

# **Predicting Permanent Magnet Resistance Of Electric Motor Using Machine Learning**

# INTRODUCTION

## 1.1 Overview

The permanent-magnet synchronous machine (PMSM) drive is one of the best choices for a full range of motion control applications. For example, the PMSM is widely used in robotics, machine tools, actuators, and it is being considered in high-power applications such as industrial drives and vehicular propulsion. It is also used for residential/commercial applications. The PMSM is known for having low torque ripple, superior dynamic performance, high efficiency, and high power density.

The task is to design a model with appropriate feature engineering that estimates the target temperature of a rotor.

In this project, we will be using algorithms such as Linear Regression, Decision Tree, Random Forest and SVM. We will train and test the data with these algorithms and select the best model. The best algorithm will be selected and saved in pkl format. We will be doing flask integration and IBM deployment.

## 1.2 Purpose

The purpose of this system is to accurately predict the temperature of a rotor in a PMSM drive, which is an important factor in determining the overall performance and efficiency of the machine. Accurate temperature predictions can help prevent damage to the rotor, as well as improve the control and performance of the PMSM drive in a wide range of applications.

By using machine learning algorithms and feature engineering, we aim to create a highly accurate model that can provide real-time predictions of the rotor temperature. The implementation of this model in a web application using Flask and IBM Cloud deployment makes it easy for users to access and use the model, providing valuable insights and predictions for a wide range of applications.

# 2.LITERATURE SURVEY

## 2.1 Existing Problem

❑ Deep learning models:

1.Convolutional Neural Networks (CNN).

2.Recurrent Neural Networks (RNN).

3.Boltzmann machine.

4.Autoencoders etc.

#### ❑ Classification:

- 1.The K-Nearest Neighbours algorithm
- 2.Decision Tree
- 3.Support Vector Machines
- 4.Naive Bayes

#### ❑ Regression:

- 1.Linear Regression
- 2.Lasso Regression
- 3.Ridge Regression
- 4.Support Vector Regression (SVR)
- 5.Ensemble Regression

#### ❑ Clustering:

- 1.K means
- 2.K means++
- 3.K medoids
- 4.Agglomerative clustering
- 5.DBSCAN

Above are some of the existing approaches or methods to solve this problem of Predicting Permanent Magnet Resistance Of Electric Motor Using Machine Learning

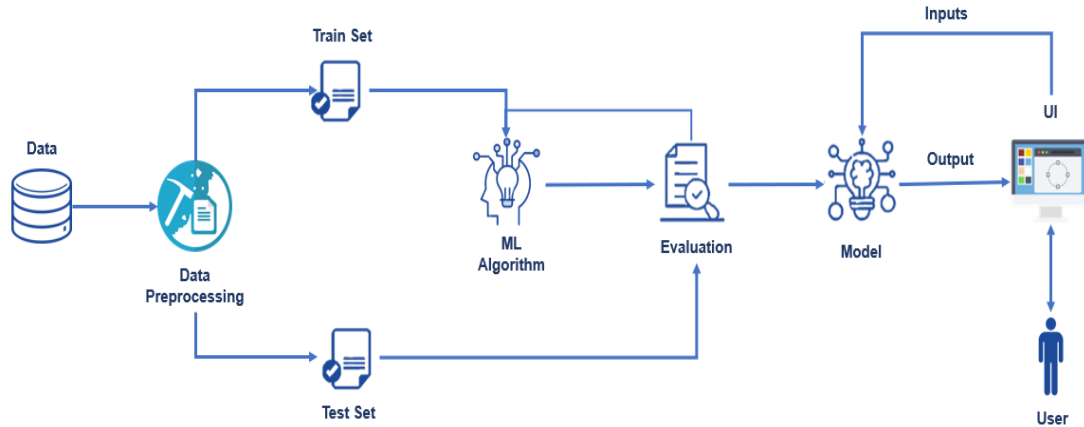
Classification problem is we are applying four Regression algorithms. The best model is saved based on its performance. To evaluate the performance of the model, we use root mean square error and r-square value.

## 2.2 Proposed solution

Proposed system is expected to improve the efficiency, performance and longevity of PMSM drives in various industries, helping to reduce maintenance costs and minimize downtime. Additionally, by having an accurate temperature prediction model, engineers and technicians can make informed decisions about the maintenance and operation of PMSM drives, leading to more reliable and efficient systems

### 3. Theoretical Analysis

#### 3.1 Block Diagram



#### 3.2 Hardware/Software Designing

##### Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It was created by Guido van Rossum , and first released on February 20, 1991. Its high-level built in data structures, combined with dynamic typing and dynamic binding , make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

##### Anaconda Navigator

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, crossplatform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder.

## Jupyter Notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter. Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

## Spyder

Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda. It includes editing, interactive testing, debugging, and introspection features. Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the community. Spyder is extensible with first-party and third party plugins includes support for interactive tools for data inspection and embeds Python specific code. Spyder is also pre-installed in Anaconda Navigator, which is included in Anaconda.

## Flask

Web framework used for building. It is a web application framework written in python which will be running in local browser with a user interface. In this application, whenever the user interacts with UI and selects emoji, it will suggest the best and top movies of that genre to the user.

## Hardware Requirements:

Operating system: window 7 and above with 64bit

Processor Type -Intel Core i3-3220 RAM: 4Gb and above

Hard disk: min 100GB

## 4.Experimental investigation

The data set comprises several sensor data collected from a permanent magnet synchronous motor (PMSM) deployed on a test bench. The PMSM represents a german OEM's prototype model. Test bench measurements were collected by the [LEA department](#) at Paderborn University.

All recordings are sampled at 2 Hz. The data set consists of multiple measurement sessions, which can be distinguished from each other by column "profile\_id". A measurement session can be between one and six hours long.

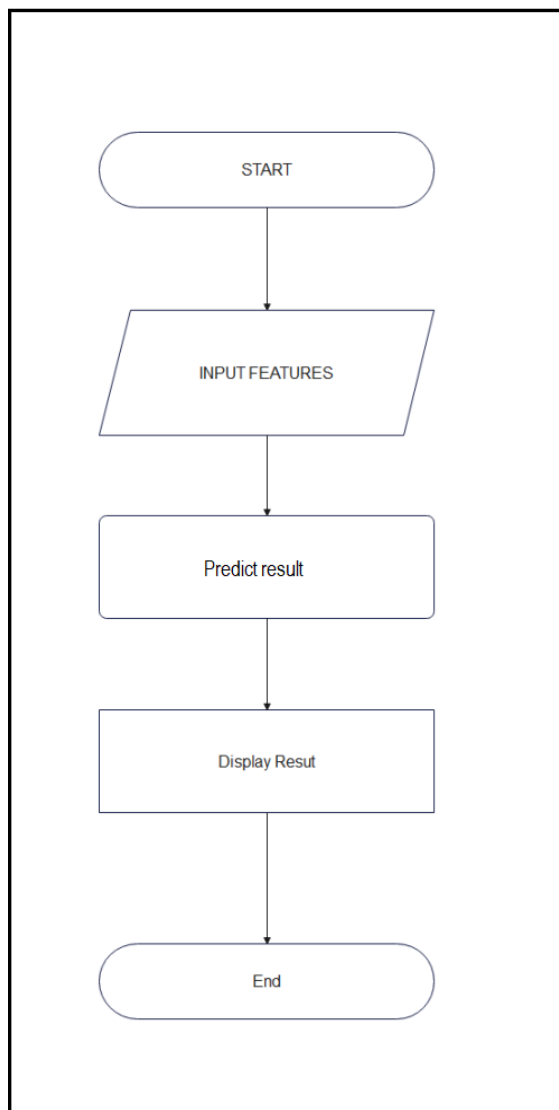
The motor is excited by hand-designed driving cycles denoting a reference motor speed and a reference torque.

Currents in d/q-coordinates (columns "i\_d" and i\_q") and voltages in d/q-coordinates (columns "u\_d" and "u\_q") are a result of a standard control strategy trying to follow the reference speed and torque.

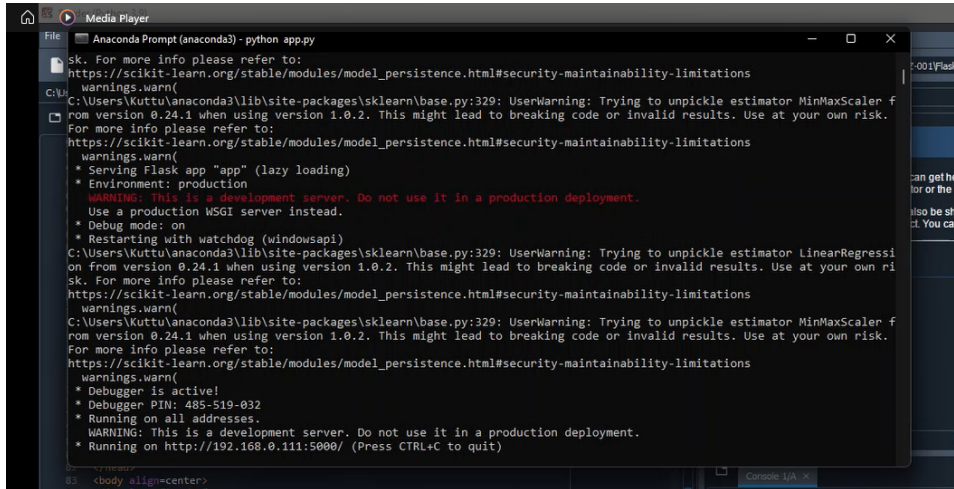
Columns "motor\_speed" and "torque" are the resulting quantities achieved by that strategy, derived from set currents and voltages.

Most driving cycles denote random walks in the speed-torque-plane in order to imitate real world driving cycles to a more accurate degree than constant excitations and ramp-ups and -downs would.

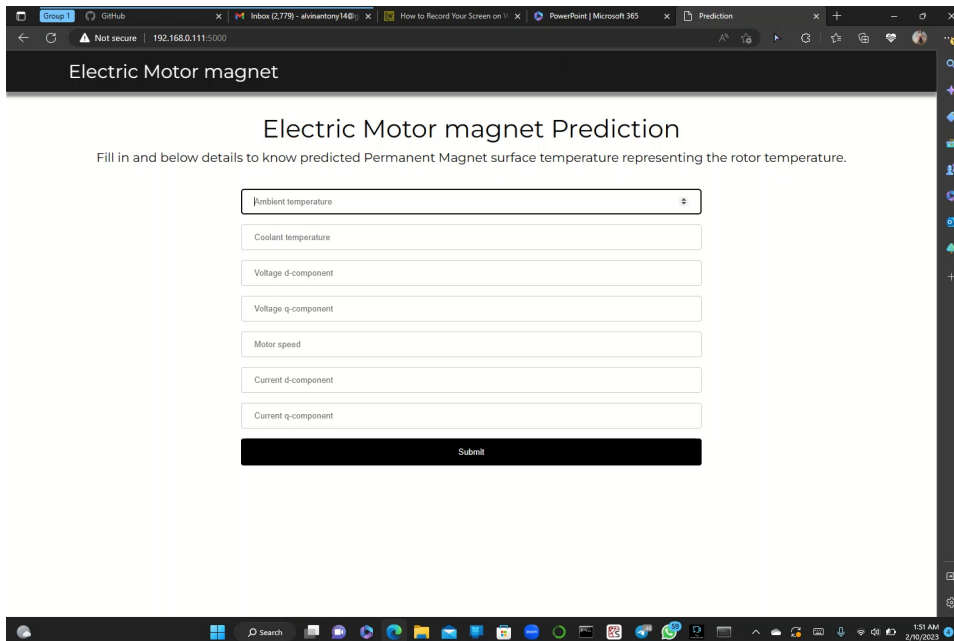
## 5.Flowchart



## 6.RESULT



```
Media Player
Anaconda Prompt (anaconda3) - python app.py
sk. For more info please refer to:
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
warnings.warn(
C:\Users\Kuttu\anaconda3\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator MinMaxScaler f
rom version 0.24.1 when using version 1.0.2. This might lead to breaking code or invalid results. Use at your own risk.
For more info please refer to:
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
warnings.warn(
* 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)
C:\Users\Kuttu\anaconda3\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator LinearRegressi
on from version 0.24.1 when using version 1.0.2. This might lead to breaking code or invalid results. Use at your own ri
sk. For more info please refer to:
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
warnings.warn(
C:\Users\Kuttu\anaconda3\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator MinMaxScaler f
rom version 0.24.1 when using version 1.0.2. This might lead to breaking code or invalid results. Use at your own risk.
For more info please refer to:
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
warnings.warn(
* Debugger is active!
* Debugger PIN: 485-519-032
* Running on all addresses.
WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.0.111:5000/ (Press CTRL+C to quit)
```



Electric Motor magnet

### Electric Motor magnet Prediction

Fill in and below details to know predicted Permanent Magnet surface temperature representing the rotor temperature.

Electric Motor magnet

## Electric Motor magnet Prediction

Fill in and below details to know predicted Permanent Magnet surface temperature representing the rotor temperature.

-1
2
1
1
-1
-2
-3

Submit

Electric Motor Temperature

## Electric Motor Temperature Prediction

Fill in and below details to know predicted Permanent Magnet surface temperature representing the rotor temperature.

(Permanent Magnet surface temperature: -, -1.6761227252902047)

Ambient temperature
Coolant temperature
Voltage d-component
Voltage q-component
Motor speed
Current d-component
Current q-component

Submit



## **7.Advantage and Disadvantage**

### **Advantages**

- **High Efficiency:** PMSM drives have a high power density and are highly efficient, which helps to reduce energy losses and improve the overall efficiency of the system.
- **Low Torque Ripple:** PMSM drives have a low torque ripple, which results in smoother and more accurate motion control.
- **Superior Dynamic Performance:** PMSM drives have a superior dynamic performance, providing fast and accurate control of the motion of machinery and equipment.
- **High Power Density:** PMSM drives have a high power density, which makes them suitable for high-power applications, such as industrial drives and vehicular propulsion.
- **Wide Range of Applications:** PMSM drives have a wide range of applications, including robotics, machine tools, actuators, and renewable energy systems.

### **Disadvantages**

- **High Cost:** PMSM drives can be more expensive than other types of drives, due to the cost of the permanent magnets and other components.
- **Complex Control Systems:** PMSM drives require complex control systems, which can make them more difficult to install and maintain.
- **Vulnerability to Overheating:** PMSM drives can be vulnerable to overheating, especially in high-power applications. Overheating can cause damage to the drive and reduce its lifespan.

- Limited Power Density: PMSM drives have a limited power density, which may make them unsuitable for applications with extremely high power requirements.
- Overall, the PMSM drive is a highly efficient and effective solution for a wide range of motion control applications. However, it is important to carefully consider the advantages and disadvantages of PMSM drives when deciding on the best drive solution for your specific application.

## **8.Application**

. Some real-world applications of the PMSM drives and their temperature monitoring system could include:

- Industrial Drives: In industrial environments, PMSM drives are commonly used to control the motion of machinery and conveyor systems. By monitoring the temperature of the rotor, manufacturers can ensure that the drive operates within safe temperature limits, which helps to prevent damage and extend the life of the equipment.
- Robotics: PMSM drives are widely used in the field of robotics, where high precision and low torque ripple are important considerations. By monitoring the temperature of the rotor, robots can be programmed to adjust their movements to prevent overheating and ensure reliable operation.
- Electric Vehicles: PMSM drives are becoming increasingly popular in electric vehicles, where high efficiency and power density are important considerations. By monitoring the temperature of the rotor, electric vehicles can be programmed to adjust their power usage to prevent overheating and ensure reliable operation.
- Renewable Energy: PMSM drives are also used in renewable energy systems, such as wind turbines and solar inverters, where efficiency and reliability are important considerations. By monitoring the temperature of the rotor, these systems can be programmed to adjust their output to prevent overheating and ensure reliable operation.

## 9.Conclusion

In conclusion, the design of a model to estimate the target temperature of a rotor in a permanent-magnet synchronous machine (PMSM) drive is a critical task that requires careful consideration of the various algorithms and feature engineering techniques available. In this project, you have chosen to use several algorithms, including Linear Regression, Decision Tree, Random Forest, and SVM, to train and test the data. The best algorithm will then be selected, saved in pkl format, and integrated into a flask application for deployment on IBM.

The choice of algorithms and feature engineering techniques will play a key role in determining the accuracy and reliability of the model, so it is important to carefully consider the strengths and weaknesses of each approach. By utilizing the best techniques available and integrating the model into a flexible and scalable deployment platform, you can ensure that your solution provides accurate and reliable estimates of the target temperature of a rotor in a PMSM drive.

## 10. Future scope

The future scope of this project could involve further improvement and optimization of the selected model to achieve higher accuracy and precision in estimating the target temperature of the rotor. Additionally, more advanced machine learning algorithms could be considered, such as deep neural networks, to see if they outperform the algorithms currently being used.

Another avenue for future work could be incorporating sensor data from other parts of the PMSM system, such as the stator, to get a more complete picture of the system's temperature and performance. This could lead to the development of even more accurate temperature models.

Finally, the integration of the model into real-world PMSM systems through the use of IoT devices and cloud computing could also be explored. This would allow for real-time monitoring and control of the temperature of PMSM systems in various applications.

## Appendix:

### Source code:

```
#Import libraries  
  
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')
```

```
#Load the dataset
```

```
df = pd.read_csv('pmsm_temperature_data.csv')
```

```
df.head()
```

```
df.info()
```

According to the dataset test data are profile id 65&72

```
df_test = df[(df['profile_id'] == 65) | (df['profile_id'] == 72)]
```

```
df = df[(df['profile_id'] != 65) & (df['profile_id'] != 72)]
```

```
df_test.info()
```

```
df.info()
```

```
df.describe()
```

```
#Check for null values
```

```
df.isnull().sum()
```

There are no missing values in the dataset.

```
plt.figure(figsize=(15,6))
```

```
df['profile_id'].value_counts().sort_values().plot(kind = 'bar')
```

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

```
df.columns
```

```
#Plotting Distribution and Boxplot for all the features to check for skewness
```

```
for i in df.columns:
```

```
    sns.distplot(df[i],color='g')
```

```
    sns.boxplot(df[i],color = 'y')
```

```
    plt.vlines(df[i].mean(),ymin = -1,ymax = 1,color = 'r')#drawing the mean line
```

```
plt.show()
```

As we can see from the 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.

Checking skewness and kurtosis numerically

Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point.

Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. That is, data sets with high kurtosis tend to have heavy tails, or outliers. Data sets with low kurtosis tend to have light tails, or lack of outliers.

#for skewness (-1,1) and (-2,2) for kurtosis is an acceptable range for being normally distributed.

```
import scipy.stats as stats
```

```
for i in df.columns:
```

```
    print(i,':\nSkew : ',df[i].skew(),': \nKurtosis : ',df[i].kurt())
```

```
    print()
```

As it is not highly skewed data and looking at the values of the dataset it seems there already has been some normalization done.

#Correlation is a statistic that measures the degree to which two variables move in relation to each other.

#It ranges from (-1,1),Nearer to -1,1 referes to high correlation and 0 means less correlation

```
plt.figure(figsize=(14,7))
```

```
sns.heatmap(df.corr(),annot=True);
```

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 windings.

For a random measurement, we can try to compare the temperatures of the 3 stator components.

```
plt.figure(figsize=(20,5))
```

```
df[df['profile_id'] == 20]['stator_yoke'].plot(label = 'stator yoke')
df[df['profile_id'] == 20]['stator_tooth'].plot(label = 'stator tooth')
df[df['profile_id'] == 20]['stator_winding'].plot(label = 'stator winding')
plt.legend();
```

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 in series of time.

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 temperature of stator winding but as we progress in time, the stator yoke temperature goes above the temperature of stator winding.

As profile\_id is an id for each measurement session, we can remove it from any further analysis and model building.

```
df.drop('profile_id',axis = 1,inplace=True)
df_test.drop('profile_id',axis = 1,inplace=True)
```

### ### Statistical Analysis of Variables

We'll see which particular variables contribute to the rotor temperature individually by checking their statistical significance.

#### #### Ambient Temperature

```
sns.distplot(df['ambient'])
```

Shapiro tests the null hypothesis that the data was drawn from a normal distribution.

```
from scipy.stats import shapiro
```

```
shapiro(df['ambient'])
```

```
shapiro(df['pm'])
```

H0 : variance\_ambient = variance\_pm #Null Hypothesis

H1 : variance\_ambient != variance\_pm # alternative hypothesis

```
from scipy.stats
```

```
import bartlett
```

bartlett(df['ambient'],df['pm'])pvalue is less than 0.05. So we reject the null hypothesis and can say that variance for ambient temperature is not equal to the variance of rotor temperature.

#### Coolant Temperature

```
sns.distplot(df['coolant'])
```

```
from scipy.stats import shapiro
```

```
shapiro(df['coolant'])
```

```
shapiro(df['pm'])
```

H0 : variance\_coolant = variance\_pm #Null Hypothesis

H1 : variance\_coolant != variance\_pm # alternative hypothesis

```
from scipy.stats import bartlett
```

```
bartlett(df['coolant'],df['pm'])
```

pvalue is less than 0.05. So we reject the null hypothesis and can say that variance for coolant temperature is not equal to the variance of rotor temperature.

#### Voltage d-component

```
sns.distplot(df['u_d'])
```

```
from scipy.stats import shapiro
```

```
shapiro(df['u_d'])
```

```
shapiro(df['pm'])
```

H0 : variance\_u\_d = variance\_pm

H1 : variance\_u\_d != variance\_pm

```
from scipy.stats import bartlett
```

```
bartlett(df['u_d'],df['pm'])
```

pvalue is less than 0.05. So we reject the null hypothesis and can say that variance for voltage d-component is not equal to the variance of rotor temperature

#### Voltage q-component

```
sns.distplot(df['u_q'])
```

```
from scipy.stats import shapiro
```

```
shapiro(df['u_q'])
```

```
shapiro(df['pm'])
```

```
H0 : variance_u_q = variance_pm
```

```
H1 : variance_u_q != variance_pm
```

```
from scipy.stats import bartlett
```

```
bartlett(df['u_q'],df['pm'])
```

pvalue is less than 0.05. So we reject the null hypothesis and can say that variance for voltage q-component is not equal to the variance of rotor temperature.

```
#### Motor speed
```

```
sns.distplot(df['motor_speed'])
```

```
from scipy.stats import shapiro
```

```
shapiro(df['motor_speed'])
```

```
shapiro(df['pm'])
```

```
H0 : variance_motor_speed = variance_pm
```

```
H1 : variance_motor_speed != variance_pm
```

```
from scipy.stats import bartlett
```

```
bartlett(df['motor_speed'],df['pm'])
```

pvalue is less than 0.05. So we reject the null hypothesis and can say that variance of motor speed is not equal to the variance of rotor temperature.

```
#### Current d-component
```

```
sns.distplot(df['i_d'])
```

```
from scipy.stats import shapiro
```

```
shapiro(df['i_d'])
```

```
shapiro(df['pm'])
```

```
H0 : variance_i_d = variance_pm
```

```
H1 : variance_i_d != variance_pm
```

```
from scipy.stats import bartlett
```

```
bartlett(df['i_d'],df['pm'])
```



pvalue is higher than 0.05. So we fail to reject the null hypothesis and can say that we do not have enough evidence to reject the null hypothesis. So we do not have enough evidence to prove that variance of d component of current is not equal to the variance of motor temperature.

#### Current q-component

```
sns.distplot(df['i_q'])
```

```
from scipy.stats import shapiro
```

```
shapiro(df['i_q'])
```

```
shapiro(df['pm'])
```

H0 : variance\_i\_q = variance\_pm

H1 : variance\_i\_q != variance\_pm

```
from scipy.stats import bartlett
```

```
bartlett(df['i_q'],df['pm'])
```

pvalue is higher than 0.05. So we fail to reject the null hypothesis and can say that we do not have enough evidence to reject the null hypothesis. So we do not have enough evidence to prove that variance of q component of current is not equal to the variance of motor temperature.

#### Shuffling the data

```
df = df.sample(frac=1,random_state=3)
```

```
df.head()
```

The data description did not provide us with any information on the units of measure. So its difficult to interpret the values measured

### Exploratory Data Analysis

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

```
sns.scatterplot(df['ambient'],df['pm'],ax=axes[0][0])
```

```
sns.scatterplot(df['coolant'],df['pm'],ax=axes[0][1])
```

```
sns.scatterplot(df['motor_speed'],df['pm'],ax=axes[0][2])
```

```
sns.scatterplot(df['i_d'],df['pm'],ax=axes[0][3])
```

```
sns.scatterplot(df['u_q'],df['pm'],ax=axes[1][0])
```

```
sns.scatterplot(df['u_d'],df['pm'],ax=axes[1][1])
```

```
sns.scatterplot(df['i_q'],df['pm'],ax=axes[1][2])
```

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 modelling.

```
from sklearn.preprocessing import MinMaxScaler
```

```
X = df.drop(['pm','stator_yoke','stator_tooth','stator_winding','torque'],axis = 1)
```

```
X_df_test = df_test.drop(['pm','stator_yoke','stator_tooth','stator_winding','torque'],axis = 1)
```

```
X
```

```
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)
```

```
X.shape
```

```
y.shape
```

```
import joblib
```

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

```
X=X.values
```

```
y=y.values
```

```
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)
```

```
from sklearn.linear_model import LinearRegression
```

```
lr=LinearRegression()
```

```
lr.fit(X_train,y_train)
y_pred=lr.predict(X_test)
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred)
mean_squared_error(y_test, y_pred, squared=False)
X_train.shape
joblib.dump(lr,"model.save")
X_test[0]
X_train[0]
p=[[-0.75214297, -1.1184461, 0.3279352, -1.2978575, -1.2224282, 1.0295724,
-0.24586003]]
lr.predict(mm.transform(p))
```