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)	□
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

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”.

```python
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 libraraies

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

```
# Generated by get-clean-log.py ($Revision: 0.1$) 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: JobId<TAB>SubmitTime<TAB>WaitTime<TAB>RunTime<TAB>NProc<TAB>AverageCPUTimeUsed<TAB>UsedMemory<TAB>ReqNProcs<TAB>ReqTime<TAB>ReqMemory<TAB>Status<TAB>UserId<TAB>Grou

0	1083658801	1	0	4	-1	-1	4	3600	-1	1	user386 group4	app34	queue0	-1	G1/site4	G1/site6/c1	UNITARY -1	-1
1	1083658849	1	19	1	-1	-1	1	3600	-1	1	user112 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
2	1083658875	2	10	5	-1	-1	5	3600	-1	1	user112 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
3	1083658891	5	8	90	-1	-1	90	3600	-1	1	user112 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
4	1083658911	5	19	100	-1	-1	100	3600	-1	1	user112 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
5	1083658944	1	25	1	-1	-1	1	3600	-1	0	user112 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
6	1083659210	1	6	1	-1	-1	1	3600	-1	1	user568 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
7	1083659322	1	43205	4	-1	-1	4	43200	-1	0	user386 group4	app0	queue0	-1	G1/site4	G1/site6/c1	UNITARY -1	-1
8	1083659636	1	5	1	-1	-1	1	3600	-1	1	user568 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
9	1083660389	-1	-1	4	-1	-1	4	9000	-1	0	user267 group5	app507	queue48	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
10	1083660523	2	156	7	-1	-1	7	18000	-1	1	user569 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
11	1083660693	1	19	7	-1	-1	7	18000	-1	1	user569 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
12	1083660719	1	4	7	-1	-1	7	18000	-1	1	user569 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
13	1083660726	1	2801	7	-1	-1	7	18000	-1	1	user569 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
14	1083660777	1	1	4	-1	-1	4	3600	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
15	1083660832	1	0	4	-1	-1	4	3600	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
16	1083660933	1	0	4	-1	-1	4	3600	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
17	1083661197	1	20992	1	-1	-1	1	36000	-1	1	user570 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
18	1083661769	1	23027	1	-1	-1	1	43200	-1	1	user571 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
19	1083661777	1	23014	1	-1	-1	1	43200	-1	1	user571 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
20	1083662072	1	22216	1	-1	-1	1	28800	-1	1	user67	group2	app0	queue0	-1	G1/site2	G1/site6/c1	UNITARY -1	-1
21	1083663533	1	13734	10	-1	-1	10	18000	-1	0	user569 group6	app0	queue0	-1	G1/site6	G1/site6/c1	UNITARY -1	-1
22	1083669430	1	1	4	-1	-1	4	3600	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
23	1083669460	1	1	4	-1	-1	4	3600	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
24	1083669628	1	1	4	-1	-1	4	3600	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
25	1083669739	1	1	1	-1	-1	1	3600	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
26	1083669841	1	0	1	-1	-1	1	9000	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
27	1083670043	1	1	4	-1	-1	4	9000	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
28	1083670602	1	1	4	-1	-1	4	9000	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
29	1083670874	1	1	4	-1	-1	4	9000	-1	1	user267 group5	app507	queue0	-1	G1/site5	G1/site6/c1	UNITARY -1	-1
30	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”.

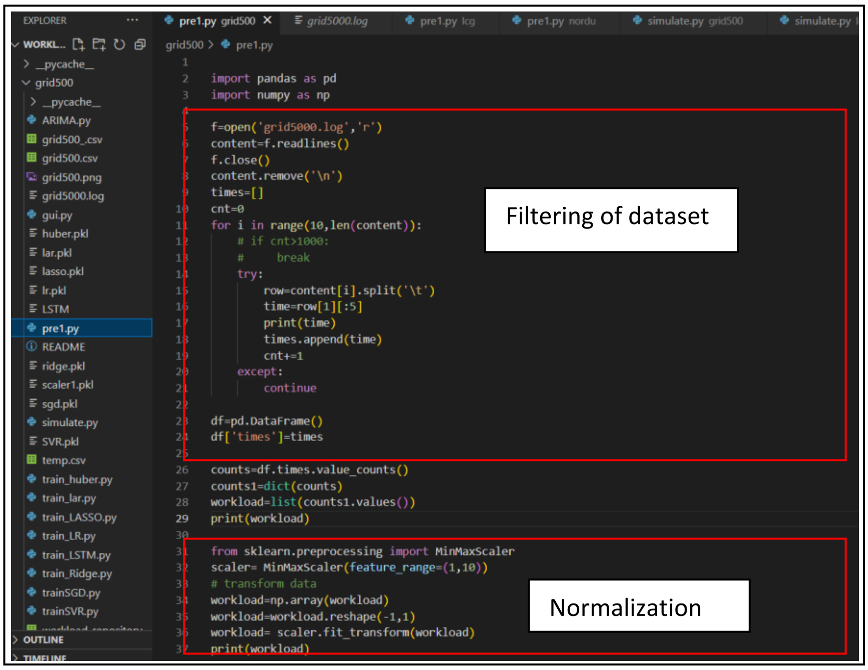


Fig 3: Snippet for Preprocessing of the dataset

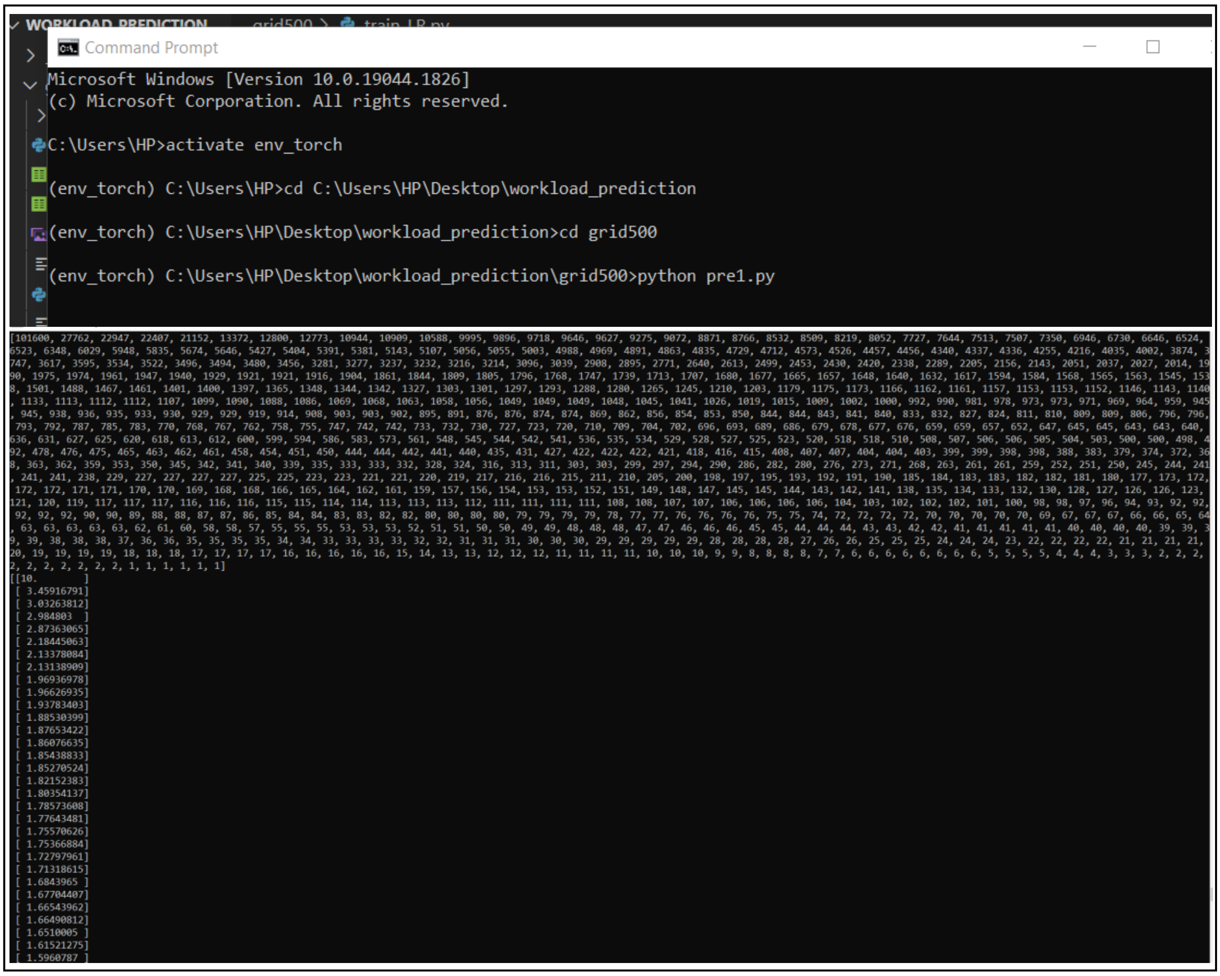
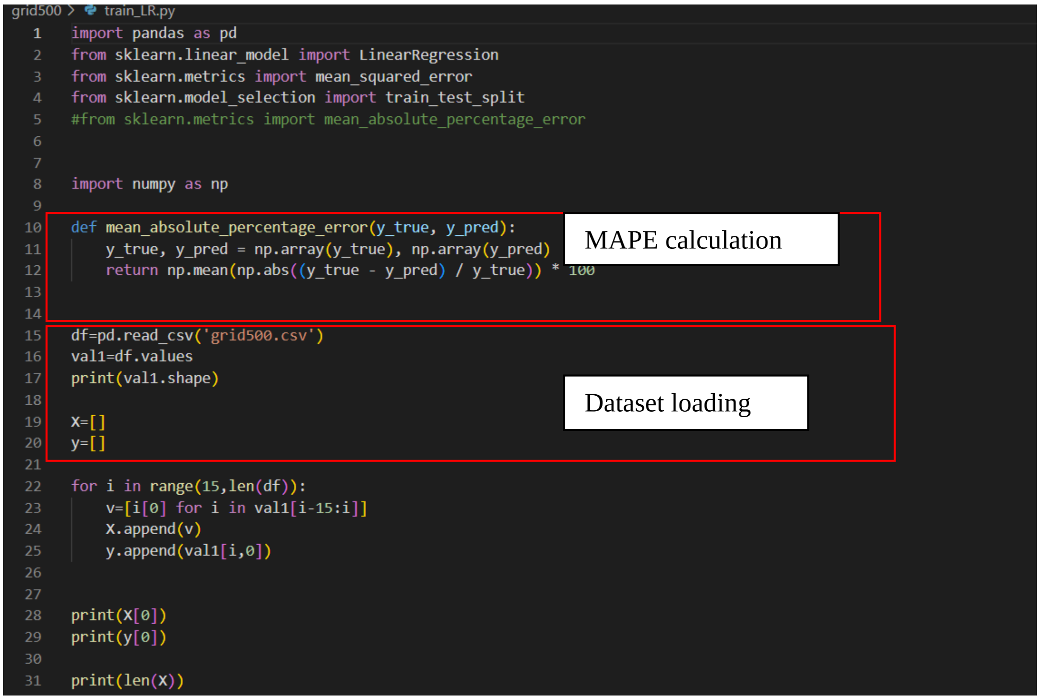


Fig 4: Output obtained for pre-processed dataset (filtering and normalization)

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.




```
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 sklearn.metrics import mean_absolute_percentage_error


import numpy as np

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


df=pd.read_csv('grid500.csv')
val1=df.values
print(val1.shape)

X=[]
y=[]

for i in range(15,len(df)):
    v=[i[0] for i in val1[i-15:i]]
    X.append(v)
    y.append(val1[i,0])


print(X[0])
print(y[0])

print(len(X))
```

```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

reg =  make_pipeline(StandardScaler(), LinearRegression())
reg.fit(x_train, y_train)

y_pred=reg.predict(x_test)

print(y_pred[:10])
print(y_train[:10])

mse=mean_absolute_percentage_error(y_test, y_pred)

print(mse)

import pickle

pickle.dump(reg,open('lr.pkl','wb'))
```

Training and testing of predictor

Fig 6 : LR code snippet

```
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python train_LR.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, 10.7383
43881337414, 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]
[1.1364186655380466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
0.3321473673606518
```

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.

```python
    from sklearn import linear_model
    from sklearn.pipeline import make_pipeline
    from sklearn.preprocessing import StandardScaler

    x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

    reg =  make_pipeline(StandardScaler(), linear_model.Lasso(alpha=0.1))
    reg.fit(x_train, y_train)

    y_pred=reg.predict(x_test)

    print(y_pred[:10])
    print(y_train[:10])

    mse=mean_absolute_percentage_error(y_test, y_pred)

    print(mse)

    import pickle

    pickle.dump(reg,open('lasso.pkl','wb'))


if __name__=="__main__":
    train('grid500.csv')
```

Fig 8: LASSO code snippet

```
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python train_LASSO.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, 10.7383
43881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
C:\Users\HP\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. Dualit
y gap: 0.3476593733219051, tolerance: 0.1385032330310563
  positive)
[1.089503   2.43081754 2.51672886 7.59990867 1.11173994 2.65756886
 1.45257145 1.12601605 1.30984328 1.87413623]
[1.1364186655380466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
3.8035052367961906
```

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.

```python
from sklearn import linear_model
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

reg =  make_pipeline(StandardScaler(), linear_model.Ridge(alpha=.5))
reg.fit(x_train, y_train)

y_pred=reg.predict(x_test)

print(y_pred[:10])
print(y_train[:10])

mse=mean_absolute_percentage_error(y_test, y_pred)

print(mse)

import pickle

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, 10.7383
43881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
[1.02805785 2.4883846  2.57779845 7.99904351 1.05238492 2.73128765
 1.42388593 1.06781612 1.26644943 1.88546705]
[1.1364186655380466, 2.5425053396194843, 1.7376352129450092, 2.112786543174637, 1.1481116940127365, 1.9383655350938491, 1.5105955767281172, 1.4862351007391803, 2.8660124607525663, 1.0935442277975176]
0.6107906828523219
```

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.

```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

reg = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
reg.fit(x_train, y_train)

y_pred=reg.predict(x_test)

print(y_pred[:10])
print(y_train[:10])

mse=mean_absolute_percentage_error(y_test, y_pred)

print(mse)

import pickle


pickle.dump(reg,open('SVR.pkl','wb'))

```

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, 10.7383
43881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
[1.1944079  2.47915516 2.59831207 7.79932449 1.18934064 2.7596937
 1.27985006 1.18695493 1.20730206 1.71141019]
[1.1364186655380466, 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.

```
from sklearn import linear_model
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

reg =  make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))
reg.fit(x_train, y_train)

y_pred=reg.predict(x_test)

print(y_pred[:10])
print(y_train[:10])

mse=mean_absolute_percentage_error(y_test, y_pred)

print(mse)

import pickle

pickle.dump(reg,open('sgd.pkl','wb'))
```

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, 10.7383
43881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
[1.05637976 2.4218249  2.5172306  7.97835927 1.0789769  2.66142713
 1.42371326 1.09315318 1.28385949 1.84440914]
[1.1364186655380466, 2.5425053396194843, 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.

```
from sklearn import linear_model
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

reg =  make_pipeline(StandardScaler(), linear_model.LassoLars(alpha=.1, normalize=False))
reg.fit(x_train, y_train)

y_pred=reg.predict(x_test)

print(y_pred[:10])
print(y_train[:10])

mse=mean_absolute_percentage_error(y_test, y_pred)

print(mse)

import pickle

pickle.dump(reg,open('lar.pkl','wb'))
```

Fig 16: LAR code snippet

```
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python train_lar.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, 10.7383
43881337414, 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.

```python
    from sklearn import linear_model
    from sklearn.linear_model import HuberRegressor
    from sklearn.pipeline import make_pipeline
    from sklearn.preprocessing import StandardScaler

    x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

    reg =  make_pipeline(StandardScaler(), HuberRegressor())
    reg.fit(x_train, y_train)

    y_pred=reg.predict(x_test)

    print(y_pred[:10])
    print(y_train[:10])

    mse=mean_absolute_percentage_error(y_test, y_pred)

    print(mse)

    import pickle

    pickle.dump(reg,open('huber.pkl','wb'))


if __name__=="__main__":
    train('grid500.csv')
```

Fig 18: HUBER code snippet

```
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python train_huber.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, 10.7383
43881337414, 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 2.50909335 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.

```python
grid500 > ARIMA.py
from matplotlib import pyplot
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
import numpy as np

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


def train(path):
    df = read_csv(path)

    X = df.values
    size = int(len(X) * 0.66)
    train, test = X[0:size], X[size:len(X)]
    history = [x for x in train]
    predictions = list()
    # walk-forward validation
    for t in range(len(test)):
        model = ARIMA(history, order=(1,1,1))
        model_fit = model.fit()
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat)
        obs = test[t]
        history.append(obs)
        print('predicted=%f, expected=%f' % (yhat, obs))

    mape = mean_absolute_percentage_error(test, predictions)
    print('MAPE: %.3f' % mape)


if __name__=="__main__":
    path='grid500.csv'
    train(path)
```

Fig 20: ARIMA code snippet

```
(env_torch) C:\Users\HP\Desktop\workload_prediction\grid500>python ARIMA.py
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

```python
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
from sklearn.metrics import mean_squared_error
#from sklearn.metrics import mean_absolute_percentage_error


import numpy as np

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


df=pd.read_csv('grid500.csv')
val1=df.values
print(val1.shape)

X=[]
y=[]

for i in range(15,len(df)):
    v=[i[0] for i in val1[i-15:i]]
    X.append(v)
    y.append(val1[i,0])


print(X[0])
print(y[0])

print(len(X))
```

```python
from sklearn import linear_model
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=True,test_size=0.2)

x_train=np.array(x_train)
x_test=np.array(x_test)
y_train=np.array(y_train)
y_test=np.array(y_test)

x_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
x_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))


model = Sequential()
model.add(LSTM(20, input_shape=(1,15)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=100, batch_size=1, verbose=2)

y_pred=model.predict(x_test)

print(y_pred[:10])
print(y_train[:10])

mse=mean_absolute_percentage_error(y_test, y_pred)

print(mse)

import pickle

model.save('LSTM')
```

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, 10.7383
43881337414, 10.641876396421225, 10.468429807379993, 10.398271636531854]
10.379757674780263
766
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
 - 2s - 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: 0.0313
```

```
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.

```python
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 sklearn.metrics import mean_absolute_percentage_error
import pickle
import numpy as np
from keras.models import load_model
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

lr=pickle.load(open('lr.pkl','rb'))
huber=pickle.load(open('huber.pkl','rb'))
lar=pickle.load(open('lar.pkl','rb'))
lasso=pickle.load(open('lasso.pkl','rb'))
ridge=pickle.load(open('ridge.pkl','rb'))
sgd=pickle.load(open('sgd.pkl','rb'))
lstm=load_model('LSTM')
svr=pickle.load(open('SVR.pkl','rb'))

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

ensemble_list=[0,1,2,3

def ensemble(feat):
    global ensemble_list
    val=0
    for i in ensemble_list:
        val+=feat[i]
    val=val/4
    return val
```

Loading the models created by each predictor

Defining list for 4 ensemble models

```python
def run():
    global ensemble_list
    df=pd.read_csv('grid500.csv')
    val1=df.values
    print(val1.shape)
    X=[]
    y=[]

    for i in range(15,len(df)):
        v=[i[0] for i in val1[i-15:i]]
        X.append(v)
        y.append(val1[i,0])

    history=val1[:15]

    #print(y)
    #return

    final_predictions=[]
    final_originals=[]

    cnt=0
    lst1=[]
    lst2=[]
    lst3=[]
    lst4=[]
    lst5=[]
    lst6=[]
    lst7=[]
    lst8=[]
    lst9=[]
    lst10=[]
    cnt2=1
```

Window declaration as 1

```
    for i in range(len(X)):

        feat=[]
        original_label=y[i]
        lst1.append(original_label)
        final_originals.append(original_label)
        #1
        pred_lr=lr.predict(X[i:i+1])[0]
        lst2.append(pred_lr)
        feat.append(pred_lr)
        #2
        pred_huber=huber.predict(X[i:i+1])[0]
        lst3.append(pred_huber)
        feat.append(pred_huber)
        #3
        pred_lar=lar.predict(X[i:i+1])[0]
        lst4.append(pred_lar)
        feat.append(pred_lar)
        #4
        pred_lasso=lasso.predict(X[i:i+1])[0]
        lst5.append(pred_lasso)
        feat.append(pred_lasso)
        #5
        pred_ridge=ridge.predict(X[i:i+1])[0]
        lst6.append(pred_ridge)
        feat.append(pred_ridge)
        #6
        pred_svr=svr.predict(X[i:i+1])[0]
        lst7.append(pred_svr)
        feat.append(pred_svr)
        #7
        pred_sgd=sgd.predict(X[i:i+1])[0]
        lst8.append(pred_sgd)
        feat.append(pred_sgd)
        #8 (LSTM)
        pred_lstm=lstm.predict(np.reshape(X[i:i+1], (1, 1, 15)))[0][0]
        lst9.append(pred_lstm)
        feat.append(pred_lstm)
        #9
        model = ARIMA(history, order=(1,1,1))
        model_fit = model.fit()
        output = model_fit.forecast()
        pred_arima = output[0]
        lst10.append(pred_arima)
        feat.append(pred_arima)
```

Initialization of each list by each predictor

```python
        cnt+=1
        val=ensemble(feat)
        final_predictions.append(val)
        print('prediction',cnt)
        if cnt>=25:
            history=history[-15:]
            print('window',cnt2)
            cnt2+=1
            cnt=0
            dict1={'lr':lst2,'huber':lst3,'lar':lst4,'lasso':lst5,'ridge':lst6,'svr':lst7,'sgd':lst8,'lstm':lst9,'arima':lst10,'original':lst1}
            # print(dict1)
            df=pd.DataFrame(dict1)
            df.to_csv('temp.csv',index=False)
```

Each window have 25 predictions (variable ; cnt)

Predictions are saved to temp.csv (workload repository)

```python
            mape1=mean_absolute_percentage_error(lst1,lst2)
            mape2=mean_absolute_percentage_error(lst1,lst3)
            mape3=mean_absolute_percentage_error(lst1,lst4)
            mape4=mean_absolute_percentage_error(lst1,lst5)
            mape5=mean_absolute_percentage_error(lst1,lst6)
            mape6=mean_absolute_percentage_error(lst1,lst7)
            mape7=mean_absolute_percentage_error(lst1,lst8)
            mape8=mean_absolute_percentage_error(lst1,lst9)
            mape9=mean_absolute_percentage_error(lst1,lst10)
```

MAPE Comparison

```python
            mape_list=[mape1,mape2,mape3,mape4,mape5,mape6,mape7,mape8,mape9]
            print(mape_list,'==========')
            mape_arr=np.array(mape_list)
            mape_arr = np.argsort(mape_arr)[:4]
            mape_list=list(mape_arr)
            ensemble_list=mape_list
```

MAPE sorting for the predictors

```python
            print(mape_arr)
```

Print the list of 4 ensemble model with least MAPE

```python
            lst1=[]
            lst2=[]
            lst3=[]
            lst4=[]
            lst5=[]
            lst6=[]
            lst7=[]
            lst8=[]
            lst9=[]
            lst10=[]


    mape=mean_absolute_percentage_error(final_originals,final_predictions)*10
    df2=pd.DataFrame()
    import pickle
    scaler=pickle.load(open('scaler1.pkl','rb'))

    final_originals=np.array(final_originals)
    final_originals=final_originals.reshape(-1,1)
    final_originals= scaler.inverse_transform(final_originals)

    final_predictions=np.array(final_predictions)
    final_predictions=final_predictions.reshape(-1,1)
    final_predictions= scaler.inverse_transform(final_predictions)


    df2['true']=final_originals.flatten()
    df2['prediction']=final_predictions.flatten()

    df2['step']=list(range(0,len(list(final_predictions))))

    df2.plot(x='step',y=['prediction','true'])
    plt.savefig("grid500.png")
```

Plotting actual and predicted values and saving the plot

```python
    print(mape)
    return mape




if __name__=="__main__":
    run()
```

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
[0.9945247885535891, 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 25
window 30
[0.12370199287938806, 0.035685313958096126, 6.1816722552354335, 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)
6.0084884688897775
```

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

```python
from tkinter import *
from tkinter.filedialog import askopenfilename
from tkinter import messagebox,DISABLED,NORMAL
# import pymysql
import datetime
from functools import partial
from PIL import Image, ImageTk
# from testing import prediction
import time
title="Workload Prediction in cloud"

main_color='#271745'

def logcheck():
    global username_var,pass_var
    uname=username_var.get()
    pass1=pass_var.get()
    if uname=="admin" and pass1=="admin":
        showcheck()
    else:
        messagebox.showinfo("alert","Wrong Credentials")
```

Loading libraries for tkinter (gui unit)

Login page and credentials setup

grid500 > gui.py

```python
# show home page


def showcheck():
    top.title(title)
    top.config(menu=menubar)
    global f
    f.pack_forget()
    f=Frame(top)
    f.config(bg=main_color)
    f.pack(side="top", fill="both", expand=True,padx=10,pady=10)


    f3=Frame(f)
    f3.pack_propagate(False)
    f3.config(bg=main_color,width=600)
    f3.pack(side="right",fill="both")

    f4=Frame(f3)
    f4.pack_propagate(False)
    f4.config(bg=main_color,height=200)
    f4.pack(side="bottom",fill="both")

    f7=Frame(f3)
    f7.pack_propagate(False)
    f7.config(height=20)
    f7.pack(side="top",fill="both",padx="3")

    l2=Label(f7,text="Process",font="Helvetica 13 bold")
    l2.pack()

    global lb1
    b2=Button(f4,text="Start",font="Verdana 10 bold",command=process1)
    b2.pack(pady=2)


    lb1=Listbox(f3,width=400,height=100,font="Helvetica 13 bold")
    lb1.pack(pady=10,padx=5)
```

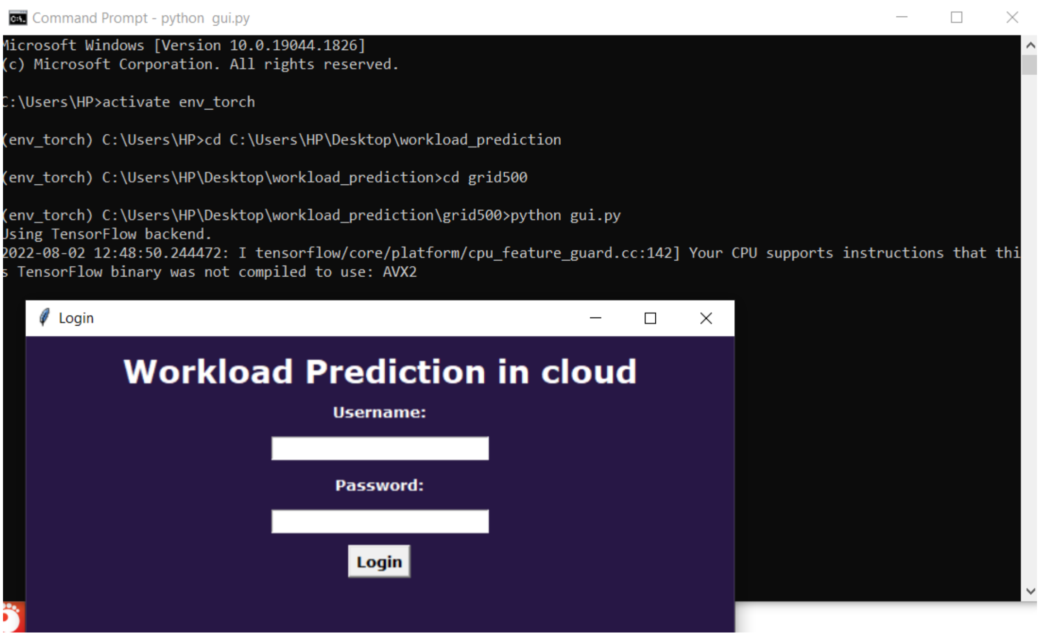
Features of box in output

```python
from simulate import run


def process1():
    global lb1
    lb1.delete(0,'end')
    lb1.after(10,delayed_insert,lb1,0,'Starting..')
    lb1.after(10,delayed_insert,lb1,0,'Selecting models')
    lb1.after(10,delayed_insert,lb1,0,'make predictions')
    #lb1.after(10,delayed_insert,lb1,3,'Done')
    lb1.update()
    mape=run()

    lb1.after(10,delayed_insert,lb1,1,str(mape)+' percentage')





def delayed_insert(label,index,message):
    label.insert(index,message)
```

Calling the function run from simulate.py grid500 > gui.py

```python

if __name__=="__main__":

    top = Tk()
    top.title("Login")
    top.geometry("600x500")
    footer = Frame(top, bg='grey', height=30)
    footer.pack(fill='both', side='bottom')

    lab1=Label(footer,text="Developed by ###",font = "Verdana 8 bold",fg="white",bg="grey")
    lab1.pack()

    menubar = Menu(top)
    # menubar.add_command(label="Home",command=showhome)
    menubar.add_command(label="Check",command=showcheck)
    top.config(bg=main_color,relief=RAISED)
    f=Frame(top)
    f.config(bg=main_color)
    f.pack(side="top", fill="both", expand=True,padx=10,pady=10)
    l=Label(f,text=title,font = "Verdana 20 bold",fg="white",bg=main_color)
    l.pack()
    l2=Label(f,text="Username:",font="Verdana 10 bold",bg=main_color,fg="white")
    l2.pack(pady=5)
    global username_var
    username_var=StringVar()
    e1=Entry(f,textvariable=username_var,font="Verdana 10 bold")
    e1.pack(pady=5)

    l3=Label(f,text="Password:",font="Verdana 10 bold",bg=main_color,fg="white")
    l3.pack(pady=5)
    global pass_var
    pass_var=StringVar()
    e2=Entry(f,textvariable=pass_var,font="Verdana 10 bold",show="*")
    e2.pack(pady=5)

    b1=Button(f,text="Login", command=logcheck,font="Verdana 10 bold")
    b1.pack(pady=5)
    top.mainloop()
```

Fig 26: Code snippet for GUI



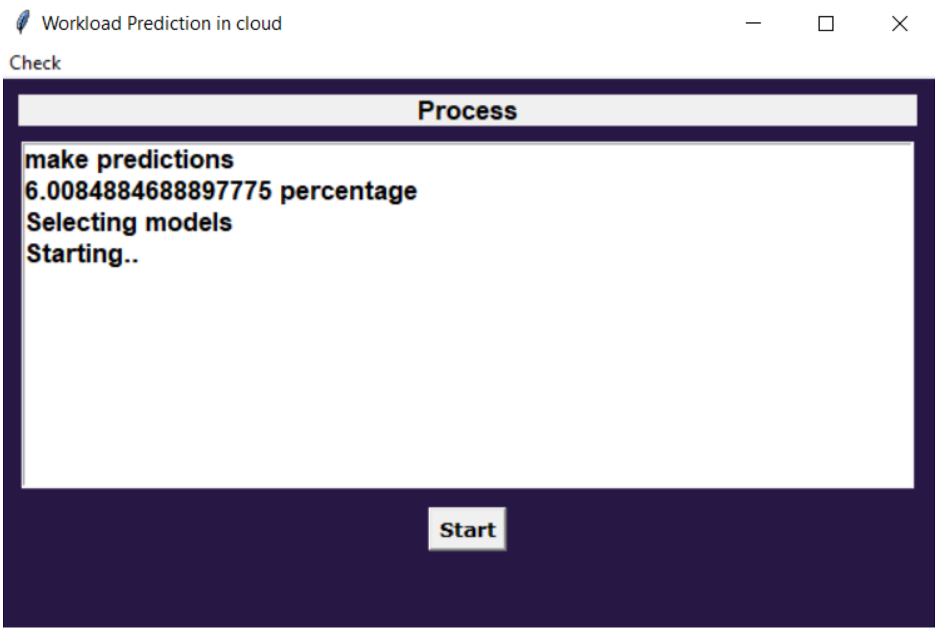


Fig 27: Output obtained for gui.py

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 " .