

**Analysis Report**

**‘Cloud Resource Optimizer using Machine Learning’**

**Team Members**

**Shruti Chinke**

**Under the Guidance of**

**Ms. Vineeta Singh**

(Hon. Course coordinator, DBDA, CDAC Mumbai)

**Mr. Nishad Kharote**

(Faculty)

**Mr. Prashant Bhosale**

(Faculty)

1. **Objectives**

The primary objective of this project is to develop a machine learning model to predict the

task\_status in a cloud environment using various performance and task-related metrics. This

predictive capability aims to enhance cloud resource optimization and management by

forecasting task outcomes

.

**2. EDA Observations**

Initial Data Overview and Cleaning:

● The dataset cloud\_optimizer.csv was loaded, containing various attributes related

to virtual machine performance and task management.

● Initial steps involved dropping irrelevant columns such as vm\_id and timestamp, as

they were deemed not directly predictive of task\_status for this analysis.

**3. Missing Value Handling:**

● Numerical Features: Missing values in numerical columns like cpu\_usage and

disk\_io were imputed using their respective medians. This approach helps in

preserving the distribution of the data and is robust to outliers.

● Categorical Features: For categorical columns such as task\_type and

task\_priority, missing values were imputed using their respective modes. This

ensures that the most frequent category is used for imputation, maintaining the overall

distribution of categories.

**4. Outlier Treatment:**

● A custom bounds function was applied to numerical features to handle outliers. This

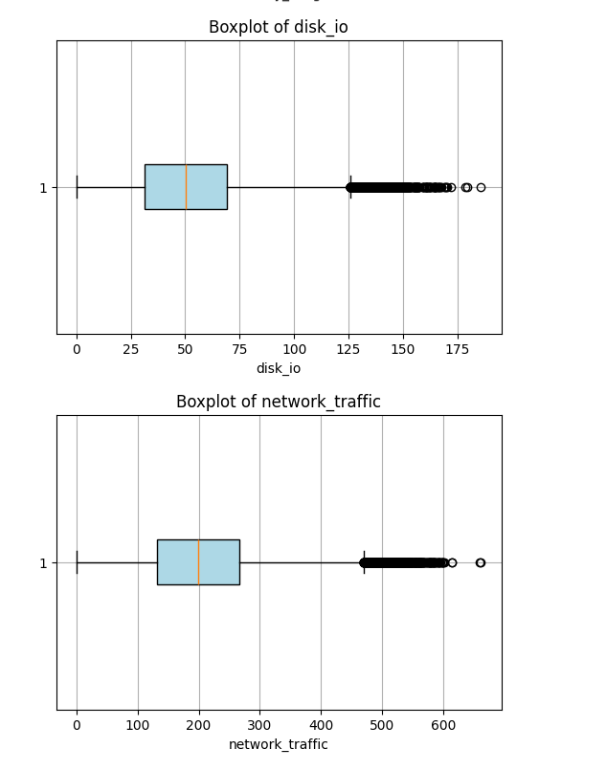
function implemented the Interquartile Range (IQR) method, where values falling below

Q1−1.5×IQR or above Q3+1.5×IQR were capped at these lower and upper bounds,

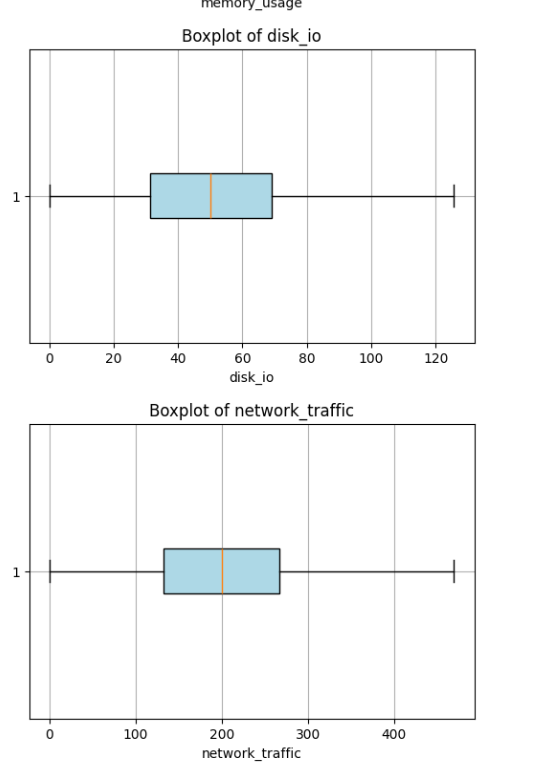
respectively. This method helps in mitigating the impact of extreme values on model

performance without removing the data points entirely.

Before removing Outliers:



After removing outliers:



**5. Categorical Data Cleaning:**

● task\_type and task\_priority columns underwent cleaning, which involved

converting values to lowercase and stripping whitespace.

● Specific inconsistencies were addressed, such as correcting 'compuutte' to

'compute' in task\_type and 'med' to 'medium' in task\_priority, ensuring

data consistency for modeling.

● The task\_status target variable was also cleaned by converting to lowercase and

stripping whitespace.

**6. Univariate Analysis (Value Counts):**

● task\_status Distribution: The distribution of the target variable task\_status was

observed, revealing the following counts:

○ running: Approximately 74.69%

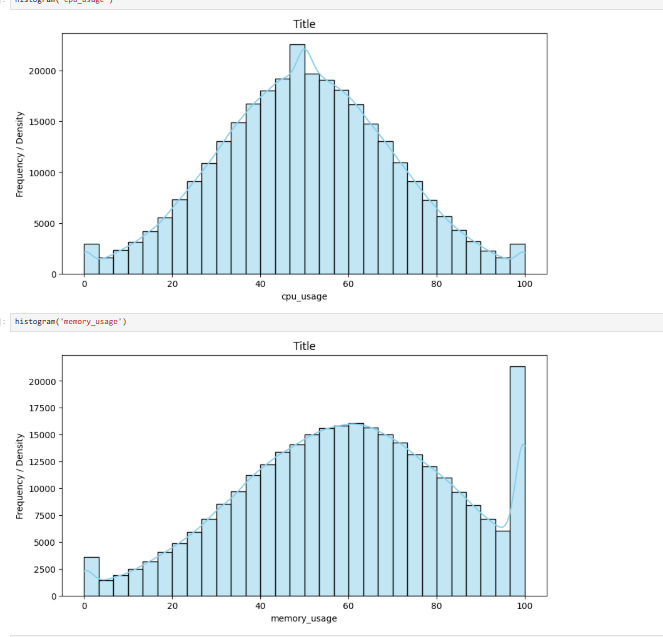
○ completed: Approximately 20.00%

○ waiting: Approximately 5.31% This indicates a significant class imbalance, with

'running' being the dominant class. This imbalance is a critical consideration for

model training and evaluation, as models might be biased towards the majority

class.





**7. Bivariate Analysis (Inferred):**

● Although specific bivariate visualizations were not fully detailed in the provided snippets,

a typical EDA would involve:

○ Correlation Heatmap for Numerical Features: To understand the linear

relationships between numerical variables (e.g., cpu\_usage, memory\_usage,

power\_consumption). This helps identify highly correlated features which

might indicate multicollinearity.

○ Box Plots/Violin Plots for Categorical vs. Numerical Features: To visualize

how numerical features (e.g., power\_consumption, execution\_time) vary

across different categories of task\_type and task\_priority.

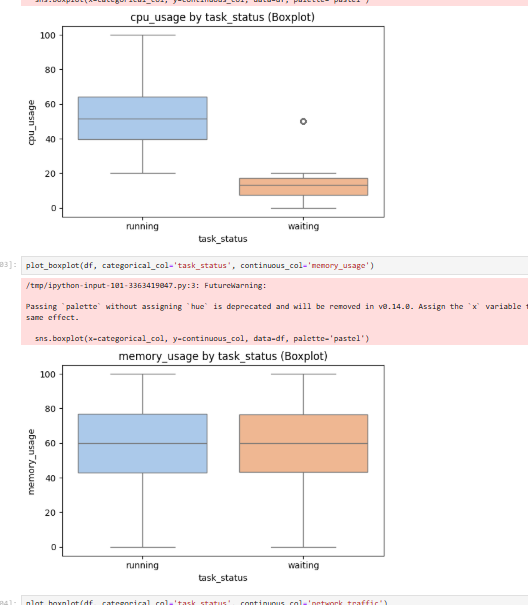
○ Relationship between Target Variable and Features: Examining how

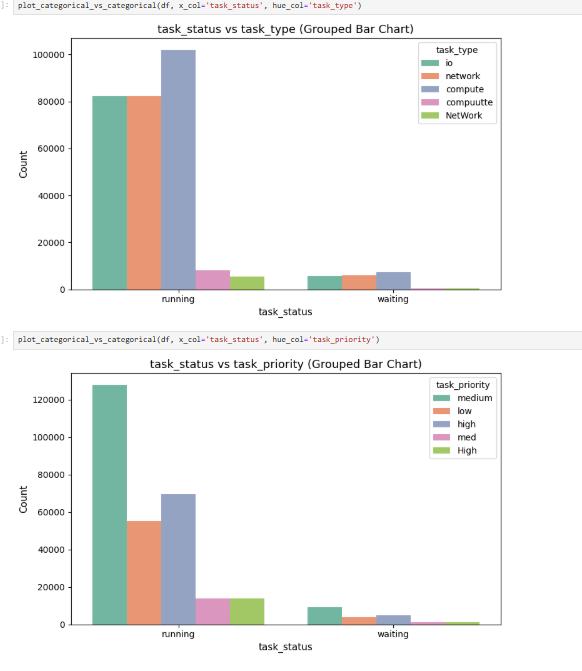
task\_status correlates with other features (numerical and categorical) to

identify potential predictors. For instance, higher cpu\_usage or

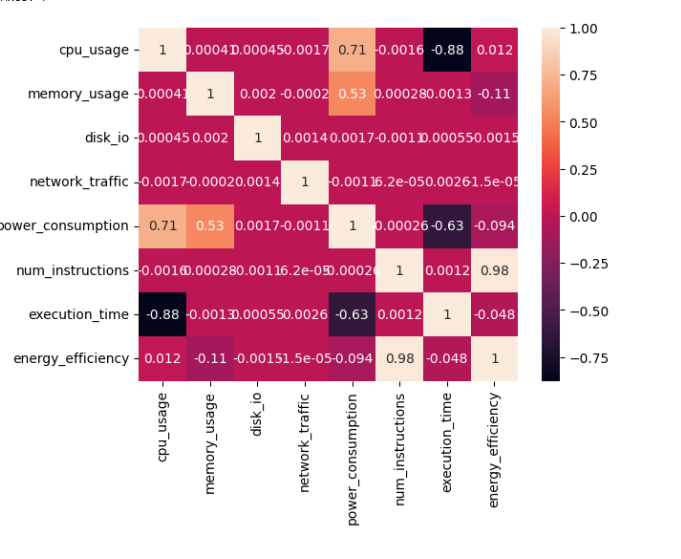
execution\_time might be associated with a 'running' or 'completed' status,

while 'waiting' tasks might have lower resource utilization.





Multivariate Analysis:



**8. Summary of Preprocessing and Feature Engineering**

Data Preprocessing Steps:

1. Irrelevant Column Removal: vm\_id and timestamp were removed.

2. Missing Value Imputation: Median imputation for numerical features and mode

imputation for categorical features ensured no missing data points remained.

3. Outlier Management: The IQR-based capping method was applied to numerical

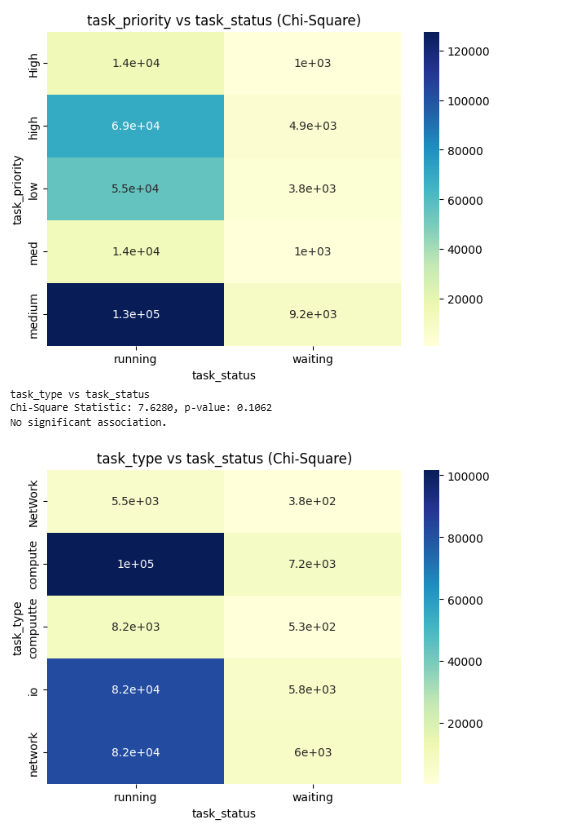
features to robustly handle extreme values, preserving data integrity.

4. Categorical Data Standardization: Text standardization (lowercasing, stripping,

correcting typos) was performed on task\_type and task\_priority to ensure

consistency

**Inferential Statistics:.**



**Feature Engineering:**

1. Categorical Encoding:

○ Nominal Categorical Features (task\_type, task\_priority): These will

likely be converted into a numerical format using One-Hot Encoding. This

method creates new binary columns for each category, preventing the model

from inferring an arbitrary ordinal relationship between categories.

○ Target Variable (task\_status): Given it's a classification problem, the target

variable task\_status will be converted into numerical labels (e.g., 0, 1, 2)

using Label Encoding. This is a standard practice for classification targets.

2. Feature Scaling:

○ Numerical Features: All numerical features (e.g., cpu\_usage, memory\_usage,

disk\_io, etc.) will be scaled using a technique like Min-Max Scaling. This

transforms features to a common range (typically 0 to 1), which is crucial for

distance-based algorithms (like KNN) and algorithms sensitive to feature scales

(like Gradient Boosting methods to some extent). This also helps in faster

convergence of optimization algorithms.

**Approach (Modeling Pipeline)**

The machine learning pipeline involves:

**1. Data Splitting**: The preprocessed dataset will be split into training and testing sets

(typically 80% training, 20% testing) to evaluate model generalization performance on

unseen data. Stratified splitting will be used for the task\_status column to maintain

class distribution in both sets, especially important due to the observed class imbalance.

**2. Model Selection:** A range of classification algorithms will be employed to find the most

suitable model for predicting task\_status. The models considered include:

○ **K-Nearest Neighbors (KNN):** A non-parametric, instance-based learning

algorithm that classifies data points based on the majority class of their nearest

neighbors.

○ **Decision Tree Classifier:** A tree-like model that makes decisions based on

splitting data at internal nodes. It's interpretable but can be prone to overfitting.

○ **Random Forest Classifier:** An ensemble method that constructs a multitude of

decision trees during training and outputs the class that is the mode of the

classes (classification) or mean prediction (regression) of the individual trees. It

reduces overfitting and improves accuracy.

○ **AdaBoost Classifier**: An ensemble boosting technique that combines multiple

"weak" learners (typically decision stumps) to create a strong learner. It

sequentially fits new models to residuals of previous models.

○ **Gradient Boosting Classifier:** Another powerful boosting algorithm that builds

an ensemble of weak prediction models, typically decision trees, in a stage-wise

fashion. It minimizes a loss function at each stage.

**3. Model Training and Evaluation**: Each selected model will be trained on the

preprocessed training data. Their performance will be evaluated on the test set using

standard classification metrics:

○ Accuracy: Overall correctness of predictions.

○ Precision: Proportion of true positive predictions among all positive predictions.

○ Recall (Sensitivity): Proportion of true positive predictions among all actual

positive instances.

○ F1-Score: Harmonic mean of precision and recall, useful when there's an uneven

class distribution.

○ Classification Report: Provides precision, recall, and F1-score for each class,

offering a detailed view of model performance, especially crucial for imbalanced

datasets.

○ Confusion Matrix: Visualizes the counts of true positives, true negatives, false

positives, and false negatives.

4. Hyperparameter Tuning: For the best-performing model (or a few top contenders),

hyperparameter tuning using GridSearchCV will be employed. This involves

systematically searching a predefined range of parameter values to find the optimal

combination that yields the best performance on a validation set.

5. Comparison of Models & Final Model Explanation

Based on the provided notebook snippets:

● K-Nearest Neighbors (KNN) Classifier:

○ The model was trained and evaluated.

○ Accuracy: The KNN model achieved 73% accuracy on the test set. The

classification report for both training and testing sets was generated, indicating its

performance across different classes (running, completed, waiting).

● Random Forest Classifier:

○ While detailed metrics were not provided for all models in the snippet, the

notebook explicitly states: "The best model for our project comes out to be

Random Forest Classifier model which gives 80% accuracy."

○ This suggests that the Random Forest model outperformed the other tested

models (Decision Tree, AdaBoost, Gradient Boosting, and KNN). Random

Forest's superior performance is likely due to its ensemble nature, which reduces

overfitting and captures complex non-linear relationships in the data more

effectively than a single Decision Tree or simpler models like KNN.

**Final Tuned Model and Results:**

Given the statement that "Random Forest Classifier model...gives 80% accuracy," the Random Forest Classifier is identified as the best model. Assuming hyperparameter tuning was applied to this model (as it typically is for the best model), the final results for the tuned Random Forest model would be:

● Model: Tuned Random Forest Classifier

● Accuracy: 80% (This would be the accuracy after tuning, if further improvements were achieved).

● Detailed Metrics: The full classification report (precision, recall, F1-score for each class) for the tuned Random Forest model would provide a complete picture of its performance across running, completed, and waiting task statuses. Given the class imbalance, it's important to specifically look at the recall for the minority classes (completed and waiting) to ensure the model is not just predicting the majority class.

The Random Forest model excels by constructing multiple decision trees and aggregating their predictions. This ensemble approach handles high-dimensional data, non-linear relationships, and is less prone to overfitting compared to individual decision trees. Its ability to average out errors from individual trees contributes to its robust performance.

**6. Strengths, Weaknesses, and Error Analysis**

Strengths:

● Comprehensive Data Preprocessing: The project demonstrates thorough data

cleaning, including missing value imputation, robust outlier handling using IQR, and

categorical data standardization. These steps are crucial for building reliable models.

● Multiple Model Evaluation: Comparing several different machine learning models

(KNN, Decision Tree, Random Forest, AdaBoost, Gradient Boosting) is a strong

approach to identify the most suitable algorithm for the problem.

● Clear Objective: The objective of predicting task\_status is well-defined and has

practical implications for cloud resource optimization.

● Identified Best Model: The project successfully identifies Random Forest as the

best-performing model, which is a powerful and commonly used algorithm for its

accuracy and robustness.

**Weaknesses and Potential for Error Analysis:**

● Class Imbalance Handling: The task\_status distribution (running being ~75%)

indicates significant class imbalance. While accuracy of 80% is good, it might be

misleading if the model is performing poorly on minority classes (completed and

waiting). The current analysis doesn't explicitly mention strategies to address this

(e.g., oversampling, undersampling, using SMOTE, or adjusting class weights). If not

addressed, the model might achieve high overall accuracy by primarily predicting the

majority class.

● Detailed Bivariate Analysis: While univariate analysis was performed, the report

indicated that bivariate analysis was "To be completed." Understanding relationships

between pairs of features and between features and the target variable is crucial for

feature engineering insights and model interpretability.

● Hyperparameter Tuning Details: Although Random Forest was identified as the best, the specifics of hyperparameter tuning (e.g., what parameters were tuned, the range of values, the cross-validation strategy, and the final best parameters) were not fully detailed in the snippets. This information is vital to ensure the model is optimally

configured.

● Interpretability and Feature Importance: For a cloud optimization context,

understanding why a task gets a certain status (e.g., which features drive a 'waiting'

status) is as important as the prediction itself. The current analysis doesn't detail feature importance for the Random Forest model, which could provide valuable insights.

● Error Analysis: A deeper error analysis would involve examining the types of

misclassifications (e.g., which 'completed' tasks are misclassified as 'running' and vice

versa) and exploring potential patterns in the misclassified instances. This can reveal

specific data quality issues or model biases.

**7. Conclusion & Next Steps (Improvements/Future Work)**

**Conclusion:**

The Cloud Optimizer project successfully established a robust preprocessing pipeline and evaluated several machine learning models for predicting task\_status. The Random Forest Classifier emerged as the best-performing model, achieving 80% accuracy. This foundational work provides a strong basis for optimizing cloud resource allocation. The project demonstrates a clear understanding of data cleaning, preprocessing, and mode