**Data Science and Decision Making Assignement 1**

---

#Brigthon Data Visualization

##Exploratory Data Analysis

```
#Brigthon Data Visualization
#Important Libraries to be used in the code
import warnings
warnings.filterwarnings('ignore')
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from matplotlib.lines import Line2D

# For seasonal decomposition
from statsmodels.tsa.seasonal import seasonal_decompose, STL
# Import additional functions needed
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.stats.diagnostic import acorr_ljungbox

#imports to get data from the GDrive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
#Mounting the path for the data
GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = os.path.join('./Project
documentation-20240129/weatherdata_for_students')
GOOGLE_DRIVE_PATH = os.path.join('gdrive',
'MyDrive',GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print('List files: ', os.listdir(GOOGLE_DRIVE_PATH))
```

```
List files:  ['colchester_001.csv', 'colchester_002.csv',
'colchester_003.csv', 'colchester_004.csv', 'colchester_005.csv',
'colchester_007.csv', 'colchester_006.csv', 'colchester_008.csv',
'colchester_009.csv', 'colchester_010.csv', 'colchester_012.csv',
'colchester_011.csv', 'colchester_013.csv', 'colchester_014.csv',
'colchester_016.csv', 'colchester_015.csv', 'colchester_018.csv',
'colchester_017.csv', 'colchester_020.csv', 'colchester_019.csv',
'colchester_021.csv', 'colchester_022.csv', 'colchester_023.csv',
'colchester_024.csv', 'colchester_025.csv', 'colchester_026.csv',
```

```
'colchester_027.csv', 'colchester_028.csv', 'colchester_029.csv',
'colchester_030.csv', 'colchester_031.csv', 'colchester_032.csv',
'colchester_034.csv', 'colchester_033.csv', 'colchester_037.csv',
'colchester_036.csv', 'colchester_035.csv', 'colchester_038.csv',
'colchester_039.csv', 'colchester_040.csv', 'colchester_042.csv',
'colchester_041.csv', 'colchester_044.csv', 'colchester_043.csv',
'colchester_045.csv', 'colchester_046.csv', 'colchester_047.csv',
'colchester_048.csv', 'colchester_049.csv', 'colchester_051.csv',
'colchester_050.csv', 'colchester_053.csv', 'colchester_052.csv',
'colchester_055.csv', 'colchester_054.csv', 'colchester_056.csv',
'colchester_057.csv', 'colchester_059.csv', 'colchester_058.csv',
'colchester_060.csv', 'colchester_062.csv', 'colchester_061.csv',
'colchester_065.csv', 'colchester_063.csv', 'colchester_064.csv',
'colchester_066.csv', 'colchester_067.csv', 'colchester_068.csv',
'colchester_069.csv', 'colchester_070.csv', 'colchester_072.csv',
'colchester_071.csv', 'colchester_073.csv', 'colchester_074.csv',
'colchester_075.csv', 'colchester_076.csv', 'colchester_077.csv',
'colchester_078.csv', 'colchester_079.csv', 'colchester_081.csv',
'colchester_080.csv', 'colchester_083.csv', 'colchester_082.csv',
'colchester_084.csv', 'colchester_085.csv', 'colchester_087.csv',
'colchester_086.csv', 'colchester_088.csv', 'colchester_089.csv',
'colchester_090.csv', 'colchester_092.csv', 'colchester_091.csv',
'colchester_094.csv', 'colchester_093.csv', 'colchester_095.csv',
'colchester_096.csv', 'colchester_097.csv', 'colchester_098.csv',
'colchester_099.csv', 'colchester_100.csv', 'colchester_101.csv',
'colchester_102.csv', 'colchester_103.csv', 'colchester_104.csv',
'colchester_105.csv', 'colchester_107.csv', 'colchester_106.csv',
'colchester_108.csv', 'colchester_109.csv', 'colchester_111.csv',
'colchester_110.csv', 'colchester_113.csv', 'colchester_112.csv',
'colchester_114.csv', 'colchester_115.csv', 'colchester_116.csv',
'colchester_118.csv', 'colchester_117.csv', 'colchester_120.csv',
'colchester_119.csv', 'colchester_122.csv', 'colchester_121.csv',
'colchester_123.csv', 'colchester_124.csv', 'colchester_125.csv',
'colchester_126.csv', 'colchester_127.csv', 'colchester_129.csv',
'colchester_128.csv', 'colchester_131.csv', 'colchester_130.csv',
'colchester_132.csv', 'colchester_133.csv', 'colchester_134.csv',
'colchester_136.csv', 'colchester_135.csv', 'colchester_137.csv',
'colchester_139.csv', 'colchester_138.csv', 'colchester_140.csv',
'colchester_141.csv', 'colchester_142.csv', 'colchester_143.csv',
'colchester_144.csv', 'colchester_145.csv', 'colchester_146.csv',
'colchester_148.csv', 'colchester_147.csv', 'colchester_149.csv',
'colchester_150.csv', 'colchester_151.csv', 'colchester_152.csv',
'colchester_153.csv', 'colchester_155.csv', 'colchester_154.csv',
'colchester_157.csv', 'colchester_156.csv', 'colchester_158.csv',
'colchester_160.csv', 'colchester_159.csv', 'colchester_161.csv',
'colchester_162.csv', 'colchester_164.csv', 'colchester_163.csv',
'colchester_165.csv', 'colchester_166.csv', 'colchester_167.csv',
'colchester_168.csv', 'colchester_169.csv', 'colchester_171.csv',
'colchester_170.csv', 'colchester_173.csv', 'colchester_172.csv',
```

```
'colchester_174.csv', 'colchester_176.csv', 'colchester_175.csv',
'colchester_177.csv', 'colchester_178.csv', 'colchester_179.csv',
'colchester_180.csv', 'colchester_181.csv', 'colchester_183.csv',
'colchester_184.csv', 'colchester_182.csv', 'colchester_186.csv',
'colchester_185.csv', 'colchester_187.csv', 'colchester_188.csv',
'colchester_189.csv', 'colchester_190.csv', 'colchester_191.csv',
'colchester_192.csv', 'colchester_193.csv', 'colchester_195.csv',
'colchester_196.csv', 'colchester_194.csv', 'brighton_003.csv',
'brighton_001.csv', 'brighton_002.csv', 'brighton_004.csv',
'brighton_005.csv', 'brighton_006.csv', 'brighton_009.csv',
'brighton_011.csv', 'brighton_014.csv', 'brighton_010.csv',
'brighton_013.csv', 'brighton_008.csv', 'brighton_007.csv',
'brighton_012.csv', 'brighton_015.csv', 'brighton_017.csv',
'brighton_018.csv', 'brighton_016.csv', 'brighton_019.csv',
'brighton_020.csv', 'brighton_021.csv', 'brighton_022.csv',
'brighton_024.csv', 'brighton_023.csv', 'brighton_025.csv',
'brighton_026.csv', 'brighton_027.csv', 'brighton_028.csv',
'brighton_029.csv', 'brighton_030.csv', 'brighton_031.csv',
'brighton_032.csv', 'brighton_033.csv', 'brighton_035.csv',
'brighton_034.csv', 'brighton_036.csv', 'brighton_037.csv',
'brighton_038.csv', 'brighton_039.csv', 'brighton_042.csv',
'brighton_040.csv', 'brighton_041.csv', 'brighton_043.csv',
'brighton_044.csv', 'brighton_045.csv', 'brighton_047.csv',
'brighton_046.csv', 'brighton_048.csv', 'brighton_049.csv',
'brighton_052.csv', 'brighton_055.csv', 'brighton_062.csv',
'brighton_056.csv', 'brighton_063.csv', 'brighton_061.csv',
'brighton_059.csv', 'brighton_060.csv', 'brighton_050.csv',
'brighton_051.csv', 'brighton_064.csv', 'brighton_053.csv',
'brighton_057.csv', 'brighton_058.csv', 'brighton_054.csv',
'brighton_071.csv', 'brighton_074.csv', 'brighton_067.csv',
'brighton_069.csv', 'brighton_081.csv', 'brighton_076.csv',
'brighton_068.csv', 'brighton_077.csv', 'brighton_073.csv',
'brighton_080.csv', 'brighton_066.csv', 'brighton_079.csv',
'brighton_075.csv', 'brighton_070.csv', 'brighton_072.csv',
'brighton_065.csv', 'brighton_078.csv', 'brighton_096.csv',
'brighton_097.csv', 'brighton_093.csv', 'brighton_082.csv',
'brighton_094.csv', 'brighton_083.csv', 'brighton_088.csv',
'brighton_100.csv', 'brighton_092.csv', 'brighton_091.csv',
'brighton_087.csv', 'brighton_099.csv', 'brighton_101.csv',
'brighton_089.csv', 'brighton_090.csv', 'brighton_084.csv',
'brighton_095.csv', 'brighton_098.csv', 'brighton_086.csv',
'brighton_085.csv', 'brighton_116.csv', 'brighton_119.csv',
'brighton_109.csv', 'brighton_111.csv', 'brighton_112.csv',
'brighton_118.csv', 'brighton_103.csv', 'brighton_110.csv',
'brighton_105.csv', 'brighton_117.csv', 'brighton_108.csv',
'brighton_114.csv', 'brighton_113.csv', 'brighton_107.csv',
'brighton_106.csv', 'brighton_102.csv', 'brighton_104.csv',
'brighton_115.csv', 'brighton_124.csv', 'brighton_128.csv',
'brighton_125.csv', 'brighton_127.csv', 'brighton_120.csv',
```

```
'brighton_123.csv', 'brighton_122.csv', 'brighton_121.csv',
'brighton_126.csv']
```

```python
import os
import pandas as pd

# Specify the folder directory where the dataset is located.
df_path = GOOGLE_DRIVE_PATH

#list of all files in the path
file_list = [file for file in os.listdir(GOOGLE_DRIVE_PATH) if
file.endswith('.csv')]

brighton_df = []

# Iterate over each CSV file
for file_ in file_list:
    # Construct the full path to the CSV file
    file_path = os.path.join(df_path, file_)

    # Check for file has an index column named '0'
    index_column = pd.read_csv(file_path, nrows=1).columns[0] == '0'

    # Change header value based on the  value of the column
    header = 1 if index_column else "infer"

    current_dataframe = pd.read_csv(file_path, header=header)

    # If the file name contains "Brighton", add its DataFrame to the
list
    if file_.startswith('brighton'):
        brighton_df.append(current_dataframe)

# Concantenate all the datasets into one
f_brighton_df = pd.concat(brighton_df, ignore_index=True)

#information about the DataFrame
print(f_brighton_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 122844 entries, 0 to 122843
Data columns (total 16 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   datetime           122844 non-null  object
 1   temp               122590 non-null  float64
 2   dew                122568 non-null  float64
 3   humidity           122575 non-null  float64
 4   precip             122565 non-null  float64
 5   precipprob         122558 non-null  float64
 6   preciptype         11277 non-null   object
```

```
 7   snow              89810 non-null   float64
 8   snowdepth         89279 non-null   float64
 9   windspeed         122583 non-null  float64
 10  winddir           122567 non-null  float64
 11  sealevelpressure  122319 non-null  float64
 12  cloudcover        122556 non-null  float64
 13  solarradiation    122514 non-null  float64
 14  solarenergy       122480 non-null  float64
 15  uvindex           122486 non-null  float64
dtypes: float64(14), object(2)
memory usage: 15.0+ MB
None
```

```
# Print dimensions of the dataset
print("Dataset dimensions:", f_brighton_df.shape)
```

```
Dataset dimensions: (122844, 16)
```

```
# Print first 2 rows of the dataset
print("First few rows of the dataset:")
print(f_brighton_df.head(2))
```

```
First few rows of the dataset:
              datetime  temp  dew  humidity  precip  precipprob
preciptype  \
0  2010-03-22T00:00:00   4.4  4.3     99.49     0.0         0.0
NaN
1  2010-03-22T01:00:00   4.8  4.8     99.82     0.0         0.0
NaN

   snow  snowdepth  windspeed  winddir  sealevelpressure
cloudcover  \
0   0.0        0.0        5.7    185.0            1021.1        40.0

1   0.0        0.0        5.9    162.0            1020.9        91.9


   solarradiation  solarenergy  uvindex
0             0.0          0.0      0.0
1             0.0          0.0      0.0
```

```
# Summary
print("Summary:")
print(f_brighton_df.describe())
```

```
Summary:
                temp            dew       humidity         precip  \
count  122590.000000  122568.000000  122575.000000  122565.000000
mean       11.059431       8.047911      82.880536       0.081954
std         5.654195       5.136014      12.511270       0.660720
min        -9.100000     -11.600000      24.340000       0.000000
```

```
25%        7.200000      4.500000     75.560000      0.000000
50%       11.000000      8.400000     85.700000      0.000000
75%       15.300000     12.000000     92.790000      0.000000
max       33.300000     20.200000    100.000000     32.385000

          precipprob          snow     snowdepth      windspeed  \
count  122558.000000  89810.000000  89279.000000  122583.000000
mean        8.883141      0.000408      0.028941      15.938294
std        28.450141      0.034549      0.441740       8.903724
min         0.000000      0.000000      0.000000       0.000000
25%         0.000000      0.000000      0.000000       9.400000
50%         0.000000      0.000000      0.000000      14.400000
75%         0.000000      0.000000      0.000000      21.300000
max       100.000000      7.870000     96.000000      72.200000

             winddir  sealevelpressure     cloudcover  solarradiation
\
count  122567.000000     122319.000000  122556.000000   122514.000000

mean      196.561486       1015.525463      60.841873      138.575974

std       106.273116         10.519485      31.879025      220.733866

min         0.700000        955.000000       0.000000        0.000000

25%       113.000000       1009.600000      36.000000        0.000000

50%       223.000000       1016.400000      68.400000        9.000000

75%       267.000000       1022.500000      89.800000      201.000000

max       360.000000       1049.300000     100.000000     1150.000000


         solarenergy        uvindex
count  122480.000000  122486.000000
mean        0.498450       1.368646
std         0.795407       2.225490
min         0.000000       0.000000
25%         0.000000       0.000000
50%         0.000000       0.000000
75%         0.700000       2.000000
max         4.100000      10.000000
```

```python
# Counting for missing values
print("Missing values:")
print(f_brighton_df.isnull().sum())
```

```
Missing values:
datetime                0
temp                  254
```

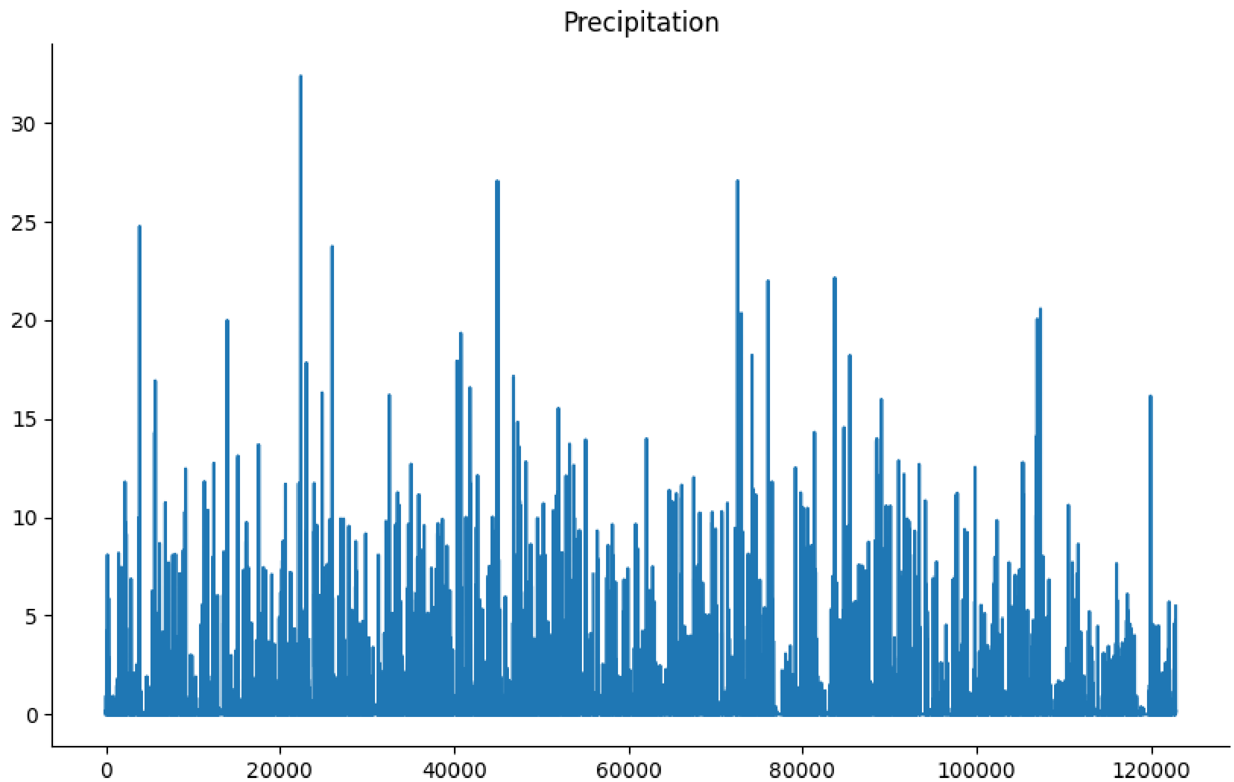
```
dew                        276
humidity                   269
precip                     279
precipprob                 286
preciptype              111567
snow                     33034
snowdepth                33565
windspeed                  261
winddir                    277
sealevelpressure           525
cloudcover                 288
solarradiation             330
solarenergy                364
uvindex                    358
dtype: int64
```

```
# Info about  categorical variables
print("Categories & frequencies for categorical variables:")
for col in f_brighton_df.select_dtypes(include='object').columns:
    print(f_brighton_df[col].value_counts())
```

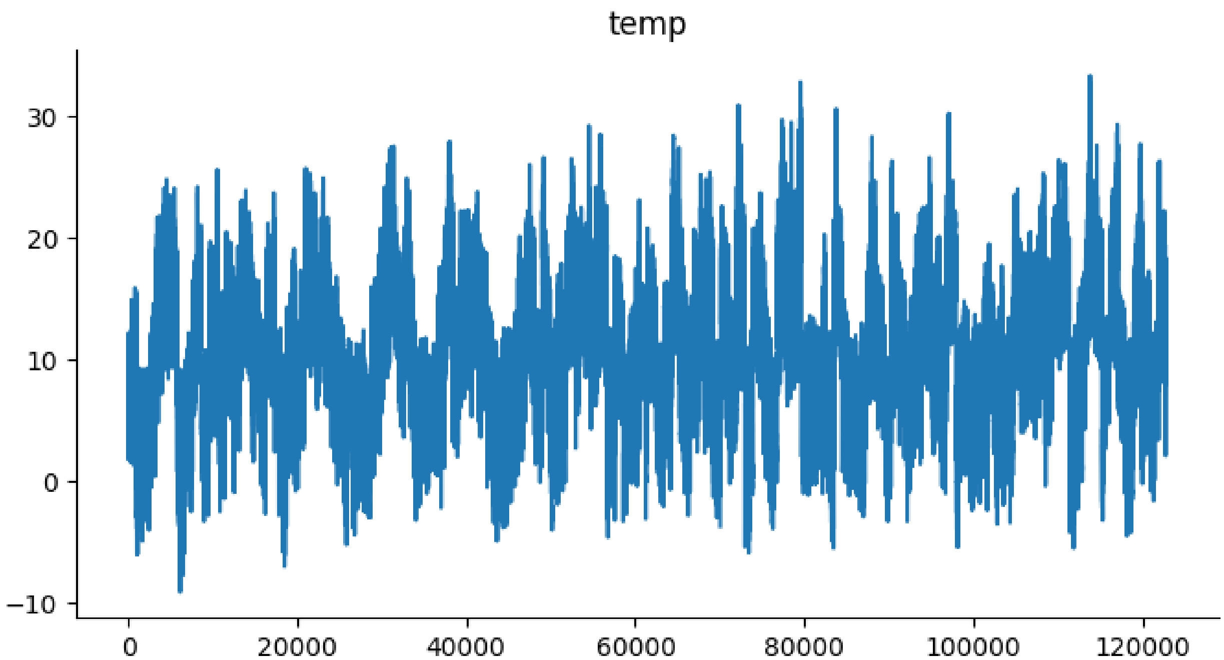
```
Categories & frequencies for categorical variables:
2020-10-25T01:00:00    2
2014-10-26T01:00:00    2
2017-10-29T01:00:00    2
2022-10-30T01:00:00    2
2023-10-29T01:00:00    2
                      ..
2014-09-03T02:00:00    1
2014-09-03T01:00:00    1
2014-09-03T00:00:00    1
2014-09-02T23:00:00    1
2023-10-19T23:00:00    1
Name: datetime, Length: 122830, dtype: int64
rain         10903
rain,snow      344
snow            30
Name: preciptype, dtype: int64
```

## Graphs for EDA

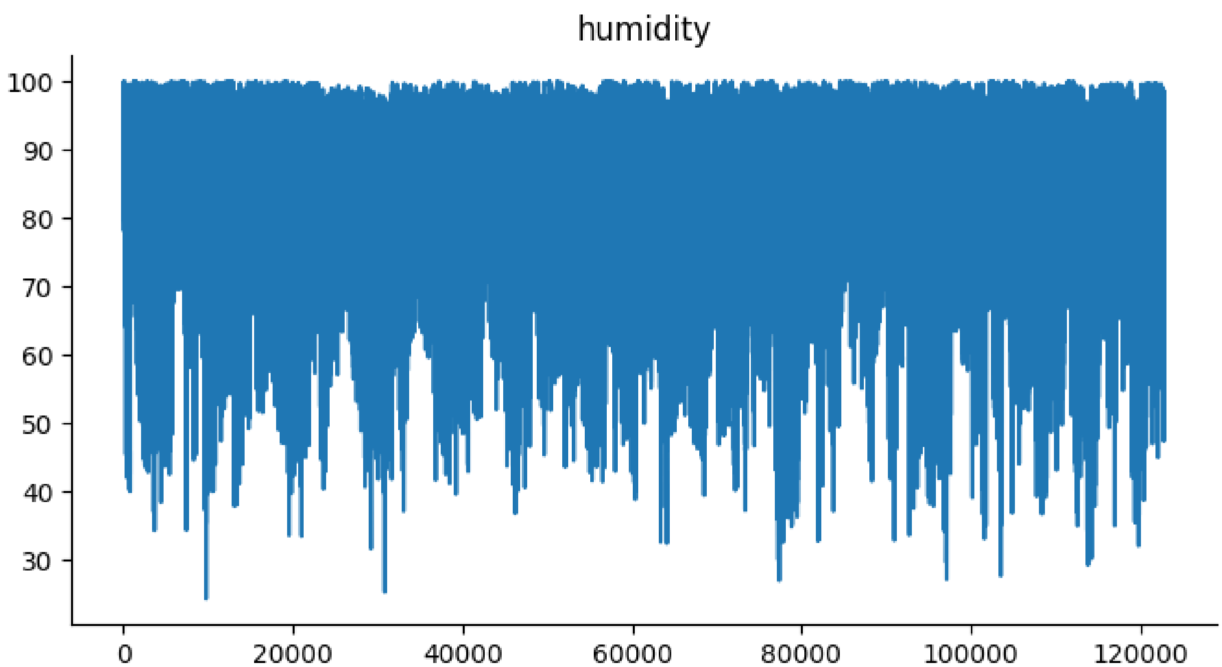
```
# @title Precipitation

from matplotlib import pyplot as plt
f_brighton_df['precip'].plot(kind='line', figsize=(10, 6),
title='Precipitation')
plt.gca().spines[['top', 'right']].set_visible(False)
```



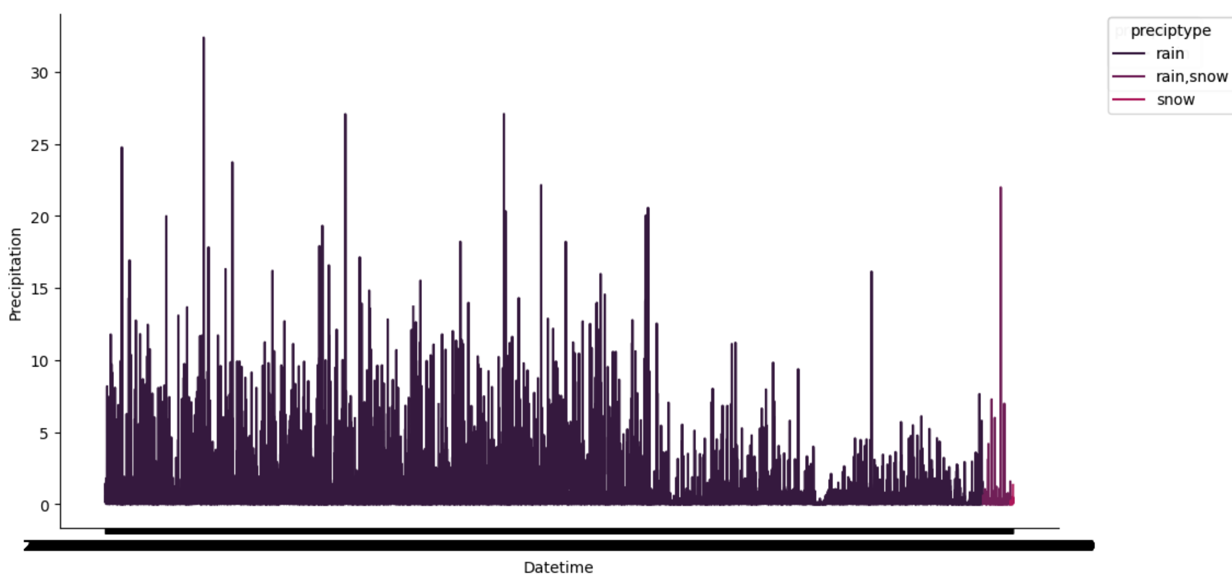

```
# @title Temperature

from matplotlib import pyplot as plt
f_brighton_df['temp'].plot(kind='line', figsize=(8, 4), title='temp')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```python
# @title Humidity

from matplotlib import pyplot as plt
f_brighton_df['humidity'].plot(kind='line', figsize=(8, 4), 
title='humidity')
plt.gca().spines[['top', 'right']].set_visible(False)
```

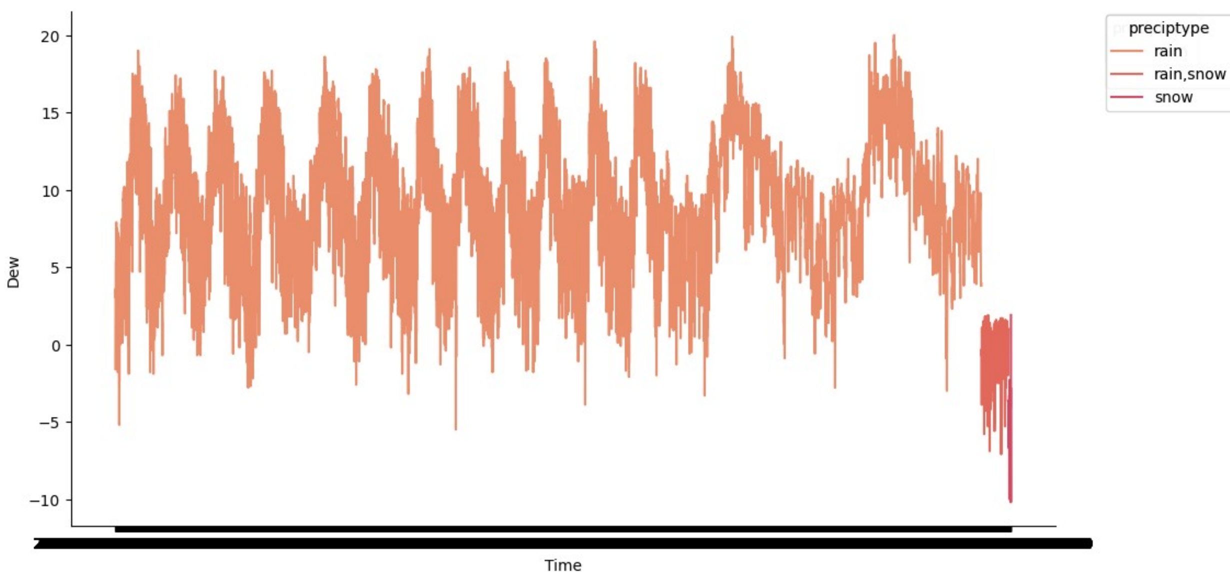


```
# @title DateTime VS Precipitation

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('rocket'))
  xs = series['datetime']
  ys = series['precip']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_brighton_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Datetime')
_ = plt.ylabel('Precipitation')
```



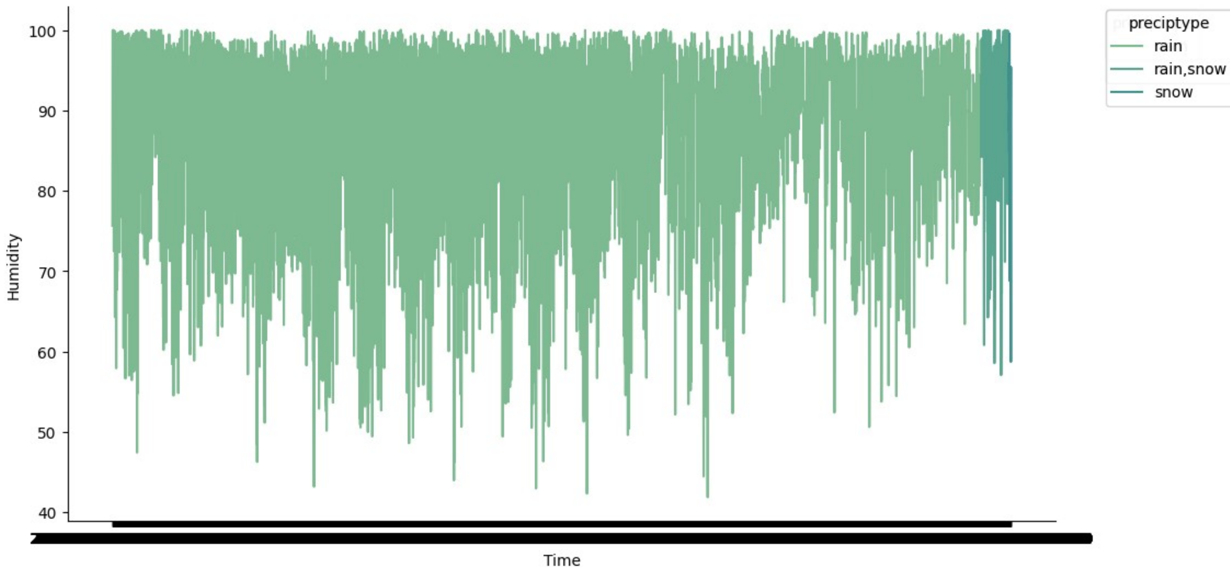

```
# @title Time VS Dew

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
```

```
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('flare'))
  xs = series['datetime']
  ys = series['dew']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_brighton_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('Dew')
```



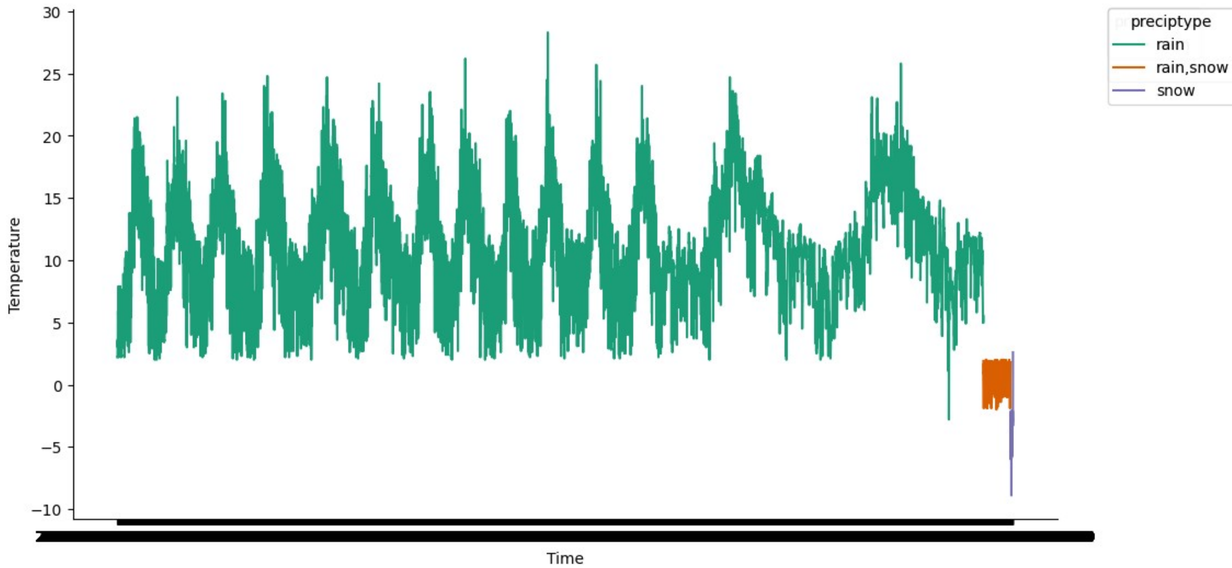

```
# @title Time VS Humidity

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('crest'))
  xs = series['datetime']
  ys = series['humidity']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
```

```
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_brighton_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('Humidity')
```

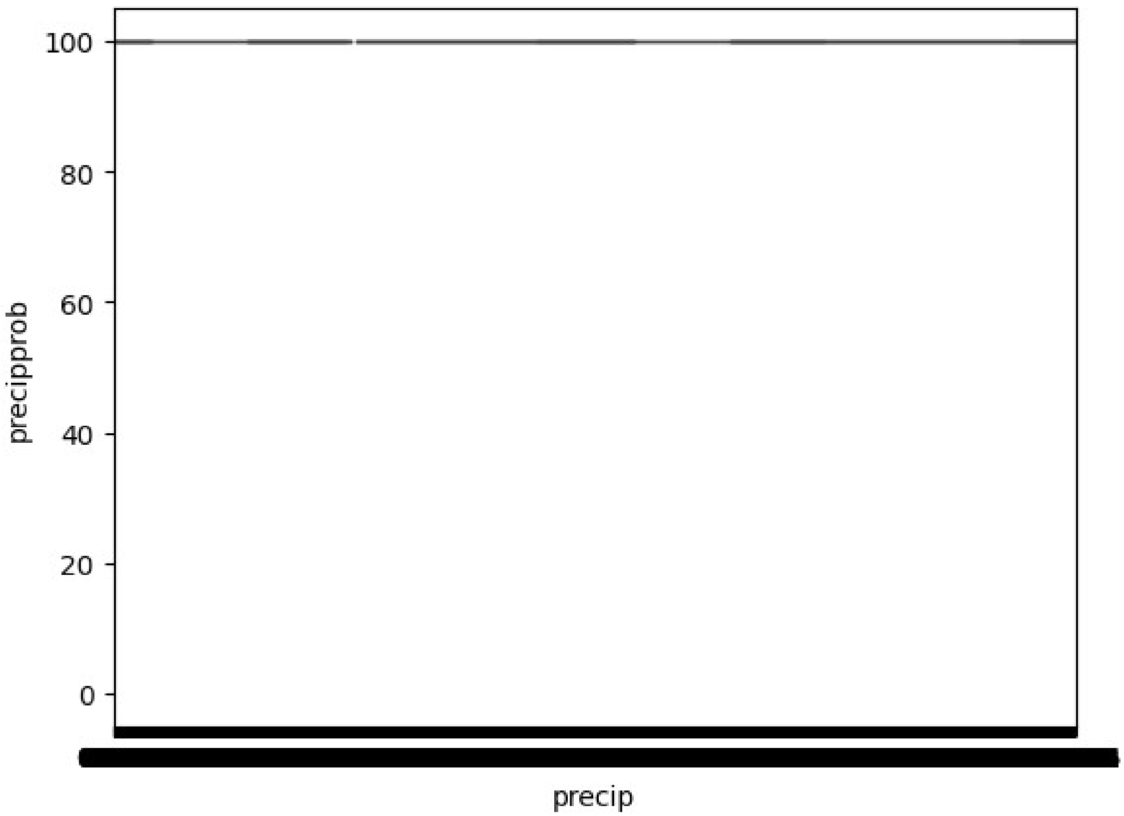



```
# @title Time vs Temperature

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['datetime']
  ys = series['temp']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_brighton_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
```

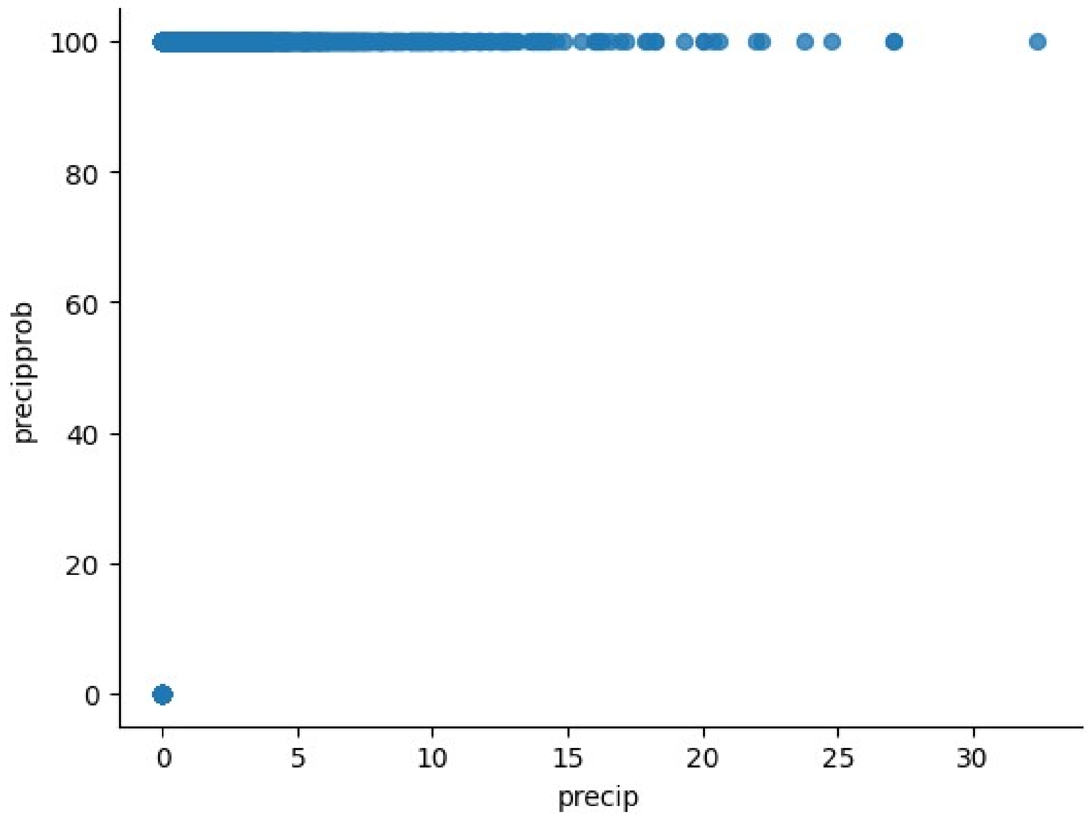
```
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('Temperature')
```



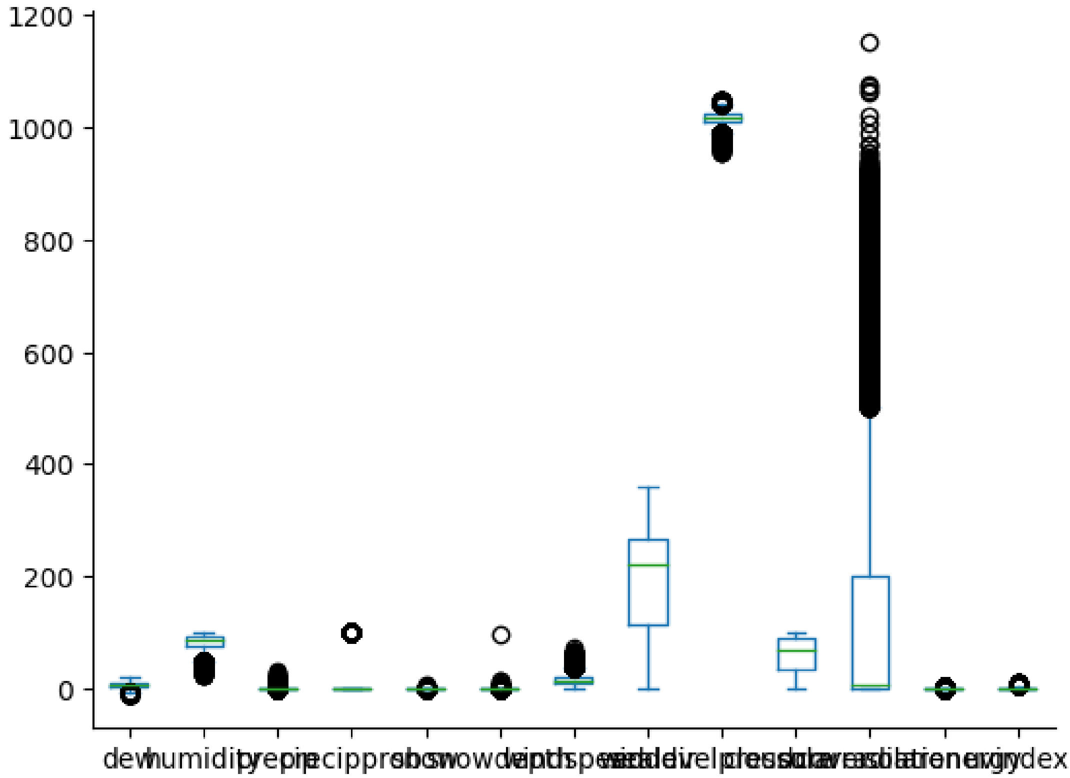

```
# @title Precipitaion VS Precipitation Probability

from matplotlib import pyplot as plt
sns.boxplot(x=f_brighton_df['precip'], y=f_brighton_df['precipprob'])

<Axes: xlabel='precip', ylabel='precipprob'>
```



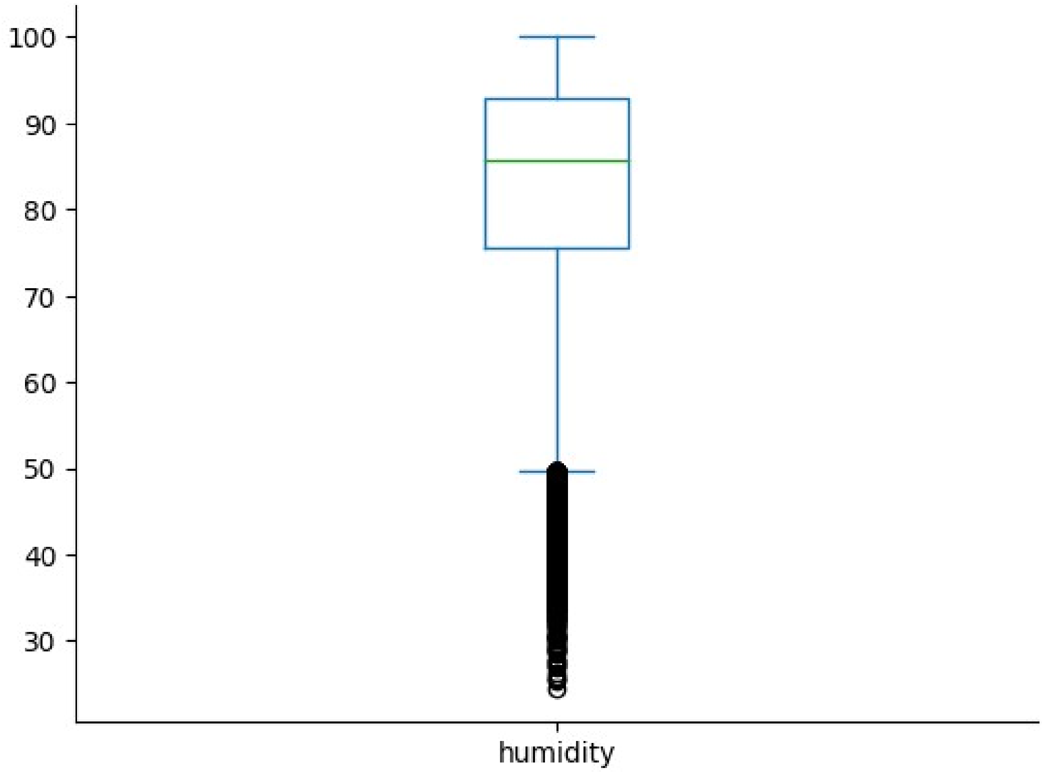

```
# @title Precipitaion VS Precipitation Probability
from matplotlib import pyplot as plt
f_brighton_df.plot(kind='scatter', x='precip', y='precipprob', s=32,
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



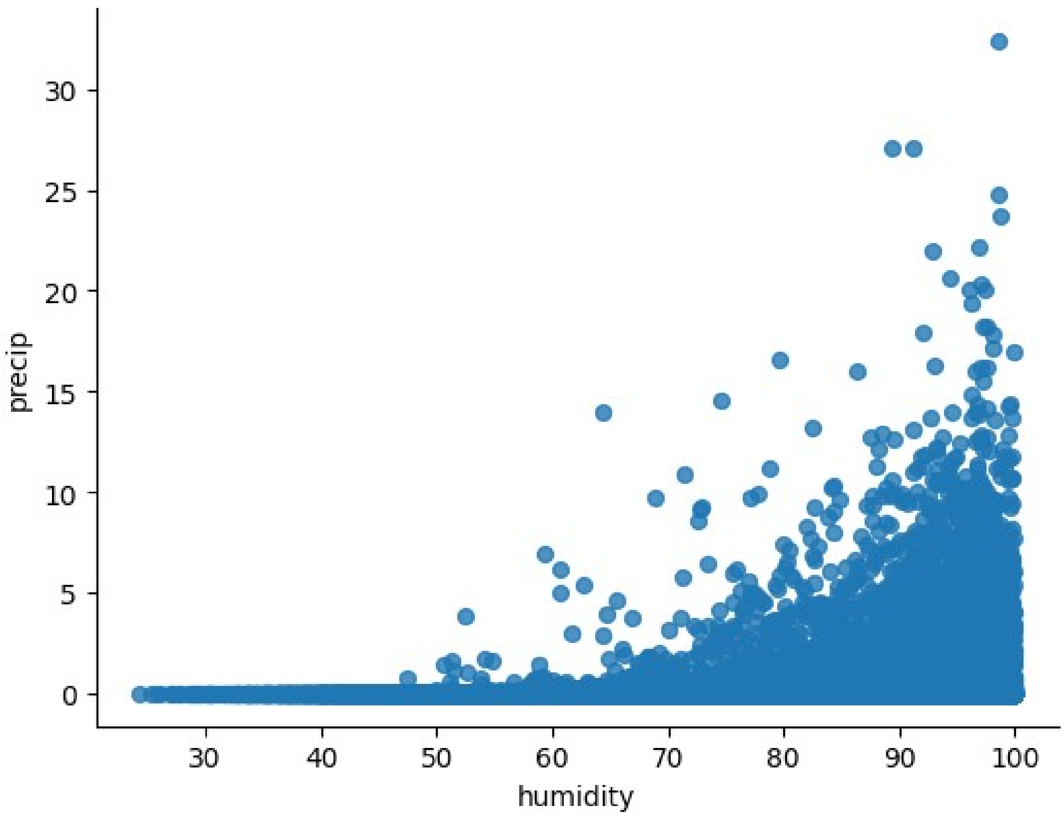

```
# @title Temperature vs Dew

from matplotlib import pyplot as plt
f_brighton_df.plot(kind='box', x='temp')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# @title Dew VS Humidity

from matplotlib import pyplot as plt
f_brighton_df.plot(kind='box', x='dew', y='humidity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

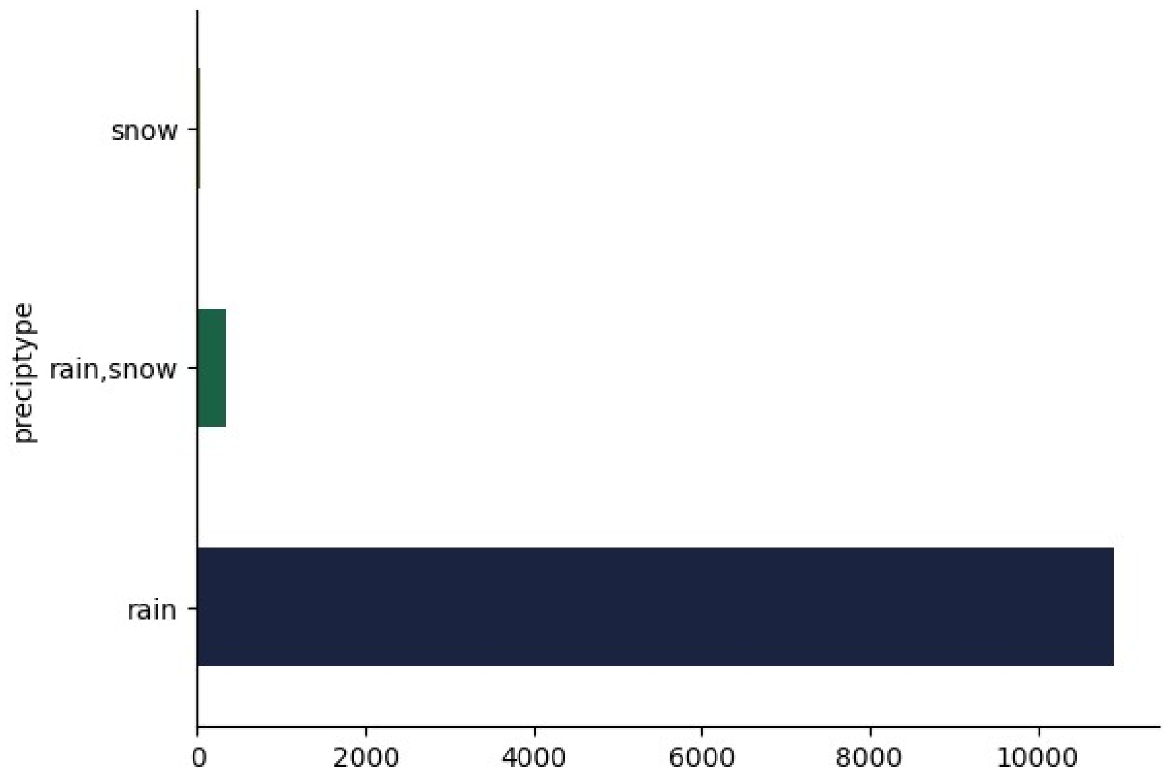



```
# @title Humidity VS Precipitation

from matplotlib import pyplot as plt
f_brighton_df.plot(kind='scatter', x='humidity', y='precip', s=32,
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

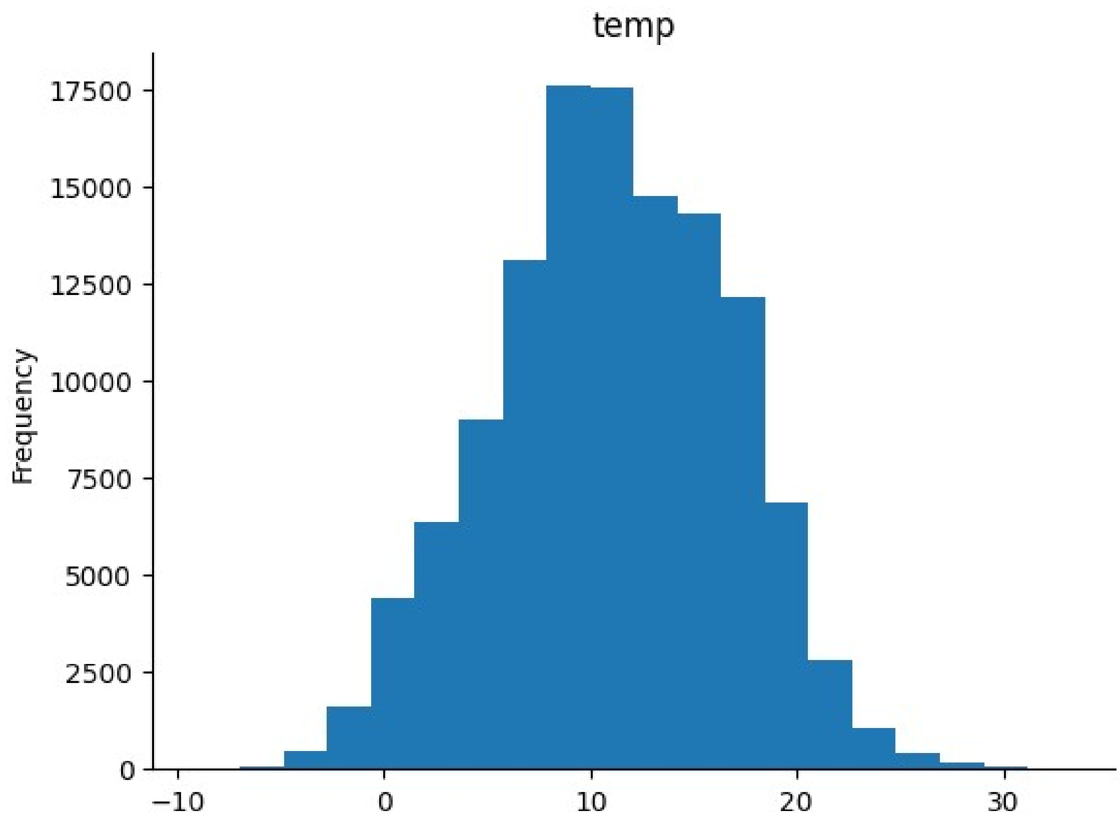



###Exploring Important Columns

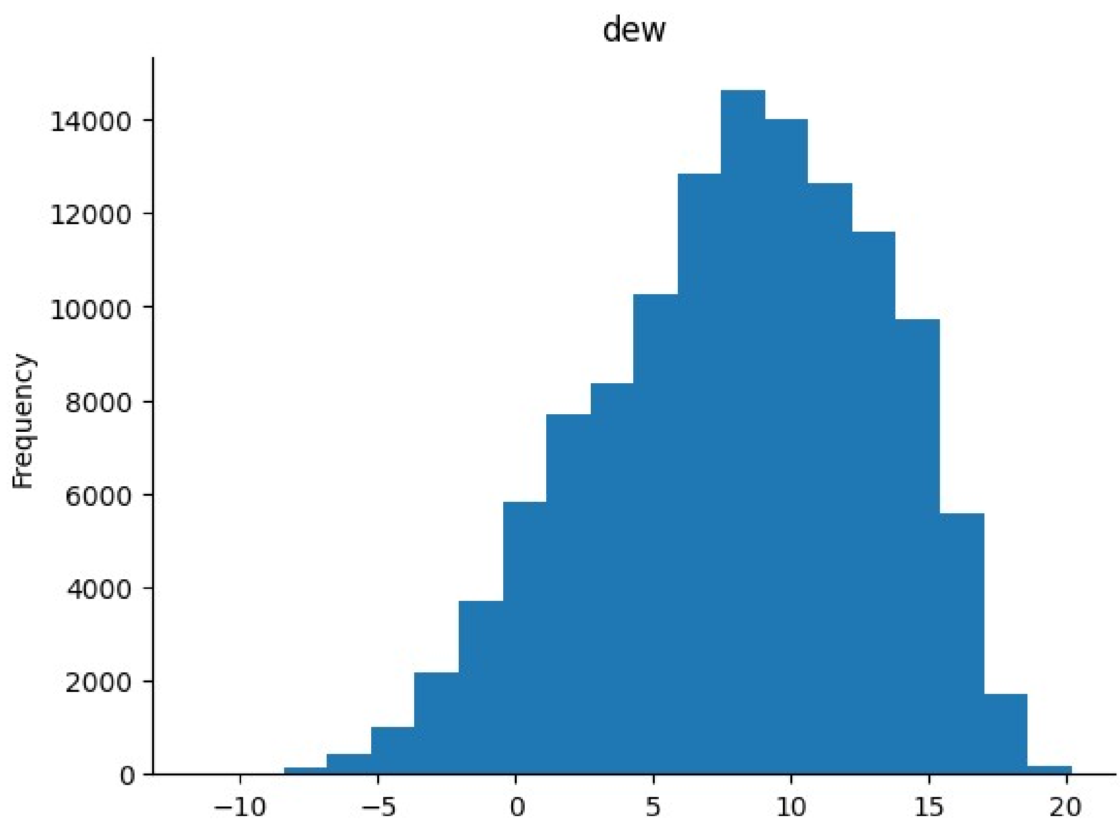
```
# @title Precipitation Type

from matplotlib import pyplot as plt
import seaborn as sns
f_brighton_df.groupby('preciptype').size().plot(kind='barh',
color=sns.palettes.mpl_palette('cubehelix'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



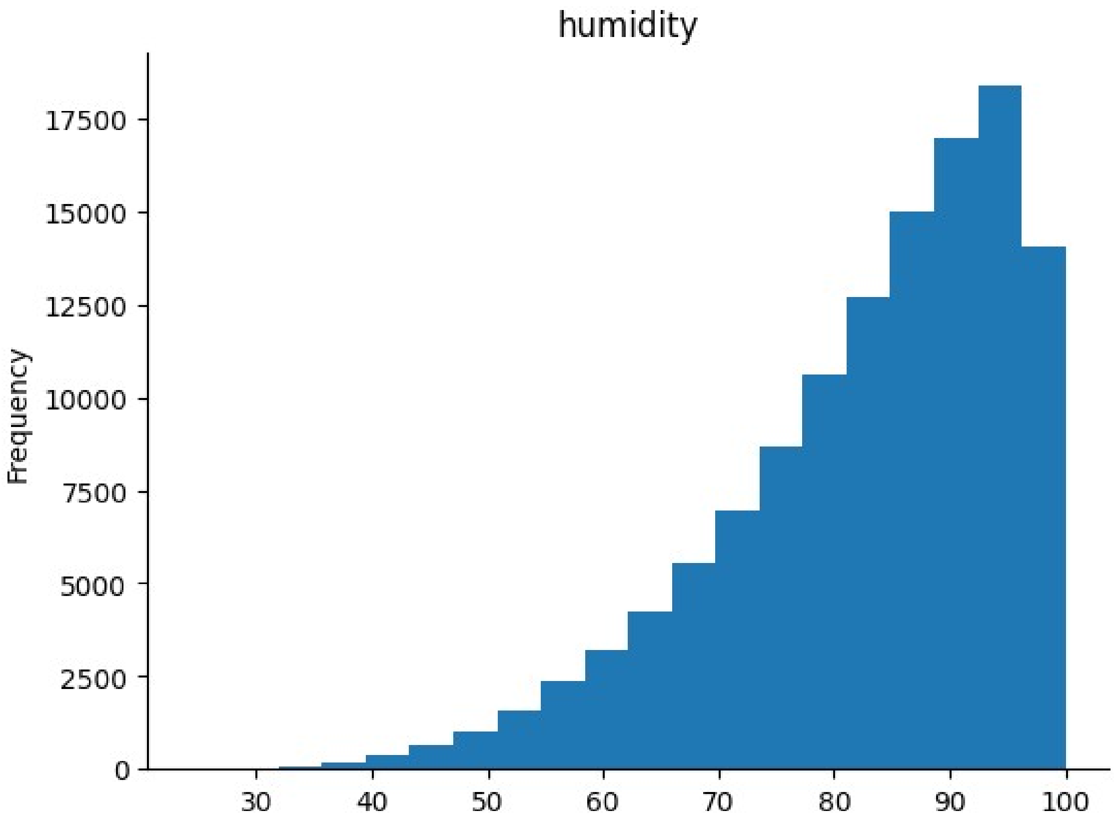

```
# @title Temperature

from matplotlib import pyplot as plt
f_brighton_df['temp'].plot(kind='hist', bins=20, title='temp')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



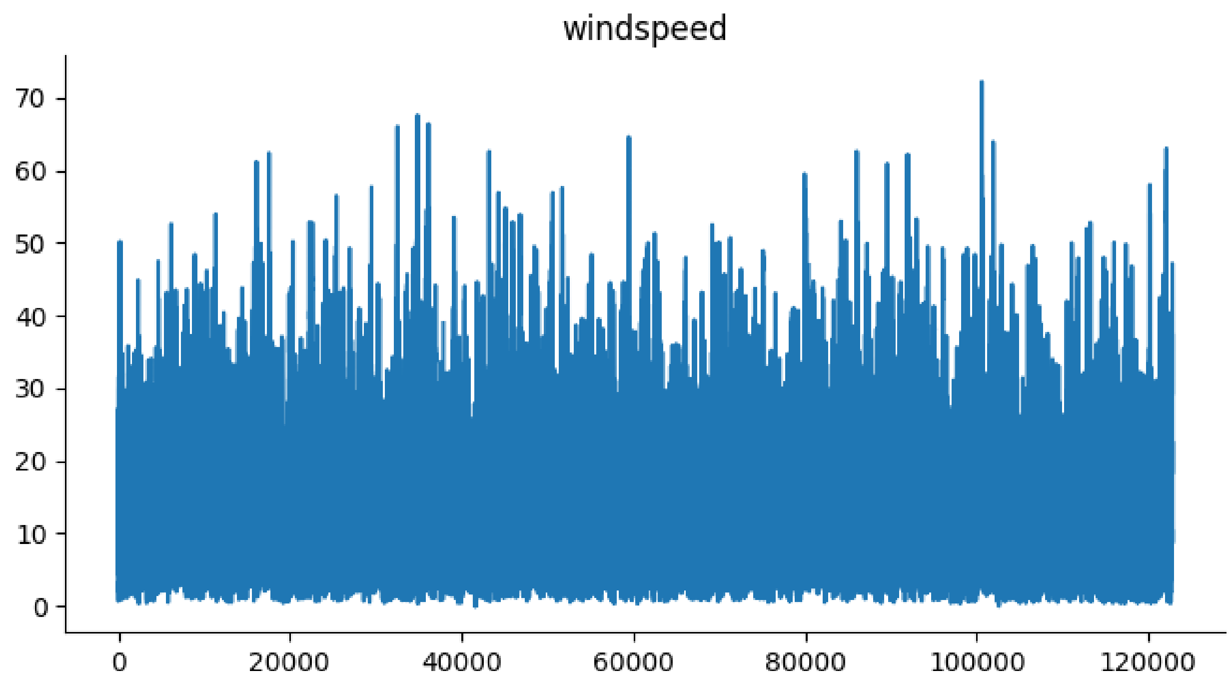

```
# @title Dew

from matplotlib import pyplot as plt
f_brighton_df['dew'].plot(kind='hist', bins=20, title='dew')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# @title Humidity

from matplotlib import pyplot as plt
f_brighton_df['humidity'].plot(kind='hist', bins=20, title='humidity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# @title Windspeed
from matplotlib import pyplot as plt
f_brighton_df['windspeed'].plot(kind='line', figsize=(8, 4),
title='windspeed')
plt.gca().spines[['top', 'right']].set_visible(False)
```




##Time Series Analysis

```
months_of_the_year = ['January', 'February', 'March', 'April', 'May',
'June', 'July', 'August', 'September', 'October', 'November',
'December']

f_brighton_df.columns

Index(['datetime', 'temp', 'dew', 'humidity', 'precip', 'precipprob',
       'preciptype', 'snow', 'snowdepth', 'windspeed', 'winddir',
       'sealevelpressure', 'cloudcover', 'solarradiation',
'solarenergy',
       'uvindex'],
      dtype='object')

# Date will be our index. Let's convert it to a datetime type
f_brighton_df['datetime'] = pd.to_datetime(f_brighton_df['datetime'],
dayfirst=True)
f_brighton_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 122844 entries, 0 to 122843
Data columns (total 16 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   datetime          122844 non-null  datetime64[ns]
 1   temp              122590 non-null  float64
 2   dew               122568 non-null  float64
 3   humidity          122575 non-null  float64
```

```
 4   precip            122565 non-null  float64
 5   precipprob        122558 non-null  float64
 6   preciptype        11277 non-null   object
 7   snow              89810 non-null   float64
 8   snowdepth         89279 non-null   float64
 9   windspeed         122583 non-null  float64
 10  winddir           122567 non-null  float64
 11  sealevelpressure  122319 non-null  float64
 12  cloudcover        122556 non-null  float64
 13  solarradiation    122514 non-null  float64
 14  solarenergy       122480 non-null  float64
 15  uvindex           122486 non-null  float64
dtypes: datetime64[ns](1), float64(14), object(1)
memory usage: 15.0+ MB
```

```
print(f_brighton_df['datetime'].min(),
f_brighton_df['datetime'].max())
```

```
2010-01-01 00:00:00 2024-01-06 11:00:00
```

```
# Let's say we want to create extra columns: month, year, and week of
the year
df = f_brighton_df.copy()
df['month'] = df['datetime'].dt.month
df['year'] = df['datetime'].dt.year
df['week_of_year'] = df['datetime'].dt.isocalendar().week
df
```

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

```
# Let's make the date column the index of the dataframe for easier
slicing
df.set_index('datetime', inplace=True) # note we can only run this
once, as it will delete the 'date' column.
df.head()
```

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

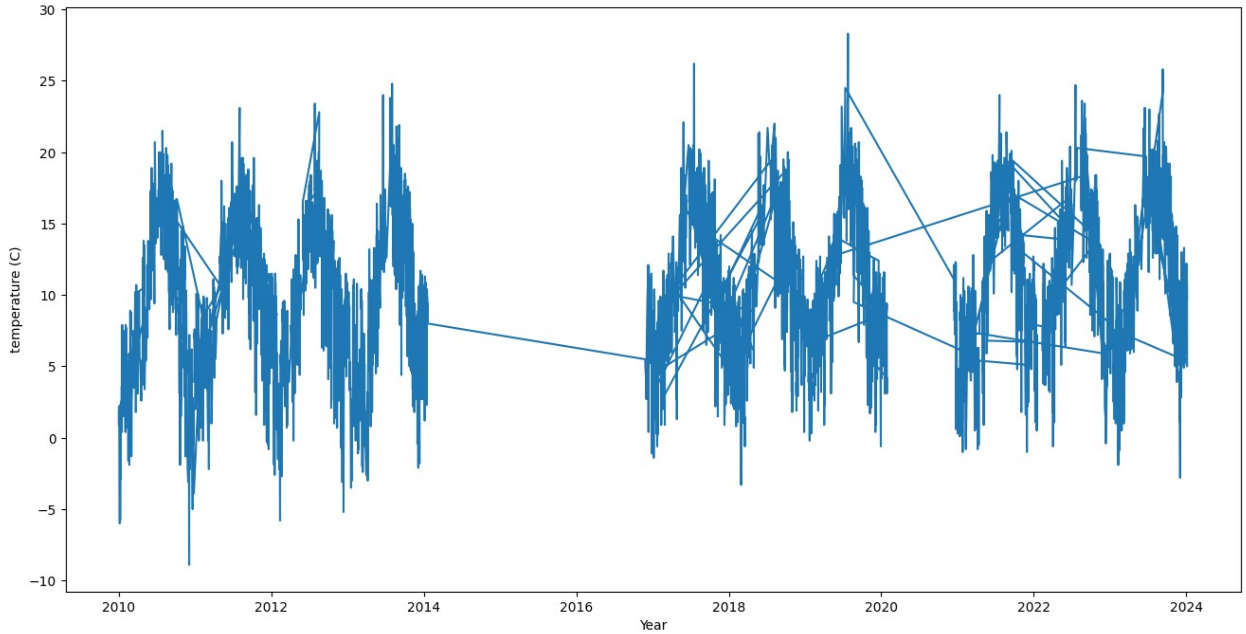
```
print(df.isna().any())
df=df.dropna()
print(df.isna().any())
```

```
temp                    True
dew                     True
humidity                True
precip                  True
precipprob              True
preciptype              True
snow                    True
snowdepth               True
windspeed               True
```

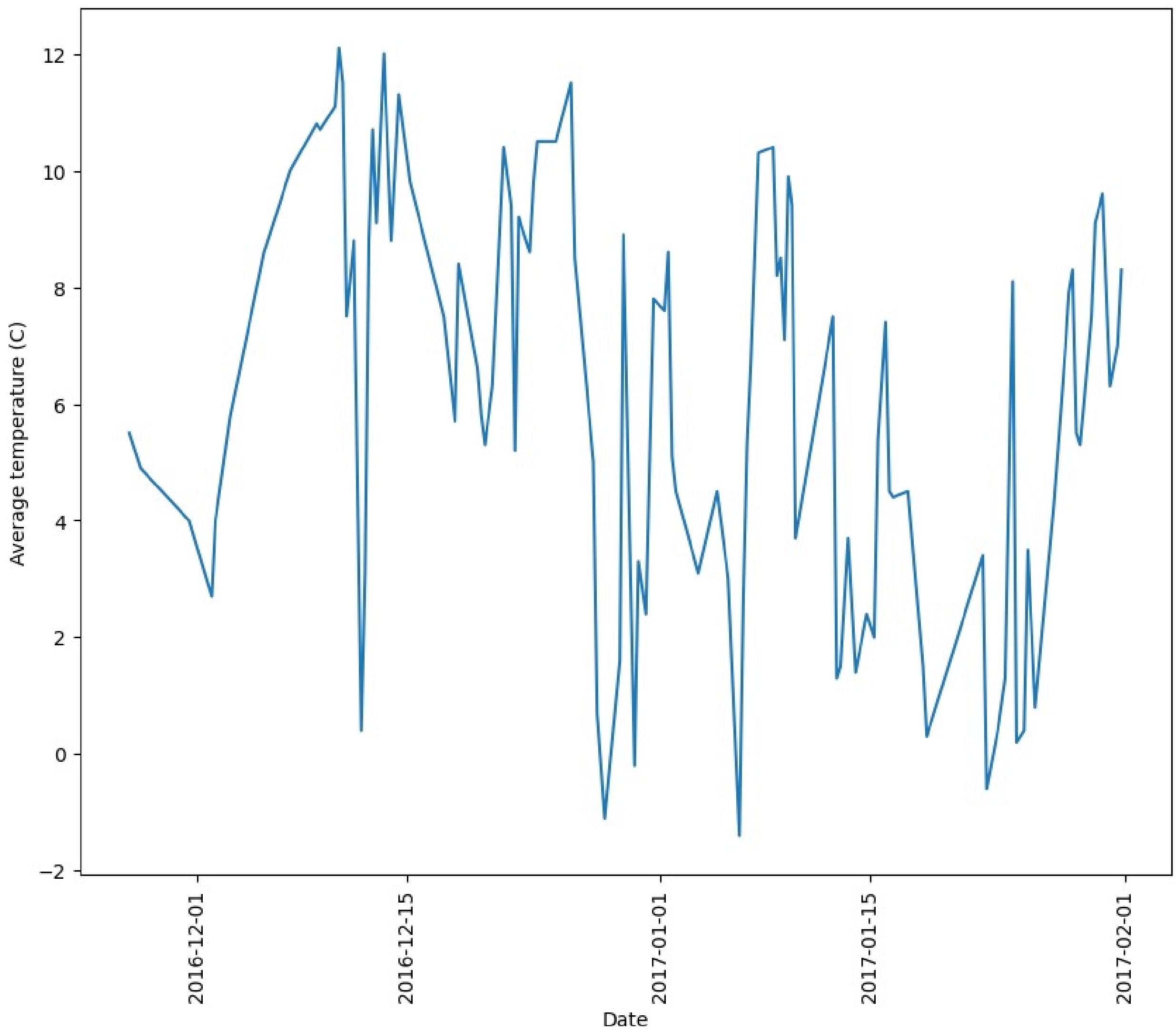
```
winddir                True
sealevelpressure       True
cloudcover             True
solarradiation         True
solarenergy            True
uvindex                True
month                 False
year                  False
week_of_year          False
dtype: bool
temp                  False
dew                   False
humidity              False
precip                False
precipprob            False
preciptype            False
snow                  False
snowdepth             False
windspeed             False
winddir               False
sealevelpressure      False
cloudcover            False
solarradiation        False
solarenergy           False
uvindex               False
month                 False
year                  False
week_of_year          False
dtype: bool
```

```
# Let's plot the data. For now we're only going to work on temp_avg,
so let's have a look
plt.figure(figsize=(16,8))
plt.plot(df.index, df['temp'])
plt.xlabel('Year')
plt.ylabel('temperature (C)')
```

```
Text(0, 0.5, 'temperature (C)')
```




```
# Let's zoom in to 2014-2017
df_chunk = df.loc['2014-12':'2017-01']  # since the date is an index, we can use it to filter our data

plt.figure(figsize=(10, 8))
plt.plot(df_chunk.index, df_chunk['temp'])
plt.xticks(rotation=90)
plt.xlabel('Date')
_=plt.ylabel('Average temperature (C)')
```




```
print(df_chunk.reindex(pd.date_range('2014-12', '2017-
01')).isnull().all(1).sum()) # 33 days missing
df_chunk.reindex(pd.date_range('2014-12', '2017-01')).isnull().all(1)
```

```
752
```

```
2014-12-01     True
2014-12-02     True
2014-12-03     True
2014-12-04     True
2014-12-05     True
              ...
2016-12-28     True
2016-12-29     True
2016-12-30     True
2016-12-31    False
```

```
2017-01-01     True
Freq: D, Length: 763, dtype: bool
```

```
df[df.index.duplicated(keep=False)].head(20)
```


{"summary":"{\n  \"name\": \"df[df\",\n  \"rows\": 2,\n  \"fields\": [\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.14142135623731025,\n        \"min\": 11.2,\n        \"max\": 11.4,\n        \"samples\": [\n          11.2,\n          11.4\n        ],\n        \"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"dew\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.2828427124746193,\n        \"min\": 10.1,\n        \"max\": 10.5,\n        \"samples\": [\n          10.1,\n          10.5\n        ],\n        \"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"humidity\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.5374011537017798,\n        \"min\": 93.11,\n        \"max\": 93.87,\n        \"samples\": [\n          93.11,\n          93.87\n        ],\n        \"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"precip\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.233194226389339,\n        \"min\": 0.678,\n        \"max\": 2.422,\n        \"samples\": [\n          2.422,\n          0.678\n        ],\n        \"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"precipprob\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 100.0,\n        \"max\": 100.0,\n        \"samples\": [\n          100.0\n        ],\n        \"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"preciptype\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"samples\": [\n          \"rain\"\n        ],\n        \"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"snow\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n        \"samples\": [\n          0.0\n        ],\n        \"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"snowdepth\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n        \"samples\": [\n          0.0\n        ],\n        \"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"windspeed\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 7.424621202458749,\n        \"min\": 11.3,\n        \"max\": 21.8,\n        \"samples\": [\n          21.8\

```
n        ],\n        \"num_unique_values\": 2,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"winddir\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3.5355339059327378,\n        \"min\": 245.0,\n        \"max\": 250.0,\
n        \"samples\": [\n          245.0\n        ],\n
\"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sealevelpressure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.6363961030678768,\n        \"min\":
983.7,\n        \"max\": 984.6,\n        \"samples\": [\n
984.6\n        ],\n        \"num_unique_values\": 2,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cloudcover\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6.505382386916239,\n        \"min\": 86.6,\n        \"max\": 95.8,\n
\"samples\": [\n          86.6\n        ],\n
\"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"solarradiation\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"solarenergy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"uvindex\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n
\"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"month\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 10,\n        \"max\": 10,\n
\"samples\": [\n          10\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"year\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 2023,\n        \"max\": 2023,\n
\"samples\": [\n          2023\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"week_of_year\",\n      \"properties\": {\n        \"dtype\":
\"UInt32\",\n        \"samples\": [\n          \"43\"\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
# Let's keep the first one only - in practice this would require more careful analysis!
df = df[~df.index.duplicated(keep='first')]
len(df)
```

```
8535
```

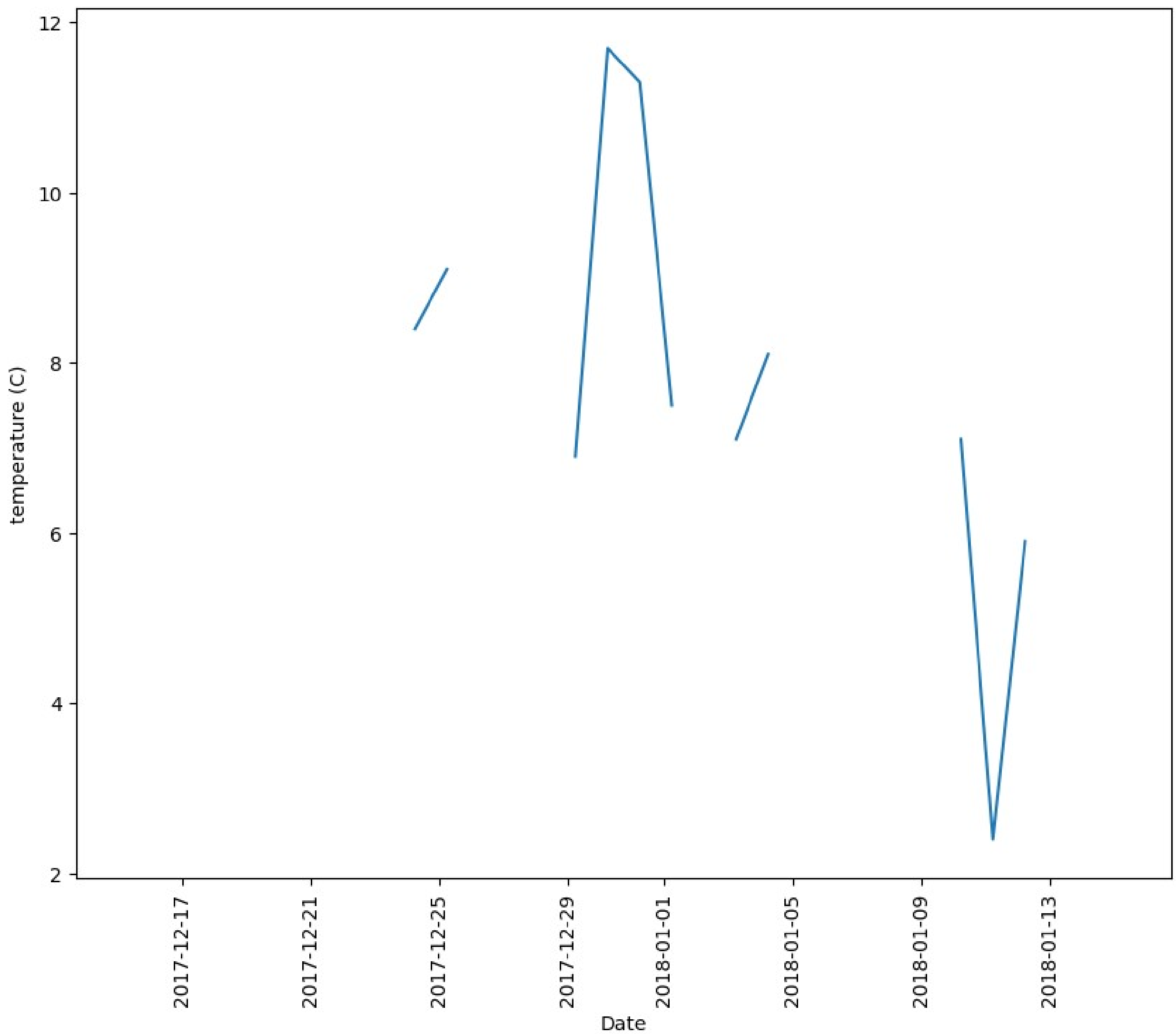
```
# Now we can reindex -- this is where the original error about duplicates was
df = df.reindex(pd.date_range(df.index[0], df.index[-1]))
print(len(df))
```

```
4960
```

```
# Now we should have missing values
print(df.isna().sum())
```

```
temp                4191
dew                 4191
humidity            4191
precip              4191
precipprob          4191
preciptype          4191
snow                4191
snowdepth           4191
windspeed           4191
winddir             4191
sealevelpressure    4191
cloudcover          4191
solarradiation      4191
solarenergy         4191
uvindex             4191
month               4191
year                4191
week_of_year        4191
dtype: int64
```

```
df_chunk = df.loc['2017-12-15':'2018-01-15']  # since the date is an index, we can use it to filter our data

plt.figure(figsize=(10, 8))
plt.plot(df_chunk.index, df_chunk['temp'])
plt.xticks(rotation=90)
plt.xlabel('Date')
_=plt.ylabel('temperature (C)')
# The missing values are clearly visible now!
```

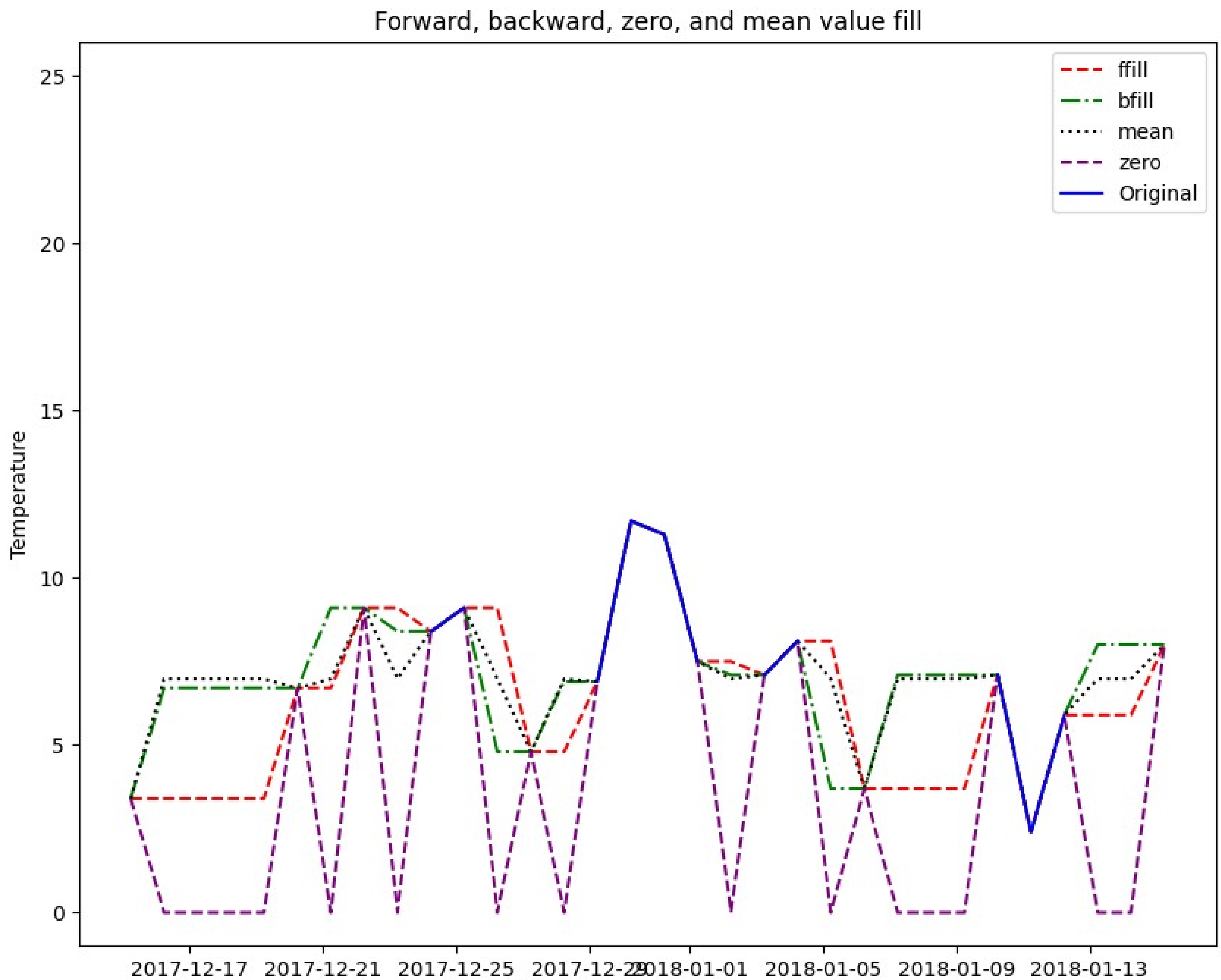



```
df2 = df_chunk.copy()
df2 = df2.loc[:, 'temp'].to_frame()
df2
```

{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 32,\n  \"fields\": [\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2.5582622778661577,\n        \"min\": 2.4,\n        \"max\": 11.7,\n        \"samples\": [\n          7.1,\n          3.7,\n          3.4\n        ],\n        \"num_unique_values\": 15,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"df2"}

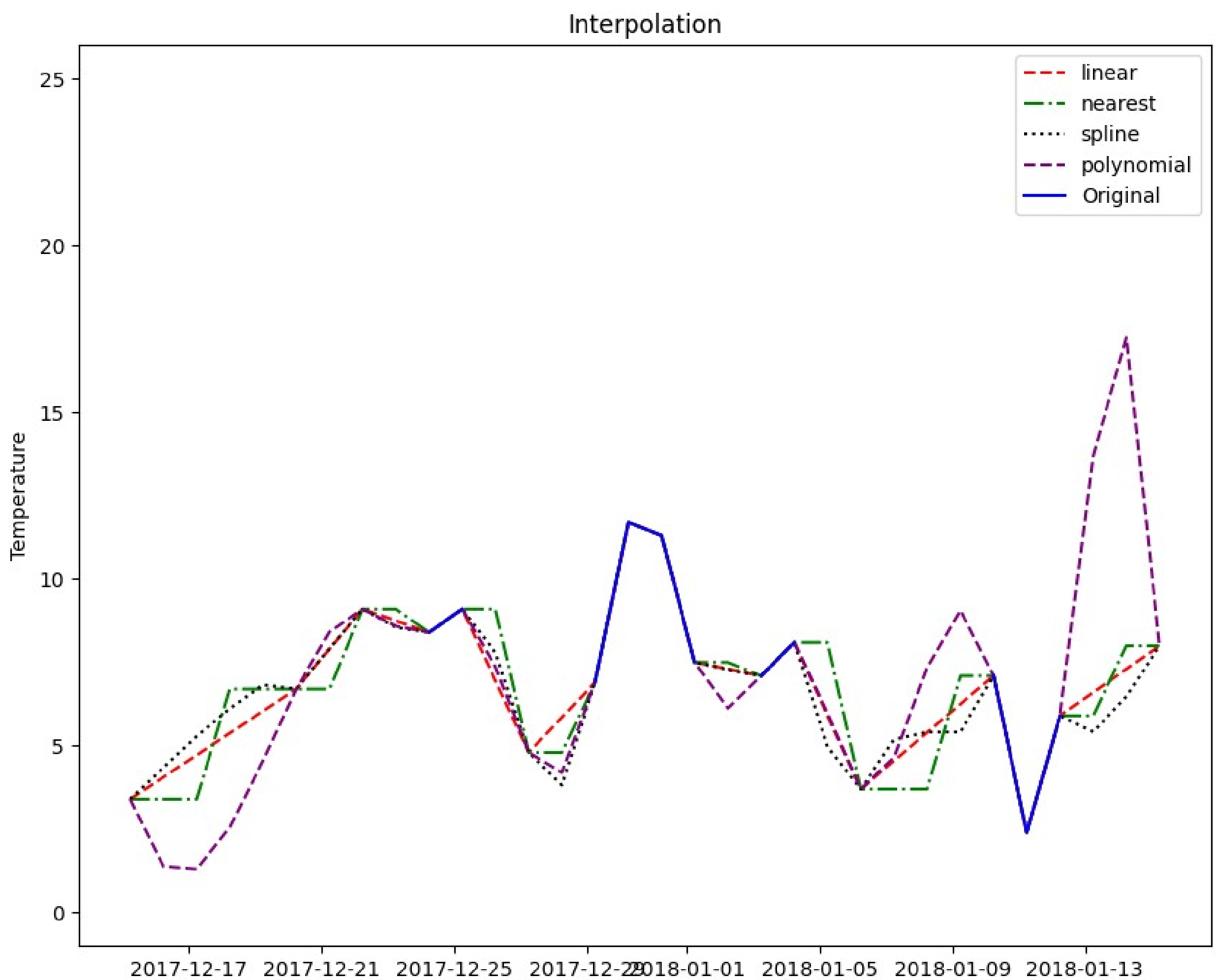
```
#Forward Fill
df2['ffill'] = df2['temp'].ffill()
# Backward Fill
df2['bfill'] = df2['temp'].bfill()
```

```python
# Mean Value Fill
df2['meanfill'] = df2['temp'].fillna(df['temp'].mean())  # Note that
we're using the mean of df, not of df2
# Fill with 0s
df2['zerofill'] = df2['temp'].fillna(0)

# Plot
fig, ax = plt.subplots(figsize=(10,8))

plt.plot(df2.index, df2['ffill'], label='ffill', linestyle='--',
color='red')
plt.plot(df2.index, df2['bfill'], label='bfill', linestyle='-.',
color='green')
plt.plot(df2.index, df2['meanfill'], label='mean', linestyle=':',
color='black')
plt.plot(df2.index, df2['zerofill'], linestyle='--', color='purple',
label='zero')
plt.plot(df2.index, df2['temp'], color='blue', label='Original')
plt.legend()
plt.ylabel('Temperature')
plt.ylim(-1, 26)
_=plt.title('Forward, backward, zero, and mean value fill')
```

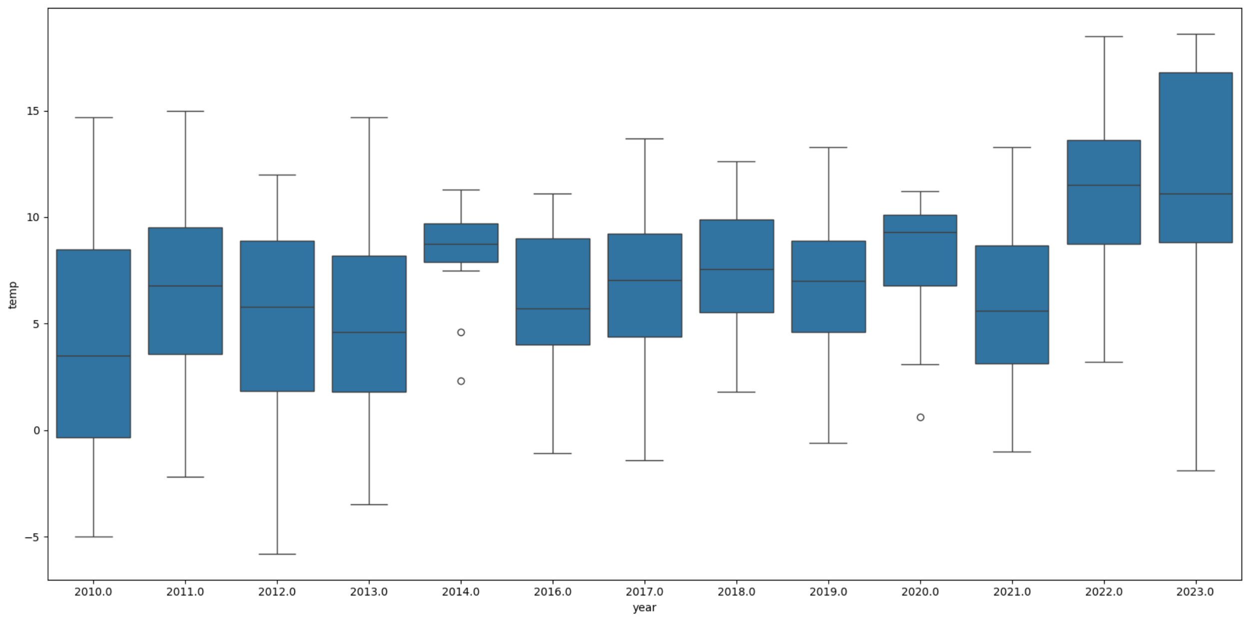



```
# Try different ways to fill the data - more advanced: interpolation

df2['linear_interp'] = df2['temp'].interpolate(method='linear')
df2['nearest_interp'] = df2['temp'].interpolate(method='nearest')
df2['spline_interp'] = df2['temp'].interpolate(method='spline',
order=2)
df2['polynomial_interp'] =
df2['temp'].interpolate(method="polynomial", order=3)

# Plot
fig, ax = plt.subplots(figsize=(10,8))

plt.plot(df2.index, df2['linear_interp'], linestyle='--', color='red',
label='linear')
plt.plot(df2.index, df2['nearest_interp'], linestyle='-.',
color='green', label='nearest')
plt.plot(df2.index, df2['spline_interp'], linestyle=':',
color='black', label='spline')
plt.plot(df2.index, df2['polynomial_interp'], linestyle='--',
```

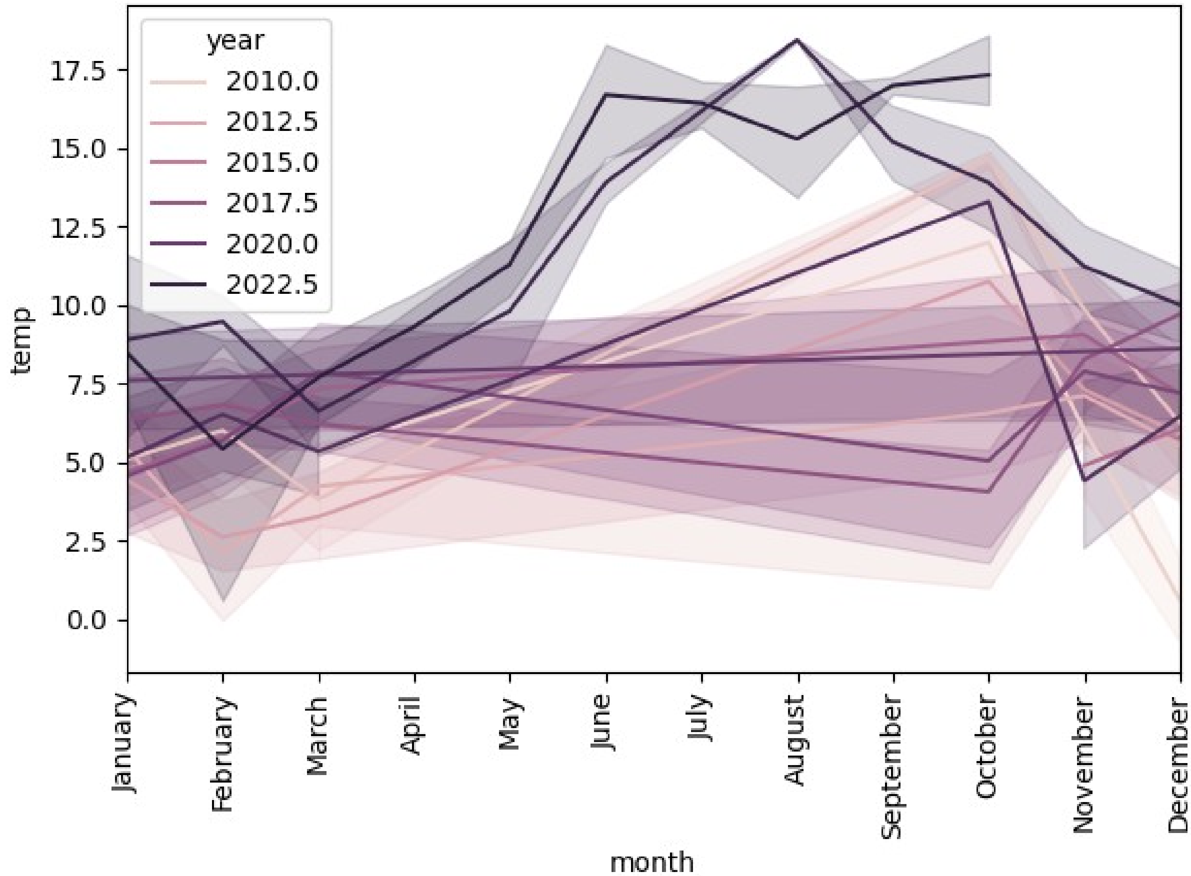
```
color='purple', label='polynomial')
plt.plot(df2.index, df2['temp'], label='Original', color='blue')

plt.legend()
plt.ylabel('Temperature')
plt.ylim(-1, 26)
_=plt.title('Interpolation')
```



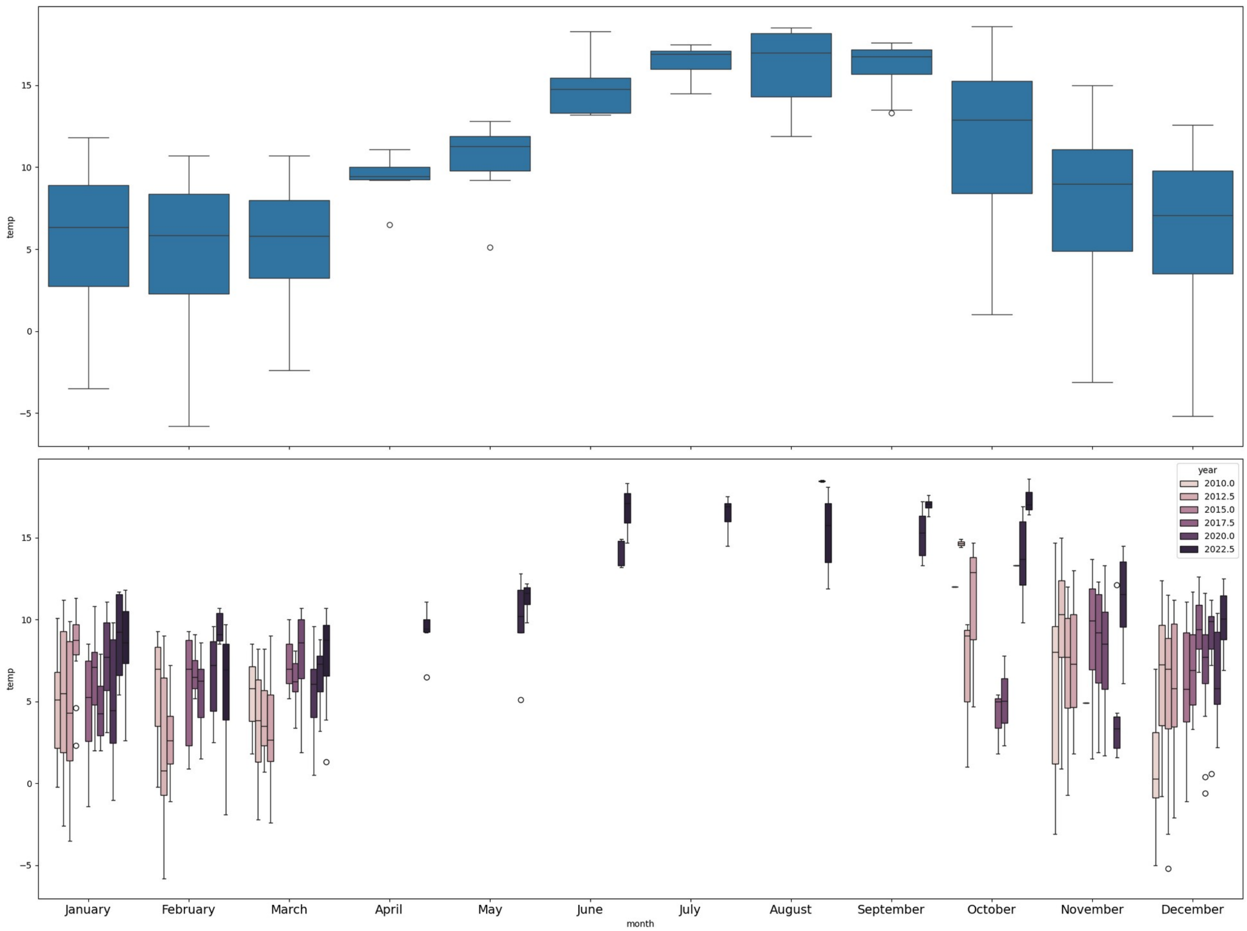

```
plt.figure(figsize=(16, 8))
_=sns.boxplot(x='year', y='temp', data=df)
_=plt.tight_layout()
```




```python
# Visualise trends across years
sns.lineplot(x='month', y='temp', data=df, hue='year')
_=plt.xticks(np.arange(1, 13), months_of_the_year, rotation=90)
_=plt.xlim(1, 12)  # limit x-axis
_=plt.tight_layout()
```



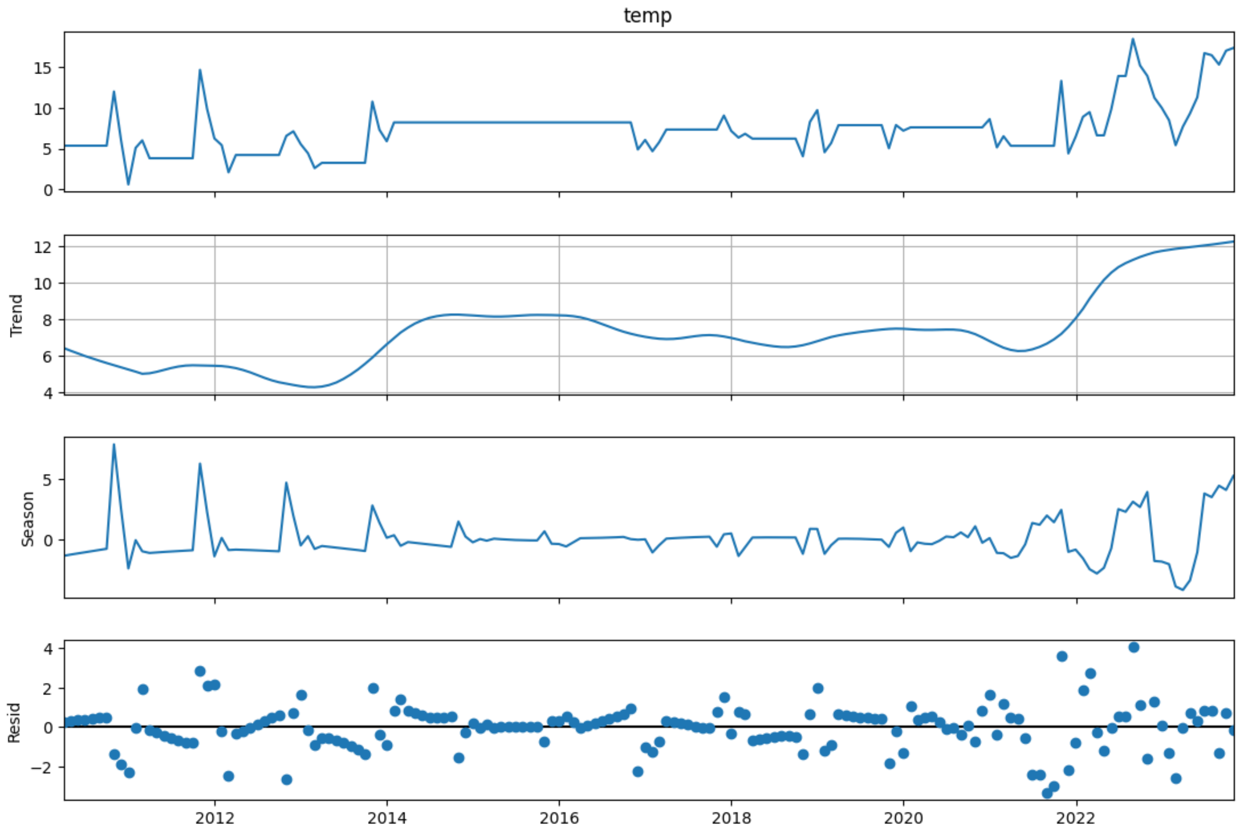

```
# Visualise trends across years
fix, ax = plt.subplots(2, 1, sharex=True, figsize=(20,15))
sns.boxplot(x='month', y='temp', data=df, ax=ax[0])  # top plot
sns.boxplot(x='month', y='temp', data=df, hue='year', ax=ax[1])  #
bottom plot
ax[1].set_xticks(np.arange(0, 12), months_of_the_year, fontsize=14)
plt.tight_layout()
```



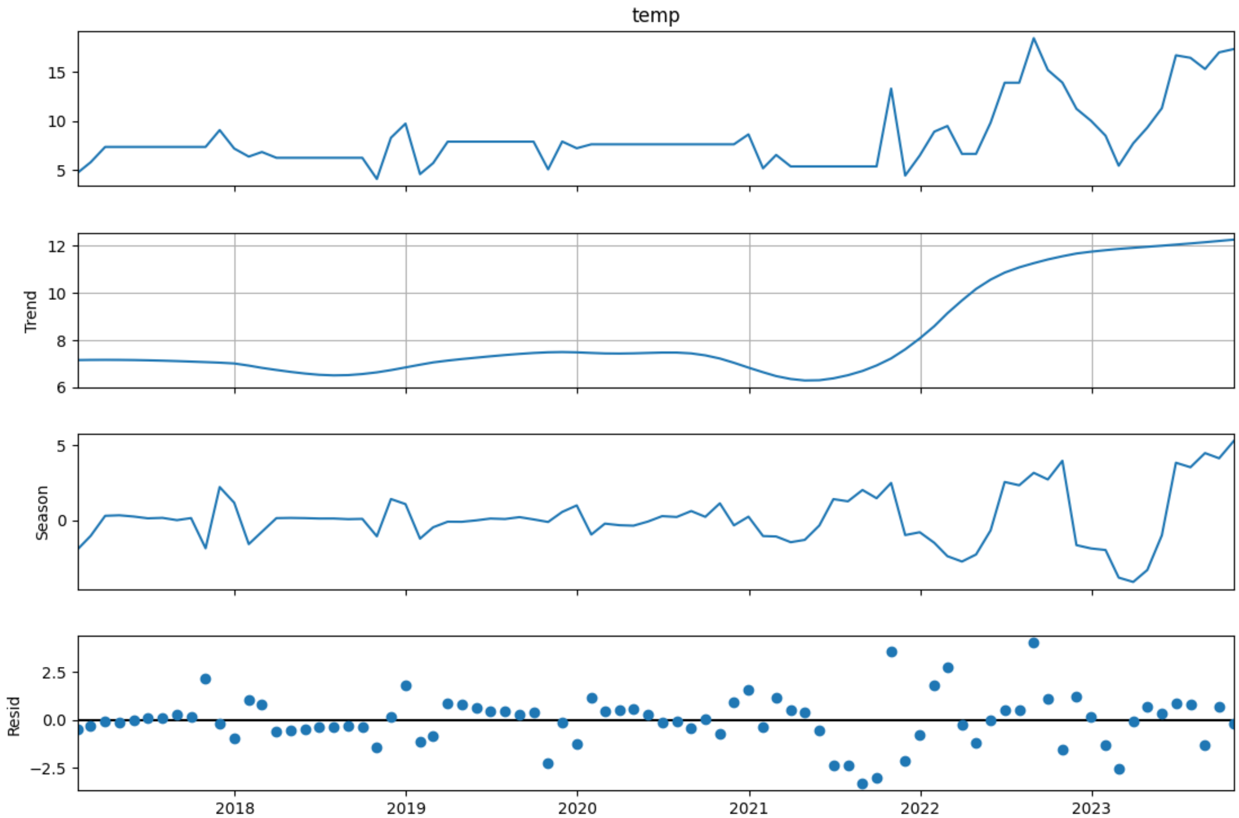

```
data_ds = df['temp'].resample('M').mean().ffill().to_frame()  # one value per month
data_ds
```

{"summary":"{\n  \"name\": \"data_ds\",\n  \"rows\": 164,\n  \"fields\": [\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 3.0209873807308147,\n        \"min\": 0.6166666666666666,\n        \"max\": 18.45,\n        \"samples\": [\n          8.216666666666667,\n          5.366666666666667,\n          6.463636363636363\n        ],\n        \"num_unique_values\": 70,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"data_ds"}

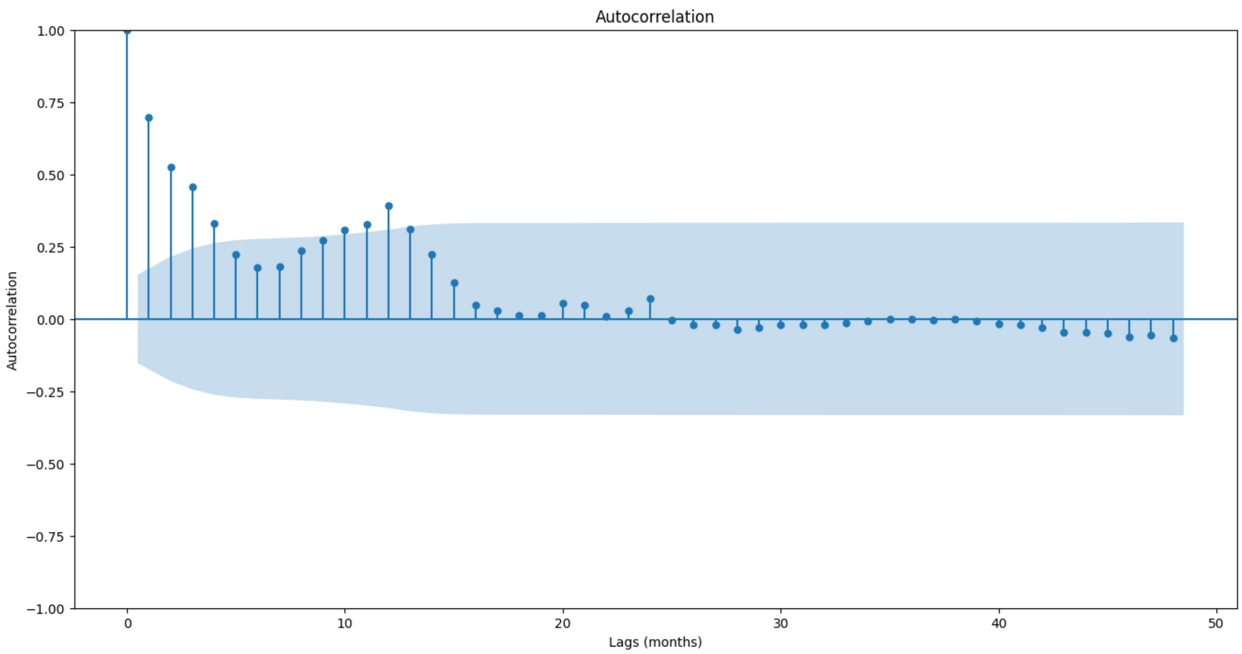
```
# Try decomposition on the resampled dataset
from statsmodels.tsa.seasonal import seasonal_decompose, STL
decomposition = STL(data_ds['temp']).fit()
fig = decomposition.plot()
fig.set_size_inches(12,8)
fig.axes[1].grid()
```



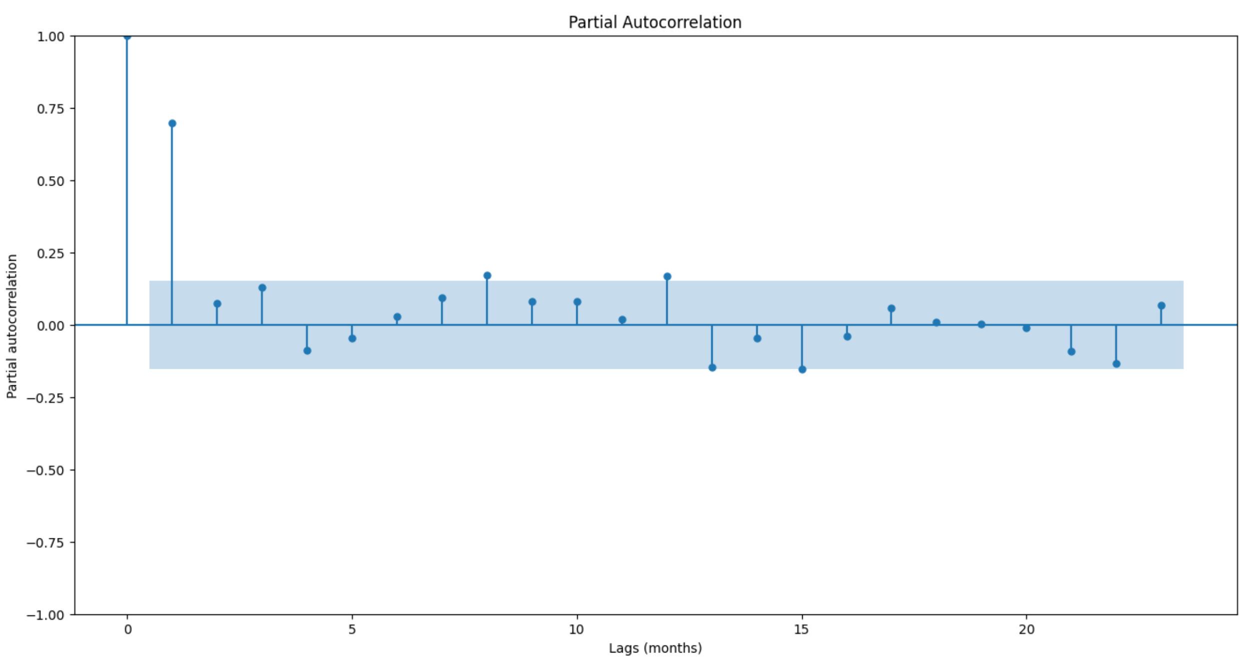

```
# Try decomposition on the resampled dataset, using only the full
years
decomposition = STL(data_ds.loc['2017':'2024', 'temp']).fit()
fig = decomposition.plot()
fig.set_size_inches(12,8)
fig.axes[1].grid()
```




```
# Statistical test for stationarity: Augmented Dickey-Fuller (ADF)
test
adf_result = adfuller(data_ds['temp'])
print('ADF Statistic %.2f:' % adf_result[0])
print('ADF p-value: %.4f:' % adf_result[1])
# p-value << 0.05 ==> timeseries does not have a unit root

ADF Statistic -0.13:
ADF p-value: 0.9465:

adf_result = adfuller(data_ds.loc['2017':'2024', 'temp'])  # ADF test
on the full years only. Is there a trend?
print('ADF Statistic %.2f:' % adf_result[0])
print('ADF p-value: %.4f:' % adf_result[1])

ADF Statistic 0.10:
ADF p-value: 0.9659:

# Autocorrelation (can help us with modelling later)
fig, ax = plt.subplots(figsize=(16,8))
_=plot_acf(data_ds['temp'], lags=48, ax=ax)  # each lag is one month,
so we're looking at 4 years worth of past data
_=plt.xlabel('Lags (months)')
_=plt.ylabel('Autocorrelation')
```

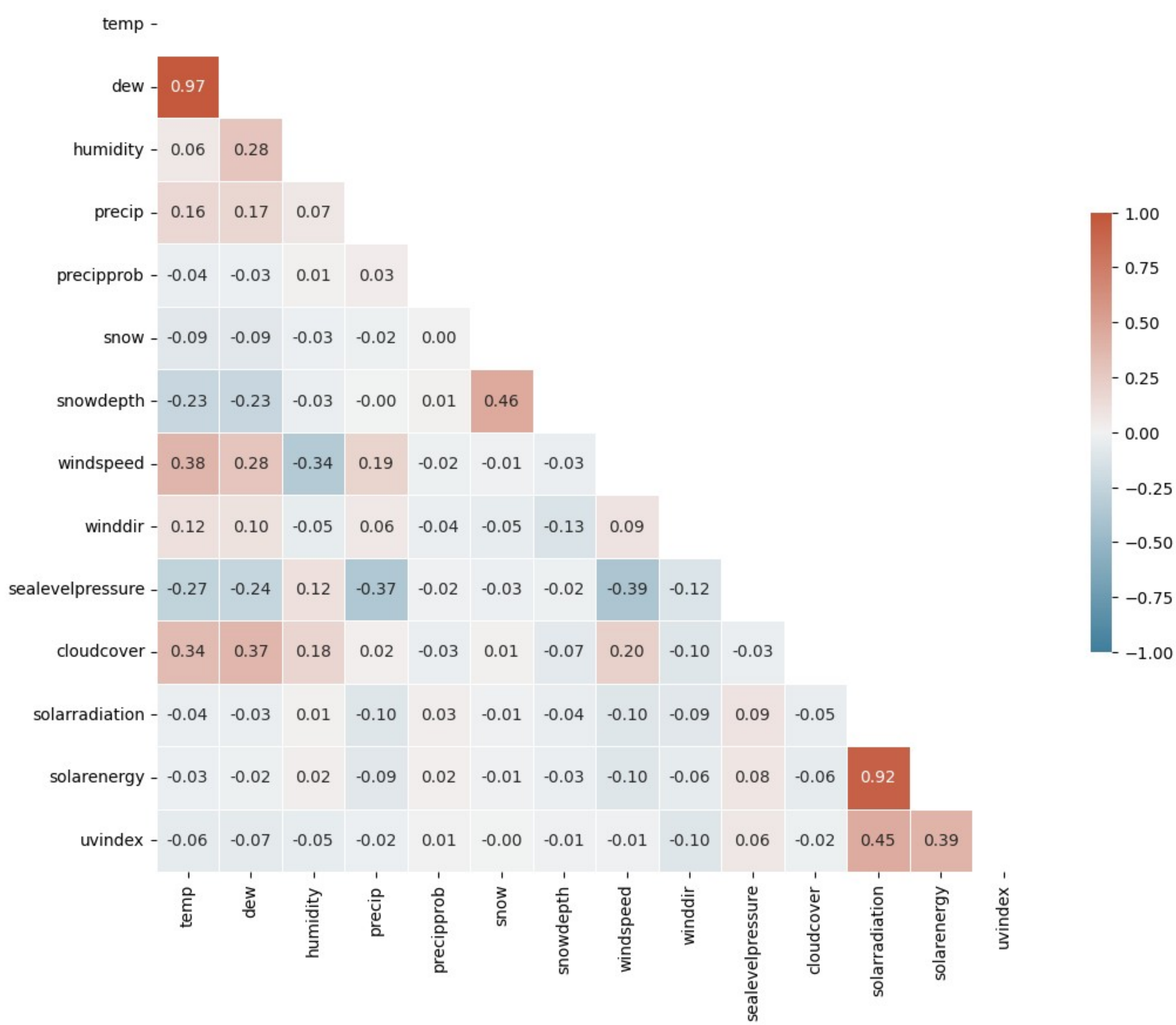



```
# Partial autocorrelation (can help us with modelling later)
fig, ax = plt.subplots(figsize=(16,8))
_=plot_pacf(data_ds['temp'], ax=ax)
_=plt.xlabel('Lags (months)')
_=plt.ylabel('Partial autocorrelation')
```

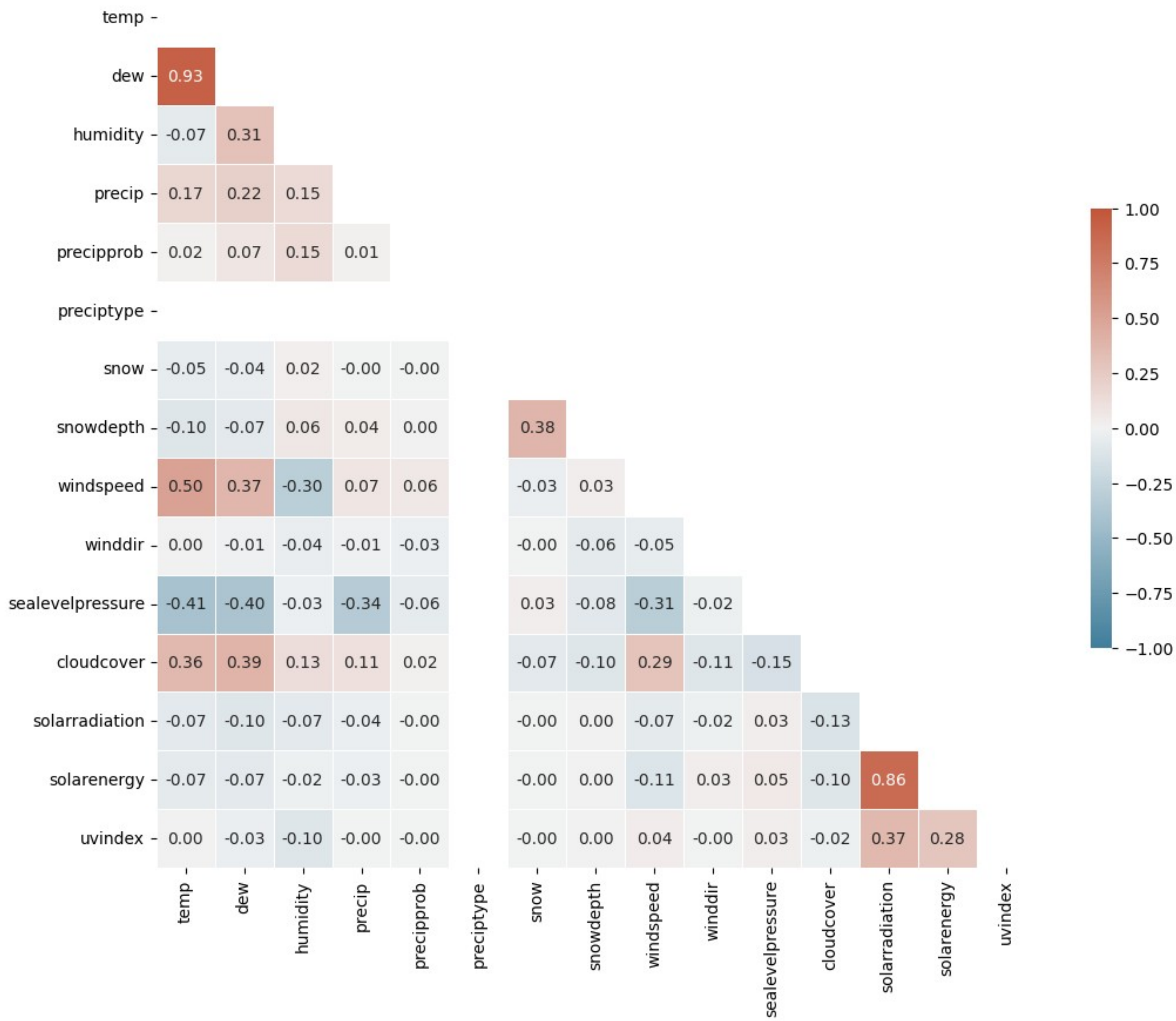



```
# https://seaborn.pydata.org/examples/many_pairwise_correlations.html

# Compute the correlation matrix
```

```
corr = df.iloc[:, :-3].corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
fmt='.2f')
plt.tight_layout()
```



```
# Convert columns to numeric type if necessary
df_numeric = df.iloc[:, :-3].apply(pd.to_numeric, errors='coerce')

# Compute the correlation matrix
corr_diff = df_numeric.diff().corr()
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_diff, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_diff, mask=mask, cmap=cmap, vmin=-1, vmax=1,
center=0, annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
fmt='.2f')
plt.tight_layout()
plt.show()
```



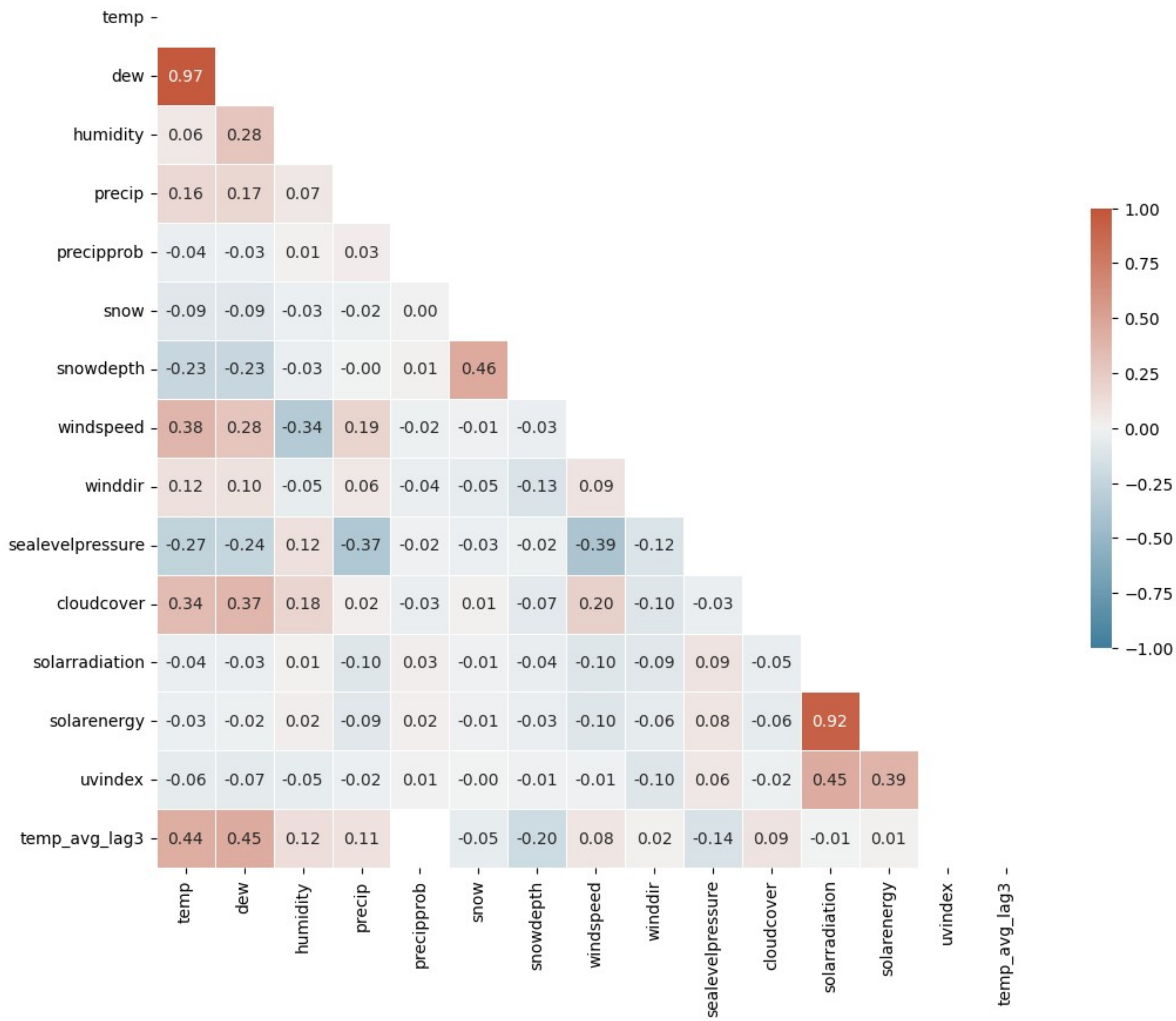

```python
df2 = df.iloc[:, :-3].copy()
df2['temp_avg_lag3'] = df2['temp'].shift(-3)
df2.head()
```

{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 4960,\n  \"fields\": [\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 4.502114672691266,\n        \"min\": -5.8,\n        \"max\": 18.6,\n        \"samples\": [\n          2.4,\n          9.4,\n          15.0\n        ],\n        \"num_unique_values\": 193,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"dew\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 4.63943049276453,\n        \"min\": -6.9,\n        \"max\": 17.5,\n        \"samples\": [\n          8.5,\n          -2.0,\n          -1.6\n        ],\n        \"num_unique_values\": 198,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"humidity\",\n      \"properties\":

```
{\n        \"dtype\": \"number\",\n        \"std\":
6.477026219315302,\n        \"min\": 64.25,\n        \"max\": 100.0,\n
\"samples\": [\n          92.36,\n          92.67,\n          92.7\n
],\n        \"num_unique_values\": 622,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"precip\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2.2224425362941083,\n        \"min\":
0.0,\n        \"max\": 23.733,\n        \"samples\": [\n
1.228,\n          0.276,\n          1.609\n        ],\n
\"num_unique_values\": 385,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"precipprob\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.042441332836129,\n        \"min\":
0.0,\n        \"max\": 100.0,\n        \"samples\": [\n          0.0,\
n          100.0\n        ],\n        \"num_unique_values\": 2,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"preciptype\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"samples\": [\n          \"rain\",\n          \"rain,snow\"\
n        ],\n        \"num_unique_values\": 3,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"snow\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.004903020264658105,\n
\"min\": 0.0,\n        \"max\": 0.13,\n        \"samples\": [\n
0.0,\n          0.04\n        ],\n        \"num_unique_values\": 3,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"snowdepth\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.39570796414247517,\n        \"min\": 0.0,\n        \"max\": 5.0,\n
\"samples\": [\n          4.75,\n          2.0\n        ],\n
\"num_unique_values\": 12,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"windspeed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 10.368344360719101,\n        \"min\":
0.8,\n        \"max\": 66.0,\n        \"samples\": [\n          30.8,\
n          15.9\n        ],\n        \"num_unique_values\": 310,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"winddir\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
101.91043086560504,\n        \"min\": 2.0,\n        \"max\": 360.0,\n
\"samples\": [\n          197.0,\n          91.0\n        ],\n
\"num_unique_values\": 293,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sealevelpressure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 13.091025102392456,\n        \"min\":
968.1,\n        \"max\": 1043.2,\n        \"samples\": [\n
1026.6,\n          987.2\n        ],\n        \"num_unique_values\":
394,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\
n      }\n    },\n    {\n      \"column\": \"cloudcover\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
27.910066073116568,\n        \"min\": 0.0,\n        \"max\": 100.0,\n
\"samples\": [\n          76.4,\n          50.3\n        ],\n
\"num_unique_values\": 378,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"solarradiation\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7.480077018670658,\n        \"min\":
0.0,\n        \"max\": 59.3,\n        \"samples\": [\n          17.6,\
n          9.0\n        ],\n        \"num_unique_values\": 97,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"solarenergy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.030652888741679738,\n        \"min\": 0.0,\n        \"max\": 0.2,\n
\"samples\": [\n          0.1,\n          0.0\n        ],\n
\"num_unique_values\": 3,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"uvindex\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.06237796929458344,\n        \"min\": 0.0,\n
\"max\": 1.0,\n        \"samples\": [\n          1.0,\n          0.0\n
],\n        \"num_unique_values\": 2,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"temp_avg_lag3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.506497352455208,\n
\"min\": -5.8,\n        \"max\": 18.6,\n        \"samples\": [\n
6.8,\n          9.4\n        ],\n        \"num_unique_values\": 193,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df2"}
```


```
# Let's see what happens if we do the differential operation again.
df2 = df.iloc[:, :-3].copy()
df2['temp_avg_lag3'] = df2['temp'].shift(-3)
corr2 = df2.corr()
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr2, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr2, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
fmt='.2f')
plt.tight_layout()
```




#Colchester Data Visualization

##Exploratory Data Analysis

```
# Specify the folder directory where the dataset is located.
df_path = GOOGLE_DRIVE_PATH

#list of all files in the path
file_list = [file for file in os.listdir(GOOGLE_DRIVE_PATH) if
file.endswith('.csv')]

brighton_df = []

# Iterate over each CSV file
for file_ in file_list:
    # Construct the full path to the CSV file
```

```python
    file_path = os.path.join(df_path, file_)

    # Check for file has an index column named '0'
    index_column = pd.read_csv(file_path, nrows=1).columns[0] == '0'

    # Change header value based on the  value of the column
    header = 1 if index_column else "infer"

    current_dataframe = pd.read_csv(file_path, header=header)

    # If the file name contains "Colchester", add its DataFrame to the
list
    if file_.startswith('colchester'):
        brighton_df.append(current_dataframe)

# Concantenate all the datasets into one
f_colchester_df = pd.concat(brighton_df, ignore_index=True)

#information about the DataFrame
print(f_colchester_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188024 entries, 0 to 188023
Data columns (total 16 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   datetime          188024 non-null  object
 1   temp              187606 non-null  float64
 2   dew               187600 non-null  float64
 3   humidity          187602 non-null  float64
 4   precip            187437 non-null  float64
 5   precipprob        187590 non-null  float64
 6   preciptype        19448 non-null   object
 7   snow              186478 non-null  float64
 8   snowdepth         186429 non-null  float64
 9   windspeed         187627 non-null  float64
 10  winddir           187595 non-null  float64
 11  sealevelpressure  180462 non-null  float64
 12  cloudcover        184405 non-null  float64
 13  solarradiation    94894 non-null   float64
 14  uvindex           94897 non-null   float64
 15  solarenergy       94908 non-null   float64
dtypes: float64(14), object(2)
memory usage: 23.0+ MB
None
```

```python
# Print dimensions of the dataset
print("Dataset dimensions:", f_colchester_df.shape)
```

```
Dataset dimensions: (188024, 16)
```

```
# Print first 2 rows of the dataset
print("First few rows of the dataset:")
print(f_colchester_df.head(2))

First few rows of the dataset:
              datetime  temp  dew  humidity  precip  precipprob
preciptype  \
0  2000-01-01T00:00:00   5.9  5.7     98.63     0.0         0.0
NaN
1  2000-01-01T01:00:00   6.4  5.8     96.12     0.0         0.0
NaN

   snow  snowdepth  windspeed  winddir  sealevelpressure
cloudcover  \
0   NaN        NaN        9.4    210.0            1020.6         NaN

1   NaN        NaN       15.1    233.0            1020.4       100.0


   solarradiation  uvindex  solarenergy
0             NaN      NaN          NaN
1             NaN      NaN          NaN

# Summary
print("Summary:")
print(f_colchester_df.describe())

Summary:
                temp            dew       humidity         precip  \
count  187606.000000  187600.000000  187602.000000  187437.000000
mean       10.503048       7.190720      81.596979       0.067773
std         6.004301       4.956358      14.085087       0.623144
min        -9.600000     -10.800000      22.430000       0.000000
25%         6.100000       3.600000      73.490000       0.000000
50%        10.300000       7.400000      85.480000       0.000000
75%        14.700000      10.900000      92.680000       0.000000
max        35.000000      21.500000     100.000000      84.324000

          precipprob           snow      snowdepth      windspeed  \
count  187590.000000  186478.000000  186429.000000  187627.000000
mean       10.276578       0.001008       0.062193      15.998126
std        30.357300       0.043173       0.602604       7.916611
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000      10.100000
50%         0.000000       0.000000       0.000000      14.600000
75%         0.000000       0.000000       0.000000      20.700000
max       100.000000       9.100000      15.230000      74.700000

             winddir  sealevelpressure     cloudcover  solarradiation
\
count  187595.000000     180462.000000  184405.000000    94894.000000
```

```
mean     198.262143       1013.922618     59.510905     122.256657
std       95.653613         29.713385     31.985577     198.631750
min        0.000000          0.000000      0.000000       0.000000
25%      126.000000       1008.300000     34.000000       0.000000
50%      218.000000       1015.500000     66.600000       9.000000
75%      267.000000       1022.000000     88.900000     169.500000
max      360.000000       1048.900000    100.000000    1054.000000

            uvindex   solarenergy
count  94897.000000  94908.000000
mean       1.203104      0.439357
std        2.006343      0.715817
min        0.000000      0.000000
25%        0.000000      0.000000
50%        0.000000      0.000000
75%        2.000000      0.600000
max       10.000000      3.800000
```

```
# Counting for missing values
print("Missing values:")
print(f_colchester_df.isnull().sum())
```

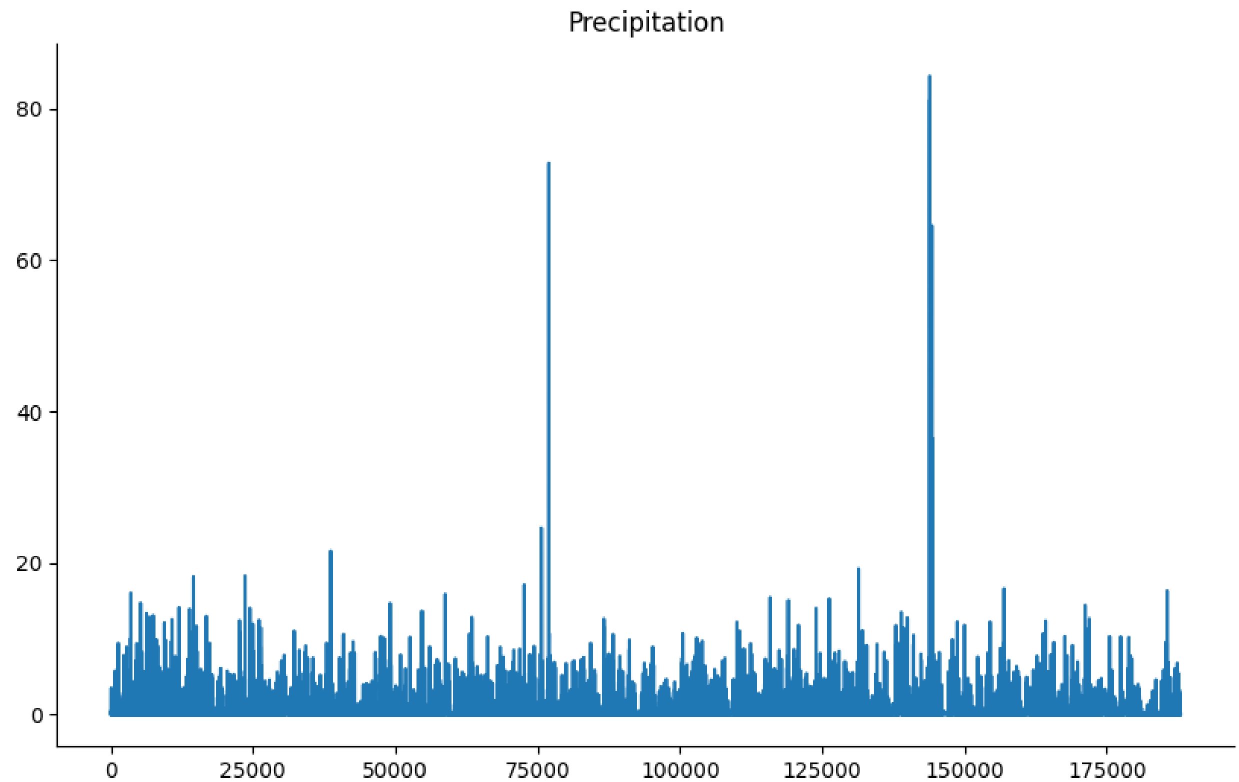
```
Missing values:
datetime                 0
temp                   418
dew                    424
humidity               422
precip                 587
precipprob             434
preciptype          168576
snow                  1546
snowdepth             1595
windspeed              397
winddir                429
sealevelpressure      7562
cloudcover            3619
solarradiation       93130
uvindex              93127
solarenergy          93116
dtype: int64
```

```
# Info about  categorical variables
print("Categories & frequencies for categorical variables:")
```

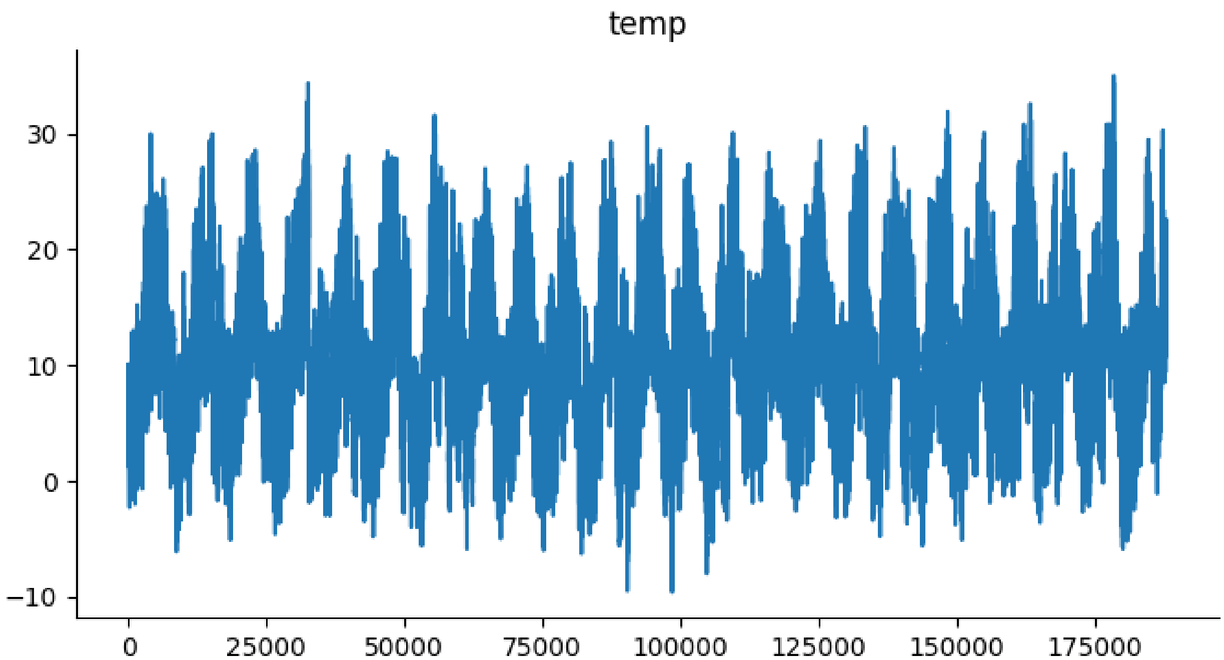
```python
for col in f_colchester_df.select_dtypes(include='object').columns:
    print(f_colchester_df[col].value_counts())
```

```
Categories & frequencies for categorical variables:
2008-10-26T01:00:00    2
2000-11-16T18:00:00    2
2000-11-16T11:00:00    2
2000-11-16T12:00:00    2
2000-11-16T13:00:00    2
                      ..
2007-05-16T01:00:00    1
2007-05-16T02:00:00    1
2007-05-16T03:00:00    1
2007-05-16T04:00:00    1
2023-10-07T23:00:00    1
Name: datetime, Length: 187979, dtype: int64
rain         18543
rain,snow      851
snow            54
Name: preciptype, dtype: int64
```

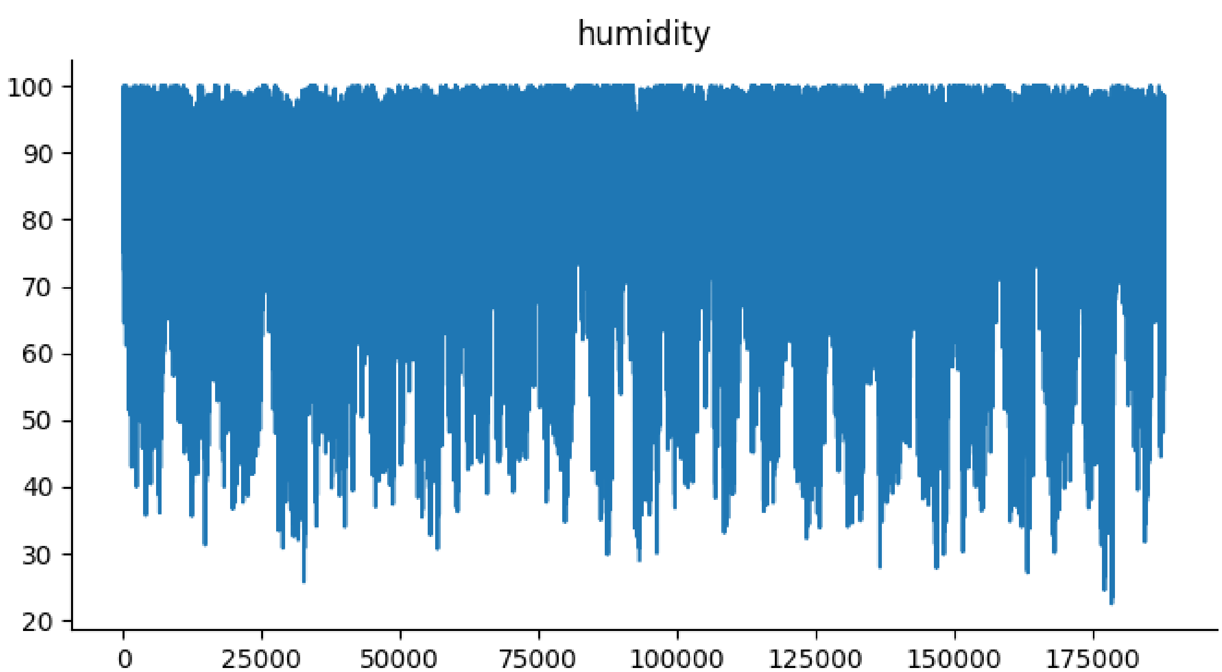
```python
# @title Precipitation

from matplotlib import pyplot as plt
f_colchester_df['precip'].plot(kind='line', figsize=(10, 6),
title='Precipitation')
plt.gca().spines[['top', 'right']].set_visible(False)
```



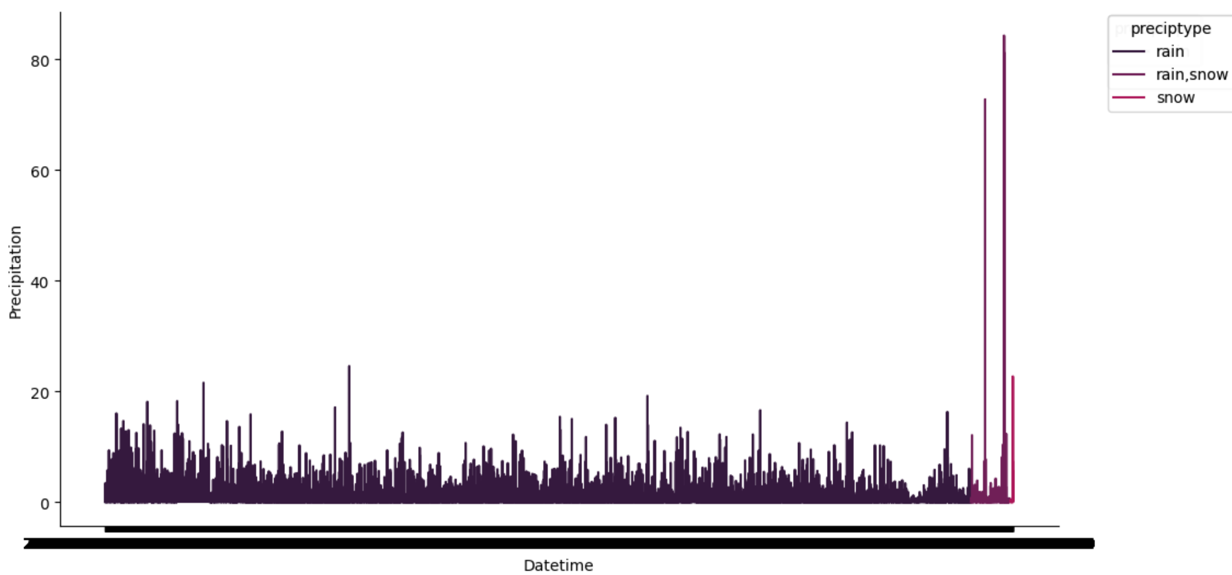

```
# @title Temperature

from matplotlib import pyplot as plt
f_colchester_df['temp'].plot(kind='line', figsize=(8, 4),
title='temp')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```python
# @title Humidity

from matplotlib import pyplot as plt
f_colchester_df['humidity'].plot(kind='line', figsize=(8, 4),
title='humidity')
plt.gca().spines[['top', 'right']].set_visible(False)
```

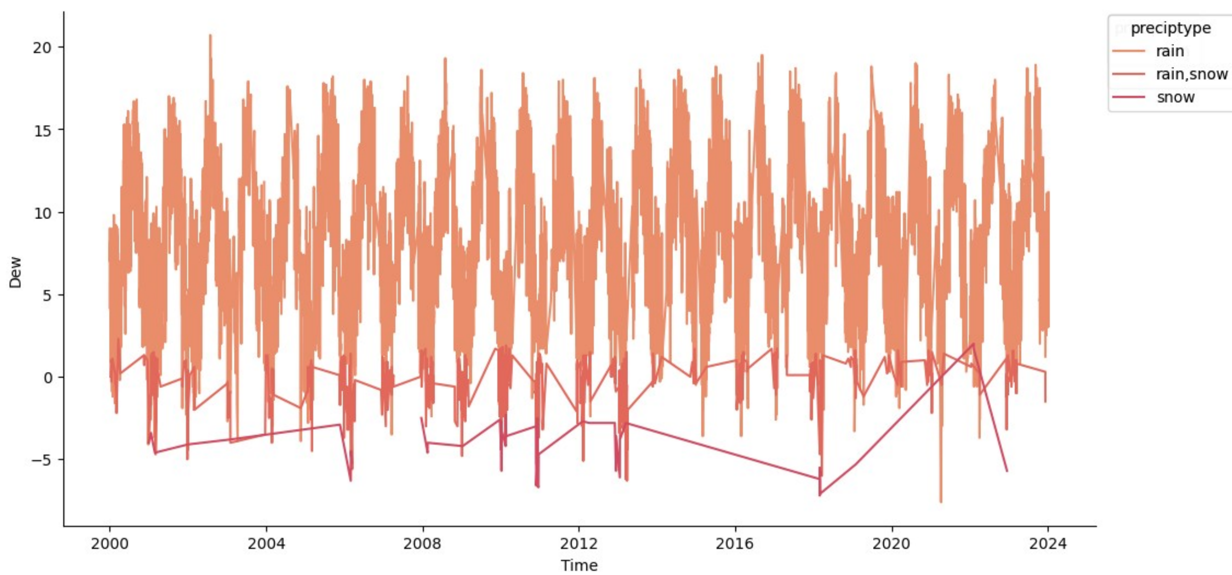


```
# @title DateTime VS Precipitation

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('rocket'))
  xs = series['datetime']
  ys = series['precip']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_colchester_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Datetime')
_ = plt.ylabel('Precipitation')
```



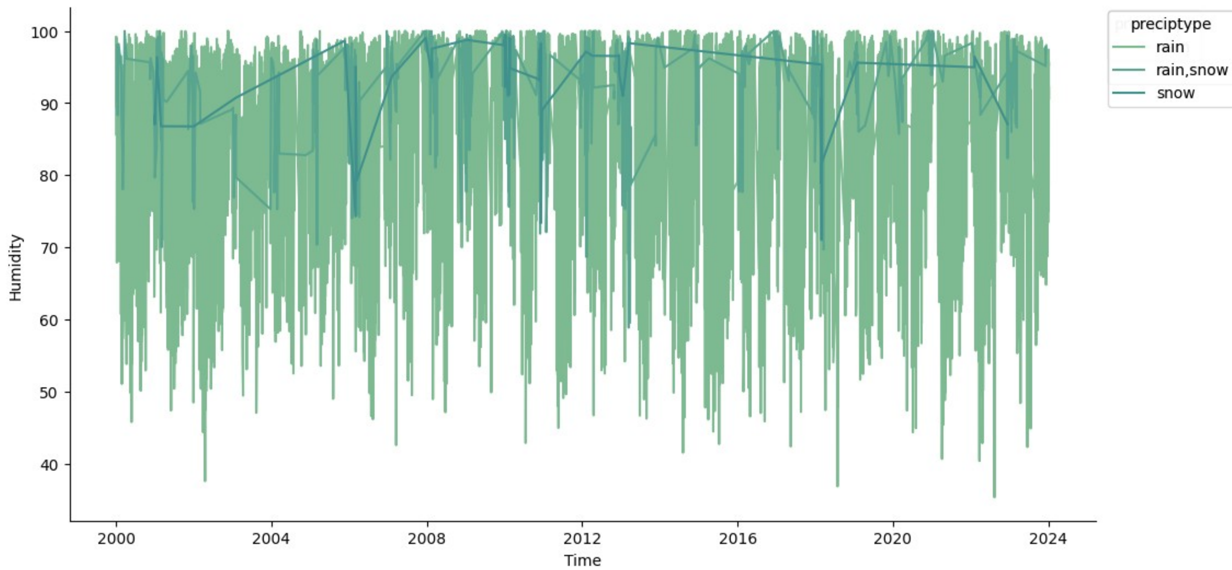

```
# @title Time VS Dew

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
```

```
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('flare'))
  xs = series['datetime']
  ys = series['dew']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_colchester_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('Dew')
```



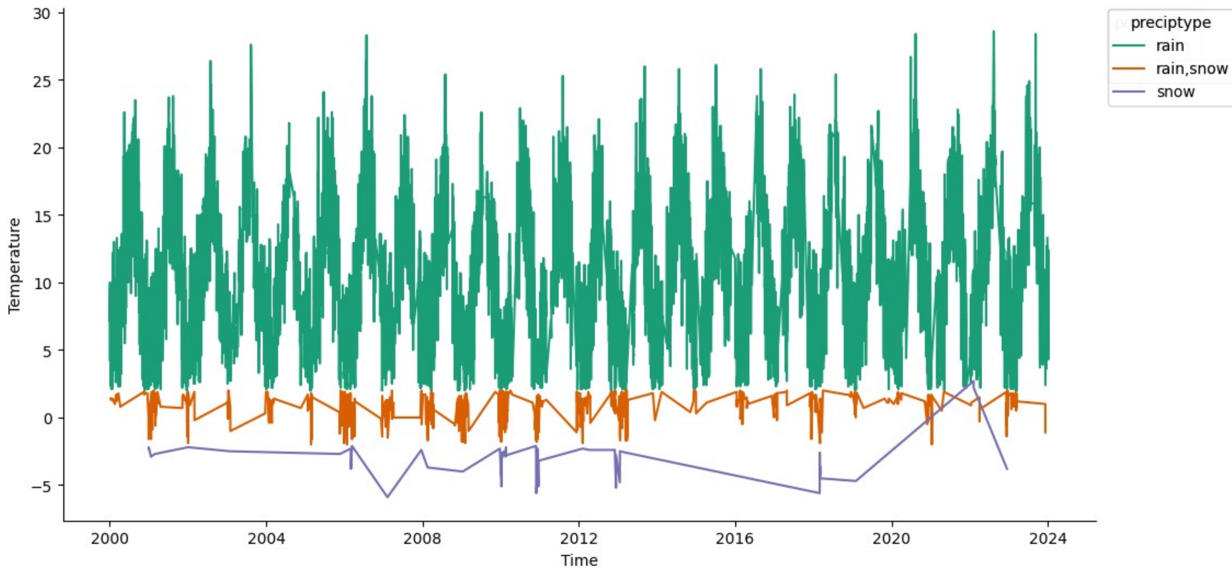

```
# @title Time VS Humidity

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('crest'))
  xs = series['datetime']
  ys = series['humidity']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
```

```
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_colchester_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('Humidity')
```



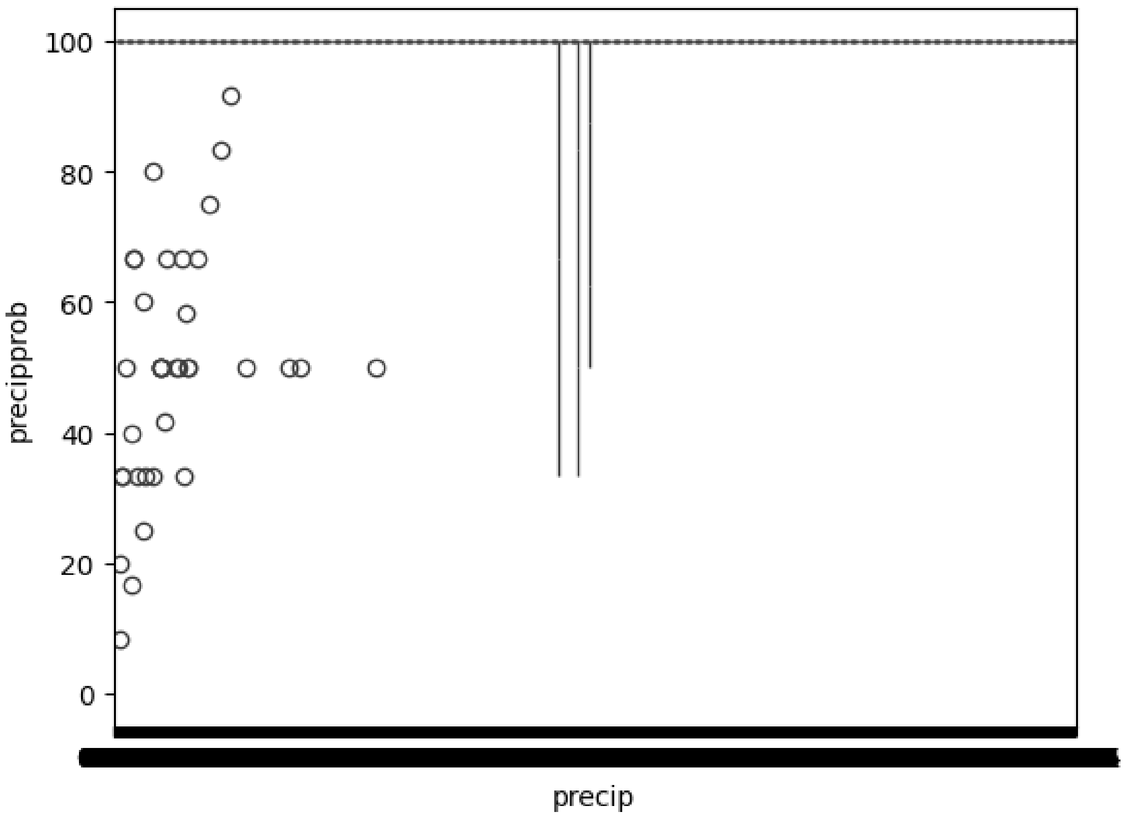

```
# @title Time vs Temperature

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  from matplotlib import pyplot as plt
  import seaborn as sns
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['datetime']
  ys = series['temp']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = f_colchester_df.sort_values('datetime', ascending=True)
for i, (series_name, series) in
enumerate(df_sorted.groupby('preciptype')):
```

```
  _plot_series(series, series_name, i)
  fig.legend(title='preciptype', bbox_to_anchor=(1, 1), loc='upper
left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('Temperature')
```

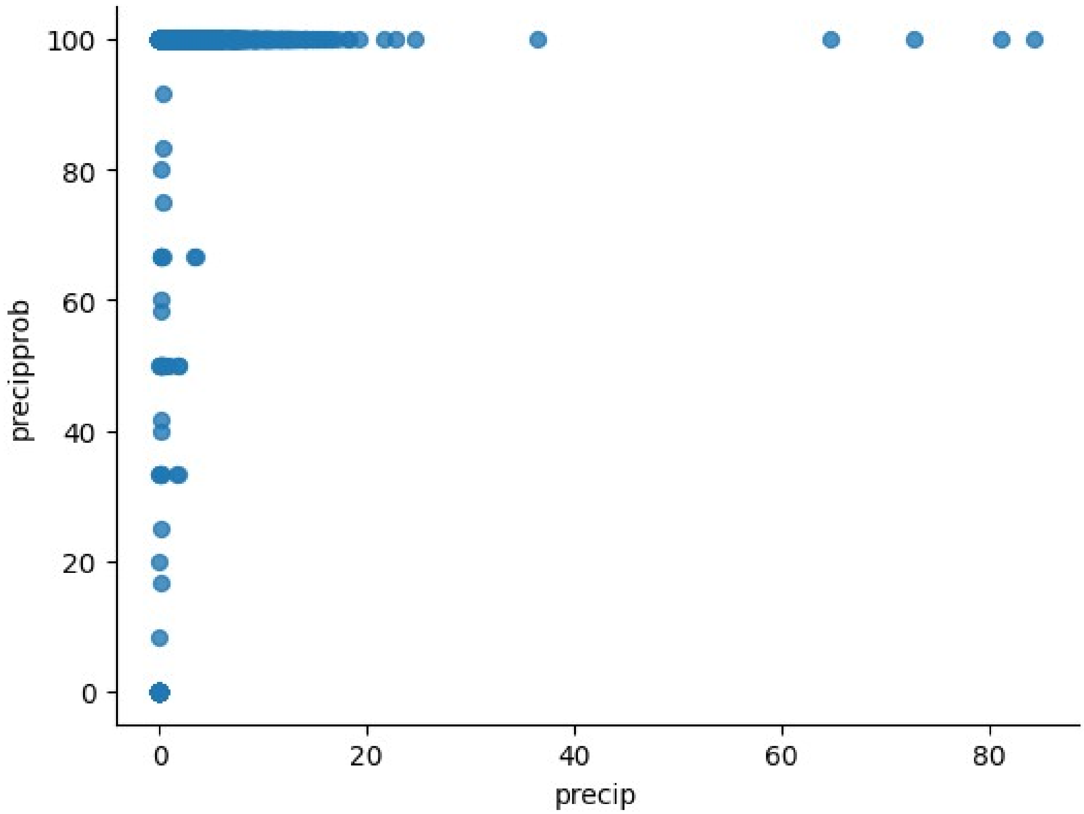



```
# @title Precipitaion VS Precipitation Probability

from matplotlib import pyplot as plt
sns.boxplot(x=f_colchester_df['precip'],
y=f_colchester_df['precipprob'])
```

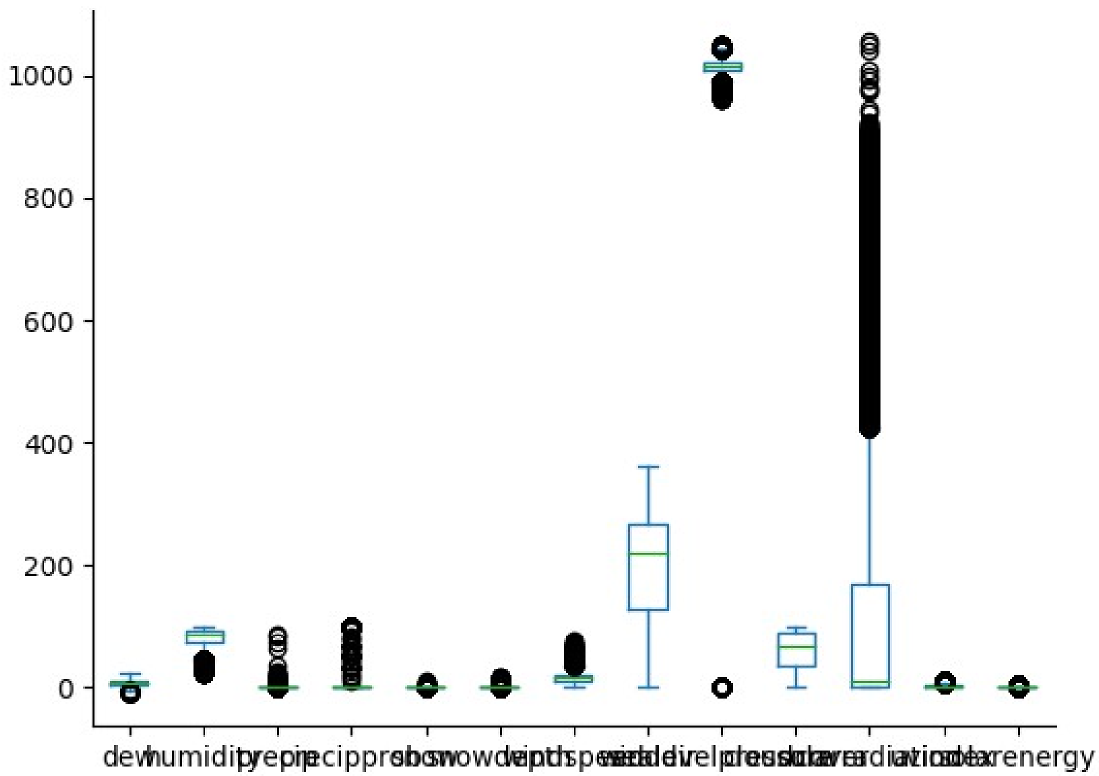
```
<Axes: xlabel='precip', ylabel='precipprob'>
```



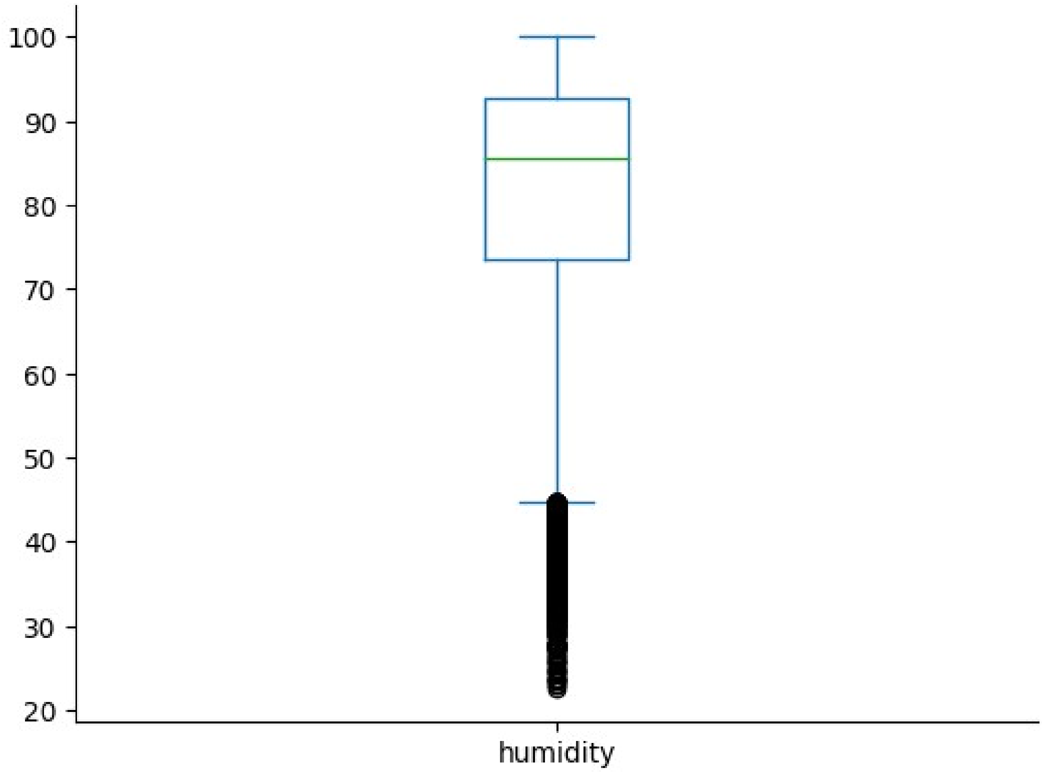

```
# @title Precipitaion VS Precipitation Probability
from matplotlib import pyplot as plt
f_colchester_df.plot(kind='scatter', x='precip', y='precipprob', s=32,
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



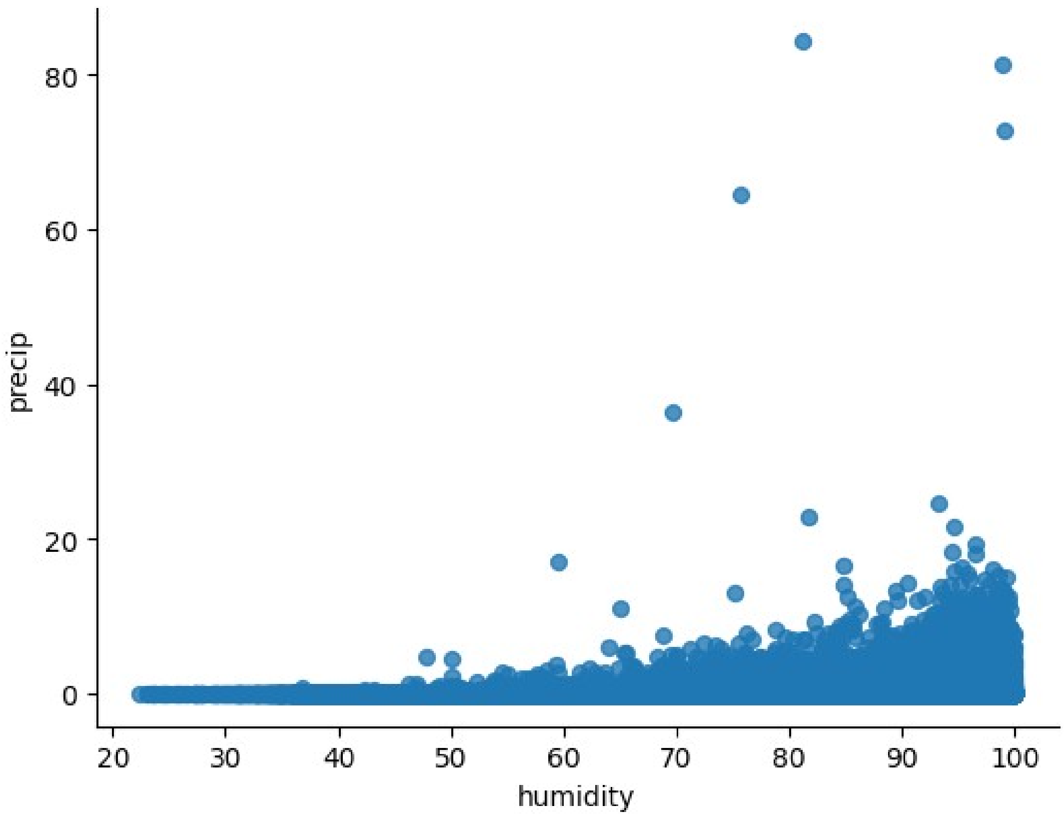

```
# @title Temperature vs Dew

from matplotlib import pyplot as plt
f_colchester_df.plot(kind='box', x='temp')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# @title Dew VS Humidity

from matplotlib import pyplot as plt
f_colchester_df.plot(kind='box', x='dew', y='humidity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

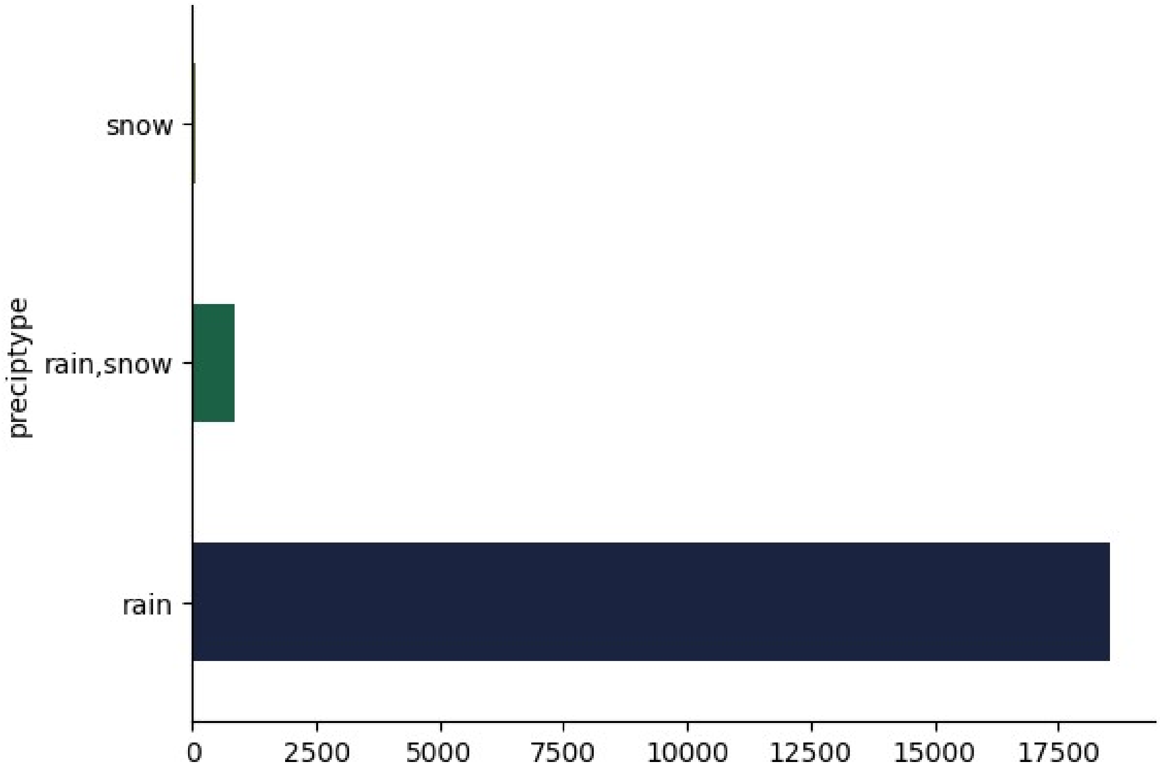



```
# @title Humidity VS Precipitation

from matplotlib import pyplot as plt
f_colchester_df.plot(kind='scatter', x='humidity', y='precip', s=32,
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

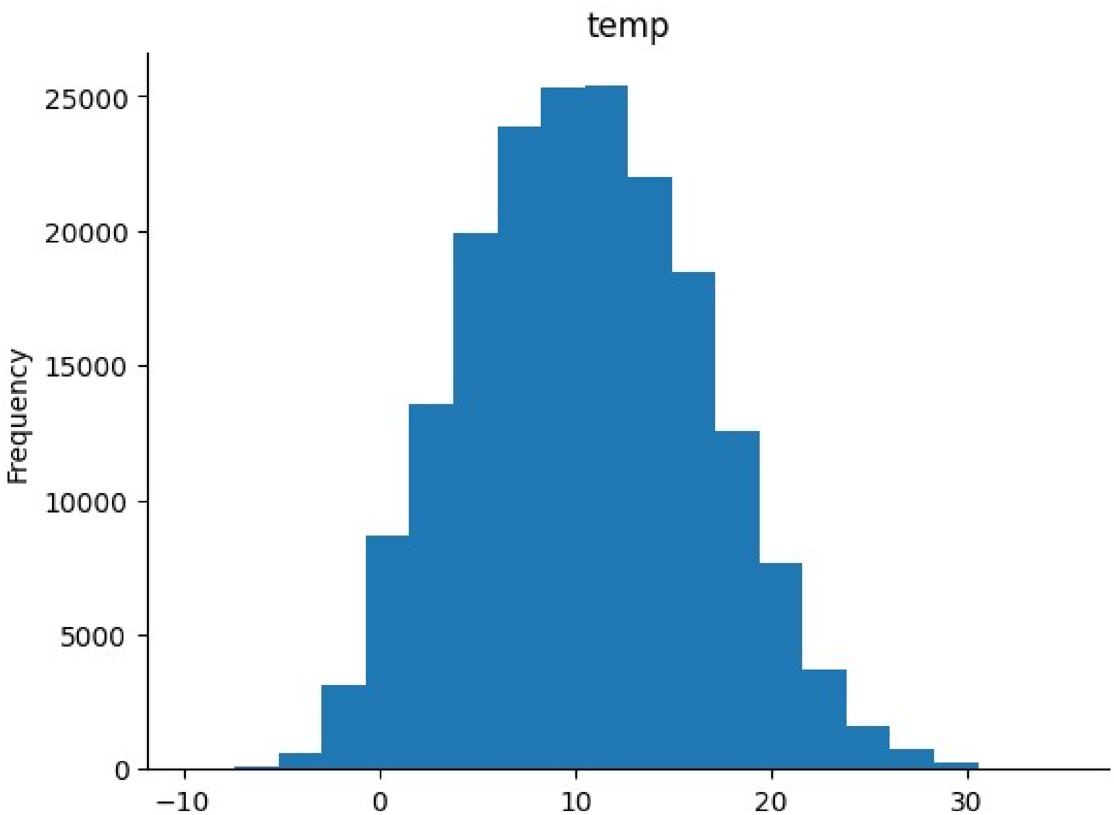



## Exploring Important Columns

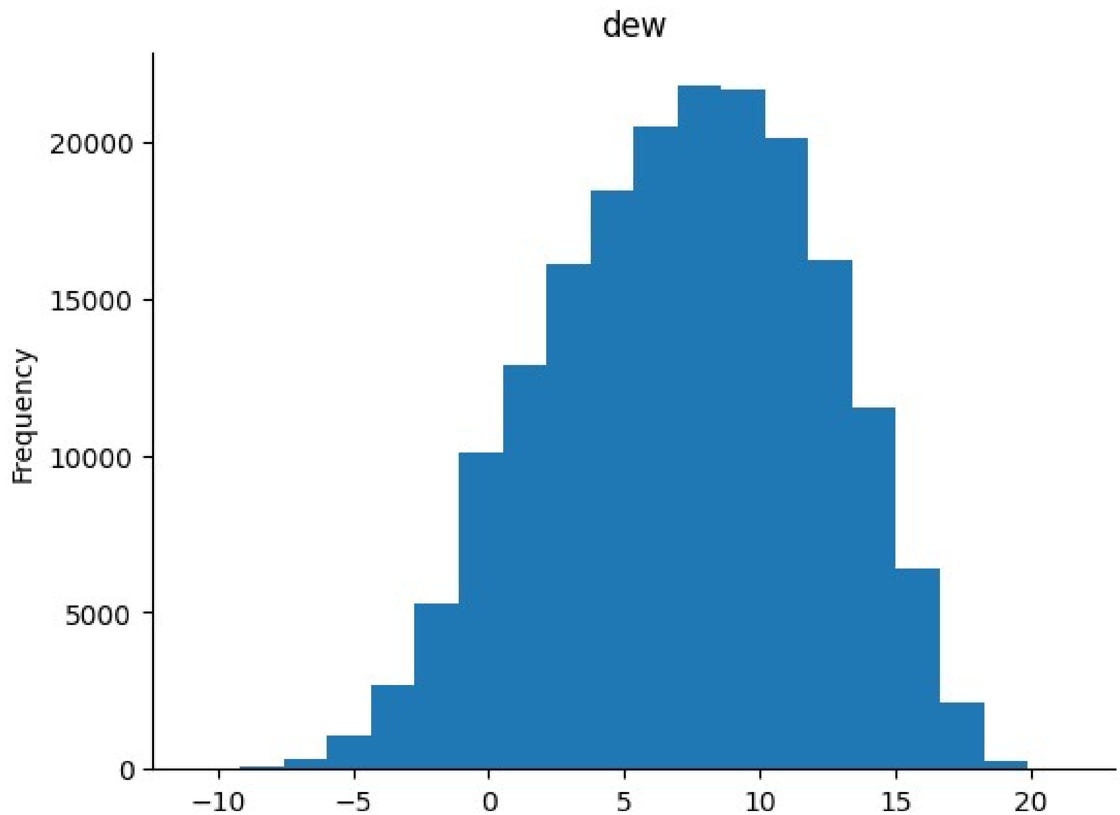
```
# @title Precipitation Type

from matplotlib import pyplot as plt
import seaborn as sns
f_colchester_df.groupby('preciptype').size().plot(kind='barh',
color=sns.palettes.mpl_palette('cubehelix'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



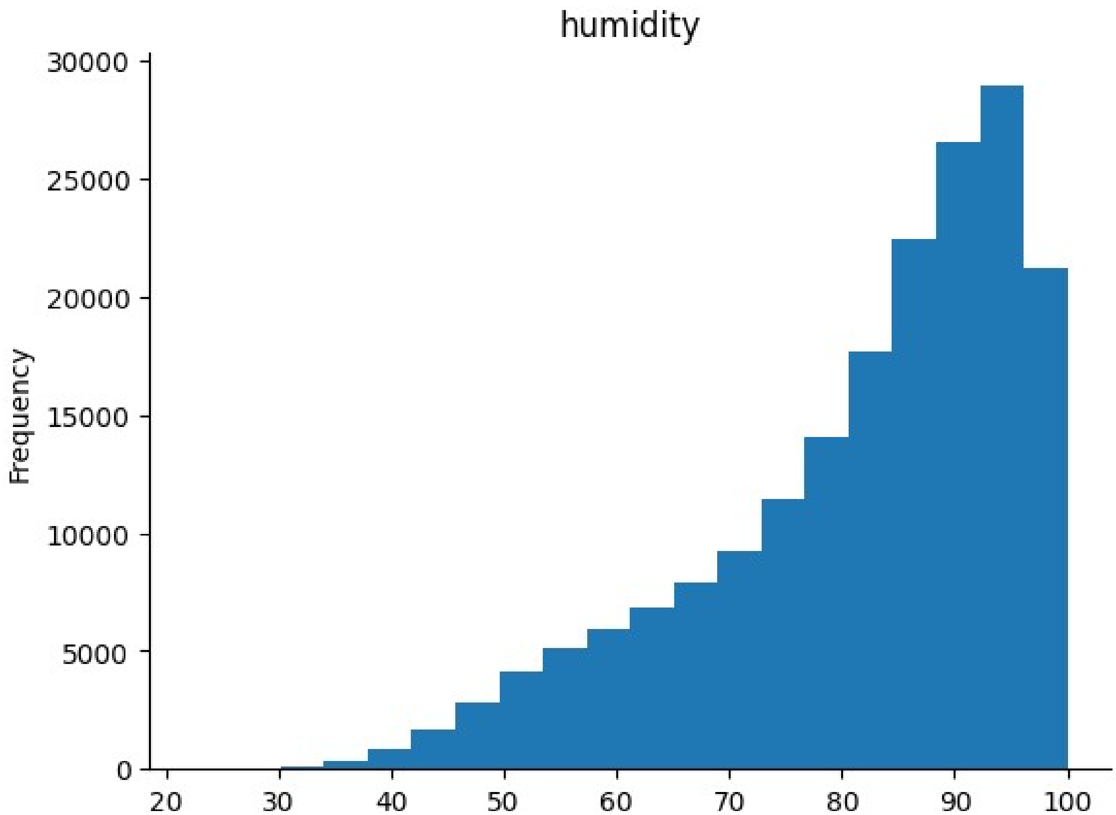

```
# @title Temperature

from matplotlib import pyplot as plt
f_colchester_df['temp'].plot(kind='hist', bins=20, title='temp')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



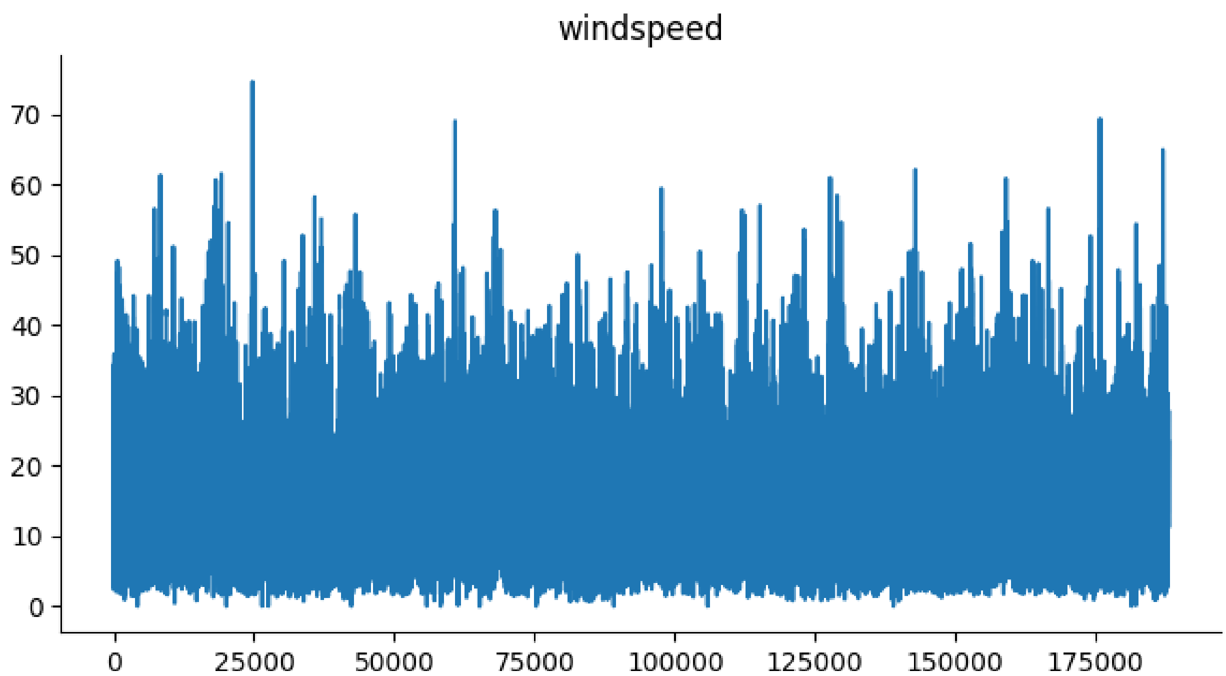

```
# @title Dew

from matplotlib import pyplot as plt
f_colchester_df['dew'].plot(kind='hist', bins=20, title='dew')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# @title Humidity

from matplotlib import pyplot as plt
f_colchester_df['humidity'].plot(kind='hist', bins=20,
title='humidity')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# @title Windspeed
from matplotlib import pyplot as plt
f_colchester_df['windspeed'].plot(kind='line', figsize=(8, 4),
title='windspeed')
plt.gca().spines[['top', 'right']].set_visible(False)
```




## Time Series Analysis

```
months_of_the_year = ['January', 'February', 'March', 'April', 'May',
'June', 'July', 'August', 'September', 'October', 'November',
'December']

f_colchester_df.columns

Index(['datetime', 'temp', 'dew', 'humidity', 'precip', 'precipprob',
       'preciptype', 'snow', 'snowdepth', 'windspeed', 'winddir',
       'sealevelpressure', 'cloudcover', 'solarradiation', 'uvindex',
       'solarenergy'],
      dtype='object')

# Date will be our index. Let's convert it to a datetime type
f_colchester_df['datetime'] =
pd.to_datetime(f_colchester_df['datetime'], dayfirst=True)
f_colchester_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188024 entries, 0 to 188023
Data columns (total 16 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   datetime          188024 non-null  datetime64[ns]
 1   temp              187606 non-null  float64
 2   dew               187600 non-null  float64
 3   humidity          187602 non-null  float64
 4   precip            187437 non-null  float64
```

```
 5   precipprob        187590 non-null  float64
 6   preciptype        19448 non-null   object
 7   snow              186478 non-null  float64
 8   snowdepth         186429 non-null  float64
 9   windspeed         187627 non-null  float64
 10  winddir           187595 non-null  float64
 11  sealevelpressure  180462 non-null  float64
 12  cloudcover        184405 non-null  float64
 13  solarradiation    94894 non-null   float64
 14  uvindex           94897 non-null   float64
 15  solarenergy       94908 non-null   float64
dtypes: datetime64[ns](1), float64(14), object(1)
memory usage: 23.0+ MB
```

```
df_c = f_colchester_df.copy()
df_c['month'] = df_c['datetime'].dt.month
df_c['year'] = df_c['datetime'].dt.year
df_c['week_of_year'] = df_c['datetime'].dt.isocalendar().week
df_c
```

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

```
# Let's make the date column the index of the dataframe for easier slicing
df_c.set_index('datetime', inplace=True) # note we can only run this once, as it will delete the 'date' column.
df_c.head()
```

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

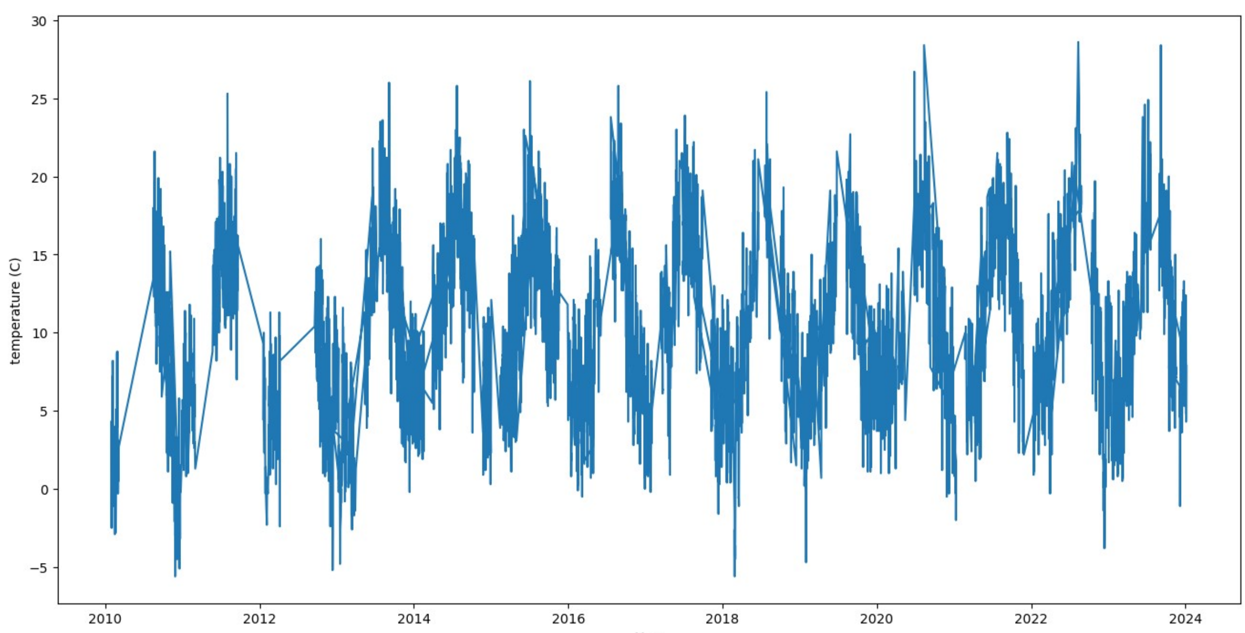
```
print(df_c.isna().any())
df_c=df_c.dropna()
print(df_c.isna().any())
```

```
temp                 True
dew                  True
humidity             True
precip               True
precipprob           True
preciptype           True
snow                 True
snowdepth            True
windspeed            True
winddir              True
sealevelpressure     True
cloudcover           True
solarradiation       True
uvindex              True
solarenergy          True
month               False
year                False
```

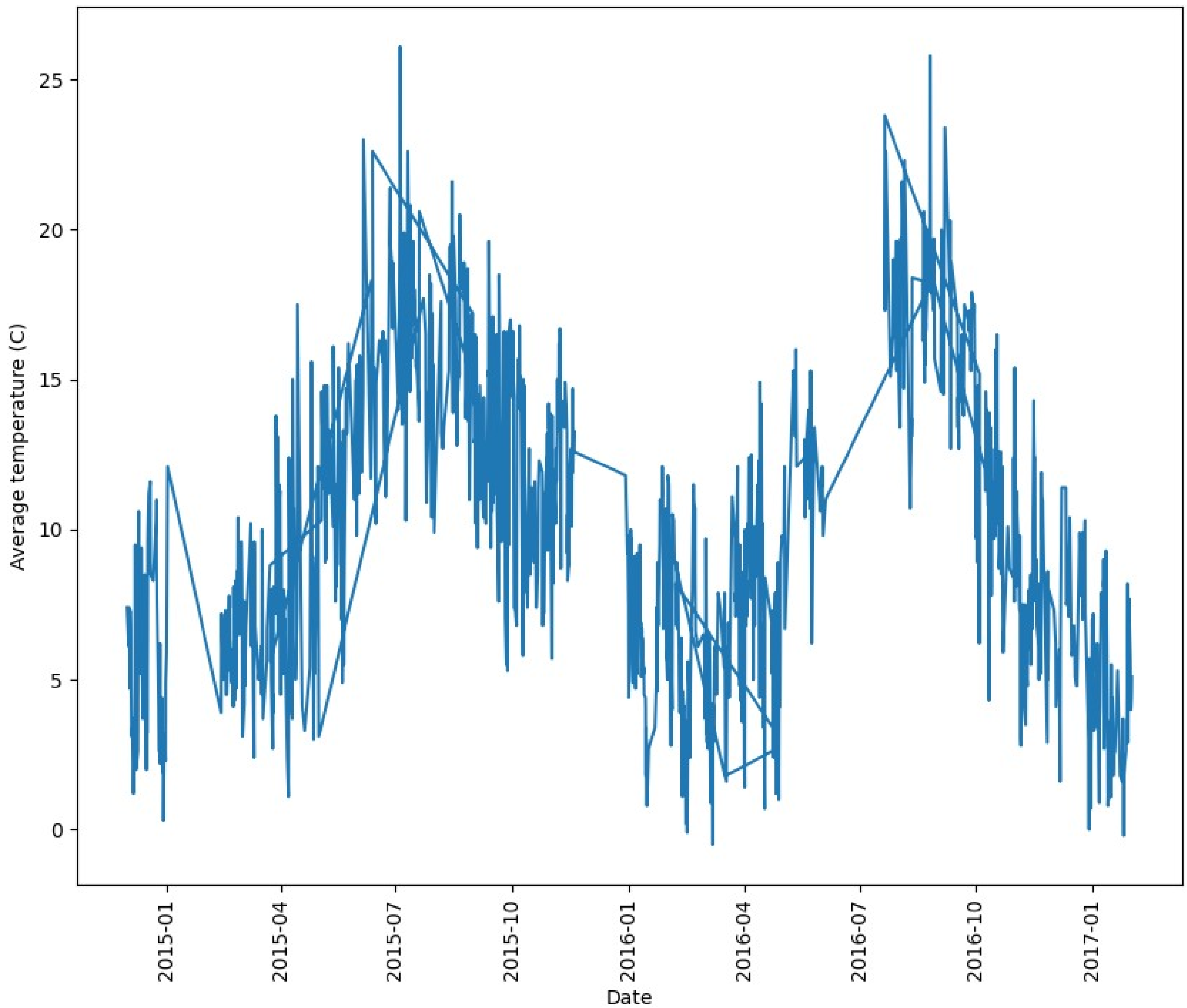
```
week_of_year         False
dtype: bool
temp                 False
dew                  False
humidity             False
precip               False
precipprob           False
preciptype           False
snow                 False
snowdepth            False
windspeed            False
winddir              False
sealevelpressure     False
cloudcover           False
solarradiation       False
uvindex              False
solarenergy          False
month                False
year                 False
week_of_year         False
dtype: bool
```

```
plt.figure(figsize=(16,8))
plt.plot(df_c.index, df_c['temp'])
plt.xlabel('Year')
plt.ylabel('temperature (C)')
```

```
Text(0, 0.5, 'temperature (C)')
```



```python
# Let's zoom in to 2014-2017
df_chunk = df_c.loc['2014-12':'2017-01']  # since the date is an index, we can use it to filter our data

plt.figure(figsize=(10, 8))
plt.plot(df_chunk.index, df_chunk['temp'])
plt.xticks(rotation=90)
plt.xlabel('Date')
_=plt.ylabel('Average temperature (C)')
```




```python
print(df_chunk.reindex(pd.date_range('2014-12', '2017-01')).isnull().all(1).sum())
df_chunk.reindex(pd.date_range('2014-12', '2017-01')).isnull().all(1)
```

```
652
```

```
2014-12-01    False
2014-12-02     True
2014-12-03    False
2014-12-04     True
2014-12-05    False
              ...
2016-12-28     True
2016-12-29     True
2016-12-30     True
2016-12-31     True
2017-01-01     True
Freq: D, Length: 763, dtype: bool
```

```
df_c[df_c.index.duplicated(keep=False)].head(20)
```


{"summary":"{\n  \"name\": \"df_c[df_c\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"temp\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.2247448713915894,\n        \"min\": 11.9,\n        \"max\": 14.3,\n
\"samples\": [\n          14.0,\n          11.9,\n          14.3\n
],\n        \"num_unique_values\": 4,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"dew\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0144785195688797,\n        \"min\":
10.3,\n        \"max\": 12.6,\n        \"samples\": [\n
12.6,\n          10.3,\n          12.2\n        ],\n
\"num_unique_values\": 4,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"humidity\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3.143897528016243,\n        \"min\":
87.17,\n        \"max\": 94.57,\n        \"samples\": [\n
91.71,\n          89.6,\n          87.17\n        ],\n
\"num_unique_values\": 4,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"precip\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3.278237788812764,\n        \"min\": 0.197,\n        \"max\":
6.905,\n        \"samples\": [\n          0.197,\n          0.686,\n
0.21\n        ],\n        \"num_unique_values\": 4,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"precipprob\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 100.0,\n        \"max\": 100.0,\n
\"samples\": [\n          100.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"preciptype\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"samples\": [\n          \"rain\"\n        ],\
n        \"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"snow\",\n      \"properties\": {\n        \"dtype\": \"number\",\n

```
\"std\": 0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n
\"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"snowdepth\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"windspeed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.883083173463291,\n        \"min\":
15.0,\n        \"max\": 34.9,\n        \"samples\": [\n          29.4\
n        ],\n        \"num_unique_values\": 4,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"winddir\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
13.32603967176045,\n        \"min\": 192.0,\n        \"max\": 223.0,\n
\"samples\": [\n          211.0\n        ],\n
\"num_unique_values\": 4,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sealevelpressure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 9.184588540956335,\n        \"min\":
981.5,\n        \"max\": 997.9,\n        \"samples\": [\n
997.3\n        ],\n        \"num_unique_values\": 4,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cloudcover\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9.40726669477732,\n        \"min\": 71.9,\n        \"max\": 93.8,\n
\"samples\": [\n          84.4\n        ],\n
\"num_unique_values\": 4,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"solarradiation\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"uvindex\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n
\"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"solarenergy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"samples\": [\n          0.0\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"month\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 10,\n        \"max\": 10,\n
\"samples\": [\n          10\n        ],\n
```

\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"year\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 5,\n        \"min\": 2013,\n        \"max\": 2023,\n
\"samples\": [\n          2023\n        ],\n
\"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"week_of_year\",\n      \"properties\": {\n        \"dtype\":
\"UInt32\",\n        \"samples\": [\n          \"43\"\n        ],\n
\"num_unique_values\": 1,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
# Let's keep the first one only - in practice this would require more
careful analysis!
df_c = df_c[~df_c.index.duplicated(keep='first')]
len(df_c)
```

```
11497
```

```
# Now we can reindex -- this is where the original error about
duplicates was
df_c = df_c.reindex(pd.date_range(df_c.index[0], df_c.index[-1]))
print(len(df_c))
```

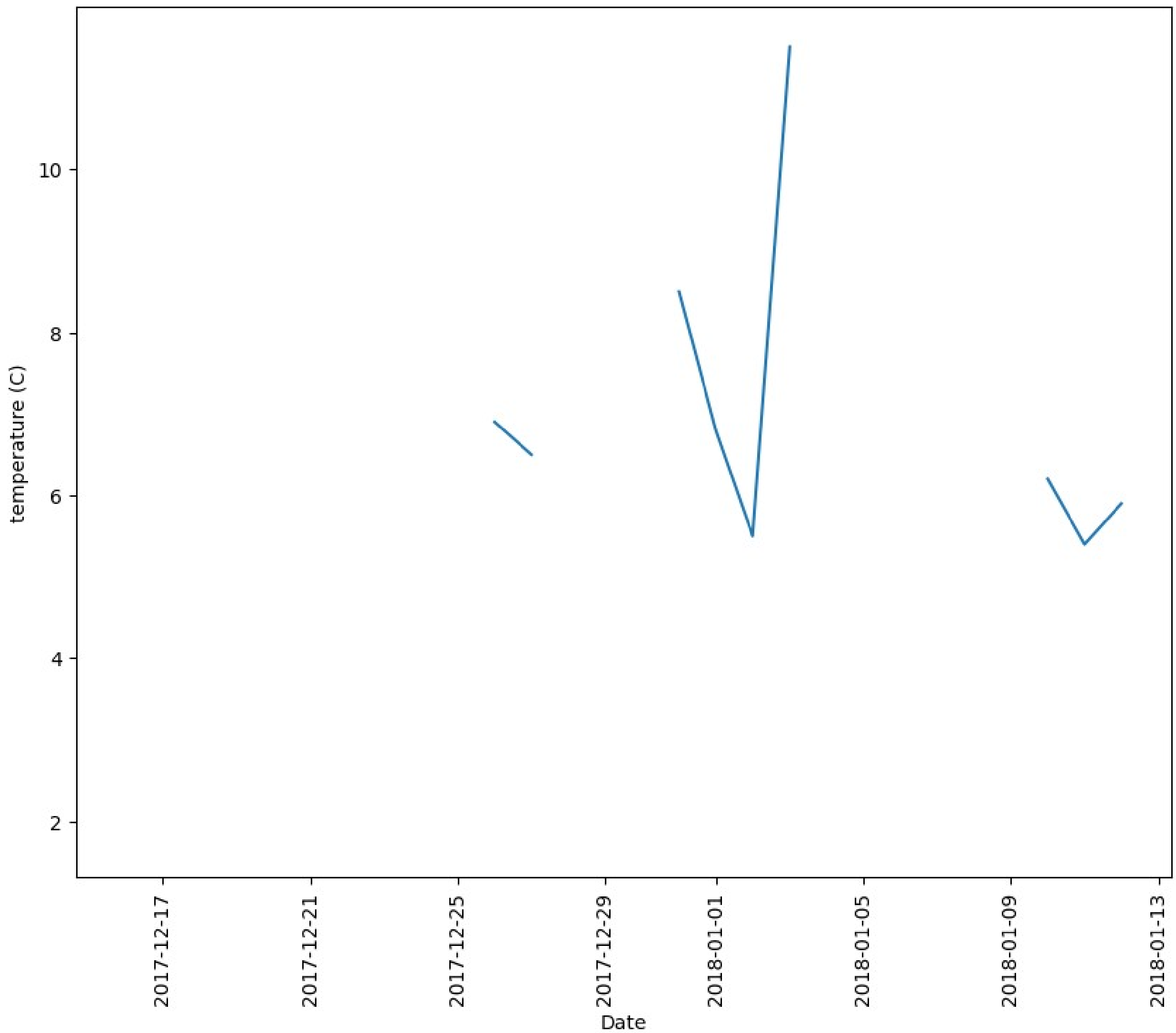
```
4996
```

```
# Now we should have missing values
print(df_c.isna().sum())
```

```
temp                4339
dew                 4339
humidity            4339
precip              4339
precipprob          4339
preciptype          4339
snow                4339
snowdepth           4339
windspeed           4339
winddir             4339
sealevelpressure    4339
cloudcover          4339
solarradiation      4339
uvindex             4339
solarenergy         4339
month               4339
year                4339
week_of_year        4339
dtype: int64
```

```
df_chunk = df_c.loc['2017-12-15':'2018-01-15']  # since the date is an
index, we can use it to filter our data
```

```python
plt.figure(figsize=(10, 8))
plt.plot(df_chunk.index, df_chunk['temp'])
plt.xticks(rotation=90)
plt.xlabel('Date')
_=plt.ylabel('temperature (C)')
# The missing values are clearly visible now!
```

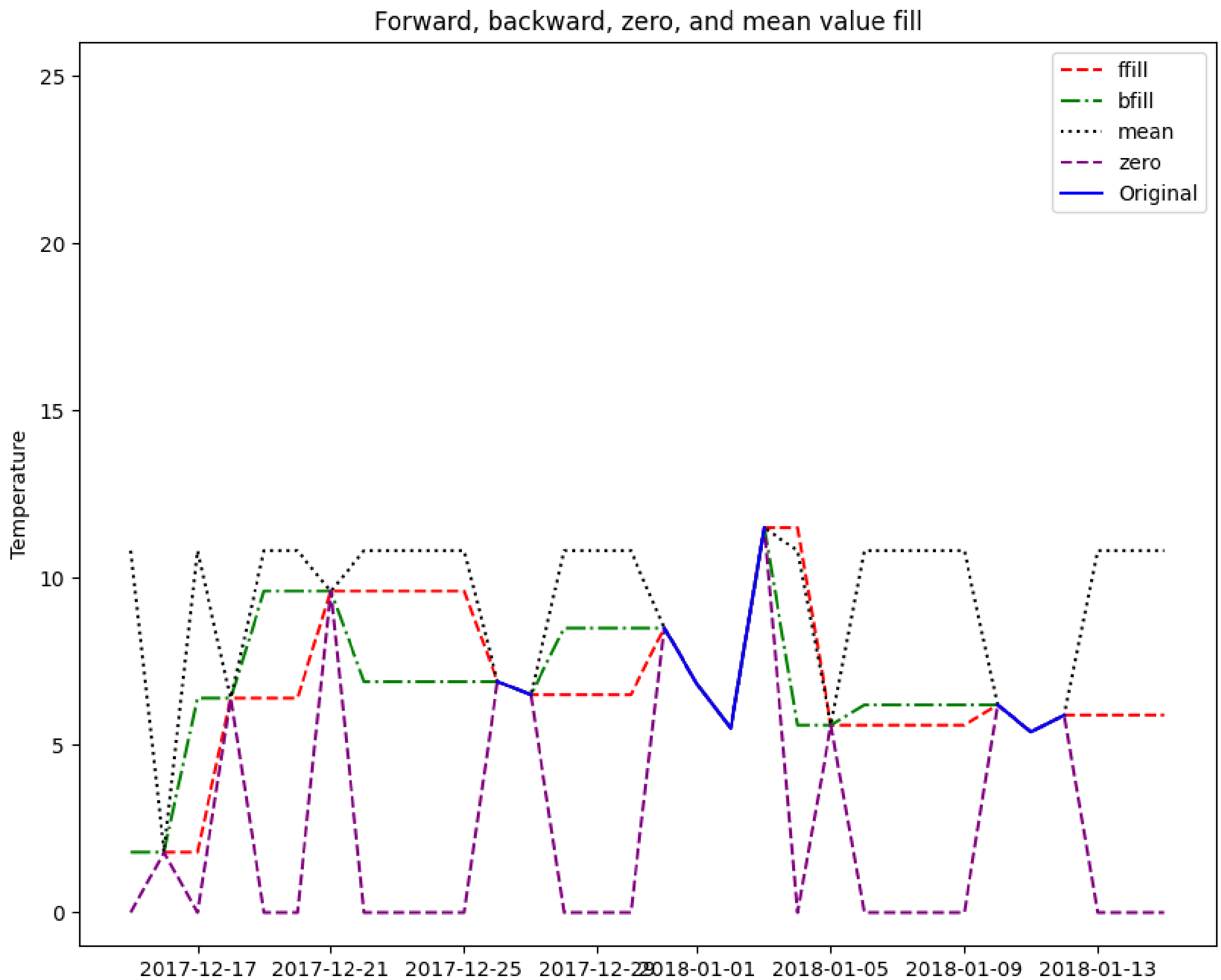



```python
df2 = df_chunk.copy()
df2 = df2.loc[:, 'temp'].to_frame()
df2
```


{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 32,\n  \"fields\": [\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2.311038172747788,\n        \"min\": 1.8,\n        \"max\": 11.5,\n        \"samples\": [\n

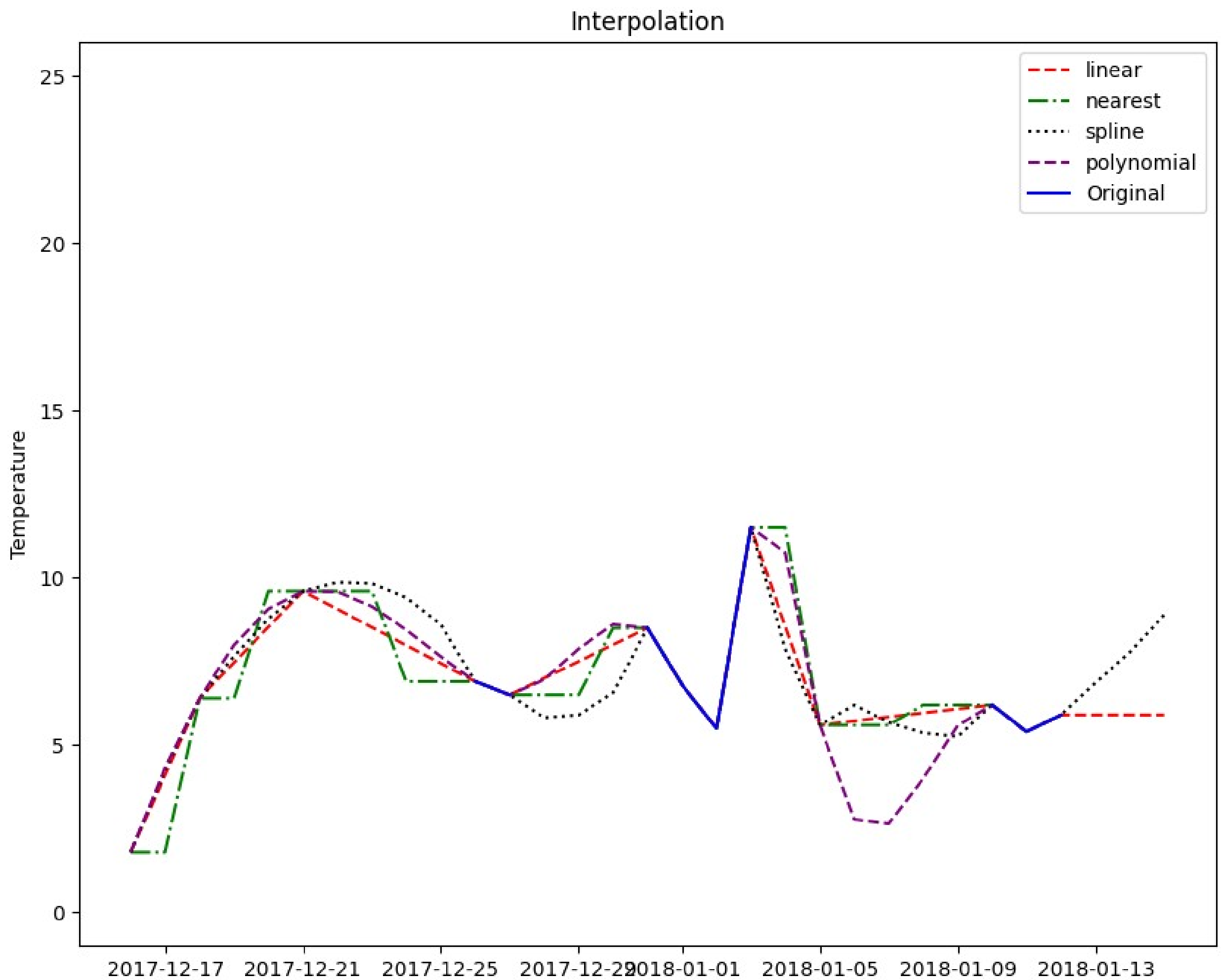
5.4,\n          5.6,\n          1.8\n        ],\n
\"num_unique_values\": 13,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df2"}


```
#Forward Fill
df2['ffill'] = df2['temp'].ffill()
# Backward Fill
df2['bfill'] = df2['temp'].bfill()
# Mean Value Fill
df2['meanfill'] = df2['temp'].fillna(df['temp'].mean())  # Note that
we're using the mean of df, not of df2
# Fill with 0s
df2['zerofill'] = df2['temp'].fillna(0)

# Plot
fig, ax = plt.subplots(figsize=(10,8))

plt.plot(df2.index, df2['ffill'], label='ffill', linestyle='--',
color='red')
plt.plot(df2.index, df2['bfill'], label='bfill', linestyle='-.',
color='green')
plt.plot(df2.index, df2['meanfill'], label='mean', linestyle=':',
color='black')
plt.plot(df2.index, df2['zerofill'], linestyle='--', color='purple',
label='zero')
plt.plot(df2.index, df2['temp'], color='blue', label='Original')
plt.legend()
plt.ylabel('Temperature')
plt.ylim(-1, 26)
_=plt.title('Forward, backward, zero, and mean value fill')
```

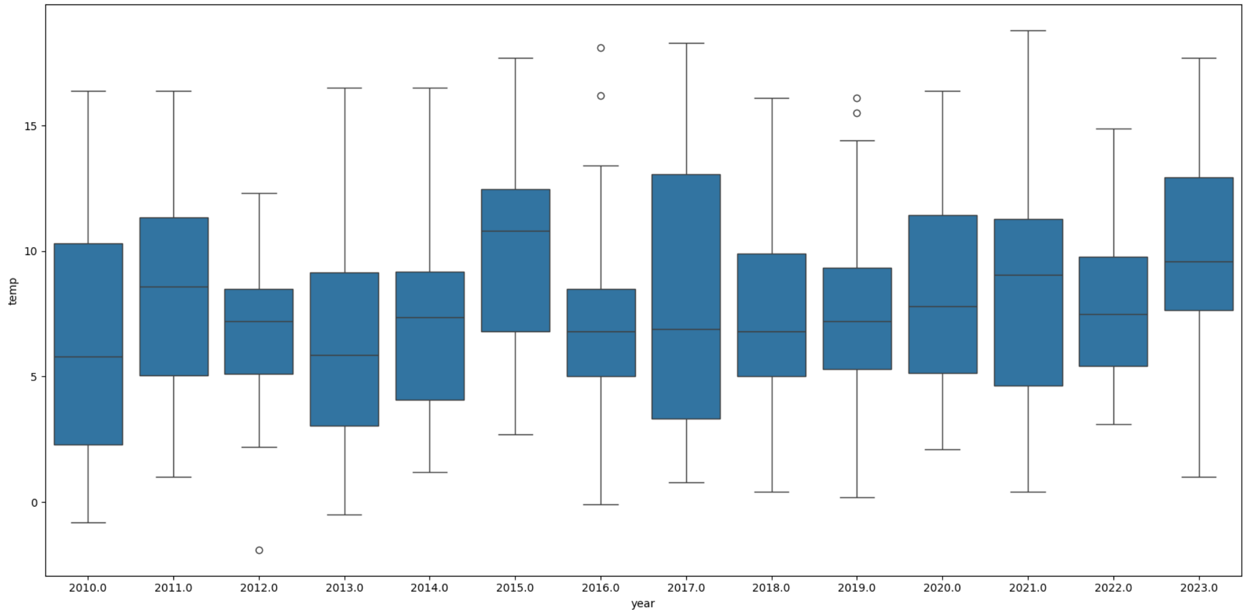



```
# Try different ways to fill the data - more advanced: interpolation

df2['linear_interp'] = df2['temp'].interpolate(method='linear')
df2['nearest_interp'] = df2['temp'].interpolate(method='nearest')
df2['spline_interp'] = df2['temp'].interpolate(method='spline',
order=2)
df2['polynomial_interp'] =
df2['temp'].interpolate(method="polynomial", order=3)

# Plot
fig, ax = plt.subplots(figsize=(10,8))

plt.plot(df2.index, df2['linear_interp'], linestyle='--', color='red',
label='linear')
plt.plot(df2.index, df2['nearest_interp'], linestyle='-.',
color='green', label='nearest')
plt.plot(df2.index, df2['spline_interp'], linestyle=':',
color='black', label='spline')
plt.plot(df2.index, df2['polynomial_interp'], linestyle='--',
```

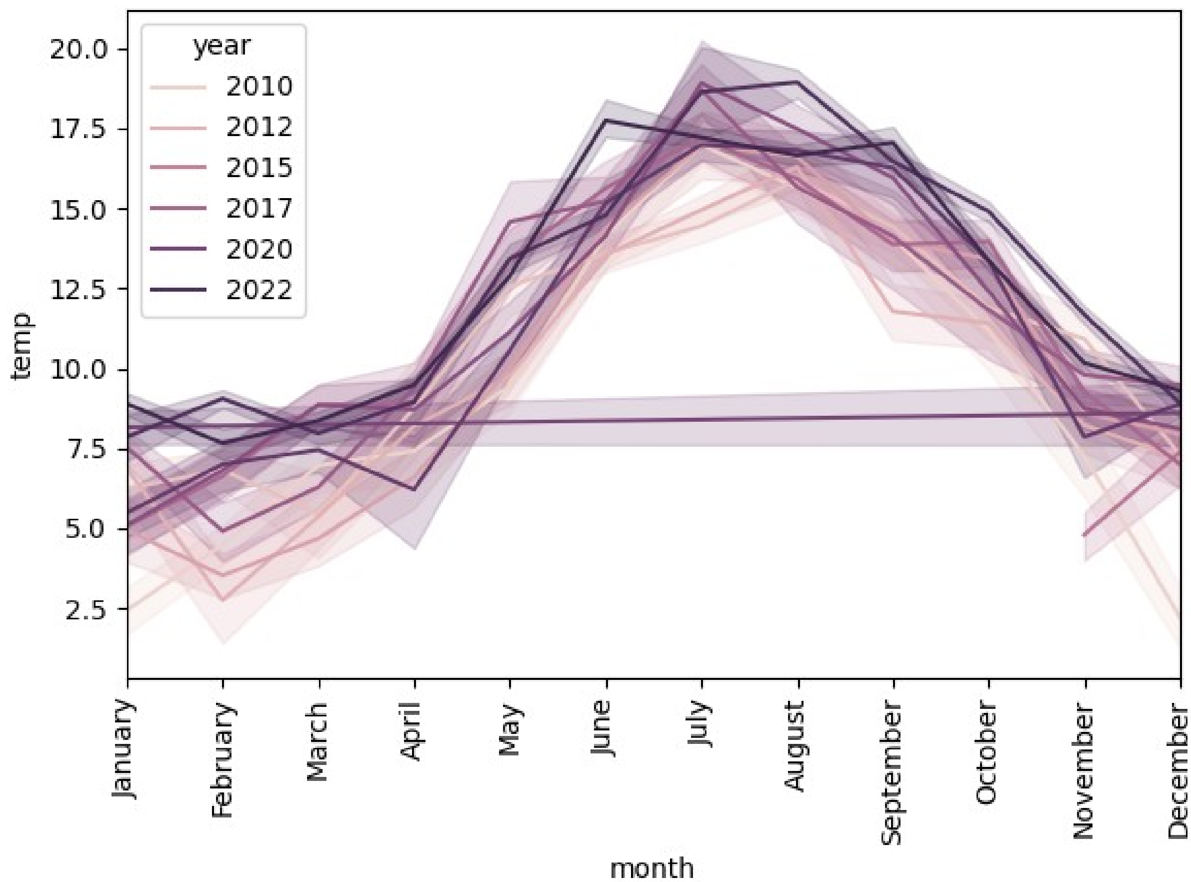
```
color='purple', label='polynomial')
plt.plot(df2.index, df2['temp'], label='Original', color='blue')

plt.legend()
plt.ylabel('Temperature')
plt.ylim(-1, 26)
_=plt.title('Interpolation')
```



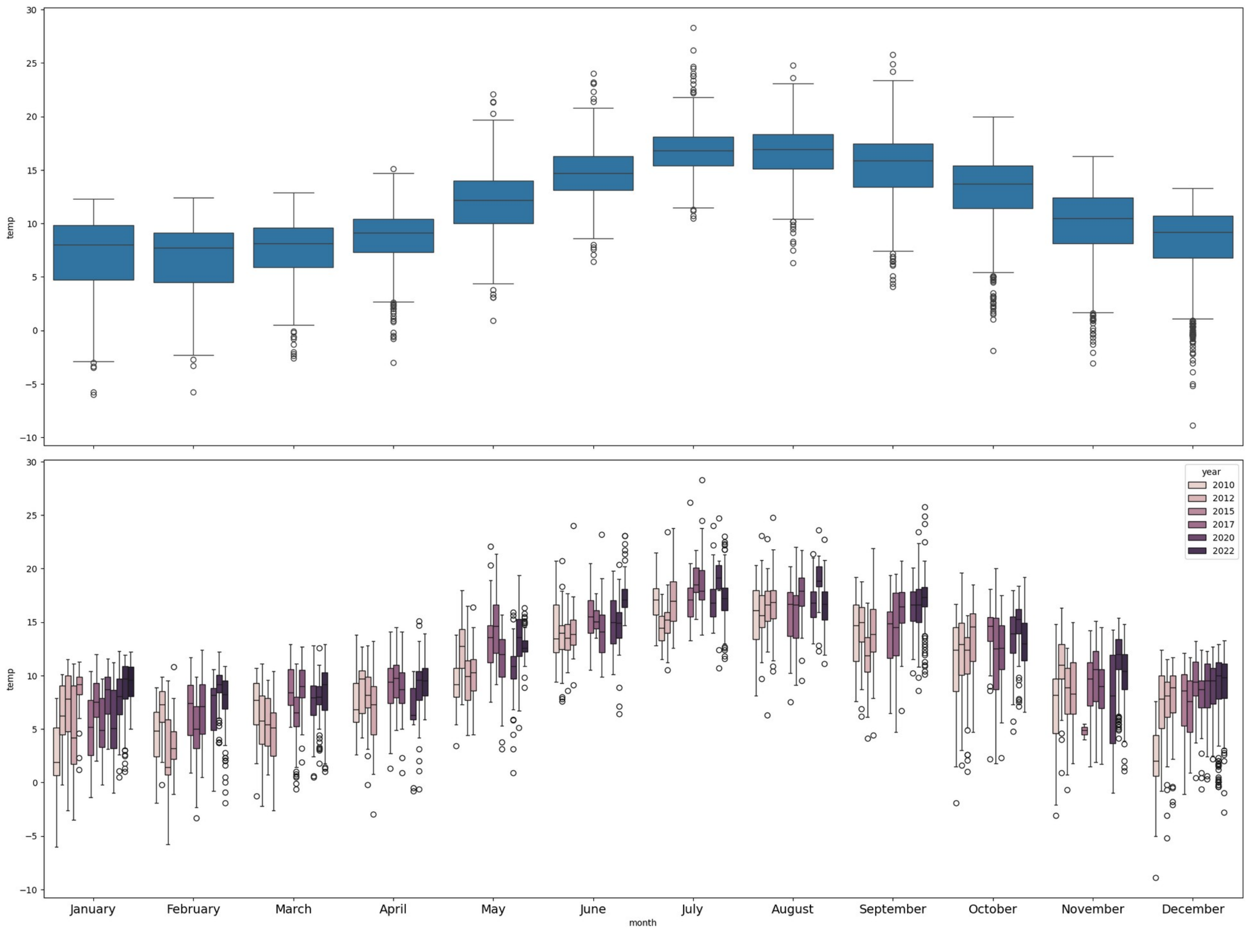

```
plt.figure(figsize=(16, 8))
_=sns.boxplot(x='year', y='temp', data=df_c)
_=plt.tight_layout()
```




```python
# Visualise trends across years
sns.lineplot(x='month', y='temp', data=df, hue='year')
_=plt.xticks(np.arange(1, 13), months_of_the_year, rotation=90)
_=plt.xlim(1, 12)  # limit x-axis
_=plt.tight_layout()
```



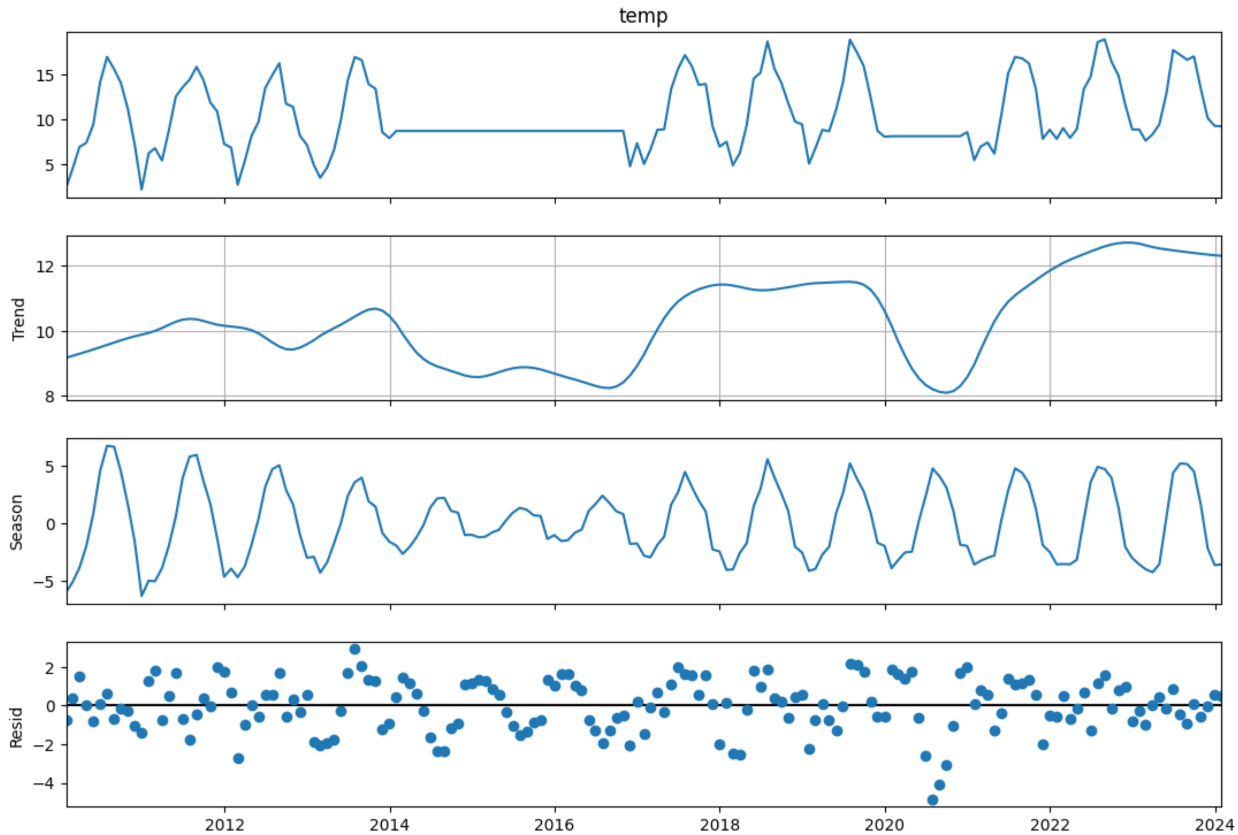

```
# Visualise trends across years
fix, ax = plt.subplots(2, 1, sharex=True, figsize=(20,15))
sns.boxplot(x='month', y='temp', data=df, ax=ax[0])  # top plot
sns.boxplot(x='month', y='temp', data=df, hue='year', ax=ax[1])  #
bottom plot
ax[1].set_xticks(np.arange(0, 12), months_of_the_year, fontsize=14)
plt.tight_layout()
```




```python
data_ds = df['temp'].resample('M').mean().ffill().to_frame()  # one value per month
data_ds
```

{"summary":"{\n  \"name\": \"data_ds\",\n  \"rows\": 169,\n  \"fields\": [\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 3.770517588760982,\n        \"min\": 2.218867924528302,\n        \"max\": 18.932758620689654,\n        \"samples\": [\n          14.439473684210528,\n          16.96818181818182,\n          4.944067796610169\n        ],\n        \"num_unique_values\": 125,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"data_ds"}

```python
# Try decomposition on the resampled dataset
from statsmodels.tsa.seasonal import seasonal_decompose, STL
decomposition = STL(data_ds['temp']).fit()
fig = decomposition.plot()
fig.set_size_inches(12,8)
fig.axes[1].grid()
```



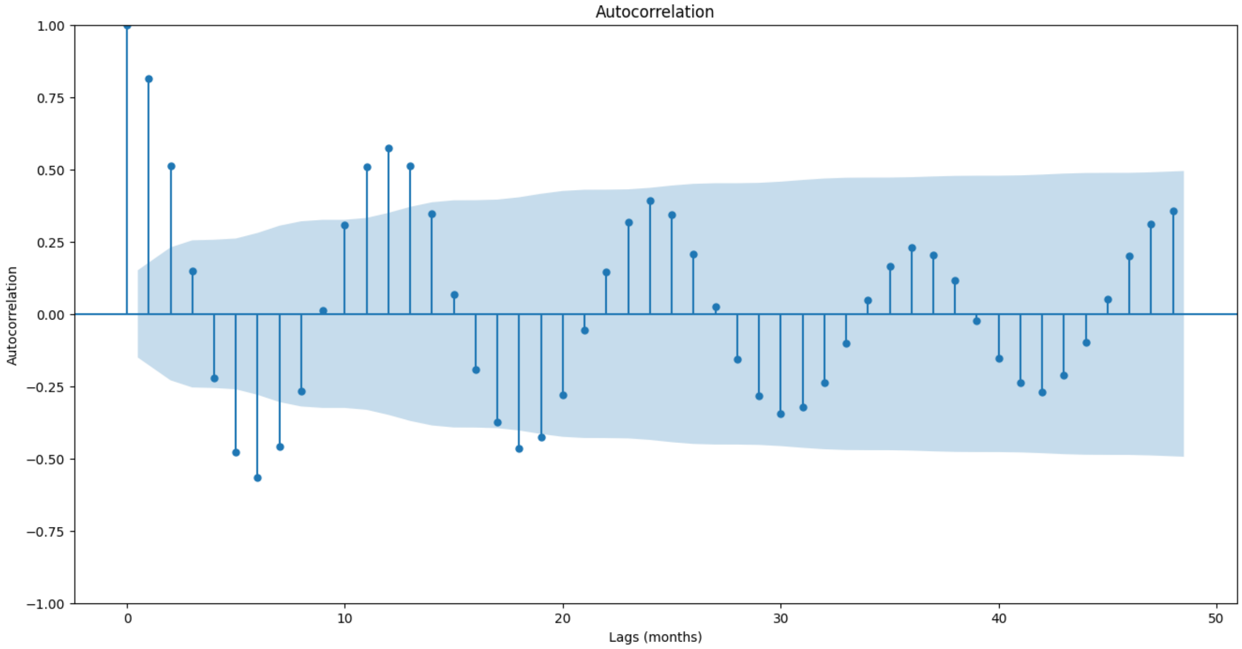

```python
# Statistical test for stationarity: Augmented Dickey-Fuller (ADF) test
adf_result = adfuller(data_ds['temp'])
print('ADF Statistic %.2f:' % adf_result[0])
print('ADF p-value: %.4f:' % adf_result[1])
# p-value << 0.05 ==> timeseries does not have a unit root
```

```
ADF Statistic -2.17:
ADF p-value: 0.2180:
```

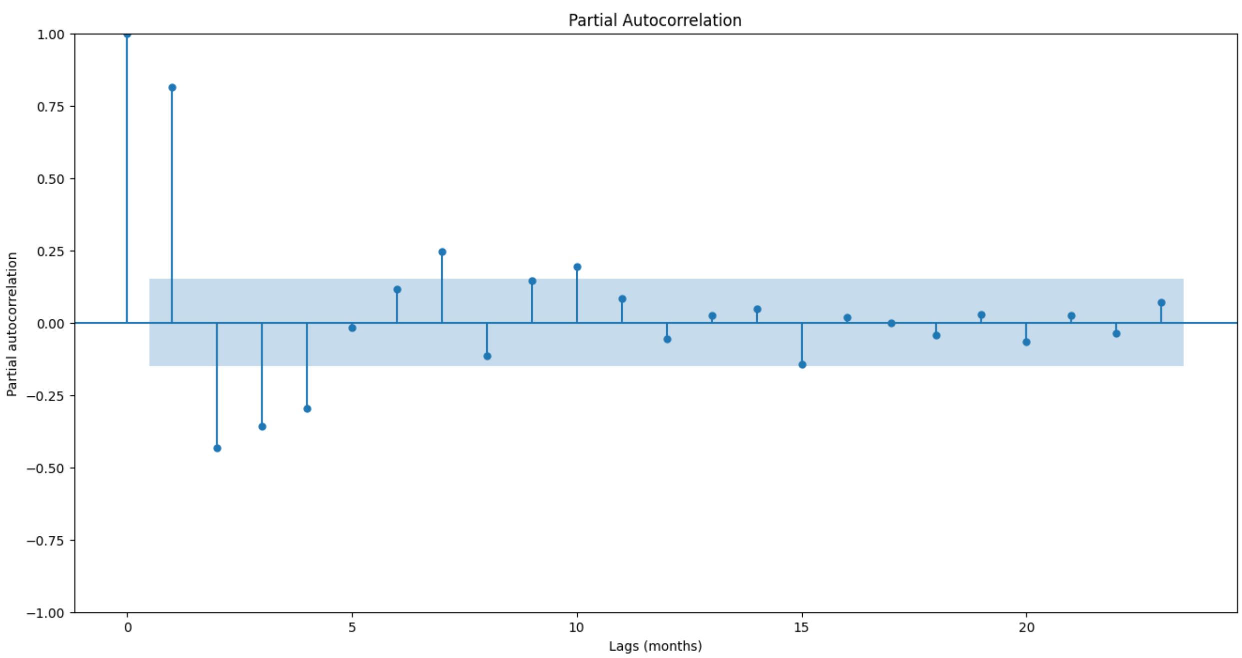
```python
adf_result = adfuller(data_ds.loc['2017':'2024', 'temp'])  # ADF test on the full years only. Is there a trend?
print('ADF Statistic %.2f:' % adf_result[0])
print('ADF p-value: %.4f:' % adf_result[1])
```

```
ADF Statistic -1.51:
ADF p-value: 0.5292:
```

```python
# Autocorrelation (can help us with modelling later)
fig, ax = plt.subplots(figsize=(16,8))
_=plot_acf(data_ds['temp'], lags=48, ax=ax)  # each lag is one month, so we're looking at 4 years worth of past data
_=plt.xlabel('Lags (months)')
_=plt.ylabel('Autocorrelation')
```

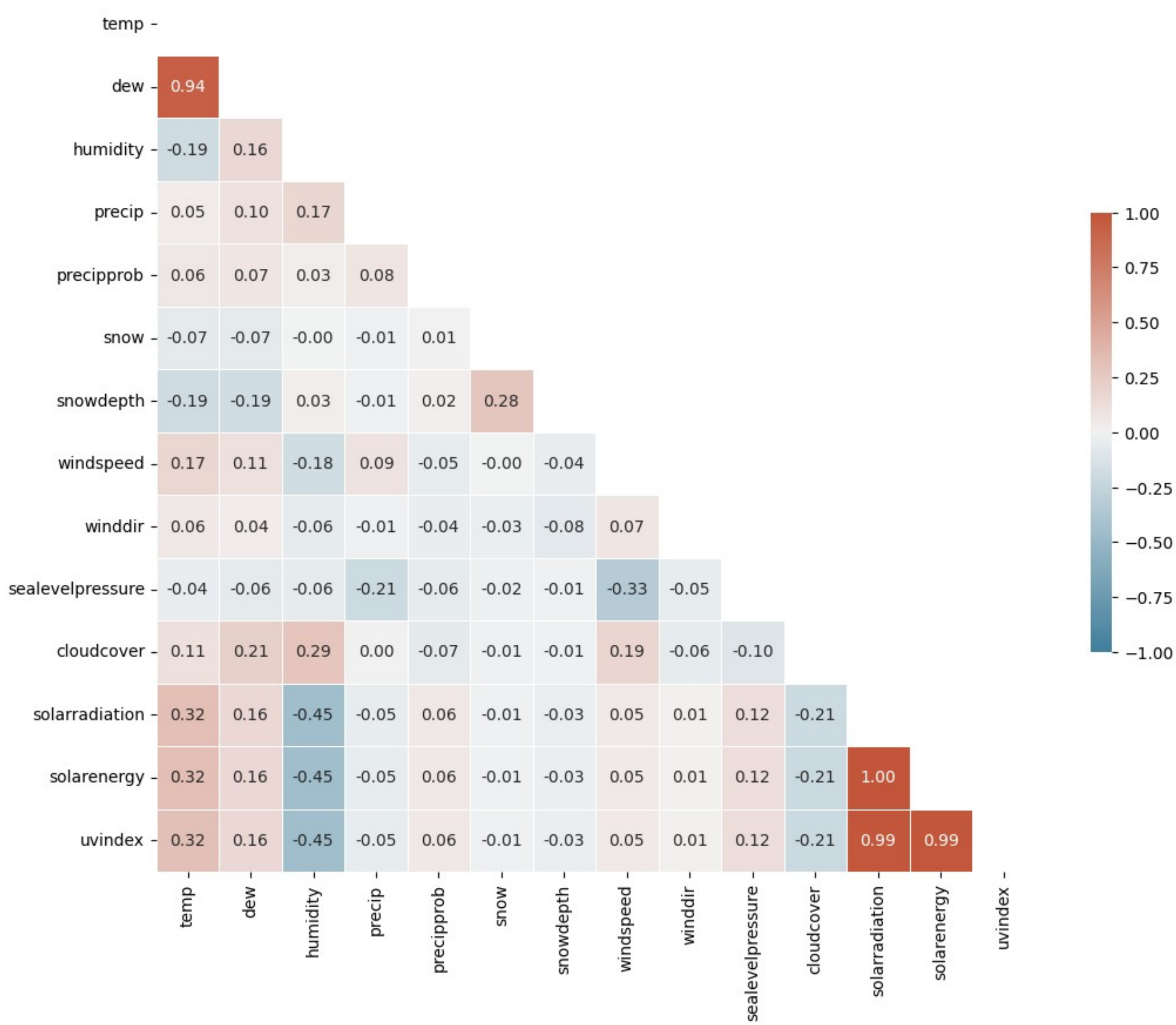



```
# Partial autocorrelation (can help us with modelling later)
fig, ax = plt.subplots(figsize=(16,8))
_=plot_pacf(data_ds['temp'], ax=ax)
_=plt.xlabel('Lags (months)')
_=plt.ylabel('Partial autocorrelation')
```

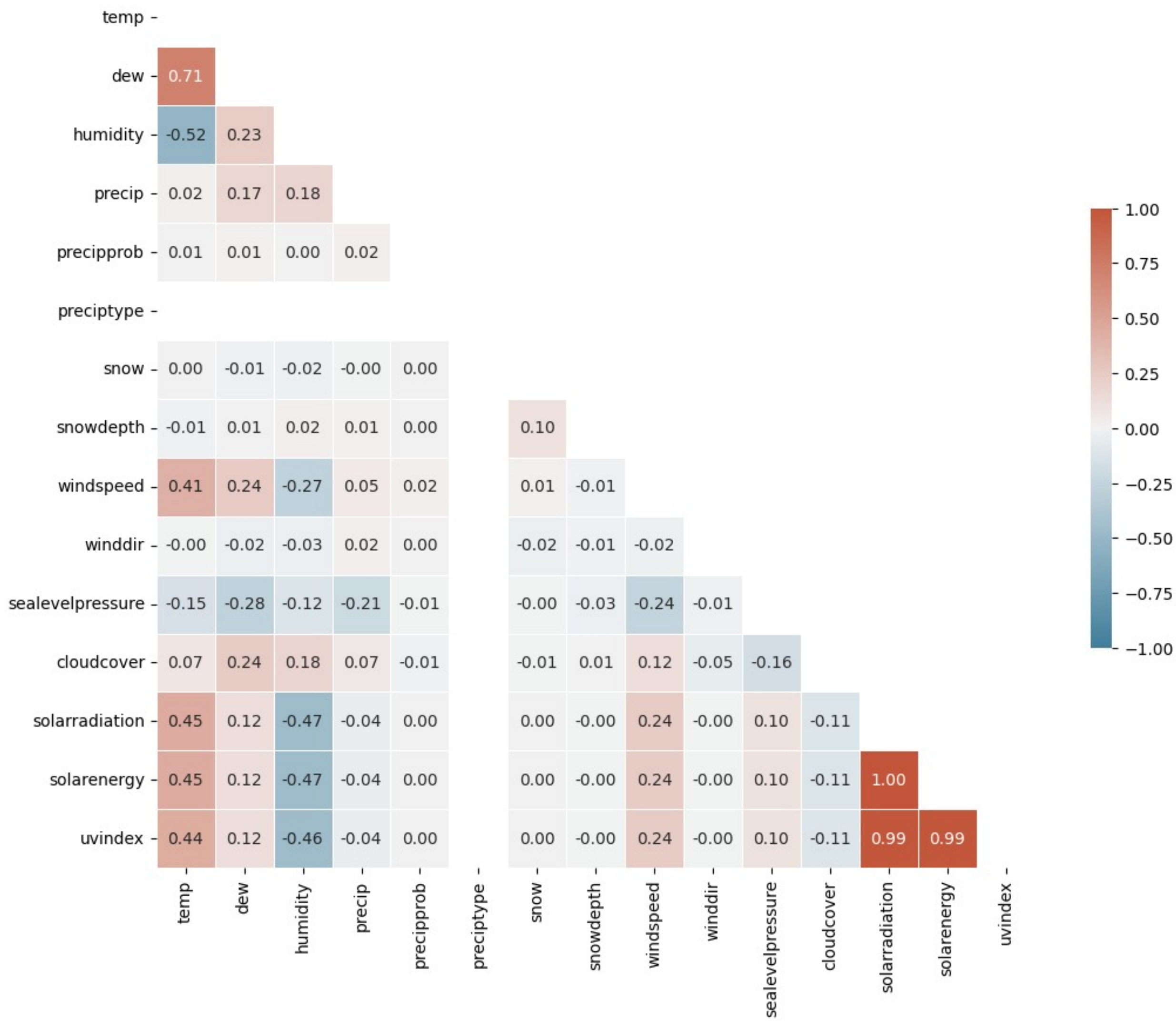



```
# https://seaborn.pydata.org/examples/many_pairwise_correlations.html

# Compute the correlation matrix
```

```python
corr = df.iloc[:, :-3].corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
fmt='.2f')
plt.tight_layout()
```



```
# Convert columns to numeric type if necessary
df_numeric = df.iloc[:, :-3].apply(pd.to_numeric, errors='coerce')

# Compute the correlation matrix
corr_diff = df_numeric.diff().corr()
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr_diff, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_diff, mask=mask, cmap=cmap, vmin=-1, vmax=1,
center=0, annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
fmt='.2f')
plt.tight_layout()
plt.show()
```



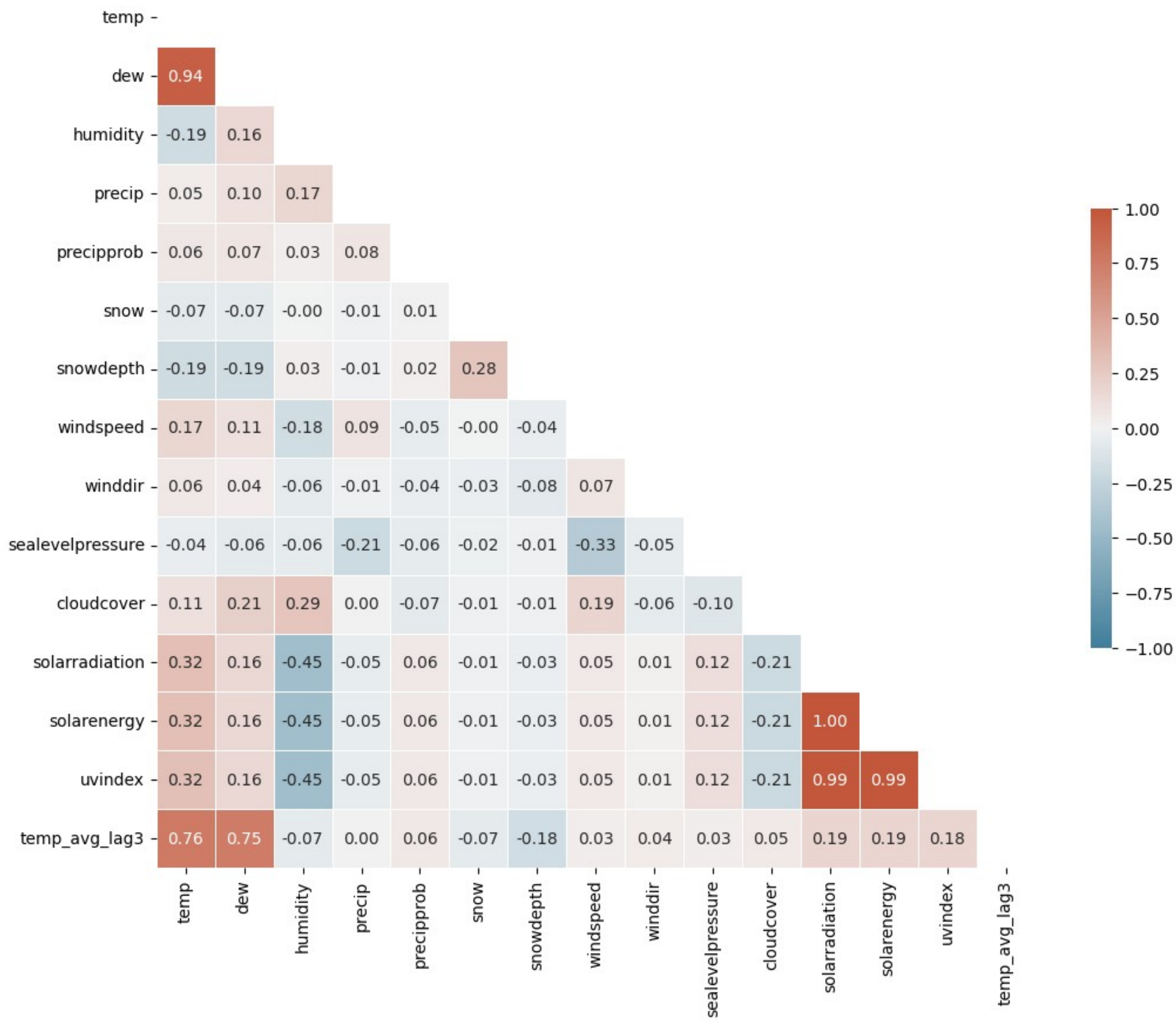

```python
df2 = df.iloc[:, :-3].copy()
df2['temp_avg_lag3'] = df2['temp'].shift(-3)
df2.head()
```

{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 8536,\n  \"fields\":
[\n    {\n      \"column\": \"temp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.640029278791179,\n
\"min\": -8.9,\n        \"max\": 28.3,\n        \"samples\": [\n
12.6,\n          13.0,\n          22.1\n        ],\n
\"num_unique_values\": 281,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"dew\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4.542494751203516,\n        \"min\": -10.2,\n        \"max\":
20.0,\n        \"samples\": [\n          4.9,\n          -6.5,\n
8.7\n        ],\n        \"num_unique_values\": 259,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"humidity\",\n      \"properties\":

{\n        \"dtype\": \"number\",\n        \"std\": 8.888867917418999,\n        \"min\": 41.89,\n        \"max\": 100.0,\n        \"samples\": [\n          72.53,\n          93.24,\n          98.62\n        ],\n        \"num_unique_values\": 2802,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"precip\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.894431542189717,\n        \"min\": 0.0,\n        \"max\": 32.385,\n        \"samples\": [\n          0.262,\n          0.074,\n          0.56\n        ],\n        \"num_unique_values\": 2216,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"precipprob\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 17.376592123895172,\n        \"min\": 0.0,\n        \"max\": 100.0,\n        \"samples\": [\n          0.0,\n          100.0\n        ],\n        \"num_unique_values\": 2,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"preciptype\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"samples\": [\n          \"rain\",\n          \"rain,snow\"\n        ],\n        \"num_unique_values\": 3,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"snow\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.0025808656410961357,\n        \"min\": 0.0,\n        \"max\": 0.13,\n        \"samples\": [\n          0.08,\n          0.13\n        ],\n        \"num_unique_values\": 5,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"snowdepth\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.3505657600610243,\n        \"min\": 0.0,\n        \"max\": 14.0,\n        \"samples\": [\n          4.91,\n          0.2\n        ],\n        \"num_unique_values\": 39,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"windspeed\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 10.354676031208625,\n        \"min\": 0.4,\n        \"max\": 72.2,\n        \"samples\": [\n          39.0,\n          12.0\n        ],\n        \"num_unique_values\": 501,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"winddir\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 89.50662453559492,\n        \"min\": 1.0,\n        \"max\": 360.0,\n        \"samples\": [\n          136.0,\n          15.0\n        ],\n        \"num_unique_values\": 361,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"sealevelpressure\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 11.43377469655269,\n        \"min\": 955.0,\n        \"max\": 1048.5,\n        \"samples\": [\n          990.2,\n          1009.2\n        ],\n        \"num_unique_values\": 669,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"cloudcover\",\n

\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
26.483150036400094,\n        \"min\": 0.0,\n        \"max\": 100.0,\n
\"samples\": [\n          36.4,\n          10.6\n        ],\n
\"num_unique_values\": 913,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"solarradiation\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 158.6456589417339,\n        \"min\":
0.0,\n        \"max\": 917.0,\n        \"samples\": [\n
248.1,\n          365.0\n        ],\n        \"num_unique_values\":
1720,\n        \"semantic_type\": \"\",\n        \"description\":
\"\"\n      }\n    },\n    {\n      \"column\": \"solarenergy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.571534271107572,\n        \"min\": 0.0,\n        \"max\": 3.3,\n
\"samples\": [\n          0.8,\n          2.3\n        ],\n
\"num_unique_values\": 34,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"uvindex\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 1.6080207673456206,\n        \"min\": 0.0,\n
\"max\": 9.0,\n        \"samples\": [\n          6.0,\n          2.0\n
],\n        \"num_unique_values\": 10,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"temp_avg_lag3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.639795491228626,\n
\"min\": -8.9,\n        \"max\": 28.3,\n        \"samples\": [\n
8.2,\n          13.4\n        ],\n        \"num_unique_values\": 281,\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df2"}


```
# Let's see what happens if we do the differential operation again.
df2 = df.iloc[:, :-3].copy()
df2['temp_avg_lag3'] = df2['temp'].shift(-3)
corr2 = df2.corr()
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr2, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr2, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
fmt='.2f')
plt.tight_layout()
```


temp
dew
humidity
precip
precipprob
snow
snowdepth
windspeed
winddir
sealevelpressure
cloudcover
solarradiation
solarenergy
uvindex
temp_avg_lag3
0.94
-0.19 0.16
0.05 0.10 0.17
0.06 0.07 0.03 0.08
-0.07 -0.07 -0.00 -0.01 0.01
-0.19 -0.19 0.03 -0.01 0.02 0.28
0.17 0.11 -0.18 0.09 -0.05 -0.00 -0.04
0.06 0.04 -0.06 -0.01 -0.04 -0.03 -0.08 0.07
-0.04 -0.06 -0.06 -0.21 -0.06 -0.02 -0.01 -0.33 -0.05
0.11 0.21 0.29 0.00 -0.07 -0.01 -0.01 0.19 -0.06 -0.10
0.32 0.16 -0.45 -0.05 0.06 -0.01 -0.03 0.05 0.01 0.12 -0.21
0.32 0.16 -0.45 -0.05 0.06 -0.01 -0.03 0.05 0.01 0.12 -0.21 1.00
0.32 0.16 -0.45 -0.05 0.06 -0.01 -0.03 0.05 0.01 0.12 -0.21 0.99 0.99
0.76 0.75 -0.07 0.00 0.06 -0.07 -0.18 0.03 0.04 0.03 0.05 0.19 0.19 0.18
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00