National College of Ireland

Configuration Manual

MSc Research Project
M.Sc. Data Analytics

Aryan Singh
Student ID: 21205523

School of Computing
National College of Ireland

Supervisor: Prof. Dr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Aryan Singh
Student ID:	21205523
Programme:	M.Sc. Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Dr. Hicham Rifai
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	769
Page Count:	5

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:	
Date:	18th September 2023

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

Aryan Singh
21205523

1 Document Structure

This document provides an overview of the code and instructions to replicate/run the code on a different system. The document is structured into:

1. Document Structure: Current section.
2. Environment Setup: Specifications of the recommended environment.
4. Defining the Simulation Model.
5. Collecting Data for training.
6. Training NN on raw simulation Data.
7. Create Graph and Vector Embedding.
8. Train GVENN on transformed data.
9. Evaluate NN and GVENN models.
10. Train and evaluate ML models.
11. Compare computation running speeds.

2 Environment Setup

All the code presented in this document has been implemented and run on a Google Colab Free tier. The free-tier colab uses an *Intel Xeon CPU with 2 vCPUs (virtual CPUs) and 13GB of RAM.*

Python=3.10 was used for this project with the default versions of the required packages. These packages should be installed with the pip command "pip install <package_name>". All the required packages have been mentioned in the beginning of the notebook.

3 Defining the Simulation Model

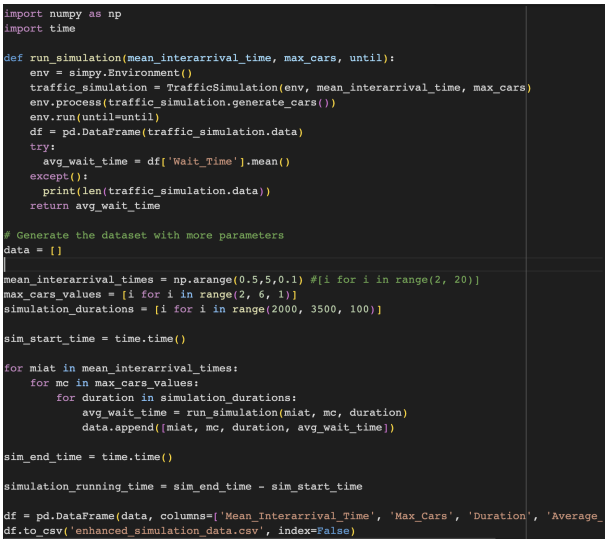
Figure 1 shows the code snippet to run a single instance of the defined simulation and then plot the relevant metrics. The parameters mean_interarrival_time, max_cars and until could be changed and run multiple times to observe the behavior of simulation runs.

```
def main():
    env = simpy.Environment()
    traffic_simulation = TrafficSimulation(env, mean_interarrival_time=1.2, max_cars=19) #st
    env.process(traffic_simulation.generate_cars())
    env.run(until=3000)  # Run the simulation until n units of time.

    # Save the data to a CSV
    df = pd.DataFrame(traffic_simulation.data)
    df.to_csv('traffic_simulation_data.csv', index=False)
    dfw = df[df['Wait_Time'] > 0 ]
    # Visualize the data
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 2, 1)
    plt.hist(dfw['Wait_Time'], bins=30, color='blue', alpha=0.7)
    plt.title('Non-Zero Wait-Time Distribution')
    plt.xlabel('Wait Time')
    plt.ylabel('Number of Cars')

    plt.subplot(1, 2, 2)
    plt.scatter(df['Arrival_Time'], df['Wait_Time'], color='red', alpha=0.7)
    plt.title('Wait Time vs Arrival Time')
    plt.xlabel('Arrival Time')
    plt.ylabel('Wait Time')

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

Figure 1: Code defining Simulation Model

4 Collecting Data for training

The code shown in Figure 2 runs multiple simulations in a loop while changing the values of the parameters in each iteration to generate the data. Modify the 3 list variables to control the range of input parameters.



```
import numpy as np
import time

def run_simulation(mean_interarrival_time, max_cars, until):
    env = simpy.Environment()
    traffic_simulation = TrafficSimulation(env, mean_interarrival_time, max_cars)
    env.process(traffic_simulation.generate_cars())
    env.run(until=until)
    df = pd.DataFrame(traffic_simulation.data)
    try:
      avg_wait_time = df['Wait_Time'].mean()
    except():
      print(len(traffic_simulation.data))
    return avg_wait_time

# Generate the dataset with more parameters
data = []

mean_interarrival_times = np.arange(0.5,5,0.1) #[i for i in range(2, 20)]
max_cars_values = [i for i in range(2, 6, 1)]
simulation_durations = [i for i in range(2000, 3500, 100)]

sim_start_time = time.time()

for miat in mean_interarrival_times:
    for mc in max_cars_values:
        for duration in simulation_durations:
            avg_wait_time = run_simulation(miat, mc, duration)
            data.append([miat, mc, duration, avg_wait_time])

sim_end_time = time.time()

simulation_running_time = sim_end_time - sim_start_time

df = pd.DataFrame(data, columns=['Mean_Interarrival_Time', 'Max_Cars', 'Duration', 'Average_
df.to_csv('enhanced_simulation_data.csv', index=False)
```

Figure 2: Generate Training Data

5 Training NN on raw simulation Data

The code shown in Figure 3 carries out the training on the NN on raw simulation data.

Figure 3: Train NN on raw simulation data.

6 Create Graph and Vector Embedding

Graph construction and Vector Embedding is carried out in Figure 4. The parameters in this section define the structure and the embeddings in the graph.

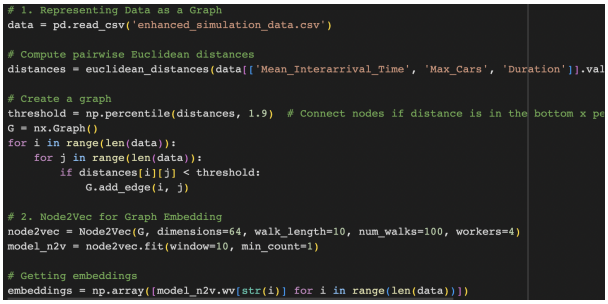


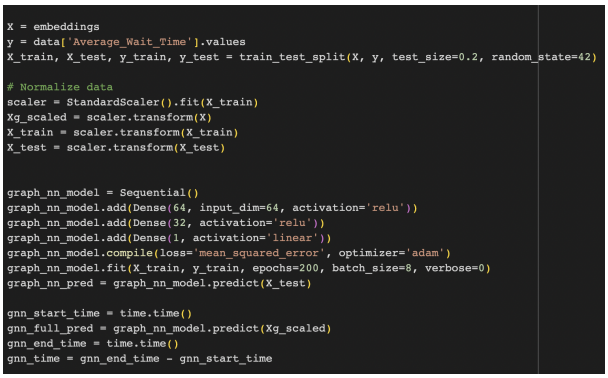
Figure 4: Graph construction and Vector Embedding

7 Train GVENN on transformed data.

A NN, GVENN with the given parameters is trained on the transformed data, Figure 5. The model could take exceptionally longer times on the machine used in this research. The time taken depends on the parameters provided in the previous section. A threshold value of less than 5 should work in cases where the number of iterations is less than 5000.

8 Train and evaluate ML models.

Two ML models, RF and XGB are also trained on the raw simulation data for comparison, shown in Figure 6. Plots showing the fit of both these models are visualized in this section.



```
X = embeddings
y = data['Average_Wait_Time'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize data
scaler = StandardScaler().fit(X_train)
Xg_scaled = scaler.transform(X)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)


graph_nn_model = Sequential()
graph_nn_model.add(Dense(64, input_dim=64, activation='relu'))
graph_nn_model.add(Dense(32, activation='relu'))
graph_nn_model.add(Dense(1, activation='linear'))
graph_nn_model.compile(loss='mean_squared_error', optimizer='adam')
graph_nn_model.fit(X_train, y_train, epochs=200, batch_size=8, verbose=0)
graph_nn_pred = graph_nn_model.predict(X_test)

gnn_start_time = time.time()
gnn_full_pred = graph_nn_model.predict(Xg_scaled)
gnn_end_time = time.time()
gnn_time = gnn_end_time - gnn_start_time
```

Figure 5: Train GVENN on transformed data

```
# Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

#Predicting on full dataset to record processing time
rf_start_time = time.time()
rf_full_pred = rf.predict(X)
rf_end_time = time.time()
rf_time = rf_end_time - rf_start_time

# XGBoost Regressor
xgb_reg = xgb.XGBRegressor(objective ='reg:squarederror')
xgb_reg.fit(X_train, y_train)
y_pred_xgb = xgb_reg.predict(X_test)

#Predicting on full dataset to record processing time
xgb_start_time = time.time()
xgb_full_pred = xgb_reg.predict(X)
xgb_end_time = time.time()
xgb_time = xgb_end_time - xgb_start_time

# Evaluation Metrics
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)
```

Figure 6: Training RF and XGB models

9 Compare computation running speeds.

The processing speeds of all the models are recorded and contrasted here for a clear comparison in performances, Figure 7.

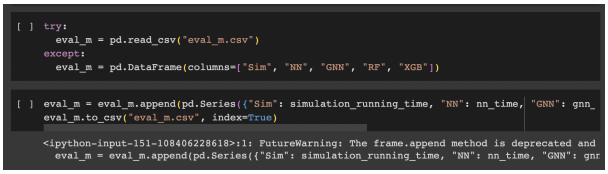


Figure 7: Record and display running times of all the models.

References