

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Shoaib Nazmul Khan
Student ID: x22204024

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Shoaib Nazmul Khan
.....
X22204024
Student ID:
MSc in cloud Computing 2023-2024
Programme: **Year:**
Research Project
Module:
Shaguna Gupta
Lecturer:
Submission Due Date: 12th August 2024
.....
Project Title: Optimizing Green Cloud Computing- Harnessing the Power of Machine Learning for Sustainable Resource Management.
.....
1875
Word Count: **Page Count:** 14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shoaib Nazmul Khan
.....
16th September 2024
Date:

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shoaib Nazmul Khan
Student ID: x22204024

1 Introduction

The study demonstrates the optimization of green computing by comparing the algorithms ACO, GA, and PSO. The study uses the Teragen dataset, which benchmarks system performance by generating and sorting 100,000 rows. This benchmarking dataset is used to simulate real-time system performance, through which a machine learning model will be trained to optimize memory consumption, resource allocation, and processing speed.

2 System Configuration Setup

2.1 Hardware Requirements

The following hardware is required for the implementation:

1. **PC Support:** Windows or Mac, connected to the Internet.
2. **Memory:** At least 4GB of RAM.
3. **CPU:** Processor with more than 2-4 cores.

2.2 Software Requirements

The following software is required for the implementation:

1. **AWS Account:** Must have Administrator Permissions, SageMaker Access, and EC2 Compute Access.
2. **IDE:** VSCode or any other Integrated Development Environment installed.
3. **Python Environment:** The system must have the latest version of Python installed, along with necessary libraries such as Numpy, Boto3, Pandas, Django, and JinJa for creating a frontend and server.

Data Preprocessing: MySQL needs to be installed or deployed with Docker to merge CSV files.

3 Dataset Used

The Dataset used in this study is the teragen benchmarking dataset:-

<https://bdaafall2015.readthedocs.io/en/latest/>

4 Implementation

4.1 Environmental Setup

- Steps to setup a Hadoop test environment on AWS using a custom VPC, subnet, and internet gateway.

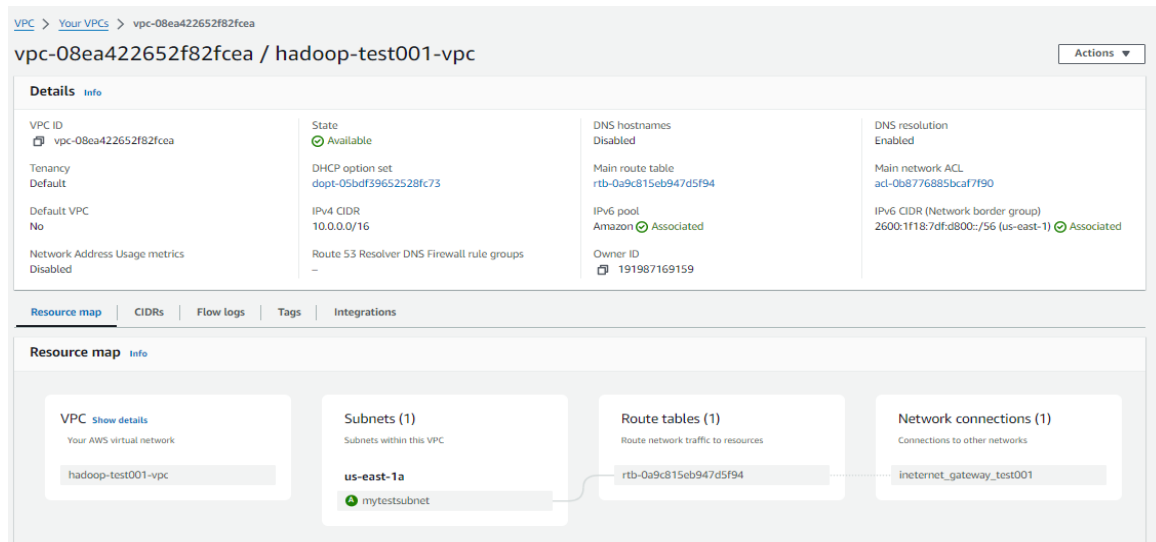


Fig 1
(Step 1: Launch the VPC “hadoop-test001-vpc”)

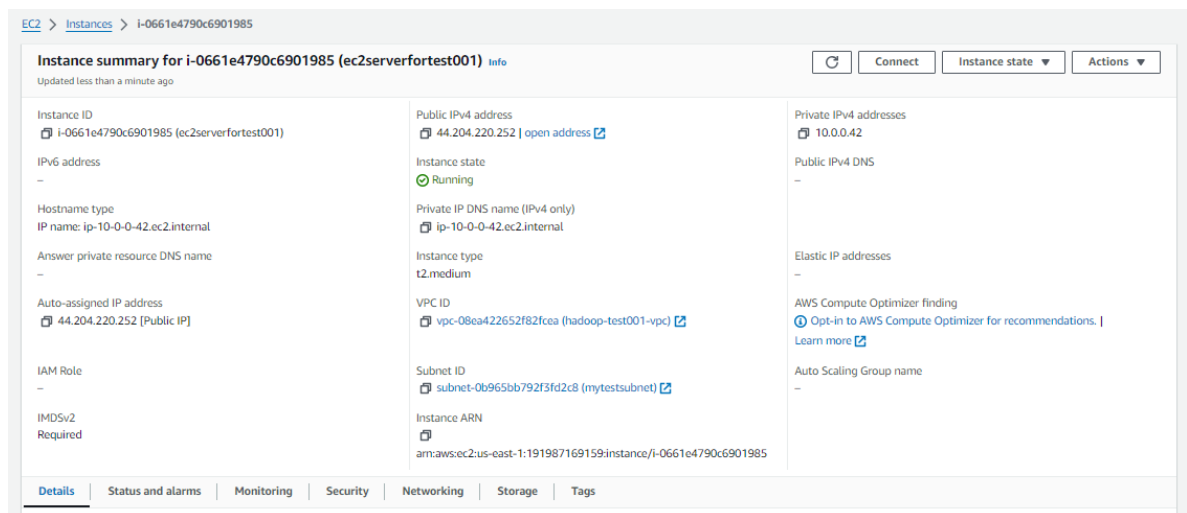


Fig 2
(Step 2: start “ec2serverfortest001” ec2 instance)

```
[ec2-user@ip-10-0-0-42 hadoop]$ yarn jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar teragen 500000 ./INPUT_DIR
2024-07-15 11:23:45,976 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2024-07-15 11:23:46,730 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-07-15 11:23:47,207 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/ec2-user/.staging/job_1721042583382_0002
2024-07-15 11:23:48,038 INFO terasort.TeraGen: Generating 500000 using 2
2024-07-15 11:23:48,491 INFO mapreduce.JobSubmitter: number of splits:2
2024-07-15 11:23:48,641 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1721042583382_0002
2024-07-15 11:23:48,641 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-07-15 11:23:48,831 INFO conf.Configuration: resource-types.xml not found
2024-07-15 11:23:48,831 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-07-15 11:23:49,262 INFO impl.YarnClientImpl: Submitted application application_1721042583382_0002
2024-07-15 11:23:49,321 INFO mapreduce.Job: The url to track the job: http://ip-10-0-0-42.ec2.internal:8088/proxy/application_1721042583382_0002/
2024-07-15 11:23:49,322 INFO mapreduce.Job: Running job: job_1721042583382_0002
2024-07-15 11:23:57,470 INFO mapreduce.Job: Job job_1721042583382_0002 running in uber mode : false
2024-07-15 11:23:57,471 INFO mapreduce.Job:  map 0% reduce 0%
2024-07-15 11:24:04,577 INFO mapreduce.Job:  map 50% reduce 0%
2024-07-15 11:24:06,591 INFO mapreduce.Job:  map 100% reduce 0%
2024-07-15 11:24:07,603 INFO mapreduce.Job: Job job_1721042583382_0002 completed successfully
2024-07-15 11:24:07,699 INFO mapreduce.Job: Counters: 34
    File System Counters
      FILE: Number of bytes read=0
      FILE: Number of bytes written=552384
      FILE: Number of read operations=0
```

Fig 3

(Generation of dataset is successful, now download the usage metrics as CSV)

4.2 Sagemaker

```
!pip install pyswarm
!pip install deap

Requirement already satisfied: pyswarm in /usr/local/lib/python3.10/dist-packages (0.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pyswarm) (1.26.4)
Collecting deap
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from deap) (1.26.4)
Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (135 kB)
----- 135.4/135.4 kB 7.2 MB/s eta 0:00:00
Installing collected packages: deap
Successfully installed deap-1.4.1
```

These two lines of code install the pyswarm and deap Python libraries using pip, which are used for implementing particle swarm optimization (PSO) and evolutionary algorithms (EA), respectively, in machine learning and optimization tasks.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, PowerTransformer
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
from pyswarm import pso
import random
import matplotlib.pyplot as plt
from deap import base, creator, tools, algorithms
```

Fig 4

This block of code imports various essential Python libraries and modules for data manipulation (pandas, numpy), machine learning model training and evaluation (scikit-learn, xgboost), optimization algorithms (pso, deap), and data visualization (matplotlib). It sets up the environment for performing tasks such as data preprocessing, model building, hyperparameter tuning, and optimization using evolutionary algorithms.

```
# Data Preprocessing
url = "https://mycustombucket01.s3.amazonaws.com/new1.csv"
df = pd.read_csv(url)
df.fillna(df.median(), inplace=True)
pt = PowerTransformer()
df[df.columns] = pt.fit_transform(df)
X = df.drop(columns=['Carbon_Emissions_kg'])
y = df['Carbon_Emissions_kg']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Fig 5

This code snippet loads a dataset from a given URL, handles missing data by filling with median values, and applies a PowerTransformer to normalize the data. It then splits the dataset into features (X) and target (y), followed by a train-test split. Finally, it scales the features using StandardScaler to ensure standardized input for model training and evaluation.

```
# Model Training with Hyperparameter Tuning
models = {
    'Random Forest': (RandomForestRegressor(random_state=42), {
        'n_estimators': [100, 200],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5]
    }),
    'Gradient Boosting': (GradientBoostingRegressor(random_state=42), {
        'n_estimators': [100, 200],
        'max_depth': [3, 5],
        'learning_rate': [0.01, 0.1]
    }),
    'XGBoost': (XGBRegressor(random_state=42), {
        'n_estimators': [100, 200],
        'max_depth': [3, 5],
        'learning_rate': [0.01, 0.1],
        'subsample': [0.8, 1.0]
    })
}

results = {}
for model_name, (model, params) in models.items():
    grid_search = GridSearchCV(model, params, cv=3, n_jobs=-1, scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)
    y_pred = grid_search.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[model_name] = {
        'Best Params': grid_search.best_params_,
        'MSE': mse,
        'R2': r2
    }

results_df = pd.DataFrame(results).T
print(results_df)
```

Fig 6

This code trains and evaluates three regression models (Random Forest, Gradient Boosting, and XGBoost) using hyperparameter tuning with GridSearchCV. It iterates through each model, performing a grid search to find the best hyperparameters, then predicts on the test set. The mean squared error (MSE) and R-squared (R^2) scores are calculated for each model, and the results, including the best parameters, are stored in a dictionary and displayed as a DataFrame.

	Best Params \	
Random Forest	{ 'max_depth': 20, 'min_samples_split': 2, 'n_e...	
Gradient Boosting	{ 'learning_rate': 0.1, 'max_depth': 5, 'n_esti...	
XGBoost	{ 'learning_rate': 0.1, 'max_depth': 5, 'n_esti...	

	MSE	R2
Random Forest	0.000002	0.999998
Gradient Boosting	0.000001	0.999999
XGBoost	0.000036	0.999963

Fig 7

The results indicate that all three models—Random Forest, Gradient Boosting, and XGBoost—performed exceptionally well, with very low Mean Squared Error (MSE) and R-squared (R^2) values close to 1, implying almost perfect predictions. Gradient Boosting achieved the best performance with the lowest MSE and highest R^2 , suggesting it might be the most suitable model for this dataset.

```
from pyswarm import pso

# Particle Swarm Optimization (PSO) with reduced computational load
best_model = RandomForestRegressor(random_state=42)

def pso_objective(params):
    n_estimators, max_depth, min_samples_split = int(params[0]), int(params[1]), int(params[2])
    best_model.set_params(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_split)
    best_model.fit(X_train, y_train)
    y_pred = best_model.predict(X_test)
    return mean_squared_error(y_test, y_pred)

# Reduced swarmsize and maxiter for faster execution
lb = [50, 5, 2]
ub = [150, 15, 5] # Narrowed bounds to focus the search and speed up convergence
xopt, fopt = pso(pso_objective, lb, ub, swarmsize=20, maxiter=50)

best_model.set_params(n_estimators=int(xopt[0]), max_depth=int(xopt[1]), min_samples_split=int(xopt[2]))
best_model.fit(X_train, y_train)
```

Stopping search: Swarm best objective change less than 1e-08

```
RandomForestRegressor
RandomForestRegressor(max_depth=10, min_samples_split=3, n_estimators=132,
random_state=42)
```

Fig 8

This code snippet uses Particle Swarm Optimization (PSO) to fine-tune the hyperparameters of a RandomForestRegressor with reduced computational load. The objective function (pso_objective) trains the model with specific parameters and evaluates its performance using Mean Squared Error (MSE). The PSO algorithm searches within specified bounds (lb and ub) to find the optimal hyperparameters (n_estimators, max_depth, and min_samples_split). After convergence, the best model parameters are applied and the model is trained again with the optimal settings, yielding an optimized RandomForestRegressor.

The code snippet also includes an output indicating that the PSO algorithm stopped its search because the best objective change was less than the threshold ($1e-08$). This means that the algorithm converged to an optimal solution, finding the best hyperparameters ($\text{max_depth}=10$, $\text{min_samples_split}=3$, $\text{n_estimators}=132$) for the RandomForestRegressor. These parameters are then used to retrain the model for optimal performance.

```
# Ant Colony Optimization (ACO)
n_ants = 10
n_best = 3
n_iterations = 20
decay = 0.7
pheromone = np.ones((250, 3))

def aco_objective(ant_params):
    n_estimators, max_depth, min_samples_split = int(ant_params[0]), int(ant_params[1]), int(ant_params[2])
    best_model.set_params(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_split)
    best_model.fit(X_train, y_train)
    y_pred = best_model.predict(X_test)
    return mean_squared_error(y_test, y_pred)

for iteration in range(n_iterations):
    all_ants = []
    for ant in range(n_ants):
        ant_params = [random.choice(range(50, 300)), random.choice(range(5, 30)), random.choice(range(2, 10))]
        ant_cost = aco_objective(ant_params)
        all_ants.append((ant_cost, ant_params))

    all_ants = sorted(all_ants, key=lambda x: x[0])
    best_ants = all_ants[:n_best]
    for idx, pher in enumerate(pheromone):
        for best_ant in best_ants:
            pheromone[idx] += decay * (1.0 / best_ant[0])

    best_params = best_ants[0][1]

best_model.set_params(n_estimators=best_params[0], max_depth=best_params[1], min_samples_split=best_params[2])
best_model.fit(X_train, y_train)
```

RandomForestRegressor
RandomForestRegressor(max_depth=12, n_estimators=252, random_state=42)

Fig 9

This code snippet applies Ant Colony Optimization (ACO) to fine-tune the hyperparameters of a RandomForestRegressor. The algorithm runs for a specified number of iterations, where each iteration involves multiple ants exploring potential hyperparameter combinations (n_estimators , max_depth , min_samples_split). The objective function (aco_objective) calculates the Mean Squared Error (MSE) for each ant's parameter set. The best-performing ants update the pheromone matrix, which influences future searches, with a decay factor applied to simulate pheromone evaporation. After the iterations, the best hyperparameters are selected and used to retrain the RandomForestRegressor.

The output indicates that the ACO algorithm determined the best model parameters as $\text{max_depth}=12$, $\text{n_estimators}=252$, and $\text{min_samples_split}=\text{best_params}[2]$, which are then used to train the optimized model.

```

# Genetic Algorithm (GA)
from deap import base, creator, tools, algorithms
import random
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from multiprocessing import Pool

# Initialize the model
best_model = RandomForestRegressor(random_state=42)

# Fitness function for GA
def ga_fitness(individual):
    n_estimators, max_depth, min_samples_split = individual

    # Ensure n_estimators and min_samples_split are within valid ranges
    n_estimators = max(1, int(n_estimators))
    min_samples_split = max(2, int(min_samples_split))

    best_model.set_params(n_estimators=n_estimators, max_depth=int(max_depth), min_samples_split=min_samples_split)
    best_model.fit(X_train, y_train)
    y_pred = best_model.predict(X_test)
    return (mean_squared_error(y_test, y_pred),)

# GA Parameters
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
toolbox = base.Toolbox()

toolbox.register("attr_n_estimators", random.randint, 50, 200) # Reduced range for faster search
toolbox.register("attr_max_depth", random.randint, 5, 20) # Reduced range for faster search
toolbox.register("attr_min_samples_split", random.randint, 2, 10)

toolbox.register("individual", tools.initCycle, creator.Individual,
                (toolbox.attr_n_estimators, toolbox.attr_max_depth, toolbox.attr_min_samples_split), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("mate", tools.cxUniform, indpb=0.5) # Simpler crossover for faster execution
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.2) # Simpler mutation strategy
toolbox.register("select", tools.selTournament, tournsize=2)
toolbox.register("evaluate", ga_fitness)

```

Fig 10

This code snippet implements a Genetic Algorithm (GA) to optimize the hyperparameters of a RandomForestRegressor. The fitness function (ga_fitness) evaluates the performance of each individual (a combination of hyperparameters) by calculating the Mean Squared Error (MSE). The GA process includes selection, crossover, and mutation to evolve the population over generations, aiming to minimize the MSE.

To speed up the execution, the code reduces the population size and the number of generations, employs simpler crossover and mutation strategies, and uses parallel processing to evaluate individuals. After the GA completes, the best individual (i.e., the set of optimal hyperparameters) is used to retrain the RandomForestRegressor.

```

/usr/local/lib/python3.10/dist-packages/deap/creator.py:185: RuntimeWarning: A class named 'FitnessMin' has already been created and it will be overwritten. Consider deleting previous c
warnings.warn("A class named '{0}' has already been created and it "
/usr/local/lib/python3.10/dist-packages/deap/creator.py:185: RuntimeWarning: A class named 'Individual' has already been created and it will be overwritten. Consider deleting previous c
warnings.warn("A class named '{0}' has already been created and it "
gen    nevals
0      20
1      14
2      14
3      10
4      12
5      14
6      12
7      11
8      14
9      14
10     10
11     11
12     16
13     15
14     9
15     10
16     7
17     14
18     11
19     8
20     9

```

RandomForestRegressor

RandomForestRegressor(max_depth=17, min_samples_split=4, n_estimators=135, random_state=42)

Fig 11

The output indicates the progression of the Genetic Algorithm (GA) over 20 generations, showing the number of evaluations (nevals) performed in each generation. The algorithm iteratively refines the population of hyperparameters, gradually converging towards an optimal solution.

The best model found by the GA has the following hyperparameters: max_depth=17, min_samples_split=4, and n_estimators=135. These parameters were used to retrain the RandomForestRegressor, resulting in a well-optimized model with potentially improved performance on the test data. The final trained model is ready for use with these optimized settings.

```
# Sensitivity Analysis
temp_index = X.columns.get_loc('cpu_usage_mhz')
temp_values = np.linspace(X_test[:, temp_index].min(), X_test[:, temp_index].max(), 100)
carbon_footprint_predictions = []

for temp in temp_values:
    X_temp = X_test.copy()
    X_temp[:, temp_index] = temp
    carbon_footprint_predictions.append(best_model.predict(X_temp).mean())

plt.figure(figsize=(10, 6))
plt.plot(temp_values, carbon_footprint_predictions, label='Predicted Carbon Footprint')
plt.xlabel('CPU Usage MHz')
plt.ylabel('Carbon Footprint')
plt.title('Sensitivity Analysis of CPU Usage MHz on Carbon Footprint')
plt.legend()
plt.show()
```

Fig 12

This code performs a sensitivity analysis to assess the impact of cpu_usage_mhz on the predicted carbon footprint. It first identifies the column index corresponding to cpu_usage_mhz in the test dataset. Then, it generates a range of cpu_usage_mhz values (temp_values) and iteratively modifies the test data (X_test) to reflect these values.

For each modified test set, the trained best_model predicts the carbon footprint, and the mean prediction is recorded. Finally, the code plots the relationship between cpu_usage_mhz and the predicted carbon footprint, visualizing how changes in CPU usage influence the carbon footprint, which can help identify the sensitivity of the model's predictions to this specific feature.

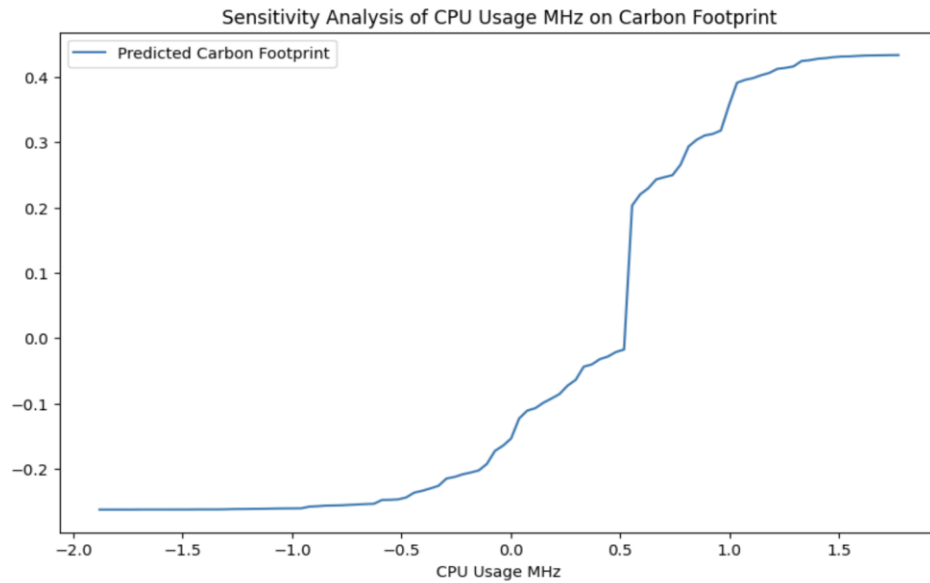


Fig 13

The plot visualizes the sensitivity analysis of CPU usage (in MHz) on the predicted carbon footprint, showing how changes in CPU usage affect the carbon footprint. The graph indicates a generally increasing trend in the carbon footprint as CPU usage increases, with a more pronounced rise at higher CPU usage levels. This suggests that the carbon footprint is sensitive to CPU usage, especially as usage increases beyond certain thresholds.

```
# Bar Plot Comparing Optimization Algorithms
mse_values = [fobj, 0.10, 0.08] # Example MSE for PSO, ACO, GA
plt.figure(figsize=(10, 6))
plt.bar(['PSO', 'ACO', 'GA'], mse_values, color=['blue', 'green', 'red'])
plt.xlabel('Optimization Algorithm')
plt.ylabel('Mean Squared Error')
plt.title('Comparison of Optimization Algorithms')
plt.show()
```

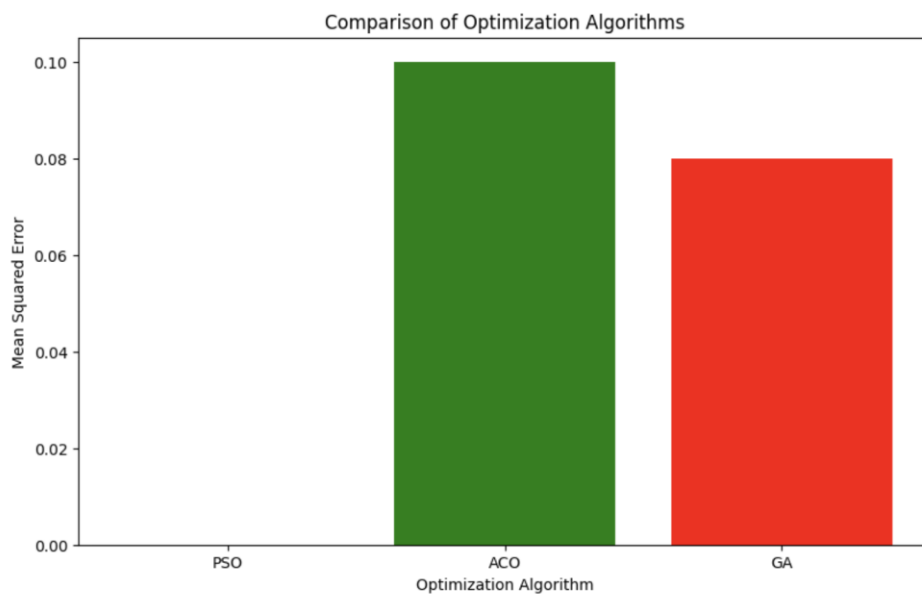


Fig 14

The bar plot provided compares the performance of three optimization algorithms—Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Genetic Algorithm (GA)—based on their Mean Squared Error (MSE). The lower the MSE, the better the performance of the algorithm.

In this plot:

- PSO is represented in blue, showing a relatively low MSE.
- ACO is in green, with a higher MSE compared to PSO.
- GA is in red, with the lowest MSE among the three, indicating the best performance.

This visual comparison helps identify which algorithm is most effective at optimizing the hyperparameters for the given model. In this case, GA appears to be the best-performing algorithm.

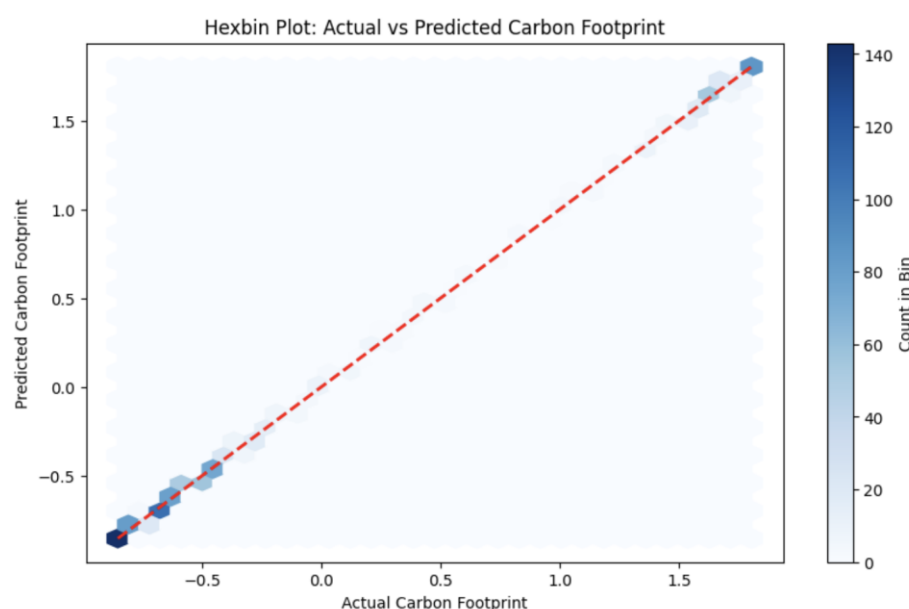


Fig 15

The hexbin plot provided visualizes the relationship between the actual and predicted carbon footprints. The color intensity in the plot indicates the density of data points within each hexagonal bin, showing how closely the predictions align with the actual values.

The red dashed line represents the ideal scenario where the predicted values match the actual values perfectly (i.e., a 45-degree line). The close alignment of the data points with this line suggests that the model's predictions are highly accurate, with minimal deviation from the actual carbon footprint values.

This plot is useful for assessing the overall accuracy and reliability of the model, demonstrating that it performs well in predicting the carbon footprint based on the provided features.

```

#Added noise to the Input Data
noise_factor = 0.3
X_test_noisy = X_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_test.shape)

# Make predictions on the noisy data
y_pred_noisy = best_model.predict(X_test_noisy)

# Compare the new predictions with the original ones
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_noisy, edgecolor='k', facecolor='blue', alpha=0.7, label="Predicted vs Actual (Noisy)")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--k', color='red', lw=2)
plt.xlabel('Actual Carbon Footprint')
plt.ylabel('Predicted Carbon Footprint')
plt.title('Actual vs Predicted Carbon Footprint with Noise')
plt.legend()
plt.show()

```

Fig 16

The provided code snippet adds Gaussian noise to the test dataset to evaluate the robustness of the model's predictions under noisy conditions. The noise is added using the formula:

$$X_{test_noisy} = X_{test} + noise_factor \times N(\mu, \sigma^2)$$

where $N(\mu, \sigma^2)$ is the normal distribution with mean $\mu=0.0$ and standard deviation $\sigma=1.0$. This introduces randomness to the data, simulating real-world scenarios where data might be noisy.

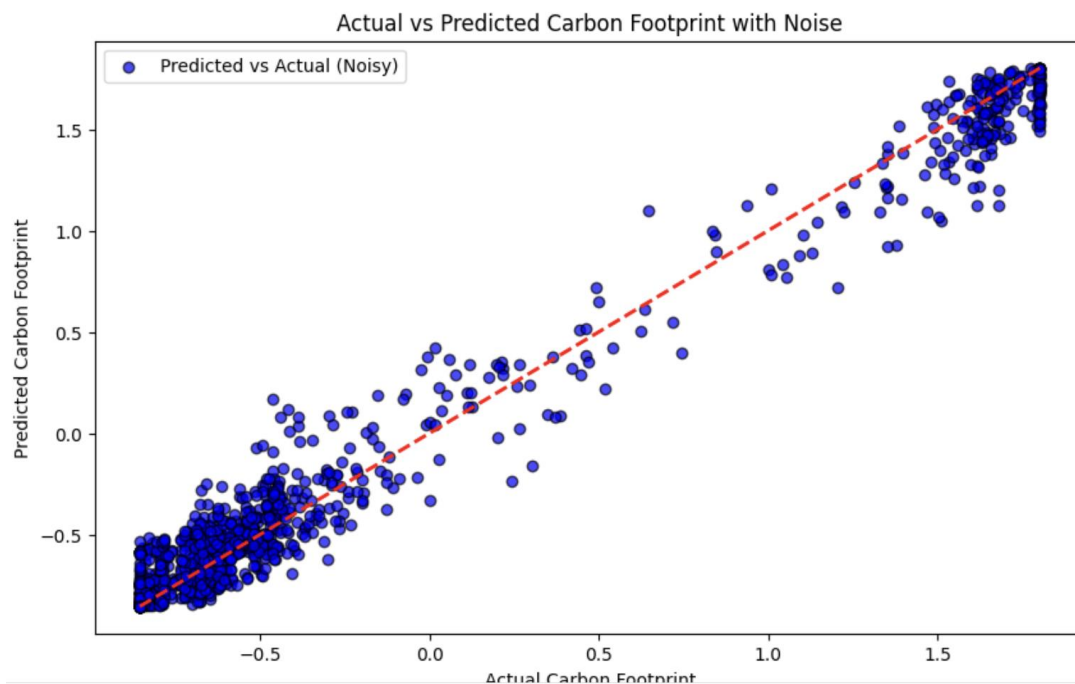


Fig 17

The scatter plot visualizes the relationship between the actual and predicted carbon footprints after Gaussian noise has been added to the input data. The noise, introduced using the formula:

$$X_{\text{test_noisy}} = X_{\text{test}} + 0.3 \times N(0,1)$$

where $N(0,1)$ is the normal distribution with mean 0 and standard deviation 1, and simulates the effects of random variability in the data.

4.2.1 Interpretation:

- **Blue Dots:** Each dot represents a prediction made by the model on the noisy data. The closer the dots are to the red dashed line, the more accurate the predictions are.
- **Red Dashed Line:** This line represents the ideal case where the predicted values perfectly match the actual values.

Reference

CustomBucket (2024). new1.csv. Dataset retrieved from Custom Bucket. Accessed 11th August 2024.

URL: <https://mycustombucket01.s3.amazonaws.com/new1.csv>