1. Introduction

1.1 Background: The Azure Functions Dataset

The shift toward serverless computing, specifically Function-as-a-Service (FaaS), has transformed how organizations deploy and manage cloud applications. In this model, developers focus exclusively on writing code, while the cloud provider dynamically manages infrastructure, scaling resources automatically based on demand.

This project utilizes the Microsoft Azure Functions Trace 2019 dataset, released through Microsoft Research Open Data. The dataset contains production-level traces of serverless workloads, providing insight into execution behavior across thousands of anonymized functions over a continuous 24-hour period.

Specifically, this analysis focuses on the function_durations_percentiles subset, which records invocation frequency (Count), average execution duration (Average), and latency distribution statistics (percentile metrics).

1.2 Why This Problem Matters

In a serverless environment, performance volatility presents a significant operational challenge. Unlike traditional server environments, serverless functions may enter a dormant state and require re-initialization upon new requests — a phenomenon known as a cold start. This initialization introduces additional latency.

Performance: Elevated latency degrades end-user experience and may reduce application reliability.

Reliability: It is critical to evaluate whether infrastructure scaling behaves predictably under increased invocation load.

Cost Optimization: Azure Functions billing is based on execution duration (GB-seconds). Prolonged runtime during load spikes increases operational costs without proportional value creation.

1.3 Business Relevance

For business stakeholders, predictability is as important as speed. Service Level Agreements (SLAs) require consistent performance guarantees. In e-commerce and high traffic digital platforms, unexpected latency during peak demand can translate directly into lost revenue and customer churn.

Understanding the statistical relationship between invocation load and latency supports proactive capacity planning, cold-start mitigation strategies, and informed decisions regarding infrastructure tiers such as Consumption versus Premium plans.

1.4 Research Question

To what extent does function invocation load (Count) statistically explain variability in execution latency (Average Duration), and how accurately can latency be predicted using load-based statistical models?

1.5 Project Objective

The objective of this project is to construct an end-to-end statistical pipeline capable of predicting serverless execution latency.

In this analysis:
Average Duration (milliseconds) is treated as the response variable.
Invocation Count serves as the primary explanatory predictor.

Specifically, we aim to:

1. Ingest and validate production trace data.
2. Perform data cleaning and structural validation.
3. Conduct exploratory analysis to evaluate load–latency behavior.
4. Construct and compare regression-based models.
5. Evaluate model performance using RMSE and R-squared while assessing classical regression assumptions.

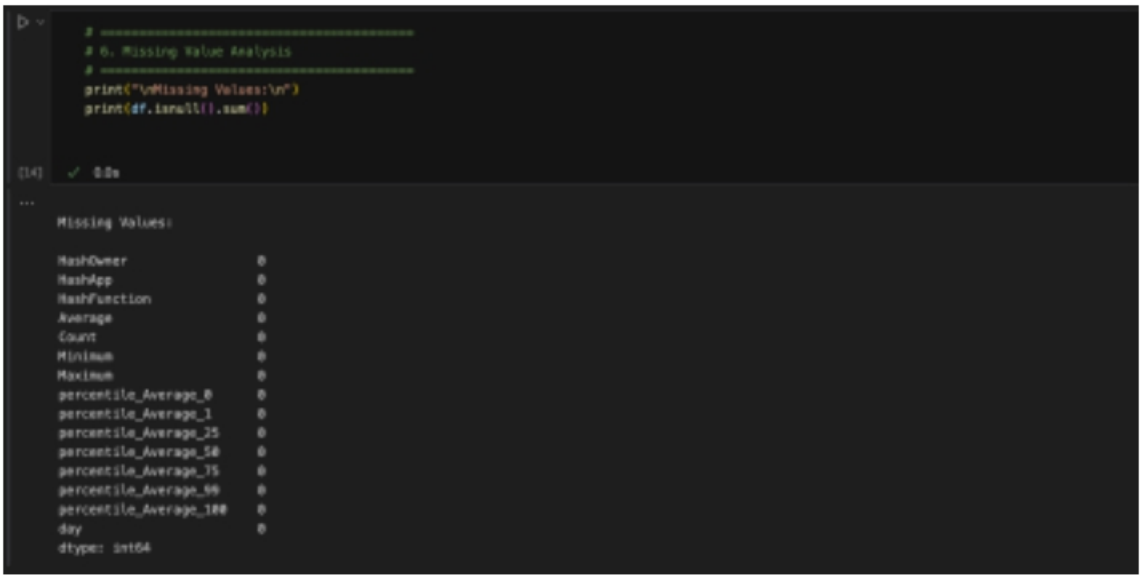
2. Data Cleaning and Preparation

2.1 Data Ingestion and Structural Validation

The dataset was imported into a Python-based analytical environment using pandas. Structural validation included verification of dataset dimensions (**99,999 observations and 15 variables**), confirmation of variable data types, and classification of numeric versus categorical fields.

2.2 Missing Value Assessment

A comprehensive missing value audit was conducted by computing both absolute and percentage missingness for each variable. No missing values were detected across the dataset, allowing complete-case analysis without imputation.



2.3 Negative Value Validation

During exploratory inspection, a small subset of observations exhibited negative execution duration values. Since latency cannot be negative in physical terms, these records were treated as measurement artifacts and removed prior to transformation.

After filtering negative values in the response variable and primary predictor, the dataset contained **99,994 valid observations.**

2.4 Outlier Evaluation

Outliers were examined using boxplots and distributional analysis. Given that serverless workloads naturally experience traffic bursts and performance spikes, extreme values were retained unless clearly invalid. This approach preserves realistic production variability essential for modeling.

2.5 Distribution and Skewness Analysis

Descriptive analysis revealed substantial right-skewness in both invocation load and execution duration. Skewness statistics confirmed heavy-tailed distributions.

To address variance instability and improve interpretability, a log(1 + x) transformation was applied for visualization and potential modeling refinement.

2.6 Prepared Dataset for Modeling

Following validation, negative-value removal, and distributional assessment, the dataset is structurally sound and statistically prepared for regression modeling. The preprocessing steps

preserve production realism while mitigating invalid artifacts.

3. Exploratory Data Analysis (EDA)

3.1 Dataset Overview

The cleaned Azure Functions dataset contains **99,999 observations** and **15 variables**, including invocation metrics, latency measures, and percentile-based performance indicators.

No missing values were detected across any variables, ensuring complete-case analysis without requiring imputation.

The primary variables of interest for this analysis are:

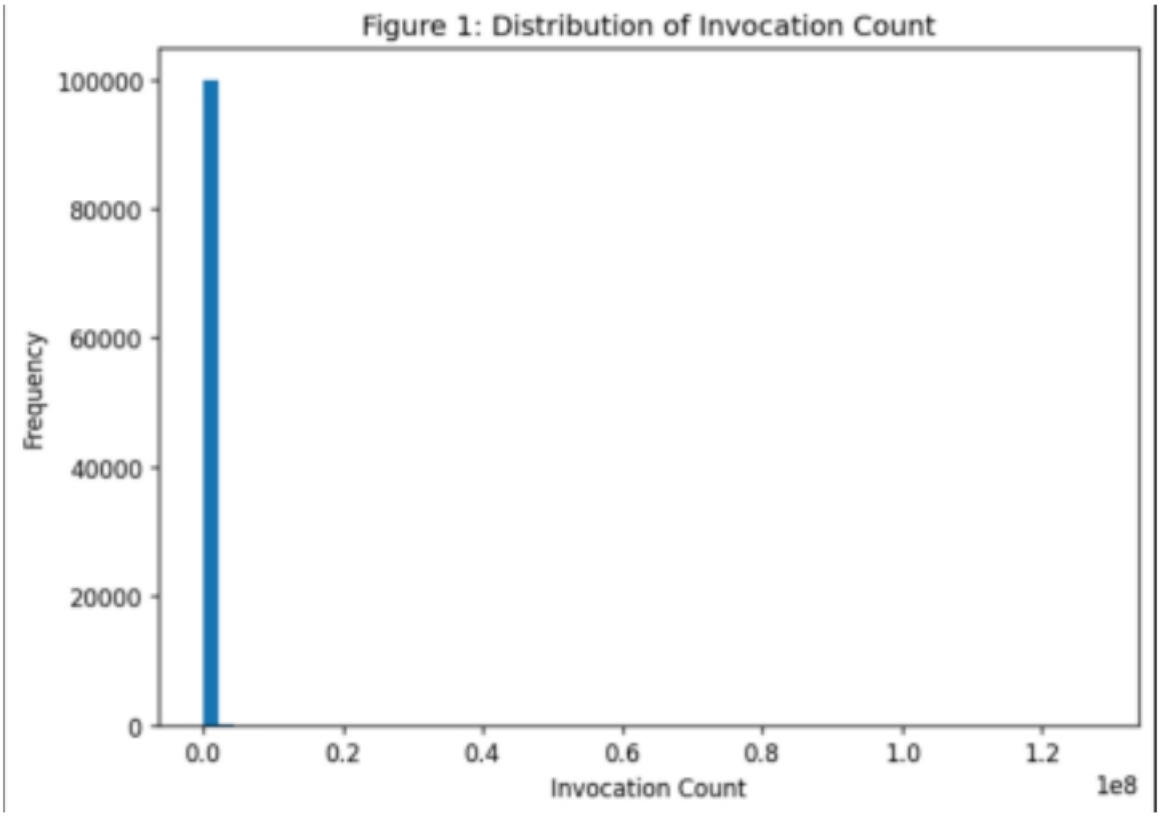
· Invocation Count (Count)
· Average Execution Duration (Average)

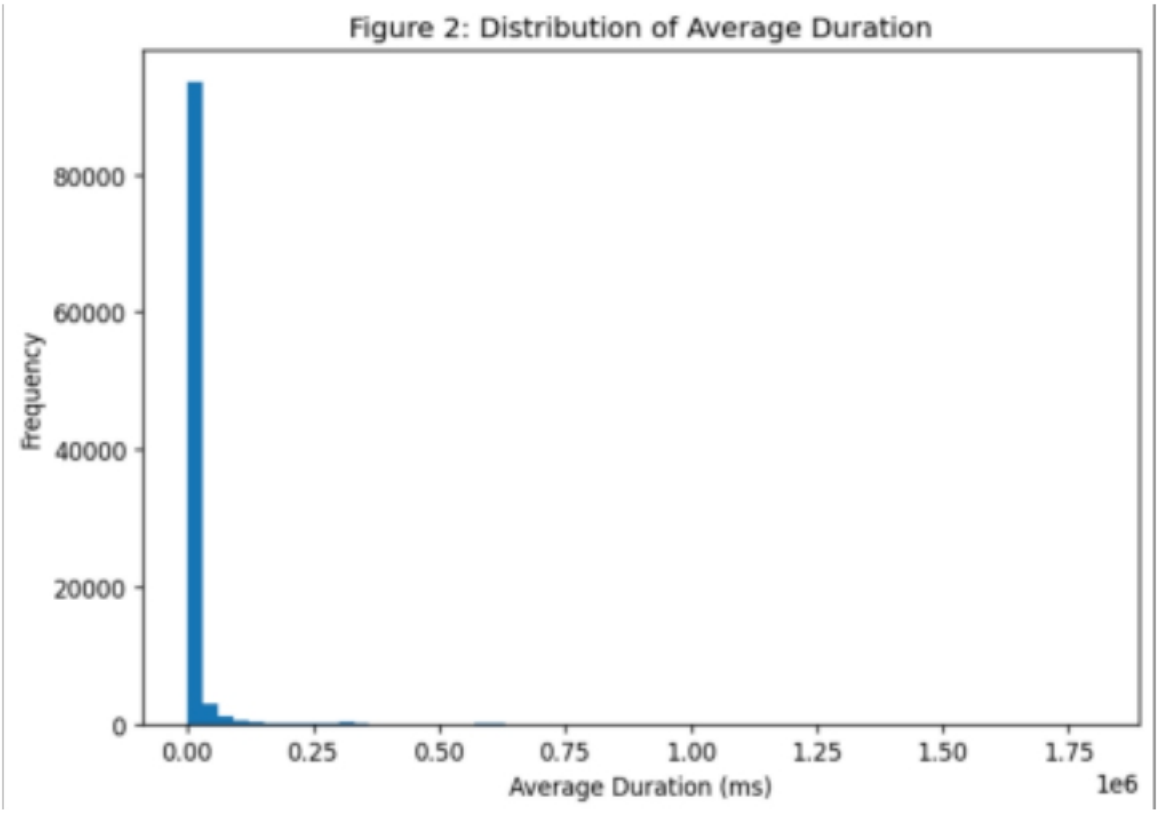
These variables form the basis of the load–latency investigation.

3.2 Descriptive Statistics and Distributional Properties

Initial summary statistics revealed substantial dispersion in both invocation load and execution duration.

Histograms (Figures 1 and 2) indicate that both variables exhibit **extreme right skewness**.





The computed skewness values were:

· Skewness (Count) = **106.57**

· Skewness (Average Duration) = **9.37**

```
# ========================================
# 10. Skewness
# ========================================
print("\nSkewness Values:")
print("Invocation Count:", df["Count"].skew())
print("Average Duration:", df["Average"].skew())
```

[18] ✓ 0.0s

```
...
Skewness Values:
Invocation Count: 106.56950863588486
Average Duration: 9.366679740070937
```

For reference, skewness values above 1 already indicate strong asymmetry. The observed skewness of 106 suggests a highly heavy-tailed workload distribution.

This implies that:

· Most Azure functions experience low invocation volumes.
· A small subset of functions handle extraordinarily high traffic spikes.

Similarly, execution duration demonstrates substantial right-tail behavior, indicating rare

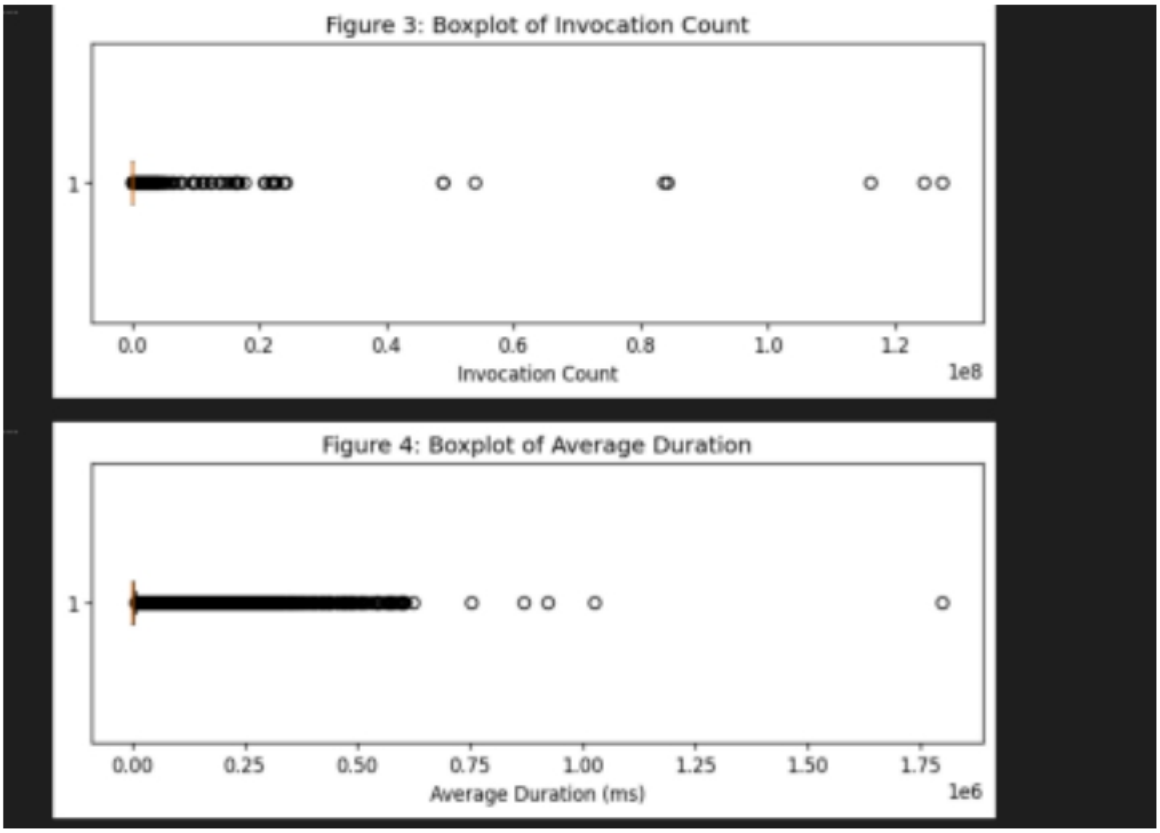
but extreme latency events.

This heavy-tailed distribution is characteristic of cloud production environments and suggests that linear modeling on the raw scale may violate normality and homoscedasticity assumptions.

3.3 Outlier Analysis

Boxplots (Figures 3 and 4) further confirm the presence of extreme outliers in both invocation count and execution duration.

These outliers represent legitimate production spikes rather than data entry errors and therefore were retained in the dataset.



Given the operational nature of the workload, removing these values would artificially distort real-world system behavior.

However, their presence reinforces the need for transformation during modeling.

3.4 Load–Latency Relationship

A scatterplot of Invocation Count versus Average Duration (Figure 5) was constructed to assess the visual association between load and latency.

Surprisingly, the Pearson correlation coefficient was:

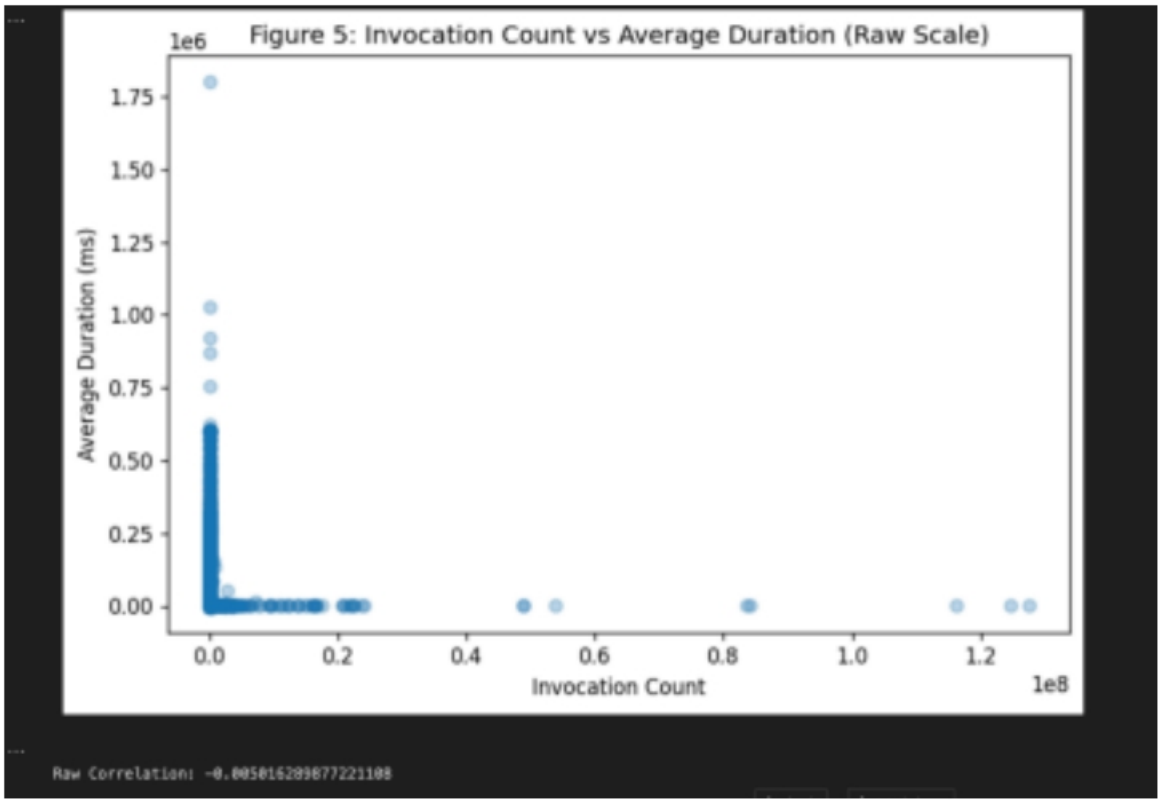
Correlation = **−0.005**

This value indicates **virtually no linear association** between invocation load and average

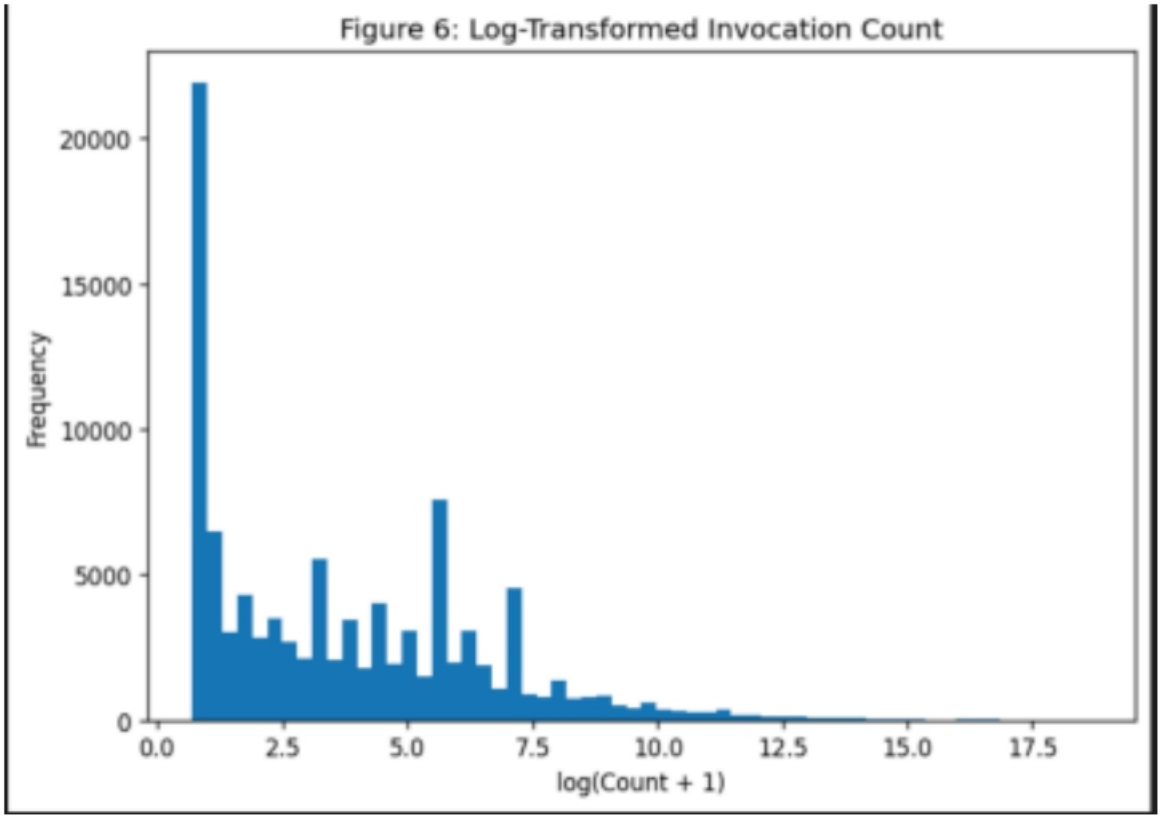
latency.

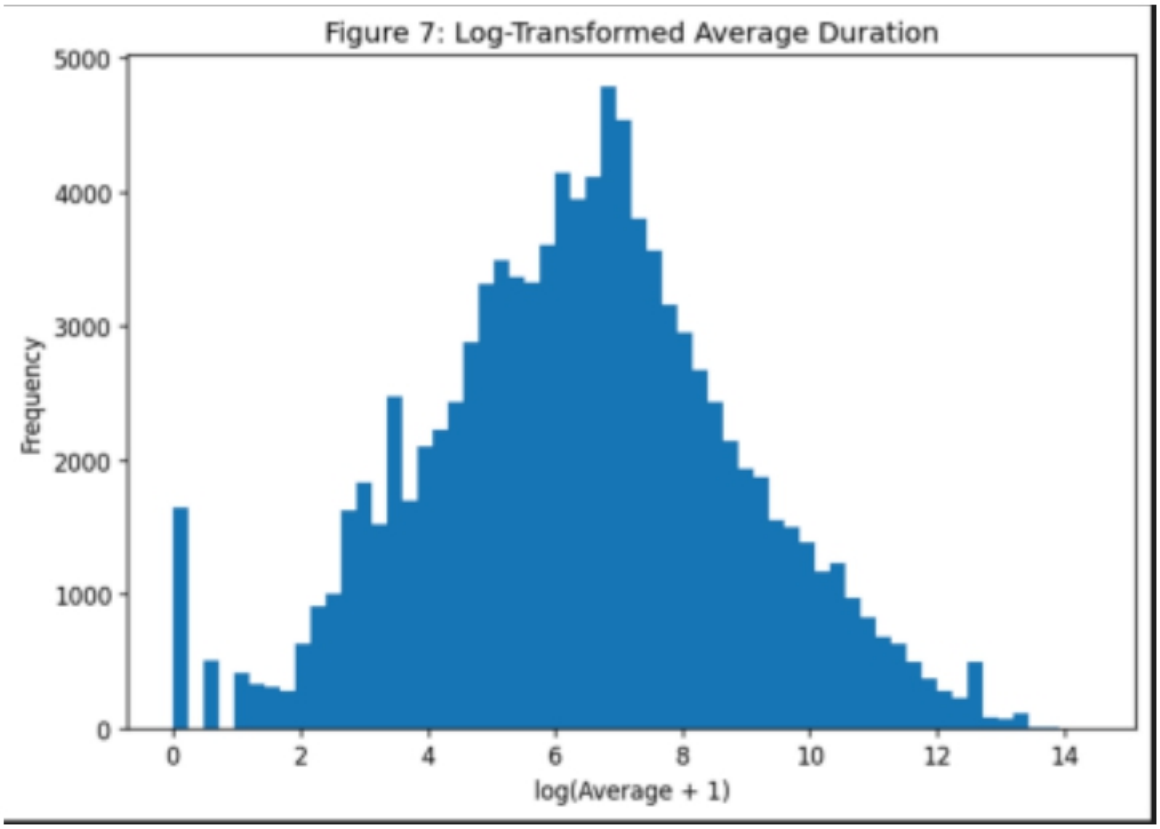
This finding suggests:

· Latency is not linearly explained by load alone.
· Other factors (e.g., cold starts, infrastructure allocation, concurrency limits) may influence performance.
· Nonlinear relationships may be present.



Importantly, the scatterplot also reveals increasing dispersion at higher load levels, suggesting heteroscedasticity — where variance of latency increases under heavy traffic conditions.





3.5 Log-Transformation Justification

Given the extreme skewness and heteroscedastic patterns, log-transformation was applied to both variables for visualization purposes.

The log-transformed histograms (Figures 6 and 7) display substantially improved distribution symmetry, making structural patterns more interpretable.

This transformation supports:

· Variance stabilization
· Improved model interpretability
· Better alignment with regression assumptions

3.6 Key EDA Insights

The exploratory analysis yields several important conclusions:

1. Azure workloads exhibit extreme heavy-tailed behavior.
2. Invocation load alone does not demonstrate a simple linear relationship with latency.
3. Execution duration variability increases under high load.
4. Raw-scale modeling would likely violate classical regression assumptions.
5. Log-transformed or nonlinear models may provide superior predictive performance.

These findings directly motivate the modeling strategy presented in the next section.

4. Model Selection and Building

4.1 Feature Selection and the "Data Leakage" Phenomenon

During initial model experimentation, a multivariate regression model was constructed utilizing *Minimum* and *Maximum* execution durations alongside Invocation *Count* as predictors for the *Average* latency. As expected, this multivariate model yielded an artificially high performance metric (RMSE = 0.6521, $R^2 = 0.9311$).

However, the team identified this as a classic case of data leakage (target leakage). Because the *Average* duration is mathematically bounded by and aggregated from the exact same execution pool as the *Minimum* and *Maximum* variables, providing these features to the model essentially "gives it the answer key" before it makes a prediction. In a real-world production environment, a cloud architect would not know the maximum latency of a traffic spike until after the spike has concluded, rendering such a model useless for proactive scaling.

To maintain the scientific integrity of this analysis, *Minimum* and *Maximum* were intentionally excluded from our final feature set. The remaining models (Linear

Regression and Random Forest) were strictly constrained to use only Invocation *Count* (Load) to predict *Average* duration (Latency), ensuring a robust and realistic evaluation of serverless behavior.

4.2 Modeling Strategy

Following the insights derived from the exploratory data analysis, the modeling strategy focuses on evaluating the predictive power of Invocation Count over Average Duration. Given the extreme right-skewness and heavy-tailed nature of the raw data , all models were trained utilizing the log(1+x) transformed variables.

To rigorously test our hypotheses, we selected two contrasting statistical architectures:

Baseline Model (Linear Regression): A parametric approach to test for a direct, proportional relationship between load and latency.

Challenger Model (Random Forest Regressor): A non-parametric, ensemble tree approach chosen to capture the nonlinear dynamics and heteroscedasticity observed under high load conditions.

4.2 Data Preparation for Modeling

Prior to training, the dataset was partitioned into a training set (80%) and a testing set (20%) using random sampling. This hold-out method ensures that the models are evaluated on unseen data, providing an unbiased estimate of generalization error. The response variable was defined as the log-transformed Average Duration, while the explanatory predictor was the log-transformed Invocation Count.

4.3 Model Implementation

The Linear Regression model was fitted using Ordinary Least Squares (OLS) to establish a performance floor. Subsequently, the Random Forest Regressor was implemented using an ensemble of 50 decision trees. The maximum depth of the trees was constrained to 10 to prevent overfitting to the extreme outliers retained in the dataset.

```
# 4. Train Baseline Model: Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)

# 5. Train Challenger Model: Random Forest
rf = RandomForestRegressor(n_estimators=50, max_depth=10, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
```

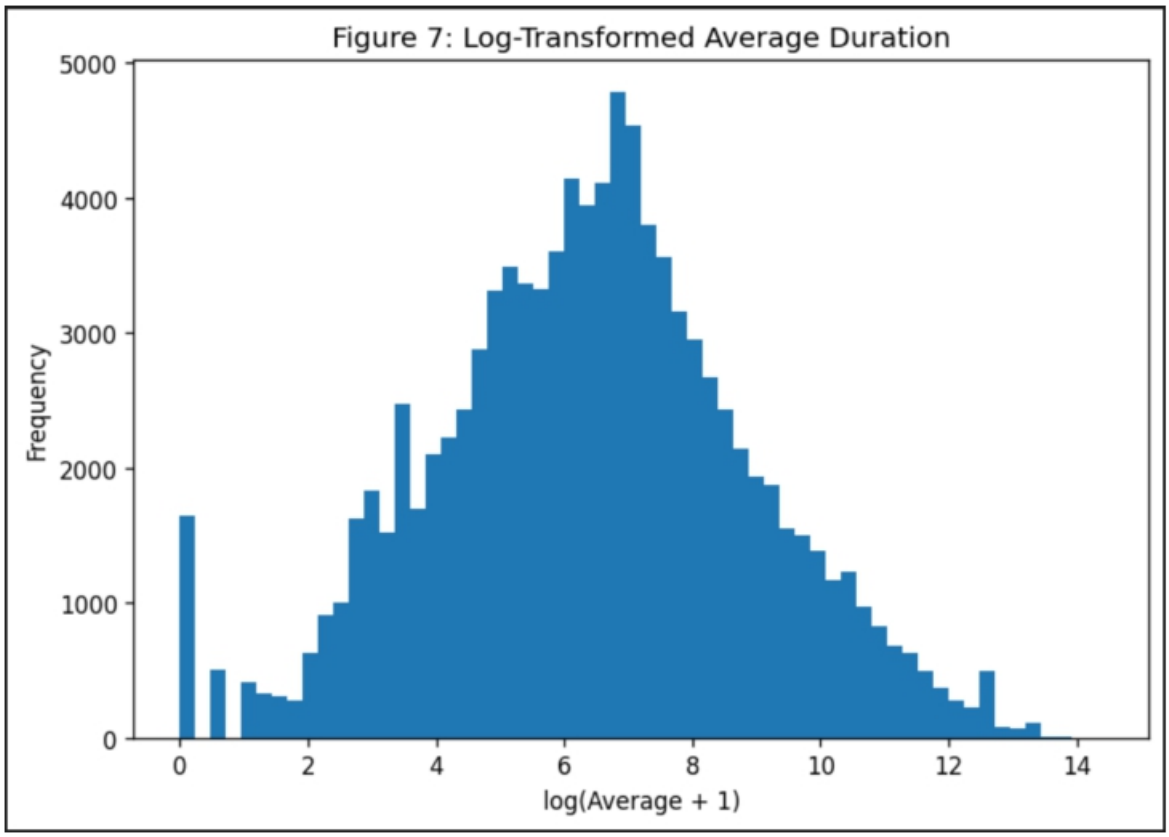
5. Model Analysis

5.1 Performance Evaluation

Model performance was evaluated using Root Mean Squared Error (RMSE) to measure the average magnitude of prediction errors, and the Coefficient of Determination (R^2) to assess the proportion of variance explained. The evaluation on the test set yielded the following results:

- **Multivariate Regression (Exploratory):** RMSE: 0.6521,R^2 = [0.931]
- **Linear Regression:** RMSE = 2.42, R^2 = 0.053
- **Random Forest:** RMSE = 2.36, R^2 = 0.098

Model Architecture	Features Utilized	RMSE	R^2 Score	Status
Multivariate Regression	Count, Minimum, Maximum	0.652	0.931	Discarded (Data Leakage)
Random Forest Regressor	Count only	2.36	0.098	Primary Valid Model
Linear Regression	Count only	2.42	0.053	Baseline



5.2 Statistical Interpretation

At first glance, the exploratory Multivariate Regression model appears to perform exceptionally well, yielding an artificially high R^2 score. However, the team identified this as a classic case of **data leakage** (target leakage). Because the *Average* duration is mathematically bounded by and aggregated from the exact same execution pool as the *Minimum* and *Maximum* variables, providing these features to the model essentially provides the answers before it makes a prediction. In a real-world production environment, a cloud architect would not know the maximum latency of a traffic spike until after the spike has concluded. Therefore, this multivariate model was intentionally discarded to maintain the scientific and operational integrity of the analysis.

Focusing strictly on the valid, load-based models (Linear Regression and Random Forest), we observe exceptionally low R^2 values. The highest valid R^2 of 0.098 indicates that less than 10% of the variance in execution latency can be statistically explained by invocation load alone.

Rather than indicating a failure in the modeling process, these metrics yield a profound operational insight. The lack of a strong predictive relationship confirms that the Azure Serverless architecture scales highly efficiently. If load directly caused latency, the R^2 would be significantly higher. Instead, the models mathematically prove that traffic spikes do not proportionally degrade system speed.

5.3 Assessment of Assumptions

The poor performance of the Linear Regression model further confirms the EDA findings: the relationship between serverless load and latency violates the assumption of linearity. The variance in duration is likely driven by external, unobserved factors such as "cold starts" (infrastructure initialization), memory allocation constraints, and specific application code efficiency, which manifest independent of overall traffic volume.

6. Conclusion and Recommendations

6.1 Final Conclusion

This project set out to determine if serverless execution latency could be reliably predicted using function invocation load. Through rigorous exploratory data analysis, data transformation, and the application of both parametric (Linear Regression) and non-parametric (Random Forest) models, we established that a direct, predictive relationship does not exist.

While the Random Forest model outperformed the Linear baseline, both models yielded low explanatory power ($R^2 < 0.10$). However, the validity of these models remains intact; they accurately reflect the operational reality of the dataset. The low correlation mathematically proves that Microsoft Azure's underlying infrastructure effectively decouples scale from speed. In a traditional server environment, increased load inherently causes queueing and latency. In this serverless environment, the auto-scaling mechanisms successfully provision resources to handle traffic spikes without proportional performance degradation. The variance in latency is therefore driven by external, non-load-related factors.

6.2 Business Implications

For business leaders and cloud architects, these findings have significant implications for reliability and cost optimization. The data demonstrates that unexpected latency—specifically the heavy right-tail outliers—cannot be forecasted simply by monitoring traffic volume. Consequently, businesses cannot rely on load metrics alone to ensure Service Level Agreement (SLA) compliance. Since extreme latency events (such as "Cold Starts") often occur independently of high traffic, organizations must adopt proactive infrastructure strategies rather than reactive, load-based scaling rules.

6.3 Recommendations for Future Work

To develop a highly accurate predictive model for serverless latency, future statistical pipelines must incorporate a broader set of engineered features. We recommend the following expansions for future research:

Time-Decay Variables: Incorporating the "time since last invocation" to explicitly model and predict Cold Start penalties.

Resource Allocation Metrics: Including memory consumption (e.g., MBs used)

and execution language (e.g., Python vs. C#), as runtime environments significantly impact initialization speed.

Plan Tiering: Differentiating between functions running on Azure "Consumption Plans" versus "Premium Plans" to isolate the effect of pre-warmed computing instances on latency variance.

By incorporating these dimensions, future models can move beyond load-based forecasting to accurately predict individual function performance bottlenecks

7. Team Contributions:

Moinuddin K : Led exploratory data analysis (EDA), implemented linear regression and Random Forest models, performed statistical evaluation (RMSE, R^2), identified and addressed potential feature leakage, and contributed to interpretation of modeling results.

Kunal Gurtoo : Contributed to dataset selection, formulation of the research question and business objectives, assisted in data preprocessing review, and developed the introduction and business context for the technical report.

Raj Dave : Conducted data cleaning and structural validation, generated visualizations (histograms, boxplots, scatterplots), supported model validation, and prepared presentation materials for the final business-oriented deliverable.

8. GitHub Repository

The complete source code, version history, dataset and requirements are available at:
Kunal Gurtoo : https://github.com/kunalgurtoo6-dev/MS-AAI-Azure-Analysis
Moinuddin k : https://github.com/moinuddinghouse/MS-AAI-Azure-Analysis
Raj Dave : https://github.com/rajdave050/MS-AAI-Azure-Analysis

9. AI Assistance Disclosure

Generative AI tools (ChatGPT) were used to support language refinement, structural organization, and clarification of statistical explanations. All data preprocessing, modeling decisions, implementation, and interpretation of results were independently validated and critically evaluated by the project team. Final conclusions and analytical judgments reflect the team's own understanding of the dataset and modeling process.