



A PROJECT REPORT ON
“ROTOR TEMPERATURE DETECTION OF A SYNCHRONOUS MOTOR”

submitted in partial fulfillment of the requirements for the degree of

B. Tech
In
Electronics and Electrical Engineering

By

Satyam Kumar(1707141)

under the guidance of

Prof. Anshuman Pattanaik

School of Electronics Engineering
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
(Deemed to be University)
BHUBANESWAR



This is to certify that the project report entitled “ **ROTOR TEMPERATURE DETECTION OF A SYNCHRONOUS MOTOR** ” submitted by

Satyam Kumar(1707141)

in partial fulfillment of the requirements for the award of the **Degree of Bachelor of Technology in Electronics and Electrical Engineering** is a bonafide record of the work carried out under my(our) guidance and supervision at School of Electronics Engineering, KIIT (Deemed to be University).

Signature of Supervisor 1

Prof. Anshuman Pattanaik

School of Electronics Engineering

KIIT (Deemed to be University)

The Project was evaluated by us on _____

EXAMINER 1

EXAMINER 3

EXAMINER 2

EXAMINER 4

ACKNOWLEDGEMENTS

We feel immense pleasure and feel privileged in expressing our deepest and most sincere gratitude to our supervisor **Prof. Anshuman Pattanaik**, for his excellent guidance throughout our project work. His kindness, dedication, hard work and attention to detail have been a great inspiration to us. Our heartfelt thanks to you sir for the unlimited support and patience shown to us. We would particularly like to thank him for all his help in patiently and carefully correcting all our manuscripts.

We are also very thankful to **Prof A. Basak and Prof. S.K. Behra** B.tech project coordinator (EEE), Associate Dean Professor **Dr. Amlan Datta** and **Professor Dr. Suprava Patnaik**, Dean (School Of Electronics) for their support and suggestions during our course of the project work in the final year of our undergraduate course.

Satyam Kumar(1707141)

ABSTRACT

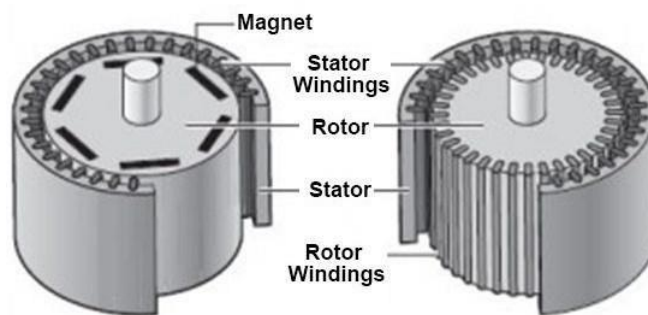
The main purpose of the data set's recording is to be able to model the rotor temperatures of a PMSM (Permanent Magnet Synchronous Motor) in real-time. Due to the intricate structure of an electric traction drive, direct measurement with thermal sensors is not possible for rotor temperatures, and even in case of the stator temperatures, sensor outage or even just deterioration can't be administered properly without redundant modelling. Anything related to the rotor in a motor is difficult to measure as it is the rotating part. Placing any sensors in the rotor area for the measurement would neither be possible nor feasible as it would result in increase in cost and also increase the weight of the motor. Measurement of quantities like temperature, torque of the rotor is important in order to design control systems to effectively control the motor. In addition, precise thermal modelling gets more and more important with the rising relevance of functional safety. We will be designing a model with appropriate feature engineering, that estimates the rotor temperature in a causal manner, In order to maintain real-time capability. The temperature estimation in production will be deployed on best-cost hardware on traction drives in an automotive environment, where lean computation and lightweight implementation is key. After creating and selecting the best regression model we will be deploying that model in a front end Application Programmable Interface created through Flask in which when we will input the independent features it will take that input as an array through numPy and the model will predict us a suitable output for it.

CONTENTS

● Introduction	6
● Literature Survey	8
● Software Requirements Specification	10
● System Design	14
● Project Planning	16
● Implementation	23
● Conclusion	32
● Reference	33

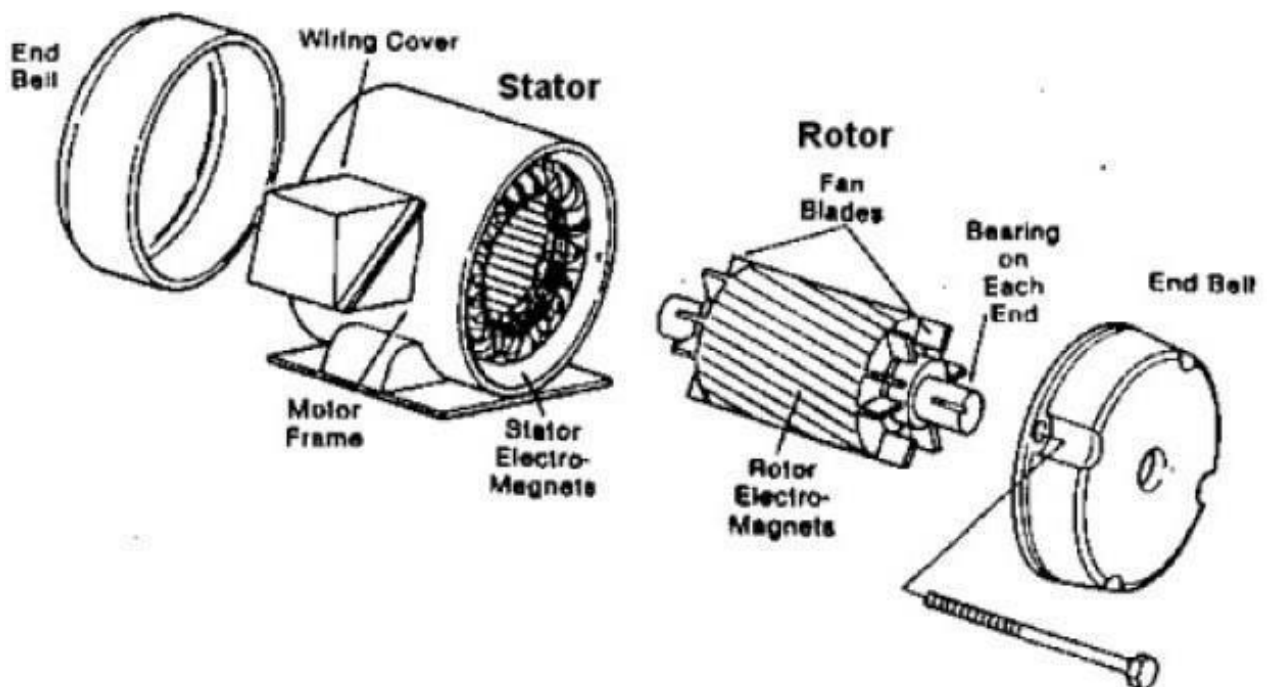
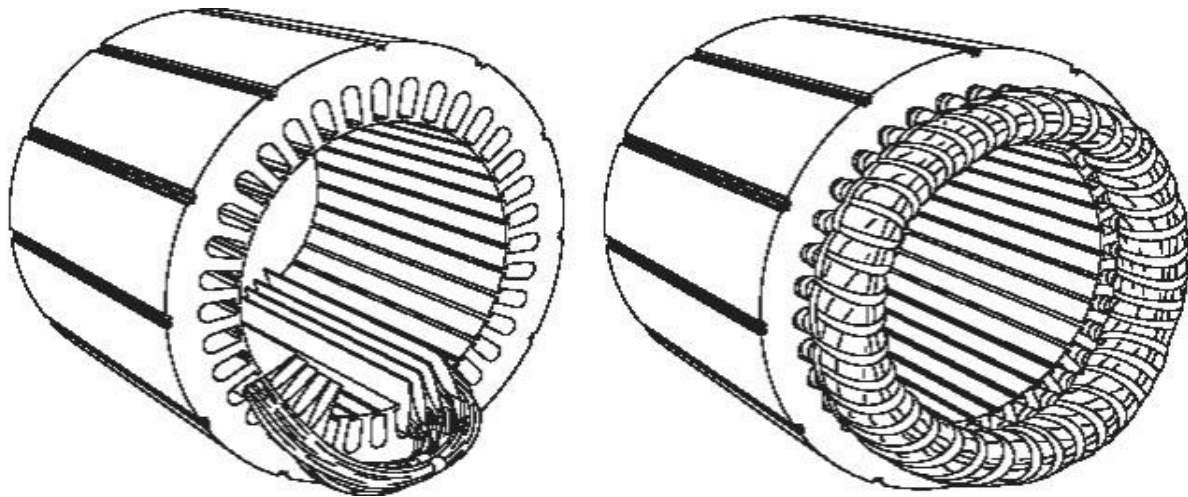
Chapter -1

INTRODUCTION



A **synchronous electric motor** is an AC motor in which, at steady state the rotation of the shaft is synchronized with the frequency of the supply current; the rotation period is exactly equal to an integral number of AC cycles. Synchronous motors contain multiphase AC electromagnets on the stator of the motor that create a magnetic field which rotates in time with the oscillations of the line current. The rotor with permanent magnets or electromagnets turns in step with the stator field at the same rate and as a result, provides the second synchronized rotating magnet field of any AC motor. A synchronous motor is termed doubly fed if it is supplied with independently excited multiphase AC electromagnets on both the rotor and stator. The synchronous motor and induction motor are the most widely used types of AC motor. The difference between the two types is that the synchronous motor rotates at a rate locked to the line frequency since it does not rely on current induction to produce the rotor's magnetic field. By contrast, the induction motor requires slip: the rotor must rotate slightly slower than the AC alternations in order to induce current in the rotor winding. Small synchronous motors are used in timing applications such as in synchronous clock, timers in appliances, tape recorders and precision servomechanism in which the motor must operate at a precise speed; speed accuracy is that of the power line frequency, which is carefully controlled in large interconnected grid systems. A **permanent-magnet synchronous motor (PMSM)** uses permanent magnets embedded in the steel rotor to create a constant magnetic field. The stator carries windings connected to an AC supply to produce a rotating magnetic field (as in an asynchronous motor). At synchronous speed the rotor poles lock to the rotating magnetic field. Permanent magnet synchronous motors are similar to brushless DC motor. Neodymium magnets are the most commonly used magnets in these motors. Most PMSMs require a Variable frequency drive to start. However, some incorporate a squirrel cage in the rotor for starting — these are known as line-start or self-starting PMSMs. These are typically used as higher-efficiency replacements for induction motors (owing to the lack of slip), but need to be specified carefully for the application to ensure that synchronous speed is reached and that the system can withstand the torque ripple during starting.

A Better View of a Synchronous Motor-



Chapter -2

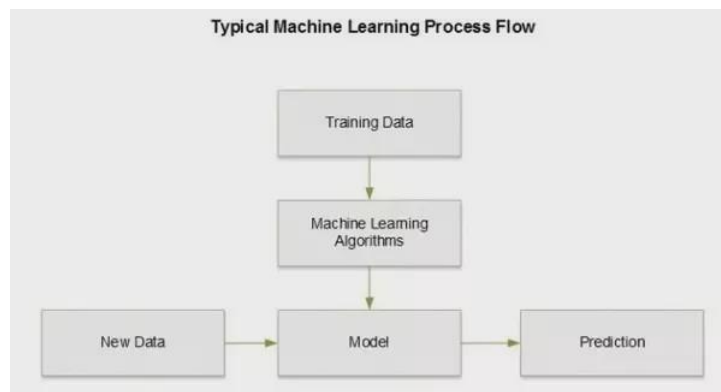
Literature Survey

What is Machine Learning ?

Machine Learning is a new trending field these days and is an application of artificial intelligence. It uses certain statistical algorithms to make computers work in a certain way without being explicitly programmed. The algorithms receive an input value and predict an output for this by the use of certain statistical methods. The main aim of machine learning is to create intelligent machines which can think and work like human beings. Among the different types of ML tasks, a crucial distinction is drawn between supervised and unsupervised learning ;Supervised machine learning:- The program is “trained” on a pre-defined set of “training examples”, which then facilitate its ability to reach an accurate conclusion when given new data. Unsupervised machine learning:- The program is given a bunch of data and must find patterns and relationships therein.

Why Machine Learning ?

Machine Learning algorithm is trained using a training data set to create a model. When new input data is introduced to the ML algorithm, it makes a prediction on the basis of the model. The prediction is evaluated for accuracy and if the accuracy is acceptable, the ML algorithm is deployed. If the accuracy is not acceptable, the ML algorithm is trained again and again with an augmented training data set.



What's required to create good Machine Learning Systems ?

Data — Input data is required for predicting the output.

Algorithms — Machine Learning is dependent on certain statistical algorithms to determine patterns.

Automation — It is the ability to make systems operate automatically.

Iteration — The complete process is an iterative i.e. repetition of the process.

Scalability — The capacity of the machine can be increased or decreased in size and scale.

Modeling — The models are created according to the demand by the process of modeling.

Predicting rotor temperature of the rotor of a Permanent Magnet Synchronous Motor (PMSM) given other sensor measurements during operation. by creating various Regressor models and selecting the best one depending on the Accuracy and Mean Square Error of the model and also implementing the best selected regressor model in a flask , that is by creating an Application Programmable Interface which will eventually show the rotor temperature by inputting the independent variables simultaneously.

Predicting techniques used are Linear Regression, Random Forest and Polynomial Regression. Linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is known as simple linear regression. For more than one explanatory variable, the process is known as multiple linear regression. Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Decision Trees are Simple to understand and to interpret. It can be visualized and requires little data preparation.

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The Random Forest is one of the most effective machine learning models for predictive analytic, making it an industrial workhorse for machine learning. All machine learning algorithms use some input data to create outputs. This input data comprise features, which are usually in the form of structured columns. Algorithms require features with some specific characteristic to work properly. Here, the need for feature engineering arises.

Chapter- 3

Software Requirement Specification

- Reading the data from the csv file, The dataset in which we'll be working our model on.

```
In [2]: # reading the data from the csv file
```

```
df1=pd.read_csv('pmsm_temperature_data.csv')
df=df1.copy()
```

```
In [3]: # top10 rows of the dataset
```

```
df.head(10)
```

Out[3]:

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator_winding	profile_id
0	-0.752143	-1.118446	0.327935	-1.297858	-1.222428	-0.250182	1.029572	-0.245860	-2.522071	-1.831422	-2.066143	-2.018033	4
1	-0.771263	-1.117021	0.329665	-1.297686	-1.222429	-0.249133	1.029509	-0.245832	-2.522418	-1.830969	-2.064859	-2.017631	4
2	-0.782892	-1.116681	0.332771	-1.301822	-1.222428	-0.249431	1.029448	-0.245818	-2.522673	-1.830400	-2.064073	-2.017343	4
3	-0.780935	-1.116764	0.333700	-1.301852	-1.222430	-0.248636	1.032845	-0.246955	-2.521639	-1.830333	-2.063137	-2.017632	4
4	-0.774043	-1.116775	0.335206	-1.303118	-1.222429	-0.248701	1.031807	-0.246610	-2.521900	-1.830498	-2.062795	-2.018145	4
5	-0.762936	-1.116955	0.334901	-1.303017	-1.222429	-0.248197	1.031031	-0.246341	-2.522203	-1.831931	-2.062549	-2.017884	4
6	-0.749228	-1.116170	0.335014	-1.302082	-1.222430	-0.247914	1.030493	-0.246162	-2.522538	-1.833012	-2.062115	-2.017243	4
7	-0.738450	-1.113986	0.336256	-1.305155	-1.222432	-0.248321	1.030107	-0.246035	-2.522844	-1.832182	-2.061953	-2.017213	4
8	-0.730910	-1.111828	0.334905	-1.303790	-1.222432	-0.247785	1.029851	-0.245981	-2.522808	-1.831576	-2.062443	-2.017739	4
9	-0.727130	-1.109486	0.335988	-1.305633	-1.222431	-0.248294	1.029636	-0.245888	-2.522677	-1.831438	-2.062317	-2.018180	4

- Total No. of rows and columns , and checking whether any null values are present or not,

```
In [4]: df.shape
```

Out[4]: (998070, 13)

```
In [5]: df.isnull().values.any()
```

Out[5]: False

A comprehensive csv files containing all measurement sessions and features. As one can see there are total of 998070 rows which represents the readings recorded for 140hrs. Sample rate is 2 Hz (One row per 0.5 seconds). Distinctive sessions are identified with "profile_id".

The dataset provided to us has already been feature scaled, ie. Converted into a simpler format in order to make the regression model more accurate. In case one wants to find the original data, the following process may come handfull,

$$\text{scaled_array} = (\text{original_array} - \text{mean_of_array}) / \text{std_of_array}$$

In Sklearn, each array column appears to be scaled in this way. To find the original data, we can simply rearrange the above, or alternatively just calculate the standard deviation and mean of each column in the unscaled data. We can then use this to transform the scaled data back to the original data at any time.

- Exploring the different variables

A brief description of data attributes:

- **Ambient - Ambient temperature**

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 998070 entries, 0 to 998069
Data columns (total 13 columns):
#   Column              Non-Null Count  Dtype
---  -
0   ambient              998070 non-null float64
1   coolant              998070 non-null float64
2   u_d                  998070 non-null float64
3   u_q                  998070 non-null float64
4   motor_speed          998070 non-null float64
5   torque               998070 non-null float64
6   i_d                  998070 non-null float64
7   i_q                  998070 non-null float64
8   pm                   998070 non-null float64
9   stator_yoke          998070 non-null float64
10  stator_tooth          998070 non-null float64
11  stator_winding        998070 non-null float64
12  profile_id            998070 non-null int64
dtypes: float64(12), int64(1)
memory usage: 99.0 MB
```

- **Coolant - Coolant temperature**
- **u_d - Voltage d-component (Active component)**
- **u_q - Voltage q-component (Reactive component)**
- **Motor speed - Speed of the motor**
- **Torque - Torque induced by current**
- **i_d - Current d-component (Active component)**
- **i_q - Current q-component (Reactive component)**
- **pm - Permanent Magnet Surface temperature(ROTOR TEMP)**
- **Stator_yoke - Stator yoke temperature**
- **Stator_tooth - Stator tooth temperature**
- **Stator_winding - Stator Stator_winding temperature**
- **Profile_id - Measurement Session id**

Describing the variables, ie. Range and how the variables are divided.

```
In [7]: df.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
ambient	998070.0	-0.003905	0.993127	-8.573954	-0.599385	0.266157	0.686675	2.967117
coolant	998070.0	0.004723	1.002423	-1.429349	-1.037925	-0.177187	0.650709	2.649032
u_d	998070.0	0.004780	0.997878	-1.655373	-0.826359	0.267542	0.358491	2.274734
u_q	998070.0	-0.005690	1.002330	-1.861463	-0.927390	-0.099818	0.852625	1.793498
motor_speed	998070.0	-0.006336	1.001229	-1.371529	-0.951892	-0.140246	0.853584	2.024164
torque	998070.0	-0.003333	0.997907	-3.345953	-0.266917	-0.187246	0.547171	3.016971
i_d	998070.0	0.006043	0.998994	-3.245874	-0.756296	0.213935	1.013975	1.060937
i_q	998070.0	-0.003194	0.997912	-3.341639	-0.257269	-0.190076	0.499260	2.914185
pm	998070.0	-0.004396	0.995686	-2.631991	-0.672308	0.094367	0.680691	2.917456
stator_yoke	998070.0	0.000609	1.001049	-1.834688	-0.747265	-0.057226	0.697344	2.449158
stator_tooth	998070.0	-0.002208	0.999597	-2.066143	-0.761951	0.005085	0.772239	2.326668
stator_winding	998070.0	-0.003935	0.998343	-2.019973	-0.725622	0.006536	0.725660	2.653781
profile_id	998070.0	50.732001	22.073125	4.000000	32.000000	56.000000	68.000000	81.000000

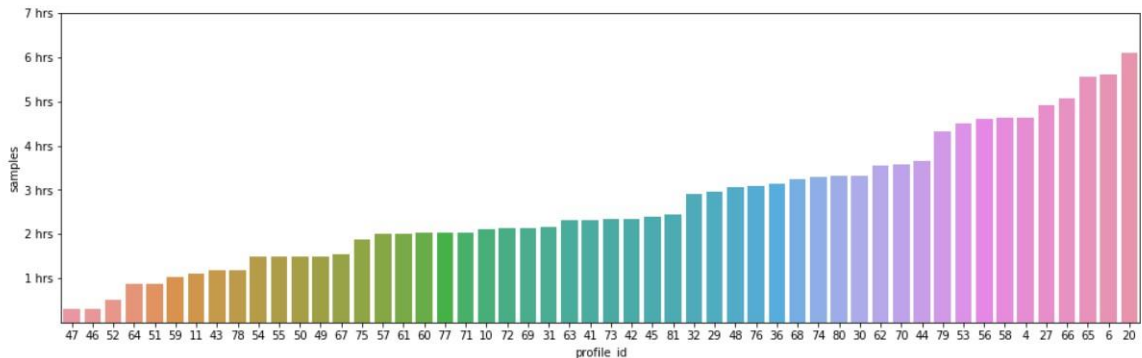
- Total number of timezones in which different sessions were recorded

```
In [8]: unique_profiles_id= df['profile_id'].unique()  
unique_profiles_id
```

Out[8]: array([4, 6, 10, 11, 20, 27, 29, 30, 31, 32, 36, 41, 42, 43, 44, 45, 46,
47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
64, 65, 66, 67, 68, 69, 70, 71, 73, 74, 75, 76, 77, 78, 79, 80, 81,
72], dtype=int64)

- Different session lengths in which measurements were taken

```
In [10]: fig = plt.figure(figsize=(17, 5))
grp = df.groupby(['profile_id'])
_df = grp.size().sort_values().rename('samples').reset_index()
ordered_ids = _df.profile_id.values.tolist()
sns.barplot(y='samples', x='profile_id', data=df, order=ordered_ids)
y_hrs = plt.yticks(2*3600*np.arange(1, 8), [f'{a} hrs' for a in range(1, 8)])
```



The plot shows that all reading sessions (profile_ids) ranges from 20 minutes to 6 hours, as each row represents one snapshot of sensor data at a certain time step. Sample rate is 2 Hz (One row per 0.5 seconds). The two short session ids "46" and "47" might be not very representative as temperatures inside electric motors need time to vary.

- **Techniques used** -
 1. Multiple Linear Regression
 2. Polynomial Regression
 3. Random Forest Regression
- **Software Requirements** -
 1. Python
 2. Jupyter
 3. WPS Office
 4. Spyder
- **Goals and Scopes** -
 1. Exploratory Analysis of the data
 2. To predict the rotor temp with the help of other variables.
 3. To select the best features for the model.
 4. Visualizing the data through Heatmaps and graphs.
- **Deliverables** -
 1. Source code
 2. Development document
 3. Training and test data.


Chapter 4

System Design

- **numPy** - It stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.
- **pandas** - Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.
- **matplotlib** - Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy, the numerical mathematics extension of Python.
- **seaborn** - Seaborn is an open source, BSD-licensed Python library providing high level API for visualizing the data using Python programming language.
- **r₂ score** - R₂ score(coefficient of determination) regression score function.R- squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination.
- **Mean Squared Error** - The mean squared error is used in this predicting model tells how close the regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them. The squaring is necessary to remove any negative signs. This is implemented using **sklearn**’s **mean_squared_error** method
- **Mean Absolute Error** - Mean absolute is the difference between the actual or true values and the values that are predicted. **MAE = True values – Predicted values** MAE takes the **average** of this error from every sample in a dataset and gives the output.This is implemented using **sklearn**’s **mean_absolute_error** method.

- **Flask Web Framework** - Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or Libraries. It supports extensions that can add application features as if they were implemented in Flask itself. In this project ,flask has been used to deploy the predicting module in web.

A screenshot of imported libraries -

```
In [1]:  # importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import explained_variance_score
from math import sqrt
```

Chapter-5

Project Planning and Feature Engineering

- **Displaying the data of a single measurement session.**

```
In [11]: measurements = df[df['profile_id'] == 62]
          measurements
```

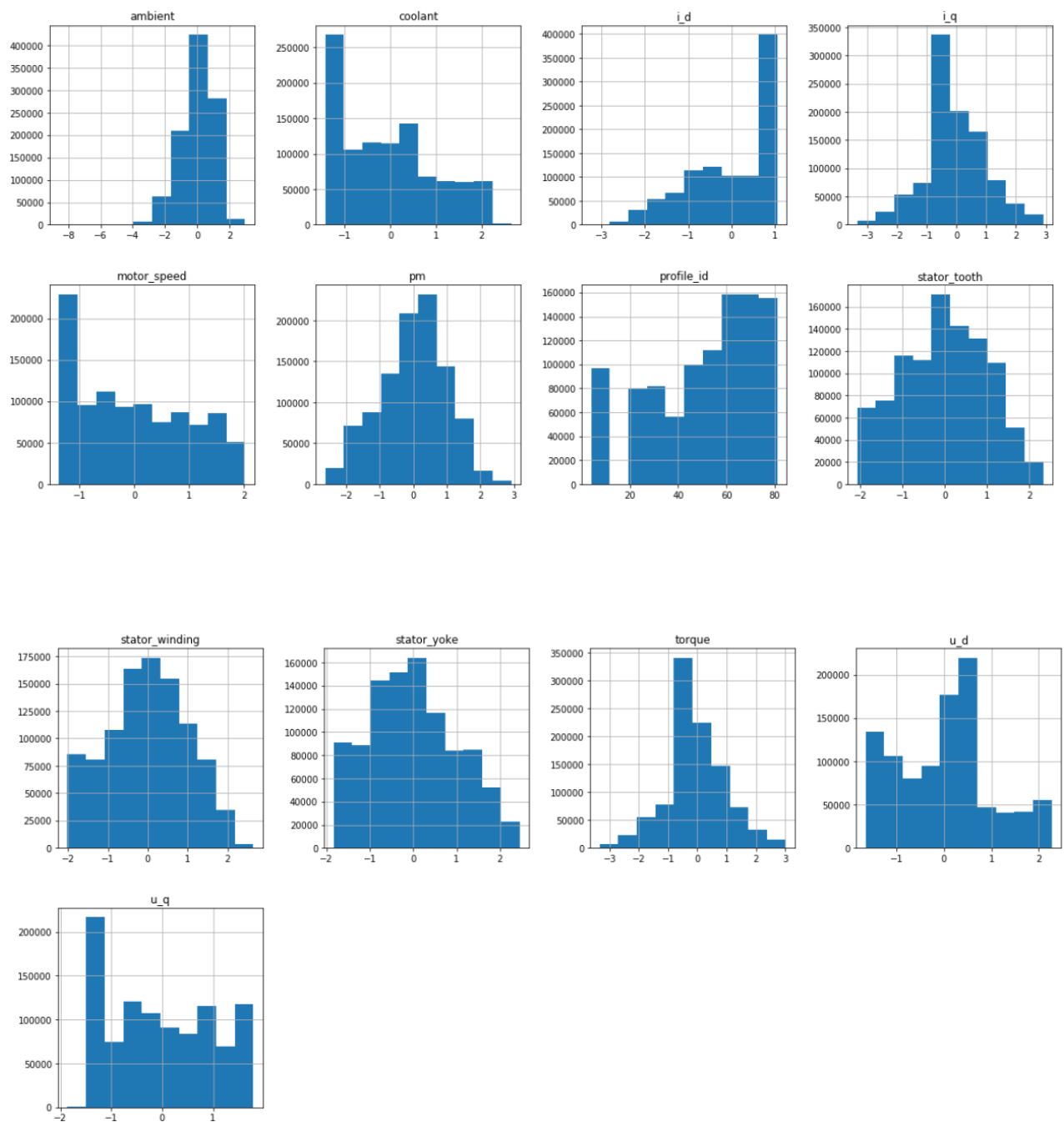
Out[11]:

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator_winding	profile_id
95628	-1.410154	-0.805239	0.343938	-1.304660	-1.222428	-0.255640	1.029165	-0.245710	-2.510381	-1.608242	-1.866557	-1.838565	62
95629	-1.410699	-0.804714	0.327403	-1.256085	-1.208100	-0.169601	1.009464	-0.142335	-2.511055	-1.607824	-1.867097	-1.838656	62
95630	-1.414667	-0.803909	0.301059	-1.144359	-1.160064	-0.099239	0.992839	-0.058038	-2.510708	-1.607688	-1.867189	-1.838910	62
95631	-1.421608	-0.802491	0.269131	-0.987023	-1.087290	-0.053933	0.983616	-0.003552	-2.509279	-1.605972	-1.866832	-1.839769	62
95632	-1.423779	-0.800906	0.236350	-0.796063	-0.996785	-0.031150	0.982299	0.024363	-2.508931	-1.603914	-1.867183	-1.840419	62
...
21222	-0.451854	-0.647815	0.316931	-1.251690	-1.222432	-0.255640	1.029141	-0.245720	-2.029862	-1.401945	-1.676269	-1.661222	62
21223	-0.445394	-0.646668	0.316149	-1.250902	-1.222432	-0.255640	1.029143	-0.245745	-2.029952	-1.402815	-1.676549	-1.661946	62
21224	-0.448949	-0.645318	0.316539	-1.250752	-1.222429	-0.255640	1.029168	-0.245694	-2.029971	-1.402027	-1.676586	-1.662592	62
21225	-0.456404	-0.644326	0.316353	-1.250179	-1.222430	-0.255640	1.029140	-0.245716	-2.029869	-1.400734	-1.676054	-1.662751	62
21226	-0.459632	-0.643422	0.316315	-1.250604	-1.222432	-0.255640	1.029140	-0.245722	-2.030152	-1.399899	-1.675672	-1.662511	62

1599 rows × 13 columns

- **Histograms of the variables**

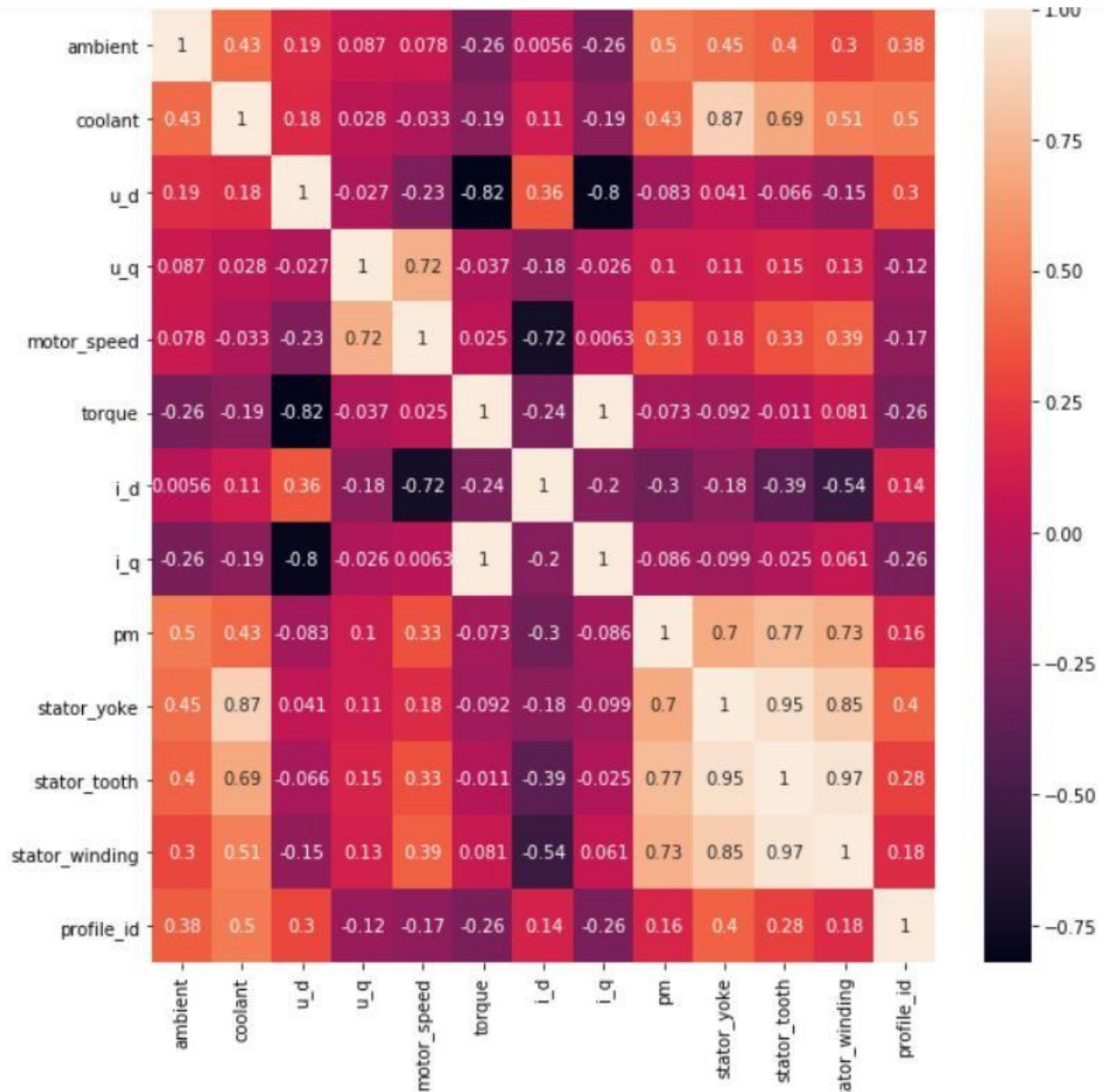

```
In [12]: df.hist(figsize=(20,20))
plt.show()
```



- **Correlation matrix**

Finding the correlations between the variables

```
In [13]: cor = df.corr()
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(cor, xticklabels=cor.columns, yticklabels=cor.columns, annot=True, ax=ax);
```



In our model, we are trying to predict the pm or the permanent magnet temperature that is the temperature of the rotor windings, so with the correlation plot we can actually see that how other variables are affecting the rotor temperature, as more +ve coefficient variables means more correlated to our dependent variable i.e. it directly affects our pm and more -ve coefficient variables means opposite to our dependent variable which can be explained as if that particular independent increases

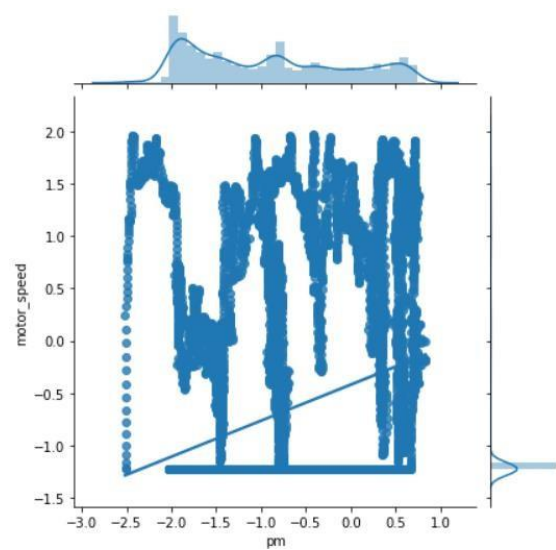
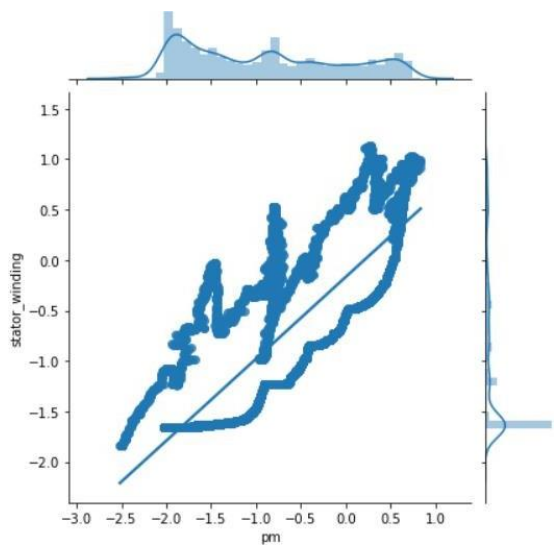
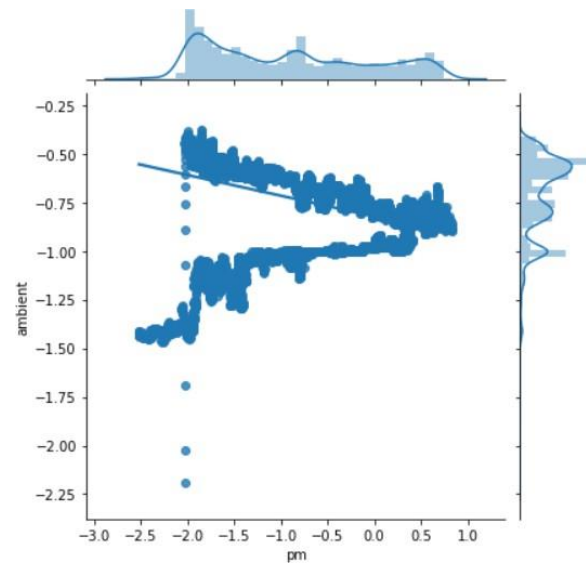
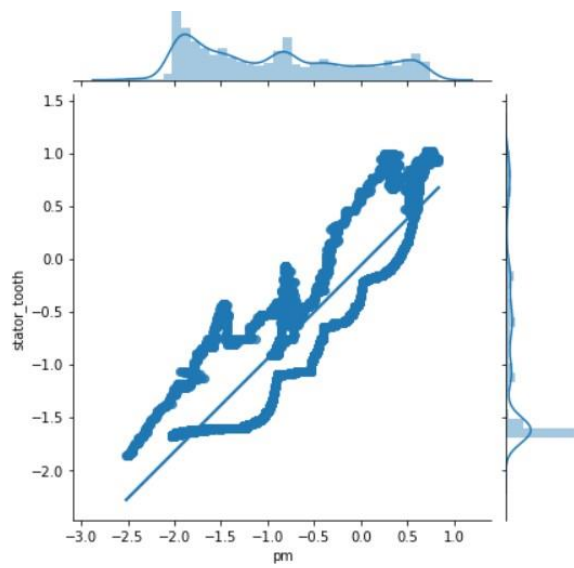
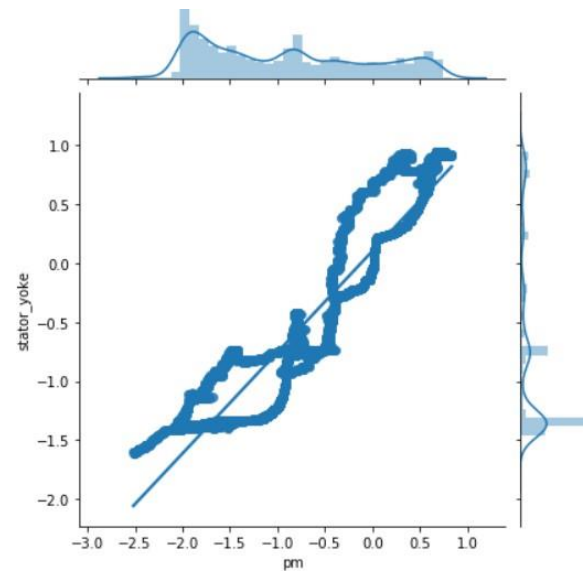
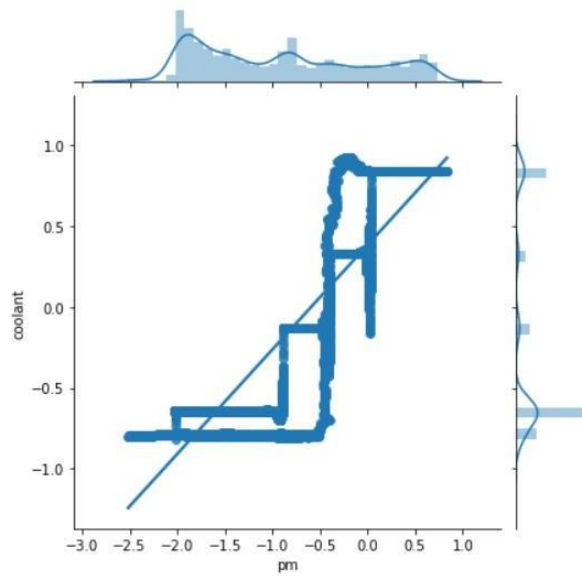
then there will be a decrease in the dependent variable or in simple words is indirectly proportional to the pm or rotor temperature.

So, we can classify that

1. temperature of stator_yoke, stator_tooth, stator_windings, ambient temperature, motor_speed, coolant has a positive co-efficient which means it is directly proportional to the rotor winding temp.
2. Voltage (active component), Current (both active and reactive components), Torque induced by current has a negative co-efficient which means it is indirectly proportional to the rotor winding temp.

- **Plots showing the relationship between the pm(rotor temp) and the positive coefficient variables of a particular profile id.**

```
In [14]: sns.jointplot(x='pm', y='coolant', data=df[df['profile_id']==62], kind='reg')
sns.jointplot(x='pm', y='stator_yoke', data=df[df['profile_id']==62], kind='reg')
sns.jointplot(x='pm', y='stator_tooth', data=df[df['profile_id']==62], kind='reg')
sns.jointplot(x='pm', y='stator_winding', data=df[df['profile_id']==62], kind='reg')
sns.jointplot(x='pm', y='ambient', data=df[df['profile_id']==62], kind='reg')
sns.jointplot(x='pm', y='motor_speed', data=df[df['profile_id']==62], kind='reg')
```



above plots shows the conclusion that were drawn regarding the relationship between the coolant, stator_yoke, stator_tooth and stator_winding, ambient. So we consider using a regression model to estimate the rotor winding temp using the four inputs stator_winding, coolant, stator_yoke and stator_tooth.

DATA PREPARATION AND HANDLING

● Dividing the data into dependent and independent variables.

```
In [15]: X= df.drop(['pm','profile_id','torque','u_d','u_q','i_d','i_q'],axis=1)
        Y= df['pm']
```

```
In [16]: #the Independent Variables
        X
```

Out[16]:

	ambient	coolant	motor_speed	stator_yoke	stator_tooth	stator_winding
0	-0.752143	-1.118446	-1.222428	-1.831422	-2.066143	-2.018033
1	-0.771263	-1.117021	-1.222429	-1.830969	-2.064859	-2.017631
2	-0.782892	-1.116681	-1.222428	-1.830400	-2.064073	-2.017343
3	-0.780935	-1.116764	-1.222430	-1.830333	-2.063137	-2.017632
4	-0.774043	-1.116775	-1.222429	-1.830498	-2.062795	-2.018145
...
998065	-0.047497	0.341638	-1.222428	1.018568	0.836084	0.494725
998066	-0.048839	0.320022	-1.222437	1.013417	0.834438	0.494279
998067	-0.042350	0.307415	-1.222430	1.002906	0.833936	0.492666
998068	-0.039433	0.302082	-1.222432	0.999157	0.830504	0.490581
998069	-0.043803	0.312666	-1.222431	0.987163	0.828046	0.489382

998070 rows × 6 columns

```
In [17]: #the Dependent Variables
        Y
```

```
Out[17]: 0      -2.522071
         1      -2.522418
         2      -2.522673
         3      -2.521639
         4      -2.521900
         ...
         998065    0.429853
         998066    0.429751
         998067    0.429439
         998068    0.429558
         998069    0.429166
         Name: pm, Length: 998070, dtype: float64
```

● Splitting the data into train and test variables.

```
In [18]: 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)
```

test_size = 0.2 , means 80% of the original data has been deployed as train values and remaining 20% of the data as test values , ie. we'll build the regressor models on these 80% of the data and will test the model on the remaining 20%.

```
In [19]: X_train
```

Out[19]:

	ambient	coolant	motor_speed	stator_yoke	stator_tooth	stator_winding
536628	0.688617	-0.679435	0.496113	-0.134675	0.396245	0.943954
362938	-0.030076	-0.613747	-0.410794	-0.812338	-0.912366	-1.015070
105287	-0.626754	-1.070268	-1.019529	-1.616032	-1.778508	-1.760606
828039	0.571349	1.349669	-0.853185	1.327840	1.346239	1.570846
41574	0.324606	0.319417	1.483027	0.555170	0.770201	0.790181
...
963395	-0.003020	1.191647	-1.222431	0.534866	0.123472	-0.122505
117952	-0.620314	-1.082453	-0.546069	-0.722736	-0.415779	-0.255078
435829	0.692343	-0.607927	1.381093	-0.164919	0.140875	0.326160
305711	-0.082905	1.256598	0.671390	0.954370	0.770257	0.462634
985772	-0.463147	2.202754	-1.222426	1.668635	1.091656	0.580089

798456 rows × 6 columns

```
In [20]: y_train
```

Out[20]:

536628	0.287970
362938	-0.186863
105287	-1.094885
828039	0.600092
41574	0.817969
...	...
963395	0.543466
117952	1.498485
435829	0.216653
305711	0.280744
985772	-0.218149

Name: pm, Length: 798456, dtype: float64

Chapter-6

IMPLEMENTATION

REGRESSION MODELS

A brief on the regression models in which we'll be working on -

The Formula for Multiple Linear Regression Is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

y_i = dependent variable

x_i = explanatory variables

β_0 = y-intercept (constant term)

β_p = slope coefficients for each explanatory variable

ϵ = the model's error term (also known as the residuals)

MULTIPLE LINEAR - Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable. In essence, multiple regression is the extension of

ordinary least-squares (OLS) regression that involves more than one explanatory variable.

POLYNOMIAL - polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an n th degree polynomial in x .

The polynomial regression model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_m x_i^m + \epsilon_i \quad (i = 1, 2, \dots, n)$$

Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y , denoted $E(y | x)$. Although *polynomial regression* fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function $E(y | x)$ is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of multiple linear regression.

RANDOM FOREST - Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn't depend on one decision tree but multiple

decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs.

● Multiple Linear Regression

```
In [23]: from sklearn.linear_model import LinearRegression
         regressor = LinearRegression()
         regressor.fit(X_train, y_train)

Out[23]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [24]: y_pred = regressor.predict(X_test)

In [25]: y_pred

Out[25]: array([-1.4250432 ,  0.33387872, -0.1439759 , ...,  0.02780903,
                0.13589972,  0.01258218])

In [26]: print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
         Mean Absolute Error: 0.39026695573369147

In [27]: print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
         Root Mean Squared Error: 0.509159749248125

In [28]: print('r2 score:', r2_score(y_test, y_pred))
         r2 score: 0.7393110007330491

In [29]: print('Explained Variance Score:', explained_variance_score(y_test, y_pred))
         Explained Variance Score: 0.7393114311016598
```

At first we imported the LinearRegression module from scikit learn library, after that we created a regressor and trained with the data of X_train and y_train.

Then with the help of that regressor we predicted the X_test values and stored it in a vector called y_pred.

Now, for testing the accuracy of our model

Mean absolute error which is the mean of the errors between y_test and y_pred which is 0.39 and can be considered as an avg model,

R2 score of the model is 0.73 , which is not that close to 1.00(ideal model), thus we can say that the regressor model is 73% accurate which is not that good.

So, we try the next type of regression

● Polynomial Regression

```
In [30]: from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X_train)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y_train)

Out[30]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [31]: y_pred = lin_reg_2.predict(poly_reg.fit_transform(X_test))

In [32]: print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
Mean Absolute Error: 0.28644334980392194

In [33]: print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
Root Mean Squared Error: 0.38115088483575366

In [34]: print('r2 score:', r2_score(y_test, y_pred))
r2 score: 0.8539140525094435

In [35]: print('Explained Variance Score:', explained_variance_score(y_test, y_pred))
Explained Variance Score: 0.8539147349074863
```

Here the Mean Absolute Error is 0.28 which is less than the previously created regressor model also the r2 score is 0.85 which is more than the Multiple Linear regressor, thus this model shows a better alternative than the multiple linear model .

● Random Forest Regression

```

In [36]: from sklearn.ensemble import RandomForestRegressor
         regressor = RandomForestRegressor(n_estimators = 4, random_state = 0)
         regressor.fit(X_train, y_train)

Out[36]: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                                max_depth=None, max_features='auto', max_leaf_nodes=None,
                                max_samples=None, min_impurity_decrease=0.0,
                                min_impurity_split=None, min_samples_leaf=1,
                                min_samples_split=2, min_weight_fraction_leaf=0.0,
                                n_estimators=4, n_jobs=None, oob_score=False,
                                random_state=0, verbose=0, warm_start=False)

In [37]: y_pred= regressor.predict(X_test)

In [38]: print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
         Mean Absolute Error: 0.010652780262612386

In [39]: print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
         Root Mean Squared Error: 0.04727440309780409

In [40]: print('r2 score:', r2_score(y_test, y_pred))
         r2 score: 0.9977526708499643

In [41]: print('Explained Variance Score:', explained_variance_score(y_test, y_pred))
         Explained Variance Score: 0.9977527407776043

```

Created with four decision trees , the Random Forest regressor outclasses both the Multiple linear model and the Polynomial model ,

As it has clearly the less Mean squared error than the other two,

The r2 Score of this model is 99%, which is a very good model but there's a chance of overfitting which may cause trouble later.

So , clearly we can say that both the Polynomial Regression model and Random Forest Regression model can best fit with the data and shows more accurate results .

DEPLOYING THE MACHINE LEARNING MODEL USING FLASK

Now we'll be developing the front end of our model.

- For this, we have our model predict file , which includes the necessary information required to build the model.

```
minor_project.py × index.html × app_minor.py × request_minor.py ×
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import seaborn as sns
5 import pickle
6
7 df1=pd.read_csv('pmsm_temperature_data.csv')
8 df=df1.copy()
9
10 X= df.drop(['pm', 'profile_id', 'torque', 'u_d', 'u_q', 'i_d', 'i_q'],axis=1)
11 Y= df['pm']
12
13 from sklearn.ensemble import RandomForestRegressor
14 regressor = RandomForestRegressor(n_estimators = 4, random_state = 0)
15 regressor.fit(X, Y)
16
17 #saving model to a disk
18 pickle.dump(regressor, open('model1.pkl', 'wb'))
19
20 #loading model to compare the results
21 model1 = pickle.load(open('model1.pkl', 'rb'))
22
23
```

- Now we need to have an index.html file which will basically act as our front end web app so that any request which we will give to our model will be in the form of api which we're going to host through flask , it will interact with it to get the particular output from the api itself.

```
minor_project.py × index.html × app_minor.py × request_minor.py ×
1 <!DOCTYPE html>
2 <html >
3 <!--From https://codepen.io/frytyler/pen/EGdtg-->
4 <head>
5   <meta charset="UTF-8">
6   <title>ML API</title>
7   <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'
8   <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
9   <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
10  <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' ty
11  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
12
13 </head>
14
15 <body>
16   <div class="Login">
17     <h1>Predict Rotor Temperature</h1>
18
19     <!-- Main Input For Receiving Query to our ML -->
20     <form action="{{ url_for('predict') }}" method="post">
21       <input type="text" name="ambient" placeholder="ambient" required="required" />
22       <input type="text" name="coolant" placeholder="coolant" required="required" />
23       <input type="text" name="motor_speed" placeholder="motor_speed" required="required" />
24       <input type="text" name="stator_yoke" placeholder="stator_yoke" required="required" />
25       <input type="text" name="stator_tooth" placeholder="stator_tooth" required="required" />
26       <input type="text" name="stator_winding" placeholder="stator_winding" required="required
27
28       <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
29     </form>
30
31     <br>
32     <br>
33     {{ prediction_text }}
34
35   </div>
36
37
38 </body>
39 </html>
40
```

- Now we'll be creating the flask environment where we will be creating our api's where we'll be reading the model1.pkl file, input the data and finally get the output.

```

minor_project.py × index.html × app_minor.py* × request_minor.py ×
1  import numpy as np
2  from flask import Flask, request, jsonify, render_template
3  import pickle
4
5  app = Flask(__name__)
6  model1 = pickle.load(open('model1.pkl', 'rb'))
7
8  @app.route('/') #root node where the api should go
9  def home():
10     return render_template('index.html') #it will directly redirect us to the .html file
11
12  @app.route('/predict',methods=['POST']) #it is used to create any no. of uri w.r.t the api's.
13  def predict():
14     '''
15     For rendering results on HTML GUI
16     '''
17     float_features = [x for x in request.form.values()]
18     final_features = [np.array(float_features)]
19     prediction = model1.predict(final_features)
20
21     output = round(prediction[0], 2)
22     return render_template('index.html', prediction_text='Rotor Temperature is {}'.format(output))
23
24  @app.route('/predict_api',methods=['POST'])
25  def predict_api():
26     '''
27     For direct API calls through request
28     '''
29     data = request.get_json(force=True)
30     prediction = model1.predict([np.array(list(data.values()))])
31
32     output = prediction[0]
33     return jsonify(output)
34
35  if __name__ == "__main__":
36     app.run(debug=True)

```

- We've to collect the json values from a random observation, for that we're creating an url

```

minor_project.py × index.html × app_minor.py* × request_minor.py ×
1  import requests
2
3  url = 'http://localhost:5000/predict_api'
4  r = requests.post(url,json={'ambient':-0.752143, 'coolant':-1.11845, 'motor_speed':-1.22243, 's':
5
6  print(r.json())

```

- Now all we've to do is to run the Anaconda prompt and paste the url, creating a link to our api.

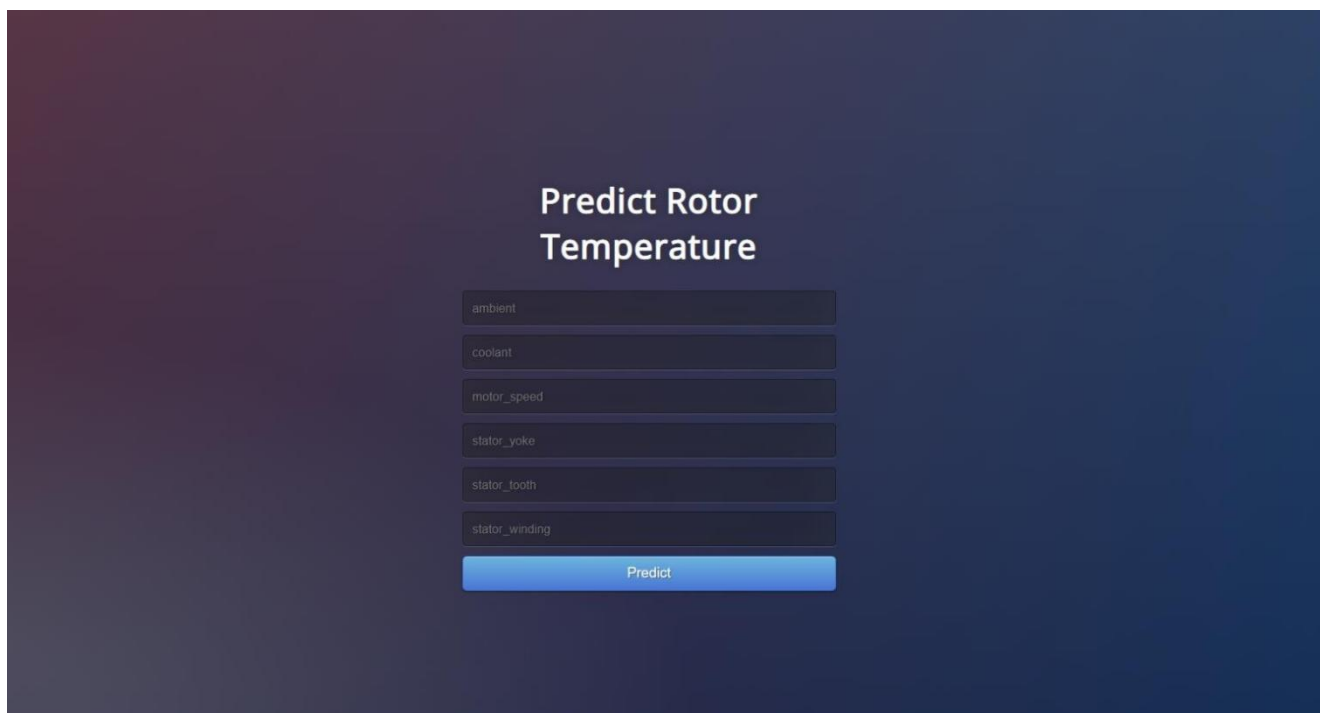
```
Anaconda Prompt (anaconda3) - python app_minor.py

(base) C:\Users\Soumya>cd Downloads

(base) C:\Users\Soumya\Downloads>cd Deployment-flask-master

(base) C:\Users\Soumya\Downloads\Deployment-flask-master>python app_minor.py
* Serving Flask app "app_minor" (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 windowsapi reloader
* Debugger is active!
* Debugger PIN: 158-300-952
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Our API looks like,



Predict Rotor Temperature

ambient

coolant

motor_speed

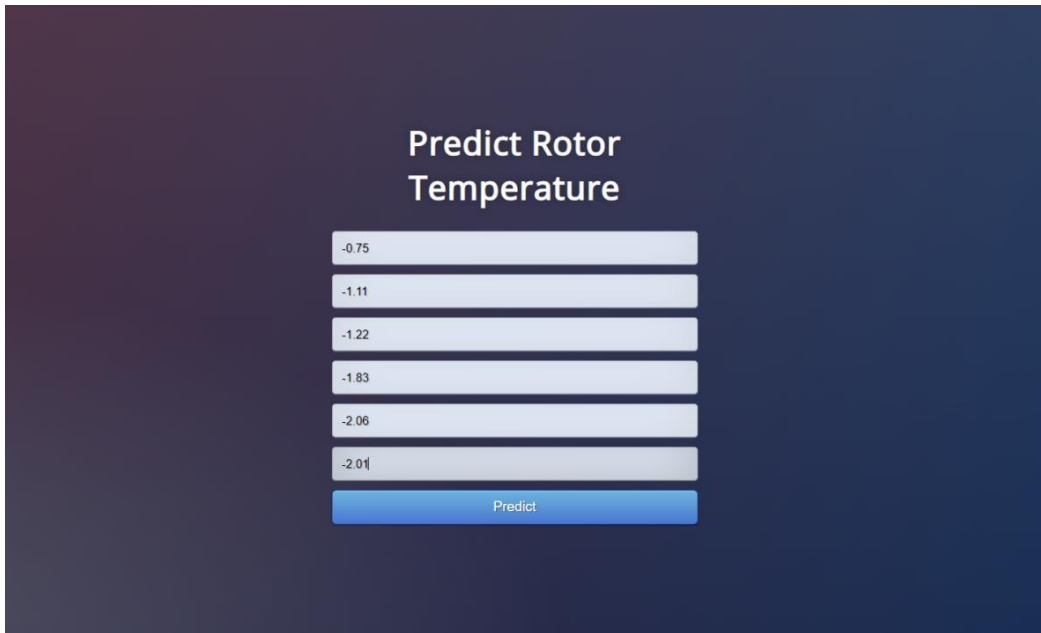
stator_yoke

stator_tooth

stator_winding

Predict

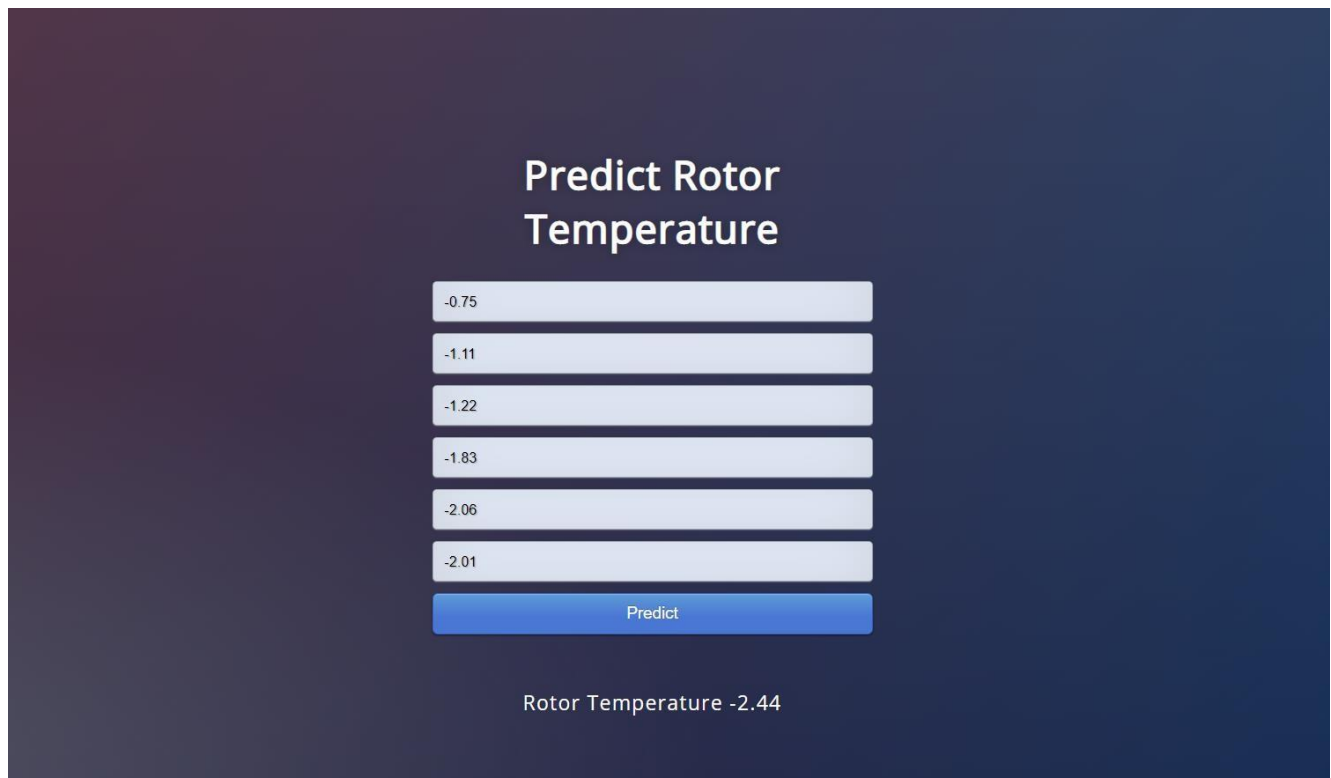
- INPUTTING DATA OF THE INDEPENDENT VARIABLES



The screenshot shows a web application titled "Predict Rotor Temperature". It features six input fields for independent variables, each containing a numerical value: -0.75, -1.11, -1.22, -1.83, -2.06, and -2.01. Below these fields is a blue button labeled "Predict".

Independent Variable
-0.75
-1.11
-1.22
-1.83
-2.06
-2.01

We get the output ie. The dependent variable as in Rotor Temperature after clicking the predict button as,



The screenshot shows the same web application as before, but now it displays the predicted output. The "Predict" button is highlighted, and below it, the text "Rotor Temperature -2.44" is displayed.

Independent Variable
-0.75
-1.11
-1.22
-1.83
-2.06
-2.01

Rotor Temperature -2.44

Chapter-7

Conclusion

A Simple conclusion to why we are creating this model is because of the functional safety, smooth functioning of the Synchronous Motor as well as cost effectiveness. As the complexity in the internal structure of an electric traction drive, direct measurement with thermal sensors is not possible for rotor temperatures, thus instead of sensing the rotor temperature through different sensor which might prove to be costlier and more cumbersome because of the rotating nature of the rotor we nullified that problem by building a model which will successfully predict the temperature of the rotor by taking into account the different features of the Synchronous motor. And showcased it with the help of an API.

Future Scope

The scope for our model is pretty clear from the abstract itself is, that in future purpose it can be used by different industries, be it small or big who like to be cost effective in its spendings and also for the user safety and for functional safeties of a Synchronous motor, as with the help of this model, after it being deployed as an API it can show the overheating of the rotor as per the industry demands the threshold respectively for the load it is supposed to produce, the the worker or the user can shut it down when the temperature gets past the threshold.

Chapter-8

References

1. The data frame containing all the recordings and measurements of the Electric Motor was obtained from kaggle, link is
<https://www.kaggle.com/wkirgsn/electric-motor-temperature>
2. The link from which we got the idea of creating an API through flask was from GitHub, the link is
<https://github.com/krishnaik06/Deployment-flask>

