PROJECT REPORT FORMAT

Electric Motor Temperature Prediction Using Machine Learning

To predict Electric Motor Temperature

1. Introduction

Electric Motor Temperature Prediction using Machine Learning is a predictive maintenance solution aimed at forecasting the temperature of electric motors in industrial settings. By analyzing historical operational data such as motor load, voltage, current, and environmental conditions, combined with machine learning algorithms, this project aims to predict motor temperatures accurately. The goal is to prevent overheating, avoid equipment failures, optimize maintenance schedules, and improve overall operational efficiency.

2. Objectives

The main objectives of this project are:

• To analyze electric motor performance parameters and predict motor temperature accurately.

• To identify potential overheating conditions before they cause system failure.

• To reduce unplanned downtime by implementing predictive maintenance strategies.

• To optimize energy consumption and operational efficiency.

• To deploy a machine learning model integrated with a web-based application.

3. Tools and Technologies Used

• Python Programming Language

• Anaconda Navigator

• Jupyter Notebook

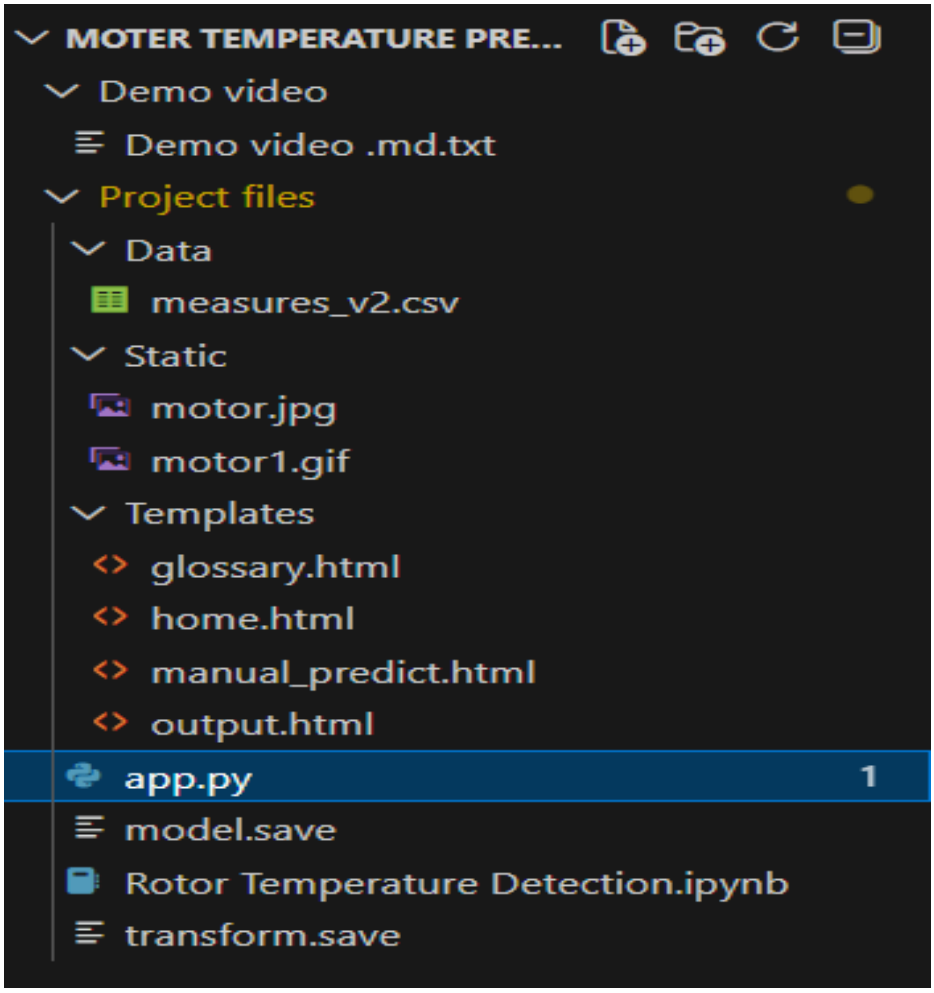
• Flask Framework for Web Application

• NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn libraries

• HTML for frontend interface

4.Project Structure

Create the project folder which contains files as shown



5.Prior Knowledge:

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

- ML Concepts

Supervised learning: https://youtu.be/QeKshry8pWQ

Unsupervised: https://youtu.be/NUXdtN1W1FE

Regression: https://youtu.be/NUXdtN1W1FE

Evalution Metrics: https://youtu.be/YSB7FtzeicA

Flask Basics: https://youtu.be/lj4I_CvBnt0

6.Project Flow

- The user interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated
- Once the model analyses the input the prediction is showcased on the UI

- To accomplish this, we have to complete all the activities listed below,

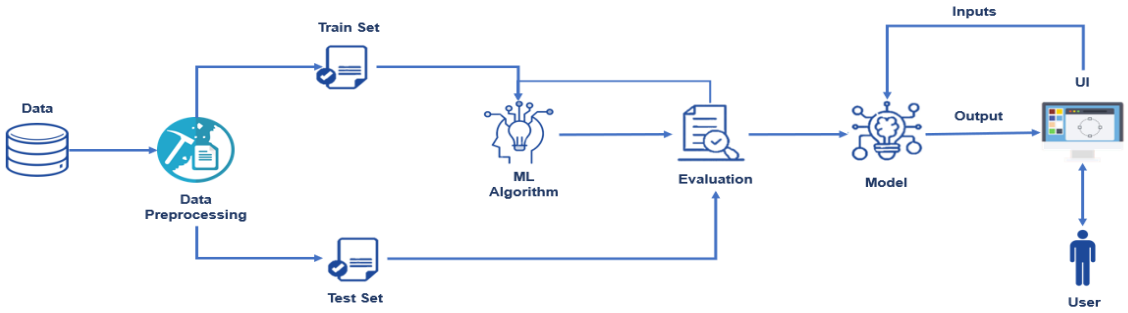
- Data collection
  - Collect the dataset or create the dataset
- Visualizing and analyzing data
  - Univariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Drop unwanted features
  - Checking for null values
  - Remove negative data
  - Handling outlier
  - Handling categorical data
  - Handling Imbalanced data
  - Splitting data into train and test
- Model building

- Import the model building libraries
- Initializing the model
- Training and testing the model
- Evaluating the performance of the model
- Save the model

- Application Building
  - Create an HTML file

7. System Architecture

The following diagram represents the workflow of the system:



8. Methodology

The methodology followed in this project includes several major phases as explained below:

- Data Collection:

The dataset containing sensor readings such as motor speed, voltage, current, and temperatures of different motor parts was collected from reliable industrial sources.

https://www.kaggle.com/datasets/wkirgsn/electric-motor-temperature

```
✅ Dataset loaded successfully!

--- First 5 Rows ---
        u_q    coolant  stator_winding       u_d  stator_tooth  motor_speed  \
0 -0.450682  18.805172       19.086670 -0.350055     18.293219     0.002866
1 -0.325737  18.818571       19.092390 -0.305803     18.294807     0.000257
2 -0.440864  18.828770       19.089380 -0.372503     18.294094     0.002355
3 -0.327026  18.835567       19.083031 -0.316199     18.292542     0.006105
4 -0.471150  18.857033       19.082525 -0.332272     18.291428     0.003133

        i_d       i_q         pm  stator_yoke    ambient    torque  profile_id
0  0.004419  0.000328  24.554214    18.316547  19.850691  0.187101          17
1  0.000606 -0.000785  24.538078    18.314955  19.850672  0.245417          17
2  0.001290  0.000386  24.544693    18.326307  19.850657  0.176615          17
3  0.000026  0.002046  24.554018    18.330833  19.850647  0.238303          17
4 -0.064317  0.037184  24.565397    18.326662  19.850639  0.208197          17
```

- Visualizing and analyzing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques

9. Importing the libraries

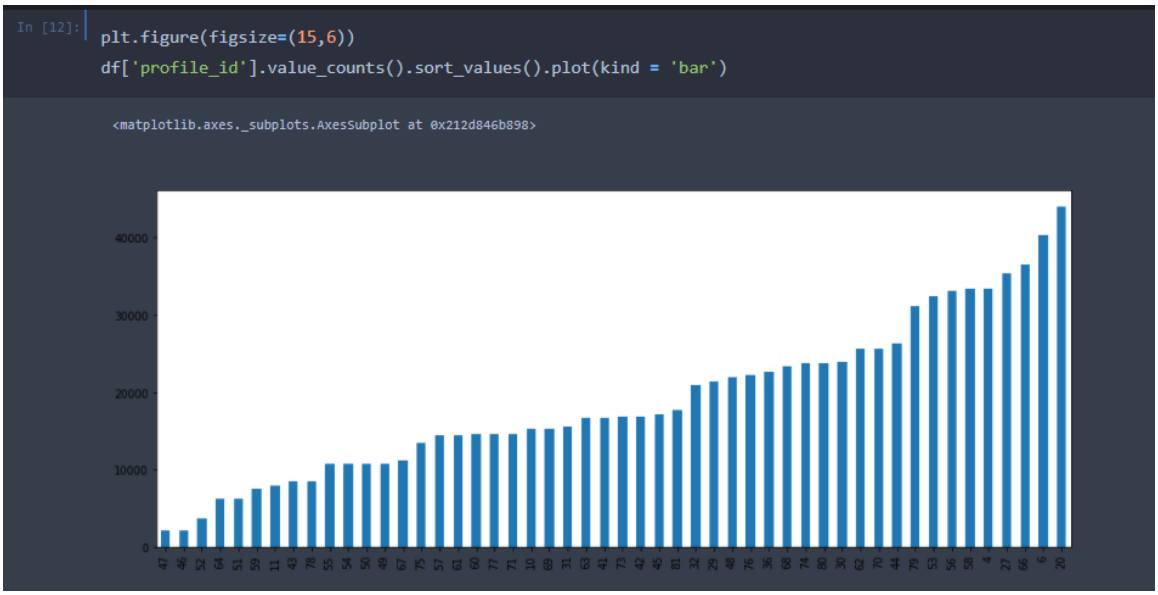
Import the necessary libraries as shown in the image To know about the packages refer to the link given on prerequisites

```
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')
```

Uni-variate analysis

Here we get to know about our data

Bar Graph: A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.



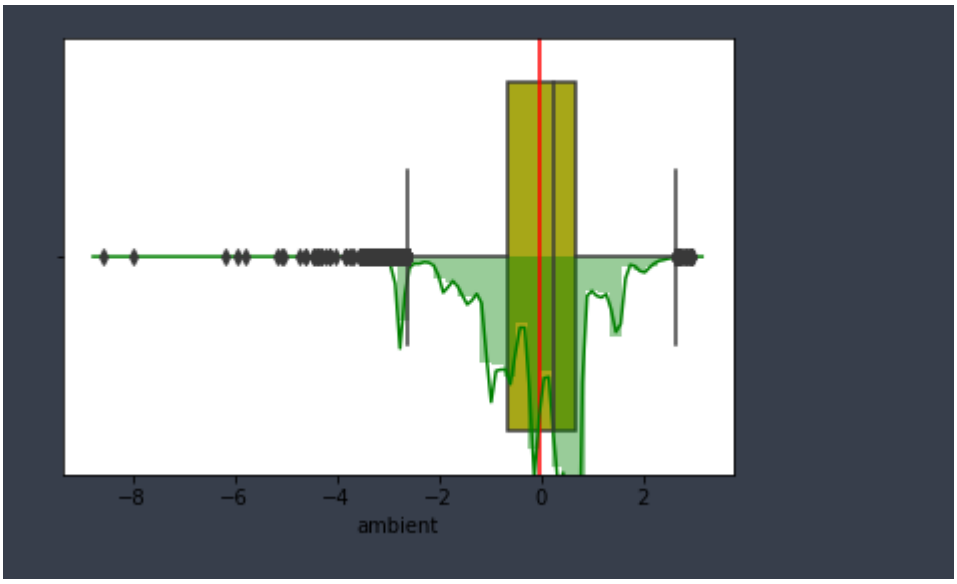
As we can see, session ids 66, 6, and 20 have the most number of measurements recorded.

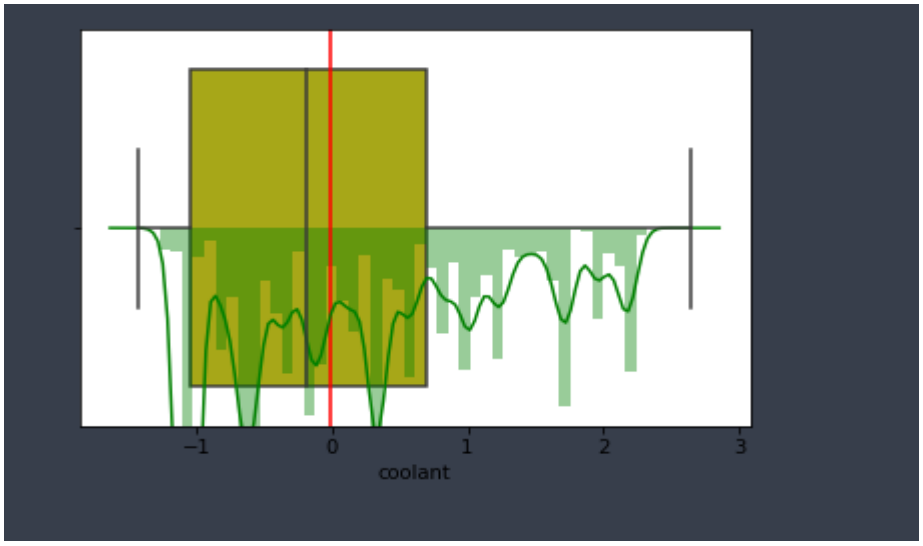
·Box plot:

A boxplot is a standardized way of displaying the distribution of data based on a five-number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about your outliers and what their values are.

Distribution plot:

The distribution plot is suitable for comparing range and distribution for groups of numerical data. Data is plotted as value points along an axis





All features boxplots are plotted and the following conclusions are drawn.

As we can see from the above plots, the mean and median for most of the plots are very close to each other. So the data seems to have low skewness for almost all variables

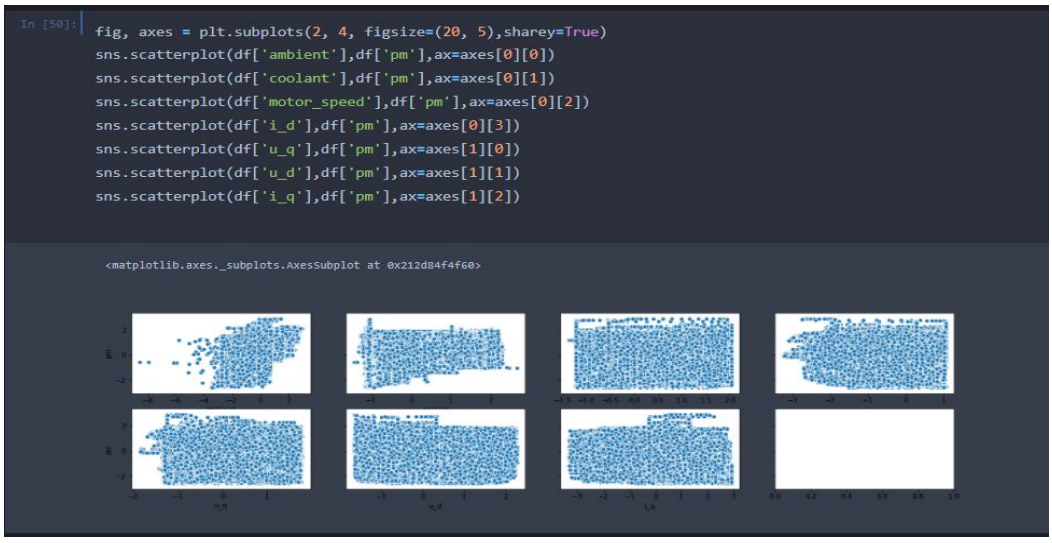
Multi-variate analysis

Multivariate analysis (MVA) is a Statistical procedure for the analysis of data involving more than one type of measurement or observation. It may also mean solving problems where more than one dependent variable is analyzed simultaneously with other variables.

Scatterplot:

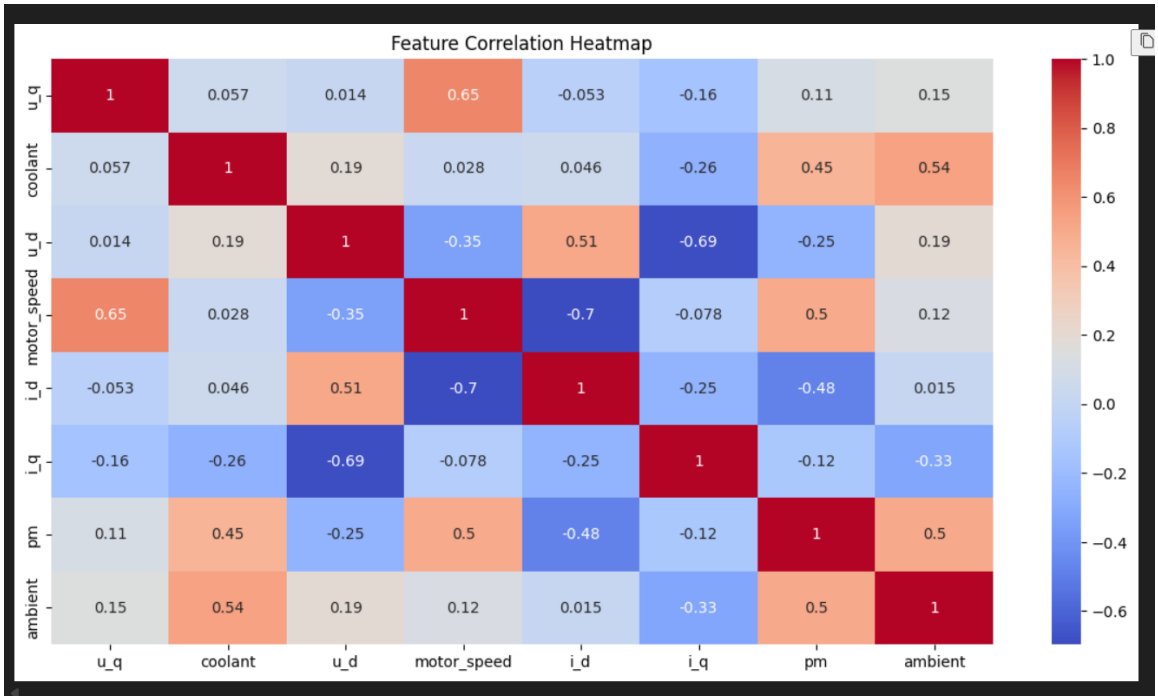
A scatter plot (also called a scatterplot, scatter graph, scatter chart, scattergram, or scatter diagram) is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data.

As we want to predict the temperatures of stator components and rotor(pm), we will drop these values from our dataset for regression. Also, torque is a quantity, which is not reliably measurable in field applications, so this feature shall be omitted in this modeling.

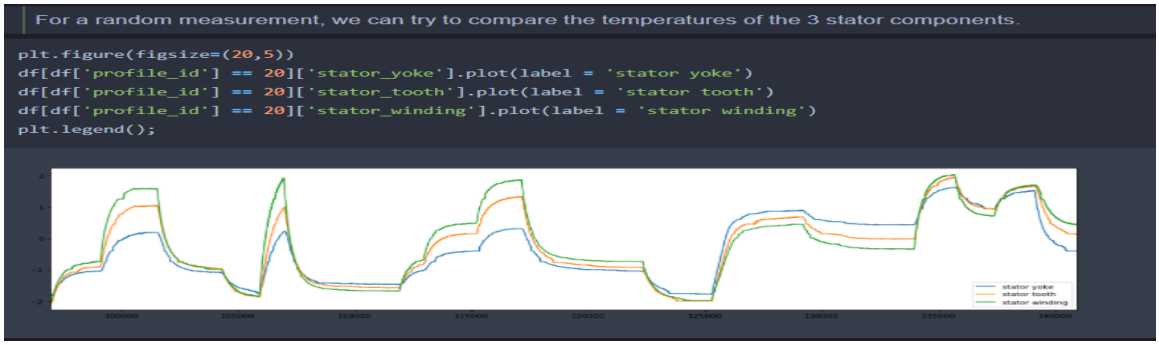


Heat-map:

A heat map is a data visualization technique that shows the magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space.



From the heatmap above, we can see that torque and q component of current are almost perfectly correlated. Also, there seems to be a very high correlation between temperature measurements of stator yoke, stator tooth, and stator winding



- As we can see from the plot, all three stator components follow a similar measurement variance.

- As the dataset author mentioned, the records in the same profile id have been sorted by time, we can assume that these recordings have been arranged a series of times.
- Due to this, we can infer that there has not been much time given for the motor to cool down in between recording the sensor data as we can see that initially the stator yoke temperature is low as compared to the temperature of stator winding but as we progress in time, the stator yoke temperature goes above the temperature of the stator winding.
- As profile_id is an id for each measurement session, we can remove it from any further analysis and model building.

```
In [20]: df.drop('profile_id',axis = 1,inplace=True)
         df_test.drop('profile_id',axis = 1,inplace=True)
```

Descriptive analysis

df.info():

This function is used to display a brief introduction about the data set such as the. of rows and columns, the Data type of each column, whether the null values are present in the column or no

t.

```
In [8]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 942677 entries, 0 to 982769
Data columns (total 13 columns):
ambient           942677 non-null float64
coolant           942677 non-null float64
u_d               942677 non-null float64
u_q               942677 non-null float64
motor_speed       942677 non-null float64
torque            942677 non-null float64
i_d               942677 non-null float64
i_q               942677 non-null float64
pm                942677 non-null float64
stator_yoke       942677 non-null float64
stator_tooth      942677 non-null float64
stator_winding    942677 non-null float64
profile_id        942677 non-null int64
dtypes: float64(12), int64(1)
memory usage: 100.7 MB
```

df.describe()

This function is used to analyze the descriptive statistics of the data such as mean, median, quartile values, maximum and minimum values of each column.

```
In [9]: df.describe()
```

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm
count	942677.000000	942677.000000	942677.000000	942677.000000	942677.000000	942677.000000	942677.000000	942677.000000	942677.
mean	-0.030010	-0.008238	-0.021835	0.007720	0.003210	0.020170	-0.002465	0.019958	-0.0050
std	1.007729	1.009503	0.994308	0.996054	0.996456	0.999063	1.000106	0.999683	1.00141
min	-8.573954	-1.429349	-1.655373	-1.861463	-1.371529	-3.345953	-3.245874	-3.341639	-2.6319
25%	-0.634542	-1.041886	-0.854390	-0.881885	-0.951878	-0.265042	-0.760764	-0.254672	-0.6678
50%	0.242495	-0.185147	0.225416	-0.089520	-0.140245	-0.121022	0.196713	-0.091449	0.09915
75%	0.681905	0.698607	0.356361	0.859836	0.855827	0.561778	1.013952	0.543750	0.67784
max	2.967117	2.649032	2.274734	1.793498	2.024164	3.016971	1.060937	2.914185	2.91745

- Data Preprocessing:

As we have understood how the data is. Let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps:

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

In the data frame, head() function is used to display the first 5 data. Our dataset has ambient, coolant,u_d,u_q,motor_speed , i_d , i_q ,stator_yoke,stator_winding and profile_id, pm(output)

The data was cleaned by handling missing values, removing unwanted features, scaling values, and splitting into training and testing datasets.

```
df.head()
```

ant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator_winding	profile_id
446	0.327935	-1.297858	-1.222428	-0.250182	1.029572	-0.245860	-2.522071	-1.831422	-2.066143	-2.018033	4
021	0.329665	-1.297686	-1.222429	-0.249133	1.029509	-0.245832	-2.522418	-1.830969	-2.064859	-2.017631	4
681	0.332771	-1.301822	-1.222428	-0.249431	1.029448	-0.245818	-2.522673	-1.830400	-2.064073	-2.017343	4
764	0.333700	-1.301852	-1.222430	-0.248636	1.032845	-0.246955	-2.521639	-1.830333	-2.063137	-2.017632	4
775	0.335206	-1.303118	-1.222429	-0.248701	1.031807	-0.246610	-2.521900	-1.830498	-2.062795	-2.018145	4

- Handling missing values

For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that the education column and previous year rating column have null values.

```
In [11]: df.isnull().sum()

ambient          0
coolant          0
u_d              0
u_q              0
motor_speed      0
torque           0
i_d              0
i_q              0
pm               0
stator_yoke      0
stator_tooth     0
stator_winding   0
profile_id       0
dtype: int64
```

- Handling outliers

With the help of a boxplot, outliers are visualized (refer to activity 3 univariate analysis). And here we are going to find the upper bound and lower bound of the Na_to_K feature with some mathematical formula.

- To find the upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find a lower bound instead of adding, subtract it with 1st quantile. Take the image attached below as your reference.
- If outliers are removed, we lose more data. It will impact model performance.
- Here removing outliers is impossible. So, the capping technique is used on outliers.
- Capping: Replacing the outliers with upper bound values.

- Normalizing the values

As we want to predict the temperatures of stator components and rotor(pm), we will drop these values from our dataset for regression. Also, torque is a quantity, which is not reliably measurable in field applications, so this feature shall be omitted in this modeling.

We are using minmax scaler ,which is a function in preprocessing module in sklearn library

```
mm = MinMaxScaler()
X = mm.fit_transform(X)
X_df_test = mm.fit_transform(X_df_test)
y = df['pm']
y_df_test = df_test['pm']
X = pd.DataFrame(X,columns = ['ambient', 'coolant', 'u_d', 'u_q', 'motor_speed', 'i_d','i_q'])
X_df_test = pd.DataFrame(X_df_test,columns = ['ambient', 'coolant', 'u_d', 'u_q', 'motor_speed', 'i_d','i_q
y.reset_index(drop = True,inplace = True)
y_df_test.reset_index(drop = True,inplace = True)
```

- Saving the transformation

```
import joblib
joblib.dump(mm,'transform.save')
```

```
['transform.save']
```

- Splitting data into train and test

Now let's split the Dataset into train and test sets. For splitting training and testing data, we are using the train_test_split() function from sklearn. As parameters, we are passing x_resample, y_resample, test_size, random_state.

For deep understanding refer to this link

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=3)
```

- Model Building:

Four regression algorithms were tested—Linear Regression, Decision Tree Regressor, Random Forest Regressor, and Support Vector Regressor (SVR). The Decision Tree model gave the best accuracy with an R^2 score of 96%.

```
import joblib
joblib.dump(mm,'transform.save')
```

```
['transform.save']
```

- Model Evaluation:

Model performance was evaluated using Root Mean Square Error (RMSE) and R^2 metrics, confirming high accuracy and minimal prediction error.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=3)
```

- Linear Regression

A function named LinearRegression is created and train and test data are passed as the parameters. Inside the function, the LinearRegression algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the performance of the model, we use root mean square error and r-square value.

- Decision tree model

A function named decision tree is created and train and test data are passed as the parameters. Inside the function, the DecisionTreeRegressor algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the performance of the model, we use root mean square error and r-square value.

- Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestRegressor algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluate the performance of the model, we use root mean square error and r-square value.

- Support Vector Machine model

A function named SVR is created and train and test data are passed as the parameters. Inside the function, SVR algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the performance of the model, we use root mean square error and r-square value.

```
In [51]: from sklearn.linear_model import LinearRegression
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.svm import SVR

In [52]: lr=LinearRegression()
         dr=DecisionTreeRegressor()
         rf =RandomForestRegressor()
         svm =SVR()
```

```
In [*]: lr.fit(X_train,y_train)
        dr.fit(X_train,y_train)
        rf.fit(X_train,y_train)
        svm.fit(X_train,y_train)
```

- Compare the model

```
from sklearn import metrics

print(metrics.r2_score(y_test,p1))
print(metrics.r2_score(y_test,p2))
print(metrics.r2_score(y_test,p3))
print(metrics.r2_score(y_test,p4))
```

```
0.9698725757690718
0.47757469778170836
```

Out of all the models. The decision Tree regressor is giving an r2-score of 96%, it means the model is able to explain 96% of the data. so we will select the decision tree model and save it.

To know more about R2-score, please refer to the below link

- Evaluating performance of the model
  Evaluating the model by using RMSE (Root Mean Squared Error)

```
In [20]: from sklearn.metrics import mean_squared_error

In [26]: print(mean_squared_error(y_test,p1))
```

```
0.030187256377134934
```

- Save the model

```
In [27]: import joblib

In [65]: joblib.dump(dr,"model.save")

         ['model.save']
```

- Application Development:

Build the python flask app

In the flask application, the user values are taken from the HTML page

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import joblib
app = Flask(__name__)
model = joblib.load("model.save")
trans=joblib.load('transform.save')


app = Flask(__name__)
```

Load the home page

```
app = Flask(__name__)

@app.route('/')
def predict():
    return render_template('Manual_predict.html')
```

Prediction function

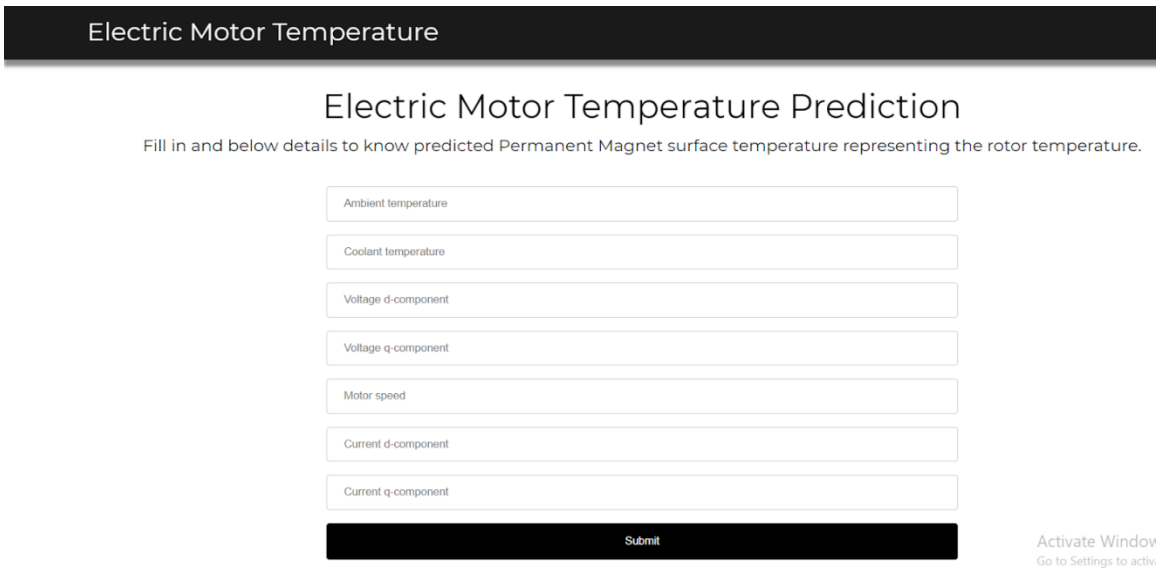
```python
@app.route('/y_predict',methods=['POST'])
def y_predict():
    x_test = [[float(x) for x in request.form.values()]]
    print('actual',x_test)
    x_test=trans.transform(x_test)
    print(x_test)
    pred = model.predict(x_test)

    return render_template('Manual_predict.html', prediction_text=('Permanent Magnet surface temperature: ',pred[0]))


if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

Run the application

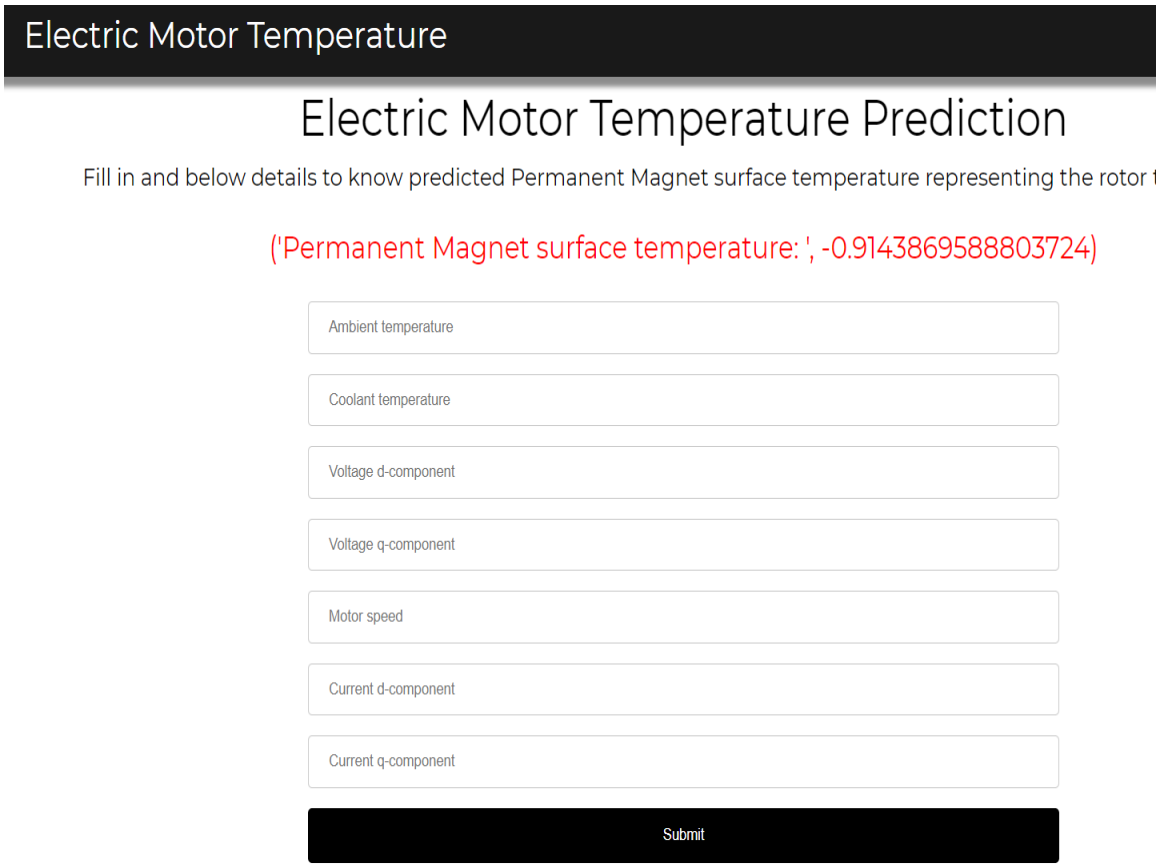
Open the anaconda prompt go to the project folder and in that go to the flask folder and run the python file by using the command "python app.py"

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 135-972-108
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

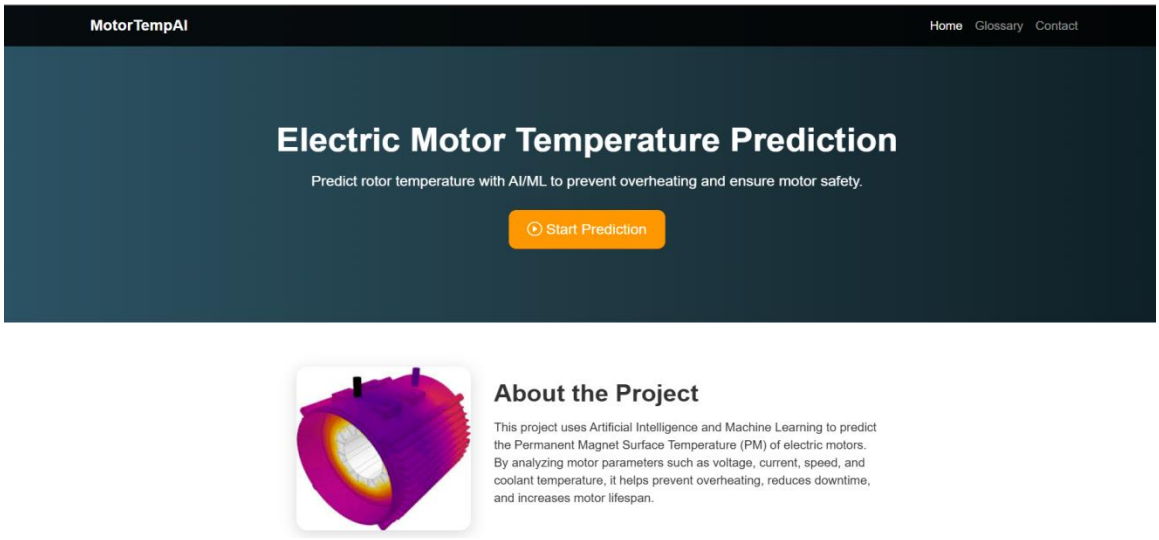
copy the HTTP link and paste it into a browser tab.
The following page will be opened



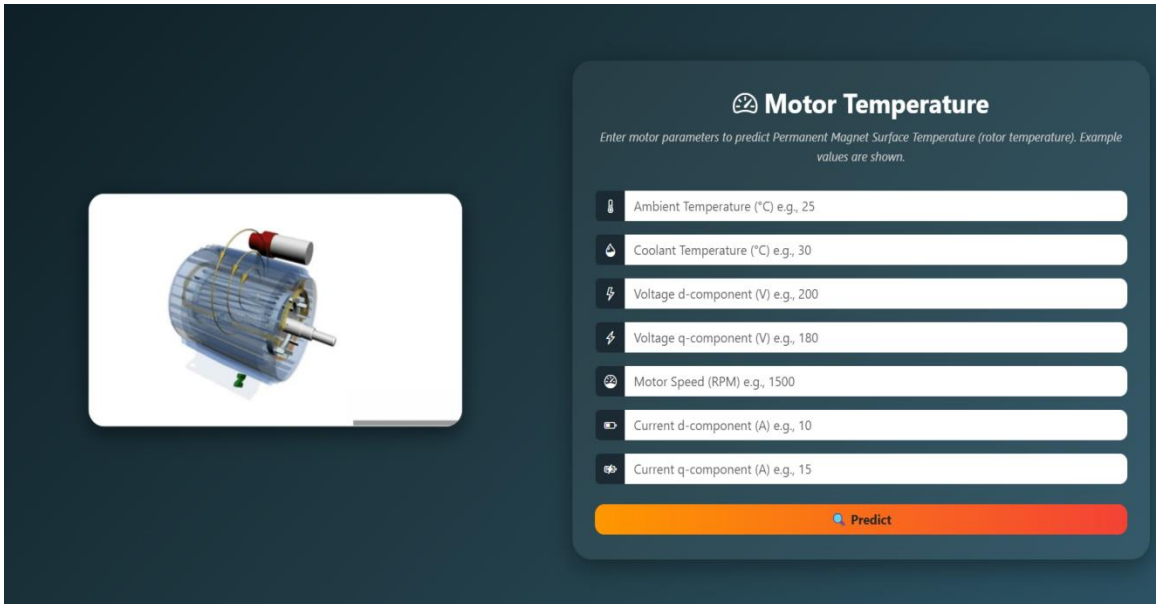
Enter the values and click on predict button, it will predict the temperature of an electric motor



- Electric motor Temeperature prediction



A Flask-based web interface was created for user input and prediction display, where users can enter motor parameters and get the predicted temperature output.



10. Advantages and Disadvantages

Advantages:

- Improves maintenance efficiency by enabling predictive insights.
- Reduces equipment downtime and operational costs.
- Enhances energy efficiency and reliability of motors.
- Provides real-time predictions via web interface.
- Enables proactive maintenance.
- Improves motor lifespan and reliability.
- Can handle complex relationships among variables.
- Helps in reducing energy costs.
- Scalable to other industrial systems.

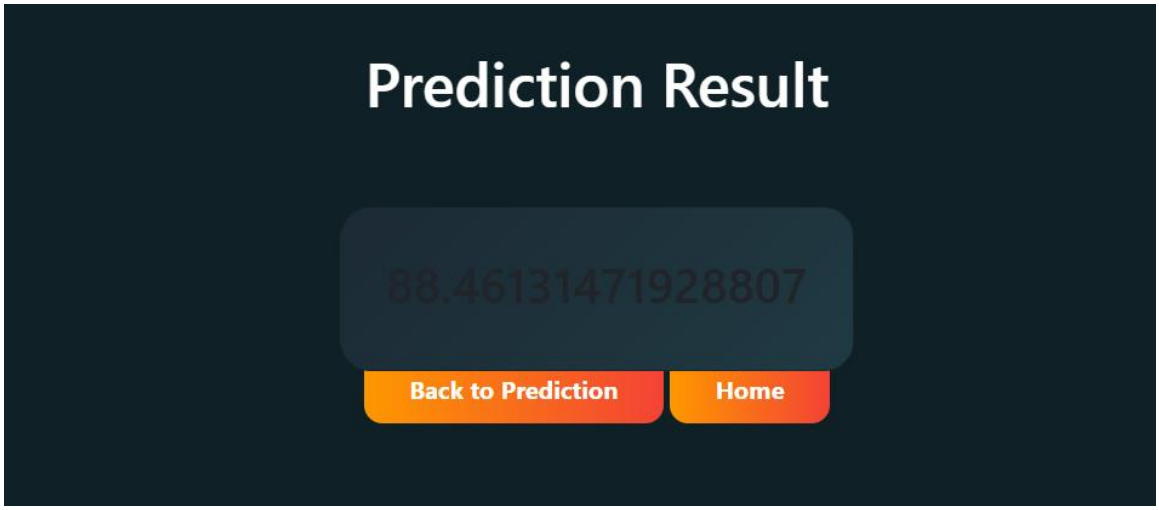
Disadvantages:

- Requires large and high-quality datasets for training.
- Dependent on sensor accuracy and consistent data input.
- Model performance can degrade with unseen environmental factors.

11. Results

After training and evaluation, the Decision Tree Regressor model achieved an R^2 score of 96% and a very low RMSE of 0.03, indicating that the predicted motor temperatures were

very close to actual values. This validates the efficiency and reliability of the proposed approach.



12. Applications / Use Cases

• Manufacturing plants for predictive maintenance and fault detection.

• HVAC systems to monitor motor temperature and avoid performance loss.

• Automotive industries for monitoring motor components in electric vehicles.

• Energy plants and large-scale industries relying on continuous motor operation.

13. Future Enhancements

• Integrate IoT-based live sensor data for real-time monitoring.

• Implement deep learning models for improved prediction accuracy.

• Deploy the system on cloud platforms for remote accessibility.

• Create a mobile-friendly interface for easy user access.

14. Conclusion

The project 'Electric Motor Temperature Prediction using Machine Learning' successfully demonstrates the application of ML algorithms for predictive maintenance. By predicting

temperature with high accuracy, it helps in preventing failures, reducing downtime, and improving overall system performance. With future integration of IoT and cloud technologies, the project can evolve into a fully automated industrial solution.