Configuration Manual for Cloud-Based Time Series Prediction of CPU Load Through Advanced Deep Learning Models

MSc Research Project
Masters In Cloud Computing

Rohan Ajila
Student ID: 22249729

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rohan Ajila
Student ID:	22249729
Programme:	MSc. Cloud Computing **Year:** 2024-25
Module:	MSc. Research Project
Lecturer:	Prof. Vikas Sahni
Submission Due Date:	24 April 2025
Project Title:	Cloud-Based Time Series Prediction of CPU Load Through Advanced Deep Learning Models
Word Count:	1410 **Page Count: 20**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rohan Ajila

Date: 24 April 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Cloud-Based Time Series Prediction of CPU Load Through Advanced Deep Learning Models

Rohan Ajila
22249729

1 Introduction

This configuration manual aims to provide a step-by-step guide for setting up the cloud-native environment for the development and code implementation for the research project titled “Cloud-Based Time Series Prediction of CPU Load Through Advanced Deep Learning Models”. The main goal of the research is to forecast CPU utilization in Microsoft Azure virtual machines using historical time series data, which helps in more proactive resource scaling in cloud environments. The study compares and evaluates the efficiency of different traditional models like Decision Tree and Gradient Boosting to the deep learning models like LSTM, BiLSTM and Multihead BiLSTM for time series forecasting.
The cloud-based implementation in this project makes use of the Amazon Web Services (AWS) like EC2 for computational purposes, S3 for storage of data and cloud 9 as the integrated development environment. Time series data from azure log files is processed using Python tools and libraries and the performance metrics used to evaluate the performance of the models are Root Mean Squared Error (RMSE) and R^2 Score.
The following manual is presented as follows: Section 2 discusses the system specifications setup on the cloud for the implementation. Section 3 displays the software installation, AWS environment setup and the Python Libraries required. Section 4 shows the implementation and evaluation of the forecasting models. Section 5 provides the conclusions of the research and section 6 lists all the references used in this project.

2 Cloud Environment System Specification

2.1 Prerequisite:

The user should have a basic knowledge of python syntax and working flow of python > 3.10 or later version, and also the concepts like machine learning and deep learning. They should also be familiar with the EC2 instances, Cloud9, S3 Bucket.

2.2 Cloud 9 Setup:

Set up the AWS Cloud9 IDE on an EC2 instance for development and project Implementation (Jupyter Notebook).

STEP 1: Open the AWS management console and select the desired location.

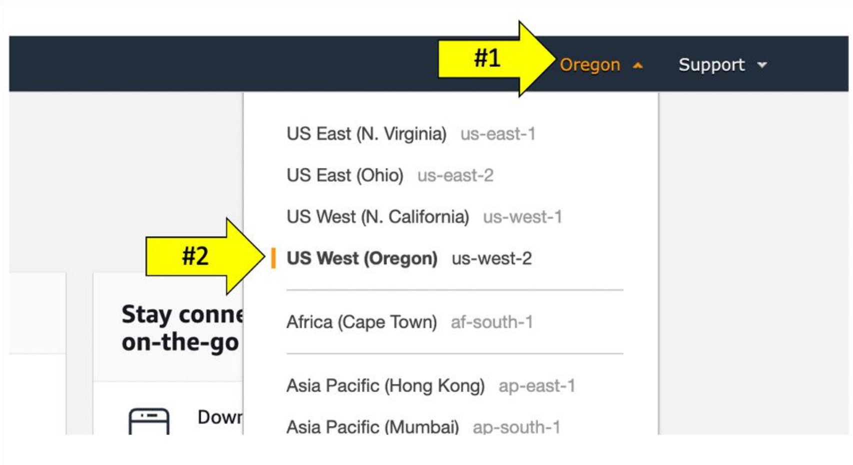


Figure 1 : Region selection on AWS Console

STEP 2: Find Cloud 9 in the search panel of the AWS Management Console

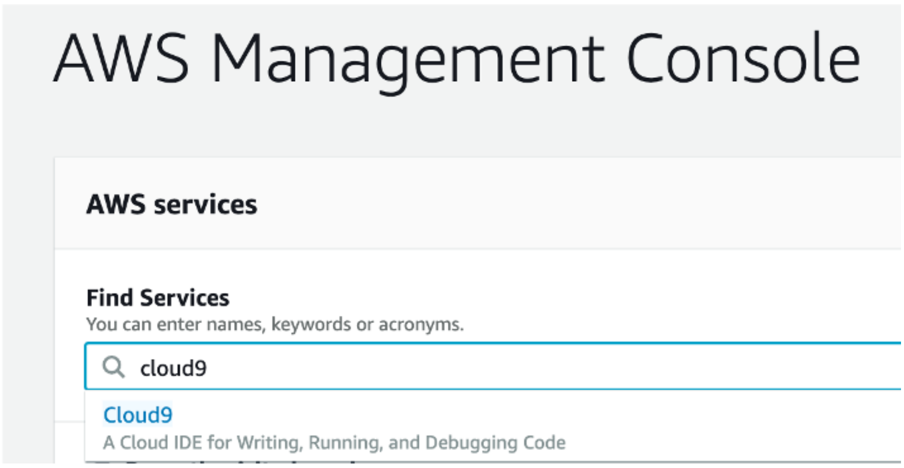


Figure 2 : Cloud 9 on AWS Console

STEP 3: Create an environment and setup the environment as shown in figure 3. Make sure to select create new EC2 instance for environment using direct access.

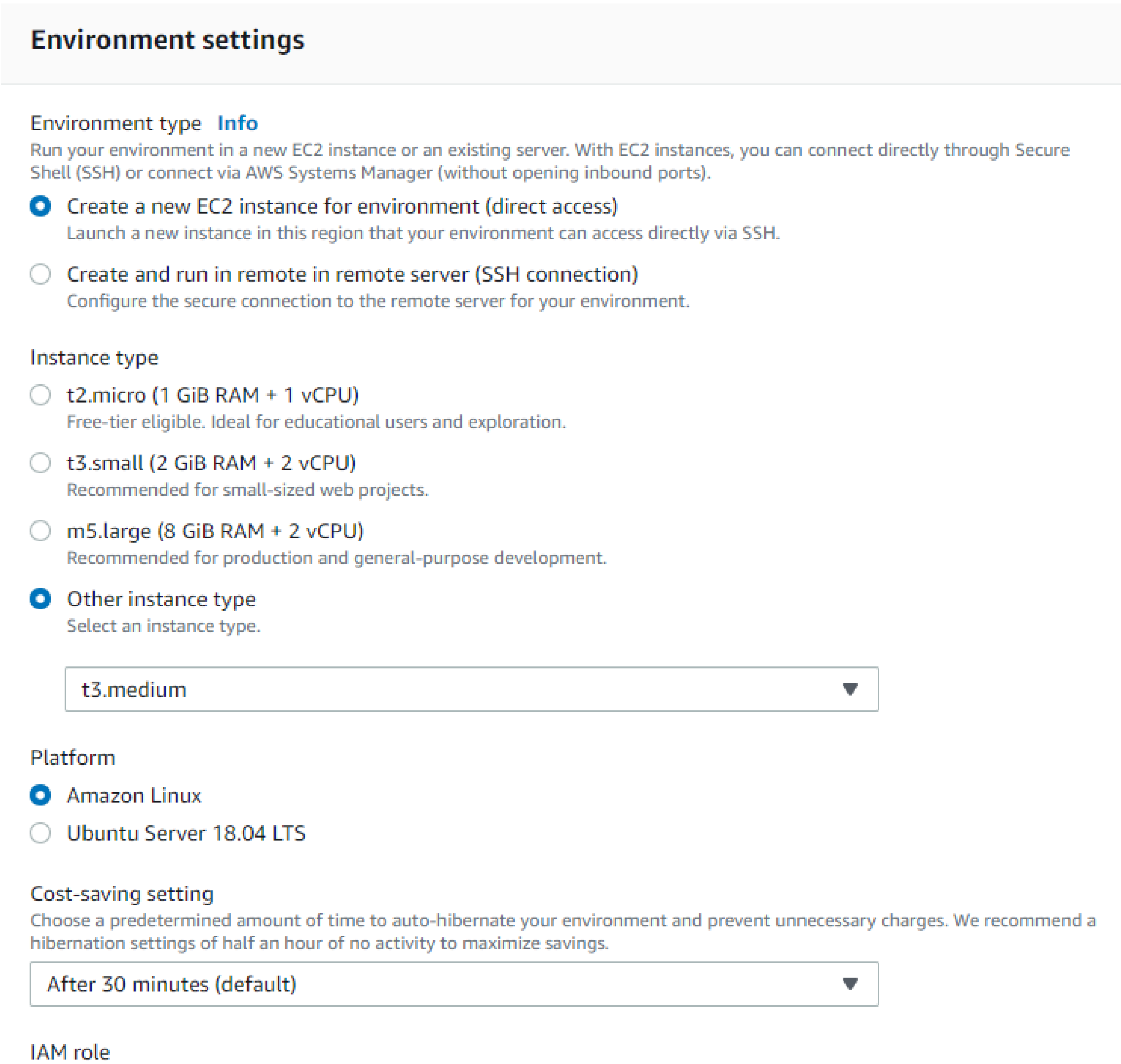


Figure 3 : Environment settings

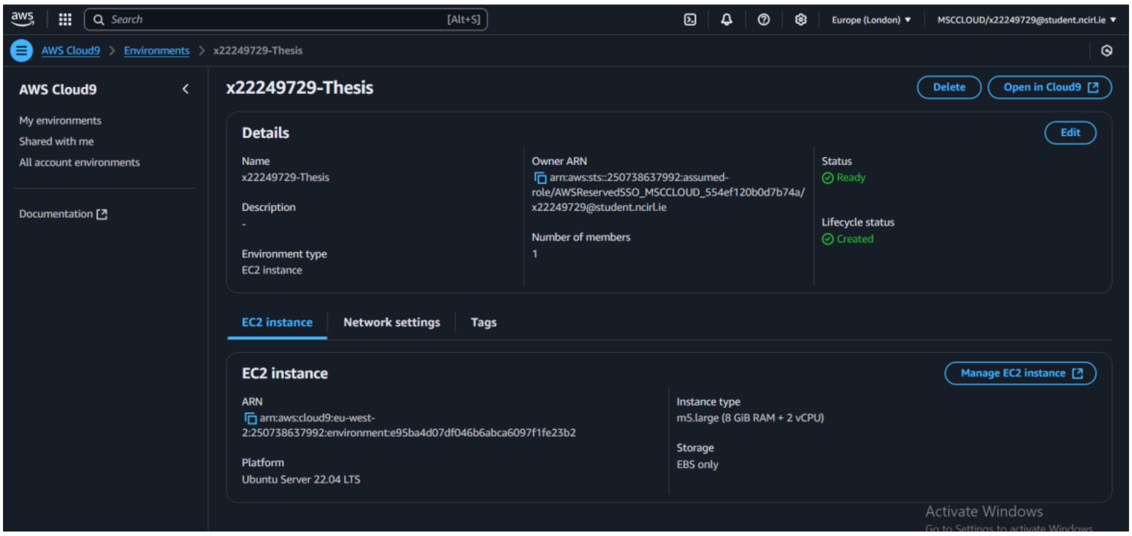


Figure 4: Cloud 9 environment setup

STEP 4: Cloud 9 environment is now setup as shown in Figure 4.

2.3 EC2 Configuration:

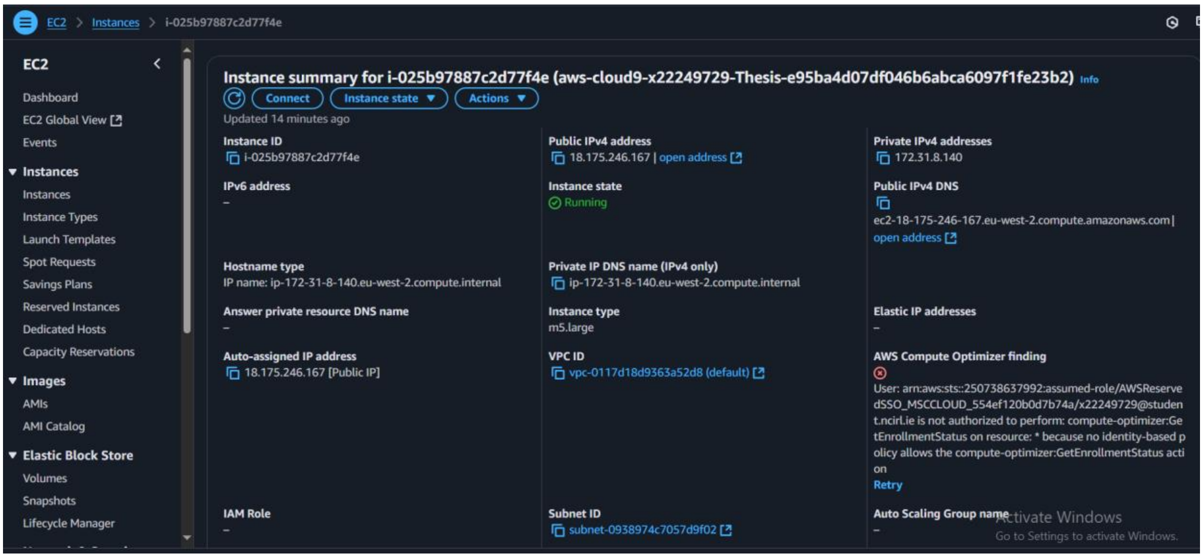


Figure 5 : EC2 instance connected to the Cloud 9 Environment

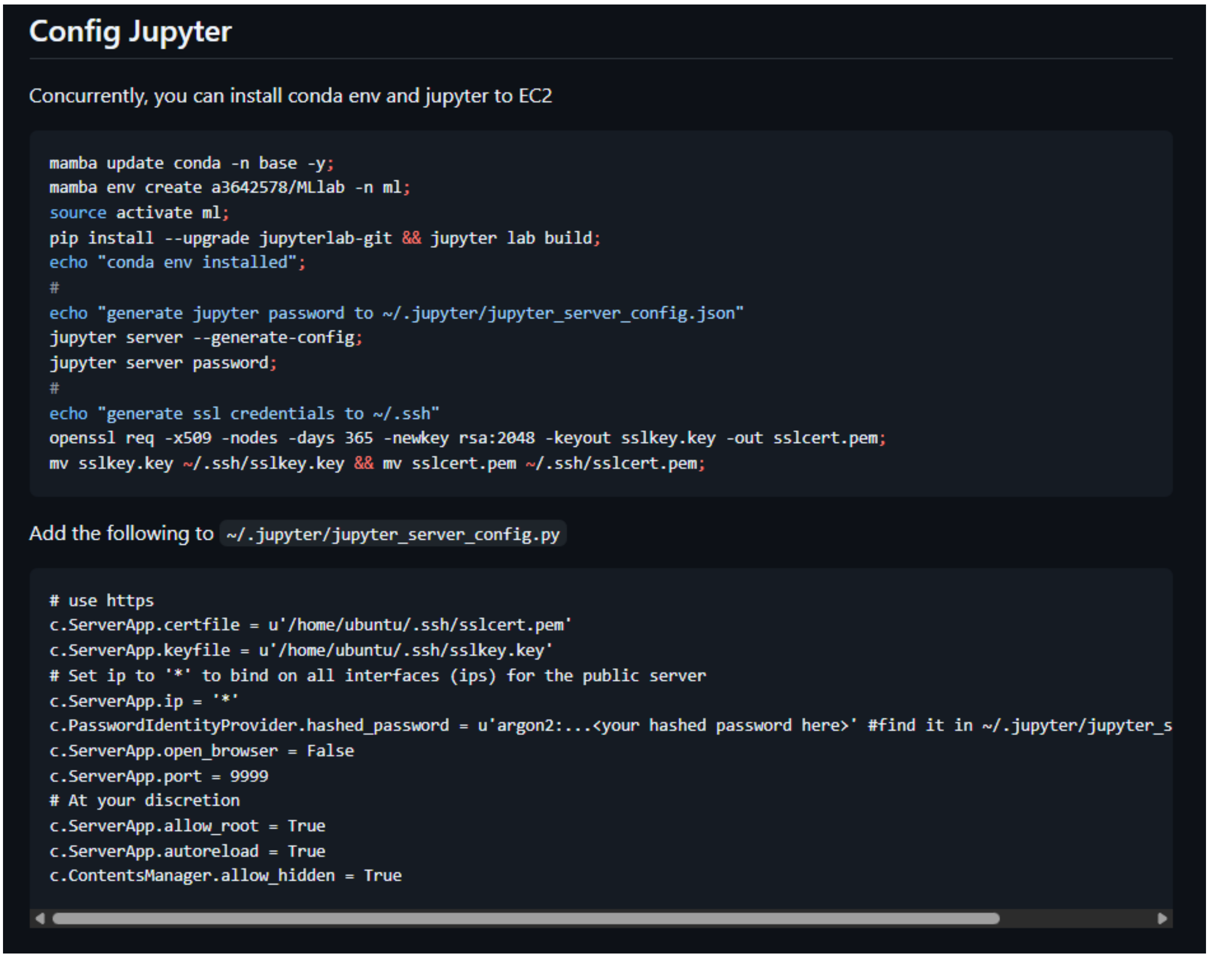


Figure 6 : Configure Jupyter notebook on EC2

Launch Jupyter from Cloud9

Set Cloud9 Project Preferrence `Stop my environemnt` to `never` Now launch jupyter from Cloud 9's terminal

Setup tunneling from local

Ubuntu / MacOS:

```
ssh -i private_key_to_ec2.pem -NfL localhost:9999:localhost:9999 ubuntu@ec2_host
```

Windows: Using MobaXterm

Figure 7 : Launch Jupyter notebook from Cloud 9

STEP 1: The EC2 instance needs to be setup to run Jupyter notebook, follow the figure 6 and 7 to setup to run Jupyter directly from cloud 9 IDE.

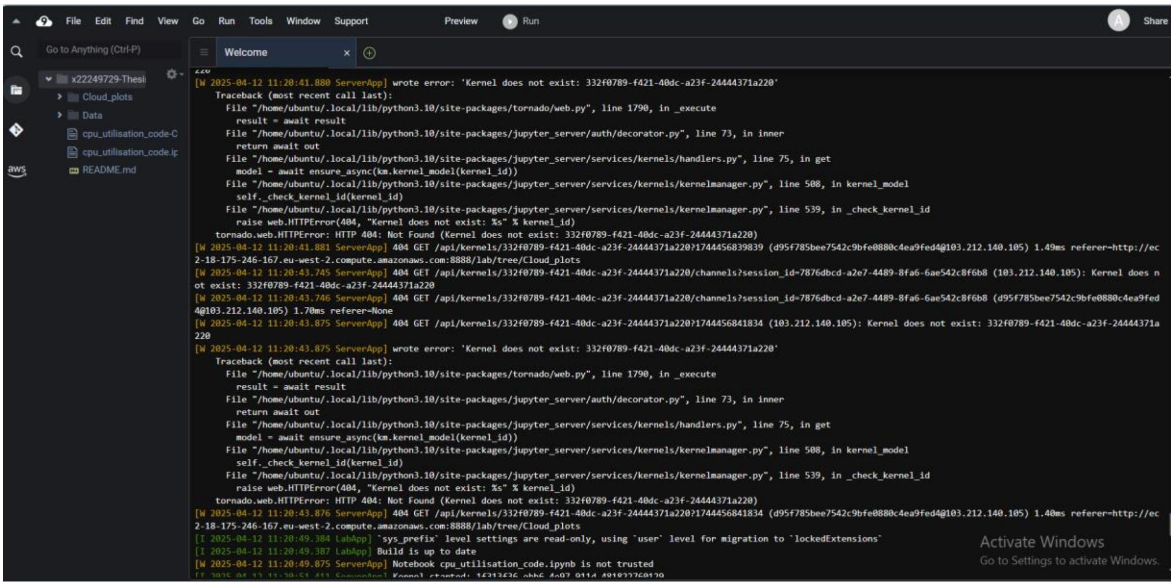


Figure 8: Running Jupyter from Cloud 9 IDE

2.4 S3 Storage Setup:

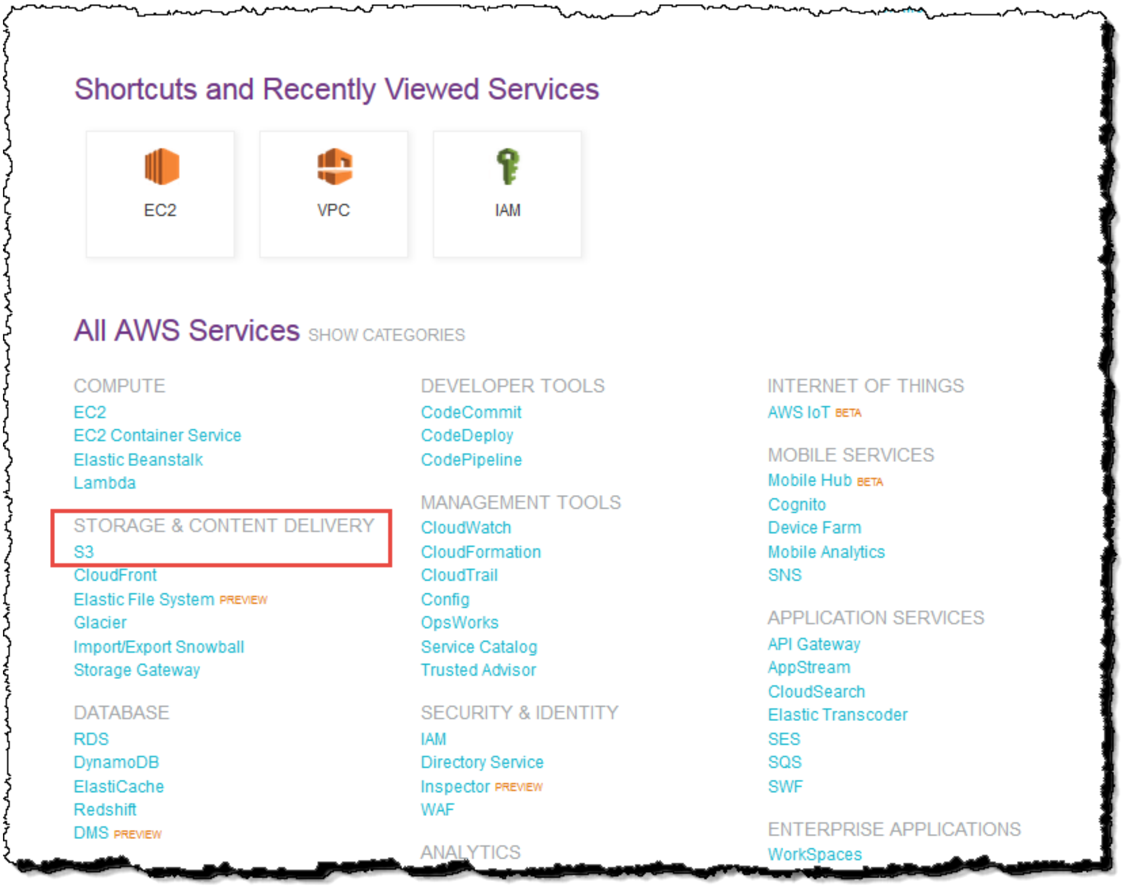


Figure 9 : AWS Storage and Content Deliver under Console

STEP 1: Find S3 under the Storage and content delivery menu in the AWS management console.

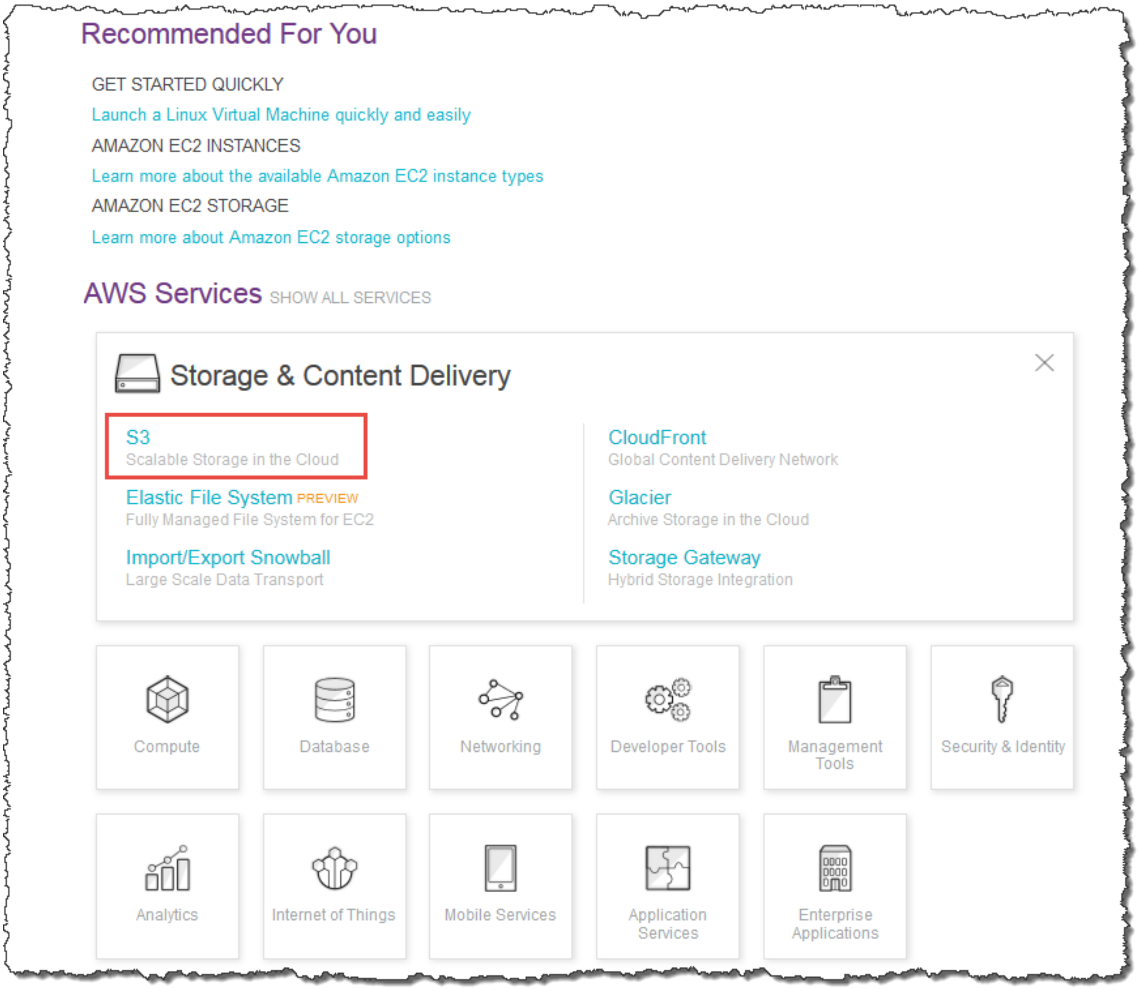


Figure 10: Select S3

STEP 2: Select S3

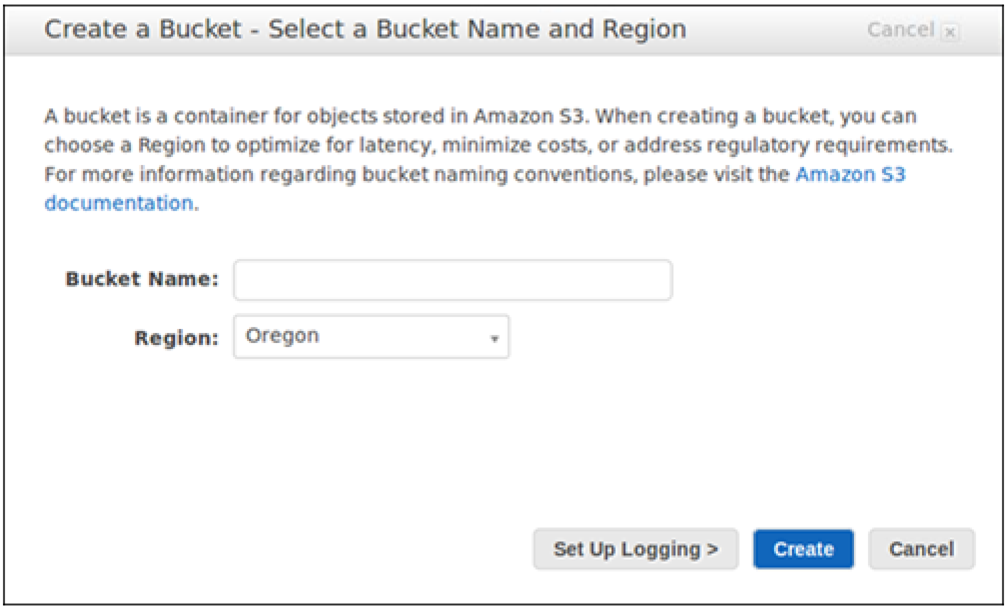


Figure 11 : Bucket Creation

STEP 3: Add the name of the bucket and select a region and click on create bucket.

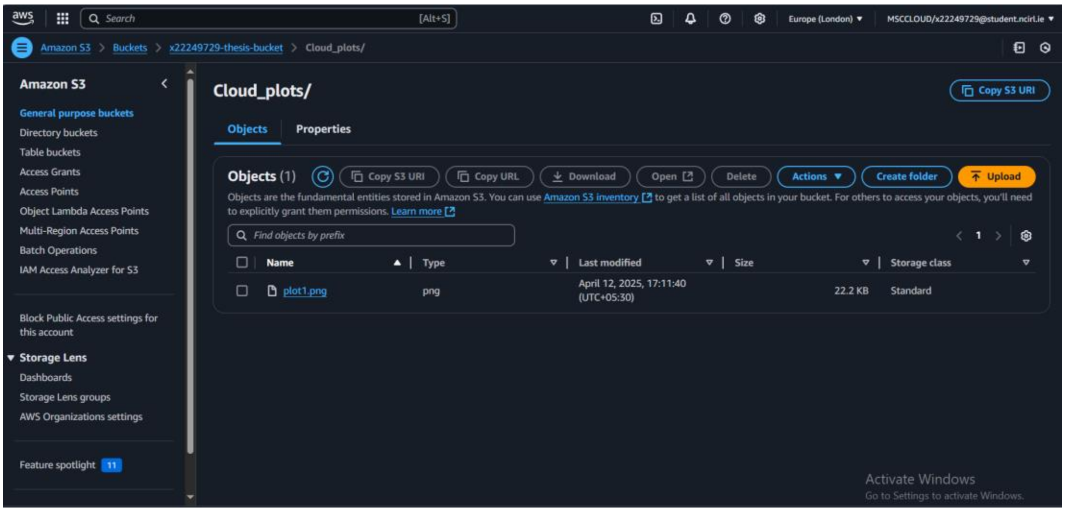


Figure 12: S3 bucket

3 Installing and Importing Python Libraries

3.1 Libraries and Packages

Boto3: An AWS SDK for Python used to interact with AWS services like S3 for cloud storage, enabling seamless data retrieval and management in cloud environments.

Plotly (express and graph objects): A powerful visualization library for creating interactive and customizable charts, graphs, and dashboards, perfect for exploring and presenting complex datasets.

Seaborn: A Python library for advanced statistical data visualization, offering aesthetic and informative charts to enhance exploratory data analysis (EDA).

TensorFlow & Keras: Components for building deep learning architectures such as LSTMs, Bidirectional LSTMs, and Cross-Attention mechanisms, essential for time-series and sequence modeling tasks.

Scikit-learn: Provides essential tools like Min-Max Scaler for data normalization and metrics like MSE and R^2 for evaluating model performance.

3.2 Installing and Importing libraries

Open Jupyter Notebook from Cloud 9 IDE and install and import all the libraries as shown in Figure 13.

Import All Libraries

```
import warnings
warnings.filterwarnings('ignore')
```

```
import glob
import boto3
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import plotly.express as px
import plotly.offline as py
from sklearn import metrics
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from tensorflow.keras import backend as K
from sklearn.preprocessing import MinMaxScaler
from botocore.exceptions import NoCredentialsError
from tensorflow.keras.models import Sequential, Model
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from tensorflow.keras.layers import LSTM, Dense, Dropout,Bidirectional, Input, Permute, Multiply, Flatten
```

```
2024-12-07 06:36:27.387217: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-12-07 06:36:27.392376: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-12-07 06:36:27.405645: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to register facto
ry for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1733553387.428697   15160 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has al
ready been registered
E0000 00:00:1733553387.435231   15160 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2024-12-07 06:36:27.460695: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in
```

Figure 13: Importing Libraries

4 Implementation and Evaluation

4.1 Data Loading

Data Loading

```
[4]: paths="materna_dataset/"

     #list of all files
     all_files = glob.glob(paths + "/*.csv")

     datavalues = []
     for filename in all_files[:150]: #loading 150 file depending upon resource
         data = pd.read_csv(filename, delimiter=';', index_col=None, header=0)
         datavalues.append(data)

     #concatenating the dataframe
     dataframe = pd.concat(datavalues, axis=0, ignore_index=True)
```

```
[5]: dataframe.head(10)
```

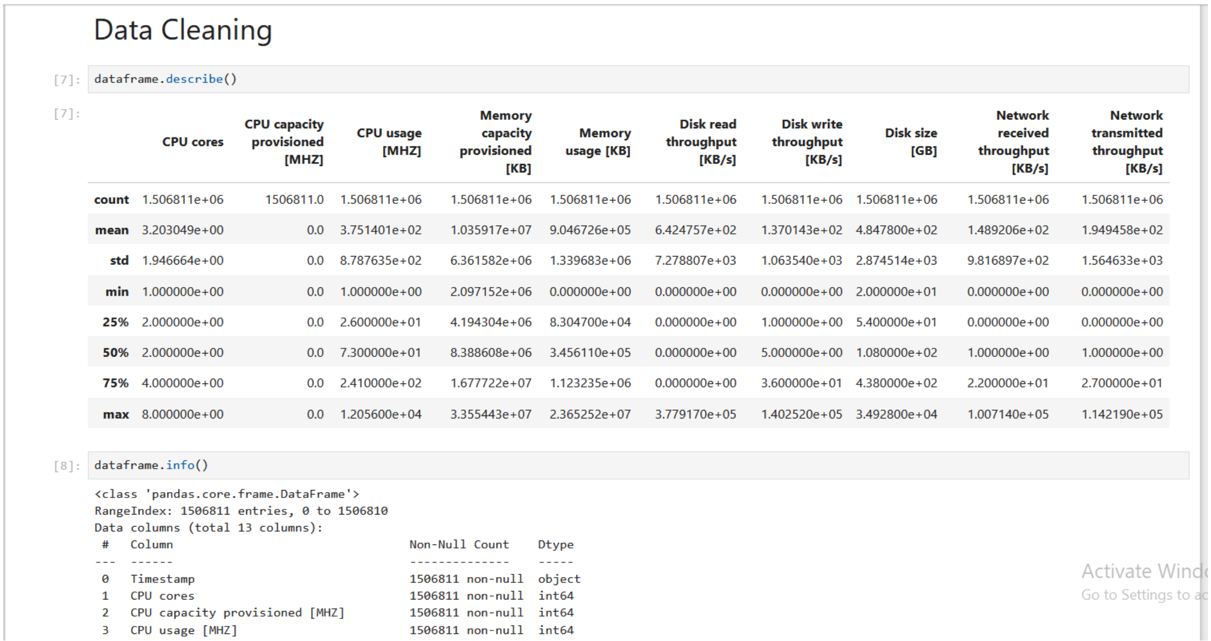
[5]:

	Timestamp	CPU cores	CPU capacity provisioned [MHZ]	CPU usage [MHZ]	CPU usage [%]	Memory capacity provisioned [KB]	Memory usage [KB]	Memory usage [%]	Disk read throughput [KB/s]	Disk write throughput [KB/s]	Disk size [GB]	Network received throughput [KB/s]	Network transmitted throughput [KB/s]
0	04.01.2016 00:00:00	2	0	115	2,51	4194304	122474	2,92	0	13	54	1	1
1	04.01.2016 00:05:00	2	0	106	2,32	4194304	119538	2,85	0	13	54	1	1
2	04.01.2016 00:10:00	2	0	119	2,61	4194304	329253	7,85	0	70	54	2	35
3	04.01.2016 00:15:00	2	0	121	2,63	4194304	284793	6,79	0	13	54	11	35

Figure 14: Loading 150 csv files using Pandas

4.2 Data Cleaning

Data Cleaning - Clean the data by addressing missing values, removing unnecessary columns, and fixing any other in consistencies.



Data Cleaning

[7]:
```
dataframe.describe()
```

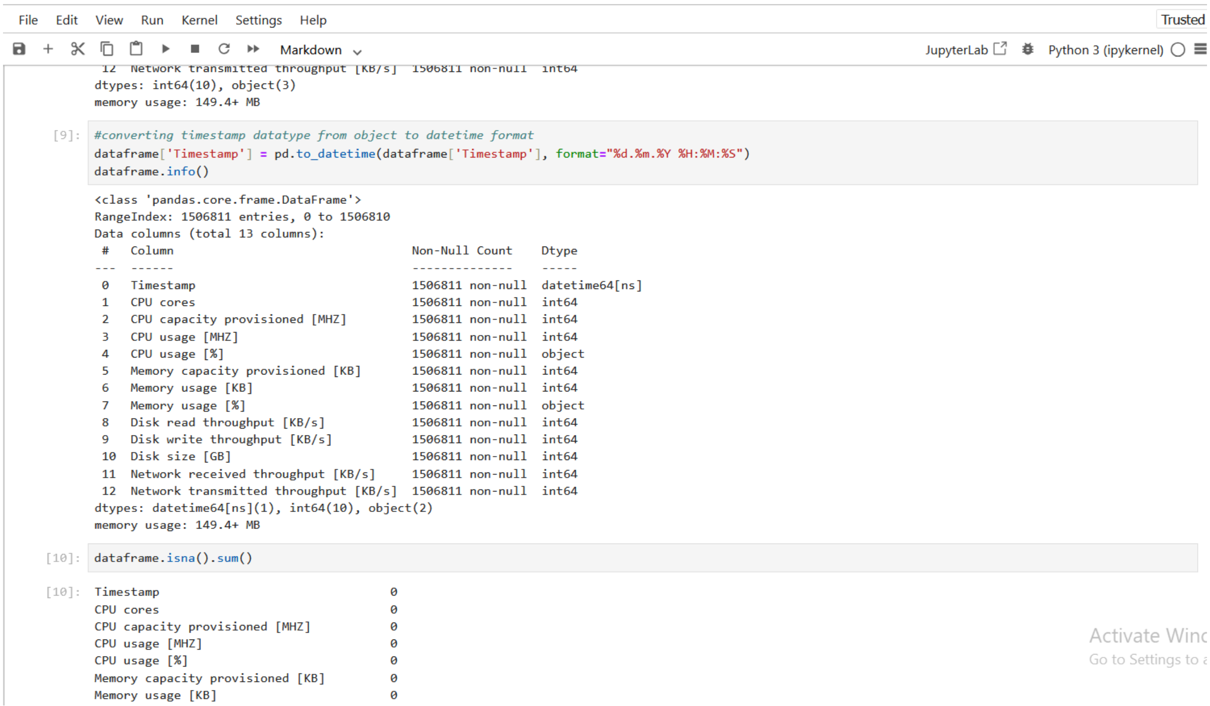
[7]:

	CPU cores	CPU capacity provisioned [MHZ]	CPU usage [MHZ]	Memory capacity provisioned [KB]	Memory usage [KB]	Disk read throughput [KB/s]	Disk write throughput [KB/s]	Disk size [GB]	Network received throughput [KB/s]	Network transmitted throughput [KB/s]
count	1.506811e+06	1506811.0	1.506811e+06	1.506811e+06	1.506811e+06	1.506811e+06	1.506811e+06	1.506811e+06	1.506811e+06	1.506811e+06
mean	3.203049e+00	0.0	3.751401e+02	1.035917e+07	9.046726e+05	6.424757e+02	1.370143e+02	4.847800e+02	1.489206e+02	1.949458e+02
std	1.946664e+00	0.0	8.787635e+02	6.361582e+06	1.339683e+06	7.278807e+03	1.063540e+03	2.874514e+03	9.816897e+02	1.564633e+03
min	1.000000e+00	0.0	1.000000e+00	2.097152e+06	0.000000e+00	0.000000e+00	0.000000e+00	2.000000e+01	0.000000e+00	0.000000e+00
25%	2.000000e+00	0.0	2.600000e+01	4.194304e+06	8.304700e+04	0.000000e+00	1.000000e+00	5.400000e+01	0.000000e+00	0.000000e+00
50%	2.000000e+00	0.0	7.300000e+01	8.388608e+06	3.456110e+05	0.000000e+00	5.000000e+00	1.080000e+02	1.000000e+00	1.000000e+00
75%	4.000000e+00	0.0	2.410000e+02	1.677722e+07	1.123235e+06	0.000000e+00	3.600000e+01	4.380000e+02	2.200000e+01	2.700000e+01
max	8.000000e+00	0.0	1.205600e+04	3.355443e+07	2.365252e+07	3.779170e+05	1.402520e+05	3.492800e+04	1.007140e+05	1.142190e+05

[8]:
```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1506811 entries, 0 to 1506810
Data columns (total 13 columns):
 #   Column                          Non-Null Count    Dtype
---  ------                          --------------    -----
 0   Timestamp                       1506811 non-null  object
 1   CPU cores                       1506811 non-null  int64
 2   CPU capacity provisioned [MHZ]  1506811 non-null  int64
 3   CPU usage [MHZ]                 1506811 non-null  int64
```


Figure 15: Checking data inconsistency




```
 12  Network transmitted throughput [KB/s]  1506811 non-null  int64
dtypes: int64(10), object(3)
memory usage: 149.4+ MB
```

[9]:
```
#converting timestamp datatype from object to datetime format
dataframe['Timestamp'] = pd.to_datetime(dataframe['Timestamp'], format="%d.%m.%Y %H:%M:%S")
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1506811 entries, 0 to 1506810
Data columns (total 13 columns):
 #   Column                                 Non-Null Count    Dtype
---  ------                                 --------------    -----
 0   Timestamp                              1506811 non-null  datetime64[ns]
 1   CPU cores                              1506811 non-null  int64
 2   CPU capacity provisioned [MHZ]         1506811 non-null  int64
 3   CPU usage [MHZ]                        1506811 non-null  int64
 4   CPU usage [%]                          1506811 non-null  object
 5   Memory capacity provisioned [KB]       1506811 non-null  int64
 6   Memory usage [KB]                      1506811 non-null  int64
 7   Memory usage [%]                       1506811 non-null  object
 8   Disk read throughput [KB/s]            1506811 non-null  int64
 9   Disk write throughput [KB/s]           1506811 non-null  int64
 10  Disk size [GB]                         1506811 non-null  int64
 11  Network received throughput [KB/s]     1506811 non-null  int64
 12  Network transmitted throughput [KB/s]  1506811 non-null  int64
dtypes: datetime64[ns](1), int64(10), object(2)
memory usage: 149.4+ MB
```

[10]:
```
dataframe.isna().sum()
```

[10]:
```
Timestamp                           0
CPU cores                           0
CPU capacity provisioned [MHZ]      0
CPU usage [MHZ]                     0
CPU usage [%]                       0
Memory capacity provisioned [KB]    0
Memory usage [KB]                   0
```


Figure 16: Checking Null Values

4.3 Data Preprocessing and Feature Extraction

Process data and extract meaningful features which include datetime conversion of date columns, extracting month, year from date for visualization.

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

```
dtype: int64
```

Feature Extraction from Date

```
[11]: dataframe['Hour'] = dataframe['Timestamp'].dt.hour
dataframe['Minute'] = dataframe['Timestamp'].dt.minute
dataframe['Year'] = dataframe['Timestamp'].dt.year
dataframe['Month'] = dataframe['Timestamp'].dt.month
dataframe['Day'] = dataframe['Timestamp'].dt.day
dataframe.head()
```

[11]:

	Timestamp	CPU cores	CPU capacity provisioned [MHZ]	CPU usage [MHZ]	CPU usage [%]	Memory capacity provisioned [KB]	Memory usage [KB]	Memory usage [%]	Disk read throughput [KB/s]	Disk write throughput [KB/s]	Disk size [GB]	Network received throughput [KB/s]	Network transmitted throughput [KB/s]	Hour	Minute	Year	Mon
0	2016-01-04 00:00:00	2	0	115	2,51	4194304	122474	2,92	0	13	54	1	1	0	0	2016	
1	2016-01-04 00:05:00	2	0	106	2,32	4194304	119538	2,85	0	13	54	1	1	0	5	2016	
2	2016-01-04 00:10:00	2	0	119	2,61	4194304	329253	7,85	0	70	54	2	35	0	10	2016	
3	2016-01-04 00:15:00	2	0	121	2,63	4194304	284793	6,79	0	13	54	11	35	0	15	2016	
4	2016-01-04 00:20:00	2	0	112	2,46	4194304	348966	8,32	0	10	54	1	1	0	20	2016	

Figure 17: Feature extraction from Date

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

Data Preprocessing for Time Series Forecasting

```
[19]: final_df = dataframe.set_index('Timestamp')
final_df = final_df[['CPU usage [MHZ]']]
final_df.sort_index(inplace=True)
final_df = final_df['CPU usage [MHZ]'].resample('5T').mean()  ##single timestamp have mutlipe record handling using resample
```

```
[20]: final_df
```

```
[20]: Timestamp
2016-01-04 00:00:00    338.049296
2016-01-04 00:05:00    357.232394
2016-01-04 00:10:00    331.542254
2016-01-04 00:15:00    336.000000
2016-01-04 00:20:00    363.676056
                          ...
2016-02-08 23:35:00    350.086667
2016-02-08 23:40:00    337.953333
2016-02-08 23:45:00    345.080000
2016-02-08 23:50:00    342.833333
2016-02-08 23:55:00    338.160000
Freq: 5min, Name: CPU usage [MHZ], Length: 10368, dtype: float64
```

Figure 18: Preparing data for Forecasting

4.4 Data Visualization

Visualize the data to identify patterns, trends, and relationships between variables.

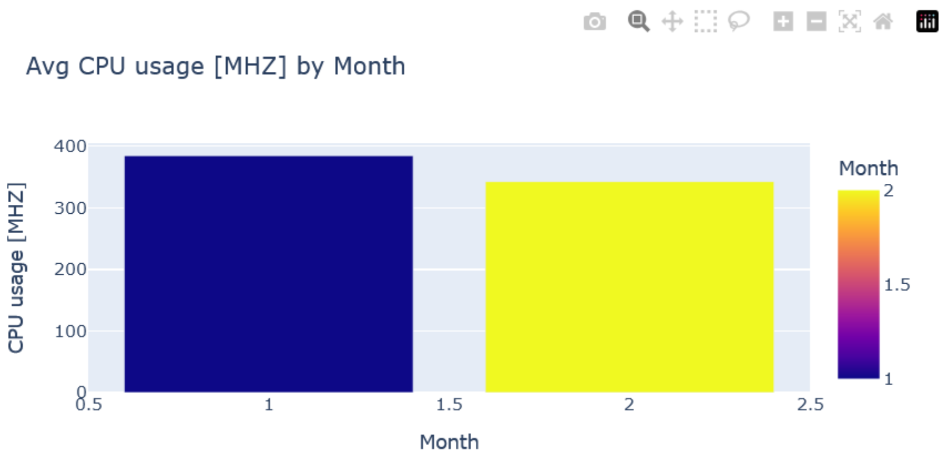


Figure 19: Avg CPU Usage [MHz] by Month

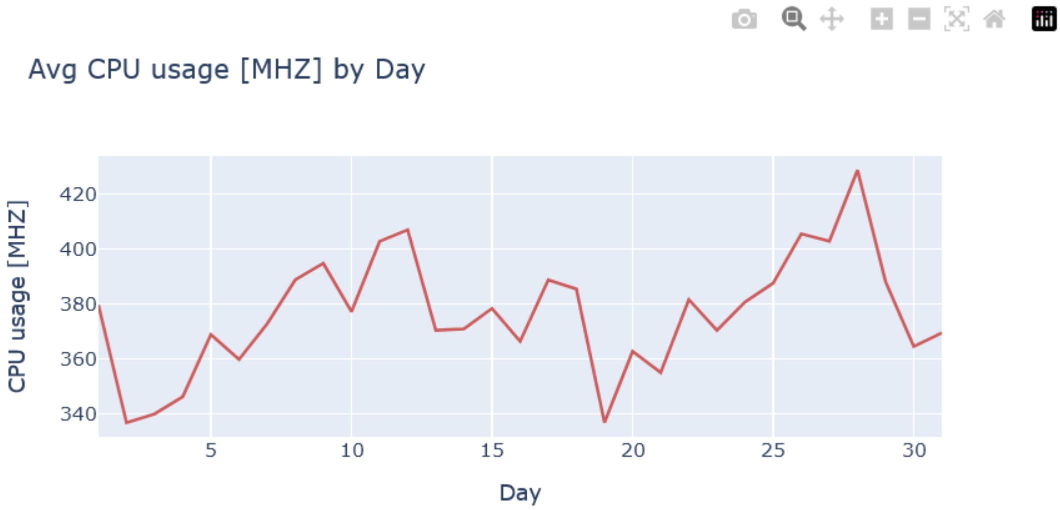


Figure 20: Avg CPU Usage [MHz] by Day

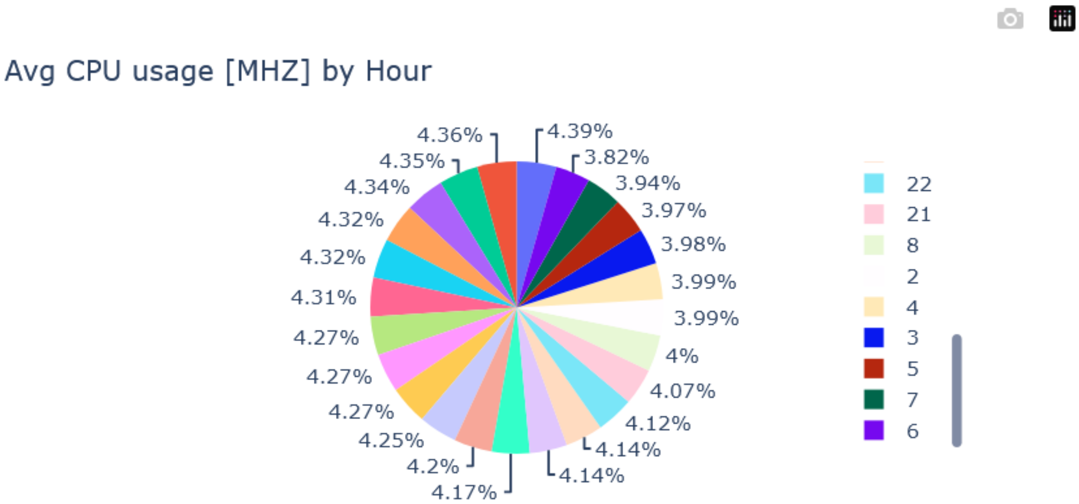


Figure 21: Avg CPU Usage [MHz] by Hour

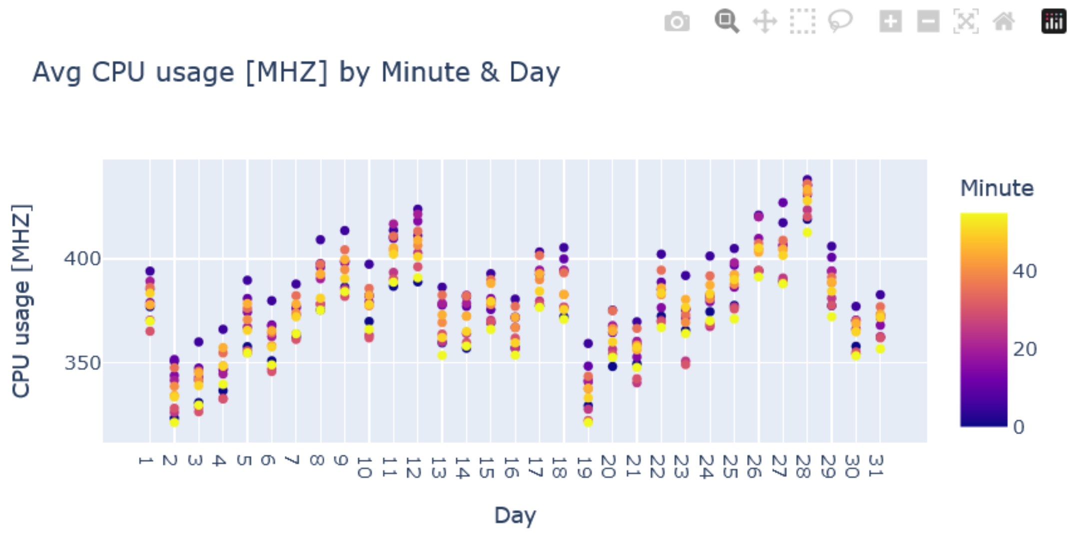


Figure 22: Avg CPU Usage [MHz] by Min and Day

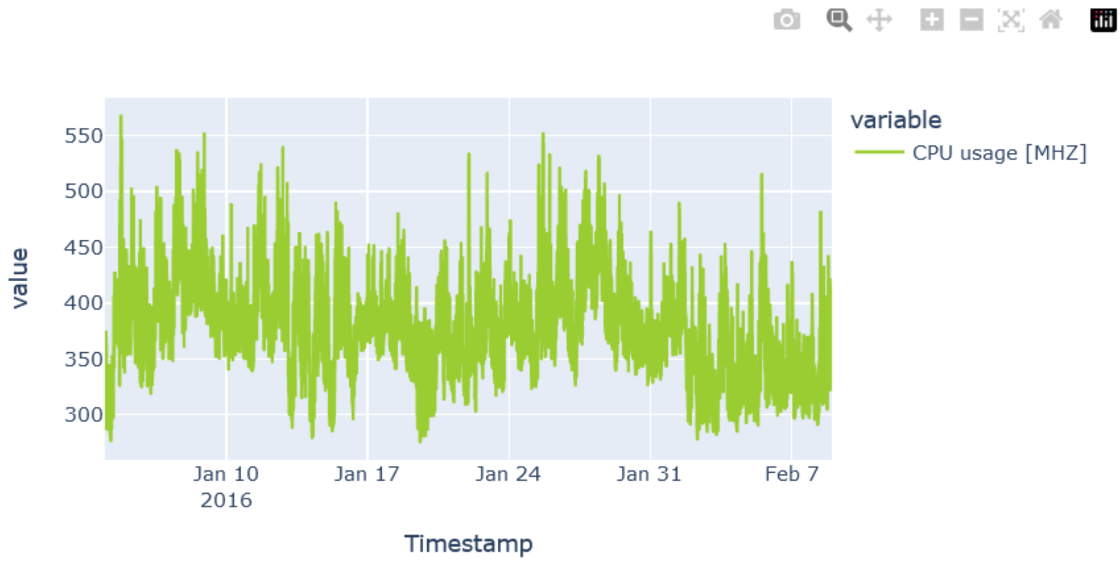


Figure 23: Time Series Plot of Usage by Date Time

4.5 Scaling data – Min-Max Normalization

Normalize the data for improved model performance.

```
#data normalization
data = final_df
scaler=MinMaxScaler(feature_range=(0,1))
dataframe=scaler.fit_transform(np.array(final_df).reshape(-1,1))
```

Figure 24: Normalization of Data

4.6 Window Rolling

Window Rolling with Timestamp (12 Previous Values)

```
# apply window rolling timestep
def window_rolling(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

```
time_step = 12
X_train, y_train = window_rolling(train_data, time_step)
X_test, y_test = window_rolling(test_data, time_step)
```

Figure 25: Apply window rolling to time step

```
[22]: #window rolling
new_final_df = final_df
rolling12_final_df = new_final_df.rolling(window=12).mean()
edadf = new_final_df.reset_index()
edadf1 = rolling12_final_df.reset_index()
newdf = edadf.merge(edadf1, on='Timestamp')
fig = px.line(newdf, x="Timestamp", y=newdf.columns, title='Actual vs WindowRolling Values')
fig.show()
file_name="./cpu_utilization_graphs/cloudcpuforecastgraph2.png"
fig.write_image(file_name)
upload_file_to_s3(file_name, bucket_name)
```

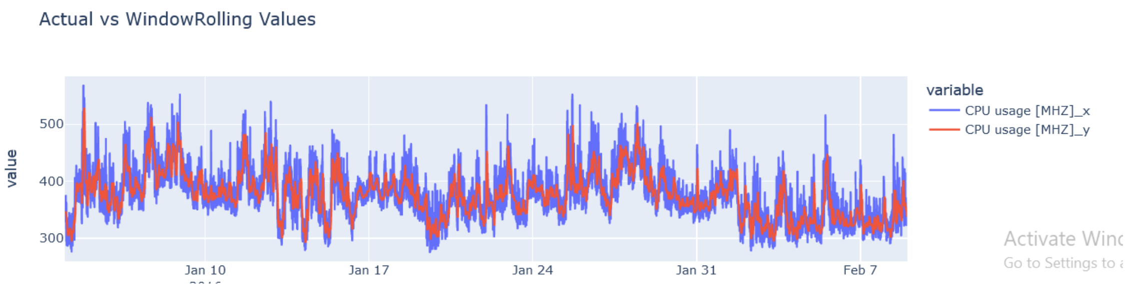


Figure 26: Comparing to actual values

4.7 Data Splitting

Split the data into training and testing (80:20) sets to evaluate model performance.

```
#splitting dataset into train and test with ration of 80:20
training_size=int(len(dataframe)*0.80)
test_size=len(dataframe)-training_size
train_data,test_data=dataframe[0:training_size,:],dataframe[training_size:len(dataframe),:1]
```

Figure 27: Splitting of Data

4.8 Data Learning Models

#Machine Learning Models

#Decision Tree Regressor

```
[31]: dtr_model = DecisionTreeRegressor(max_depth=2)
      dtr_model.fit(X_train, y_train)
      dtr_model.predict(X_test)
      predicted = dtr_model.predict(X_test)
      print(dtr_model.score(X_test, y_test))

      0.8661293319861584

[32]: print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
      print('R-2 Score :', metrics.r2_score(y_test, predicted))

      predicted_value = scaler.inverse_transform(predicted.reshape(-1, 1))
      real_value = scaler.inverse_transform(y_test.reshape(-1, 1))

      outputdf = pd.DataFrame({
          "Real": real_value.ravel(),
          "Predicted": predicted_value.ravel()
      }, index = data.index[-len(real_value): ])
      outputdf.head()

      Root Mean Squared Error: 0.04806755142422494
      R-2 Score : 0.8661293319861584
[32]:
```

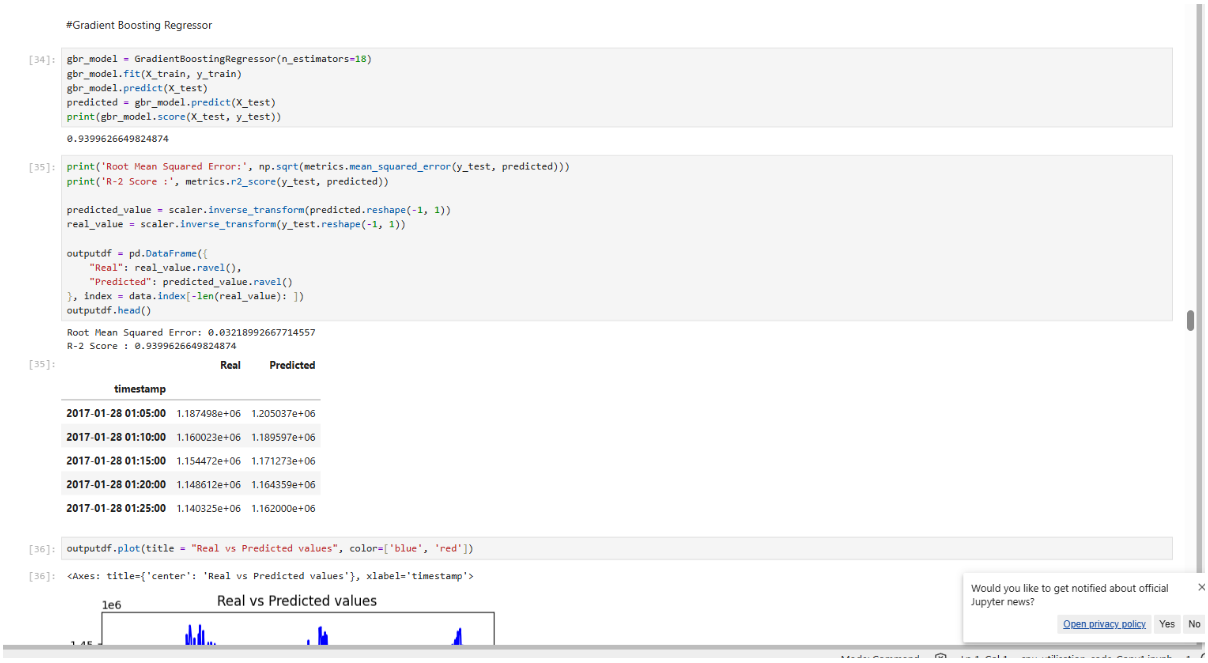
timestamp	Real	Predicted
2017-01-28 01:05:00	1.187498e+06	1.172015e+06
2017-01-28 01:10:00	1.160023e+06	1.172015e+06
2017-01-28 01:15:00	1.154472e+06	1.172015e+06
2017-01-28 01:20:00	1.148612e+06	1.172015e+06
2017-01-28 01:25:00	1.140325e+06	1.172015e+06

```
[33]: outputdf.plot(title = "Real vs Predicted values", color=['blue', 'red'])

[33]: <Axes: title={'center': 'Real vs Predicted values'}, xlabel='timestamp'>
```

Would you like to get notified about official Jupyter news?
Open privacy policy Yes No

Figure 28: Running the Decision Tree Regressor



#Gradient Boosting Regressor

```
[34]: gbr_model = GradientBoostingRegressor(n_estimators=18)
      gbr_model.fit(X_train, y_train)
      gbr_model.predict(X_test)
      predicted = gbr_model.predict(X_test)
      print(gbr_model.score(X_test, y_test))

      0.9399626649824874

[35]: print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
      print('R-2 Score :', metrics.r2_score(y_test, predicted))

      predicted_value = scaler.inverse_transform(predicted.reshape(-1, 1))
      real_value = scaler.inverse_transform(y_test.reshape(-1, 1))

      outputdf = pd.DataFrame({
          "Real": real_value.ravel(),
          "Predicted": predicted_value.ravel()
      }, index = data.index[-len(real_value): ])
      outputdf.head()

      Root Mean Squared Error: 0.03218992667714557
      R-2 Score : 0.9399626649824874
[35]:
```

timestamp	Real	Predicted
2017-01-28 01:05:00	1.187498e+06	1.205037e+06
2017-01-28 01:10:00	1.160023e+06	1.189597e+06
2017-01-28 01:15:00	1.154472e+06	1.171273e+06
2017-01-28 01:20:00	1.148612e+06	1.164359e+06
2017-01-28 01:25:00	1.140325e+06	1.162000e+06

```
[36]: outputdf.plot(title = "Real vs Predicted values", color=['blue', 'red'])

[36]: <Axes: title={'center': 'Real vs Predicted values'}, xlabel='timestamp'>
```


Would you like to get notified about official Jupyter news?
Open privacy policy Yes No

Figure 29: Running the Gradient Boosting Regressor

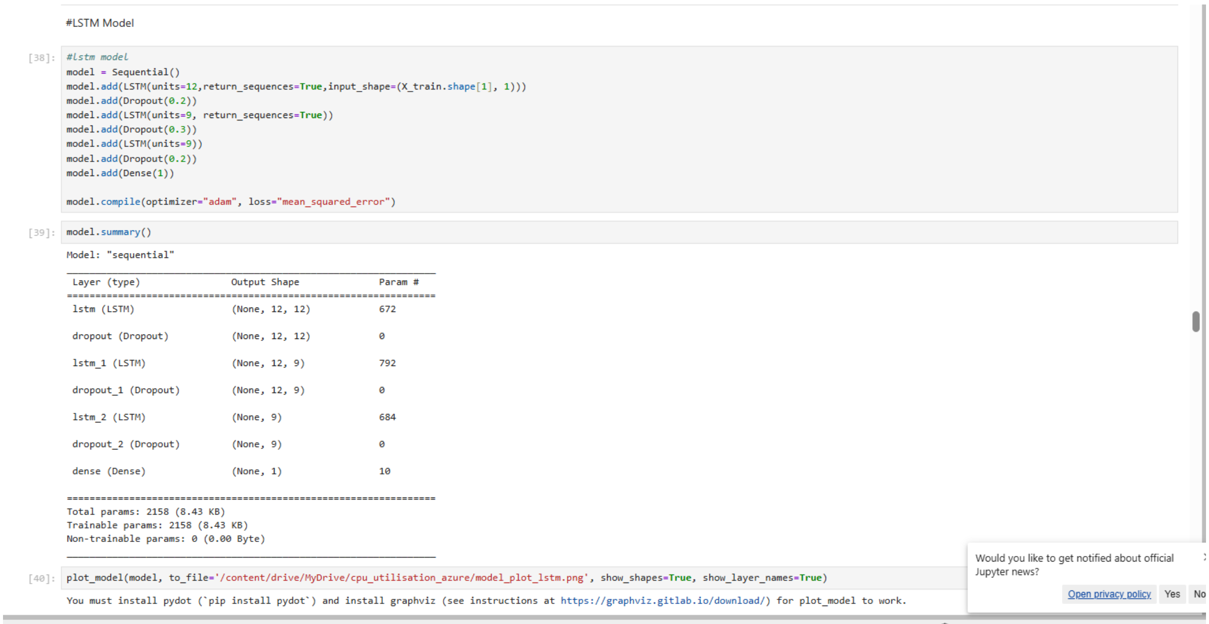


Figure 30: Running Deep Learning Models (LSTM)

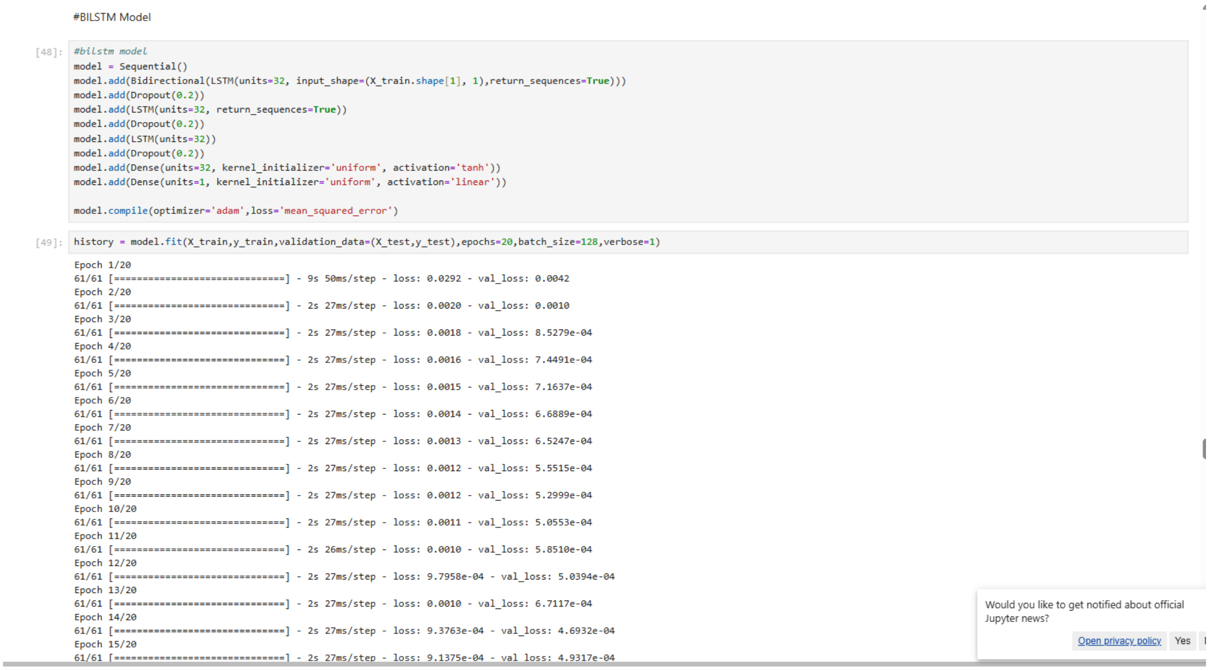


Figure 31: BiLSTM Model Run

Multihead bilstm

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Bidirectional, Dense, Dropout, Concatenate

# Input layer
input_layer = Input(shape=(X_train.shape[1], 1))

# === BiLSTM Head 1 ===
head1 = Bidirectional(LSTM(32, return_sequences=True))(input_layer)
head1 = Dropout(0.2)(head1)
head1 = LSTM(32, return_sequences=True)(head1)
head1 = Dropout(0.2)(head1)
head1 = LSTM(32)(head1)
head1 = Dropout(0.2)(head1)

# === BiLSTM Head 2 ===
head2 = Bidirectional(LSTM(32, return_sequences=True))(input_layer)
head2 = Dropout(0.2)(head2)
head2 = LSTM(32, return_sequences=True)(head2)
head2 = Dropout(0.2)(head2)
head2 = LSTM(32)(head2)
head2 = Dropout(0.2)(head2)

# === BiLSTM Head 3 (different configuration) ===
head3 = Bidirectional(LSTM(16, return_sequences=True))(input_layer)
head3 = Dropout(0.2)(head3)
head3 = LSTM(16)(head3)
head3 = Dropout(0.2)(head3)

# === Merge all heads ===
merged = Concatenate()([head1, head2, head3])

# === Dense layers ===
dense = Dense(32, kernel_initializer='uniform', activation='tanh')(merged)
output = Dense(1, kernel_initializer='uniform', activation='linear')(dense)
```

Figure 32: Multihead BiLSTM Model Run

```
outputdf.head()
```

```
Root Mean Squared Error: 0.021343953440310356
R-Score : 0.9736044229354797
```

timestamp	Real	Predicted
2017-01-28 01:05:00	1.187498e+06	1182782.875
2017-01-28 01:10:00	1.160023e+06	1184300.750
2017-01-28 01:15:00	1.154472e+06	1176124.125
2017-01-28 01:20:00	1.148612e+06	1168153.125
2017-01-28 01:25:00	1.140325e+06	1160207.000

```
outputdf.plot(title = "Real vs Predicted values", color=['blue', 'red'])
```

```
&lt;Axes: title={'center': 'Real vs Predicted values'}, xlabel='timestamp'&gt;
```

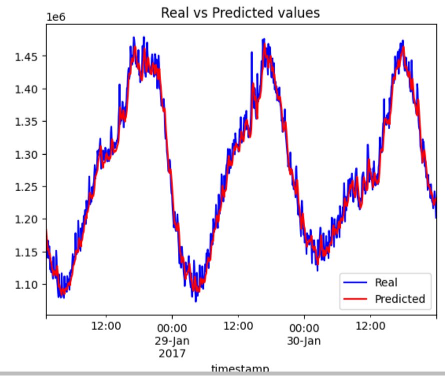


Figure 33: Model Results for Multihead BiLSTM Model

5 Conclusion

The steps and procedures explained in this configuration manual will serve as a guide to any researcher trying to replicate the implementation of the cloud-based CPU Utilization forecasting system that is described in the research project. By following the instructions on the setup of the environment, data preprocessing and model training using all the AWS services used in this project by using the Python libraries, users can get the same experimental outcomes and performance metrics that have achieved in the current study. The visualizations and evaluation techniques demonstrate the objectives of the study that prove the better performance of the deep learning models particularly Multihead BiLSTM in predicting time-series prediction in cloud environments.

6 References

Amazon Web Services (n.d.) *Step 1: Create an Amazon S3 bucket*. Available at: https://docs.aws.amazon.com/quickstarts/latest/s3backup/step-1-create-bucket.html (Accessed: 23 April 2025).

JFrog (n.d.) *Cloud9 setup – JFrog AWS Workshop*. Available at: https://jfrog.awsworkshop.io/2_self_guided_setup/22_cloud9.html (Accessed: 23 April 2025).

Zhou, L. (n.d.) *Data visualization and exploration in pandas and matplotlib*. GitHub Gist. Available at: https://gist.github.com/LiutongZhou/44655c8ed8e1c77d3f6a035e2b83e1f7 (Accessed: 23 April 2025).