Table of Contents

- Data Cleaning
- Univariate Data Analysis
- Data Frequency
- Binning the Time-Series Data Behaviors
- Exploring Relationships between Independent Variables (Intra-Inter Relation) (Bivariate Optimization)
- Predicting Sound by using the Inputs as Series
- Predicting Sound Level Pressure for Deployment DATA

Data Cleaning

```
import pandas as pd
import numpy as np
df = pd.read_csv('/content/airfoil_self_noise.dat', header=None)
df.shape
```

```
(1503, 6)
```

```
df.head()
df.tail()
df.info()
df.describe()
```

	0	1	2	3	4	5
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.sample(5)
```

	0	1	2	3	4	5
1	800	5000	0.050000	39.6000000000000	0.002000	125.201000000000
2	1000	2000	0.090000	39.6000000000000	0.002000	123.943000000000
3	1250	2000	0.090000	39.6000000000000	0.002000	124.365000000000
4	1600	2000	0.090000	39.6000000000000	0.002000	125.797000000000
5	2000	2000	0.090000	39.6000000000000	0.002000	127.591000000000
6	2500	2000	0.090000	39.6000000000000	0.002000	129.039000000000
7	3150	2000	0.090000	39.6000000000000	0.002000	130.474000000000
8	4000	2000	0.090000	39.6000000000000	0.002000	132.080000000000
9	5000	2000	0.090000	39.6000000000000	0.002000	133.474000000000
10	6300	2000	0.090000	39.6000000000000	0.002000	135.080000000000
11	8000	2000	0.090000	39.6000000000000	0.002000	136.474000000000
12	10000	2000	0.090000	39.6000000000000	0.002000	138.080000000000
13	12500	2000	0.090000	39.6000000000000	0.002000	139.474000000000
14	16000	2000	0.090000	39.6000000000000	0.002000	141.080000000000
15	20000	2000	0.090000	39.6000000000000	0.002000	142.474000000000
16	800	2500	0.090000	39.6000000000000	0.002000	124.365000000000
17	1000	2500	0.090000	39.6000000000000	0.002000	125.797000000000
18	1250	2500	0.090000	39.6000000000000	0.002000	127.591000000000
19	1600	2500	0.090000	39.6000000000000	0.002000	129.039000000000
20	2000	2500	0.090000	39.6000000000000	0.002000	130.474000000000
21	2500	2500	0.090000	39.6000000000000	0.002000	132.080000000000
22	3150	2500	0.090000	39.6000000000000	0.002000	133.474000000000
23	4000	2500	0.090000	39.6000000000000	0.002000	135.080000000000
24	5000	2500	0.090000	39.6000000000000	0.002000	136.474000000000
25	6300	2500	0.090000	39.6000000000000	0.002000	138.080000000000
26	8000	2500	0.090000	39.6000000000000	0.002000	139.474000000000
27	10000	2500	0.090000	39.6000000000000	0.002000	141.080000000000
28	12500	2500	0.090000	39.6000000000000	0.002000	142.474000000000
29	16000	2500	0.090000	39.6000000000000	0.002000	144.080000000000
30	20000	2500	0.090000	39.6000000000000	0.002000	145.474000000000
31	800	3150	0.090000	39.6000000000000	0.002000	125.797000000000
32	1000	3150	0.090000	39.6000000000000	0.002000	127.591000000000
33	1250	3150	0.090000	39.6000000000000	0.002000	129.039000000000
34	1600	3150	0.090000	39.6000000000000	0.002000	130.474000000000
35	2000	3150	0.090000	39.6000000000000	0.002000	132.080000000000
36	2500	3150	0.090000	39.6000000000000	0.002000	133.474000000000
37	3150	3150	0.090000	39.6000000000000	0.002000	135.080000000000
38	4000	3150	0.090000	39.6000000000000	0.002000	136.474000000000
39	5000	3150	0.090000	39.6000000000000	0.002000	138.080000000000
40	6300	3150	0.090000	39.6000000000000	0.002000	139.474000000000
41	8000	3150	0.090000	39.6000000000000	0.002000	141.080000000000
42	10000	3150	0.090000	39.6000000000000	0.002000	142.474000000000
43	12500	3150	0.090000	39.6000000000000	0.002000	144.080000000000
44	16000	3150	0.090000	39.6000000000000	0.002000	145.474000000000
45	20000	3150	0.090000	39.6000000000000	0.002000	147.080000000000
46	800	4000	0.090000	39.6000000000000	0.002000	127.591000000000
47	1000	4000	0.090000	39.6000000000000	0.002000	129.039000000000
48	1250	4000	0.090000	39.6000000000000	0.002000	130.474000000000
49	1600	4000	0.090000	39.6000000000000	0.002000	132.080000000000
50	2000	4000	0.090000	39.6000000000000	0.002000	133.474000000000
51	2500	4000	0.090000	39.6000000000000	0.002000	135.080000000000
52	3150	4000	0.090000	39.6000000000000	0.002000	136.474000000000
53	4000	4000	0.090000	39.6000000000000	0.002000	138.080000000000
54	5000	4000	0.090000	39.6000000000000	0.002000	139.474000000000
55	6300	4000	0.090000	39.6000000000000	0.002000	141.080000000000
56	8000	4000	0.090000	39.6000000000000	0.002000	142.474000000000
57	10000	4000	0.090000	39.6000000000000	0.002000	144.080000000000
58	12500	4000	0.090000	39.6000000000000	0.002000	145.474000000000
59	16000	4000	0.090000	39.6000000000000	0.002000	147.080000000000
60	20000	4000	0.090000	39.6000000000000	0.002000	148.474000000000
61	800	5000	0.090000	39.6000000000000	0.002000	129.039000000000
62	1000	5000	0.090000	39.6000000000000	0.002000	130.474000000000
63	1250	5000	0.090000	39.6000000000000	0.002000	132.080000000000
64	1600	5000	0.090000	39.6000000000000	0.002000	133.474000000000
65	2000	5000	0.090000	39.6000000000000	0.002000	135.080000000000
66	2500	5000	0.090000	39.6000000000000	0.002000	136.474000000000
67	3150	5000	0.090000	39.6000000000000	0.002000	138.080000000000
68	4000	5000	0.090000	39.6000000000000	0.002000	139.474000000000
69	5000	5000	0.090000	39.6000000000000	0.002000	141.080000000000
70	6300	5000	0.090000	39.6000000000000	0.002000	142.474000000000
71	8000	5000	0.090000	39.6000000000000	0.002000	144.080000000000
72	10000	5000	0.090000	39.6000000000000	0.002000	145.474000000000
73	12500	5000	0.090000	39.6000000000000	0.002000	147.080000000000
74	16000	5000	0.090000	39.6000000000000	0.002000	148.474000000000
75	20000	5000	0.090000	39.6000000000000	0.002000	150.080000000000

```
df.dtypes
```

```
0    int64
1    int64
2    int64
3    int64
4    int64
5    int64
dtype: object
```

```
df.isnull()
```

	0	1	2	3	4	5
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False
11	False	False	False	False	False	False
12	False	False	False	False	False	False
13	False	False	False	False	False	False
14	False	False	False	False	False	False
15	False	False	False	False	False	False
16	False	False	False	False	False	False
17	False	False	False	False	False	False
18	False	False	False	False	False	False
19	False	False	False	False	False	False
20	False	False	False	False	False	False
21	False	False	False	False	False	False
22	False	False	False	False	False	False
23	False	False	False	False	False	False
24	False	False	False	False	False	False
25	False	False	False	False	False	False
26	False	False	False	False	False	False
27	False	False	False	False	False	False
28	False	False	False	False	False	False
29	False	False	False	False	False	False
30	False	False	False	False	False	False
31	False	False	False	False	False	False
32	False	False	False	False	False	False
33	False	False	False	False	False	False
34	False	False	False	False	False	False
35	False	False	False	False	False	False
36	False	False	False	False	False	False
37	False	False	False	False	False	False
38	False	False	False	False	False	False
39	False	False	False	False	False	False
40	False	False	False	False	False	False
41	False	False	False	False	False	False
42	False	False	False	False	False	False
43	False	False	False	False	False	False
44	False	False	False	False	False	False
45	False	False	False	False	False	False
46	False	False	False	False	False	False
47	False	False	False	False	False	False
48	False	False	False	False	False	False
49	False	False	False	False	False	False
50	False	False	False	False	False	False
51	False	False	False	False	False	False
52	False	False	False	False	False	False
53	False	False	False	False	False	False
54	False	False	False	False	False	False
55	False	False	False	False	False	False
56	False	False	False	False	False	False
57	False	False	False	False	False	False
58	False	False	False	False	False	False
59	False	False	False	False	False	False
60	False	False	False	False	False	False
61	False	False	False	False	False	False
62	False	False	False	False	False	False
63	False	False	False	False	False	False
64	False	False	False	False	False	False
65	False	False	False	False	False	False
66	False	False	False	False	False	False
67	False	False	False	False	False	False
68	False	False	False	False	False	False
69	False	False	False	False	False	False
70	False	False	False	False	False	False
71	False	False	False	False	False	False
72	False	False	False	False	False	False
73	False	False	False	False	False	False
74	False	False	False	False	False	False
75	False	False	False	False	False	False

```
df.isnull().sum()
```

```
0    0
1    0
2    0
3    0
4    0
5    0
dtype: int64
```

```
df.rename(columns={0:'Frequency', 1:'Angle', 2:'Chord length', 3:'Free stream velocity', 4:'Suction side displacement thickness', 5:'Sound'},inplace=True)
df.shape
```

```
(1503, 6)
```

```
df.head()
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.sample(5)
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
1	800	5000	0.050000	39.6000000000000	0.002000	125.201000000000
2	1000	2000	0.090000	39.6000000000000	0.002000	123.943000000000
3	1250	2000	0.090000	39.6000000000000	0.002000	124.365000000000
4	1600	2000	0.090000	39.6000000000000	0.002000	125.797000000000
5	2000	2000	0.090000	39.6000000000000	0.002000	127.591000000000
6	2500	2000	0.090000	39.6000000000000	0.002000	129.039000000000
7	3150	2000	0.090000	39.6000000000000	0.002000	130.474000000000
8	4000	2000	0.090000	39.6000000000000	0.002000	132.080000000000
9	5000	2000	0.090000	39.6000000000000	0.002000	133.474000000000
10	6300	2000	0.090000	39.6000000000000	0.002000	135.080000000000
11	8000	2000	0.090000	39.6000000000000	0.002000	136.474000000000
12	10000	2000	0.090000	39.6000000000000	0.002000	138.080000000000
13	12500	2000	0.090000	39.6000000000000	0.002000	139.474000000000
14	16000	2000	0.090000	39.6000000000000	0.002000	141.080000000000
15	20000	2000	0.090000	39.6000000000000	0.002000	142.474000000000
16	800	2500	0.090000	39.6000000000000	0.002000	124.365000000000
17	1000	2500	0.090000	39.6000000000000	0.002000	125.797000000000
18	1250	2500	0.090000	39.6000000000000	0.002000	127.591000000000
19	1600	2500	0.090000	39.6000000000000	0.002000	129.039000000000
20	2000	2500	0.090000	39.6000000000000	0.002000	130.474000000000
21	2500	2500	0.090000	39.6000000000000	0.002000	132.080000000000
22	3150	2500	0.090000	39.6000000000000	0.002000	133.474000000000
23	4000	2500	0.090000	39.6000000000000	0.002000	135.080000000000
24	5000	2500	0.090000	39.6000000000000	0.002000	136.474000000000
25	6300	2500	0.090000	39.6000000000000	0.002000	138.080000000000
26	8000	2500	0.090000	39.6000000000000	0.002000	139.474000000000
27	10000	2500	0.090000	39.6000000000000	0.002000	141.080000000000
28	12500	2500	0.090000	39.6000000000000	0.002000	142.474000000000
29	16000	2500	0.090000	39.6000000000000	0.002000	144.080000000000
30	20000	2500	0.090000	39.6000000000000	0.002000	145.474000000000
31	800	3150	0.090000	39.6000000000000	0.002000	125.797000000000
32	1000	3150	0.090000	39.6000000000000	0.002000	127.591000000000
33	1250	3150	0.090000	39.6000000000000	0.002000	129.039000000000
34	1600	3150	0.090000	39.6000000000000	0.002000	130.474000000000
35	2000	3150	0.090000	39.6000000000000	0.002000	132.080000000000
36	2500	3150	0.090000	39.6000000000000	0.002000	133.474000000000
37	3150	3150	0.090000	39.6000000000000	0.002000	135.080000000000
38	4000	3150	0.090000	39.6000000000000	0.002000	136.474000000000
39	5000	3150	0.090000	39.6000000000000	0.002000	138.080000000000
40	6300	3150	0.090000	39.6000000000000	0.002000	139.474000000000
41	8000	3150	0.090000	39.6000000000000	0.002000	141.080000000000
42	10000	3150	0.090000	39.6000000000000	0.002000	142.474000000000
43	12500	3150	0.090000	39.6000000000000	0.002000	144.080000000000
44	16000	3150	0.090000	39.6000000000000	0.002000	145.474000000000
45	20000	3150	0.090000	39.6000000000000	0.002000	147.080000000000
46	800	4000	0.090000	39.6000000000000	0.002000	127.591000000000
47	1000	4000	0.090000	39.6000000000000	0.002000	129.039000000000
48	1250	4000	0.090000	39.6000000000000	0.002000	130.474000000000
49	1600	4000	0.090000	39.6000000000000	0.002000	132.080000000000
50	2000	4000	0.090000	39.6000000000000	0.002000	133.474000000000
51	2500	4000	0.090000	39.6000000000000	0.002000	135.080000000000
52	3150	4000	0.090000	39.6000000000000	0.002000	136.474000000000
53	4000	4000	0.090000	39.6000000000000	0.002000	138.080000000000
54	5000	4000	0.090000	39.6000000000000	0.002000	139.474000000000
55	6300	4000	0.090000	39.6000000000000	0.002000	141.080000000000
56	8000	4000	0.090000	39.6000000000000	0.002000	142.474000000000
57	10000	4000	0.090000	39.6000000000000	0.002000	144.080000000000
58	12500	4000	0.090000	39.6000000000000	0.002000	145.474000000000
59	16000	4000	0.090000	39.6000000000000	0.002000	147.080000000000
60	20000	4000	0.090000	39.6000000000000	0.002000	148.474000000000
61	800	5000	0.090000	39.6000000000000	0.002000	129.039000000000
62	1000	5000	0.090000	39.6000000000000	0.002000	130.474000000000
63	1250	5000	0.090000	39.6000000000000	0.002000	132.080000000000
64	1600	5000	0.090000	39.6000000000000	0.002000	133.474000000000
65	2000	5000	0.090000	39.6000000000000	0.002000	135.080000000000
66	2500	5000	0.090000	39.6000000000000	0.002000	136.474000000000
67	3150	5000	0.090000	39.6000000000000	0.002000	138.080000000000
68	4000	5000	0.090000	39.6000000000000	0.002000	139.474000000000
69	5000	5000	0.090000	39.6000000000000	0.002000	141.080000000000
70	6300	5000	0.090000	39.6000000000000	0.002000	142.474000000000
71	8000	5000	0.090000	39.6000000000000	0.002000	144.080000000000
72	10000	5000	0.090000	39.6000000000000	0.002000	145.474000000000
73	12500	5000	0.090000	39.6000000000000	0.002000	147.080000000000
74	16000	5000	0.090000	39.6000000000000	0.002000	148.474000000000
75	20000	5000	0.090000	39.6000000000000	0.002000	150.080000000000

```
df.dtypes
```

```
Frequency                                    int64
Angle                                        int64
Chord length                               float64
Free stream velocity                       float64
Suction side displacement thickness        float64
Sound                                      float64
dtype: object
```

```
df.isnull()
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False
11	False	False	False	False	False	False
12	False	False	False	False	False	False
13	False	False	False	False	False	False
14	False	False	False	False	False	False
15	False	False	False	False	False	False
16	False	False	False	False	False	False
17	False	False	False	False	False	False
18	False	False	False	False	False	False
19	False	False	False	False	False	False
20	False	False	False	False	False	False
21	False	False	False	False	False	False
22	False	False	False	False	False	False
23	False	False	False	False	False	False
24	False	False	False	False	False	False
25	False	False	False	False	False	False
26	False	False	False	False	False	False
27	False	False	False	False	False	False
28	False	False	False	False	False	False
29	False	False	False	False	False	False
30	False	False	False	False	False	False
31	False	False	False	False	False	False
32	False	False	False	False	False	False
33	False	False	False	False	False	False
34	False	False	False	False	False	False
35	False	False	False	False	False	False
36	False	False	False	False	False	False
37	False	False	False	False	False	False
38	False	False	False	False	False	False
39	False	False	False	False	False	False
40	False	False	False	False	False	False
41	False	False	False	False	False	False
42	False	False	False	False	False	False
43	False	False	False	False	False	False
44	False	False	False	False	False	False
45	False	False	False	False	False	False
46	False	False	False	False	False	False
47	False	False	False	False	False	False
48	False	False	False	False	False	False
49	False	False	False	False	False	False
50	False	False	False	False	False	False
51	False	False	False	False	False	False
52	False	False	False	False	False	False
53	False	False	False	False	False	False
54	False	False	False	False	False	False
55	False	False	False	False	False	False
56	False	False	False	False	False	False
57	False	False	False	False	False	False
58	False	False	False	False	False	False
59	False	False	False	False	False	False
60	False	False	False	False	False	False
61	False	False	False	False	False	False
62	False	False	False	False	False	False
63	False	False	False	False	False	False
64	False	False	False	False	False	False
65	False	False	False	False	False	False
66	False	False	False	False	False	False
67	False	False	False	False	False	False
68	False	False	False	False	False	False
69	False	False	False	False	False	False
70	False	False	False	False	False	False
71	False	False	False	False	False	False
72	False	False	False	False	False	False
73	False	False	False	False	False	False
74	False	False	False	False	False	False
75	False	False	False	False	False	False

```
df.isnull().sum()
```

```
Frequency                                    0
Angle                                        0
Chord length                                 0
Free stream velocity                         0
Suction side displacement thickness          0
Sound                                        0
dtype: int64
```

Univariate Data Analysis

```
df.describe()
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.describe(include='object')
```

```
ValueError: Cannot describe on DataFrame with 0 rows and 0 columns without at least one dtype.
Consider select_dtypes, e.g. df.select_dtypes(include=['object']).
```

```
df.describe(include='float_64_dtype')
```

```
ValueError: Cannot describe on DataFrame with 0 rows and 0 columns without at least one dtype.
Consider select_dtypes, e.g. df.select_dtypes(include=['float_64_dtype']).
```

```
df.describe(include='int')
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.describe(include='float')
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.describe(include='all')
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.describe(exclude='object')
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.describe(exclude='int')
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.describe(exclude='float')
```

	Frequency	Angle	Chord length	Free stream velocity	Suction side displacement thickness	Sound
count	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000	1503.000000
mean	2903.865603	6782.454151	0.136548	50.860117	0.010196	124.835943
std	2162.827065	5476.636739	0.093541	15.572784	0.013150	6.898657
min	200.000000	500.000000	0.002000	31.700000	0.000400	103.380000
25%	800.000000	2000.000000	0.050000	39.000000	0.002000	120.191000
50%	2500.000000	5000.000000	0.090000	39.600000	0.004957	125.997000
75%	4000.000000	10000.000000	0.137000	71.300000	0.015576	129.067000
max	20000.000000	20000.000000	0.300000	71.300000	0.058411	140.987000

```
df.describe(exclude='all')
```

```
ValueError: Nothing matchs to exclude
```

Binning the Time-Series Data Behaviors

```
df['Frequency'].value_counts
```

```
<bound_method_NDFrame.value_counts of 0       800
1      1000
2      1250
3      1600
4      2000
       ...
1498   8000
1499  10000
1500  12500
1501  16000
1502  20000
Name: Frequency, Length: 1503, dtype: int64>
```

```
df['Frequency'].value_counts()
```

```
800     51
1000    51
1250    51
1600    51
2000    51
2500    51
3150    51
4000    51
5000    51
6300    51
8000    51
10000   51
12500   51
16000   51
20000   51
200     50
500     50
2000    50
5000    50
10000   50
Name: Frequency, dtype: int64
```

```
df['Frequency'].value_counts(ascending=True)
```

```
200     50
500     50
2000    50
5000    50
10000   50
800     51
1000    51
1250    51
1600    51
2000    51
2500    51
3150    51
4000    51
5000    51
6300    51
8000    51
10000   51
12500   51
16000   51
20000   51
Name: Frequency, dtype: int64
```

```
df['Frequency'].value_counts(normalize=True)
```

```
800     0.033932
1000    0.033932
1250    0.033932
1600    0.033932
2000    0.033932
2500    0.033932
3150    0.033932
4000    0.033932
5000    0.033932
6300    0.033932
8000    0.033932
10000   0.033932
12500   0.033932
16000   0.033932
20000   0.033932
200     0.033267
500     0.033267
2000    0.033267
5000    0.033267
10000   0.033267
Name: Frequency, dtype: float64
```

```
df['Frequency'].value_counts(normalize=True).sum()
```

```
1.0
```

```
df['Frequency'].value_counts(normalize=True).mean()
```

```
0.0333333333333333
```

```
df['Frequency'].value_counts(normalize=True).mode
```

```
```

Exploratory Data Analysis

In [54]:
```
df=data.copy()
df
```

Out[54]:

States	date	andhra pradesh	arunachal pradesh	assam	bihar	chandigarh	chhattisgarh	dd	delhi	dnh	...	pondy	punjab	rajasthan	sikkim	tamil nadu
0	2013-01-06	251.5	1.6	21.3	40.9	3.8	60.3	5.8	66.0	15.0	...	5.6	95.2	192.0	1.7	254.5
1	2013-01-07	257.9	1.7	21.2	39.6	3.8	64.3	5.8	67.2	15.1	...	5.5	92.8	192.9	1.8	262.3
2	2013-01-08	265.1	1.6	20.4	39.2	3.9	63.0	5.9	66.7	15.2	...	5.8	93.9	197.1	1.8	256.6
3	2013-01-09	261.0	1.6	21.4	39.5	3.7	65.0	5.8	67.4	15.2	...	5.9	90.1	197.6	1.9	254.6
4	2013-01-10	260.0	1.6	21.1	39.7	3.8	66.5	5.9	67.8	14.9	...	5.9	93.5	199.9	1.9	255.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2590	2020-11-11	173.2	2.1	25.3	71.9	3.2	71.3	7.4	64.4	18.2	...	7.5	111.3	248.4	1.5	301.4
2591	2020-11-12	163.7	2.1	24.9	74.1	3.1	73.7	7.4	62.8	18.1	...	7.7	113.9	248.2	1.5	299.1
2592	2020-11-13	157.4	2.2	25.6	76.0	3.1	73.9	6.9	61.6	17.7	...	7.1	110.1	244.2	1.5	259.9
2593	2020-11-14	157.4	2.2	23.5	78.5	2.7	67.7	4.2	54.4	12.6	...	5.0	88.7	224.2	1.3	197.1
2594	2020-11-15	152.0	2.1	22.0	77.7	2.5	63.1	3.3	49.5	12.1	...	5.4	64.8	205.9	1.2	194.6

2595 rows × 37 columns

In [81]:
```
pd.set_option('max_columns', None)
df.describe()
```

Out[81]:

States	andhra pradesh	arunachal pradesh	assam	bihar	chandigarh	chhattisgarh	dd	delhi	dnh	c
count	2595.000000	2595.000000	2595.000000	2595.000000	2595.000000	2595.000000	2595.000000	2595.000000	2595.000000	2595.0000
mean	165.737187	2.137380	24.491021	69.155915	4.407206	77.161118	6.458497	84.131561	16.001040	60.5209
std	33.701368	1.976552	4.287087	18.782690	1.154916	10.750122	1.010118	22.798622	2.512435	6.2297
min	62.300000	0.600000	10.300000	0.000000	2.200000	26.100000	1.300000	41.800000	1.300000	27.7000
25%	143.600000	1.700000	21.600000	56.950000	3.400000	70.200000	6.000000	62.900000	15.200000	57.0000
50%	160.000000	2.000000	23.400000	67.600000	4.100000	76.400000	6.600000	81.600000	16.500000	61.2000
75%	179.000000	2.200000	26.900000	79.400000	5.400000	84.250000	7.200000	103.900000	17.500000	64.5000
max	284.800000	19.100000	38.100000	124.700000	7.400000	111.600000	15.300000	147.100000	19.500000	77.3000

As you can see above , the column like essar steel , kerla , mizoram , nagaland are having 0 mega units as minimum number which is not possible , so we are going to remove those values while doing data preprocessing

Below we are going to plot the data using folium choropleth to visualize the total number of megaruns used by different states

In [80]:
```
import json
indian_states = json.load(open("states_india.geojson","r"))  # loading india states geojson file

state_id_map={}                                              # creating a dictionary that has states linked with t
he id of geojson file
for feature in indian_states['features']:
    feature['id'] = feature['properties']['st_nm']
    state_id_map[feature['properties']['st_nm'].lower()] = feature['id']

edg df = df.T.reset_index()

edg df = edg df.iloc[1:].reset_index(drop=True)
edg df.rename(columns={'index':'States'},inplace=True)
edg df['States'] = edg df['States'].str.lower()
# edg df[edg df.States == 'dadra and nagar haveli'].index
edg df.drop(1,inplace=True)
edg df.drop(2,inplace=True)
# edg df['sum'] = edg df.sum()                               # creating a row which contains the sum of MegaHits of each st
ate
edg df
```

Out[80]:

	States	0	1	2	3	4	5	6	7	8	...	2586	2587	2588	2589	2590	2591	2592	2593
0	andhra pradesh	251.5	257.9	265.1	261.0	260.0	254.5	257.4	221.1	221.3	...	174.2	168.2	170.4	170.9	173.2	183.7	197.4	197.4
1	arunachal pradesh	1.6	1.7	1.6	1.6	1.6	1.5	1.6	1.2	1.1	...	2.1	2.0	2.1	2.1	2.1	2.1	2.2	2.2
2	assam	21.3	21.2	20.4	21.4	21.1	20.9	20.2	24.8	24.7	...	25.7	23.9	23.4	24.9	25.3	24.9	24.6	23.5
3	bihar	40.9	39.9	39.2	39.3	39.7	39.8	39.9	45.0	43.5	...	68.7	70.6	71.3	73.9	71.9	74.1	76.0	78.5
4	chandigarh	3.8	3.8	3.9	3.7	3.8	3.6	3.4	5.9	5.7	...	3.0	2.7	3.1	3.1	3.2	3.1	3.1	2.7
5	chhattisgarh	60.3	64.3	63.0	65.0	66.5	64.3	65.1	83.8	83.9	...	71.7	71.2	71.3	70.8	71.3	73.7	73.9	67.7
6	dd	5.8	5.8	5.9	5.8	5.9	5.8	5.8	6.0	6.2	...	7.6	8.8	7.2	7.4	7.4	7.4	6.9	4.2
7	delhi	66.0	67.2	66.7	67.4	67.8	65.8	61.7	102.4	103.7	...	61.6	59.8	63.2	63.0	64.4	62.8	61.6	54.4
8	dn	15.0	15.1	15.2	15.2	14.9	14.7	14.8	13.3	14.4	...	18.0	17.3	18.1	18.3	18.2	18.1	17.7	12.6
9	goa	8.6	7.9	7.6	7.9	7.9	7.5	6.7	6.8	7.1	...	10.1	9.5	10.1	9.6	9.8	9.9	9.2	8.2
10	gujarat	238.7	232.9	242.6	244.3	244.8	240.7	235.9	195.1	198.1	...	357.0	345.3	355.2	352.1	343.7	337.8	317.4	265.9
11	haryana	180.4	170.0	110.0	107.4	107.4	112.9	113.7	108.4	170.0	...	122.4	113.8	120.4	122.8	121.6	119.3	113.0	91.0
12	hp	24.9	25.7	25.6	25.6	26.0	26.0	23.9	24.7	24.2	...	28.6	26.7	26.5	26.4	26.8	26.5	27.4	27.4
13	jk	39.6	40.6	40.4	37.8	41.2	41.5	42.0	29.9	31.5	...	47.2	48.2	47.0	47.6	47.5	52.0	45.3	42.0
14	jharkhand	18.9	20.8	22.6	22.0	22.3	21.5	21.1	22.6	22.6	...	24.1	24.8	24.4	24.6	25.4	25.1	25.4	26.2
15	karnataka	178.7	179.7	181.5	182.4	184.5	185.6	175.4	141.9	141.1	...	186.4	172.9	168.7	190.1	190.0	187.2	184.6	171.9
16	kerala	57.8	58.6	56.7	59.6	60.3	59.5	54.7	53.5	53.0	...	73.1	66.7	73.5	74.3	72.5	72.8	72.3	68.3
17	maharashtra	362.5	365.0	365.0	366.3	366.3	355.6	352.2	334.7	335.5	...	430.0	428.9	430.1	427.8	425.4	436.2	436.1	398.8
18	manipur	1.5	1.6	1.6	1.9	1.9	2.0	1.5	1.3	1.3	...	1.5	1.3	1.5	1.5	1.5	1.5	1.5	1.3
19	meghalaya	5.4	5.3	4.8	5.1	5.4	4.9	5.1	4.1	4.7	...	5.9	5.6	5.5	5.8	5.6	5.7	5.6	5.7
20	mizoram	1.6	1.2	1.3	1.0	1.0	1.0	1.0	1.0	1.0	...	1.7	1.6	1.5	1.5	1.6	1.5	1.5	1.7
21	mp	164.1	170.8	179.1	178.9	178.4	169.9	170.8	102.6	110.8	...	276.6	277.4	279.8	284.4	283.7	286.5	286.1	275.8
22	nagaland	1.6	1.6	1.6	1.6	1.6	1.6	1.5	1.4	1.5	...	2.4	2.2	2.4	2.4	2.3	2.1	2.1	2.2
23	odisha	68.4	62.4	65.5	66.7	72.0	68.0	70.3	74.8	74.8	...	96.5	90.4	90.0	89.9	88.6	72.9	71.2	69.6
24	pondy	5.6	5.5	5.8	5.9	5.9	5.9	5.8	6.2	6.1	...	8.0	7.0	7.3	7.9	7.5	7.7	7.1	5.0
25	punjab	95.2	92.9	93.9	90.1	93.5	96.1	95.5	194.7	195.9	...	111.7	108.9	111.4	112.0	111.3	113.9	110.1	88.7
26	rajasthan	192.6	192.9	197.1	197.6	198.9	197.5	195.4	142.6	146.3	...	248.5	247.8	251.9	250.8	246.4	246.2	244.2	224.2
27	sikkim	1.7	1.8	1.6	1.9	1.9	2.0	1.5	1.3	1.3	...	1.5	1.3	1.5	1.5	1.5	1.5	1.5	1.3
28	tamil nadu	254.5	262.3	258.6	254.8	255.3	256.3	246.8	250.4	249.2	...	296.2	299.7	290.0	300.6	301.4	299.1	299.8	197.1
29	telangana	127.6	129.2	131.9	130.5	130.5	132.0	130.9	119.6	115.9	...	147.3	137.1	138.9	137.8	138.9	139.9	142.4	132.5
30	tripura	2.6	2.6	2.1	2.7	3.2	2.7	3.0	3.0	3.0	...	3.6	3.7	3.9	4.1	3.8	3.9	4.1	4.0
31	up	225.0	219.1	239.7	239.1	234.7	222.9	223.8	242.5	240.6	...	279.2	277.1	283.3	279.4	280.1	284.9	286.8	280.7
32	uttarakhand	33.6	34.2	33.9	32.7	35.4	35.4	32.4	35.0	34.2	...	35.0	33.9	35.8	35.1	36.8	36.4	33.9	27.0
33	west bengal	107.5	106.6	103.9	105.7	105.3	107.4	87.8	135.7	135.7	...	111.7	105.2	114.8	118.9	123.0	120.9	121.8	113.5

34 rows × 2597 columns

In [81]:
```
edg df = edg df[~df['States']=='sum']                        # removing the state names proper
ly for merging with id
edg df = edg df.replace('arunachal pradesh','arunachal pradesh')
edg df = edg df.replace('j&k','jammu & kashmir')
edg df = edg df.replace('dadra & nagar haveli','dadra & nagar haveli')
edg df = edg df.replace('j&k','jammu & kashmir')
edg df = edg df.replace('mp','madhya pradesh')
edg df = edg df.replace('up','uttar pradesh')
edg df = edg df.replace('hp','himachal pradesh')
edg df = edg df.replace('pondy','puducherry')
edg df = edg df.replace('hp','himachal pradesh')
edg df['id']=edg df['States'].apply(lambda x: state_id_map[x])   # linking the id with state names
edg df
```

Out[81]:

	States	sum	id
0	andhra pradesh	2791.7	28
1	arunachal pradesh	28|8.0	12
2	assam	2604.8	18
3	bihar	2584.1	10
4	chandigarh	2853.9	4
5	chhattisgarh	2817.2	22
6	daman & diu	2774.3	26
7	nct of delhi	2748.1	7
8	dadra & nagar haveli	2758.9	26
9	goa	2755.1	30
10	gujarat	2604.7	24
11	haryana	2612.8	6
12	himachal pradesh	2701.2	2
13	jammu & kashmir	2669.7	1
14	jharkhand	2700.4	20
15	karnataka	2686.5	29
16	kerala	2697.4	32
17	maharashtra	2682.2	27
18	manipur	2641.9	14
21	meghalaya	2711.6	17
22	mizoram	2716.1	15
23	madhya pradesh	2697.6	23
24	nagaland	2695.6	13
25	odisha	2706.1	21
26	puducherry	2743.0	34
27	punjab	2694.1	3
28	rajasthan	2724.1	8
29	sikkim	2724.2	11
30	tamil nadu	2733.6	33
31	telangana	2490.7	0
32	tripura	2548.5	16
33	uttar pradesh	2537.4	9
34	uttarakhand	2514.7	5
35	west bengal	2586.6	19

In [18]:
```
import folium
import json

india_map = folium.Map(
    location=[19.630562, 78.272066],
    zoom_start=4.5,
    tiles='Stamen Terrain'
)
india_map.choropleth(geo_data=indian_states, data=edg_df, columns=['id','sum'], key_on='feature.properti
ies.state_code',
                     fill_color='YlOrr', fill_opacity=0.7, line_opacity=0.2, nan_fill_color='black'
, nan_fill_opacity=0.2,
                     legend_name='Total MegaHits used')
folium.LayerControl().add_to(india_map)

india_map
```

Out[18]:

From the above we can visualize that chhattisgarh,assam and bihar are using higher megaruns when compared to other states

Plotting the distribution of all the columns

In [84]:
```
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 320
fig, axs = plt.subplots(nrows=1,ncols=12,figsize=(20,40))
c=0
i=0
for l in data.columns[1:]:
    sns.boxplot(data=data, x=l, ax=axs[i],color='andybrown')
    c=c+1
    if c==0:
        c=0
        i=i+1
```

As you can see the data has longer tails which tells that it contains outliers . Most of the columns are either left skewed or right skewed.We need to deal with outliers and try to transform the data which approximately looks like a bell curve.

Plotting the boxplot of each column to visualize the outliers properly

In [85]:
```
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 320
fig, axs = plt.subplots(nrows=1,ncols=12,figsize=(20,40))
c=0
i=0
for l in data.columns[1:]:
    sns.boxplot(data=data, x=l,ax=axs[i],color='andybrown')
    c=c+1
    if c==0:
        c=0
        i=i+1
```


Lets visualize the data of each column whether the data in each column is stationary or not

```
In [18]: import matplotlib.pyplot as plt
         %matplotlib inline
         for key, ldf in zip(ldfs.keys(), ldfs.values):
             print(key)
             plt.figure(figsize=(15,5))
             ldf['Ceiling'].plot()
             ldf['Ceiling_Wear'].plot()
             ldf['Ceiling_Tear'].plot()
             plt.legend(loc='upper right')
             plt.title('Ceiling Wear & Tearing for {}\n{}\n{}\n{
```

Data Preprocessing

Removing outliers using zscore

In [25]:
```
df=data.copy()
from scipy import stats
df = df[(np.abs(stats.zscore(df[df.columns[1:]])) < 3).all(axis=1)]
df
```

Out[25]:

States	date	andhra pradesh	arunachal pradesh	assam	bihar	chandigarh	chhattisgarh	dd	delhi	dnh	...	pondy	punjab	rajasthan	sikkim	tamil nadu
0	2013-01-06	251.5	1.6	21.3	40.9	3.8	60.3	5.8	66.0	15.0	...	5.6	95.2	192.0	1.7	254.5
1	2013-01-07	257.9	1.7	21.2	39.6	3.8	64.3	5.8	67.2	15.1	...	5.5	92.8	192.9	1.8	262.3
2	2013-01-08	265.1	1.6	20.4	39.2	3.9	63.0	5.9	66.7	15.2	...	5.8	93.9	197.1	1.8	256.6
3	2013-01-09	261.0	1.6	21.4	39.5	3.7	65.0	5.8	67.4	15.2	...	5.9	90.1	197.6	1.9	254.6
4	2013-01-10	260.0	1.6	21.1	39.7	3.8	66.5	5.9	67.8	14.9	...	5.9	93.5	199.9	1.9	255.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2589	2020-11-10	170.9	2.1	24.9	73.9	3.1	70.8	7.4	63.0	18.3	...	7.9	112.0	250.8	1.5	300.6
2590	2020-11-11	173.2	2.1	25.3	71.9	3.2	71.3	7.4	64.4	18.2	...	7.5	111.3	248.4	1.5	301.4
2591	2020-11-12	163.7	2.1	24.9	74.1	3.1	73.7	7.4	62.8	18.1	...	7.7	113.9	248.2	1.5	299.1
2592	2020-11-13	157.4	2.2	25.6	76.0	3.1	73.9	6.9	61.6	17.7	...	7.1	110.1	244.2	1.5	259.9
2593	2020-11-14	157.4	2.2	23.5	78.5	2.7	67.7	4.2	54.4	12.6	...	5.0	88.7	224.2	1.3	197.1

2268 rows × 37 columns

Removing negative values because MegaUnits cant be -ve for a state

In [28]: **Out** [28]:
```
Index(['Unnamed: 0', 'id', 'author', 'subreddit',
       'score', 'ups', 'downs', 'date', 'created', 'retrieved',
       'num_comments', 'num_crossposts', 'body', 'sentiment',
       'vaderneg', 'vaderneu', 'vaderpos', 'vadercompound', 'textblob',
       'textblobpolarity', 'textblobsubjectivity', 'week', 'month',
       'year', 'day', 'hour', 'minute', 'second', 'weekday', 'timeofpost',
       'timeofpostcat', 'wordcount', 'charcount', 'avgwordlen',
       'hashtagcount', 'mentioncount', 'emojicount', 'urlcount',
       'capscount', 'lowercount', 'punctcount', 'stopwordcount',
       'stopwordratio', 'uppercount', 'upperratio', 'punctratio',
       'lowerratio', 'hashtagratio', 'mentionratio', 'emojiratio',
       'urlratio', 'capsratio', 'stopwordcat', 'wordcountcat',
       'charcountcat', 'avgwordlencat', 'hashtagcountcat',
       'mentioncountcat', 'emojicountcat', 'urlcountcat',
       'capscountcat', 'lowercountcat', 'punctcountcat',
       'stopwordcountcat', 'uppercountcat', 'punctuation', 'stopwords',
       'lemmatized', 'stemmed', 'nouns', 'verbs', 'adjectives',
       'adverbs', 'pronouns', 'interjections', 'proper_nouns',
       'numerals', 'symbols', 'others', 'lemmatized_nouns',
       'lemmatized_verbs', 'lemmatized_adjectives', 'lemmatized_adverbs',
       'lemmatized_pronouns', 'lemmatized_interjections',
       'lemmatized_proper_nouns', 'lemmatized_numerals',
       'lemmatized_symbols', 'lemmatized_others', 'emojis',
       'emojicodes', 'urls', 'hashtags', 'mentions', 'capsword',
       'lowerword', 'punctuations', 'uppercase', 'lowercase',
       'stopwordcount2', 'stopwordratio2', 'punctuationcount',
       'punctuationratio', 'uppercount2', 'upperratio2', 'lowercount2',
       'lowerratio2', 'hashtagcount2', 'hashtagratio2', 'mentioncount2',
       'mentionratio2', 'emojicount2', 'emojiratio2', 'urlcount2',
       'urlratio2', 'capscount2', 'capsratio2', 'wordcount2',
       'charcount2', 'avgwordlen2', 'stopwordcat2', 'wordcountcat2',
       'charcountcat2', 'avgwordlencat2', 'hashtagcountcat2',
       'mentioncountcat2', 'emojicountcat2', 'urlcountcat2',
       'capscountcat2', 'lowercountcat2', 'punctcountcat2',
       'stopwordcountcat2', 'uppercountcat2', 'punctuation2',
       'stopwords2', 'lemmatized2', 'stemmed2', 'nouns2', 'verbs2',
       'adjectives2', 'adverbs2', 'pronouns2', 'interjections2',
       'proper_nouns2', 'numerals2', 'symbols2', 'others2',
       'lemmatized_nouns2', 'lemmatized_verbs2', 'lemmatized_adjectives2',
       'lemmatized_adverbs2', 'lemmatized_pronouns2',
       'lemmatized_interjections2', 'lemmatized_proper_nouns2',
       'lemmatized_numerals2', 'lemmatized_symbols2',
       'lemmatized_others2', 'emojis2', 'emojicodes2', 'urls2',
       'hashtags2', 'mentions2', 'capsword2', 'lowerword2',
       'punctuations2', 'uppercase2', 'lowercase2', 'stopwordcount3',
       'stopwordratio3', 'punctuationcount3', 'punctuationratio3',
       'uppercount3', 'upperratio3', 'lowercount3', 'lowerratio3',
       'hashtagcount3', 'hashtagratio3', 'mentioncount3',
       'mentionratio3', 'emojicount3', 'emojiratio3', 'urlcount3',
       'urlratio3', 'capscount3', 'capsratio3', 'wordcount3',
       'charcount3', 'avgwordlen3', 'stopwordcat3', 'wordcountcat3',
       'charcountcat3', 'avgwordlencat3', 'hashtagcountcat3',
       'mentioncountcat3', 'emojicountcat3', 'urlcountcat3',
       'capscountcat3', 'lowercountcat3', 'punctcountcat3',
       'stopwordcountcat3', 'uppercountcat3', 'punctuation3',
       'stopwords3', 'lemmatized3', 'stemmed3', 'nouns3', 'verbs3',
       'adjectives3', 'adverbs3', 'pronouns3', 'interjections3',
       'proper_nouns3', 'numerals3', 'symbols3', 'others3',
       'lemmatized_nouns3', 'lemmatized_verbs3', 'lemmatized_adjectives3',
       'lemmatized_adverbs3', 'lemmatized_pronouns3',
       'lemmatized_interjections3', 'lemmatized_proper_nouns3',
       'lemmatized_numerals3', 'lemmatized_symbols3',
       'lemmatized_others3', 'emojis3', 'emojicodes3', 'urls3',
       'hashtags3', 'mentions3', 'capsword3', 'lowerword3',
       'punctuations3', 'uppercase3', 'lowercase3', 'stopwordcount4',
       'stopwordratio4', 'punctuationcount4', 'punctuationratio4',
       'uppercount4', 'upperratio4', 'lowercount4', 'lowerratio4',
       'hashtagcount4', 'hashtagratio4', 'mentioncount4',
       'mentionratio4', 'emojicount4', 'emojiratio4', 'urlcount4',
       'urlratio4', 'capscount4', 'capsratio4', 'wordcount4',
       'charcount4', 'avgwordlen4', 'stopwordcat4', 'wordcountcat4',
       'charcountcat4', 'avgwordlencat4', 'hashtagcountcat4',
       'mentioncountcat4', 'emojicountcat4', 'urlcountcat4',
       'capscountcat4', 'lowercountcat4', 'punctcountcat4',
       'stopwordcountcat4', 'uppercountcat4', 'punctuation4',
       'stopwords4', 'lemmatized4', 'stemmed4', 'nouns4', 'verbs4',
       'adjectives4', 'adverbs4', 'pronouns4', 'interjections4',
       'proper_nouns4', 'numerals4', 'symbols4', 'others4',
       'lemmatized_nouns4', 'lemmatized_verbs4', 'lemmatized_adjectives4',
       'lemmatized_adverbs4', 'lemmatized_pronouns4',
       'lemmatized_interjections4', 'lemmatized_proper_nouns4',
       'lemmatized_numerals4', 'lemmatized_symbols4',
       'lemmatized_others4', 'emojis4', 'emojicodes4', 'urls4',
       'hashtags4', 'mentions4', 'capsword4', 'lowerword4',
       'punctuations4', 'uppercase4', 'lowercase4', 'stopwordcount5',
       'stopwordratio5', 'punctuationcount5', 'punctuationratio5',
       'uppercount5', 'upperratio5', 'lowercount5', 'lowerratio5',
       'hashtagcount5', 'hashtagratio5', 'mentioncount5',
       'mentionratio5', 'emojicount5', 'emojiratio5', 'urlcount5',
       'urlratio5', 'capscount5', 'capsratio5', 'wordcount5',
       'charcount5', 'avgwordlen5', 'stopwordcat5', 'wordcountcat5',
       'charcountcat5', 'avgwordlencat5', 'hashtagcountcat5',
       'mentioncountcat5', 'emojicountcat5', 'urlcountcat5',
       'capscountcat5', 'lowercountcat5', 'punctcountcat5',
       'stopwordcountcat5', 'uppercountcat5', 'punctuation5',
       'stopwords5', 'lemmatized5', 'stemmed5', 'nouns5', 'verbs5',
       'adjectives5', 'adverbs5', 'pronouns5', 'interjections5',
       'proper_nouns5', 'numerals5', 'symbols5', 'others5',
       'lemmatized_nouns5', 'lemmatized_verbs5', 'lemmatized_adjectives5',
       'lemmatized_adverbs5', 'lemmatized_pronouns5',
       'lemmatized_interjections5', 'lemmatized_proper_nouns5',
       'lemmatized_numerals5', 'lemmatized_symbols5',
       'lemmatized_others5', 'emojis5', 'emojicodes5', 'urls5',
       'hashtags5', 'mentions5', 'capsword5', 'lowerword5',
       'punctuations5', 'uppercase5', 'lowercase5', 'stopwordcount6',
       'stopwordratio6', 'punctuationcount6', 'punctuationratio6',
       'uppercount6', 'upperratio6', 'lowercount6', 'lowerratio6',
       'hashtagcount6', 'hashtagratio6', 'mentioncount6',
       'mentionratio6', 'emojicount6', 'emojiratio6', 'urlcount6',
       'urlratio6', 'capscount6', 'capsratio6', 'wordcount6',
       'charcount6', 'avgwordlen6', 'stopwordcat6', 'wordcountcat6',
       'charcountcat6', 'avgwordlencat6', 'hashtagcountcat6',
       'mentioncountcat6', 'emojicountcat6', 'urlcountcat6',
       'capscountcat6', 'lowercountcat6', 'punctcountcat6',
       'stopwordcountcat6', 'uppercountcat6', 'punctuation6',
       'stopwords6', 'lemmatized6', 'stemmed6', 'nouns6', 'verbs6',
       'adjectives6', 'adverbs6', 'pronouns6', 'interjections6',
       'proper_nouns6', 'numerals6', 'symbols6', 'others6',
       'lemmatized_nouns6', 'lemmatized_verbs6', 'lemmatized_adjectives6',
       'lemmatized_adverbs6', 'lemmatized_pronouns6',
       'lemmatized_interjections6', 'lemmatized_proper_nouns6',
       'lemmatized_numerals6', 'lemmatized_symbols6',
       'lemmatized_others6', 'emojis6', 'emojicodes6', 'urls6',
       'hashtags6', 'mentions6', 'capsword6', 'lowerword6',
       'punctuations6', 'uppercase6', 'lowercase6', 'stopwordcount7',
       'stopwordratio7', 'punctuationcount7', 'punctuationratio7',
       'uppercount7', 'upperratio7', 'lowercount7', 'lowerratio7',
       'hashtagcount7', 'hashtagratio7', 'mentioncount7',
       'mentionratio7', 'emojicount7', 'emojiratio7', 'urlcount7',
       'urlratio7', 'capscount7', 'capsratio7', 'wordcount7',
       'charcount7', 'avgwordlen7', 'stopwordcat7', 'wordcountcat7',
       'charcountcat7', 'avgwordlencat7', 'hashtagcountcat7',
       'mentioncountcat7', 'emojicountcat7', 'urlcountcat7',
       'capscountcat7', 'lowercountcat7', 'punctcountcat7',
       'stopwordcountcat7', 'uppercountcat7', 'punctuation7',
       'stopwords7', 'lemmatized7', 'stemmed7', 'nouns7', 'verbs7',
       'adjectives7', 'adverbs7', 'pronouns7', 'interjections7',
       'proper_nouns7', 'numerals7', 'symbols7', 'others7',
       'lemmatized_nouns7', 'lemmatized_verbs7', 'lemmatized_adjectives7',
       'lemmatized_adverbs7', 'lemmatized_pronouns7',
       'lemmatized_interjections7', 'lemmatized_proper_nouns7',
       'lemmatized_numerals7', 'lemmatized_symbols7',
       'lemmatized_others7', 'emojis7', 'emojicodes7', 'urls7',
       'hashtags7', 'mentions7', 'capsword7', 'lowerword7',
       'punctuations7', 'uppercase7', 'lowercase7', 'stopwordcount8',
       'stopwordratio8', 'punctuationcount8', 'punctuationratio8',
       'uppercount8', 'upperratio8', 'lowercount8', 'lowerratio8',
       'hashtagcount8', 'hashtagratio8', 'mentioncount8',
       'mentionratio8', 'emojicount8', 'emojiratio8', 'urlcount8',
       'urlratio8', 'capscount8', 'capsratio8', 'wordcount8',
       'charcount8', 'avgwordlen8', 'stopwordcat8', 'wordcountcat8',
       'charcountcat8', 'avgwordlencat8', 'hashtagcountcat8',
       'mentioncountcat8', 'emojicountcat8', 'urlcountcat8',
       'capscountcat8', 'lowercountcat8', 'punctcountcat8',
       'stopwordcountcat8', 'uppercountcat8', 'punctuation8',
       'stopwords8', 'lemmatized8', 'stemmed8', 'nouns8', 'verbs8',
       'adjectives8', 'adverbs8', 'pronouns8', 'interjections8',
       'proper_nouns8', 'numerals8', 'symbols8', 'others8',
       'lemmatized_nouns8', 'lemmatized_verbs8', 'lemmatized_adjectives8',
       'lemmatized_adverbs8', 'lemmatized_pronouns8',
       'lemmatized_interjections8', 'lemmatized_proper_nouns8',
       'lemmatized_numerals8', 'lemmatized_symbols8',
       'lemmatized_others8', 'emojis8', 'emojicodes8', 'urls8',
       'hashtags8', 'mentions8', 'capsword8', 'lowerword8',
       'punctuations8', 'uppercase8', 'lowercase8', 'stopwordcount9',
       'stopwordratio9', 'punctuationcount9', 'punctuationratio9',
       'uppercount9', 'upperratio9', 'lowercount9', 'lowerratio9',
       'hashtagcount9', 'hashtagratio9', 'mentioncount9',
       'mentionratio9', 'emojicount9', 'emojiratio9', 'urlcount9',
       'urlratio9', 'capscount9', 'capsratio9', 'wordcount9',
       'charcount9', 'avgwordlen9', 'stopwordcat9', 'wordcountcat9',
       'charcountcat9', 'avgwordlencat9', 'hashtagcountcat9',
       'mentioncountcat9', 'emojicountcat9', 'urlcountcat9',
       'capscountcat9', 'lowercountcat9', 'punctcountcat9',
       'stopwordcountcat9', 'uppercountcat9', 'punctuation9',
       'stopwords9', 'lemmatized9', 'stemmed9', 'nouns9', 'verbs9',
       'adjectives9', 'adverbs9', 'pronouns9', 'interjections9',
       'proper_nouns9', 'numerals9', 'symbols9', 'others9',
       'lemmatized_nouns9', 'lemmatized_verbs9', 'lemmatized_adjectives9',
       'lemmatized_adverbs9', 'lemmatized_pronouns9',
       'lemmatized_interjections9', 'lemmatized_proper_nouns9',
       'lemmatized_numerals9', 'lemmatized_symbols9',
       'lemmatized_others9', 'emojis9', 'emojicodes9', 'urls9',
       'hashtags9', 'mentions9', 'capsword9', 'lowerword9',
       'punctuations9', 'uppercase9', 'lowercase9', 'stopwordcount10',
       'stopwordratio10', 'punctuationcount10', 'punctuationratio10',
       'uppercount10', 'upperratio10', 'lowercount10', 'lowerratio10',
       'hashtagcount10', 'hashtagratio10', 'mentioncount10',
       'mentionratio10', 'emojicount10', 'emojiratio10', 'urlcount10',
       'urlratio10', 'capscount10', 'capsratio10', 'wordcount10',
       'charcount10', 'avgwordlen10', 'stopwordcat10', 'wordcountcat10',
       'charcountcat10', 'avgwordlencat10', 'hashtagcountcat10',
       'mentioncountcat10', 'emojicountcat10', 'urlcountcat10',
       'capscountcat10', 'lowercountcat10', 'punctcountcat10',
       'stopwordcountcat10', 'uppercountcat10', 'punctuation10',
       'stopwords10', 'lemmatized10', 'stemmed10', 'nouns10', 'verbs10',
       'adjectives10', 'adverbs10', 'pronouns10', 'interjections10',
       'proper_nouns10', 'numerals10', 'symbols10', 'others10',
       'lemmatized_nouns10', 'lemmatized_verbs10', 'lemmatized_adjectives10',
       'lemmatized_adverbs10', 'lemmatized_pronouns10',
       'lemmatized_interjections10', 'lemmatized_proper_nouns10',
       'lemmatized_numerals10', 'lemmatized_symbols10',
       'lemmatized_others10', 'emojis10', 'emojicodes10', 'urls10',
       'hashtags10', 'mentions10', 'capsword10', 'lowerword10',
       'punctuations10', 'uppercase10', 'lowercase10'],
      dtype=object)
```

checking if there is any missing value

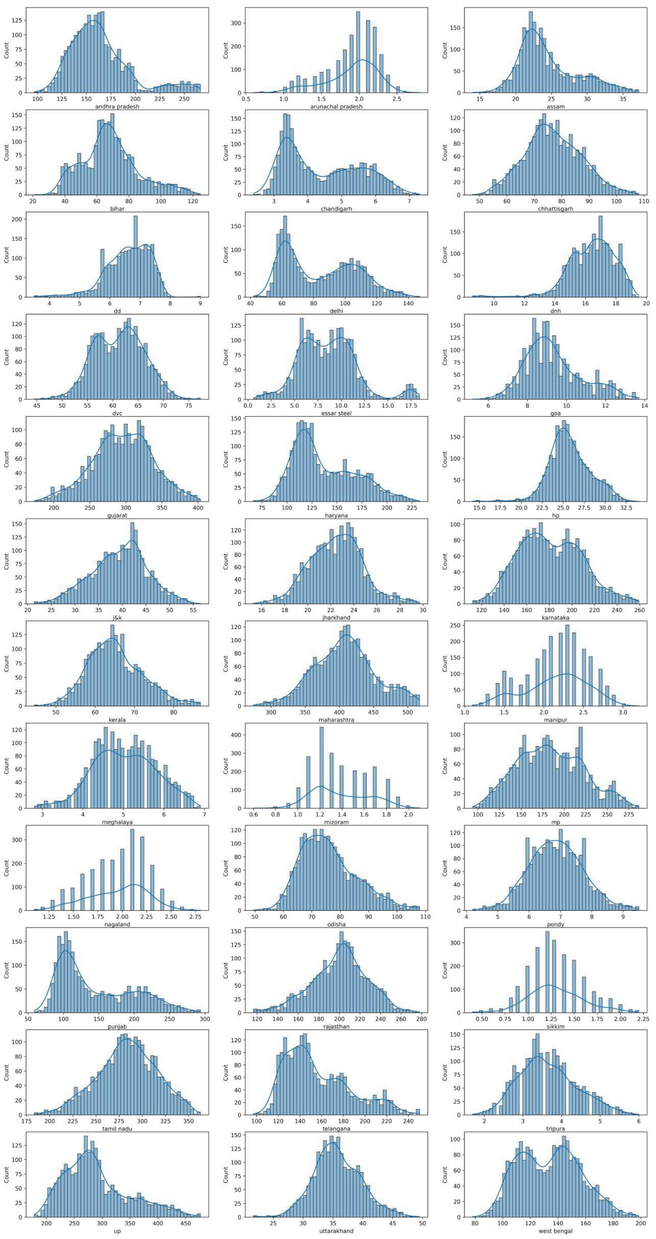
In [29]: **Out** [29]:
```
Unnamed: 0                 0
id                         0
author                     0
subreddit                  0
score                      0
ups                        0
downs                      0
date                       0
created                    0
retrieved                  0
num_comments               0
num_crossposts             0
body                       0
sentiment                  0
vaderneg                   0
vaderneu                   0
vaderpos                   0
vadercompound              0
textblob                   0
textblobpolarity           0
textblobsubjectivity       0
week                       0
month                      0
year                       0
day                        0
hour                       0
minute                     0
second                     0
weekday                    0
timeofpost                 0
timeofpostcat              0
wordcount                  0
charcount                  0
avgwordlen                 0
hashtagcount               0
mentioncount               0
emojicount                 0
urlcount                   0
capscount                  0
lowercount                 0
punctcount                 0
stopwordcount              0
stopwordratio              0
uppercount                 0
upperratio                 0
punctratio                 0
lowerratio                 0
hashtagratio               0
mentionratio               0
emojiratio                 0
urlratio                   0
capsratio                  0
stopwordcat                0
wordcountcat               0
charcountcat               0
avgwordlencat              0
hashtagcountcat            0
mentioncountcat            0
emojicountcat              0
urlcountcat                0
capscountcat               0
lowercountcat              0
punctcountcat              0
stopwordcountcat           0
uppercountcat              0
punctuation                0
stopwords                  0
lemmatized                 0
stemmed                    0
nouns                      0
verbs                      0
adjectives                 0
adverbs                    0
pronouns                   0
interjections              0
proper_nouns               0
numerals                   0
symbols                    0
others                     0
lemmatized_nouns           0
lemmatized_verbs           0
lemmatized_adjectives      0
lemmatized_adverbs         0
lemmatized_pronouns        0
lemmatized_interjections   0
lemmatized_proper_nouns    0
lemmatized_numerals        0
lemmatized_symbols         0
lemmatized_others          0
emojis                     0
emojicodes                 0
urls                       0
hashtags                   0
mentions                   0
capsword                   0
lowerword                  0
punctuations               0
uppercase                  0
lowercase                  0
stopwordcount2             0
stopwordratio2             0
punctuationcount           0
punctuationratio           0
uppercount2                0
upperratio2                0
lowercount2                0
lowerratio2                0
hashtagcount2              0
hashtagratio2              0
mentioncount2              0
mentionratio2              0
emojicount2                0
emojiratio2                0
urlcount2                  0
urlratio2                  0
capscount2                 0
capsratio2                 0
wordcount2                 0
charcount2                 0
avgwordlen2                0
stopwordcat2               0
wordcountcat2              0
charcountcat2              0
avgwordlencat2             0
hashtagcountcat2           0
mentioncountcat2           0
emojicountcat2             0
urlcountcat2               0
capscountcat2              0
lowercountcat2             0
punctcountcat2             0
stopwordcountcat2          0
uppercountcat2             0
punctuation2               0
stopwords2                 0
lemmatized2                0
stemmed2                   0
nouns2                     0
verbs2                     0
adjectives2                0
adverbs2                   0
pronouns2                  0
interjections2             0
proper_nouns2              0
numerals2                  0
symbols2                   0
others2                    0
lemmatized_nouns2          0
lemmatized_verbs2          0
lemmatized_adjectives2     0
lemmatized_adverbs2        0
lemmatized_pronouns2       0
lemmatized_interjections2  0
lemmatized_proper_nouns2   0
lemmatized_numerals2       0
lemmatized_symbols2        0
lemmatized_others2         0
emojis2                    0
emojicodes2                0
urls2                      0
hashtags2                  0
mentions2                  0
capsword2                  0
lowerword2                 0
punctuations2              0
uppercase2                 0
lowercase2                 0
stopwordcount3             0
stopwordratio3             0
punctuationcount3          0
punctuationratio3          0
uppercount3                0
upperratio3                0
lowercount3                0
lowerratio3                0
hashtagcount3              0
hashtagratio3              0
mentioncount3              0
mentionratio3              0
emojicount3                0
emojiratio3                0
urlcount3                  0
urlratio3                  0
capscount3                 0
capsratio3                 0
wordcount3                 0
charcount3                 0
avgwordlen3                0
stopwordcat3               0
wordcountcat3              0
charcountcat3              0
avgwordlencat3             0
hashtagcountcat3           0
mentioncountcat3           0
emojicountcat3             0
urlcountcat3               0
capscountcat3              0
lowercountcat3             0
punctcountcat3             0
stopwordcountcat3          0
uppercountcat3             0
punctuation3               0
stopwords3                 0
lemmatized3                0
stemmed3                   0
nouns3                     0
verbs3                     0
adjectives3                0
adverbs3                   0
pronouns3                  0
interjections3             0
proper_nouns3              0
numerals3                  0
symbols3                   0
others3                    0
lemmatized_nouns3          0
lemmatized_verbs3          0
lemmatized_adjectives3     0
lemmatized_adverbs3        0
lemmatized_pronouns3       0
lemmatized_interjections3  0
lemmatized_proper_nouns3   0
lemmatized_numerals3       0
lemmatized_symbols3        0
lemmatized_others3         0
emojis3                    0
emojicodes3                0
urls3                      0
hashtags3                  0
mentions3                  0
capsword3                  0
lowerword3                 0
punctuations3              0
uppercase3                 0
lowercase3                 0
stopwordcount4             0
stopwordratio4             0
punctuationcount4          0
punctuationratio4          0
uppercount4                0
upperratio4                0
lowercount4                0
lowerratio4                0
hashtagcount4              0
hashtagratio4              0
mentioncount4              0
mentionratio4              0
emojicount4                0
emojiratio4                0
urlcount4                  0
urlratio4                  0
capscount4                 0
capsratio4                 0
wordcount4                 0
charcount4                 0
avgwordlen4                0
stopwordcat4               0
wordcountcat4              0
charcountcat4              0
avgwordlencat4             0
hashtagcountcat4           0
mentioncountcat4           0
emojicountcat4             0
urlcountcat4               0
capscountcat4              0
lowercountcat4             0
punctcountcat4             0
stopwordcountcat4          0
uppercountcat4             0
punctuation4               0
stopwords4                 0
lemmatized4                0
stemmed4                   0
nouns4                     0
verbs4                     0
adjectives4                0
adverbs4                   0
pronouns4                  0
interjections4             0
proper_nouns4              0
numerals4                  0
symbols4                   0
others4                    0
lemmatized_nouns4          0
lemmatized_verbs4          0
lemmatized_adjectives4     0
lemmatized_adverbs4        0
lemmatized_pronouns4       0
lemmatized_interjections4  0
lemmatized_proper_nouns4   0
lemmatized_numerals4       0
lemmatized_symbols4        0
lemmatized_others4         0
emojis4                    0
emojicodes4                0
urls4                      0
hashtags4                  0
mentions4                  0
capsword4                  0
lowerword4                 0
punctuations4              0
uppercase4                 0
lowercase4                 0
stopwordcount5             0
stopwordratio5             0
punctuationcount5          0
punctuationratio5          0
uppercount5                0
upperratio5                0
lowercount5                0
lowerratio5                0
hashtagcount5              0
hashtagratio5              0
mentioncount5              0
mentionratio5              0
emojicount5                0
emojiratio5                0
urlcount5                  0
urlratio5                  0
capscount5                 0
capsratio5                 0
wordcount5                 0
charcount5                 0
avgwordlen5                0
stopwordcat5               0
wordcountcat5              0
charcountcat5              0
avgwordlencat5             0
hashtagcountcat5           0
mentioncountcat5           0
emojicountcat5             0
urlcountcat5               0
capscountcat5              0
lowercountcat5             0
punctcountcat5             0
stopwordcountcat5          0
uppercountcat5             0
punctuation5               0
stopwords5                 0
lemmatized5                0
stemmed5                   0
nouns5                     0
verbs5                     0
adjectives5                0
adverbs5                   0
pronouns5                  0
interjections5             0
proper_nouns5              0
numerals5                  0
symbols5                   0
others5                    0
lemmatized_nouns5          0
lemmatized_verbs5          0
lemmatized_adjectives5     0
lemmatized_adverbs5        0
lemmatized_pronouns5       0
lemmatized_interjections5  0
lemmatized_proper_nouns5   0
lemmatized_numerals5       0
lemmatized_symbols5        0
lemmatized_others5         0
emojis5                    0
emojicodes5                0
urls5                      0
hashtags5                  0
mentions5                  0
capsword5                  0
lowerword5                 0
punctuations5              0
uppercase5                 0
lowercase5                 0
stopwordcount6             0
stopwordratio6             0
punctuationcount6          0
punctuationratio6          0
uppercount6                0
upperratio6                0
lowercount6                0
lowerratio6                0
hashtagcount6              0
hashtagratio6              0
mentioncount6              0
mentionratio6              0
emojicount6                0
emojiratio6                0
urlcount6                  0
urlratio6                  0
capscount6                 0
capsratio6                 0
wordcount6                 0
charcount6                 0
avgwordlen6                0
stopwordcat6               0
wordcountcat6              0
charcountcat6              0
avgwordlencat6             0
hashtagcountcat6           0
mentioncountcat6           0
emojicountcat6             0
urlcountcat6               0
capscountcat6              0
lowercountcat6             0
punctcountcat6             0
stopwordcountcat6          0
uppercountcat6             0
punctuation6               0
stopwords6                 0
lemmatized6                0
stemmed6                   0
nouns6                     0
verbs6                     0
adjectives6                0
adverbs6                   0
pronouns6                  0
interjections6             0
proper_nouns6              0
numerals6                  0
symbols6                   0
others6                    0
lemmatized_nouns6          0
lemmatized_verbs6          0
lemmatized_adjectives6     0
lemmatized_adverbs6        0
lemmatized_pronouns6       0
lemmatized_interjections6  0
lemmatized_proper_nouns6   0
lemmatized_numerals6       0
lemmatized_symbols6        0
lemmatized_others6         0
emojis6                    0
emojicodes6                0
urls6                      0
hashtags6                  0
mentions6                  0
capsword6                  0
lowerword6                 0
punctuations6              0
uppercase6                 0
lowercase6                 0
stopwordcount7             0
stopwordratio7             0
punctuationcount7          0
punctuationratio7          0
uppercount7                0
upperratio7                0
lowercount7                0
lowerratio7                0
hashtagcount7              0
hashtagratio7              0
mentioncount7              0
mentionratio7              0
emojicount7                0
emojiratio7                0
urlcount7                  0
urlratio7                  0
capscount7                 0
capsratio7                 0
wordcount7                 0
charcount7                 0
avgwordlen7                0
stopwordcat7               0
wordcountcat7              0
charcountcat7              0
avgwordlencat7             0
hashtagcountcat7           0
mentioncountcat7           0
emojicountcat7             0
urlcountcat7               0
capscountcat7              0
lowercountcat7             0
punctcountcat7             0
stopwordcountcat7          0
uppercountcat7             0
punctuation7               0
stopwords7                 0
lemmatized7                0
stemmed7                   0
nouns7                     0
verbs7                     0
adjectives7                0
adverbs7                   0
pronouns7                  0
interjections7             0
proper_nouns7              0
numerals7                  0
symbols7                   0
others7                    0
lemmatized_nouns7          0
lemmatized_verbs7          0
lemmatized_adjectives7     0
lemmatized_adverbs7        0
lemmatized_pronouns7       0
lemmatized_interjections7  0
lemmatized_proper_nouns7   0
lemmatized_numerals7       0
lemmatized_symbols7        0
lemmatized_others7         0
emojis7                    0
emojicodes7                0
urls7                      0
hashtags7                  0
mentions7                  0
capsword7                  0
lowerword7                 0
punctuations7              0
uppercase7                 0
lowercase7                 0
stopwordcount8             0
stopwordratio8             0
punctuationcount8          0
punctuationratio8          0
uppercount8                0
upperratio8                0
lowercount8                0
lowerratio8                0
hashtagcount8              0
hashtagratio8              0
mentioncount8              0
mentionratio8              0
emojicount8                0
emojiratio8                0
urlcount8                  0
urlratio8                  0
capscount8                 0
capsratio8                 0
wordcount8                 0
charcount8                 0
avgwordlen8                0
stopwordcat8               0
wordcountcat8              0
charcountcat8              0
avgwordlencat8             0
hashtagcountcat8           0
mentioncountcat8           0
emojicountcat8             0
urlcountcat8               0
capscountcat8              0
lowercountcat8             0
punctcountcat8             0
stopwordcountcat8          0
uppercountcat8             0
punctuation8               0
stopwords8                 0
lemmatized8                0
stemmed8                   0
nouns8                     0
verbs8                     0
adjectives8                0
adverbs8                   0
pronouns8                  0
interjections8             0
proper_nouns8              0
numerals8                  0
symbols8                   0
others8                    0
lemmatized_nouns8          0
lemmatized_verbs8          0
lemmatized_adjectives8     0
lemmatized_adverbs8        0
lemmatized_pronouns8       0
lemmatized_interjections8  0
lemmatized_proper_nouns8   0
lemmatized_numerals8       0
lemmatized_symbols8        0
lemmatized_others8         0
emojis8                    0
emojicodes8                0
urls8                      0
hashtags8                  0
mentions8                  0
capsword8                  0
lowerword8                 0
punctuations8              0
uppercase8                 0
lowercase8                 0
stopwordcount9             0
stopwordratio9             0
punctuationcount9          0
punctuationratio9          0
uppercount9                0
upperratio9                0
lowercount9                0
lowerratio9                0
hashtagcount9              0
hashtagratio9              0
mentioncount9              0
mentionratio9              0
emojicount9                0
emojiratio9                0
urlcount9                  0
urlratio9                  0
capscount9                 0
capsratio9                 0
wordcount9                 0
charcount9                 0
avgwordlen9                0
stopwordcat9               0
wordcountcat9              0
charcountcat9              0
avgwordlencat9             0
hashtagcountcat9           0
mentioncountcat9           0
emojicountcat9             0
urlcountcat9               0
capscountcat9              0
lowercountcat9             0
punctcountcat9             0
stopwordcountcat9          0
uppercountcat9             0
punctuation9               0
stopwords9                 0
lemmatized9                0
stemmed9                   0
nouns9                     0
verbs9                     0
adjectives9                0
adverbs9                   0
pronouns9                  0
interjections9             0
proper_nouns9              0
numerals9                  0
symbols9                   0
others9                    0
lemmatized_nouns9          0
lemmatized_verbs9          0
lemmatized_adjectives9     0
lemmatized_adverbs9        0
lemmatized_pronouns9       0
lemmatized_interjections9  0
lemmatized_proper_nouns9   0
lemmatized_numerals9       0
lemmatized_symbols9        0
lemmatized_others9         0
emojis9                    0
emojicodes9                0
urls9                      0
hashtags9                  0
mentions9                  0
capsword9                  0
lowerword9                 0
punctuations9              0
uppercase9                 0
lowercase9                 0
stopwordcount10            0
stopwordratio10            0
punctuationcount10         0
punctuationratio10         0
uppercount10               0
upperratio10               0
lowercount10               0
lowerratio10               0
hashtagcount10             0
hashtagratio10             0
mentioncount10             0
mentionratio10             0
emojicount10               0
emojiratio10               0
urlcount10                 0
urlratio10                 0
capscount10                0
capsratio10                0
wordcount10                0
charcount10                0
avgwordlen10               0
stopwordcat10              0
wordcountcat10             0
charcountcat10             0
avgwordlencat10            0
hashtagcountcat10          0
mentioncountcat10          0
emojicountcat10            0
urlcountcat10              0
capscountcat10             0
lowercountcat10            0
punctcountcat10            0
stopwordcountcat10         0
uppercountcat10            0
punctuation10              0
stopwords10                0
lemmatized10               0
stemmed10                  0
nouns10                    0
verbs10                    0
adjectives10               0
adverbs10                  0
pronouns10                 0
interjections10            0
proper_nouns10             0
numerals10                 0
symbols10                  0
others10                   0
lemmatized_nouns10         0
lemmatized_verbs10         0
lemmatized_adjectives10    0
lemmatized_adverbs10       0
lemmatized_pronouns10      0
lemmatized_interjections10 0
lemmatized_proper_nouns10  0
lemmatized_numerals10      0
lemmatized_symbols10       0
lemmatized_others10        0
emojis10                   0
emojicodes10               0
urls10                     0
hashtags10                 0
mentions10                 0
capsword10                 0
lowerword10                0
punctuations10             0
uppercase10                0
lowercase10                0
dtype: int64
```

Finding top and after removing outliers

In [30]:
```
sns.set(rc={'figure.figure.size':(11,8)})
plot_cols = df[['sentiment','vaderneg','vaderneu','vaderpos',
                'vadercompound','textblob']]
for col in plot_cols:
    plt.figure()
    sns.boxplot(x=plot_cols[col])
```

In [31]: **Out** [31]:
```
Top 10 Outliers
Top 10 Outliers
Top 10 Outliers
```

In [38]:
```
mpl.rcParams['figure.dpi'] = 300
fig, axs = plt.subplots(ncols=3,nrows=12,figsize=(20,40))
c=0
l=0
for i in df.columns[1:]:
    sns.histplot(data=df, x=i,ax=axs[l,c],kde=True,legend=True,bins=50)
    c+=1
    if c==3:
        c=0
        l+=1
```



As you can see above the data is nearly normal distributed , Now we need to save this dataframe into an csv file and keep it as a backup

- Next we are going to make the data stationary for making the data suitable for time series forecasting

In [39]:
```
df3 = df.copy()
df3 = df3.sort_values('date',ascending=True).reset_index(drop=True)    # sorting the df by date column
df3
```

Out[39]:

	States	date	andhra pradesh	arunachal pradesh	assam	bihar	chandigarh	chhattisgarh	dd	delhi	dnh	dvc	essar steel	goa	gujarat	haryana	hp	j
0		2013-01-06	251.5	1.6	21.3	40.9	3.8	60.3	5.8	66.0	15.0	56.2	7.8	8.6	238.7	100.4	24.9	39
1		2013-01-07	257.9	1.7	21.2	39.6	3.8	64.3	5.8	67.2	15.1	59.9	7.8	7.9	232.9	110.0	25.7	40
2		2013-01-08	265.1	1.6	20.4	39.2	3.9	63.0	5.9	66.7	15.2	60.3	7.9	7.6	242.6	110.0	25.8	40
3		2013-01-09	261.0	1.6	21.4	39.5	3.7	65.0	5.8	67.4	15.2	60.0	8.0	7.9	244.3	107.4	25.6	37
4		2013-01-10	260.0	1.6	21.1	39.7	3.8	66.5	5.9	67.8	14.9	58.9	8.0	7.8	244.8	112.9	26.0	4
...		...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2262		2020-11-10	170.9	2.1	24.9	73.9	3.1	70.8	7.4	63.0	18.3	63.6	17.5	9.6	352.1	122.8	29.4	4
2263		2020-11-11	173.2	2.1	25.3	71.9	3.2	71.3	7.4	64.4	18.2	63.9	17.6	8.9	343.7	121.6	28.8	4
2264		2020-11-12	163.7	2.1	24.9	74.1	3.1	73.7	7.4	62.8	18.1	63.6	16.4	9.8	337.8	119.3	29.5	52
2265		2020-11-13	157.4	2.2	25.6	76.0	3.1	73.9	6.9	61.6	17.7	65.4	17.8	9.9	317.4	113.0	27.4	45
2266		2020-11-14	157.4	2.2	23.5	78.5	2.7	67.7	4.2	54.4	12.6	64.4	17.5	8.2	265.9	91.0	21.4	42

2267 rows × 37 columns

In [96]:
```
%store  df3   # storing the df
```

```
Stored 'df3' (DataFrame)
```

In [229]:
```
df3.to_csv('pycharm_data.csv')  # saving the df into csv file
```

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
new_df = pd.read_csv('pycharm_data.csv').drop(columns='Unnamed: 0')  # importing the saved df
new_df
```

Out[2]:

	date	andhra pradesh	arunachal pradesh	assam	bihar	chandigarh	chhattisgarh	dd	delhi	dnh	...	pondy	punjab	rajasthan	sikkim	tamil nadu
0	2013-01-06	251.5	1.6	21.3	40.9	3.8	60.3	5.8	66.0	15.0	...	5.6	95.2	192.0	1.7	254.5
1	2013-01-07	257.9	1.7	21.2	39.6	3.8	64.3	5.8	67.2	15.1	...	5.5	92.8	192.9	1.8	262.3
2	2013-01-08	265.1	1.6	20.4	39.2	3.9	63.0	5.9	66.7	15.2	...	5.8	93.9	197.1	1.8	256.6
3	2013-01-09	261.0	1.6	21.4	39.5	3.7	65.0	5.8	67.4	15.2	...	5.9	90.1	197.6	1.9	254.6
4	2013-01-10	260.0	1.6	21.1	39.7	3.8	66.5	5.9	67.8	14.9	...	5.9	93.5	199.9	1.9	255.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2262	2020-11-10	170.9	2.1	24.9	73.9	3.1	70.8	7.4	63.0	18.3	...	7.9	112.0	250.8	1.5	300.6
2263	2020-11-11	173.2	2.1	25.3	71.9	3.2	71.3	7.4	64.4	18.2	...	7.5	111.3	248.4	1.5	301.4
2264	2020-11-12	163.7	2.1	24.9	74.1	3.1	73.7	7.4	62.8	18.1	...	7.7	113.9	248.2	1.5	299.1
2265	2020-11-13	157.4	2.2	25.6	76.0	3.1	73.9	6.9	61.6	17.7	...	7.1	110.1	244.2	1.5	259.9
2266	2020-11-14	157.4	2.2	23.5	78.5	2.7	67.7	4.2	54.4	12.6	...	5.0	88.7	224.2	1.3	197.1

2267 rows × 37 columns

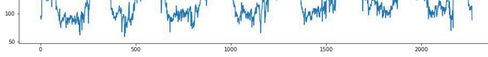
Note: I didn't apply minmax scalar or standardscalar because i saw there is a decline in accuracy when applying these methods on both test data and future predictions.

Making the Time-Series Data Stationary

The overall process i am going to do inorder to make the data stationary:

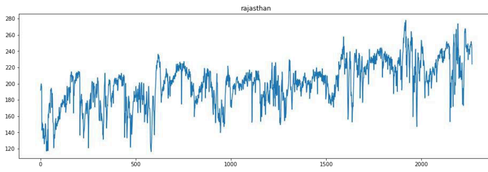
- At first we are going to visualize the data and see the trend and seasonality
- To confirm that data is stationary or not , we are going to use **Augumented Dicky Fuller test** and consider the pvalue
- If p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary. (non - stationary)
- If p-value <= 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is stationary. (stationary)
- We are going to apply seasonal decompose to see the trend and seasonality of the data separately
- We are going to tranform the non-stationary data to stationary using differencing transform

```
# of product 2.25 kcal/kcal is obtained from total hydrothermal (HTC) and dark fermentation (DF) and dark fermentation and
# photofermentation (PF) and photofermentation (PF) and dark fermentation (DF) and dark fermentation and photofermentation
# (PF) and photofermentation (PF) and dark fermentation (DF) and dark fermentation and photofermentation (PF) and photofermentation
# (PF) and photofermentation (PF) and dark fermentation (DF) and dark fermentation and photofermentation (PF) and photofermentation
import matplotlib.pyplot as plt
from sklearn.ensemble import (
    AdaBoostClassifier,
    gradientboosting,
    ExtraTreesClassifier,
    RandomForestClassifier,
    VotingClassifier,
    BaggingClassifier,
    HistGradientBoostingClassifier,
    StackingClassifier,
    GradientBoostingClassifier,
    AdaBoostClassifier
```



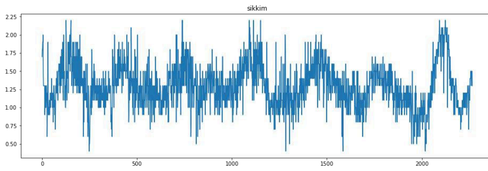
```
mean1 = 136.340071, mean2 = 154.176896
variance1 = 2373.623143, variance2 = 3039.296680

ADF Statistic: -3.780689
p-value: 0.003111
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

rajasthan



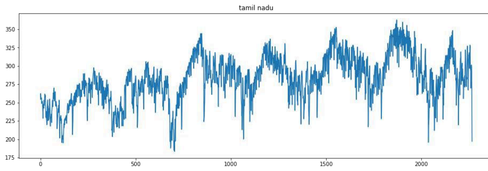
```
mean1 = 186.818182, mean2 = 213.591799
variance1 = 561.364894, variance2 = 537.677208

ADF Statistic: -4.439963
p-value: 0.000252
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

sikkim



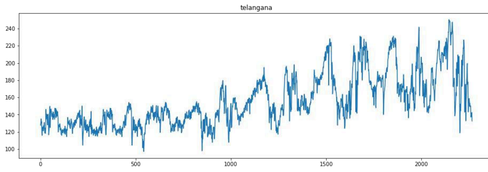
```
mean1 = 1.327096, mean2 = 1.276543
variance1 = 0.079698, variance2 = 0.098039

ADF Statistic: -3.823819
p-value: 0.002673
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

tamil nadu



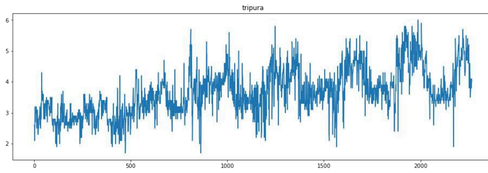
```
mean1 = 268.290203, mean2 = 298.752381
variance1 = 846.044750, variance2 = 764.089708

ADF Statistic: -3.608001
p-value: 0.005610
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

telangana



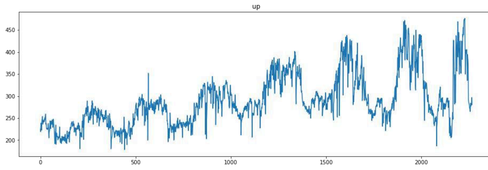
```
mean1 = 134.909003, mean2 = 177.338272
variance1 = 173.025250, variance2 = 750.491322

ADF Statistic: -3.039949
p-value: 0.031328
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

tripura



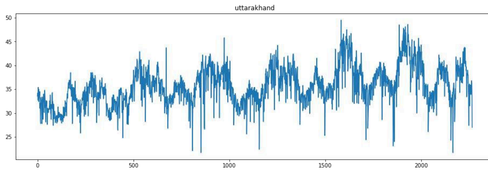
```
mean1 = 3.272462, mean2 = 4.019841
variance1 = 0.348174, variance2 = 0.486952

ADF Statistic: -3.517696
p-value: 0.007544
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

up



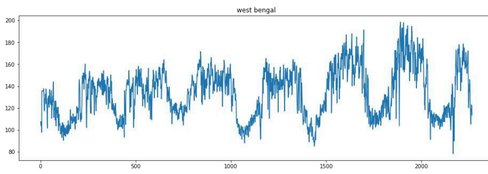
```
mean1 = 254.247308, mean2 = 325.232363
variance1 = 1114.348424, variance2 = 3278.550284

ADF Statistic: -3.217290
p-value: 0.019003
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

uttarakhand



```
mean1 = 34.191792, mean2 = 37.290476
variance1 = 10.966967, variance2 = 15.824900

ADF Statistic: -4.214149
p-value: 0.000623
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

west bengal



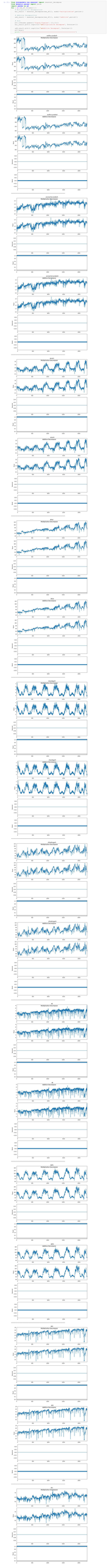
```
mean1 = 128.881906, mean2 = 139.182363
variance1 = 321.319858, variance2 = 610.573234

ADF Statistic: -3.422909
p-value: 0.010202
Critical Values:
	1%: -3.433
	5%: -2.863
	10%: -2.567
==========================================================================================
=========
```

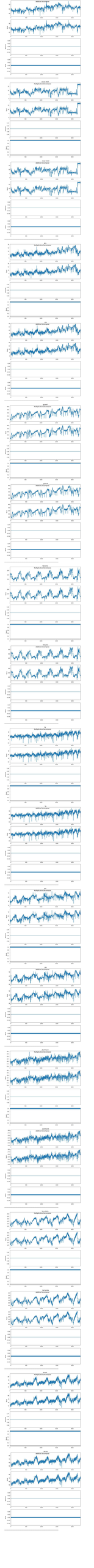
As you can see many columns have p value less the 0.05 in adftest, but by seeing the data visually we can see that every column has the some trend which is directing upwards. Eventhough adftest says it is stationary we should give importance to the visuals

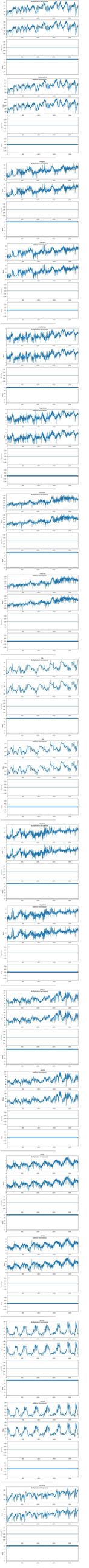
- Now we are going to see the trend and seasionality separately with the help of seasional decompose library
- we are going to use both multiplicative and additive decompose methods

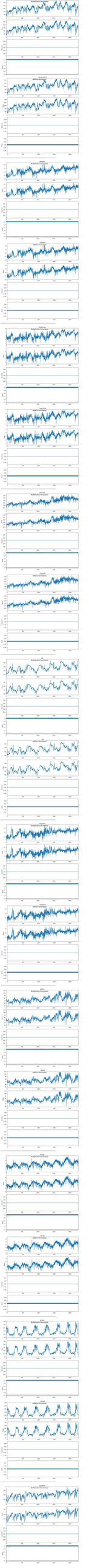
```
In [4]: from sklearn.preprocessing import Normalizer, StandardScaler
        from sklearn.pipeline import Pipeline
        from sklearn.cluster import KMeans
        from sklearn.decomposition import PCA
        # Pipeline for normalization and PCA
        normalizer = Normalizer(norm='l2', copy=True)
        pca = PCA(n_components=2, whiten=False, random_state=None, svd_solver='auto')
        # Pipeline for normalization, PCA, and clustering
        pipeline = Pipeline([('normalizer', normalizer), ('pca', pca)])
        # Transform the data using the pipeline
        X_pca = pipeline.fit_transform(X_scaled)
        print("Shape after PCA: {}".format(X_pca.shape))
        # Clustering on the PCA-transformed data
        kmeans = KMeans(n_clusters=3, random_state=42)
        clusters = kmeans.fit_predict(X_pca)
```

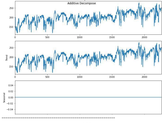
```
Shape after PCA: (1000, 2)
```

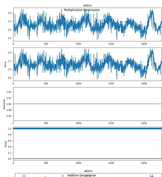


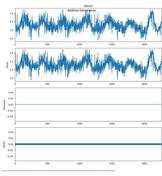

Adding Noise (Gaussian)
Adding Noise (Gaussian)
Adding Noise (Gaussian)

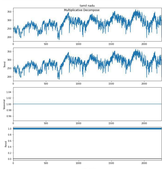


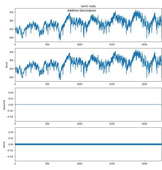


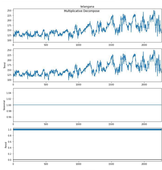


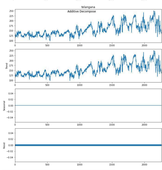


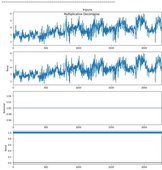


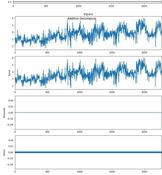


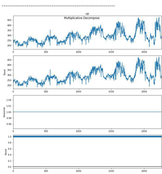


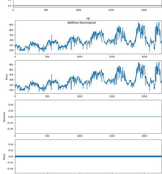


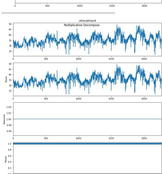


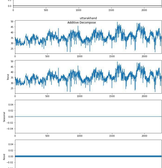


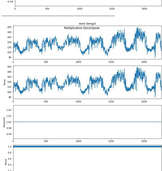


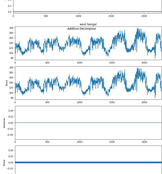












From the above graphs we can conclude the the data has trend but seasonality is normal

We are going to apply differential transformation, and do ADF test again to check the p-value

```
In [4]: for col in df.columns[1:]:
            print(f"{col} stationarity:")
            ts = df[col].dropna()
            result = adfuller(ts, autolag='AIC')
            print(f'ADF Statistic: {result[0]}')
            print(f'p-value: {result[1]}')
            print('Critical Values:')
            for key, value in result[4].items():
                print(f'\t{key}: {value}')
            if result[1] <= 0.05:
                print("\n✅ Stationary (reject H0)")
            else:
                print("\n❌ Non-stationary (fail to reject H0)")
            print("\n" + "-"*50 + "\n")
            ts = ts - ts.shift(1)
            ts = ts.dropna()
            result = adfuller(ts, autolag='AIC')
            print(f'After differencing:')
            print(f'ADF Statistic: {result[0]}')
            print(f'p-value: {result[1]}')
            print('Critical Values:')
            for key, value in result[4].items():
                print(f'\t{key}: {value}')
            if result[1] <= 0.05:
                print("\n✅ Stationary (reject H0)")
            else:
                print("\n❌ Non-stationary (fail to reject H0)")
            print("\n" + "-"*50 + "\n")
            plt.figure(figsize=(12,5))
            plt.plot(ts)
            plt.title(f'After differencing transformation for the column {f'{col}'}')
            plt.grid(True)
            plt.show()
            ts = df[col].dropna()
            ts = ts - ts.shift(1)
            ts = ts.dropna()
            ts = np.log(ts)
            result = adfuller(ts, autolag='AIC')
            print(f'After log transformation:')
            print(f'ADF Statistic: {result[0]}')
            print(f'p-value: {result[1]}')
            print('Critical Values:')
            for key, value in result[4].items():
                print(f'\t{key}: {value}')
            if result[1] <= 0.05:
                print("\n✅ Stationary (reject H0)")
            else:
                print("\n❌ Non-stationary (fail to reject H0)")
            print("\n" + "-"*50 + "\n")
            plt.figure(figsize=(12,5))
            plt.plot(ts)
            plt.title(f'After differencing transformation for the column {f'{col}'}')
            plt.grid(True)
            plt.show
```

```
Traceback (most recent call last):
  File "<stackexchange_replication_1>", line 1, in <module>
  File "<stackexchange_replication_1>", line 2, in <module>
  File "<stackexchange_replication_1>", line 3, in <module>
  File "<stackexchange_replication_1>", line 4, in <module>
  File "<stackexchange_replication_1>", line 5, in <module>
  File "<stackexchange_replication_1>", line 6, in <module>
  File "<stackexchange_replication_1>", line
```

```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.047000

ADF Statistic: -0.747378
p-value: 0.839000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -0.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```



```
Mean(t-1)=0.000000, mean2=0.000000
Variance(t-1)=0.000000, variance2=0.000000

ADF Statistic: -10.000000
p-value: 0.000000
Critical Values:
   1%: -3.466
   5%: -2.877
   10%: -2.575
```

```
count    1.000000
mean     0.000000
std      0.000000
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      0.000000
dtype: float64
```

```
CITIZ CODE 2 (column)
```

```
In [74]: [ORIGINAL_DATA_COL_NAME_LIST[i] for i in range(len(ORIGINAL_DATA_COL_NAME_LIST))]
```

```
Out[74]: ['CITIZ CODE',
          'bankruptcies',
          'income',
          'mortgage',
          'income',
          'mortgage1',
          'dependents',
          'years',
          'payed',
          'capital',
          'other',
          'bad debt',
          'mortgage1',
          'claim',
          'dwelling',
          'damagepaid',
          'claim1',
          'up',
          'attempted',
          'damage']
```

Step 2: Dividing the original dataset into two parts

Step 3: Applying Augmented Dickey Fuller Test on each column and transform the trend and seasonality if the column is non stationary

```
In [75]: print("Original Dimensions:",
         ORIGINAL_DATA_DF.shape)
         print("Dimensions of Data Set for ADF Test: ",len(ORIGINAL_DATA_COL_NAME_LIST))
         print("Dimensions of Data Set for Forecasting: ",len(ORIGINAL_DATA_COL_NAME_LIST))
         ORIGINAL_DATA_DF.to_csv("/content/drive/MyDrive/Colab Notebooks/Forecasting/ORIGINAL_DATA.csv")
         print("ORIGINAL_DATA.csv Saved to Forecasting Folder")
         print("Dimensions of Data Set for Forecasting Model: ",
         ORIGINAL_DATA_DF.shape)
         ORIGINAL_DATA_DF.tail(5)
```

```
Original Dimensions: (1000, 20)
Dimensions of Data Set for ADF Test:  20
Dimensions of Data Set for Forecasting:  20
ORIGINAL_DATA.csv Saved to Forecasting Folder
Dimensions of Data Set for Forecasting Model:  (1000, 20)
```


	CITIZ CODE	bankruptcies	income	mortgage	income	mortgage1	dependents	years	payed	capital	other	bad debt	mortgage1	claim	dwelling	damagepaid	claim1	up	attempted	damage
995	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
996	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
997	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
998	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
999	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [8]:
```
new_lstm_log_diff.dropna(inplace=True)
new_lstm_log_diff = new_lstm_log_diff.reset_index(drop=True)
new_lstm_log_diff
```

Out[8]:

	andhra pradesh
0	6.4
1	7.2
2	-4.1
3	-1.0
4	-5.5
...	...
2261	0.5
2262	2.3
2263	-9.5
2264	-6.3
2265	0.0

2266 rows × 1 columns

As you can see above the data became stationary (mean and variance is not varying much), we are going to apply differencing transformation before applying LSTM

Creating Univariate LSTM model

Below I had created a function model_pre which is only used for creating model for columns another protect and nitrogenous

The things it does is explained below

- Extracts the data of those two columns from the input
- Extracts the NO value, because I am going to predict the data from the year 2011, by NO and TL have data from 2010 year, this value is used for creating a column for differencing the data and the data of year 2010 is dropped
- Then, taking the DO data for training and DNO data as a test data. Incentivizing for the whole data
- Then normalizing the data using MinMaxScaler and then using the data to fit a Sequential Keras model with two unstructure units or Dense1 unless a default, Dense1 means one neuron and two hidden layers
- Then I am creating the training and testing data
- Then I fit data to the training and the testing is for training the data
- Then I am predicting the data for the testing data
- I am going to create a DATE field using pd.Timestamp(2010,1,1) and then
- I am creating a dataset with the actual data and predicted data and I am sorting the dataset values within the given range
- After entering the model, I am going to get the mean score from all the folds and use it to test the data
- I am using the model to predict the data for the year 2011
- I am going to save the model as **column+name_model_fold.h5**
- I am also saving the model history as **column+name_model_fold.npy** and I am going to **column+name_model_fold.txt**
- Later I will use the saved models to make future predictions

In [1]:
```
%load_ext autotime
%matplotlib inline
%matplotlib inline
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers.recurrent import LSTM
from keras.callbacks import History
history = History()
from keras import callbacks
cbks = [history]
import matplotlib.pyplot as plt
from matplotlib import rcParams
import math
from sklearn.metrics import mean_squared_error
import pickle
import pickle as cPickle
import gzip
import os
import sys
import datetime
import dateutil.parser
from dateutil import parser
from dateutil import rrule
from datetime import datetime, time, date
from sklearn.ensemble import RandomForestRegressor
from sklearn import cross_validation
from sklearn import ensemble
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import KFold
from sklearn.grid_search import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from keras.wrappers.scikit_learn import KerasClassifier
from keras.wrappers.scikit_learn import KerasRegressor
from keras.utils.np_utils import to_categorical
from keras.models import Sequential, Graph
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM
from keras.optimizers import SGD
def load_data(data, seq_len):
    amount_of_features = len(data.columns)
    data = data.as_matrix()
    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])
    result = np.array(result)
    row = round(0.9 * result.shape[0])
    train = result[:int(row), :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[int(row):, :-1]
    y_test = result[int(row):, -1]
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], amount_of_features))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], amount_of_features))
    return [x_train, y_train, x_test, y_test]
def build_model(layers):
    model = Sequential()
    model.add(LSTM(layers[1], return_sequences=True,
    activation="tanh",
    inner_activation="hard_sigmoid"))
    model.add(Dropout(0.2))
    model.add(LSTM(layers[2],
    activation="tanh",
    inner_activation="hard_sigmoid"))
    model.add(Dropout(0.2))
    model.add(Dense(layers[3],
    activation="linear"))
    model.compile(loss="mse", optimizer="rmsprop")
    print("Compilation Time : %s" % (time.ctime()))
    return model
def inverse_transform(scaler, data, steps):
    print("Inverse transforming the values with {0} data for future predictions".\
    format(steps))
    for i in range(0, steps):
        data[:, i] = scaler.inverse_transform(data[:, i])
    return data
def predict_point_by_point(model, data):
    print("Predicting the model with {0} data for future predictions".\
    format(data.shape[0]))
    predicted = model.predict(data)
    print("Reshaping the predicted data to 1D and returning back to calling function")
    predicted = np.reshape(predicted, (predicted.size,))
    return predicted
def plot_results_multiple(predicted_data, true_data, prediction_len):
    print("Plotting the predicted data and true data for comparison")
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    print("Length of the predicted data", len(predicted_data))
    for i, p in enumerate(predicted_data):
        padding = [None for x in range(i * prediction_len)]
        plt.plot(padding + p, label='Prediction')
        print("Adding padding to the predicted data, padding is", len(padding))
    print("Displaying the plot")
    plt.legend()
    plt.show(block=False)
def plot_results(predicted_data, true_data):
    print("Plotting the predicted data and true data for comparison")
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    print("Length of the predicted data", len(predicted_data))
    plt.plot(predicted_data, label='Prediction')
    print("Displaying the plot")
    plt.legend()
    plt.show(block=False)
def model_score(model, X_train, y_train, X_test, y_test):
    print("Evaluating the model with X_train and y_train data")
    train_score = model.evaluate(X_train, y_train, verbose=0)
    print('Train score: %.5f MSE (%.2f RMSE)' % (train_score, math.sqrt(train_score)))
    print("Evaluating the model with X_test and y_test data")
    test_score = model.evaluate(X_test, y_test, verbose=0)
    print('Test score: %.5f MSE (%.2f RMSE)' % (test_score, math.sqrt(test_score)))
    return train_score, test_score
def plot_mse(history):
    print("Plotting mean squared error vs epochs")
    plt.plot(history.history['mean_squared_error'],
    label='Mean Squared Error')
    plt.legend()
    plt.show(block=False)
def model_score(model, X_train, y_train, X_test, y_test):
    print("Evaluating the model with X_train and y_train data")
    train_score = model.evaluate(X_train, y_train, verbose=0)
    print('Train score: %.5f MSE (%.2f RMSE)' % (train_score, math.sqrt(train_score)))
    print("Evaluating the model with X_test and y_test data")
    test_score = model.evaluate(X_test, y_test, verbose=0)
    print('Test score: %.5f MSE (%.2f RMSE)' % (test_score, math.sqrt(test_score)))
    return train_score, test_score
def plot_mse(history):
    print("Plotting mean squared error vs epochs")
    plt.plot(history.history['mean_squared_error'],
    label='Mean Squared Error')
    plt.legend()
    plt.show(block=False)
def model_pre(df, seq_len, name):
    print("Extracting the data from the dataframe")
    df = df[['DO', 'NO', 'TL']]
    df['NO'] = df['NO'].shift(-1)
    df['DO'] = df['DO'] - df['NO']
    df['TL'] = df['TL'] - df['NO']
    df = df.dropna()
    print("Dropping the NO data")
    df = df[['DO', 'TL']]
    print("Taking the DO data for training and DNO data as a test data. Incentivizing for the whole data")
    df = df.ix[:df.shape[0]]
    print("Normalizing the data using MinMaxScaler and then using the data to fit a Sequential Keras model with two unstructure units or Dense1 unless a default, Dense1 means one neuron and two hidden layers")
    scaler = MinMaxScaler(feature_range=(0, 1))
    df = scaler.fit_transform(df)
    print("Creating the training data X, y")
    X, y = df[:, 0:-1], df[:, -1]
    print("Reshaping the training data to be compatible with Keras")
    X = np.reshape(X, (X.shape[0], 1, X.shape[1]))
    print("Creating the testing data X_test, y_test")
    X_test = X[int(X.shape[0]*0.9):]
    y_test = y[int(y.shape[0]*0.9):]
    print("Creating a sequential model")
    model = Sequential()
    print("Adding the first layer with 50 units")
    model.add(LSTM(50, return_sequences=True, activation="tanh", inner_activation="hard_sigmoid"))
    print("Adding a dropout layer with 20% dropout")
    model.add(Dropout(0.2))
    print("Adding the second layer with 100 units")
    model.add(LSTM(100, activation="tanh", inner_activation="hard_sigmoid"))
    print("Adding a dropout layer with 20% dropout")
    model.add(Dropout(0.2))
    print("Adding a dense layer with 1 unit")
    model.add(Dense(1, activation="linear"))
    print("Compiling the model with loss as mse and optimizer as rmsprop")
    model.compile(loss="mse", optimizer="rmsprop")
    print("Fitting the model with 10 epochs, 512 batch_size and 0.1 validation_split")
    model.fit(X, y, nb_epoch=10, batch_size=512, validation_split=0.1, callbacks=cbks)
    print("Evaluating the model with X_train and y_train data")
    train = model.evaluate(X, y, verbose=0)
    print("Evaluating the model with X_test and y_test data")
    test = model.evaluate(X_test, y_test, verbose=0)
    print("Predicting the model with X data for future predictions")
    predicted = model.predict(X)
    print("Reshaping the predicted data to 1D and returning back to calling function")
    predicted = np.reshape(predicted, (predicted.size,))
    print("Inverse transforming the values with {0} data for future predictions".\
    format(predicted.shape[0]))
    for i in range(0, predicted.shape[0]):
        predicted[i] = scaler.inverse_transform(predicted[i])
    print("Inverse transforming the values with {0} data for future predictions".\
    format(y.shape[0]))
    for i in range(0, y.shape[0]):
        y[i] = scaler.inverse_transform(y[i])
    print("Creating a dataset with the actual data and predicted data and I am sorting the dataset values within the given range")
    print("After entering the model, I am going to get the mean score from all the folds and use it to test the data")
    print("I am using the model to predict the data for the year 2011")
    print("I am going to save the model as {0}_model_fold.h5".\
    format(name))
    print("I am also saving the model history as {0}_model_fold.npy and I am going to {0}_model_fold.txt".\
    format(name))
    print("Later I will use the saved models to make future predictions")
```

In [2]: time: [0] (0 s)

In [3]: time: [0] (0 s)

In [4]: time: [0] (0 s)

In [5]: time: [0] (0 s)

In [6]: time: [0] (0 s)

In [7]: time: [0] (0 s)

In

In [28]:
```
# final model for states other than AP and TELENGANA without mimax
# accuracy check

def predict_rem(col_name):
    print('Creating a best model for predicting {} MU and Checking its accuracy with test data'.format(
col_name))
    import pandas as pd
    import warnings

    warnings.filterwarnings("ignore")

    df3 = pd.read_csv('pycharm_data.csv')
    df3 = df3.drop(columns=['Unnamed: 0', 'date'])
    new_df = df3[col_name].copy()
    new_df

    # create a differenced series
    from pandas import Series                                   # creating a function for diferencing tra
nsformation
    def difference(dataset, interval=1):
        diff = list()
        for i in range(interval, len(dataset)):
            value = dataset[i] - dataset[i - interval]
            diff.append(value)
        return Series(diff)


    differenced = difference(new_df,1)                        # applying the differencing transformation
 for the whole data
    differenced.head()

    X = []
    y = []
    for i in range(0, differenced.shape[0] - 48):
        X.append(differenced.iloc[i:i + 48])                   # taking 48 rows for training incrementall
y
        y.append(differenced.iloc[i + 48])

    import numpy as np
    X, y = np.array(X,dtype=np.float32), np.array(y,dtype=np.float32) # converting the data to array
    y = np.reshape(y, (len(y), 1))                                    # reshaping the y data


    X_train, X_test = X[:-480], X[-480:]                    # splitting the data for training and testing

    y_train, y_test = y[:-480], y[-480:]

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))  # reshaping the data to 3d
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))      # reshaping the data to 3d


    import warnings                                            # importing required libraries for creatin
g models
    warnings.filterwarnings("ignore")
    import keras
    from keras.layers import LSTM, Activation, Dense
    from keras import optimizers
    from keras.callbacks import ModelCheckpoint, EarlyStopping
    from keras.layers import ReLU,LeakyReLU
    import h5py
    from keras.models import load_model
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import Dropout
    from keras.layers import LSTM
    from kerastuner.tuners import BayesianOptimization

    def build_model(hp):    # In keras-tuner we are using BayesianOptimization for searching the best p
arams
        for i in range(hp.Int('num_layers', 2, 5)):
            model = keras.Sequential()
            model.add(LSTM(units=hp.Int('units_' + str(i), min_value=32,max_value=300,step=32)))
            model.add(ReLU())
            model.add(Dropout(rate=hp.Float(
                    'dropout_1',
                    min_value=0.1,
                    max_value=0.5,
                    default=0.1,
                    step=0.05)))
            model.add(Dense(1, activation='linear'))
            model.compile(
            optimizer=keras.optimizers.Adam(
                hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
            loss='mean_absolute_error',
            metrics=['mse'])
        return model


    bayesian_opt_tuner = BayesianOptimization(
        build_model,
        objective='mse',
        max_trials=3,
        executions_per_trial=1,
        overwrite=True)

    bayesian_opt_tuner.search(X_train, y_train,epochs=1000,validation_data=(X_test,y_test),callbacks =[
keras.callbacks.EarlyStopping(monitor='val_loss',patience=100)], verbose=0, batch_size=5)
                                                            # searching the best parameters for our lstm mo
del
                                                           # for validation we are using test data set
    best_model = bayesian_opt_tuner.get_best_models(num_models=1)
    model = best_model[0]                             # taking the best model out of all

    predictions = model.predict(X_test)               # predicting the output for test data
    final_pred = []                                   # applying inverse differencing transformation to t
he predicted values
    for i in range(len(predictions)):
        y_test_diff =predictions[i]+new_df[1786+i]
        final_pred.append(y_test_diff[0])
    final_pred

    y_test_orig = []                                  # applying inverse differencing transformation to t
he original values
    for i in range(len(y_test)):
        y_test_diff = y_test[i]+new_df[1786+i]
        y_test_orig.append(y_test_diff[0])
    y_test_orig

    total = 0
    for i, j in zip(y_test_orig, final_pred):
        value = abs(i - j) / abs(i)
        total += value
    error = float(total * 100 / (len(y_test_orig)))  # calculate mape
    mape = round(error, 1)                           # round to 3 significant figures
    accuracy = 100 - mape                            # Calculate accuracy

    import matplotlib.pyplot as plt
    print("The LSTM's accuracy in predicting the mega units on test data : " + str(accuracy) + "%")
    print(' ')
    df = pd.DataFrame(y_test_orig, final_pred).reset_index() # ploting the graph between predicted and
 actual values
    df.columns = ['original','predicted']
    df.plot(title='Original data vs Predicted data (for test data)')
    plt.show()
    print(' ')
    print('==================================================================================')

    print('')
    print('Now training the model with 100% data for future predictions....')
                                     # Now we are going to train the same model with 100% data for bett
er predictions
    X_train = X                      # taking the whole X data for training
    y_train = y                      # taking whole y data for training
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))   # reshaping the data to 3d
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))       # reshaping the data to 3d

    callbacks = [EarlyStopping(monitor = 'val_loss',patience=100)]
                                # creating a call-back ( if there is no decrease in loss after 100 epochs
 , it stops training)
    model.fit(X_train,y_train,validation_split=0.2,epochs=1000,callbacks=callbacks,batch_size=5,shuffle
=False)
    model.save('models_with_tuner\\'+col_name+'_model_100.h5')  # saved the model after training with 1
00% data
    print(' ')
    print('saved the model successfully')
```

```
tf.keras.layers.Dense(1,activation='sigmoid')
# creating a model from predicting scores 0% and checking its accuracy with test data
model.fit(x_train,y_train,epochs=10,validation_data=(x_test,y_test))
```

Trial complete

Trial summary

```
|-Trial ID: 0f1d7d5bf115bc9b9d2d8a8a
|-Score: 0.7142857142857143
|-Best step: 0
```

Hyperparameters:

```
|-learning_rate: 0.01000000000000004
|-Loss_0: 128
|-Loss_1: 256
|-Loss_2: 32
|-Loss_3: 128
|-Loss_4: 128
|-Loss_5: 32
|-Loss_6: 128
|-Loss_7: 32
|-Loss_8: 128
|-Loss_9: 32
```

Trial complete

Trial summary

```
|-Trial ID: 1d7d5bf115bc9b9d2d8a8a00
|-Score: 0.7142857142857143
|-Best step: 0
```

Hyperparameters:

```
|-learning_rate: 0.01000000000000004
|-Loss_0: 128
|-Loss_1: 256
|-Loss_2: 32
|-Loss_3: 128
|-Loss_4: 128
|-Loss_5: 32
|-Loss_6: 128
|-Loss_7: 32
|-Loss_8: 128
|-Loss_9: 32
```

Trial complete

Trial summary

```
|-Trial ID: 2d7d5bf115bc9b9d2d8a8a00
|-Score: 0.7142857142857143
|-Best step: 0
```

Hyperparameters:

```
|-learning_rate: 0.01000000000000004
|-Loss_0: 128
|-Loss_1: 256
|-Loss_2: 32
|-Loss_3: 128
|-Loss_4: 128
|-Loss_5: 32
|-Loss_6: 128
|-Loss_7: 32
|-Loss_8: 128
|-Loss_9: 32
```

Trial complete

Trial summary

```
|-Trial ID: 3d7d5bf115bc9b9d2d8a8a00
|-Score: 0.7142857142857143
|-Best step: 0
```

Hyperparameters:

```
|-learning_rate: 0.01000000000000004
|-Loss_0: 128
|-Loss_1: 256
|-Loss_2: 32
|-Loss_3: 128
|-Loss_4: 128
|-Loss_5: 32
|-Loss_6: 128
|-Loss_7: 32
|-Loss_8: 128
|-Loss_9: 32
```

```
Model training the model with 1700 data for future predictions...
Epoch 1/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 2/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 3/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 4/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 5/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 6/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 7/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 8/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 9/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Epoch 10/10
53/53 [==========]- 0ms/step - loss: 0.5977 - mae: 0.5977 - mse: 0.5977 - val_loss: 0.5978 - val_mae: 0.5978 - val_mse: 0.5978
Model training
```

```
{"data":[{"id":"1","name":"Alex Ng","class":"C-101","date":"2025-01-01","time":"08:00","status":"Present"},{"id":"2","name":"Bella Tan","class":"C-102","date":"2025-01-02","time":"08:05","status":"Late"},{"id":"3","name":"Carlos Mendes","class":"C-103","date":"2025-01-03","time":"08:10","status":"Absent"},{"id":"4","name":"Dana Lim","class":"C-104","date":"2025-01-04","time":"08:15","status":"Present"},{"id":"5","name":"Eric Wong","class":"C-105","date":"2025-01-05","time":"08:20","status":"Late"},{"id":"6","name":"Fiona Chen","class":"C-106","date":"2025-01-06","time":"08:25","status":"Absent"},{"id":"7","name":"Gabriel Ong","class":"C-107","date":"2025-01-07","time":"08:30","status":"Present"},{"id":"8","name":"Hannah Koh","class":"C-108","date":"2025-01-08","time":"08:35","status":"Late"},{"id":"9","name":"Ivan Teo","class":"C-109","date":"2025-01-09","time":"08:40","status":"Absent"},{"id":"10","name":"Jessica Ang","class":"C-110","date":"2025-01-10","time":"08:45","status":"Present"},{"id":"11","name":"Kevin Sim","class":"C-111","date":"2025-01-11","time":"08:50","status":"Late"},{"id":"12","name":"Lucas Ng","class":"C-112","date":"2025-01-12","time":"08:55","status":"Absent"},{"id":"13","name":"Mia Tan","class":"C-113","date":"2025-01-13","time":"09:00","status":"Present"},{"id":"14","name":"Nathan Lim","class":"C-114","date":"2025-01-14","time":"09:05","status":"Late"},{"id":"15","name":"Olivia Wong","class":"C-115","date":"2025-01-15","time":"09:10","status":"Absent"},{"id":"16","name":"Pauline Chen","class":"C-116","date":"2025-01-16","time":"09:15","status":"Present"},{"id":"17","name":"Quinn Ong","class":"C-117","date":"2025-01-17","time":"09:20","status":"Late"},{"id":"18","name":"Rachel Koh","class":"C-118","date":"2025-01-18","time":"09:25","status":"Absent"},{"id":"19","name":"Sam Teo","class":"C-119","date":"2025-01-19","time":"09:30","status":"Present"},{"id":"20","name":"Tina Ang","class":"C-120","date":"2025-01-20","time":"09:35","status":"Late"},{"id":"21","name":"Ulysses Sim","class":"C-121","date":"2025-01-21","time":"09:40","status":"Absent"},{"id":"22","name":"Vanessa Ng","class":"C-122","date":"2025-01-22","time":"09:45","status":"Present"},{"id":"23","name":"Wilson Tan","class":"C-123","date":"2025-01-23","time":"09:50","status":"Late"},{"id":"24","name":"Xavier Lim","class":"C-124","date":"2025-01-24","time":"09:55","status":"Absent"},{"id":"25","name":"Yvonne Wong","class":"C-125","date":"2025-01-25","time":"10:00","status":"Present"},{"id":"26","name":"Zoe Chen","class":"C-126","date":"2025-01-26","time":"10:05","status":"Late"},{"id":"27","name":"Aaron Ong","class":"C-127","date":"2025-01-27","time":"10:10","status":"Absent"},{"id":"28","name":"Bella Koh","class":"C-128","date":"2025-01-28","time":"10:15","status":"Present"},{"id":"29","name":"Carlos Teo","class":"C-129","date":"2025-01-29","time":"10:20","status":"Late"},{"id":"30","name":"Dana Ang","class":"C-130","date":"2025-01-30","time":"10:25","status":"Absent"},{"id":"31","name":"Eric Sim","class":"C-131","date":"2025-01-31","time":"10:30","status":"Present"},{"id":"32","name":"Fiona Ng","class":"C-132","date":"2025-02-01","time":"10:35","status":"Late"},{"id":"33","name":"Gabriel Tan","class":"C-133","date":"2025-02-02","time":"10:40","status":"Absent"},{"id":"34","name":"Hannah Lim","class":"C-134","date":"2025-02-03","time":"10:45","status":"Present"},{"id":"35","name":"Ivan Wong","class":"C-135","date":"2025-02-04","time":"10:50","status":"Late"},{"id":"36","name":"Jessica Chen","class":"C-136","date":"2025-02-05","time":"10:55","status":"Absent"},{"id":"37","name":"Kevin Ong","class":"C-137","date":"2025-02-06","time":"11:00","status":"Present"},{"id":"38","name":"Lucas Koh","class":"C-138","date":"2025-02-07","time":"11:05","status":"Late"},{"id":"39","name":"Mia Teo","class":"C-139","date":"2025-02-08","time":"11:10","status":"Absent"},{"id":"40","name":"Nathan Ang","class":"C-140","date":"2025-02-09","time":"11:15","status":"Present"},{"id":"41","name":"Olivia Sim","class":"C-141","date":"2025-02-10","time":"11:20","status":"Late"},{"id":"42","name":"Pauline Ng","class":"C-142","date":"2025-02-11","time":"11:25","status":"Absent"},{"id":"43","name":"Quinn Tan","class":"C-143","date":"2025-02-12","time":"11:30","status":"Present"},{"id":"44","name":"Rachel Lim","class":"C-144","date":"2025-02-13","time":"11:35","status":"Late"},{"id":"45","name":"Sam Wong","class":"C-145","date":"2025-02-14","time":"11:40","status":"Absent"},{"id":"46","name":"Tina Chen","class":"C-146","date":"2025-02-15","time":"11:45","status":"Present"},{"id":"47","name":"Ulysses Ong","class":"C-147","date":"2025-02-16","time":"11:50","status":"Late"},{"id":"48","name":"Vanessa Koh","class":"C-148","date":"2025-02-17","time":"11:55","status":"Absent"},{"id":"49","name":"Wilson Teo","class":"C-149","date":"2025-02-18","time":"12:00","status":"Present"},{"id":"50","name":"Xavier Ang","class":"C-150","date":"2025-02-19","time":"12:05","status":"Late"},{"id":"51","name":"Yvonne Sim","class":"C-151","date":"2025-02-20","time":"12:10","status":"Absent"},{"id":"52","name":"Zoe Ng","class":"C-152","date":"2025-02-21","time":"12:15","status":"Present"},{"id":"53","name":"Aaron Tan","class":"C-153","date":"2025-02-22","time":"12:20","status":"Late"},{"id":"54","name":"Bella Lim","class":"C-154","date":"2025-02-23","time":"12:25","status":"Absent"},{"id":"55","name":"Carlos Wong","class":"C-155","date":"2025-02-24","time":"12:30","status":"Present"},{"id":"56","name":"Dana Chen","class":"C-156","date":"2025-02-25","time":"12:35","status":"Late"},{"id":"57","name":"Eric Ong","class":"C-157","date":"2025-02-26","time":"12:40","status":"Absent"},{"id":"58","name":"Fiona Koh","class":"C-158","date":"2025-02-27","time":"12:45","status":"Present"},{"id":"59","name":"Gabriel Teo","class":"C-159","date":"2025-02-28","time":"12:50","status":"Late"},{"id":"60","name":"Hannah Ang","class":"C-160","date":"2025-03-01","time":"12:55","status":"Absent"},{"id":"61","name":"Ivan Sim","class":"C-161","date":"2025-03-02","time":"13:00","status":"Present"},{"id":"62","name":"Jessica Ng","class":"C-162","date":"2025-03-03","time":"13:05","status":"Late"},{"id":"63","name":"Kevin Tan","class":"C-163","date":"2025-03-04","time":"13:10","status":"Absent"},{"id":"64","name":"Lucas Lim","class":"C-164","date":"2025-03-05","time":"13:15","status":"Present"},{"id":"65","name":"Mia Wong","class":"C-165","date":"2025-03-06","time":"13:20","status":"Late"},{"id":"66","name":"Nathan Chen","class":"C-166","date":"2025-03-07","time":"13:25","status":"Absent"},{"id":"67","name":"Olivia Ong","class":"C-167","date":"2025-03-08","time":"13:30","status":"Present"},{"id":"68","name":"Pauline Koh","class":"C-168","date":"2025-03-09","time":"13:35","status":"Late"},{"id":"69","name":"Quinn Teo","class":"C-169","date":"2025-03-10","time":"13:40","status":"Absent"},{"id":"70","name":"Rachel Ang","class":"C-170","date":"2025-03-11","time":"13:45","status":"Present"},{"id":"71","name":"Sam Sim","class":"C-171","date":"2025-03-12","time":"13:50","status":"Late"},{"id":"72","name":"Tina Ng","class":"C-172","date":"2025-03-13","time":"13:55","status":"Absent"},{"id":"73","name":"Ulysses Tan","class":"C-173","date":"2025-03-14","time":"14:00","status":"Present"},{"id":"74","name":"Vanessa Lim","class":"C-174","date":"2025-03-15","time":"14:05","status":"Late"},{"id":"75","name":"Wilson Wong","class":"C-175","date":"2025-03-16","time":"14:10","status":"Absent"},{"id":"76","name":"Xavier Chen","class":"C-176","date":"2025-03-17","time":"14:15","status":"Present"},{"id":"77","name":"Yvonne Ong","class":"C-177","date":"2025-03-18","time":"14:20","status":"Late"},{"id":"78","name":"Zoe Koh","class":"C-178","date":"2025-03-19","time":"14:25","status":"Absent"},{"id":"79","name":"Aaron Teo","class":"C-179","date":"2025-03-20","time":"14:30","status":"Present"},{"id":"80","name":"Bella Ang","class":"C-180","date":"2025-03-21","time":"14:35","status":"Late"},{"id":"81","name":"Carlos Sim","class":"C-181","date":"2025-03-22","time":"14:40","status":"Absent"},{"id":"82","name":"Dana Ng","class":"C-182","date":"2025-03-23","time":"14:45","status":"Present"},{"id":"83","name":"Eric Tan","class":"C-183","date":"2025-03-24","time":"14:50","status":"Late"},{"id":"84","name":"Fiona Lim","class":"C-184","date":"2025-03-25","time":"14:55","status":"Absent"},{"id":"85","name":"Gabriel Wong","class":"C-185","date":"2025-03-26","time":"15:00","status":"Present"},{"id":"86","name":"Hannah Chen","class":"C-186","date":"2025-03-27","time":"15:05","status":"Late"},{"id":"87","name":"Ivan Ong","class":"C-187","date":"2025-03-28","time":"15:10","status":"Absent"},{"id":"88","name":"Jessica Koh","class":"C-188","date":"2025-03-29","time":"15:15","status":"Present"},{"id":"89","name":"Kevin Teo","class":"C-189","date":"2025-03-30","time":"15:20","status":"Late"},{"id":"90","name":"Lucas Ang","class":"C-190","date":"2025-03-31","time":"15:25","status":"Absent"},{"id":"91","name":"Mia Sim","class":"C-191","date":"2025-04-01","time":"15:30","status":"Present"},{"id":"92","name":"Nathan Ng","class":"C-192","date":"2025-04-02","time":"15:35","status":"Late"},{"id":"93","name":"Olivia Tan","class":"C-193","date":"2025-04-03","time":"15:40","status":"Absent"},{"id":"94","name":"Pauline Lim","class":"C-194","date":"2025-04-04","time":"15:45","status":"Present"},{"id":"95","name":"Quinn Wong","class":"C-195","date":"2025-04-05","time":"15:50","status":"Late"},{"id":"96","name":"Rachel Chen","class":"C-196","date":"2025-04-06","time":"15:55","status":"Absent"},{"id":"97","name":"Sam Ong","class":"C-197","date":"2025-04-07","time":"16:00","status":"Present"},{"id":"98","name":"Tina Koh","class":"C-198","date":"2025-04-08","time":"16:05","status":"Late"},{"id":"99","name":"Ulysses Teo","class":"C-199","date":"2025-04-09","time":"16:10","status":"Absent"},{"id":"100","name":"Vanessa Ang","class":"C-200","date":"2025-04-10","time":"16:15","status":"Present"},{"id":"101","name":"Wilson Sim","class":"C-201","date":"2025-04-11","time":"16:20","status":"Late"},{"id":"102","name":"Xavier Ng","class":"C-202","date":"2025-04-12","time":"16:25","status":"Absent"},{"id":"103","name":"Yvonne Tan","class":"C-203","date":"2025-04-13","time":"16:30","status":"Present"},{"id":"104","name":"Zoe Lim","class":"C-204","date":"2025-04-14","time":"16:35","status":"Late"},{"id":"105","name":"Aaron Wong","class":"C-205","date":"2025-04-15","time":"16:40","status":"Absent"},{"id":"106","name":"Bella Chen","class":"C-206","date":"2025-04-16","time":"16:45","status":"Present"},{"id":"107","name":"Carlos Ong","class":"C-207","date":"2025-04-17","time":"16:50","status":"Late"},{"id":"108","name":"Dana Koh","class":"C-208","date":"2025-04-18","time":"16:55","status":"Absent"},{"id":"109","name":"Eric Teo","class":"C-209","date":"2025-04-19","time":"17:00","status":"Present"},{"id":"110","name":"Fiona Ang","class":"C-210","date":"2025-04-20","time":"17:05","status":"Late"},{"id":"111","name":"Gabriel Sim","class":"C-211","date":"2025-04-21","time":"17:10","status":"Absent"},{"id":"112","name":"Hannah Ng","class":"C-212","date":"2025-04-22","time":"17:15","status":"Present"},{"id":"113","name":"Ivan Tan","class":"C-213","date":"2025-04-23","time":"17:20","status":"Late"},{"id":"114","name":"Jessica Lim","class":"C-214","date":"2025-04-24","time":"17:25","status":"Absent"},{"id":"115","name":"Kevin Wong","class":"C-215","date":"2025-04-25","time":"17:30","status":"Present"},{"id":"116","name":"Lucas Chen","class":"C-216","date":"2025-04-26","time":"17:35","status":"Late"},{"id":"117","name":"Mia Ong","class":"C-217","date":"2025-04-27","time":"17:40","status":"Absent"},{"id":"118","name":"Nathan Koh","class":"C-218","date":"2025-04-28","time":"17:45","status":"Present"},{"id":"119","name":"Olivia Teo","class":"C-219","date":"2025-04-29","time":"17:50","status":"Late"},{"id":"120","name":"Pauline Ang","class":"C-220","date":"2025-04-30","time":"17:55","status":"Absent"},{"id":"121","name":"Quinn Sim","class":"C-221","date":"2025-05-01","time":"18:00","status":"Present"},{"id":"122","name":"Rachel Ng","class":"C-222","date":"2025-05-02","time":"18:05","status":"Late"},{"id":"123","name":"Sam Tan","class":"C-223","date":"2025-05-03","time":"18:10","status":"Absent"},{"id":"124","name":"Tina Lim","class":"C-224","date":"2025-05-04","time":"18:15","status":"Present"},{"id":"125","name":"Ulysses Wong","class":"C-225","date":"2025-05-05","time":"18:20","status":"Late"},{"id":"126","name":"Vanessa Chen","class":"C-226","date":"2025-05-06","time":"18:25","status":"Absent"},{"id":"127","name":"Wilson Ong","class":"C-227","date":"2025-05-07","time":"18:30","status":"Present"},{"id":"128","name":"Xavier Koh","class":"C-228","date":"2025-05-08","time":"18:35","status":"Late"},{"id":"129","name":"Yvonne Teo","class":"C-229","date":"2025-05-09","time":"18:40","status":"Absent"},{"id":"130","name":"Zoe Ang","class":"C-230","date":"2025-05-10","time":"18:45","status":"Present"},{"id":"131","name":"Aaron Sim","class":"C-231","date":"2025-05-11","time":"18:50","status":"Late"},{"id":"132","name":"Bella Ng","class":"C-232","date":"2025-05-12","time":"18:55","status":"Absent"},{"id":"133","name":"Carlos Tan","class":"C-233","date":"2025-05-13","time":"19:00","status":"Present"},{"id":"134","name":"Dana Lim","class":"C-234","date":"2025-05-14","time":"19:05","status":"Late"},{"id":"135","name":"Eric Wong","class":"C-235","date":"2025-05-15","time":"19:10","status":"Absent"},{"id":"136","name":"Fiona Chen","class":"C-236","date":"2025-05-16","time":"19:15","status":"Present"},{"id":"137","name":"Gabriel Ong","class":"C-237","date":"2025-05-17","time":"19:20","status":"Late"},{"id":"138","name":"Hannah Koh","class":"C-238","date":"2025-05-18","time":"19:25","status":"Absent"},{"id":"139","name":"Ivan Teo","class":"C-239","date":"2025-05-19","time":"19:30","status":"Present"},{"id":"140","name":"Jessica Ang","class":"C-240","date":"2025-05-20","time":"19:35","status":"Late"},{"id":"141","name":"Kevin Sim","class":"C-241","date":"2025-05-21","time":"19:40","status":"Absent"},{"id":"142","name":"Lucas Ng","class":"C-242","date":"2025-05-22","time":"19:45","status":"Present"},{"id":"143","name":"Mia Tan","class":"C-243","date":"2025-05-23","time":"19:50","status":"Late"},{"id":"144","name":"Nathan Lim","class":"C-244","date":"2025-05-24","time":"19:55","status":"Absent"},{"id":"145","name":"Olivia Wong","class":"C-245","date":"2025-05-25","time":"20:00","status":"Present"},{"id":"146","name":"Pauline Chen","class":"C-246","date":"2025-05-26","time":"20:05","status":"Late"},{"id":"147","name":"Quinn Ong","class":"C-247","date":"2025-05-27","time":"20:10","status":"Absent"},{"id":"148","name":"Rachel Koh","class":"C-248","date":"2025-05-28","time":"20:15","status":"Present"},{"id":"149","name":"Sam Teo","class":"C-249","date":"2025-05-29","time":"20:20","status":"Late"},{"id":"150","name":"Tina Ang","class":"C-250","date":"2025-05-30","time":"20:25","status":"Absent"},{"id":"151","name":"Ulysses Sim","class":"C-251","date":"2025-05-31","time":"20:30","status":"Present"},{"id":"152","name":"Vanessa Ng","class":"C-252","date":"2025-06-01","time":"20:35","status":"Late"},{"id":"153","name":"Wilson Tan","class":"C-253","date":"2025-06-02","time":"20:40","status":"Absent"},{"id":"154","name":"Xavier Lim","class":"C-254","date":"2025-06-03","time":"20:45","status":"Present"},{"id":"155","name":"Yvonne Wong","class":"C-255","date":"2025-06-04","time":"20:50","status":"Late"},{"id":"156","name":"Zoe Chen","class":"C-256","date":"2025-06-05","time":"20:55","status":"Absent"},{"id":"157","name":"Aaron Ong","class":"C-257","date":"2025-06-06","time":"21:00","status":"Present"},{"id":"158","name":"Bella Koh","class":"C-258","date":"2025-06-07","time":"21:05","status":"Late"},{"id":"159","name":"Carlos Teo","class":"C-259","date":"2025-06-08","time":"21:10","status":"Absent"},{"id":"160","name":"Dana Ang","class":"C-260","date":"2025-06-09","time":"21:15","status":"Present"},{"id":"161","name":"Eric Sim","class":"C-261","date":"2025-06-10","time":"21:20","status":"Late"},{"id":"162","name":"Fiona Ng","class":"C-262","date":"2025-06-11","time":"21:25","status":"Absent"},{"id":"163","name":"Gabriel Tan","class":"C-263","date":"2025-06-12","time":"21:30","status":"Present"},{"id":"164","name":"Hannah Lim","class":"C-264","date":"2025-06-13","time":"21:35","status":"Late"},{"id":"165","name":"Ivan Wong","class":"C-265","date":"2025-06-14","time":"21:40","status":"Absent"},{"id":"166","name":"Jessica Chen","class":"C-266","date":"2025-06-15","time":"21:45","status":"Present"},{"id":"167","name":"Kevin Ong","class":"C-267","date":"2025-06-16","time":"21:50","status":"Late"},{"id":"168","name":"Lucas Koh","class":"C-268","date":"2025-06-17","time":"21:55","status":"Absent"},{"id":"169","name":"Mia Teo","class":"C-269","date":"2025-06-18","time":"22:00","status":"Present"},{"id":"170","name":"Nathan Ang","class":"C-270","date":"2025-06-19","time":"22:05","status":"Late"},{"id":"171","name":"Olivia Sim","class":"C-271","date":"2025-06-20","time":"22:10","status":"Absent"},{"id":"172","name":"Pauline Ng","class":"C-272","date":"2025-06-21","time":"22:15","status":"Present"},{"id":"173","name":"Quinn Tan","class":"C-273","date":"2025-06-22","time":"22:20","status":"Late"},{"id":"174","name":"Rachel Lim","class":"C-274","date":"2025-06-23","time":"22:25","status":"Absent"},{"id":"175","name":"Sam Wong","class":"C-275","date":"2025-06-24","time":"22:30","status":"Present"},{"id":"176","name":"Tina Chen","class":"C-276","date":"2025-06-25","time":"22:35","status":"Late"},{"id":"177","name":"Ulysses Ong","class":"C-277","date":"2025-06-26","time":"22:40","status":"Absent"},{"id":"178","name":"Vanessa Koh","class":"C-278","date":"2025-06-27","time":"22:45","status":"Present"},{"id":"179","name":"Wilson Teo","class":"C-279","date":"2025-06-28","time":"22:50","status":"Late"},{"id":"180","name":"Xavier Ang","class":"C-280","date":"2025-06-29","time":"22:55","status":"Absent"},{"id":"181","name":"Yvonne Sim","class":"C-281","date":"2025-06-30","time":"23:00","status":"Present"},{"id":"182","name":"Zoe Ng","class":"C-282","date":"2025-07-01","time":"23:05","status":"Late"},{"id":"183","name":"Aaron Tan","class":"C-283","date":"2025-07-02","time":"23:10","status":"Absent"},{"id":"184","name":"Bella Lim","class":"C-284","date":"2025-07-03","time":"23:15","status":"Present"},{"id":"185","name":"Carlos Wong","class":"C-285","date":"2025-07-04","time":"23:20","status":"Late"},{"id":"186","name":"Dana Chen","class":"C-286","date":"2025-07-05","time":"23:25","status":"Absent"},{"id":"187","name":"Eric Ong","class":"C-287","date":"2025-07-06","time":"23:30","status":"Present"},{"id":"188","name":"Fiona Koh","class":"C-288","date":"2025-07-07","time":"23:35","status":"Late"},{"id":"189","name":"Gabriel Teo","class":"C-289","date":"2025-07-08","time":"23:40","status":"Absent"},{"id":"190","name":"Hannah Ang","class":"C-290","date":"2025-07-09","time":"23:45","status":"Present"},{"id":"191","name":"Ivan Sim","class":"C-291","date":"2025-07-10","time":"23:50","status":"Late"},{"id":"192","name":"Jessica Ng","class":"C-292","date":"2025-07-11","time":"23:55","status":"Absent"},{"id":"193","name":"Kevin Tan","class":"C-293","date":"2025-07-12","time":"00:00","status":"Present"},{"id":"194","name":"Lucas Lim","class":"C-294","date":"2025-07-13","time":"00:05","status":"Late"},{"id":"195","name":"Mia Wong","class":"C-295","date":"2025-07-14","time":"00:10","status":"Absent"},{"id":"196","name":"Nathan Chen","class":"C-296","date":"2025-07-15","time":"00:15","status":"Present"},{"id":"197","name":"Olivia Ong","class":"C-297","date":"2025-07-16","time":"00:20","status":"Late"},{"id":"198","name":"Pauline Koh","class":"C-298","date":"2025-07-17","time":"00:25","status":"Absent"},{"id":"199","name":"Quinn Teo","class":"C-299","date":"2025-07-18","time":"00:30","status":"Present"},{"id":"200","name":"Rachel Ang","class":"C-300","date":"2025-07-19","time":"00:35","status":"Late"},{"id":"201","name":"Sam Sim","class":"C-301","date":"2025-07-20","time":"00:40","status":"Absent"},{"id":"202","name":"Tina Ng","class":"C-302","date":"2025-07-21","time":"00:45","status":"Present"},{"id":"203","name":"Ulysses Tan","class":"C-303","date":"2025-07-22","time":"00:50","status":"Late"},{"id":"204","name":"Vanessa Lim","class":"C-304","date":"2025-07-23","time":"00:55","status":"Absent"},{"id":"205","name":"Wilson Wong","class":"C-305","date":"2025-07-24","time":"01:00","status":"Present"},{"id":"206","name":"Xavier Chen","class":"C-306","date":"2025-07-25","time":"01:05","status":"Late"},{"id":"207","name":"Yvonne Ong","class":"C-307","date":"2025-07-26","time":"01:10","status":"Absent"},{"id":"208","name":"Zoe Koh","class":"C-308","date":"2025-07-27","time":"01:15","status":"Present"},{"id":"209","name":"Aaron Teo","class":"C-309","date":"2025-07-28","time":"01:20","status":"Late"},{"id":"210","name":"Bella Ang","class":"C-310","date":"2025-07-29","time":"01:25","status":"Absent"},{"id":"211","name":"Carlos Sim","class":"C-311","date":"2025-07-30","time":"01:30","status":"Present"},{"id":"212","name":"Dana Ng","class":"C-312","date":"2025-07-31","time":"01:35","status":"Late"},{"id":"213","name":"Eric Tan","class":"C-313","date":"2025-08-01","time":"01:40","status":"Absent"},{"id":"214","name":"Fiona Lim","class":"C-314","date":"2025-08-02","time":"01:45","status":"Present"},{"id":"215","name":"Gabriel Wong","class":"C-315","date":"2025-08-03","time":"01:50","status":"Late"},{"id":"216","name":"Hannah Chen","class":"C-316","date":"2025-08-04","time":"01:55","status":"Absent"},{"id":"217","name":"Ivan Ong","class":"C-317","date":"2025-08-05","time":"02:00","status":"Present"},{"id":"218","name":"Jessica Koh","class":"C-318","date":"2025-08-06","time":"02:05","status":"Late"},{"id":"219","name":"Kevin Teo","class":"C-319","date":"2025-08-07","time":"02:10","status":"Absent"},{"id":"220","name":"Lucas Ang","class":"C-320","date":"2025-08-08","time":"02:15","status":"Present"},{"id":"221","name":"Mia Sim","class":"C-321","date":"2025-08-09","time":"02:20","status":"Late"},{"id":"222","name":"Nathan Ng","class":"C-322","date":"2025-08-10","time":"02:25","status":"Absent"},{"id":"223","name":"Olivia Tan","class":"C-323","date":"2025-08-11","time":"02:30","status":"Present"},{"id":"224","name":"Pauline Lim","class":"C-324","date":"2025-08-12","time":"02:35","status":"Late"},{"id":"225","name":"Quinn Wong","class":"C-325","date":"2025-08-13","time":"02:40","status":"Absent"},{"id":"226","name":"Rachel Chen","class":"C-326","date":"2025-08-14","time":"02:45","status":"Present"},{"id":"227","name":"Sam Ong","class":"C-327","date":"2025-08-15","time":"02:50","status":"Late"},{"id":"228","name":"Tina Koh","class":"C-328","date":"2025-08-16","time":"02:55","status":"Absent"},{"id":"229","name":"Ulysses Teo","class":"C-329","date":"2025-08-17","time":"03:00","status":"Present"},{"id":"230","name":"Vanessa Ang","class":"C-330","date":"2025-08-18","time":"03:05","status":"Late"},{"id":"231","name":"Wilson Sim","class":"C-331","date":"2025-08-19","time":"03:10","status":"Absent"},{"id":"232","name":"Xavier Ng","class":"C-332","date":"2025-08-20","time":"03:15","status":"Present"},{"id":"233","name":"Yvonne Tan","class":"C-333","date":"2025-08-21","time":"03:20","status":"Late"},{"id":"234","name":"Zoe Lim","class":"C-334","date":"2025-08-22","time":"03:25","status":"Absent"},{"id":"235","name":"Aaron Wong","class":"C-335","date":"2025-08-23","time":"03:30","status":"Present"},{"id":"236","name":"Bella Chen","class":"C-336","date":"2025-08-24","time":"03:35","status":"Late"},{"id":"237","name":"Carlos Ong","class":"C-337","date":"2025-08-25","time":"03:40","status":"Absent"},{"id":"238","name":"Dana Koh","class":"C-338","date":"2025-08-26","time":"03:45","status":"Present"},{"id":"239","name":"Eric Teo","class":"C-339","date":"2025-08-27","time":"03:50","status":"Late"},{"id":"240","name":"Fiona Ang","class":"C-340","date":"2025-08-28","time":"03:55","status":"Absent"},{"id":"241","name":"Gabriel Sim","class":"C-341","date":"2025-08-29","time":"04:00","status":"Present"},{"id":"242","name":"Hannah Ng","class":"C-342","date":"2025-08-30","time":"04:05","status":"Late"},{"id":"243","name":"Ivan Tan","class":"C-343","date":"2025-08-31","time":"04:10","status":"Absent"},{"id":"244","name":"Jessica Lim","class":"C-344","date":"2025-09-01","time":"04:15","status":"Present"},{"id":"245","name":"Kevin Wong","class":"C-345","date":"2025-09-02","time":"04:20","status":"Late"},{"id":"246","name":"Lucas Chen","class":"C-346","date":"2025-09-03","time":"04:25","status":"Absent"},{"id":"247","name":"Mia Ong","class":"C-347","date":"2025-09-04","time":"04:30","status":"Present"},{"id":"248","name":"Nathan Koh","class":"C-348","date":"2025-09-05","time":"04:35","status":"Late"},{"id":"249","name":"Olivia Teo","class":"C-349","date":"2025-09-06","time":"04:40","status":"Absent"},{"id":"250","name":"Pauline Ang","class":"C-350","date":"2025-09-07","time":"04:45","status":"Present"},{"id":"251","name":"Quinn Sim","class":"C-351","date":"2025-09-08","time":"04:50","status":"Late"},{"id":"252","name":"Rachel Ng","class":"C-352","date":"2025-09-09","time":"04:55","status":"Absent"},{"id":"253","name":"Sam Tan","class":"C-353","date":"2025-09-10","time":"05:00","status":"Present"},{"id":"254","name":"Tina Lim","class":"C-354","date":"2025-09-11","time":"05:05","status":"Late"},{"id":"255","name":"Ulysses Wong","class":"C-355","date":"2025-09-12","time":"05:10","status":"Absent"},{"id":"256","name":"Vanessa Chen","class":"C-356","date":"2025-09-13","time":"05:15","status":"Present"},{"id":"257","name":"Wilson Ong","class":"C-357","date":"2025-09-14","time":"05:20","status":"Late"},{"id":"258","name":"Xavier Koh","class":"C-358","date":"2025-09-15","time":"05:25","status":"Absent"},{"id":"259","name":"Yvonne Teo","class":"C-359","date":"2025-09-16","time":"05:30","status":"Present"},{"id":"260","name":"Zoe Ang","class":"C-360","date":"2025-09-17","time":"05:35","status":"Late"},{"id":"261","name":"Aaron Sim","class":"C-361","date":"2025-09-18","time":"05:40","status":"Absent"},{"id":"262","name":"Bella Ng","class":"C-362","date":"2025-09-19","time":"05:45","status":"Present"},{"id":"263","name":"Carlos Tan","class":"C-363","date":"2025-09-20","time":"05:50","status":"Late"},{"id":"264","name":"Dana Lim","class":"C-364","date":"2025-09-21","time":"05:55","status":"Absent"},{"id":"265","name":"Eric Wong","class":"C-365","date":"2025-09-22","time":"06:00","status":"Present"},{"id":"266","name":"Fiona Chen","class":"C-366","date":"2025-09-23","time":"06:05","status":"Late"},{"id":"267","name":"Gabriel Ong","class":"C-367","date":"2025-09-24","time":"06:10","status":"Absent"},{"id":"268","name":"Hannah Koh","class":"C-368","date":"2025-09-25","time":"06:15","status":"Present"},{"id":"269","name":"Ivan Teo","class":"C-369","date":"2025-09-26","time":"06:20","status":"Late"},{"id":"270","name":"Jessica Ang","class":"C-370","date":"2025-09-27","time":"06:25","status":"Absent"},{"id":"271","name":"Kevin Sim","class":"C-371","date":"2025-09-28","time":"06:30","status":"Present"},{"id":"272","name":"Lucas Ng","class":"C-372","date":"2025-09-29","time":"06:35","status":"Late"},{"id":"273","name":"Mia Tan","class":"C-373","date":"2025-09-30","time":"06:40","status":"Absent"},{"id":"274","name":"Nathan Lim","class":"C-374","date":"2025-10-01","time":"06:45","status":"Present"},{"id":"275","name":"Olivia Wong","class":"C-375","date":"2025-10-02","time":"06:50","status":"Late"},{"id":"276","name":"Pauline Chen","class":"C-376","date":"2025-10-03","time":"06:55","status":"Absent"},{"id":"277","name":"Quinn Ong","class":"C-377","date":"2025-10-04","time":"07:00","status":"Present"},{"id":"278","name":"Rachel Koh","class":"C-378","date":"2025-10-05","time":"07:05","status":"Late"},{"id":"279","name":"Sam Teo","class":"C-379","date":"2025-10-06","time":"07:10","status":"Absent"},{"id":"280","name":"Tina Ang","class":"C-380","date":"2025-10-07","time":"07:15","status":"Present"},{"id":"281","name":"Ulysses Sim","class":"C-381","date":"2025-10-08","time":"07:20","status":"Late"},{"id":"282","name":"Vanessa Ng","class":"C-382","date":"2025-10-09","time":"07:25","status":"Absent"},{"id":"283","name":"Wilson Tan","class":"C-383","date":"2025-10-10","time":"07:30","status":"Present"},{"id":"284","name":"Xavier Lim","class":"C-384","date":"2025-10-11","time":"07:35","status":"Late"},{"id":"285","name":"Yvonne Wong","class":"C-385","date":"2025-10-12","time":"07:40","status":"Absent"},{"id":"286","name":"Zoe Chen","class":"C-386","date":"2025-10-13","time":"07:45","status":"Present"},{"id":"287","name":"Aaron Ong","class":"C-387","date":"2025-10-14","time":"07:50","status":"Late"},{"id":"288","name":"Bella Koh","class":"C-388","date":"2025-10-15","time":"07:55","status":"Absent"},{"id":"289","name":"Carlos Teo","class":"C-389","date":"2025-10-16","time":"08:00","status":"Present"},{"id":"290","name":"Dana Ang","class":"C-390","date":"2025-10-17","time":"08:05","status":"Late"},{"id":"291","name":"Eric Sim","class":"C-391","date":"2025-10-18","time":"08:10","status":"Absent"},{"id":"292","name":"Fiona Ng","class":"C-392","date":"2025-10-19","time":"08:15","status":"Present"},{"id":"293","name":"Gabriel Tan","class":"C-393","date":"2025-10-20","time":"08:20","status":"Late"},{"id":"294","name":"Hannah Lim","class":"C-394","date":"2025-10-21","time":"08:25","status":"Absent"},{"id":"295","name":"Ivan Wong","class":"C-395","date":"2025-10-22","time":"08:30","status":"Present"},{"id":"296","name":"Jessica Chen","class":"C-396","date":"2025-10-23","time":"08:35","status":"Late"},{"id":"297","name":"Kevin Ong","class":"C-397","date":"2025-10-24","time":"08:40","status":"Absent"},{"id":"298","name":"Lucas Koh","class":"C-398","date":"2025-10-25","time":"08:45","status":"Present"},{"id":"299","name":"Mia Teo","class":"C-399","date":"2025-10-26","time":"08:50","status":"Late"},{"id":"300","name":"Nathan Ang","class":"C-400","date":"2025-10-27","time":"08:55","status":"Absent"},{"id":"301","name":"Olivia Sim","class":"C-401","date":"2025-10-28","time":"09:00","status":"Present"},{"id":"302","name":"Pauline Ng","class":"C-402","date":"2025-10-29","time":"09:05","status":"Late"},{"id":"303","name":"Quinn Tan","class":"C-403","date":"2025-10-30","time":"09:10","status":"Absent"},{"id":"304","name":"Rachel Lim","class":"C-404","date":"2025-10-31","time":"09:15","status":"Present"},{"id":"305","name":"Sam Wong","class":"C-405","date":"2025-11-01","time":"09:20","status":"Late"},{"id":"306","name":"Tina Chen","class":"C-406","date":"2025-11-02","time":"09:25","status":"Absent"},{"id":"307","name":"Ulysses Ong","class":"C-407","date":"2025-11-03","time":"09:30","status":"Present"},{"id":"308","name":"Vanessa Koh","class":"C-408","date":"2025-11-04","time":"09:35","status":"Late"},{"id":"309","name":"Wilson Teo","class":"C-409","date":"2025-11-05","time":"09:40","status":"Absent"},{"id":"310","name":"Xavier Ang","class":"C-410","date":"2025-11-06","time":"09:45","status":"Present"},{"id":"311","name":"Yvonne Sim","class":"C-411","date":"2025-11-07","time":"09:50","status":"Late"},{"id":"312","name":"Zoe Ng","class":"C-412","date":"2025-11-08","time":"09:55","status":"Absent"},{"id":"313","name":"Aaron Tan","class":"C-413","date":"2025-11-09","time":"10:00","status":"Present"},{"id":"314","name":"Bella Lim","class":"C-414","date":"2025-11-10","time":"10:05","status":"Late"},{"id":"315","name":"Carlos Wong","class":"C-415","date":"2025-11-11","time":"10:10","status":"Absent"},{"id":"316","name":"Dana Chen","class":"C-416","date":"2025-11-12","time":"10:15","status":"Present"},{"id":"317","name":"Eric Ong","class":"C-417","date":"2025-11-13","time":"10:20","status":"Late"},{"id":"318","name":"Fiona Koh","class":"C-418","date":"2025-11-14","time":"10:25","status":"Absent"},{"id":"319","name":"Gabriel Teo","class":"C-419","date":"2025-11-15","time":"10:30","status":"Present"},{"id":"320","name":"Hannah Ang","class":"C-420","date":"2025-11-16","time":"10:35","status":"Late"},{"id":"321","name":"Ivan Sim","class":"C-421","date":"2025-11-17","time":"10:40","status":"Absent"},{"id":"322","name":"Jessica Ng","class":"C-422","date":"2025-11-18","time":"10:45","status":"Present"},{"id":"323","name":"Kevin Tan","class":"C-423","date":"2025-11-19","time":"10:50","status":"Late"},{"id":"324","name":"Lucas Lim","class":"C-424","date":"2025-11-20","time":"10:55","status":"Absent"},{"id":"325","name
```

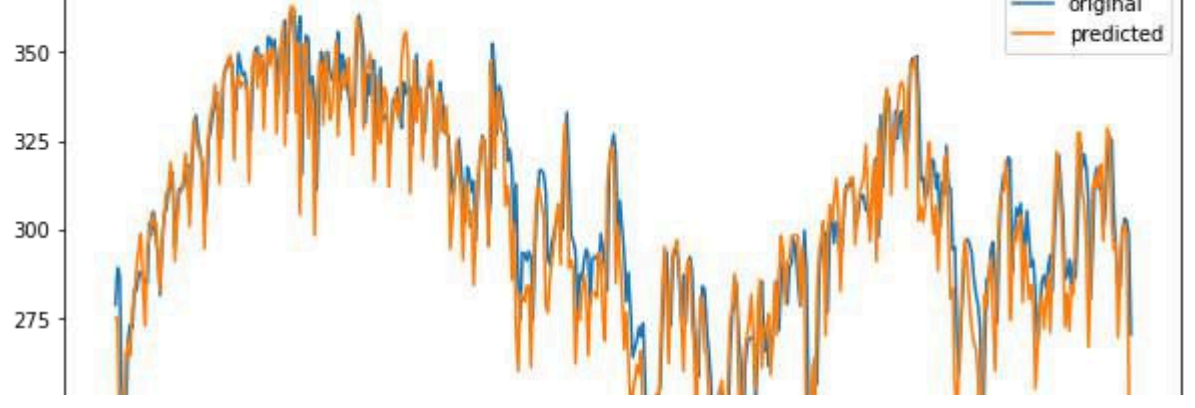
```
01 - mse: 0.899 - ETA: 1s - loss: 0.6276 - mse: 0.924 - ETA: 1s - loss: 0.6771 - mse: 1.049 - ETA: 1s
 - loss: 0.7049 - mse: 1.073 - ETA: 1s - loss: 0.6825 - mse: 1.059 - ETA: 1s - loss: 0.6736 - mse: 1.
025 - ETA: 1s - loss: 0.7010 - mse: 1.099 - ETA: 1s - loss: 0.7219 - mse: 1.140 - ETA: 0s - loss: 0.7
399 - mse: 1.184 - ETA: 0s - loss: 0.7452 - mse: 1.183 - ETA: 0s - loss: 0.7271 - mse: 1.131 - ETA: 0
s - loss: 0.7065 - mse: 1.075 - ETA: 0s - loss: 0.7550 - mse: 1.238 - ETA: 0s - loss: 0.7652 - mse:
 1.251 - ETA: 0s - loss: 0.7723 - mse: 1.263 - ETA: 0s - loss: 0.7718 - mse: 1.264 - ETA: 0s - loss:
 0.7549 - mse: 1.214 - ETA: 0s - loss: 0.7581 - mse: 1.233 - ETA: 0s - loss: 0.7782 - mse: 1.297 - ET
A: 0s - loss: 0.7814 - mse: 1.316 - ETA: 0s - loss: 0.7926 - mse: 1.347 - ETA: 0s - loss: 0.7876 - ms
e: 1.367 - ETA: 0s - loss: 0.7710 - mse: 1.322 - ETA: 0s - loss: 0.7797 - mse: 1.343 - ETA: 0s - los
s: 0.7827 - mse: 1.346 - ETA: 0s - loss: 0.7849 - mse: 1.343 - ETA: 0s - loss: 0.7903 - mse: 1.349 -
 ETA: 0s - loss: 0.7842 - mse: 1.323 - 2s 5ms/step - loss: 0.7814 - mse: 1.3178 - val_loss: 1.4465 -
 val_mse: 4.0509
Epoch 122/1000
355/355 [==============================] - ETA: 0s - loss: 0.2873 - mse: 0.189 - ETA: 1s - loss: 0.55
33 - mse: 0.790 - ETA: 1s - loss: 0.5847 - mse: 0.811 - ETA: 1s - loss: 0.6617 - mse: 0.990 - ETA: 1s
 - loss: 0.6811 - mse: 0.998 - ETA: 1s - loss: 0.6600 - mse: 0.989 - ETA: 1s - loss: 0.6513 - mse: 0.
959 - ETA: 1s - loss: 0.6790 - mse: 1.046 - ETA: 1s - loss: 0.6999 - mse: 1.065 - ETA: 0s - loss: 0.7
172 - mse: 1.119 - ETA: 0s - loss: 0.7201 - mse: 1.119 - ETA: 0s - loss: 0.7008 - mse: 1.073 - ETA: 0
s - loss: 0.7019 - mse: 1.071 - ETA: 0s - loss: 0.7480 - mse: 1.213 - ETA: 0s - loss: 0.7505 - mse:
 1.201 - ETA: 0s - loss: 0.7597 - mse: 1.225 - ETA: 0s - loss: 0.7542 - mse: 1.212 - ETA: 0s - loss:
 0.7334 - mse: 1.154 - ETA: 0s - loss: 0.7448 - mse: 1.196 - ETA: 0s - loss: 0.7640 - mse: 1.256 - ET
A: 0s - loss: 0.7731 - mse: 1.292 - ETA: 0s - loss: 0.7844 - mse: 1.373 - ETA: 0s - loss: 0.7724 - ms
e: 1.332 - ETA: 0s - loss: 0.7653 - mse: 1.318 - ETA: 0s - loss: 0.7759 - mse: 1.338 - ETA: 0s - los
s: 0.7724 - mse: 1.320 - ETA: 0s - loss: 0.7845 - mse: 1.347 - ETA: 0s - loss: 0.7807 - mse: 1.330 -
 2s 5ms/step - loss: 0.7724 - mse: 1.3106 - val_loss: 1.4652 - val_mse: 4.1396
Epoch 123/1000
355/355 [==============================] - ETA: 0s - loss: 0.3649 - mse: 0.156 - ETA: 1s - loss: 0.58
87 - mse: 0.799 - ETA: 1s - loss: 0.6153 - mse: 0.842 - ETA: 1s - loss: 0.6765 - mse: 1.036 - ETA: 1s
 - loss: 0.6992 - mse: 1.040 - ETA: 1s - loss: 0.6922 - mse: 1.016 - ETA: 1s - loss: 0.6695 - mse: 1.
017 - ETA: 1s - loss: 0.7016 - mse: 1.094 - ETA: 1s - loss: 0.7208 - mse: 1.147 - ETA: 1s - loss: 0.7
352 - mse: 1.184 - ETA: 0s - loss: 0.7359 - mse: 1.176 - ETA: 0s - loss: 0.7250 - mse: 1.150 - ETA: 0
s - loss: 0.7055 - mse: 1.093 - ETA: 0s - loss: 0.7257 - mse: 1.146 - ETA: 0s - loss: 0.7597 - mse:
 1.255 - ETA: 0s - loss: 0.7752 - mse: 1.280 - ETA: 0s - loss: 0.7647 - mse: 1.256 - ETA: 0s - loss:
 0.7514 - mse: 1.218 - ETA: 0s - loss: 0.7453 - mse: 1.190 - ETA: 0s - loss: 0.7639 - mse: 1.258 - ET
A: 0s - loss: 0.7812 - mse: 1.317 - ETA: 0s - loss: 0.7860 - mse: 1.326 - ETA: 0s - loss: 0.7846 - ms
e: 1.363 - ETA: 0s - loss: 0.7682 - mse: 1.316 - ETA: 0s - loss: 0.7727 - mse: 1.329 - ETA: 0s - los
s: 0.7797 - mse: 1.346 - ETA: 0s - loss: 0.7825 - mse: 1.346 - ETA: 0s - loss: 0.7877 - mse: 1.357 -
 ETA: 0s - loss: 0.7792 - mse: 1.330 - 2s 5ms/step - loss: 0.7765 - mse: 1.3254 - val_loss: 1.4441 -
 val_mse: 4.0012
Epoch 124/1000
355/355 [==============================] - ETA: 0s - loss: 0.3661 - mse: 0.176 - ETA: 1s - loss: 0.53
15 - mse: 0.754 - ETA: 1s - loss: 0.5850 - mse: 0.825 - ETA: 1s - loss: 0.6541 - mse: 0.993 - ETA: 1s
 - loss: 0.7016 - mse: 1.067 - ETA: 1s - loss: 0.6498 - mse: 0.941 - ETA: 1s - loss: 0.6435 - mse: 0.
948 - ETA: 1s - loss: 0.6630 - mse: 0.990 - ETA: 1s - loss: 0.6995 - mse: 1.097 - ETA: 1s - loss: 0.7
036 - mse: 1.092 - ETA: 1s - loss: 0.7266 - mse: 1.154 - ETA: 0s - loss: 0.7244 - mse: 1.143 - ETA: 0
s - loss: 0.6963 - mse: 1.070 - ETA: 0s - loss: 0.7187 - mse: 1.116 - ETA: 0s - loss: 0.7498 - mse:
 1.211 - ETA: 0s - loss: 0.7607 - mse: 1.234 - ETA: 0s - loss: 0.7513 - mse: 1.219 - ETA: 0s - loss:
 0.7375 - mse: 1.173 - ETA: 0s - loss: 0.7370 - mse: 1.180 - ETA: 0s - loss: 0.7504 - mse: 1.222 - ET
A: 0s - loss: 0.7673 - mse: 1.290 - ETA: 0s - loss: 0.7750 - mse: 1.317 - ETA: 0s - loss: 0.7741 - ms
e: 1.347 - ETA: 0s - loss: 0.7583 - mse: 1.299 - ETA: 0s - loss: 0.7669 - mse: 1.319 - ETA: 0s - los
s: 0.7682 - mse: 1.317 - ETA: 0s - loss: 0.7742 - mse: 1.326 - ETA: 0s - loss: 0.7796 - mse: 1.336 -
 ETA: 0s - loss: 0.7723 - mse: 1.310 - 2s 5ms/step - loss: 0.7699 - mse: 1.3075 - val_loss: 1.4714 -
 val_mse: 4.1651
Epoch 125/1000
355/355 [==============================] - ETA: 0s - loss: 0.3584 - mse: 0.172 - ETA: 1s - loss: 0.56
82 - mse: 0.828 - ETA: 1s - loss: 0.6130 - mse: 0.858 - ETA: 1s - loss: 0.6848 - mse: 1.008 - ETA: 1s
 - loss: 0.6787 - mse: 0.959 - ETA: 1s - loss: 0.6642 - mse: 0.948 - ETA: 1s - loss: 0.6688 - mse: 0.
968 - ETA: 1s - loss: 0.7024 - mse: 1.048 - ETA: 0s - loss: 0.7118 - mse: 1.061 - ETA: 0s - loss: 0.7
296 - mse: 1.128 - ETA: 0s - loss: 0.7239 - mse: 1.112 - ETA: 0s - loss: 0.6949 - mse: 1.043 - ETA: 0
s - loss: 0.7154 - mse: 1.091 - ETA: 0s - loss: 0.7514 - mse: 1.231 - ETA: 0s - loss: 0.7620 - mse:
 1.244 - ETA: 0s - loss: 0.7517 - mse: 1.218 - ETA: 0s - loss: 0.7409 - mse: 1.189 - ETA: 0s - loss:
 0.7327 - mse: 1.159 - ETA: 0s - loss: 0.7518 - mse: 1.228 - ETA: 0s - loss: 0.7695 - mse: 1.289 - ET
A: 0s - loss: 0.7743 - mse: 1.303 - ETA: 0s - loss: 0.7760 - mse: 1.339 - ETA: 0s - loss: 0.7626 - ms
e: 1.298 - ETA: 0s - loss: 0.7655 - mse: 1.305 - ETA: 0s - loss: 0.7694 - mse: 1.310 - ETA: 0s - los
s: 0.7712 - mse: 1.304 - ETA: 0s - loss: 0.7800 - mse: 1.322 - ETA: 0s - loss: 0.7723 - mse: 1.296 -
 2s 5ms/step - loss: 0.7681 - mse: 1.2885 - val_loss: 1.4699 - val_mse: 4.1717
Epoch 126/1000
355/355 [==============================] - ETA: 0s - loss: 0.3233 - mse: 0.139 - ETA: 1s - loss: 0.56
82 - mse: 0.783 - ETA: 1s - loss: 0.6127 - mse: 0.849 - ETA: 1s - loss: 0.6941 - mse: 1.054 - ETA: 1s
 - loss: 0.7153 - mse: 1.081 - ETA: 1s - loss: 0.6905 - mse: 1.061 - ETA: 1s - loss: 0.6924 - mse: 1.
066 - ETA: 1s - loss: 0.7194 - mse: 1.131 - ETA: 0s - loss: 0.7250 - mse: 1.122 - ETA: 0s - loss: 0.7
493 - mse: 1.210 - ETA: 0s - loss: 0.7381 - mse: 1.185 - ETA: 0s - loss: 0.7097 - mse: 1.112 - ETA: 0
s - loss: 0.7266 - mse: 1.160 - ETA: 0s - loss: 0.7571 - mse: 1.263 - ETA: 0s - loss: 0.7707 - mse:
 1.283 - ETA: 0s - loss: 0.7609 - mse: 1.253 - ETA: 0s - loss: 0.7463 - mse: 1.216 - ETA: 0s - loss:
 0.7386 - mse: 1.189 - ETA: 0s - loss: 0.7582 - mse: 1.262 - ETA: 0s - loss: 0.7776 - mse: 1.333 - ET
A: 0s - loss: 0.7854 - mse: 1.353 - ETA: 0s - loss: 0.7816 - mse: 1.372 - ETA: 0s - loss: 0.7664 - ms
e: 1.325 - ETA: 0s - loss: 0.7758 - mse: 1.346 - ETA: 0s - loss: 0.7792 - mse: 1.352 - ETA: 0s - los
s: 0.7815 - mse: 1.346 - ETA: 0s - loss: 0.7870 - mse: 1.355 - ETA: 0s - loss: 0.7800 - mse: 1.328 -
 2s 5ms/step - loss: 0.7757 - mse: 1.3202 - val_loss: 1.4550 - val_mse: 4.0386
Epoch 127/1000
355/355 [==============================] - ETA: 0s - loss: 0.3358 - mse: 0.147 - ETA: 1s - loss: 0.56
99 - mse: 0.822 - ETA: 1s - loss: 0.6066 - mse: 0.856 - ETA: 1s - loss: 0.6659 - mse: 1.013 - ETA: 1s
 - loss: 0.6996 - mse: 1.061 - ETA: 1s - loss: 0.6641 - mse: 1.026 - ETA: 1s - loss: 0.6633 - mse: 1.
002 - ETA: 1s - loss: 0.6824 - mse: 1.036 - ETA: 1s - loss: 0.7196 - mse: 1.140 - ETA: 0s - loss: 0.7
298 - mse: 1.189 - ETA: 0s - loss: 0.7303 - mse: 1.187 - ETA: 0s - loss: 0.7114 - mse: 1.133 - ETA: 0
s - loss: 0.7049 - mse: 1.119 - ETA: 0s - loss: 0.7524 - mse: 1.266 - ETA: 0s - loss: 0.7488 - mse:
 1.233 - ETA: 0s - loss: 0.7542 - mse: 1.250 - ETA: 0s - loss: 0.7479 - mse: 1.234 - ETA: 0s - loss:
 0.7305 - mse: 1.183 - ETA: 0s - loss: 0.7438 - mse: 1.228 - ETA: 0s - loss: 0.7597 - mse: 1.277 - ET
A: 0s - loss: 0.7670 - mse: 1.301 - ETA: 0s - loss: 0.7802 - mse: 1.381 - ETA: 0s - loss: 0.7647 - ms
e: 1.335 - ETA: 0s - loss: 0.7593 - mse: 1.327 - ETA: 0s - loss: 0.7677 - mse: 1.340 - ETA: 0s - los
s: 0.7623 - mse: 1.319 - ETA: 0s - loss: 0.7742 - mse: 1.343 - ETA: 0s - loss: 0.7707 - mse: 1.327 -
 2s 5ms/step - loss: 0.7630 - mse: 1.3097 - val_loss: 1.4554 - val_mse: 4.0521
Epoch 128/1000
355/355 [==============================] - ETA: 0s - loss: 0.3665 - mse: 0.195 - ETA: 1s - loss: 0.59
27 - mse: 0.849 - ETA: 1s - loss: 0.5972 - mse: 0.823 - ETA: 1s - loss: 0.6803 - mse: 1.021 - ETA: 1s
 - loss: 0.6865 - mse: 0.998 - ETA: 1s - loss: 0.6727 - mse: 1.002 - ETA: 1s - loss: 0.6750 - mse: 0.
995 - ETA: 1s - loss: 0.7101 - mse: 1.087 - ETA: 0s - loss: 0.7141 - mse: 1.092 - ETA: 0s - loss: 0.7
332 - mse: 1.165 - ETA: 0s - loss: 0.7249 - mse: 1.133 - ETA: 0s - loss: 0.7028 - mse: 1.081 - ETA: 0
s - loss: 0.7123 - mse: 1.110 - ETA: 0s - loss: 0.7468 - mse: 1.224 - ETA: 0s - loss: 0.7551 - mse:
 1.218 - ETA: 0s - loss: 0.7600 - mse: 1.225 - ETA: 0s - loss: 0.7451 - mse: 1.191 - ETA: 0s - loss:
 0.7315 - mse: 1.152 - ETA: 0s - loss: 0.7446 - mse: 1.201 - ETA: 0s - loss: 0.7620 - mse: 1.263 - ET
A: 0s - loss: 0.7740 - mse: 1.304 - ETA: 0s - loss: 0.7800 - mse: 1.368 - ETA: 0s - loss: 0.7659 - ms
e: 1.322 - ETA: 0s - loss: 0.7674 - mse: 1.329 - ETA: 0s - loss: 0.7760 - mse: 1.346 - ETA: 0s - los
s: 0.7761 - mse: 1.335 - ETA: 0s - loss: 0.7802 - mse: 1.341 - ETA: 0s - loss: 0.7709 - mse: 1.314 -
 2s 5ms/step - loss: 0.7679 - mse: 1.3090 - val_loss: 1.4404 - val_mse: 4.0065
Epoch 129/1000
355/355 [==============================] - ETA: 0s - loss: 0.4565 - mse: 0.266 - ETA: 1s - loss: 0.57
08 - mse: 0.800 - ETA: 1s - loss: 0.5722 - mse: 0.814 - ETA: 1s - loss: 0.6613 - mse: 1.013 - ETA: 1s
 - loss: 0.6731 - mse: 0.987 - ETA: 1s - loss: 0.6645 - mse: 0.993 - ETA: 1s - loss: 0.6867 - mse: 1.
026 - ETA: 1s - loss: 0.7280 - mse: 1.130 - ETA: 0s - loss: 0.7290 - mse: 1.120 - ETA: 0s - loss: 0.7
460 - mse: 1.176 - ETA: 0s - loss: 0.7398 - mse: 1.153 - ETA: 0s - loss: 0.7111 - mse: 1.087 - ETA: 0
s - loss: 0.7287 - mse: 1.127 - ETA: 0s - loss: 0.7572 - mse: 1.222 - ETA: 0s - loss: 0.7662 - mse:
 1.239 - ETA: 0s - loss: 0.7586 - mse: 1.235 - ETA: 0s - loss: 0.7407 - mse: 1.184 - ETA: 0s - loss:
 0.7405 - mse: 1.188 - ETA: 0s - loss: 0.7513 - mse: 1.224 - ETA: 0s - loss: 0.7670 - mse: 1.285 - ET
A: 0s - loss: 0.7775 - mse: 1.314 - ETA: 0s - loss: 0.7741 - mse: 1.338 - ETA: 0s - loss: 0.7569 - ms
e: 1.290 - ETA: 0s - loss: 0.7696 - mse: 1.322 - ETA: 0s - loss: 0.7674 - mse: 1.312 - ETA: 0s - los
s: 0.7729 - mse: 1.322 - ETA: 0s - loss: 0.7784 - mse: 1.336 - ETA: 0s - loss: 0.7697 - mse: 1.308 -
 2s 5ms/step - loss: 0.7693 - mse: 1.3102 - val_loss: 1.4497 - val_mse: 4.0660
Epoch 130/1000
355/355 [==============================] - ETA: 0s - loss: 0.5598 - mse: 0.368 - ETA: 1s - loss: 0.55
71 - mse: 0.800 - ETA: 1s - loss: 0.5737 - mse: 0.805 - ETA: 1s - loss: 0.6479 - mse: 0.969 - ETA: 1s
 - loss: 0.6573 - mse: 0.951 - ETA: 1s - loss: 0.6382 - mse: 0.951 - ETA: 1s - loss: 0.6438 - mse: 0.
962 - ETA: 1s - loss: 0.6766 - mse: 1.046 - ETA: 0s - loss: 0.6957 - mse: 1.067 - ETA: 0s - loss: 0.7
113 - mse: 1.128 - ETA: 0s - loss: 0.7024 - mse: 1.098 - ETA: 0s - loss: 0.6793 - mse: 1.033 - ETA: 0
s - loss: 0.6996 - mse: 1.089 - ETA: 0s - loss: 0.7368 - mse: 1.210 - ETA: 0s - loss: 0.7491 - mse:
 1.221 - ETA: 0s - loss: 0.7404 - mse: 1.193 - ETA: 0s - loss: 0.7258 - mse: 1.161 - ETA: 0s - loss:
 0.7171 - mse: 1.134 - ETA: 0s - loss: 0.7328 - mse: 1.193 - ETA: 0s - loss: 0.7504 - mse: 1.256 - ET
A: 0s - loss: 0.7611 - mse: 1.287 - ETA: 0s - loss: 0.7579 - mse: 1.309 - ETA: 0s - loss: 0.7419 - ms
e: 1.263 - ETA: 0s - loss: 0.7530 - mse: 1.285 - ETA: 0s - loss: 0.7531 - mse: 1.284 - ETA: 0s - los
s: 0.7556 - mse: 1.285 - ETA: 0s - loss: 0.7618 - mse: 1.299 - ETA: 0s - loss: 0.7550 - mse: 1.276 -
 2s 5ms/step - loss: 0.7541 - mse: 1.2768 - val_loss: 1.4607 - val_mse: 4.0615
Epoch 131/1000
355/355 [==============================] - ETA: 0s - loss: 0.4470 - mse: 0.233 - ETA: 1s - loss: 0.56
69 - mse: 0.796 - ETA: 1s - loss: 0.5920 - mse: 0.874 - ETA: 1s - loss: 0.6802 - mse: 1.079 - ETA: 1s
 - loss: 0.6933 - mse: 1.047 - ETA: 1s - loss: 0.6657 - mse: 1.014 - ETA: 1s - loss: 0.6765 - mse: 1.
022 - ETA: 1s - loss: 0.7073 - mse: 1.100 - ETA: 0s - loss: 0.7228 - mse: 1.122 - ETA: 0s - loss: 0.7
455 - mse: 1.195 - ETA: 0s - loss: 0.7426 - mse: 1.175 - ETA: 0s - loss: 0.7172 - mse: 1.110 - ETA: 0
s - loss: 0.7259 - mse: 1.137 - ETA: 0s - loss: 0.7665 - mse: 1.269 - ETA: 0s - loss: 0.7706 - mse:
 1.258 - ETA: 0s - loss: 0.7669 - mse: 1.258 - ETA: 0s - loss: 0.7539 - mse: 1.223 - ETA: 0s - loss:
 0.7448 - mse: 1.191 - ETA: 0s - loss: 0.7619 - mse: 1.261 - ETA: 0s - loss: 0.7797 - mse: 1.339 - ET
A: 0s - loss: 0.7811 - mse: 1.335 - ETA: 0s - loss: 0.7842 - mse: 1.382 - ETA: 0s - loss: 0.7680 - ms
e: 1.335 - ETA: 0s - loss: 0.7706 - mse: 1.342 - ETA: 0s - loss: 0.7734 - mse: 1.348 - ETA: 0s - los
s: 0.7726 - mse: 1.338 - ETA: 0s - loss: 0.7825 - mse: 1.358 - ETA: 0s - loss: 0.7747 - mse: 1.330 -
 2s 5ms/step - loss: 0.7725 - mse: 1.3280 - val_loss: 1.4417 - val_mse: 4.0788
Epoch 132/1000
355/355 [==============================] - ETA: 0s - loss: 0.4707 - mse: 0.283 - ETA: 1s - loss: 0.65
77 - mse: 1.001 - ETA: 1s - loss: 0.6282 - mse: 0.915 - ETA: 1s - loss: 0.6900 - mse: 1.083 - ETA: 1s
 - loss: 0.7238 - mse: 1.128 - ETA: 1s - loss: 0.6835 - mse: 0.995 - ETA: 1s - loss: 0.6894 - mse: 1.
079 - ETA: 1s - loss: 0.7144 - mse: 1.134 - ETA: 1s - loss: 0.7336 - mse: 1.166 - ETA: 1s - loss: 0.7
434 - mse: 1.206 - ETA: 0s - loss: 0.7495 - mse: 1.217 - ETA: 0s - loss: 0.7268 - mse: 1.152 - ETA: 0
s - loss: 0.7140 - mse: 1.113 - ETA: 0s - loss: 0.7671 - mse: 1.282 - ETA: 0s - loss: 0.7645 - mse:
 1.254 - ETA: 0s - loss: 0.7670 - mse: 1.259 - ETA: 0s - loss: 0.7581 - mse: 1.237 - ETA: 0s - loss:
 0.7396 - mse: 1.183 - ETA: 0s - loss: 0.7501 - mse: 1.222 - ETA: 0s - loss: 0.7675 - mse: 1.286 - ET
A: 0s - loss: 0.7760 - mse: 1.310 - ETA: 0s - loss: 0.7894 - mse: 1.384 - ETA: 0s - loss: 0.7747 - ms
e: 1.338 - ETA: 0s - loss: 0.7683 - mse: 1.328 - ETA: 0s - loss: 0.7769 - mse: 1.342 - ETA: 0s - los
s: 0.7773 - mse: 1.334 - ETA: 0s - loss: 0.7826 - mse: 1.338 - ETA: 0s - loss: 0.7827 - mse: 1.337 -
 ETA: 0s - loss: 0.7736 - mse: 1.311 - 2s 5ms/step - loss: 0.7736 - mse: 1.3112 - val_loss: 1.4549 -
 val_mse: 4.0838
Epoch 133/1000
355/355 [==============================] - ETA: 0s - loss: 0.2794 - mse: 0.121 - ETA: 1s - loss: 0.57
00 - mse: 0.783 - ETA: 1s - loss: 0.6091 - mse: 0.848 - ETA: 1s - loss: 0.6700 - mse: 0.976 - ETA: 1s
 - loss: 0.7069 - mse: 1.037 - ETA: 1s - loss: 0.6817 - mse: 1.011 - ETA: 1s - loss: 0.6673 - mse: 0.
971 - ETA: 1s - loss: 0.7016 - mse: 1.069 - ETA: 1s - loss: 0.7224 - mse: 1.100 - ETA: 0s - loss: 0.7
312 - mse: 1.127 - ETA: 0s - loss: 0.7305 - mse: 1.125 - ETA: 0s - loss: 0.7162 - mse: 1.081 - ETA: 0
s - loss: 0.7202 - mse: 1.100 - ETA: 0s - loss: 0.7639 - mse: 1.239 - ETA: 0s - loss: 0.7727 - mse:
 1.236 - ETA: 0s - loss: 0.7774 - mse: 1.250 - ETA: 0s - loss: 0.7605 - mse: 1.209 - ETA: 0s - loss:
 0.7489 - mse: 1.176 - ETA: 0s - loss: 0.7622 - mse: 1.216 - ETA: 0s - loss: 0.7811 - mse: 1.281 - ET
A: 0s - loss: 0.7901 - mse: 1.306 - ETA: 0s - loss: 0.7996 - mse: 1.385 - ETA: 0s - loss: 0.7854 - ms
e: 1.339 - ETA: 0s - loss: 0.7843 - mse: 1.341 - ETA: 0s - loss: 0.7886 - mse: 1.349 - ETA: 0s - los
s: 0.7954 - mse: 1.356 - ETA: 0s - loss: 0.8047 - mse: 1.377 - ETA: 0s - loss: 0.7958 - mse: 1.347 -
 2s 5ms/step - loss: 0.7895 - mse: 1.3363 - val_loss: 1.4395 - val_mse: 4.0295
Epoch 134/1000
355/355 [==============================] - ETA: 0s - loss: 0.3499 - mse: 0.236 - ETA: 1s - loss: 0.58
70 - mse: 0.846 - ETA: 1s - loss: 0.5887 - mse: 0.814 - ETA: 1s - loss: 0.6728 - mse: 1.010 - ETA: 1s
 - loss: 0.6785 - mse: 0.996 - ETA: 1s - loss: 0.6565 - mse: 0.975 - ETA: 1s - loss: 0.6631 - mse: 0.
978 - ETA: 1s - loss: 0.7071 - mse: 1.084 - ETA: 0s - loss: 0.7129 - mse: 1.084 - ETA: 0s - loss: 0.7
300 - mse: 1.146 - ETA: 0s - loss: 0.7190 - mse: 1.118 - ETA: 0s - loss: 0.6945 - mse: 1.051 - ETA: 0
s - loss: 0.7180 - mse: 1.112 - ETA: 0s - loss: 0.7492 - mse: 1.223 - ETA: 0s - loss: 0.7620 - mse:
 1.243 - ETA: 0s - loss: 0.7525 - mse: 1.219 - ETA: 0s - loss: 0.7362 - mse: 1.180 - ETA: 0s - loss:
 0.7307 - mse: 1.158 - ETA: 0s - loss: 0.7449 - mse: 1.220 - ETA: 0s - loss: 0.7675 - mse: 1.297 - ET
A: 0s - loss: 0.7760 - mse: 1.318 - ETA: 0s - loss: 0.7759 - mse: 1.349 - ETA: 0s - loss: 0.7589 - ms
e: 1.301 - ETA: 0s - loss: 0.7663 - mse: 1.321 - ETA: 0s - loss: 0.7679 - mse: 1.324 - ETA: 0s - los
s: 0.7710 - mse: 1.323 - ETA: 0s - loss: 0.7760 - mse: 1.327 - ETA: 0s - loss: 0.7665 - mse: 1.296 -
 2s 5ms/step - loss: 0.7653 - mse: 1.2964 - val_loss: 1.4765 - val_mse: 4.1857

saved the model successfully
```

- Training all the the columns simultaneously making the juypter notebook freeze
- so i ran the same code in pycharm and trained all the models similar to the above
- **check the training.py file**

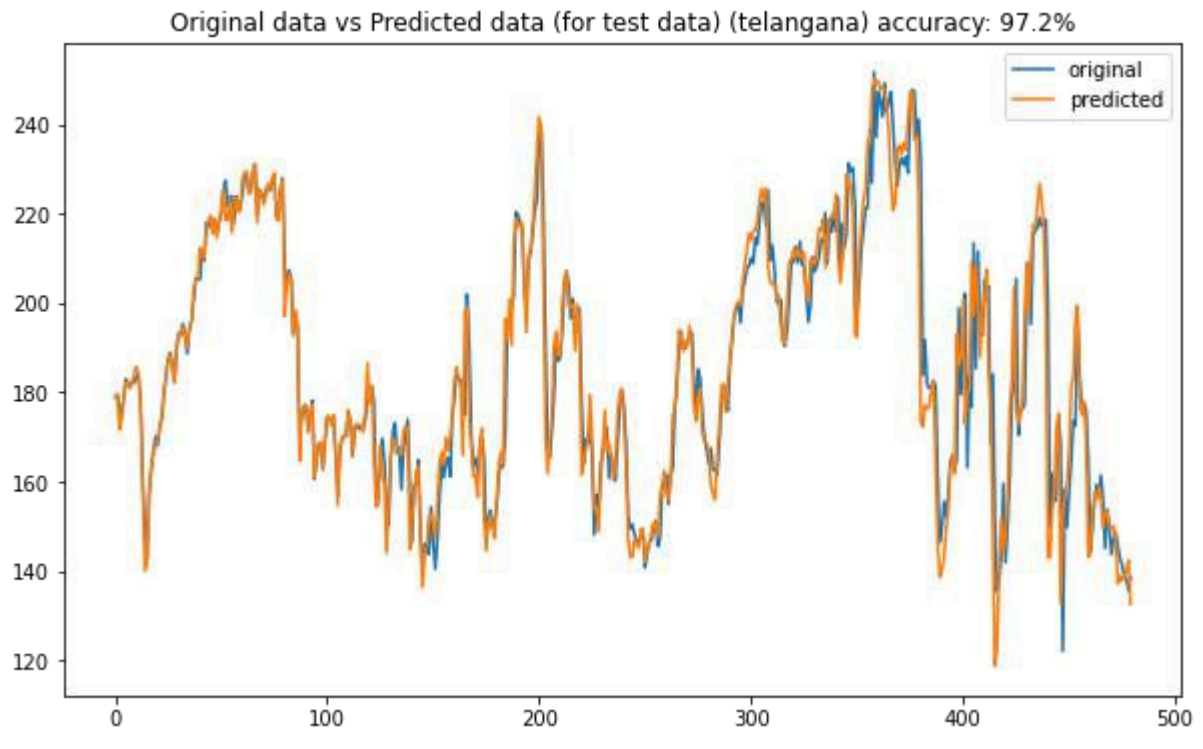
In [2]:
```
# import pandas as pd
# import time

# df3 = pd.read_csv('pycharm_data.csv')
# df3 = df3.drop(columns=['Unnamed: 0', 'date'])


# for i in df3.columns:
#     if i in ['telangana','andhra pradesh','assam'];
#         continue
#     else:
#         predict_rem(i)
#         time.sleep(15)
```

Visualizing the accuracy of saved models on the test data

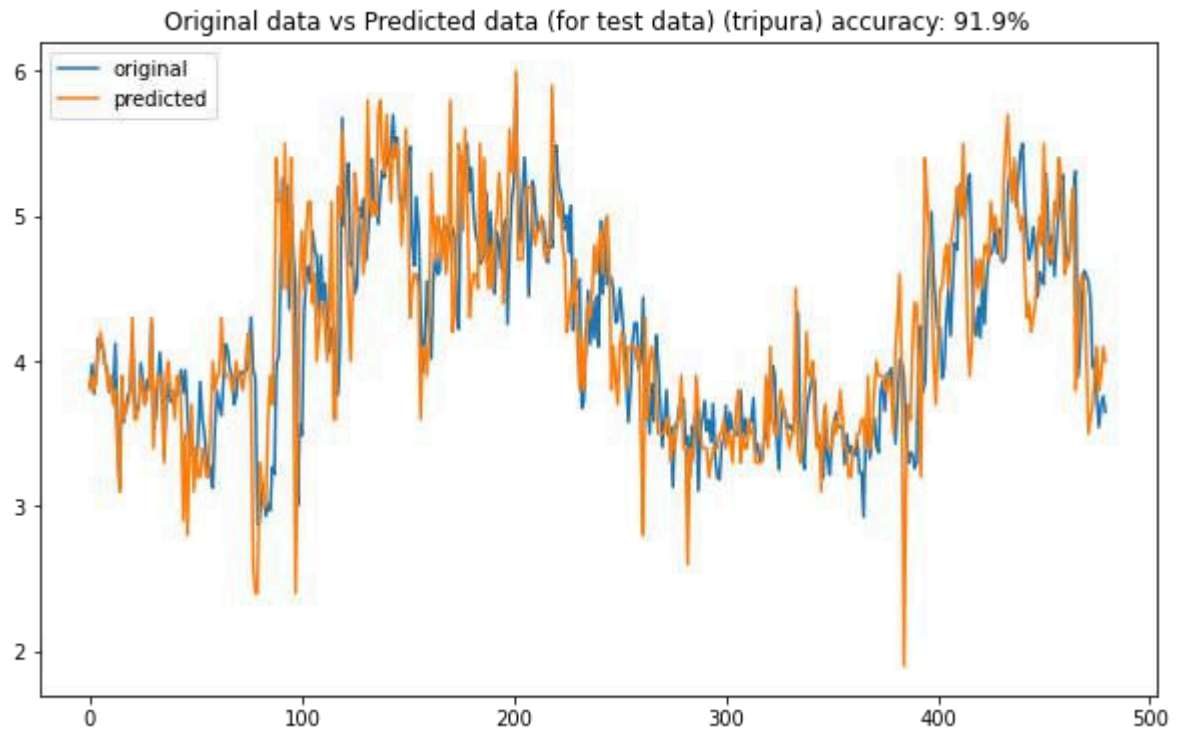
```
import pandas as pd
import numpy as np
import os
DATA_DIR = '/kaggle/input/finance-data.zip/'
df1 = pd.read_csv(os.path.join(DATA_DIR, 'Sheet1.csv'))
df1.shape

def fix_cols(df, prefix=None):
    '''Fix column names by stripping whitespace and optionally adding prefix.'''
    from pandas import DataFrame
    assert isinstance(df, DataFrame)
    df.columns = [c.strip().replace(' ', '_').lower() for c in df.columns]
    if prefix: df.columns = [prefix + '_' + c for c in df.columns if c != 'date']
    return df

def drop_duplicate_dates(df, verbose=1):
    '''Remove duplicate dates.'''
    ndrop = len(df.index.duplicated())
    if ndrop and verbose: print('Dropping %d duplicate dates.' % ndrop)
    return df.groupby(df.index)
    .mean().sort_index()

df1 = fix_cols(df1)
df1 = drop_duplicate_dates(df1)

# Convert to datetime index.
df1.index = pd.to_datetime(df1.index)

# Sort by date.
df1 = df1.sort_index()

# Show shape and head.
df1.shape, df1.head()

# Save to CSV.
df1.to_csv('/kaggle/working/finance_data.csv')

# Load back and check.
df = pd.read_csv('/kaggle/working/finance_data.csv', index_col=0, parse_dates=True)
df.shape, df.head()

# Check for missing values.
df.isnull().sum()

# Check for infinite values.
import numpy as np
df.isin([np.inf, -np.inf]).sum()

# Describe the data.
df.describe()

# Save to CSV.
df
```


original
predicted
350
325
300
275

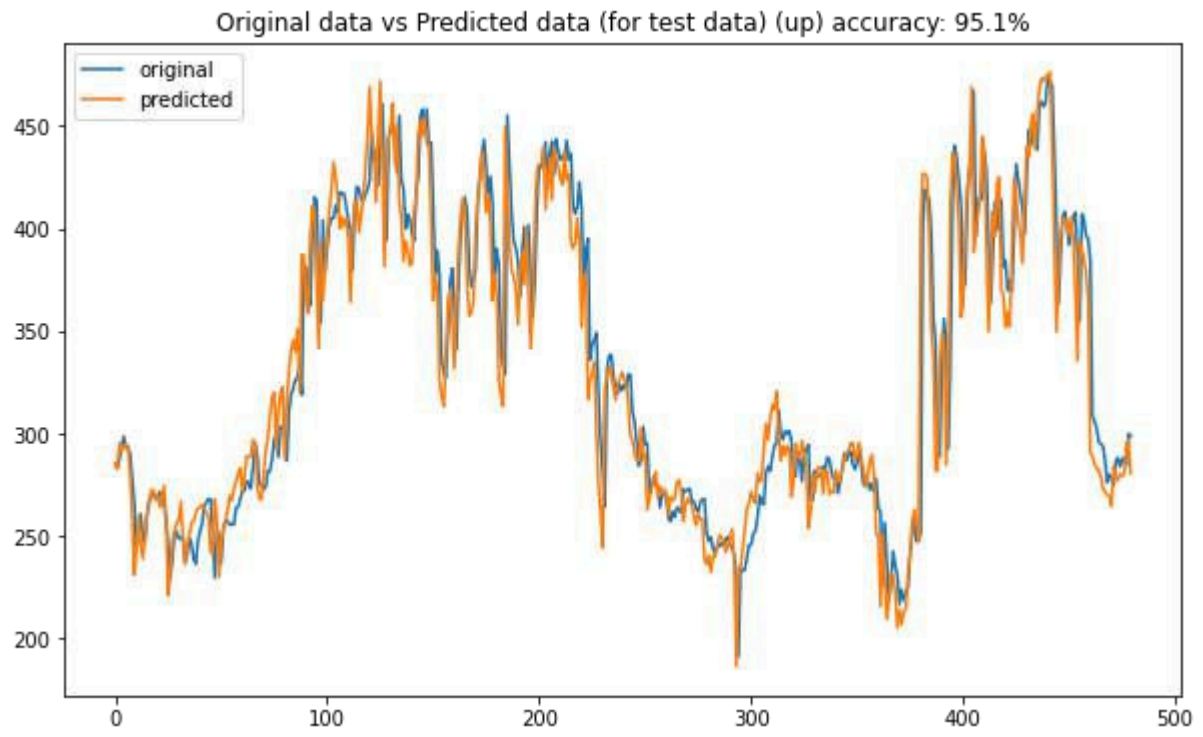
==========


Original data vs Predicted data (for test data) (telangana) accuracy: 97.2%
original
predicted
240
220
200
180
160
140
120
0
100
200
300
400
500

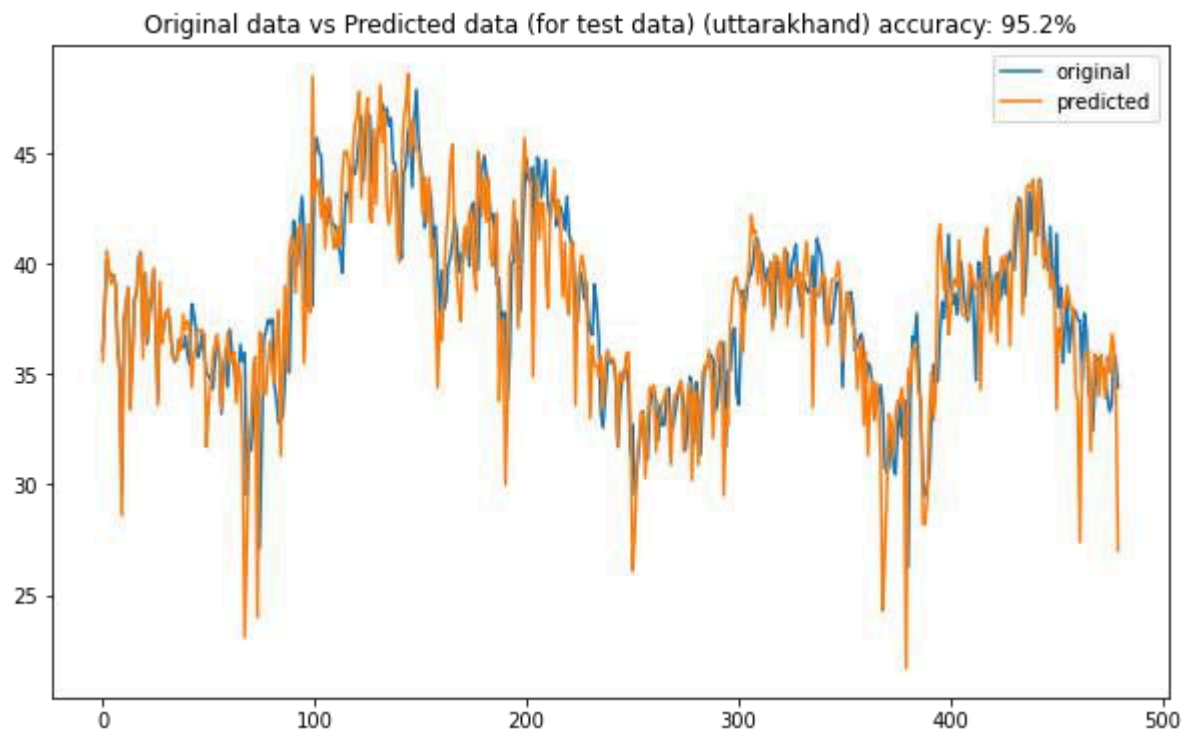
==========


Original data vs Predicted data (for test data) (tripura) accuracy: 91.9%
original
predicted
6
5
4
3
2
0
100
200
300
400
500

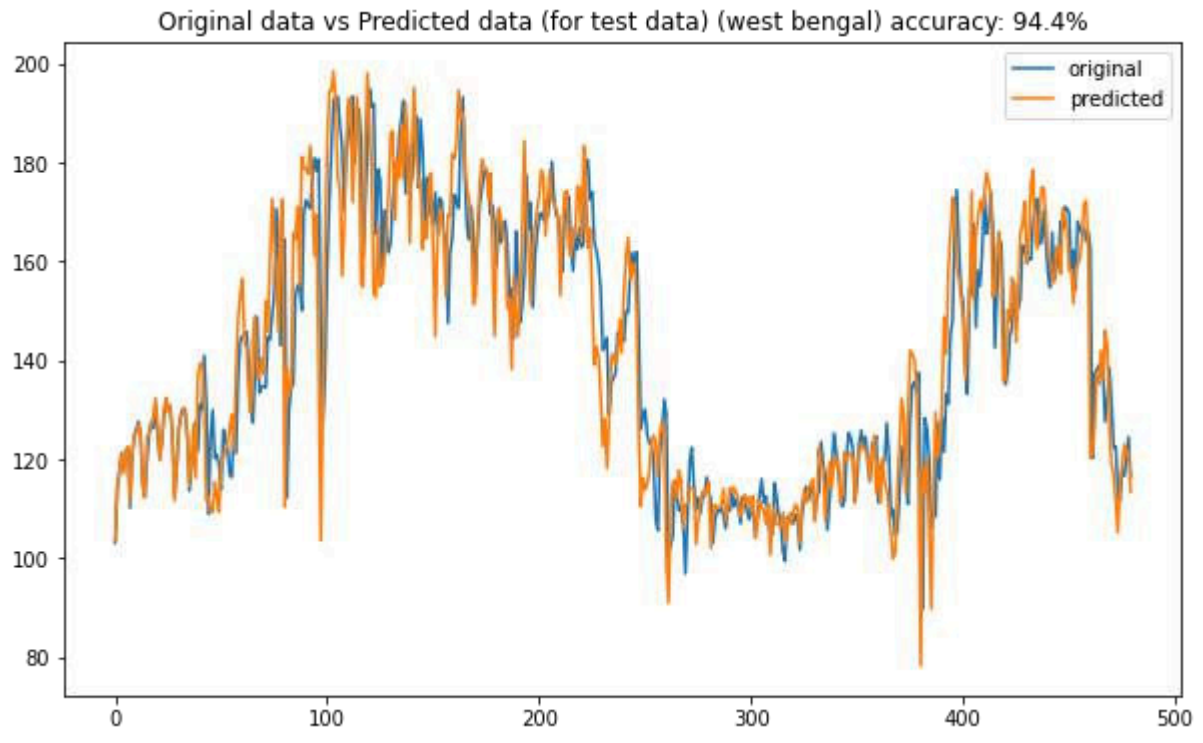
==========


Original data vs Predicted data (for test data) (up) accuracy: 95.1%
original
predicted
450
400
350
300
250
200
0
100
200
300
400
500

==========


Original data vs Predicted data (for test data) (uttarakhand) accuracy: 95.2%
original
predicted
45
40
35
30
25
0
100
200
300
400
500

==========


Original data vs Predicted data (for test data) (west bengal) accuracy: 94.4%
original
predicted
200
180
160
140
120
100
80
0
100
200
300
400
500

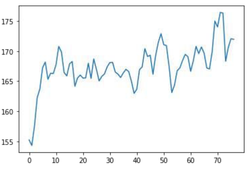
==========

Predicting future

In [28]:
```
from pandas import Series
from datetime import date
from datetime import datetime
from datetime import timedelta
import matplotlib.pyplot as plt
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)


    differenced = difference(new_df,1)
    differenced.head()
def future_pred(col_name,day,month,year):
    from pandas import Series
    from datetime import date
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    d0 = date(year, month, day)
    d1 = date(2020, 11, 14)
    delta = d0 - d1
    no_days = delta.days

    if col_name in ['telangana','andhra pradesh']:
        from keras.models import load_model
        model = load_model('models_with_tuner\\'+col_name+'_model_100.h5')
        model.load_weights('models_with_tuner\\'+col_name+'_model_100.h5')

        data1 = pd.read_csv('pycharm_data.csv')
        data = list(data1[col_name][-39:])
        differenced = difference(data, 1)
        X_test = pd.DataFrame(differenced)
        x_input = np.array(X_test)
        lst_output = []
        i = 0

        while (i < no_days):
            if (len(data) > 39):
                X_test = pd.DataFrame(difference(data[i:], 1))
                x_input = np.array(X_test)
                x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))
                x_input = np.reshape(x_input, (1, x_input.shape[0], 1))
                yhat = model.predict(x_input, verbose=0)
                yhat = yhat + data[-1]
                day = d1 + timedelta(days=i+1)
                day_str = day.strftime("%m/%d/%Y")
                print("{} day output {}".format(day_str, yhat[0][0]))
                lst_output.append(yhat[0][0])
                data.append(yhat[0][0])
                i = i + 1
            else:
                x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))
                x_input = np.reshape(x_input, (1, x_input.shape[0], 1))
                yhat = model.predict(x_input, verbose=0)
                yhat = yhat + data[-1]
                day = d1 + timedelta(days=i+1)
                day_str = day.strftime("%m/%d/%Y")
                print("{} day output {}".format(day_str, yhat[0][0]))
                lst_output.append(yhat[0][0])
                data.append(yhat[0][0])
                i = i + 1
        print('xxxxxxxxxxxxxxx DONE xxxxxxxxxxxxxxxxxxxxx')
        import seaborn as sns
        sns.lineplot(x=[x for x in range(len(lst_output))], y=lst_output)
        plt.show()

    else:
        from keras.models import load_model
        model = load_model('models_with_tuner\\'+col_name+'_model_100.h5')
        model.load_weights('models_with_tuner\\'+col_name+'_model_100.h5')

        data1 = pd.read_csv('pycharm_data.csv')
        data = list(data1[col_name][-49:])
        differenced = difference(data,1)
        X_test = pd.DataFrame(differenced)
        x_input = np.array(X_test)
        lst_output = []
        lst_dates = []
        i = 0
        while (i < no_days):
            if (len(data) > 49):
                X_test = pd.DataFrame(difference(data[i:],1))
                x_input = np.array(X_test)
                # print("{} day input {}".format(i,x_input))
                x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))
                x_input = np.reshape(x_input, (1, x_input.shape[0], 1))
                yhat = model.predict(x_input, verbose=0)
                yhat = yhat+data[-1]
                day = d1 + timedelta(days=i+1)
                day_str = day.strftime("%m/%d/%Y")
                print("{} day output {}".format(day_str, yhat[0][0]))
                lst_dates.append(day_str)
                lst_output.append(yhat[0][0])
                data.append(yhat[0][0])
                i=i+1
            else:
                x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))
                x_input = np.reshape(x_input, (1, x_input.shape[0], 1))
                yhat = model.predict(x_input, verbose=0)
                yhat = yhat+data[-1]
                day = d1 + timedelta(days=i+1)
                day_str = day.strftime("%m/%d/%Y")
                print("{} day output {}".format(day_str, yhat[0][0]))
                lst_dates.append(day_str)
                lst_output.append(yhat[0][0])
                data.append(yhat[0][0])
                i = i + 1
        print(lst_output[-1])
        print('xxxxxxxxxxxxxxx DONE xxxxxxxxxxxxxxxxxxxxx')
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.lineplot(x=[x for x in range(len(lst_output))], y=lst_output)
        plt.show()
    prediction1 = lst_output[-1]
    day_str = d0.strftime("%m/%d/%Y")
    print('On {} the number of mega units used by the state {} is {}'.format(day_str,col_name,predictio
n1))
```

In [29]:
```
future_pred('andhra pradesh',30,1,2021)                # giving date , month , year and the name of the s
tate you want to predict
```

```
11/15/2020 day output 155.24122619628906
11/16/2020 day output 154.32852172851562
11/17/2020 day output 157.55992126464844
11/18/2020 day output 162.25616455078125
11/19/2020 day output 163.72984313964844
11/20/2020 day output 167.22276306152344
11/21/2020 day output 168.17184448242188
11/22/2020 day output 165.3292694091797
11/23/2020 day output 166.33262634277344
11/24/2020 day output 166.27250671338672
11/25/2020 day output 167.7413787841797
11/26/2020 day output 170.77264404296875
11/27/2020 day output 169.86293029785156
11/28/2020 day output 166.43453979492188
11/29/2020 day output 165.88597106933594
11/30/2020 day output 167.85690307617188
12/01/2020 day output 168.2665557861328
12/02/2020 day output 164.15447998046875
12/03/2020 day output 165.5413360595703
12/04/2020 day output 166.02456665039062
12/05/2020 day output 165.50990295410156
12/06/2020 day output 165.55386352539062
12/07/2020 day output 167.99160766601562
12/08/2020 day output 165.46490478515625
12/09/2020 day output 168.6893768310547
12/10/2020 day output 166.976638793453
12/11/2020 day output 165.04026794433594
12/12/2020 day output 165.79513549804688
12/13/2020 day output 166.21470642089844
12/14/2020 day output 167.39442443847656
12/15/2020 day output 168.1018524169922
12/16/2020 day output 168.11734008789062
12/17/2020 day output 166.5160675048828
12/18/2020 day output 166.17767333984375
12/19/2020 day output 165.60745239257812
12/20/2020 day output 166.39430236816406
12/21/2020 day output 166.96958923339844
12/22/2020 day output 166.59054565429688
12/23/2020 day output 164.9849395751953
12/24/2020 day output 162.98086547851562
12/25/2020 day output 163.6806182861328
12/26/2020 day output 166.9229736328125
12/27/2020 day output 167.37469482421875
12/28/2020 day output 170.40286254882812
12/29/2020 day output 169.10284423828125
12/30/2020 day output 169.3097686767578
12/31/2020 day output 166.15823364257812
01/01/2021 day output 169.28211975097656
01/02/2021 day output 171.4698028564453
01/03/2021 day output 172.8902130126953
01/04/2021 day output 171.06948852539062
01/05/2021 day output 170.92066955566406
01/06/2021 day output 167.42138671875
01/07/2021 day output 163.1173553466797
01/08/2021 day output 164.34739685058594
01/09/2021 day output 166.7374725341797
01/10/2021 day output 167.2136993408203
01/11/2021 day output 168.4334259033203
01/12/2021 day output 169.4495086669922
01/13/2021 day output 169.06285095214844
01/14/2021 day output 166.6407470703125
01/15/2021 day output 168.40411376953125
01/16/2021 day output 170.7858123779297
01/17/2021 day output 169.6023101866406
01/18/2021 day output 170.67173767089844
01/19/2021 day output 169.67303466796875
01/20/2021 day output 167.189453125
01/21/2021 day output 167.03623962402344
01/22/2021 day output 169.9117889404297
01/23/2021 day output 174.98692321777344
01/24/2021 day output 174.00701904296875
01/25/2021 day output 176.42709350585938
01/26/2021 day output 176.31639099121094
01/27/2021 day output 168.31446838378906
01/28/2021 day output 170.650146484375
01/29/2021 day output 172.05844116210938
01/30/2021 day output 171.95529174804688
xxxxxxxxxxxxxxx DONE xxxxxxxxxxxxxxxxxxxxx
```




```
on 01/30/2021 the number of mega units used by the state andhra pradesh is 171.95529174804688
```

Creating the required files for deployment (GCP)

- In order to deploy the application to GCP we need to create the below files
  - main.py file which contain the main code to run the application
  - app.yaml
  - .ebextenstions folder. In that folder we need to ceate python.config file
  - A file named Procfile with no extension
  - .ebignore file with no extension
  - requirements.txt
  - templates folder which contain index.html,base.html and results.html
  - static folder which has css files
  - A folder which contains all the models we saved

Below is the code used for creating main.py using flask

In [ ]:
```
# importing the necessary dependencies
from flask import Flask, render_template, request,send_file # importing flask library
import matplotlib.pyplot as plt                             # importing matplotlib as plt
import matplotlib
import io                                                   # importing io to manage the file-related input and
 output operations
import base64
matplotlib.use('Agg')

main = Flask(__name__)                                      # initializing a flask app


@main.route('/',methods=['GET'])                            # route to display the home page
def homePage():
    return render_template("index.html")

@main.route('/predict',methods=['POST','GET'])              # route to show the predictions in a web UI
def index():
    if request.method == 'POST':
        try:
            col_name=str(request.form['State_name'])#  reading the inputs given by the user
            Year = int(request.form['Year'])
            Day = int(request.form['Day'])
            Month = int(request.form['Month'])

            from datetime import datetime                   # importing all the libraries required
            from datetime import timedelta
            from pandas import Series
            from datetime import date
            import pandas as pd
            import numpy as np

            def difference(dataset, interval=1):            # creating a differencing transform function
                diff = list()
                for i in range(interval, len(dataset)):
                    value = dataset[i] - dataset[i - interval]
                    diff.append(value)
                return Series(diff)

                differenced = difference(new_df, 1) # applying differencing transform to the data
                differenced.head()

            d0 = date(Year, Month, Day)
            d1 = date(2020, 11, 14)
            delta = d0 - d1
            no_days = delta.days                            # finding the number of days we need to predict
            from keras.models import load_model
            filename1 =  'models_with_tuner/'+col_name+'_model_100.h5'

            model = load_model( filepath=filename1 )# loading the model file from the storage
            model.load_weights( filepath=filename1 )

            if col_name in ['telangana', 'andhra pradesh']:
                data1 = pd.read_csv('pycharm_data.csv')
                data = list(data1[col_name][-39:])  # We need to use only 39 values as input incrementa
lly

                differenced = difference(data, 1)   # difference transform the data
                X_test = pd.DataFrame(differenced)
                x_input = np.array(X_test)          # converting the data into an array
                lst_output = []                     # list to store all the outputs
                i = 0

                while (i < no_days):
                    if (len(data) > 39):
                        X_test = pd.DataFrame(difference(data[i:], 1)) # the input increments for every
 iteration and applying

                                                                        # differencing transform
                                                                        # to predict the next day output
                        x_input = np.array(X_test)                      # converting the input to an arr
ay
                        x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))   # resh
aping the input
                        x_input = np.reshape(x_input, (1, x_input.shape[0], 1)) # reshaping the input
                        yhat = model.predict(x_input, verbose=0)        # predicting the output
                        yhat = yhat + data[-1]                          # inverse differencing transform

                        lst_output.append(yhat[0][0])                   # appending to the lst_output
                        data.append(yhat[0][0])                         # appending the value to the dat
a
                        i = i + 1

                    else:
                        x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))# reshapi
ng the input

                        x_input = np.reshape(x_input, (1, x_input.shape[0], 1))# reshaping the input
                        yhat = model.predict(x_input, verbose=0)        # predicting the output
                        yhat = yhat + data[-1]                          # inverse differencing transform

                        lst_output.append(yhat[0][0])                   # appending to the lst_output
                        data.append(yhat[0][0])                         # appending the value to the dat
a
                        i = i + 1

                import matplotlib                                       # importing required libraries f
or ploting
                matplotlib.use('Agg')               # we are going to plot the output from 14/11/2020 to
 the data given as input
                import matplotlib.pyplot as plt
                import seaborn as sns
                import io
                import base64
                img = io.BytesIO()                  # our graph is kept as bytes in an in-memory buffer whe
n we use the io.BytesIO()
                sns.set_style("dark")

                y = lst_output                      # getting the output
                x = [x for x in range(len(lst_output))]

                plt.plot(x, y)                      # plotting the graph
                plt.xlabel('Day count after 14/11/2020')
                plt.ylabel('MegaUnits')
                plt.savefig(img, format='png')
                plt.close()
                img.seek(0)
                                                    # converting img to base64 string and decoding it to ut
f8
                plot_url = base64.b64encode(img.getvalue()).decode('utf8')

            else:

                data1 = pd.read_csv('pycharm_data.csv')
                data = list(data1[col_name][-49:]) # We need to use only 49 values as input incremental
ly
                differenced = difference(data, 1)  # difference transform the data
                X_test = pd.DataFrame(differenced)
                x_input = np.array(X_test)          # converting the data into an array
                lst_output = []                     # list to store all the outputs
                i = 0
                while (i < no_days):
                    if (len(data) > 49):
                        X_test = pd.DataFrame(difference(data[i:], 1))# the input increments for every
 iteration and applying

                                                                        # differencing transform
                                                                        # to predict the next day output
                        x_input = np.array(X_test)                      # converting the input to an arra
y
                        x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))# reshapi
ng the input
                        x_input = np.reshape(x_input, (1, x_input.shape[0], 1))# reshaping the input
                        yhat = model.predict(x_input, verbose=0)        # predicting the output
                        yhat = yhat + data[-1]                          # inverse differencing transform
                        lst_output.append(yhat[0][0])                   # appending to the lst_output
                        data.append(yhat[0][0])                         # appending the value to the data
                        i = i + 1
                    else:
                        x_input = np.reshape(x_input, (x_input.shape[0], x_input.shape[1], 1))# reshapi
ng the input
                        x_input = np.reshape(x_input, (1, x_input.shape[0], 1))# reshaping the input
                        yhat = model.predict(x_input, verbose=0)      # predicting the output
                        yhat = yhat + data[-1]                        # inverse differencing transform
                        lst_output.append(yhat[0][0])                 # appending to the lst_output
                        data.append(yhat[0][0])                       # appending the value to the data
                        i = i + 1
                import matplotlib
                matplotlib.use('Agg')         # we are going to plot the output from 14/11/2020 to the d
ata given as input
                import matplotlib.pyplot as plt
                import seaborn as sns
                import io
                import base64

                img = io.BytesIO()            # our graph is kept as bytes in an in-memory buffer when we
 use the io.BytesIO()
                sns.set_style("dark")  # E.G.

                y = lst_output
                x = [x for x in range(len(lst_output))]

                plt.plot(x, y)
                plt.xlabel('Day count after 14/11/2020')
                plt.ylabel('MegaUnits')
                plt.savefig(img, format='png')
                plt.close()
                img.seek(0)
                                                    # converting img to base64 string and decoding it to utf8
                plot_url = base64.b64encode(img.getvalue()).decode('utf8')
                print(plot_url)

            prediction1 = lst_output[-1]
                                                    # rendering the results on result.html
            return render_template('results.html',prediction=prediction1,State_name=col_name,Date=d0.st
rftime("%m/%d/%Y"),plot_url=plot_url)



        except Exception as e:
            print('The Exception message is: ',e)
            return 'something is wrong'
    else:
        return render_template('index.html')



if __name__ == "__main__":
    #app.run(host='127.0.0.1', port=8001, debug=True)
    main.run(debug=True)  # running the app
```

- The above file is saved as main.py file. you can check that file in my repository
- I had created index.html,base.html and results.html using https://codepen.io/.
- You can look at the html files in templates folder
- All the remaining files are created which are mentioned in the above markdown cell , you can check those files in my repository
- The below is link which will redirect you the deployed model website created using Google cloud platform
  https://forecast-1111.df.r.appspot.com/

Thanking you for looking at my project . For any suggestions or queries mail me at nitishkumar2902@gmail.com