```
from google.colab import files
import io
import pandas as pd
data = files.upload()
```

Choose Files intern_data_spm.csv

- **intern_data_spm.csv**(text/csv) - 1046715 bytes, last modified: 6/6/2023 - 100% done

```
Saving intern_data_spm.csv to intern_data_spm (1).csv
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
%matplotlib inline
```

```
df=pd.read_csv(io.StringIO(data['intern_data_spm.csv'].decode('utf-8')))
```

```
df.shape
```

```
(5000, 16)
```

```
df.head()
```

	CLOCK	X1	X2	X3	X4	X5	X6	X7	X8	X9	
0	2022-12-18 14:50:00	48.65105	38.97555	35.33245	36.00630	63.58980	10.96650	145.15495	384.60800	919.95	2
1	2022-12-31	46.30760	36.69085	32.61530	34.56650	62.49190	10.48560	153.96970	389.11030	880.05	1
2	12-23	48.56460	38.86065	34.98840	35.50790	65.73585	19.72395	161.72370	383.47905	712.95	1

Saving...

```
df.tail()
```

	CLOCK	X1	X2	X3	X4	X5	X6	X7	X
4995	2022-12-29 17:50:00	48.970250	38.739500	35.964150	36.376850	65.717900	8.880750	152.399100	408.22870
4996	2022-12-29 00:40:00	47.482000	38.186750	34.010450	35.813450	64.686100	17.171500	134.963450	368.41175
4997	2022-12-19	47.177000	38.093600	34.074500	35.629350	63.249700	11.424150	135.836050	378.53305

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   CLOCK   5000 non-null   object 
 1   X1      5000 non-null   float64
 2   X2      5000 non-null   float64
 3   X3      5000 non-null   float64
 4   X4      5000 non-null   float64
 5   X5      5000 non-null   float64
 6   X6      5000 non-null   float64
 7   X7      5000 non-null   float64
 8   X8      5000 non-null   float64
 9   X9      5000 non-null   float64
 10  X10     5000 non-null   float64
 11  X11     5000 non-null   float64
 12  X12     5000 non-null   float64
 13  X13     5000 non-null   float64
 14  X14     5000 non-null   float64
 15  output  5000 non-null   float64
dtypes: float64(15), object(1)
memory usage: 625.1+ KB
```

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

```
CLOCK     0
X1        0
X2        0
X3        0
```

```
X4        0
X5        0
X6        0
X7        0
X8        0
X9        0
X10       0
X11       0
X12       0
X13       0
X14       0
output    0
dtype: int64
```

```
[features for features in df.columns if df[features].isnull().sum()>0]
```

```
[]
```

NO NULL VALUES SO THERE ARE NO MISSING VALUES

```
df.dtypes
```

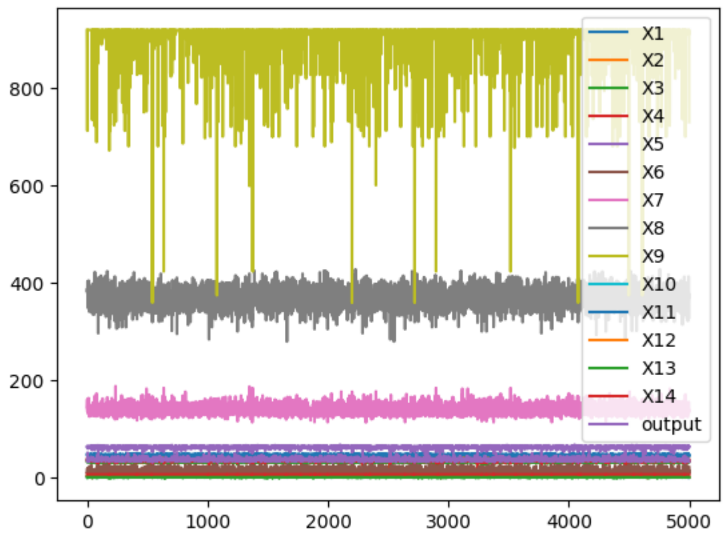
```
CLOCK      object
X1        float64
X2        float64
X3        float64
X4        float64
X5        float64
X6        float64
X7        float64
X8        float64
X9        float64
X10       float64
X11       float64
[illegible]
```

Saving...

```
output    float64
dtype: object
```

```
df.plot()
```

```
<Axes: >
```

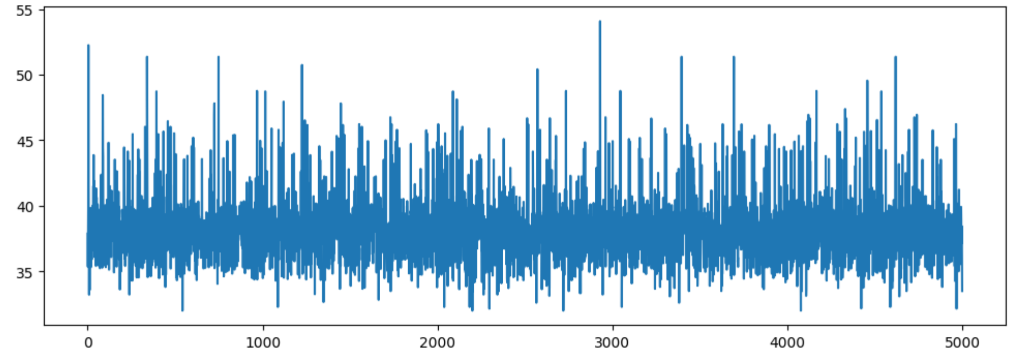



```
df.sort_values(['CLOCK','X1','X2','X3','X4','X5','X6','X7','X8','X9','X10','X11','X12','X13','X14','output'])
```

	CLOCK	X1	X2	X3	X4	X5	X6	X7	X8	X9
1703	2022-12-10 06:30:00	49.52130	39.18235	35.92655	36.67485	63.90860	2.63820	142.41390	387.24310	920.00
3094	2022-12-10 06:50:00	49.55760	39.07140	35.79370	36.81315	62.28685	2.79050	146.99295	402.32190	920.05

```
df['output'].plot(figsize=(12,4))
```

```
<Axes: >
```



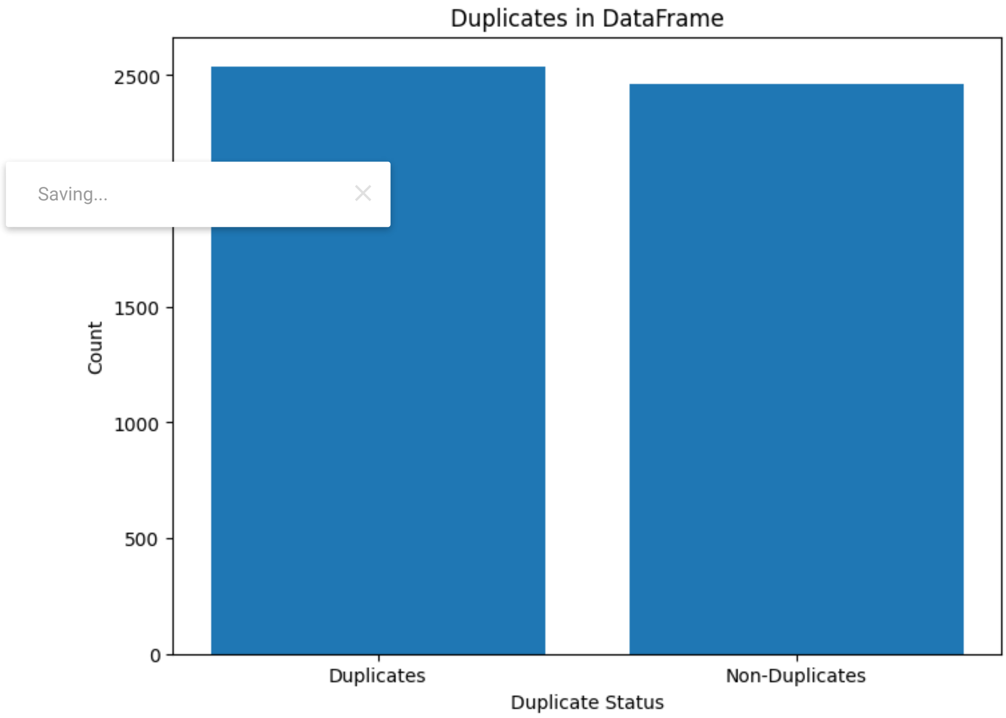

```
cols=df.columns
Saving...
'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
output']].duplicated()
duplicated_rows=df[duplicates]
print(duplicated_rows)
print(duplicates)
```

```
Index(['CLOCK', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
       'X11', 'X12', 'X13', 'X14', 'output'],
      dtype='object')
                    CLOCK         X1         X2         X3         X4  \
52    2022-12-13 06:00:00  48.111000  39.059250  34.437700  37.116600
107   2022-12-17 01:20:00  46.827900  37.523900  33.374050  35.422150
108   2022-12-19 17:00:00  47.433500  38.433700  34.218200  35.896800
118   2022-12-15 10:00:00  47.569650  38.379900  34.173000  36.184050
145   2022-12-13 06:00:00  48.111000  39.059250  34.437700  37.116600
...                   ...        ...        ...        ...        ...
4994  2022-12-25 07:40:00  46.379550  36.759050  33.223450  34.295950
4995  2022-12-29 17:50:00  48.970250  38.739500  35.964150  36.376850
4996  2022-12-29 00:40:00  47.482000  38.186750  34.010450  35.813450
4997  2022-12-19 03:20:00  47.177000  38.093600  34.074500  35.629350
4998  2022-12-12 13:50:00  48.468167  39.066556  34.451111  36.946222

             X5         X6          X7          X8          X9      X10  \
52    62.575800  17.052400  132.687400  358.084200  919.950000  2.17830
107   65.338500  10.199800  136.478150  369.819700  919.950000  1.93795
108   64.227100   8.185350  136.866000  363.454600  920.000000  2.05755
118   66.840300   8.052250  138.086150  379.072150  919.950000  1.80095
145   62.575800  17.052400  132.687400  358.084200  919.950000  2.17830
...         ...        ...         ...         ...         ...      ...
4994  64.590650   6.571300  156.399550  413.743950  920.000000  1.68955
4995  65.717900   8.880750  152.399100  408.228700  909.900000  1.79045
4996  64.686100  17.171500  134.963450  368.411750  919.950000  1.89235
4997  63.249700  11.424150  135.836050  378.533050  920.050000  2.04950
4998  62.825278  10.619111  135.960722  368.527111  919.944444  2.13950

           X11       X12       X13       X14     output
52    2.829300  3.804500  3.799000  7.694000  38.972250
107   2.679350  2.913000  2.923500  7.693500  38.436000
108   2.783350  3.659000  3.665000  7.693300  39.025950
118   2.540200  3.097000  3.095500  7.693850  36.270600
145   2.829300  3.804500  3.799000  7.694000  38.972250
...        ...       ...       ...       ...        ...
4994  2.508800  2.712500  2.703000  7.693900  38.868850
4995  2.555800  2.760500  2.788500  7.693900  37.104800
4996  2.610600  2.799000  2.839500  7.693900  38.265500
4997  2.748300  3.621500  3.631500  7.693300  38.432150
4998  2.801833  3.478889  3.467222  7.693833  35.841333

[2539 rows x 16 columns]
0       False
1       False
```

```
2       False
3       False
4       False
        ...
4995     True
4996     True
4997     True
4998     True
4999    False
Length: 5000, dtype: bool
```

CHEKING DUPLICATES

```
duplicates = df.duplicated()
df1 = df[duplicates == True]
df2 = df[duplicates == False]
x = ['Duplicates', 'Non-Duplicates']
y = [len(df1), len(df2)]

# Create a bar plot
plt.figure(figsize=(8, 6))
plt.bar(x, y)
plt.xlabel('Duplicate Status')
plt.ylabel('Count')
plt.title('Duplicates in DataFrame')
plt.show()
```




```
#REMOVING DUPLICATES
df=df.drop_duplicates()
print(df)
```

```
                   CLOCK        X1        X2        X3        X4        X5  \
0    2022-12-18 14:50:00  48.65105  38.97555  35.33245  36.00630  63.58980
1    2022-12-31 12:10:00  46.30760  36.69085  32.61530  34.56650  62.49190
2    2022-12-23 18:10:00  48.56460  38.86065  34.98840  35.50790  65.73585
3    2022-12-19 05:30:00  47.59175  38.35315  34.27105  35.68830  64.44080
4    2022-12-10 14:00:00  49.20060  38.82830  35.15390  36.69245  63.76605
...                  ...       ...       ...       ...       ...       ...
4987 2022-12-21 17:10:00  46.74265  37.91030  33.77445  35.46620  62.01295
4988 2022-12-20 13:30:00  46.81005  37.49205  34.34830  35.62205  62.44480
4989 2022-12-24 04:50:00  46.55275  38.00450  32.84530  35.70645  64.54105
4990 2022-12-13 22:20:00  49.12230  39.84660  35.99265  37.92645  65.72715
4999 2022-12-23 13:30:00  44.97165  35.83190  33.48865  34.23095  64.12685

           X6         X7         X8      X9      X10      X11     X12  \
0    10.96650  145.15495  384.60800  919.95  2.05030  2.66385  3.5270
1    10.48560  153.96970  389.11030  880.05  1.59045  1.83155  1.9555
2    19.72395  161.72370  383.47905  712.95  1.06160  1.54015  3.5815
3    11.15725  136.59115  379.70615  920.05  2.04615  2.74185  3.6125
4     4.46650  149.19130  403.56375  919.95  2.18880  2.68110  3.7480
...       ...        ...        ...     ...      ...      ...     ...
4987  7.81940  132.32080  356.39805  919.95  2.09040  2.86435  3.7385
4988 11.28075  141.45355  322.63470  920.00  1.75145  2.33580  3.2625
```

```
4989  10.52350  135.02065  357.26920  919.90  1.99050  2.47880  3.5585
4990  15.68075  138.12390  365.56105  920.00  2.05900  2.87455  2.9855
4999  22.19980  139.35930  329.82085  731.00  0.99180  1.25110  1.8820

         X13      X14    output
0     3.5270  7.69330  37.87175
1     1.9835  7.69370  35.35060
2     0.0000  7.69370  37.73035
3     3.5880  7.69310  39.72865
4     3.7475  7.69395  52.24655
...      ...      ...       ...
4987  3.7295  7.69375  39.84960
4988  3.2635  7.69350  38.20440
4989  3.5740  7.69355  39.89885
4990  2.9465  7.69375  38.44980
4999  1.8485  7.69380  33.49360

[2461 rows x 16 columns]
```

```
df.shape
```

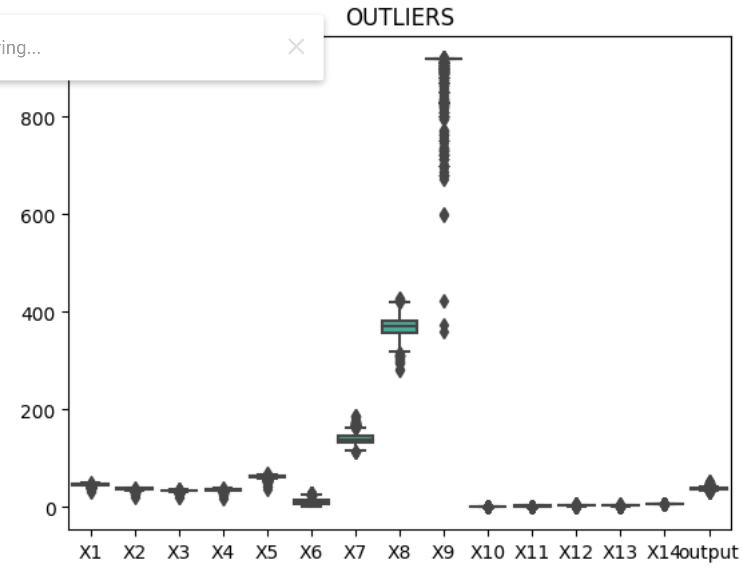
```
(2461, 16)
```

OUTLIER DETECTION by using z score

```
numerical_columns = ['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'output']
dataf=df[numerical_columns]
sns.boxplot(data=dataf)
plt.show
plt.title('OUTLIERS')
```

```
Text(0.5, 1.0, 'OUTLIERS')
```

Saving...




```
numerical_columns = ['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'output']
dataf=df[numerical_columns]
sns.scatterplot(data=dataf)
plt.show
plt.title('OUTLIERS')
```

```
Text(0.5, 1.0, 'OUTLIERS')
```


```python
def find_outliers(data, threshold=3):   #defining a funciton to check wheather the dataset have outlier or not

    mean = data.mean()
    std = data.std()
    z_scores = (data - mean) / std
    outliers = data[np.abs(z_scores) > threshold]

    return outliers
numerical_cols = ['CLOCK', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'output']
numeric_data = df[numerical_cols]


outliers = find_outliers(numeric_data)
print(outliers)
```

```
      CLOCK  X1  X2  X3  X4  X5  X6  X7  X8      X9     X10      X11  X12  X13  \
0       NaN NaN NaN NaN NaN NaN NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
1       NaN NaN NaN NaN NaN NaN NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
2       NaN NaN NaN NaN NaN NaN NaN NaN NaN  712.95  1.0616  1.54015  NaN  0.0
3       NaN NaN NaN NaN NaN NaN NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
4       NaN NaN NaN NaN NaN NaN NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
...     ...  ..  ..  ..  ..  ..  ..  ..  ..     ...     ...      ...  ...  ...
4987    NaN NaN NaN NaN NaN NaN NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
4988    NaN NaN NaN NaN NaN NaN NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
    Saving...                   NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
                                NaN NaN NaN     NaN     NaN      NaN  NaN  NaN
                                NaN NaN NaN  731.00  0.9918  1.25110  NaN  NaN

      X14    output
0     NaN       NaN
1     NaN       NaN
2     NaN       NaN
3     NaN       NaN
4     NaN  52.24655
...   ...       ...
4987  NaN       NaN
4988  NaN       NaN
4989  NaN       NaN
4990  NaN       NaN
4999  NaN       NaN

[2461 rows x 16 columns]
<ipython-input-20-51f23597b19d>:3: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future ve
  mean = data.mean()
<ipython-input-20-51f23597b19d>:4: FutureWarning: The default value of numeric_only in DataFrame.std is deprecated. In a future ver
  std = data.std()
```

```python
sns.scatterplot(data=outliers)
plt.title('OUTLIERS DETECTION USING Z-SCORE')
plt.legend()
plt.show()
```

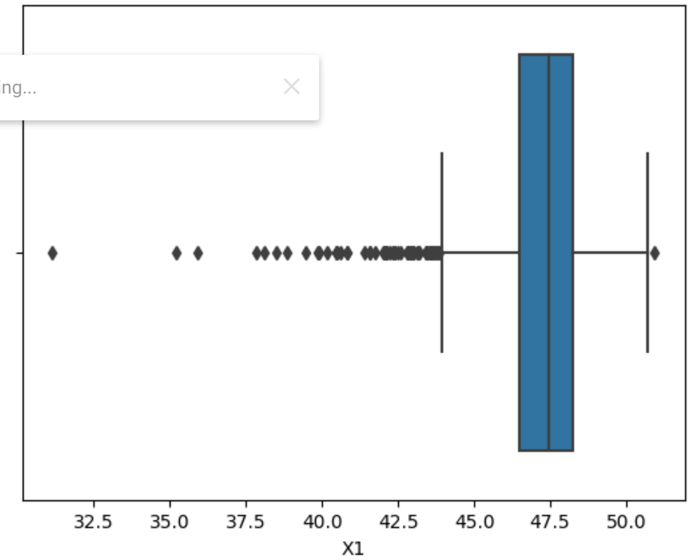

```
df.describe()
```

	X1	X2	X3	X4	X5	X6	X7	
count	2461.000000	2461.000000	2461.000000	2461.000000	2461.000000	2461.000000	2461.000000	246
mean	47.253695	38.028817	33.910237	35.610200	63.473492	11.900316	139.989673	36
std	1.566158	1.365561	1.412881	1.285951	2.218384	5.644267	9.797301	1
min	31.142350	21.108950	20.374150	18.402300	35.847700	2.638200	114.054950	28
25%	46.488500	37.420200	33.135550	34.958650	62.674750	8.118750	133.079150	35
50%	47.467450	38.215800	34.064750	35.754150	63.784650	10.385750	138.226850	37
75%	48.248800	38.846333	34.885500	36.344500	64.767450	14.785750	145.166050	38
max	50.932850	41.017300	37.655100	38.548050	68.600700	27.933500	188.816050	42

OUTLIER REMOVAL BY IQR METHOD

```
sns.boxplot(data=df, x='X1')
```

```
<Axes: xlabel='X1'>
```

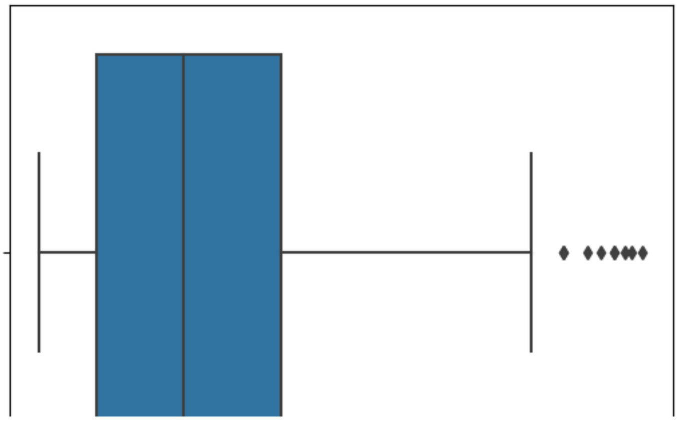



```
df['X1'].skew()
percentile25=df['X1'].quantile(0.25)
percentile75=df['X1'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X1']<upper_limit)&(df['X1']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
175
```

```
sns.boxplot(data=newdf, x='X1')
```

```
<Axes: xlabel='X1'>
```

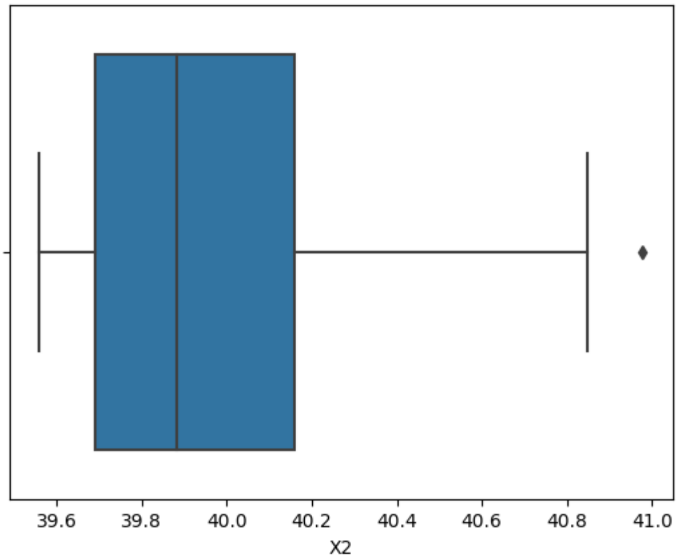


```
df['X2'].skew()
percentile25=df['X2'].quantile(0.25)
percentile75=df['X2'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X2']<upper_limit)&(df['X2']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
198
```

Saving...

```
<Axes: xlabel='X2'>
```




```
sns.boxplot(data=df, x='X3')
```

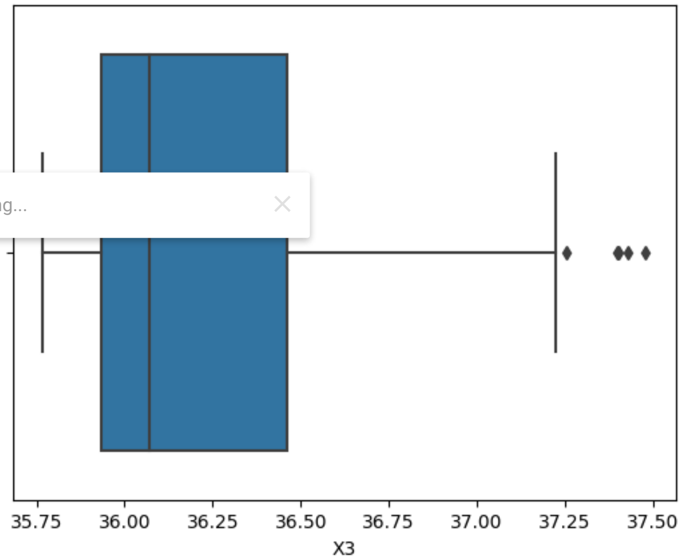
```
<Axes: xlabel='X3'>
```

```
df['X3'].skew()
percentile25=df['X3'].quantile(0.25)
percentile75=df['X3'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X3']<upper_limit)&(df['X3']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
180
```

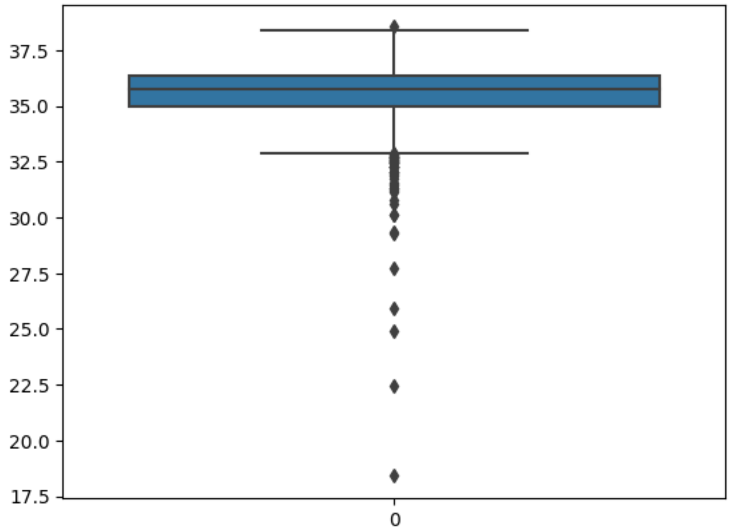
```
sns.boxplot(data=newdf, x='X3')
```

```
<Axes: xlabel='X3'>
```




```
sns.boxplot(df['X4'])
```

```
<Axes: >
```



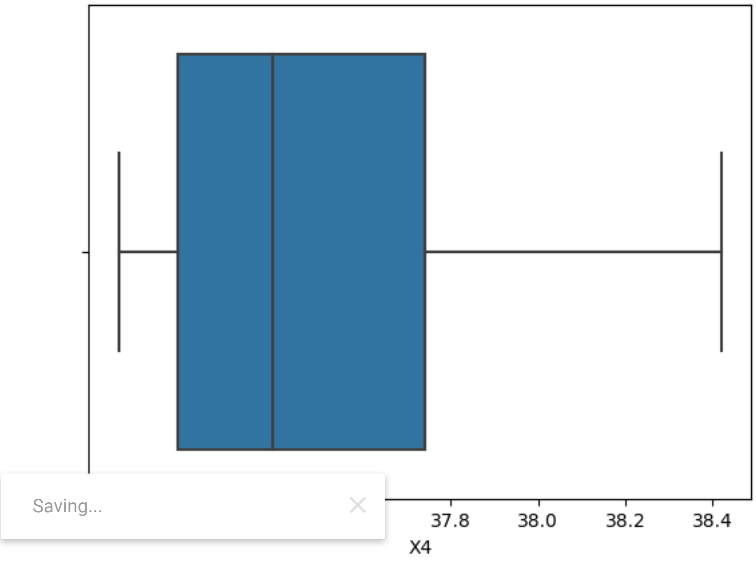

```
df['X4'].skew()
percentile25=df['X4'].quantile(0.25)
percentile75=df['X4'].quantile(0.75)
iqr=percentile75-percentile25
```

```
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X4']<upper_limit)&(df['X4']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
243
```

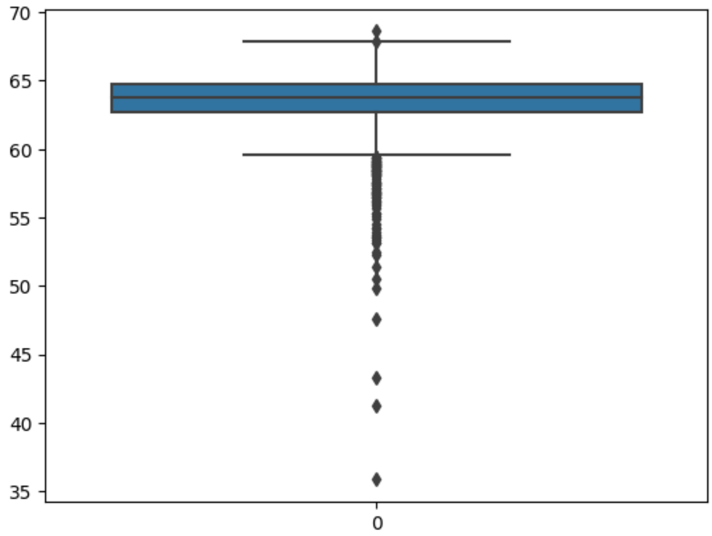
```
sns.boxplot(data=newdf, x='X4')
```

```
<Axes: xlabel='X4'>
```




```
sns.boxplot(df['X5'])
```

```
<Axes: >
```

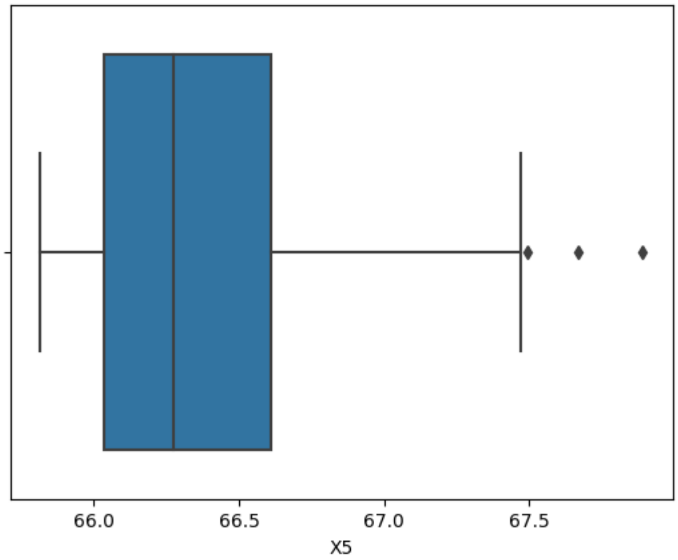



```
df['X5'].skew()
percentile25=df['X5'].quantile(0.25)
percentile75=df['X5'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X5']<upper_limit)&(df['X5']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
197
```

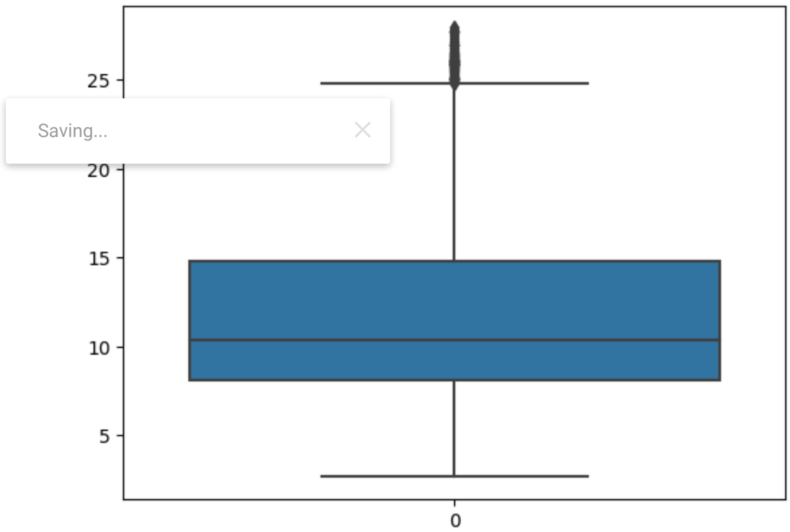
```
sns.boxplot(data=newdf, x='X5')
```

```
<Axes: xlabel='X5'>
```




```
sns.boxplot(df['X6'])
```

```
<Axes: >
```

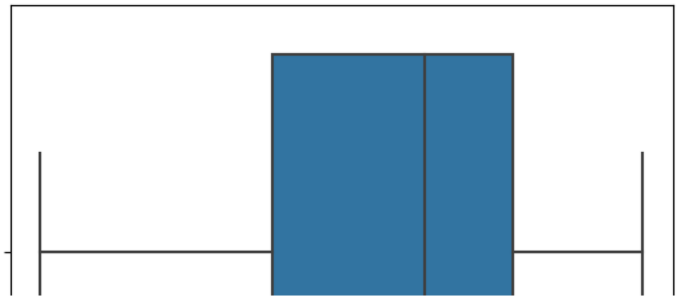



```
df['X6'].skew()
percentile25=df['X6'].quantile(0.25)
percentile75=df['X6'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X6']<upper_limit)&(df['X6']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
227
```

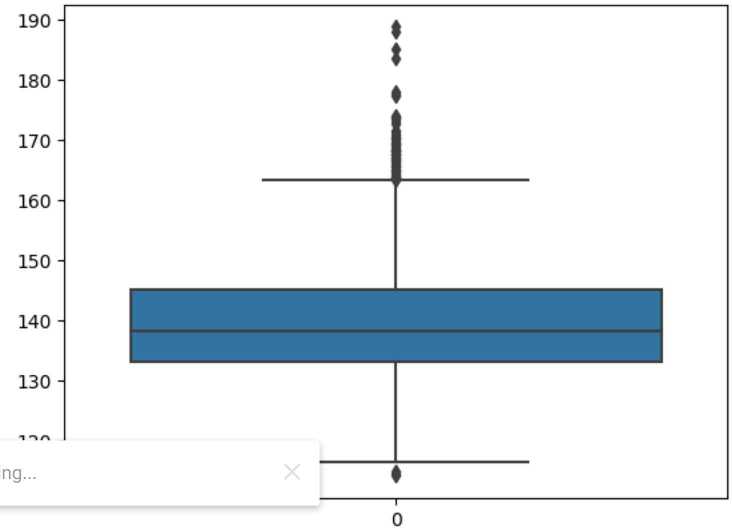
```
sns.boxplot(data=newdf, x='X6')
```

```
<Axes: xlabel='X6'>
```



```
sns.boxplot(df['X7'])
```

```
<Axes: >
```

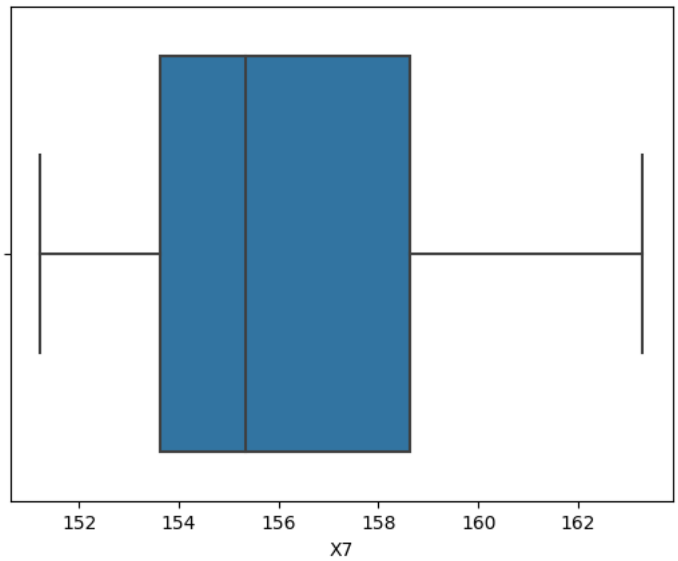



```
df['X1'].skew()
percentile25=df['X7'].quantile(0.25)
percentile75=df['X7'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X7']<upper_limit)&(df['X7']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
266
```

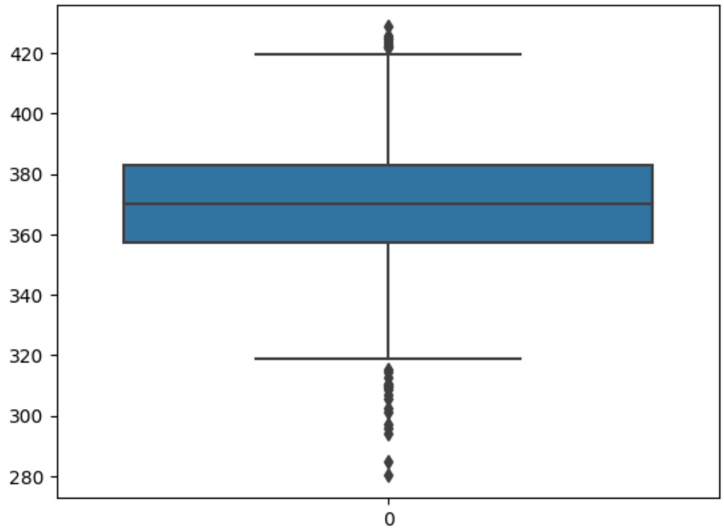
```
sns.boxplot(data=newdf, x='X7')
```

```
<Axes: xlabel='X7'>
```



```
sns.boxplot(df['X8'])
```

```
<Axes: >
```



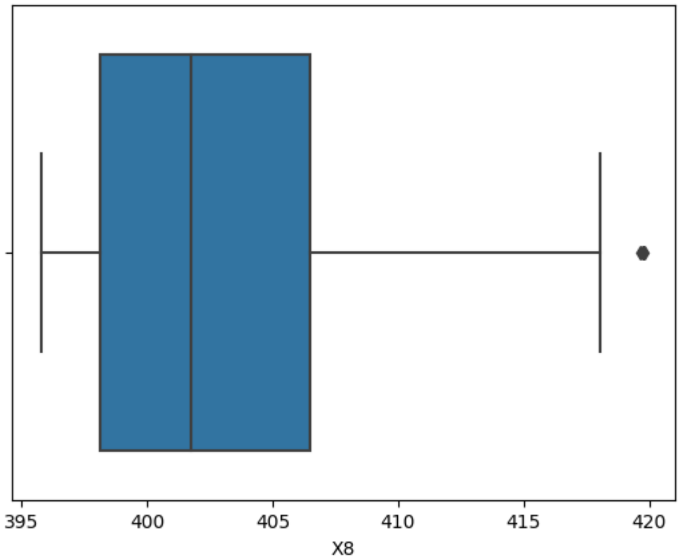

```
df['X8'].skew()
percentile25=df['X8'].quantile(0.25)
percentile75=df['X8'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X8']<upper_limit)&(df['X8']>lower_limit)]
```

Saving...

```
2461
234
```

```
sns.boxplot(data=newdf, x='X8')
```

```
<Axes: xlabel='X8'>
```




```
sns.boxplot(data=df, x='X9')
```

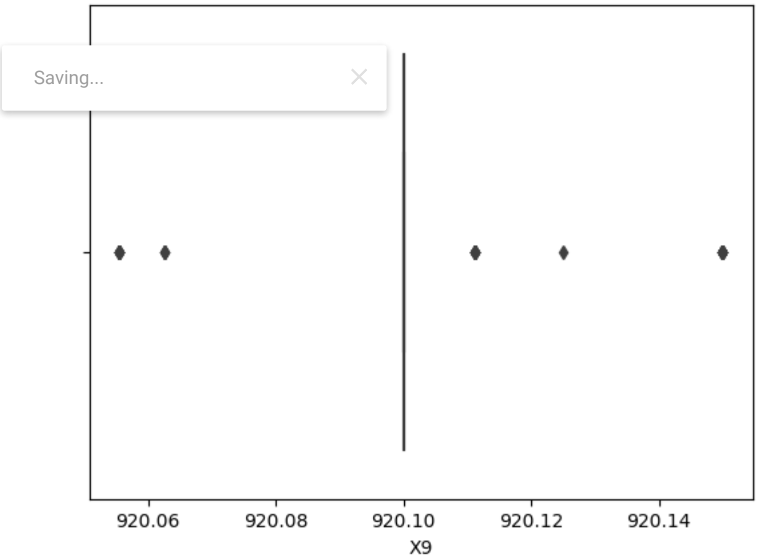
```
<Axes: xlabel='X9'>
```

```
df['X9'].skew()
percentile25=df['X9'].quantile(0.25)
percentile75=df['X9'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X9']<upper_limit)&(df['X9']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
189
```

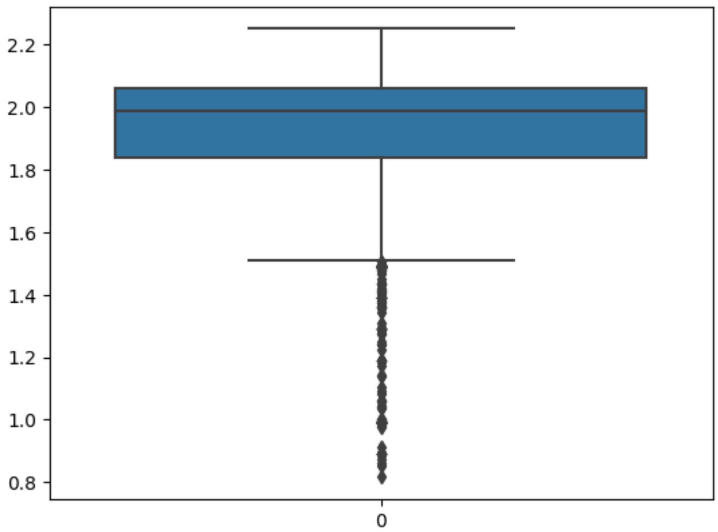
```
sns.boxplot(data=newdf, x='X9')
```

```
<Axes: xlabel='X9'>
```




```
sns.boxplot(df['X10'])
```

```
<Axes: >
```

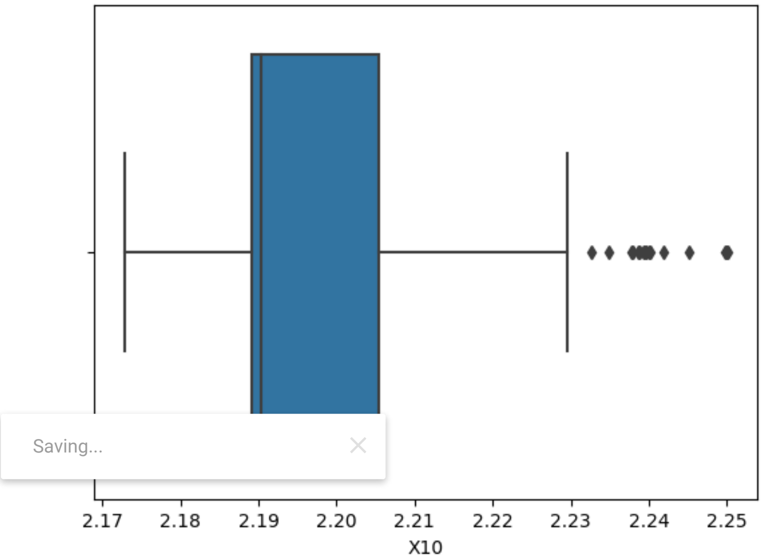


```
df['X10'].skew()
percentile25=df['X10'].quantile(0.25)
percentile75=df['X10'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X10']<upper_limit)&(df['X10']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
150
```

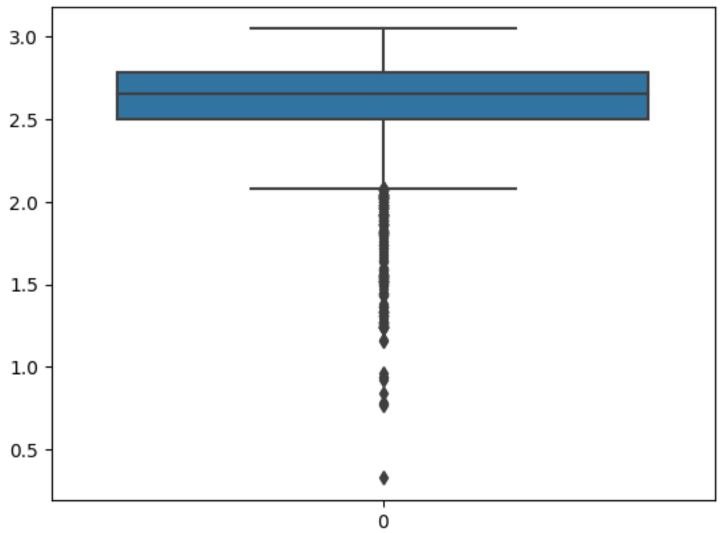
```
sns.boxplot(data=newdf, x='X10')
```

```
<Axes: xlabel='X10'>
```




```
sns.boxplot(df['X11'])
```

```
<Axes: >
```

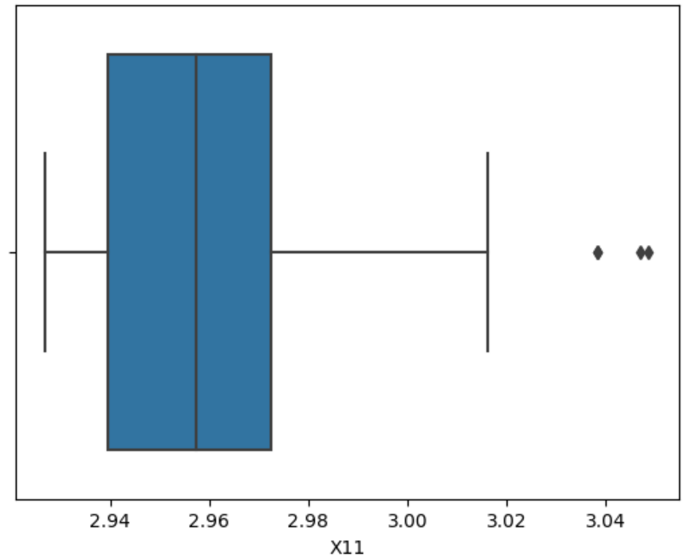



```
df['X11'].skew()
percentile25=df['X11'].quantile(0.25)
percentile75=df['X11'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X11']<upper_limit)&(df['X11']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
108
```

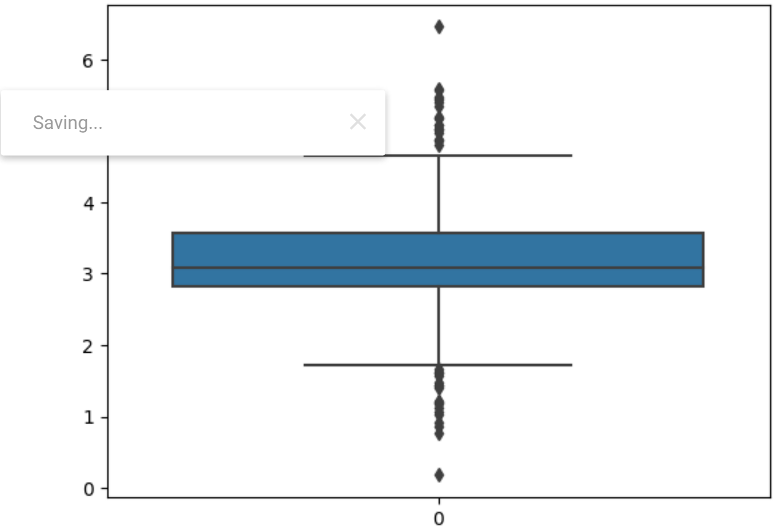
```
sns.boxplot(data=newdf, x='X11')
```

```
<Axes: xlabel='X11'>
```




```
sns.boxplot(df['X12'])
```

```
<Axes: >
```

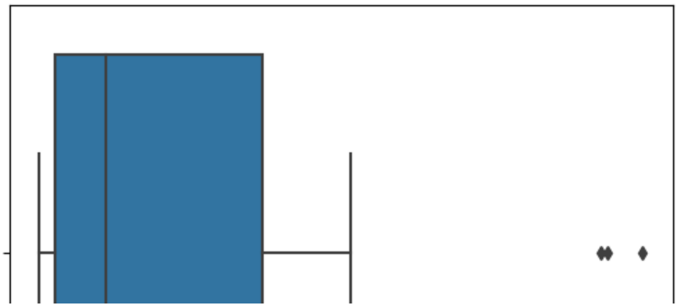



```
df['X12'].skew()
percentile25=df['X12'].quantile(0.25)
percentile75=df['X12'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X12']<upper_limit)&(df['X12']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
23
```

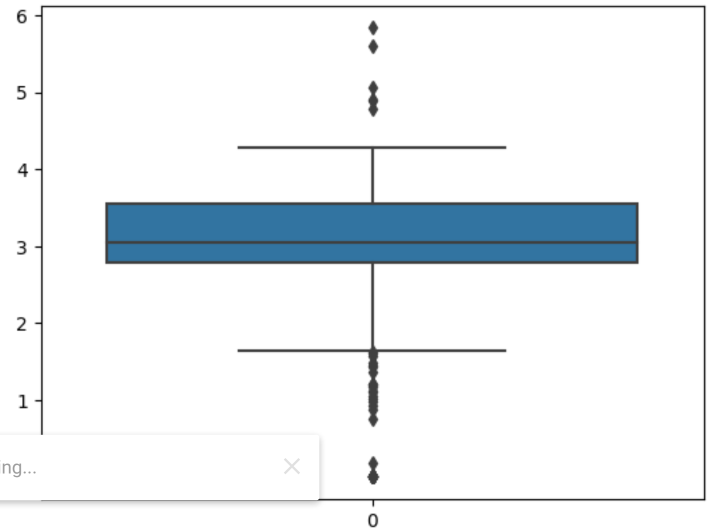
```
sns.boxplot(data=newdf, x='X12')
```

```
<Axes: xlabel='X12'>
```



```
sns.boxplot(df['X13'])
```

```
<Axes: >
```

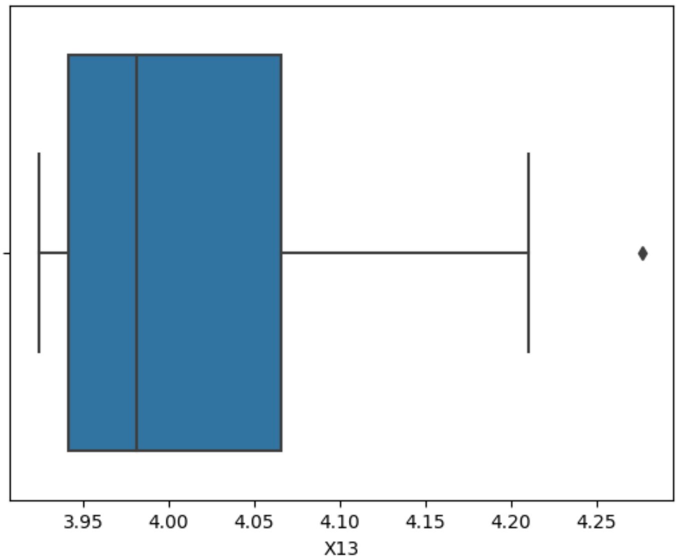



```
df['X13'].skew()
percentile25=df['X13'].quantile(0.25)
percentile75=df['X13'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X13']<upper_limit)&(df['X13']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
21
```

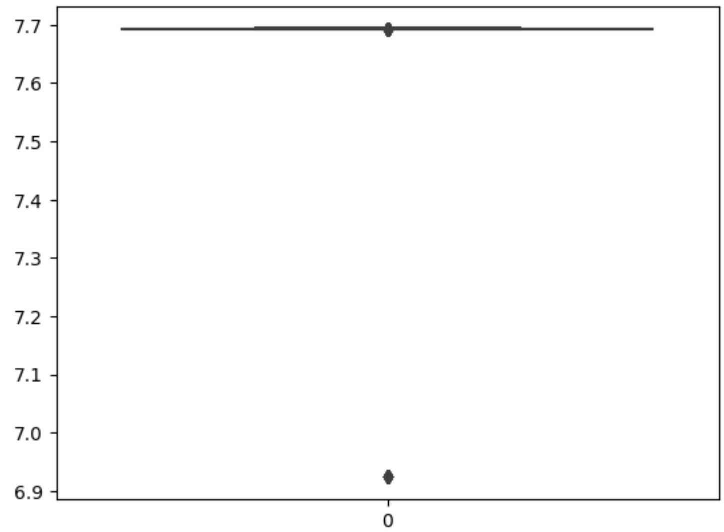
```
sns.boxplot(data=newdf, x='X13')
```

```
<Axes: xlabel='X13'>
```



```
sns.boxplot(df['X14'])
```

```
<Axes: >
```



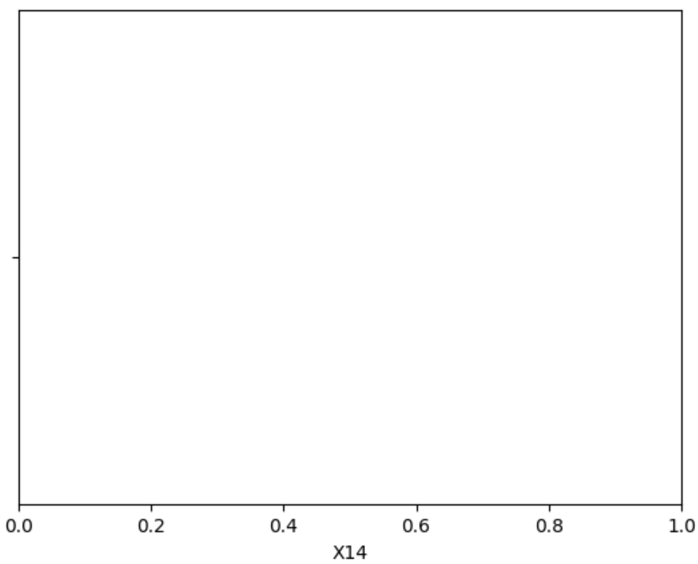

```
df['X14'].skew()
percentile25=df['X14'].quantile(0.25)
percentile75=df['X14'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['X14']<upper_limit)&(df['X14']>lower_limit)]
```

Saving...

```
2461
0
```

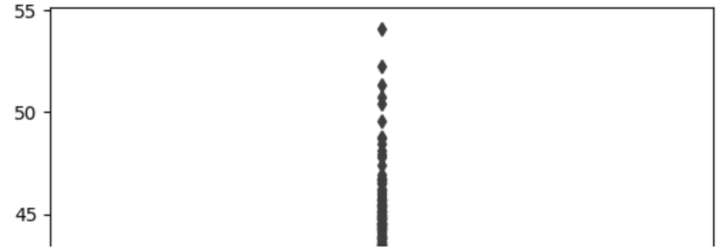
```
sns.boxplot(data=newdf, x='X14')
```

```
<Axes: xlabel='X14'>
```




```
sns.boxplot(df['output'])
```

```
<Axes: >
```

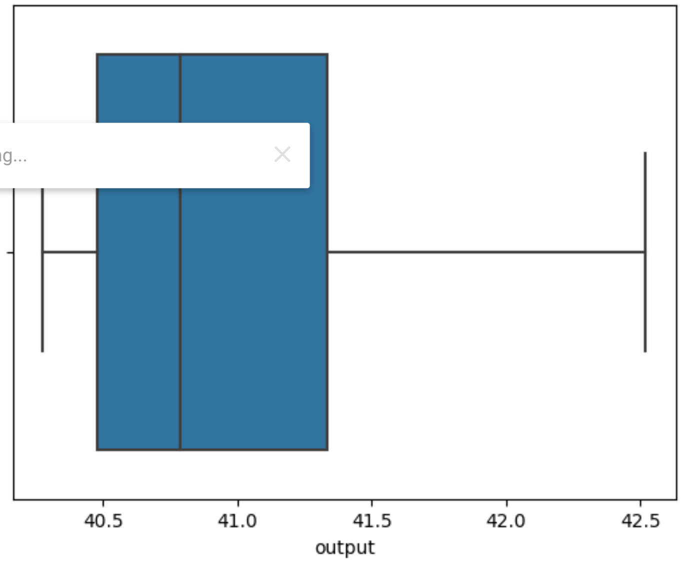



```
df['output'].skew()
percentile25=df['output'].quantile(0.25)
percentile75=df['output'].quantile(0.75)
iqr=percentile75-percentile25
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25+1.5*iqr

newdf=df.loc[(df['output']<upper_limit)&(df['output']>lower_limit)]
print(len(df))
print(len(newdf))
```

```
2461
137
```

```
sns.boxplot(data=newdf, x='output')
```

```
<Axes: xlabel='output'>
```

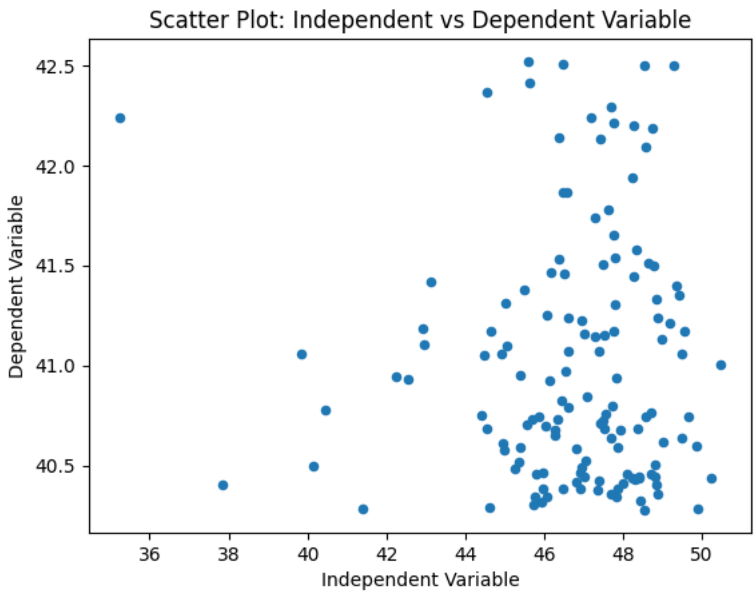



```
newdf
```

	CLOCK	X1	X2	X3	X4	X5	X6	X7	X8	X9
5	2022-12-26 03:00:00	46.49890	36.73950	34.69115	35.00760	60.91395	8.79055	155.91180	387.32905	919.85
32	2022-12-20 01:00:00	48.78120	39.10060	34.89470	36.61255	66.95260	9.58575	138.03350	389.17670	920.00
38	2022-12-29 07:30:00	46.57750	37.72965	33.14280	35.49450	64.09080	13.58100	134.12380	361.28640	920.00
46	2022-12-30 02:20:00	45.03310	36.36985	32.58205	34.48370	59.32215	7.38100	143.52900	336.35205	920.00
49	2022-12-19 13:50:00	47.80290	38.69705	34.91170	35.98200	65.99985	7.82405	138.09640	377.56425	919.80
...	...	...	...	...	...	...	...	...	...	...
	2022-									

```
newdf.plot.scatter(x='X1', y='output')
plt.xlabel('Independent Variable')
plt.ylabel('Dependent Variable')
```

```
plt.title('Scatter Plot: Independent vs Dependent Variable')
plt.show()
```



CHECKING STATIONARITY

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(newdf['X1'])
```

Saving...

```
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 5.745986305315026e-21
The time series is stationary.
```

```
result = adfuller(newdf['X2'])
p_value = result[1]
print("ADF test p-value:", p_value)

if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 1.3674325296808186e-21
The time series is stationary.
```

```
result = adfuller(newdf['X3'])
p_value = result[1]
print("ADF test p-value:", p_value)

if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 1.0011844372785953e-20
The time series is stationary.
```

```
result = adfuller(newdf['X4'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 1.462674513731191e-21
The time series is stationary.
```

```
result = adfuller(newdf['X5'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
else:
 print("The time series is notstationary.")
```

```
ADF test p-value: 5.516367337689574e-22
The time series is stationary.
```

```
result = adfuller(newdf['X6'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 3.920498714625382e-21
The time series is stationary.
```

```
result = adfuller(newdf['X7'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 1.465504552333475e-07
The time series is stationary.
```

```
result = adfuller(newdf['X8'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 3.701978429574922e-19
The time series is stationary.
```

```
result = adfuller(newdf['X9'])
p_value = result[1]
```

Saving...

```
    print("The time series is stationary.")
```

```
ADF test p-value: 6.483972675406812e-21
The time series is stationary.
```

```
result = adfuller(newdf['X10'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 8.499578137477644e-17
The time series is stationary.
```

```
result = adfuller(newdf['X11'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 4.817530518378106e-21
The time series is stationary.
```

```
result = adfuller(newdf['X12'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 6.870695129139611e-20
The time series is stationary.
```

```
result = adfuller(newdf['X13'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 4.556792446926935e-23
The time series is stationary.
```

```
result = adfuller(df['X14'])
p_value = result[1]
print("ADF test p-value:", p_value)
if p_value <= 0.05:
    print("The time series is stationary.")
```

```
ADF test p-value: 0.0
The time series is stationary.
```

```
correlation=newdf.corr()
print(correlation)
```

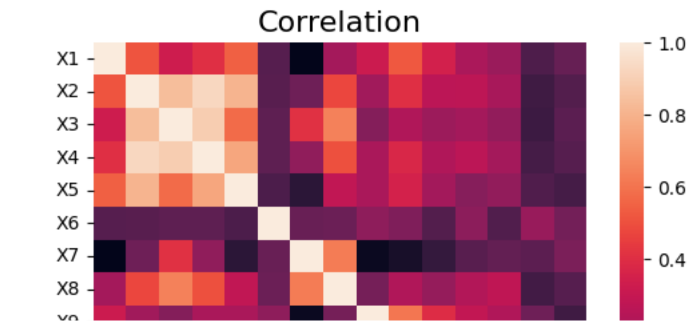
```
              X1        X2        X3        X4        X5        X6        X7  \
X1      1.000000  0.514831  0.327415  0.407710  0.545590 -0.058986 -0.363575
X2      0.514831  1.000000  0.839376  0.928894  0.805550 -0.054538  0.024117
X3      0.327415  0.839376  1.000000  0.892398  0.579092 -0.030495  0.410912
X4      0.407710  0.928894  0.892398  1.000000  0.760239 -0.031635  0.128197
X5      0.545590  0.805550  0.579092  0.760239  1.000000 -0.091912 -0.205320
X6     -0.058986 -0.054538 -0.030495 -0.031635 -0.091912  1.000000 -0.001052
X7     -0.363575  0.024117  0.410912  0.128197 -0.205320 -0.001052  1.000000
X8      0.194413  0.477915  0.641767  0.501176  0.281711  0.012600  0.623369
X9      0.323272  0.183526  0.093327  0.210846  0.210467  0.124879 -0.324439
X10     0.521576  0.405116  0.232883  0.371852  0.354414  0.078284 -0.275992
X11     0.345737  0.265453  0.168345  0.232029  0.189636 -0.068958 -0.175923
X12     0.216819  0.273203  0.195061  0.268187  0.099261  0.118027 -0.051588
X13     0.160047  0.206203  0.134088  0.193876  0.134144 -0.070946 -0.016231
X14    -0.081586 -0.139709 -0.147480 -0.116103 -0.080135  0.153524 -0.037462
output -0.002562 -0.068807 -0.042989 -0.057900 -0.114618  0.035996  0.064654

              X8        X9       X10       X11       X12       X13       X14  \
X1      0.194413  0.323272  0.521576  0.345737  0.216819  0.160047 -0.081586
X2      0.477915  0.183526  0.405116  0.265453  0.273203  0.206203 -0.139709
X3      0.641767  0.093327  0.232883  0.168345  0.195061  0.134088 -0.147480
X4      0.501176  0.210846  0.371852  0.232029  0.268187  0.193876 -0.116103
X5      0.281711  0.210467  0.354414  0.189636  0.099261  0.134144 -0.080135
X6      0.012600  0.124879  0.078284 -0.068958  0.118027 -0.070946  0.153524
[illegible] 75992 -0.175923 -0.051588 -0.016231 -0.037462
[illegible] 31357  0.147690  0.235578  0.279499 -0.126703
[illegible] 07546  0.392958  0.286456  0.230730  0.022969
X10     0.231357  0.607546  1.000000  0.736117  0.559524  0.724307 -0.023356
X11     0.147690  0.392958  0.736117  1.000000  0.598965  0.699599 -0.106642
X12     0.235578  0.286456  0.559524  0.598965  1.000000  0.486341 -0.243951
X13     0.279499  0.230730  0.724307  0.699599  0.486341  1.000000 -0.139195
X14    -0.126703  0.022969 -0.023356 -0.106642 -0.243951 -0.139195  1.000000
output -0.056289 -0.138181 -0.033596  0.064076  0.060252  0.064252  0.037144

          output
X1     -0.002562
X2     -0.068807
X3     -0.042989
X4     -0.057900
X5     -0.114618
X6      0.035996
X7      0.064654
X8     -0.056289
X9     -0.138181
X10    -0.033596
X11     0.064076
X12     0.060252
X13     0.064252
X14     0.037144
output  1.000000
<ipython-input-83-987ea4cd76c3>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
  correlation=newdf.corr()
```

Saving...

CHECKING CORRELATION

```
plt.title('Correlation',y=1,size=16)
sns.heatmap(correlation,square=True)
```

```
<Axes: title={'center': 'Correlation'}>
```

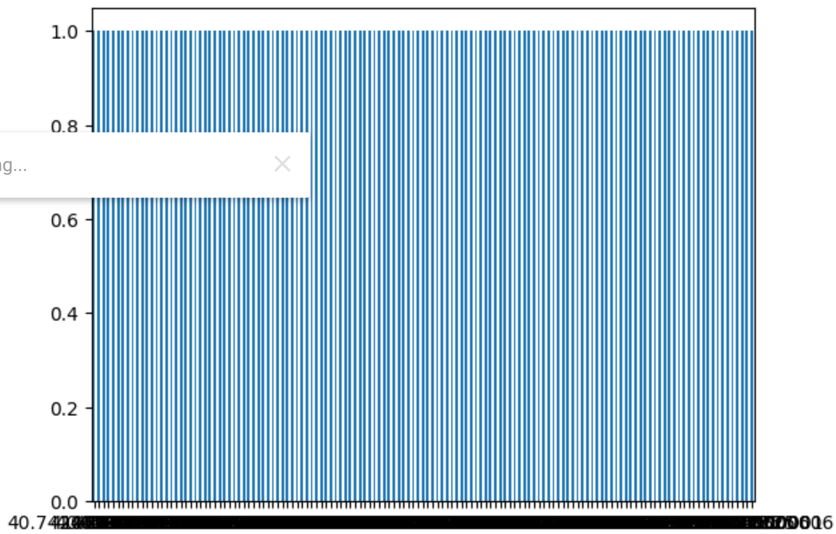



```
x=(newdf.drop(['output','CLOCK'],axis=1))

y=(newdf['output'])
print(x.shape)
print(y.shape)
```

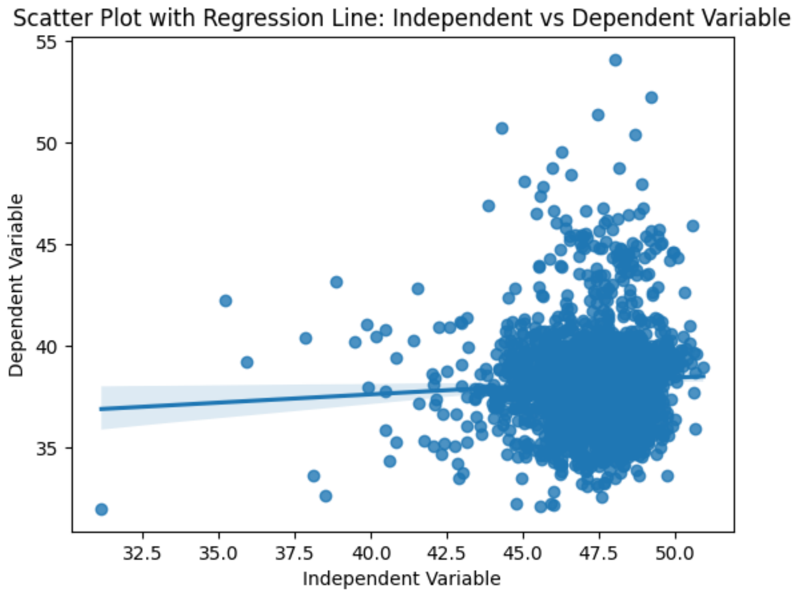
```
(137, 14)
(137,)
```

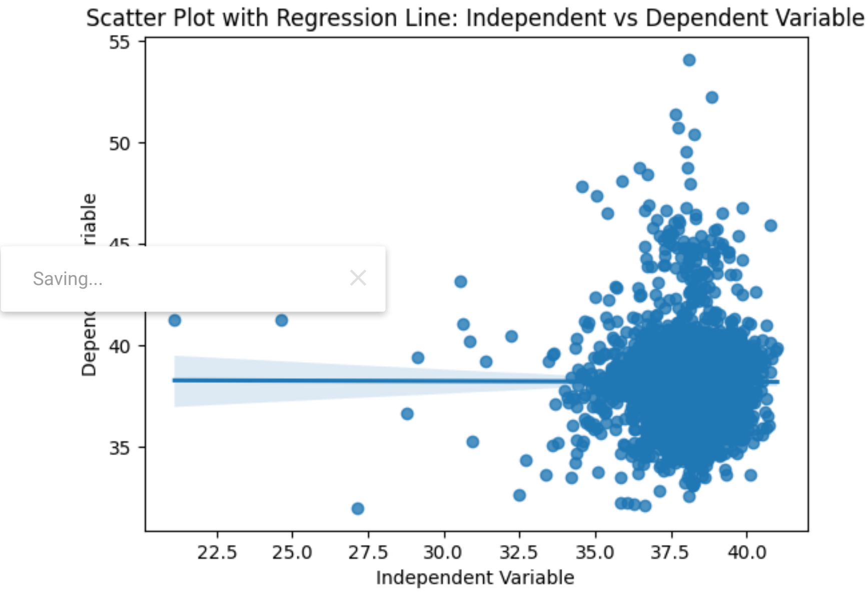
```
count_classes=pd.value_counts(newdf['output'],sort=True)
count_classes.plot(kind='bar',rot=0)
```

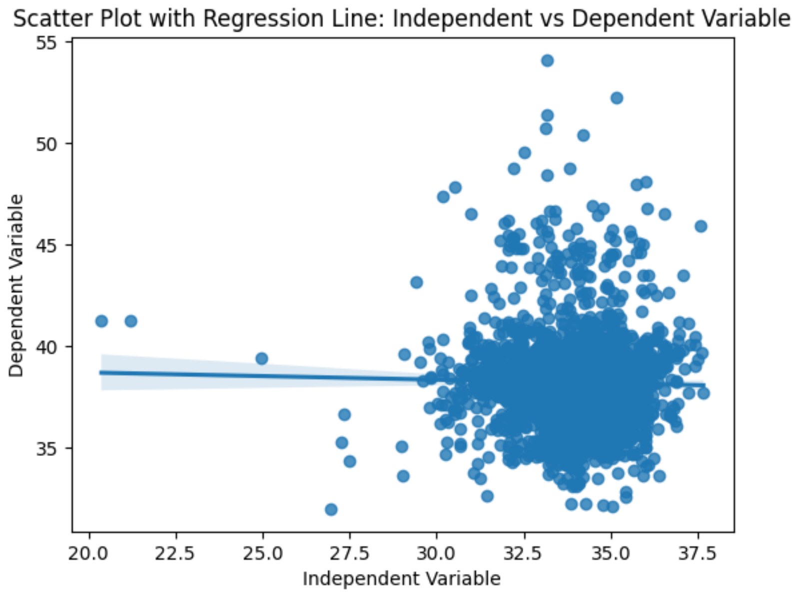
```
<Axes: >
```

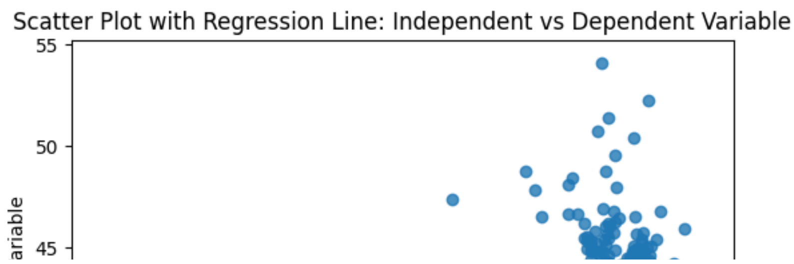


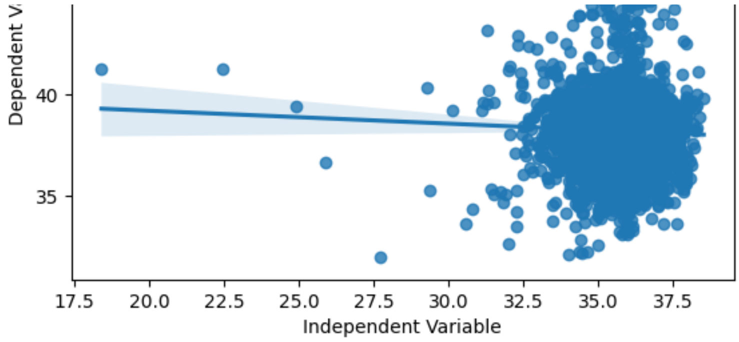

```
for col in df.select_dtypes(include='number'):
  sns.regplot(x=col, y='output', data=df)
  plt.xlabel('Independent Variable')
  plt.ylabel('Dependent Variable')
  plt.title('Scatter Plot with Regression Line: Independent vs Dependent Variable')
  plt.show()
```


Scatter Plot with Regression Line: Independent vs Dependent Variable
Dependent Variable
55
50
45
40
35
32.5 35.0 37.5 40.0 42.5 45.0 47.5 50.0
Independent Variable

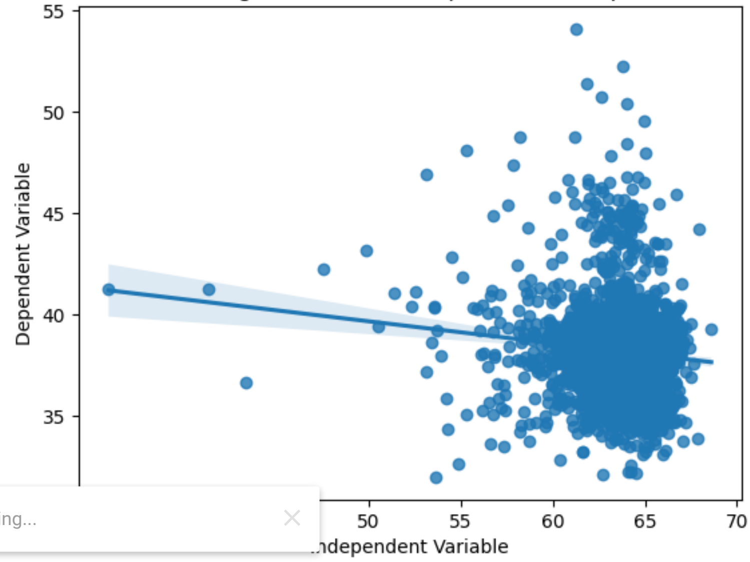

Scatter Plot with Regression Line: Independent vs Dependent Variable
55
50
45
40
35
22.5 25.0 27.5 30.0 32.5 35.0 37.5 40.0
Independent Variable
Saving...


Scatter Plot with Regression Line: Independent vs Dependent Variable
Dependent Variable
55
50
45
40
35
20.0 22.5 25.0 27.5 30.0 32.5 35.0 37.5
Independent Variable


Scatter Plot with Regression Line: Independent vs Dependent Variable
55
50
45

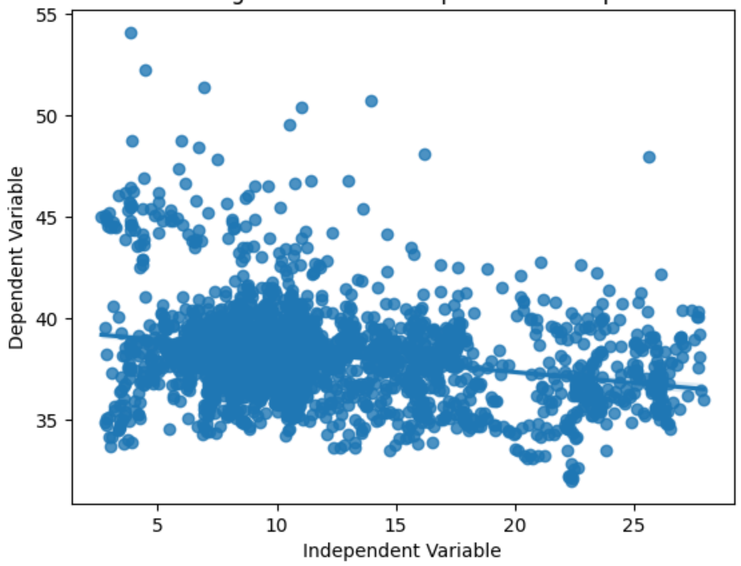


Scatter Plot with Regression Line: Independent vs Dependent Variable

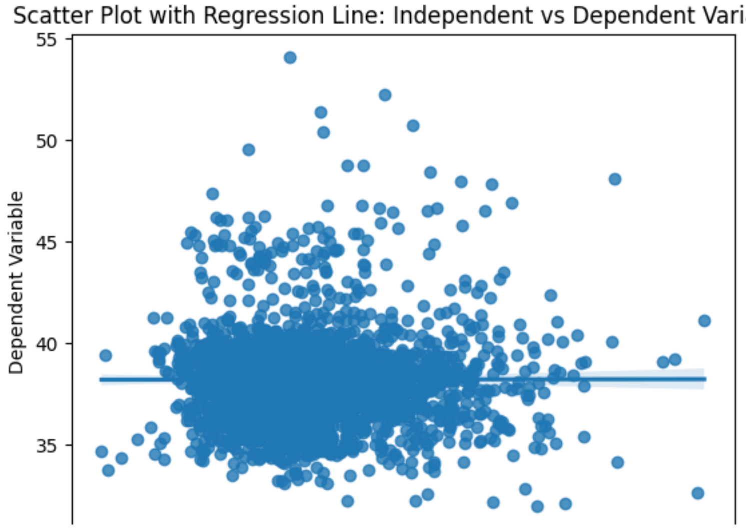


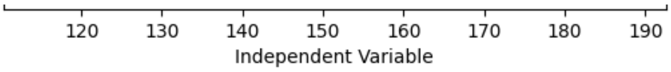
Saving...

Scatter Plot with Regression Line: Independent vs Dependent Variable

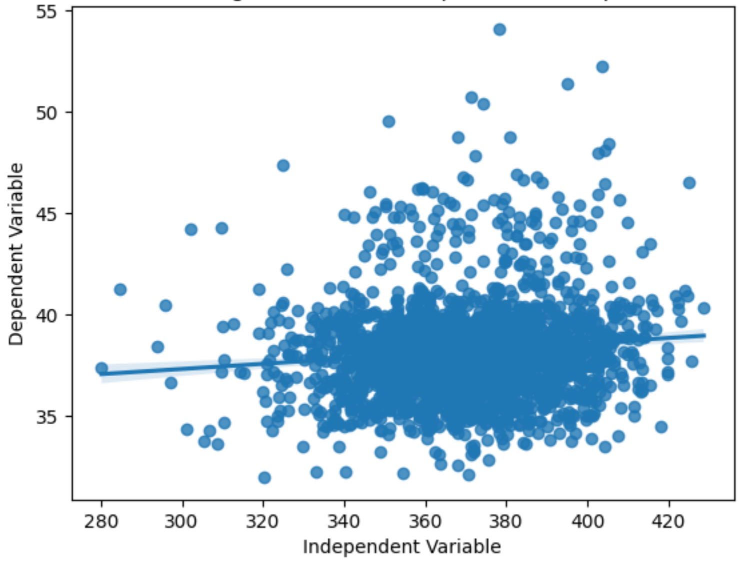


Scatter Plot with Regression Line: Independent vs Dependent Variable

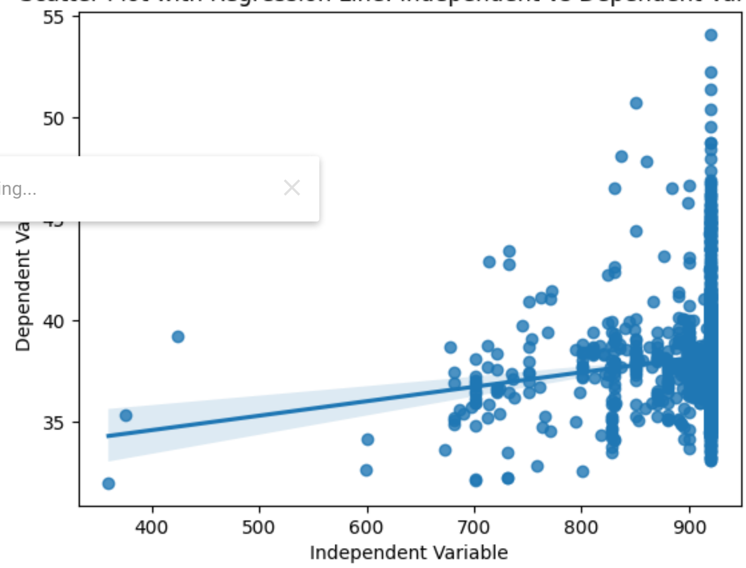




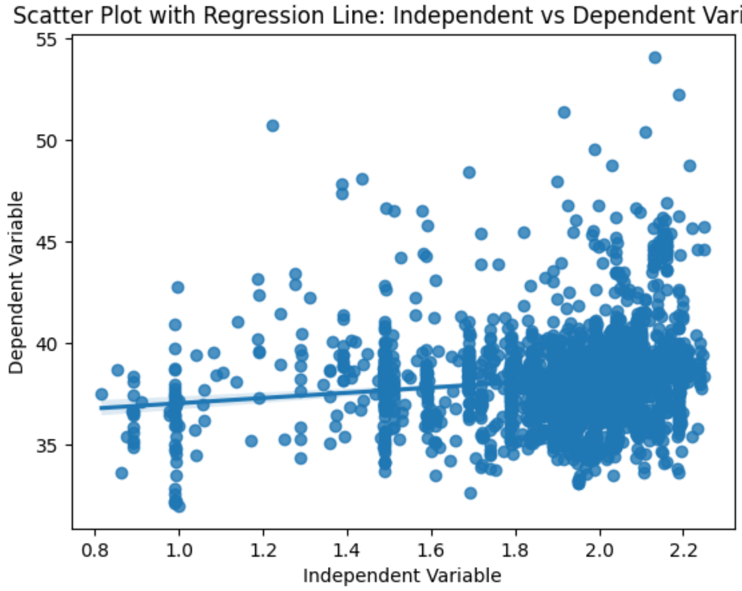
Scatter Plot with Regression Line: Independent vs Dependent Variable



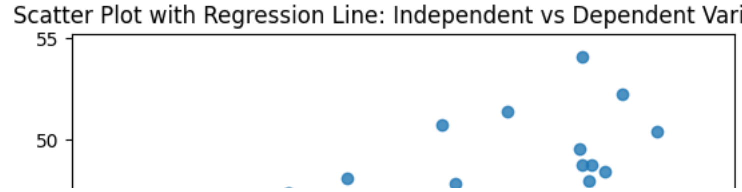
Scatter Plot with Regression Line: Independent vs Dependent Variable

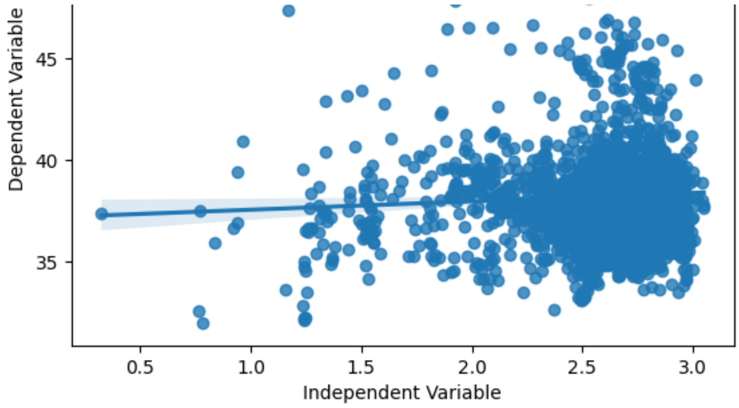


Scatter Plot with Regression Line: Independent vs Dependent Variable

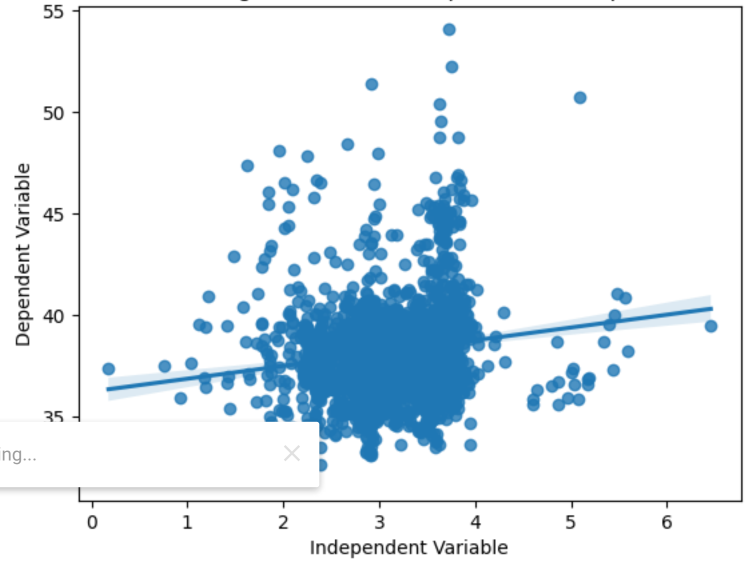


Scatter Plot with Regression Line: Independent vs Dependent Variable

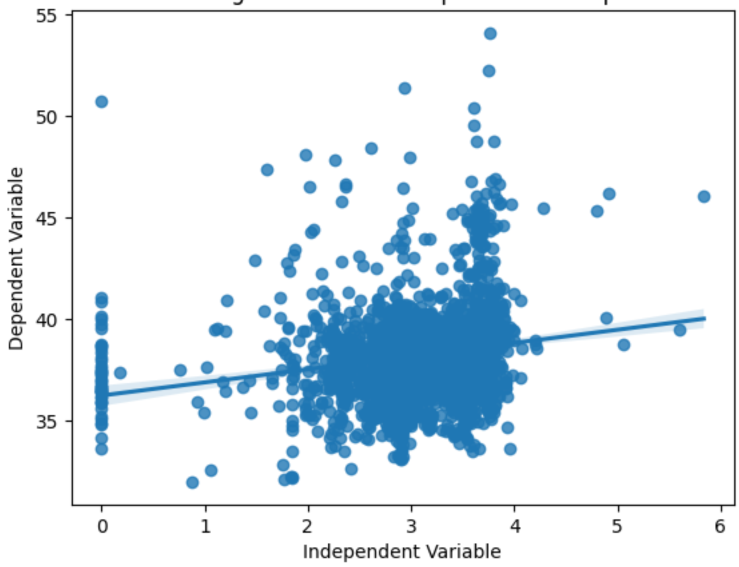




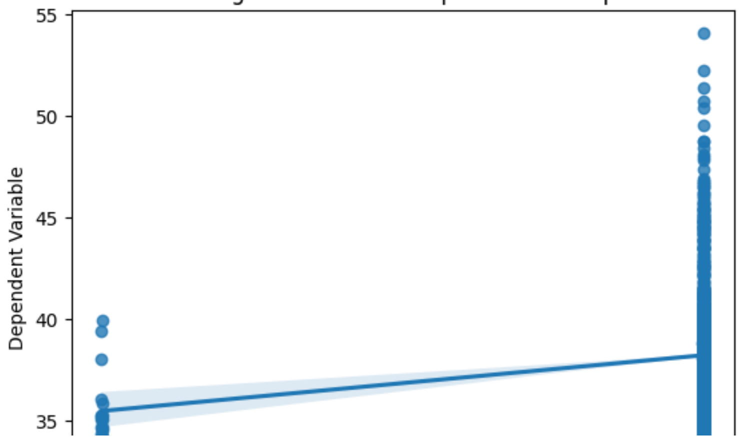
Scatter Plot with Regression Line: Independent vs Dependent Variable



Scatter Plot with Regression Line: Independent vs Dependent Variable



Scatter Plot with Regression Line: Independent vs Dependent Variable



Scatter Plot with Regression Line: Independent vs Dependent Variable

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
X_train.shape,X_test.shape
```

```
((95, 14), (42, 14))
```

APPLYING NORMALISATION

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

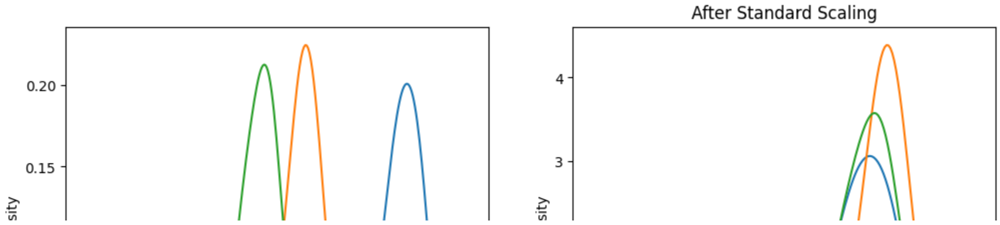
```
X_train_scaled=pd.DataFrame(X_train_scaled,columns=X_train.columns)
X_test_scaled=pd.DataFrame(X_test_scaled,columns=X_test.columns)
```

```
np.round(X_train.describe(),1)
```

Saving...

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5.0	95.0	95.0	95.0	95.0	95.0	95.0	95.0	95.0	95.0
mean	46.8	37.4	33.6	35.0	61.7	12.2	141.7	371.7	913.3	1.9	2.6	3.3	3.2	7.7
std	2.3	2.7	2.4	2.6	4.9	5.6	12.0	26.0	26.3	0.3	0.3	0.6	0.7	0.0
min	35.2	21.1	20.4	18.4	35.8	3.1	120.5	284.6	762.2	1.1	1.5	1.7	0.0	7.7
25%	46.0	36.9	32.5	34.5	61.2	8.8	133.0	357.4	919.9	1.8	2.6	2.9	2.9	7.7
50%	47.3	37.9	33.9	35.6	63.1	10.5	139.6	375.8	920.0	2.0	2.7	3.5	3.4	7.7
75%	48.4	38.8	35.0	36.2	64.2	14.6	148.7	388.8	920.0	2.1	2.8	3.7	3.7	7.7
max	50.5	40.7	37.2	38.4	67.0	27.7	188.8	428.7	920.2	2.2	3.0	5.6	4.0	7.7

```
fig,(ax1, ax2)= plt.subplots(ncols=2,figsize=(12, 5))

#before scaling axi.set_title('Before Scaling')

sns.kdeplot(X_train['X1'],ax=ax1)
sns.kdeplot(X_train['X2'],ax=ax1)
sns.kdeplot(X_train['X3'],ax=ax1)
#after scaling
ax2.set_title('After Standard Scaling')
sns.kdeplot(X_train_scaled['X1'],ax=ax2)
sns.kdeplot(X_train_scaled['X2'],ax=ax2)
sns.kdeplot(X_train_scaled['X3'],ax=ax2)

plt.show()
```



ML MODEL USING LINEAR REGRESSION

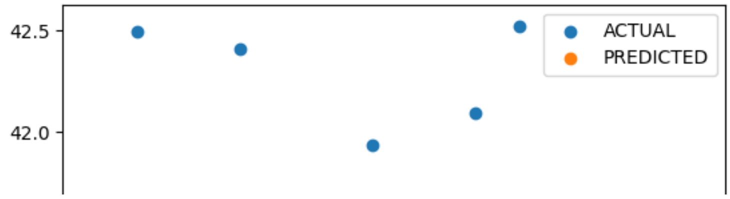
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.model_selection import  GridSearchCV
from sklearn.linear_model import Ridge
model = LinearRegression()

param_grid = {'alpha': [0.1, 1.0, 10.0], 'max_iter': [10, 100, 1000],'fit_intercept':[True,False]}
ridge = Ridge(param_grid)

#cross-validation
grid_search = GridSearchCV(ridge, param_grid, cv=5)
grid_search.fit(X_train_scaled,y_train)

print("hyperparameters:", grid_search.best_params_)
print("score:", grid_search.best_score_)

model = grid_search.best_estimator_
y_pred = model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
```



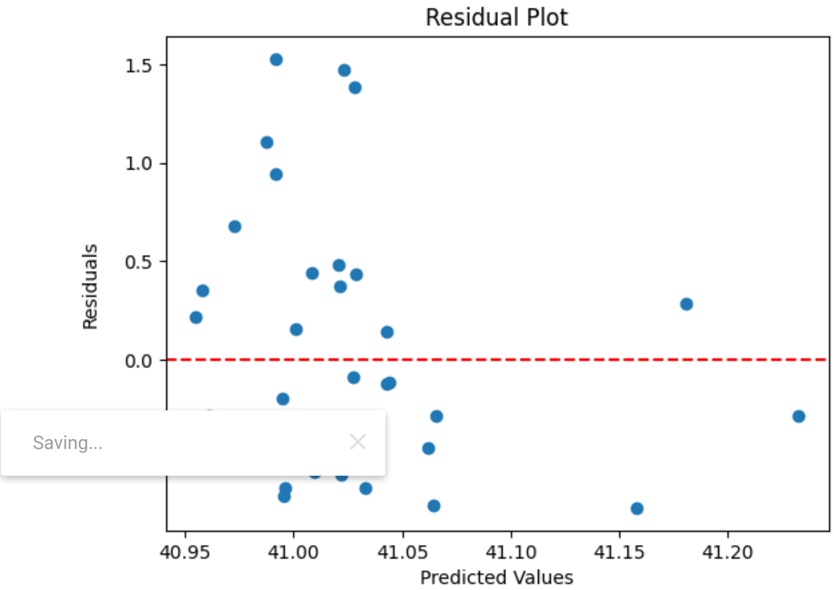
```
intercept =model.intercept_

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse1)
print("Coefficients:", coefficients)
print("Intercept:", intercept)
print(y_pred)
```

```
hyperparameters: {'alpha': 10.0, 'fit_intercept': True, 'max_iter': 10}
score: -0.09888222757513718
Mean Squared Error: 0.39221779052798167
Root Mean Squared Error: 0.6262729361292739
Coefficients: [-0.01153665 -0.02485005  0.00069192 -0.01478967 -0.05382455  0.04310128
  0.10618496 -0.02874678 -0.11032604 -0.05280059 -0.0331805  -0.00433651
  0.01812458  0.0482288 ]
Intercept: 41.196426214273984
[41.01036894 41.02120848 41.04439714 41.02290897 41.1805145  41.04303165
 40.98302172 40.96117219 41.1582513  41.0053333  41.02816517 41.01113833
 40.99259039 41.2325796  41.06444147 41.02082327 40.99231579 40.9910005
 41.06584699 40.99168217 41.04286046 40.95769599 41.03297301 40.99586992
 40.99512223 41.0272108  40.98747652 41.02186067 41.00108212 40.99164623
 41.00984398 40.99552306 41.06183716 41.02138876 41.00831152 41.01250068
 40.95507321 41.02847313 41.01660515 40.97256688 41.01351563 40.96107077]
```

```
import matplotlib.pyplot as plt

plt.scatter(range(len(y_test)), y_test)
plt.scatter(range(len(y_pred)), y_pred)
plt.legend(['ACTUAL', 'PREDICTED'])
plt.show()
```




```
residuals = y_test - y_pred

plt.scatter(y_pred, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residual Plot')
plt.show()
```



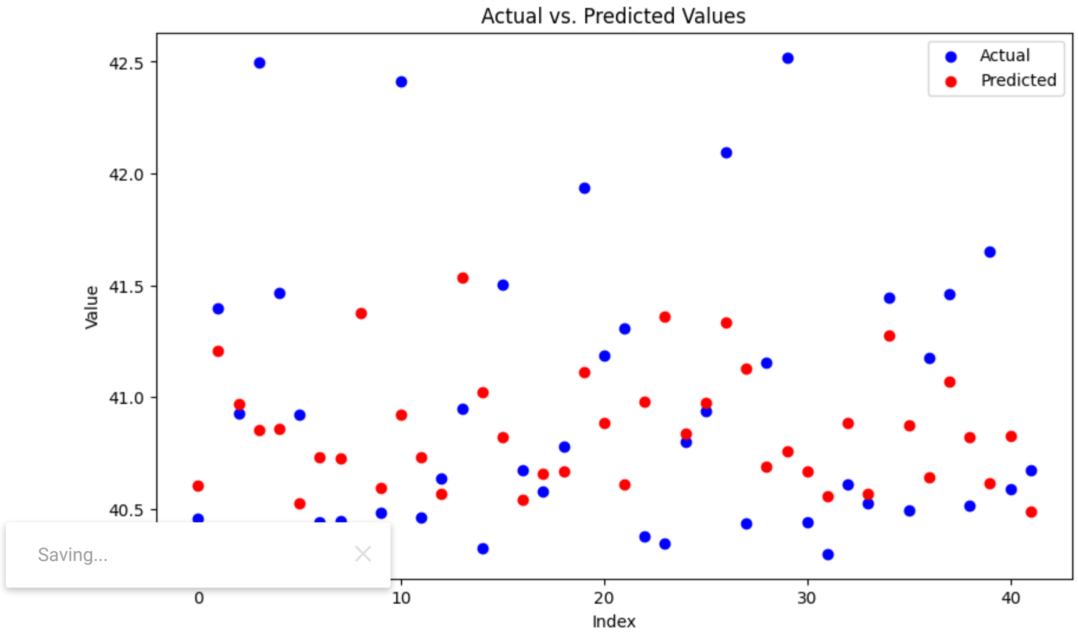
ML MODEL USING XGBOOST

```
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': list(range(100, 1000, 100))
}
model = xgb.XGBRegressor(objective='reg:squarederror', max_depth=3, learning_rate=0.1, n_estimators=100)

grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train_scaled, y_train)

print("Hyperparameters:", grid_search.best_params_)
print("Best score:", -grid_search.best_score_)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse2= np.sqrt(mse)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse2)

r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)
```

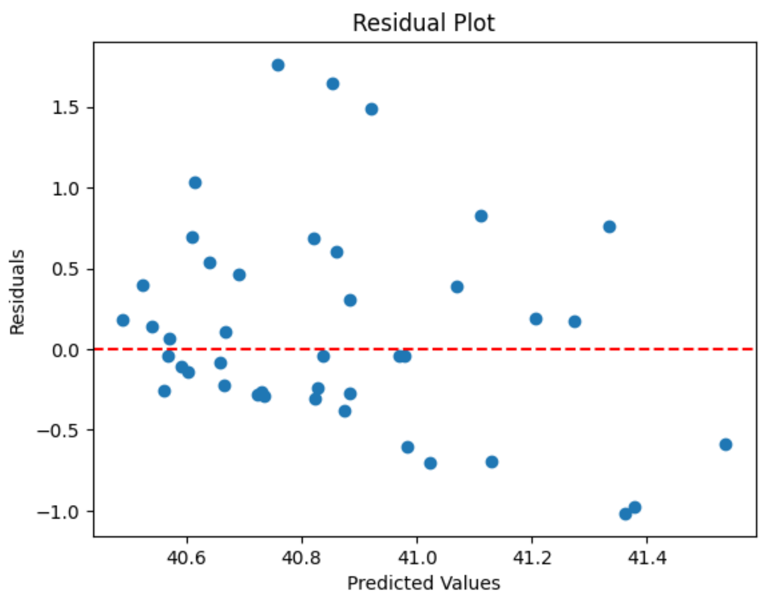
```
Hyperparameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 600}
Best score: 0.41745777882228535
Mean Squared Error: 0.4085904635994367
Mean Absolute Error: 0.47753576543898746
```

```
Root Mean Squared Error: 0.6392108131121036
R-squared Score: -0.06538119922619368
```

```
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='b', label='Actual')
plt.scatter(range(len(y_pred)), y_pred, color='r', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```




```
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residual Plot')
plt.show()
```



ML MODEL USING SUPPORT VECTOR REGRESSION

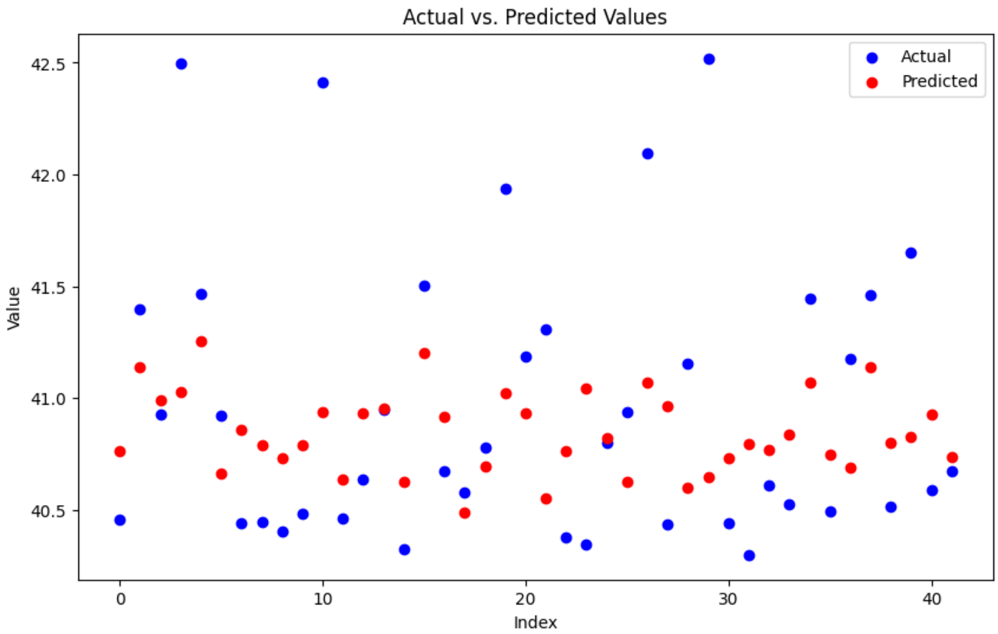
```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
from sklearn.model_selection import GridSearchCV

model = SVR(kernel='rbf')
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.01, 0.1, 1],
    'epsilon': [0.01, 0.1, 1],
    'kernel': ['rbf', 'linear', 'poly']
}
# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train_scaled, y_train)

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse3 = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Hyperparameters:", grid_search.best_params_)
print("Best score:", -grid_search.best_score_)
print("Root Mean Squared Error:", rmse3)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.43944475567837415
Hyperparameters: {'C': 1, 'epsilon': 0.01, 'gamma': 1, 'kernel': 'rbf'}
Best score: 0.3817036498095305
Root Mean Squared Error: 0.5916615002961743
Mean Squared Error: 0.3500633309327198
R-squared: 0.08722565860019482
```

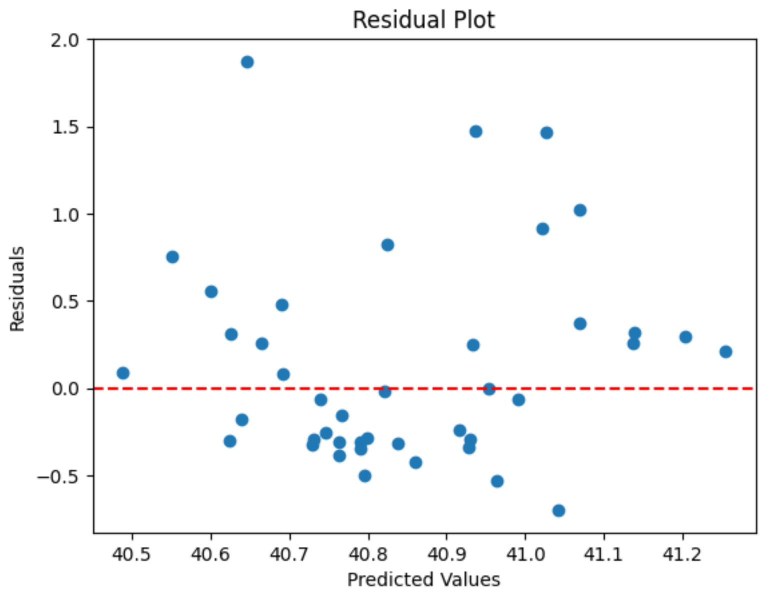


```
plt.scatter(range(len(y_test)), y_test, color='b', label='Actual')
plt.scatter(range(len(y_pred)), y_pred, color='r', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```




```
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
```

```
plt.title('Residual Plot')
plt.show()
```

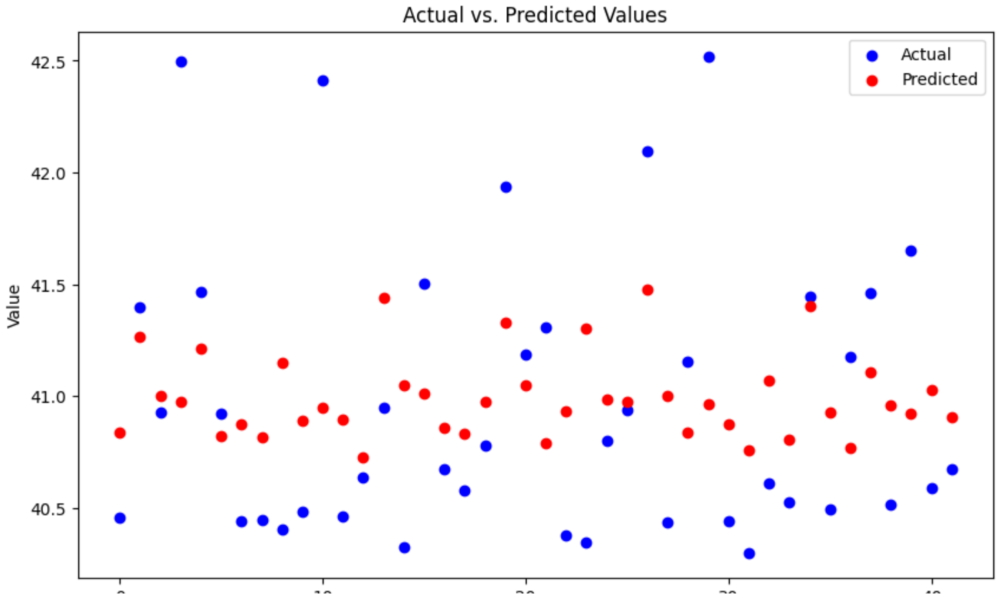


ML MODEL USING RANDOM FOREST REGRESSOR

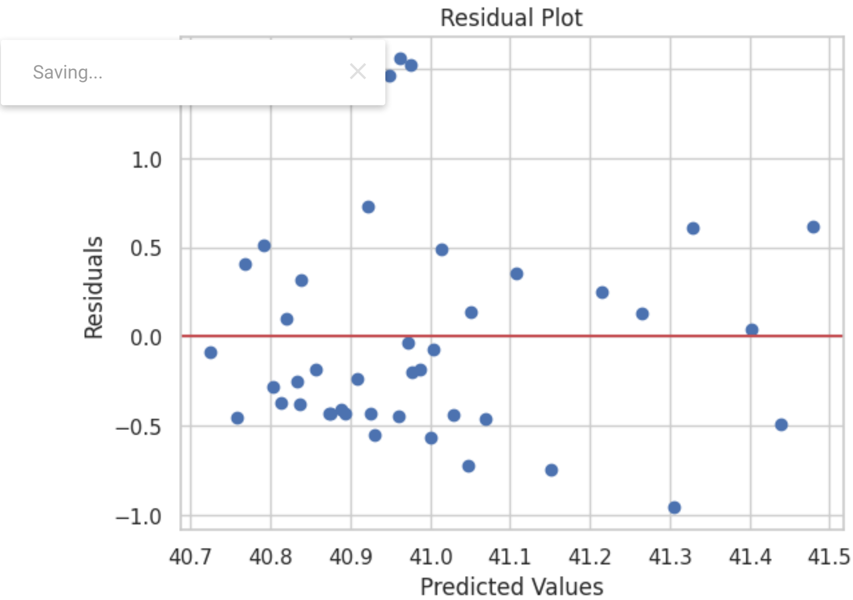
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV
Saving...                                      1000, 100)),
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
model = RandomForestRegressor(random_state=42)
# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train_scaled, y_train)

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test_scaled)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse4 = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Hyperparameters:", grid_search.best_params_)
print("Best score:", -grid_search.best_score_)
print("Root Mean Squared Error:", rmse4)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.4640138351644905
Hyperparameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 700}
Best score: 0.37442932730958683
Root Mean Squared Error: 0.5846155985097846
Mean Squared Error: 0.34177539802095364
R-squared: 0.10883606973622173
```

```
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='b', label='Actual')
plt.scatter(range(len(y_pred)), y_pred, color='r', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```




```
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r')
plt.title('Residual Plot')
plt.show()
```



MODEL ACCURACY COMPARISION

```
models = ['Linear Regression','XGBoost','SVR','Random Forest']
rmse= [rmse1,rmse2,rmse3,rmse4]
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=rmse)
plt.title('Accuracy of Different Models')
plt.xlabel('Models')
plt.ylabel('RMSE')
plt.show()
```