Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Data science
2025 | LIU-IDA/LITH-EX-A--2025/005--SE

Machine Learning-Based Cloud Cost Estimation: Developing and Evaluating Predictive Models

Maskininlärningsbaserad Kostnadsestimering för Molntjänster: Utveckling och Utvärdering av Prediktiva Modeller

Linus Rundin

Supervisor : Jorke de vlas
Examiner : Victor Lagerkvist

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Cloud services have revolutionized the way an organization manages or provides its resources, enabling easy scalability and flexibility. However, cloud service cost estimation remains critical for companies due to the constantly changing patterns of services. This research is focused on designing and implementing machine-learned models to predict cloud computing costs, explicitly considering GCP cloud computing services.

The study evaluates four machine learning models, Linear Regression, Random Forest, Feedforward Neural Network (FNN) and Long Short-Term Memory (LSTM), to identify the most effective approach to predict cloud costs based on historical usage data. Data collected from GCP billing records was preprocessed, aggregated, and analyzed to extract meaningful patterns and features for model training and evaluation. The models were assessed using standard evaluation metrics, including the mean squared error (MSE), the root mean squared error (RMSE), the mean absolute error (MAE), and the R^2 score.

The results indicate that the Feedforward Neural Network achieved the highest predictive accuracy, outperforming other models in multiple metrics. However, trade-offs between complexity and performance were observed, with simpler models like Random Forest and Linear Regression providing competitive results with reduced computational overhead.

The findings of this thesis contribute to a broader understanding of machine learning applications in cost prediction for cloud services, offering a foundation for improved financial planning and resource allocation. Future work includes further comparison with GCP's native pricing tools and extending the models to other cloud providers, such as AWS and Azure.

Acknowledgments

I want to thank my examinator Victor Lagerkvist and supervisor Jorke de vlas for helping me with the thesis. Also, a special thank you to my supervisor at QTE Development, Albin Henriksson, for helping me with the technical aspect of the thesis.

Contents

List of Tables

1 Introduction

The cloud has revolutionized how software projects are developed and deployed, with services such as Google Cloud Platform (GCP), Amazon Web Services, and Microsoft Azure. These services provide a scalable infrastructure to meet the needs of all types of modern businesses. For QTE Development, a company that consistently uses cloud solutions for its project, accurately estimating cloud costs is crucial to maintaining resource allocation and profitability. GCP offers its pricing calculator to assist users in predicting expenses; however, this tool has limitations. Mostly when accounting for historical usage patterns, fluctuating demand, and increased user bases. This project aims to create a predictive model that improves upon these limitations by using historical data and valuable metrics to provide a more accurate cost estimate. By doing so, QTE Development can make more informed decisions about future projects, ensuring better cost control and resource management. For QTE Development, the reliance on GCP for a majority of their projects presents a real challenge: providing a good and reliable way to estimate cloud computation cost. This is beneficial for QTE Development in many ways, more obviously for the reason of accurately negotiating with customers and being able to allocate company resources efficiently. Failing to estimate cloud costs can lead to overspending or project inefficiencies, which directly impacts profitability and customer satisfaction.

1.1 Problem statement

Traditional methods of calculating cloud costs based on exact memory and up-time usage of various GCP services offer limited foresight, as these costs are often fixed and predictable. However, more realistic variables, such as storage requirements, the number of unique users accessing the website, and other project-specific metrics, provide a more dynamic and practical basis for cost estimation. The study aims to create a model that can predict cloud expenses based on expected growth and usage patterns by analyzing these variables. The primary objective is to train multiple machine learning models, evaluate their predictive power against each other and finally present a model that can accurately estimate cloud costs. This model will be invaluable not only on a project level but also for providing an overarching view of QTE Development's cloud resource utilization. By incorporating factors such as constant costs, licenses, database instances, and variable costs, such as storage needs and unique user

access, the model will offer a comprehensive forecast, facilitating better financial planning and resource allocation.

1.2 Research question

To encapsulate the intention of this thesis, the goal is to answer the following research question.

- Which machine learning model provides the most accurate cost prediction for GCP usage for QTE Development?

1.3 Aim

Complementary to the research question, this thesis aims to create a predictive model that outperforms Google's GCP pricing calculator in estimating cloud computing costs for QTE Development, using historical data to improve cost accuracy. The objective is to:

- Implement and compare three machine learning algorithms.
- Identify the model with the lowest prediction error.
- Identify which model from the model selection 1.3.1, with the lowest prediction error.
- Evaluate the suitability of each model based on QTE Development's project needs.

1.3.1 Model selection

We concentrated on the following models:

1. Random Forest,
2. Linear Regression,
3. Feedforward Neural Network (FNN),
4. Long Short-Term Memory Network (LSTM).

The motivation for choosing these different machine learning models is that these were the most popular models occurring in the related works. And most of these models have been used in similar cases, and that is why we choose to use just these ones. There are many different methods for cost prediction, and comparing every single one would not be feasible under almost any circumstance. However, two types of prediction methods stood out: Neural networks and regression methods. Regression methods that are simpler to implement could show strength in "easier-to-understand" and faster results but fall short in accuracy. In contrast, an artificial neural network could produce better predictions with the cost of complexity. Hashemi et al. have done a comprehensive literature study on cost estimation and prediction for construction projects [20]. Their findings tell us that of all machine learning techniques used, Artificial neural network(ANN) and regression analysis(RA) were the most popular methods. Furthermore, these methods were also the most successful methods [20].

2 Theory

This chapter provides a brief introduction and explanation to the machine learning models and evaluation metrics used in this study.

2.1 Machine Learning Models

This section provides an overview of the machine learning models considered in this thesis for cloud cost estimation. These models were selected based on their popularity and proven effectiveness in similar predictive tasks.

2.1.1 Regression analysis

Regression analysis is a technique to model relationships between variables. The use cases for this are applicable in various areas, such as engineering, chemistry, and economics [14]. These variables can be singular and multiple. When there is only a single response variable, the regression analysis is referred to as a univariate regression. Consequently, when dealing with two or more response variables, a regression analysis of this kind is called multivariate regression.

2.1.2 Linear regression

Linear regression is a version of regression analysis in which the modeling technique uses the variables to make linear equations for the observed data [14]. Linear regression is modeled as follows:

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{2.1}$$

Where Y is the dependent variable, X is independent variable. β_0 and β_1 are the intercept and slope, respectively. Lastly, epsilon represents the error term. Linear regression has the ability to interpret and is efficient in exploring relationships between variables [6].

2.1.3 Multi linear regression

Multiple linear regression extends the linear regression model to consider multiple predictor variables [6]. In this model, several independent variables can be used to predict the response variable. The formula for multiple linear regression can be described as follows:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon \tag{2.2}$$

where Y is the response variable, $X_1, X_2, \ldots, X_p$ are the predictor variables, and p is the total number of predictors. The coefficients $\beta_0, \beta_1, \ldots, \beta_p$ represent the weights. Lastly ϵ represents the error between the predicted and actual values [6].

2.1.4 Random Forest

Random Forest (RF) is a machine learning method that uses multiple decision trees to improve its prediction accuracy and minimize overfitting [18, 2]. A decision tree is a model where data is split into branches based on feature values. Forming a tree-like structure where each branch is a decision outcome. In Random Forest regression, predictions from multiple decision trees, each trained on different subsets of the data are averaged out. This method produces a more resilient model compared to a single decision tree [2]. The formula for random forest regression can be described as follows [18]:

$$\hat{f}_{RF}(x) = \frac{1}{K} \sum_{k=1}^{K} T(x; \Theta_k) \tag{2.3}$$

Where:

- The input feature vector x is used to grow K decision trees, which are denoted by $\{T(x; \Theta_k)\}$, where $k = 1, \ldots, K$.
- Each Θ_k represents a randomly chosen subset of the training data through a process called "bagging", which involves resampling the original data set with a replacement.
- The final RF prediction, $\hat{f}_{RF}(x)$, is calculated as the average of the predictions made by each individual tree

This process helps reduce variance by combining multiple weak learners, resulting in a strong learner. Bagging also ensures that individual trees are less correlated with each other, thereby minimizing the overall prediction error [16].

2.1.5 Feedforward Neural Networks (FNN)

Feedforward Neural Networks (FNNs) are essential types of artificial neural networks. This machine learning model frequently used for pattern recognition and regression problems [13]. The structure of an FFN consists of three main components: an input layer, one or more hidden layers, and an output layer [13]. Information flows through the network in a forward motion, from the input layer to the hidden layers, and ends up in the output layer [17]. This is done without forming any cycles or loops in the network topology [17].

Common activation functions include sigmoid, ReLU (Rectified Linear Unit), and tanh. These functions introduce nonlinearity into the network and enable it to learn complex mappings between inputs and outputs.

The primary goal of training an FNN is to minimize the difference between predicted outputs and the true target values [17].

FNNs are particularly effective for generalization, which means they can make accurate predictions on data not seen during training [13]. Their ability to learn complex mappings makes them suitable for tasks such as image recognition, signal processing, and time series prediction [13].

2.1.6 Long Short-Term Memory Networks (LSTM)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to overcome the limitations of traditional RNNs in learning long-term dependencies. This concept was presented by Hochreiter and Schmidhuber in 1997 [9]. Recurrent neural networks are designed to process sequential data by maintaining a hidden state that captures information about previous time steps. However, standard RNNs suffer from vanishing and exploding gradient problems when training in long sequences, severely limiting their ability to learn dependencies over extended periods [7]. LSTMs address this issue through a memory cell and a gating mechanism that allow the network to retain or discard information as needed selectively. An LSTM network consists of a sequence of repeating units, each containing a memory cell and three gates: the input gate, the forget gate, and the output gate. These gates regulate the flow of information to, from and within the memory cell, allowing the network to preserve long-term dependencies while mitigating the problem of vanishing gradients [9].

2.2 Evaluation Metrics

Several evaluation metrics are used to assess the performance of the predictive models developed in this thesis. These metrics provide quantitative measures to compare the accuracy, efficiency, and robustness of the models.

2.2.1 Mean Squared Error (MSE)

The mean squared error (MSE) is a widely used metric in supervised learning to evaluate the performance of a model during training [10]. Measures the average square difference between the predicted values (P_j) and the actual target values (A_j) for all data points in a data set. A lower MSE value indicates better model performance, as it means that the predictions are closer to the true values [10]. The formula for MSE is the following.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(P_j - A_j)^2 \tag{2.4}$$

Where:

- P_j is the predicted value for the j-th instance,
- A_j is the actual target value for the j-th instance,
- n is the total number of instances.

During the training process of models such as Learning Vector Quantization (LVQ), MSE is used to evaluate how well the model is learning to classify data [10]. The solution with the lowest MSE is typically chosen as the best model, as it minimizes the error between the predictions and the targets. Despite these limitations, MSE remains a standard metric for regression tasks and a useful tool for evaluating model performance when used alongside other metrics [10].

2.2.2 Root Mean Squared Error (RMSE)

The root mean squared error metric is the square root of MSE, making it more interpretable in practical scenarios. Although MSE is easier to compute and analyze mathematically, RMSE is often preferred when focusing on understanding the scale of errors in the data context.

2.2.3 R^2 Score

The R^2 score quantifies how well the predictions of a model explain the variability of the target variable. It measures the proportion of variance in the dependent variable that is predictable from the independent variables.

- R^2 ranges from 0 to 1:
  - **1**: Perfect fit. The model explains all variability in the target variable.
  - **0**: The model explains none of the variability and performs no better than simply predicting the mean of the target variable.
  - **Negative Values**: Indicates the model performs worse than a mean prediction, suggesting poor predictive power.
- A higher R^2 score indicates a better fit of the model to the data but does not necessarily indicate good generalization to unseen data.

2.2.4 Mean Absolute Error (MAE)

The MAE measures the average absolute difference between the predicted values and the actual target values. It is a straightforward, linear metric that calculates the average magnitude of errors without considering their direction. The formula is as follows:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |P_i - A_i|$$

Where:

- P_i: Predicted value for instance i.
- A_i: Actual value for instance i.
- n: Total number of predictions.

The MAE can be summarized by:

- **Smaller MAE**: Indicates more accurate predictions.
- MAE is easy to interpret as it is in the same unit as the dependent variable.
- Unlike metrics such as MSE or RMSE, MAE is less sensitive to outliers and does not overly penalize large errors.

2.2.5 Evaluation Criteria

The evaluation of models will primarily focus on minimizing MAE, MSE, and RMSE, as they directly reflect predictive accuracy. R^2 will be used to assess how well the model captures the overall variability in the data. MAPE will provide additional insights into percentage-based accuracy, particularly for comparisons across different services or datasets.

The selection of these metrics ensures a comprehensive evaluation of both absolute and relative performance, allowing a robust comparison of the models.

3 Background and Related Work

This chapter will give some insight into the topic of cloud computing, cloud estimation, and machine learning prediction models. Alongside some interesting related work, this chapter aims to give a broad and clear understanding of the subject.

3.1 Overview of cloud computing

The phrase cloud computing can be used to define many things these days. Surbiryala and Rong claim that cloud computing integrates several services. Such services are Distributed computing, Utility Computing, Parallel computing working together with network storage and virtualization, and other related technologies [19]. The National Institute of Standards and Technology(NIST) defines cloud computing as a model for accessing and using services like servers, storage, and applications over the Internet whenever you need them, allowing you to quickly setup or remove these resources with little hassle or direct interaction with the service provider [5]. Cloud computing is a service that many attributed to it, which comes with benefits and complications. Looking closely at what GCP offers as a cloud computing service, the most obvious is Google Compute Engine (GCE). GCE is a service that enables users to run applications and various workloads in the "cloud" by providing virtual machines [cite shitty source]. Another service Google offers is Cloud Run. Google Cloud Run allows it to run front- and back-end services, batch jobs, host LLMs, and queue processing workloads without the need to manage infrastructure [4].

3.2 Predictive machine learning models

Choosing which machine learning model that is best suited to the task depends on several different reasons. Candanedo et al. present two supervised machine learning algorithms, logistic regression, and Random Forest, to predict the performance of HVAC systems based on temperature data. Both algorithms have their strengths and weaknesses, and the choice between them should be informed by factors such as the nature of the dataset, the specific prediction goals, and computational constraints. Logistic regression, as a parametric model, is straightforward and easy to interpret. It works well when there is a linear relationship between the independent variables and the outcome. They found that in the context of

HVAC systems, logistic regression can be useful to quickly identify binary outcomes, such as whether a system is functioning within the normal range or experiencing a failure [3]. On the other hand, Random Forest is a nonparametric method that builds multiple decision trees and combines their outputs to improve prediction accuracy. It is particularly robust in handling non-linear relationships and complex interactions among variables. In the HVAC case study, Random Forest was found to be slightly less accurate than logistic regression, but its ability to capture more nuanced patterns could make it advantageous when dealing with more complex datasets, especially as the size of the dataset increases [3]. Furthermore, Wang et al. present a systematic review of predictive machine learning models with structured data to predict outcomes. They found that the most commonly used machine learning models were random forests, support vector machines, decision trees, and artificial neural networks. Unfortunately, performance was not presented or evaluated in this study, but frequency can tell us what models are commonly used in this type of practice [24]. Furthermore, Wang found the same result as they found in their paper on predictive machine learning models of methane emissions using farm environmental data. They found that the Random Forest model had the best performance [23].

3.3 Cost components in cloud estimation

There are components that contribute to the overall cost of a cloud computing service. Such components are data storage, computational machines, and data transfers in the cloud. The cost of these components is provided by the cloud service, in this case, the GCP [4], [22]. Additional costs, such as licenses, should also be considered. Data storage costs vary according to the amount of data stored, how often the data is accessed, and the provider's pricing model. The computational cost of machines is typically measured on virtual machines. This means that you could get instances of a machine or get CPU hours of a machine. In this case, costs fluctuate depending on the number of CPUs, memory, and duration of use. Data transfer costs are charged depending on how much data is moved in and out of the cloud [4].

4 Method

The method for this project is divided into two parts. The first part is querying the large data set from GCP and manipulating it into a data set that can be used to apply the different models. The second part shows the four different models and how they were implemented.

4.1 Data collection

This section will present how the data is gathered and where it comes from.

4.1.1 Google Cloud Platform

All data come from GCP. An existing account has been present from 2021 where QTE Development has opened up their admin account to track billing data for all projects hosted on GCP. This includes multiple services like Google Cloud Run, SQL instances, and more. Due to administration rules and security, a separate GCP project was created for this thesis where all data was copied over so queries and other administration tasks could be performed without interfering with any live operation from qte.

4.1.2 BigQuery

BigQuery is a tool provided by Google that allows us to query and download sets and subsets of the data stored. Google provides a simple solution to store all billing and computational data from GCP and present them in a data set. This data set is then accessible from BigQuery.

4.1.3 Querying and partitioning the data

At the time of querying the data, there were more than 23 million rows of usage and cost data. Without doing any operation, the data are presented as every bit of action that happens on any project in GCP. Due to this level of precision, manipulating the data is crucial to be able to use it for anything useful. Below, we present major steps and criteria used to make the data manageable and useful for modeling.

4.1.4 Data characteristic

The raw data presents challenges due to:

1. Time coverage: The data includes all actions performed during the collection period, regardless of the status or timeline of the project.

2. Cost variation: Many projects generate minimal or no cost, often remaining within GCP's free usage thresholds, which can skew cost analysis.

4.2 Data collection steps

The data collection can be described in the following steps:

4.2.1 Step 1: Identifying Active Projects

The initial step involved identifying active projects within a designated time frame. Active projects were specified as those with recorded usage after December 31, 2022. The query for this step can be found in A.1. This ensures that only projects with significant historical data are included, excluding projects that started in the middle of the time period.

4.2.2 Step 2: Aggregating Cost Data

Using the active projects identified in Step 1, the next step aggregates cost data for these projects over a specific analysis period (2023-2024). The projects were further filtered by restricting the analysis to high-usage services such as Compute Engine, Cloud SQL, Cloud Storage, Cloud Firestore, and Artifact Registry. The query for this step is shown in A.2. This filtering reduces noise from low-cost projects and ensures that only relevant services are included in the analysis. After applying all filtering criteria, a total of 46 projects remain.

4.2.3 Step 3: Calculating Rolling Averages

The last part of the query calculates monthly usage metrics for projects and services. The final query joins this summarized monthly average usage with the original filtered data on project ID, service, and month, allowing each hourly record to include its corresponding average monthly usage A.3.

4.2.4 Partitioning and Rolling Averages

Partitioning by service and project_id allows the calculation of rolling averages specific to each service and project. This approach ensures that trends are not skewed by aggregated data from unrelated services or projects. By structuring the query in this manner, the data was effectively transformed into a manageable format for further analysis. This provided insights into usage trends and cost patterns while addressing the challenges of granularity and volume in the original dataset.

4.2.5 Visualisation of data

When querying the data set, regular visualization scripts were programmed to understand the data. To clarify whether the queried data seemed usable, Python and a visualization tool called Matplotlib were used to visualize the data with plots and figures. By doing this, you get a good understanding of what the data is and can find anomalies that might skew your data.

4.2.6 Tools and Libraries Utilized

This research used Python as the programming language to develop machine learning models. A set of machine learning and deep learning frameworks were implemented to support model development. Specifically:

- **Scikit-learn**: Used for data preprocessing, data set splitting, and regression analysis [15].
- **TensorFlow and Keras**: These libraries were used in creating deep learning models, such as LSTM and FFN [1].
- **Pandas and NumPy**: Used for data manipulation, cleaning, and aggregation. [12, 8].
- **Matplotlib**: Used to visualize results, including metrics such as validation loss and mean squared error [11].

4.3 Implementation

The four approaches were coded in the same programming language, Python 3. Python is simple and well-suited for machine learning and data handling.

4.3.1 Predictive Modeling Overview

To analyze and predict future trends based on the processed dataset, we implemented and evaluated four distinct predictive modeling algorithms: Linear Regression, Random Forest, LSTM, and FNN.

4.3.2 Data splitting

The data was split into 80 percent training data, and 20 percent validation data. The 80-20 split is a standard practice that provides the model with sufficient data for learning while preserving enough data for an unbiased assessment of its predictive capabilities. Furthermore, the split was performed randomly to minimize potential bias. A 90-10 split was also considered due to the data set being large, meaning you could train the data on a bigger split without risking the chance of overfitting. However, this was ultimately not chosen and the more standard approach was used.

4.3.3 Feature Engineering

The features selected for the models where:

1. **Service:** This represents the service utilized (e.q., Compute Engine, Cloud SQL). Depending on which service is used, a different amount of computational time and cost correspond directly to the service.
2. **Hourly usage:** This represents the granular usage data, documenting the resource consumption in a time-sensitive matter. This allows the model to capture, fluctuations and trends in a more short-term cost prediction.
3. **Average monthly usage:** By including an average of usage data, the model gains a broader perspective on long-term trends. This helps mitigate the noise introduced by the short-term variation.

These features were chosen after analyzing the potential output that the GCP data set provides. The goal is to consider as many features as possible without causing potential overfitting. Since the service is derived from the dataset as a string, a simple encoding of services

from string to integer is needed for the models to interpret the data correctly. By analyzing the input of Google's pricing calculator, we can deduce that they give estimates depending on usage and type of service. This approach makes it applicable to compare and evaluate with their own solution.

4.3.4 Features considered but not included

The number of active users during a given time period was considered a potential feature. However, the user count data was not part of the data set. Furthermore, to acquire the user count data, an integration with another system would be needed, which is beyond the scope of this project.

4.3.5 Additional filtration

Some additional data has been filtered from the original dataset derived from GCP. There were a total of 64 rows that included a time span with a negative hourly cost. This could occur due to many things. The most probable cause for this entry is that the QTE overpaid for said service and a refund was observed, leading to the cost being negative for that time instance. With this knowledge, we decided to remove these instances to make sure the model trains on only true data.

4.3.6 Model Training and Evaluation

During training, a simple integration of early stopping was used to determine when overfitting or generalization occurred. By monitoring the performance with the validation set, the model can stop training when no further improvement is possible. The metric used for early stopping was validation loss after each epoch.

4.4 Model comparision

After the four models were finished training and the best model from each was selected, the next step was to evaluate their predictive capabilities with real-world data. This section details the process of testing the models with the unseen dataset, comparing their prediction to the actual cost, and evaluating their performance based on MSE, RMSE, MAE, and R^2 score.

4.4.1 Test data

The test data comes from the time period of 2024-04-01 until 2024-09-31. With the exception that the data do not come from any project already present in the training set. Making this data totally unseen to the models. See the listing A.4 for the entire query. After applying all filtering criteria, a total of 40 projects remain.

4.4.2 Implementation shared among models

There are plenty of code that are reused for all four models. The train and test data set is imported once by using pandas, both sets are csv's. Some minor data preparations are done to the data sets where the field service is encoded and changed to the field "service_encoded" with its encoded value instead of a string. Moreover, the fields "hour", "day_of_week" and "date" are converted to their pandas own time functions for each respective column. Sklearn's own function for splitting into tests and validation with a random state is used for all models during training.

4.4.3 Implementation of Linear Regression

The implementation of linear regression in Python uses the linear model for sklearn. The model is fitted to the training data and then tries to predict the test set. Using the built-in function from sklearn, we can derive all the evaluation metrics needed for the result.

4.4.4 Implementation of Random Forest

Similar to the implementation of LR, RF uses Sklearn's random forest function with the default number of estimators and state. Then, the model is fitted to the data and predicted on the test set.

4.4.5 Implementation of Feed-forward neural network.

The FNN model comes from the TensorFlow sequential library, which adds three layers to implement the feed-forward neural network. The model has three layers and an output layer. The model uses Adam as the optimizer and MSE as the loss function. The model is set to be trained for 10 epochs with a basic early stoppage function to prevent overtraining.

4.4.6 Implementation of Long short term memory network

The LSTM model is a bit more complicated than the other models due to the training data needing to be in sequences. We used numpy's hstack and a function from **??** called split_sequence to create a sequences of the training data. The input length of the sequence was picked from the recommendation from the source, which is 7. When the data had been transformed into sequences, the tensorflow model was trained with 150 epochs and with the same early stopping function as FNN. Also, using the Adam optimizer with MSE as the loss function.

4.4.7 Hyperparameter Selection

The hyperparameters for both the FNN and LSTM models were chosen based on standard recommendations from TensorFlow documentation to ensure stable and efficient training [21].

5 Results

In this chapter, the results and performance of the models are presented. The results are presented in tables to compare the evaluation metrics across the different models.

5.1 Training results

Table 5.1 shows the validation and test MSE values for the best performing FNN and LSTM models, together with the number of epochs each model trained. The FNN achieved a slightly better test MSE of 8.168 compared to the LSTM's 8.866. The FNN also required only three epochs to achieve its performance, compared to the two epochs LSTM did. These results demonstrate that FNN was more effective at generalizing the validation and the test data. There is also large discrepancy between validation MSE and test MSE for both FNN and LSTM, this is discussed in more detailed in section 6.1.2.

Model	Validation MSE	Test MSE	Epoch
Long Short-Term Memory Network	0.061	8.866	2
Feed-Forward Neural Network	0.037	8.168	3

Table 5.1: Best models for FNN and LSTM.

Table 5.2 compares the MSE for all four models. The FNN achieved the lowest MSE of 8.168, followed by the LSTM with 8.866. RF and LR with 8.944 and 9.141, respectively. The results indicate that neural network-based models, particularly the FNN, are slightly more capable of understanding the relationships within the data set compared to the regression models LR and RF.

Model	Mean Squared Error (MSE)
Linear Regression	9.141
Random Forest	8.944
Long Short-Term Memory Network	8.866
Feed-Forward Neural Network	8.168

Table 5.2: Mean squared error (MSE) for different models.

Table 5.3 reports the MAE for all models. The LSTM showed the lowest MAE of 0.759, closely followed by the FNN with 0.786. RF with 0.953 and LR with 0.844 performed slightly worse, indicating that the neural network models are a little better suited to minimize the average absolute deviation of the predictions from the actual values.

Model	Mean Absolute Error (MAE)
Linear Regression	0.844
Random Forest	0.953
Long Short-Term Memory Network	0.759
Feed-Forward Neural Network	0.786

Table 5.3: Mean Absolute Error (MAE) for different models.

Table 5.4 shows the R^2 score that evaluates how well each model explains the variability in the data set. The FNN achieved the highest R^2 score of 0.1191, indicating better performance compared to LSTM with 0.0439, RF with 0.0355, and LR with 0.0142. However, the relatively low R^2 values across all models suggest significant variability in the dataset, leading to low scores for all of them.

Model	R^2 Score (R^2)
Linear Regression	0.0142
Random Forest	0.0355
Long Short-Term Memory Network	0.0439
Feed-Forward Neural Network	0.1191

Table 5.4: R^2 Score (R^2) for different models.

Table 5.5 summarizes the RMSE values for all models, where lower values indicate better predictive performance. The FNN demonstrated the best performance with an RMSE of 2.858, followed by LSTM with 2.978, RF with 2.991, and LR with 3.023. The result shows us the error in the same unit as it is trained to predict on, meaning that the difference between the best model, FFN with a RMSE of 2.858 and the worst model, LR with a RMSE of 3.023, have a difference of 0.165 SEK on an hourly basis.

Model	Root Mean Squared Error (RMSE)
Linear Regression	3.023
Random Forest	2.991
Long Short-Term Memory Network	2.978
Feed-Forward Neural Network	2.858

Table 5.5: Root Mean Squared Error (RMSE) for different models.

Table 5.6 presents the RMSE values on a daily basis, indicating the models' prediction errors in the same unit as the cost being predicted. The difference between the best model, FNN, and the worst model, LR, is 5.96 SEK per day. While this margin may seem small, it accumulates significantly over time.

Model	Daily Root Mean Squared Error (RMSE)
Linear Regression	72.552
Random Forest	69.784
Long Short-Term Memory Network	68.941
Feed-Forward Neural Network	66.592

Table 5.6: Daily Root Mean Squared Error (RMSE) for different models.

Table 5.7 provides the RMSE values for monthly cost predictions. The difference between the best model FNN and the worst LR amounts to 33.13 SEK per month.

Model	Monthly Root Mean Squared Error (RMSE)
Linear Regression	396.334
Random Forest	381.275
Long Short-Term Memory Network	376.219
Feed-Forward Neural Network	363.204

Table 5.7: Monthly Root Mean Squared Error (RMSE) for different models.

Table 5.8 shows the RMSE values over a yearly period, illustrating the long-term prediction accuracy of the models. The difference between the FNN and LR, the best and worst models, is 124.23 SEK per year. In an annual time frame, this difference underscores the better performance of FNN.

Model	Yearly Root Mean Squared Error (RMSE)
Linear Regression	1447.683
Random Forest	1391.231
Long Short-Term Memory Network	1372.813
Feed-Forward Neural Network	1323.455

Table 5.8: Yearly Root Mean Squared Error (RMSE) for different models.

6 Discussion

In this chapter, the discussion of the result and method are presented, respectfully.

6.1 Results

This section covers the discussion of the results from 5. By analyzing the result from each metric with the purpose of understanding the performance of each model.

6.1.1 Analysis of Results

The performance metrics, including Mean Square Error, Root Mean Squared Error, and R^2 scores, comprehensively evaluate the predictive accuracy and effectiveness of the different machine learning models. The feed-forward neural network generally showed the best overall performance in all metrics.

6.1.2 Best FNN and LSTM models

The best FNN and LSTM models demonstrated a significant disparity between their validation MSE and test MSE. The validation set was randomly selected and amounted to 20 percent of the training data. These results show that the models are better at predicting data from the same dataset than the entirely new dataset. In most scenarios, this is completely normal. However, when such a significant disparity is like this, we must verify whether the model is overfitting and why. We observed no difference in the best models' predictive power during validation when changing:

1. The models' training and validation distribution, and
2. the number of epochs the models trained on.

We saw no improvement in changing the training data distribution, but we saw a slight decrease in performance when the distribution lowered to around 10 - 30 percent of the training data of the whole set. Regarding the number of epochs, as stated in 4, early stopping makes it so that we know that the chosen epoch gave the best training result.

With this in mind, why do we see such a significant disparity? The answer is probably in the training data set. Due to the training data set being a set of many different projects with

their own distribution of services, these projects vary greatly. One project can use the same services, such as Cloud SQL and Cloud Computing. Comparing this project to another that uses the same services can yield entirely different results due to significant variations in how each project operates. For example, one project can be an online website, and the other can be a mobile application that displays videos. These projects are different and act differently, but modeling them as the same can lead to the models not understanding the broader difference between the data. This is also amplified when testing on the test set, which is a different set of projects. These projects can be vastly different in how they operate and in how they are built to use their resources. That is, the data is vastly diverse, both for training and test data. By training the model on the training data, the models learn to predict only the training data, making it very good and learning the imperfections of just the training data. However, this makes it much worse when data is not similar to what it's trained on, leading to it not being as good as predicting new unsolicited data.

6.1.3 MSE and RMSE

The MSE and RMSE metrics offer insight into the predictive accuracy and practical utility of the models implemented. As shown in 5.2 we get the lowest measured MSE for FNN with a MSE of 8.168. The other models are much closer to each other, and LSTM is the best among them. However, because MSE shows us a magnified interpreter and the predictive error of the models, we can see that there is no significant difference between the different models. This is much clearer in 5.5 where the error is presented in their actual magnitude. The results show clearly that the worst model, LR, is very close to the same hourly error as the other models. We can scale this up and start looking at daily, monthly, and even yearly errors. In a year, we get a difference of roughly 120 Swedish crowns. For a whole year, the improvement in the implementation of neural networks is there but in a tiny magnitude. The increased complexity of developing neural networks does lead to improvement. However, not to the extent one would think.

6.1.4 R^2 score

Comparing the R^2 scores among the models, we can see a clear outlier, FNN, with a score of 0.11191. This is vastly better than the next-best model, LSTM, which had an R^2 score of only 0.10439. At the bottom, we have LR with an underwhelming result of 0.10142, which is very close to the lowest possible score, 0. The performance of FNN suggests that its architecture and implementation are better at understanding the underlying patterns in the data compared to the LSTM, RF, and LR models. However, although 0.11191 is vastly better than the other results, it is still considered a poor score. This means that the model only explains 11,91 percent of the variance of the target variable. This is linked to 6.1.2, where we know that there is a large diversity of all the projects in both the training and the test dataset. This introduces a lot of variability, making it difficult for the models to achieve a high R^2 score.

6.1.5 MAE

For the models evaluated, MAE serves as a straightforward metric to understand how far, on average, the predictions deviate from the actual cloud costs. LSTM achieved the lowest MAE, with a value of 0.1759, indicating that its predictions were closest to actual values on average. FNN followed closely, with an MAE of 0.1786, showing a performance comparable to LSTM in minimizing average prediction errors. Random Forest and Linear Regression lagged behind, with MAEs of 0.1953 and 0.1844, respectively, reflecting less accurate predictions. The ability of LSTM to capture temporal relationships likely contributed to its better performance, minimizing absolute errors, as it could leverage sequential data more effectively than the others. Despite FNN lacking temporal modeling capabilities, the FNN performed nearly as well

as the LSTM, highlighting its strength in capturing nonlinear patterns. The higher MAEs for RF and LR suggest their inability to fully capture the complexities and variability of the dataset. LR's simplicity and RF's reliance on discrete splits in decision trees might have limited their predictive accuracy.

6.2 Method

In this section, we will discuss the method used. It is divided into three subsections: data gathering and processing, machine learning models used, and evaluation.

6.2.1 Data

The methodology used in this study provided a solid foundation for evaluating the performance of various machine learning models in predicting cloud costs. One of its strengths was the approach to data preprocessing, which ensured that the models were trained on clean, well-structured data. By aggregating, partitioning, and filtering the raw dataset, the study emphasized relevant patterns and allowed the models to focus on full meaningful features.

Due to the large amount of data stored in GCP, there were many challenges in filtering and selecting the most relevant data for this study. While the dataset was comprehensive, covering various projects and services, the process of gathering data could have been significantly improved with a more precise understanding of the specific features and patterns needed for predictive models. A substantial addition to the data set would have been additional data for every project, detailing the number of users from the dedicated time period. In addition, information about the project, such as if it was a mobile app, a web app, or another service, could help to better categorize the projects. This would help categorize the data into more specific areas, which would be possible because the data set already is very large. This means that it would not lack in the number of data points.

An added method for testing the best model for future projects could involve extending the study timeline to evaluate the model's predictive power on actual future project data, allowing for a real-world comparison between predicted and actual costs over time. This approach would provide valuable insights into the model's ability to generalize and adapt to evolving project characteristics and resource usage patterns. This would give additional evidence to the predictive power of the generated models.

6.2.2 Machine learning models

The implementation of machine learning models was straightforward for LR, RF and FNN. Following the docs from [21] and sklearn, implementing these models was quite simple. LR and RF are especially notable because of their simpler complexity. Although their simpler complexation resulted in a relatively good way to their more complex neural network approaches. The implementation for FNN was straightforward and was easy to integrate using the TensorFlow documentation, with a developing time almost similar to that of the regression models, it also yielded the best result.

Implementing LSTM contributed the largest amount of development time. Due to its architecture of needing the data to be in sequences, additional computation and complexities were necessary for the model to function. Moreover, despite the longer development time, the results did not impress compared to LR and RF. The dataset's diverse and project-specific nature may have favored models like RF and FNN, which are better equipped to handle heterogeneous and non-sequential patterns. This indicates that while LSTMs are powerful tools for time-series analysis, their applicability may be limited when the temporal structure of the data is weak or overshadowed by other factors. Future work could explore whether segmenting the data by project type or introducing additional temporal features could enhance the

LSTM performance. However, for the current dataset, simpler models demonstrated competitive results with significantly shorter development time and complexity.

6.2.3 Evaluation

The evaluation process provided critical information about the performance and limitations of machine learning models. Using a robust and trusted set of metrics, the study ensured that multiple aspects of model performance were assessed, from accuracy and error magnitude to the ability to explain variance in the data. One of the main strengths of the evaluation was the use of a completely unseen test dataset, which included projects not presented in the training data. This approach effectively tested the model's ability to generalize and demonstrated how they might perform on future projects. In addition, the combination of the evaluation metrics ensured a balanced assessment of both the scale of prediction error and the explanatory power of each model.

Despite these strengths, the evaluation revealed limitations in the model's ability to handle the diverse nature of the test data set. The significant disparity between the validation and test metrics, especially for LSTM and FNN, suggests that these models struggled to generalize beyond the training data. The evaluation also highlighted an interesting trade-off between model complexity and performance. Although FNN and LSTM showed some improvements in capturing relationships in the training data, simpler models like RF and LR provided competitive results with far less computational effort. This suggests that, in this context, the added complexity of neural networks may not always translate to improved real-world performance.

6.3 Source Criticism

The literature on cloud cost estimation is relatively specific, making it difficult to find sources that align directly with the objectives of this study. However, a more general approach to estimation and prediction, as applied in this study, remains valuable. Although the literature primarily focuses on cost estimation in general rather than cloud costs specifically, it still offers relevant insights. Furthermore, priority was given to sources that were peer-reviewed and published by well-known, reputable outlets.

7 Conclusion

This chapter contains the summarization of the study, the conclusion of the research questions, and provides a detailed plan for future work.

7.1 The goal

This study successfully achieved its goal of developing a predictive model for cloud cost estimation based on extensive data from the Google Cloud Platform (GCP). The data derived from GCP was thorough and comprehensive, capturing a wide range of relevant usage patterns and cost components, which provided a solid foundation for the model. Using historical GCP billing data, the model can forecast future costs, aligning with QTE's objectives of optimizing cloud resource allocation and ensuring more precise financial planning. However, to fully evaluate whether the best performing model in this study outperforms the existing GCP pricing calculator, additional comparative analysis is necessary. For a deeper exploration of this topic, including specific recommendations for conducting such an evaluation, refer to the future work section 7.3.

7.2 Answer to the research question

The research question "Which machine learning model provides the most accurate cost prediction for GCP usage for QTE Development?" was thoroughly investigated through the implementation and evaluation of four machine learning models. Linear Regression, Random Forest, Feedforward Neural Network, and Long Short-Term Memory networks.

Among the models tested, the Feedforward Neural Network emerged as the most accurate for GCP cost prediction. This was evident across multiple evaluation metrics, including the lowest Mean Squared Error, Root Mean Squared Error, and the highest R^2 score. The FNN achieved an MSE of 8.168 and an R^2 score of 0.11191, indicating its relative success in capturing variability in cloud costs compared to other models.

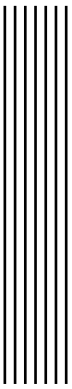
Although the Long-Short-Term Memory network also performed well, with slightly better accuracy than the Random Forest and Linear Regression models, its complexity and the need for sequence preparation introduced additional challenges. While the Random Forest and Linear Regression models demonstrated reasonable predictive capabilities with significantly

lower computational requirements, they lacked the nuanced understanding of the dataset's variability that the FNN and LSTM models showed.

Overall, the results indicate that while FNN is the best-performing model for this specific task, the trade-off between accuracy and complexity must be carefully considered depending on the application. In scenarios where computational simplicity and interpretability are more critical, Random Forest or even Linear Regression may be the right choice.

7.3 Future Work: Further Evaluation and Development

Based on the findings of this study, an obvious approach would be to perform an extended evaluation, comparing the predictive models developed with the existing GCP pricing calculator. This comparison could use the same input parameters and features as those used by thee GCP calculator to ensure a direct and fair evaluation. Some areas that would be interesting and valuable is a direct model comparison. Using identical data sets for both machine learning models and the GCP pricing calculator to see who performs better after a specific time period. Moreover, one way to further study this subject could be to introduce additional performance metrics. Performance metrics could be user satisfaction, execution time, complexity, and ease of use. This could give a broader view of the strengths and weaknesses of the models. If we take an even broader view of the problem, a possible future work could focus on other cloud providers such as AWS and Azure. This could provide insight on the robustness of the models developed in this study, but also investigate if other providers could present another set of information that could be used to train models. There is also the possibility to use more than just one source of data, in this study the only source of data was from GCP. In the future, a combination of multiple sources could use more advanced relationships, use a broader set of variables, and provide a broader view on the subject.



Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[3] Inés Sittón Candanedo, Elena Hernández Nieves, Sara Rodríguez González, M Teresa Santos Martín, and Alfonso González Briones. "Machine learning predictive model for industry 4.0". In: *Knowledge Management in Organizations: 13th International Conference*. Springer. 2018, pp. 501–510.

[4] Google Cloud. *Google Cloud Documentation*. `https://cloud.google.com/docs`. Accessed: 2024-12-12.

[5] Hybrid Cloud. "The nist definition of cloud computing". In: *National institute of science and technology, special publication* 800.2011 (2011), p. 145.

[6] Nelson Fumo and MA Rafe Biswas. "Regression analysis for prediction of residential energy consumption". In: *Renewable and sustainable energy reviews* 47 (2015), pp. 332–343.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[8] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020), pp. 357–362.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] Mohammad Hossin and Md Nasir Sulaiman. "A review on evaluation metrics for data classification evaluations". In: *International journal of data mining & knowledge management process* 5.2 (2015), p. 1.

[11] John D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95.

[12] Wes McKinney et al. "pandas: a foundational Python library for data analysis and statistics". In: *Python for high performance and scientific computing* 14.9 (2011), pp. 1–9.

[13] David J Montana, Lawrence Davis, et al. "Training feedforward neural networks using genetic algorithms." In: *IJCAI*. Vol. 89. 1989. 1989, pp. 762–767.

[14] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[15] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[16] Victor Rodriguez-Galiano, Manuel Sanchez-Castillo, M Chica-Olmo, and MJOGR Chica-Rivas. "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines". In: *Ore Geology Reviews* 71 (2015), pp. 804–818.

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[18] Mark Segal and Yuanyuan Xiao. "Multivariate random forests". In: *Wiley interdisciplinary reviews: Data mining and knowledge discovery* 1.1 (2011), pp. 80–87.

[19] Jayachander Surbiryala and Chunming Rong. "Cloud computing: History and overview". In: *2019 IEEE Cloud Summit*. IEEE. 2019, pp. 1–7.

[20] Sanaz Tayefeh Hashemi, Omid Mahdi Ebadati, and Harleen Kaur. "Cost estimation and prediction in construction projects: A systematic review on machine learning techniques". In: *SN Applied Sciences* 2.10 (2020), p. 1703.

[21] TensorFlow. *TensorFlow API Documentation*. Accessed: 2025-02-27. 2024. URL: https://www.tensorflow.org/api_docs/python/tf.

[22] Hong-Linh Truong and Schahram Dustdar. "Composable cost estimation and monitoring for computational applications in cloud computing environments". In: *Procedia Computer Science* 1.1 (2010), pp. 2175–2184.

[23] Peiran Wang. "Predictive Machine Learning Models of Methane Emissions Using Farm Environmental Data". In: *Proceedings of the 2023 5th International Conference on Internet of Things, Automation and Artificial Intelligence*. 2023, pp. 881–887.

[24] Wenjuan Wang, Martin Kiik, Niels Peek, Vasa Curcin, Iain J Marshall, Anthony G Rudd, Yanzhong Wang, Abdel Douiri, Charles D Wolfe, and Benjamin Bray. "A systematic review of machine learning models for predicting outcomes of stroke with structured data". In: *PloS one* 15.6 (2020), e0234722. DOI: https://doi.org/10.1371/journal.pone.0234722.

A Appendix for all SQL queries used.

```sql
WITH active_projects_2022 AS (
  SELECT
    project.id AS project_id
  FROM
    `my_billing_dataset.gcp_billing_export`
  WHERE
    DATE(usage_start_time) BETWEEN '2022-01-01' AND '2022-12-31'
  GROUP BY
    project_id
  HAVING
    MIN(usage_start_time) <= '2022-12-31'
)
```

Listing A.1: SQL Query for Identifying Active Projects

```sql
, filtered_data AS (
  SELECT
    TIMESTAMP_TRUNC(usage_start_time, HOUR) AS hour,
    service.description AS service,
    project.id AS project_id,
    SUM(cost) AS hourly_cost,
    SUM(usage.amount) AS hourly_usage,
  FROM
    `my_billing_dataset.gcp_billing_export`
  WHERE
    DATE(usage_start_time) BETWEEN '2023-01-01' AND '2024-09-30'
    AND project.id IN (SELECT project_id FROM active_projects_2022)
    AND service.description IN ('Compute Engine', 'Cloud SQL', 'Cloud Storage', '
        Cloud Firestore', 'Artifact Registry')
  GROUP BY
    hour, service, project_id
)
```

Listing A.2: SQL Query for aggregate cost data only for active projects

```sql
, growth_rates AS (
  SELECT
    project_id,
    service,
    EXTRACT(MONTH FROM hour) AS month,
    AVG(hourly_usage) AS avg_monthly_usage
  FROM
    filtered_data
  GROUP BY
    project_id, service, month
)

-- Final Output
SELECT
  fd.hour,
  fd.service,
  fd.project_id,
  fd.hourly_cost,
  fd.hourly_usage,
  gr.avg_monthly_usage
FROM
  filtered_data fd
LEFT JOIN
  growth_rates gr
ON
  fd.project_id = gr.project_id AND fd.service = gr.service AND EXTRACT(MONTH FROM
      fd.hour) = gr.month
ORDER BY
  fd.hour, fd.service, fd.project_id;
```

Listing A.3: SQL Query on monthly average and final output

```sql
WITH active_projects_2022 AS (
  SELECT
    project.id AS project_id
  FROM
    `Billing_table`
  WHERE
    DATE(usage_start_time) BETWEEN '2022-01-01' AND '2022-12-31'
  GROUP BY
    project.id
  HAVING
    MIN(usage_start_time) <= '2022-12-31'
),
filtered_data AS (
  SELECT
    TIMESTAMP_TRUNC(usage_start_time, HOUR) AS hour,
    service.description AS service,
    project.id AS project_id,
    SUM(cost) AS hourly_cost,
    SUM(usage.amount) AS hourly_usage
  FROM
    `Billing_table`
  WHERE
    DATE(usage_start_time) BETWEEN '2024-04-01' AND '2024-09-30'
    AND project.id IN (
      SELECT
        all_projects.project_id
      FROM
        (SELECT DISTINCT project.id AS project_id
         FROM `Billing_table`) AS all_projects
       LEFT JOIN
        active_projects_2022
       ON
        all_projects.project_id = active_projects_2022.project_id
       WHERE
        active_projects_2022.project_id IS NULL
    )
    AND service.description IN ('Compute Engine', 'Cloud SQL', 'Cloud Storage', 'Cloud Firestore'
        , 'Artifact Registry')
  GROUP BY
    hour, service, project_id
),
growth_rates AS (
  SELECT
    project_id,
    service,
    EXTRACT(MONTH FROM hour) AS month,
    AVG(hourly_usage) AS avg_monthly_usage
  FROM
    filtered_data
  GROUP BY
    project_id, service, month
)

-- Final Output
SELECT
  fd.hour,
  fd.service,
  fd.project_id,
  fd.hourly_cost,
  fd.hourly_usage,
  gr.avg_monthly_usage
FROM
  filtered_data fd
LEFT JOIN
  growth_rates gr
ON
  fd.project_id = gr.project_id AND fd.service = gr.service AND EXTRACT(MONTH FROM fd.hour) = gr.
      month
ORDER BY
  fd.hour, fd.service, fd.project_id;
```

Listing A.4: SQL Query retrieving the test data set