In [138…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import openpyxl as xls
from sklearn.svm import SVR  # support vector regressor ( imported when line of coc
%matplotlib inline
```

In [2]:
```
dataframe=pd.read_csv("H:\\Realdata.csv")
dataframe.head()
```

Out[2]:

	DateTime	kVA	Min Temp	Max Temp
0	4/14/2017 1:00	726.0	12.5	28.2
1	4/14/2017 2:00	704.0	12.5	28.2
2	4/14/2017 3:00	704.0	12.5	28.2
3	4/14/2017 4:00	748.0	12.5	28.2
4	4/14/2017 5:00	814.0	12.5	28.2

In [85]:
```
dataframe['DateTime']=pd.to_datetime(dataframe['DateTime'])
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23359 entries, 0 to 23358
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   DateTime  23359 non-null  datetime64[ns]
 1   kVA       22857 non-null  float64
 2   Min Temp  23359 non-null  float64
 3   Max Temp  23359 non-null  float64
 4   AvgTemp   23359 non-null  float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 912.6 KB
```

In [89]:
```
dataframe=dataframe.set_index('DateTime')
```

In [3]:
```
dataframe.shape
```

Out[3]:
```
(23359, 4)
```

In [4]:
```
dataframe['AvgTemp']=((dataframe['Min Temp']+dataframe['Max Temp'])/2)
```

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

Out[5]:

	DateTime	kVA	Min Temp	Max Temp	AvgTemp
0	4/14/2017 1:00	726.0	12.5	28.2	20.35
1	4/14/2017 2:00	704.0	12.5	28.2	20.35
2	4/14/2017 3:00	704.0	12.5	28.2	20.35
3	4/14/2017 4:00	748.0	12.5	28.2	20.35
4	4/14/2017 5:00	814.0	12.5	28.2	20.35

In [6]:
```
copydata=dataframe.copy()
```

In [7]:
```
copydata['AvgTemp']=copydata['Min Temp']+ copydata['Max Temp']/2
```

In [8]:
```
copydata['AvgTemp'].value_counts()
```

Out[8]:
```
34.00    384
35.00    320
34.50    320
34.25    288
34.30    288
        ...
9.95      14
10.30      7
10.10      4
10.20      2
10.37      1
Name: AvgTemp, Length: 365, dtype: int64
```

In [9]:
```
copydata.isnull().sum()
```

Out[9]:
```
DateTime      0
kVA         502
Min Temp      0
Max Temp      0
AvgTemp       0
dtype: int64
```

In [10]:
```
copydata[copydata['kVA'].isnull()].head()
```

Out[10]:

	DateTime	kVA	Min Temp	Max Temp	AvgTemp
36	4/15/2017 5:00	NaN	12.8	28.8	27.2
37	4/15/2017 6:00	NaN	12.8	28.8	27.2
38	4/15/2017 6:30	NaN	12.8	28.8	27.2
39	4/15/2017 7:00	NaN	12.8	28.8	27.2
40	4/15/2017 7:30	NaN	12.8	28.8	27.2

In [11]:
```
copydata.isnull().sum()
```

Out[11]:
```
DateTime      0
kVA         502
Min Temp      0
Max Temp      0
AvgTemp       0
dtype: int64
```

Separating null values from copydata dataframe

In [12]:
```
test_data=copydata[copydata['kVA'].isnull()]
```

In [13]:
```
test_data
```

Out[13]:

	DateTime	kVA	Min Temp	Max Temp	AvgTemp
36	4/15/2017 5:00	NaN	12.8	28.8	27.2
37	4/15/2017 6:00	NaN	12.8	28.8	27.2
38	4/15/2017 6:30	NaN	12.8	28.8	27.2
39	4/15/2017 7:00	NaN	12.8	28.8	27.2
40	4/15/2017 7:30	NaN	12.8	28.8	27.2
...	...	...	...	...	...
23113	4/6/2019 8:00	NaN	12.2	17.6	21.0
23114	4/6/2019 8:30	NaN	12.2	17.6	21.0
23122	4/6/2019 15:00	NaN	12.2	17.6	21.0
23201	4/9/2019 2:00	NaN	13.3	24.4	25.5
23217	4/9/2019 14:00	NaN	13.3	24.4	25.5

502 rows × 5 columns

Dropping null values from copydata dataframe and considering as train data

In [20]:
```
copydata.dropna(inplace=True)
```

In [21]:
```
copydata.size
```

Out[21]:
```
114285
```

In [22]:
```
copydata
```

Out[22]:

	DateTime	kVA	Min Temp	Max Temp	AvgTemp
0	2017-04-14 01:00:00	726.0	12.5	28.2	26.60
1	2017-04-14 02:00:00	704.0	12.5	28.2	26.60
2	2017-04-14 03:00:00	704.0	12.5	28.2	26.60
3	2017-04-14 04:00:00	748.0	12.5	28.2	26.60
4	2017-04-14 05:00:00	814.0	12.5	28.2	26.60
...	...	...	...	...	...
23354	2019-04-13 20:00:00	1474.0	16.0	29.5	30.75
23355	2019-04-13 20:30:00	1397.0	16.0	29.5	30.75
23356	2019-04-13 21:00:00	1254.0	16.0	29.5	30.75
23357	2019-04-13 22:00:00	1056.0	16.0	29.5	30.75
23358	2019-04-13 23:00:00	825.0	16.0	29.5	30.75

22857 rows × 5 columns

In [23]:
```
copydata.isnull().sum()
```

Out[23]:
```
DateTime    0
kVA         0
Min Temp    0
Max Temp    0
AvgTemp     0
dtype: int64
```

In [24]:
```
copydata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22857 entries, 0 to 23358
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   DateTime  22857 non-null  datetime64[ns]
 1   kVA       22857 non-null  float64
 2   Min Temp  22857 non-null  float64
 3   Max Temp  22857 non-null  float64
 4   AvgTemp   22857 non-null  float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 1.0 MB
```

In [25]:
```
copydata['DateTime']=pd.to_datetime(copydata['DateTime'])
```

In [26]:
```
copydata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22857 entries, 0 to 23358
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   DateTime  22857 non-null  datetime64[ns]
 1   kVA       22857 non-null  float64
 2   Min Temp  22857 non-null  float64
 3   Max Temp  22857 non-null  float64
 4   AvgTemp   22857 non-null  float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 1.0 MB
```

Creating X_train and y_train from copydata

In [34]:
```
y_train=copydata['kVA']
```

In [35]:
```
y_train
```

Out[35]:
```
0         726.0
1         704.0
2         704.0
3         748.0
4         814.0
          ...
23354    1474.0
23355    1397.0
23356    1254.0
23357    1056.0
23358     825.0
Name: kVA, Length: 22857, dtype: float64
```

In [29]:
```
y_train.shape
```

Out[29]:
```
(22857,)
```

In [36]:
```
X_train=copydata.drop(columns=['DateTime','kVA'])
```

In [37]:
```
X_train.shape
```

Out[37]:
```
(22857, 3)
```

In [38]:
```
X_train
```

Out[38]:

	Min Temp	Max Temp	AvgTemp
0	12.5	28.2	26.60
1	12.5	28.2	26.60
2	12.5	28.2	26.60
3	12.5	28.2	26.60
4	12.5	28.2	26.60
...	...	...	...
23354	16.0	29.5	30.75
23355	16.0	29.5	30.75
23356	16.0	29.5	30.75
23357	16.0	29.5	30.75
23358	16.0	29.5	30.75

22857 rows × 3 columns

Building the linear regression model

In [39]:
```
from sklearn.linear_model import LinearRegression
linreg=LinearRegression()
```

In [98]:
```
# train the model withX_train and y_train
```

In [40]:
```
linreg.fit(X_train,y_train)
```

Out[40]:
```
LinearRegression
LinearRegression()
```

Creating X_test from test data where only null values are stored of kVA

In [41]:
```
X_test=test_data.drop(columns=['DateTime','kVA'])
```

In [42]:
```
X_test
```

Out[42]:

	Min Temp	Max Temp	AvgTemp
36	12.8	28.8	27.2
37	12.8	28.8	27.2
38	12.8	28.8	27.2
39	12.8	28.8	27.2
40	12.8	28.8	27.2
...	...	...	...
23113	12.2	17.6	21.0
23114	12.2	17.6	21.0
23122	12.2	17.6	21.0
23201	13.3	24.4	25.5
23217	13.3	24.4	25.5

502 rows × 3 columns

Applying the trained model in X_test and predicting the values

In [43]:
```
y_pred=linreg.predict(X_test)
```

In [44]:
```
y_pred
```

Out[44]:
```
array([1612.64570052, 1612.64570052, 1612.64570052, 1612.64570052,
       1612.64570052, 1612.64570052, 1612.64570052, 1730.81764476,
       1782.90354764, 1782.90354764, 1734.3137212 , 1734.3137212 ,
       1734.3137212 , 1635.09889476, 1611.00514476, 1591.38406832,
       1640.66139476, 1686.23174188, 1659.78639476, 1659.78639476,
       1673.15354764, 1654.71604764, 1680.47389476, 1680.47389476,
       1666.03639476, 1666.03639476, 1651.59889476, 1651.59889476,
       1735.51299188, 1680.09104764, 1680.09104764, 1753.958339  ,
       1753.958339  , 1753.958339  , 1715.520839  , 1708.08320052,
       1708.08320052, 1708.08320052, 1708.08320052, 1708.08320052,
       1708.08320052, 1689.65354764, 1689.65354764, 1689.65354764,
       1689.65354764, 1689.65354764, 1689.65354764, 1689.65354764,
       1689.65354764, 1689.65354764, 1689.65354764, 1689.65354764,
       1706.53639476, 1706.53639476, 1724.10674188, 1724.10674188,
       1726.42695052, 1708.88014476, 1707.32549188, 1716.09889476,
       1716.09889476, 1737.64570052, 1739.41139476, 1706.645839  ,
       1753.84889476, 1753.84889476, 1714.44264476, 1776.19264476,
       1761.75514476, 1761.75514476, 1761.75514476, 1761.75514476,
       1761.75514476, 1761.75514476, 1783.30195052, 1783.30195052,
       1783.30195052, 1783.30195052, 1732.19264476, 1732.19264476,
       1732.19264476, 1732.19264476, 1699.41924188, 1699.41924188,
       1699.41924188, 1660.19264476, 1682.53639476, 1704.302089  ,
       1704.302089  , 1704.302089  , 1704.302089  , 1704.302089  ,
       1704.302089  , 1704.302089  , 1704.302089  , 1704.302089  ,
       1734.52070052, 1734.52070052, 1692.208339  , 1692.208339  ,
       1692.208339  , 1692.208339  , 1692.208339  , 1692.208339  ,
       1696.08320052, 1696.08320052, 1687.30979764, 1677.770839  ,
       1677.770839  , 1677.770839  , 1677.770839  , 1677.770839  ,
       1677.770839  , 1677.770839  , 1677.770839  , 1713.75514476,
       1717.73945052, 1717.73945052, 1717.73945052, 1717.73945052,
       1715.41139476, 1662.52070052, 1662.52070052, 1662.52070052,
       1673.94264476, 1673.94264476, 1673.94264476, 1673.94264476,
       1673.94264476, 1673.94264476, 1673.94264476, 1657.16139476,
       1628.35281832, 1641.91139476, 1641.91139476, 1629.04031832,
       1614.79424188, 1632.6262212 , 1632.6262212 , 1637.5949712 ,
       1641.60281832, 1585.2512212 , 1585.2512212 , 1578.91139476,
       1601.6887212 , 1609.97193298, 1609.97193298, 1609.97193298,
       1609.97193298, 1609.97193298, 1609.97193298, 1609.97193298,
       1609.97193298, 1609.97193298, 1609.97193298, 1609.97193298,
       1613.19460654, 1609.87818298, 1608.91335654, 1636.38799188,
       1636.38799188, 1636.38799188, 1636.38799188, 1767.6887212 ,
       1767.6887212 , 1767.6887212 , 1669.78443298, 1553.1574712 ,
       1579.41139476, 1693.41139476, 1620.10281832, 1620.10281832,
       1620.10281832, 1620.10281832, 1620.10281832, 1651.57549188,
       1651.57549188, 1651.57549188, 1626.60674188, 1626.60674188,
       1611.48174188, 1611.48174188, 1611.48174188, 1611.48174188,
       1611.48174188, 1611.48174188, 1611.48174188, 1611.48174188,
       1611.48174188, 1611.48174188, 1593.91139476, 1593.91139476,
       1593.91139476, 1593.91139476, 1593.91139476, 1593.91139476,
       1593.91139476, 1593.91139476, 1593.91139476, 1593.91139476,
       1593.91139476, 1593.91139476, 1593.91139476, 1593.91139476,
       1593.91139476, 1574.79031832, 1574.79031832, 1574.79031832,
       1574.79031832, 1577.8137212 , 1577.8137212 , 1577.8137212 ,
       1577.8137212 , 1577.8137212 , 1577.8137212 , 1588.06764476,
       1588.06764476, 1585.72781832, 1618.60674188, 1618.60674188,
       1639.38406832, 1639.38406832, 1639.38406832, 1688.07549188,
       1688.07549188, 1675.98174188, 1675.98174188, 1675.98174188,
       1675.98174188, 1675.98174188, 1675.98174188, 1675.98174188,
       1675.98174188, 1695.1887212 , 1695.1887212 , 1695.1887212 ,
       1650.32549188, 1650.32549188, 1650.32549188, 1650.32549188,
       1645.34889476, 1745.52854764, 1745.52854764, 1745.52854764,
       1745.52854764, 1745.52854764, 1745.52854764, 1745.52854764,
       1745.52854764, 1745.52854764, 1745.52854764, 1777.54424188,
       1674.91139476, 1674.91139476, 1665.63014476, 1749.15354764,
```

```
1667.10674188, 1660.19264476, 1626.91139476, 1626.91139476,
1626.91139476, 1626.91139476, 1626.91139476, 1626.91139476,
1707.50514476, 1707.50514476, 1707.50514476, 1707.50514476,
1657.14962408, 1657.14962408, 1657.14962408, 1697.66139476,
1697.66139476, 1687.98945052, 1691.41139476, 1704.989589  ,
1728.08320052, 1728.08320052, 1684.19264476, 1684.19264476,
1684.19264476, 1684.19264476, 1683.40354764, 1683.40354764,
1699.31764476, 1699.31764476, 1699.31764476, 1803.50514476,
1803.50514476, 1825.15354764, 1756.19264476, 1749.864589  ,
1749.864589  , 1749.864589  , 1749.864589  , 1713.65354764,
1713.65354764, 1713.65354764, 1713.65354764, 1713.65354764,
1713.65354764, 1713.65354764, 1713.65354764, 1713.65354764,
1713.65354764, 1713.65354764, 1713.65354764, 1713.65354764,
1730.52854764, 1730.52854764, 1700.114589  , 1700.114589  ,
1700.114589  , 1698.552089  , 1698.552089  , 1689.57549188,
1689.57549188, 1723.31764476, 1699.21604764, 1699.21604764,
1699.21604764, 1754.63799188, 1754.63799188, 1754.63799188,
1725.864589  , 1725.864589  , 1801.16139476, 1684.09104764,
1723.41924188, 1723.41924188, 1723.41924188, 1723.41924188,
1723.41924188, 1740.20049188, 1740.20049188, 1740.20049188,
1740.20049188, 1694.52854764, 1694.52854764, 1711.20820052,
1711.20820052, 1711.20820052, 1696.10674188, 1696.10674188,
1696.10674188, 1696.10674188, 1696.10674188, 1696.10674188,
1696.10674188, 1696.98174188, 1696.98174188, 1728.88014476,
1720.86445052, 1720.86445052, 1720.86445052, 1777.84889476,
1777.84889476, 1713.64570052, 1770.63014476, 1770.63014476,
1770.63014476, 1720.86445052, 1720.86445052, 1720.86445052,
1720.86445052, 1720.86445052, 1715.30979764, 1715.30979764,
1683.427089  , 1683.427089  , 1683.427089  , 1683.427089  ,
1683.427089  , 1756.29424188, 1756.29424188, 1756.29424188,
1756.29424188, 1756.29424188, 1756.29424188, 1756.29424188,
1756.29424188, 1756.29424188, 1756.29424188, 1756.29424188,
1756.29424188, 1761.85674188, 1761.85674188, 1761.85674188,
1761.85674188, 1761.85674188, 1761.85674188, 1761.85674188,
1761.85674188, 1761.85674188, 1692.989589  , 1692.989589  ,
1692.989589  , 1684.88014476, 1684.88014476, 1744.79424188,
1688.08320052, 1730.53639476, 1730.53639476, 1721.84104764,
1752.27070052, 1752.27070052, 1752.27070052, 1687.30979764,
1673.84104764, 1673.84104764, 1691.00514476, 1691.00514476,
1691.00514476, 1691.00514476, 1691.00514476, 1691.00514476,
1691.00514476, 1669.54031832, 1649.53639476, 1649.53639476,
1649.53639476, 1649.53639476, 1649.53639476, 1649.53639476,
1617.91139476, 1617.91139476, 1617.91139476, 1617.91139476,
1597.80979764, 1596.19264476, 1596.19264476, 1596.19264476,
1654.44656832, 1620.5324712 , 1644.19264476, 1644.19264476,
1644.19264476, 1644.19264476, 1657.59693298, 1642.81764476,
1597.78443298, 1597.78443298, 1597.78443298, 1592.5637212 ,
1657.59889476, 1622.40943298, 1622.40943298, 1622.40943298,
1622.40943298, 1634.91139476, 1634.91139476, 1634.91139476,
1626.66139476, 1622.66139476, 1622.66139476, 1622.66139476,
1622.66139476, 1622.66139476, 1622.66139476, 1622.66139476,
1615.78639476, 1615.78639476, 1615.78639476, 1615.78639476,
1615.78639476, 1625.79031832, 1802.91335654, 1802.91335654,
1802.91335654, 1802.91335654, 1802.91335654, 1602.31568298,
1642.97781832, 1731.958339  , 1731.958339  , 1731.958339  ,
1731.958339  , 1731.958339  , 1731.958339  , 1731.958339  ,
1731.958339  , 1731.958339  , 1731.958339  , 1731.958339  ,
1731.958339  , 1731.958339  , 1586.27854764, 1586.27854764,
1578.0949712 , 1565.32549188, 1564.45049188, 1564.45049188,
1564.45049188, 1569.02854764, 1616.94264476, 1688.6574712 ,
1778.78639476, 1778.78639476, 1778.78639476, 1778.78639476,
1688.84889476, 1688.84889476])
```

In [45]:
```
y_pred.shape
```

Out[45]:
```
(502,)
```

Replacing the missing values with predicted values

In [46]:
```
test_data.loc[test_data.kVA.isnull(),'kVA']=y_pred
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_1764\3744497328.py:1: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_data.loc[test_data.kVA.isnull(),'kVA']=y_pred
```

In [48]:
```
test_data
```

Out[48]:

	DateTime	kVA	Min Temp	Max Temp	AvgTemp
36	4/15/2017 5:00	1612.645701	12.8	28.8	27.2
37	4/15/2017 6:00	1612.645701	12.8	28.8	27.2
38	4/15/2017 6:30	1612.645701	12.8	28.8	27.2
39	4/15/2017 7:00	1612.645701	12.8	28.8	27.2
40	4/15/2017 7:30	1612.645701	12.8	28.8	27.2
...	...	...	...	...	...
23113	4/6/2019 8:00	1778.786395	12.2	17.6	21.0
23114	4/6/2019 8:30	1778.786395	12.2	17.6	21.0
23122	4/6/2019 15:00	1778.786395	12.2	17.6	21.0
23201	4/9/2019 2:00	1688.848895	13.3	24.4	25.5
23217	4/9/2019 14:00	1688.848895	13.3	24.4	25.5

502 rows × 5 columns

Combining filled data with original dataset

In [50]:
```
new_copy=dataframe.copy()
```

In [51]:
```
new_copy.shape
```

Out[51]:
```
(23359, 5)
```

In [52]:
```
new_copy=new_copy.astype(test_data.dtypes)
```

In [53]:
```
updated_fill=new_copy.copy()
```

In [54]:
```
null_mask=new_copy.isnull()
```

In [55]:
```
for column in new_copy.columns:
    updated_fill.loc[null_mask[column],column]=test_data.loc[null_mask[column],colu
```

In [56]:
```
print("old_dataframe")
```

```
old_dataframe
```

In [57]:
```
print(new_copy)
```

```
              DateTime      kVA  Min Temp  Max Temp  AvgTemp
0       4/14/2017 1:00    726.0      12.5      28.2    20.35
1       4/14/2017 2:00    704.0      12.5      28.2    20.35
2       4/14/2017 3:00    704.0      12.5      28.2    20.35
3       4/14/2017 4:00    748.0      12.5      28.2    20.35
4       4/14/2017 5:00    814.0      12.5      28.2    20.35
...                ...      ...       ...       ...      ...
23354  4/13/2019 20:00   1474.0      16.0      29.5    22.75
23355  4/13/2019 20:30   1397.0      16.0      29.5    22.75
23356  4/13/2019 21:00   1254.0      16.0      29.5    22.75
23357  4/13/2019 22:00   1056.0      16.0      29.5    22.75
23358  4/13/2019 23:00    825.0      16.0      29.5    22.75

[23359 rows x 5 columns]
```

In [58]:
```
print('Updated dataframe')
```

```
Updated dataframe
```

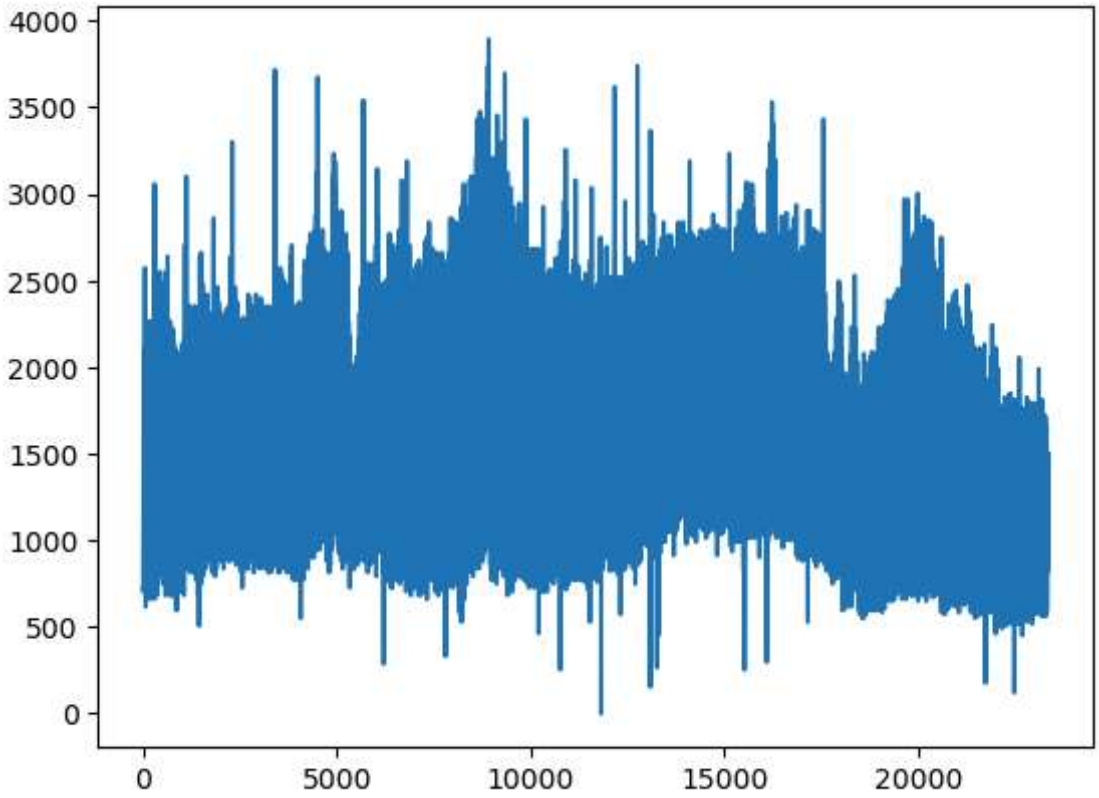
In [59]:
```
print(updated_fill)
```

```
              DateTime      kVA  Min Temp  Max Temp  AvgTemp
0       4/14/2017 1:00    726.0      12.5      28.2    20.35
1       4/14/2017 2:00    704.0      12.5      28.2    20.35
2       4/14/2017 3:00    704.0      12.5      28.2    20.35
3       4/14/2017 4:00    748.0      12.5      28.2    20.35
4       4/14/2017 5:00    814.0      12.5      28.2    20.35
...                ...      ...       ...       ...      ...
23354  4/13/2019 20:00   1474.0      16.0      29.5    22.75
23355  4/13/2019 20:30   1397.0      16.0      29.5    22.75
23356  4/13/2019 21:00   1254.0      16.0      29.5    22.75
23357  4/13/2019 22:00   1056.0      16.0      29.5    22.75
23358  4/13/2019 23:00    825.0      16.0      29.5    22.75

[23359 rows x 5 columns]
```

In [61]:
```
updated_fill.isnull().sum()
```

Out[61]:
```
DateTime    0
kVA         0
Min Temp    0
Max Temp    0
AvgTemp     0
dtype: int64
```

In [63]:
```
plt.plot(updated_fill['kVA'])
plt.show()
```



In [65]:
```
updated_fill.head()
```

Out[65]:

	DateTime	kVA	Min Temp	Max Temp	AvgTemp
0	4/14/2017 1:00	726.0	12.5	28.2	20.35
1	4/14/2017 2:00	704.0	12.5	28.2	20.35
2	4/14/2017 3:00	704.0	12.5	28.2	20.35
3	4/14/2017 4:00	748.0	12.5	28.2	20.35
4	4/14/2017 5:00	814.0	12.5	28.2	20.35

Saving/Exporting dataframe to excel

Main works started

In [50]:
```
updated_fill=updated_fill.set_index('DateTime')
```

In [51]:
```
updated_fill.head()
```

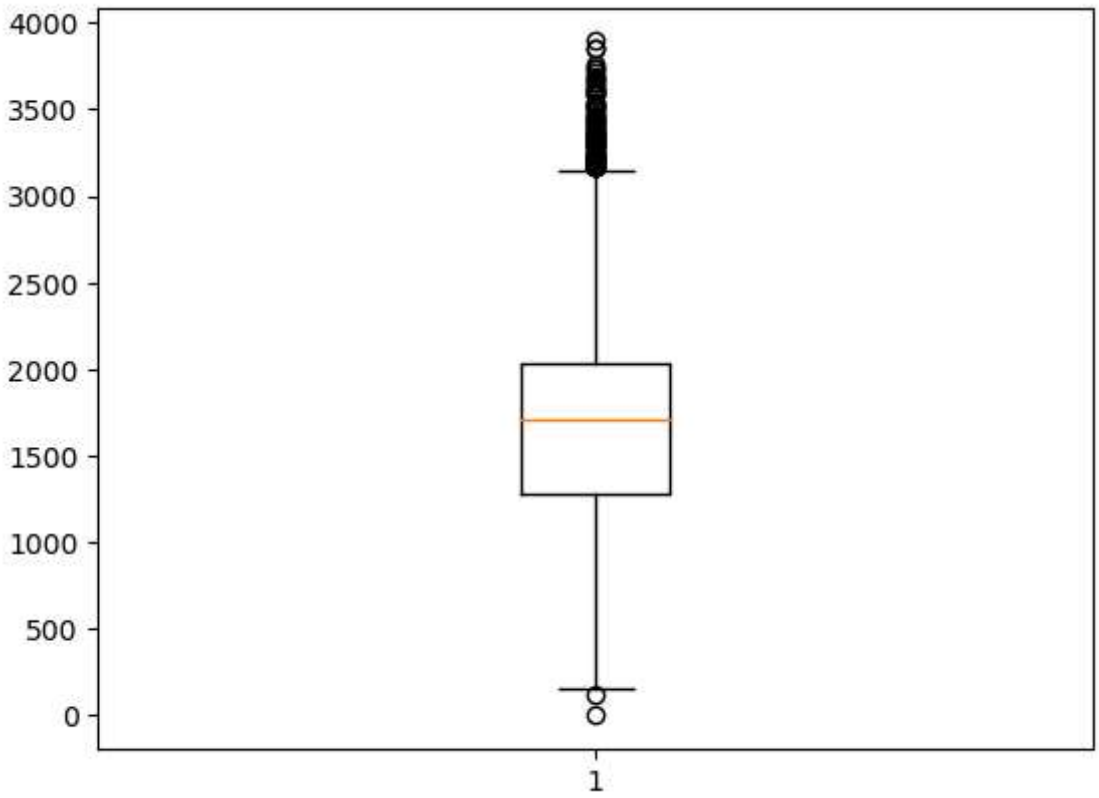
Out[51]:

DateTime	kVA	Min Temp	Max Temp	AvgTemp
4/14/2017 1:00	726.0	12.5	28.2	20.35
4/14/2017 2:00	704.0	12.5	28.2	20.35
4/14/2017 3:00	704.0	12.5	28.2	20.35
4/14/2017 4:00	748.0	12.5	28.2	20.35
4/14/2017 5:00	814.0	12.5	28.2	20.35

In [52]:
```
updated_fill.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 23359 entries, 4/14/2017 1:00 to 4/13/2019 23:00
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   kVA       23359 non-null  float64
 1   Min Temp  23359 non-null  float64
 2   Max Temp  23359 non-null  float64
 3   AvgTemp   23359 non-null  float64
dtypes: float64(4)
memory usage: 912.5+ KB
```

In [82]:
```
plt.boxplot(updated_fill['kVA'])
plt.show()
```



In [83]:
```
updated_fill.isnull().sum()
```

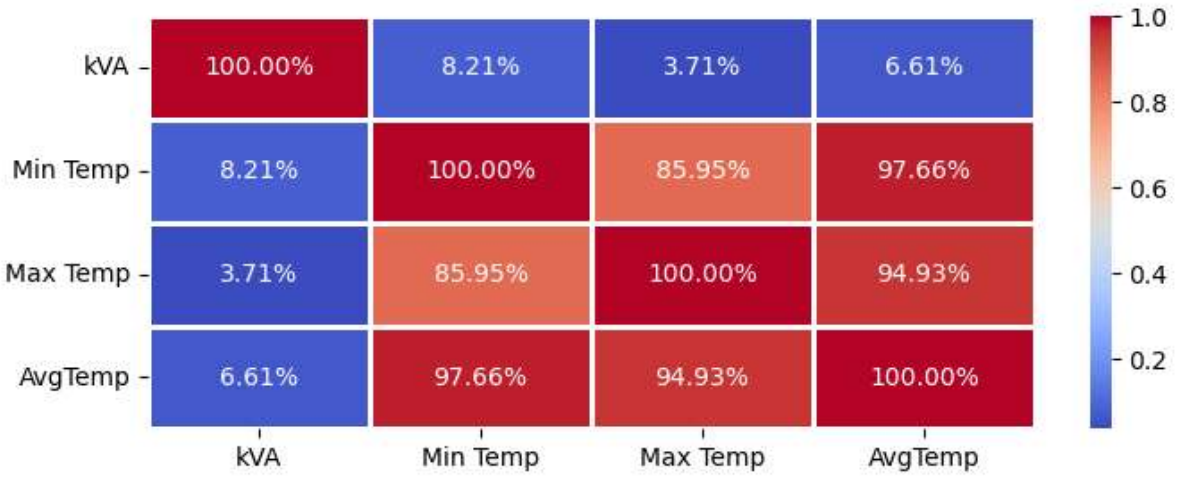
Out[83]:
```
kVA         0
Min Temp    0
Max Temp    0
AvgTemp     0
dtype: int64
```

Checking correlation between variable

In [66]:
```
plt.figure(figsize=(8,3))
sns.heatmap(updated_fill.corr(),annot=True, linewidths=1,fmt='.2%',cmap='coolwarm')
plt.xticks(rotation='horizontal')
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_1764\1192481284.py:2: FutureWarning: Th
e default value of numeric_only in DataFrame.corr is deprecated. In a future versi
on, it will default to False. Select only valid columns or specify the value of nu
meric_only to silence this warning.
  sns.heatmap(updated_fill.corr(),annot=True, linewidths=1,fmt='.2%',cmap='coolwar
m')
```

Out[66]:
```
(array([0.5, 1.5, 2.5, 3.5]),
 [Text(0.5, 0, 'kVA'),
  Text(1.5, 0, 'Min Temp'),
  Text(2.5, 0, 'Max Temp'),
  Text(3.5, 0, 'AvgTemp')])
```



Energy demand plot

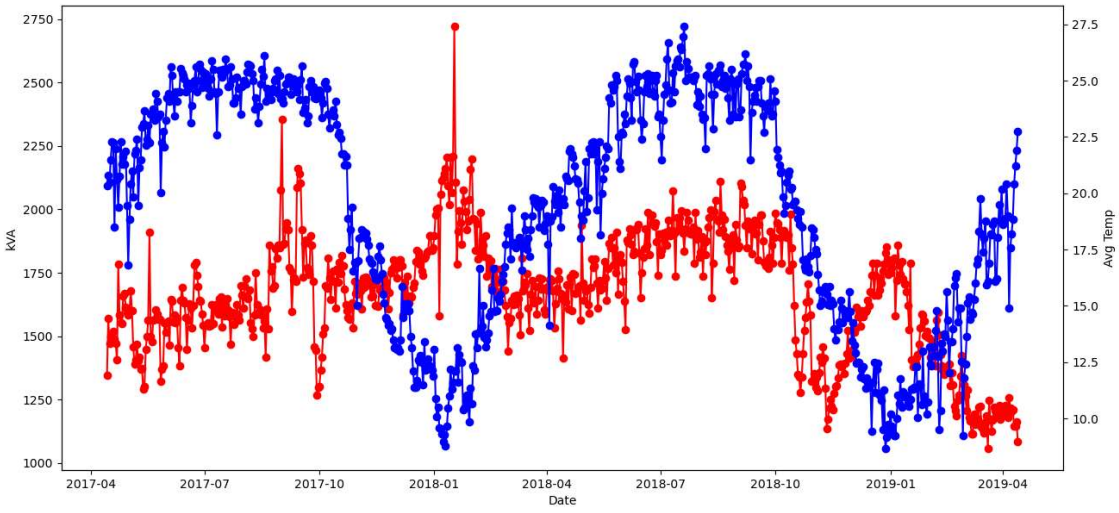
In [67]:
```
updated_fill['DateTime']=pd.to_datetime(updated_fill['DateTime'])
```

In [68]:
```
updated_fill.set_index('DateTime', inplace=True)
```

In [69]:
```
# Resample the data weekly and calculate the mean
df_sum_weekly = updated_fill['kVA'].resample('D').mean()
df_feature1 = updated_fill['AvgTemp'].resample('D').mean()
```

In [70]:
```
fig,ax=plt.subplots(figsize=(15,7))
ax.plot(df_sum_weekly.index,df_sum_weekly, color='red',marker="o",label='kVA')
ax.set_ylabel('kVA')
ax.set_xlabel('Date')
ax2=ax.twinx()
ax2.plot(df_sum_weekly.index,df_feature1, color="blue",marker="o")
ax2.set_ylabel('Avg Temp')
fig.legend('Resampled Weekly energy demand','Resampled weekly min temperature',loc=
fig.show()
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_1764\3888261947.py:8: UserWarning: Lege
nd does not support handles for str instances.
A proxy artist may be used instead.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#contro
lling-the-legend-entries
  fig.legend('Resampled Weekly energy demand','Resampled weekly min temperature',l
oc='lower right')
C:\Users\Dell\AppData\Local\Temp\ipykernel_1764\3888261947.py:9: UserWarning: Matp
lotlib is currently using module://matplotlib_inline.backend_inline, which is a no
n-GUI backend, so cannot show the figure.
  fig.show()
```



In [56]:
```
updated_fill['AvgTemp'].max()
```

Out[56]:
```
27.4
```

In [125…
```
filename='Final_filled.xlsx'
updated_fill.to_excel(filename)
```

In [126…
```
 updated_fill= updated_fill.applymap(lambda x: str(x).encode('utf-8').decode('utf-8
```

In [129…
```
updated_fill.isnull().sum()
```

Out[129]:
```
kVA          0
Min Temp     0
Max Temp     0
AvgTemp      0
dtype: int64
```

In [57]:
```
updated_fill.tail()
```

Out[57]:

DateTime	kVA	Min Temp	Max Temp	AvgTemp
4/13/2019 20:00	1474.0	16.0	29.5	22.75
4/13/2019 20:30	1397.0	16.0	29.5	22.75
4/13/2019 21:00	1254.0	16.0	29.5	22.75
4/13/2019 22:00	1056.0	16.0	29.5	22.75
4/13/2019 23:00	825.0	16.0	29.5	22.75

In [140…
```
updated_fill.head()
```

Out[140]:

DateTime	kVA	Min Temp	Max Temp	AvgTemp
2017-04-14 01:00:00	726.0	12.5	28.2	20.35
2017-04-14 02:00:00	704.0	12.5	28.2	20.35
2017-04-14 03:00:00	704.0	12.5	28.2	20.35
2017-04-14 04:00:00	748.0	12.5	28.2	20.35
2017-04-14 05:00:00	814.0	12.5	28.2	20.35

In [72]:
```
weather=pd.read_csv('H:\\ML\\energyforecast\\weather_data.csv',usecols=['date','tav
weather.head()
```

Out[72]:

	date	tavg	tmin	prcp	wspd
0	2017-04-14	20.4	12.5	0.0	6.1
1	2017-04-15	17.3	12.8	2.8	5.8
2	2017-04-16	20.4	12.5	0.0	5.6
3	2017-04-17	21.2	13.4	0.0	5.4
4	2017-04-18	22.2	15.8	NaN	6.2

In [73]:
```
weather.isnull().sum()
```

Out[73]:
```
date      0
tavg      0
tmin      0
prcp    113
wspd      5
dtype: int64
```

In [74]:
```
weather['tavg'].shape
```

Out[74]:
```
(730,)
```

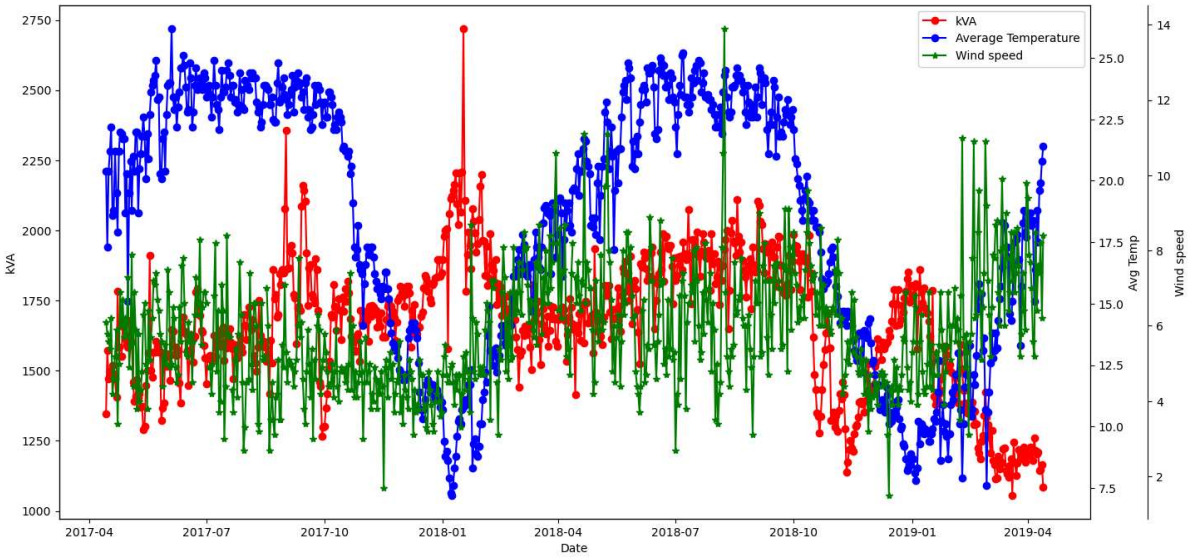
In [75]:
```
weather['wspd'].isnull().sum()
```

Out[75]:
```
5
```

In [76]:
```
df_sum_weekly = updated_fill['kVA'].resample('D').mean()
df_feature1 = weather['tavg']#.resample('D').mean()
df_feature2=weather['wspd']
```

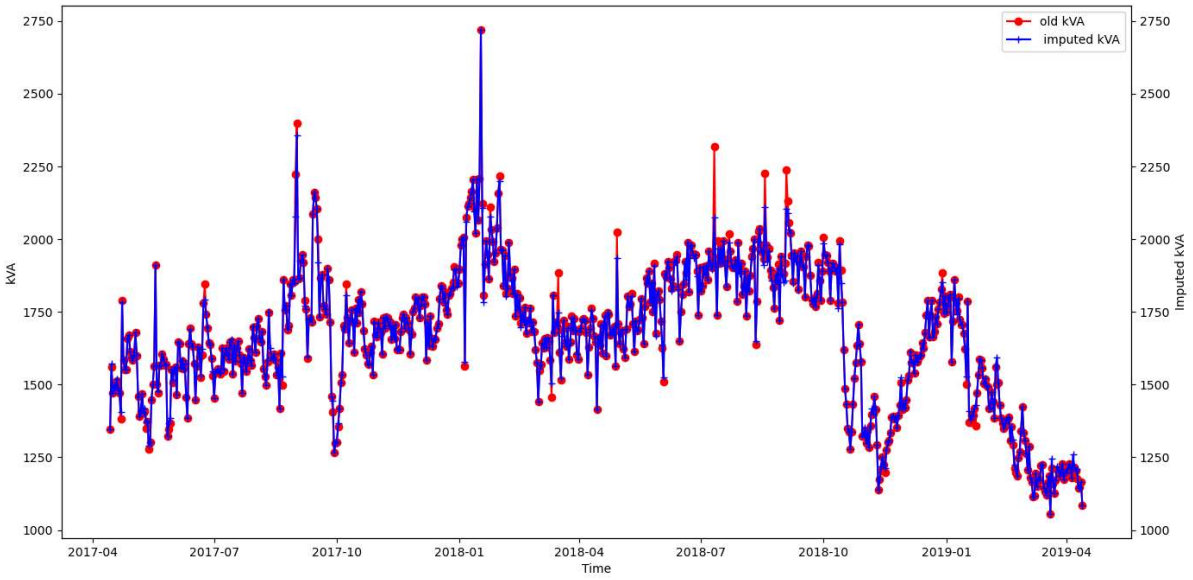
In [77]:
```
# setting up the environment
fig,ax=plt.subplots(figsize=(16,8))
# plotting the first graph in primary axis
ax.plot(df_sum_weekly.index,df_sum_weekly, color='red',marker="o",label='kVA')
ax.set_ylabel('kVA')
ax.set_xlabel('Date')
#ax.grid('True')
# making second plot
ax2=ax.twinx()
ax2.plot(df_sum_weekly.index,df_feature1, color="blue",marker="o",label='Average Te
ax2.set_ylabel('Avg Temp')

# Plotting the data on series
```

```
ax3=ax.twinx()
ax3.plot(df_sum_weekly.index,df_feature2,color="green",marker="*",label='Wind speed
ax3.set_ylabel('Wind speed')

ax3.spines['right'].set_position(('outward',50))

lines_1, labels_1 = ax.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
lines_3, labels_3 = ax3.get_legend_handles_labels()
ax.legend(lines_1 + lines_2 + lines_3, labels_1 + labels_2 + labels_3, loc='upper r

#fig.legend('Resampled Weekly energy demand','Resampled weekly min temperature',loc
plt.show()
```



In [125…

```
df_sum_weekly=dataframe['kVA'].resample('D').mean()
df_f1=updated_fill['kVA'].resample('D').mean()
fig,ax=plt.subplots(figsize=(16,8))
ax.plot(df_sum_weekly.index,df_sum_weekly, color='red',marker="o",label='old kVA ')
ax.set_xlabel('Time')
ax.set_ylabel('kVA')

ax2=ax.twinx()
ax2.plot(df_sum_weekly.index,df_f1, color='blue',marker="+", label=' imputed kVA')
ax2.set_ylabel('Imputed kVA')

lines_1, labels_1 = ax.get_legend_handles_labels()
lines_2, labels_2 = ax2.get_legend_handles_labels()
#lines_3, labels_3 = ax3.get_legend_handles_labels()
ax.legend(lines_1 + lines_2 + lines_3, labels_1 + labels_2,loc='upper right')

plt.show()
```



In [102…
```
updated_fill.shape
```

Out[102]:
```
(23359, 4)
```

Copy_updated_fill_to_newdataframe

In [104…
```
new_dataframe=updated_fill.copy()
```

In [105…
```
new_dataframe.shape
```

Out[105]:
```
(23359, 4)
```

In [168…
```
new_dataframe.head()
```

Out[168]:

DateTime	kVA	Min Temp	Max Temp	AvgTemp
2017-04-14 01:00:00	726.0	12.5	28.2	20.35
2017-04-14 02:00:00	704.0	12.5	28.2	20.35
2017-04-14 03:00:00	704.0	12.5	28.2	20.35
2017-04-14 04:00:00	748.0	12.5	28.2	20.35
2017-04-14 05:00:00	814.0	12.5	28.2	20.35

In [109…
```
new_dataframe.columns
```

Out[109]:
```
Index(['kVA', 'Min Temp', 'Max Temp', 'AvgTemp'], dtype='object')
```

In [110…
```
features1=new_dataframe[['Min Temp','Max Temp','AvgTemp']]
```

In [113…
```
features1.shape
```

Out[113]:
```
(23359, 3)
```

In [114…
```
demand1=new_dataframe['kVA']
```

In [116…
```
demand1.shape[0]
```

Out[116]:
```
23359
```

Creating ML model SVM

In [117…
```
new_dataframe['Min Temp'].isnull().sum()
```

Out[117]:
```
0
```

In [118…
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(features1,demand1,test_size=0.2,rand
```

In [120…
```
y_train.shape
```

Out[120]:
```
(18687,)
```

In [121…
```
y_test.shape
```

Out[121]:
```
(4672,)
```

In [124…
```
y_train
```

Out[124]:
```
DateTime
2017-09-19 07:00:00    1650.0
2018-02-07 18:30:00    2662.0
2017-10-01 18:30:00    1782.0
2017-10-06 07:30:00    1650.0
2017-04-23 19:00:00    2266.0
                        ...
2018-05-29 04:00:00    1045.0
2018-12-19 01:00:00     737.0
2018-02-15 17:30:00    1848.0
2018-03-17 12:00:00    1760.0
2017-07-08 09:30:00    1870.0
Name: kVA, Length: 18687, dtype: float64
```

In [126…
```
## Converting two dimensional array to one d array
```

In [130…
```
y_train=y_train.ravel()
```

In [131…
```
y_train
```

Out[131]:
```
array([1650., 2662., 1782., ..., 1848., 1760., 1870.])
```

In [132…
```
y_train.ndim  # dimension has been changed
```

Out[132]:
```
1
```

In [133…
```
# similary changing test dimension to 1 d
```

In [134…
```
y_test
```

Out[134]:
```
DateTime
2019-03-29 07:00:00    1364.0
2018-03-11 03:00:00     748.0
2018-08-13 05:00:00    1144.0
2018-05-18 07:00:00    1727.0
2017-08-19 05:00:00     682.0
                        ...
2018-10-02 14:00:00    2112.0
2019-02-14 09:30:00    1474.0
2018-04-26 06:00:00    1320.0
2018-03-17 08:30:00    2002.0
2018-11-18 06:00:00    1045.0
Name: kVA, Length: 4672, dtype: float64
```

In [135…
```
y_test.ndim
```

Out[135]:
```
1
```

In [136…
```
y_test=y_test.ravel()
```

In [137…
```
y_test
```

Out[137]:
```
array([1364.,  748., 1144., ..., 1320., 2002., 1045.])
```

Importing SVR: Support vector Regression model in

In [139…
```
# since we already imported the regessor in preamble we now make tha object of the
```

In [142…
```
supvreg= SVR(kernel='rbf') # for non-linear regression
supvreg.fit(X_train,y_train)
```

Out[142]:
```
SVR
SVR()
```

Predicting kVA on the basis of only training data that we fed

In [144…
```
predictedkVA_train=supvreg.predict(X_train)
```

In [145…
```
predictedkVA_train
```

Out[145]:
```
array([1770.2962611 , 1628.45438638, 1781.53185729, ..., 1590.38721826,
       1568.38156369, 1796.93741921])
```

In [146…
```
predictedkVA_train.shape
```

Out[146]:
```
(18687,)
```

accuracy score

In [157…
```
# since it is a continuos data we cannot use the accuracy_score becoz it is only fo
```

In [160…
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

In [161…
```
print(r2_score(y_train,predictedkVA_train))
print(mean_squared_error(y_train,predictedkVA_train))
```

```
0.032067277046605835
285187.8027602161
```

We have not normalized the data of kVA so there's a place to improve the score

In [165…
```
#importing standand scaler model
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
```

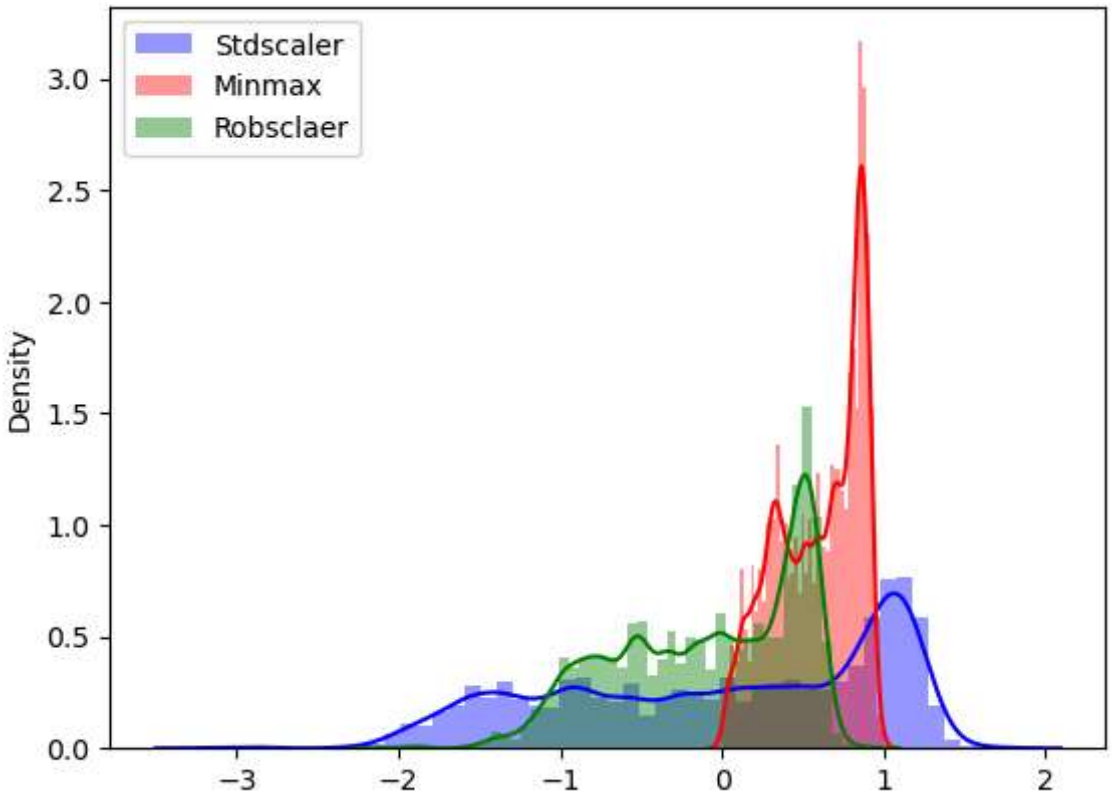
In [163…
```
# creating instance of object and checking which scaler gives us optimum value
```

In [164…
```
scl1=StandardScaler()
scl2=MinMaxScaler()
scl3=RobustScaler()
```

In [167…
```
feat1=scl1.fit_transform(features1)
feat2=scl2.fit_transform(features1)
feat3=scl3.fit_transform(features1)
```

In [176…
```
sns.distplot(feat1,color='blue',label='Stdscaler',kde=True)
sns.distplot(feat2,color='red',label='Minmax',kde=True)
sns.distplot(feat3,color='green',label='Robsclaer',kde=True)
plt.legend()
plt.show()
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_1764\708105003.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feat1,color='blue',label='Stdscaler',kde=True)
C:\Users\Dell\AppData\Local\Temp\ipykernel_1764\708105003.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feat2,color='red',label='Minmax',kde=True)
C:\Users\Dell\AppData\Local\Temp\ipykernel_1764\708105003.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feat3,color='green',label='Robsclaer',kde=True)
```



Testing each scaler in training and testing from above scaled

In [197…
```
Xstd_train,Xstd_test,ystd_train,ystd_test=train_test_split(feat1,demand1,test_size=
```

In [198…
```
ystd_train
```

Out[198]:
```
DateTime
2017-09-19 07:00:00    1650.0
2018-02-07 18:30:00    2662.0
2017-10-01 18:30:00    1782.0
2017-10-06 07:30:00    1650.0
2017-04-23 19:00:00    2266.0
                        ...
2018-05-29 04:00:00    1045.0
2018-12-19 01:00:00     737.0
2018-02-15 17:30:00    1848.0
2018-03-17 12:00:00    1760.0
2017-07-08 09:30:00    1870.0
Name: kVA, Length: 18687, dtype: float64
```

In [199…
```
ystd_train=ystd_train.values.ravel()
```

In [200…
```
ystd_train
```

Out[200]:
```
array([1650., 2662., 1782., ..., 1848., 1760., 1870.])
```

In [201…
```
ystd_test
```

Out[201]:
```
DateTime
2019-03-29 07:00:00    1364.0
2018-03-11 03:00:00     748.0
2018-08-13 05:00:00    1144.0
2018-05-18 07:00:00    1727.0
2017-08-19 05:00:00     682.0
                        ...
2018-10-02 14:00:00    2112.0
2019-02-14 09:30:00    1474.0
2018-04-26 06:00:00    1320.0
2018-03-17 08:30:00    2002.0
2018-11-18 06:00:00    1045.0
Name: kVA, Length: 4672, dtype: float64
```

In [202…
```
ystd_test=ystd_test.values.ravel()
```

In [203…
```
ystd_test
```

Out[203]:
```
array([1364.,  748., 1144., ..., 1320., 2002., 1045.])
```

again fitting the regressor model after normalize or standarize or minmax scaler

In [204…
```
supvreg=SVR(kernel='rbf')
supvreg=supvreg.fit(Xstd_train,ystd_train) # X_train chai feat1 vitra cha
supvreg
```

Out[204]:
```
▾ SVR
SVR()
```

In [217…
```
supvreg.fit(Xstd_train,ystd_train)
forecasted_train=supvreg.predict(Xstd_train)
```

```
forecasted_train
```

Out[217]:
```
array([1765.68170715, 1622.35739515, 1781.0223144 , ..., 1563.43874812,
       1562.34765637, 1797.92318306])
```

In [206…
```
## testing the model accuracy again
print(r2_score(ystd_train,forecasted_train))
print(mean_squared_error(ystd_train,forecasted_train))
```

```
0.03375359985225512
284690.951394121
```

In [207…
```
## predcit on test data
test_predict=supvreg.predict(Xstd_test)
print(r2_score(ystd_test,test_predict))
print(mean_squared_error(ystd_test,test_predict))
```

```
0.03610924309412644
280555.03988106997
```

Minmax scaler training and testing

In [209…
```
Xmm_train,Xmm_test,ymm_train,ymm_test=train_test_split(feat2,demand1,test_size=0.2,
ymm_train.shape
```

Out[209]:
```
(18687,)
```

In [210…
```
ymm_test.shape
```

Out[210]:
```
(4672,)
```

In [211…
```
ymm_train=ymm_train.values.ravel()
ymm_test=ymm_test.values.ravel()
```

In [212…
```
ymm_train
```

Out[212]:
```
array([1650., 2662., 1782., ..., 1848., 1760., 1870.])
```

In [214…
```
supvreg=SVR(kernel='rbf')
supvreg=supvreg.fit(Xmm_train,ymm_train)
```

In [215…
```
### Predicting
forecasted_train2=supvreg.predict(Xmm_train)
```

In [216…
```
forecasted_train2
```

Out[216]:
```
array([1769.05980841, 1619.36825171, 1782.29167946, ..., 1572.8280361 ,
       1558.78450654, 1797.68111173])
```

In [219…
```
print(r2_score(ymm_train,forecasted_train2))
print(mean_squared_error(ymm_train,forecasted_train2))
```

```
0.034075882563346815
284595.9953129138
```

Testing different fitting algorithm for accuracy improvement

In [221…
```
X1_train,X1_test,y1_train,y1_test=train_test_split(features1,demand1,test_size=0.2,
```

In [222…
```
y1_train=y1_train.values.ravel()
```

In [223…
```
y1_train
```

Out[223]:
```
array([ 726.,  704.,  704., ..., 1463., 1111., 1056.])
```

In [224…
```
y1_test=y1_test.values.ravel()
```

In [225…
```
reg1=SVR(kernel='poly',degree=5)
```

In [226…
```
reg1.fit(X1_train,y1_train)
```

Out[226]:
```
SVR(degree=5, kernel='poly')
```

In [227…
```
prediction=reg1.predict(X1_train)
```

In [228…
```
print(r2_score(y1_train,prediction))
```

```
0.009461935202173644
```

In [229…
```
print(mean_squared_error(y1_train,prediction))
```

```
279836.4956017564
```

In [230…
```
prediction2=reg1.predict(X1_test)
```

In [231…
```
prediction2
```

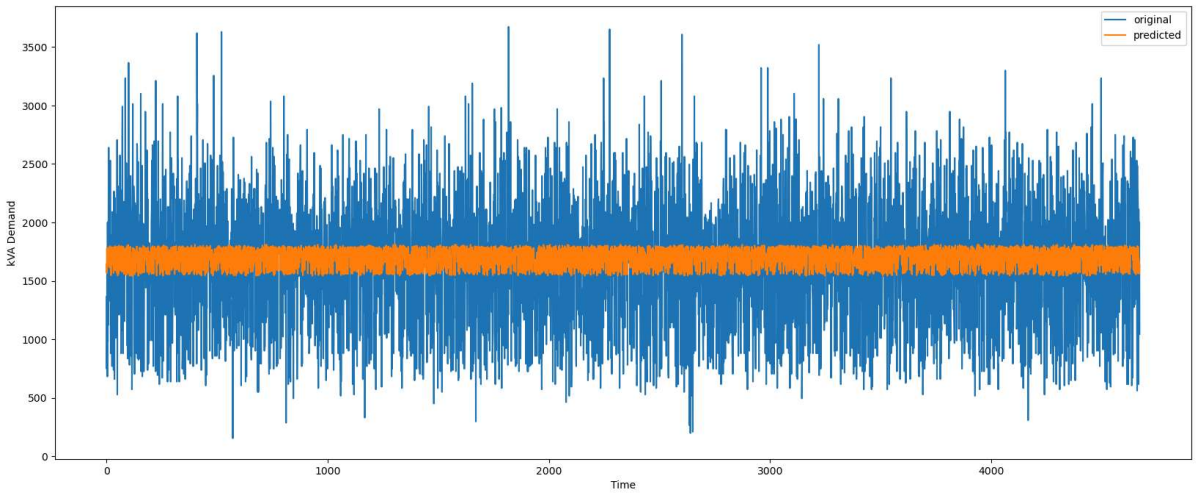
Out[231]:
```
array([1753.55398897, 1753.55398897, 1753.55398897, ..., 1711.84292948,
       1711.84292948, 1711.84292948])
```

In [233…
```
print(r2_score(y1_test,prediction2))
```

```
-0.41752665262899735
```

In [234…
```
## Plotting to see the predcited data
```

In [236…
```
plt.figure(figsize=(20,8))
plt.plot(ystd_test,label='original')
plt.plot(test_predict,label='predicted')
plt.legend(loc='best')
plt.xlabel('Time')
plt.ylabel('kVA Demand')
plt.show()
```


original
predicted
kVA Demand
Time
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000

In [ ]: