```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
from google.colab import files
```

```
uploaded = files.upload()
```

```
Choose Files   delhi_elect...emapnd.csv
delhi_electricity_demapnd.csv(text/csv) - 2687390 bytes, last modified: n/a - 100% done
Saving delhi_electricity_demapnd.csv to delhi_electricity_demapnd.csv
```

```
data = pd.read_csv('delhi_electricity_demapnd.csv')
```

```
data.shape
```

```
(43848, 9)
```

```
data.head(-10)
```

```
       Timestamp  hour  dayofweek  month    year  dayofyear  Temperature  Humi
    0  01-Jan-20   0.0        2.0    1.0  2020.0        1.0     3.000000  61.28
    1  01-Jan-20   1.0        2.0    1.0  2020.0        1.0     3.000000  52.87
    2  01-Jan-20   2.0        2.0    1.0  2020.0        1.0     4.244482  36.34
    3  01-Jan-20   3.0        2.0    1.0  2020.0        1.0     3.000000  72.62
    4  01-Jan-20   4.0        2.0    1.0  2020.0        1.0     3.881208  90.58
  ...        ...   ...        ...    ...     ...        ...          ...
43833  31-Dec-24   9.0        1.0   12.0  2024.0      366.0    16.896276  72.38
43834  31-Dec-24  10.0        1.0   12.0  2024.0      366.0    21.984042  77.27
43835  31-Dec-24  11.0        1.0   12.0  2024.0      366.0    20.659219  69.22
43836  31-Dec-24  12.0        1.0   12.0  2024.0      366.0    19.112534  52.68
43837  31-Dec-24  13.0        1.0   12.0  2024.0      366.0    17.862808  64.74

43838 rows × 9 columns
```

Next steps: Generate code with data   View recommended plots   New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43848 entries, 0 to 43847
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Timestamp    43848 non-null  object 
 1   hour         43837 non-null  float64
 2   dayofweek    43839 non-null  float64
 3   month        43840 non-null  float64
 4   year         43843 non-null  float64
 5   dayofyear    43843 non-null  float64
 6   Temperature  43841 non-null  float64
 7   Humidity     43838 non-null  float64
 8   Demand       43841 non-null  float64
dtypes: float64(8), object(1)
memory usage: 3.0+ MB
```

the **Timestamp** column was initially a an object data type and below we are converting it to *datetime* format.

```
data['Timestamp'] = pd.to_datetime(data['Timestamp'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43848 entries, 0 to 43847
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   Timestamp    43848 non-null  datetime64[ns]
 1   hour         43837 non-null  float64       
 2   dayofweek    43839 non-null  float64       
 3   month        43840 non-null  float64       
 4   year         43843 non-null  float64       
 5   dayofyear    43843 non-null  float64       
 6   Temperature  43841 non-null  float64       
 7   Humidity     43838 non-null  float64       
 8   Demand       43841 non-null  float64       
dtypes: datetime64[ns](1), float64(8)
memory usage: 3.0 MB
/tmp/ipython-input-7-3315307892.py:1: UserWarning: Could not infer format,
  data['Timestamp'] = pd.to_datetime(data['Timestamp'])
```

```
data = data.set_index('Timestamp')
data
```

```
            hour  dayofweek  month    year  dayofyear  Temperature   Humidity
Timestamp
2020-01-01   0.0        2.0    1.0  2020.0        1.0     3.000000  61.288951  2
2020-01-01   1.0        2.0    1.0  2020.0        1.0     3.000000  52.873702  2
2020-01-01   2.0        2.0    1.0  2020.0        1.0     4.244482  36.341783  2
2020-01-01   3.0        2.0    1.0  2020.0        1.0     3.000000  72.629378  2
2020-01-01   4.0        2.0    1.0  2020.0        1.0     3.881208  90.582444  2
    ...      ...        ...    ...     ...        ...          ...        ...
2024-12-31  19.0        1.0   12.0  2024.0      366.0     3.956838  43.287161  4
2024-12-31  20.0        1.0   12.0  2024.0      366.0     3.118824  51.705756  4
2024-12-31  21.0        1.0   12.0  2024.0      366.0     3.000000  40.565916  4
2024-12-31  22.0        1.0   12.0  2024.0      366.0     3.000000  51.998107  3
2024-12-31  23.0        1.0   12.0  2024.0      366.0     6.037472  59.931925  3
43848 rows × 8 columns
```

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data[['Temperature', 'Humidity', 'Demand']].describe()
```

```
        Temperature      Humidity        Demand
count  43841.000000  43838.000000  43841.000000
mean      25.067788     59.903007   5000.790976
std       12.821725     18.342604   1412.527409
min        3.000000     20.000000   1611.954020
25%       15.210186     46.241224   4015.668472
50%       25.003212     59.986720   5013.053367
75%       34.740971     73.796820   6000.803082
max       50.000000     95.000000  11910.705100
```

Dealing with Missing Data

```
print(data.isnull().sum())
```

```
hour           11
dayofweek       9
month           8
year            5
dayofyear       5
Temperature     7
Humidity       10
Demand          7
dtype: int64
```

```
data[data.isna().any(axis=1)]
```

	hour	dayofweek	month	year	dayofyear	Temperature	Humidity
Timestamp							
2020-04-30	NaN	3.0	4.0	2020.0	121.0	21.820261	41.353675
2020-07-21	NaN	1.0	7.0	2020.0	203.0	36.555833	62.779665
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2021-02-27	NaN	5.0	2.0	2021.0	58.0	24.001003	53.303268
2021-12-13	23.0	0.0	12.0	2021.0	NaN	3.000000	75.457130
2021-12-14	NaN	1.0	12.0	2021.0	348.0	5.061319	47.215825
2021-12-14	14.0	1.0	12.0	2021.0	348.0	15.403995	74.011557
2022-06-17	13.0	4.0	6.0	2022.0	168.0	NaN	43.171391
2022-06-17	NaN	4.0	6.0	2022.0	168.0	42.924693	43.645711
2022-06-17	17.0	4.0	NaN	2022.0	168.0	37.115634	37.554842
2022-06-17	18.0	4.0	6.0	2022.0	168.0	39.886471	NaN
2022-06-17	21.0	NaN	6.0	2022.0	168.0	28.471494	54.964101
2022-06-18	5.0	5.0	6.0	2022.0	169.0	38.349174	58.285814
2023-08-30	NaN	2.0	8.0	2023.0	242.0	25.707715	69.868663
2023-08-31	0.0	NaN	8.0	2023.0	243.0	22.705122	81.691817

2023-08-31	5.0	3.0	NaN	2023.0	243.0	29.081792	81.270824
2023-08-31	11.0	3.0	8.0	NaN	243.0	44.270455	75.563954
2023-08-31	14.0	3.0	8.0	2023.0	243.0	NaN	NaN
2024-01-27	19.0	5.0	1.0	2024.0	27.0	14.351975	NaN
2024-01-28	2.0	NaN	1.0	2024.0	28.0	3.000000	68.899943
2024-01-28	NaN	6.0	1.0	2024.0	28.0	9.081186	78.163106
2024-09-24	19.0	NaN	9.0	2024.0	268.0	25.632052	74.123843
2024-09-24	21.0	1.0	9.0	2024.0	268.0	19.380978	NaN
2024-09-24	22.0	1.0	9.0	2024.0	268.0	NaN	66.618690
2024-09-25	7.0	2.0	NaN	2024.0	269.0	31.190258	95.000000
2024-12-18	11.0	2.0	12.0	2024.0	353.0	23.075591	NaN
2024-12-18	12.0	2.0	12.0	2024.0	353.0	20.012449	NaN
2024-12-18	21.0	NaN	NaN	2024.0	353.0	3.000000	64.041109

```
data[data.isna().all(axis=1)]
##this shows the subset of the data where all the values in each of the columns
```

	hour	dayofweek	month	year	dayofyear	Temperature	Humidity	De
Timestamp								
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2021-02-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

As there is no scope of trying to work our way around with the data where no entries are present, it would be a wise decision to drop these rows of data.

Below we are dropping all the rows which had no data in them at all!

```
data = data.dropna(how='all')
data
```

	hour	dayofweek	month	year	dayofyear	Temperature	Humidity	
Timestamp								
2020-01-01	0.0	2.0	1.0	2020.0	1.0	3.000000	61.288951	2
2020-01-01	1.0	2.0	1.0	2020.0	1.0	3.000000	52.873702	2
2020-01-01	2.0	2.0	1.0	2020.0	1.0	4.244482	36.341783	2
2020-01-01	3.0	2.0	1.0	2020.0	1.0	3.000000	72.629378	2
2020-01-01	4.0	2.0	1.0	2020.0	1.0	3.881208	90.582444	2
...	...	...	...	...	...	...	...	
2024-12-31	19.0	1.0	12.0	2024.0	366.0	3.956838	43.287161	4
2024-12-31	20.0	1.0	12.0	2024.0	366.0	3.118824	51.705756	4
2024-12-31	21.0	1.0	12.0	2024.0	366.0	3.000000	40.565916	4
2024-12-31	22.0	1.0	12.0	2024.0	366.0	3.000000	51.998107	3
2024-12-31	23.0	1.0	12.0	2024.0	366.0	6.037472	59.931925	3

43844 rows × 8 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

Now to fill in the remaining empty values we can use bfill and ffill

```
data[['hour', 'dayofweek', 'month', 'year', 'dayofyear']] = data[['hour', 'dayc
```

```
/tmp/ipython-input-14-3402763247.py:1: SettingWithCopyWarning:
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
  data[['hour', 'dayofweek', 'month', 'year', 'dayofyear']] = data[['hour',
data[['Temperature','Humidity']] = data[['Temperature','Humidity']].bfill()
```

```
/tmp/ipython-input-15-2229112482.py:1: SettingWithCopyWarning: 
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
  data[['Temperature','Humidity']] = data[['Temperature','Humidity']].bfill
```

```
data['Demand'] = data['Demand'].interpolate(method='time')
```

```
/tmp/ipython-input-16-3652729601.py:1: SettingWithCopyWarning: 
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
  data['Demand'] = data['Demand'].interpolate(method='time')
```

```
data.isnull().sum()
```

```
                0
hour            0
dayofweek       0
month           0
year            0
dayofyear       0
Temperature     0
Humidity        0
Demand          0

dtype: int64
```

Feature Engineering

```
data
```

	hour	dayofweek	month	year	dayofyear	Temperature	Humidity	
Timestamp								
2020-01-01	0.0	2.0	1.0	2020.0	1.0	3.000000	61.288951	2
2020-01-01	1.0	2.0	1.0	2020.0	1.0	3.000000	52.873702	2
2020-01-01	2.0	2.0	1.0	2020.0	1.0	4.244482	36.341783	2
2020-01-01	3.0	2.0	1.0	2020.0	1.0	3.000000	72.629378	2
2020-01-01	4.0	2.0	1.0	2020.0	1.0	3.881208	90.582444	2
...	...	...	...	...	...	...	...	
2024-12-31	19.0	1.0	12.0	2024.0	366.0	3.956838	43.287161	4
2024-12-31	20.0	1.0	12.0	2024.0	366.0	3.118824	51.705756	4
2024-12-31	21.0	1.0	12.0	2024.0	366.0	3.000000	40.565916	4
2024-12-31	22.0	1.0	12.0	2024.0	366.0	3.000000	51.998107	3
2024-12-31	23.0	1.0	12.0	2024.0	366.0	6.037472	59.931925	3

43844 rows × 8 columns

Next steps: Generate code with `data` View recommended plots New interactive sheet

```
data.insert(5, 'quarter', data.index.quarter)
data
```

	hour	dayofweek	month	year	dayofyear	quarter	Temperature	Hu
Timestamp								
2020-01-01	0.0	2.0	1.0	2020.0	1.0	1	3.000000	61
2020-01-01	1.0	2.0	1.0	2020.0	1.0	1	3.000000	52
2020-01-01	2.0	2.0	1.0	2020.0	1.0	1	4.244482	36
2020-01-01	3.0	2.0	1.0	2020.0	1.0	1	3.000000	72
2020-01-01	4.0	2.0	1.0	2020.0	1.0	1	3.881208	90
...	...	...	...	...	...	...	...	
2024-12-31	19.0	1.0	12.0	2024.0	366.0	4	3.956838	43
2024-12-31	20.0	1.0	12.0	2024.0	366.0	4	3.118824	51
2024-12-31	21.0	1.0	12.0	2024.0	366.0	4	3.000000	40
2024-12-31	22.0	1.0	12.0	2024.0	366.0	4	3.000000	51
2024-12-31	23.0	1.0	12.0	2024.0	366.0	4	6.037472	59

43844 rows × 9 columns

Next steps: Generate code with data View recommended plots New interactive sheet

```
data[['hour', 'dayofweek', 'month', 'year', 'dayofyear']] = data[['hour', 'dayc
```

```
/tmp/ipython-input-20-3222519661.py:1: SettingWithCopyWarning:
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
  data[['hour', 'dayofweek', 'month', 'year', 'dayofyear']] = data[['hour',
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 43844 entries, 2020-01-01 to 2024-12-31
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   hour         43844 non-null  int64  
 1   dayofweek    43844 non-null  int64  
 2   month        43844 non-null  int64  
 3   year         43844 non-null  int64  
 4   dayofyear    43844 non-null  int64  
 5   quarter      43844 non-null  int32  
 6   Temperature  43844 non-null  float64
 7   Humidity     43844 non-null  float64
 8   Demand       43844 non-null  float64
dtypes: float64(3), int32(1), int64(5)
memory usage: 3.2 MB
```

```
data.insert(5, 'weekofyear', data.index.isocalendar().week.astype(int))
```

```
data
```

	hour	dayofweek	month	year	dayofyear	weekofyear	quarter	Temp
Timestamp								
2020-01-01	0	2	1	2020	1	1	1	
2020-01-01	1	2	1	2020	1	1	1	
2020-01-01	2	2	1	2020	1	1	1	
2020-01-01	3	2	1	2020	1	1	1	
2020-01-01	4	2	1	2020	1	1	1	
...	...	...	...	...	...	...	...	
2024-12-31	19	1	12	2024	366	1	4	
2024-12-31	20	1	12	2024	366	1	4	
2024-12-31	21	1	12	2024	366	1	4	
2024-12-31	22	1	12	2024	366	1	4	
2024-12-31	23	1	12	2024	366	1	4	

43844 rows × 10 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.insert(7, 'is_weekend' , data.index.dayofweek.isin([5,6]).astype(int))
```

```
data
```

	hour	dayofweek	month	year	dayofyear	weekofyear	quarter	is_w
Timestamp								
2020-01-01	0	2	1	2020	1	1	1	
2020-01-01	1	2	1	2020	1	1	1	
2020-01-01	2	2	1	2020	1	1	1	
2020-01-01	3	2	1	2020	1	1	1	
2020-01-01	4	2	1	2020	1	1	1	
...	...	...	...	...	...	...	...	
2024-12-31	19	1	12	2024	366	1	4	
2024-12-31	20	1	12	2024	366	1	4	
2024-12-31	21	1	12	2024	366	1	4	
2024-12-31	22	1	12	2024	366	1	4	
2024-12-31	23	1	12	2024	366	1	4	

43844 rows × 11 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

Also we will check if there was any public holiday during the timestamp of the dataset

```
import holidays
```

```
data['holidays'] = holidays.IN(years = data.year)
```

```
/tmp/ipython-input-27-2470672226.py:1: SettingWithCopyWarning:
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
  data['holidays'] = holidays.IN(years = data.year)
```

```
data.holidays.value_counts()
```

```
                         count
        holidays
----------------------------------
     Republic Day          120
   Maha Shivaratri         120
   Mahavir Jayanti         120
     Good Friday           120
   Buddha Purnima          120
     Eid al-Fitr           120
     Eid al-Adha           120
     Janmashtami           120
  Independence Day         120
       Ashura              120
   Gandhi Jayanti          120
      Dussehra             120
  Prophet's Birthday       120
       Diwali              120
 Guru Nanak Jayanti        120
      Christmas            120

dtype: int64
```

```
data = data.drop(columns = ['holidays'])
```

```
data['Demand_lag_24hr'] = data['Demand'].shift(24)
```

```
data['Demand_lag_week168hr'] = data['Demand'].shift(168)
```

Rolling Meand and Rolling Standard Deviation

```
data['Demand_rolling_mean_24hr'] = data['Demand'].rolling(window=24).mean()
```

```
data['Demand_rolling_std_24hr'] = data['Demand'].rolling(window=24).std()
```

```
data.head(30)
```

	hour	dayofweek	month	year	dayofyear	weekofyear	quarter	is_w
Timestamp								
2020-01-01	0	2	1	2020	1	1	1	
2020-01-01	1	2	1	2020	1	1	1	
2020-01-01	2	2	1	2020	1	1	1	
2020-01-01	3	2	1	2020	1	1	1	
2020-01-01	4	2	1	2020	1	1	1	
2020-01-01	5	2	1	2020	1	1	1	
2020-01-01	6	2	1	2020	1	1	1	
2020-01-01	7	2	1	2020	1	1	1	
2020-01-01	8	2	1	2020	1	1	1	
2020-01-01	9	2	1	2020	1	1	1	
2020-01-01	10	2	1	2020	1	1	1	
2020-01-01	11	2	1	2020	1	1	1	
2020-01-01	12	2	1	2020	1	1	1	
2020-01-01	13	2	1	2020	1	1	1	
2020-01-01	14	2	1	2020	1	1	1	
2020-01-01	15	2	1	2020	1	1	1	
2020-01-01	16	2	1	2020	1	1	1	
2020-01-01	17	2	1	2020	1	1	1	
2020-01-01	18	2	1	2020	1	1	1	
2020-01-01	19	2	1	2020	1	1	1	
2020-01-01	20	2	1	2020	1	1	1	
2020-01-01	21	2	1	2020	1	1	1	
2020-01-01	22	2	1	2020	1	1	1	
2020-01-01	23	2	1	2020	1	1	1	
2020-01-02	0	3	1	2020	2	1	1	
2020-01-02	1	3	1	2020	2	1	1	

2020-01-02	2	3	1	2020	2	1	1
2020-01-02	3	3	1	2020	2	1	1
2020-01-02	4	3	1	2020	2	1	1
2020-01-02	5	3	1	2020	2	1	1

Next steps: Generate code with data View recommended plots New interactive sheet

```
##now dropping all the rows will null entries
data = data.dropna()
```

```
data
```

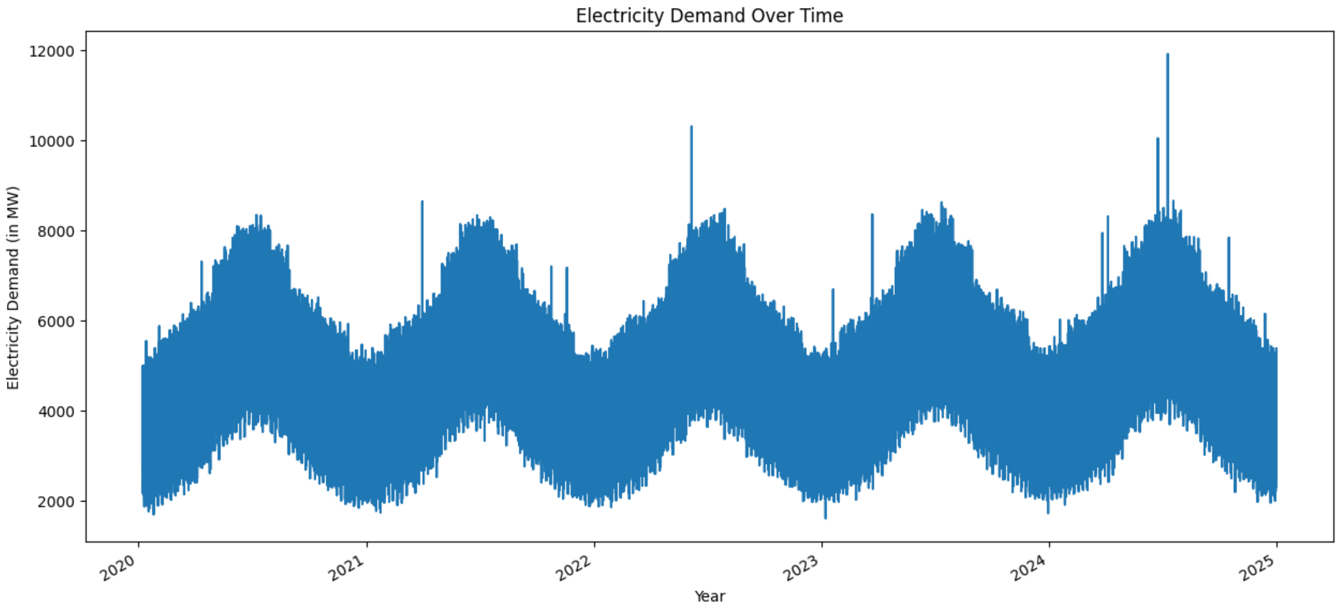
	hour	dayofweek	month	year	dayofyear	weekofyear	quarter	is_w
Timestamp								
2020-01-08	0	2	1	2020	8	2	1	
2020-01-08	1	2	1	2020	8	2	1	
2020-01-08	2	2	1	2020	8	2	1	
2020-01-08	3	2	1	2020	8	2	1	
2020-01-08	4	2	1	2020	8	2	1	
...	...	...	...	...	...	...	...	
2024-12-31	19	1	12	2024	366	1	4	
2024-12-31	20	1	12	2024	366	1	4	
2024-12-31	21	1	12	2024	366	1	4	
2024-12-31	22	1	12	2024	366	1	4	
2024-12-31	23	1	12	2024	366	1	4	

43676 rows × 15 columns

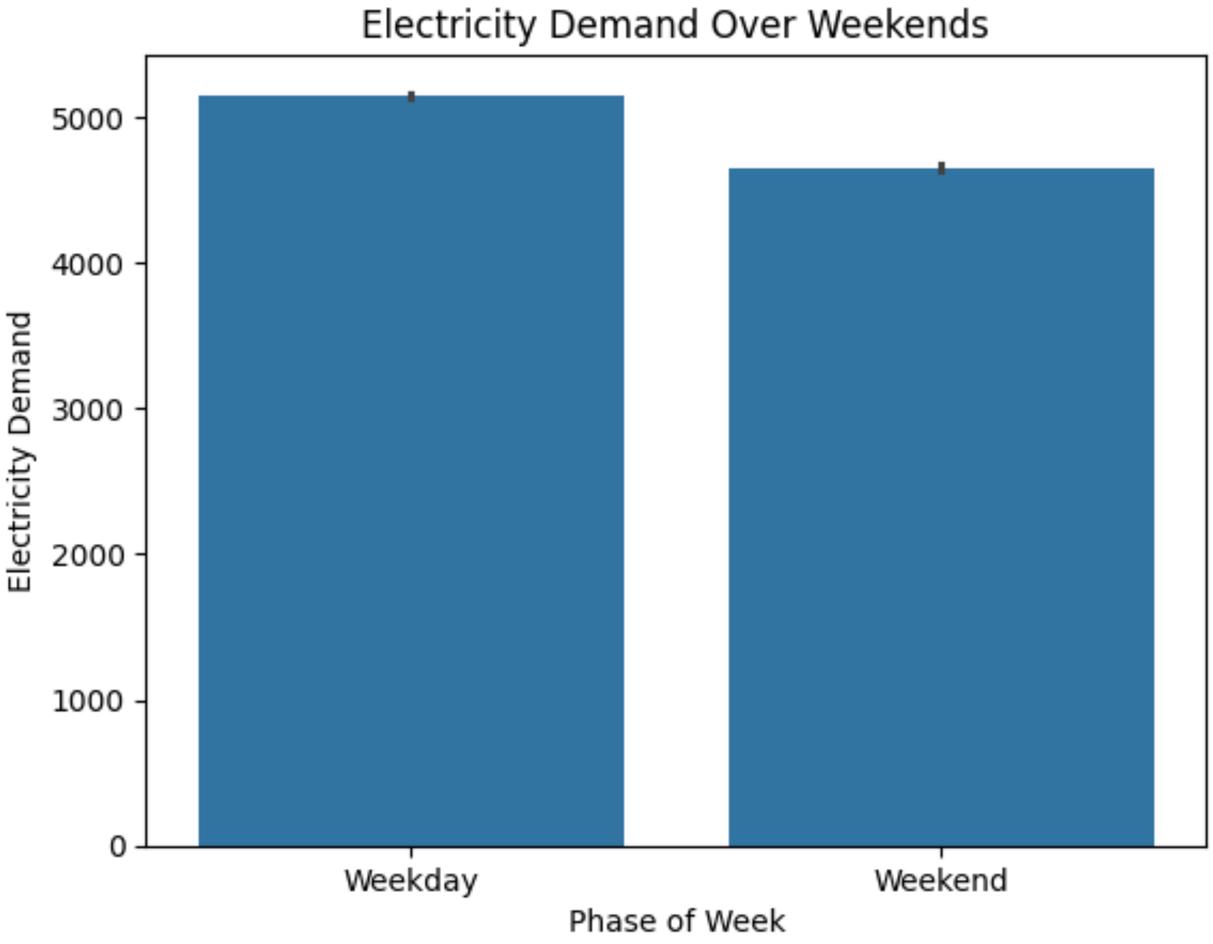
Next steps: Generate code with data View recommended plots New interactive sheet

Visualization

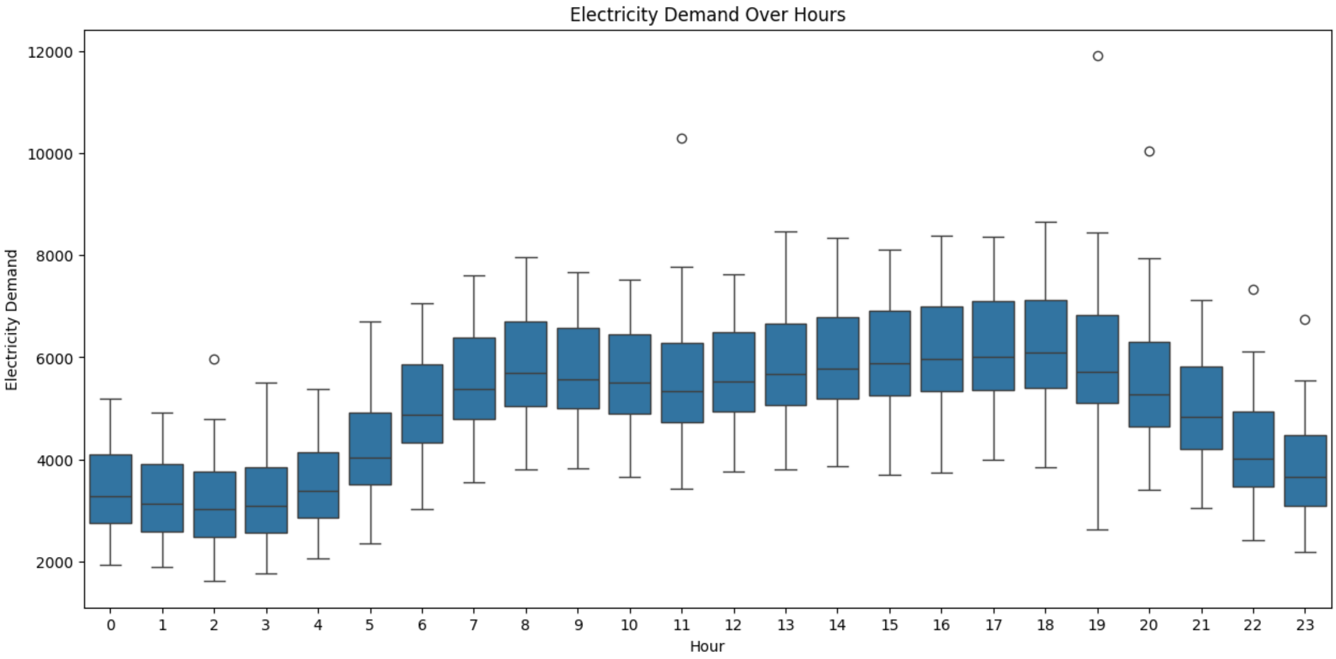
```
data['Demand'].plot(title = 'Electricity Demand Over Time', figsize = (15,7))
plt.xlabel('Year')
plt.ylabel('Electricity Demand (in MW)')
plt.show()
```



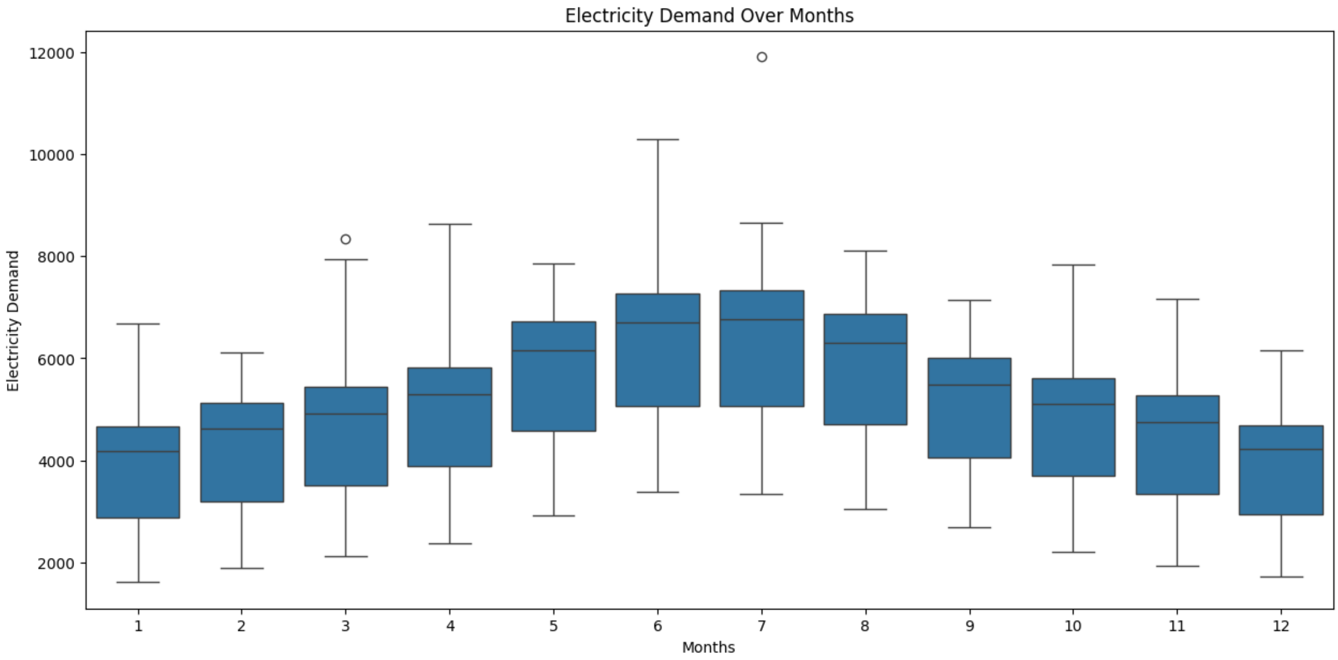
```
sns.barplot(x=data.is_weekend, y=data.Demand)
plt.title('Electricity Demand Over Weekends')
plt.xticks([0, 1], ['Weekday', 'Weekend'])
plt.xlabel('Phase of Week')
plt.ylabel('Electricity Demand')
plt.show()
```



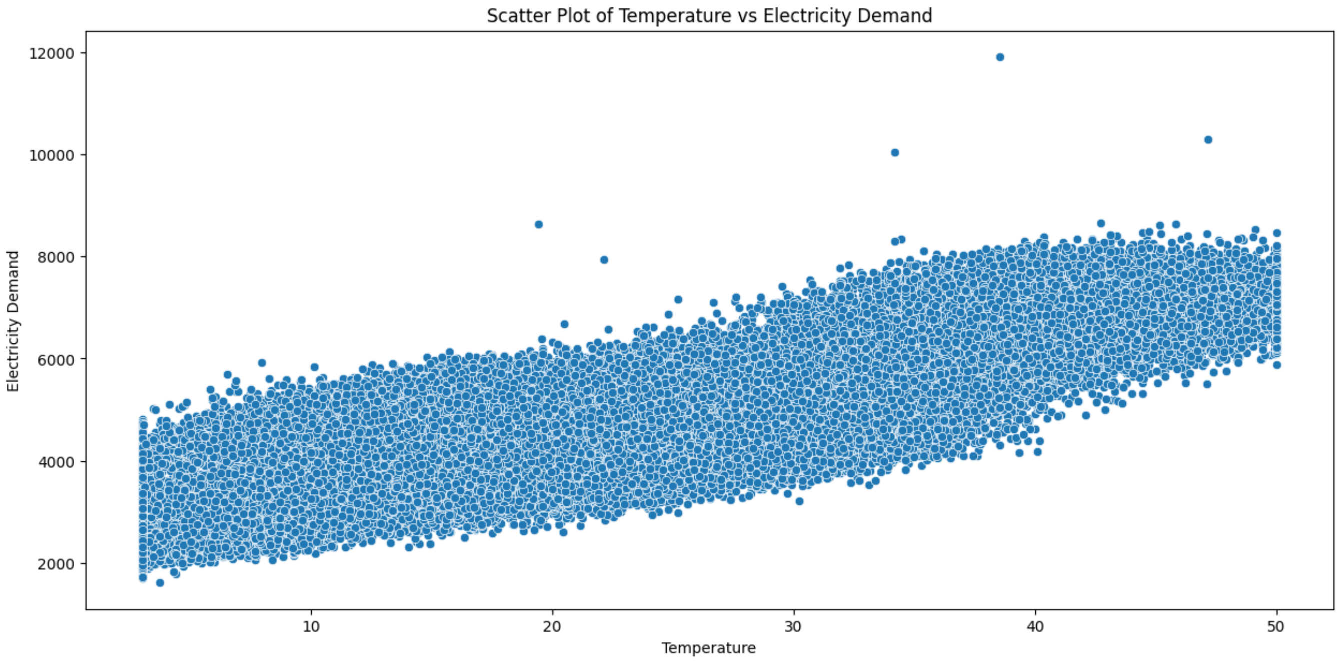
```
plt.figure(figsize=(15,7))
sns.boxplot(x=data.hour, y=data.Demand)
plt.title('Electricity Demand Over Hours')
plt.xlabel('Hour')
plt.ylabel('Electricity Demand')
plt.show()
```



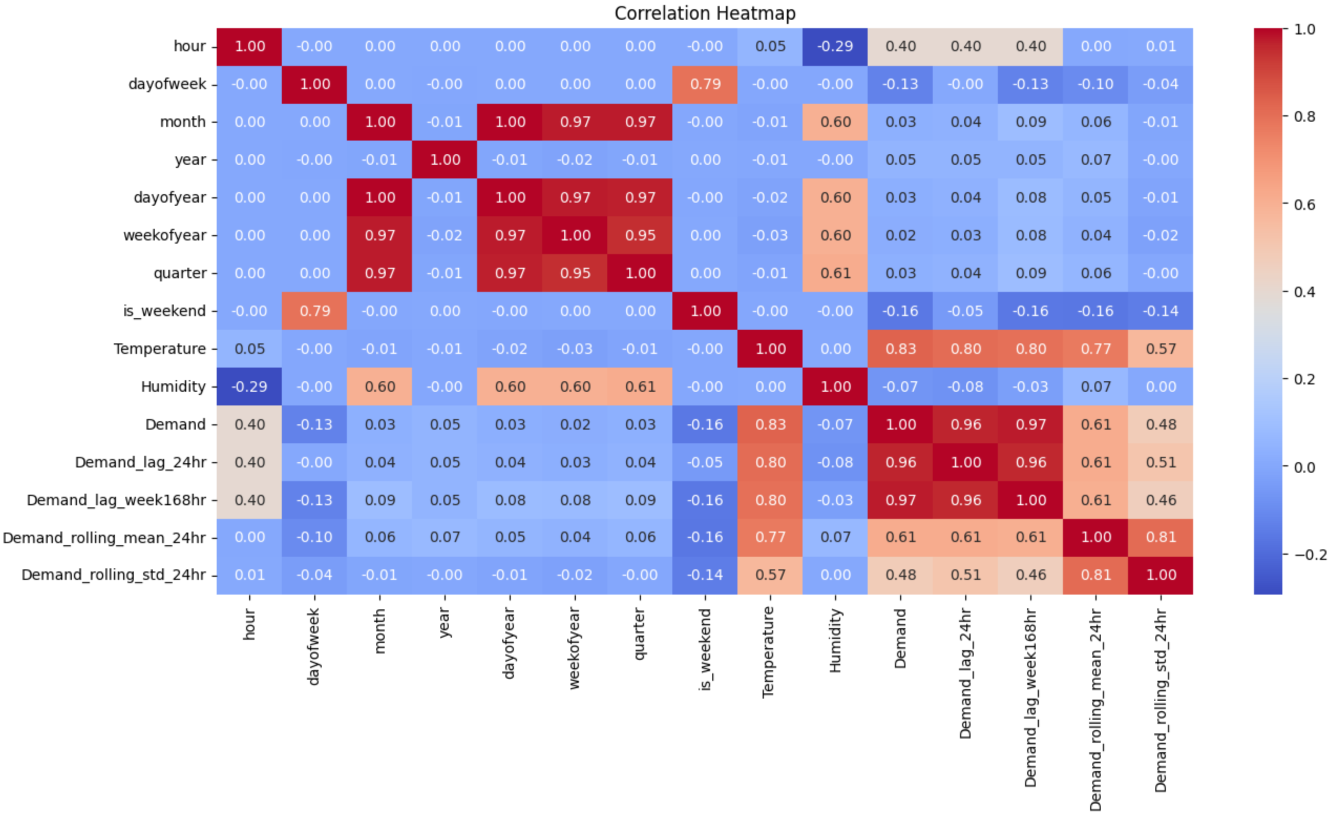
```
plt.figure(figsize=(15,7))
sns.boxplot(x=data.month, y=data.Demand)
plt.title('Electricity Demand Over Months')
plt.xlabel('Months')
plt.ylabel('Electricity Demand')
plt.show()
```



```
plt.figure(figsize=(15,7))
sns.scatterplot(x=data.Temperature, y=data.Demand)
plt.title('Scatter Plot of Temperature vs Electricity Demand')
plt.xlabel('Temperature')
plt.ylabel('Electricity Demand')
plt.show()
```



```
plt.figure(figsize=(15,7))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



Correlation heatmaps shows how one column's values change w.r.t other column according to their correlation points (CP).

- when CP = 1, it means to columns are directly correlated
- when CP = -1, it means to columns are inversely correlated

Model Building

```
# assigning dependen variable
y = data.Demand
```

```
# dependent variables consists of all the columns(parameters) except the Demand
x = data.drop(columns = ['Demand'], axis=1)
```

```
y
```

```
                 Demand
Timestamp
2020-01-08  2363.060115
2020-01-08  2282.558766
2020-01-08  2193.324174
2020-01-08  2208.724679
2020-01-08  2402.611018
    ...          ...
2024-12-31  4689.693109
2024-12-31  4331.249224
2024-12-31  4015.979957
2024-12-31  3353.241682
2024-12-31  3219.023339

43676 rows × 1 columns

dtype: float64
```

```
x
```

	hour	dayofweek	month	year	dayofyear	weekofyear	quarter	is_w
Timestamp								
2020-01-08	0	2	1	2020	8	2	1	
2020-01-08	1	2	1	2020	8	2	1	
2020-01-08	2	2	1	2020	8	2	1	
2020-01-08	3	2	1	2020	8	2	1	
2020-01-08	4	2	1	2020	8	2	1	
...	...	...	...	...	...	...	...	
2024-12-31	19	1	12	2024	366	1	4	
2024-12-31	20	1	12	2024	366	1	4	
2024-12-31	21	1	12	2024	366	1	4	
2024-12-31	22	1	12	2024	366	1	4	
2024-12-31	23	1	12	2024	366	1	4	

43676 rows × 14 columns

Next steps: Generate code with x View recommended plots New interactive sheet

Splitting the data into 80 - 20 for training and testing

```
x_train = x.loc[:'2023-12-31']

y_train = y.loc[:'2023-12-31']

x_test = x.loc['2024-01-01':]

y_test = y.loc['2024-01-01':]
```

XGBoost Model

As it can handle non-linear data easily for timeseries forcasting

```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import TimeSeriesSplit
```

Training the model

```
model_xgb = XGBRegressor(n_estimators=1000,
                         learning_rate=0.01,
                         early_stopping_rounds=50,
                         random_state=42,
                         objective='reg:squarederror')
```

In short:

The XGBRegressor initialization creates a model that will train for up to 1000 boosting rounds (trees) but will stop early if performance on a validation set doesn't improve for 50 consecutive rounds.

Each tree's contribution is scaled down by a learning rate of 0.01 to prevent overfitting.

This model is designed to minimize the squared error for regression tasks.

```
model_xgb.fit(x_train, y_train, eval_set=[(x_test, y_test)], verbose=False)
```

```
▾                                XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=None
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.01, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=1
             n_jobs=None, num_parallel_tree=None, ...)
```

```
# making predictions
y_pred = model_xgb.predict(x_test)
```

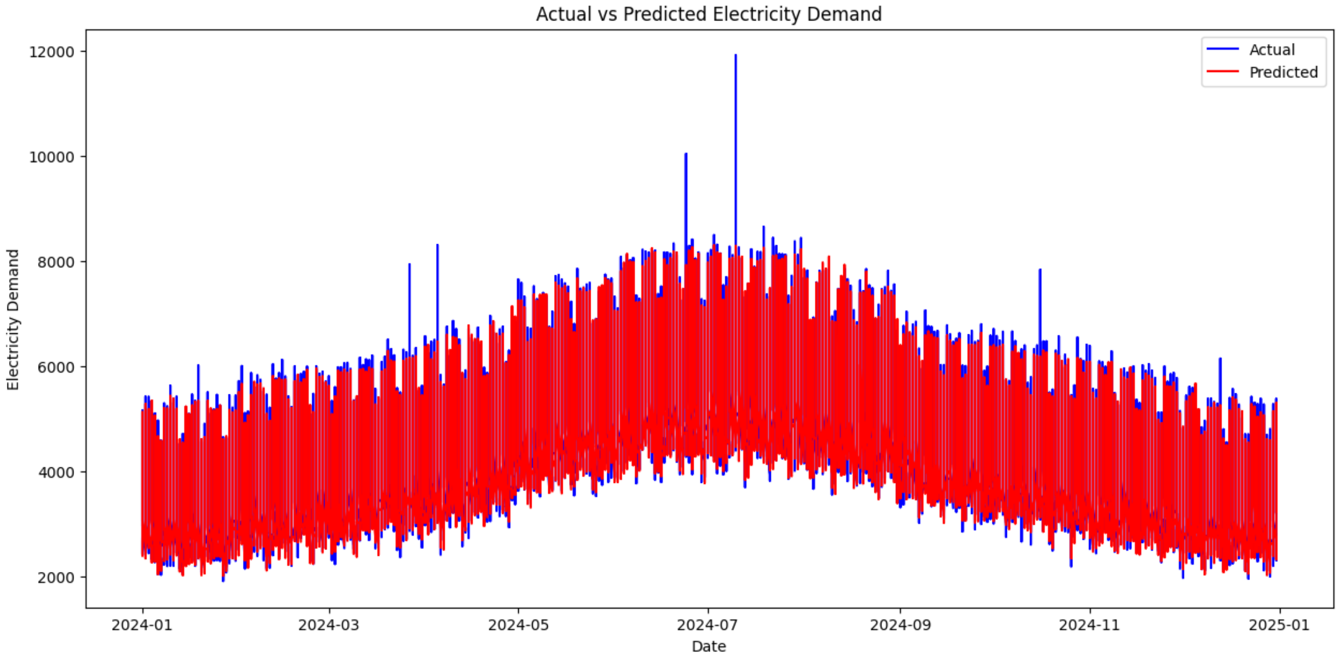
```
# evaluating its findings
root_mse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
```

```
print("Root Mean Squared Error:", root_mse)
print("Mean Absolute Error:", mae)
```

```
Root Mean Squared Error: 175.22716387271916
Mean Absolute Error: 123.47612356015286
```

Visualising the Model Predictions

```
plt.figure(figsize=(15,7))
plt.plot(y_test.index, y_test.values, label='Actual', color='blue')
plt.plot(y_test.index, y_pred, label='Predicted', color='red')
plt.title('Actual vs Predicted Electricity Demand')
plt.xlabel('Date')
plt.ylabel('Electricity Demand')
plt.legend()
plt.show()
```



```
# saving the model
import joblib
joblib.dump(model_xgb, 'xgb_model.pkl')
```

```
['xgb_model.pkl']
```

```
files.download('xgb_model.pkl')
```

We can use this .pkl file to instantiatate the model we just created.

```
# to load the pkl file
model_new = joblib.load('xgb_model.pkl')


# verify the model
model_new
```

```
▼                                 XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=None
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.01, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=1
             n_jobs=None, num_parallel_tree=None, ...)
```