Colab now has AI features powered by Gemini. The video below provides information on how to use these features, whether you're new to Python, or a seasoned veteran.

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab or Colab Features You May Have Missed to learn more, or just get started below!

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g',
alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"data:image/png;base64,
{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)

<IPython.core.display.Markdown object>
```

Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

For example, if you find yourself waiting for **pandas** code to finish running and want to go faster, you can switch to a GPU Runtime and use libraries like RAPIDS cuDF that provide zero-code-change acceleration.

To learn more about accelerating pandas on Colab, see the 10 minute guide or US stock market data analysis demo.

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow

- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

- Overview of Colab
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets
- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Intro to RAPIDS cuDF to accelerate pandas
- Linear regression with tf.keras using synthetic data
- TensorFlow with GPUs
- TensorFlow with TPUs
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

data_e=pd.read_csv('energy_dataset.csv')
data_w=pd.read_csv('weather_features.csv')

import holidays
```

```
data_e['time'] = pd.to_datetime((data_e['time']),utc=True)
data_w['dt_iso'] = pd.to_datetime((data_w['dt_iso']),utc=True)

spain_holidays = holidays.Spain()

def is_holiday(date):
    return 1 if date in spain_holidays else 0

data_e['holiday'] = data_e['time'].dt.date.apply(is_holiday)
data_w['holiday'] = data_w['dt_iso'].dt.date.apply(is_holiday)

data_e.head()
```

{"type":"dataframe","variable_name":"data_e"}

```
data_e.shape
```

```
(35064, 30)
```

```
data_e.columns
```

```
Index(['time', 'generation biomass', 'generation fossil brown
coal/lignite',
       'generation fossil coal-derived gas', 'generation fossil gas',
       'generation fossil hard coal', 'generation fossil oil',
       'generation fossil oil shale', 'generation fossil peat',
       'generation geothermal', 'generation hydro pumped storage
aggregated',
       'generation hydro pumped storage consumption',
       'generation hydro run-of-river and poundage',
       'generation hydro water reservoir', 'generation marine',
       'generation nuclear', 'generation other', 'generation other
renewable',
       'generation solar', 'generation waste', 'generation wind
offshore',
       'generation wind onshore', 'forecast solar day ahead',
       'forecast wind offshore eday ahead', 'forecast wind onshore day
ahead',
       'total load forecast', 'total load actual', 'price day ahead',
       'price actual', 'holiday'],
      dtype='object')
```

```
data_e.duplicated().sum()
```

```
0
```

```
data_e.isnull().sum()
```

```
time                                                    0
generation biomass                                     19
generation fossil brown coal/lignite                   18
generation fossil coal-derived gas                     18
```

```
generation fossil gas                                  18
generation fossil hard coal                            18
generation fossil oil                                  19
generation fossil oil shale                            18
generation fossil peat                                 18
generation geothermal                                  18
generation hydro pumped storage aggregated          35064
generation hydro pumped storage consumption            19
generation hydro run-of-river and poundage             19
generation hydro water reservoir                       18
generation marine                                      19
generation nuclear                                     17
generation other                                       18
generation other renewable                             18
generation solar                                       18
generation waste                                       19
generation wind offshore                               18
generation wind onshore                                18
forecast solar day ahead                                0
forecast wind offshore eday ahead                   35064
forecast wind onshore day ahead                         0
total load forecast                                     0
total load actual                                      36
price day ahead                                         0
price actual                                            0
holiday                                                 0
dtype: int64
```

```
data_e.interpolate(method='linear', limit_direction='forward',
inplace=True)
```

```
data_e.isnull().sum()
```

```
time                                                    0
generation biomass                                      0
generation fossil brown coal/lignite                    0
generation fossil coal-derived gas                      0
generation fossil gas                                   0
generation fossil hard coal                             0
generation fossil oil                                   0
generation fossil oil shale                             0
generation fossil peat                                  0
generation geothermal                                   0
generation hydro pumped storage aggregated          35064
generation hydro pumped storage consumption             0
generation hydro run-of-river and poundage              0
generation hydro water reservoir                        0
generation marine                                       0
generation nuclear                                      0
generation other                                        0
```

```
generation other renewable                                    0
generation solar                                              0
generation waste                                              0
generation wind offshore                                      0
generation wind onshore                                       0
forecast solar day ahead                                      0
forecast wind offshore eday ahead                         35064
forecast wind onshore day ahead                               0
total load forecast                                           0
total load actual                                             0
price day ahead                                               0
price actual                                                  0
holiday                                                       0
dtype: int64
```

```
data_e.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 30 columns):
 #   Column                                       Non-Null Count
Dtype
---  ------                                       --------------
-----
 0   time                                         35064 non-null
datetime64[ns, UTC]
 1   generation biomass                           35064 non-null
float64
 2   generation fossil brown coal/lignite         35064 non-null
float64
 3   generation fossil coal-derived gas           35064 non-null
float64
 4   generation fossil gas                        35064 non-null
float64
 5   generation fossil hard coal                  35064 non-null
float64
 6   generation fossil oil                        35064 non-null
float64
 7   generation fossil oil shale                  35064 non-null
float64
 8   generation fossil peat                       35064 non-null
float64
 9   generation geothermal                        35064 non-null
float64
 10  generation hydro pumped storage aggregated   0 non-null
float64
 11  generation hydro pumped storage consumption  35064 non-null
float64
 12  generation hydro run-of-river and poundage   35064 non-null
float64
```

```
 13  generation hydro water reservoir              35064 non-null
float64
 14  generation marine                             35064 non-null
float64
 15  generation nuclear                            35064 non-null
float64
 16  generation other                              35064 non-null
float64
 17  generation other renewable                    35064 non-null
float64
 18  generation solar                              35064 non-null
float64
 19  generation waste                              35064 non-null
float64
 20  generation wind offshore                      35064 non-null
float64
 21  generation wind onshore                       35064 non-null
float64
 22  forecast solar day ahead                      35064 non-null
int64
 23  forecast wind offshore eday ahead             0 non-null
float64
 24  forecast wind onshore day ahead               35064 non-null
int64
 25  total load forecast                           35064 non-null
int64
 26  total load actual                             35064 non-null
float64
 27  price day ahead                               35064 non-null
float64
 28  price actual                                  35064 non-null
float64
 29  holiday                                       35064 non-null
int64
dtypes: datetime64[ns, UTC](1), float64(25), int64(4)
memory usage: 8.0 MB
```

```
data_e.describe().round(2)
```

{"type":"dataframe"}

```
data_e =data_e.drop(['generation hydro pumped storage aggregated',
                                'forecast wind offshore eday ahead'],
axis = 1)
```

```
data_e.nunique()
```

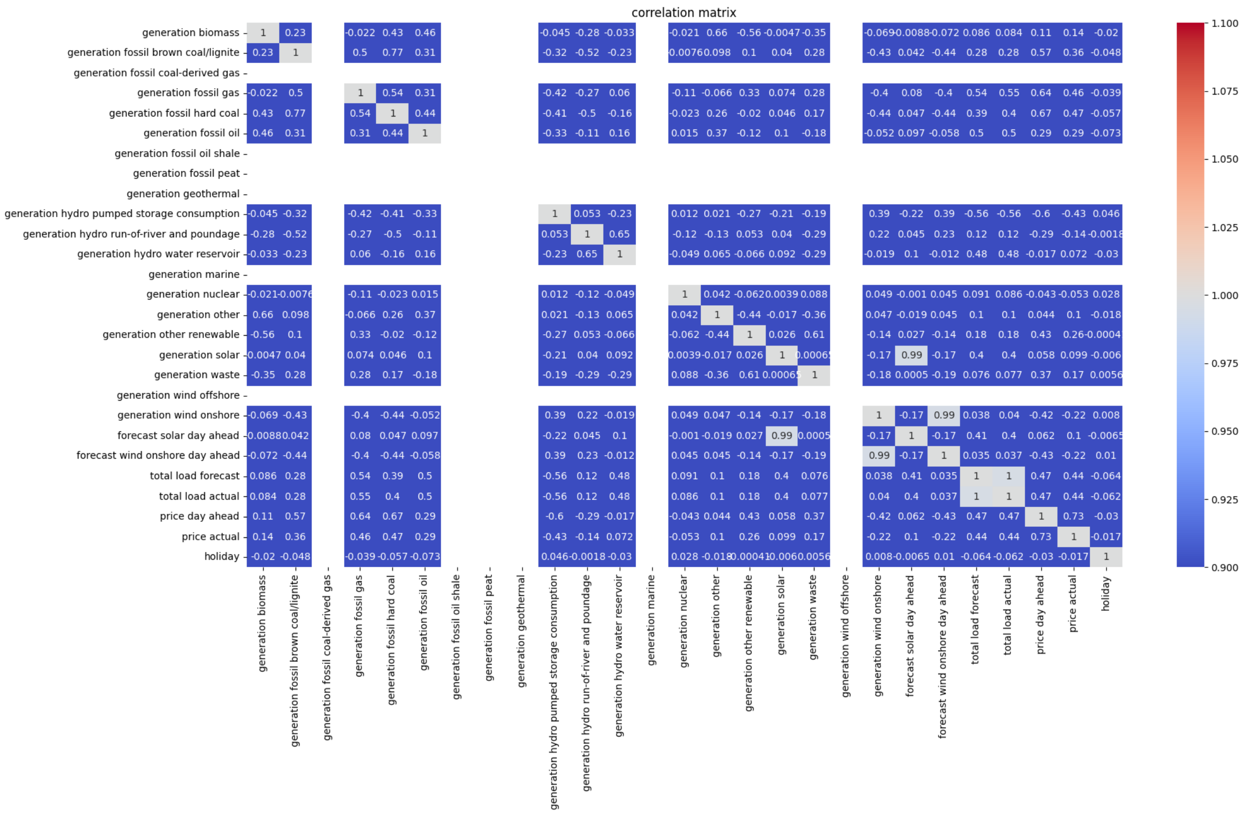
```
time                                           35064
generation biomass                               435
generation fossil brown coal/lignite             964
```

```
generation fossil coal-derived gas                    1
generation fossil gas                              8310
generation fossil hard coal                        7279
generation fossil oil                               334
generation fossil oil shale                           1
generation fossil peat                                1
generation geothermal                                 1
generation hydro pumped storage consumption        3319
generation hydro run-of-river and poundage         1697
generation hydro water reservoir                   7040
generation marine                                     1
generation nuclear                                 2396
generation other                                    112
generation other renewable                           87
generation solar                                   5344
generation waste                                    268
generation wind offshore                              1
generation wind onshore                           11477
forecast solar day ahead                           5356
forecast wind onshore day ahead                   11332
total load forecast                               14790
total load actual                                 15149
price day ahead                                    5747
price actual                                       6653
holiday                                               2
dtype: int64
```

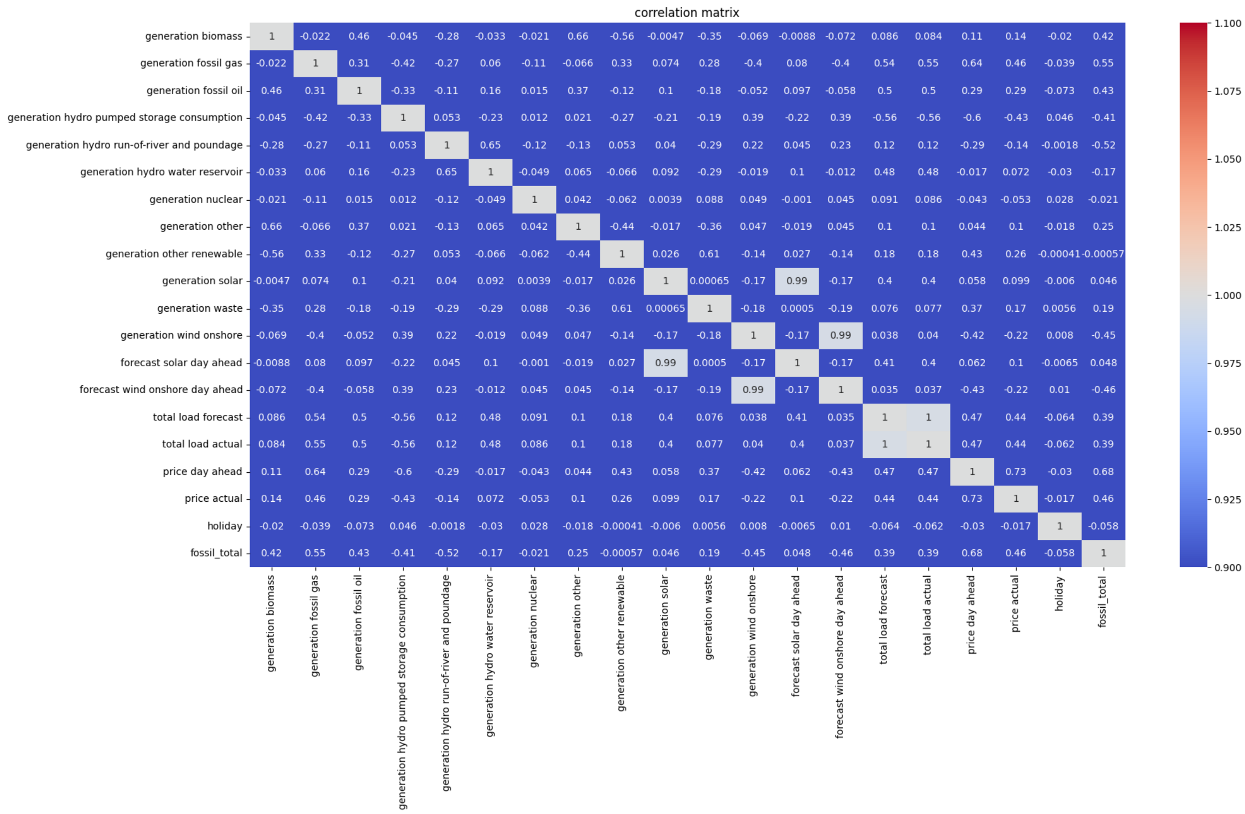
```
data_e['time'] = pd.to_datetime(data_e['time'])
data_e =data_e.set_index('time')
data_e
```

{"type":"dataframe","variable_name":"data_e"}

```
def plot_heatmap(info_e):
  corr=info_e.corr()
  plt.figure(figsize=(min(20, corr.shape[1] * 1.2), 10))
  sns.heatmap(corr,annot=True,cmap="coolwarm",vmin=1,vmax=1)
  plt.title('correlation matrix')
  return plt.show()

plot_heatmap(data_e)
```




```
data_e ["fossil_total"]=data_e['generation fossil hard coal'] +
data_e['generation fossil brown coal/lignite']
data_e.drop(['generation fossil hard coal', 'generation fossil brown
coal/lignite','generation fossil coal-derived gas' , 'generation
fossil peat','generation fossil oil shale',
               'generation geothermal', 'generation
marine','generation wind offshore'],axis=1,inplace=True)

plot_heatmap(data_e)
```




```
data_w.head()
```

{"type":"dataframe","variable_name":"data_w"}

```
data_w.columns
```

```
Index(['dt_iso', 'city_name', 'temp', 'temp_min', 'temp_max',
'pressure',
       'humidity', 'wind_speed', 'wind_deg', 'rain_1h', 'rain_3h',
'snow_3h',
       'clouds_all', 'weather_id', 'weather_main',
'weather_description',
       'weather_icon', 'holiday'],
      dtype='object')
```

```
data_w.shape
```

```
(178396, 18)
```

```
data_w.isnull().sum()
```

```
dt_iso                 0
city_name              0
temp                   0
temp_min               0
temp_max               0
```

```
pressure                 0
humidity                 0
wind_speed               0
wind_deg                 0
rain_1h                  0
rain_3h                  0
snow_3h                  0
clouds_all               0
weather_id               0
weather_main             0
weather_description      0
weather_icon             0
holiday                  0
dtype: int64
```

```
data_w.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178396 entries, 0 to 178395
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   dt_iso               178396 non-null  datetime64[ns, UTC]
 1   city_name            178396 non-null  object
 2   temp                 178396 non-null  float64
 3   temp_min             178396 non-null  float64
 4   temp_max             178396 non-null  float64
 5   pressure             178396 non-null  int64
 6   humidity             178396 non-null  int64
 7   wind_speed           178396 non-null  int64
 8   wind_deg             178396 non-null  int64
 9   rain_1h              178396 non-null  float64
 10  rain_3h              178396 non-null  float64
 11  snow_3h              178396 non-null  float64
 12  clouds_all           178396 non-null  int64
 13  weather_id           178396 non-null  int64
 14  weather_main         178396 non-null  object
 15  weather_description  178396 non-null  object
 16  weather_icon         178396 non-null  object
 17  holiday              178396 non-null  int64
dtypes: datetime64[ns, UTC](1), float64(6), int64(7), object(4)
memory usage: 24.5+ MB
```

```
data_w.describe().round(2)
```


```
{"summary":"{\n  \"name\": \"data_w\",\n  \"rows\": 8,\n  \"fields\":
[\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 62984.53119863224,\n
\"min\": 8.03,\n        \"max\": 178396.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          289.62,\n
```

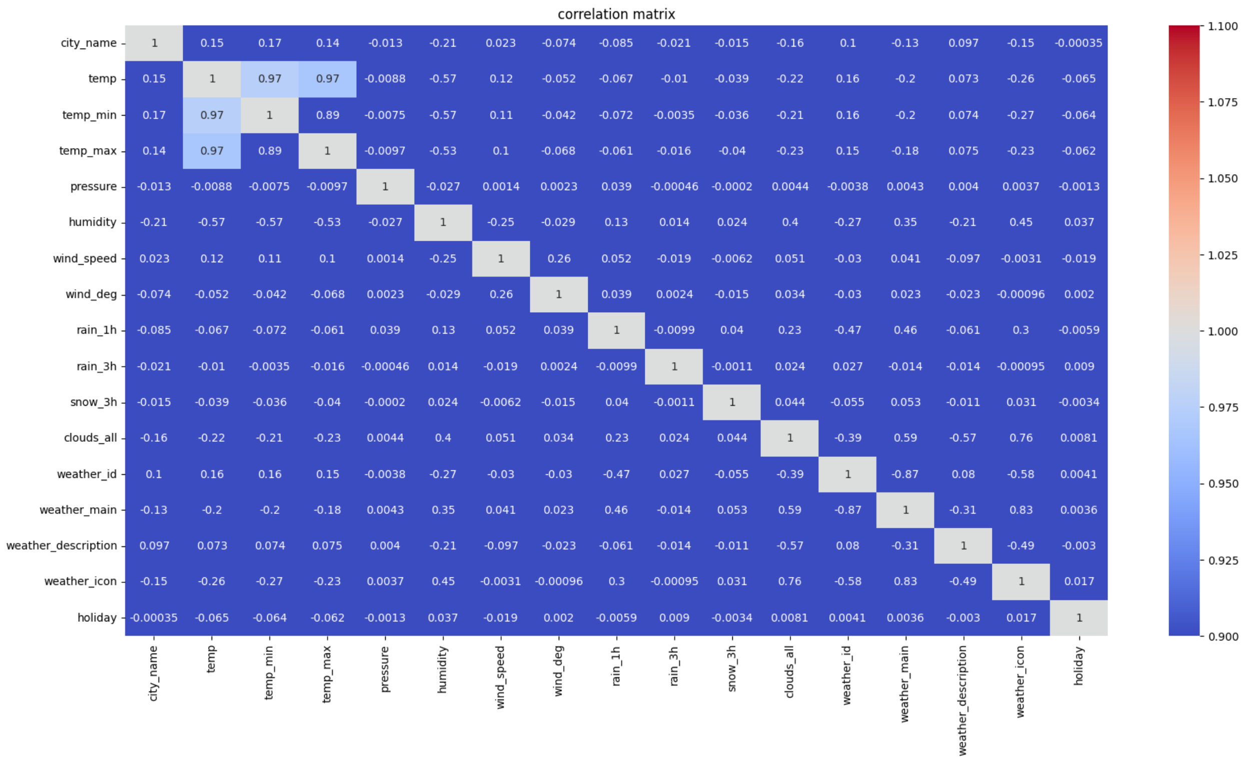
289.15,\n   178396.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"temp_min\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 62984.80447098353,\n   \"min\":
7.96,\n   \"max\": 178396.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   288.33,\n   288.15,\n
178396.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"temp_max\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 62983.94737640082,\n   \"min\":
8.61,\n   \"max\": 178396.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   291.09,\n   290.15,\n
178396.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"pressure\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 352464.42432672356,\n   \"min\":
0.0,\n   \"max\": 1008371.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   1069.26,\n   1018.0,\n
178396.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"humidity\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 63052.19904917059,\n   \"min\":
0.0,\n   \"max\": 178396.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   68.42,\n   72.0,\n   178396.0\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"wind_speed\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 63065.225469373356,\n   \"min\":
0.0,\n   \"max\": 178396.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   2.47,\n   2.0,\n   178396.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"wind_deg\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
63014.77231336905,\n   \"min\": 0.0,\n   \"max\": 178396.0,\
n   \"num_unique_values\": 8,\n   \"samples\": [\n
166.59,\n   177.0,\n   178396.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"rain_1h\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
63071.88047115948,\n   \"min\": 0.0,\n   \"max\": 178396.0,\
n   \"num_unique_values\": 5,\n   \"samples\": [\n
0.08,\n   12.0,\n   0.4\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"rain_3h\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
63072.39299072351,\n   \"min\": 0.0,\n   \"max\": 178396.0,\
n   \"num_unique_values\": 4,\n   \"samples\": [\n
0.0,\n   2.32,\n   178396.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

n   },\n   {\n   \"column\": \"snow_3h\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
63071.414089844664,\n   \"min\": 0.0,\n   \"max\":
178396.0,\n   \"num_unique_values\": 4,\n   \"samples\": [\n
0.0,\n   21.5,\n   178396.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"clouds_all\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
63061.616940185224,\n   \"min\": 0.0,\n   \"max\":
178396.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n
178396.0,\n   25.07,\n   40.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"weather_id\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
62857.33163058068,\n   \"min\": 108.73,\n   \"max\":
178396.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n
178396.0,\n   759.83,\n   801.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"holiday\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
63072.45157529076,\n   \"min\": 0.0,\n   \"max\": 178396.0,\
n   \"num_unique_values\": 5,\n   \"samples\": [\n
0.02,\n   1.0,\n   0.15\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}


```
data_w['dt_iso' ] = pd.to_datetime((data_w['dt_iso']),utc=True)
data_w =data_w.set_index('dt_iso')
data_w
```


{"type":"dataframe","variable_name":"data_w"}


```
data_temp = data_w.copy(deep = True)
labels = ['weather_id',
'weather_main','weather_description','weather_icon','city_name']
for col in labels:
    data_temp[col] = LabelEncoder().fit_transform(data_w[col])

plot_heatmap(data_temp)
```



```
col_drop_name = ['weather_id',
'weather_main','weather_description','weather_icon', 'temp_min',
'temp_max']
# col_drop_name = ['weather_id',
'weather_main','weather_description','weather_icon']
data_w.drop(col_drop_name, axis = 1 , inplace = True)

data_w.duplicated().sum()
```

```
13051
```

```
data_w=data_w.reset_index().drop_duplicates()

data_w.isnull().sum()
```

```
dt_iso         0
city_name      0
temp           0
pressure       0
humidity       0
wind_speed     0
wind_deg       0
rain_1h        0
rain_3h        0
snow_3h        0
clouds_all     0
```

```
holiday        0
dtype: int64

# Convert 'dt_iso' column to datetime format
data_w['time'] = pd.to_datetime(data_w['dt_iso'])

# Drop the 'dt_iso' column after conversion
data_w.drop(["dt_iso"], axis=1, inplace=True)

# Set 'time' as the index
data_w.set_index('time', inplace=True)

# Check if 'index' column exists before attempting to drop
if 'index' in data_w.columns:
    data_w.drop(["index"], axis=1, inplace=True)

data_w
```

{"type":"dataframe","variable_name":"data_w"}

```
data_e
```

{"summary":"{\n \"name\": \"data_e\",\n \"rows\": 35064,\n \"fields\": [\n   {\n   \"column\": \"time\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"2014-12-31 23:00:00+00:00\",\n   \"max\": \"2018-12-31 22:00:00+00:00\",\n   \"num_unique_values\": 35064,\n   \"samples\": [\n   \"2015-09-10 21:00:00+00:00\",\n   \"2018-09-20 07:00:00+00:00\",\n   \"2016-01-04 13:00:00+00:00\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"generation biomass\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 85.34626560634285,\n   \"min\": 0.0,\n   \"max\": 592.0,\n   \"num_unique_values\": 435,\n   \"samples\": [\n   313.0,\n   375.0,\n   481.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"generation fossil gas\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2201.5109844400663,\n   \"min\": 0.0,\n   \"max\": 20034.0,\n   \"num_unique_values\": 8310,\n   \"samples\": [\n   3808.0,\n   8358.0,\n   4506.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"generation fossil oil\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 52.51992700618621,\n   \"min\": 0.0,\n   \"max\": 449.0,\n   \"num_unique_values\": 334,\n   \"samples\": [\n   205.0,\n   174.0,\n   333.85714285714283\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"generation hydro pumped storage consumption\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 792.3127041043937,\n   \"min\":

```
0.0,\n        \"max\": 4523.0,\n        \"num_unique_values\": 3319,\n
\"samples\": [\n          2518.0,\n          3243.0,\n
2326.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"generation hydro run-of-river and poundage\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
400.7123041191516,\n        \"min\": 0.0,\n        \"max\": 2000.0,\n
\"num_unique_values\": 1697,\n        \"samples\": [\n
482.0,\n          779.0,\n          687.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"generation hydro water
reservoir\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 1835.1750784195215,\n        \"min\": 0.0,\n
\"max\": 9728.0,\n        \"num_unique_values\": 7040,\n
\"samples\": [\n          1875.0,\n          648.0,\n          1333.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"generation nuclear\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 840.272333029197,\n        \"min\":
0.0,\n        \"max\": 7117.0,\n        \"num_unique_values\": 2396,\n
\"samples\": [\n          6956.0,\n          7060.0,\n
6446.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"generation other\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 20.238791913007326,\n        \"min\":
0.0,\n        \"max\": 106.0,\n        \"num_unique_values\": 112,\n
\"samples\": [\n          59.0,\n          71.0,\n          45.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"generation other renewable\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
14.076947707437599,\n        \"min\": 0.0,\n        \"max\": 119.0,\n
\"num_unique_values\": 87,\n        \"samples\": [\n          110.0,\n
73.0,\n          71.85714285714286\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"generation solar\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1679.961732996324,\n        \"min\": 0.0,\n        \"max\": 5792.0,\n
\"num_unique_values\": 5344,\n        \"samples\": [\n
970.0,\n          1057.0,\n          2536.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"generation waste\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
50.21842280627276,\n        \"min\": 0.0,\n        \"max\": 357.0,\n
\"num_unique_values\": 268,\n        \"samples\": [\n          271.0,\
n          316.0,\n          178.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"generation wind onshore\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3213.586295667971,\n        \"min\": 0.0,\n        \"max\": 17436.0,\n
```

```
\"num_unique_values\": 11477,\n        \"samples\": [\n
14509.0,\n          14329.0,\n          8953.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"forecast solar day ahead\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1677,\n        \"min\": 0,\n        \"max\": 5836,\n
\"num_unique_values\": 5356,\n        \"samples\": [\n          4378,\
n          2872,\n          4343\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"forecast wind onshore day
ahead\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3176,\n        \"min\": 237,\n        \"max\": 17430,\n
\"num_unique_values\": 11332,\n        \"samples\": [\n
9682,\n          12055,\n          11745\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total load forecast\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4594,\n        \"min\": 18105,\n        \"max\": 41390,\n
\"num_unique_values\": 14790,\n        \"samples\": [\n
21970,\n          30045,\n          23844\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total load actual\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4575.828853961441,\n        \"min\": 18041.0,\n        \"max\":
41015.0,\n        \"num_unique_values\": 15149,\n        \"samples\":
[\n          34630.0,\n          21475.0,\n          31084.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"price day ahead\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 14.618899685404326,\n        \"min\":
2.06,\n        \"max\": 101.99,\n        \"num_unique_values\": 5747,\
n        \"samples\": [\n          33.63,\n          45.73,\n
32.64\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"price actual\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 14.204083293241405,\n        \"min\":
9.33,\n        \"max\": 116.8,\n        \"num_unique_values\": 6653,\n
\"samples\": [\n          83.51,\n          57.22,\n          54.1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"holiday\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fossil_total\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2246.1062774346974,\n        \"min\":
0.0,\n        \"max\": 9320.0,\n        \"num_unique_values\": 8114,\n
\"samples\": [\n          5956.0,\n          8960.0\n        ],\n
```

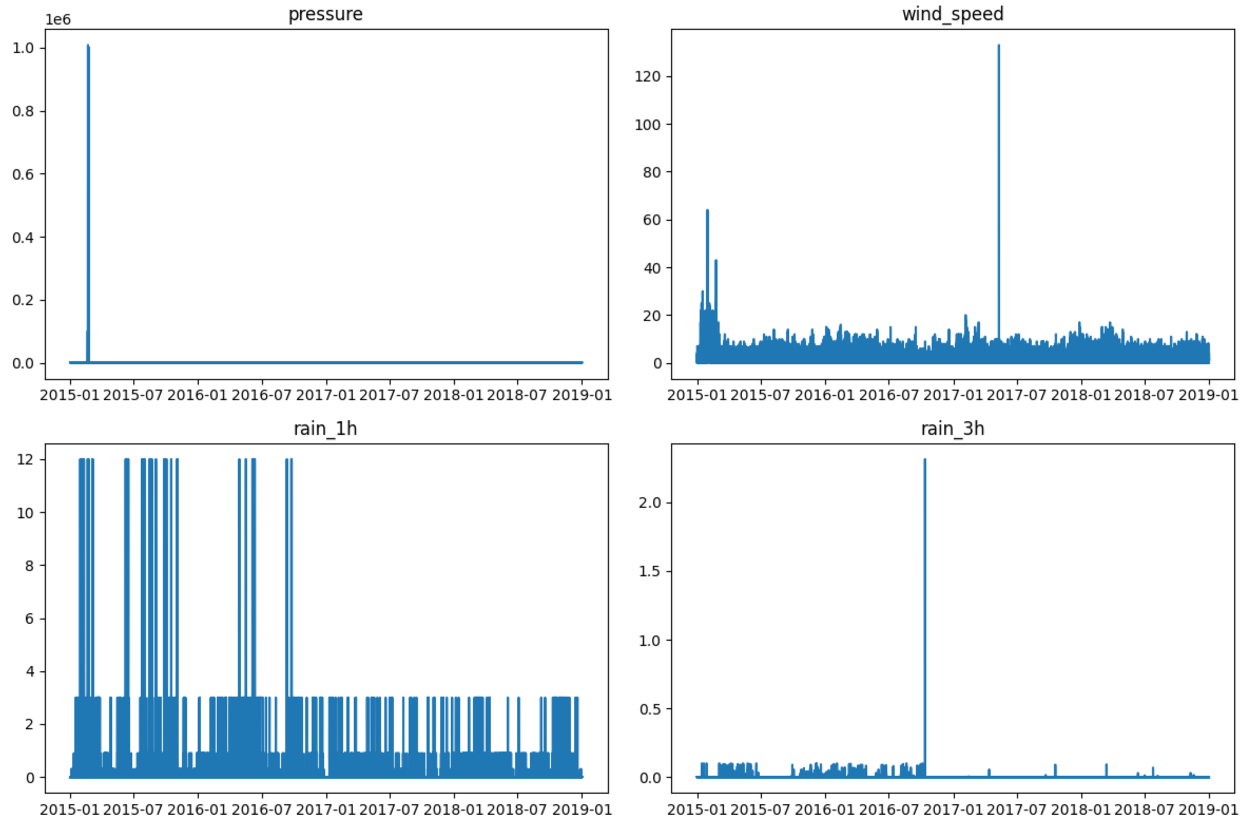
```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"data_e"}
```


```
data_w.describe().round(2)
```

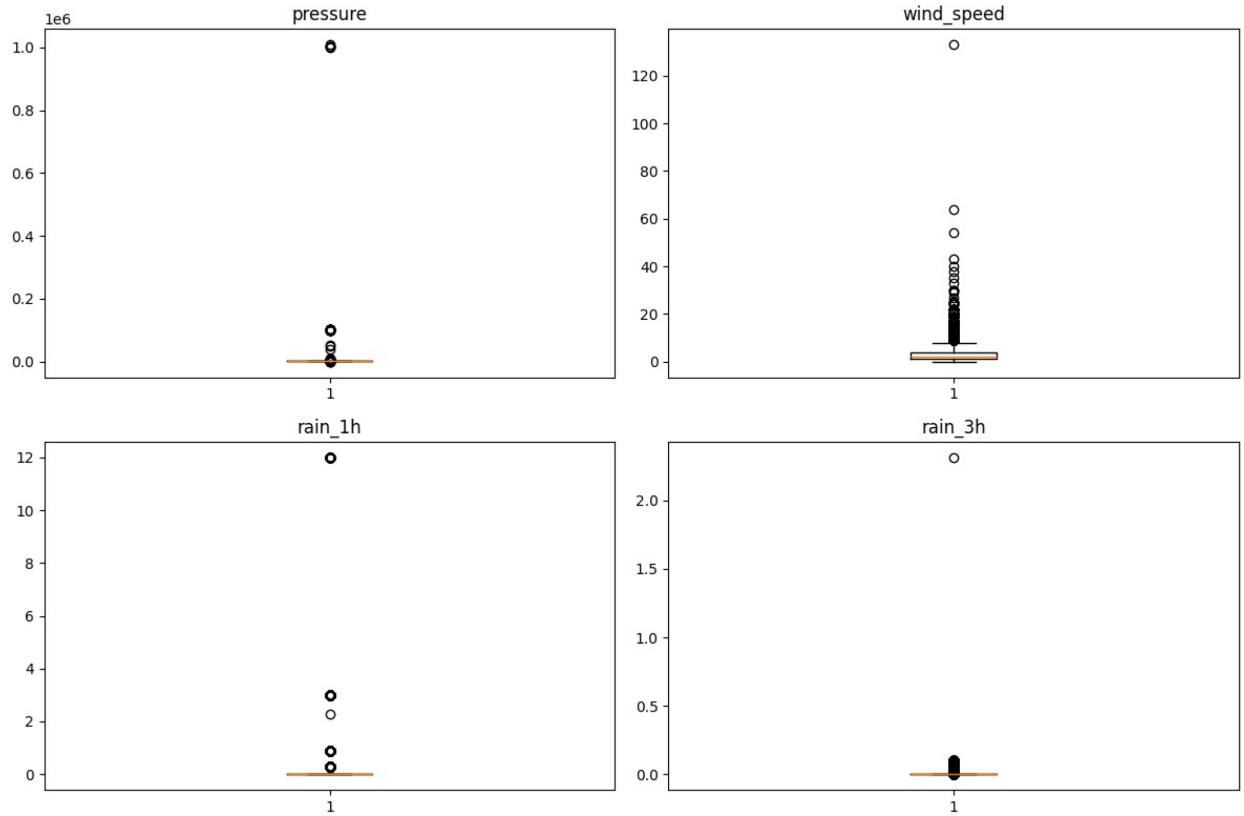

```
{"summary":"{\n  \"name\": \"data_w\",\n  \"rows\": 8,\n  \"fields\":
[\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 61896.985718306394,\n
\"min\": 8.02,\n        \"max\": 175320.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          289.71,\n
289.15,\n          175320.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"pressure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 352427.1251218861,\n        \"min\":
0.0,\n        \"max\": 1008371.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          1070.2,\n          1018.0,\n
175320.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"humidity\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 61964.69169446349,\n        \"min\":
0.0,\n        \"max\": 175320.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          68.03,\n          72.0,\n          175320.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"wind_speed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 61977.695532748236,\n        \"min\":
0.0,\n        \"max\": 175320.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          2.47,\n          2.0,\n          175320.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"wind_deg\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
61927.139131176096,\n        \"min\": 0.0,\n        \"max\":
175320.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
166.72,\n          178.0,\n          175320.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"rain_1h\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
61984.351254340865,\n        \"min\": 0.0,\n        \"max\":
175320.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.07,\n          12.0,\n          0.39\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"rain_3h\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
61984.862761350145,\n        \"min\": 0.0,\n        \"max\":
175320.0,\n        \"num_unique_values\": 4,\n        \"samples\": [\n
0.0,\n          2.32,\n          175320.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"snow_3h\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
61983.883868226585,\n        \"min\": 0.0,\n        \"max\":
```

```
175320.0,\n        \"num_unique_values\": 4,\n        \"samples\": [\n
0.0,\n          21.5,\n          175320.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"clouds_all\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
61974.34759213272,\n        \"min\": 0.0,\n        \"max\": 175320.0,\
n        \"num_unique_values\": 7,\n        \"samples\": [\n
175320.0,\n          24.34,\n          40.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"holiday\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 61984.9213458423,\
n        \"min\": 0.0,\n        \"max\": 175320.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          0.02,\n
1.0,\n          0.15\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```python
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12,8))

# select the columns to plot
columns_to_plot = ['pressure', 'wind_speed', 'rain_1h', 'rain_3h']

# loop through the subplots and plot each column
for i, ax in enumerate(axes.flat):
    if i < len(columns_to_plot):
        ax.plot(data_w.index,data_w[columns_to_plot[i]])
        ax.set_title(columns_to_plot[i])
    else:
        ax.set_visible(False)

plt.tight_layout()  # adjust the spacing between subplots
plt.show()  # display the plot
```




```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12,8))

# select the columns to plot
columns_to_plot = ['pressure', 'wind_speed', 'rain_1h', 'rain_3h']

# loop through the subplots and plot each column
for i, ax in enumerate(axes.flat):
    if i < len(columns_to_plot):  # Fixed indentation and syntax here
        ax.boxplot(x = data_w[columns_to_plot[i]])
        ax.set_title(columns_to_plot[i])
    else:
        ax.set_visible(False)

plt.tight_layout()  # adjust the spacing between subplots
plt.show()  # display the plot
```



Maximum pressure on earth is recorded as 1080hPa : https://wmo.asu.edu/content/world-highest-sea-level-air-pressure-below-700m Minimum pressure on earth is recorded as 870hPa : https://wmo.asu.edu/content/world-lowest-sea-level-air-pressure-excluding-tornadoes Maximum wind speed on earth is recorded as 113 m/s : https://www.wunderground.com/cat6/the-highest-anemometer-measured-wind-speeds-on-earth

So we set the max threshold as these numbers and replace the outliers with Nan values and then use interpolation to fill the gap.

```
data_w.loc[data_w['pressure']  > 1080,  'pressure'] = np.nan
data_w.loc[data_w['pressure']   < 870,  'pressure'] = np.nan
data_w.loc[data_w['wind_speed'] > 113, 'wind_speed'] = np.nan

data_w.interpolate(method ='linear', limit_direction ='forward',
inplace = True)

<ipython-input-46-4cf3945930da>:5: FutureWarning:
DataFrame.interpolate with object dtype is deprecated and will raise
in a future version. Call obj.infer_objects(copy=False) before
interpolating instead.
  data_w.interpolate(method ='linear', limit_direction ='forward',
inplace = True)

data_w.drop(['rain_3h'], axis = 1 , inplace = True)
```

```
# Select only numeric columns
numeric_data_w = data_w.select_dtypes(include=['number'])

# Calculate the mean for each city
mean_weather_by_city =
numeric_data_w.groupby(data_w['city_name']).mean()
temp_w = data_w.duplicated(keep='first').sum()
print('there are {} duplicate rows in data_w based on all
columns.'.format(temp_w))


there are 9976 duplicate rows in data_w based on all columns.

# Print the number of observations in data_e
print('There are {} observations in data_e.'.format(data_e.shape[0]))

# List of unique cities and grouping by 'city_name'
cities = data_w['city_name'].unique()
grouped_weather = data_w.groupby('city_name')

# Loop through each city and print the number of observations for that
city
for city in cities:
    city_count = grouped_weather.get_group(city).shape[0]
    print('There are {} observations in data_w for city:
{}'.format(city_count, city))

There are 35064 observations in data_e.
There are 35064 observations in data_w for city: Valencia
There are 35064 observations in data_w for city: Madrid
There are 35064 observations in data_w for city: Bilbao
There are 35064 observations in data_w for city:  Barcelona
There are 35064 observations in data_w for city: Seville

# Remove duplicate rows based on 'time' and 'city_name', keeping the
last occurrence, and reset the index to 'time'
data_w_last = data_w.reset_index().drop_duplicates(subset=['time',
'city_name'], keep='last').set_index('time')

# Remove duplicate rows based on 'time' and 'city_name', keeping the
first occurrence, and reset the index to 'time'
data_w_first = data_w.reset_index().drop_duplicates(subset=['time',
'city_name'], keep='first').set_index('time')

# Print the number of observations in data_e
print('There are {} observations in data_e.'.format(data_e.shape[0]))

# List of unique cities and grouping by 'city_name'
cities = data_w['city_name'].unique()
grouped_weather = data_w.groupby('city_name')
```

```
# Loop through each city and print the number of observations for that city
for city in cities:
    city_count = grouped_weather.get_group(city).shape[0]
    print('There are {} observations in data_w for city: {}'.format(city_count, city))
```

```
There are 35064 observations in data_e.
There are 35064 observations in data_w for city: Valencia
There are 35064 observations in data_w for city: Madrid
There are 35064 observations in data_w for city: Bilbao
There are 35064 observations in data_w for city:  Barcelona
There are 35064 observations in data_w for city: Seville
```

```
print('there are {} missing values or nans in data_w.'.format(data_w.isnull().values.sum()))
```

```
there are 0 missing values or nans in data_w.
```

```
data_w_all_cities = [grouped_weather.get_group(x) for x in grouped_weather.groups]
```

```
data_w_all_cities[0]
```


{"summary":"{\n \"name\": \"data_w_all_cities[0]\",\n \"rows\": 35064,\n \"fields\": [\n   {\n   \"column\": \"time\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"2014-12-31 23:00:00+00:00\",\n   \"max\": \"2018-12-31 22:00:00+00:00\",\n   \"num_unique_values\": 35064,\n   \"samples\": [\n   \"2015-09-10 21:00:00+00:00\",\n   \"2018-09-20 07:00:00+00:00\",\n   \"2016-01-04 13:00:00+00:00\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"city_name\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \" Barcelona\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"temp\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 6.723623493789164,\n   \"min\": 262.24,\n   \"max\": 309.15,\n   \"num_unique_values\": 5296,\n   \"samples\": [\n   280.96\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pressure\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 7.582256321982248,\n   \"min\": 918.0,\n   \"max\": 1039.0,\n   \"num_unique_values\": 109,\n   \"samples\": [\n   1006.5833333333334\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"humidity\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 17,\n   \"min\": 0,\n   \"max\": 100,\n   \"num_unique_values\": 99,\n

\"samples\": [\n   31\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"wind_speed\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1.9960811671624321,\n
\"min\": 0.0,\n   \"max\": 15.0,\n   \"num_unique_values\":
16,\n   \"samples\": [\n   7.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"wind_deg\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 108,\n
\"min\": 0,\n   \"max\": 360,\n   \"num_unique_values\":
361,\n   \"samples\": [\n   192\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"rain_1h\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.6677707337523273,\n   \"min\": 0.0,\n   \"max\": 12.0,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   0.3\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"snow_3h\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.0,\n   \"min\": 0.0,\n   \"max\": 0.0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   0.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"clouds_all\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
27,\n   \"min\": 0,\n   \"max\": 100,\n
\"num_unique_values\": 86,\n   \"samples\": [\n   49\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"holiday\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0,\n   \"min\": 0,\n   \"max\": 1,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   1\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

```
data_weather_energy = data_e

for data_city in data_w_all_cities:
    city_name = data_city.iloc[0]['city_name']
    data_temp_city = data_city.add_suffix(f'_{city_name}')
    data_weather_energy = pd.concat([data_weather_energy,
data_temp_city], axis=1)
    data_weather_energy =
data_weather_energy.drop(f'city_name_{city_name}' , axis=1)

data_weather_energy.columns
```

```
Index(['generation biomass', 'generation fossil gas', 'generation
fossil oil',
       'generation hydro pumped storage consumption',
       'generation hydro run-of-river and poundage',
```

```
       'generation hydro water reservoir', 'generation nuclear',
       'generation other', 'generation other renewable', 'generation
solar',
       'generation waste', 'generation wind onshore',
       'forecast solar day ahead', 'forecast wind onshore day ahead',
       'total load forecast', 'total load actual', 'price day ahead',
       'price actual', 'holiday', 'fossil_total', 'temp_ Barcelona',
       'pressure_ Barcelona', 'humidity_ Barcelona', 'wind_speed_
Barcelona',
       'wind_deg_ Barcelona', 'rain_1h_ Barcelona', 'snow_3h_
Barcelona',
       'clouds_all_ Barcelona', 'holiday_ Barcelona', 'temp_Bilbao',
       'pressure_Bilbao', 'humidity_Bilbao', 'wind_speed_Bilbao',
       'wind_deg_Bilbao', 'rain_1h_Bilbao', 'snow_3h_Bilbao',
       'clouds_all_Bilbao', 'holiday_Bilbao', 'temp_Madrid',
'pressure_Madrid',
       'humidity_Madrid', 'wind_speed_Madrid', 'wind_deg_Madrid',
       'rain_1h_Madrid', 'snow_3h_Madrid', 'clouds_all_Madrid',
       'holiday_Madrid', 'temp_Seville', 'pressure_Seville',
       'humidity_Seville', 'wind_speed_Seville', 'wind_deg_Seville',
       'rain_1h_Seville', 'snow_3h_Seville', 'clouds_all_Seville',
       'holiday_Seville', 'temp_Valencia', 'pressure_Valencia',
       'humidity_Valencia', 'wind_speed_Valencia',
'wind_deg_Valencia',
       'rain_1h_Valencia', 'snow_3h_Valencia', 'clouds_all_Valencia',
       'holiday_Valencia'],
      dtype='object')
```

```
data_weather_energy.duplicated().sum()
```

```
0
```

```
data_weather_energy.isnull().sum()
```

```
generation biomass                             0
generation fossil gas                          0
generation fossil oil                          0
generation hydro pumped storage consumption    0
generation hydro run-of-river and poundage     0
                                              ..
wind_deg_Valencia                              0
rain_1h_Valencia                               0
snow_3h_Valencia                               0
clouds_all_Valencia                            0
holiday_Valencia                               0
Length: 65, dtype: int64
```

```
data_weather_energy['hour'] = data_weather_energy.index.map(lambda x :
x.hour)
data_weather_energy['weekday'] = data_weather_energy.index.map(lambda
```

```
x : x.weekday())
data_weather_energy['month'] =data_weather_energy.index.map(lambda x :
x.month)
data_weather_energy['year'] = data_weather_energy.index.map(lambda x:
x.year)

data_weather_energy.columns

Index(['generation biomass', 'generation fossil gas', 'generation
fossil oil',
       'generation hydro pumped storage consumption',
       'generation hydro run-of-river and poundage',
       'generation hydro water reservoir', 'generation nuclear',
       'generation other', 'generation other renewable', 'generation
solar',
       'generation waste', 'generation wind onshore',
       'forecast solar day ahead', 'forecast wind onshore day ahead',
       'total load forecast', 'total load actual', 'price day ahead',
       'price actual', 'holiday', 'fossil_total', 'temp_ Barcelona',
       'pressure_ Barcelona', 'humidity_ Barcelona', 'wind_speed_
Barcelona',
       'wind_deg_ Barcelona', 'rain_1h_ Barcelona', 'snow_3h_
Barcelona',
       'clouds_all_ Barcelona', 'holiday_ Barcelona', 'temp_Bilbao',
       'pressure_Bilbao', 'humidity_Bilbao', 'wind_speed_Bilbao',
       'wind_deg_Bilbao', 'rain_1h_Bilbao', 'snow_3h_Bilbao',
       'clouds_all_Bilbao', 'holiday_Bilbao', 'temp_Madrid',
'pressure_Madrid',
       'humidity_Madrid', 'wind_speed_Madrid', 'wind_deg_Madrid',
       'rain_1h_Madrid', 'snow_3h_Madrid', 'clouds_all_Madrid',
       'holiday_Madrid', 'temp_Seville', 'pressure_Seville',
       'humidity_Seville', 'wind_speed_Seville', 'wind_deg_Seville',
       'rain_1h_Seville', 'snow_3h_Seville', 'clouds_all_Seville',
       'holiday_Seville', 'temp_Valencia', 'pressure_Valencia',
       'humidity_Valencia', 'wind_speed_Valencia',
'wind_deg_Valencia',
       'rain_1h_Valencia', 'snow_3h_Valencia', 'clouds_all_Valencia',
       'holiday_Valencia', 'hour', 'weekday', 'month', 'year'],
      dtype='object')

data_weather_energy.shape

(35064, 69)

data_weather_energy
```

{"type":"dataframe","variable_name":"data_weather_energy"}

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

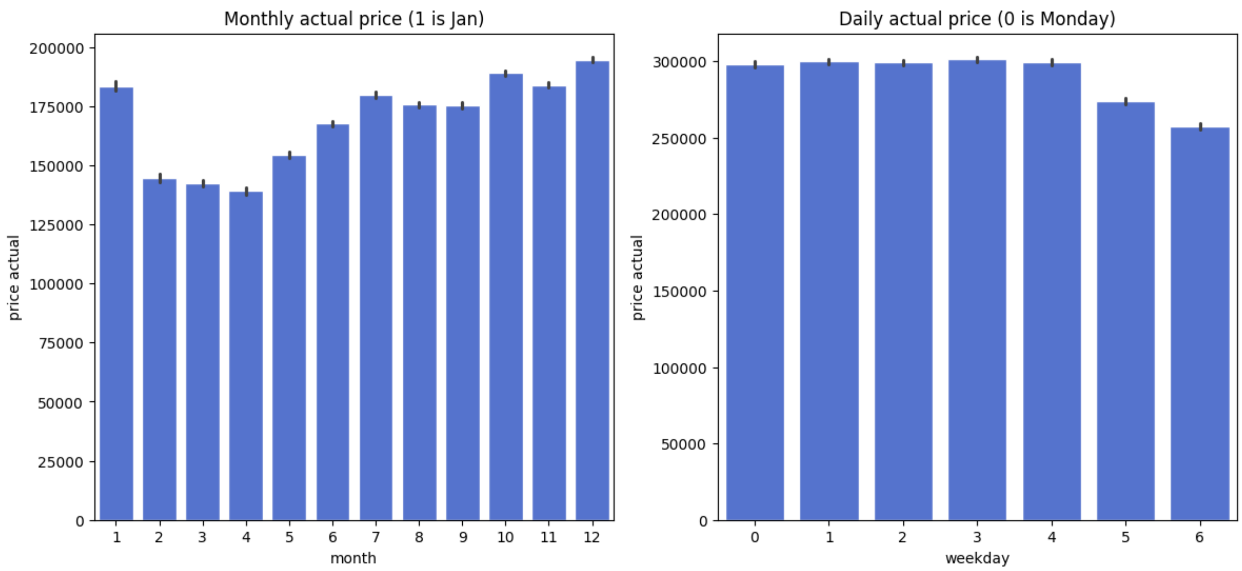
```
fig = make_subplots()

fig.add_trace(
    go.Line(x=data_weather_energy.index, y=data_weather_energy["price
actual"],
                                name="price actual"))
fig.add_trace(

go.Line(x=data_weather_energy.index,y=data_weather_energy.rolling(wind
ow=24).mean()["price actual"],
                                        name="rolling window = daily
ave"))
fig.add_trace(

go.Line(x=data_weather_energy.index,y=data_weather_energy.rolling(wind
ow=24*7).mean()["price actual"],
                                        name="rolling window = weekly
ave"))
# fig.update_xaxes(rangeslider_visible=True)
fig.show()
```

```
/usr/local/lib/python3.10/dist-packages/plotly/graph_objs/
_deprecations.py:378: DeprecationWarning:

plotly.graph_objs.Line is deprecated.
Please replace it with one of the following more specific types
  - plotly.graph_objs.scatter.Line
  - plotly.graph_objs.layout.shape.Line
  - etc.
```

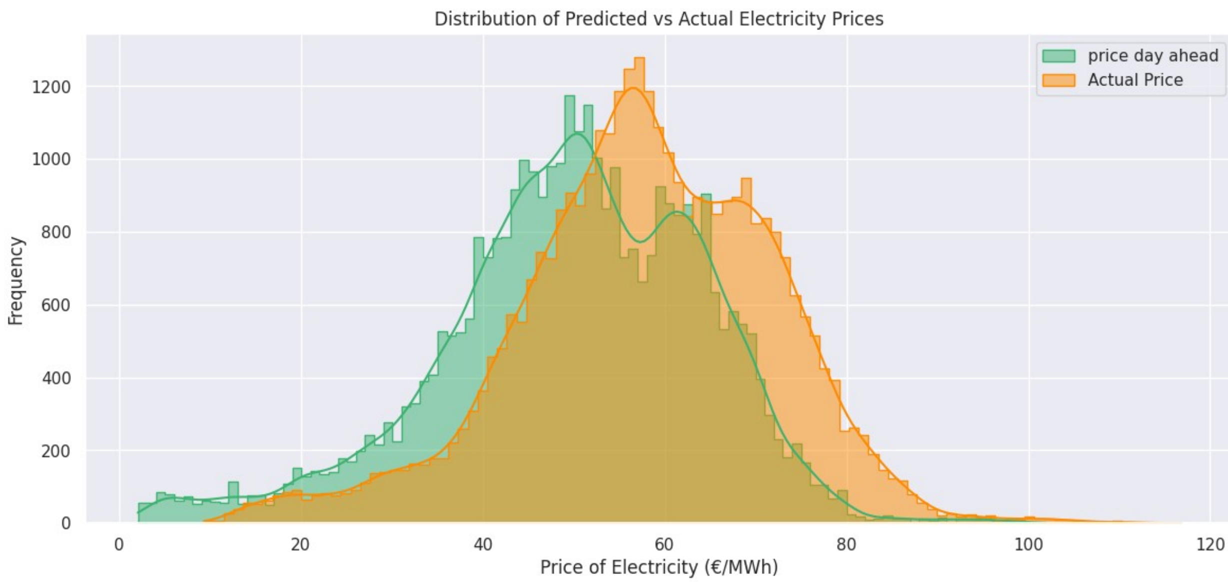
```
fig, axes = plt.subplots(ncols=2, figsize=(14, 6))
sns.set(style="darkgrid")

sns.barplot(
    x="month",
    y="price actual",
    data=data_weather_energy,
    estimator=sum,
    color='royalblue',
    ax=axes[0]);
axes[0].set_title('Monthly actual price (1 is Jan)')

sns.barplot(
    x="weekday",
    y="price actual",
    data=data_weather_energy,
    estimator=sum,
    color='royalblue',
```

```
    ax=axes[1]);
axes[1].set_title('Daily actual price (0 is Monday)')
```

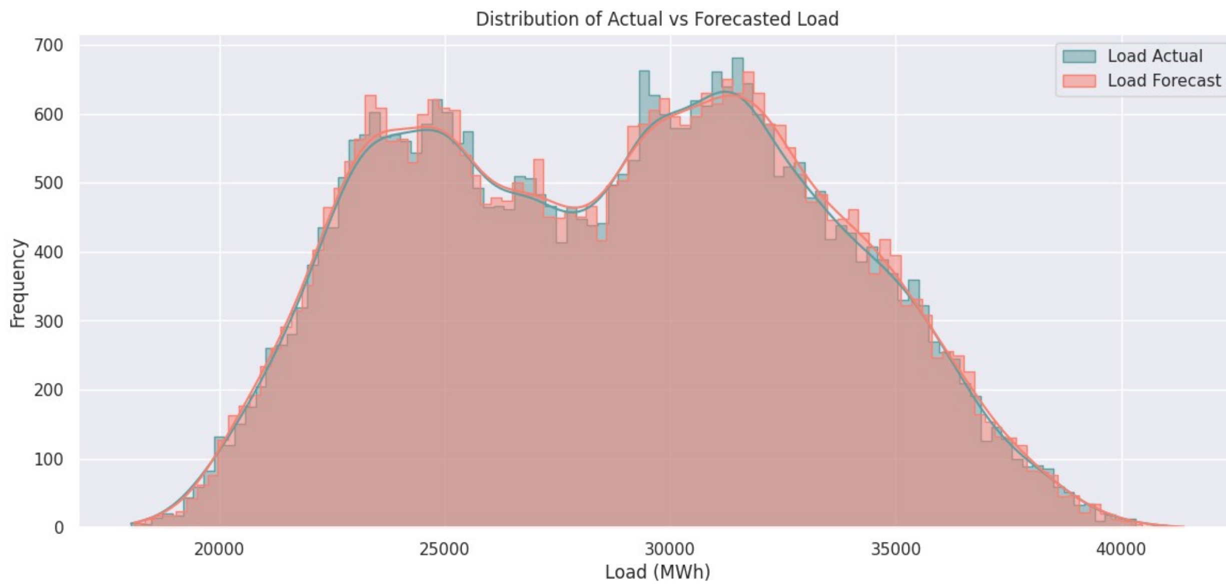
```
Text(0.5, 1.0, 'Daily actual price (0 is Monday)')
```



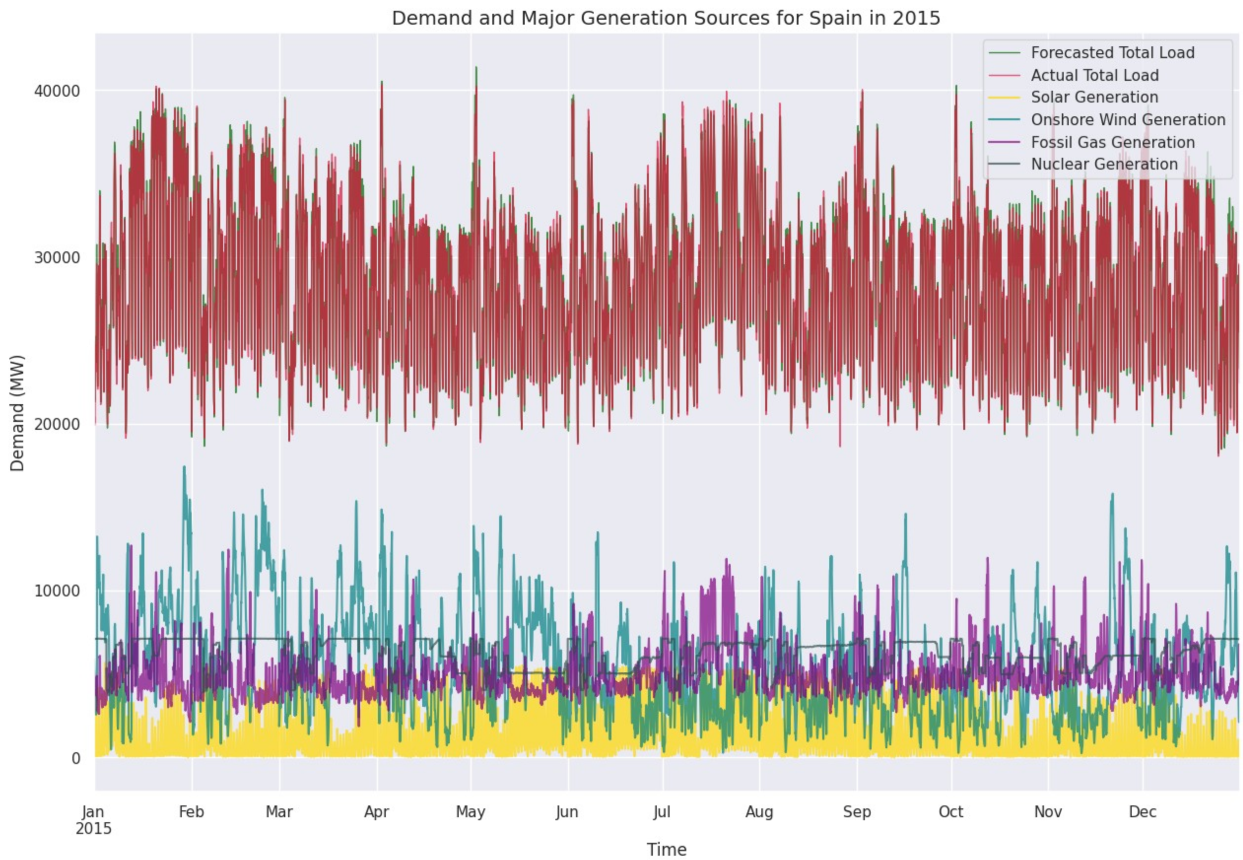

```
plt.figure(figsize=(14,6))

gr = sns.histplot(data_weather_energy['price day ahead'],
                  bins=100,
                  label='price day ahead',
                  element="step",
                  color='mediumseagreen',  # Changed to
'mediumseagreen'
                  kde=True)

gr = sns.histplot(data_weather_energy['price actual'],
                  bins=100,
                  label='Actual Price',
                  element="step",
                  color='darkorange',  # Changed to 'darkorange'
                  kde=True)

gr.set(xlabel="Price of Electricity (€/MWh)", ylabel="Frequency")
plt.legend()
plt.title("Distribution of Predicted vs Actual Electricity Prices")  #
Added title
plt.show()
```



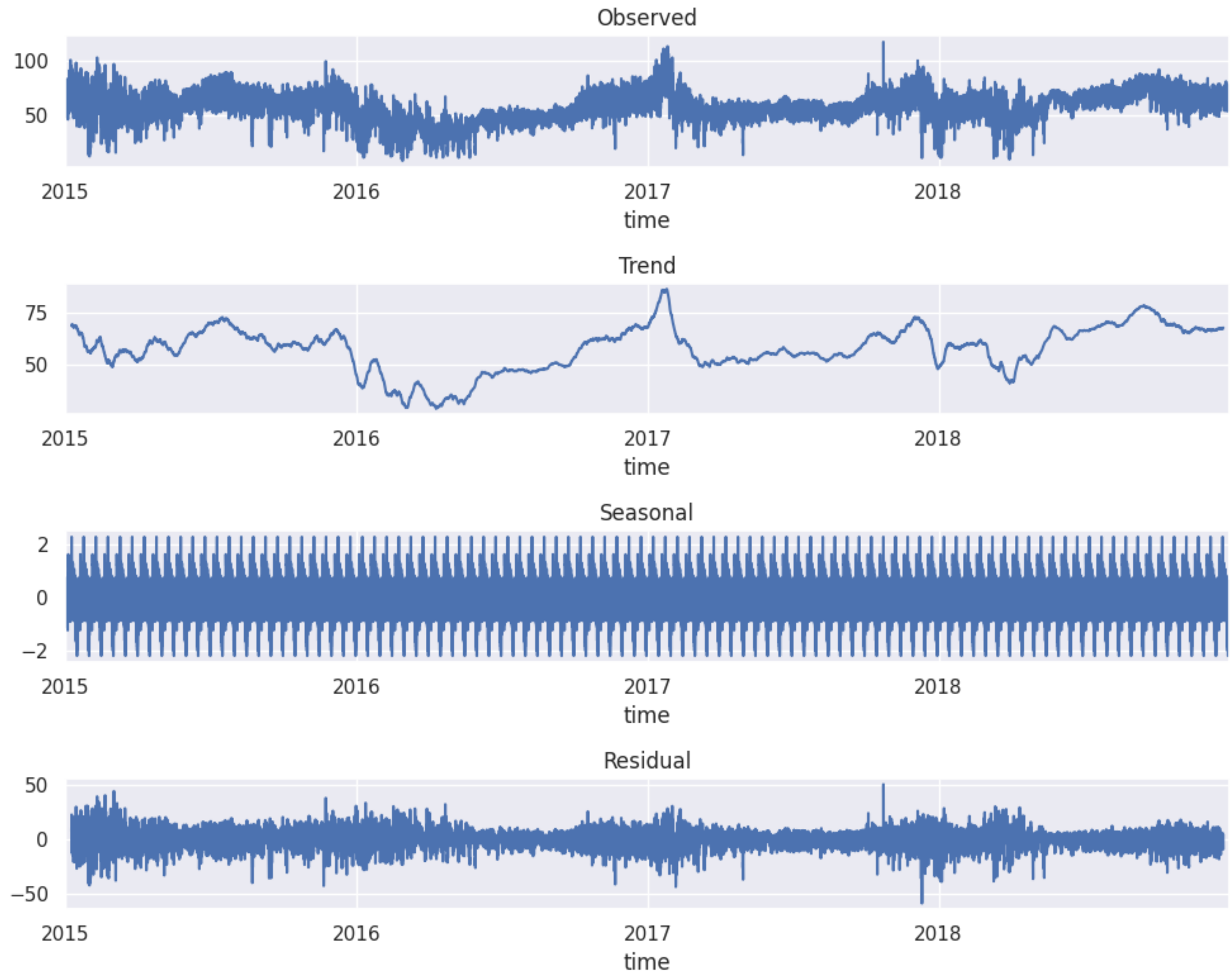

```
plt.figure(figsize=(14,6))

gr = sns.histplot(data_weather_energy['total load actual'],
                  bins=100,
                  label='Load Actual',
                  element="step",
                  color='cadetblue',  # Changed to 'cadetblue'
                  kde=True)

gr = sns.histplot(data_weather_energy['total load forecast'],
                  bins=100,
                  label='Load Forecast',
                  element="step",
                  color='salmon',  # Changed to 'salmon'
                  kde=True)

gr.set(xlabel="Load (MWh)", ylabel="Frequency")
plt.legend()
plt.title("Distribution of Actual vs Forecasted Load")  # Added title
plt.show()
```




```
energy_2015 = data_weather_energy.loc['2015']
fig = plt.figure(figsize=(15,10))

# Plot forecasted and actual total load
energy_2015['total load forecast'].plot(linestyle='-', linewidth=1,
label='Forecasted Total Load', color='darkgreen', alpha=0.7)
energy_2015['total load actual'].plot(linestyle='-', linewidth=1,
label='Actual Total Load', color='crimson', alpha=0.7)

# Plot major generation sources
energy_2015['generation solar'].plot(label='Solar Generation',
color='gold', alpha=0.7)
energy_2015['generation wind onshore'].plot(label='Onshore Wind
Generation', color='teal', alpha=0.7)
energy_2015['generation fossil gas'].plot(label='Fossil Gas
Generation', color='purple', alpha=0.7)
energy_2015['generation nuclear'].plot(label='Nuclear Generation',
color='darkslategray', alpha=0.7)

# Set plot labels and title
plt.ylabel('Demand (MW)', fontsize=12)
plt.xlabel('Time', fontsize=12)
plt.title('Demand and Major Generation Sources for Spain in 2015',
fontsize=14)

# Display legend
plt.legend()

# Show the plot
plt.show()
```

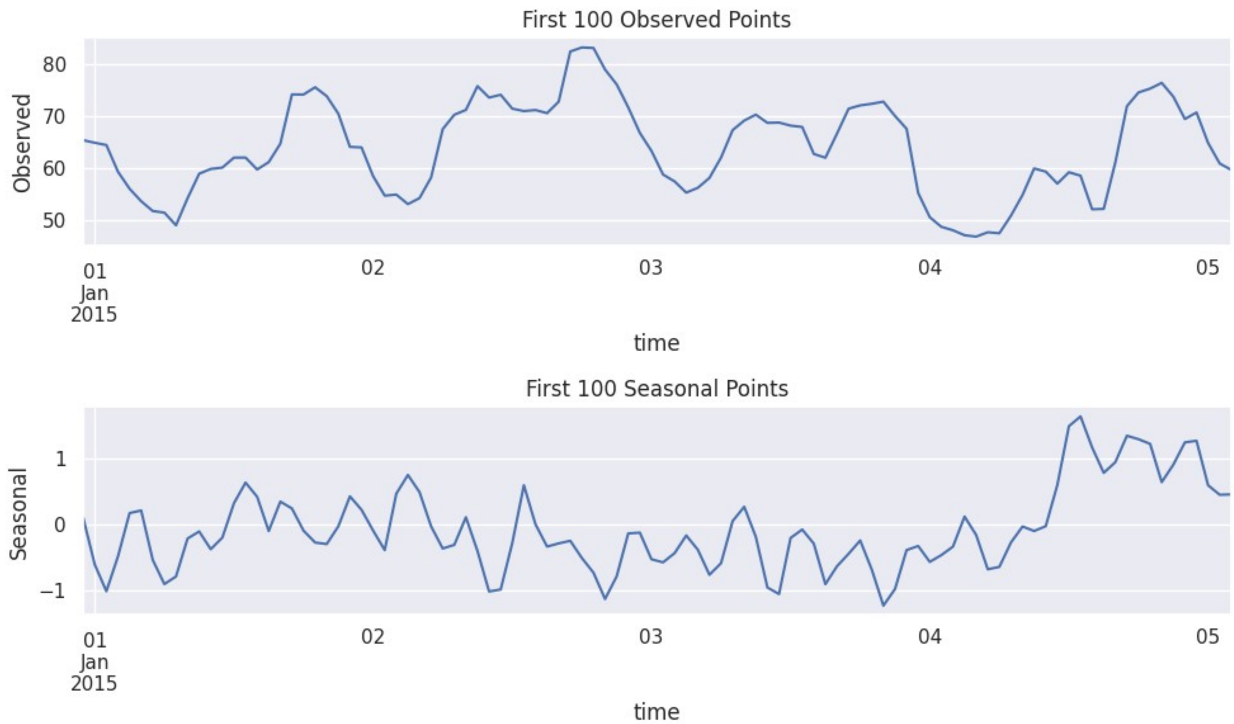



```
from statsmodels.tsa.seasonal import seasonal_decompose
decom_data = data_weather_energy[['price actual']].copy()
decomposition = seasonal_decompose(decom_data, model='additive',
period=365)

# Plot the decomposition
fig, axes = plt.subplots(4, 1, figsize=(10, 8))
decomposition.observed.plot(ax=axes[0], title='Observed')
decomposition.trend.plot(ax=axes[1], title='Trend')
decomposition.seasonal.plot(ax=axes[2], title='Seasonal')
decomposition.resid.plot(ax=axes[3], title='Residual')
plt.tight_layout()
plt.show()
```




```
# Import necessary libraries
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt

# Perform seasonal decomposition (copy this from your 'ipython-input-
0-cd1c58b197a9' cell)
decom_data = data_weather_energy[['price actual']].copy()
decomposition = seasonal_decompose(decom_data, model='additive',
period=365)

# Now create your plots
fig, axes = plt.subplots(2, 1, figsize=(10, 6))

# Plot the first 100 points of the observed component
decomposition.observed[:100].plot(ax=axes[0], title='First 100
Observed Points')
axes[0].set_ylabel('Observed')

# Plot the first 100 points of the seasonal component
decomposition.seasonal[:100].plot(ax=axes[1], title='First 100
```

```
Seasonal Points')
axes[1].set_ylabel('Seasonal')

plt.tight_layout()
plt.show()
```



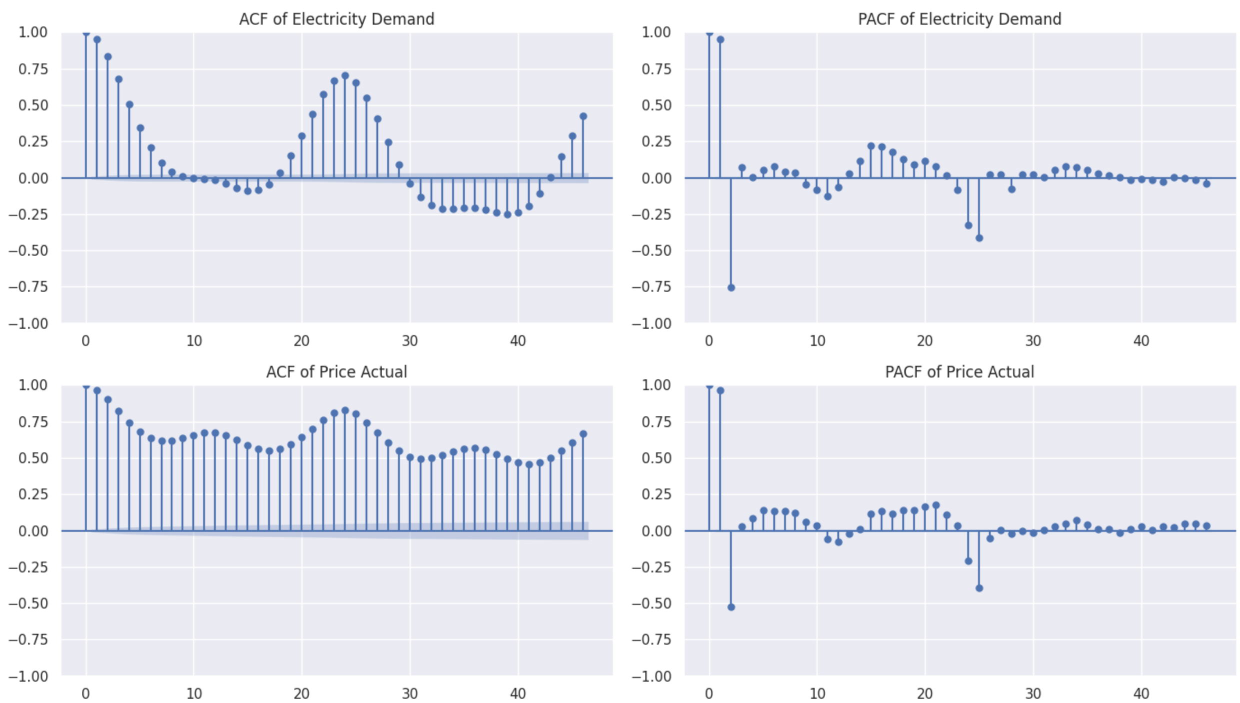

```
from statsmodels.tsa.stattools import adfuller
result = adfuller(data_weather_energy[['price actual']])
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:', result[4])
```

```
ADF Statistic: -9.147016232851199
p-value: 2.750493484934111e-15
Critical Values: {'1%': -3.4305367814665044, '5%': -
2.8616225527935106, '10%': -2.566813940257257}
```

```
from statsmodels.tsa.stattools import adfuller
result = adfuller(data_weather_energy[['total load actual']])
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:', result[4])
```

```
ADF Statistic: -21.42031575696054
p-value: 0.0
Critical Values: {'1%': -3.43053679213716, '5%': -2.8616225575095284,
'10%': -2.566813942767471}
```

```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
#  'total load actual' and 'price actual' are the columns we want to analyze
demand_series = data_weather_energy['total load actual']
price_series = data_weather_energy['price actual']

# Plot ACF and PACF for Electricity Demand
fig, axes = plt.subplots(2, 2, figsize=(14, 8))

# ACF and PACF for electricity demand
plot_acf(demand_series.dropna(), ax=axes[0, 0], title='ACF of Electricity Demand')
plot_pacf(demand_series.dropna(), ax=axes[0, 1], title='PACF of Electricity Demand')

# ACF and PACF for price actual
plot_acf(price_series.dropna(), ax=axes[1, 0], title='ACF of Price Actual')
plot_pacf(price_series.dropna(), ax=axes[1, 1], title='PACF of Price Actual')

plt.tight_layout()
plt.show()
```




```python
import tensorflow as tf

from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
```

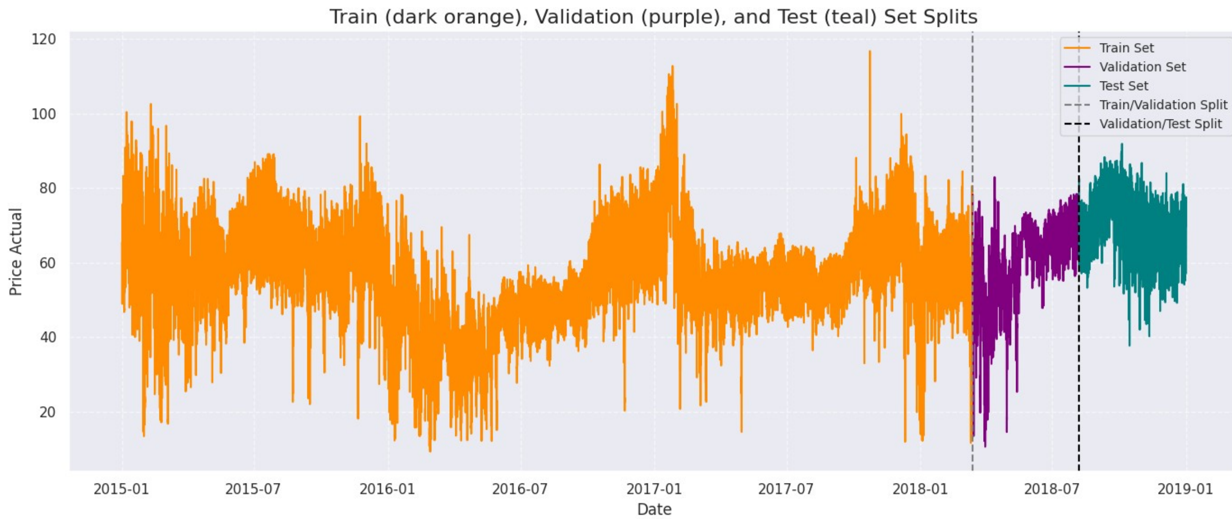
```
from sklearn.pipeline import make_pipeline
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Assuming df_weather_energy is your main dataframe
X = data_weather_energy.drop(['price actual'], axis=1)  # Features
y = data_weather_energy['price actual']  # Target variable


def apply_PCA(X_input, cum_variance=0.8, if_apply=True):

    if if_apply:
        # Create a pipeline that scales data and applies PCA
        pca = PCA(n_components=cum_variance)
        scaler_pca = make_pipeline(MinMaxScaler(), pca)

        # Apply scaling and PCA transformation
        X_pca = scaler_pca.fit_transform(X_input)

        # Print explained variance ratio to understand how much
variance is retained
        explained_variance = pca.explained_variance_ratio_.sum()
        print(f"PCA applied. Explained variance ratio:
{explained_variance:.2f}")

        return X_pca
    else:
        print("PCA not applied, returning original data.")
        return np.array(X_input)

params_pca = {'cum_variance': 0.8, 'if_apply': True}
X_pca = apply_PCA(X, **params_pca)

# Check and print the shape of the resulting data
print(f"Shape of X after PCA: {X_pca.shape}")

# If using this data for LSTM or other models, reshape if necessary
if len(X_pca.shape) == 1:
    # Ensure it's 2D if PCA reduces it to 1D
    X_pca = X_pca.reshape(-1, 1)

print(f"Final shape of X_pca: {X_pca.shape}")

PCA applied. Explained variance ratio: 0.81
Shape of X after PCA: (35064, 16)
Final shape of X_pca: (35064, 16)

# Step 2: Data Windowing for LSTM
def windowing(X_input, y_input, history_size):
```

```
    data = []
    labels = []
    for i in range(history_size, len(y_input)):
        data.append(X_input[i - history_size: i, :])
        labels.append(y_input[i])
    return np.array(data), np.array(labels).reshape(-1, 1)

history_size = 30  # Use past 30 days for prediction
X_lstm, y_lstm = windowing(X_pca, y.values, history_size)

train_cutoff = int(0.8*X_pca.shape[0])
val_cutoff   = int(0.9*X_pca.shape[0])

scaler_y = MinMaxScaler()
# Reshape y to a 2D array before fitting the scaler
scaler_y.fit(y[:train_cutoff].values.reshape(-1, 1))
y_norm = scaler_y.transform(y.values.reshape(-1, 1))


train_cutoff = int(0.8 * X_lstm.shape[0])
val_cutoff = int(0.9 * X_lstm.shape[0])

X_train = X_lstm[:train_cutoff]
X_val = X_lstm[train_cutoff:val_cutoff]
X_test = X_lstm[val_cutoff:]

y_train = y_lstm[:train_cutoff]
y_val = y_lstm[train_cutoff:val_cutoff]
y_test = y_lstm[val_cutoff:]

# Step 4: Build LSTM Model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(history_size,
X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/
rnn.py:204: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
```

```
hist_size= 24
data_norm = np.concatenate((X_pca,y_norm), axis = 1)
```

```
X_train, y_train =
windowing(data_norm[:train_cutoff,:],data_norm[:train_cutoff,-1],
hist_size)
X_val, y_val     =
windowing(data_norm[train_cutoff :val_cutoff,:],data_norm[train_cutoff
:val_cutoff,-1], hist_size)
X_test, y_test   =
windowing(data_norm[val_cutoff :,:],data_norm[val_cutoff:,-1],
hist_size)

fig, axes = plt.subplots(figsize=(14, 6))

# Plotting the train set in dark orange
axes.plot(data_weather_energy['price actual'].iloc[:train_cutoff],
color='darkorange', label='Train Set')

# Plotting the validation set in purple
axes.plot(data_weather_energy['price actual'].iloc[train_cutoff +
1:val_cutoff], color='purple', label='Validation Set')

# Plotting the test set in teal
axes.plot(data_weather_energy['price actual'].iloc[val_cutoff + 1:],
color='teal', label='Test Set')

# Adding vertical lines to indicate splits
axes.axvline(x=data_weather_energy.index[train_cutoff], color='gray',
linestyle='--', label='Train/Validation Split')
axes.axvline(x=data_weather_energy.index[val_cutoff], color='black',
linestyle='--', label='Validation/Test Split')

# Adding title and labels
axes.set_title('Train (dark orange), Validation (purple), and Test
(teal) Set Splits', fontsize=16)
axes.set_xlabel('Date', fontsize=12)
axes.set_ylabel('Price Actual', fontsize=12)

# Adding legend for clarity
axes.legend(loc='best', fontsize=10)

# Adding grid for better readability
axes.grid(True, linestyle='--', alpha=0.6)

# Display the plot
plt.tight_layout()
plt.show()
```




```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Flatten,
BatchNormalization

def base_model_lstm():
    model = Sequential()

    # First LSTM layer with return_sequences=True for stacking layers
    model.add(LSTM(units=64, return_sequences=True, activation='relu',
input_shape=X_train.shape[-2:]))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    # Second LSTM layer (optional, can be removed if overfitting)
    model.add(LSTM(units=32, return_sequences=False,
activation='relu'))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    # Flattening the output of the LSTM before feeding it to Dense
layers
    model.add(Flatten())

    # Dense layers for additional learning capacity
    model.add(Dense(units=128, activation='relu'))
    model.add(Dropout(0.3))  # Increased dropout to reduce overfitting
    model.add(BatchNormalization())

    # Final output layer
    model.add(Dense(1))

    return model

# Instantiate and compile the model
```

```
lstm_model = base_model_lstm()
lstm_model.compile(optimizer='adam', loss='mean_absolute_error',
metrics=['mae'])

# Display model summary
lstm_model.summary()
```

```
Model: "sequential_5"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ lstm_10 (LSTM)                        │ (None, 24, 64)              │
20,992 │
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_9 (Dropout)                   │ (None, 24, 64)              │
0 │
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_6                 │ (None, 24, 64)              │
256 │
│ (BatchNormalization)                  │                             │
│
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ lstm_11 (LSTM)                        │ (None, 32)                  │
12,416 │
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_10 (Dropout)                  │ (None, 32)                  │
0 │
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_7                 │ (None, 32)                  │
128 │
│ (BatchNormalization)                  │                             │
│
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ flatten_2 (Flatten)                   │ (None, 32)                  │
0 │
├───────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_7 (Dense)                       │ (None, 128)                 │
4,224 │
├───────────────────────────────────────┼─────────────────────────────┼
```

```
──────────────────┤
│ dropout_11 (Dropout)                      │ (None, 128)                         │
0 │
├───────────────────────────────────────────┼─────────────────────────────────────┼
──────────────────┤
│ batch_normalization_8                     │ (None, 128)                         │
512 │
│ (BatchNormalization)                      │                                     │
│
├───────────────────────────────────────────┼─────────────────────────────────────┼
──────────────────┤
│ dense_8 (Dense)                           │ (None, 1)                           │
129 │
└───────────────────────────────────────────┴─────────────────────────────────────┴
──────────────────┘

 Total params: 38,657 (151.00 KB)

 Trainable params: 38,209 (149.25 KB)

 Non-trainable params: 448 (1.75 KB)
```

```
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, ModelCheckpoint

# Define training parameters
epochs = 50             # Number of epochs (adjust as needed)
batch_size = 32         # Batch size (adjust as needed)
learning_rate = 0.001   # Learning rate (adjust as needed)

# Define callbacks
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,                # Stop if no improvement after 10
epochs
    restore_best_weights=True
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,                # Reduce learning rate by a factor of
0.2
    patience=5,                # If no improvement after 5 epochs,
reduce LR
    min_lr=1e-5               # Set a lower bound for the learning
rate
)

model_checkpoint = ModelCheckpoint(
    'best_lstm_model.keras',
    monitor='val_loss',
```

```
    save_best_only=True,
    verbose=1
)

# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
lstm_model.compile(optimizer=optimizer, loss='mean_absolute_error')

# Fit the model
history = lstm_model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    epochs=epochs,             # Use the defined number of epochs
    batch_size=batch_size,
    callbacks=[early_stopping, reduce_lr, model_checkpoint],
    verbose=2
)

# Model Summary
lstm_model.summary()
```

```
Epoch 1/50

Epoch 1: val_loss improved from inf to 0.09751, saving model to
best_lstm_model.keras
876/876 - 18s - 20ms/step - loss: 0.2657 - val_loss: 0.0975 -
learning_rate: 0.0010
Epoch 2/50

Epoch 2: val_loss improved from 0.09751 to 0.06262, saving model to
best_lstm_model.keras
876/876 - 13s - 15ms/step - loss: 0.0802 - val_loss: 0.0626 -
learning_rate: 0.0010
Epoch 3/50

Epoch 3: val_loss improved from 0.06262 to 0.05615, saving model to
best_lstm_model.keras
876/876 - 10s - 11ms/step - loss: 0.0648 - val_loss: 0.0562 -
learning_rate: 0.0010
Epoch 4/50

Epoch 4: val_loss improved from 0.05615 to 0.04351, saving model to
best_lstm_model.keras
876/876 - 7s - 8ms/step - loss: 0.0580 - val_loss: 0.0435 -
learning_rate: 0.0010
Epoch 5/50

Epoch 5: val_loss did not improve from 0.04351
876/876 - 6s - 7ms/step - loss: 0.0527 - val_loss: 0.0564 -
```

```
learning_rate: 0.0010
Epoch 6/50

Epoch 6: val_loss improved from 0.04351 to 0.04301, saving model to
best_lstm_model.keras
876/876 - 10s - 12ms/step - loss: 0.0485 - val_loss: 0.0430 -
learning_rate: 0.0010
Epoch 7/50

Epoch 7: val_loss improved from 0.04301 to 0.03370, saving model to
best_lstm_model.keras
876/876 - 7s - 8ms/step - loss: 0.0456 - val_loss: 0.0337 -
learning_rate: 0.0010
Epoch 8/50

Epoch 8: val_loss improved from 0.03370 to 0.03338, saving model to
best_lstm_model.keras
876/876 - 6s - 7ms/step - loss: 0.0431 - val_loss: 0.0334 -
learning_rate: 0.0010
Epoch 9/50

Epoch 9: val_loss did not improve from 0.03338
876/876 - 10s - 12ms/step - loss: 0.0406 - val_loss: 0.0421 -
learning_rate: 0.0010
Epoch 10/50

Epoch 10: val_loss did not improve from 0.03338
876/876 - 11s - 12ms/step - loss: 0.0389 - val_loss: 0.0366 -
learning_rate: 0.0010
Epoch 11/50

Epoch 11: val_loss did not improve from 0.03338
876/876 - 10s - 12ms/step - loss: 0.0383 - val_loss: 0.0368 -
learning_rate: 0.0010
Epoch 12/50

Epoch 12: val_loss improved from 0.03338 to 0.03267, saving model to
best_lstm_model.keras
876/876 - 10s - 11ms/step - loss: 0.0380 - val_loss: 0.0327 -
learning_rate: 0.0010
Epoch 13/50

Epoch 13: val_loss improved from 0.03267 to 0.01832, saving model to
best_lstm_model.keras
876/876 - 6s - 7ms/step - loss: 0.0365 - val_loss: 0.0183 -
learning_rate: 0.0010
Epoch 14/50

Epoch 14: val_loss did not improve from 0.01832
876/876 - 11s - 12ms/step - loss: 0.0355 - val_loss: 0.0333 -
```

```
learning_rate: 0.0010
Epoch 15/50

Epoch 15: val_loss did not improve from 0.01832
876/876 - 6s - 7ms/step - loss: 0.0353 - val_loss: 0.0264 -
learning_rate: 0.0010
Epoch 16/50

Epoch 16: val_loss did not improve from 0.01832
876/876 - 10s - 12ms/step - loss: 0.0352 - val_loss: 0.0240 -
learning_rate: 0.0010
Epoch 17/50

Epoch 17: val_loss did not improve from 0.01832
876/876 - 7s - 8ms/step - loss: 0.0349 - val_loss: 0.0270 -
learning_rate: 0.0010
Epoch 18/50

Epoch 18: val_loss did not improve from 0.01832
876/876 - 10s - 11ms/step - loss: 0.0342 - val_loss: 0.0300 -
learning_rate: 0.0010
Epoch 19/50

Epoch 19: val_loss did not improve from 0.01832
876/876 - 10s - 11ms/step - loss: 0.0326 - val_loss: 0.0279 -
learning_rate: 2.0000e-04
Epoch 20/50

Epoch 20: val_loss did not improve from 0.01832
876/876 - 10s - 12ms/step - loss: 0.0324 - val_loss: 0.0353 -
learning_rate: 2.0000e-04
Epoch 21/50

Epoch 21: val_loss did not improve from 0.01832
876/876 - 10s - 12ms/step - loss: 0.0321 - val_loss: 0.0356 -
learning_rate: 2.0000e-04
Epoch 22/50

Epoch 22: val_loss did not improve from 0.01832
876/876 - 11s - 12ms/step - loss: 0.0325 - val_loss: 0.0299 -
learning_rate: 2.0000e-04
Epoch 23/50

Epoch 23: val_loss did not improve from 0.01832
876/876 - 6s - 7ms/step - loss: 0.0318 - val_loss: 0.0325 -
learning_rate: 2.0000e-04

Model: "sequential_5"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
```

```
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ lstm_10 (LSTM)                       │ (None, 24, 64)              │
20,992 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_9 (Dropout)                  │ (None, 24, 64)              │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_6                │ (None, 24, 64)              │
256 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ lstm_11 (LSTM)                       │ (None, 32)                  │
12,416 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_10 (Dropout)                 │ (None, 32)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_7                │ (None, 32)                  │
128 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ flatten_2 (Flatten)                  │ (None, 32)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_7 (Dense)                      │ (None, 128)                 │
4,224 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_11 (Dropout)                 │ (None, 128)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_8                │ (None, 128)                 │
512 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼
```

```
──────────────────┘
│ dense_8 (Dense)                      │ (None, 1)                    │
129 │
└──────────────────────────────────────┴──────────────────────────────┴
──────────────────┘

 Total params: 115,077 (449.52 KB)

 Trainable params: 38,209 (149.25 KB)

 Non-trainable params: 448 (1.75 KB)

 Optimizer params: 76,420 (298.52 KB)
```

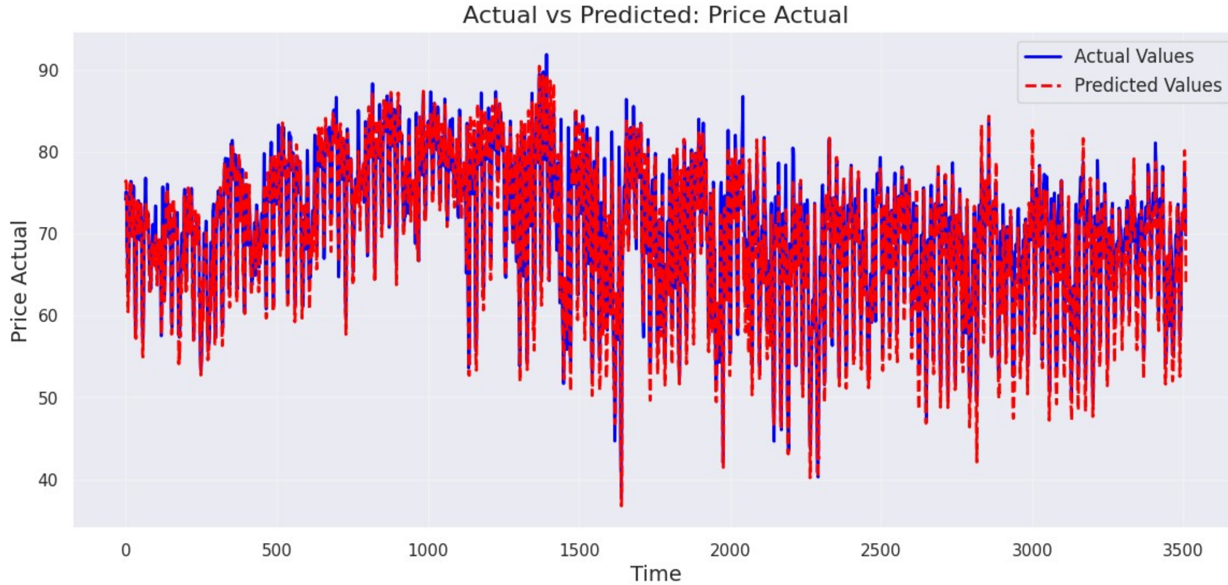
```
y_pred = lstm_model.predict(X_test)

# Predictions and actual values (inverse transformed if necessary)
y_pred_actual = scaler_y.inverse_transform(y_pred)
y_test_actual = scaler_y.inverse_transform(y_test)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import numpy as np

y_pred = lstm_model.predict(X_test)

# Predictions and actual values (inverse transformed if necessary)
y_pred_actual = scaler_y.inverse_transform(y_pred)
y_test_actual = scaler_y.inverse_transform(y_test)

# Calculate metrics
mae = mean_absolute_error(y_test_actual, y_pred_actual)
rmse = np.sqrt(mean_squared_error(y_test_actual, y_pred_actual))
mape = np.mean(np.abs((y_test_actual - y_pred_actual) /
y_test_actual)) * 100
r2 = r2_score(y_test_actual, y_pred_actual)

# Define accuracy as (1 - MAPE)
accuracy = 100 - mape

# Print results
print('---------------- Evaluation Metrics ----------------')
print(f'MAE: {mae:.3f}')
print(f'RMSE: {rmse:.3f}')
print(f'MAPE: {mape:.2f}%')
print(f'R^2: {r2:.3f}')
print(f'Accuracy (1 - MAPE): {accuracy:.2f}%')
print('----------------------------------------------------')
```

```
---------------- Evaluation Metrics ----------------
MAE: 2.023
RMSE: 2.615
MAPE: 2.95%
R^2: 0.907
Accuracy (1 - MAPE): 97.05%
---------------------------------------------------
```

```python
import matplotlib.pyplot as plt

def plot_actual_vs_predicted(y_actual, y_predicted, title="Actual vs
Predicted"):
    """
    Plots actual vs predicted values for visual comparison.

    Parameters:
    - y_actual: Actual values (array-like)
    - y_predicted: Predicted values (array-like)
    - title: Title of the plot (string)
    """
    plt.figure(figsize=(14, 6))

    # Plot actual and predicted values
    plt.plot(y_actual, label='Actual Values', color='blue',
linewidth=2)
    plt.plot(y_predicted, label='Predicted Values', color='red',
linestyle='--', linewidth=2)

    # Add title, labels, and legend
    plt.title(title, fontsize=16)
    plt.xlabel("Time", fontsize=14)
    plt.ylabel("Price Actual", fontsize=14)
    plt.legend(fontsize=12)
    plt.grid(alpha=0.3)

    # Show the plot
    plt.show()

# Example usage:
# Assuming y_test_actual and y_pred_actual contain the respective
values
plot_actual_vs_predicted(y_test_actual, y_pred_actual, title="Actual
vs Predicted: Price Actual")
```



Total Load **Actual** Prediction

```
# Assuming df_weather_energy is your main dataframe
X = data_weather_energy.drop(['total load actual'], axis=1)  #
Features
y = data_weather_energy['total load actual']  # Target variable

def apply_PCA(X_input, cum_variance=0.8, if_apply=True):

    if if_apply:
        # Create a pipeline that scales data and applies PCA
        pca = PCA(n_components=cum_variance)
        scaler_pca = make_pipeline(MinMaxScaler(), pca)

        # Apply scaling and PCA transformation
        X_pca = scaler_pca.fit_transform(X_input)

        # Print explained variance ratio to understand how much
variance is retained
        explained_variance = pca.explained_variance_ratio_.sum()
        print(f"PCA applied. Explained variance ratio:
{explained_variance:.2f}")

        return X_pca
    else:
        print("PCA not applied, returning original data.")
        return np.array(X_input)

params_pca = {'cum_variance': 0.8, 'if_apply': True}
X_pca = apply_PCA(X, **params_pca)

# Check and print the shape of the resulting data
```

```python
print(f"Shape of X after PCA: {X_pca.shape}")

# If using this data for LSTM or other models, reshape if necessary
if len(X_pca.shape) == 1:
    # Ensure it's 2D if PCA reduces it to 1D
    X_pca = X_pca.reshape(-1, 1)

print(f"Final shape of X_pca: {X_pca.shape}")
```

```
PCA applied. Explained variance ratio: 0.81
Shape of X after PCA: (35064, 16)
Final shape of X_pca: (35064, 16)
```

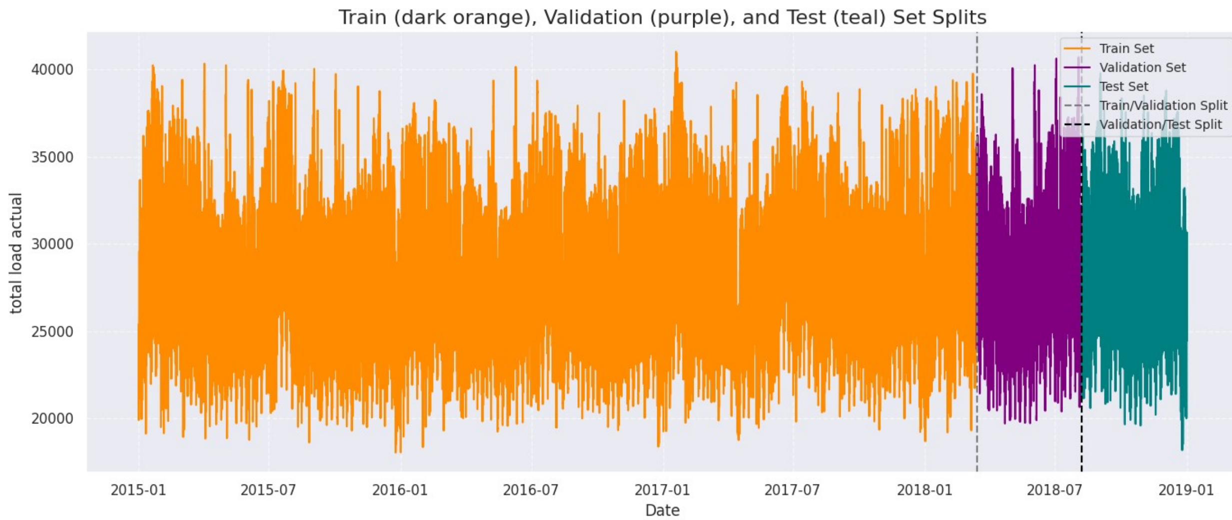
```python
# Step 2: Data Windowing for LSTM
def windowing(X_input, y_input, history_size):
    data = []
    labels = []
    for i in range(history_size, len(y_input)):
        data.append(X_input[i - history_size: i, :])
        labels.append(y_input[i])
    return np.array(data), np.array(labels).reshape(-1, 1)

history_size = 30  # Use past 30 days for prediction
X_lstm, y_lstm = windowing(X_pca, y.values, history_size)

train_cutoff = int(0.8*X_pca.shape[0])
val_cutoff   = int(0.9*X_pca.shape[0])

scaler_y = MinMaxScaler()
# Reshape y to a 2D array before fitting the scaler
scaler_y.fit(y[:train_cutoff].values.reshape(-1, 1))
y_norm = scaler_y.transform(y.values.reshape(-1, 1))

train_cutoff = int(0.8 * X_lstm.shape[0])
val_cutoff = int(0.9 * X_lstm.shape[0])

X_train = X_lstm[:train_cutoff]
X_val = X_lstm[train_cutoff:val_cutoff]
X_test = X_lstm[val_cutoff:]

y_train = y_lstm[:train_cutoff]
y_val = y_lstm[train_cutoff:val_cutoff]
y_test = y_lstm[val_cutoff:]

# Step 4: Build LSTM Model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(history_size,
X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dense(1))
```

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/
rnn.py:204: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
```

```
hist_size= 24
data_norm = np.concatenate((X_pca,y_norm), axis = 1)

X_train, y_train =
windowing(data_norm[:train_cutoff,:],data_norm[:train_cutoff,-1],
hist_size)
X_val, y_val     =
windowing(data_norm[train_cutoff :val_cutoff,:],data_norm[train_cutoff
:val_cutoff,-1], hist_size)
X_test, y_test   =
windowing(data_norm[val_cutoff :,:],data_norm[val_cutoff:,-1],
hist_size)

fig, axes = plt.subplots(figsize=(14, 6))

# Plotting the train set in dark orange
axes.plot(data_weather_energy['total load
actual'].iloc[:train_cutoff], color='darkorange', label='Train Set')

# Plotting the validation set in purple
axes.plot(data_weather_energy['total load actual'].iloc[train_cutoff +
1:val_cutoff], color='purple', label='Validation Set')

# Plotting the test set in teal
axes.plot(data_weather_energy['total load actual'].iloc[val_cutoff +
1:], color='teal', label='Test Set')

# Adding vertical lines to indicate splits
axes.axvline(x=data_weather_energy.index[train_cutoff], color='gray',
linestyle='--', label='Train/Validation Split')
axes.axvline(x=data_weather_energy.index[val_cutoff], color='black',
linestyle='--', label='Validation/Test Split')

# Adding title and labels
axes.set_title('Train (dark orange), Validation (purple), and Test
(teal) Set Splits', fontsize=16)
axes.set_xlabel('Date', fontsize=12)
axes.set_ylabel('total load actual', fontsize=12)

# Adding legend for clarity
```

```
axes.legend(loc='best', fontsize=10)

# Adding grid for better readability
axes.grid(True, linestyle='--', alpha=0.6)

# Display the plot
plt.tight_layout()
plt.show()
```




```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Flatten,
BatchNormalization

def base_model_lstm():
    model = Sequential()

    # First LSTM layer with return_sequences=True for stacking layers
    model.add(LSTM(units=64, return_sequences=True, activation='relu',
input_shape=X_train.shape[-2:]))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    # Second LSTM layer (optional, can be removed if overfitting)
    model.add(LSTM(units=32, return_sequences=False,
activation='relu'))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    # Flattening the output of the LSTM before feeding it to Dense
layers
    model.add(Flatten())

    # Dense layers for additional learning capacity
```

```
    model.add(Dense(units=128, activation='relu'))
    model.add(Dropout(0.3))  # Increased dropout to reduce overfitting
    model.add(BatchNormalization())

    # Final output layer
    model.add(Dense(1))

    return model

# Instantiate and compile the model
lstm_model = base_model_lstm()
lstm_model.compile(optimizer='adam', loss='mean_absolute_error',
metrics=['mae'])

# Display model summary
lstm_model.summary()
```

```
Model: "sequential_7"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ lstm_14 (LSTM)                        │ (None, 24, 64)               │
20,992 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dropout_13 (Dropout)                  │ (None, 24, 64)               │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ batch_normalization_9                 │ (None, 24, 64)               │
256 │
│ (BatchNormalization)                  │                              │
│
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ lstm_15 (LSTM)                        │ (None, 32)                   │
12,416 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dropout_14 (Dropout)                  │ (None, 32)                   │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ batch_normalization_10                │ (None, 32)                   │
128 │
│ (BatchNormalization)                  │                              │
```

```
│
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ flatten_3 (Flatten)                  │ (None, 32)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_10 (Dense)                     │ (None, 128)                 │
4,224 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_15 (Dropout)                 │ (None, 128)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_11               │ (None, 128)                 │
512 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_11 (Dense)                     │ (None, 1)                   │
129 │
└──────────────────────────────────────┴─────────────────────────────┴
─────────────────┘

 Total params: 38,657 (151.00 KB)

 Trainable params: 38,209 (149.25 KB)

 Non-trainable params: 448 (1.75 KB)

from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, ModelCheckpoint

# Define training parameters
epochs = 50             # Number of epochs (adjust as needed)
batch_size = 32         # Batch size (adjust as needed)
learning_rate = 0.001   # Learning rate (adjust as needed)

# Define callbacks
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,                # Stop if no improvement after 10
epochs
    restore_best_weights=True
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
```

```
    factor=0.2,                # Reduce learning rate by a factor of
0.2
    patience=5,                # If no improvement after 5 epochs,
reduce LR
    min_lr=1e-5               # Set a lower bound for the learning
rate
)

model_checkpoint = ModelCheckpoint(
    'best_lstm_model.keras',
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)

# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
lstm_model.compile(optimizer=optimizer, loss='mean_absolute_error')

# Fit the model
history = lstm_model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    epochs=epochs,             # Use the defined number of epochs
    batch_size=batch_size,
    callbacks=[early_stopping, reduce_lr, model_checkpoint],
    verbose=2
)

# Model Summary
lstm_model.summary()
```

```
Epoch 1/50

Epoch 1: val_loss improved from inf to 0.14029, saving model to
best_lstm_model.keras
876/876 - 16s - 19ms/step - loss: 0.3510 - val_loss: 0.1403 -
learning_rate: 0.0010
Epoch 2/50

Epoch 2: val_loss improved from 0.14029 to 0.07862, saving model to
best_lstm_model.keras
876/876 - 13s - 15ms/step - loss: 0.1378 - val_loss: 0.0786 -
learning_rate: 0.0010
Epoch 3/50

Epoch 3: val_loss improved from 0.07862 to 0.05165, saving model to
best_lstm_model.keras
876/876 - 7s - 8ms/step - loss: 0.0863 - val_loss: 0.0516 -
```

```
learning_rate: 0.0010
Epoch 4/50

Epoch 4: val_loss did not improve from 0.05165
876/876 - 6s - 7ms/step - loss: 0.0697 - val_loss: 0.0517 -
learning_rate: 0.0010
Epoch 5/50

Epoch 5: val_loss improved from 0.05165 to 0.04524, saving model to
best_lstm_model.keras
876/876 - 7s - 8ms/step - loss: 0.0615 - val_loss: 0.0452 -
learning_rate: 0.0010
Epoch 6/50

Epoch 6: val_loss improved from 0.04524 to 0.04184, saving model to
best_lstm_model.keras
876/876 - 6s - 7ms/step - loss: 0.0582 - val_loss: 0.0418 -
learning_rate: 0.0010
Epoch 7/50

Epoch 7: val_loss improved from 0.04184 to 0.03509, saving model to
best_lstm_model.keras
876/876 - 11s - 12ms/step - loss: 0.0554 - val_loss: 0.0351 -
learning_rate: 0.0010
Epoch 8/50

Epoch 8: val_loss improved from 0.03509 to 0.02706, saving model to
best_lstm_model.keras
876/876 - 6s - 7ms/step - loss: 0.0526 - val_loss: 0.0271 -
learning_rate: 0.0010
Epoch 9/50

Epoch 9: val_loss did not improve from 0.02706
876/876 - 10s - 11ms/step - loss: 0.0522 - val_loss: 0.0324 -
learning_rate: 0.0010
Epoch 10/50

Epoch 10: val_loss did not improve from 0.02706
876/876 - 10s - 12ms/step - loss: 0.0508 - val_loss: 0.0303 -
learning_rate: 0.0010
Epoch 11/50

Epoch 11: val_loss did not improve from 0.02706
876/876 - 7s - 7ms/step - loss: 0.0497 - val_loss: 0.0350 -
learning_rate: 0.0010
Epoch 12/50

Epoch 12: val_loss did not improve from 0.02706
876/876 - 6s - 7ms/step - loss: 0.0492 - val_loss: 0.0353 -
learning_rate: 0.0010
```

```
Epoch 13/50

Epoch 13: val_loss did not improve from 0.02706
876/876 - 7s - 7ms/step - loss: 0.0478 - val_loss: 0.0284 -
learning_rate: 0.0010
Epoch 14/50

Epoch 14: val_loss improved from 0.02706 to 0.02097, saving model to
best_lstm_model.keras
876/876 - 6s - 7ms/step - loss: 0.0437 - val_loss: 0.0210 -
learning_rate: 2.0000e-04
Epoch 15/50

Epoch 15: val_loss did not improve from 0.02097
876/876 - 7s - 8ms/step - loss: 0.0440 - val_loss: 0.0224 -
learning_rate: 2.0000e-04
Epoch 16/50

Epoch 16: val_loss did not improve from 0.02097
876/876 - 10s - 11ms/step - loss: 0.0436 - val_loss: 0.0280 -
learning_rate: 2.0000e-04
Epoch 17/50

Epoch 17: val_loss did not improve from 0.02097
876/876 - 7s - 7ms/step - loss: 0.0444 - val_loss: 0.0222 -
learning_rate: 2.0000e-04
Epoch 18/50

Epoch 18: val_loss improved from 0.02097 to 0.01714, saving model to
best_lstm_model.keras
876/876 - 10s - 12ms/step - loss: 0.0436 - val_loss: 0.0171 -
learning_rate: 2.0000e-04
Epoch 19/50

Epoch 19: val_loss did not improve from 0.01714
876/876 - 10s - 11ms/step - loss: 0.0435 - val_loss: 0.0214 -
learning_rate: 2.0000e-04
Epoch 20/50

Epoch 20: val_loss did not improve from 0.01714
876/876 - 6s - 7ms/step - loss: 0.0433 - val_loss: 0.0238 -
learning_rate: 2.0000e-04
Epoch 21/50

Epoch 21: val_loss did not improve from 0.01714
876/876 - 6s - 7ms/step - loss: 0.0424 - val_loss: 0.0253 -
learning_rate: 2.0000e-04
Epoch 22/50

Epoch 22: val_loss did not improve from 0.01714
```

```
876/876 - 10s - 12ms/step - loss: 0.0422 - val_loss: 0.0304 -
learning_rate: 2.0000e-04
Epoch 23/50

Epoch 23: val_loss did not improve from 0.01714
876/876 - 10s - 12ms/step - loss: 0.0427 - val_loss: 0.0277 -
learning_rate: 2.0000e-04
Epoch 24/50

Epoch 24: val_loss did not improve from 0.01714
876/876 - 7s - 7ms/step - loss: 0.0421 - val_loss: 0.0237 -
learning_rate: 4.0000e-05
Epoch 25/50

Epoch 25: val_loss did not improve from 0.01714
876/876 - 10s - 11ms/step - loss: 0.0418 - val_loss: 0.0295 -
learning_rate: 4.0000e-05
Epoch 26/50

Epoch 26: val_loss did not improve from 0.01714
876/876 - 6s - 7ms/step - loss: 0.0427 - val_loss: 0.0264 -
learning_rate: 4.0000e-05
Epoch 27/50

Epoch 27: val_loss did not improve from 0.01714
876/876 - 11s - 12ms/step - loss: 0.0424 - val_loss: 0.0254 -
learning_rate: 4.0000e-05
Epoch 28/50

Epoch 28: val_loss did not improve from 0.01714
876/876 - 10s - 11ms/step - loss: 0.0413 - val_loss: 0.0284 -
learning_rate: 4.0000e-05

Model: "sequential_7"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape              ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
━━━━━━━━━━━━━━━━━┩
│ lstm_14 (LSTM)                        │ (None, 24, 64)            │
20,992 │
├───────────────────────────────────────┼───────────────────────────┤
─────────────────┤
│ dropout_13 (Dropout)                  │ (None, 24, 64)            │
0 │
├───────────────────────────────────────┼───────────────────────────┤
─────────────────┤
│ batch_normalization_9                 │ (None, 24, 64)            │
```

```
256 │
│ (BatchNormalization)                    │                             │
│
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ lstm_15 (LSTM)                          │ (None, 32)                  │
12,416 │
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_14 (Dropout)                    │ (None, 32)                  │
0 │
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_10                  │ (None, 32)                  │
128 │
│ (BatchNormalization)                    │                             │
│
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ flatten_3 (Flatten)                     │ (None, 32)                  │
0 │
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_10 (Dense)                        │ (None, 128)                 │
4,224 │
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dropout_15 (Dropout)                    │ (None, 128)                 │
0 │
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ batch_normalization_11                  │ (None, 128)                 │
512 │
│ (BatchNormalization)                    │                             │
│
├─────────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ dense_11 (Dense)                        │ (None, 1)                   │
129 │
└─────────────────────────────────────────┴─────────────────────────────┴
─────────────────┘

 Total params: 115,077 (449.52 KB)

 Trainable params: 38,209 (149.25 KB)

 Non-trainable params: 448 (1.75 KB)

 Optimizer params: 76,420 (298.52 KB)
```

```
y_pred = lstm_model.predict(X_test)

# Predictions and actual values (inverse transformed if necessary)
y_pred_actual = scaler_y.inverse_transform(y_pred)
y_test_actual = scaler_y.inverse_transform(y_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import numpy as np

y_pred = lstm_model.predict(X_test)

# Predictions and actual values (inverse transformed if necessary)
y_pred_actual = scaler_y.inverse_transform(y_pred)
y_test_actual = scaler_y.inverse_transform(y_test)

# Calculate metrics
mae = mean_absolute_error(y_test_actual, y_pred_actual)
rmse = np.sqrt(mean_squared_error(y_test_actual, y_pred_actual))
mape = np.mean(np.abs((y_test_actual - y_pred_actual) /
y_test_actual)) * 100
r2 = r2_score(y_test_actual, y_pred_actual)

# Define accuracy as (1 - MAPE)
accuracy = 100 - mape

# Print results
print('---------------- Evaluation Metrics ----------------')
print(f'MAE: {mae:.3f}')
print(f'RMSE: {rmse:.3f}')
print(f'MAPE: {mape:.2f}%')
print(f'R^2: {r2:.3f}')
print(f'Accuracy (1 - MAPE): {accuracy:.2f}%')
print('---------------------------------------------------')

---------------- Evaluation Metrics ----------------
MAE: 430.107
RMSE: 590.709
MAPE: 1.51%
R^2: 0.983
Accuracy (1 - MAPE): 98.49%
---------------------------------------------------

import matplotlib.pyplot as plt

def plot_actual_vs_predicted(y_actual, y_predicted, title="Actual vs
Predicted"):
    """
    Plots actual vs predicted values for visual comparison.

    Parameters:
```

```
    - y_actual: Actual values (array-like)
    - y_predicted: Predicted values (array-like)
    - title: Title of the plot (string)
    """
    plt.figure(figsize=(14, 6))

    # Plot actual and predicted values
    plt.plot(y_actual, label='Actual Values', color='blue',
linewidth=2)
    plt.plot(y_predicted, label='Predicted Values', color='red',
linestyle='--', linewidth=2)

    # Add title, labels, and legend
    plt.title(title, fontsize=16)
    plt.xlabel("Time", fontsize=14)
    plt.ylabel("total load actual", fontsize=14)
    plt.legend(fontsize=12)
    plt.grid(alpha=0.3)

    # Show the plot
    plt.show()

# Example usage:
# Assuming y_test_actual and y_pred_actual contain the respective
values
plot_actual_vs_predicted(y_test_actual, y_pred_actual, title="Actual
vs Predicted: total load Actual")
```