Configuration Manual

MSc Research Project
MSc In Artificial Intelligence

Balaji Dinakaran
Student ID: x22249842

School of Computing
National College of Ireland

Supervisor: Prof. Victor Del Rosal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Balaji Dinakaran
Student ID:	x22249842
Programme:	Master of Science in Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Victor Del Rosal
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Page Count:	10

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:	Balaji Dinakaran
Date:	12 August 2024

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

Balaji Dinakaran
x22249842

1 System Configuration

The project is done on 64-Bit Windows 11 pro–operating system with 8 GB RAM with Intel® Core™ i5-8259U Processor having a base clock speed of 3.30 GHz.

Item	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.19045 Build 19045
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	DESKTOP-CJA0QNI
System Manufacturer	Apple Inc.
System Model	MacBookPro15,2
System Type	x64-based PC
System SKU	
Processor	Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz, 2301 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	Apple Inc. 2020.61.1.0.0 (iBridge: 21.16.2057.0.0,0), 11-11-2023
SMBIOS Version	3.3
Embedded Controller V...	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Apple Inc.
BaseBoard Product	Mac-827FB448E656EC26
BaseBoard Version	MacBookPro15,2
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Binding Not Possible
Windows Directory	C:\Windows
System Directory	C:\Windows\system32
Boot Device	\Device\HarddiskVolume1
Locale	United Kingdom
Hardware Abstraction L...	Version = "10.0.19041.3636"
Username	DESKTOP-CJA0QNI\Balaji_Dinakaran
Time Zone	India Standard Time
Installed Physical Mem...	8.00 GB
Total Physical Memory	7.85 GB
Available Physical Mem...	2.57 GB
Total Virtual Memory	14.6 GB
Available Virtual Memory	6.63 GB

Figure 1: System Conifguration

2 Software Requirements

For the project we have used following software:

1. Python 3.11.4
2. Anaconda 2.4.3
3. VS Code
4. Jupyter Notebook

3 Python Libraries

The project uses following python libraries:

1. TensorFlow
2. Keras

3. numpy
4. matplotlib
5. pandas
6. sklearn
7. itertools
8. nltk
9. transformers
10. scipy

4 Dataset

1. The dataset is readily available at Kaggle and licensed by MIT. This dataset provides synthetic data related to vehicle maintenance to help predict whether a vehicle requires maintenance or not based on various features.
2. The model uses a dataset of 50,000 vehicles, featuring both categorical and numerical data on specifications, maintenance, and operational metrics.
3. Link - https://www.kaggle.com/datasets/vehicle-maintenance-data

5 Data Preprocessing

1. Data is cleaned and process to remove unwanted columns from the data frame.
2. The data is label-encoded to convert categorical variables into a numerical format suitable for model training.

```python
# Label encode
label_columns = ['Vehicle_Model','Maintenance_History', 'Fuel_Type', 'Transmission_Type','Tire_Condition', 'Brake_Condition', 'Battery_Status',
                 'Owner_Type']

# Custom transformer for label encoding multiple columns
def label_encode_columns(X):
    label_encoders = {col: LabelEncoder() for col in X.columns}
    return pd.DataFrame({col: label_encoders[col].fit_transform(X[col]) for col in X.columns})

# Define the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('label', FunctionTransformer(label_encode_columns, validate=False), label_columns)
    ], remainder='passthrough'
)

# Apply the transformations
df_transformed = preprocessor.fit_transform(df)

# Display the transformed DataFrame
print(df_transformed)
```

Figure 2: Label Encoded Data

3. Following label encoding, feature selection such as spearman, pearson and SelectKBase to retain the most relevant variables that significantly contribute to the model performance.

```python
#Feature Selection using spearman and pearson
# List of date columns
date_columns = ['Service_Date', 'Warranty_Expiry_Date','Service_History']
filtered_df = pred_df.drop(columns=date_columns)

# Assuming df is your DataFrame and target is your target variable
spearman_corr = filtered_df.corr(method='spearman')
pearson_corr = filtered_df.corr(method='pearson')

spearman_target_corr = spearman_corr['Need_Maintenance']
pearson_target_corr = pearson_corr['Need_Maintenance']

# Define different thresholds
low_threshold = 0.1   # Very weak to weak correlations
moderate_threshold = 0.3   # Moderate to strong correlations
high_threshold = 0.5  # Strong correlations

# Determine a threshold for significance, e.g., 0.1
significant_spearman_features = spearman_target_corr[abs(spearman_target_corr) > low_threshold].index
significant_pearson_features = pearson_target_corr[abs(pearson_target_corr) > low_threshold].index

# Combine significant features from both methods
significant_features = list(set(significant_spearman_features) | set(significant_pearson_features))
# print(significant_features)

# Threshold for multicollinearity, e.g., 0.9
threshold = 0.9
to_remove = set()

for feature in significant_features:
    for other_feature in significant_features:
        # print(other_feature)
        # print(abs(pearson_corr.loc[feature, other_feature]))
        if feature != other_feature and abs(pearson_corr.loc[feature, other_feature]) > threshold:
            to_remove.add(other_feature)

first_selected_features = [feature for feature in significant_features if feature not in to_remove]
print(first_selected_features)
#using spearman and pearson with low thershold
```

Figure 3: Feature Selection using Spearman and Pearson

```python
# Feature Selection using SelectKBest
# Select top 10 features based on chi-squared
X = filtered_df.drop(columns='Need_Maintenance')
y = filtered_df['Need_Maintenance'].astype('int')

select_k_best = SelectKBest(chi2, k=10)
X_new = select_k_best.fit_transform(X, y)

# Get the scores for each feature
feature_scores = select_k_best.scores_
feature_names = X.columns  # assuming X is a DataFrame
selected_features = feature_names[select_k_best.get_support()]

print("Selected features:", selected_features)
print("Feature scores:", feature_scores)

#training and test set based on selected features
X = pred_df[selected_features]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# print(X)
# print(y)
# filtered_df.head()
```

Figure 4: Feature Selection using SelectKBase

4. The figure 5 displays the final processed and cleaned dataset which serves as the foundation for training both supervised and deep learning algorithms.

	Vehicle_Model	Maintenance_History	Fuel_Type	Transmission_Type	Tire_Condition	Brake_Condition	Battery_Status	Owner_Type	Mileage	Reported_Issues	Engine_Size	Odometer_Reading	Service_Date	Warranty_Expiry_Date	Insurance_Premium	Service_History	Accident_History	Fuel_Efficiency	Need_Maintenance
0	4	1	1	0	1	1	2	1	58765	0	2000	28524	23-11-2023	24-06-2025	20782	6	3	13.622204	1
1	5	0	1	0	1	1	2	1	60353	1	2500	133630	21-09-2023	04-06-2025	23489	7	0	13.625307	1
2	0	2	1	0	1	0	2	0	68072	0	1500	34022	27-06-2023	27-04-2025	17979	7	0	14.306302	1
3	0	0	2	0	1	2	1	1	60849	4	2500	81636	24-08-2023	05-11-2025	6220	7	3	18.709467	1
4	0	2	2	1	0	0	2	2	45742	5	2000	97162	25-05-2023	14-09-2025	16446	6	2	16.977483	1

Figure 5: Final Label Encoded Data Frame

6 Data Analysis

1. The distribution of Need Maintenance with respect to Device whereas 1 represents maintenance required and 0 represents maintenance not required. This shows dataset contains different model with almost equivalent quantity to conduct further study.

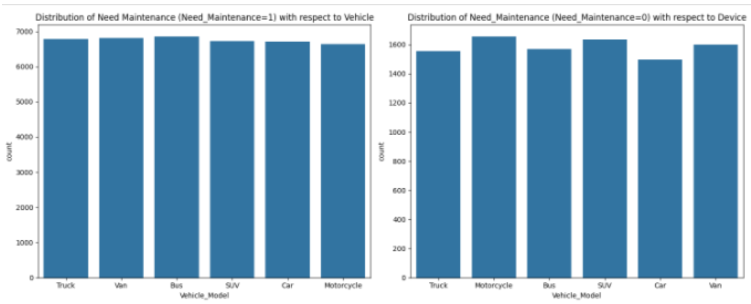


Figure 6: Distribution of Need Maintenance with respect to vehicle

2. Distribution of 'maintenance' based on different issues as hue

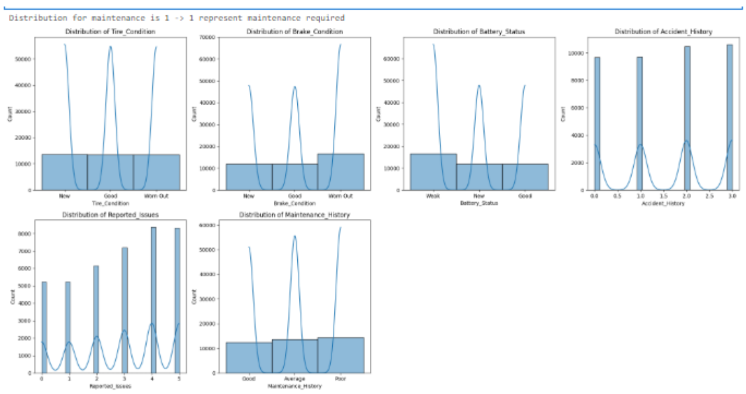


Figure 7: Distribution of 'maintenance' based on different issues as hue

7 Model Training and Testing

Vehicle maintenance prediction using supervised learning and deep learning (Neural network) algorithms.

1. Logistic Regression

```
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Predictions on the test set
y_pred = lr.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f' % (accuracy))

precision = precision_score(y_test, y_pred)
print('Precision: %.2f' % precision)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print('F1 Score: %.2f' % f1)

# Calculate recall
recall = recall_score(y_test, y_pred)
print('Recall: %.2f' % recall)

# print(y_pred)
```

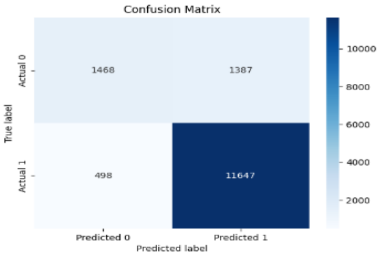
Figure 8: Model Training

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix using seaborn for better visualization
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# [[TN, FP],
#  [FN, TP]]
#TN - vechicle not required maintenance and predicted as no maintenance required (Correct)
#FP - vechicle not required maintenance but predicted as maintenance required
#FN - vechicle required maintenance but predicted as no maintenance required.
#TP - vehicle required maintenance and predicted as maintenance required (correct)
```

```
Confusion Matrix:
[[ 1468  1387]
 [  498 11647]]
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.51      0.61      2855
           1       0.89      0.96      0.93     12145

    accuracy                           0.87     15000
   macro avg       0.82      0.74      0.77     15000
weighted avg       0.87      0.87      0.86     15000
```

Figure 9: Model Evaluation Report

2. Decision Tree

```
# Initialize and fit the DecisionTreeClassifier
DTC = DecisionTreeClassifier()
DTC = DTC.fit(X_train, y_train)

# Predict the test set results
y_pred = DTC.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f' % (accuracy))

# Calculate precision
precision = precision_score(y_test, y_pred)
print('Precision: %.2f' % precision)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print('F1 Score: %.2f' % f1)

# Calculate recall
recall = recall_score(y_test, y_pred)
print('Recall: %.2f' % recall)
```

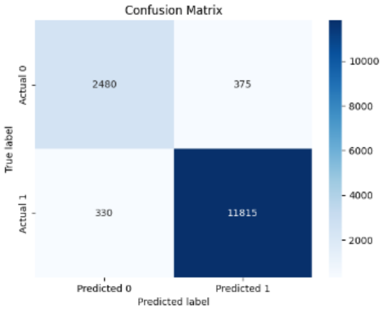
Figure 10: Model Training

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix using seaborn for better visualization
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# [[TN, FP],
#  [FN, TP]]
#TN - vechicle not required maintenance and predicted as no maintenance required (Correct)
#FP - vechicle not required maintenance but predicted as maintenance required
#FN - vechicle required maintenance but predicted as no maintenance required.
#TP - vehicle required maintenance and predicted as maintenance required (correct)
```

```
Confusion Matrix:
[[ 2480   375]
 [  330 11815]]
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.88      2855
           1       0.97      0.97      0.97     12145

    accuracy                           0.95     15000
   macro avg       0.93      0.92      0.92     15000
weighted avg       0.95      0.95      0.95     15000
```

Figure 11: Model Evaluation Report

3. Gradient Boosting Regressor

```
# Initialize GradientBoostingClassifier
gb_classifier = GradientBoostingClassifier()

# Train the model
gb_classifier.fit(X_train, y_train)

# Predictions on the test set
y_pred = gb_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f' % (accuracy))

# Calculate Predictions
precision = precision_score(y_test, y_pred)
print('Precision: %.2f' % precision)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print('F1 Score: %.2f' % f1)

# Predictions on the test set
y_pred = gb_classifier.predict(X_test)

# Calculate recall
recall = recall_score(y_test, y_pred)
print('Recall: %.2f' % recall)
```

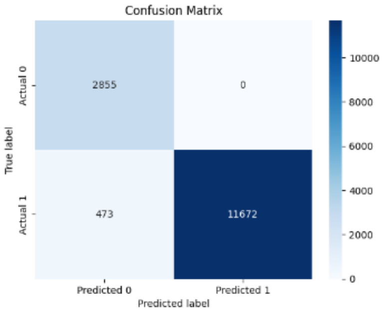
Figure 12: Model Training

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix using seaborn for better visualization
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# [[TN, FP],
#  [FN, TP]]
#TN - vechicle not required maintenance and predicted as no maintenance required (Correct)
#FP - vechicle not required maintenance but predicted as maintenance required
#FN - vechicle required maintenance but predicted as no maintenance required.
#TP - vehicle required maintenance and predicted as maintenance required (correct)
```

```
Confusion Matrix:
[[ 2855     0]
 [  473 11672]]
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2855
           1       1.00      0.96      0.98     12145

    accuracy                           0.97     15000
   macro avg       0.93      0.98      0.95     15000
weighted avg       0.97      0.97      0.97     15000
```

Figure 13: Model Evaluation Report

4. Random Forest

```
RFC = RandomForestClassifier()
RFC = RFC.fit(X_train, y_train)
y_pred = RFC.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f' % (accuracy))

precision = precision_score(y_test, y_pred)
print('Precision: %.2f' % precision)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print('F1 Score: %.2f' % f1)

# Calculate recall
recall = recall_score(y_test, y_pred)
print('Recall: %.2f' % recall)
```

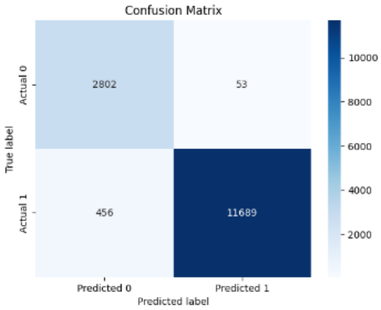
Figure 14: Model Training

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix using seaborn for better visualization
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# [[TN, FP],
#  [FN, TP]]
#TN - vechicle not required maintenance and predicted as no maintenance required (Correct)
#FP - vechicle not required maintenance but predicted as maintenance required
#FN - vechicle required maintenance but predicted as no maintenance required.
#TP - vehicle required maintenance and predicted as maintenance required (correct)
```

```
Confusion Matrix:
[[ 2802    53]
 [  456 11689]]
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.98      0.92      2855
           1       1.00      0.96      0.98     12145

    accuracy                           0.97     15000
   macro avg       0.93      0.97      0.95     15000
weighted avg       0.97      0.97      0.97     15000
```

Figure 15: Model Evaluation Report

5. Gated Recurrent Unit

```
#Gated Recurrent Unit (GRU)
# Assuming X and y are your features and labels
# Split the data into training and test sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the model
model = Sequential()
model.add(GRU(units=50, input_shape=(X_train.shape[1], 1), return_sequences=False))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for 10 epochs
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')

# Predict on the test data
y_pred_prob = model.predict(X_test)
y_pred = np.round(y_pred_prob)  # Since the output is probabilistic, round to get class predictions

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f' % accuracy)

# Calculate precision
precision = precision_score(y_test, y_pred)
print('Precision: %.2f' % precision)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print('F1 Score: %.2f' % f1)

# Calculate recall
recall = recall_score(y_test, y_pred)
print('Recall: %.2f' % recall)
```

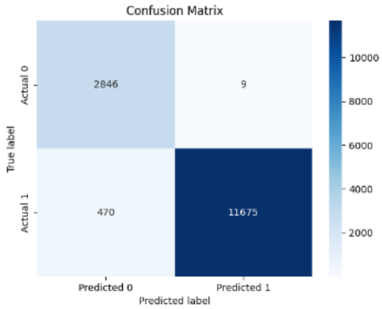
Figure 16: Model Training

```python
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix using seaborn for better visualization
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# [[TN, FP],
#  [FN, TP]]
#TN - vechicle not required maintenance and predicted as no maintenance required (Correct)
#FP - vechicle not required maintenance but predicted as maintenance required
#FN - vechicle required maintenance but predicted as no maintenance required.
#TP - vehicle required maintenance and predicted as maintenance required (correct)
```

```
Confusion Matrix:
[[ 2846     9]
 [  470 11675]]
```




```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2855
           1       1.00      0.96      0.98     12145

    accuracy                           0.97     15000
   macro avg       0.93      0.98      0.95     15000
weighted avg       0.97      0.97      0.97     15000
```

Figure 17: Model Evaluation Report

6. Long Short-Term Memory

```python
#Long Short-Term Memory (LSTM)
# Assuming X and y are your features and labels
# Split the data into training and test sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the model
model = Sequential()
model.add(LSTM(units=50, input_shape=(X_train.shape[1], 1), return_sequences=False))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')

# Predict on the test data
y_pred_prob = model.predict(X_test)
y_pred = np.round(y_pred_prob)  # Since the output is probabilistic, round to get class predictions

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f' % accuracy)

# Calculate precision
precision = precision_score(y_test, y_pred)
print('Precision: %.2f' % precision)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print('F1 Score: %.2f' % f1)

# Calculate recall
recall = recall_score(y_test, y_pred)
print('Recall: %.2f' % recall)
```

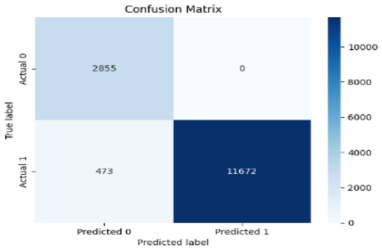
Figure 18: Model Training

```python
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix using seaborn for better visualization
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# [[TN, FP],
#  [FN, TP]]
#TN - vechicle not required maintenance and predicted as no maintenance required (Correct)
#FP - vechicle not required maintenance but predicted as maintenance required
#FN - vechicle required maintenance but predicted as no maintenance required.
#TP - vehicle required maintenance and predicted as maintenance required (correct)
```

```
Confusion Matrix:
[[ 2855     0]
 [  473 11672]]
```




```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2855
           1       1.00      0.96      0.98     12145

    accuracy                           0.97     15000
   macro avg       0.93      0.98      0.95     15000
weighted avg       0.97      0.97      0.97     15000
```

Figure 19: Model Evaluation Report

7. Minimal Gated Unit

```python
#Minimal Gated Unit (MGU)
# Define the Minimal Gated Unit (MGU) as a custom layer
class MGUCell(Layer):
    def __init__(self, units):
        super(MGUCell, self).__init__()
        self.units = units
        self.state_size = units  # Add the state_size attribute

    def build(self, input_shape):
        self.Wx = self.add_weight(shape=(input_shape[-1], self.units), initializer='glorot_uniform', name='Wx')
        self.Wh = self.add_weight(shape=(self.units, self.units), initializer='glorot_uniform', name='Wh')
        self.Wr = self.add_weight(shape=(input_shape[-1], self.units), initializer='glorot_uniform', name='Wr')
        self.Ur = self.add_weight(shape=(self.units, self.units), initializer='glorot_uniform', name='Ur')
        self.bh = self.add_weight(shape=(self.units,), initializer='zeros', name='bh')
        self.br = self.add_weight(shape=(self.units,), initializer='zeros', name='br')

    def call(self, inputs, states):
        h_prev = states[0]
        r = tf.sigmoid(tf.matmul(inputs, self.Wr) + tf.matmul(h_prev, self.Ur) + self.br)
        h_candidate = tf.tanh(tf.matmul(inputs, self.Wx) + r * (tf.matmul(h_prev, self.Wh) + self.bh))
        h = (1 - r) * h_prev + r * h_candidate
        return h, [h]

# Define a wrapper layer to create an RNN with MGU
class MGU(Layer):
    def __init__(self, units):
        super(MGU, self).__init__()
        self.units = units
        self.rnn = RNN(MGUCell(units))  # RNN layer wrapping the custom MGUCell

    def call(self, inputs):
        return self.rnn(inputs)

# Assuming X and y are your features and labels
# X = np.array(...)  # Define or load your feature dataset here
# y = np.array(...)  # Define or load your labels here

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Reshape the data to fit the MGU input requirements
X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)

# Define the MGU-based model
model = Sequential()
model.add(MGU(units=50))  # Use the custom MGU layer
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for 10 epochs
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')

# Predict on the test data
y_pred_prob = model.predict(X_test)
y_pred = np.round(y_pred_prob)  # Since the output is probabilistic, round to get class predictions

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f' % accuracy)

# Calculate precision
precision = precision_score(y_test, y_pred)
print('Precision: %.2f' % precision)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print('F1 Score: %.2f' % f1)

# Calculate recall
recall = recall_score(y_test, y_pred)
print('Recall: %.2f' % recall)
```

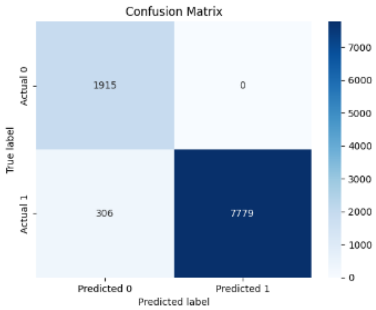
Figure 20: Model Training

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix using seaborn for better visualization
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# [[TN, FP],
#  [FN, TP]]
#TN - vechicle not required maintenance and predicted as no maintenance required. (Correct)
#FP - vechicle not required maintenance but predicted as maintenance required.
#FN - vechicle required maintenance but predicted as no maintenance required.
#TP - vehicle required maintenance and predicted as maintenance required. (correct)
```

```
Confusion Matrix:
[[1915    0]
 [ 306 7779]]
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.93      1915
           1       1.00      0.96      0.98      8085

    accuracy                           0.97     10000
   macro avg       0.93      0.98      0.95     10000
weighted avg       0.97      0.97      0.97     10000
```

Figure 21: Model Evaluation Report