Electric Motor Temperature Prediction using Machine Learning

Project Description

Electric motors are widely used in electric vehicles, industrial automation, robotics, and manufacturing systems. One of the critical factors affecting motor performance and lifespan is temperature, especially the internal Permanent Magnet (PM) temperature. Excessive heating can lead to reduced efficiency, insulation damage, and unexpected motor failure.

However, directly measuring internal motor temperature using physical sensors is costly, complex, and not always feasible in real-time industrial environments. Therefore, there is a need for a predictive solution that can estimate motor temperature using available operational data.

Monitoring and predicting motor temperature is essential for:

- Preventive maintenance
- Improving energy efficiency
- Avoiding unexpected breakdowns
- Enhancing equipment reliability

In this project, we build a Machine Learning Regression Model that predicts the Permanent Magnet temperature using electrical and environmental operational parameters.

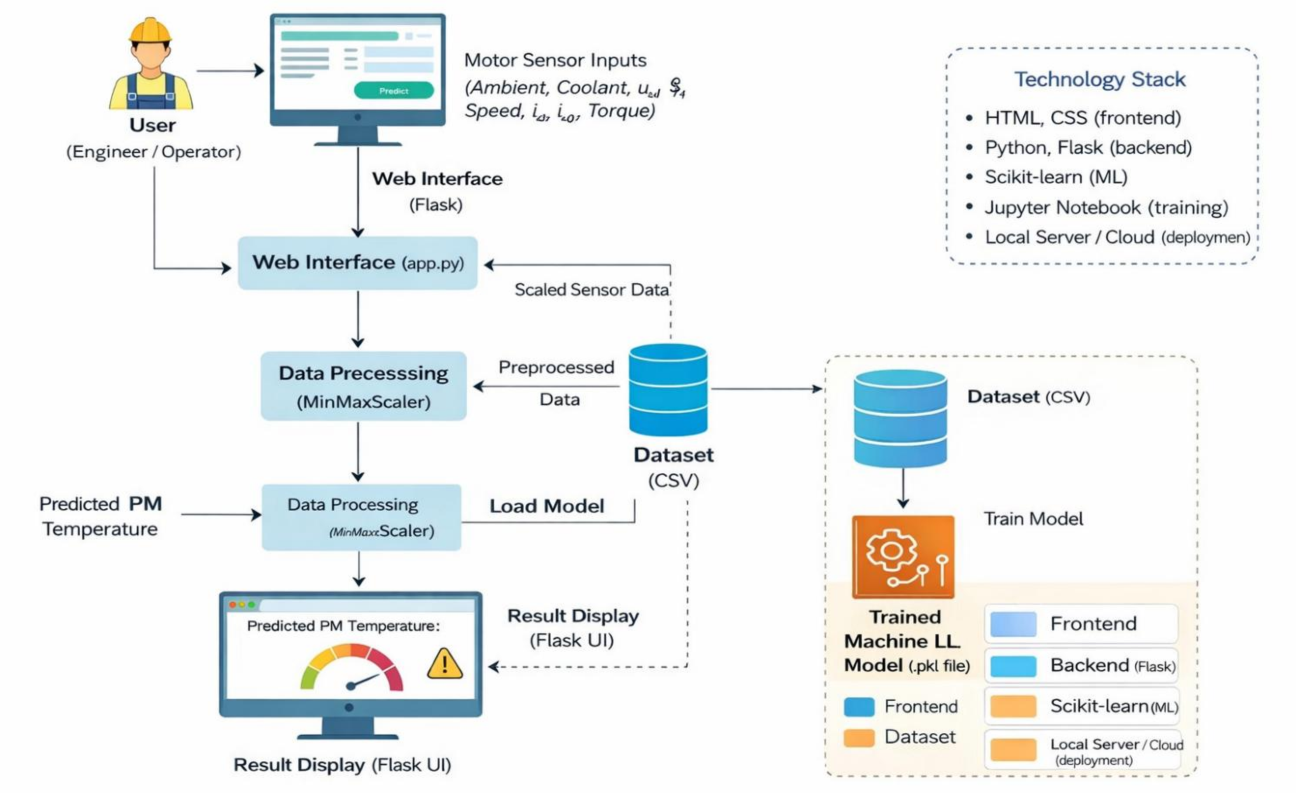
We train and test the dataset using regression algorithms. The best performing model is selected and saved in Save.pkl format. The model is then integrated into a Flask web application for real-time prediction

The objective of this project is to develop a Machine Learning-based system that predicts the permanent magnet temperature of an electric motor using real-time sensor parameters such as:

- Ambient temperature
- Coolant temperature
- Direct-axis voltage (u_d)
- Quadrature-axis voltage (u_q)
- Motor speed
- Direct-axis current (i_d)
- Quadrature-axis current (i_q)
- Torque

The system aims to provide accurate, fast, and cost-effective temperature prediction through a web-based interface.

Technical Architecture



Pre-Requisites

- **Software Requirements**
  - **Python 3.x** (Recommended: Python 3.8 or above)
  - **Anaconda Navigator** (for managing environments and Jupyter Notebook)
  - **Jupyter Notebook** (for model building and experimentation)
  - **VS Code** (for Flask application development)
  - **Web Browser** (Chrome / Edge / Firefox)
- **Required Python Packages**

  The following libraries must be installed before running the project:

  Open Anaconda Prompt or Command Prompt and execute:

  pip install numpy
  pip install pandas
  pip install scikit-learn
  pip install matplotlib
  pip install seaborn
  pip install flask
- **Package Usage in Project**

- **NumPy** → Numerical operations
- **Pandas** → Data loading and preprocessing
- **Scikit-learn** → Model training and evaluation
- **Matplotlib & Seaborn** → Data visualization
- **Flask** → Web application framework
- **Pickle** → Model serialization (built-in Python module, no need to install separately)

Note: pickle-mixin is NOT required because Python already includes pickle by default.

- **Hardware Requirements**
  - Minimum 4 GB RAM
  - Intel i3 Processor or above
  - 1 GB free storage space

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- ML Concepts

  Supervised learning: https://www.javatpoint.com/supervised-machine-learning

- Regression and classification

  Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

  Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm

  KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

  Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

  Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

To develop a Machine Learning-based system that accurately predicts the permanent magnet (PM) temperature of an electric motor using operational sensor data, thereby reducing the need for costly internal temperature sensors.

Specific Objectives

1. **To analyze electric motor sensor data**
   Understand the relationship between motor parameters such as voltage, current, torque, speed, and temperature.
2. **To preprocess and prepare data effectively**
   Handle missing values, normalize features using MinMaxScaler, and split data into training and testing sets.
3. **To build and train a regression model**
   Develop a machine learning model capable of predicting permanent magnet temperature with high accuracy.
4. **To evaluate model performance**
   Measure prediction accuracy using metrics such as R^2 Score, Mean Absolute Error (MAE), and Mean Squared Error (MSE).
5. **To deploy the trained model using Flask**
   Create a web-based interface where users can input motor parameters and obtain temperature predictions in real-time.
6. **To enable predictive maintenance**
   Provide early detection of overheating conditions to prevent motor failure and improve operational efficiency.
7. **To design a scalable and modular system**
   Ensure the architecture supports future integration with IoT sensors and cloud deployment.

Project Flow

The project follows a structured flow starting from data collection and model development to deployment and real-time prediction using a web application.

Phase 1: Data Collection

- Collect electric motor dataset containing operational parameters.
- Identify input features and target variable.
- Understand relationships between sensor readings and temperature.

Phase 2: Data Preprocessing

- Load dataset using Pandas.
- Handle missing or inconsistent values.
- Perform feature scaling using **MinMaxScaler**.
- Split dataset into training and testing sets.

Phase 3: Model Building & Training

- Select regression algorithm.

- Train model using training dataset.
- Evaluate model performance using:
    - R^2 Score
    - Mean Absolute Error (MAE)
    - Mean Squared Error (MSE)

Phase 4: Model Saving

- Save trained model using pickle.
- Store as model.pkl file.
- Prepare model for deployment.

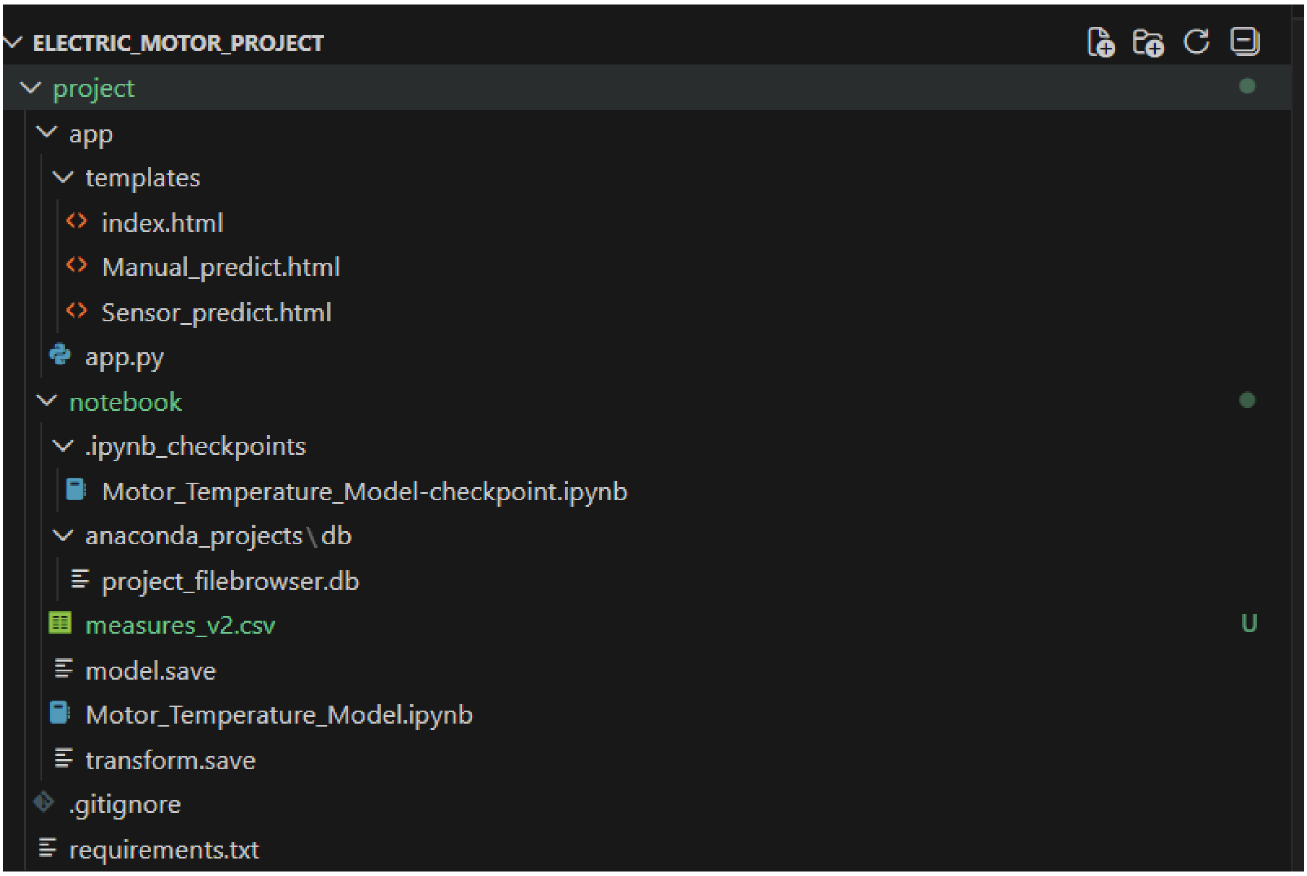
Phase 5: Web Application Development

- Develop Flask backend (app.py).
- Design HTML form for user input.
- Create routes for prediction.
- Integrate ML model with Flask application.

Phase 6: Real-Time Prediction

1. User enters motor parameters:
    - Ambient temperature
    - Coolant temperature
    - Voltage values (u_d, u_q)
    - Current values (i_d, i_q)
    - Motor speed
    - Torque
2. Input data is:
    - Validated
    - Scaled using saved scaler
3. Preprocessed data is passed to trained ML model.
4. Model predicts permanent magnet temperature.
5. Result is displayed on web interface.
6. If temperature exceeds safe threshold:
    - Warning message is shown.

Project Structure



We are building a Flask application which needs HTML pages stored in the templates folder and a Python script app.py for scripting.

model.save → trained Random Forest model

transform.save → MinMaxScaler object

notebook folder → model training

templates folder → web interface

Milestone 1: Data Collection and Understanding

Machine learning models depend heavily on data quality. In this project, we use an electric motor dataset containing operational parameters such as voltage, current, speed, coolant temperature, and torque. The target variable is the permanent magnet temperature (pm), which we aim to predict.

Activity 1: Importing Libraries and Loading Dataset

We first imported required libraries such as NumPy, Pandas, Matplotlib, Seaborn, and Scikit-learn. The dataset was then loaded using the pandas read_csv() function to begin analysis.

```
[1]: import numpy as np
     import pandas as pd
     from scipy import stats
     import matplotlib.pyplot as plt
     import seaborn as sns
     import warnings
     warnings.filterwarnings('ignore')
```

```
[2]: df = pd.read_csv('measures_v2.csv')
     df.head()
```

[2]:

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_yoke	ambient	torque	profile_id
0	-0.450682	18.805172	19.086670	-0.350055	18.293219	0.002866	0.004419	0.000328	24.554214	18.316547	19.850691	0.187101	17
1	-0.325737	18.818571	19.092390	-0.305803	18.294807	0.000257	0.000606	-0.000785	24.538078	18.314955	19.850672	0.245417	17
2	-0.440864	18.828770	19.089380	-0.372503	18.294094	0.002355	0.001290	0.000386	24.544693	18.326307	19.850657	0.176615	17
3	-0.327026	18.835567	19.083031	-0.316199	18.292542	0.006105	0.000026	0.002046	24.554018	18.330833	19.850647	0.238303	17
4	-0.471150	18.857033	19.082525	-0.332272	18.291428	0.003133	-0.064317	0.037184	24.565397	18.326662	19.850639	0.208197	17

Activity 2: Understanding Dataset Structure

After loading the dataset, we examined the column names, checked data types, and verified whether any missing values were present. The info() function confirmed that all features are numerical and there were no null values. The describe() function was used to analyze statistical properties such as mean, minimum, maximum, and standard deviation.

```
[10]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1330816 entries, 0 to 1330815
Data columns (total 13 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   u_q             1330816 non-null  float64
 1   coolant         1330816 non-null  float64
 2   stator_winding  1330816 non-null  float64
 3   u_d             1330816 non-null  float64
 4   stator_tooth    1330816 non-null  float64
 5   motor_speed     1330816 non-null  float64
 6   i_d             1330816 non-null  float64
 7   i_q             1330816 non-null  float64
 8   pm              1330816 non-null  float64
 9   stator_yoke     1330816 non-null  float64
 10  ambient         1330816 non-null  float64
 11  torque          1330816 non-null  float64
 12  profile_id      1330816 non-null  int64
dtypes: float64(12), int64(1)
memory usage: 132.0 MB
```

```
[11]: df.describe()
```

[11]:

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_yoke	am
count	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816
mean	5.427900e+01	3.622999e+01	6.634275e+01	-2.513381e+01	5.687858e+01	2.202081e+03	-6.871681e+01	3.741278e+01	5.850678e+01	4.818796e+01	2.456526
std	4.417323e+01	2.178615e+01	2.867206e+01	6.309197e+01	2.295223e+01	1.859663e+03	6.493323e+01	9.218188e+01	1.900150e+01	1.999100e+01	1.929522
min	-2.529093e+01	1.062375e+01	1.858582e+01	-1.315304e+02	1.813398e+01	-2.755491e+02	-2.780036e+02	-2.934268e+02	2.085696e+01	1.807669e+01	8.783478
25%	1.206992e+01	1.869814e+01	4.278796e+01	-7.869090e+01	3.841601e+01	3.171107e+02	-1.154061e+02	1.095863e+00	4.315158e+01	3.199033e+01	2.318480
50%	4.893818e+01	2.690014e+01	6.511013e+01	-7.429755e+00	5.603635e+01	1.999977e+03	-5.109376e+01	1.577401e+01	6.026629e+01	4.562551e+01	2.479733

Milestone 2: Data Visualization

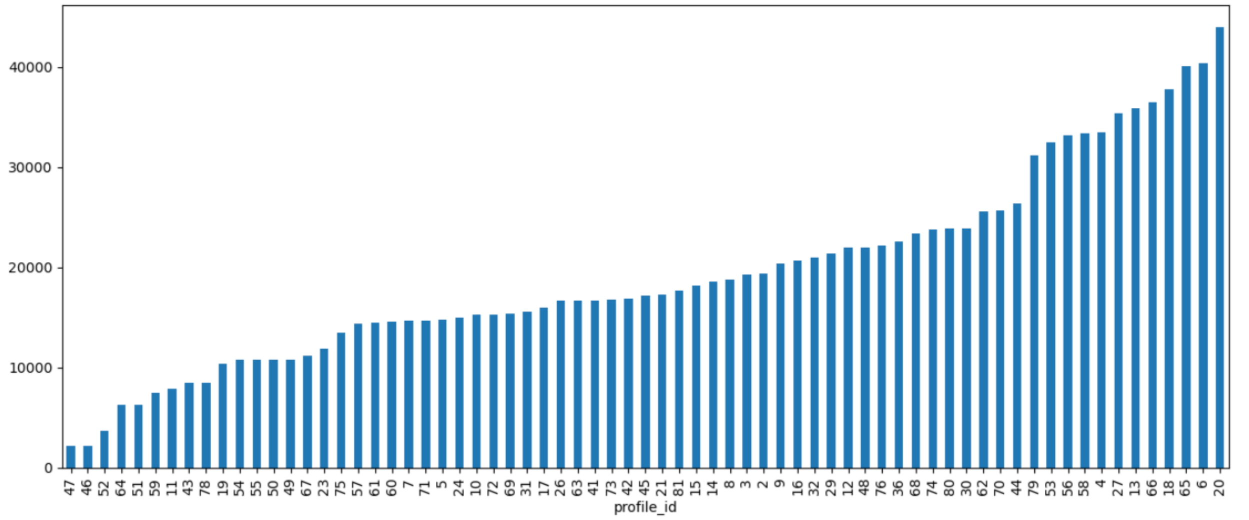
Data visualization helps in understanding feature distribution and relationships between variables.

Activity 1: Distribution and Outlier Analysis

Univariate analysis was performed using distribution plots and box plots. This helped in identifying skewness and detecting potential outliers in the dataset.

```
[3]: plt.figure(figsize=(15,6))
     df['profile_id'].value_counts().sort_values().plot(kind='bar')
```

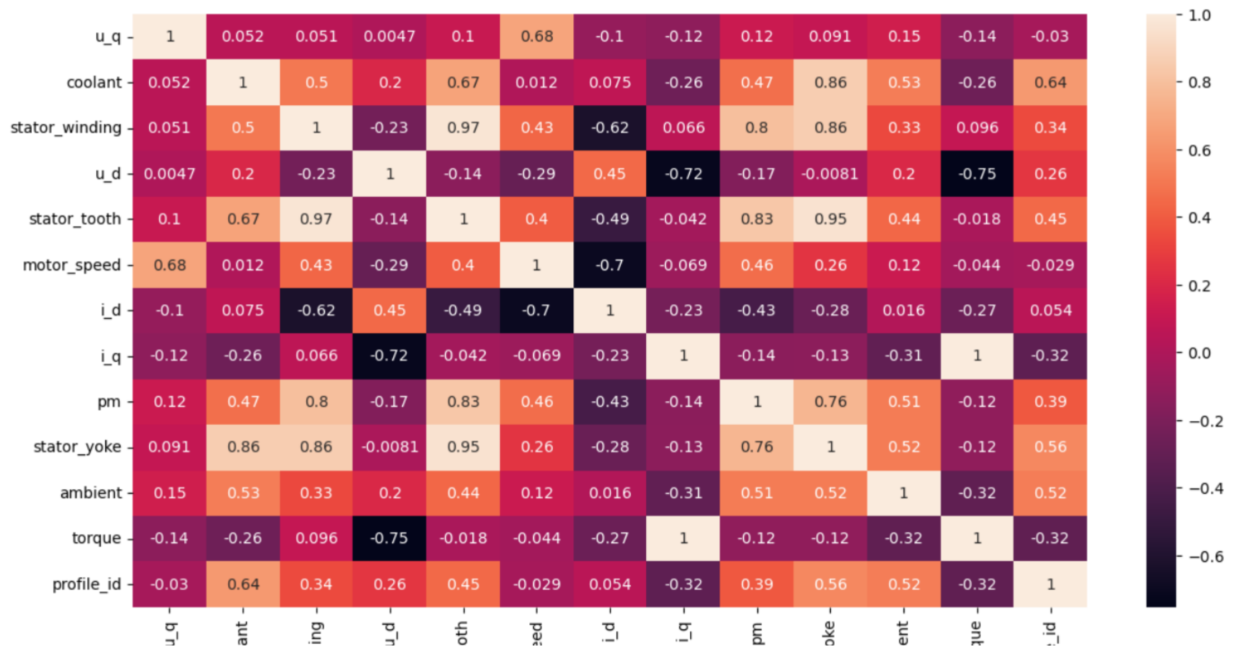
```
[3]: <Axes: xlabel='profile_id'>
```



Activity 2: Correlation Analysis

A heatmap was generated to study the correlation between features. It showed that stator temperature variables and current values have strong relationships with the permanent magnet temperature.

```
[8]: plt.figure(figsize=(14,7))
     sns.heatmap(df.corr(),annot=True);
```



Activity 3: Relationship Analysis

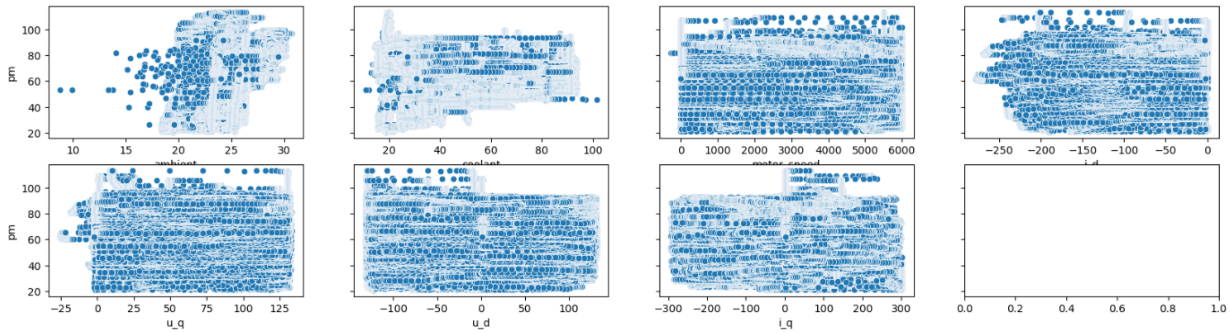
Scatter plots were created between pm and key features such as coolant temperature, motor speed, and current values. These plots visually demonstrated how input features influence the target variable.

```
[7]: fig, axes = plt.subplots(2, 4, figsize=(20, 5), sharey=True)

     sns.scatterplot(x=df['ambient'], y=df['pm'], ax=axes[0][0])
     sns.scatterplot(x=df['coolant'], y=df['pm'], ax=axes[0][1])
     sns.scatterplot(x=df['motor_speed'], y=df['pm'], ax=axes[0][2])
     sns.scatterplot(x=df['i_d'], y=df['pm'], ax=axes[0][3])

     sns.scatterplot(x=df['u_q'], y=df['pm'], ax=axes[1][0])
     sns.scatterplot(x=df['u_d'], y=df['pm'], ax=axes[1][1])
     sns.scatterplot(x=df['i_q'], y=df['pm'], ax=axes[1][2])
```

```
[7]: <Axes: xlabel='i_q', ylabel='pm'>
```



Milestone 3: Data Preprocessing

Before training the model, the dataset was cleaned and prepared to ensure better performance.

Activity 1: Feature Selection and Cleaning

Some less relevant features were removed to reduce redundancy. We also verified that there were no missing values in the dataset. This step ensures that the model is trained on meaningful input features.

```
[12]: # Drop unwanted features
      df = df.drop(columns=[
          'stator_yoke',
          'stator_tooth',
          'stator_winding',
          'torque'
      ])

      # Define features and target
      X = df.drop('pm', axis=1)
      y = df['pm']
```

Activity 2: Train-Test Split and Feature Scaling

The dataset was divided into training and testing sets using an 80:20 ratio. After splitting, MinMaxScaler was applied to normalize feature values between 0 and 1. Scaling ensures consistent ranges across all features and improves model accuracy.

```
[15]: from sklearn.model_selection import train_test_split

      X_train, X_test, y_train, y_test = train_test_split(
          X, y, test_size=0.2, random_state=42
      )
```

```
[16]: from sklearn.preprocessing import MinMaxScaler

      mm = MinMaxScaler()

      # Fit only on training data
      X_train = mm.fit_transform(X_train)

      # Transform test data
      X_test = mm.transform(X_test)
```

Milestone 4: Model Building and Evaluation

Multiple regression models were trained to compare their performance in predicting PM temperature.

Activity 1: Training Different Models

Various models such as Linear Regression, Decision Tree, Random Forest, and Support Vector Machine were initialized and trained using the training dataset.

Activity 2: Model Evaluation and Comparison

The performance of each model was evaluated using RMSE and R^2 score. Among all models, the Random Forest Regressor achieved the highest R^2 score and lowest error, making it the best performing model.

[31]:

```
lr = LinearRegression()

dt = DecisionTreeRegressor(
    max_depth=12,
    random_state=3
)

rf = RandomForestRegressor(
    n_estimators=30,     # reduced trees
    max_depth=12,        # limited depth
    min_samples_split=5,
    n_jobs=-1,
    random_state=3
)

svm = LinearSVR(
    max_iter=5000,
    random_state=3
)
```

[32]:

```
evaluate_model(lr, X_train, X_test, y_train, y_test)
evaluate_model(dt, X_train, X_test, y_train, y_test)
evaluate_model(rf, X_train, X_test, y_train, y_test)
evaluate_model(svm, X_train, X_test, y_train, y_test)
```

```
LinearRegression
RMSE: 11.94739860505927
R2 Score: 0.6035557458782114
----------------------------------
DecisionTreeRegressor
RMSE: 5.314042011420618
R2 Score: 0.9215694270020754
----------------------------------
RandomForestRegressor
RMSE: 4.8745272697207005
R2 Score: 0.9340066113060523
----------------------------------
LinearSVR
RMSE: 12.070150199507934
R2 Score: 0.5953674930338954
----------------------------------
```

```
[35]: from sklearn import metrics

print("Linear Regression R2:", metrics.r2_score(y_test, p1))
print("Decision Tree R2:", metrics.r2_score(y_test, p2))
print("Random Forest R2:", metrics.r2_score(y_test, p3))
print("SVM R2:", metrics.r2_score(y_test, p4))
```

```
Linear Regression R2: 0.6035557458782114
Decision Tree R2: 0.9215694270020754
Random Forest R2: 0.9340066113060523
SVM R2: 0.5953674930338954
```

Activity 3: Saving the Best Model

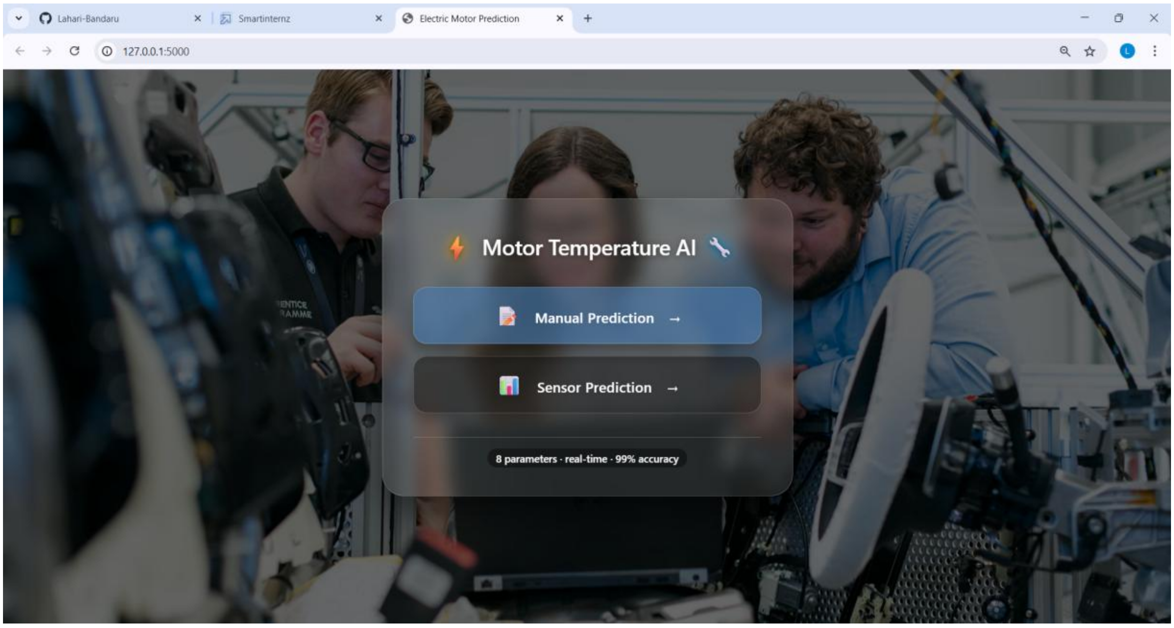
The best performing model (Random Forest) was saved using joblib so that it can be used later for real-time prediction in the Flask application.

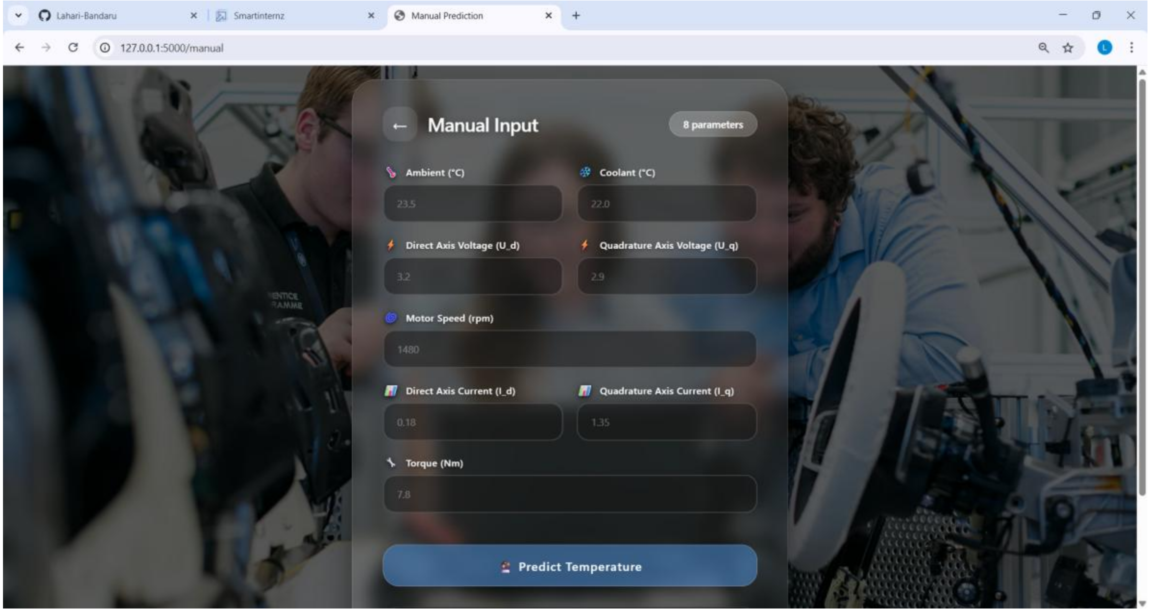
```
[41]: joblib.dump(rf, "model.save")

[41]: ['model.save']
```

Milestone 5: Application Building

The trained model was integrated into a Flask web application. Users can enter motor parameters through the web interface, and the system predicts the permanent magnet temperature instantly. The saved scaler ensures that input values are transformed in the same way as during training.





Future Enhancements

The system can be enhanced by integrating real-time IoT sensor data for automatic monitoring and deploying it on cloud platforms for better scalability and availability. Advanced machine learning models such as Random Forest, XGBoost, or Deep Learning can improve prediction accuracy. A multi-motor dashboard with real-time graphs and alert logs can be developed. Future improvements may also include automated SMS/Email alerts, database storage for historical analysis, mobile application support, and integration with complete predictive maintenance systems including vibration analysis and fault detection.

Conclusion

The Electric Motor Temperature Prediction System successfully applies machine learning to estimate internal motor temperature using operational sensor data. It eliminates the need for costly physical sensors and provides accurate real-time predictions through a Flask-based web interface. The system improves motor safety, reduces maintenance costs, and offers a scalable foundation for future enhancements such as IoT integration and cloud deployment.