

Configuration Manual

MSc Cloud Computing
Research Project

Kripa Mariam Joy
Student ID: 20217986

School of Computing
National College of Ireland

Supervisor: Shivani Jaswal

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name:

Kripa Mariam Joy

Student ID:

20217986

Programme:

MSc. Cloud Computing

Year: 2022

Module:

Research Project

Lecturer:

Shivani Jaswal

Submission Due

Date:

15/08/2022

Project Title:

DynamicForecast: Experts Council with optimized Workload
Prediction Framework for Cloud Computing

Word Count:

.....1200.....

Page Count:

.....21.....

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:

KripaMJoy

Date:

02/08/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Kripa Mariam Joy
Student ID: 20217986

1 Introduction

Configuration manual aids the reader to gain knowledge regarding system requirements, setup, and software specifications used for research DynamicForecast (DF).

2 System Configuration

2.1 Hardware Specification

- Model: HP Pavilion x360 Convertible 14-dg0xxx
- Processor: Intel(R) Core (TM) i5-8265U CPU @1.60GHz 1.80 GHz
- Operating System: Windows 10
- RAM: 8.00 GB (7.83 GB usable)
- Hard Disk: 256 GB

3 Software Used

To implement the proposed system, DF, set up the platform and environment in such a way that it should run machine learning and deep learning models.

The software used for implementing DF is listed below;

- Visual Studio Code
- Anaconda

To install packages of deep learning and machine learning, anaconda is used. Python is the programming language used to code this system.

4 Procedures

Step 1: Install all the software needed and install all libraries such as NumPy, sklearn, pickle, tkinter and so on using the command `pip install "package"`.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
import pickle
```

Fig 1: Importing librairaies

Step 2: Pre-process the dataset. Figure 2 indicates snap of one dataset used.

```
Generated by get-clean-log.py ($Revision: 0.15) on Tue February 20, 2007, at 09:48:14 PM
Authors: Alexandru Iosup and Mathieu Jan ([A.Iosup|M.Jan] at tudelft.nl)
The Grid workloads Archive (http://gwa.ewi.tudelft.nl/)
External coallocated_jobs info file: grid5000_coallocated_jobs.log
External interactive_jobs info file: grid5000_interactive_jobs.log
External reservation_jobs info file: grid5000_reservation_jobs.log
External sites_time info file: grid5000_sites_time.log
External user_to_group info file: grid5000_user_to_group.log

Grid workloads Format: JobIdCTAB>SubmitTimeCTAB>WaitTimeCTAB>RunTimeCTAB>HPProcCTAB>AverageCPUTimeUsedCTAB>UsedMemoryCTAB>ReqHPProcCTAB>ReqLineCTAB>ReqMemoryCTAB>StatusCTAB>UserIdCTAB>Group
1 1083658801 1 0 4 -1 -1 4 3600 -1 1 user386 group4 app34 queue0 -1 G1/site4 G1/site6/c1 UNITARY -1 -1
2 1083658849 1 19 1 -1 -1 1 3600 -1 1 user112 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
3 1083658875 2 10 5 -1 -1 5 3600 -1 1 user112 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
4 1083658891 5 8 90 -1 -1 90 3600 -1 1 user112 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
5 1083658911 5 19 100 -1 -1 100 3600 -1 1 user112 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
6 1083658944 1 25 1 -1 -1 1 3600 -1 0 user112 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
7 1083659210 1 6 1 -1 -1 1 3600 -1 1 user568 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
8 1083659322 1 43205 4 -1 -1 4 43200 -1 0 user386 group4 app0 queue0 -1 G1/site4 G1/site6/c1 UNITARY -1 -1
9 1083659636 1 5 1 -1 -1 1 3600 -1 1 user568 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
10 1083660389 -1 -1 4 -1 -1 4 9000 -1 0 user267 group5 app507 queue48 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
11 1083660523 2 156 7 -1 -1 7 18000 -1 1 user569 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
12 1083660693 1 19 7 -1 -1 7 18000 -1 1 user569 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
13 1083660719 1 4 7 -1 -1 7 18000 -1 1 user569 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
14 1083660726 1 2801 7 -1 -1 7 18000 -1 1 user569 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
15 1083660777 1 1 4 -1 -1 4 3600 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
16 1083660832 1 0 4 -1 -1 4 3600 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
17 1083660933 1 0 4 -1 -1 4 3600 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
18 1083661197 1 20992 1 -1 -1 1 36000 -1 1 user570 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
19 1083661760 1 23027 1 -1 -1 1 43200 -1 1 user571 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
20 1083661777 1 23014 1 -1 -1 1 43200 -1 1 user571 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
21 1083662072 1 22216 1 -1 -1 1 28800 -1 1 user67 group2 app0 queue0 -1 G1/site2 G1/site6/c1 UNITARY -1 -1
22 1083662523 1 13734 10 -1 -1 10 18000 -1 0 user569 group6 app0 queue0 -1 G1/site6 G1/site6/c1 UNITARY -1 -1
23 1083662430 1 1 4 -1 -1 4 3600 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
24 1083662460 1 1 4 -1 -1 4 3600 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
25 1083662628 1 1 4 -1 -1 4 3600 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
26 1083662739 1 1 1 -1 -1 1 3600 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
27 1083662841 1 0 1 -1 -1 1 9000 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
28 1083670843 1 1 4 -1 -1 4 9000 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
29 1083670602 1 1 4 -1 -1 4 9000 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
30 1083670874 1 1 4 -1 -1 4 9000 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
31 1083671023 1 1 4 -1 -1 4 9000 -1 1 user267 group5 app507 queue0 -1 G1/site5 G1/site6/c1 UNITARY -1 -1
```

Fig 2: Grid 5000 workload dataset

This dataset is pre-processed by discarding all the data except job arrival time. So that dynamic variation can be captured. Figure 3 depicts the code for removing unnecessary data, normalization of data using the MinMaxScalar function, and printing needed data. Figure 4 indicates the output for pre-processing step. The output is obtained from the command prompt after running the command “python pre1.py”.

```

1
2 import pandas as pd
3 import numpy as np
4
5 f=open('grid5000.log','r')
6 content=f.readlines()
7 f.close()
8 content.remove('\n')
9 times=[]
10 cnt=0
11 for i in range(10,len(content)):
12     # if cnt>1000:
13     #     break
14     try:
15         row=content[i].split('\t')
16         time=row[1][:5]
17         print(time)
18         times.append(time)
19         cnt+=1
20     except:
21         continue
22
23 df=pd.DataFrame()
24 df['times']=times
25
26 counts=df.times.value_counts()
27 counts1=dict(counts)
28 workload=list(counts1.values())
29 print(workload)
30
31 from sklearn.preprocessing import MinMaxScaler
32 scaler=MinMaxScaler(feature_range=(1,10))
33 # transform data
34 workload=np.array(workload)
35 workload=workload.reshape(-1,1)
36 workload= scaler.fit_transform(workload)
37 print(workload)

```

Fig 3: Snippet for Preprocessing of the dataset

```

C:\Users\HP>activate env_torch
(env_torch) C:\Users\HP>cd C:\Users\HP\Desktop\workload_prediction
(env_torch) C:\Users\HP\Desktop\workload_prediction>cd grid500
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python pre1.py
[101600, 2762, 2247, 22407, 21152, 11372, 12800, 12773, 10944, 18909, 10588, 9995, 9806, 9718, 9646, 9627, 9275, 9072, 8871, 8768, 8532, 8309, 8219, 8052, 7727, 7644, 7513, 7507, 7350, 6946, 6730, 6646, 6524, 6523, 6348, 6029, 5948, 5835, 5674, 5646, 5427, 5484, 5391, 5381, 5143, 5107, 5056, 5055, 5003, 4888, 4869, 4891, 4803, 4835, 4729, 4712, 4573, 4526, 4457, 4456, 4340, 4337, 4336, 4255, 4216, 4035, 4002, 3874, 3747, 3617, 3595, 3534, 3522, 3496, 3494, 3480, 3456, 3281, 3277, 3237, 3232, 3216, 3214, 3096, 3039, 2908, 2895, 2771, 2640, 2613, 2499, 2453, 2430, 2420, 2338, 2289, 2205, 2156, 2143, 2051, 2037, 2027, 2014, 1970, 1975, 1974, 1961, 1947, 1940, 1929, 1921, 1921, 1916, 1904, 1861, 1844, 1809, 1805, 1796, 1768, 1747, 1739, 1713, 1707, 1680, 1677, 1665, 1657, 1648, 1640, 1632, 1617, 1594, 1584, 1568, 1565, 1563, 1545, 1533, 1501, 1488, 1467, 1461, 1401, 1400, 1397, 1365, 1348, 1344, 1342, 1327, 1303, 1301, 1297, 1293, 1288, 1280, 1265, 1245, 1210, 1203, 1179, 1175, 1173, 1166, 1162, 1161, 1157, 1153, 1153, 1152, 1146, 1143, 1140, 1133, 1113, 1112, 1107, 1099, 1096, 1088, 1086, 1068, 1063, 1058, 1056, 1049, 1049, 1049, 1043, 1043, 1041, 1026, 1019, 1015, 1001, 1002, 1000, 992, 990, 981, 978, 973, 973, 971, 969, 964, 959, 945, 945, 938, 936, 935, 933, 930, 929, 929, 919, 914, 908, 903, 903, 902, 899, 891, 876, 876, 874, 874, 869, 862, 856, 854, 853, 850, 844, 844, 843, 841, 840, 833, 832, 827, 824, 811, 810, 809, 809, 806, 796, 796, 793, 792, 787, 785, 783, 770, 768, 767, 762, 758, 755, 747, 742, 742, 733, 732, 730, 727, 723, 720, 710, 709, 704, 702, 696, 693, 689, 686, 679, 678, 677, 676, 659, 659, 657, 652, 647, 645, 645, 643, 643, 640, 636, 631, 627, 625, 620, 618, 613, 612, 600, 599, 594, 586, 583, 573, 561, 548, 545, 544, 542, 541, 536, 535, 534, 529, 528, 527, 525, 523, 520, 518, 518, 510, 508, 507, 506, 506, 505, 504, 503, 500, 500, 498, 492, 478, 476, 475, 465, 463, 462, 461, 458, 454, 451, 450, 444, 444, 442, 441, 440, 435, 431, 427, 422, 422, 422, 421, 418, 416, 415, 408, 407, 407, 404, 404, 403, 399, 399, 398, 398, 388, 383, 379, 374, 372, 369, 363, 362, 359, 353, 350, 345, 342, 341, 340, 339, 338, 333, 333, 332, 328, 324, 316, 313, 311, 303, 303, 299, 297, 294, 290, 286, 282, 280, 276, 273, 271, 269, 263, 261, 261, 259, 252, 251, 250, 245, 244, 241, 241, 241, 238, 229, 227, 227, 227, 225, 225, 223, 223, 221, 221, 220, 219, 217, 216, 216, 215, 211, 210, 205, 200, 198, 197, 195, 193, 192, 191, 190, 185, 184, 183, 183, 182, 182, 181, 180, 177, 173, 172, 172, 172, 171, 171, 170, 170, 169, 168, 168, 166, 165, 164, 162, 161, 159, 157, 156, 154, 153, 153, 152, 151, 149, 148, 147, 145, 145, 144, 143, 142, 141, 138, 135, 134, 133, 132, 130, 128, 127, 126, 126, 123, 121, 120, 119, 117, 117, 116, 116, 116, 115, 115, 114, 114, 113, 113, 112, 111, 111, 111, 111, 108, 108, 107, 107, 106, 106, 106, 106, 104, 103, 102, 102, 102, 101, 100, 98, 98, 97, 96, 94, 93, 92, 92, 92, 92, 90, 90, 89, 88, 88, 87, 87, 86, 85, 84, 84, 83, 83, 82, 82, 80, 80, 80, 80, 79, 79, 79, 78, 77, 77, 76, 76, 76, 75, 75, 74, 72, 72, 72, 72, 70, 70, 70, 70, 69, 67, 67, 66, 66, 65, 64, 63, 63, 63, 63, 62, 61, 60, 58, 58, 57, 55, 55, 55, 53, 53, 52, 51, 51, 50, 50, 49, 49, 48, 48, 48, 47, 47, 46, 46, 46, 45, 45, 44, 44, 44, 43, 43, 42, 42, 41, 41, 41, 41, 41, 40, 40, 40, 40, 39, 39, 39, 39, 38, 38, 37, 36, 36, 35, 35, 35, 35, 34, 34, 33, 33, 33, 32, 32, 31, 31, 31, 30, 30, 30, 29, 29, 29, 29, 28, 28, 28, 27, 26, 26, 25, 25, 24, 24, 24, 23, 22, 22, 22, 21, 21, 21, 21, 20, 19, 19, 19, 18, 18, 18, 17, 17, 17, 17, 16, 16, 16, 16, 15, 14, 13, 13, 12, 12, 12, 11, 11, 11, 11, 10, 10, 10, 9, 9, 8, 8, 8, 7, 7, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 4, 4, 4, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1]
[10]
[ 3.45916791]
[ 3.03263812]
[ 2.984803 ]
[ 2.87363865]
[ 2.18445063]
[ 2.13378884]
[ 2.13138909]
[ 1.96936978]
[ 1.96626935]
[ 1.93783483]
[ 1.88530399]
[ 1.87653422]
[ 1.86076635]
[ 1.85438833]
[ 1.85270524]
[ 1.82152383]
[ 1.80354137]
[ 1.78573688]
[ 1.77643481]
[ 1.75578626]
[ 1.75366884]
[ 1.72797961]
[ 1.71318615]
[ 1.6842965 ]
[ 1.67784407]
[ 1.66543962]
[ 1.66490812]
[ 1.6510005 ]
[ 1.61521275]
[ 1.5968787 ]

```

Fig 4: Output obtained for pre-processed dataset (filtering and normalization)

Step 3a) Code each and every predictor in the pool [1] of DF. Figure 5 illustrates the predictors in the pool. While coding the predictor, a portion of dataset is given for testing and training.



Fig 5: Predictors in predictor pool

Step 3b) The figure 6 depicts the code for predictor LR (Logistic Regression). The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 7 illustrates the output obtained for LR model.

```

grid500 > train_LR.py
1  import pandas as pd
2  from sklearn.linear_model import LinearRegression
3  from sklearn.metrics import mean_squared_error
4  from sklearn.model_selection import train_test_split
5  #from sklearn.metrics import mean_absolute_percentage_error
6
7
8  import numpy as np
9
10 def mean_absolute_percentage_error(y_true, y_pred):
11     y_true, y_pred = np.array(y_true), np.array(y_pred)
12     return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
13
14
15 df=pd.read_csv('grid500.csv')
16 val1=df.values
17 print(val1.shape)
18
19 X=[]
20 y=[]
21
22 for i in range(15,len(df)):
23     v=[i[0] for i in val1[i-15:i]]
24     X.append(v)
25     y.append(val1[i,0])
26
27
28 print(X[0])
29 print(y[0])
30
31 print(len(X))

```

```

32
33 from sklearn.pipeline import make_pipeline
34 from sklearn.preprocessing import StandardScaler
35
36 x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
37
38 reg = make_pipeline(StandardScaler(), LinearRegression())
39 reg.fit(x_train, y_train)
40
41 y_pred=reg.predict(x_test)
42
43 print(y_pred[:10])
44 print(y_train[:10])
45
46 mse=mean_absolute_percentage_error(y_test, y_pred)
47
48 print(mse)
49
50 import pickle
51
52 pickle.dump(reg, open('lr.pkl', 'wb'))
53
54

```

Training and testing of predictor

Fig 6 : LR code snippet

```

(env_torch) C:\Users\VP\Desktop\workload_prediction\grid500\python train_lr.py
(781, 1)
[100.0, 28.05884695715589, 23.359019281685843, 22.832833080324884, 21.60993710568817, 14.02895697792301, 13.471589287296133, 13.445279973228882, 11.66386754987746, 11.628962883492944, 11.316174371794993, 10.738343881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
[1.02246036 2.50896631 2.58856872 8.06984697 1.04726161 2.73665711
 1.42497925 1.06267058 1.26233374 1.89929074]
[1.1364186653388466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8668124607525663, 1.0935442277975176]
0.321473673600518

```

Fig 7: Output for LR

Step 3c) The figure 8 depicts the code for predictor LASSO (Least Absolute Shrinkage and Selection Operator). The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 9 illustrates the output obtained for LASSO model.

```

35 from sklearn import linear_model
36 from sklearn.pipeline import make_pipeline
37 from sklearn.preprocessing import StandardScaler
38
39 x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
40
41 reg = make_pipeline(StandardScaler(), linear_model.Lasso(alpha=0.1))
42 reg.fit(x_train, y_train)
43
44 y_pred=reg.predict(x_test)
45
46 print(y_pred[:10])
47 print(y_train[:10])
48
49 mse=mean_absolute_percentage_error(y_test, y_pred)
50
51 print(mse)
52
53 import pickle
54
55 pickle.dump(reg, open('lasso.pkl', 'wb'))
56
57
58 ~ if __name__ == "__main__":
59     train('grid500.csv')

```

Fig 8: LASSO code snippet

```

(env_torch) C:\Users\VP\Desktop\workload_prediction\grid500\python train_lasso.py
(781, 1)
[100.0, 28.05884695715589, 23.359019281685843, 22.832833080324884, 21.60993710568817, 14.02895697792301, 13.471589287296133, 13.445279973228882, 11.66386754987746, 11.628962883492944, 11.316174371794993, 10.738343881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
C:\Users\VP\anaconda3\envs\env_torch\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:532: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 0.3476593733219051, tolerance: 0.1385032330310563
  positive)
[1.089583 2.43881754 2.51672886 7.59990867 1.11173994 2.65756886
 1.45257145 1.12681005 1.30984328 1.87413623]
[1.1364186653388466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8668124607525663, 1.0935442277975176]
1.8835852367961986

```

Fig 9: Output for LASSO

Step 3d) The figure 10 depicts the code for predictor RIDGE. The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 11 illustrates the output obtained for RIDGE model.

```

33 from sklearn import linear_model
34 from sklearn.pipeline import make_pipeline
35 from sklearn.preprocessing import StandardScaler
36
37 x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
38
39 reg = make_pipeline(StandardScaler(), linear_model.Ridge(alpha=.5))
40 reg.fit(x_train, y_train)
41
42 y_pred=reg.predict(x_test)
43
44 print(y_pred[:10])
45 print(y_train[:10])
46
47 mse=mean_absolute_percentage_error(y_test, y_pred)
48
49 print(mse)
50
51 import pickle
52
53 pickle.dump(reg,open('ridge.pkl','wb'))

```

Fig 10: RIDGE code snippet

```

(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python train_Ridge.py
(781, 1)
[100.0, 28.05084695715509, 23.359019281685843, 22.832833000324804, 21.60993710568017, 14.02895697792301, 13.471589287296133, 13.445279973228082, 11.66306754987746, 11.628962883492944, 11.316174371794993, 10.738343881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674788263
766
[1.02805785 2.4883846 2.57779845 7.99904351 1.05238492 2.73128765
 1.42388593 1.06781612 1.26644943 1.88546705]
[1.1364186653380466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
9.487286519485908

```

Fig 11: Output for RIDGE

Step 3e) The figure 12 depicts the code for predictor SVR (Support Vector Regression). The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 13 illustrates the output obtained for SVR model.

```

34 from sklearn.pipeline import make_pipeline
35 from sklearn.preprocessing import StandardScaler
36
37 x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
38
39 reg = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
40 reg.fit(x_train, y_train)
41
42 y_pred=reg.predict(x_test)
43
44 print(y_pred[:10])
45 print(y_train[:10])
46
47 mse=mean_absolute_percentage_error(y_test, y_pred)
48
49 print(mse)
50
51 import pickle
52
53
54 pickle.dump(reg,open('SVR.pkl','wb'))
55

```

Fig 12: SVR code snippet

```

(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python trainSVR.py
(781, 1)
[100.0, 28.05084695715509, 23.359019281685843, 22.832833000324804, 21.60993710568017, 14.02895697792301, 13.471589287296133, 13.445279973228082, 11.66306754987746, 11.628962883492944, 11.316174371794993, 10.738343881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674788263
766
[1.1944079 2.47915516 2.59831207 7.79932449 1.18934064 2.7596937
 1.27985006 1.18695493 1.20730286 1.71141819]
[1.1364186653380466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
9.487286519485908

```

Fig 13: SVR output

Step 3f) The figure 14 depicts the code for predictor Stochastic Gradient Descent (SGD). The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 15 illustrates the output obtained for SGD model.

```

33 from sklearn import linear_model
34 from sklearn.linear_model import SGDRegressor
35 from sklearn.pipeline import make_pipeline
36 from sklearn.preprocessing import StandardScaler
37
38 x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
39
40 reg = make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))
41 reg.fit(x_train, y_train)
42
43 y_pred=reg.predict(x_test)
44
45 print(y_pred[:10])
46 print(y_train[:10])
47
48 mse=mean_absolute_percentage_error(y_test, y_pred)
49
50 print(mse)
51
52 import pickle
53
54 pickle.dump(reg,open('sgd.pkl','wb'))
55
56

```

Fig 14: SGD code snippet

```

(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python trainSGD.py
(781, 1)
[100.0, 28.05084695715509, 23.359019281685843, 22.832833000324804, 21.60993710568017, 14.02895697792301, 13.471589287296133, 13.445279973228082, 11.66306754987746, 11.628962883492944, 11.316174371794993, 10.738343881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
785
[1.05637976 2.4218249 2.5172306 7.97835927 1.0789769 2.66142713
 1.42371326 1.09315318 1.28385949 1.84440914]
[1.1364186655380466, 2.5425853396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
2.338366101383751

```

Fig 15: Output for SGD

Step 3g) The figure 16 depicts the code for predictor LAR (Least Angle Regression). The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 17 illustrates the output obtained for LAR model.

```

33 from sklearn import linear_model
34 from sklearn.pipeline import make_pipeline
35 from sklearn.preprocessing import StandardScaler
36
37 x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
38
39 reg = make_pipeline(StandardScaler(), linear_model.LassoLars(alpha=.1, normalize=False))
40 reg.fit(x_train, y_train)
41
42 y_pred=reg.predict(x_test)
43
44 print(y_pred[:10])
45 print(y_train[:10])
46
47 mse=mean_absolute_percentage_error(y_test, y_pred)
48
49 print(mse)
50
51 import pickle
52
53 pickle.dump(reg,open('lar.pkl','wb'))
54
55

```

Fig 16: LAR code snippet

```
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500\python train_lar.py
(781, 1)
[180.0, 28.05084695715509, 23.359019281685843, 22.832833000324884, 21.60993718568017, 14.02895697792301, 13.471589287296133, 13.445279973228082, 11.66306754987746, 11.628962883492944, 11.316174371794993, 10.738343881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
[1.08193715 2.45850685 2.54253124 7.63136976 1.10517794 2.67482495
 1.4555775 1.11947996 1.30540626 1.89178919]
[1.1364186655380466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
3.4791713243271536
```

Fig 17: Output for LAR

Step 3h) The figure 18 depicts the code for predictor HUBER. The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 19 illustrates the output obtained for HUBER model.

```
35 from sklearn import linear_model
36 from sklearn.linear_model import HuberRegressor
37 from sklearn.pipeline import make_pipeline
38 from sklearn.preprocessing import StandardScaler
39
40 x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
41
42 reg = make_pipeline(StandardScaler(), HuberRegressor())
43 reg.fit(x_train, y_train)
44
45 y_pred=reg.predict(x_test)
46
47 print(y_pred[:10])
48 print(y_train[:10])
49
50 mse=mean_absolute_percentage_error(y_test, y_pred)
51
52 print(mse)
53
54 import pickle
55
56 pickle.dump(reg,open('huber.pkl','wb'))
57
58
59 if __name__=="__main__":
60     train('grid500.csv')
```

Fig 18: HUBER code snippet

```
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500\python train_huber.py
(781, 1)
[180.0, 28.05084695715509, 23.359019281685843, 22.832833000324884, 21.60993718568017, 14.02895697792301, 13.471589287296133, 13.445279973228082, 11.66306754987746, 11.628962883492944, 11.316174371794993, 10.738343881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
C:\Users\HP\anaconda3\envs\env_torch\lib\site-packages\sklearn\linear_model\_huber.py:296: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
[1.02127251 2.50009335 2.59915034 8.07390463 1.04601681 2.74102869
 1.42575374 1.06176193 1.26274419 1.90038339]
[1.1364186655380466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
0.27355103513925066
```

Fig 19: Output for HUBER

Step 3i) The figure 20 depicts the code for predictor ARIMA (Autoregressive Integrated Moving Average). The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 21 illustrates the output obtained for ARIMA model.

```

grid500 7 ARIMA.py
1 from pandas import read_csv
2 from matplotlib import pyplot
3 from statsmodels.tsa.arima.model import ARIMA
4 from sklearn.metrics import mean_squared_error
5 from math import sqrt
6 import numpy as np
7
8
9 def mean_absolute_percentage_error(y_true, y_pred):
10     y_true, y_pred = np.array(y_true), np.array(y_pred)
11     return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
12
13
14 def train(path):
15     df = read_csv(path)
16
17     X = df.values
18     size = int(len(X) * 0.66)
19     train, test = X[0:size], X[size:len(X)]
20     history = [x for x in train]
21     predictions = list()
22     # walk-forward validation
23     for t in range(len(test)):
24         model = ARIMA(history, order=(1,1,1))
25         model_fit = model.fit()
26         output = model_fit.forecast()
27         yhat = output[0]
28         predictions.append(yhat)
29         obs = test[t]
30         history.append(obs)
31         print('predicted=%f, expected=%f' % (yhat, obs))
32
33     mape = mean_absolute_percentage_error(test, predictions)
34     print('MAPE: %.3f' % mape)
35
36
37 if __name__ == "__main__":
38     path = 'grid500.csv'
39     train(path)

```

Fig 20: ARIMA code snippet

```

(predicted=1.111895, expected=1.111084
predicted=1.109830, expected=1.111084
predicted=1.111547, expected=1.110109
predicted=1.108624, expected=1.110109
predicted=1.110658, expected=1.109135
predicted=1.107618, expected=1.109135
predicted=1.109695, expected=1.109135
predicted=1.108928, expected=1.108161
predicted=1.106922, expected=1.107186
predicted=1.106328, expected=1.107186
predicted=1.107503, expected=1.107186
predicted=1.107069, expected=1.107186
predicted=1.107229, expected=1.104263
predicted=1.100302, expected=1.104263
predicted=1.105725, expected=1.103288
predicted=1.101434, expected=1.103288
predicted=1.103973, expected=1.102314
predicted=1.100746, expected=1.102314
predicted=1.102893, expected=1.102314
predicted=1.102100, expected=1.102314
predicted=1.102393, expected=1.100365
predicted=1.097705, expected=1.099391
predicted=1.099057, expected=1.098416
predicted=1.097224, expected=1.098416
predicted=1.098857, expected=1.098416
predicted=1.001576, expected=1.002923
predicted=1.003422, expected=1.002923
predicted=1.002739, expected=1.001949
predicted=1.000695, expected=1.001949
predicted=1.002413, expected=1.001949
predicted=1.001777, expected=1.000974
predicted=0.999716, expected=1.000974
predicted=1.001440, expected=1.000974
predicted=1.000802, expected=1.000974
predicted=1.001038, expected=1.000974
predicted=1.000951, expected=1.000974
predicted=1.000983, expected=1.000974
predicted=1.000971, expected=1.000974
predicted=1.000976, expected=1.000974
predicted=1.000974, expected=1.000974
predicted=1.000975, expected=1.000000
predicted=0.998678, expected=1.000000
predicted=1.000490, expected=1.000000
predicted=0.999819, expected=1.000000
predicted=1.000067, expected=1.000000
predicted=0.999975, expected=1.000000
MAPE: 3.588

```

Fig 21: Output for ARIMA

Step 3j) The figure 22 depicts the code for predictor Long Short Term Memory (LSTM). The highlighted area shows that 20% of data has been given to testing and training of the model. Figure 23 illustrates the output obtained for LSTM model.

```

grid500 > train_LSTM.py
1  import pandas as pd
2  from sklearn.linear_model import LinearRegression
3  from sklearn.metrics import mean_squared_error
4  from sklearn.model_selection import train_test_split
5  from keras.models import Sequential
6  from keras.layers import Dense
7  from keras.layers import LSTM
8  from sklearn.preprocessing import MinMaxScaler
9  from sklearn.metrics import mean_squared_error
10 #from sklearn.metrics import mean_absolute_percentage_error
11
12
13  import numpy as np
14
15  def mean_absolute_percentage_error(y_true, y_pred):
16      y_true, y_pred = np.array(y_true), np.array(y_pred)
17      return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
18
19
20  df=pd.read_csv('grid500.csv')
21  val1=df.values
22  print(val1.shape)
23
24  X=[]
25  y=[]
26
27  for i in range(15,len(df)):
28      v=[i[0] for i in val1[i-15:i]]
29      X.append(v)
30      y.append(val1[i,0])
31
32
33  print(X[0])
34  print(y[0])
35
36  print(len(X))
37
38  from sklearn import linear_model
39  from sklearn.pipeline import make_pipeline
40  from sklearn.preprocessing import StandardScaler
41
42  x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True, test_size=0.2)
43
44  x_train=np.array(x_train)
45  x_test=np.array(x_test)
46  y_train=np.array(y_train)
47  y_test=np.array(y_test)
48
49  x_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
50  x_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
51
52
53  model = Sequential()
54  model.add(LSTM(20, input_shape=(1,15)))
55  model.add(Dense(1))
56  model.compile(loss='mean_squared_error', optimizer='adam')
57  model.fit(x_train, y_train, epochs=100, batch_size=1, verbose=2)
58
59  y_pred=model.predict(x_test)
60
61  print(y_pred[:10])
62  print(y_train[:10])
63
64  mse=mean_absolute_percentage_error(y_test, y_pred)
65
66  print(mse)
67
68  import pickle
69
70  model.save('LSTM')
71

```

Code for Adams optimization

Fig 22: LSTM with Adams optimization code snippet

```

(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500\python train_LSTM.py
Using TensorFlow backend.
(781, 1)
[100, 0, 28.05084695715509, 23.359019281685843, 22.832833000324804, 21.60993710568017, 14.02895697792301, 13.471589287296133, 13.445279973228082, 11.66306754987746, 11.628962883492944, 11.316174371794993, 10.73834381337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
786
2022-07-29 14:25:36.809947: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
Epoch 1/100
- 1s - loss: 1.0751
Epoch 2/100
- 1s - loss: 0.3848
Epoch 3/100
- 1s - loss: 0.2284
Epoch 4/100
- 1s - loss: 0.1544
Epoch 5/100
- 1s - loss: 0.1133
Epoch 6/100
- 1s - loss: 0.0948
Epoch 7/100
- 1s - loss: 0.0753
Epoch 8/100
- 1s - loss: 0.0685
Epoch 9/100
- 1s - loss: 0.0537
Epoch 10/100
- 1s - loss: 0.0558
Epoch 11/100
- 1s - loss: 0.0542
Epoch 12/100
- 1s - loss: 0.0352
Epoch 13/100
- 1s - loss: 0.0359
Epoch 14/100
- 1s - loss: 0.0271
Epoch 15/100
- 1s - loss: 0.0482
Epoch 16/100
- 1s - loss: 0.0322
Epoch 17/100
- 1s - loss: 0.0310
Epoch 95/100
- 1s - loss: 0.0107
Epoch 96/100
- 1s - loss: 0.0302
Epoch 97/100
- 1s - loss: 0.0116
Epoch 98/100
- 1s - loss: 0.0157
Epoch 99/100
- 1s - loss: 0.0092
Epoch 100/100
- 1s - loss: 0.0129
[[1.0354334]
[2.4031105]
[2.493092 ]
[7.2851176]
[1.0625341]
[2.6192222]
[1.4431186]
[1.0787549]
[1.2906835]
[1.8656316]]
[1.13641867 2.54250534 1.73763521 2.11278654 1.14811169 1.93836554
1.51059558 1.4862351 2.86601246 1.09354423]
71.27236720771022

```

Fig 23: Output of LSTM module with Adams optimization

Step 4) The next step is to create four ensemble models according to the least MAPE value obtained. The figure 24 and figure 25 illustrate the code and output for ensemble model creation respectively.

```

1  import pandas as pd
2  from sklearn.linear_model import LinearRegression
3  from sklearn.metrics import mean_squared_error
4  from sklearn.model_selection import train_test_split
5  #from sklearn.metrics import mean_absolute_percentage_error
6  import pickle
7  import numpy as np
8  from keras.models import load_model
9  from statsmodels.tsa.arima.model import ARIMA
10 import matplotlib.pyplot as plt
11
12 lr=pickle.load(open('lr.pkl','rb'))
13 huber=pickle.load(open('huber.pkl','rb'))
14 lar=pickle.load(open('lar.pkl','rb'))
15 lasso=pickle.load(open('lasso.pkl','rb'))
16 ridge=pickle.load(open('ridge.pkl','rb'))
17 sgd=pickle.load(open('sgd.pkl','rb'))
18 lstm=load_model('LSTM')
19 svr=pickle.load(open('SVR.pkl','rb'))
20
21 def mean_absolute_percentage_error(y_true, y_pred):
22     y_true, y_pred = np.array(y_true), np.array(y_pred)
23     return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
24
25 ensemble_list=[0,1,2,3]
26
27 def ensemble(feat):
28     global ensemble_list
29     val=0
30     for i in ensemble_list:
31         val+=feat[i]
32     val=val/4
33     return val

```

Loading the models created by each predictor

Defining list for 4 ensemble models

```

38 def run():
39     global ensemble_list
40     df=pd.read_csv('grid500.csv')
41     val1=df.values
42     print(val1.shape)
43     X=[]
44     y=[]
45
46     for i in range(15,len(df)):
47         v=[i[0] for i in val1[i-15:i]]
48         X.append(v)
49         y.append(val1[i,0])
50
51     history=val1[:15]
52
53     #print(y)
54     #return
55
56     final_predictions=[]
57     final_originals=[]
58
59     cnt=0
60     lst1=[]
61     lst2=[]
62     lst3=[]
63     lst4=[]
64     lst5=[]
65     lst6=[]
66     lst7=[]
67     lst8=[]
68     lst9=[]
69     lst10=[]
70     cnt2=1

```

Window declaration as 1

```

73     for i in range(len(X)):
74
75         feat=[]
76         original_label=y[i]
77         lst1.append(original_label)
78         final_originals.append(original_label)
79         #1
80         pred_lr=lr.predict(X[i:i+1])[0]
81         lst2.append(pred_lr)
82         feat.append(pred_lr)
83         #2
84         pred_huber=huber.predict(X[i:i+1])[0]
85         lst3.append(pred_huber)
86         feat.append(pred_huber)
87         #3
88         pred_lar=lar.predict(X[i:i+1])[0]
89         lst4.append(pred_lar)
90         feat.append(pred_lar)
91         #4
92         pred_lasso=lasso.predict(X[i:i+1])[0]
93         lst5.append(pred_lasso)
94         feat.append(pred_lasso)
95         #5
96         pred_ridge=ridge.predict(X[i:i+1])[0]
97         lst6.append(pred_ridge)
98         feat.append(pred_ridge)
99         #6
100        pred_svr=svr.predict(X[i:i+1])[0]
101        lst7.append(pred_svr)
102        feat.append(pred_svr)
103        #7
104        pred_sgd=sgd.predict(X[i:i+1])[0]
105        lst8.append(pred_sgd)
106        feat.append(pred_sgd)
107        #8 (LSTM)
108        pred_lstm=lstm.predict(np.reshape(X[i:i+1], (1, 1, 15)))[0][0]
109        lst9.append(pred_lstm)
110        feat.append(pred_lstm)
111        #9
112        model = ARIMA(history, order=(1,1,1))
113        model_fit = model.fit()
114        output = model_fit.forecast()
115        pred_arma = output[0]
116        lst10.append(pred_arma)
117        feat.append(pred_arma)

```

Initialization of each list by each predictor


```

125     cnt+=1
126     val=ensemble(feats)
127     final_predictions.append(val)
128     print('prediction',cnt)
129     if cnt>=25:
130         history=history[-15:]
131         print('window',cnt2)
132         cnt2+=1
133         cnt=0
134         dict1={'lr':lst2,'huber':lst3,'lar':lst4,'lasso':lst5,'ridge':lst6,'svr':lst7,'sgd':lst8,'lstm':lst9,'arima':lst10,'original':lst1}
135         # print(dict1)
136         df=pd.DataFrame(dict1)
137         df.to_csv('temp.csv',index=False)
138
139         mape1=mean_absolute_percentage_error(lst1,lst2)
140         mape2=mean_absolute_percentage_error(lst1,lst3)
141         mape3=mean_absolute_percentage_error(lst1,lst4)
142         mape4=mean_absolute_percentage_error(lst1,lst5)
143         mape5=mean_absolute_percentage_error(lst1,lst6)
144         mape6=mean_absolute_percentage_error(lst1,lst7)
145         mape7=mean_absolute_percentage_error(lst1,lst8)
146         mape8=mean_absolute_percentage_error(lst1,lst9)
147         mape9=mean_absolute_percentage_error(lst1,lst10)
148
149         mape_list=[mape1,mape2,mape3,mape4,mape5,mape6,mape7,mape8,mape9]
150         print(mape_list,'=====')
151         mape_arr=np.array(mape_list)
152         mape_arr = np.argsort(mape_arr)[:4]
153         mape_list=list(mape_arr)
154         ensemble_list=mape_list
155
156     print(mape_arr)
157
158     lst1=[]
159     lst2=[]
160     lst3=[]
161     lst4=[]
162     lst5=[]
163     lst6=[]
164     lst7=[]
165     lst8=[]
166     lst9=[]
167     lst10=[]
168
169
170     mape=mean_absolute_percentage_error(final_originals,final_predictions)*10
171     df2=pd.DataFrame()
172     import pickle
173     scaler=pickle.load(open('scaler1.pkl','rb'))
174
175     final_originals=np.array(final_originals)
176     final_originals=final_originals.reshape(-1,1)
177     final_originals= scaler.inverse_transform(final_originals)
178
179     final_predictions=np.array(final_predictions)
180     final_predictions=final_predictions.reshape(-1,1)
181     final_predictions= scaler.inverse_transform(final_predictions)
182
183
184     df2['true']=final_originals.flatten()
185     df2['prediction']=final_predictions.flatten()
186
187     df2['step']=list(range(0,len(list(final_predictions))))
188
189
190
191     df2.plot(x='step',y=['prediction','true'])
192     plt.savefig("grid500.png")
193
194
195     print(mape)
196     return mape
197
198
199
200
201 if __name__=="__main__":
202     run()

```

Each window have 25 predictions (variable ; cnt)

Predictions are saved to temp.csv (workload repository)

MAPE Comparison

MAPE sorting for the predictors

Print the list of 4 ensemble model with least MAPE

Plotting actual and predicted values and saving the plot

Fig 24: Code snippet for ensemble model creation

```

(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python simulate.py
Using TensorFlow backend.
2022-08-01 00:28:21.483630: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
(781, 1)
prediction 1
prediction 2
prediction 3
prediction 4
prediction 5
prediction 6
prediction 7
prediction 8
prediction 9
prediction 10
prediction 11
prediction 12
prediction 13
prediction 14
prediction 15
prediction 16
prediction 17
prediction 18
prediction 19
prediction 20
prediction 21
prediction 22
prediction 23
prediction 24
prediction 25
window 1
[0.994524788535891, 1.2444069406266733, 4.7213707658494, 4.197344796153198, 1.2221142052396945, 6.665173538916106, 1.9376392047484152, 6.016878328048576, 28.653541566309578] *****
[0 4 1 6]
prediction 1
prediction 2
prediction 3
prediction 4
prediction 5
prediction 6
prediction 7
prediction 8
prediction 9
prediction 10
prediction 11
prediction 12
prediction 13
prediction 14
prediction 15
prediction 16
prediction 17
prediction 18
prediction 19
prediction 20
prediction 21
prediction 22
prediction 23
prediction 24
prediction 25
window 30
[0.12270195287938806, 0.035685313958096126, 6.1816722525254335, 6.9480150106955945, 0.7030619602936604, 19.314746109492834, 3.6229739981878306, 1.2603289698715956, 933.0445879992634] *****
[1 0 4 7]
prediction 1
prediction 2
prediction 3
prediction 4
prediction 5
prediction 6
prediction 7
prediction 8
prediction 9
prediction 10
prediction 11
prediction 12
prediction 13
prediction 14
prediction 15
prediction 16
C:\Users\HP\anaconda3\envs\env_torch\lib\site-packages\sklearn\base.py:315: UserWarning: Trying to unpickle estimator MinMaxScaler from version 0.23.1 when using version 0.24.1. This might lead to breaking code
or invalid results. Use at your own risk.
  UserWarning)
9.008488468897775

```

Fig 25: Output obtained for simulate.py

Step 5) Tkinter is the UI used, which shows the MAPE value obtained after selecting the predictor. The figure 26 and 27 demonstrates the code snippet and output obtained for the same.

```

grid500 > gui.py
1  from tkinter import *
2  from tkinter.filedialog import askopenfilename
3  from tkinter import messagebox,DISABLED,NORMAL
4  # import pymysql
5  import datetime
6  from functools import partial
7  from PIL import Image, ImageTk
8  # from testing import prediction
9  import time
10 title="Workload Prediction in cloud"
11
12 main_color='#271745'
13
14 def logcheck():
15     global username_var,pass_var
16     uname=username_var.get()
17     pass1=pass_var.get()
18     if uname=="admin" and pass1=="admin":
19         showcheck()
20     else:
21         messagebox.showinfo("alert","Wrong Credentials")
22

```

Loading libraries for tkinter (gui unit)

Login page and credentials setup

```

23 # show nome page
24
25
26 def showcheck():
27     top.title(title)
28     top.config(menu=menubar)
29     global f
30     f.pack_forget()
31     f=Frame(top)
32     f.config(bg=main_color)
33     f.pack(side="top", fill="both", expand=True,padx=10,pady=10)
34
35
36     f3=Frame(f)
37     f3.pack_propagate(False)
38     f3.config(bg=main_color,width=600)
39     f3.pack(side="right",fill="both")
40
41     f4=Frame(f3)
42     f4.pack_propagate(False)
43     f4.config(bg=main_color,height=200)
44     f4.pack(side="bottom",fill="both")
45
46     f7=Frame(f3)
47     f7.pack_propagate(False)
48     f7.config(height=20)
49     f7.pack(side="top",fill="both",padx="3")
50
51     l2=Label(f7,text="Process",font="Helvetica 13 bold")
52     l2.pack()
53
54     global lb1
55     b2=Button(f4,text="Start",font="Verdana 10 bold",command=process1)
56     b2.pack(pady=2)
57
58
59     lb1=Listbox(f3,width=400,height=100,font="Helvetica 13 bold")
60     lb1.pack(pady=10,padx=5)

```

Features of box in output

```

65 from simulate import run
66
67
68 def process1():
69     global lb1
70     lb1.delete(0,'end')
71     lb1.after(10,delayed_insert,lb1,0,'Starting..')
72     lb1.after(10,delayed_insert,lb1,0,'Selecting models')
73     lb1.after(10,delayed_insert,lb1,0,'make predictions')
74     #lb1.after(10,delayed_insert,lb1,3,'Done')
75     lb1.update()
76     mape=run()
77     lb1.after(10,delayed_insert,lb1,1,str(mape)+' percentage')
78
79
80
81
82
83
84 def delayed_insert(label,index,message):
85     label.insert(index,message)
86

```

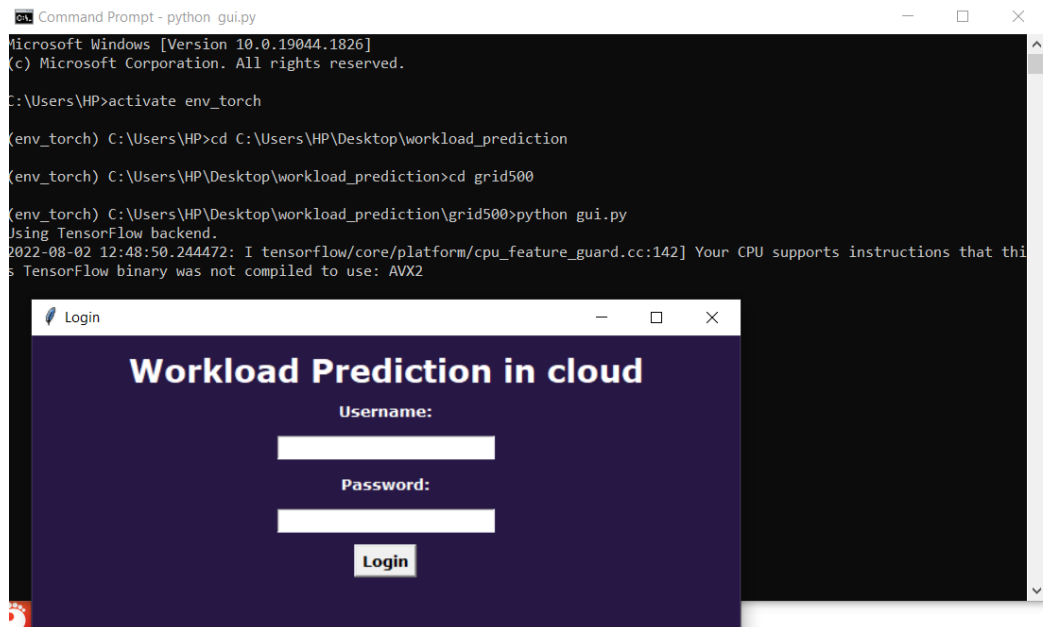
Calling the function run from simulate.py

```

grid500 > gui.py
87
88 if __name__ == "__main__":
89
90     top = Tk()
91     top.title("Login")
92     top.geometry("600x500")
93     footer = Frame(top, bg='grey', height=30)
94     footer.pack(fill='both', side='bottom')
95
96     lab1=Label(footer,text="Developed by ###",font = "Verdana 8 bold",fg="white",bg="grey")
97     lab1.pack()
98
99     menubar = Menu(top)
100     # menubar.add_command(label="Home",command=showhome)
101     menubar.add_command(label="Check",command=showcheck)
102     top.config(bg=main_color,relief=RAISED)
103     f=Frame(top)
104     f.config(bg=main_color)
105     f.pack(side="top", fill="both", expand=True,padx=10,pady=10)
106     l=Label(f,text=title,font = "Verdana 20 bold",fg="white",bg=main_color)
107     l.pack()
108     l2=Label(f,text="Username:",font="Verdana 10 bold",bg=main_color,fg="white")
109     l2.pack(pady=5)
110     global username_var
111     username_var=StringVar()
112     e1=Entry(f,textvariable=username_var,font="Verdana 10 bold")
113     e1.pack(pady=5)
114
115     l3=Label(f,text="Password:",font="Verdana 10 bold",bg=main_color,fg="white")
116     l3.pack(pady=5)
117     global pass_var
118     pass_var=StringVar()
119     e2=Entry(f,textvariable=pass_var,font="Verdana 10 bold",show="*")
120     e2.pack(pady=5)
121
122     b1=Button(f,text="Login", command=logcheck,font="Verdana 10 bold")
123     b1.pack(pady=5)
124     top.mainloop()

```

Fig 26: Code snippet for GUI



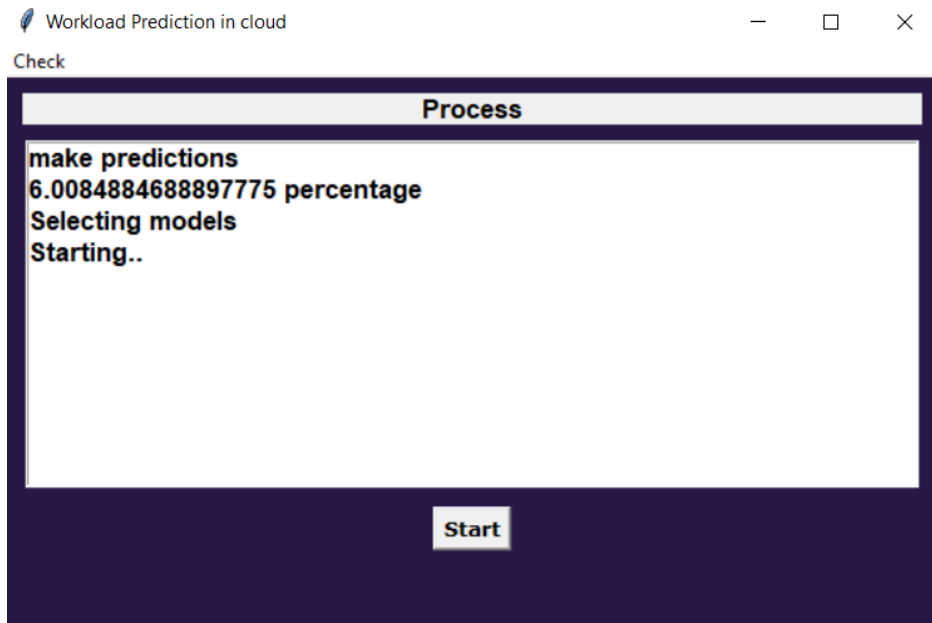


Fig 27: Output obtained for gui.py

Step 6) Similarly, repeat the steps from 2. Use the datasets LCG and NORDU from [2] instead of Grid 5000.

References

[1] Y. Q. I. K. Kim, W. Wang and M. Humphrey, "Forecasting cloud application workloads with cloud- insight for predictive resource management," IEEE Trans. Cloud Comput., pp. 1–16, May 2020, doi:10.1109/TCC.2020.2998017. JCR Impact Factor 2021: 5.938.

[2] Dataset; TU Delft. The Grid Workloads Archive available online on "<http://gwa.ewi.tudelft.nl/datasets/?msclkid=d9112cc7b2ea11eca7067512037bc5d6>".