Internship Project Report

Predicting the critical parameters of Blast Furnace using Machine Learning

A report submitted as a part of the industrial Orientation Training in IT & ERP Department

M. Lalith Sri Sai

V. Naga Koushik

M. Sampath Akash

M.V. Chandra Shekara Reddy

RASHTRIYA ISPAT NIGAM LIMITED (RINL), Visakhapatnam

DURATION: 6 May 2024 – 29 June 2024

Andhra University College of Engineering

Department of Computer Science Engineering

CERTIFICATE

This is to certify that following students of Andhra University College of Engineering, Visakhapatnam is engaged in the project work titled "**A Project Based on Python (ML)**"

- **M. Lalith Sri Sai** Trainee No: 100034218
- **V. Naga Koushik** Trainee No: 100033965
- **M. Sampath Akash** Trainee No: 100034340
- **M. Chandra Sekhara Reddy** Trainee No: 100034221

In partial fulfilment of the degree of Bachelor of Technology in Computer Science and Engineering stream at Andhra University College of Engineering, Visakhapatnam, this certificate acknowledges their bonafide work carried out under my guidance and supervision during the period from 6 May 2024 to 29 June 2024.

Date: 29 June 2024

Place: Visakhapatnam

Mr. T.V. Kameshwar Rao

D.G.M. (IT&ERP),

IT & ERP Department,

RINL- VSP

ACKNOWLEDGEMENT

I would like to extend my sincere appreciation to the individuals whose guidance and support have been pivotal in the completion of this project. Their expertise and assistance have been invaluable throughout, and I am deeply grateful for their contributions.

I wish to express my gratitude to Mr. T.V.K. Rao, D.G.M(IT & ERP), Visakhapatnam Steel Plant, for his extensive knowledge and guidance, which have been instrumental in navigating the complexities of this project and ensuring its successful completion.

Furthermore, my appreciation extends to the dedicated employees of the IT and ERP Department at Visakhapatnam Steel Plant for their unwavering support and guidance throughout this endeavour.

Their collective efforts have greatly enhanced the outcome of this project, and I am sincerely thankful for their invaluable contributions.

DECLARATION

I affirm that the work titled "A Project Based on ML" presented herein is entirely original and has not been previously submitted to any university or college to fulfil the requirements of any academic course or degree. The opinions expressed and conclusions drawn within this work are solely my own.

TABLE OF CONTENTS

ABSTRACT

A blast furnace (BF) is a critical reactor and heat exchanger, serving as the first step in steel production. As one of the most complex industrial reactors, accurately modelling a blast furnace mathematically is challenging. The operation and control of an industrial blast furnace present significant difficulties. To address this, we employ various machine learning algorithms to predict key temperatures, such as Skin Temperature, at different intervals. Optimizing these temperature distributions can result in substantial savings in the blast furnace's input materials.

This project utilizes diverse regression techniques to predict these output parameters, aiming to enhance overall efficiency, reduce operational costs, and decrease fuel consumption, thereby improving productivity.

The input parameters considered include Cold Blast Flow, Cold Blast Pressure, Cold Blast Temperature, Steam Flow, O2 Flow, O2 Percentage, Pulverized Coal Injection Rate, Atmospheric Humidity, Hot Blast Temperature, Hot Blast Pressure, and CO, CO2, H2 Content. The best-performing algorithm is selected and deployed using Flask to provide an interactive user interface.

A BRIEF OVERVIEW OF VSP

Visakhapatnam Steel Plant (VSP), the integrated steel plant of Rashtriya Ispat Nigam Limited (RINL), was founded in 1971 in Visakhapatnam, Andhra Pradesh. VSP is renowned for its cutting-edge technology, superior product quality, and picturesque location.

The decision to establish VSP was announced by then Prime Minister Smt. Indira Gandhi in Parliament on January 17, 1971. VSP is India's first coastal-based integrated steel plant, located 16 km west of Visakhapatnam. With a modern infrastructure, VSP has an installed capacity of 3 million tons per annum of liquid steel and 2.656 million tons of saleable steel. The plant produces a variety of products, including wire rod coils, structural steel, special steel, rebar, and forged rounds.

VSP has achieved notable certifications, becoming the first integrated steel plant in India to be certified for ISO 9001 (Quality), ISO 14001 (Environment Management), and OHSAS 18001 (Occupational Health & Safety). VSP's marketing network includes four regional offices, 20 branch offices, and 22 stockyards across the country.

VSP is committed to environmental sustainability, having invested Rs. 460 crores in pollution and environment control equipment and transforming the once-barren landscape with over 3 million plants. The plant's products, including export-quality pig iron and steel, are sold in international markets such as Sri Lanka, Myanmar, Nepal, the Middle East, the USA, and Southeast Asia. RINL-VSP was awarded "Star

Trading House" status for 1997-2000, highlighting its established presence and dependability in the export market.

Different sections at the RINL VSP:

- Coke Ovens & Coal Chemical plant
- Thermal Power Plant
- Chemical Power Plant
- Rolling Mills
- Blast Furnace Section
- Steel Melting Shop (SMS)
- Sinter Plant
- Raw Material Handling Plant (RMHP)
- Research and Development (R&D) Center
- Calcining & Refractive materials plant
- Continuous casting machine
- Light & Medium machine mills

Blast Furnace Operations at RINL-VSP:

RINL-VSP's Blast Furnaces:

The Rashtriya Ispat Nigam Limited - Visakhapatnam Steel Plant (RINL-VSP) boasts three of India's most colossal blast furnaces, each with a volumetric capacity of **3200 cubic meters**. These state-of-the-art furnaces incorporate Petworth Bell-less top technology alongside advanced conveyor charging systems. Two of these giants are named

‘Godavari’ and ‘Krishna’, honoring the revered rivers of Andhra Pradesh. Their operation is pivotal to the region’s industrial growth.

Innovative Features and Production:

The plant is equipped to handle the granulation of all the liquid slag produced, directly at the blast furnace cast house. The blast furnace gas is harnessed at a maximum pressure of **1.5 to 2.0 atmospheres** to generate **12 MW** of electricity per furnace via gas expansion turbines. Each furnace’s unique circular cast house, complete with four tap holes, enables a daily output of **9720 tons** of hot metal. This translates to an impressive annual production capacity of **3.4 million tons** of low-sulphur hot metal.

INPUT PARAMETERS

1. Cold Blast Pressure:

This refers to the pressure of the air before it enters the stove system, where it is heated. Maintaining the correct cold blast pressure is essential for ensuring that the air can be properly heated and delivered to the furnace.

2. Cold Blast Temperature:

This is the temperature of the air before it is heated by the stoves. The cold blast is typically at ambient temperature before it undergoes the heating process.

3. Steam Flow:

The rate at which steam is introduced into the blast furnace. Steam is used to regulate the temperature within the furnace and can also aid in the reduction of iron ore by contributing to the chemical reactions.

4. Steam Temperature:

The temperature of the steam that is injected into the blast furnace. Proper control of steam temperature is important for effective thermal management and reaction control within the furnace.

5. Steam Pressure:

The pressure at which steam is injected into the blast furnace. High-pressure steam can be more effective in penetrating the furnace and influencing the internal reactions.

6. O2 Pressure:

The pressure of oxygen supplied to the blast furnace. Oxygen is often enriched in the air blast to increase combustion efficiency and support the reduction process.

7. O2 Flow:

The rate at which oxygen is delivered to the blast furnace. Accurate control of oxygen flow is critical for maintaining the desired reaction rates and temperatures within the furnace.

8. O2 Percentage:

The proportion of oxygen in the air or gas mixture used in the blast furnace. Increasing the oxygen percentage can enhance combustion and improve furnace efficiency.

9. Pulverized Coal Injection (PCI):

A technique where finely ground coal is injected directly into the blast furnace through the tuyeres. PCI reduces the need for coke, lowers production costs, and can improve furnace efficiency.

10. Atmospheric Humidity:

The amount of water vapor present in the air surrounding the blast furnace. High atmospheric humidity can affect the performance of the furnace by influencing the moisture content of the blast air.

11. Hot Blast Temperature:

The temperature of the air after it has been heated by the stoves and before it enters the blast furnace. High hot blast temperatures improve the efficiency of the furnace by reducing the need for additional fuel.

12. Hot Blast Pressure:

The pressure of the air after it has been heated and before it is injected into the blast furnace. Maintaining the correct hot blast pressure ensures that the hot air can properly penetrate the furnace and support the smelting process.

13. Top Gas Temperature:

The temperature of the gas exiting the top of the blast furnace. This gas consists of byproducts of the smelting process and its temperature can indicate the overall efficiency of the furnace.

14. Top Gas Pressure:

The pressure of the gas collected from the top of the blast furnace. Monitoring this pressure helps in controlling the gas flow and ensuring the safe operation of the furnace.

15. Top Gas Spray:

A method used to cool the top gas by spraying water into it. This process reduces the temperature of the gas before it is processed further or emitted.

16. CO (Carbon Monoxide):

A byproduct gas produced in the blast furnace from the combustion of coke. CO acts as a reducing agent, reacting with iron ore to produce molten iron and carbon dioxide.

17. CO2 (Carbon Dioxide):

Another byproduct gas from the blast furnace, produced primarily from the combustion of carbon-based materials.

18. H2 (Hydrogen):

A gas that can be present in the blast furnace, either as a byproduct of reactions involving water vapor or as an intentional additive to enhance reduction processes. Hydrogen can contribute to reducing iron ore to iron.

Output Parameters

Average Skin Temperature:

Description: The average temperature of the blast furnace's outer shell, monitored to ensure the structural integrity and safety of the furnace.

Impact: Maintaining proper skin temperatures is vital to prevent damage to the refractory lining. Excessive skin temperatures can lead to refractory failure and potential leaks, posing safety hazards. Thus, controlling skin temperature helps in protecting the furnace and ensuring its longevity.

In this project, we will be predicting *Average Skin Temperature in real-time and also, forecast the skin temperature after 1, 2, 3, 4 hrs respectively*.

METHODOLOGY

To accurately predict output parameters, we need to follow a systematic machine learning workflow. This process involves several critical phases that contribute to the development and effectiveness of the model. These stages are:

1. Data Collection: Gather relevant and high-quality data from various sources to ensure a robust dataset.

2. Data Preparation: Clean, preprocess, and transform the data to make it suitable for analysis. This step includes handling missing values and normalizing data and detecting outliers.

3. Choosing Learning Algorithm: Select the appropriate machine learning algorithm based on the nature of the data and the prediction goals.

4. Training the Model: Use the training dataset to train the chosen algorithm, allowing it to learn the underlying patterns and relationships within the data.

5. Evaluating the Model: Assess the model's performance using the testing dataset and relevant metrics to ensure accuracy and generalizability.

6. Predicting the Output: Apply the trained model to new, unseen data to generate predictions for the desired output parameters.

DATA COLLECTION

The data was collected from the Vizag Steel Plant (VSP) over a five-month period, from July 1, 2021, to December 31, 2021, at ten-minute intervals. The dataset consists of 25,405 rows and 26 columns. It contains several missing values and other errors that need to be addressed.

DATA PREPARATION

We take the following steps to prepare data for model training:

1. Data Importing
2. Removing unnecessary columns
3. Remove rows with null values in SKIN_TEMP_AVG
4. Convert DATE_TIME to ‘datetime’ datatype
5. Find distribution of each feature and choose the fill strategy
6. Derive SKIN_TEMP_AVG for 1, 2, 3, 4 hrs

- # DATA IMPORTING

We import the data as follows:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df_og = pd.read_excel('dataset.xlsx')
df = df_og.copy(deep = True)
df.head()
```

Python

	DATE_TIME	CB_FLOW	CB_PRESS	CB_TEMP	STEAM_FLOW	STEAM_TEMP	STEAM_PRESS	O2_PRESS	O2_FLOW	O2_PER	...	TOP_TEMP
0	01-07-21 00:10	311727.0	3.15	129.0	4.0	213.0	3.34	3.20	7296.0	23.08	...	135.
1	01-07-21 00:20	315163.0	3.16	129.0	4.0	209.0	3.35	3.20	7829.0	23.08	...	143.
2	01-07-21 00:30	314595.0	3.16	128.0	4.0	205.0	3.35	3.21	7904.0	23.08	...	138.
3	01-07-21 00:40	312465.0	3.16	127.0	4.0	200.0	3.35	3.21	7919.0	23.08	...	128.
4	01-07-21 00:50	302981.0	3.11	126.0	4.0	194.0	3.29	3.16	7938.0	23.08	...	139.

- # REMOVING UNNECESSARY COLUMNS

There exists a column TOP_TEMP which is the average of TOP_TEMP1, TOP_TEMP2, TOP_TEMP3, TOP_TEMP4. So, we remove these extra columns. We also remove TOP_PRESS_1 for the same reason.

```
df.drop(['TOP_TEMP1', 'TOP_TEMP2', 'TOP_TEMP3', 'TOP_TEMP4', 'TOP_PRESS_1'], axis = 1, inplace = True)
df.head()
```

Python

	DATE_TIME	CB_FLOW	CB_PRESS	CB_TEMP	STEAM_FLOW	STEAM_TEMP	STEAM_PRESS	O2_PRESS	O2_FLOW	O2_PER	...	ATM_HUMI
0	01-07-21 00:10	311727.0	3.15	129.0	4.0	213.0	3.34	3.20	7296.0	23.08	...	24.!
1	01-07-21 00:20	315163.0	3.16	129.0	4.0	209.0	3.35	3.20	7829.0	23.08	...	24.!
2	01-07-21 00:30	314595.0	3.16	128.0	4.0	205.0	3.35	3.21	7904.0	23.08	...	24.!
3	01-07-21 00:40	312465.0	3.16	127.0	4.0	200.0	3.35	3.21	7919.0	23.08	...	25.(
4	01-07-21 00:50	302981.0	3.11	126.0	4.0	194.0	3.29	3.16	7938.0	23.08	...	25.:

- REMOVE ROWS THAT CONTAIN NULL FOR SKIN_TEMP_AVG:

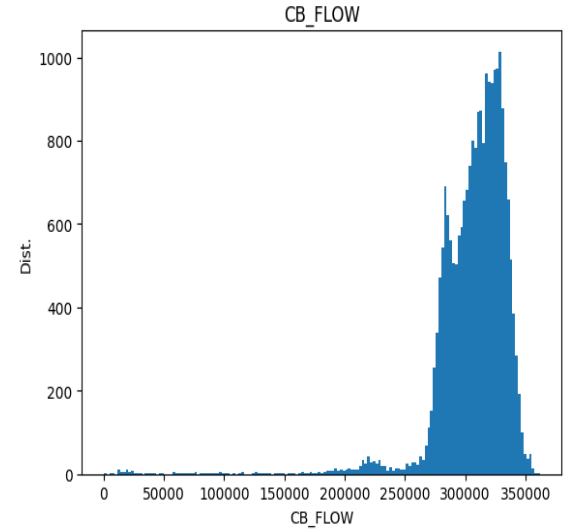
```
df.drop(df[df['SKIN_TEMP_AVG'].isna()].index, axis = 0, inplace = True)
df.isna().sum()
```

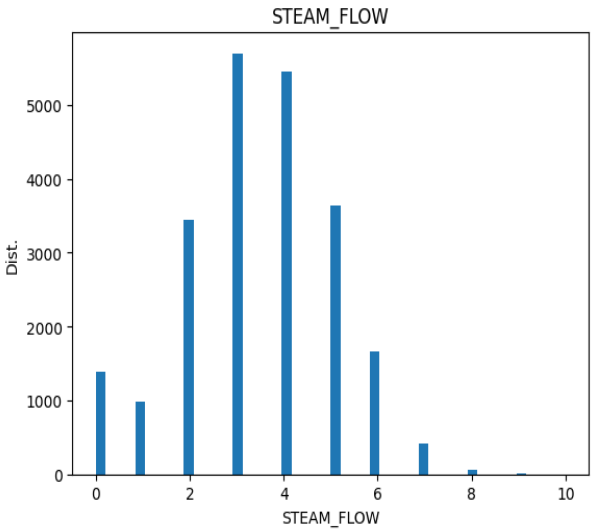
```
DATE_TIME          0
CB_FLOW         2633
CB_PRESS           0
CB_TEMP            0
STEAM_FLOW      2633
STEAM_TEMP         0
STEAM_PRESS        0
O2_PRESS           0
O2_FLOW         2633
O2_PER             0
PCI             2633
ATM_HUMID          0
HB_TEMP         3785
HB_PRESS        2714
TOP_PRESS       2633
TOP_SPRAY          0
TOP_TEMP           0
CO              2633
CO2             2633
H2              2633
SKIN_TEMP_AVG      0
```

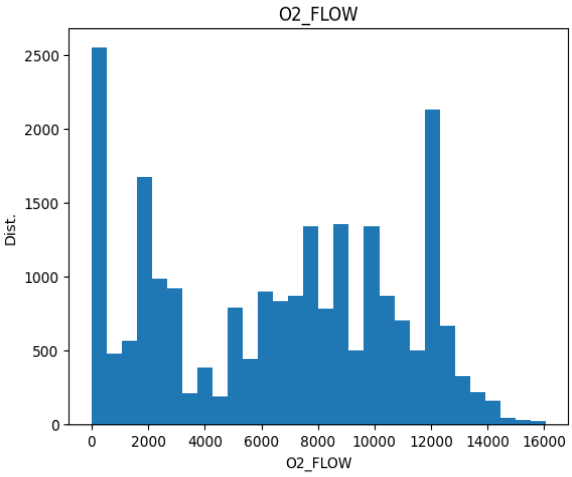
- CHECK DISTRIBUTION OF FEATURES

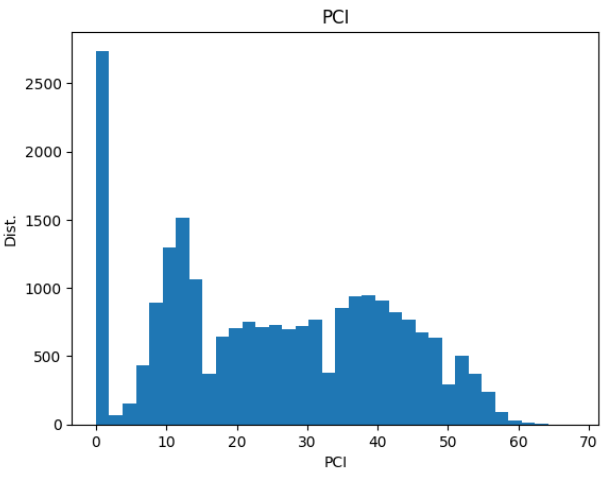
We used histogram with Freeman-Diaconis bins to observe the distribution of features with null values, to decide the fill strategy.

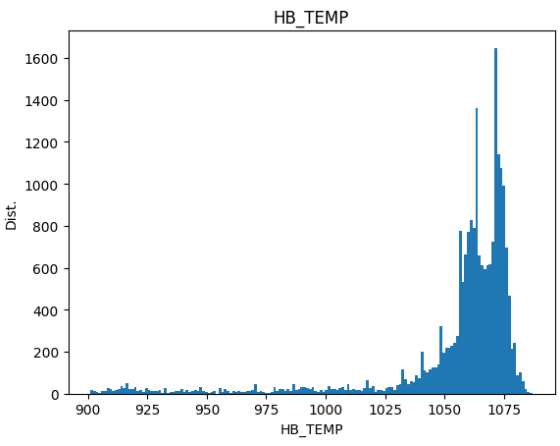
The distribution is as follows:

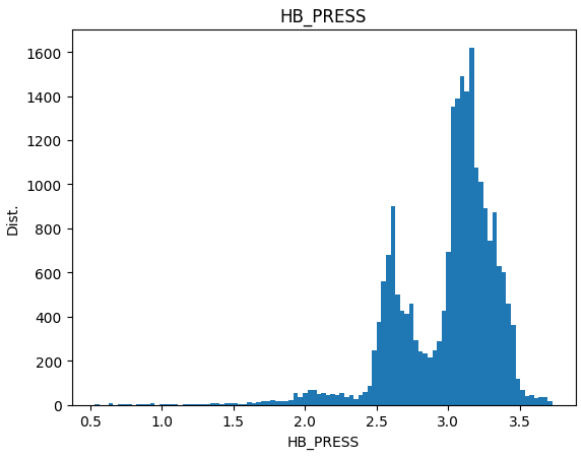

CB_FLOW
Dist.
CB_FLOW
0
200
400
600
800
1000
0
50000
100000
150000
200000
250000
300000
350000

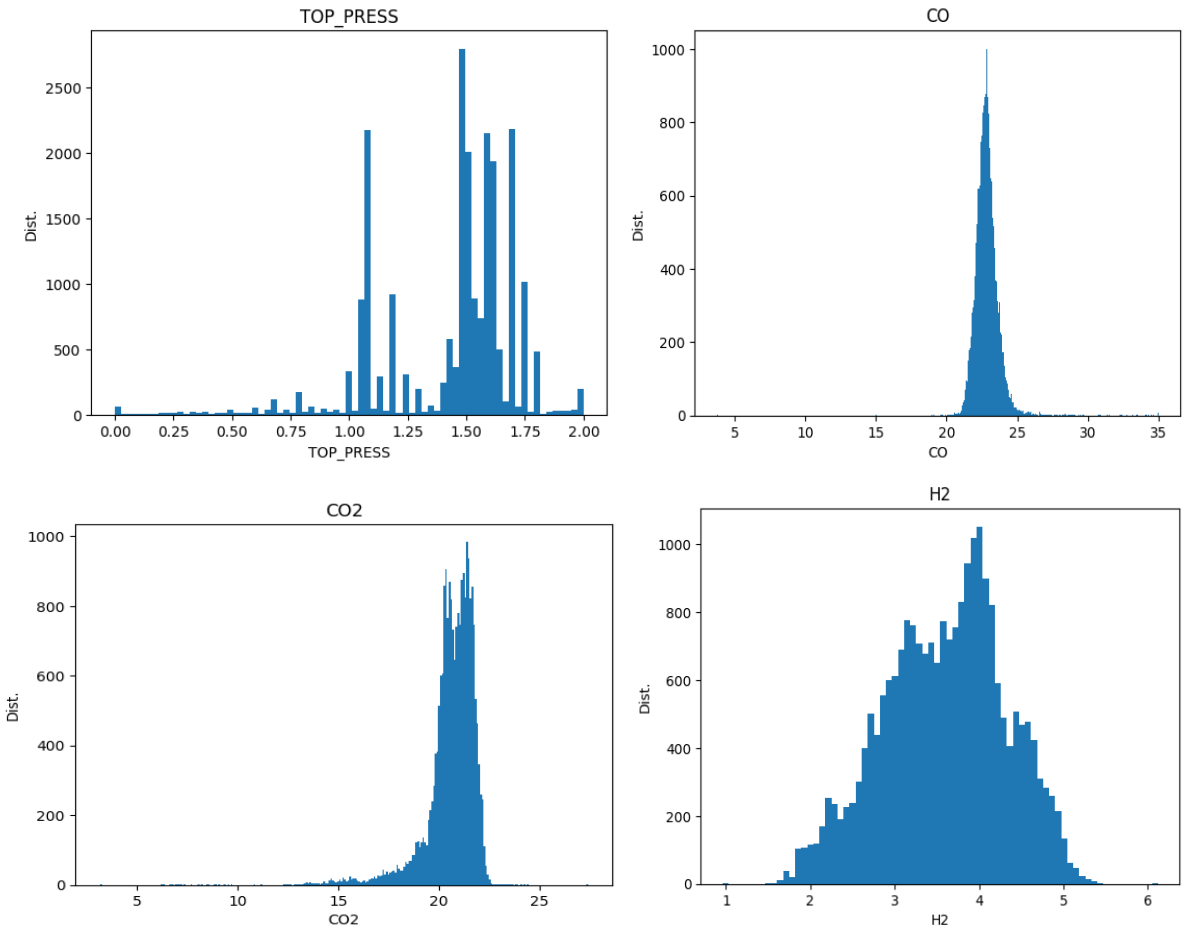

STEAM_FLOW
Dist.
STEAM_FLOW
0
1000
2000
3000
4000
5000
0
2
4
6
8
10


O2_FLOW
Dist.
O2_FLOW
0
500
1000
1500
2000
2500
0
2000
4000
6000
8000
10000
12000
14000
16000


PCI
Dist.
PCI
0
500
1000
1500
2000
2500
0
10
20
30
40
50
60
70


HB_TEMP
Dist.
HB_TEMP
0
200
400
600
800
1000
1200
1400
1600
900
925
950
975
1000
1025
1050
1075


HB_PRESS
Dist.
HB_PRESS
0
200
400
600
800
1000
1200
1400
1600
0.5
1.0
1.5
2.0
2.5
3.0
3.5



From the distribution, we observe the following:

```
CB_FLOW, HB_TEMP, CO(very mildly): skewed left
STEAM_FLOW (value lies from 1 to 2): discrete
O2_FLOW, PCI, CO2, HB_PRESS, TOP_PRESS: irregular
H2: (near) Gaussian
```

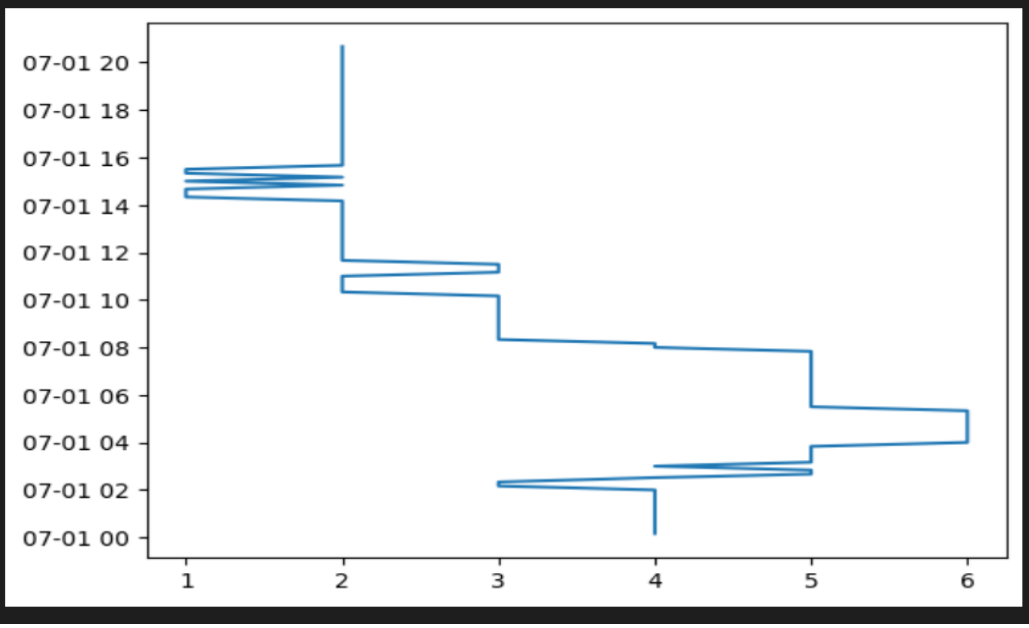
And from before, we observe that nearly 10% of values are missing in the feature columns listed above.

This means that, using statistical method to fill the data will bias the data and might skew the distribution.

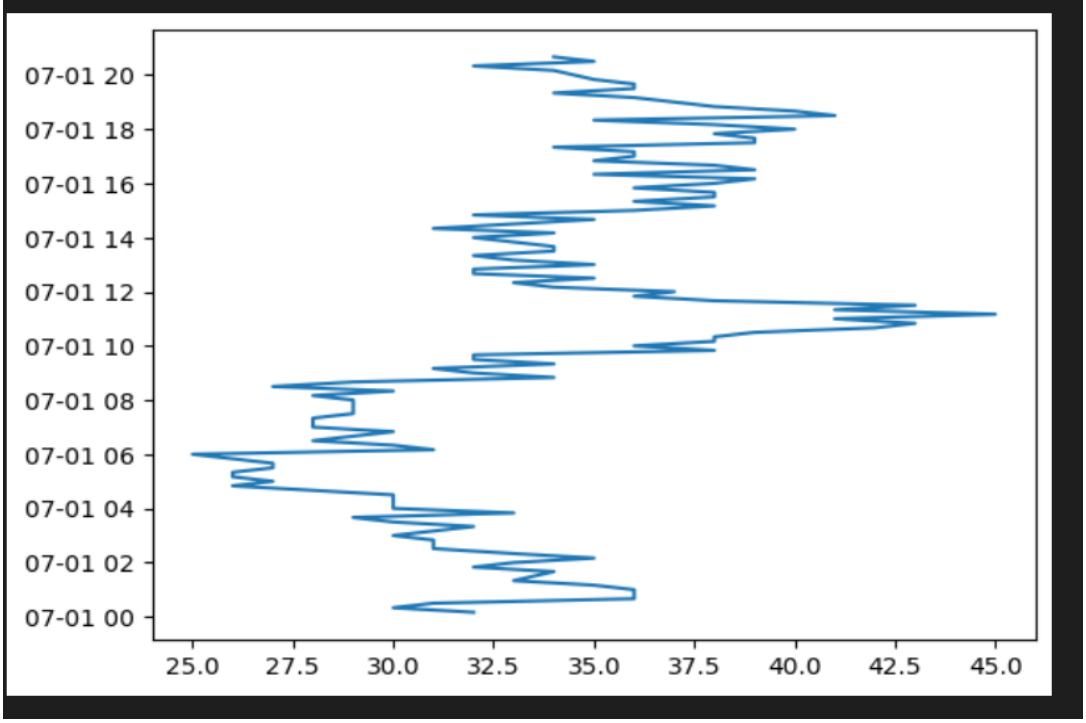
So, we cannot use SimpleImputer(). We will check if KNNImputer() would be useful in this situation.

To understand if KNNImputer() is a good choice, we plot a graph of feature values against time. It looks as follows:

```
plt.plot(df['STEAM_FLOW'][0:120], df['DATE_TIME'][0:120])
plt.show()
```




```
plt.plot(df['PCI'][0:120], df['DATE_TIME'][0:120])
plt.show()
```



Values are continuous. KNNImputer(), which would find the mean of n neighbour's values, would be a good fit. We put n_neighbors = 6, to replace the missing value with its hour's mean.

```
from sklearn.impute import KNNImputer
df_copy = df.copy(deep = True)
imputer = KNNImputer(n_neighbors = 6) # n_neighbors based on 1hr

df_copy.loc[:, [column for column in df_copy.columns if column != 'DATE_TIME']] =
imputer.fit_transform(df_copy.loc[:, [column for column in df_copy.columns if column
!= 'DATE_TIME']])

# Rounding off the values to keep the column values discrete in nature
df_copy.loc[:, ['STEAM_FLOW',]] = np.round(df_copy.loc[:, ['STEAM_FLOW',]])

df_copy.isna().sum()

df = df_copy.copy(deep = True)
```

This will keep the distribution intact even after imputing.

- ## DERIVE SKIN TEMPERATURE FOR 1, 2, 3, 4 hrs

The current dataset has rows set apart 10 minutes from each other. We need to forecast the skin temperature 1 hr apart till 4 hrs. So, we need to create 4 new columns (SKIN_TEMP_AVG_1, SKIN_TEMP_AVG_2, SKIN_TEMP_AVG_3, SKIN_TEMP_AVG_4).

Then, we shift SKIN_TEMP_AVG by 6, 12, 18, 24 hrs to SKIN_TEMP_AVG_1, SKIN_TEMP_AVG_2, SKIN_TEMP_AVG_3, SKIN_TEMP_AVG_4 respectively.

```python
feature_cols = ['SKIN_TEMP_AVG_1', 'SKIN_TEMP_AVG_2', 'SKIN_TEMP_AVG_3', 'SKIN_TEMP_AVG_4']

for i in range(1, 5):
    df[feature_cols[i-1]] = df['SKIN_TEMP_AVG'].shift(-6*i)
```

Python

```python
df.head(7)
```

Python

2_PRESS	O2_FLOW	O2_PER	...	TOP_SPRAY	TOP_TEMP	CO	CO2	H2	SKIN_TEMP_AVG	SKIN_TEMP_AVG_1	SKIN_TEMP_AVG_2	SKIN_TEMP_AVG_3	SKIN_TEMP_AVG_4
3.20	7296.0	23.08	...	0.0	121.0	22.22	21.00	3.88	69.940478	73.583364	77.198904	82.604995	84.877672
3.20	7829.0	23.08	...	0.0	125.0	22.56	21.00	3.94	71.454476	74.666066	78.518159	84.475989	83.397999
3.21	7904.0	23.08	...	0.0	124.0	22.49	21.08	3.94	70.579462	75.008361	80.865417	84.880888	83.368013
3.21	7919.0	23.08	...	0.0	115.0	22.36	21.13	3.99	70.179791	75.799102	82.564532	84.282448	84.592822
3.16	7938.0	23.08	...	0.0	125.0	22.25	21.30	4.10	70.728470	77.342288	82.636160	84.636354	83.540342
3.25	7916.0	23.08	...	0.0	128.0	22.30	21.10	4.12	72.222703	77.713731	82.332439	85.074844	82.441827
3.23	7925.0	23.08	...	0.0	129.0	22.25	21.00	4.09	73.583364	77.198904	82.604995	84.877672	81.730883

SELECTING PREDICTIVE MODEL

We will be using the following models:

1. **Multi-Linear Regressor**
2. **Polynomial Regressor**
3. **Random Forest Regressor (Ensemble)**
4. **Histogram Gradient Boosting Regressor (Ensemble)**
5. **KNN Regressor**

These models are available in sklearn (or scikit-learn) module in Python.

- ## Multi-Linear Regressor

Multi-linear regression is a statistical technique used to model the relationship between one dependent variable and two or more independent variables. It extends simple linear regression, which involves just one independent variable.

The general form of a multi-linear regression model is: $Y=\beta0+\beta1X1+\beta2X2+...+\beta pXp+\epsilon$ where:

- Y is the dependent variable.
- X1, X2, ..., Xp are the independent variables (predictors).
- $\beta0$ is the y-intercept.
- $\beta1, \beta2, ..., \beta p$ are the coefficients that represent the relationship between each independent variable and the dependent variable.
- ϵ is the error term, representing the variation in Y not explained by the predictors.

- ## Polynomial Regressor

Polynomial regression is a form of regression analysis in which the relationship between the independent variable X and the dependent variable Y is modeled as an n-th degree polynomial. It is an extension of linear regression that allows for modeling non-linear relationships.

The general form of a polynomial regression model of degree n is: $Y=\beta0+\beta1X+\beta2X2+\beta3X3+...+\beta nXn+\epsilon$ where:

- Y is the dependent variable.

- $X, X^2, X^3, \ldots, X^n$ are the independent variables raised to the power of 1, 2, 3, ..., n.
- $\beta 0, \beta 1, \beta 2, \beta 3, \ldots, \beta n$ are the coefficients.
- ϵ is the error term, representing the variation in Y not explained by the predictors.

If degree is 1, i.e., n=1, then Polynomial Regression regresses to multi-linear regression.

- ## Random Forest Regressor

Random Forest Regressor is an ensemble learning method used for regression tasks. It operates by constructing multiple decision trees during training and outputting the average prediction of the individual trees to improve accuracy and control over-fitting.

Components and Concepts:

1. **Decision Trees:**

   The basic building block of a random forest, where each tree makes a prediction based on the training data.

2. **Ensemble Learning:**

   Combines predictions from multiple models (decision trees) to produce a single, improved result.

3. **Bootstrap Aggregation (Bagging):**

Each tree in the forest is trained on a random subset of the training data with replacement. This helps in reducing variance.

4. **Random Feature Selection:**

   At each split in a tree, a random subset of features is considered for splitting, which helps in making the model more robust and less correlated.

- ## Histogram Gradient Boosting Regressor

Histogram Gradient Boosting Regressor (HGBR) is an advanced gradient boosting algorithm designed to handle large datasets efficiently. It improves on traditional gradient boosting methods by binning continuous features into discrete bins, which accelerates the training process and reduces memory usage.

How it Works:

1. **Initialization**:
   - Begin with an initial prediction (often the mean of the target values).

2. **Binning**:
   - Continuous features are binned into discrete bins. This reduces the computational burden by transforming continuous features into categorical ones.
   - Each feature is divided into a specified number of bins, and the values are replaced by their corresponding bin indices.

3. **Boosting**:
    - **Step 1**: Calculate the pseudo-residuals, which are the gradients of the loss function with respect to the predictions.
    - **Step 2**: Fit a decision tree (usually shallow) to the pseudo-residuals.
    - **Step 3**: Update the predictions by adding the weighted contribution of the new tree.
    - **Step 4**: Repeat the process for a specified number of iterations or until convergence.

4. **Prediction**:
    - The final prediction is obtained by summing the initial prediction and the contributions of all the trees in the ensemble.

- # KNN Regressor

K Neighbors Regressor is a type of instance-based learning or non-generalizing learning method in which the prediction for a new data point is made based on the target values of its k nearest neighbors in the feature space. It is a simple, yet effective, method for regression tasks.

Step 1: For a given test data point, calculate the distance between this point and all points in the training data. Common distance metrics

include Euclidean distance, Manhattan distance, and Minkowski distance.

Step 2: Identify the k training points that are closest to the test point.

Step 3: The prediction for the test point is typically the average of the target values of these k nearest neighbors.

TRAINING, CROSS-VALIDATION, TESTING

- **train_test_split():**

This method divides the dataset into two parts – training and testing.

The training dataset is solely used for training the model and testing is used to test the model's accuracy on unseen data.

This prevents overfitting, where the model, rather than generalizing the training dataset, remembers it.

```
from sklearn.model_selection import train_test_split, cross_validate, GridSearchCV
from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, R2_score, mean_squared_error
from numpy import inf

excluded_cols = feature_cols + ['DATE_TIME', 'SKIN_TEMP_AVG'] # Excluding the target
features and the date time column
X = df[[x for x in df.columns if x not in excluded_cols]]
y = df['SKIN_TEMP_AVG']

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

Also, the dataset is divided by input features (X) and target feature (y).

The train dataset is 80% of the total dataset whereas the test dataset is 20% of the total dataset.

- **Selecting a scaler:**

From the observed distribution of data, we can see that there are not of the same scale.

A model performs tremendously better, if all the input features are scaled to the same units.

As Gaussian distribution does not exist, we will be using MinMaxScaler() which scales the input features in the range [0, 1].

- **GridSearchCV():**

GridSearchCV() is an exhaustive hyperparameter-tuning method which searches through the given hyperparameters and cross-validates by dividing the dataset into multiple folds, training the model on one-fold and testing it on the remaining folds.

After the exhaustive search, it will return the best estimator along with scores.

This is computationally very exhaustive, so we used AWS SageMaker to perform GridSearchCV().

- **Pipeline:**

We use pipeline to streamline the model creation process. It automates the scaler-application and other data pre-processing.

We will perform GridSearchCV() by passing a pipeline as a parameter.

- **Random Forest Regressor:**

```
rf_param_grid = {
    'regressor__n_estimators' : [x for x in range(100, 600, 100)],
    'regressor__max_depth' : [None, 10, 20, 30, 40, 50],
    'regressor__max_features' : ['auto', 'sqrt', 'log2', 0.3, 0.6, 0.9],
    'regressor__min_samples_split' : [2, 5, 10],
    'regressor__min_samples_leaf' : [1, 2, 4],
}

rf_pipeline = Pipeline([('scaler', MinMaxScaler()), ('regressor',
RandomForestRegressor(random_state=42))])

rf_grid_search = GridSearchCV(rf_pipeline, rf_param_grid, cv = 5, scoring = ('R2',
'neg_mean_absolute_error', 'neg_mean_squared_error'), refit =
'neg_mean_squared_error', n_jobs = -1)

rf_grid_search.fit(X_train, y_train)

print(rf_grid_search.best_params_)
print(rf_grid_search.best_score_)
```

- **KNN Regressor:**

```
knn_params = {
    'regressor__n_neighbors' : [x for x in range(1, 11)],
    'regressor__weights' : ['uniform', 'distance'],
    'regressor__algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'regressor__leaf_size' : [x for x in range(10, 51, 10)]
}

knn_pipeline = Pipeline([('scaler', MinMaxScaler()), ('regressor',
KNeighborsRegressor())])
knn_grid_search = GridSearchCV(knn_pipeline, knn_params, cv = 5, scoring = ('R2',
'neg_mean_absolute_error', 'neg_mean_squared_error'), refit =
'neg_mean_squared_error', n_jobs = -1)
knn_grid_search.fit(X_train, y_train)

print(knn_grid_search.best_params_)
print(knn_grid_search.best_score_)
```

COMPARISON

MODEL	HYPERPARAMETERS-SCORE
Multi-linear Regressor	76% variance
Polynomial Regressor	Degree: 5 MAE = 7.968 R2 = 83.36% variance
Random Forest Regressor	No. of estimators = 900 Maximum features = 0.3 Max depth = 30 Score - 93.90% MAE - 4.559 MSE - 43.482
Histogram Gradient Boosting Regressor	loss = 'gamma' learning rate= 0.1 max leaf nodes = No limit max iter = 500 MAE -- 4.134 R2 -- 94.78%
KNeighbors Regressor	Neighbors = 2 Weights = Distance Algorithm = Auto Leaf size = 10 p = 1 Score - 97.049% MAE - 2.299 MSE - 21.038

All these results are after cross-validation and testing.

From these scores, we conclude that KNeighbor Regressor is highly accurate.

Create chained models:

By replacing the target variable by the next SKIN_TEMP column, we create models for predicting the skin temperature for 1, 2, 3, 4 hours respectively.

```
for i in range(1,5):
    X_nxt_hr = X.copy(deep = True)
    if i-1 != 0:
        X_nxt_hr.loc[:, 'SKIN_TEMP_AVG'] = df['SKIN_TEMP_AVG_' + str(i-1)]
    else:
        X_nxt_hr.loc[:, 'SKIN_TEMP_AVG'] = df['SKIN_TEMP_AVG']

    y_nxt_hr = df['SKIN_TEMP_AVG_' + str(i)]

    X_train, X_test, y_train, y_test = train_test_split(X_nxt_hr, y_nxt_hr,
test_size=0.2, random_state=42)

    print('R2: ' + str(R2_score(y_test, y_pred)))
    print('MAE: ' + str(mean_absolute_error(y_test, y_pred)))
    print('MSE: ' + str(mean_squared_error(y_test, y_pred)))
```

The following are the scores for all the models generated:

Skin Temp 1:

- **R2:** 0.96967
- **MAE:** 2.28521
- **MSE:** 21.37588

Skin Temp 2:

- **R2:** 0.97093
- **MAE:** 2.29158
- **MSE:** 21.36960

Skin Temp 3:

- **R2:** 0.97063
- **MAE:** 2.31440
- **MSE:** 21.69500

Skin Temp 4:

- **R2:** 0.97277
- **MAE:** 2.26228
- **MSE:** 20.26573

Deploying the Model

For model deployment, we utilized Flask, a Python web framework known for its simplicity and scalability. Here's a breakdown of how the model deployment was achieved:

Model Serialization:

The trained machine learning model was serialized using joblib, ensuring efficient storage and quick retrieval. This allows us to load the model instantly whenever predictions are required.

Flask Web Application:

Flask was chosen to build the web interface due to its lightweight nature and ease of development. The Flask application serves several routes:

- **Main Route ('/'):** Renders the main predictor interface (predictor.html), where users can input data for predictions.
- **About Page ('/about.html'):** Provides information about the project or application.
- **Credits Page ('/credits.html'):** Acknowledges contributors or sources.

Prediction Handling:

When users submit input data via a form, Flask processes the data using Pandas to create a DataFrame. The model then predicts the current skin temperature and forecasts the next four skin temperatures based on the input. These results are displayed on the results page (results.html).

```
from flask import Flask, render_template, redirect, request
import pandas as pd
import joblib

pred_model = joblib.load('knn_1hr_models.pk1')
features = joblib.load('df_cols.pkl')

app = Flask(__name__)

@app.route('/')
def index():
  return render_template('predictor.html')

@app.route('/about.html')
def about():
  return render_template('about.html')

@app.route('/credits.html')
def credits():
  return render_template('credits.html')

@app.route('/predictor.html')
def predictor_index():
  return redirect('/')

@app.route('/predict', methods=['POST'])
def predict():
  data = [float(x) for x in request.form.values()]
  df = pd.DataFrame([data], columns = features)

  skin_temp_nxt = pred_model[0].predict(df)
  skin_temps = [skin_temp_nxt[0], ]
  for i in range(1, 5):
```

```
        df_next = df.copy(deep = True)
        df_next.loc[:, 'SKIN_TEMP_AVG'] = skin_temp_nxt[0]
        skin_temp_nxt[0] = pred_model[i].predict(df_next)
        skin_temps.append(skin_temp_nxt[0])

    return render_template('results.html', skin_temp_curr = skin_temps[0], skin_temp_1 = skin_temps[1],
skin_temp_2 = skin_temps[2], skin_temp_3 = skin_temps[3], skin_temp_4 = skin_temps[4])

if __name__ == '__main__':
    app.run()
```

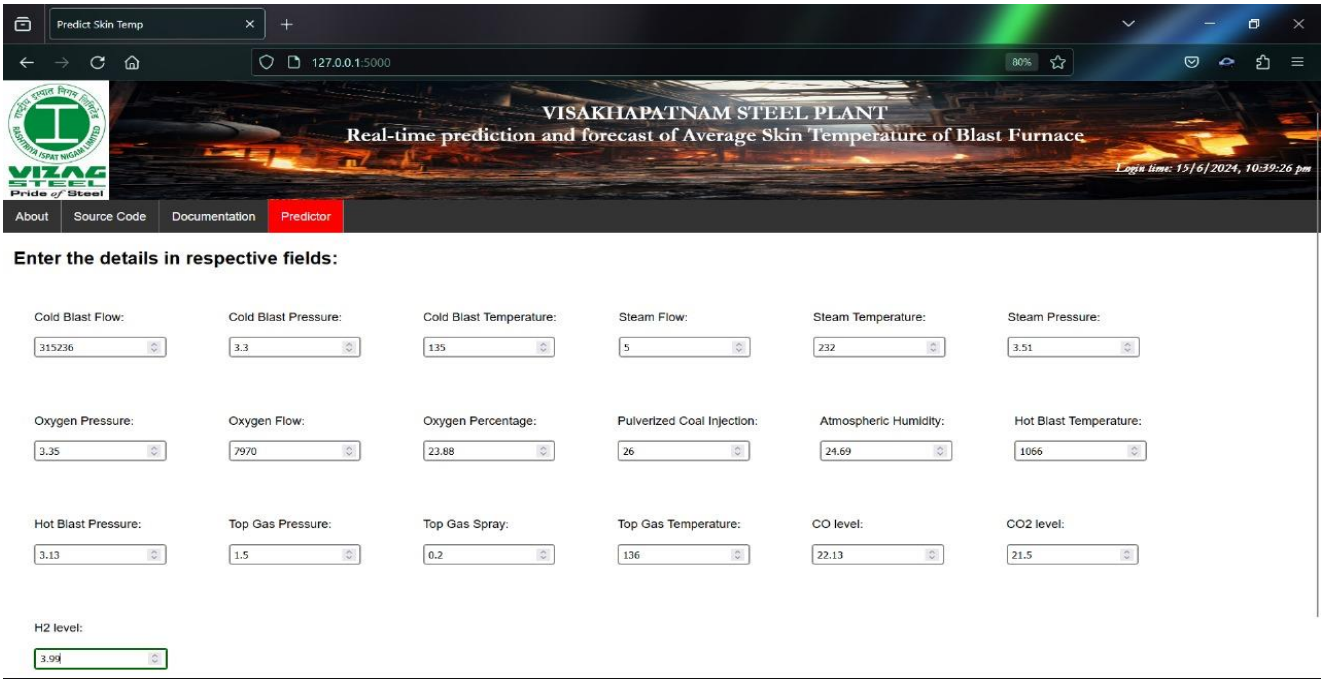
User Interaction:

Upon running the Flask server, users interact with the model through a user-friendly web interface. They input data relevant to the prediction task and receive immediate feedback in the form of predicted and forecasted values.

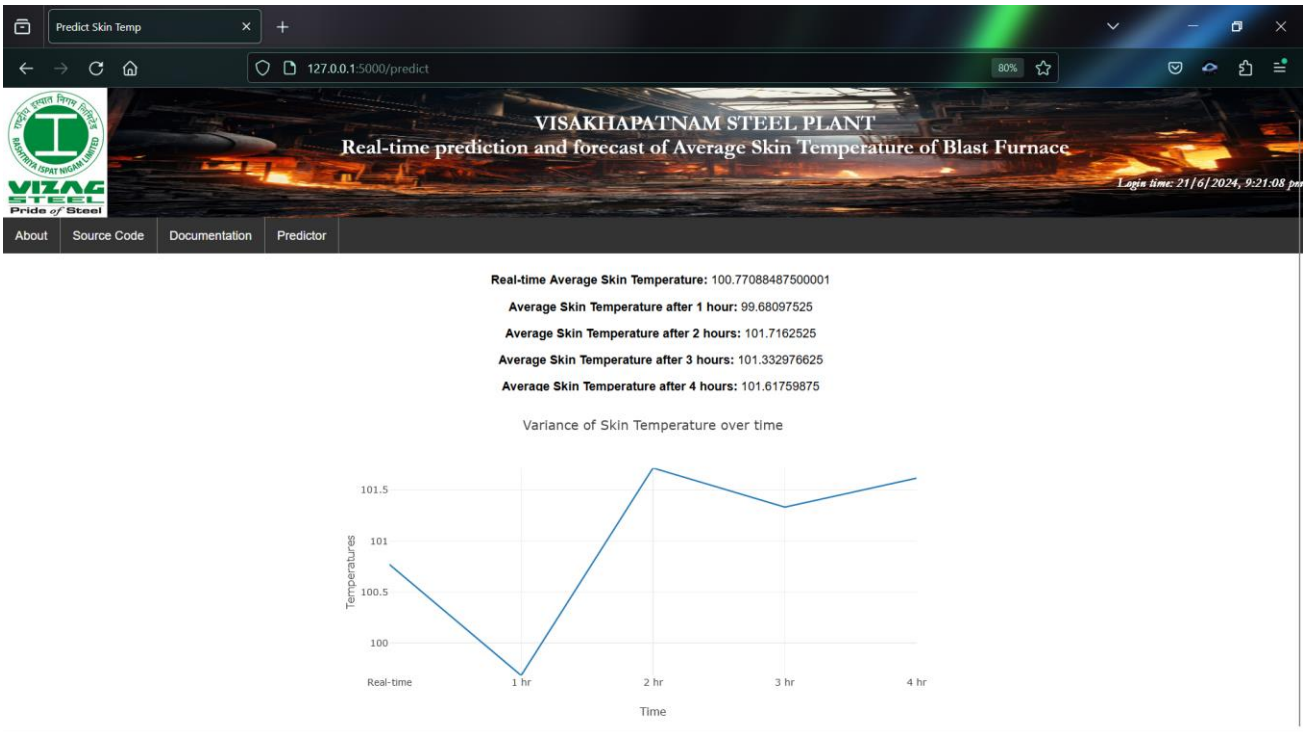
Conclusion:

This deployment setup integrates machine learning seamlessly into a web environment, providing real-time predictions and insights accessible via a standard web browser. The combination of Flask and joblib enables effective model deployment, demonstrating practical applications of machine learning in a web-based scenario.

INPUTTING VARIABLES



OUTPUT PREDICTIONS



CREDITS

M. Lalith Sri Sai	**7013031211**
V. Naga Koushik	**8074769900**
M. Sampath Akash	**6304389049**
M. Chandra Shekara Reddy	**8008581408**

PROJECT LINK

- https://github.com/TheStarSlayer/Prediction-and-Forecasting-Skin-Temperature-of-Blast-Furnace

SOURCES & REFERENCES

- VSCode - For Deployment
- HTML, CSS, JAVASCRIPT – For Website Deployment
- Python – Language Used
- ChatGPT
- Google, YouTube, W3Schools