encounter failure [5]. Hence Machine learning Operations simply known as MLOps provides an effective systematic approach to close this gap. MLOps is defined as the culmination of people, processes, practices, and underpinning technologies that automate the development, deployment, monitoring, and management of machine learning (ML) models into production in a scalable and fully governed way to finally provide measurable business value from machine learning [6]. It involves the application of software Development and Operations (DevOps) principles to machine learning with a focus on streamlining the end-to-end process of taking machine learning models from development to deployment to production while maintaining and monitoring them. These DevOps principles include [7]:

- **Collaboration:** among the team of data scientists, DevOps engineers, and IT.
- **Automation:** of all the steps in the Machine learning life cycle, using workflows and pipelines to integrate the ML system and continuously operate it in production.
- **Continuous Integration (CI):** Involves frequently integrating code changes into a shared repository, using git for versioning, keeping code up-to-date as well as detecting and addressing issues early.
- **Continuous Deployment(CD):** Extends continuous integration to automatically deploy code changes to production environments.

To implement these automation in MLOps, cloud platforms like Microsoft Azure and AWS are instrumental in providing a robust infrastructure that supports the seamless implementation of machine learning workflows. Cloud computing which refers to the on-demand delivery of scalable IT resources over the Internet, eliminates the need for individuals and businesses to self-manage physical resources themselves, and only pay for what they use. [8] Among the cloud platforms, Microsoft Azure stands out as a comprehensive and powerful ecosystem for MLOps through services like Azure machine Learning (AML) and Azure DevOps. Azure ML streamlines MLOps with its end-to-end solution, designed for seamless integration with diverse developer tools and platforms. This compatibility is crucial for deploying models effectively in business operations and also ensure scalability, reliability, and efficient maintenance throughout the entire machine learning life-cycle.

The insurance industry use case for this project involves developing a ML classification model to predict the responses of existing health insurance policyholders (customers) to a newly offered vehicle insurance service. However, the focus of the project is to successfully implement MLOps to operationalise the model in a real-life production environment. The subsections of this Introductory chapter will delve deeper into the steps in MLOps cycle, explain the problem that this project aims to address and state the objective of this research work.

1.1 Overview of MLOps cycle

The MLOps cycle shows the workflow that connects the processes in each stage of a machine learning life-cycle. MLOps is a continuous automated end-to-end process from data ingestion, through model development and deployment to model monitoring and retraining. At the outset of any standard project lies a crucial planning phase where the business problem or opportunity that the machine learning model aims to address is defined [9]. Planning also includes data collection, model selection, and establishing success criteria. Each step in the MLOps cycle in figure 1, is explained below:

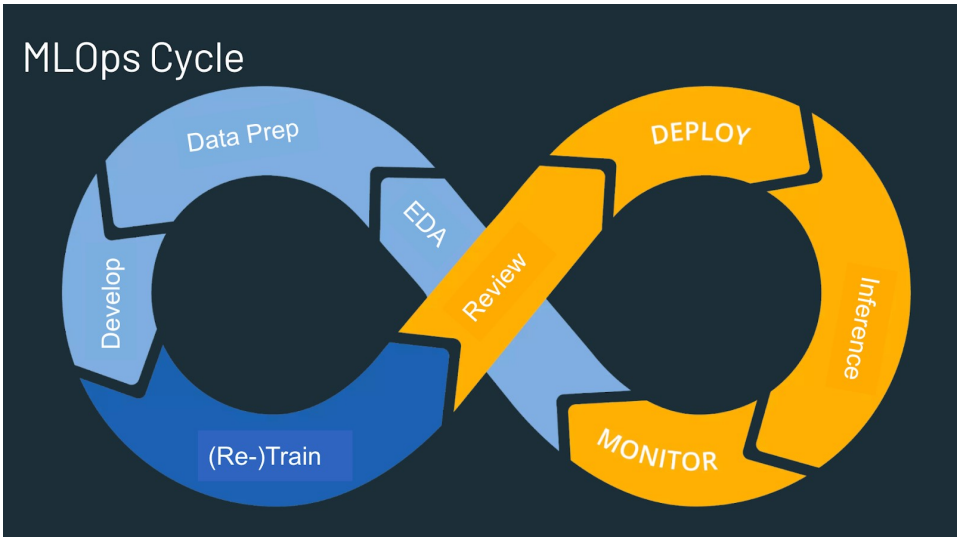


Figure 1: Machine learning Operations life cycle [10]

1. **Exploratory data Analysis(EDA):** is a preliminary process in developing a ML model, typically carried out by data scientists. It involves the use of visualisations and correlations tools to analyse the raw data, aiming to unveil obvious patterns and relationships among features. EDA not only offers insights into the data's inherent structure but also helps confirm whether stakeholders are asking the right questions. EDA serves as a dynamic and interactive phase that contributes to a comprehensive understanding of the dataset, facilitating informed decisions throughout subsequent stages in the machine learning journey. [5]

2. **Data Preparation**: is a pivotal process that influences the final result of the machine learning model. Most times, the data provided by companies and organisations is usually large, contains missing information, and is provided in formats that are unsuitable for learning algorithms. Hence, data preparation involves the data cleaning, imputation, normalisation/standardisation, feature selection and engineering that collectively transforms the raw data into a suitable input for effective model training. [5]

3. **Model Development**: involves selecting an appropriate machine learning algorithm (between supervised or unsupervised learning depending on the data), designing the model architecture and using pipelines to combine the data preprocessing steps with the model training. During model development, the data scientist also defines the strategy by experimenting with various models and parameters. The outcome of the model development is a well-structured code artifact with interconnected components that is ready to execute model training. [5]

4. **(Re-)Train:** During this step in the MLOps cycle, the selected machine learning model undergoes an iterative process during which the algorithm learns the relationship between the features in the data. The transformed

training data from the data preparation step is employed to train the model built earlier. As a result of this training process, a model with optimized parameters is obtained. A crucial part of MLOps practices is retraining an existing machine learning model to adapt to new data or changes in the underlying patterns of the data. Retraining ensures that the model in production is maintained, up-to-date and continues to make accurate predictions over time. [11]

5. **Review:** The trained machine learning model undergoes evaluation on a testing/validation data to review its prediction performance, effectiveness and suitability for the business problem. Depending on the type of model, for a classification or regression problem, metrics like accuracy score, recall, precision, F1 score, AUC-ROC curve, Mean squared error(MSE), mean absolute error (MAE), and R-squared are calculated to know the model's performance. The review step encompasses not only quantitative assessments but also qualitative aspects such as code reviews and packaging of the code using reproducible, readable and modular steps, connected using pipelines in line with best practices. Additionally, interpretability of the model's predictions is assessed in order to gain insights into feature importance and the factors influencing the model's predictions.

   Based on the review outcome and feedback from the stakeholders, an informed decision is made on whether to proceed with model deployment, refine the model further, or consider loop back to earlier stages in the MLOps cycle to make improvements [12].

6. **Model Deployment:** A model that is considered good enough solution for the business is saved to a pickle file and versioned ready for deployment to the production environment. "Model deployment is the process of moving a model from an offline development environment to an online system or a large, powerful, and secure database or server where it can be used to make real life predictions on new data." [12]. The true value of the model is gotten from its performance in the production environment which may have been written in different languages than the model development environment. Hence the need to have a well structured workflow for the model deployment process.

   Firstly, the ML model and its dependencies are packaged into a container (e.g., a Docker container). Containerization ensures that the model, its dependencies, and the runtime environment are encapsulated together, facilitating portability and reproducibility. [1] The next step is to set up the production infrastructure, including the endpoint (real-time or batch endpoint), and configure it to access resources such as compute power and storage. The deployment endpoint is the API (Application Programming Interface) where external systems or applications can send requests to obtain predictions from the model.

   For model to be deployed, the code for an inference pipeline is written to automate the processes involved in making predictions or inferences using the trained machine learning model. Consequently, before finally deploying the model to production, a testing/staging phase is done, where the inference pipeline is made to mimic the production environment as closely as possible to ensure that its works as expected.

7. **Inference:** In the MLOps cycle, the inference step, also known as score serving, involves utilizing the deployed model to generate predictions for new, unseen data. In this phase, the model which is now live, actively receives real-world data through the inference pipeline which is designed to take input data, optionally transform it, and provide predictions as output to the users. Figure 2 below illustrates the function of the inference pipeline from ingesting input data from either a batch (S3) or real-time requests (DB1 or DB2). The process involves validating the data, transforming it into a format suitable for the ML model, and then making predictions. It is

important that the transformation code used is similar to the preprocessing code used to prepare the training data.

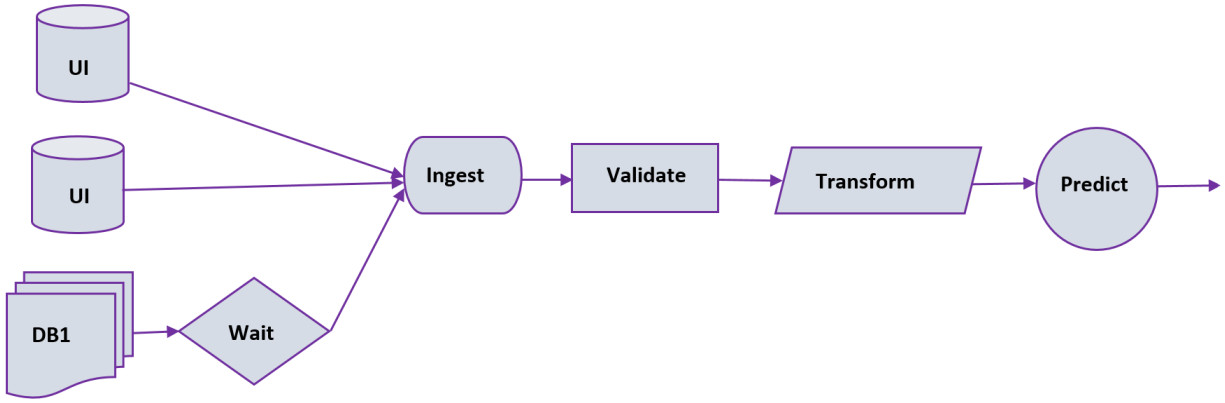


Figure 2: Steps in the Inference pipeline for a Model in production.

8. **Monitor**: A model in production requires continuous monitoring of the model's performance accuracy, to detect any drift in model behavior and identify potential issues that might arise as the model interacts with real-world data. An example of such issues can be the occurrence of model drift and data drift which tends to reduce production performance over time.

   Monitoring also involves keeping track of the workload processed by the model and resources consumed in the production operations. Information about compute power, memory usage, and storage requirements helps to maintain optimal performance and identify potential scalability needs.

The outcome of monitoring will help the team decide the trigger for retraining a new model for continuous improvement. Hence the MLOps cycle is a continuous process that requires collaboration between the machine learning data scientist, DevOps team and IT operations team.

1.2 Problem statement

In the field of data science and machine learning (ML), the development of strong algorithms and high-performing models are widespread. However, the seamless integration of these models into real-world operational environments, a crucial step for realizing their business value, poses a significant challenge. Albeit the potential benefits of MLOps are recognized to eliminate this gap between model development and deployment, its implementation introduces intricacies. Traditionally, the manual handover of models from data scientists to software development teams or ML engineers results in an inefficient process due to disconnected machine learning life cycle processes. Data scientists, trained to experiment in interactive Jupyter notebooks, often face challenges in aligning their code structures with the modular, repeatable, and scalable formats preferred by ML engineers for model operationalisation [13]. Lack of collaboration between the team members makes it time-consuming and costly to deploy a model to production because the software development team may have to embark on the task of refactoring the code.

Moreover, the transition from model development to production demands additional infrastructure and expertise. Tasks such as building CI/CD pipelines, establishing endpoints for the production environment, containerizing the environment, and implementing version control and tracking systems become essential. Managing production environments using on-prem servers poses problem of managing cost and compute resources especially when scaling becomes

necessary. Also, an essential requirement of any AI solution is the need for it to meet set governance considerations such as clear accountability, keeping logs for audit trail, and transparency throughout the model's lifecycle.

Even after successful deployment, sustaining optimal model performance becomes a concern [14]. Data drift, characterized by gradual changes in the distribution of real-world production data over time, challenges the assumption of static underlying data. External factors, seasonal variations, and shifts in user behavior contribute to evolving input data distributions. Simultaneously, model drift may occur when the relationship between input features and the target variable shifts, even if the data distribution remains constant. This makes it pertinent to achieve a fully matured MLOps cycle which includes the continuous monitoring, retraining, and maintenance of the model in production, in order to prevent the problem of model degradation.

Addressing these challenges requires a comprehensive approach to MLOps implementation, considering the unique interplay between data science and software engineering practices. By leveraging the capabilities of the cloud solution, specifically Azure Machine Learning (Azure ML), as a primary platform for MLOps practices and focusing on real-world use cases within the insurance industry, this project endeavors to surmount the identified obstacles. The ultimate goal is to deliver a fully operational model in the production environment, capable of offering tangible business value that can empower insurance companies with insightful decision-making capabilities.

1.3 Project Objectives and Research Statements

The overall objective of this project is to develop an end-to-end fully matured Machine Learning operations in azure. The specific research statements to be achieved are listed enumerated below:

1. Successfully develop a machine learning model to predict Insurance customers' interests in vehicle insurance.
2. Build and automate all the steps in a machine learning life cycle.
3. Deploy the model to production in Microsoft Azure.
4. Develop a CI/CD pipeline for continuous improvement of the MLOps system.

1.4 Significance of the Project

Similar to the financial sector, the insurance industry is undergoing digital transformations in addressing challenges such as fraud detection, risk assessment, customer satisfaction, claims management, and predictive analysis. This dissertation makes a substantive contribution by proposing and implementing an MLOps framework designed to operationalise machine learning models specifically for insurance use cases. The particular use case develops a cross-selling model capable of predicting which existing health insurance customers would be interested in a vehicle insurance policy. With such a robust model in production, insurance companies can make insightful decisions and continually identify customers to channel new services to while saving costs on acquiring new customers.

The project's significance lies in its automation of the machine learning life cycle within a data science project. The effective implementation of MLOps holds the potential to directly impact business outcomes in the insurance industry, offering opportunities to reduce costs, enable continuous improvements, and gain a competitive advantage in the market.

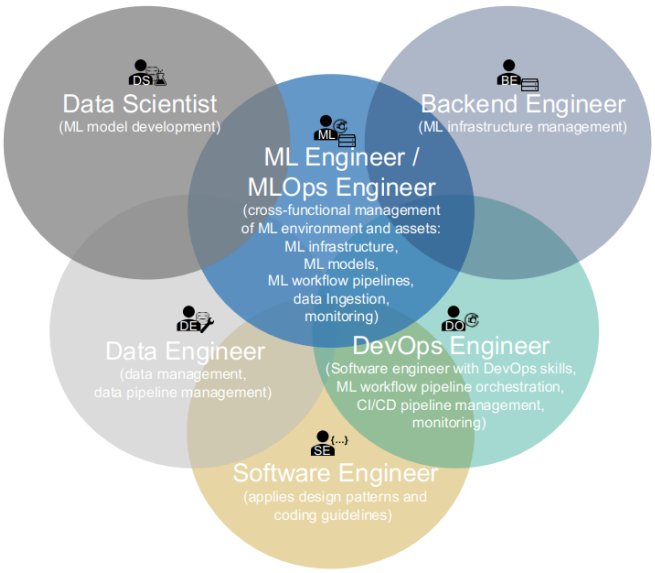
Moreover, the interdisciplinary nature of the topic fosters cross-disciplinary learning, bringing together expertise from data science, cloud computing, and operational DevOps practices. This prepares MSc students to navigate the intricate landscape of modern technology holistically.

2 BACKGROUND THEORY AND LITERATURE REVIEW

2.1 Origin and evolution of MLOps

MLOps is a relatively new practice which was first coined in 2015, in a research paper [15] which analysed the 'technical debt' being incurred by the existing ML practices. The paper highlighted the challenges associated with maintaining machine learning (ML) systems over time, for business applications. Drawing parallels with traditional software engineering, the paper identifies ML-specific limiting factors, such as not using modular design to create maintainable code which makes it is easy to make isolated changes and improvements. Amongst other factors mentioned, is ML model dependence on data that qualitatively or quantitatively changes behavior over time. As ML systems demand considerable resources and engagement from diverse teams, challenges emerged from a lack of collaboration and misinterpretations between data scientists and IT teams regarding the most effective development processes. The paper proposed the establishment of a structured "MLOps" process that utilizes DevOps Continuous Integration/Continuous Deployment (CI/CD) methodology to enhance integration of future improvements, reduce errors, and improve maintainability.

In an effort to provide a clear structure for MLOps principles and best practices, Dominik et al [16] conducted a mixed-method research involving a systematic literature review, a tool review, and expert interviews of ML professionals with profound MLOps knowledge on a global level. The result of this research provided a detailed definition and architecture for MLOps which aligns with the principles and best practices applied by the top industry leaders [17–20]. While team structures may differ across organizations, Figure 3 below, presents a comprehensive visualization illustrating the cross-functional collaboration among professionals in MLOps."



(a) Roles and their intersections in MLOps paradigm

P1	CI/CD automation
P2	Workflow orchestration
P3	Reproducibility
P4	Versioning of data, code, model
P5	Collaboration
P6	Continuous ML training & evaluation
P7	ML metadata tracking
P8	Continuous monitoring
P9	Feedback loops

(b) MLOPS Principles and best practices

Figure 3: Professionals and best practices of MLOps (Source [16])

With the growing awareness of the need for MLOps within the ML community, several open source tools were being built to facilitate the implementation of MLOps. A comparative analysis of these tools like Kubeflow, Mlflow, Datarobot, was conducted by Hewage et al to assess their performance and use cases [21]. The examination extends to the features and usability of MLOps tools, aiming to pinpoint the most suitable tool support for specific scenarios [22]. However, the research identifies a deficiency in the availability of a comprehensive MLOps platform capable of automating processes and minimizing human intervention. These necessitates the need for continuous exploration of

best practices in implementing MLOps especially with the more sophisticated features continuously provided by cloud services like Azure Machine Learning version 2. This project aims to overcome the challenges pointed out N. Hewage et al by using Azure to achieve a fully mature MLOps model.

2.2 Benefits of MLOps to Data Science Projects

The remarkable success achieved by applying DevOps principles to enable continuous software development [23, 24] has sparked a growing interest in extending these practices to facilitate the real-time deployment of machine learning components. As the foundational step in developing an ML model involves data, projects in this domain are commonly referred to as data science projects [25]. Organizations providing machine learning solutions [26] are increasingly embracing the MLOps approach because it offers a host of significant benefits, some of which are outlined below.

1. **Automation:** MLOps automates the end-to-end machine learning lifecycle. This automation accelerates the development cycle, reduces manual errors, and promotes a more agile and responsive development process.

2. **Improved Efficiency and productivity**: of the team and process since MLOps enables the building of reusable and reproducible pipelines that can be used across different projects. This enhances overall project efficiency

3. **Collaboration:** and communication among data scientists, engineers, and other stakeholders make the team more efficient in achieving the goals of the project.

4. **Scalability and Resource Optimization:** With MLOps, models can be deployed and scaled seamlessly as the workloads change and resource usage is optimised.

5. **Mitigate risk**: of the model failing in production since MLOps practices provide a structured approach to validate Model and testing of code/pipelines before deployment. Also, the robust continuous monitoring and maintenance that MLOps offers help the team to handle occurrences like data and model drift throughout the lifetime of the model.

6. **Continuous Integration and Continuous Deployment (CI/CD):** MLOps embraces CI/CD principles, enabling continuous integration of new features and improvements and the automated deployment of models into production. Resulting in faster delivery cycles and quicker response to changing requirements.

2.3 MLOps Maturity Levels

Operationalising a machine learning model using the MLOps cycle introduced earlier is a very challenging process [27] to implement and automate. Hence, organisations gradually evolve in their MLOps maturity level. MLOps maturity levels refer to the stages of development and implementation of MLOps practices within an organization. Maturity levels provide a way to assess how well an organization is integrating and managing machine learning operations across its processes. These levels typically evolve from manual isolated to fully optimised and automated processes. According to the top industry leaders [17, 18, 28, 29] and research conducted across companies who practice MLOps [30–33], the attributes observed at each of the MLOps maturity levels is presented in the table below:

Table 1: Progression and Characteristic of MLOps maturity levels.

Maturity levels	Personnels and roles collaboration	Characteristics of scope of work	Level of automation
Level 0	• Driven by Data scientist working in isolation, handing models to the deployment team. • ML engineers are siloed. • Software engineers work remotely.	• No MLOps, No CI/CD. • No collaboration within the team. • Involves experimentation and use of notebooks. • No tracking of logs and metrics. • Infrequent retraining or release of models.	• Manual steps for data and model development. • Each process in the machine learning life cycle is manually executed. • Automated data collection is present for organizations with processes for aggregating data.
Level 1	• Data scientist, ML engineers, and Software engineers not in regular communication. • Software engineers work remotely. Models are still received as artifacts.	• Continuous training of the model • Faster and more efficient training and retraining. • Difficult to trace/reproduce results. • No tracking, no keeping of logs and no active monitoring.	• The ML and model development steps are automated in a ML pipeline. • Manual model releases. • Manual deployments.
Level 2	• Data scientists work with data engineers to convert experimentation code into reproducible scripts. • Software engineers work remotely and receive models remotely.	• code and resulting models are version controlled. • Implementation of centralized tracking of ML metadata. • Reproducible training environment and models.	• Automated training. • Manual model releases. • Manual deployments.
Level 3	• Team communicate frequently. • Software engineers collaborate with Data engineers to automate model integration into production.	• code and resulting models are version controlled. • Full traceability from deployment back to original data. • Training pipeline is connected to deployment pipeline.	• Automated model deployment. • Releases are automated by continuous delivery (CI/CD) pipelines.
Level 4	• Collaboration among Data scientists, Data engineers, and Software engineers. • Software engineers implement post-deployment monitoring and share metrics gathered for continuous improvements.	• Retraining triggered automatically based on production metrics. • Production performance and resources (compute, storage, etc.) usage are monitored. • Full version control.	• Full MLOps automation, including monitoring, triggering retraining, and auto-scaling in the production environment.

2.4 Literature review on Azure cloud services for MLOps

2.4.1 Fundamentals of Cloud Computing

Organizations are increasingly considering migrating on-premises data centers to the cloud, leveraging extensive and robust on-demand services from major cloud providers [34]. This shift to cloud computing offers a contemporary alternative to traditional methods, delivering computing infrastructures like virtual machines, software, storage, databases, and networking over the internet to organizations. The three categories of cloud computing services below define the shared responsibilities between cloud providers and customers based on specific needs [35]:

1. **Infrastructure as a Service (IaaS)**: Provides virtual computing resources while users manage operating systems, middleware, applications, and data.
2. **Platform as a Service (PaaS**): Offers a platform, tools for application development, and handling of infrastructure management while users manage applications, data and focus on coding.
3. **Software as a Service (SaaS):** Delivers software applications over the internet as a subscription; maintenance, updates, and security are handled by the service provider.

The user will always be responsible for the information and data stored as well as accounts of the people and devices allowed to connect to the cloud. The cloud service provider will always be responsible for the physical data center, network and the physical hosts. Although internet computing has advantages such as easing scalability, providing centralized computation and storage, utilizing high-performance machines and efficient management, [34] raised concerns about privacy and security being important considerations for the future of cloud computing.

2.4.2 Azure Machine Learning service

Azure Cloud Services is a cloud-based platform provided by Microsoft that offers an extensive range of services from web-based no-code solutions to developer tools. To implement MLOps, the Azure Machine Learning Services (AML) serves as a PaaS for training, deploying, automating, managing, and monitoring machine learning models. It supports automated model training and tuning through the AML AutoML designer which is a no-code user interface. Azure also has a comprehensive documentation on how professionals can utilize the AML Python software development kit (SDK) and the Azure command line interface (CLI) to carry out all steps of the MLOps. Additionally, Azure DevOps streamlines pipeline automation for handling machine learning model development, packaging, model validation, and deployment. Figure 4 below shows the end-to-end MLOps framework in AML with a sequence of the workflow and the infrastructure options offered within Azure.

The research conducted by [36, 37] emphasized the growing significance of incorporating responsible AI (RAI) into the MLOps process, marking it as a trending topic in the ongoing development and future trajectory of AI. The study proposed that achieving sustainable MLOps requires careful consideration of fairness, explainability, accountability, and sustainability throughout every stage of the MLOps cycle. Moreover, Azure ML governance is strategically designed to mitigate the potential risks associated with ML solutions across their lifecycle [38]. It achieves this by incorporating provisions for privacy, access control, auditability, logging, RAI dashboard setup, monitoring, and a resilient infrastructure capable of recovering from failures. AML provides a range of managed tools, including managed online and batch endpoints for serving models, and supports 'Bring Your Own Device' (BYOD). This means that developers can harness Kubernetes technology in AML for container orchestration, and seamlessly integrate tracking and logging tools from other open-source platforms such as MLFlow.

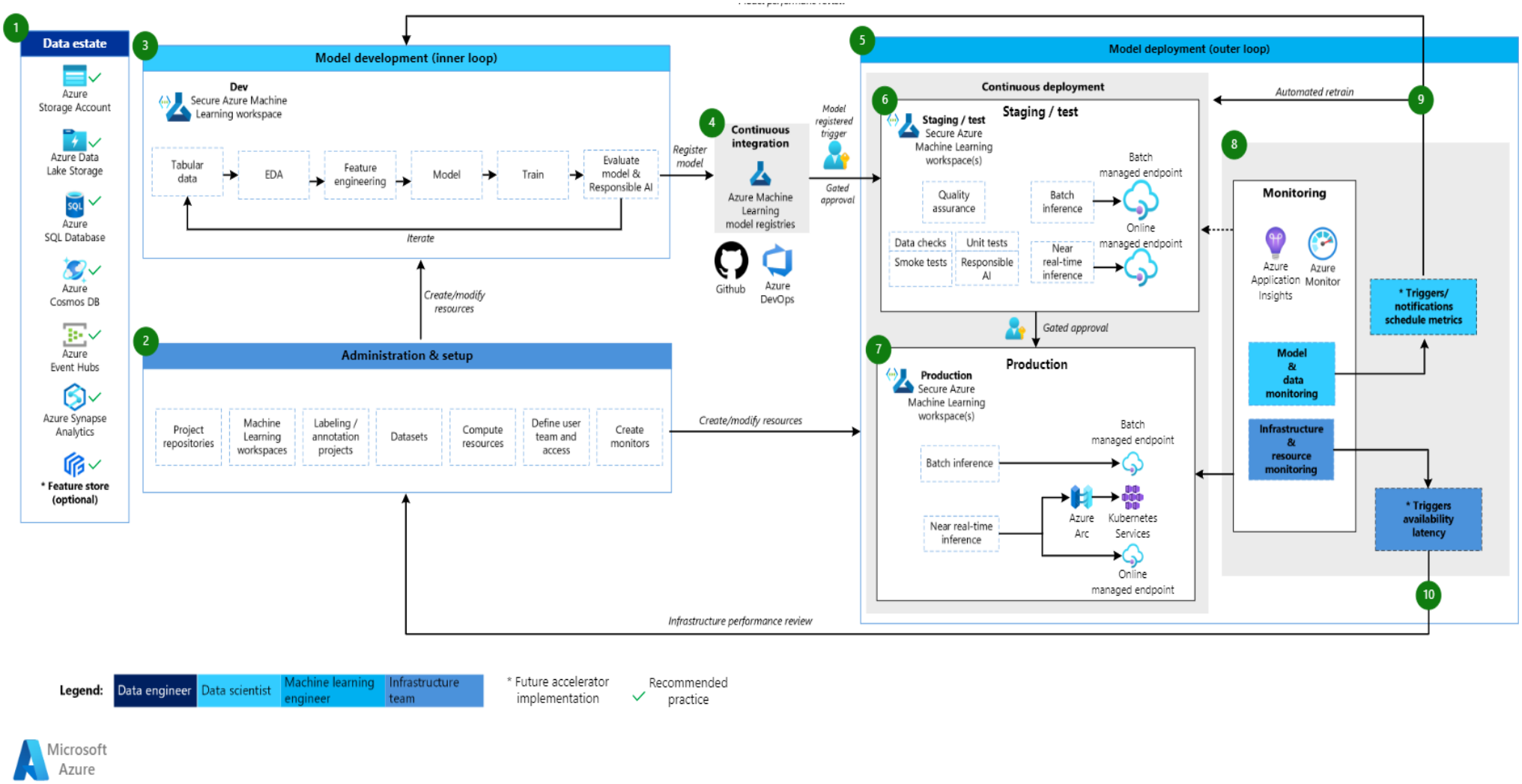


Figure 4: Azure MLOps structure [39]

2.5 Review of Research involving MLOps in the insurance industry

In recent years, the insurance sector, like many other industries, has witnessed a significant surge in the adoption of machine learning innovations to enhance service delivery. As contained in the 2023 AI Readiness Report, companies within the insurance industry lead the list of sectors planning substantial increases in their AI budgets over the next three years. Insurers are actively seeking ways to optimize their risk assessment, policy underwriting, and claims processing by leveraging machine learning and AI technologies. Their overarching objective is to achieve cost reduction, enhance customer experiences, and improve overall operational efficiency [40]. Realizing the maximum return on investments in AI goes beyond a narrow focus on ML technologies; it requires embracing change in process architecture and team structures to enable frameworks like MLOps.

Sriram et al [41] leveraged ML to develop a holistic AI System for data preparation on insurance policy listings, named ECLIPSE (Elegant Cleaning and Labeling of Insurance Policies while Standardizing Entities). The work aimed to reduce the time often spent by actuaries in the insurance industry, on data preparation and cleaning to regain more time for modeling. This often dirty data generated within the insurance industry stems from the fact that insurers are still lacking a standard industry data schema. Hence policies being underwritten by insurers have an ever-changing pattern. ECLIPSE design used a labeling framework and MLOps user interface (UI) to balance the trade-off between full automation of the ML system and the need for a human-in-the-loop. The UI enabled users to review and correct schema suggestions through a drop-down menu which saves user-confirmed labels to the training dataset for periodic retraining and continuous improvement of model performance. The result of applying ECLIPSE using MLflow MLOps tool at Guy Carpenter was a 50% reduction in time spent to build ML models for insurers.

A lot of research involving the development of machine learning models to solve problems in the insurance industry has been conducted [42–45]. By utilizing exploratory data analysis (EDA), visualisation and ML algorithms to identify meaningful and decisive factors for claim filing and acceptance, [46] was able to prove that "Insurtech" companies will contribute to making the claims management process more efficient. Despite the high-performing models trained in this paper, the author pointed out the need for future works to consider methods for handling imbalanced data which is quite prevalent in real-life insurance data. Professionals in Insurtech jointly agree that MLOps is still in its early phase of adoption in the insurance industry [47, 48]. Hence it is not surprising that there are rarely previous research papers that implement MLOps in the insurance sector up till model deployment in production. From the online search conducted, the only documented attempt to implement the MLOps strategy for an insurance use case is the prediction of insurance premium carried out by Silva et al [49]. Even though the model had a RMSE of 76.93, a fully matured MLOps model was not achieved as only the continuous integration pipeline was automated without reaching continuous deployment.

2.6 Impact of Machine Learning in Cross-Selling

The effectiveness of employing a data-centered approach for predictive analysis to identify cross-selling opportunities is substantial. Cross-selling is a sales strategy where a business promotes additional products or services to existing customers [50]. The customer plays a crucial role in the insurance sector, given that insurance companies do not produce tangible goods or services. The frequency of customer-company interactions increases with additional purchases, hence providing the company with valuable insights into customer behaviors and preferences [51].

Yunus et al [51] harnessed the power of decision tree algorithms to develop a cross-selling model aimed at assisting insurers in making strategic decisions when introducing new products to existing customers. The effectiveness of the

models which had 98%-99% prediction accuracy where tested by the company segmenting its customers. Specifically, engaging customers identified by the model as likely to make a positive purchase yielded positive outcomes. Simultaneously, a more intelligent, targeted, and differentiated communication channel was employed for customers whom the model predicted to show no interest in the new product. Comparing the results with previous periods where cross-selling was conducted without insights from the cross-selling model, the company experienced a significant 25% increase in the acceptance of new products by existing customers. This demonstrates the substantial impact and value of employing data-driven models in optimizing cross-selling strategies within the insurance industry.

2.7 Conclusion of Literature review

In conclusion, the comprehensive review of past literatures in this chapter has provided a clearer understanding of the dissertation's topic. Additionally, it has outlined the evolutionary trajectory of MLOps, offering a roadmap to address the challenges in operationalising machine learning models effectively. It has shown that the objective of this project, which is to implement a fully mature MLOps in Azure would will significantly contribute to existing research in this area. The utilisation of DevOps principles to automate the MLOps cycle will not only facilitate the achievement of this objective but also establish a framework for the seamless reuse of components or entire pipelines to develop and deploy other models for the insurance sector and across diverse sectors.

3 PROJECT METHODOLOGY

This chapter will provide a detailed report of all methods and steps used in implementing MLOps in this project.

3.1 Data Sourcing and Experimentation

3.1.1 Data Sourcing

Given that the primary goal of the project is to implement the MLOps methodology for automating the end-to-end machine learning lifecycle, the first step was to source for a suitable open-source insurance data set available online. The chosen data set for this project was the "Insurance Cross-Sell Prediction" obtained from Kaggle [52], which is licensed under the General Public License version 2 (GPL-2.0).

This dataset is for an insurance company that provides health insurance to its customers and would like to build a cross-selling model to predict whether current policyholders of health insurance would be interested purchasing the vehicle insurance offered by the company. The dataset includes features with information about customers and their vehicles, as detailed in Table 2:

Table 2: Description of the features in the Cross-selling insurance data set

S/N	Feature name	Feature Description	Type of Feature
1	Gender	Gender of the customer	Binary
2	Age	Age of the customer in years	Numeric continuous
3	Driving_License	0 : Customer has no DL; 1 : Customer has DL.	Binary (qualitative)
4	Region_Code	Unique code for the region of the customer	Qualitative categorical
5	Previously_Insured	0: Customer has no previous vehicle Insurance ; 1: Customer already has Vehicle Insurance.	Binary (qualitative)
6	Vehicle_Age	Categories of Vehicle age	Ordinal categorical
7	Vehicle_Damage	0: No damages to customer's vehicle in the past ; 1: Customer vehicle got damaged in the past.	Binary (qualitative)
8	Annual_Premium	Amount customer pays as premium in the year.	Numeric continuous
9	Policy_Sales_Channel	Anonymized code for channels used to reach customers (Different agents, mail, phone, In-person)	Qualitative categorical
10	Vintage	Number of days customer has been with company.	Numeric continuous
11	Response	0: Customer is not interested in Vehicle Insurance 1: Customer is interested in Vehicle Insurance	Binary (target variable)

As indicated in the table S/N 2 to 11 are the independent features while the target variable which the model will predict is the Response. The data has two files

id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0

Figure 5: A glimpse into the first five rows of the cross-selling insurance dataset

- **Training data:** CSV file with size 381109 × 11 (rows representing unique customers and columns for the customers' features.)
- **Inference data**: This CSV file with size 127037 × 10 (without the target variable) has information for customers that will be served as input to the model when it is in production.

3.1.2 Data Preparation

The raw data set had no missing values so there was no need for any form of data imputation. However, as shown in Figure 5, some of the columns needed label encoding and changing some features from categorical strings to categorical numerical and vice versa. The data cleaning and preparation that was done on the data is explained below.

Table 3: Label encoding and mapping of categorical features in the insurance data set

S/N	Feature name	Type of preparation	Data preparation Description
1	Gender	Label encoding	Replaced strings 'Male' and 'Female' with binary numerical values 0 and 1
2	Vehicle_Age	Label encoding	Replaced vehicle ages ('>2 Years', '1-2 Year', '< 1 Year') with categorical numerical values (3, 2, 1) that retains the ordinal relationship in the age.
3	Vehicle_Damage	Label encoding	Replaced strings 'No' and 'Yes' with binary numerical values 0 and 1
4	Policy_Sales_Channel Region_Code	Grouping	• Number of unique Region_Code = 53 • Number of unique Policy_Sales_Channel = 155 • frequency distribution for each region/channel. • Regions and channels with fractional distribution less than 0.01 were respectively grouped together as one region and channel named 'other'. • This reduced number of unique regions to 26 and number of unique channels to 10. • Changed both features type to string

3.1.3 Exploratory Data analysis(EDA)

Referring to the definition and explanation of EDA provided in the introduction chapter under the MLOps cycle, an exploratory data analysis (EDA) was conducted on the cross-selling insurance training dataset. This analysis aimed to gain insights into the characteristics of the data by using data visualization to explore relationships between different features and understand how these features influence the policyholder's interest in vehicle insurance.

1. **Check for data Imbalance**

   The distribution of the target variable (Response) in the training data shows that 87.74% of the customer data are those who are not interested in vehicle insurance, leaving the percentage of interested customer at 12.26%. As illustrated in Figure 6, the dataset exhibits an imbalance, highlighting the need for careful consideration in machine learning modeling.

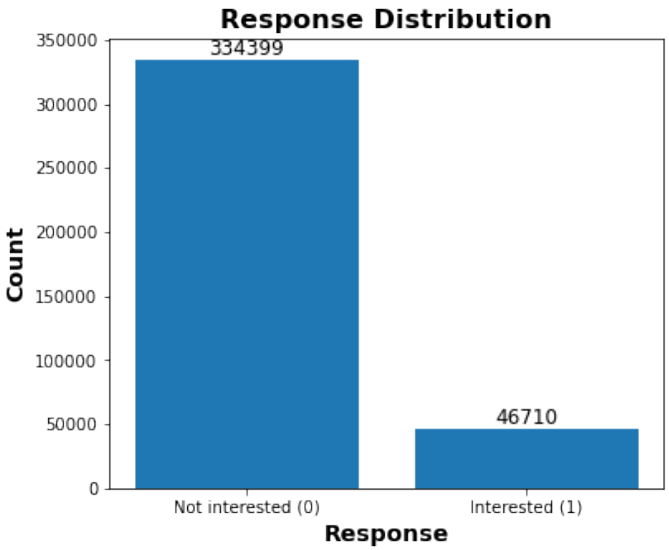


Figure 6: Distribution ratio of customers response in the cross-selling insurance dataset

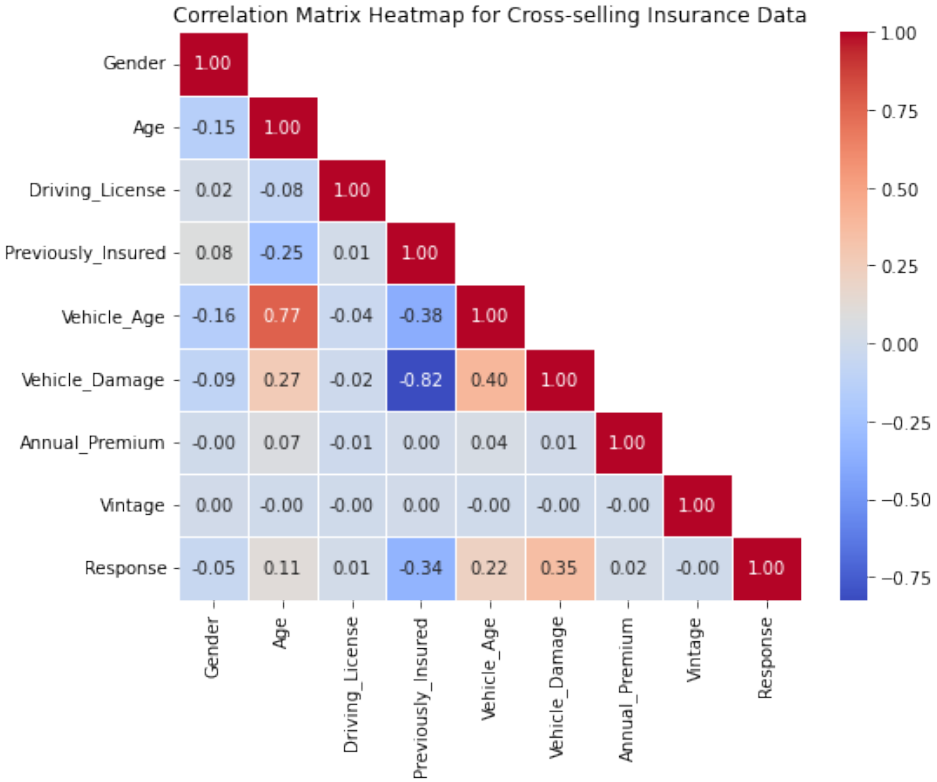


Figure 7: Correlation Matrix for numerical features in the cross-selling insurance data

2. **Correlation between data Features**

Figure 7 depicts the correlation matrix that shows how every pair of features are correlated through the correlation coefficients. +1 signifies perfect positive correlation (both variables increase or decrease proportionally together), and -1 signifies perfect negative correlation (as one variable increases, the other decreases). Most pairs of features in the dataset have weak correlation or no linear relationship between the features since their coefficients are close to zero. There is a notably positive correlation between vehicle age and policyholder age, indicating that older customers are more likely to own older vehicles, and conversely, younger customers tend to have newer vehicles. Also a strong negative correlation between vehicle_Damage and Previously_insured implies that there is

a higher tendency for customers who have not previously purchased vehicle insurance to have had their vehicles damaged in the past, and vice versa.

3. **Distribution of the binary categorical variables in the data.** The bar charts provide insight that customers who are likely to be interested in the vehicle insurance are those who have had their Vehicle damaged in the past and those without a previous vehicle insurance.

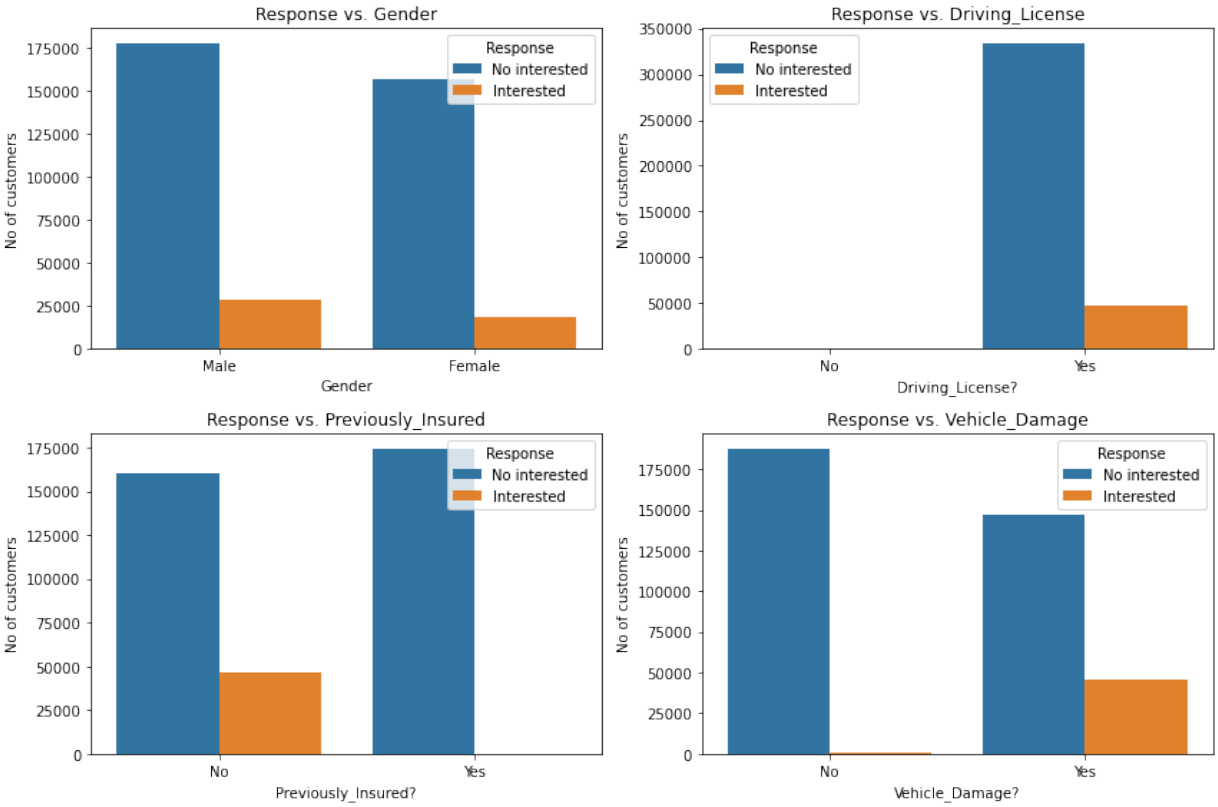


Figure 8: Distribution of categorical variables in each class of Customer's response

4. **Distribution of the continuous variables in the data.** The Kernel Density Estimation (KDE) was done to estimate the underlying probability density function for each of the continuous variables. As deduced from the plots in Figure 9, the distribution of customers interested and not interested in vehicle insurance appears to be uniform across various values of Age, Annual Premium, and Vintage. This implies that there is no discernible concentration of responses within specific ranges of these features. Suggesting a lack of a strong correlation between the continuous features and the likelihood of a customer being interested in vehicle insurance.

5. **Distribution of customer vehicle_age across the classes of response.**

   The pie chart in Figure 10 illustrates the distribution of vehicle ages among policyholders in the dataset. It indicates that a significant proportion of the vehicle ages falls within the 1-2 years range. From the bar chart in Figure 10, the ratio of customer response (interested/Not interested) in each category of vehicle, suggests that customers with older vehicles are more likely to express interest in vehicle insurance compared to customers with newer vehicles.

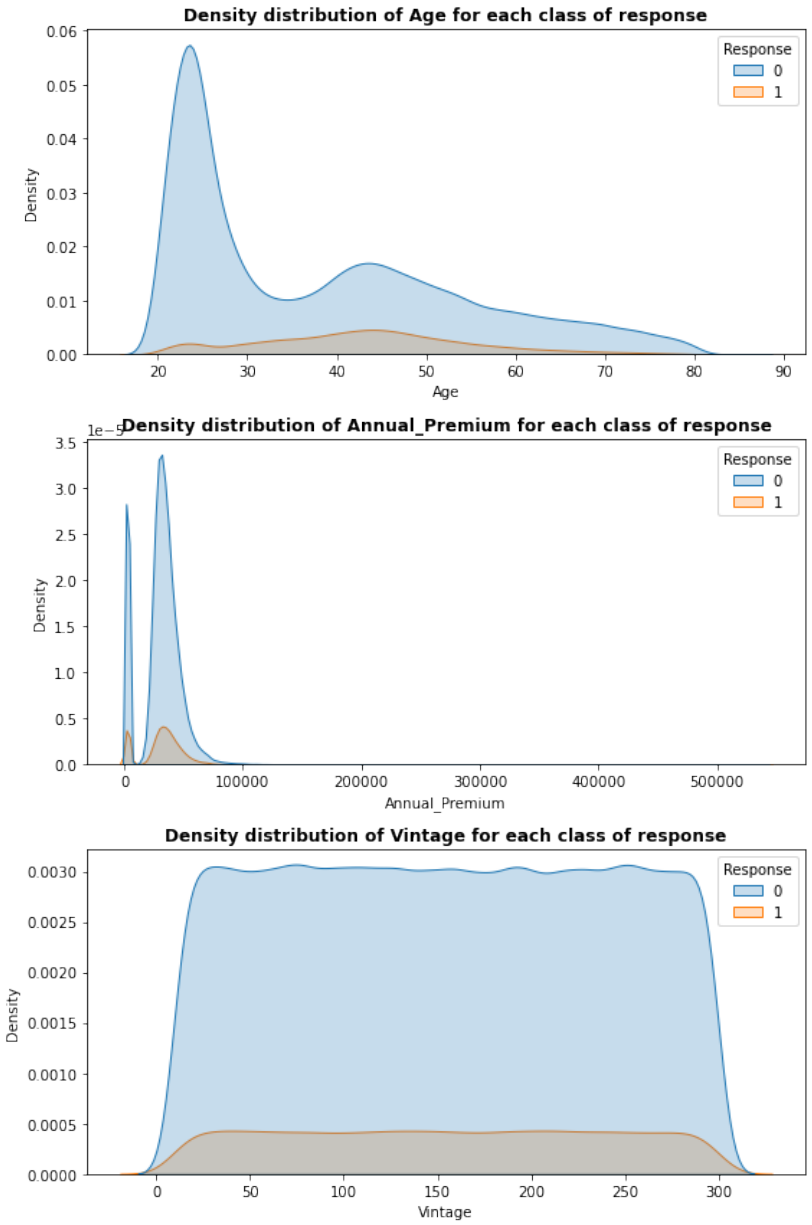


Figure 9: Distribution of continuous variables in each class of Customer's response

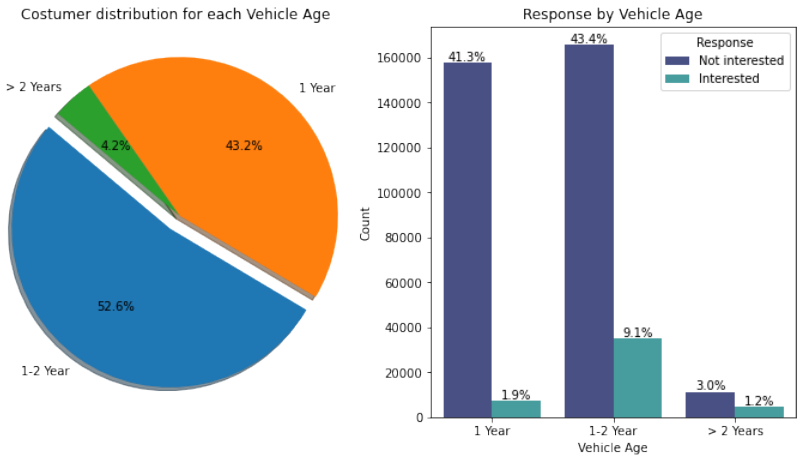


Figure 10: Vehicle_age distribution in the cross-selling insurance data

In summary, the conducted EDA provides insights into the distribution of the data, guiding decisions on subsequent steps such as data preprocessing and the formulation of a feature engineering strategy.

3.1.4 Data Pre-processing

In order to make the data presentable and suitable for the Machine learning models to efficiently learn the underlying relationships in the data, some preprocessing steps were carried out on the data;

1. **Train/Test data Splitting:** The cleaned data set from the data preparation step was split into training data (80%) and testing data (20%). Splitting the data before preprocessing was done to ensure that the preprocessing steps are fitted separately to the training and the test data remain unseen till it gets transformed using same preprocessors. This helps prevent data leakage, where information from the test set unintentionally influences the model during training and its generalization performance. The size of the resulting training and testing data set was (304887,11) and (76222,11) respectively.

2. **Handling Data Imbalance**: A classification data set is said to be imbalanced when the class proportions is skewed. In the Cross-selling Insurance data set, majority (78.74%) of the data set is made up of customers who were not interested in the vehicle insurance. When such imbalanced data is fed into machine learning algorithms, it may result in sub-optimal model performance as the models often prioritize the majority class, neglecting the minority class. In the insurance data, the minority class, representing interested customers, holds greater significance for the insurer than those not interested. Various data re-sampling methods were experimented with to address the data imbalance:

   - **Undersampling:** also known as Downsampling, involves randomly removing some of the data samples in the majority class [54].
   - **Oversampling:** also known as Upsampling, involves randomly duplicating some of the observations within the minority class [54].
   - **Synthetic Minority Oversampling technique for Nominal and continuous features (SMOTE_NC):** is an oversampling technique that generates synthetic data for the minority target class in the data set [53]. The algorithm synthesizes the new data samples by randomly selecting k-nearest neighbors for a minority instance, interpolates values for continuous features, and choosing values for categorical features based on their frequencies.

Each of the data balancing techniques above were applied separately on the training data. Subsequently, the resulting balanced training data from each technique was used to train a Logistic regression model and the model's performance evaluated on the testing data. Also, same logistic regression model was trained on the imbalance data set without any re-sampling and the models performance evaluated on the test data. The evaluation metrics for all four scenarios is presented in Table 4 to assess the effects of each data balancing method on model's performance.

In other to deal with the imbalanced nature of the cross-selling insurance data, Recall was chosen as the preferred metrics for measuring model's performance. Recall which is the ratio of true positive predictions to the total true positives and false negatives, indicates how well the model avoids false negative errors. A high recall means the model is avoiding misclassification of interested customers as not interested, which aligns with the goal of the Insurance company.

Table 4: Evaluation metrics for comparing outcomes of data balancing techniques

	model	precision_train	precision_test	recall_train	recall_test	f1_train	f1_test
0	Baseline_model_with_no_resampling	0.298	0.244	0.001	0.001	0.002	0.002
1	Undersampled_model_1	0.733	0.277	0.936	0.935	0.822	0.427
2	Oversampled_model_1	0.732	0.277	0.936	0.936	0.822	0.427
3	SMOTENC_model_0.25	0.410	0.277	0.946	0.934	0.572	0.427

The number at the end of the model name in the table indicates the value for the sampling_strategy parameter which was used to specify the desired ratio of the number of samples in the minority class to the majority class after resampling.

From Table 4, the metrics obtained from all three Data sampling methods are relatively close in terms of the recall scores. Although the SMOTENC method gave the highest recall score in the training, Undersampling was selected as the preferred data balancing method because it required lesser computational resources and time to run while still maximizing the Recall scores for both training and testing. Applying undersampling to balance the data set resulted in a training data of size (74736, 11).

3. **One-hot encoding**: was used to convert nominal categorical features (Region_Code, Policy_Sales_Channel) into a binary matrix format which is suitable for machine learning algorithms. Each column of this matrix represents dummy variables for each of the 26 regions and 10 channels in the data set. This made the size of the training data set to be become (74736, 43).

4. **Data Normalization**: Sci-kit learn Min-max StandardScaler was used to standardise continuous features (Age, Annual_Premium and Vintage) to a similar scale without actually altering the distribution of the data set. Standardization, also known as z-score normalisation is applied to transform the features of a dataset such that they have a mean of 0 and a standard deviation of 1. Hence, enabling the ML algorithms to converge faster and prevent features with larger scales from dominating the learning process. The formula used for the Min-max standardization is:

$$X_{scaled} = \frac{X - \text{mean}}{\text{Standard deviation}}$$

5. **Feature Selection**: was done to identify features that have the strongest relationship with the target feature, Response. This would improve the effectiveness of the Machine learning and also reduce the complexity and computational resources of the process. The Scikit-Learn SelectKBest package was employed to determine the optimal 36 input features for model training by utilizing mutual information (MI), a metric that measures the dependency between each feature and the target variable. The remaining 7 features which had MI scores less than 0.001 had negligible impact on customer response, leading to their exclusion from model training.

In line with best practice, preprocessing steps 3-5 enumerated above were combined with the model training in one pipeline [55]. Separate pipeline steps for one-hot encoding and normalisation were enclosed within a column transformer, given that one-hot encoding and normalisation transform each feature independently. Finally, the pipeline was structured to execute the column transformer as the initial step, followed by feature selection as the second step, and fitting of the model to the preprocessed data as the final step.

This integration of preprocessing tasks into the overall model training workflow not only prevents data leakage but

also ensures that the trained model from the pipeline encapsulates these preprocessors for seamless transformation of test data or inference data when the model is deployed for predictions.

3.1.5 Model selection, training and Evaluation

The classification problem to be solved using the cross-selling insurance data requires supervised machine learning. Four ML classification models were fitted to the training data using the preproccessing-training pipeline that was built. The four chosen supervised ML models are described below

1. **Logistic Regression (LR):** is a classification algorithm that predicts the likelihood of policyholders belonging to each of the binary Responses ("Interested" or "Not interested"). LR models the relationship between the input features and the target feature using a logistic function, also known as a sigmoid function.

2. **Random Forest:** Random Forest is an ensemble technique that combines multiple decision trees to make accurate predictions about customer responses. By aggregating individual tree predictions, it creates a robust model that considers feature importance and reduces over-fitting.

3. **Gradient Boosting (GB):** Gradient Boosting is another ensemble approach that sequentially builds weak learners to correct the errors of the previous ones, effectively improving predictive performance for customer Responses.

4. **K Nearest Neighbours (KNN):** is a supervised learning algorithm that assesses the similarity between data points based on metrics like Euclidean distance between the feature vectors of different policyholders. KNN leverages the assumption that policyholders with similar profiles or characteristics are likely to share common a interest in vehicle insurance.

The trained models were then used to predict the responses of the policyholders in the test dataset. To evaluate a model's performance and generalization power, the predicted responses were compared with the actual responses from the testing data. The classification metrics defined below were calculated to review the predictive performance of the model with higher values indicating better performance than lower values:

- **Accuracy:** is the proportion of the predicted responses that were correct out of the total testing data predictions, providing an overall measure of model performance.

- **Precision:** is the ratio of true positive predictions to the total predicted positives, indicating how well the model avoids false positive errors.

- **Recall:** (Sensitivity or True Positive Rate) is the ratio of true positive predictions to the total true positives and false negatives, indicating how well the model avoids false negative errors.

- **F1 Score:** is the harmonic mean of the precision and recall.

- **Receiver operating characteristics (ROC)** is a graphical plot of the true positive rate on the y-axis against the false positive rate, indicating the model's performance across different classification thresholds.

- **Area under the curve (AUC)**: is a numerical quantification of the area under the ROC curve.

Table 5: Evaluation metrics for the trained Insurance cross-selling models

	Model	Accuracy	Precision	Recall	F1 Score	AUC Score
0	Logistic Regression	0.578	0.206	0.858	0.332	0.738
1	Random Forest	0.635	0.208	0.703	0.320	0.711
2	Gradient Boosting	0.605	0.214	0.831	0.341	0.753
3	k Nearest Neighbours	0.606	0.177	0.607	0.274	0.606

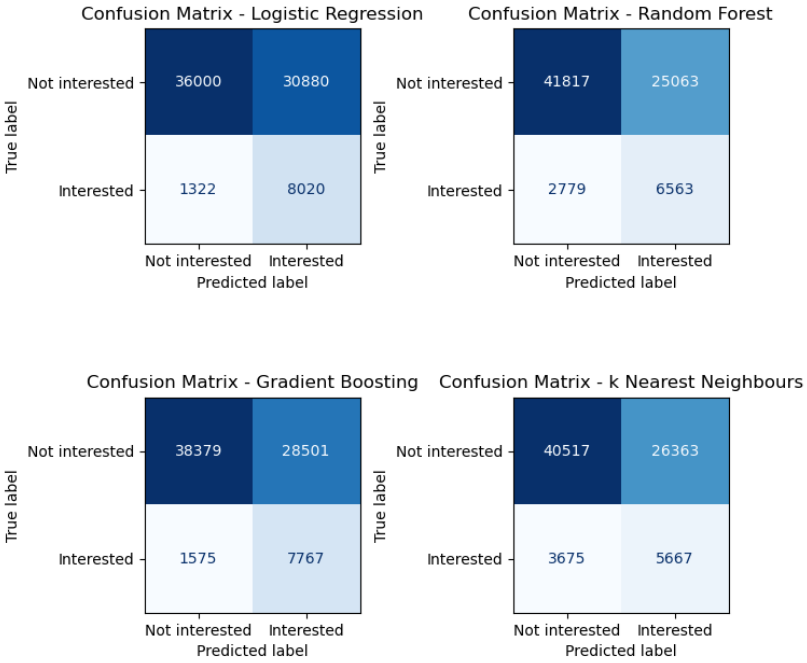


Figure 11: Confusion matrices for the trained models prediction on the test data

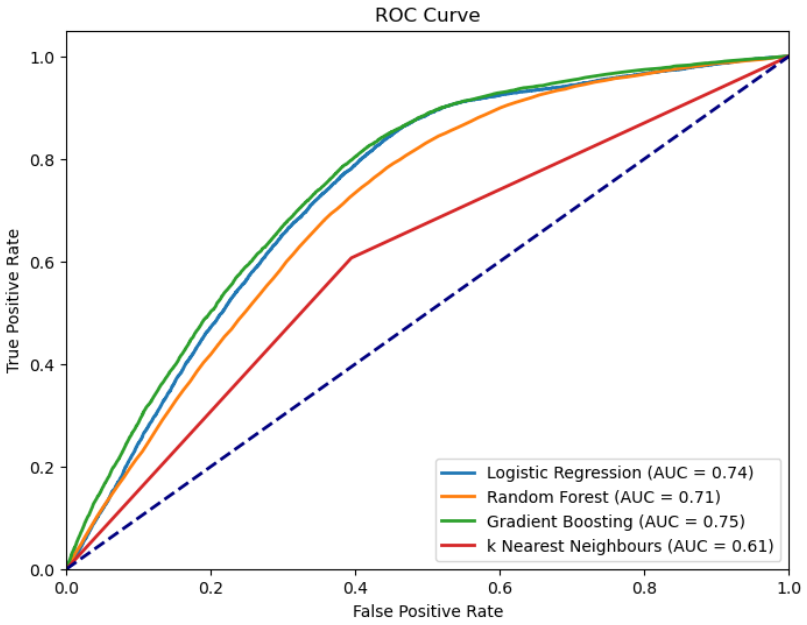


Figure 12: ROC evaluation of the trained models prediction on the test data

The evaluation was done using a for loop to iterate across all four models and the metrics, confusion matrix and Area under the ROC curve were presented for review.

The criterion for choosing the best-performing model was based on achieving the highest recall. In cross-selling, the objective of the insurance company is to identify customers who express interest in the Vehicle insurance policy. Maximizing recall ensures that policyholders interested in the product are not misclassified as 'Not interested.'.

From the evaluation results in Table 5 and Figure 11, the Logistics regression was selected as the best model to proceed with the MLOps process. Hence, it was to save out in a serialized format using the pickle or joblib function. The experimentation phase covered the first 3 processes in the MLOps cycle of Figure 1. The deliverables from the experimentation were the Jupyter notebook containing the Python code and a trained model in a pickle file which should be packaged for deployment.

3.2 Preparation of Resources and infrastructure for MLOps

Following the initial experimentation phase, the focus shifted to building the infrastructure. This involved setting up Azure machine learning resources and continuous integration pipelines within GitHub. Much of the coding utilized the Azure Machine Learning Python SDK v2. For terminal command execution, the Azure command line interface (CLI) proved effective and the AML Studio served as a user interface, offering a central hub for viewing code outputs, logs, and results. While development primarily took place locally in Visual Studio Code, experiments and jobs were submitted to AML by connecting to the workspace with the subscription and workspace credentials.

The flow diagram in Figure 13 outlines the different steps that were implemented to build and operate MLOps for the insurance cross-selling model. It shows how the pipelines were integrated/automated and the resources/infrastructures that were setup to achieve the desired outcome.

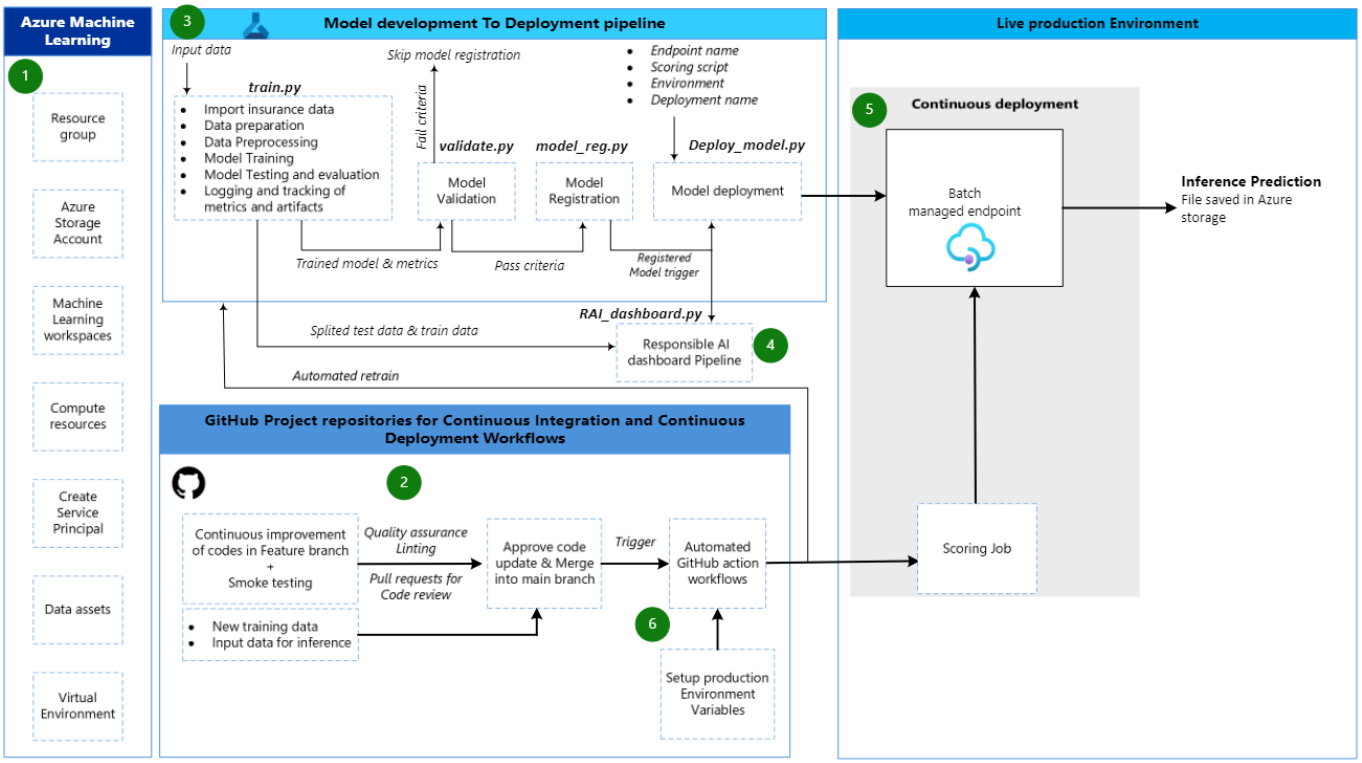


Figure 13: Flow diagram of MLOps implemented in Azure for Vehicle Insurance model

3.2.1 Azure machine learning workspace configuration

An Azure account was created using student Aston's email address and this activated a free student subscription with $100 Azure credit from Microsoft. The section labeled '1' in Figure 13 contains the resources that were set up for utilisation of Azure machine learning services. Their definition and application are briefly explained below [56]:

- **Resource Group (RG):** is a logical container designed within Azure to organize and streamline the management of a collection of related resources and their metadata. The RG was created using the Azure CLI by specifying a distinctive name, and the hosting region (UK West). See fig A.1.1 in the appendix to view the RG for this project.

- **Azure Machine Learning Workspace:** as the name implies is the central workspace that provides tools and resources for MLOps workloads. A workspace was created using the Azure CLI by providing a name, access to the subscription and RG. Take a look at Figure A.1.2 in the appendix to get a preview of the project AML Studio workspace which provides access to view all the models, pipelines, endpoints, compute resources, and other project-related assets. When a workspace is provisioned, AML automatically sets up support resources like storage account(Azure blob storage) and Azure Container Registry for storing virtual environment images.

- **Compute resource:** is the virtual machine (VM) that was set up to perform various computing tasks. AML offers various types of scalable VM with varying capacities and costs. A compute instance was created for running most of the workloads during development. However, the non-interactive automated production workflows were configured to run on a compute cluster. This choice was made due to the cluster's inherent scalability, enabling it to utilize more than one node simultaneously and scale down when not in use.

- **Service Principal (SP)**: To establish an identity for GitHub to authenticate and access the Azure Machine Learning (AML) workspace, a Service Principal was created using Azure Microsoft Entra ID. The Service Principal was then assigned a contributor role to function as a non-interactive identity with access to run automated GitHub actions workflows in the MLOps production environment.

- **Data assets:** In the AML workspace, data assets offer a means to store data in distinct formats (URI file, URI folder, MLtable). These assets support versioning of the data and provide a convenient way to reference data stored in a data store.

- **Virtual Environment**: In line with best practice, a new conda environment was created locally to isolate dependencies and packages for this MLOps project. This was done to create a self-contained environment, ensuring easy reproducibility both within the AML workspace and in the production environment. The dependencies were exported to a conda.yml file for building a virtual environment container registry in the workspace using the Linux ubuntu base image. Once the environment was successfully built, its name and version were referenced in the code and pipelines, specifying the environment in which to execute the code. Appendix Figure A.1.3 shows a screenshot of the environment build in the AML workspace.

3.2.2 Continuous Integration (CI) with Git

A shared GitHub repository was created for the project in Kainos account which granted access to the student and supervisors to collaborate. The repository provides a central location for storing code and files for the project with a record of each file's versions and revision history. Continuous Integration allows for the creation of temporary

feature branches derived from the main branch. These branches are pulled into the local project directory for code development and testing to ensure their proper functionality. Once updates or file additions are deemed ready for integration, the modifications are staged and committed to the branch. Subsequently, a pull request is created for supervisors to review the code, offer feedback, approve the request, and merge the feature branch into the main branch.

The section labeled 2 in Figure 13 presents a visual flow of the CI strategy that was used. In other to maintain code quality standard, GitHub action workflow was configured to run a linting check with Flake8 package, across all files in the feature branch, whenever a pull request is created. See Appendix Figure A.1.4 for details of the linting Workflow in the YAML configuration file.

3.2.3 Convert Notebooks to scripts

In line with the DevOps principles (CI/CD, automation and collaboration), there was need to convert the Python code in experimentation Jupyter notebook to Python scripts (.py). This involved removing all unnecessary code, outputs, markdown cells, and refactoring code into modular functions that execute small individual tasks. The Python scripts were organised in a readable manner, well commented, with an explanation of what the script does and packages imported at the beginning of the scripts. More details about the Python scripts for the MLOps are provided in the section below.

3.3 Develop the end-to-end Machine Learning Operations

This phase of the methodology will delve into the Python scripts that were written to execute the steps within the MLOps cycle and how AML Python SDK was leveraged to connect the scripts.

3.3.1 Description of the Python scripts from Model development to deployment

The section labeled 3 in Figure 13 shows the Model Training to model deployment pipeline and the connection between the steps. First, the individual Python scripts were developed and tested using a smoke test. The smoke testing process included the execution of the code to identify and fix any bugs or defects that might prevent the script from achieving the intended outcome outlined in the acceptance criteria. Below is a description of the individual scripts:

1. **Model Training script:** As illustrated in Figure 13, the train.py script comprises distinct functions to perform tasks such as:

    - Importing the insurance cross-selling data
    - Label_encoding
    - Train/test data splitting
    - Check for imbalance in training data and handling imbalance
    - Data preprocessing-Model training pipeline explained in the experimentation method.
    - Evaluate model performance
    - Saveout metrics dataframe in a CSV file and log metric using MLFlow Auto-log
    - Save trained model with highest recall to a pickle file and also log the trained model as MLFlow artifact.

    Although the code still allows for iteration over four classification models training and saving the model with the highest recall, only the logistic regression model was passed in the train.py script to reduce computational

time. MLflow logging was employed due to its ability to treat each run as an experiment and package the model and its dependencies in a deployable format. By leveraging the azuremL-mlflow package, metrics and artifacts logged with MLflow during each run were readily accessible within the AML workspace.

It is essential to note that the argparse package was used to facilitate the passing of arguments and variables to the scripts, serving as both inputs and outputs. For instance, in the case of train.py, which is designed as a reusable script for training various datasets, the path to the raw cross-selling data was specified as args.input_data. This approach not only grants flexibility in adjusting argument variables (or specifying default) but also provides a way to reference output arguments from one script as inputs for another when building pipelines. As depicted in Figure 13, outputs from the train.py script, the trained model artifact and evaluation metrics file became the input to the validate.py, while outputs split testing data and balanced training_data were passed to the RAI dashboard construction pipeline.

2. **Model validation for deployment:** The next step is carried out by the validate.py script, whose objective is to assess whether a newly trained model meets the criteria set for redeployment. Redeployment refers to the process of updating or replacing an existing deployed model with a newly trained version of the model in the production environment.

   The script contains three functions described below:

   - **get_evaluation_recall**: imports the evaluation_metric file by passing the output from train.py as input argument path and returns the recall score of the newly trained model assigned to a variable "retrained_model_recall" .
   - **get_last_model_recall:** obtain details of the last version of the vehicle insurance model registered in the AML workspace. The recall which was included as a property of the model during model registration, is extracted and assigned to a variable "deployed_model_recall".
   - **main:** this main function of the script executes the first two functions and runs an 'If loop' logic which checks the criteria "retrained_model_recall_value >= deployed_model_recall". If the criteria is met, it writes the deployment_decision "Yes, Model should be deployed" to a 'validation_result.txt' file. Else, it writes the deployment_decision "No, Model should not be deployed" to a 'validation_result.txt' file. This deployment_decision in the file path becomes the output from the validate.py script.

   The first model deployment will execute all steps in the pipeline labeled 2 in Figure 13. However, for subsequent model retraining, the model validation script will determine if the pipeline run will register and deploy the new model when running the next two scripts.

3. **Model registration:** A trained model will be registered in the AML workspace model registry only if it satisfies the validation criteria. The model_reg.py script responsible for this task comprises distinct functions, including one that takes the path to the 'validation_result.txt' as input argument and assigns the content as deployment decision. Another function receives the evaluation metrics from train.py as input and returns the recall of the model to be passed as model property in the model registration function.

   Finally, the script's main function uses AML SDK Model class to register the model to the workspace if the deployment decision is "Yes, Model should be deployed", then writes the model registration name to a registered_model.txt file. Else it skips the model registration and writes "No model was registered" to a registered_model.txt file. This file is passed as output from this step and an input to the model deployment script.

4. **Model deployment:** The last step in the train_to_deploy pipeline is execution of the deploy_model.py which happens when a new model is registered. As indicated in Figure 13, in addition to the registered_model.txt path, the four other requirements in bullet points are needed for a successful model deployment to an AML managed batch endpoint:

   - **Endpoint:** is the URL through which the model receives requests and returns predictions. BatchEndpoint class in AML SDK was used to create a batch endpoint that is suitable for the particular use case of predicting insurance customers' Responses. The script written for creating a batch endpoint required an **endpoint name** and **description** and it was run just once to set up the batch endpoint called 'batch-insurance'. Once the endpoint is created in the AML workspace, multiple models can be deployed to the same endpoint by specifying the endpoint name. This is depicted in the section labeled 5 in Figure 13.
   - **Scoring script:** The Scoring script named batch_driver.py which executes the task depicted in Figure 2, was developed. It includes two important functions init() and run(), following the format specified in the AML SDK. The former is called when the deployment container is initialized/started and used to load the model. While the run() function is called at every invocation of the endpoint to do the actual scoring and prediction. In addition, same label_encoding function, that was used to prepare the input data for training was included in the scoring script. In summary, the scoring script loads the model from the AML model registry, imports the batch inference data, transforms it using label_encoding and passes the data for prediction. Recall that the registered model includes the preprocessing transformations in the serialized format.
   - **Environment**: The virtual environment that was earlier built in the AML workspace was referenced by using, env = ml_client.environments.get(name="Aml-env", version="7") to get the specified environment required for the deployed model.
   - **Deployment name:** A function to generate a unique name for a model deployment was included in deploy_model.py. It uses a specified base name "insurance-classifier"+ timestamp to make each deployment name unique.

   Appendix Figure A.1.5 shows how the information above is used to configure AML BatchDeployment function in the deploy_model.py script, in addition to the last registered model in the workspace. The main function of the script reads the content of registered_model.txt and checks if it is $\neq$ "No model was registered", then it contains the name of the model that was registered in the previous script. This satisfies the criteria to run the batch deployment function and also sets the new deployment as the default deployment in the endpoint. Hence whenever a batch scoring job invokes the endpoint, the default model deployment is accessed for receiving requests and returning predictions.

3.3.2 Automate Model Training to Deployment with Pipelines

The four major scripts described in section 3.3.1 were developed and tested as stand-alone scripts to confirm that they carry out the intended functionality. However, the goal of MLOps practice is to automate the end-to-end ML cycle. So AML SDK was employed to create a component with each of the scripts in Figure 13, label 3, using yaml configuration files. The components were then integrated to run sequentially in a pipeline to automate Model training to deployment to the batch endpoint. Refer to Appendix Figure A.1.6 for an excerpt from the Train_deploy_pipeline.py

script, demonstrating the interconnection of components with outputs from previous steps served as inputs to later steps.

3.3.3 Responsible AI (RAI) dashboard pipeline

AML machine learning has a robust infrastructure that allows pipeline code to be written for constructing RAI dashboard interface in the AML workspace. Label 4 in Figure 13 shows that the model registration triggers execution of the pipeline script for creating a RAI dashboard for the registered model. Also, the test dataset and train dataset outputs from the train.py are utilized in the RAI analysis. Responsible AI dashboard in Azure integrates ethical considerations such as model explainability, error analysis, interpretability, and feature importance transparency into the MLOps development.

RAI Inbuilt components listed below, were loaded from in AML curated component registry to run the steps in the pipeline. See Appendix Figure A.1.7 for the RAI dashboard pipeline code.

- **rai_constructor_component:** configures the dashboard setup by defining details like the title, task type, the input data, the reference to the latest model path in Azure, target column name and categorical columns in the data set. The output from this step is an empty RAI insights dashboard.
- **rai_erroranalysis_component:** This component adds insight about the model's error analysis to the RAI insights dashboard.
- **rai_explanation_component:** takes the RAI insights dashboard as input and add model explanation insights like feature importance plots and confusion matrix breakdown.
- **rai_gather_component:** consolidates the insights generated from the previous steps into a final interactive Responsible AI dashboard for the model that was registered.

The limitations to use the current version of AML RAI dashboard include a maximum allowance of 5000 rows for each of the test and training datasets and they must be in mltable tabular data format [57]. Hence the RAI_dashboard.py included a preliminary function to generate 5000 random policyholder samples from the test data which is originally 76,222 rows and the balance training data originally 74,736 rows. Other functions converted the data to the supported mltable and registered the data as data assets in the AML workspace. The final part of the script submits a pipeline job to create the RAI dashboard which can be monitored and reviewed in the workspace.

3.3.4 Scoring or Inference

A model that has been deployed to the batch endpoint is ready to be used for prediction. In AML, this is accomplished by invoking the endpoint, that is serving data to the deployed model to receive real-time predictions from the model. Recall that despite having one endpoint and being able to redeploy several models to the 'batch-insurance', the deployment script sets the latest deployed model as the default model in the endpoint. This means that when the endpoint is invoked, the inference data is routed to the default model for prediction.

The script for submitting a scoring job to the 'batch-insurance' endpoint was created and named score_job.py. It runs whenever new policyholders' data in a csv file is added to a folder named 'inf_data'. The two functions in the script are explained below:

- **reg_data:** Registers the 'inf_data' folder and its content (which can be more than one CSV file) as a Uri_folder data asset in the AML workspace and returns the AML reference path to the data which is then passed as input to the invoke function.
- **main:** The main function of the script uses AML Python SDK command "job = ml_client.batch_endpoints.invoke(endpoint_
insurance', input=input)", for invoking the endpoint. Then it submits the scoring job for execution and indicates the path to download the output prediction file in the workspace.

Details about this is shown in the section labeled 5 in Figure 13. The scoring job is monitored in AML workspace and the resulting predictions are located in a file named "predicted.csv" generated by the job.

3.4 Automate MLOps with GitHub action

The final phase of the MLOps implementation aimed to automate running of the pipelines developed in section 3.3 above, enabling them to function seamlessly and autonomously in a production environment. This was achieved by using GitHub action workflows to automate continuous training and model deployment as well as the submission of a scoring job as represented in the section labeled 6 in Figure 13.

Firstly, a production environment was set up in the project GitHub repository. The client ID, subscription ID, tenant ID and client secret associated with the Service principal created earlier in Azure were securely stored as environment variables for the production environment setup. This configuration facilitates a continuous delivery process, for integrating code changes and enhancements into the ML operations pipeline in production.

The two GitHub workflows that were created using yaml configuration files as described below:

1. **Continuous Training and Continuous Deployment workflow:** is located in the GitHub repository path "/.GitHub/workflows/Deploy_train_prod.yml". It contains a defined sequence of steps to execute an automated process for retraining and redeploying updated vehicle insurance model by running the Train_deploy_pipeline.py. Authoring with YAML syntax, the workflow is triggered either manually or by code push to specified directories in the main repository branch and then runs a job in the production environment following the steps below:
    - Checks-out to the project repository
    - Installs the AML CLI,
    - Login to Azure using the saved credentials in the environment variables.
    - Installs Python
    - Creates an AMl compute workspace
    - Reproduce the conda environment for the project using the conda.yml file
    - Runs the Train_deploy_pipeline.py which submits the pipeline job to AML workspace.
2. **Batch Scoring job workflow:** for automating submission of inference request to the model in production is located in the GitHub repository path "/.GitHub/workflows/score_prod.yml". The workflow was authored to run the same steps outlined above except that it executes the score_job.py script instead of Train_deploy_pipeline.py. Also, this workflow is triggered by either a manual workflow dispatch or a push to the 'main' branch, specifically when changes are made within the inference directory.

Both workflows were tested and debugged to confirm that they work as expected while logs and results of the pipeline jobs are viewed in the AML workspace.

3.5 Project management and Azure cost management

During the three months of this project, GitHub project board was leveraged as a pivotal tool for project management. Although the project was primarily executed by the student, weekly meetings were held with the supervisors at Kainos during which discussions were had about the intended direction of the project. Agile project management was applied to break down the project into small and adaptable weekly tasks set up as tickets in the project board. The GitHub project board provided a visual and collaborative platform, for organizing, prioritizing tasks, and tracking progress (See snap shot of the project board in Appendix Figure A.1.8 and A.1.9). The contributions of the industry supervisors, especially in reviewing the pull requests before approving for merging of feature branches into the main GitHub branch, serves as the collaboration component essential for the successful implementation of MLOps.

A part of project management is cost control and monitoring which becomes even more crucial when working with cloud computing platforms like Azure. Some of the ways employed to manage and reduce cost spent in the Azure credit are enumerated below:

- Keeping track of Azure subscription cost monitor for each resource and try to understand which resources such as storage, container registry, workspace etc is taking up cost.
- The decision to maintain all files related to the project in the GitHub repository without having to upload and store them in Azure helped to reduce storage costs.
- Compute resources like the compute instance were set to stop or go off after 15 minutes of inactivity. On some instances, it was deleted and created only when needed.
- Most of the code development and testing were done locally in the Visual studio code editor instead of using AML notebook or web CLI which needs the compute to be run.
- Resources such as registered versions of model, data sets, and environments that are not actively needed were archived and failed experiments pipeline jobs that had successfully been debugged were deleted.

4 RESULTS AND DISCUSSIONS

The main outcome of this industry-based project is a functional MLOps system documented and stored within the project repository designed to be both reusable and reproducible. This repository can serve as foundation for implementing future MLOps projects and be readily adapted for various ML model contexts. Once an Azure subscription is available and the AML resources outlined in Section 3.2.1 are set up, it can be readily utilized. As illustrated in Figure 14, the repository is organized such that the **.GitHub** folder accommodates the automated continuous delivery workflows, the **setup** directory contains scripts and YAML files for AML resources setup, and the **src** folder encompasses the directories for the three main pipelines

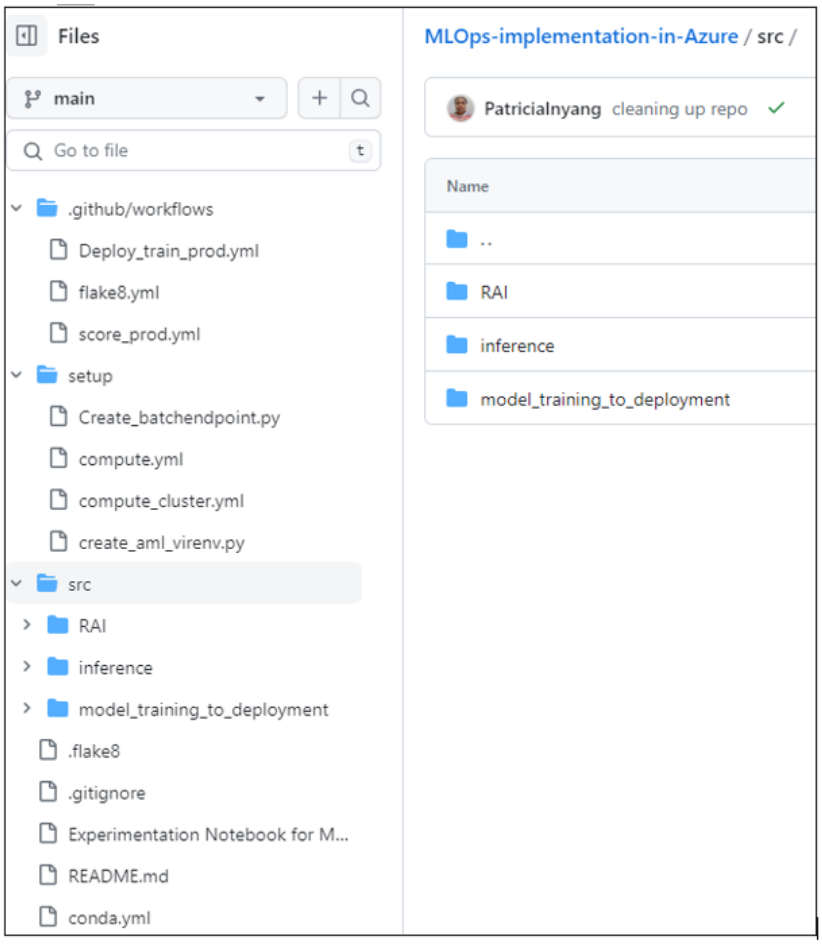


Figure 14: Structure of the MLOps project GitHub repository

4.1 Result from the Model training to deployment pipeline

The "/src/ model_training_to_deployment/" directory contains the four Python scripts for the steps explained in methodology 3.3.1, which can be run as standalone scripts, but were connected in the pipeline explained in 3.3.2. Figure 15 shows a visualisation of a successful run of the Train_to_deploy.py pipeline job in the AML workspace and the connection between the steps as explained in 3.3.1.

1. **Result of model development step:** As a result of tracking with MLflow, details of the trained Logistic regression model like the performance metric and artifacts are accessible by clicking on the train_model step of the completed pipeline job.

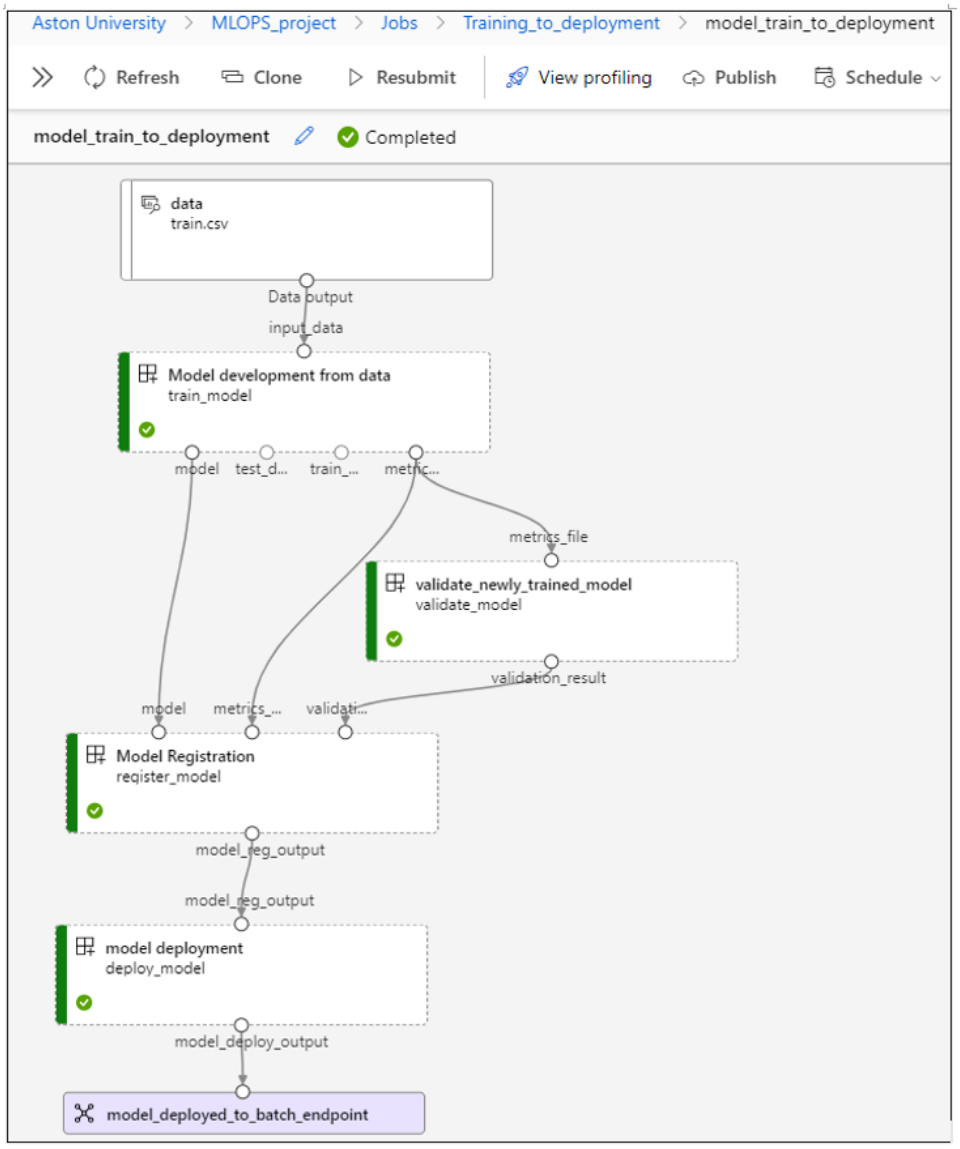


Figure 15: Visual presentation of a successful Model training to deployment pipeline job

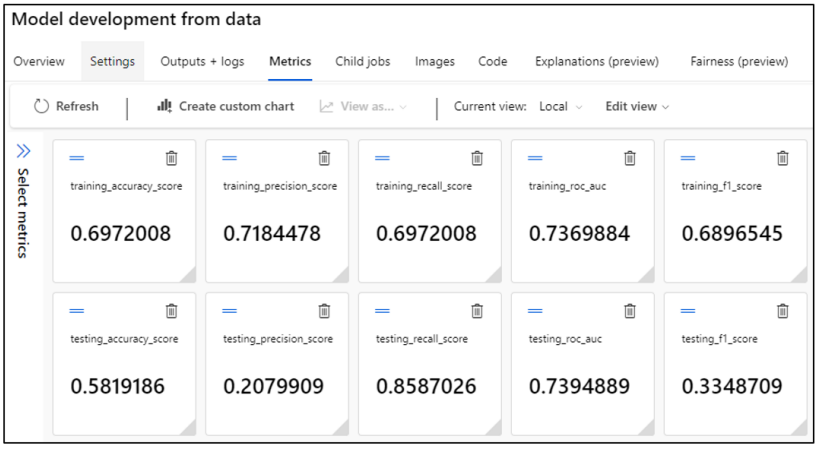


Figure 16: Evaluation metrics for the trained model tracked in AML workspace

Figure 16 above shows the evaluation metrics of the model, where the training scores at the top row provide insights into how well the model training learned underlying relationships in the training dataset. The testing scores in the second row represent the performance of the trained model in predicting the Responses for

Policyholders in the test dataset. While the overall prediction accuracy indicates that the model performed less effectively in generalizing on unseen data, the selected reference metric for this project insurance use case, Recall, displayed a higher score on the testing data. The low accuracy implies that approximately 42% of test dataset customers' responses were classified incorrectly. However, the recall score, highlighted in the bottom right section (dark blue) of the confusion matrix (CM) in Figure 17, reveals that approximately 85.9% of the interested customers were predicted correctly.

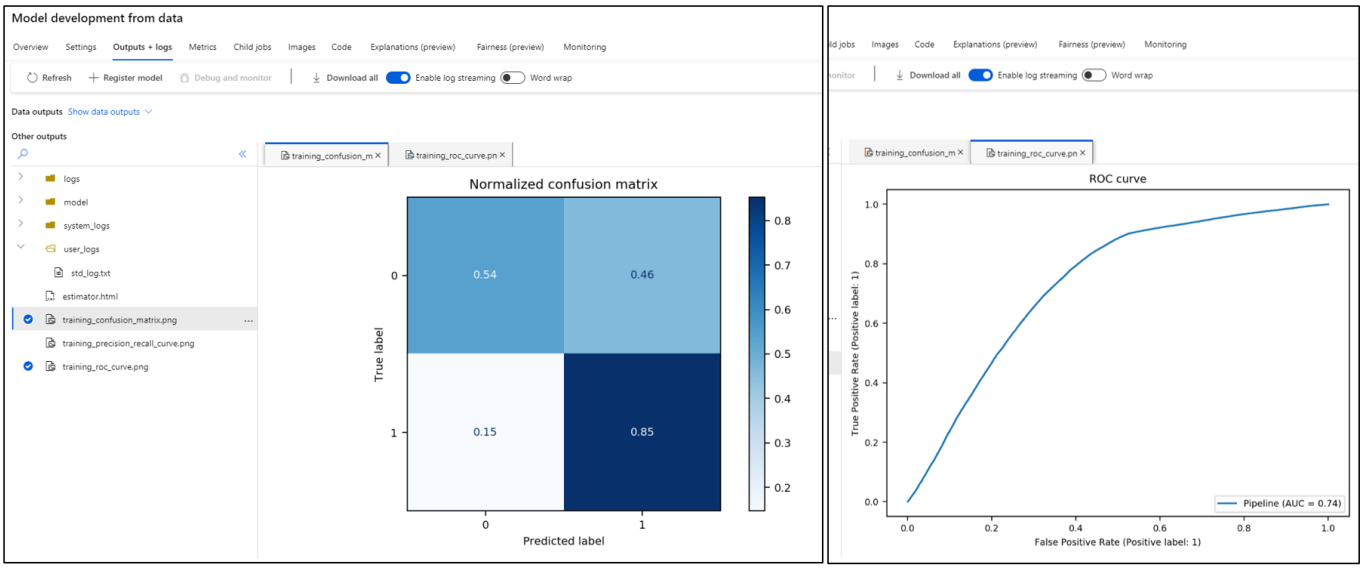


Figure 17: Confusion matrix and ROC curve for the trained Logistic regression model

The impact of misclassifying an interested customer (True label = 1) as 'not interested' (predicted label = 0) is more significant for the insurer because it results in a loss of cross-selling opportunities. In comparison, predicting a non-interested customer as 'interested' has a less severe consequence. Although the model generated by training.py may not be the optimal achievable model, it still aligns with the overarching goal of maximizing recall (True positive rate = 85.9%) and minimizing false negatives (bottom left section of the confusion matrix)

Additionally, one of the challenges that implementing MLOps aims to address is the reduction of the time taken to get model deployed. With this in mind, the first version of the model met the criteria for deployment, particularly considering that the designed MLOps process includes automated continuous retraining and redeployment of improved models to the production endpoint. Later versions of the Vehicle_insurance_model were determined by the result from the next pipeline step - Model validation.

2. **Result of the model validation step:** Figure 18 presents a comprehensive overview of the completed validate.py job, consolidating four windows into a single image. Observe how crucial information about the resources used for the completed validate.py job is provided in the AML workspace, including details about script input and output files. Notice also that the run was initiated by the non-interactive service principal identity, employed for authenticating GitHub and the Azure workspace. The red arrows are used to show how to navigate from the overview page to the deployment decision contained in the job's output file.

3. **Result of the model registration step:** is the Vehicle_insurance_model registered in AML workspace model registry as shown in Figure 19. As long as retrained models from this pipeline maintain same registration name, Azure will register a new version for any model that passes the validation.

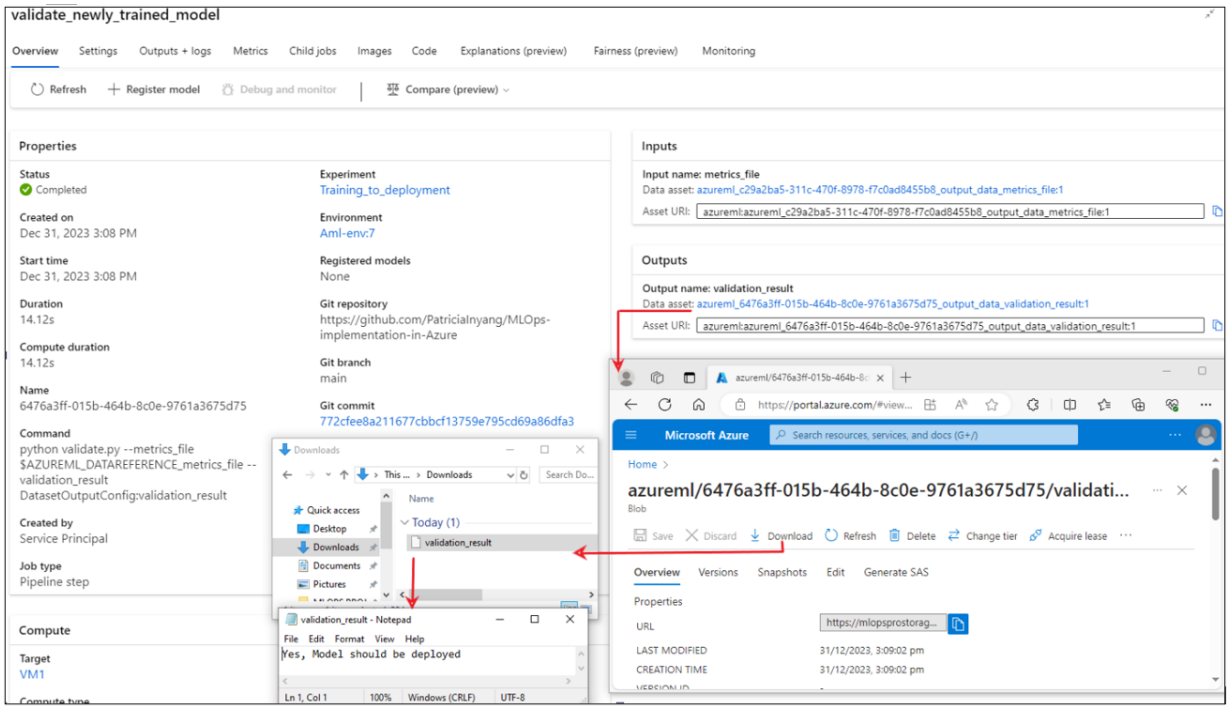


Figure 18: Review of result for a completed model validation step

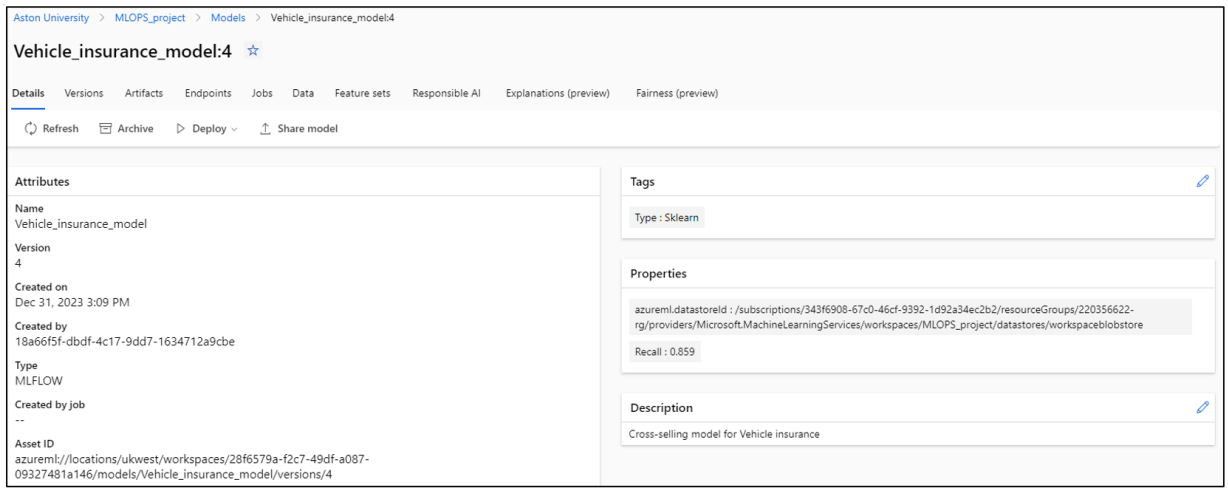


Figure 19: Details of registered Vehicle_insurance_model in AML model registry

4. **Result of the model deployment step:** The final step in this pipeline executes deploy_model.py, deploying the latest registered version of the Vehicle_insurance_model to the batch-insurance endpoint. Figure 20 shows the default model and the previous model deployments coexisting within the same endpoint. The REST endpoint is the URL through which users (in this case, the service principal) access the model for inference. Observe the unique names generated for each deployment based on the timestamp of deployment, facilitating distinction between different versions.

4.2 Review of results from the RAI insight dashboard pipeline

Azure Responsible AI dashboard provides up to four types of analysis to explain and interpret models. The successful run of the RAI (Responsible AI) pipeline job in Figure 21, shows that Error analysis and Explanation were performed to understand the behavior of the model. The final pipeline output 'dashboard' links to the populated model RAI

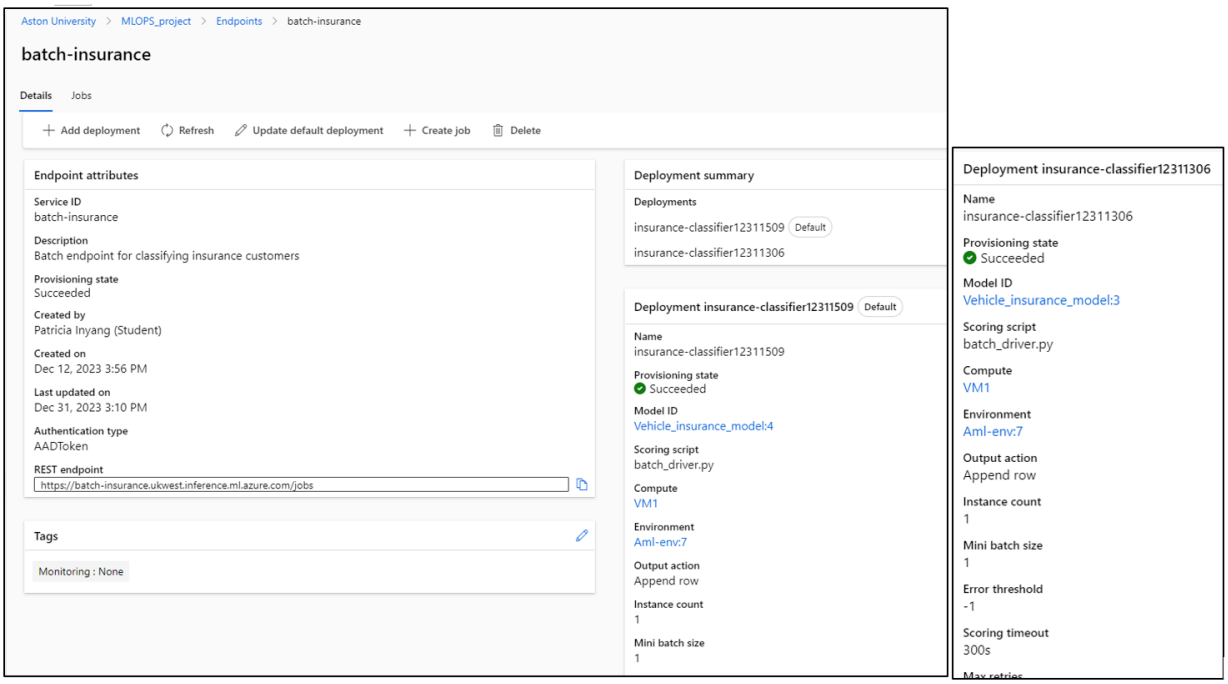


Figure 20: Vehicle_insurance_model deployments existing in Azure managed batch endpoints

dashboard which is interactive and allows for selection of features and metrics reveal various insights.

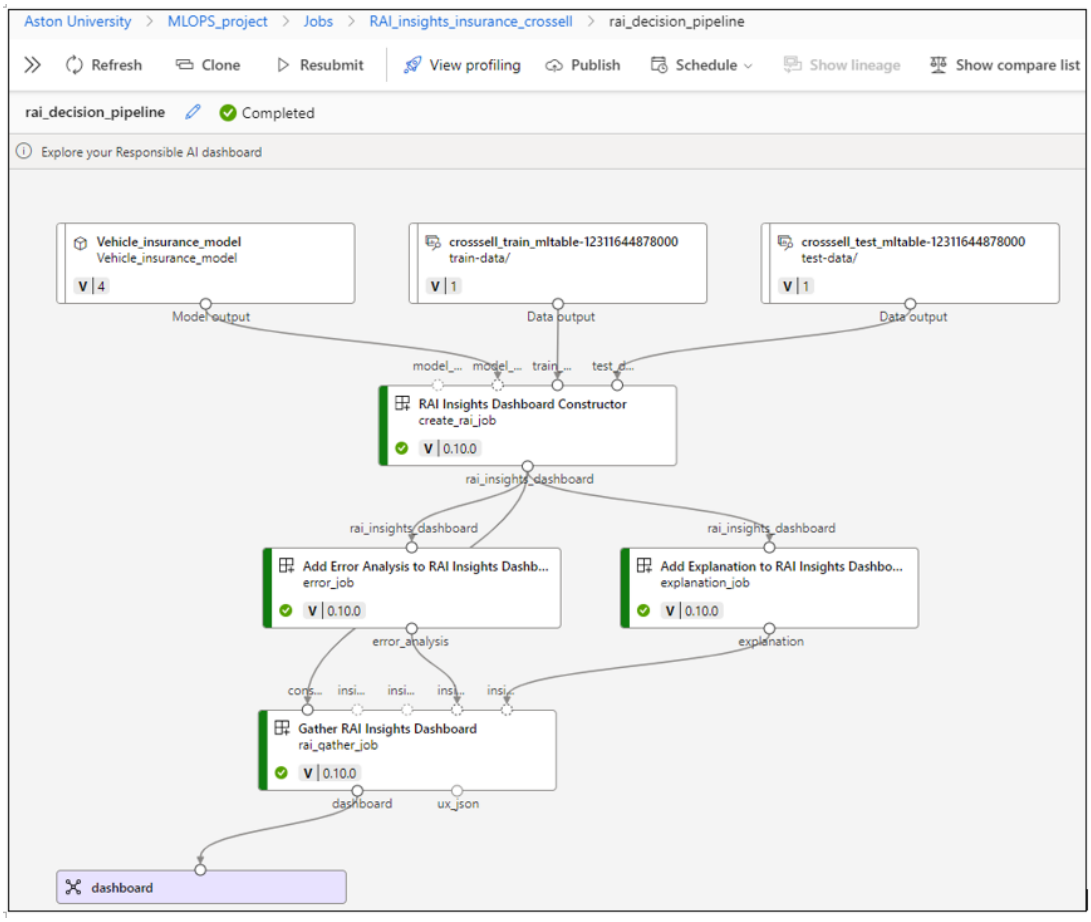


Figure 21: Completed pipeline job for Vehicle_insurance_model RAI insight dashboard

With reference to the constraints mentioned in Section 3.3.3, the 5000 policyholder data samples passed as test and train dataset inputs to the RAI pipeline represent a very insignificant fraction of the actual train and test data

used in training the model. Hence, the insights in the RAI dashboard may not reflect the true behavior of the model. However, the RAI pipeline, built as part of the MLOps infrastructure, functions as expected and will be more effective when reused to assess models with dataset rows not exceeding 5,000. The error analysis results from the AML RAI insight dashboard are presented and explained in Appendix figures A.2.10 to A.2.12. The actual feature importance of the Logistic regression model based on the trained model coefficient is depicted in Figure 22, revealing top features that influence policyholders' responses.

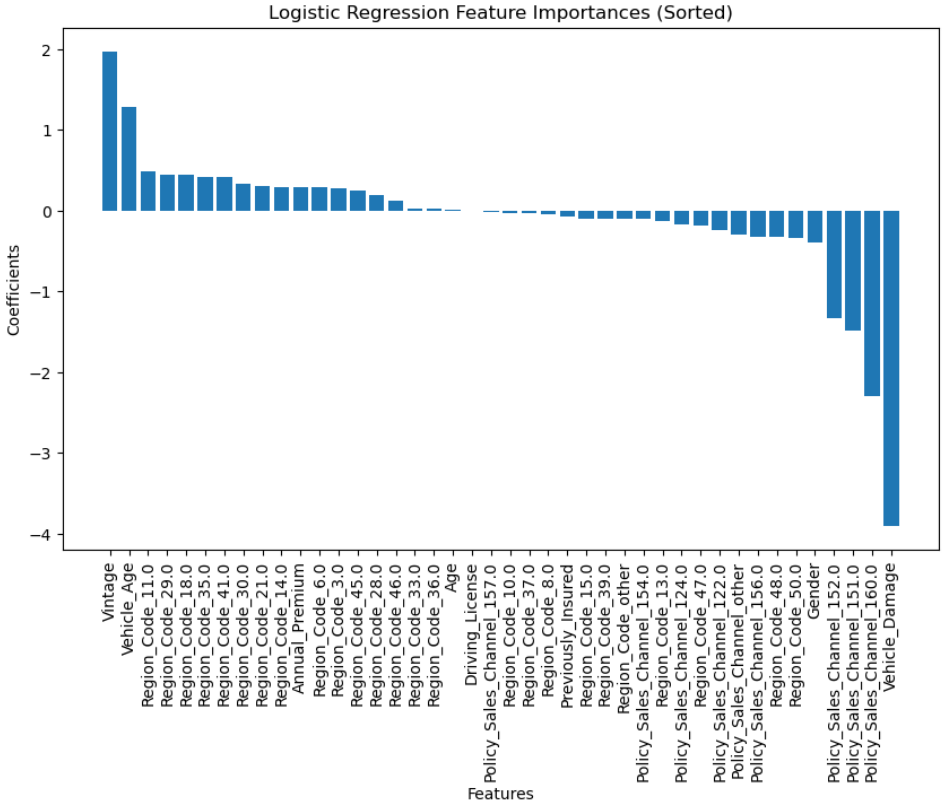


Figure 22: Feature importance plot for the trained Vehicle_insurance_model

The negative coefficients attributed to Vehicle_damage and policy sales channels 160.0, 152.0 and 151.0 means that these features have a strong influence on customer's responses, but in an inversely proportional way. That implies that a customer whose vehicle was damaged in the past (1) is likely to be 'not interested' (0) in the vehicle insurance policy and vice versa. The features with high positive coefficient imply that customers who have been with the insurance company for a longer period (vintage) and those with older vehicles tend to be interested in the Vehicle insurance offer. These interpretations provide valuable insights that will guide the Insurer's business decisions and cross-selling marketing strategies.

4.3 Result for the Inference pipeline:

The inference pipeline submits batch scoring jobs to the operationalised model by invoking the Azure ML managed batch endpoint. Within the inference directory, a folder named "inf_data" serves as the designated path for uploading mini-batches of CSV files containing unlabeled policyholder data for the model to make predictions. Figure 23 shows a successfully completed batch scoring pipeline job and the output scores in a prediction.csv file which is saved in Azure blob storage and also downloaded to the project repository.

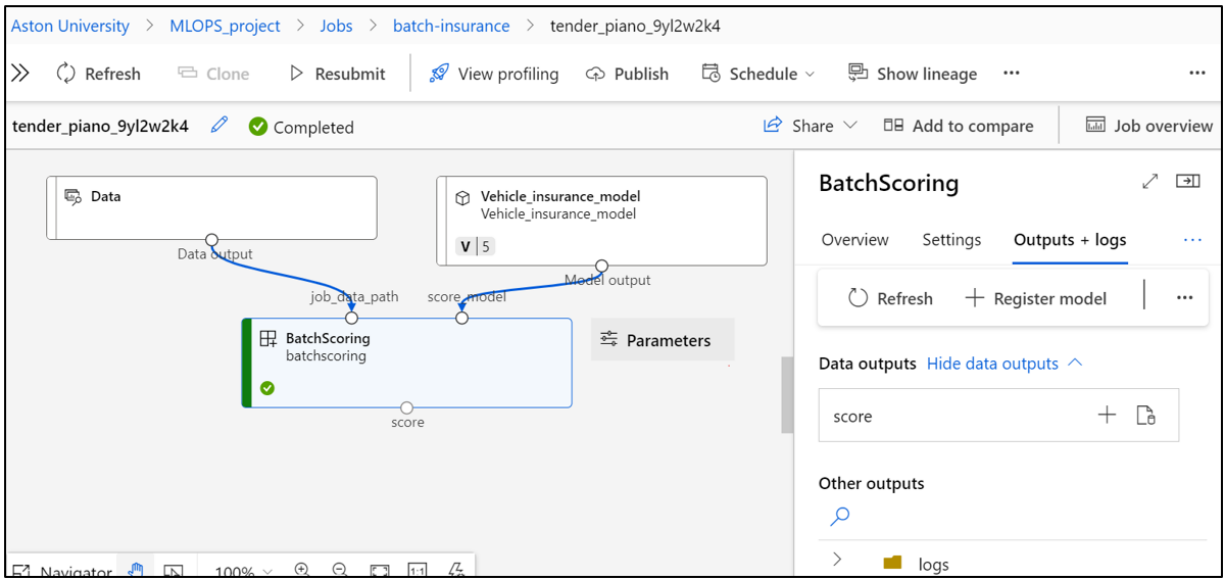


Figure 23: Successful scoring job using the deployed Vehicle_insurance_model

4.4 Results of MLOps Automation and CI/CD with GitHub action

The preceding result sections demonstrate the fully functional pipelines execution of MLOps culminating in the model utilisation for scoring in the AML endpoint. MLOps seeks to seamlessly integrate all these pipelines in a system that allows automatic execution of continuous (re)training, redeployment and improvement. This objective was achieved by the three GitHub workflows that were set up to run in the production environment. Figure 24 presents screenshots of the steps completed in each of the successful workflow runs.

The linting workflow primarily serves as a continuous integration process, ensuring that the code quality meets acceptable standards each time a pull request is created for newly committed code changes to be reviewed. The workflow will fail if the pushed code does not adhere to the Flake8 standards. The "PROD-Deploy_training_deployment" and "PROD-Scoring job" automate the release of the training_to_deployment pipeline and batch scoring job pipelines respectively, in the production environment as detailed in earlier Section 3.4.

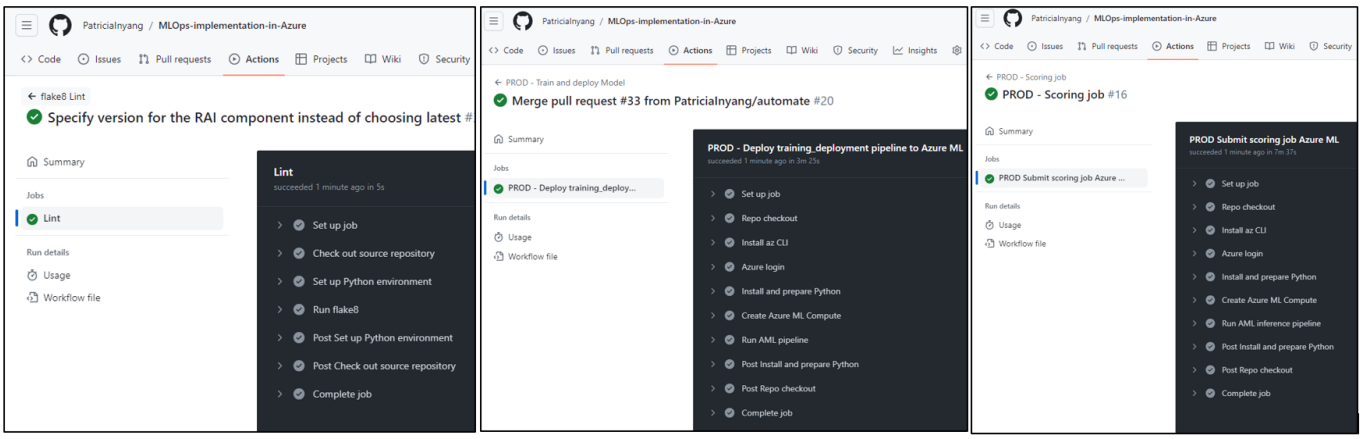


Figure 24: Successful runs of automated GitHub workflows for Azure MLOps pipelines

4.5 Summary discussion of project results

The results were initially presented in phases using same breakdown in the methodology subsections before demonstrating how CI/CD and automation were included to implement a functional MLOps systems for the cross-selling insurance case model. GitHub was effective in enabling collaboration and version control of the code artifacts in the project repository. Simultaneously, pipeline scripts, datasets and models were also versioned and tracked in Azure providing full traceability and maintaining audit trails. Azure proved to be a reliable governance and secure platform for hosting end-to-end MLOps by utilizing Azure credentials to establish connections with the AML workspace. The service principal identity enabled authentication between GitHub and the AML workspace, facilitating the execution of production environment workloads in non-interactive mode.

Considering that the primary focus of this project is not to create a high-performing model but to design and implement MLOps using Azure Machine Learning services, the research successfully achieved its goals. Comparing the results of this project with the MLOps maturity level described in Table 1, it is clear that the MLOps implemented in the research project attained the 3^{rd} maturity level.

5 CONCLUSIONS

This dissertation worked on overcoming the challenges involved in operationalising machine learning models by implementing an end-to-end MLOps in Azure ML. It progressed beyond the experimentation phase which produced a trained Logistic regression model for predicting responses (interested or not interested), from existing health insurance policyholders regarding the insurer's vehicle insurance offer. The model's 85.9% Recall score deemed it suitable for deployment since it identifies significant True positive cases that are most crucial for the Insurance company. Version 2 of AML Python SDK and CLI were utilized to build modular and reproducible scripts for executing the steps in the machine learning life cycle. The individual scripts were configured as AML components and linked together using pipelines resulting in a Model training to deployment pipeline, RAI dashboard pipeline and a scoring job pipeline for submitting batch inference requests to the deployed model. Finally, the entire MLOps process was automated for execution in a production environment that was configured with Azure service principal credentials. This automation achieved through GitHub action workflows, simultaneously facilitates continuous integration and continuous delivery for ongoing model improvements.

Despite the successful MLOps infrastructure built in this project, it is important to note that it currently operates at the 3rd maturity level. This is still far from the fully matured MLOps outlined in Table 1, which involves continuous operation, handling of real-life data and ongoing monitoring of production workloads to detect issues like data and model drifts. The subsequent sections in this concluding chapter will highlight the limitations of the results achieved in this research and suggest recommendations for future works.

5.1 Project Limitations

The limitations discussed below are the factors that impacted the findings from this research and the constraints to be aware of when using the designed MLOps system as a foundation for other machine learning models.

1. **Time Limitation:** The three-month period from when this project was initiated in October 2023 and the deadline for submission was insufficient to achieve a fully matured 4^{th} level MLOps which executes the continuous monitoring step. Also, the true reliability of the implemented MLOps requires running over time and monitoring the production metrics which should also trigger retraining, redeployment and model improvements.

2. **Limited robust testing:** Comparing Azure recommended MLOps structure in Figure 4 with the implemented MLOps flow diagram in Figure 13, reveals that the model deployment endpoint was run directly in the production environment. Best practices require testing the deployments by first operating it in a staging environment which mimics the production environment. A contributing factor was the fact that the Azure student subscription had restricted access to register the application needed to create the Service identity in Azure. The solve request raised for the Aston digital team to create the SP took about two weeks to be resolved, further affecting the timeline for testing in the staging environment before approval for production. Additionally, while the smoke testing method employed throughout this project confirmed the functionality of the pipelines and scripts, ideally, it should serve as a preliminary test. Subsequent, more robust tests, such as unit tests and User Acceptance Tests (UAT), are essential for uncovering vulnerabilities in the system

3. **Absence of continuous real-life generated data:** Although the cross-selling data that was used for training and evaluating the model represents a real-life scenario, it is not sufficient for retraining of new model ver-

sions. Hence, although the retraining and redeployment pipeline is confirmed to be triggered by the automated workflows when changes are pushed to the repository main branch, it is actually reproducing same model.

4. **Model performance:** The performance of the currently deployed Logistic regression model is not good enough, especially with an overall accuracy which indicates that the model misclassifies a lot of the customers. Since a high-performing model was not the focus of this project and recall was emphasized as the key metric for the model evaluation, the model qualified for deployment. This model is considered to be a limitation because there is need for future improvements to train and deploy an optimal performing model.

5. **RAI dashboard pipeline:** The script for the RAI dashboard construction pipeline does not currently run automatically. While the idea of configuring Azure Event Hub to trigger the RAI pipeline upon the registration of a new model to the Azure Machine Learning (AML) workspace was proposed, it has not been implemented. Instead, the script has been deployed as a pipeline component to a batch endpoint, necessitating manual human-in-the-loop invocation to run the pipeline when needed.

5.2 Recommendations for research continuation

MLOps is still a relatively new field with more areas to explore, hence the recommendations below for future works:

1. **Explore Azure DevOps:** The full end-to-end MLops can be achieved by using various DevOps tools provided by Microsoft in Azure instead of GitHub. This will make the system to be even more seamlessly integrated, with functional tools like Azure data lake, Azure pipelines Azure event hub, and Azure Boards.

2. Utilize MLFlow for more of the model training and experimentation.

3. **Develop a self-managed Build Your Own (BYOD) container:** instead of using Azure managed endpoints like the managed batch endpoint used in this project. This allows for more flexibility in making the MLOps infrastructure custom based for an organisations machine learning models.

4. **Writing of tests:** Implement a comprehensive testing strategy for the MLOps pipeline. This involves creating unit tests, integration tests, and potentially user acceptance tests (UAT) to ensure the reliability, robustness, and effectiveness of the MLOps system.

References

[1] Raj, E. (2021): Engineering MLOps: Rapidly build, test, and manage production-ready machine learning life cycles at scale. Packt Publishing Ltd, ISBN 978-1-80056-288-2 pg 12-13, 18-24

[2] IBM (2022): IBM Global AI Adoption Index 2022 New research commissioned by IBM in partnership with Morning Consult. [online] Available at: https://www.ibm.com/downloads/cas/GVAGA3JP.

[3] Thormundsson Bergur. (2023). Machine learning - statistics and facts. Statista website:https://www.statista.com/topics/9583/machine-learning/#editorsPicks

[4] Ana De Las Heras, Amalia Luque-Sendra, and Francisco Zamora-Polo. (2020): Machine learning technologies for sustainability in smart cities in the post-covid era.

[5] Treveil, M., et al (2020): Introducing MLOps. 'O'Reilly Media, Inc.', ISBN 978-1-492-08329-0, pp 23-65.

[6] DataRobot. (n.d.): *MLOps 101: The Foundation for Your AI Strategy.* Retrieved from https://www.datarobot.com/wp-content/uploads/2023/08/DataRobot_MLOps_101-_The_Foundation_for_Your_AI_Strategy_eBook.pdf

[7] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo (2015): Dimensions of DevOps. In International conference on agile software development, pages 212–217. Springer.

[8] Madhavaiah, C. and Bashir, I. (2012): Defining Cloud Computing in Business Perspective: A Review of Research. Metamorphosis: A Journal of Management Research, 11(2), pp.50–65. doi:https://doi.org/10.1177/0972622520120205.

[9] : Chip Huyen (2022): Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications, 1st Edition, 'O'Reilly Media, Inc, ISBN 978-1098107963, pages 25-35.

[10] Pathak, A. (2022): Learn MLOps with these 10 Courses.Geekflare. https://geekflare.com/mlops-courses

[11] Faubel, L., Schmid, K. and Eichelberger, H. (2023): MLOps Challenges in Industry 4.0. SN Computer Science, 4(6).Doi: https://doi.org/10.1007/s42979-023-02282-2.

[12] Kordon, A.K. (2020): The Model Deployment Life Cycle. Springer eBooks. https://doi.org/10.1007/978-3-030-36375-8_11 pp 315-318.

[13] Romero, O.; Wrembel, R.; Song, I.Y.(2020): An Alternative View on Data Processing Pipelines from the DOLAP 2019 Perspective. J. Inf. Syst., 92, 101489.

[14] Gartner. (2022): Gartner Predicts Half of Finance AI Projects Will Be Delayed or Cancelled By 2024. Available at: https://www.gartner.com/en/newsroom/press-releases/2022-06-07-gartner-predicts-half-of-finance-ai-projects-will-be-delayed-or-cancelled-by-2024.

[15] Sculley, D., et al.(2015): Hidden technical debt in machine learning systems. In: NIPS, pp. 2494–2502

[16] Kreuzberger, D., Kühl, N. and Hirschl, S. (2022). Machine Learning Operations (MLOps): Overview, Definition, and Architecture. Available at: https://arxiv.org/abs/2205.02302.

[17] Google Cloud. (2020): MLOps: Continuous delivery and automation pipelines in machine learning. Retrieved from https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[18] AWS (n.d.;. What is MLOps? - Machine Learning Operations Explained - AWS. Available at: https://aws.amazon.com/what-is/mlops/.

[19] Microsoft (2023): MLOps 101 - Microsoft Solutions Playbook. Available at: https://playbook.microsoft.com/code-with-mlops/technology-guidance/mlops/#standardize-on-repeatable-architectural-patterns.

[20] IBM (n.d.): Automation, IBM Data Science Best Practices. Available at: https://ibm.GitHub.io/data-science-best-practices/automation.html.

[21] N. Hewage, D. Meedeniya (2022): Machine Learning Operations: A Survey on MLOps Tool Support, pp. 1-12, Arxiv.2202.10169. DOI: https://doi.org/10.48550/arXiv.2202.10169

[22] Ruf, P., Madan, M., Reich, C. and Ould-Abdeslam, D. (2021): Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools. Applied Sciences, 11(19), p.8861. doi:https://doi.org/10.3390/app11198861.

[23] Brian Fitzgerald and Klaas-Jan Stol (2014): Continuous software engineering and beyond: trends and challenges. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pages 1–9.

[24] Brian Fitzgerald and Klaas-Jan Stol (2017). Continuous software engineering:A roadmap and agenda. Journal of Systems and Software, 123:176–189.

[25] Mäkinen, S., Skogström, H., Turku, V., Laaksonen, E. and Mikkonen, T. (2021): Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help? [online] Available at: https://arxiv.org/pdf/2103.08942.pdf.

[26] IBM Data and AI Team (2023). MLOps and the evolution of data science. Available at: https://www.ibm.com/blog/mlops-and-the-evolution-of-data-science/.

[27] Singh, P. (2021): Model Deployment and Challenges. In: Deploy Machine Learning Models to Production. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6546-8_2

[28] Martin Ekuan (nd): Machine Learning operations maturity model - Azure Architecture Center. Available at: https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/mlops-maturity-model.

[29] Amazon Web Services (2020): MLOps- Continuous Delivery for Machine Learning on AWS. https://d1.awsstatic.com/whitepapers/mlops-continuous-delivery-machine-learning-on-aws.pdf

[30] John, M.M., Olsson, H.H. and Bosch, J. (2021: Towards MLOps: A Framework and Maturity Model. IEEE Xplore. doi:https://doi.org/10.1109/SEAA53835.2021.00050.

[31] Ikram, A. and Tabassam, U. (2023): MLOps: A Step Forward to Enterprise Machine Learning. Available at: https://arxiv.org/ftp/arxiv/papers/2305/2305.19298.pdf.

[32] Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H. and Crnkovic, I. (2019): A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. Lecture Notes in Business Information Processing, (Vol. 355, p. 227–243). Springer International Publishing. https://doi.org/10.1007/978-3-030-19034-7_14.

[33] Poduska, J. (2022): The Seven Stages of MLOps Maturity. [online] Medium. Available at: https://towardsdatascience.com/the-seven-stages-of-mlops-maturity-ccb029530f2.

[34] Hayes, B. (2008): Cloud computing. Communications of the ACM, 51(7), p.9. doi:https://doi.org/10.1145/1364782.1364786.

[35] Collier, M. and Shahan, R. (2015): Fundamentals of Azure Microsoft Azure Essentials. ISBN: 978-0-7356-9722-5 pp 17-20, PUBLISHED BY Microsoft Press, Available at: https://scadahacker.com/library/Documents/eBooks/MS%20Press%20-%20Fundamentals%20of%20Azure.pdf.

[36] Tamburri, D.A. (2020): Sustainable MLOps: Trends and Challenges — IEEE Conference Publication — IEEE Xplore, Available at: https://ieeexplore.ieee.org/abstract/document/9356947/

[37] Mayumi, B. and Denise Hideko Goya (2022): MLOps:A Guide to its Adoption in the Context of Responsible AI. Association for Computing Machinery, ACM ISBN 978-1-4503-9319-5/22/05. doi: https://doi.org/10.1145/3526073.3527591.

[38] Ratilainen, K.-M. (2023): Adopting Machine Learning Pipeline in Existing Environment. [online] Available at: https://helda.helsinki.fi/server/api/core/bitstreams/1ccd55c5-399b-4e6e-9927-afb9a9cde0a2/content.

[39] sdonohoo (2023): Machine learning operations (MLOps) v2 - Azure Architecture Center. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/machine-learning-operations-v2.

[40] Zeitgeist (2023): AI Readiness Report. Available at: https://go.scale.com/hubfs/Scale-Zeitgeist-AI-Readiness-Report-2023.pdf.

[41] Sriram, V., Fan, Z. and Liu, N. (2022): ECLIPSE: Holistic AI System for Preparing Insurer Policy Data. Risks, 11(1), p.4. doi: https://doi.org/10.3390/risks11010004.

[42] Y. Choi, J. An, S. Ryu, and J. Kim, (2022): "Development and Evaluation of Machine Learning-Based High-Cost Prediction Model Using Health Check-Up Data by the National Health Insurance Service of Korea," Int J Environ Res Public Health, vol. 19, no. 20, doi: 10.3390/IJERPH192013672.

[43] R. D, M. S. K, and D. J,(2022): "Health Insurance Cost Prediction using Machine Learning Algorithms," in International Conference on Edge Computing and Applications,ICECAA, Institute of Electrical and Electronics Engineers (IEEE), pp.1381–1384. doi: 10.1109/ICECAA55415.2022.9936153.

[44] L. Xiaoqun and L. Run, (2022): "An Improved K-means Clustering Model Based onSupport Vector Machine for Health Insurance Cost Prediction," IEEE 2nd International Conference on Electronic Technology, Communication and Information, ICETCI 2022, pp. 521–525, doi: 10.1109/ICETCI55101.2022.9832085.

[45] S. Panda, B. Purkayastha, D. Das, M. Chakraborty, and S. K. Biswas, (2022): "Health Insurance Cost Prediction Using Regression Models," in International Conference on Machine Learning, Big Data, Cloud and Parallel Computing, COM-IT-CON, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 168–173. doi:10.1109/COM-IT-CON54601.2022.9850653.

[46] S. Rawat, A. Rawat, D. Kumar, and A. S. Sabitha, (2021): "Application of machine learning and data visualization techniques for decision support in the insurance sector," International Journal of Information Management Data Insights, vol. 1, no. 2, p. 100012, doi: 10.1016/J.JJIMEI.2021.100012.

[47] Capegemini (2022); Top trends in insurance: 2022 eBook. Available at: https://www.capgemini.com/insights/research-library/top-trends-in-insurance-2022-ebook/

[48] Mckinsey (2022): The future of insurance: Creating value, finding focus — McKinsey. Available at: https://www.mckinsey.com/industries/financial-services/our-insights/the-future-of-insurance-creating-value-finding-focus.

[49] Silva, C., Brito, M.A. and Duarte, D.N. (2023): Machine Learning on Insurance Premium Prediction. https://doi.org/10.1145/3605423.3605450.

[50] Kamakura, W.A. (2008): Cross-Selling. Journal of Relationship Marketing, pp.41–58. :https://doi.org/10.1300/j366v06n03_03.

[51] Makalesi, A., Ozdemir, Y., Bayrakli, S., Üniversitesi, M., Harp, H., Dekanlığı, O., Bölümü, M., İstanbul, Y., Üzerine, O., Örnek, B. and Öz, Ç. (2022): A Case Study on Building a Cross-Selling Model through Machine Learning in the Insurance Industry. European Journal of Science and Technology, pp.364–372. doi:https://doi.org/10.31590/ejosat.895069.

[52] Anmol Kumar. (2018): Health Insurance Cross Sell Prediction, Available at: https://www.kaggle.com/datasets/anmolkumar/health-insurance-cross-sell-prediction/data.

[53] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, (2002): "SMOTE: Synthetic minority oversampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321-357.

[54] A. Estabrooks, T. Jo, and N. Japkowicz, (2004):"A multiple resampling method for learning from imbalanced data sets," Computational Intelligence, vol. 20, no. 1, 18-36.

[55] Garreta, R., Moncecchi, G., Hauck, T. and Hackeling, G. (2017). scikit-learn : Machine Learning Simplified: Implement scikit-learn into every step of the data science pipeline. Packt Publishing Ltd, ISBN 978-1-84719-752-8, pages 119-121.

[56] Microsoft learn (n.d.). Exam DP-100: Designing and Implementing a Data Science Solution on Azure - Certifications. Available at: https://learn.microsoft.com/en-us/credentials/certifications/exams/dp-100/.

[57] minthigpen (2023): Generate a Responsible AI insights with YAML and Python - Azure Machine Learning. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-responsible-ai-insights-sdk-cli?view=azureml-api-2&tabs=yaml#input-constraints.

A APPENDIX

All source codes and related files for this project as explained in the body of the report are accessible in the project GitHub repository in the link below: https://github.com/PatriciaInyang/MLOps-implementation-in-Azure

A.1 Appendices for Methodology

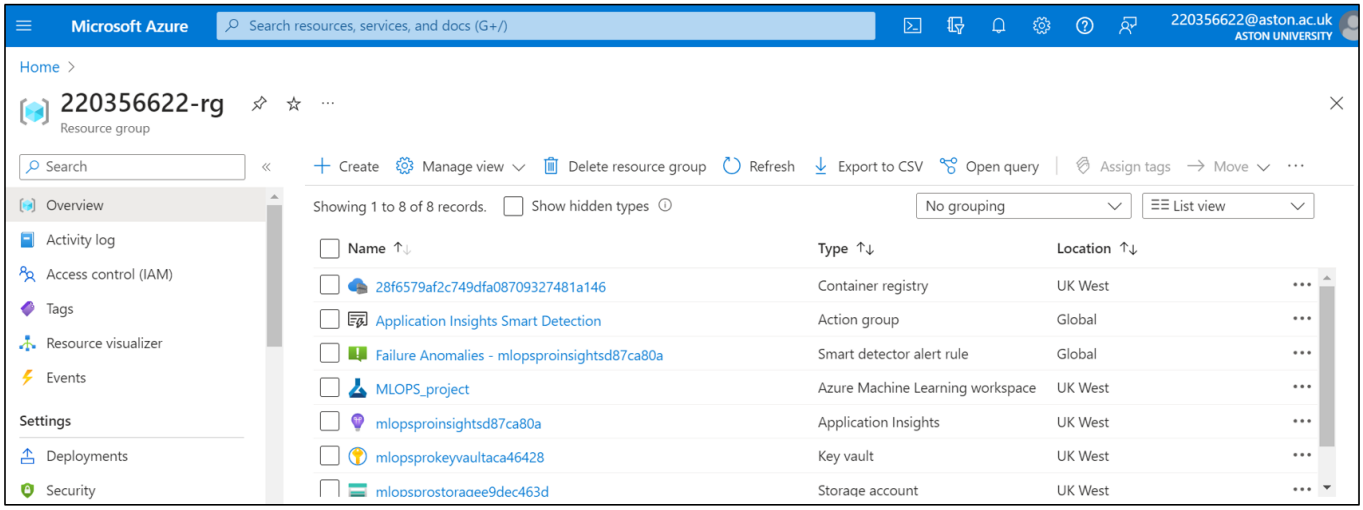


Figure A.1.1: Screenshot of the Resource group hosting all the resources for the project

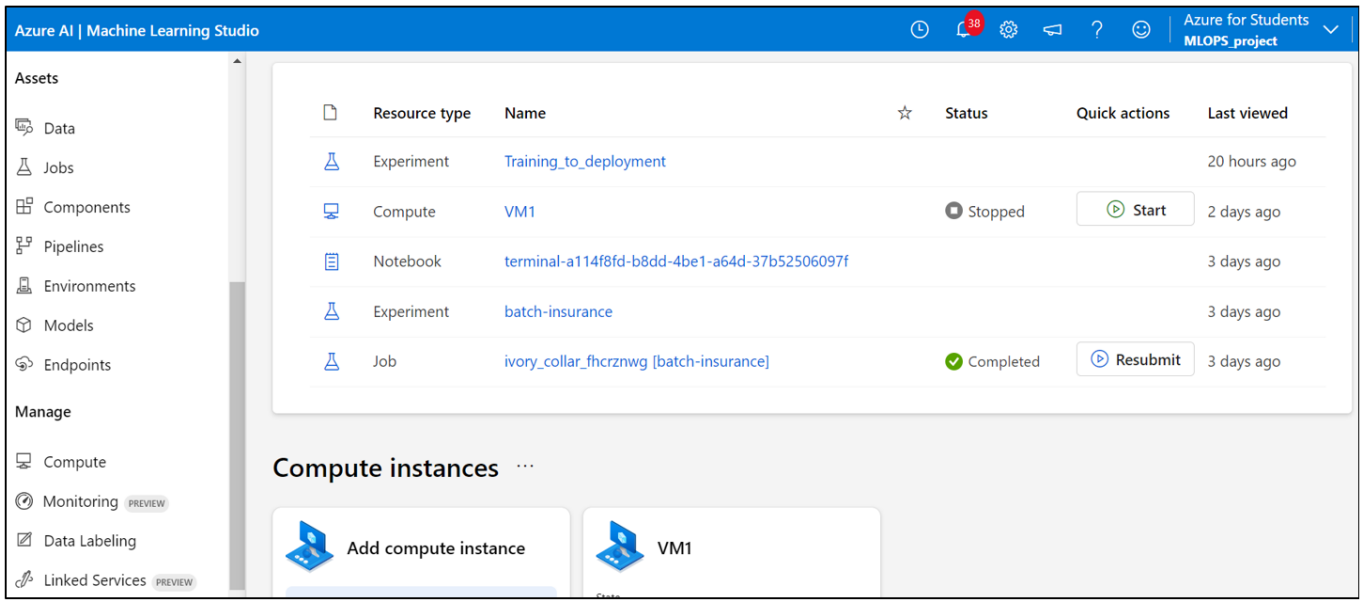


Figure A.1.2: Screenshot of the MLOps_project AML workspace

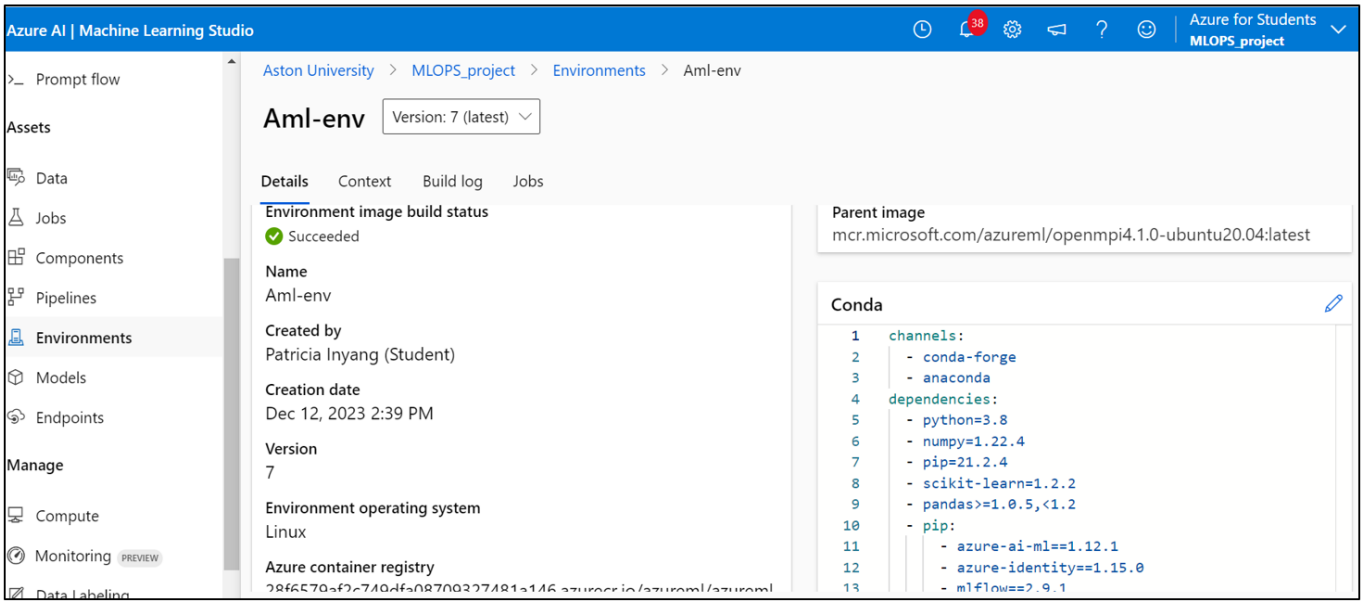


Figure A.1.3: Screenshot of the project environment setup in AML workspace

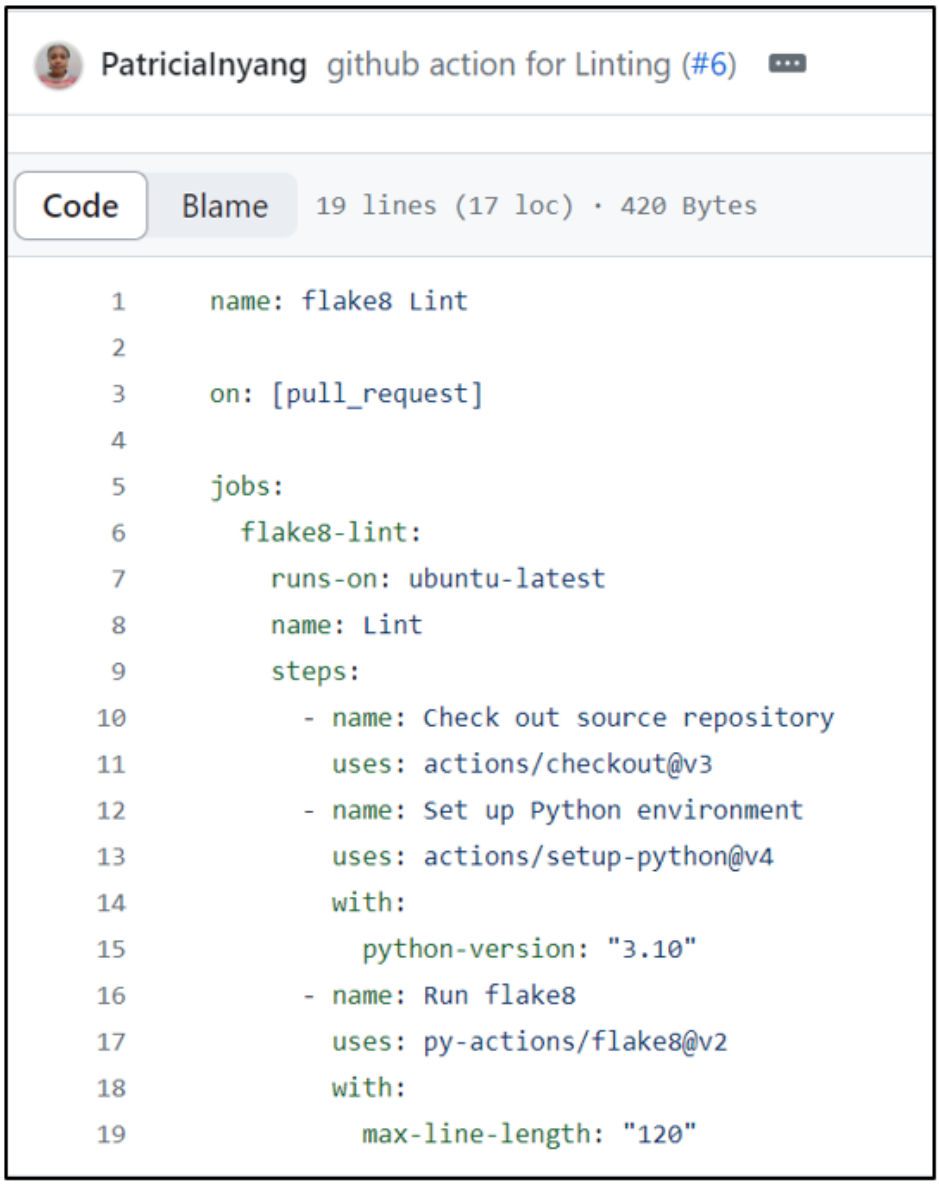


Figure A.1.4: Screenshot of GitHub action workflow for linting repository code

main MLOps-implementation-in-Azure / src / model_training_to_deployment / deploy_model.py

Blame 117 lines (89 loc) · 4.56 KB Code 55% faster with GitHub Copilot Raw

```python
def create_batch_deployment(deployment_name, description, model, env, ml_client):
    deployment = BatchDeployment(
        name=deployment_name,
        description=description,
        endpoint_name='batch-insurance',
        model=model,
        code_configuration=CodeConfiguration(
            code="code",
            scoring_script="batch_driver.py"
        ),
        compute="VM1",
        environment=env,
        instance_count=1,
        max_concurrency_per_instance=2,
        mini_batch_size=1,
        output_action=BatchDeploymentOutputAction.APPEND_ROW,
        output_file_name="predictions.csv",
        retry_settings=BatchRetrySettings(max_retries=3, timeout=300),
        logging_level="info"
    )

    ml_client.batch_deployments.begin_create_or_update(deployment).result()
```

Figure A.1.5: Screenshot of configuration for deploying model to Batch process endpoint in Azure

```python
# Create components for the model training to deployment steps
parent_dir = "."
train_model_component = load_component(source=parent_dir + "/train.yml")
validate_model_component = load_component(source=parent_dir + "/validate.yml")
reg_model_component = load_component(source=parent_dir + "/model_reg.yml")
deploy_model_component = load_component(source=parent_dir + "/deploy_model.yml")

# Create the pipeline using the @pipeline function to automate the training to model deployment components
@pipeline()
def model_train_to_deployment(pipeline_job_input):
    # 1st step - Training and evaluate model
    train_model = train_model_component(input_data=pipeline_job_input)

    # 2nd step - Validate criteria to check if the model should be deployed
    validate_model = validate_model_component(metrics_file=train_model.outputs.metrics_file)

    # 3rd step - Register model and version it
    register_model = reg_model_component(model=train_model.outputs.model,
                                         metrics_file=train_model.outputs.metrics_file,
                                         validation_result=validate_model.outputs.validation_result,
                                         )
    # 4th step - Deploy model to batch-insurance endpoint and make it the default deployment
    deploy_model = deploy_model_component(model_reg_output=register_model.outputs.model_reg_output)

    return {"automated_model_training_to_deployment_pipeline": deploy_model.outputs.model_deploy_output}

# Define input data  using the data asset registered in the workspace
pipeline_job_input = Input(type=AssetTypes.URI_FILE, path=f"azureml:data:{Label}")
```

Figure A.1.6: Screenshot of components pipeline to Automate the model training to deployment

MLOps-implementation-in-Azure / src / RAI / RAI_dashboard.py

Code | Blame | 284 lines (216 loc) · 10.8 KB | Code 55% faster with GitHub Copilot | Raw

```python
def rai_decision_pipeline(target_column_name, train_data, test_data):

    # Create the RAI insights dashboard
    create_rai_job = rai_constructor_component(
        title="RAI dashboard insurance",
        task_type="classification",
        model_info=expected_model_id,
        model_input=Input(type=AssetTypes.MLFLOW_MODEL, path=azureml_model_id),
        train_dataset=train_data,
        test_dataset=test_data,
        categorical_column_names='["Region_Code", "Policy_Sales_Channel"]',
        target_column_name=target_column_name,
    )
    create_rai_job.set_limits(timeout=300)

    # Add error analysis
    error_job = rai_erroranalysis_component(rai_insights_dashboard=create_rai_job.outputs.rai_insights_dashboard
    error_job.set_limits(timeout=300)

    # Add explanations
    explanation_job = rai_explanation_component(
        rai_insights_dashboard=create_rai_job.outputs.rai_insights_dashboard,
        comment="add explanation"
        )
    explanation_job.set_limits(timeout=300)

    # Combine insights
    rai_gather_job = rai_gather_component(
        constructor=create_rai_job.outputs.rai_insights_dashboard,
        insight_3=error_job.outputs.error_analysis,
        insight_4=explanation_job.outputs.explanation,
    )
    rai_gather_job.set_limits(timeout=300)

    # Set dashboard mode to upload
    rai_gather_job.outputs.dashboard.mode = "upload"

    # Return the dashboard path
    return {"dashboard": rai_gather_job.outputs.dashboard}
```

Figure A.1.7: Screenshot of RAI dashboard pipeline for registered models

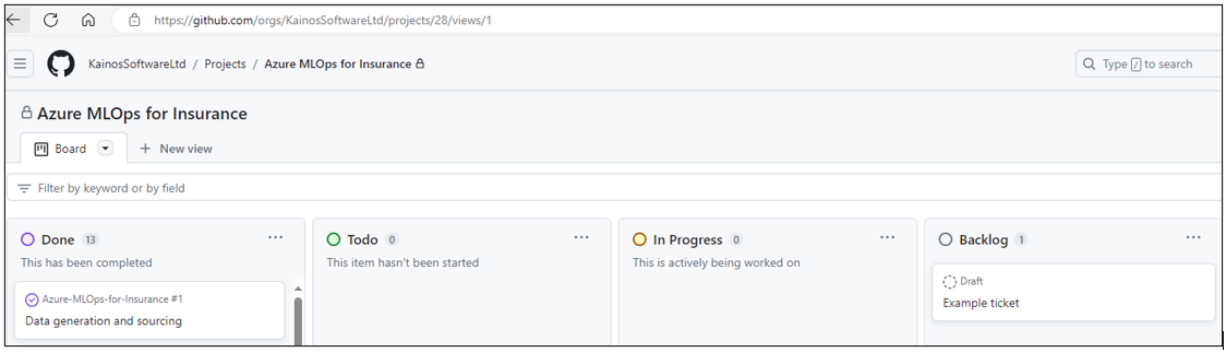


Figure A.1.8: Screenshot of project board organisation in GitHub

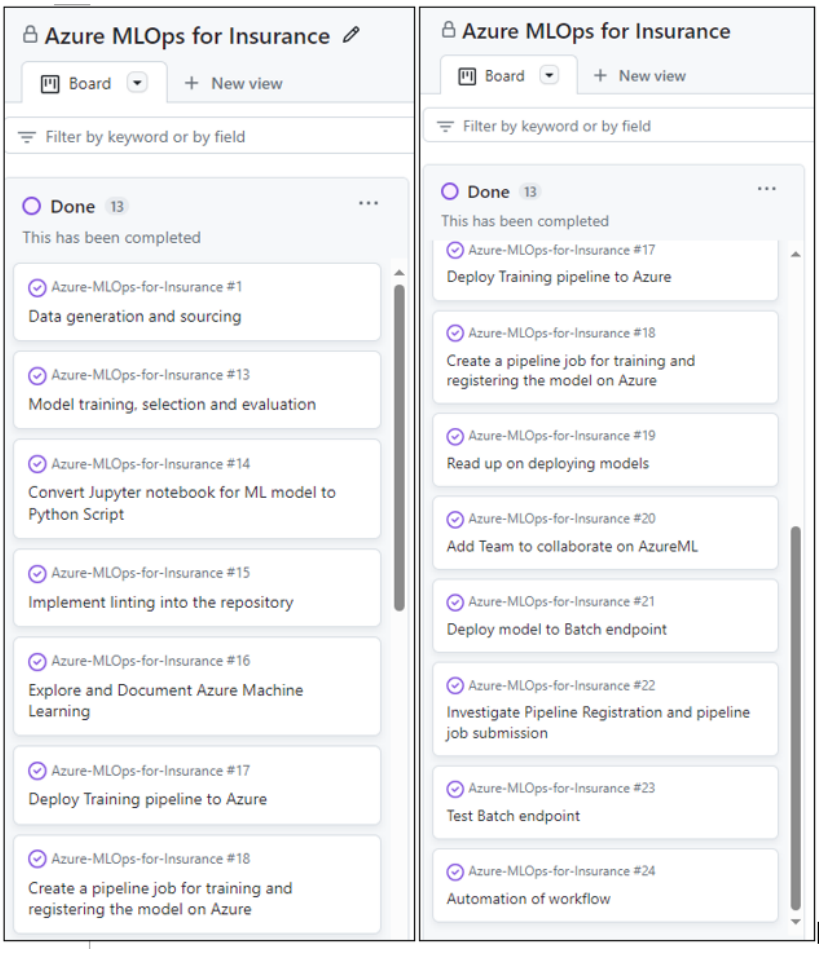


Figure A.1.9: A glimpse into the weekly tickets for project plans and tasks

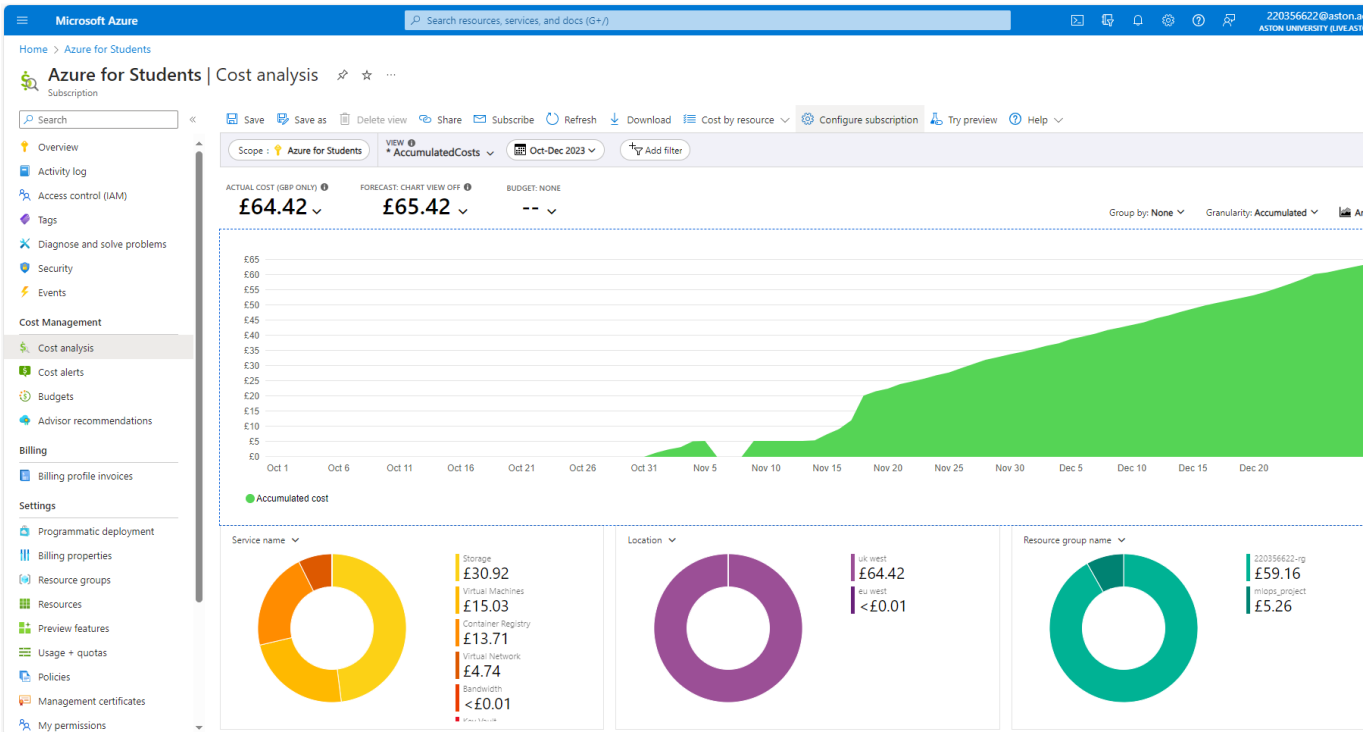


Figure A.1.10: Azure subscription Cost breakdown based on resources

A.2 Appendices for Results

This RAI insights results from the RAI dashboard, presented in this appendix, shows that the RAI pipeline works as expected but it does not reflect the true behaviour of the model because it used a sample of 5000 policyholders each from the 76,222 rows test dataset and 74,736 row train dataset.

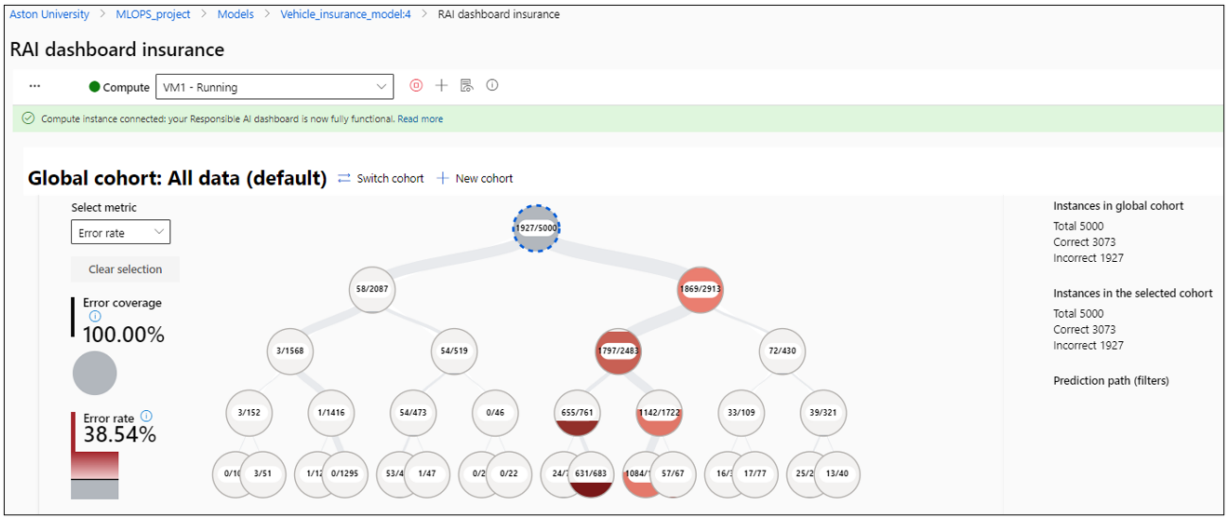


Figure A.2.1: Error rate insights from the Vehicle-insurance model RAI dashboard

To get the most interpretability from the error tree requires interaction with the live dashboard when the workspace compute is connected. Observe that the interconnecting lines are of varying thickness and they represent instances of the features used in generating the tree. High error rates are depicted in nodes with a stronger red color and a higher fill line (i.e., high error coverage). For example, hovering the mouse cursor on the bigger line on the right sub-tree in the AML live dashboard shows that the highest error rate (misclassification of policyholders) occurred in those with Policy_sales_channel ! = 152.0|160.0.

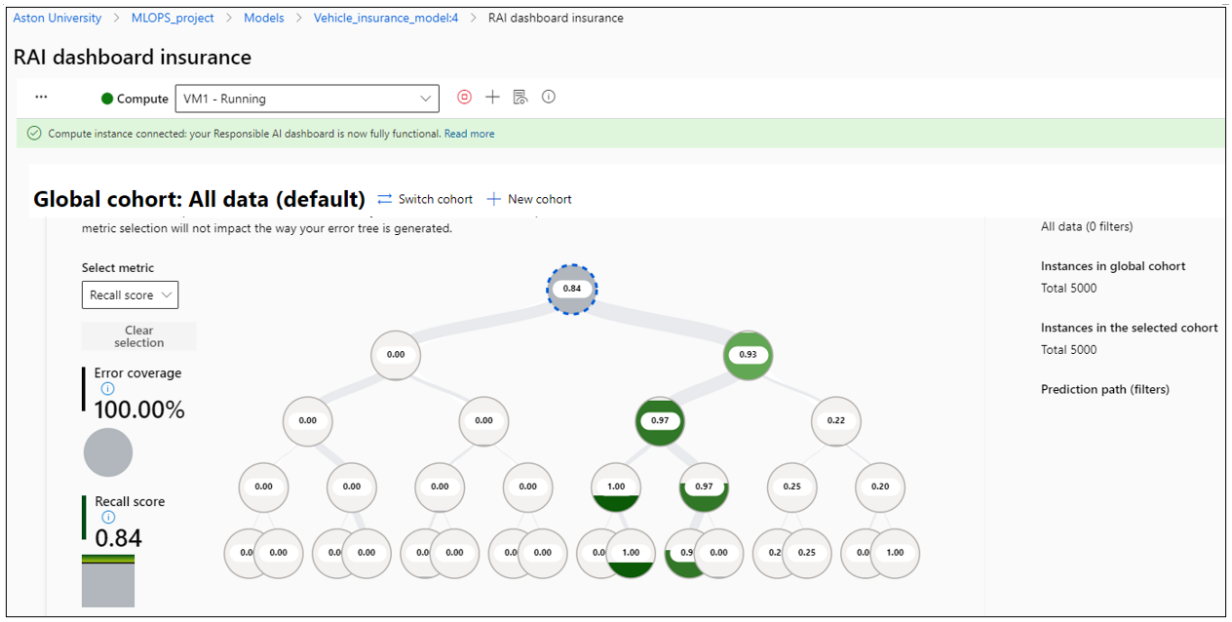


Figure A.2.2: Recall analysis from the Vehicle-insurance-model RAI dashboard

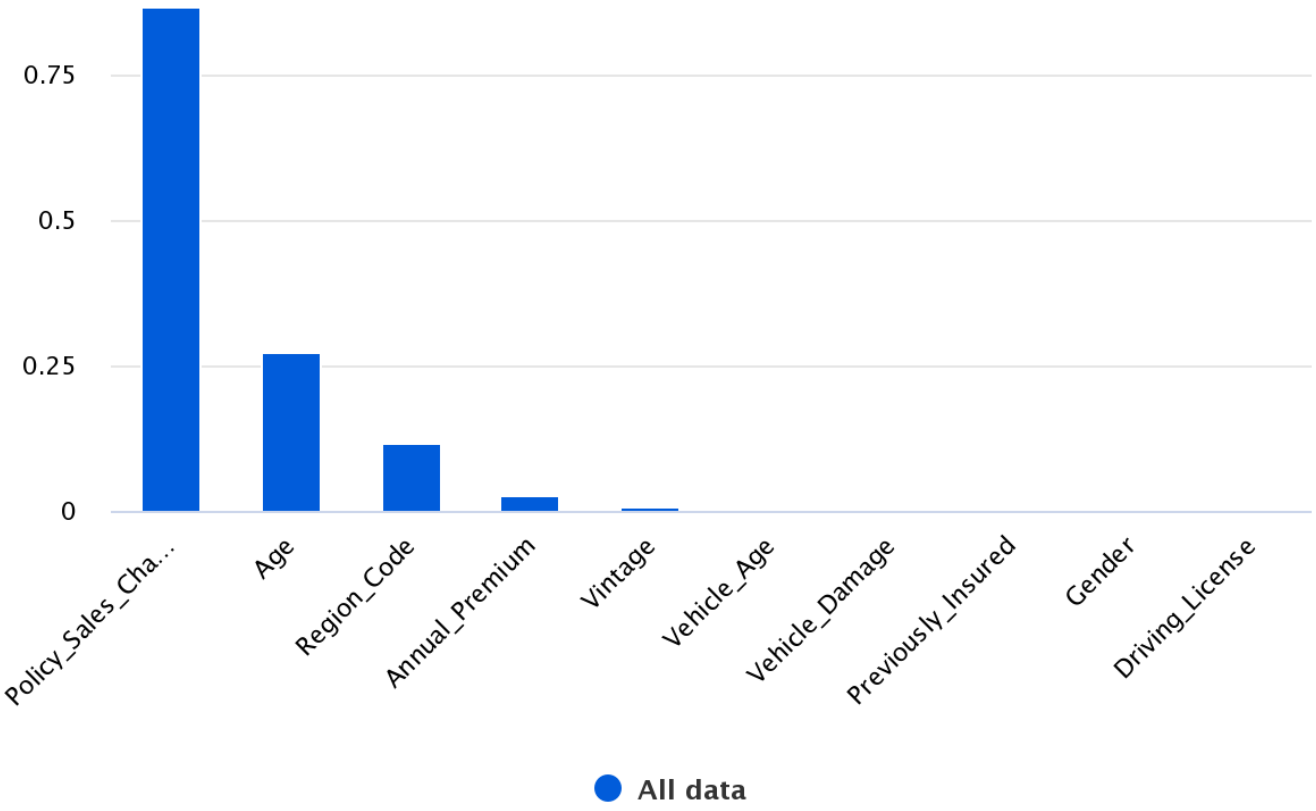


Figure A.2.3: Feature importance insights from the Vehicle-insurance-model RAI dashboard