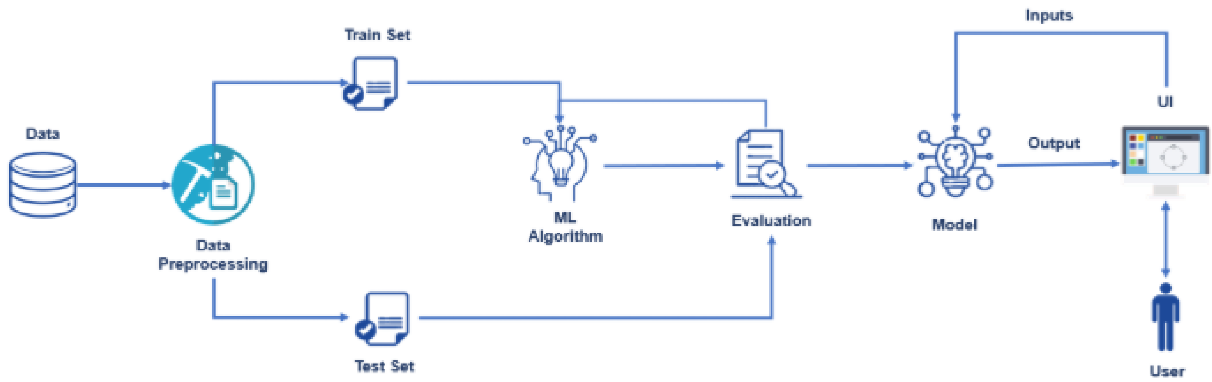
Electric Motor Temperature Prediction Using Machine Learning

Group Members :-
Pratik Ghadge
Tapan Belapurkar
Nupoor Raut
Yash Kataria

Electric Motor Temp Prediction Using Machine Learning

The Electric Motor Rotor Temperature Prediction project uses a machine learning model to estimate rotor temperature in PMSM drives. It takes 10 input features like voltage, current, speed, torque, and stator/coolant temperatures. Built with Flask, it provides a web interface for real-time predictions.

Technical Architecture:



Project Flow:

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

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

- Define Problem / Problem Understanding
  - 0 Specify the business problem
    Business requirements
  - ○ Literature Survey
  - ○ Social or Business Impact.
- Data Collection & Preparation
  - 0 Collect the dataset
  - ○ Data Preparation
- Exploratory Data Analysis
  - 0 Descriptive statistical
  - ○ Visual Analysis
- Model Building
  - 0 Training the model in multiple algorithms
  - ○ Testing the model
- Performance Testing & Hyperparameter Tuning
  - 0 Testing model with multiple evaluation metrics
  - ○ Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
  - 0 Save the best model
  - ○ Integrate with Web Framework
- Project Demonstration & Documentation
  - 0 Record explanation Video for project end to end solution
  - ○ Project Documentation-Step by step project development procedure

Prior Knowledge:

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

· ML Concepts

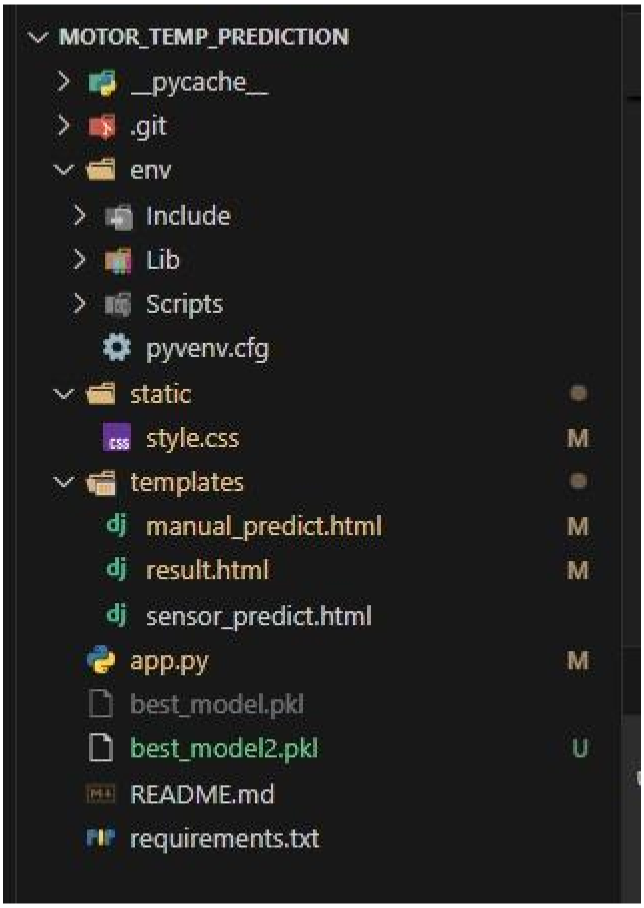
○ Supervised learning: https://www.javatpoint.com/supervised-machine-learning ○ Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning

- 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 Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- best_model.pkl is our saved model. Further we will use this model for flask integration.
- Data Folder contains the Dataset used
- Manual_Predict.html is the code for UI of application.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Rotor temperature in electric motors is a critical parameter that affects performance, efficiency, and safety. However, direct measurement of rotor temperature is often impractical in real-world applications due to cost and sensor limitations. This project aims to build a machine learning model that predicts rotor temperature using accessible sensor data such as voltage, current, torque, and ambient conditions. Accurate prediction enables proactive maintenance, reduces downtime, and enhances motor reliability across industrial and automotive systems.

Activity 2: Business requirements

An electric motor temperature prediction system must meet several business requirements to be effective and scalable:

- **Accurate and Real-Time Predictions:** The model should use reliable sensor data to deliver precise rotor temperature estimates, ensuring timely decision-making for maintenance and safety.
- **Flexibility and Scalability:** The system should be adaptable to different motor types, operating conditions, and sensor configurations without major reengineering.
- **Integration Capability:** The model must be easily integrable with existing industrial monitoring systems or embedded platforms for seamless deployment.
- **User-Friendly Interface:** The web-based UI should be intuitive and accessible for engineers, technicians, and maintenance personnel to input data and interpret results.
- **Cost Efficiency:** The solution should minimize the need for expensive hardware by leveraging existing sensor data, reducing operational costs.

.

Activity 3: Literature Survey (Student Will Write)

Recent advancements in machine learning have significantly improved the accuracy of rotor temperature prediction in Permanent Magnet Synchronous Motors (PMSMs). While traditional thermal models, such as Lumped Parameter Thermal Networks (LPTNs) and Finite Element Analysis (FEA), provide foundational insights, they often struggle with adapting to real-time, dynamic operating conditions. This has led to the rise of data-driven approaches, where sensor measurements and operational parameters are leveraged to train predictive models.

In this project, four machine learning algorithms—**Linear Regression**, **Decision Tree**, **Random Forest**, and **Support Vector Machine (SVM)**—were evaluated for rotor temperature prediction. The **Random Forest Regressor** emerged as the best-performing model, achieving an **R^2 score of 0.9999246** and a **Mean Squared Error (MSE) of 0.01309**, demonstrating exceptional predictive accuracy. This result aligns with previous studies in electric drive systems and electric vehicles, where Random Forest has been shown to effectively capture complex nonlinear relationships between motor parameters and thermal behavior. Research published in journals such as *IEEE Transactions on Industrial Electronics* and *Applied Sciences* highlights its robustness, ability to handle noisy sensor data, and generalization performance.

The **Decision Tree Regressor** also performed remarkably well, with an **R^2 score of 0.9997512** and an **MSE of 0.04320**, confirming findings from earlier works where tree-based models provided competitive accuracy with high interpretability. Similarly, **Support Vector Machine (SVM)** achieved an **R^2 score of 0.99744**, reflecting its strong performance in handling high-dimensional feature spaces, as reported in literature on motor diagnostics and fault prediction.

Interestingly, **Linear Regression**, while simpler, attained an **R^2 score of 0.990077**, validating its relevance for quick baseline modeling, though it falls short in capturing complex patterns compared to ensemble and nonlinear methods.

These results reinforce existing research trends—ensemble methods such as Random Forest and Gradient Boosting consistently outperform simpler models in PMSM temperature prediction due to their ability to model complex feature interactions, maintain robustness against outliers, and adapt well to varying operational conditions.

The best-performing model in this project—RandomForestRegressor—was saved as `best_model2.pkl` and is ready for deployment in real-time motor monitoring systems.

Activity 4: Social or Business Impact.

Social Impact :- Accurate prediction of rotor temperature enhances the safety and reliability of electric motors used in vehicles, industrial machinery, and renewable energy systems. By enabling early detection of overheating, this system helps prevent mechanical failures, reduces the risk of accidents, and supports the transition to smarter, more sustainable technologies.

Business Model/Impact :- From a business perspective, predictive temperature modeling reduces maintenance costs and unplanned downtime. Manufacturers and fleet operators can optimize motor design, schedule proactive servicing, and extend equipment lifespan. This leads to improved operational efficiency, reduced warranty claims, and greater customer satisfaction in sectors like automotive, aerospace, and manufacturing

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/code/sumeetsawant/electrical-motor-temperature-pmsm

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

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

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import joblib
```

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

```
import pandas as pd
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

- For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
         u_q   coolant  stator_winding       u_d  stator_tooth  motor_speed  \
0  -2.269314 -0.447817       -1.865502  2.010356       -2.0296    -2.805925
1  -2.269314 -0.419236       -1.865502  2.010356       -2.0296    -2.805925
2  -2.269314 -0.397481       -1.865502  2.010356       -2.0296    -2.805925
3  -2.269314 -0.382981       -1.865502  2.010356       -2.0296    -2.805925
4  -2.269314 -0.337194       -1.865502  2.010356       -2.0296    -2.805925
5  -2.269314 -0.242239       -1.865502  2.010356       -2.0296    -2.805925
6  -2.269314 -0.156569       -1.865502  2.010356       -2.0296    -2.805925
7  -2.269314 -0.115721       -1.865502  2.010356       -2.0296    -2.805925
8  -2.269314 -0.088665       -1.865502  2.010356       -2.0296    -2.805925
9  -2.269314 -0.058233       -1.865502  2.010356       -2.0296    -2.805925
10 -2.269314 -0.035006       -1.865502  2.010356       -2.0296    -2.805925
11 -2.269314 -0.023056       -1.865502  2.010356       -2.0296    -2.805925
12 -2.269314 -0.053436       -1.865502  2.010356       -2.0296    -2.805925
13 -2.269314 -0.105965       -1.865502  2.010356       -2.0296    -2.805925
14 -2.269314 -0.183457       -1.865502  2.010356       -2.0296    -2.805925

         i_d       i_q  stator_yoke    torque        pm
0   1.892995 -2.047535    -2.186312 -2.140795  50.38169
1   1.892995 -2.047535    -2.186312 -2.140795  50.38169
2   1.892995 -2.047535    -2.186312 -2.140795  50.38169
3   1.892995 -2.047535    -2.186312 -2.140795  50.38169
4   1.892995 -2.047535    -2.186312 -2.140795  50.38169
5   1.892995 -2.047535    -2.186312 -2.140795  50.38169
6   1.892995 -2.047535    -2.186312 -2.140795  50.38169
7   1.892995 -2.047535    -2.186312 -2.140795  50.38169
8   1.892995 -2.047535    -2.186312 -2.140795  50.38169
9   1.892995 -2.047535    -2.186312 -2.140795  50.38169
10  1.892995 -2.047535    -2.186312 -2.140795  50.38169
11  1.892995 -2.047535    -2.186312 -2.140795  50.38169
12  1.892995 -2.047535    -2.186312 -2.140795  50.38169
13  1.892995 -2.047535    -2.186312 -2.140795  50.38169
14  1.892995 -2.047535    -2.186312 -2.140795  50.38169
```

Activity 2: Data Preparation

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 Outliers

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.

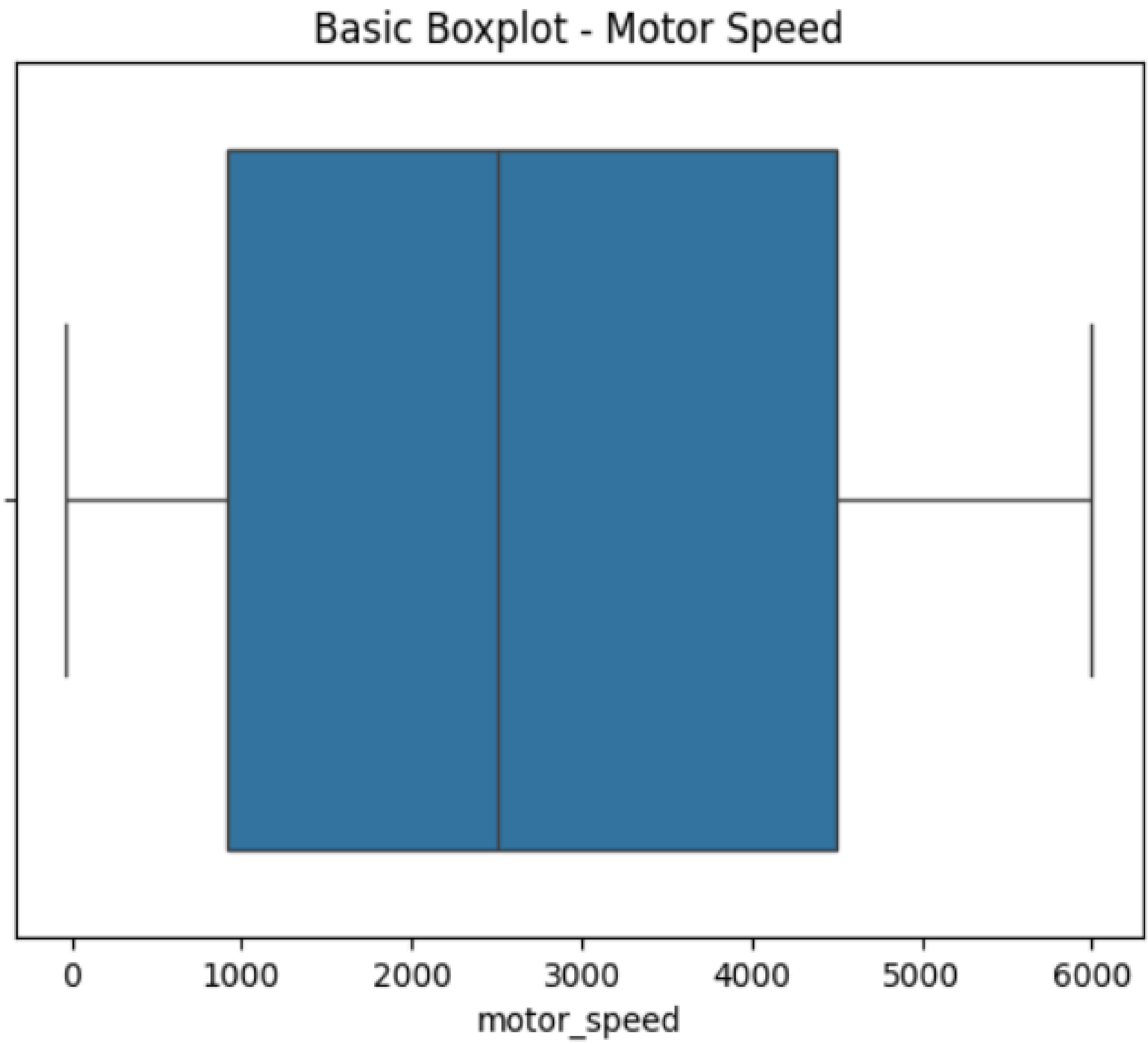
Activity 2.1: Handling missing values

- For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

Activity 2.2: Handling Outliers

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound motor speed feature with some mathematical formula.

- From the below diagram, we could visualize that the **motor_speed** feature has **no visible outliers**. The **boxplot** from the `seaborn` library is used here to represent the distribution. The box represents the **Interquartile Range (IQR)**, which is the difference between the third quartile (Q3) and the first quartile (Q1). To find the **upper bound**, we multiply the IQR by **1.5** and add it to Q3. To find the **lower bound**, we subtract **1.5 × IQR** from Q1. Any data points lying beyond these bounds are considered outliers. In this case, all data points fall within the whiskers, indicating there are no outliers in the motor_speed feature..


Basic Boxplot - Motor Speed
0
1000
2000
3000
4000
5000
6000
motor_speed

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

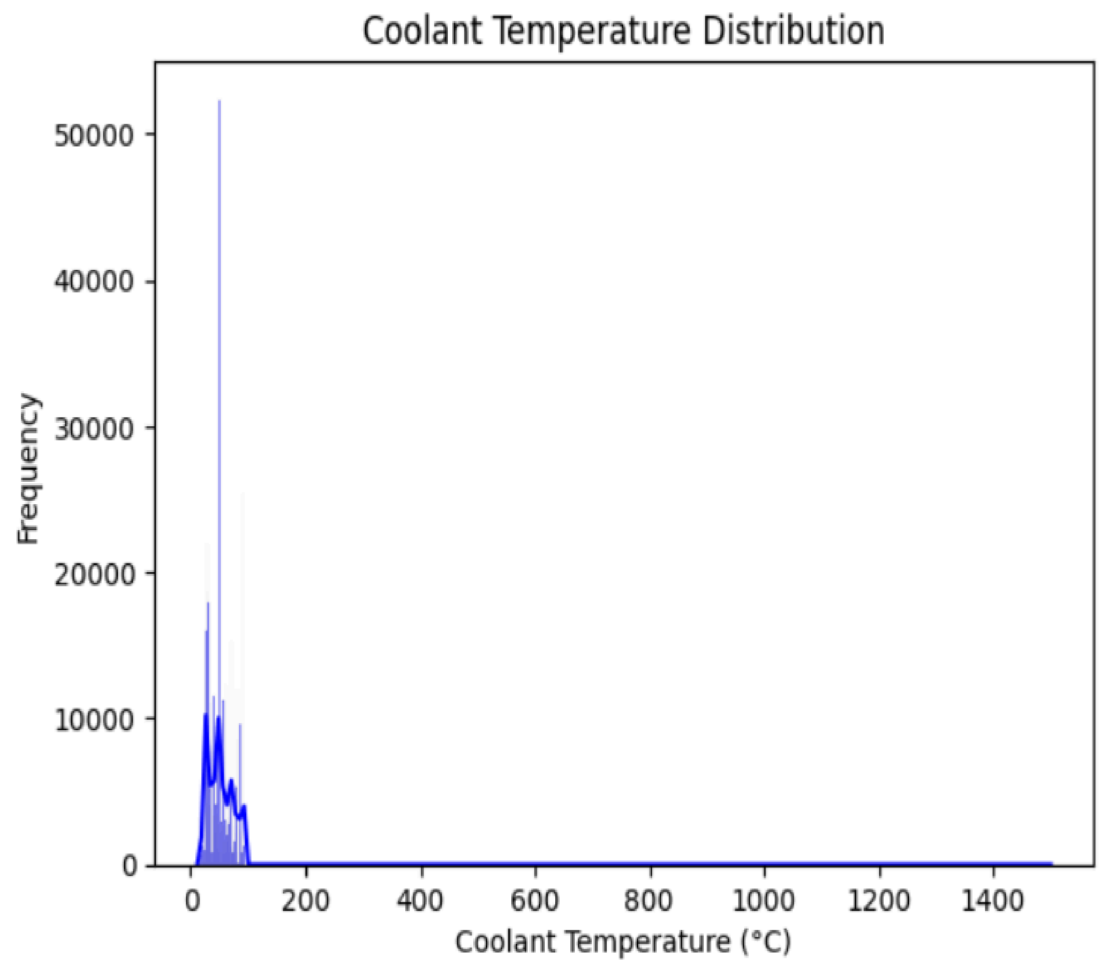
```
              622552  1499.9962059786096  -15.499266522499031  \
count  450668.000000       450668.000000        450668.000000
mean       66.127199         2081.769998           -58.869911
std        20.400765         1794.360195            57.905013
min        18.268347           -0.031091          -269.072268
25%        51.290298            0.006728          -100.384398
50%        64.940965         1907.049938           -46.504988
75%        83.243935         3632.480584            -2.001338
max       111.946423         5981.344042             0.006143

       70.79406241903523  62.948756763751675  82.70612192314437  \
count      450668.000000       450668.000000      450668.000000
mean           10.837978           64.762317          59.264565
std            91.835745           14.023592          18.901180
min          -293.426793           21.266710          18.281317
25%           -22.331519           55.740580          44.456936
50%             1.097740           65.517263          57.456392
75%            61.145339           74.509924          75.270239
max           299.618486           97.552674         101.147964

       23.75990468468147  50.00000000009659             45
count      450668.000000       4.506680e+05  450668.000000
mean           25.698206       8.558179e+00      61.816051
std             1.290279       7.623624e+01      10.738274
min             8.783478      -2.464667e+02      42.000000
25%            24.864168      -1.863306e+01      56.000000
50%            25.806514      3.705492e-320      63.000000
75%            26.384509       5.000000e+01      69.000000
max            30.171988       2.486054e+02      79.000000
```

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

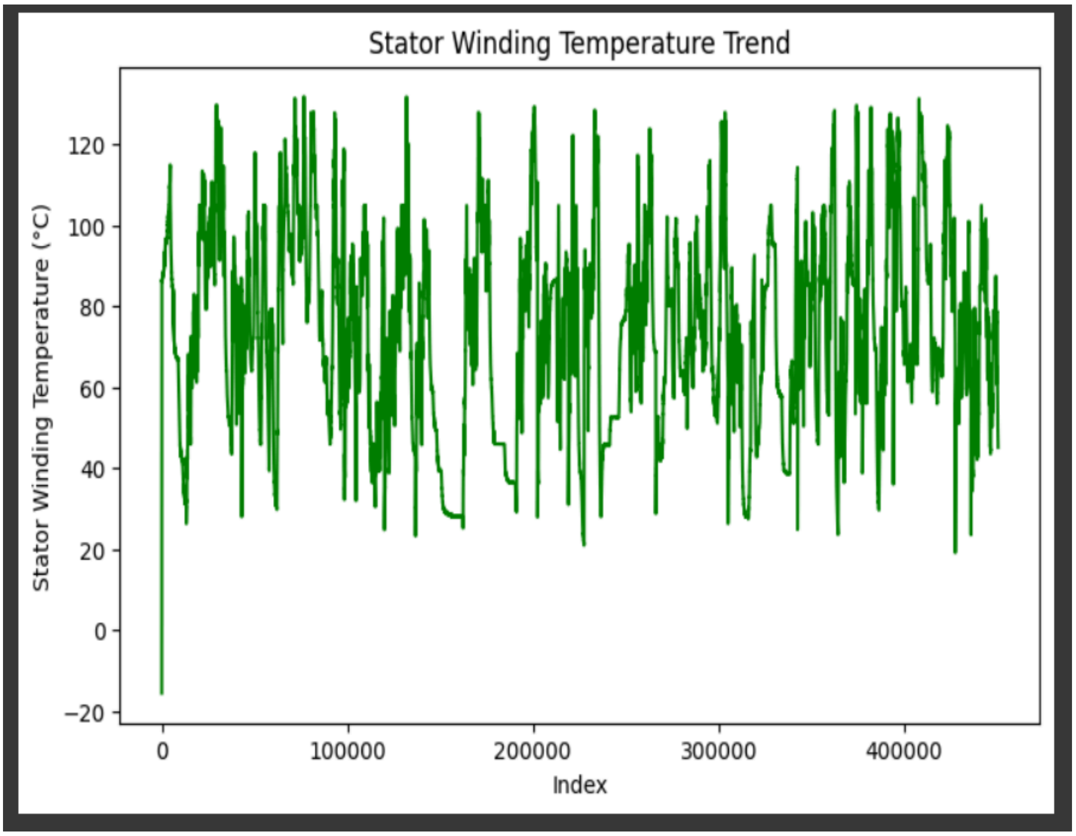
Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature.



The first plot shows the distribution of coolant temperature in °C.

- The x-axis represents coolant temperature, and the y-axis shows the frequency of occurrences.
- Most readings are concentrated between **0°C and 200°C**, with a sharp spike at certain values indicating frequent measurement at those points.
- The long tail towards higher temperatures suggests a few extreme readings, but they are rare.
- This distribution helps in understanding the normal operating range of coolant temperature and identifying possible outliers.



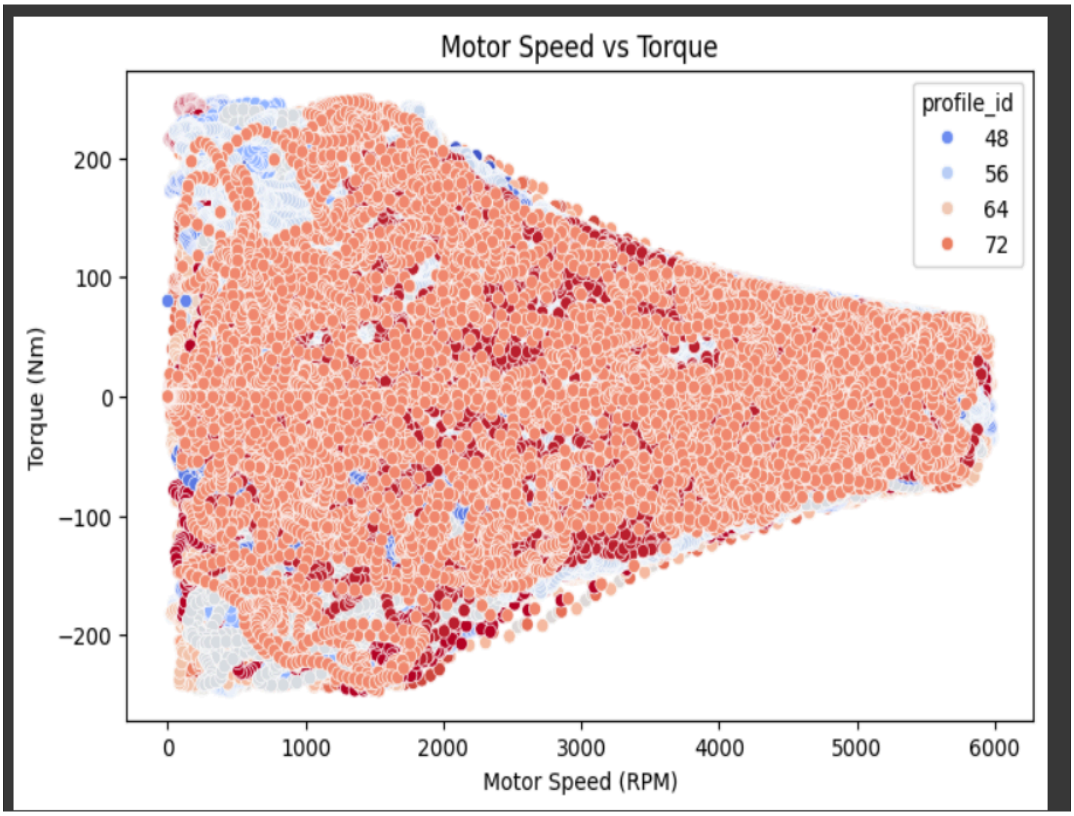
The second graph represents how **stator winding temperature** varies over the dataset's index (likely a time sequence).

- The y-axis shows temperature in °C, and the x-axis represents the sequence of measurements.
- The temperatures fluctuate significantly, ranging from around **-20°C to above 120°C**.
- Frequent variations indicate dynamic changes in motor operation, possibly due to changes in load, speed, or cooling system efficiency.
- Observing the peaks and drops can help in detecting operational anomalies or potential overheating issues.

Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis.

.



The third plot visualizes the relationship between **motor speed (RPM)** and **torque (Nm)**, with data points colored by `profile_id`.

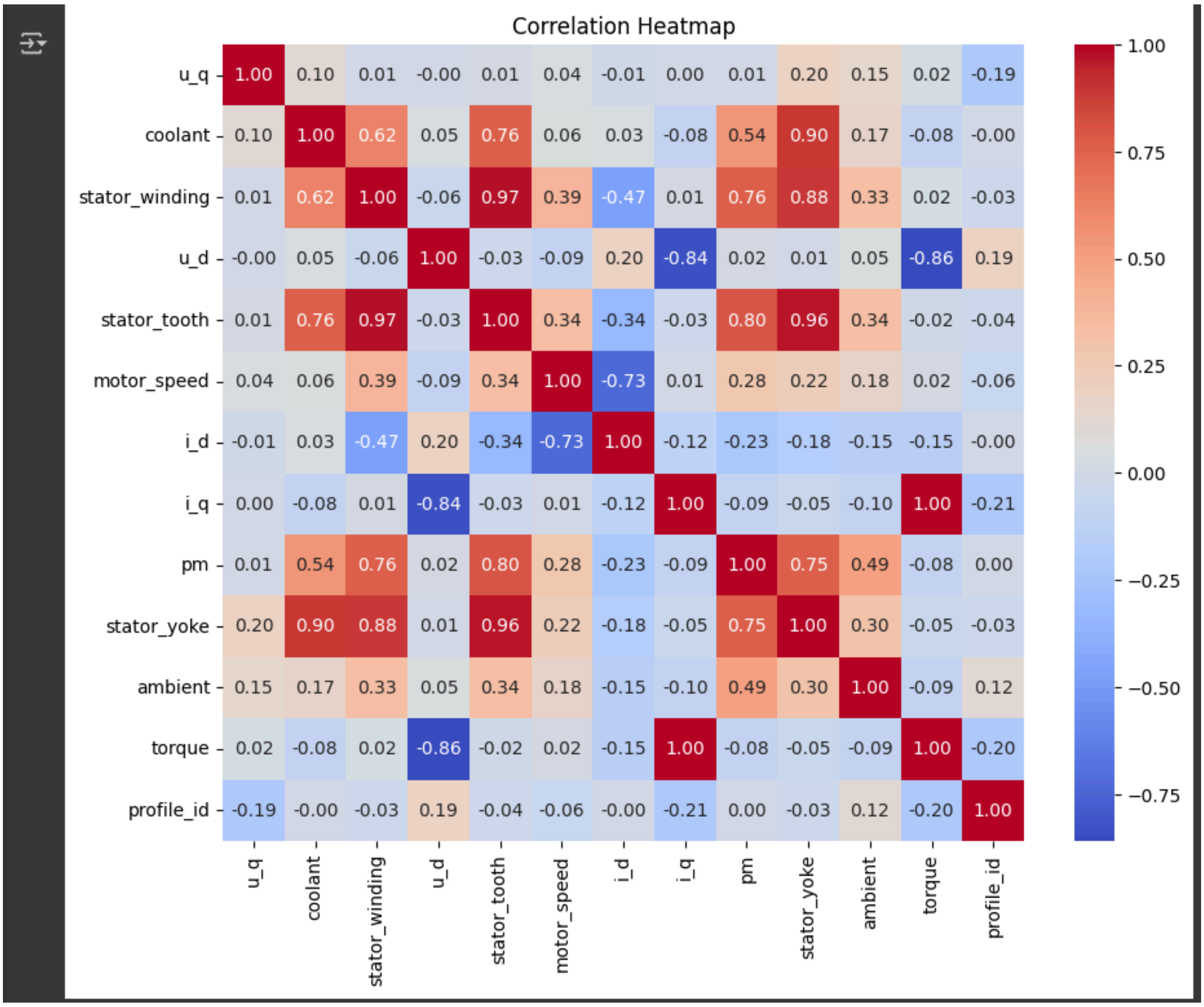
- The scatter of points shows that higher motor speeds are generally associated with a narrowing range of torque values.
- At low motor speeds (near 0 RPM), torque values vary widely from **-200 Nm to +200 Nm**.
- As motor speed increases beyond 3000 RPM, torque variation becomes more constrained.
- The color encoding for `profile_id` indicates operational profiles, with some profiles more common at specific speed–torque combinations.
- This plot helps in understanding performance patterns across different operating profiles and can be useful in detecting unusual operating conditions.

Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package.of 22

The heatmap visualizes the pairwise correlations between various variables in a dataset, using a color-coded matrix to represent the strength and direction of these relationships. Each cell shows the

Pearson correlation coefficient between two variables, ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation). Reddish tones indicate strong positive correlations, while bluish tones indicate strong negative correlations. The diagonal elements are always 1.00, representing perfect self-correlation. This visualization helps identify patterns, detect multicollinearity, and guide feature selection for further analysis or modeling.



Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
x=daata1.iloc[:,0:30]
y=daata1.iloc[:,30:]
```

```
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=0)
```

Handling Imbalanced dataset

- Imbalanced data is a common problem in machine learning and data analysis, where the number of observations in one class is significantly higher or lower than the other class. Handling imbalanced data is important to ensure that the model is not biased towards the majority class and can accurately predict the minority class.
- Here we are using SMOTE Technique.

```
from imblearn.over_sampling import SMOTE
smt=SMOTE()
X_train, y_train = smt.fit_resample(X_train, y_train)
```

Scaling

- Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.
- Here we are using Standard Scaler.
- This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by:

  X_scaled = (X - X_mean) / X_std

```
from sklearn.preprocessing import StandardScaler
Std_scaler=StandardScaler()
```

```
X_train = Std_scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train, columns=x.columns)
```

```
X_test = Std_scaler.transform(X_test)
X_test = pd.DataFrame(X_test, columns=x.columns)
```

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance.

Activity 1.1: Linear Regression Model

First, `LinearRegression` was imported from the `sklearn.linear_model` library. The training (`X_train`, `y_train`) and testing (`X_test`, `y_test`) datasets were loaded using `pandas.read_csv()`.
The `LinearRegression` model was initialized and trained using the `.fit()` method on the training data. Predictions on the test data were generated using `.predict()`.
The model's performance was evaluated using the Mean Squared Error (MSE) and R^2 score from the `sklearn.metrics` library.
Finally, the trained model was saved in `.pkl` format using the `joblib.dump()` function for future use.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import joblib

print("Linear Regression..........")
X_train = pd.read_csv('X_train.csv')
X_test = pd.read_csv('X_test.csv')
y_train = pd.read_csv('y_train.csv').values.ravel()
y_test = pd.read_csv('y_test.csv').values.ravel()
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
print("Linear Regression - MSE:", mse_lr, "R2:", r2_lr)
joblib.dump(lr_model, 'linear_regression_model.pkl')
print("Linear Regression model saved as linear_regression_model.pkl
```

Activity 1.2: Decision Tree Model

The `DecisionTreeRegressor` was imported from the `sklearn.tree` library. The datasets were loaded similarly using `pandas.read_csv()`.
A `DecisionTreeRegressor` model was initialized with a fixed `random_state` for reproducibility and trained using `.fit()` on the training dataset.
Predictions were made on the test dataset, and the performance was measured using MSE and R^2 metrics.
The trained model was stored in `.pkl` format with `joblib.dump()` for later deployment.

```python
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib

print("Decision Tree Model...........")
X_train = pd.read_csv('X_train.csv')
X_test = pd.read_csv('X_test.csv')
y_train = pd.read_csv('y_train.csv').values.ravel()
y_test = pd.read_csv('y_test.csv').values.ravel()
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)
print("Decision Tree - MSE:", mse_dt, "R2:", r2_dt)
joblib.dump(dt_model, 'decision_tree_model.pkl')
print("Decision Tree model saved as decision_tree_model.pkl")
```

Activity 1.3: Random Forest Model

The `RandomForestRegressor` was imported from the `sklearn.ensemble` library. After loading the datasets, a `RandomForestRegressor` model was initialized with 100 estimators and a fixed `random_state`.

The model was trained on the training dataset and used to predict the test dataset.

Performance metrics (MSE and R^2) were calculated to evaluate accuracy.

The trained model was saved in `.pkl` format for further use in the application.

```python
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [10, 20, 30, 40, 50, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
print("Hyperparameter grid defined.")
```

```python
rf = RandomForestRegressor(random_state=42)
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_grid, n_iter=50, cv=5, scoring='r2', random_state=42, n_j
random_search.fit(X_train, y_train)
print("Best hyperparameters found by RandomizedSearchCV:")
print(random_search.best_params_)
```

```python
best_rf_model = RandomForestRegressor(**random_search.best_params_, random_state=4
best_rf_model.fit(X_train, y_train)
print("New Random Forest model trained with best hyperparameters.")
```

Activity 1.4: Support Vector Machine (SVM) Model

The `SVR` model from the `sklearn.svm` library was used for regression. After loading the datasets, the SVR model was initialized with an RBF kernel.

It was trained using `.fit()` and predictions were made on the test set.

The model's MSE and R^2 score were computed to assess prediction quality.

The trained model was stored as a `.pkl` file using `joblib.dump()` for future integration into the application.

```
import pandas as pd
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_sc
import joblib

print("Support Vector Machine Model............")
X_train = pd.read_csv('X_train.csv')
X_test = pd.read_csv('X_test.csv')
y_train = pd.read_csv('y_train.csv').values.ravel()
y_test = pd.read_csv('y_test.csv').values.ravel()
svm_model = SVR(kernel='rbf')
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
mse_svm = mean_squared_error(y_test, y_pred_svm)
r2_svm = r2_score(y_test, y_pred_svm)
print("MSE:", mse_svm, "R2:", r2_svm)
joblib.dump(svm_model, 'svm_model.pkl')
print("SVM model saved as svm_model.pkl")
```

Activity 2: Testing the model

Here we have tested with Decision Tree algorithm. You can test with all algorithm. With the help of predict() function.

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

Activity 1.1: Compare the model

For comparing the above four models

```python
import pandas as pd
import joblib
from sklearn.metrics import mean_squared_error, r2_score

print("Comparing the Models..........")
X_test = pd.read_csv('X_test.csv')
y_test = pd.read_csv('y_test.csv').values.ravel()
models = {
    'Linear Regression': joblib.load('linear_regression_model.pkl'),
    'Decision Tree': joblib.load('decision_tree_model.pkl'),
    'Random Forest': joblib.load('random_forest_model.pkl'),
    'SVM': joblib.load('svm_model.pkl')
}
results = {}
for name, model in models.items():
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[name] = {'MSE': mse, 'R2': r2}
    print("-", name, "- MSE:", mse, "R2:", r2)
results_df = pd.DataFrame(results).T
results_df.to_csv('model_comparison.csv')
print("Model comparison saved to model_comparison.csv")
```

```
Comparing the Models..........
- Linear Regression - MSE: 1.723213475860319 R2: 0.9900773400384608
- Decision Tree - MSE: 0.04320327264529003 R2: 0.9997512256086143
- Random Forest - MSE: 0.013091178485523904 R2: 0.9999246179800545
- SVM - MSE: 0.444410243447976 R2: 0.9974409834933785
Model comparison saved to model_comparison.csv
```

```python
import pandas as pd
import joblib
from sklearn.metrics import mean_squared_error, r2_score
from google.colab import files

print(" Starting: Evaluating Performance and Saving the Model")
X_test = pd.read_csv('X_test.csv')
y_test = pd.read_csv('y_test.csv').values.ravel()
# Load all models to find the best
models = {
    'Linear Regression': joblib.load('linear_regression_model.pkl'),
    'Decision Tree': joblib.load('decision_tree_model.pkl'),
    'Random Forest': joblib.load('random_forest_model.pkl'),
    'SVM': joblib.load('svm_model.pkl')
}
results = {name: r2_score(y_test, model.predict(X_test)) for name, model in models.ite
best_model_name = max(results, key=results.get)
best_model = models[best_model_name]
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(" Best Model (", best_model_name, ") Performance - MSE:", mse_best, "R2:", r2_be
joblib.dump(best_model, 'best_model.pkl')
print(" Best model saved as best_model.pkl")
```

```
Starting: Evaluating Performance and Saving the Model
Best Model ( Random Forest ) Performance - MSE: 0.013091178485523904 R2: 0.9999246179
Best model saved as best_model.pkl
```

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance.This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
joblib.dump(best_rf_model, 'best_model2.pkl')
print("Enhanced Random Forest model saved as best_model2.pkl")
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building Html Page:

For this project create HTML file namely

- index.html

and save them in the templates folder. Refer this link for templates.

Activity 2.2: Build Python code:

Import the libraries

```
app.py > home
from flask import Flask, request, render_template
import joblib
import numpy as np
import pandas as pd
import warnings
```

This code imports essential libraries for a Flask web application that serves machine learning predictions. It brings in Flask components for web functionality (`Flask`, `request`, `render_template`), `joblib` for loading trained models, and `numpy`/`pandas` for data processing. The `warnings` import helps manage system alerts. These imports form the foundation for an application that will receive data, process it, run ML predictions, and display results.

—---

```python
app = Flask(__name__)

# Load the model
try:
    model = joblib.load('best_model2.pkl')
    print("Model loaded successfully!")
except FileNotFoundError:
    print("Warning: best_model2.pkl not found. Please ensure the file exists in the project directory.")
    model = None
```

This code initializes a Flask web application and loads a machine learning model. The `Flask(__name__)` line creates the application instance. The `try-except` block attempts to load a pre-trained model (`best_model2.pkl`) using joblib, printing a success message if loaded or a warning if the file is missing. The model is stored in the `model` variable for later use in predictions. This setup is crucial for enabling the app to make ML-based inferences.

—---

```python
@app.route('/')
def home():
    """Home route that renders the main prediction page"""
    return render_template('manual_predict.html')
```

This code defines the root route (`/`) for a Flask web application. When users access the homepage, it renders and displays the `manual_predict.html` template file, which likely contains a form for submitting prediction inputs. The docstring briefly explains the route's purpose as serving the main prediction interface. This is a standard Flask route that connects the web frontend to the backend application.

—---

Prediction Endpoint (`/y_predict`)

This POST route handles motor temperature predictions by:

1. Collecting Input Parameters:
    - Extracts 11 numerical values from form data (voltage components, temperatures, current, speed, torque)
    - Converts them to floats for model compatibility
2. Data Processing:

    ○ Logs received values for debugging
    ○ Structures inputs into a pandas DataFrame with matching feature names
3. Prediction Flow:
    ○ Intended to pass the DataFrame to a pre-loaded ML model (though model inference code is cut off in snippet)
    ○ Follows standard ML deployment practices for web services

Key Features:

- RESTful design (POST method for data submission)
- Input validation through float conversion
- Structured data preparation matching model training format

Note: The incomplete snippet suggests additional code exists for:

- Model inference (`model.predict()`)
- Result formatting/return

```python
@app.route('/y_predict', methods=['POST'])
def y_predict():
    """
    Prediction route that processes form data and returns prediction
    """
    try:
        # Collect form values as float inputs
        u_q = float(request.form['u_q'])
        coolant = float(request.form['coolant'])
        stator_winding = float(request.form['stator_winding'])
        u_d = float(request.form['u_d'])
        stator_tooth = float(request.form['stator_tooth'])
        motor_speed = float(request.form['motor_speed'])
        i_d = float(request.form['i_d'])
        i_q = float(request.form['i_q'])
        stator_yoke = float(request.form['stator_yoke'])
        ambient = float(request.form['ambient'])
        torque = float(request.form['torque'])

        print(f"Received values: u_q={u_q}, coolant={coolant}, stator_winding={stator_winding}, "
              f"u_d={u_d}, stator_tooth={stator_tooth}, motor_speed={motor_speed}, "
              f"i_d={i_d}, i_q={i_q}, stator_yoke={stator_yoke}, ambient={ambient}, torque={torque}")

        input_data = pd.DataFrame({
            'u_q': [u_q],
            'coolant': [coolant],
            'stator_winding': [stator_winding],
            'u_d': [u_d],
            'stator_tooth': [stator_tooth],
            'motor_speed': [motor_speed],
            'i_d': [i_d],
            'i_q': [i_q],
            'stator_yoke': [stator_yoke],
            'ambient': [ambient],
            'torque': [torque]
        })
```

—---

This code launches the Flask development server when the script is executed directly (not imported as a module). Key features:

1. Development Mode:
    - `debug=True` enables auto-reloading and detailed error pages
    - Should be disabled in production
2. Network Accessibility:
    - `host='0.0.0.0'` makes the server accessible on all network interfaces
    - Default `port=5000` serves the application
3. Execution Control:
    - The `__name__ == '__main__'` check ensures this only runs when the file is executed directly

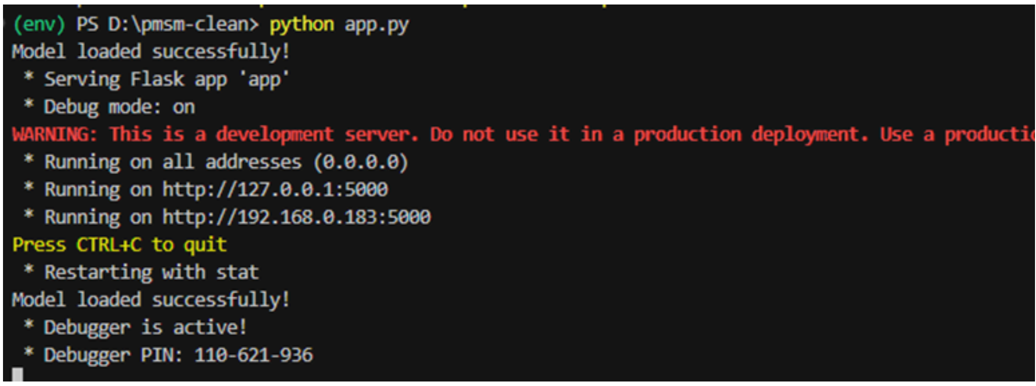
Typical for local testing and development environments.

```python
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

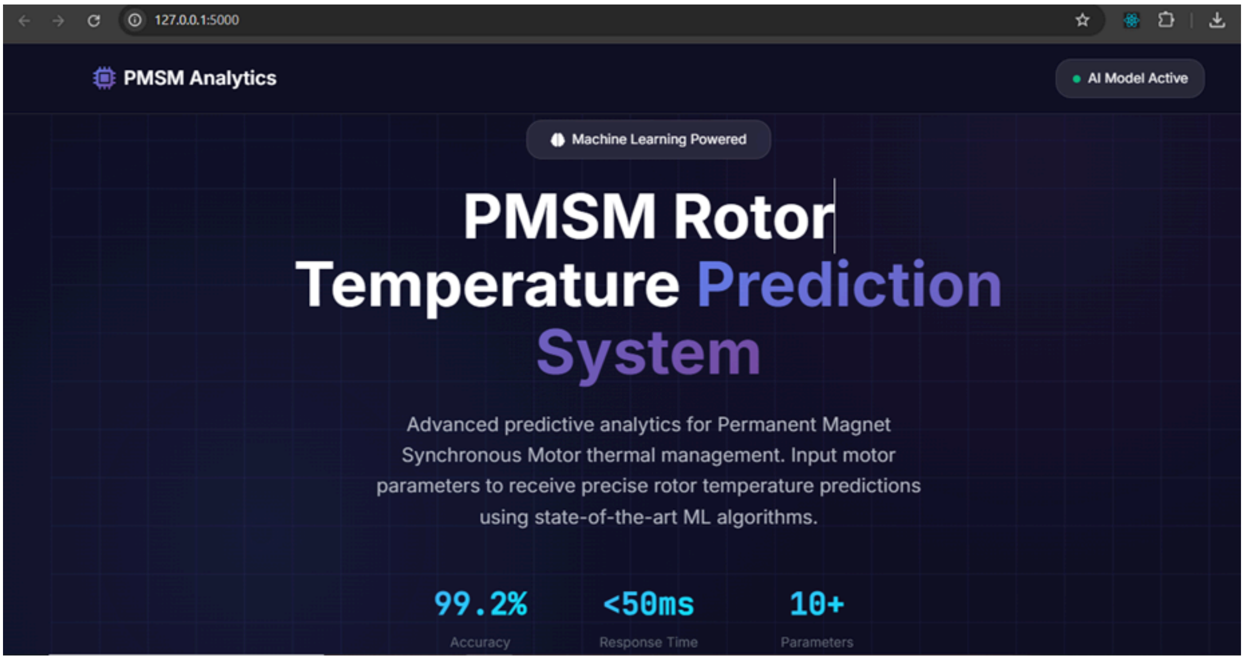
Activity 2.3: Run the web application

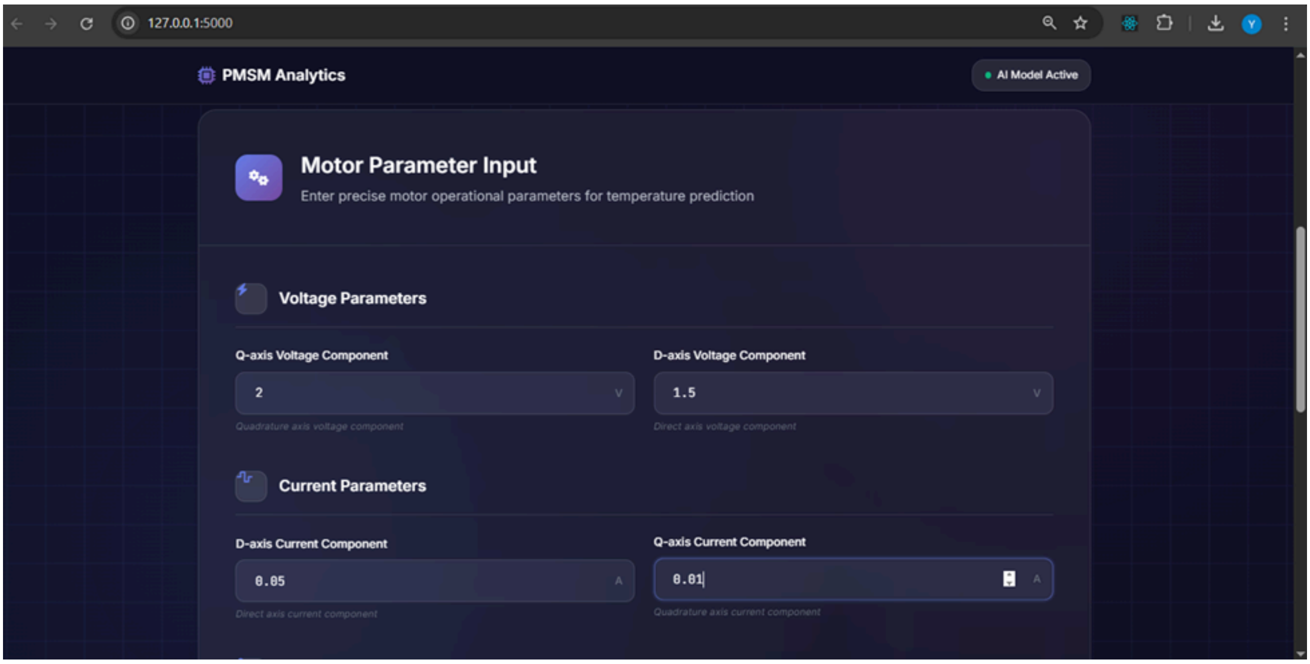
This console output shows:

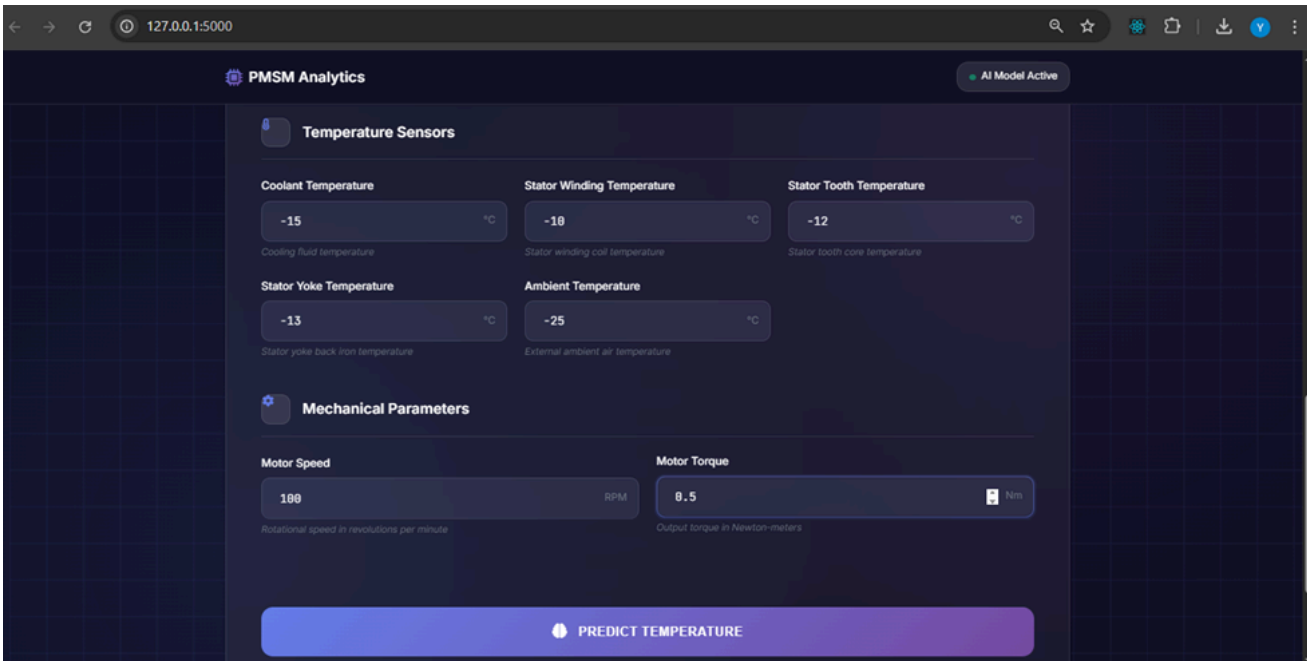
1. Successful Model Loading
    - The ML model (`best_model2.pkl`) loaded twice due to Flask's debug reloader
2. Server Configuration
    - Running in debug mode (auto-reloads code changes)
    - Accessible via:
        - Localhost: `http://127.0.0.1:5000`
        - Network IP: `http://192.168.0.183:5000`
3. Important Warnings
    - Development server not suitable for production
    - Debugger PIN provided for secure console access
4. Execution Context
    - Running in a Python virtual environment (`env`)
    - Launched from `D:\pmsm-clean\app.py`

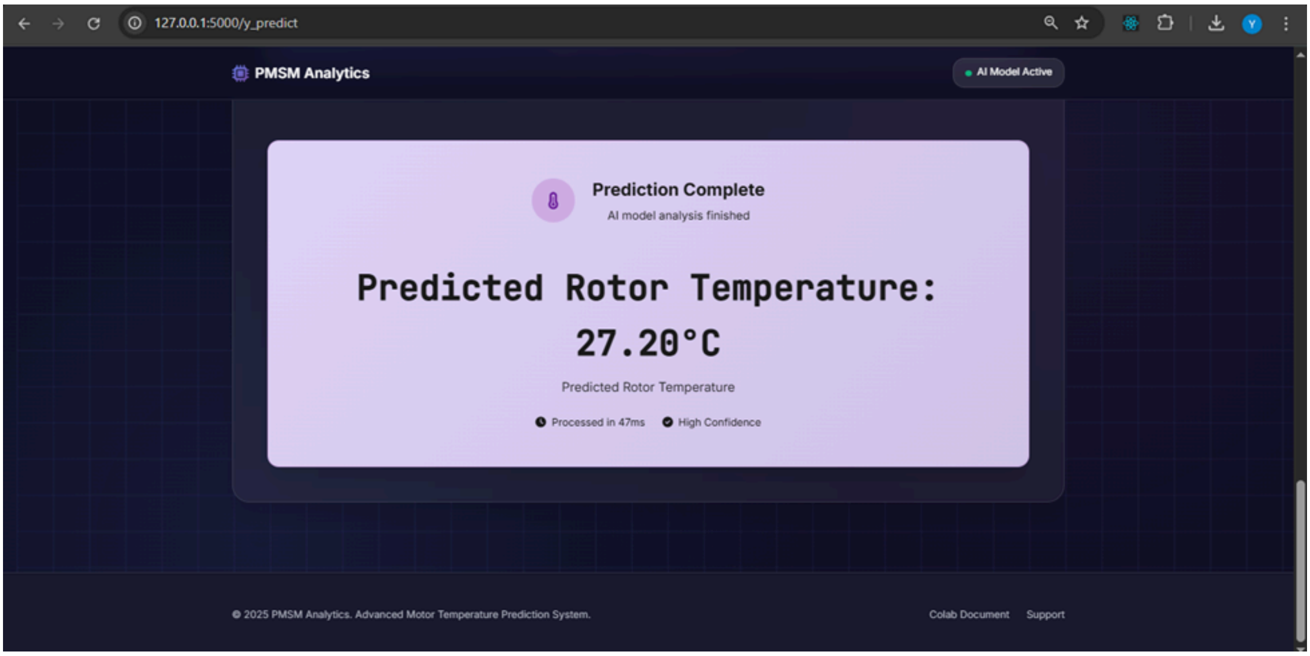

(env) PS D:\pmsm-clean> python app.py
Model loaded successfully!
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a producti
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.183:5000
Press CTRL+C to quit
* Restarting with stat
Model loaded successfully!
* Debugger is active!
* Debugger PIN: 110-621-936

The Output→


127.0.0.1:5000
PMSM Analytics
AI Model Active
Machine Learning Powered
PMSM Rotor Temperature Prediction System
Advanced predictive analytics for Permanent Magnet Synchronous Motor thermal management. Input motor parameters to receive precise rotor temperature predictions using state-of-the-art ML algorithms.
99.2%
Accuracy
<50ms
Response Time
10+
Parameters


127.0.0.1:5000
PMSM Analytics
AI Model Active
Motor Parameter Input
Enter precise motor operational parameters for temperature prediction
Voltage Parameters
Q-axis Voltage Component
2
Quadrature axis voltage component
D-axis Voltage Component
1.5
Direct axis voltage component
Current Parameters
D-axis Current Component
0.05
Direct axis current component
Q-axis Current Component
0.01
Quadrature axis current component





Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure

Create document as per the template provided