Model Development Phase Template

Date	20 SEP 2024
Team ID	738336
Project Title	Electric Motor Temperature Prediction using Machine Learning
Maximum Marks	4 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model Training Code:

Paste the screenshot of the model training code

Model Validation and Evaluation Report:

Model	Classification Report Better	Accuracy	Confusion Matrix Better

Model 1		100 %	

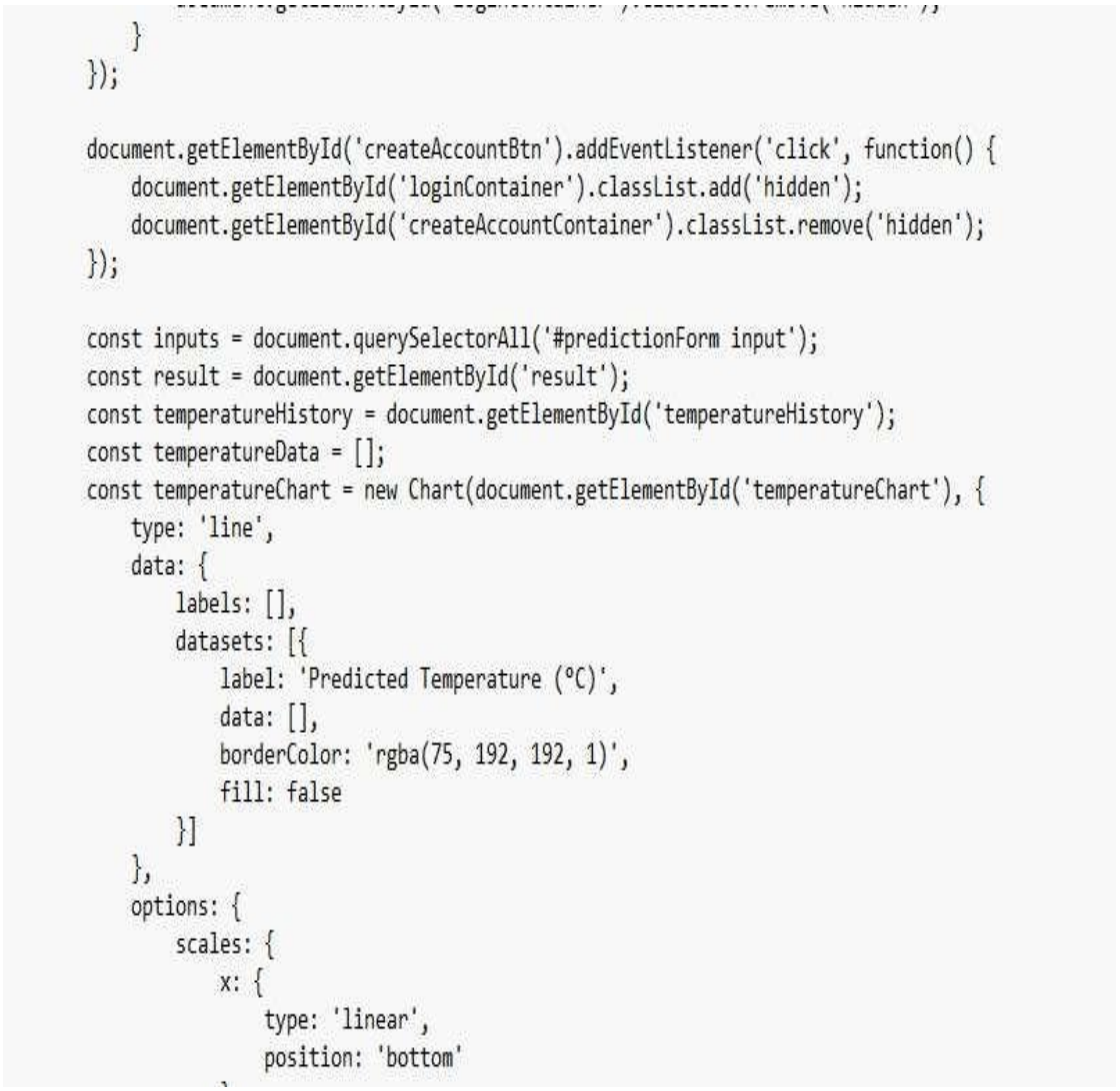
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PMSM Temperature Prediction</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background: url('https://www.itwm.fraunhofer.de/en/departments/sys/machine-monitoring-and-control/predictive-maintenance-
machinelearning/jcr:content/stage/stageParsys/stage_slide/image.img.jpg/1689858387339/1440x448-Predictive-Maintenance.jpg') no-repeat center center fixed;
            background-size: cover;
            transition: background-color 0.5s;
        }
        body.dark-mode {
            background-color: #2c2c2c;
            color: white;
        }
        .container {
            max-width: 400px;
            padding: 20px;
            background: rgba(255, 255, 255, 0.9);
            border-radius: 10px;
            box-shadow: 0 4px 20px rgba(0, 0, 0, 0.2);
            transition: background 0.5s;
        }
```

```
body.dark-mode .container {
    background: rgba(50, 50, 50, 0.9);
}
h1 {
    text-align: center;
    color: #333;
}
label {
    margin-bottom: 5px;
}
input {
    width: 100%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ddd;
    border-radius: 5px;
}
button {
    width: 100%;
    padding: 10px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
button:hover {
    background-color: #4cae4c;
```

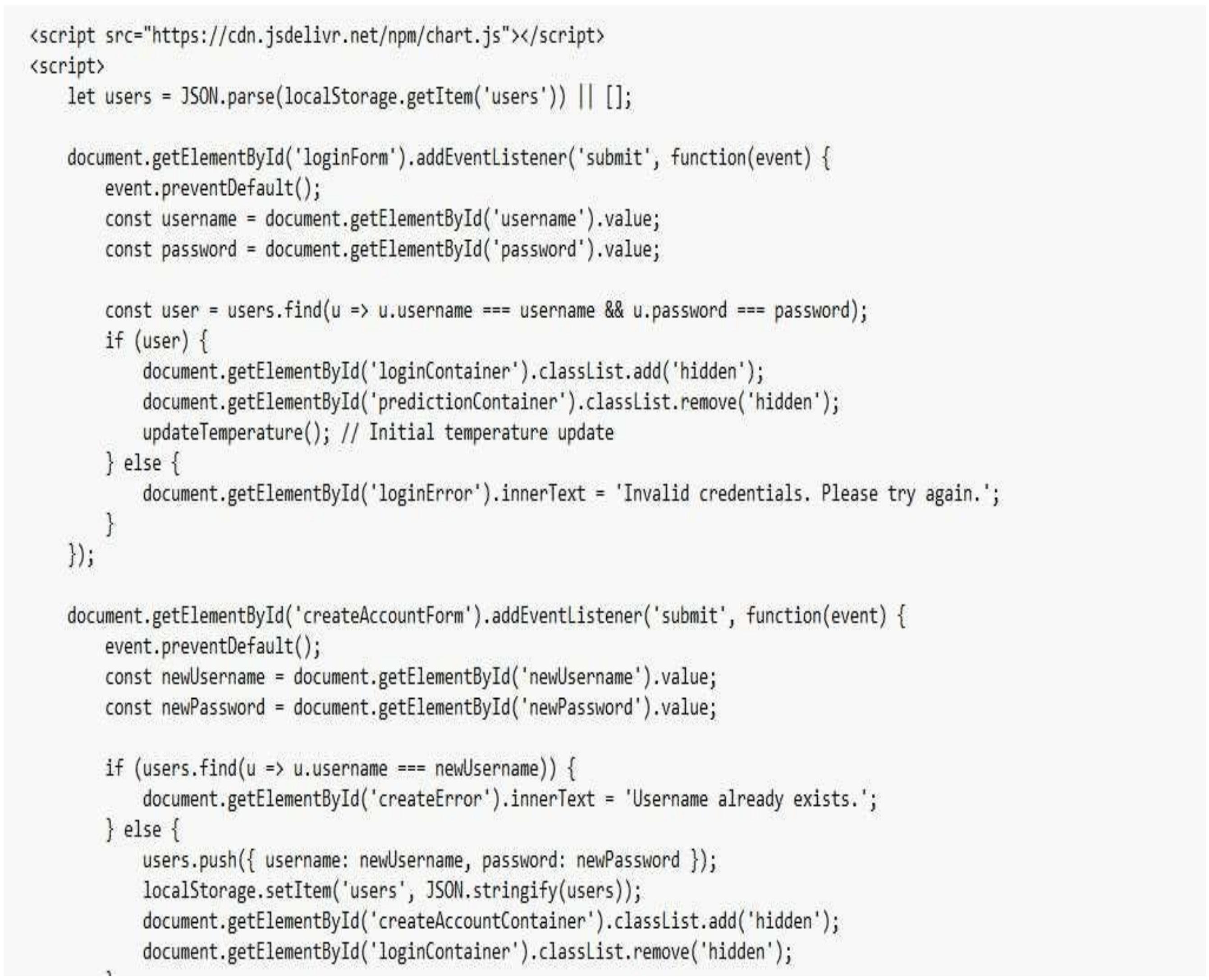
```
        background-color: #4cae4c;
    }
    .result {
        margin-top: 20px;
        font-size: 1.2em;
        text-align: center;
        color: #333;
    }
    .hidden {
        display: none;
    }
    .error {
        color: red;
        text-align: center;
        margin-top: 10px;
    }
    #temperatureHistory {
        margin-top: 20px;
        max-height: 200px;
        overflow-y: auto;
        border: 1px solid #ddd;
        border-radius: 5px;
        padding: 10px;
    }
    canvas {
        max-width: 100%;
        margin-top: 20px;
```

```
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="container" id="loginContainer">
        <h1>Login</h1>
        <form id="loginForm">
            <label for="username">Username:</label>
            <input type="text" id="username" required>
            <label for="password">Password:</label>
            <input type="password" id="password" required>
            <button type="submit">Login</button>
        </form>
        <div class="error" id="loginError"></div>
        <button id="createAccountBtn">Create Account</button>
    </div>

    <div class="container hidden" id="createAccountContainer">
        <h1>Create Account</h1>
        <form id="createAccountForm">
            <label for="newUsername">Username:</label>
            <input type="text" id="newUsername" required>
            <label for="newPassword">Password:</label>
            <input type="password" id="newPassword" required>
            <button type="submit">Create Account</button>
        </form>
        <div class="error" id="createError"></div>
    </div>
```

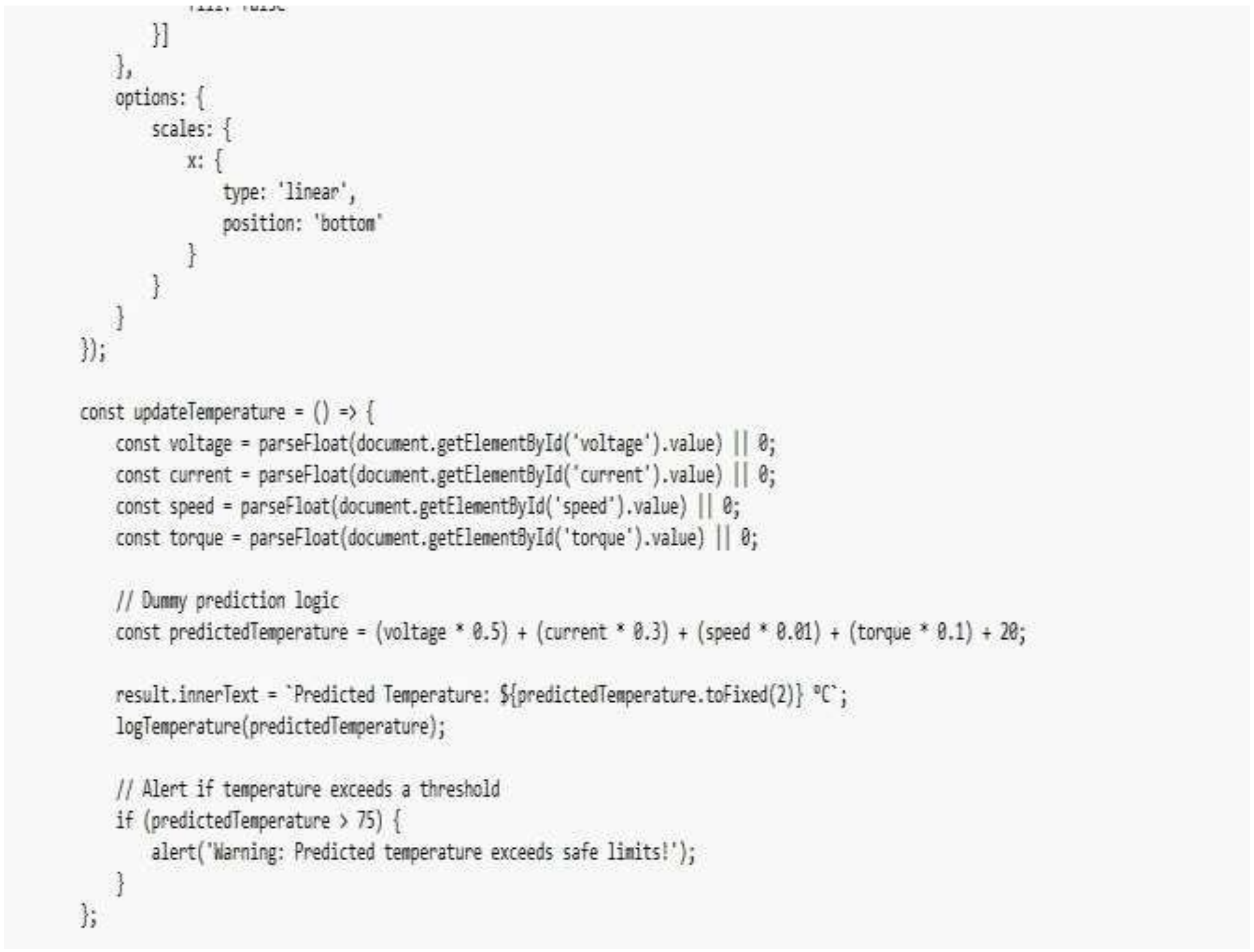
```
    <div class="container hidden" id="predictionContainer">
        <h1>PMSM Rotor Temperature Prediction</h1>
        <form id="predictionForm">
            <label for="voltage">Voltage (V):</label>
            <input type="number" id="voltage" required min="0">
            <label for="current">Current (A):</label>
            <input type="number" id="current" required min="0">
            <label for="speed">Speed (RPM):</label>
            <input type="number" id="speed" required min="0">
            <label for="torque">Torque (Nm):</label>
            <input type="number" id="torque" required min="0">
            <button type="button" id="logoutBtn">Logout</button>
            <button type="button" id="exportDataBtn">Export Data</button>
            <button type="button" id="toggleDarkModeBtn">Toggle Dark Mode</button>
        </form>
        <div id="result" class="result">Predicted Temperature: 0.00 °C</div>
        <div id="temperatureHistory"></div>
        <canvas id="temperatureChart" width="400" height="200"></canvas>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script>
        let users = JSON.parse(localStorage.getItem('users')) || [];

        document.getElementById('loginForm').addEventListener('submit', function(event) {
            event.preventDefault();
            const username = document.getElementById('username').value;
            const password = document.getElementById('password').value;
```



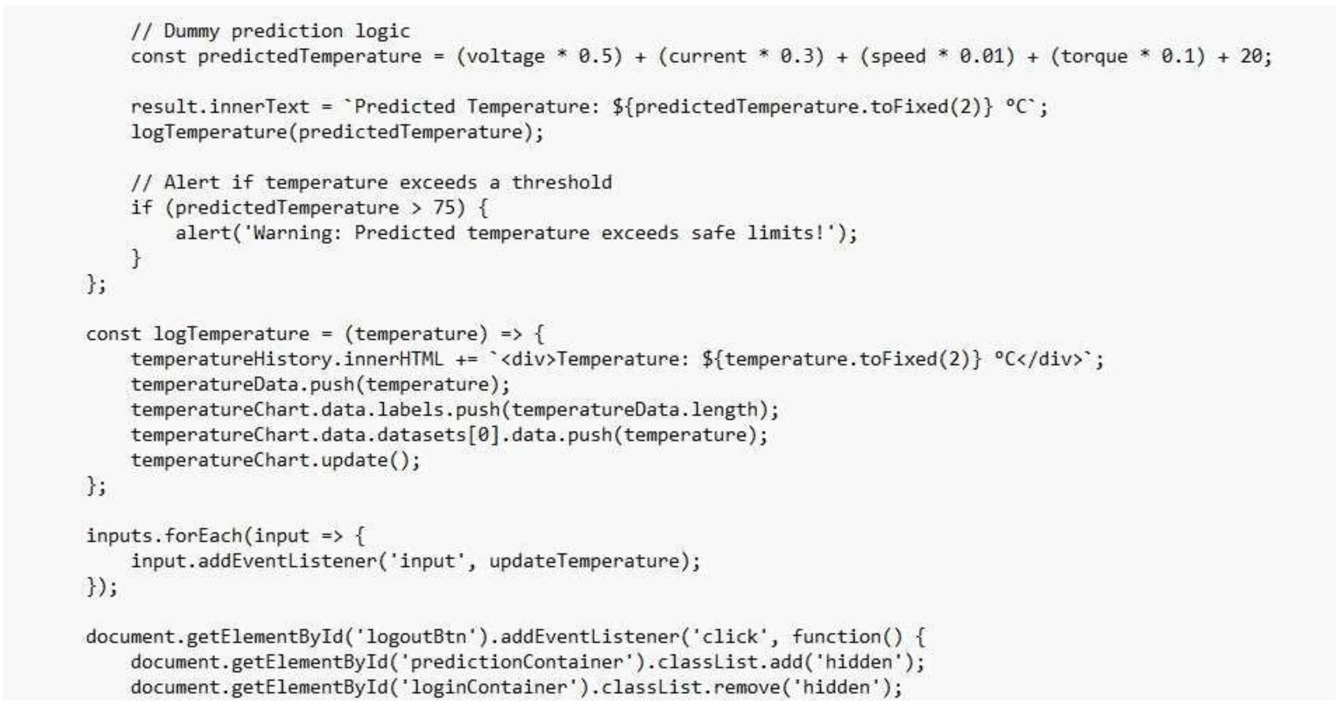
```
    }
});

document.getElementById('createAccountBtn').addEventListener('click', function() {
    document.getElementById('loginContainer').classList.add('hidden');
    document.getElementById('createAccountContainer').classList.remove('hidden');
});

const inputs = document.querySelectorAll('#predictionForm input');
const result = document.getElementById('result');
const temperatureHistory = document.getElementById('temperatureHistory');
const temperatureData = [];
const temperatureChart = new Chart(document.getElementById('temperatureChart'), {
    type: 'line',
    data: {
        labels: [],
        datasets: [{
            label: 'Predicted Temperature (°C)',
            data: [],
            borderColor: 'rgba(75, 192, 192, 1)',
            fill: false
        }]
    },
    options: {
        scales: {
            x: {
                type: 'linear',
                position: 'bottom'
```



```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
    let users = JSON.parse(localStorage.getItem('users')) || [];

    document.getElementById('loginForm').addEventListener('submit', function(event) {
        event.preventDefault();
        const username = document.getElementById('username').value;
        const password = document.getElementById('password').value;

        const user = users.find(u => u.username === username && u.password === password);
        if (user) {
            document.getElementById('loginContainer').classList.add('hidden');
            document.getElementById('predictionContainer').classList.remove('hidden');
            updateTemperature(); // Initial temperature update
        } else {
            document.getElementById('loginError').innerText = 'Invalid credentials. Please try again.';
        }
    });

    document.getElementById('createAccountForm').addEventListener('submit', function(event) {
        event.preventDefault();
        const newUsername = document.getElementById('newUsername').value;
        const newPassword = document.getElementById('newPassword').value;

        if (users.find(u => u.username === newUsername)) {
            document.getElementById('createError').innerText = 'Username already exists.';
        } else {
            users.push({ username: newUsername, password: newPassword });
            localStorage.setItem('users', JSON.stringify(users));
            document.getElementById('createAccountContainer').classList.add('hidden');
            document.getElementById('loginContainer').classList.remove('hidden');
```



```
                fill: false
            }]
        },
        options: {
            scales: {
                x: {
                    type: 'linear',
                    position: 'bottom'
                }
            }
        }
    });

    const updateTemperature = () => {
        const voltage = parseFloat(document.getElementById('voltage').value) || 0;
        const current = parseFloat(document.getElementById('current').value) || 0;
        const speed = parseFloat(document.getElementById('speed').value) || 0;
        const torque = parseFloat(document.getElementById('torque').value) || 0;

        // Dummy prediction logic
        const predictedTemperature = (voltage * 0.5) + (current * 0.3) + (speed * 0.01) + (torque * 0.1) + 20;

        result.innerText = `Predicted Temperature: ${predictedTemperature.toFixed(2)} °C`;
        logTemperature(predictedTemperature);

        // Alert if temperature exceeds a threshold
        if (predictedTemperature > 75) {
            alert('Warning: Predicted temperature exceeds safe limits!');
        }
    };
```



```
        // Dummy prediction logic
        const predictedTemperature = (voltage * 0.5) + (current * 0.3) + (speed * 0.01) + (torque * 0.1) + 20;

        result.innerText = `Predicted Temperature: ${predictedTemperature.toFixed(2)} °C`;
        logTemperature(predictedTemperature);

        // Alert if temperature exceeds a threshold
        if (predictedTemperature > 75) {
            alert('Warning: Predicted temperature exceeds safe limits!');
        }
    };
```

```
    const logTemperature = (temperature) => {
        temperatureHistory.innerHTML += `<div>Temperature: ${temperature.toFixed(2)} °C</div>`;
        temperatureData.push(temperature);
        temperatureChart.data.labels.push(temperatureData.length);
        temperatureChart.data.datasets[0].data.push(temperature);
        temperatureChart.update();
    };

    inputs.forEach(input => {
        input.addEventListener('input', updateTemperature);
    });

    document.getElementById('logoutBtn').addEventListener('click', function() {
        document.getElementById('predictionContainer').classList.add('hidden');
        document.getElementById('loginContainer').classList.remove('hidden');
        document.getElementById('toggleDarkModeBtn').addEventListener('click', function() {
            document.body.classList.toggle('dark-mode');
        });
    </script>
</body>
</html>
```

Model Building

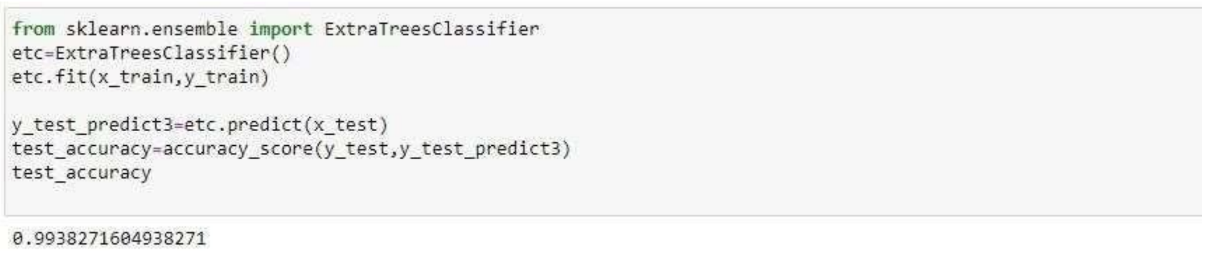
1. Building a model for predicting electric motor temperature involves several steps, from data preprocessing to training and evaluating the model. Below is a structured approach to building your predictive model using machine learning.

Random Forest Classifier

2. While the original task involves predicting temperature (a regression problem), if you are looking to categorize temperature into discrete classes (e.g., normal, high, low), you can use a **Random Forest Classifier**. Here’s a structured approach to build and evaluate a Random Forest Classifier.

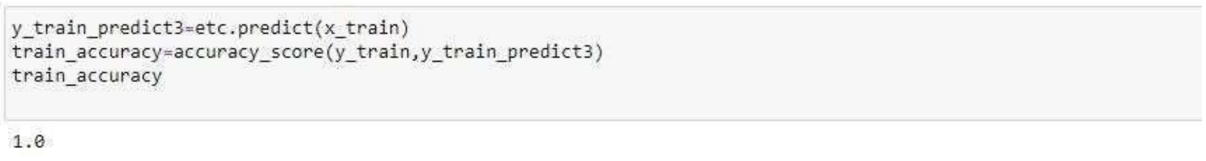
Decision Tree Classifier

ExtraTrees Classifier



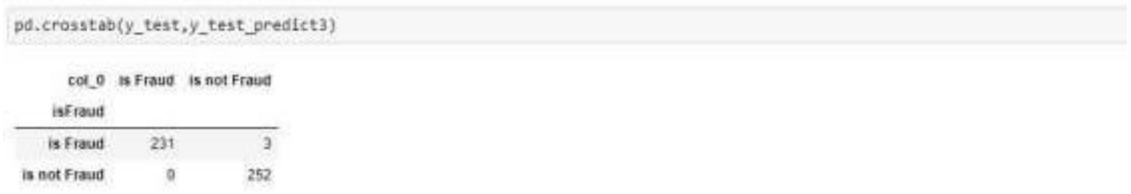
```
from sklearn.ensemble import ExtraTreesClassifier
etc=ExtraTreesClassifier()
etc.fit(x_train,y_train)

y_test_predict3=etc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict3)
test_accuracy
```

```
0.9938271604938271
```



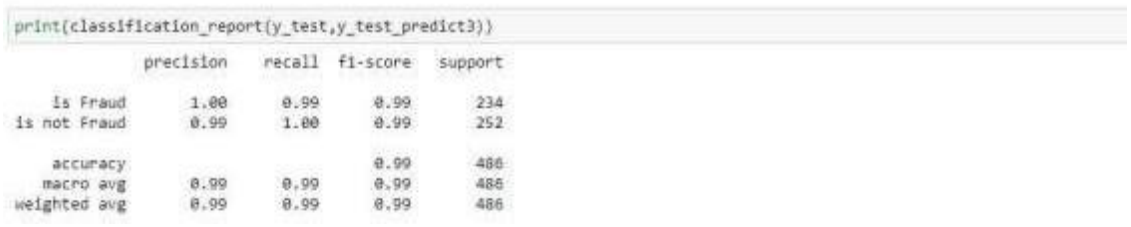
```
y_train_predict3=etc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict3)
train_accuracy
```

```
1.0
```



```
pd.crosstab(y_test,y_test_predict3)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	231	3
is not Fraud	0	252

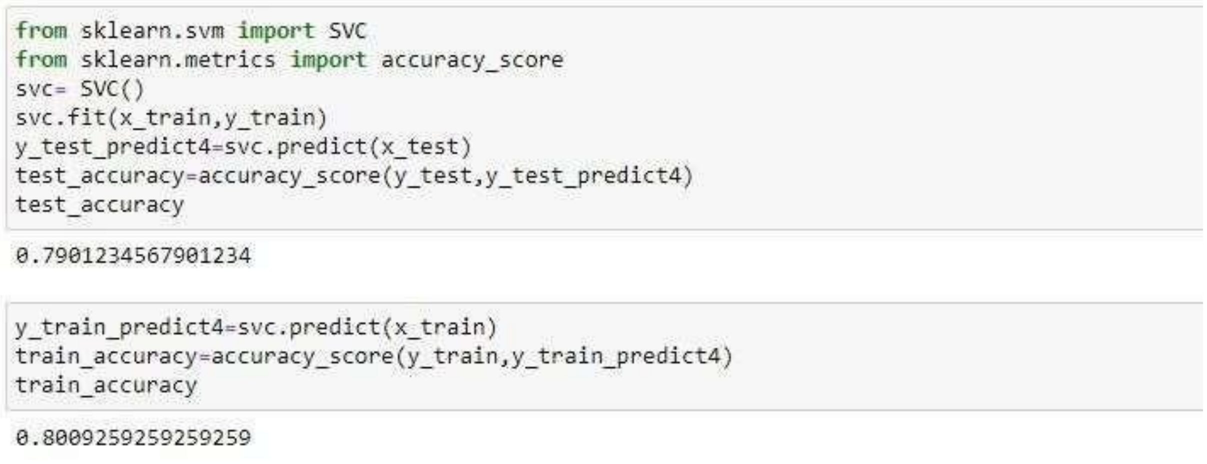


```
print(classification_report(y_test,y_test_predict3))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.99      0.99       234
is not Fraud       0.99      1.00      0.99       252

    accuracy                           0.99       486
   macro avg       0.99      0.99      0.99       486
weighted avg       0.99      0.99      0.99       486
```

Support Vector Machine Classifier

A function named SupportVector is created and train and test data are passed as the parameters. Inside the function, the SupportVectorClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, confusion matrix and classification report is done

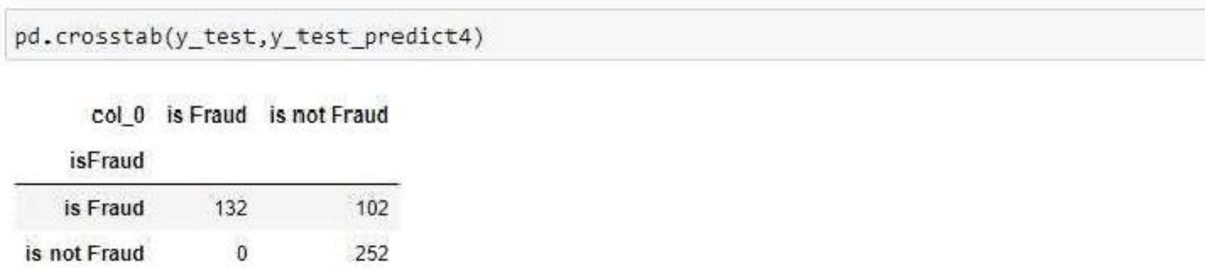


```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc= SVC()
svc.fit(x_train,y_train)
y_test_predict4=svc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict4)
test_accuracy
```

```
0.7901234567901234
```

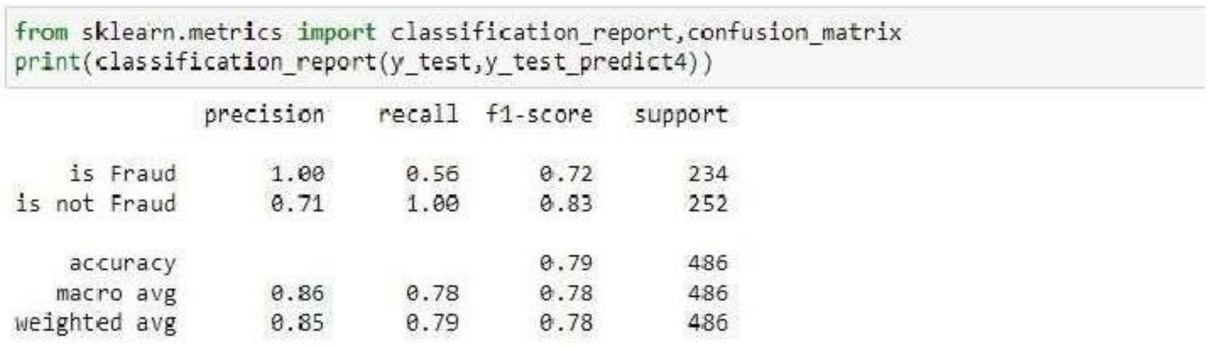
```
y_train_predict4=svc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict4)
train_accuracy
```

```
0.8009259259259259
```



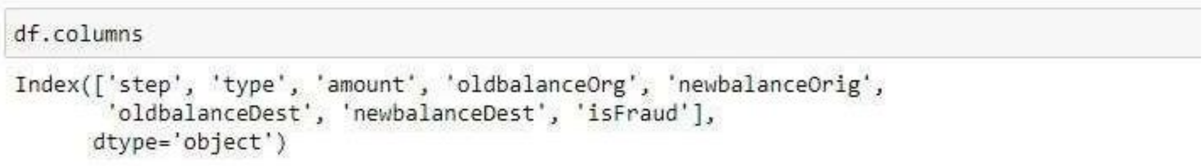
```
pd.crosstab(y_test,y_test_predict4)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	132	102
is not Fraud	0	252



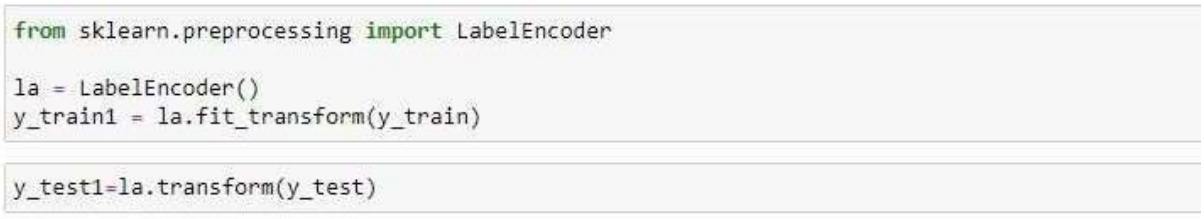
```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,y_test_predict4))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.56      0.72       234
is not Fraud       0.71      1.00      0.83       252

    accuracy                           0.79       486
   macro avg       0.86      0.78      0.78       486
weighted avg       0.85      0.79      0.78       486
```



```
df.columns
```

```
Index(['step', 'type', 'amount', 'oldbalanceOrg', 'newbalanceOrig',
       'oldbalanceDest', 'newbalanceDest', 'isFraud'],
      dtype='object')
```



```
from sklearn.preprocessing import LabelEncoder

la = LabelEncoder()
y_train1 = la.fit_transform(y_train)
```

```
y_test1=la.transform(y_test)
```

preprocessing class of sklearn. LabelEncoder[source] 0 to n classes-1 as the range for the target labels to be encoded. Instead of encoding the input X, the target values, i.e. y, should be encoded using this transformer.