MOD550 | Final Project | Solar radiation

Reynel Villabona | Daniel Fylling

Data Exploration and Estimation input

We found that Norway has a website (Ref.2) where they storage the solar irradiance historical data from 2016 to the present time, we took until 22/04/2023. The data that we got it was from one station (SN50539:0), located in Bergen.

We got the data from met.no throught their FROST API

In [ ]:
```
import requests
import pandas as pd
```

Insert your own client ID here. (This client ID was given by the met.no website when we signed up with our e-mail)

In [ ]:
```
client_id = '96c403d9-7ff5-48ce-bb39-39d3ccf215f5'
```

Define endpoint and parameters (we found that solar irrandiance is the one working to do the calculations)

In [ ]:
```
endpoint = 'https://frost.met.no/observations/v0.jsonld'
parameters = {
    'sources': 'SN50539:0',
    'elements': 'mean(solar_irradiance PT1H)',
    'referencetime': '2016-02-11/2023-04-22',
}
# Issue an HTTP GET request
r = requests.get(endpoint, parameters, auth=(client_id, ''))
# Extract JSON data
json = r.json()
```

Check if the request worked, print out any errors

In [ ]:
```
if r.status_code == 200:
    data = json['data']
    print('Data retrieved from frost.met.no!')
else:
    print('Error! Returned status code %s' % r.status_code)
    print('Message: %s' % json['error']['message'])
    print('Reason: %s' % json['error']['reason'])
```

```
Data retrieved from frost.met.no!
```

Here we return a Dataframe with all the observation in a table format.

In [ ]:
```
df = pd.DataFrame()
for i in range(len(data)):
    row = pd.DataFrame(data[i]['observations'])
```

```
    row['referenceTime'] = data[i]['referenceTime']
    row['sourceId'] = data[i]['sourceId']
    df = df.append(row)

df = df.reset_index()
```

Because last line used to take around 25min - 30min to run, we decided to save the file in CSV format and then read it.

In [ ]:

```
# Save the DataFrame as a CSV file
df.to_csv('solar_irradiance_data.csv', index=False)
```

Data Exploration and forescating input

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd
from shapely.geometry import Point
from pykrige.ok import OrdinaryKriging
from sklearn.ensemble import RandomForestRegressor
```

Read the data CSV, that was previously saved

In [ ]:
```
df = pd.read_csv('solar_irradiance_data.csv', index_col=None)
df = df.drop('index', axis=1)
```

This is how initially the head of the dataframe looks like but we will not use all of the data shown.

In [ ]:
```
df.head()
```

Out[ ]:

	elementId	value	unit	timeOffset	timeResolution	timeSeriesId	performanceCategory	
0	mean(solar_irradiance PT1H)	0.737	W/m2	PT0H	PT1H	0	C	
1	mean(solar_irradiance PT1H)	0.301	W/m2	PT0H	PT1H	0	C	
2	mean(solar_irradiance PT1H)	37.689	W/m2	PT0H	PT1H	0	C	
3	mean(solar_irradiance PT1H)	91.967	W/m2	PT0H	PT1H	0	C	
4	mean(solar_irradiance PT1H)	15.464	W/m2	PT0H	PT1H	0	C	

We eliminated some columns and kept the important ones, which form a new table called "df2"

In [ ]:
```
# These additional columns will be kept
columns = ['sourceId', 'referenceTime',
           'elementId', 'value', 'unit', 'timeOffset']
df2 = df[columns].copy()
# Convert the time value to something Python understands
df2['referenceTime'] = pd.to_datetime(df2['referenceTime'])
```

Now the table looks like this.

In [ ]:
```
# Preview the result
df2
```

Out[ ]:

	sourceId	referenceTime	elementId	value	unit	timeOffset
0	SN50539:0	2016-02-11 06:00:00+00:00	mean(solar_irradiance PT1H)	0.737	W/m2	PT0H
1	SN50539:0	2017-02-11 06:00:00+00:00	mean(solar_irradiance PT1H)	0.301	W/m2	PT0H
2	SN50539:0	2017-08-05 08:00:00+00:00	mean(solar_irradiance PT1H)	37.689	W/m2	PT0H
3	SN50539:0	2017-08-05 09:00:00+00:00	mean(solar_irradiance PT1H)	91.967	W/m2	PT0H
4	SN50539:0	2017-08-05 10:00:00+00:00	mean(solar_irradiance PT1H)	15.464	W/m2	PT0H
...	...	...	...	...	...	...
49012	SN50539:0	2023-04-21 19:00:00+00:00	mean(solar_irradiance PT1H)	325.800	W/m2	PT0H
49013	SN50539:0	2023-04-21 20:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
49014	SN50539:0	2023-04-21 21:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
49015	SN50539:0	2023-04-21 22:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
49016	SN50539:0	2023-04-21 23:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H

49017 rows × 6 columns

We print the type of data that we will handling

In [ ]:
```
print(df2["sourceId"].dtype)
print(df2["referenceTime"].dtype)
print(df2["elementId"].dtype)
print(df2["value"].dtype)
print(df2["unit"].dtype)
```

```
object
datetime64[ns, UTC]
object
float64
object
```

Data Exploration

We group the column by day and take the mean of the irradiance solar daily and We remove Nan values from the value column. We create a new table called "df_daily" to storage this information.

In [ ]:
```
df_daily = df2.groupby(pd.Grouper(key='referenceTime', freq='D'))[
    'value'].mean().reset_index()  # group by day and take mean
df_daily = df_daily.dropna(subset=['value']).reset_index(drop=True)
df_daily
```

Out[ ]:

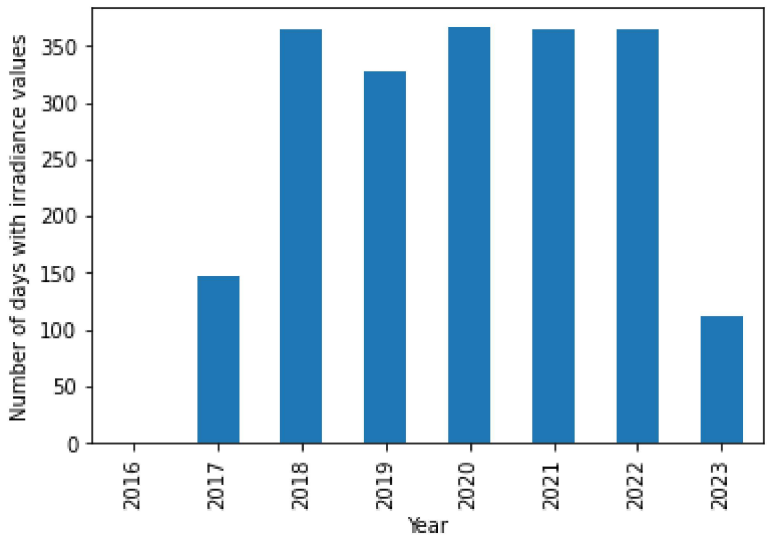
	referenceTime	value
0	2016-02-11 00:00:00+00:00	0.737000
1	2017-02-11 00:00:00+00:00	0.301000
2	2017-08-05 00:00:00+00:00	8.530563
3	2017-08-06 00:00:00+00:00	-0.196458

	referenceTime	value
4	2017-08-07 00:00:00+00:00	59.771125
...	...	...
2043	2023-04-17 00:00:00+00:00	389.614292
2044	2023-04-18 00:00:00+00:00	397.421167
2045	2023-04-19 00:00:00+00:00	432.240833
2046	2023-04-20 00:00:00+00:00	443.350708
2047	2023-04-21 00:00:00+00:00	448.077125

2048 rows × 2 columns

We will see how many samples we have per year, so if we have every day a sample, we should see something close to 365 samples.

In [ ]:
```
# assuming df2 is the DataFrame with referenceTime and value columns
# create a new column with the year
df_daily['year'] = pd.DatetimeIndex(df_daily['referenceTime']).year

# group by year and count the number of unique days
counts = df_daily.groupby('year')['referenceTime'].nunique()

# plot the results
counts.plot(kind='bar')
plt.xlabel('Year')
plt.ylabel('Number of days with irradiance values')
plt.show()
```



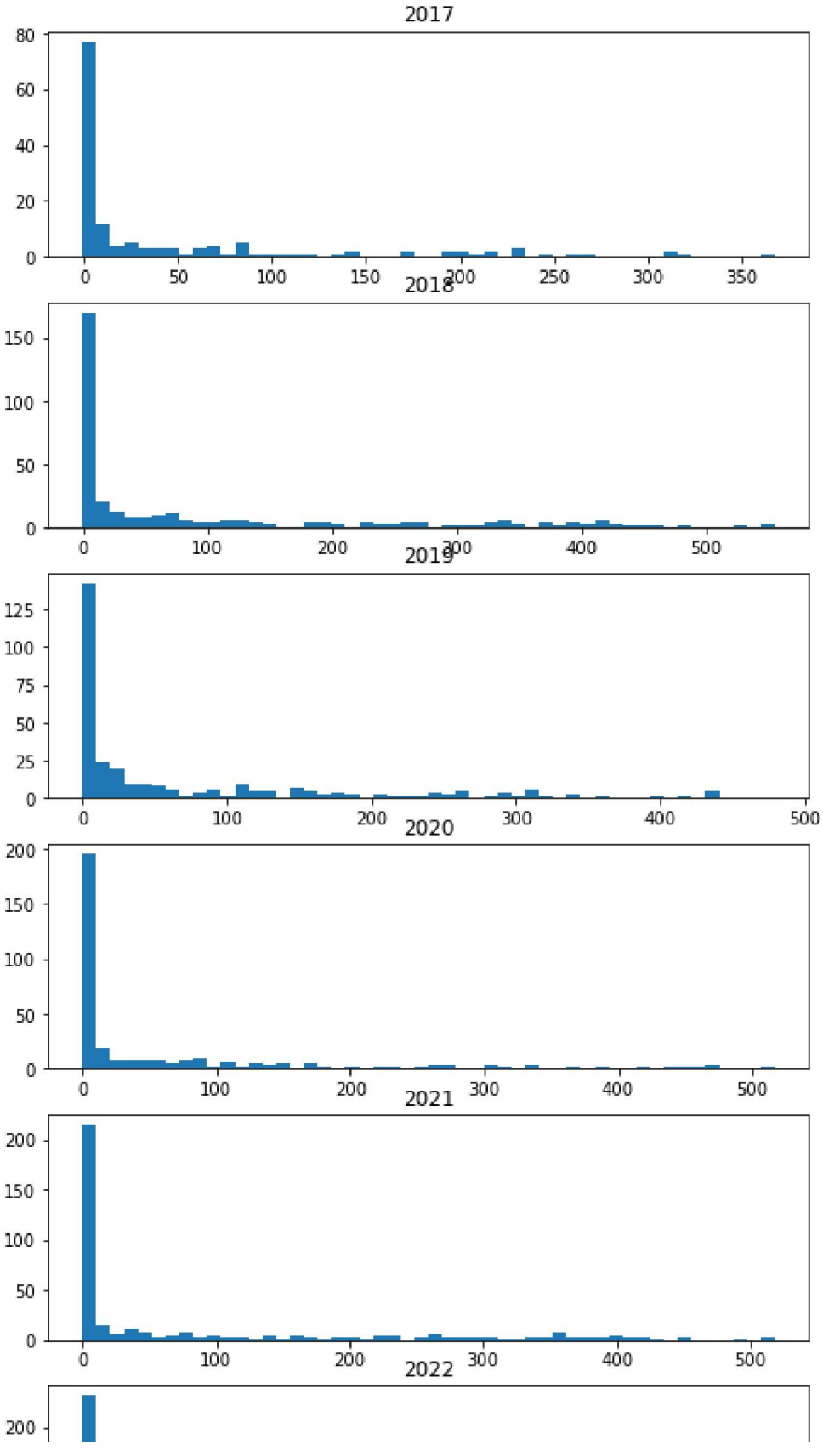
From the last graph we can see that 2016 is not representative, so we will discard it from now on.

Now we want to see in each year how is the distribution of the samples taken.

In [ ]:
```
# Create subplots with 7 rows and 1 column
fig, axes = plt.subplots(nrows=7, ncols=1, figsize=(8, 20))
```

```python
# Iterate over each subplot and plot the histogram for each year
for i, ax in enumerate(axes):
    year = 2017 + i
    df_year = df_daily[df_daily['referenceTime'].dt.year == year]
    ax.hist(df_year['value'], bins=50)
    ax.set_title(f'{year}')

# Add axis labels and a title to the plot
fig.suptitle('Solar Irradiance Distribution by Year (2017-2023)')
plt.xlabel('Solar Irradiance (W/m^2)')
plt.ylabel('Count')
plt.show()
```

Solar Irradiance Distribution by Year (2017-2023)



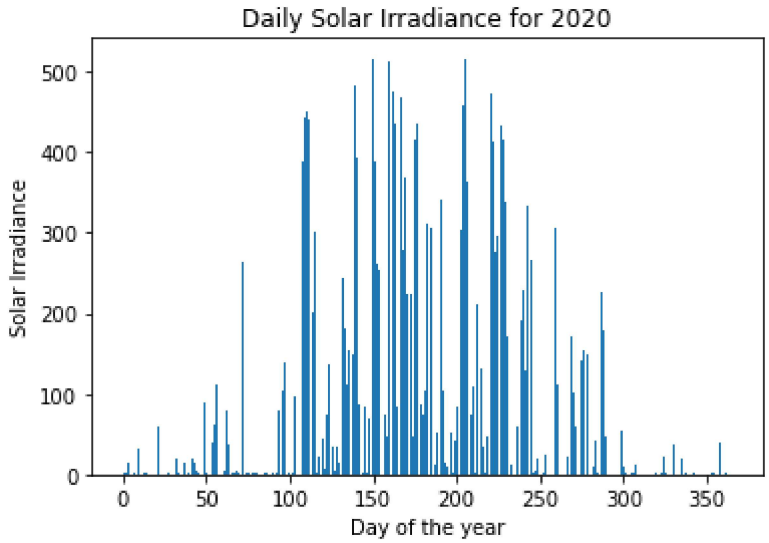
The above graphs made us think that there are too many values close to 0. That is a bit weird so we will see closer to an specific day and year. We choose 2020 to look closer.

In [ ]:
```
# Filter the year 2020
df_2020 = df_daily[df_daily['year'] == 2020]

# Create bins for the histogram
bins = [0, 100, 200, 300]

# Count the number of values in each bin
hist, edges = np.histogram(df_2020['value'], bins=bins)

# Print the histogram counts
print("Number of values between 0 and 100:", hist[0])
print("Number of values between 100 and 200:", hist[1])
print("Number of values between 200 and 300:", hist[2])
```

```
Number of values between 0 and 100: 273
Number of values between 100 and 200: 33
Number of values between 200 and 300: 21
```

Almost most of the days of the year are between 0 and 100 W/m2. We will see closer, through plotting every samples for 2020 in a bar plot.

In [ ]:
```
# Filter the DataFrame to keep only the rows for the year 2018
df_2020 = df_daily[df_daily['year'] == 2020]

# Create a new DataFrame with date as index and daily values as column
df_2020_daily = df_2020.set_index('referenceTime')[
    'value'].resample('D').mean().reset_index()

# Create a bar plot with 365 days on the x-axis and daily values on the y-axis
plt.bar(df_2020_daily.index, df_2020_daily['value'])

# Customize the plot with axis labels and title
plt.xlabel('Day of the year')
plt.ylabel('Solar Irradiance')
plt.title('Daily Solar Irradiance for 2020')
plt.show()
```



Above we can see that even in march there were so many values close to "0 W/m2". We filter "df2" table to see how were the measurements made on "2020-03-29".

In [ ]:
```
# Filter the dataframe to the desired date
desired_date = pd.to_datetime('2020-03-29').date()
filtered_df = df2[df2['referenceTime'].dt.date == desired_date]
filtered_df
```

Out[ ]:

	sourceId	referenceTime	elementId	value	unit	timeOffset
22183	SN50539:0	2020-03-29 00:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22184	SN50539:0	2020-03-29 01:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22185	SN50539:0	2020-03-29 02:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22186	SN50539:0	2020-03-29 03:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22187	SN50539:0	2020-03-29 04:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22188	SN50539:0	2020-03-29 05:00:00+00:00	mean(solar_irradiance PT1H)	1.072	W/m2	PT0H
22189	SN50539:0	2020-03-29 06:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22190	SN50539:0	2020-03-29 07:00:00+00:00	mean(solar_irradiance PT1H)	0.809	W/m2	PT0H
22191	SN50539:0	2020-03-29 08:00:00+00:00	mean(solar_irradiance PT1H)	0.579	W/m2	PT0H
22192	SN50539:0	2020-03-29 09:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22193	SN50539:0	2020-03-29 10:00:00+00:00	mean(solar_irradiance PT1H)	0.527	W/m2	PT0H
22194	SN50539:0	2020-03-29 11:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22195	SN50539:0	2020-03-29 12:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22196	SN50539:0	2020-03-29 13:00:00+00:00	mean(solar_irradiance PT1H)	4.240	W/m2	PT0H
22197	SN50539:0	2020-03-29 14:00:00+00:00	mean(solar_irradiance PT1H)	0.398	W/m2	PT0H
22198	SN50539:0	2020-03-29 15:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22199	SN50539:0	2020-03-29 16:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22200	SN50539:0	2020-03-29 17:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H

	sourceId	referenceTime	elementId	value	unit	timeOffset
22201	SN50539:0	2020-03-29 18:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22202	SN50539:0	2020-03-29 19:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22203	SN50539:0	2020-03-29 20:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22204	SN50539:0	2020-03-29 21:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22205	SN50539:0	2020-03-29 22:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H
22206	SN50539:0	2020-03-29 23:00:00+00:00	mean(solar_irradiance PT1H)	0.000	W/m2	PT0H

We identified that there are some weirds lectures of solar irradiation. Most of the lectures are "0" even in the afternoon which may make us think that something might be wrong on the lecture of the data. But we decided to move forward.

We can see below the average per year, we can see that on 2018 had the highest average.

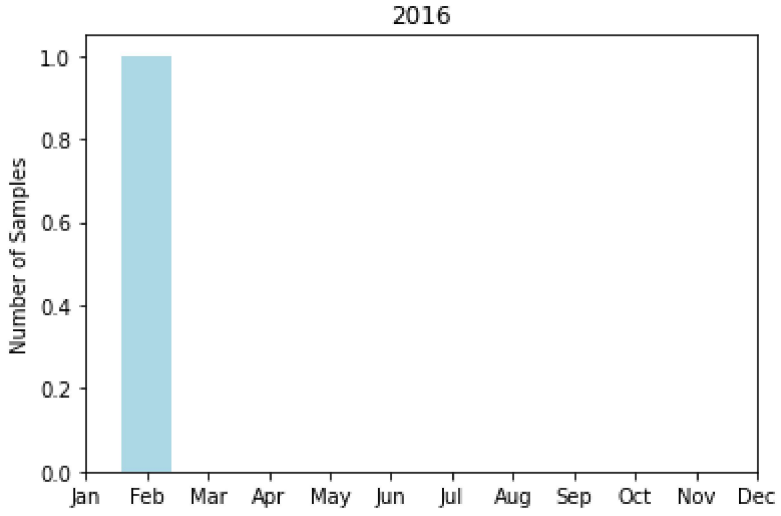
In [ ]:
```
yearly_avg = df_daily.groupby(df_daily["referenceTime"].dt.year)["value"].mean()
yearly_avg
```

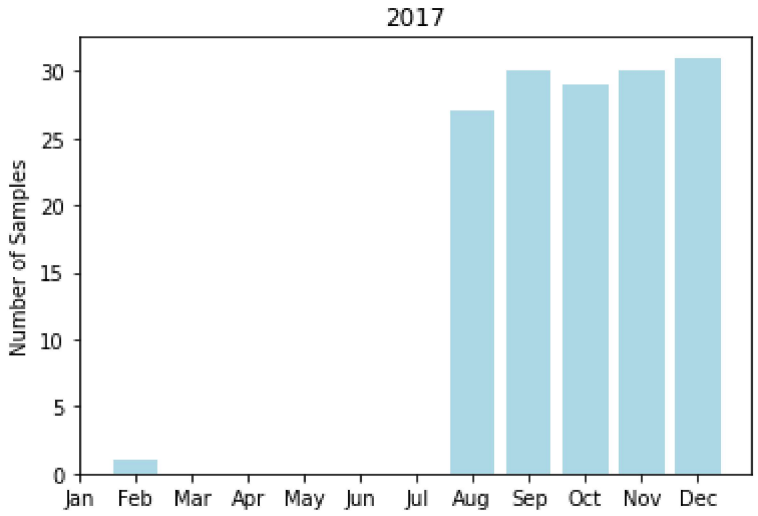
Out[ ]:
```
referenceTime
2016      0.737000
2017     48.665130
2018    102.817962
2019     80.811857
2020     79.947009
2021     77.014838
2022     58.094770
2023     38.994755
Name: value, dtype: float64
```

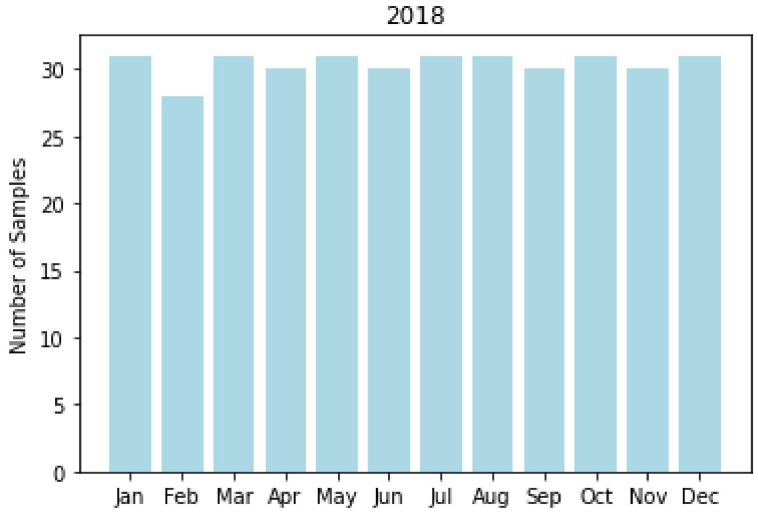
Here We want to see how many samples were taken per month, so if every day during a month samples were taken we should see values close to 30 - 31 samples.

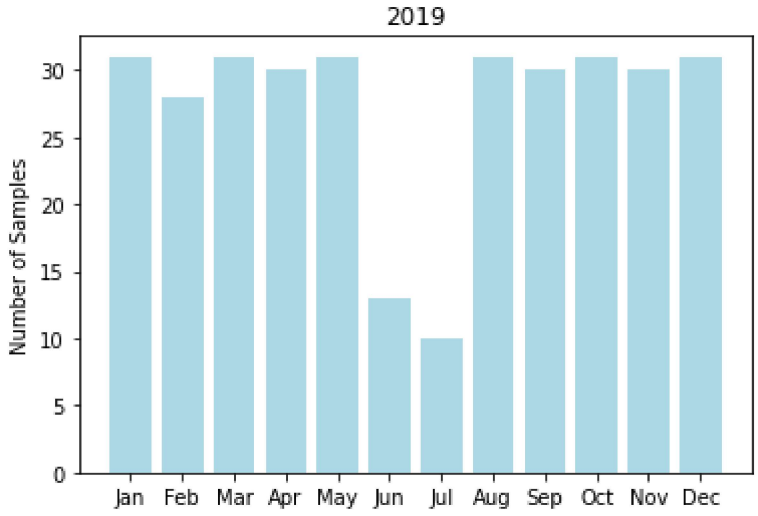
In [ ]:
```
# Group by year and month
grouped = df_daily.groupby([df_daily['referenceTime'].dt.year,
                            df_daily['referenceTime'].dt.month])
# Get the count of samples for each group
counts = grouped.size()

# Loop over each year
for year in range(2016, 2024):
    # Create a new figure for each year
    plt.figure()
    plt.title(str(year))

    # Loop over each month
    for month in range(1, 13):
        # Check if there are any samples for this month and year
        if (year, month) in counts.index:
            # Get the count of samples for this month and year
            count = counts.loc[(year, month)]
            # Plot the count as a bar
            plt.bar(month, count, color="lightblue")

    # Set the x-axis label to show the month names
```

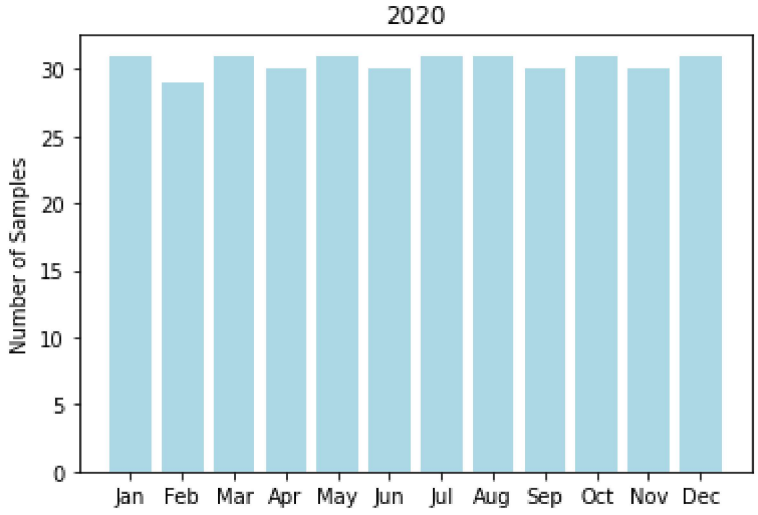
```
    plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr',
               'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

    # Set the y-axis label
    plt.ylabel('Number of Samples')

    # Save the figure
    plt.savefig(f'sample_count_{year}.png')
```

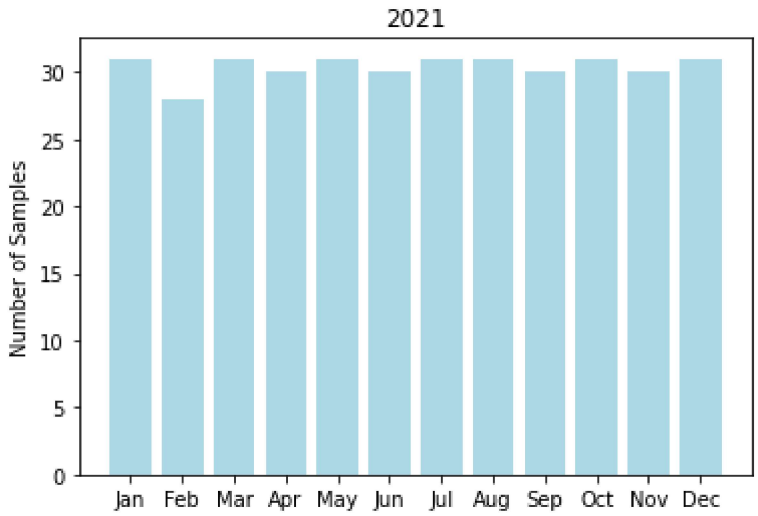


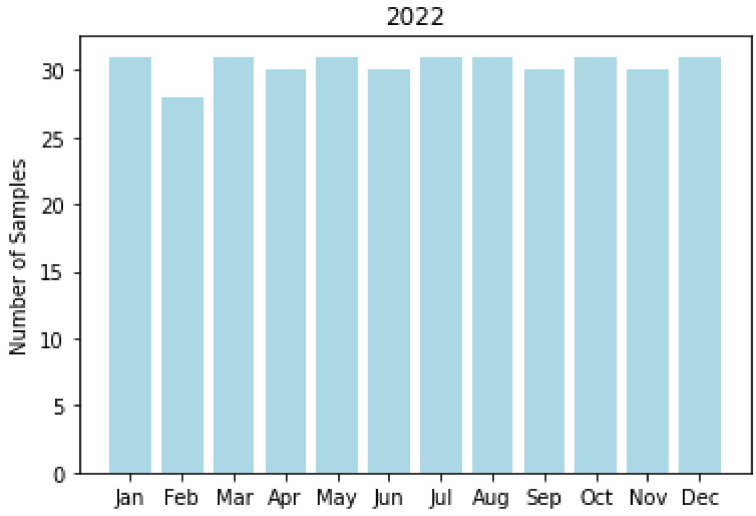


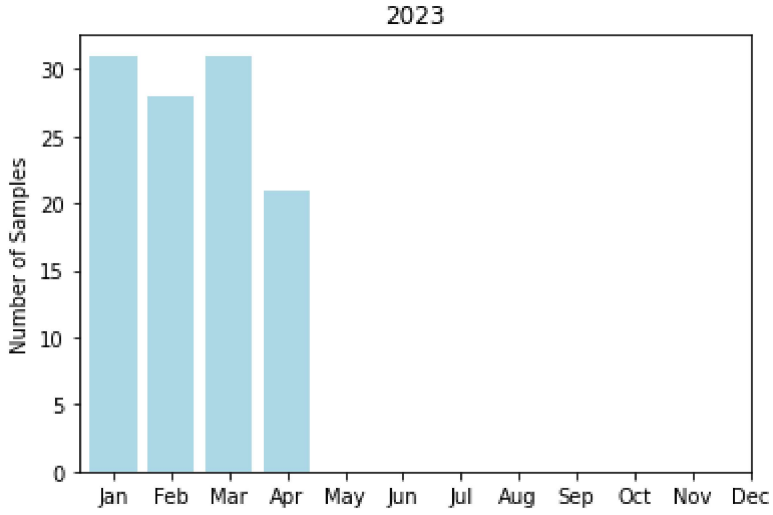



2019
Number of Samples
0
5
10
15
20
25
30
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec


2020
Number of Samples
0
5
10
15
20
25
30
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec


2021
Number of Samples
0
5
10
15
20
25
30
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec





We have decided to work only with the years where the data is completed (2018,2020,2021,2022). We keep 2023 data to compare estimations later on.

Here we did an average per month each year, since before we had samples average per day. We created another table called "df_monthly" to storage this information.

In [ ]:

```
df_monthly = df2.groupby(pd.Grouper(key='referenceTime', freq='M'))[
    'value'].mean().reset_index()  # group by day and take mean
df_monthly = df_monthly.dropna(subset=['value']).reset_index(drop=True)

# extract the year from referenceTime
df_monthly['year'] = df_monthly['referenceTime'].dt.year

# filter the DataFrame based on the desired years
years = [2018, 2020, 2021, 2022, 2023]
df_monthly = df_monthly[df_monthly['year'].isin(years)]

# drop the 'year' column
df_monthly.drop('year', axis=1, inplace=True)

df_monthly = df_monthly.reset_index(drop=True)

df_monthly
```

Out[ ]:

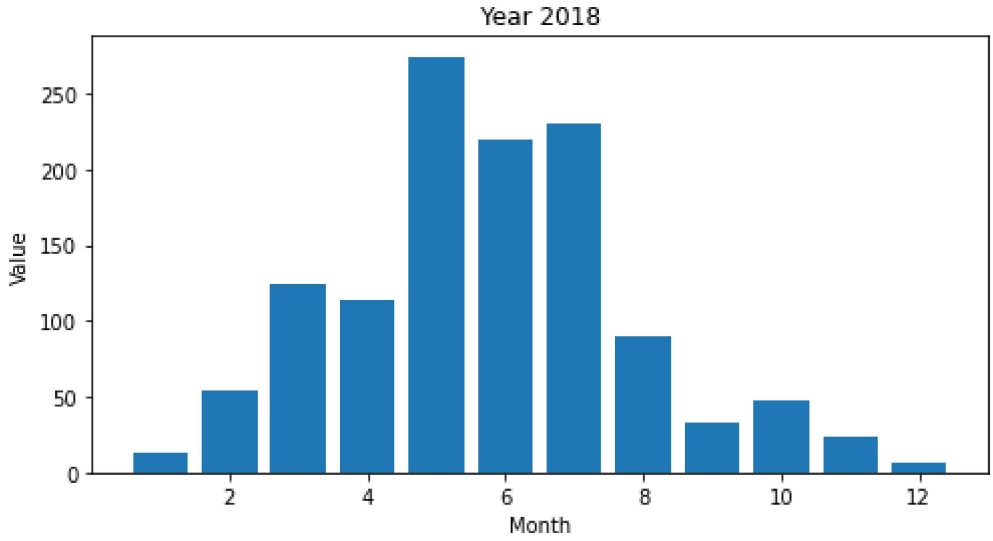
	referenceTime	value
0	2018-01-31 00:00:00+00:00	12.893884
1	2018-02-28 00:00:00+00:00	53.589844
2	2018-03-31 00:00:00+00:00	123.750011
3	2018-04-30 00:00:00+00:00	113.232876
4	2018-05-31 00:00:00+00:00	274.477007
5	2018-06-30 00:00:00+00:00	219.122131
6	2018-07-31 00:00:00+00:00	230.426753
7	2018-08-31 00:00:00+00:00	90.421403
8	2018-09-30 00:00:00+00:00	32.905474
9	2018-10-31 00:00:00+00:00	47.286329
10	2018-11-30 00:00:00+00:00	23.319613
11	2018-12-31 00:00:00+00:00	6.683515
12	2020-01-31 00:00:00+00:00	4.329312
13	2020-02-29 00:00:00+00:00	22.901684
14	2020-03-31 00:00:00+00:00	17.131618
15	2020-04-30 00:00:00+00:00	138.769993
16	2020-05-31 00:00:00+00:00	151.512310
17	2020-06-30 00:00:00+00:00	223.265718
18	2020-07-31 00:00:00+00:00	134.312187
19	2020-08-31 00:00:00+00:00	169.409163
20	2020-09-30 00:00:00+00:00	46.390482
21	2020-10-31 00:00:00+00:00	45.264501
22	2020-11-30 00:00:00+00:00	3.004583
23	2020-12-31 00:00:00+00:00	2.348430
24	2021-01-31 00:00:00+00:00	16.400786
25	2021-02-28 00:00:00+00:00	2.471936
26	2021-03-31 00:00:00+00:00	36.400992
27	2021-04-30 00:00:00+00:00	63.905121
28	2021-05-31 00:00:00+00:00	183.082876
29	2021-06-30 00:00:00+00:00	196.748235
30	2021-07-31 00:00:00+00:00	123.098610
31	2021-08-31 00:00:00+00:00	222.829415
32	2021-09-30 00:00:00+00:00	63.127483
33	2021-10-31 00:00:00+00:00	1.700034

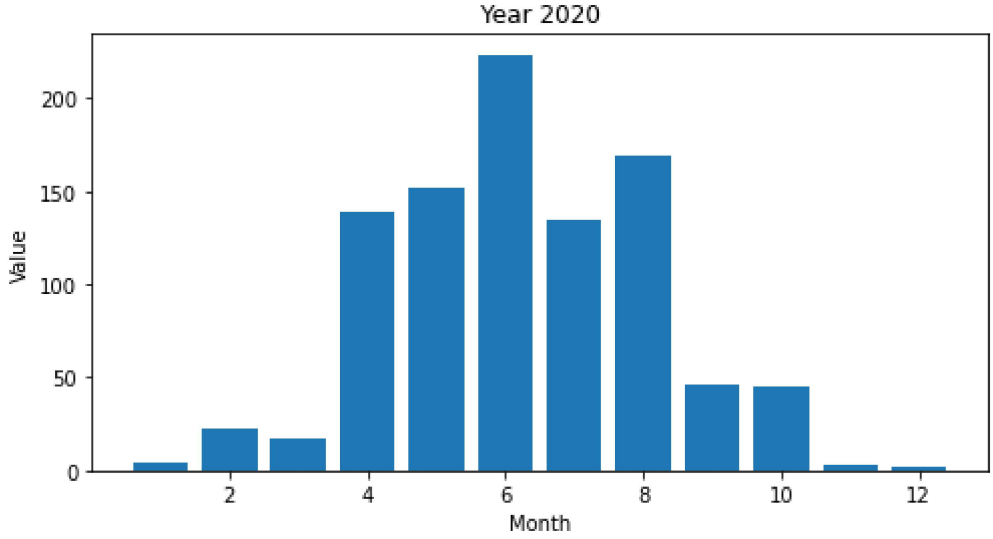
	referenceTime	value
34	2021-11-30 00:00:00+00:00	5.094903
35	2021-12-31 00:00:00+00:00	1.683159
36	2022-01-31 00:00:00+00:00	2.328355
37	2022-02-28 00:00:00+00:00	10.818632
38	2022-03-31 00:00:00+00:00	78.312202
39	2022-04-30 00:00:00+00:00	125.141301
40	2022-05-31 00:00:00+00:00	96.292082
41	2022-06-30 00:00:00+00:00	131.983147
42	2022-07-31 00:00:00+00:00	11.230421
43	2022-08-31 00:00:00+00:00	74.057589
44	2022-09-30 00:00:00+00:00	139.043600
45	2022-10-31 00:00:00+00:00	10.880946
46	2022-11-30 00:00:00+00:00	12.873458
47	2022-12-31 00:00:00+00:00	5.334525
48	2023-01-31 00:00:00+00:00	4.165316
49	2023-02-28 00:00:00+00:00	6.257792
50	2023-03-31 00:00:00+00:00	39.747470
51	2023-04-30 00:00:00+00:00	132.947776

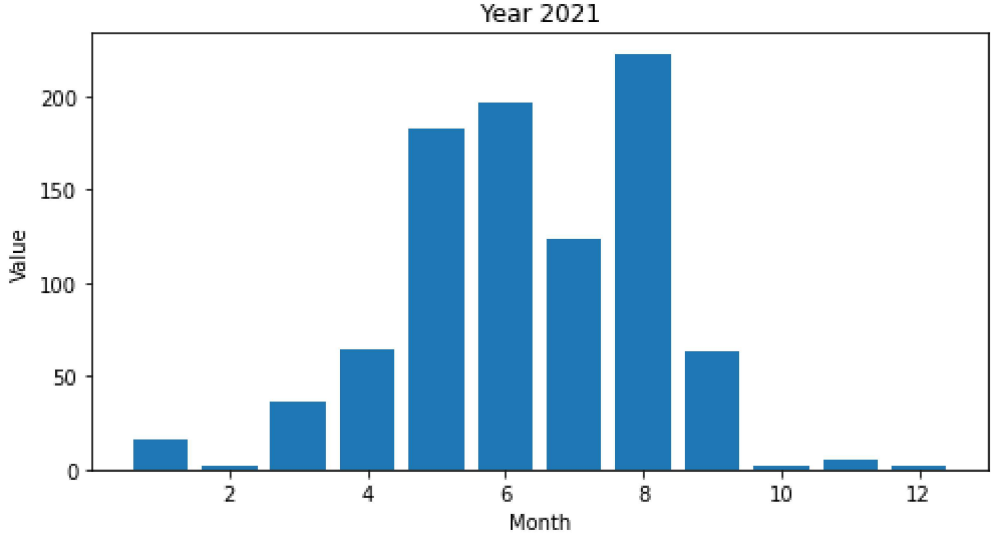
Now we want to graph each year and see how each month the value of solar irradation change

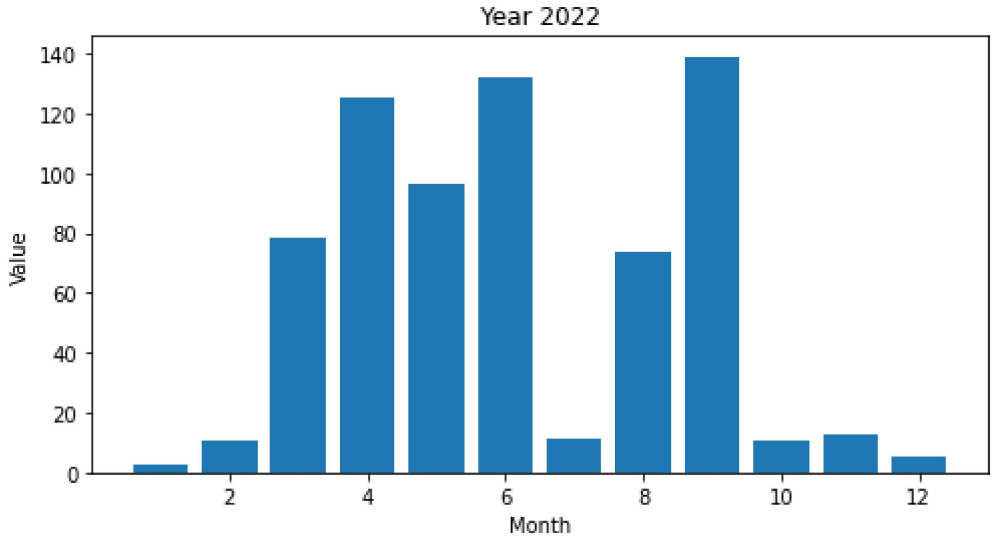
In [ ]:

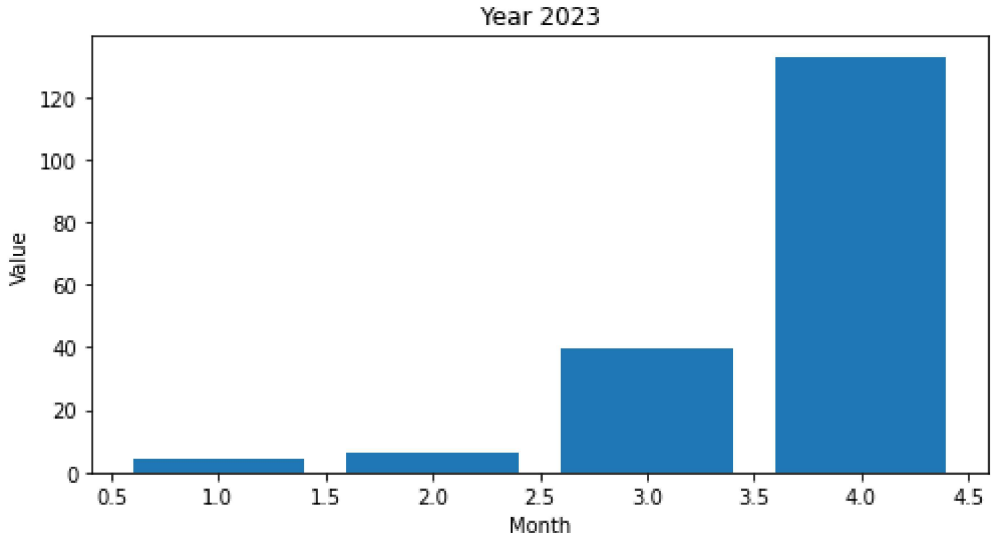
```
# create a separate plot for each year
years = df_monthly['referenceTime'].dt.year.unique()
for year in years:
    # filter the grouped DataFrame by year
    df_year = df_monthly[df_monthly['referenceTime'].dt.year == year]

    # create a bar plot of the monthly values for the year
    plt.figure(figsize=(8, 4))
    plt.bar(df_year['referenceTime'].dt.month, df_year['value'])
    plt.title(f'Year {year}')
    plt.xlabel('Month')
    plt.ylabel('Value')
    plt.show()
```


Year 2018
Value
Month
0
50
100
150
200
250
2
4
6
8
10
12


Year 2020
Value
Month
0
50
100
150
200
2
4
6
8
10
12


Year 2021
Value
Month
0
50
100
150
200
2
4
6
8
10
12





We can see how on summer solar irradiance increase its value. Also, how in winter decrease significantly.

Estimation Solar irradiance for 2023

Geometric Brownian Motion (GBM) and Geometric Ornstein-Uhlenbeck Process (GOU) are typically used to model stock prices and financial data, where the underlying process is assumed to be continuous and stationary over time. However, solar irradiance measurements are influenced by many factors that are not stationary, such as weather patterns, atmospheric conditions, and time of day. The cyclic nature of the solar irradiance data, combined with the non-stationary nature of the underlying process, makes it difficult to use GBM and GOU models to accurately estimate solar irradiance measurements.

Therefore we have decided to do estimation using Random Forest because it can handle non-stationary data and capture the complex interactions between different features that affect solar irradiance, such as weather patterns, time of day, and seasonal changes. Random Forest models are able to account for the nonlinear relationships between these features, which makes them a good choice for modeling solar irradiance data.

We filter the table we had by months "df_monthly" so we do not have 2023 data.

In [ ]:

```
desired_date = pd.to_datetime('2022-12-31')
df_monthly_wo_2023 = df_monthly[df_monthly['referenceTime'].dt.year <=
                                desired_date.year]
df_monthly_wo_2023
```

Out[ ]:

	referenceTime	value
0	2018-01-31 00:00:00+00:00	12.893884
1	2018-02-28 00:00:00+00:00	53.589844
2	2018-03-31 00:00:00+00:00	123.750011
3	2018-04-30 00:00:00+00:00	113.232876
4	2018-05-31 00:00:00+00:00	274.477007
5	2018-06-30 00:00:00+00:00	219.122131
6	2018-07-31 00:00:00+00:00	230.426753
7	2018-08-31 00:00:00+00:00	90.421403
8	2018-09-30 00:00:00+00:00	32.905474
9	2018-10-31 00:00:00+00:00	47.286329
10	2018-11-30 00:00:00+00:00	23.319613
11	2018-12-31 00:00:00+00:00	6.683515
12	2020-01-31 00:00:00+00:00	4.329312
13	2020-02-29 00:00:00+00:00	22.901684
14	2020-03-31 00:00:00+00:00	17.131618
15	2020-04-30 00:00:00+00:00	138.769993
16	2020-05-31 00:00:00+00:00	151.512310
17	2020-06-30 00:00:00+00:00	223.265718
18	2020-07-31 00:00:00+00:00	134.312187
19	2020-08-31 00:00:00+00:00	169.409163
20	2020-09-30 00:00:00+00:00	46.390482
21	2020-10-31 00:00:00+00:00	45.264501
22	2020-11-30 00:00:00+00:00	3.004583
23	2020-12-31 00:00:00+00:00	2.348430
24	2021-01-31 00:00:00+00:00	16.400786
25	2021-02-28 00:00:00+00:00	2.471936
26	2021-03-31 00:00:00+00:00	36.400992
27	2021-04-30 00:00:00+00:00	63.905121
28	2021-05-31 00:00:00+00:00	183.082876
29	2021-06-30 00:00:00+00:00	196.748235

	referenceTime	value
30	2021-07-31 00:00:00+00:00	123.098610
31	2021-08-31 00:00:00+00:00	222.829415
32	2021-09-30 00:00:00+00:00	63.127483
33	2021-10-31 00:00:00+00:00	1.700034
34	2021-11-30 00:00:00+00:00	5.094903
35	2021-12-31 00:00:00+00:00	1.683159
36	2022-01-31 00:00:00+00:00	2.328355
37	2022-02-28 00:00:00+00:00	10.818632
38	2022-03-31 00:00:00+00:00	78.312202
39	2022-04-30 00:00:00+00:00	125.141301
40	2022-05-31 00:00:00+00:00	96.292082
41	2022-06-30 00:00:00+00:00	131.983147
42	2022-07-31 00:00:00+00:00	11.230421
43	2022-08-31 00:00:00+00:00	74.057589
44	2022-09-30 00:00:00+00:00	139.043600
45	2022-10-31 00:00:00+00:00	10.880946
46	2022-11-30 00:00:00+00:00	12.873458
47	2022-12-31 00:00:00+00:00	5.334525

Here we will create a models array with a model for each month across all the years, since for example the idea it is to estimate January 2023, taking into account January month of 2018,2020,2021 and 2022

In [ ]:

```
# Extract the year and month from the "referenceTime" column
df_monthly_wo_2023 = df_monthly_wo_2023.assign(
    year=df_monthly_wo_2023['referenceTime'].dt.year)
df_monthly_wo_2023 = df_monthly_wo_2023.assign(
    month=df_monthly_wo_2023['referenceTime'].dt.month)

# Create a list to store the trained models
models = []

# Loop through each month
for month in range(1, 13):
    # Extract the X and y data for the current month across all years
    X_month = df_monthly_wo_2023[df_monthly_wo_2023['month']
                                 == month]['year'].values.reshape(-1, 1)
    y_month = df_monthly_wo_2023[df_monthly_wo_2023['month']
                                 == month]['value'].values

    # Select data up to 2022-12-31 for training
    X_train = X_month[X_month[:, 0] < 2023]
    y_train = y_month[X_month[:, 0] < 2023]
```

```
    # Train a random forest regressor model for the current month
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Add the trained model to the list of models
    models.append(model)
```

Here we make predicitons for each of the months of 2023 so at the end we will have an array of 12 estimations.

In [ ]:
```
# Use the trained models to predict values for January to December 2023
X_test = pd.date_range(start='2023-01-01', end='2023-12-01',
                       freq='MS').to_numpy().reshape(-1, 1)


y_pred_energ_produced = np.zeros(12)

for month in range(1, 13):
    # Extract the X data for the current month in 2023
    X_month = X_test[np.where(X_test[:, 0].astype(
        'datetime64[M]').astype(int) % 12 + 1 == month)]



    # Predict the y values for the current month in 2023
    model = models[month - 1]

    y_month = model.predict(X_month)

    y_pred_energ_produced[np.where(X_test[:, 0].astype('datetime64[M]').astype(
        int) % 12 + 1 == month)] = y_month
```

Here we compare the values that we already have for 2023 and the ones we have predicted for January, February and March

In [ ]:
```
start_date = '2023-01-01'
end_date = '2023-03-31'
filtered_df = df_monthly[(df_monthly['referenceTime'] >= start_date)
                         & (df_monthly['referenceTime'] <= end_date)]

# Corroborate predicted values for January, February, and March 2023
actual_values = filtered_df['value'].to_numpy()
predicted_values = y_pred_energ_produced[:3]

for actual, predicted in zip(actual_values, predicted_values):
    print(f"Actual value: {actual:.2f}  Predicted value: {predicted:.2f}")
```

```
Actual value: 4.17  Predicted value: 5.56
Actual value: 6.26  Predicted value: 9.83
Actual value: 39.75  Predicted value: 64.81
```

The rest of the months solar irradiance values are here.

In [ ]:
```
# Print predicted values for April to December 2023
for i, value in enumerate(y_pred_energ_produced[3:], start=4):
    print(f"Predicted value for {i:02d}-2023: {value:.2f}")
```

```
Predicted value for 04-2023: 112.62
Predicted value for 05-2023: 119.25
Predicted value for 06-2023: 152.62
Predicted value for 07-2023: 44.46
Predicted value for 08-2023: 113.46
Predicted value for 09-2023: 115.86
Predicted value for 10-2023: 11.27
Predicted value for 11-2023: 10.47
Predicted value for 12-2023: 4.32
```

Now we have to add all these values that we estimated from March/2023 until Dec/2023 into the already created table that has every month solar irradiance average "df_monthly".

In [ ]:

```
# create a dataframe with the new rows
new_rows = pd.DataFrame({
    'referenceTime': ['2023-05-31', '2023-06-30', '2023-07-31', '2023-08-31', '2023-09-
    'value': y_pred_energ_produced[4:]
})
new_rows['referenceTime'] = pd.to_datetime(new_rows['referenceTime'])
new_rows['referenceTime'] = new_rows['referenceTime'].dt.strftime(
    '%Y-%m-%d %H:%M:%S+00:00')

# append the new rows to the existing dataframe
df_monthly = df_monthly.append(new_rows, ignore_index=True)
df_monthly
```

Out[ ]:

	referenceTime	value
0	2018-01-31 00:00:00+00:00	12.893884
1	2018-02-28 00:00:00+00:00	53.589844
2	2018-03-31 00:00:00+00:00	123.750011
3	2018-04-30 00:00:00+00:00	113.232876
4	2018-05-31 00:00:00+00:00	274.477007
5	2018-06-30 00:00:00+00:00	219.122131
6	2018-07-31 00:00:00+00:00	230.426753
7	2018-08-31 00:00:00+00:00	90.421403
8	2018-09-30 00:00:00+00:00	32.905474
9	2018-10-31 00:00:00+00:00	47.286329
10	2018-11-30 00:00:00+00:00	23.319613
11	2018-12-31 00:00:00+00:00	6.683515
12	2020-01-31 00:00:00+00:00	4.329312
13	2020-02-29 00:00:00+00:00	22.901684
14	2020-03-31 00:00:00+00:00	17.131618
15	2020-04-30 00:00:00+00:00	138.769993
16	2020-05-31 00:00:00+00:00	151.512310
17	2020-06-30 00:00:00+00:00	223.265718
18	2020-07-31 00:00:00+00:00	134.312187

	referenceTime	value
19	2020-08-31 00:00:00+00:00	169.409163
20	2020-09-30 00:00:00+00:00	46.390482
21	2020-10-31 00:00:00+00:00	45.264501
22	2020-11-30 00:00:00+00:00	3.004583
23	2020-12-31 00:00:00+00:00	2.348430
24	2021-01-31 00:00:00+00:00	16.400786
25	2021-02-28 00:00:00+00:00	2.471936
26	2021-03-31 00:00:00+00:00	36.400992
27	2021-04-30 00:00:00+00:00	63.905121
28	2021-05-31 00:00:00+00:00	183.082876
29	2021-06-30 00:00:00+00:00	196.748235
30	2021-07-31 00:00:00+00:00	123.098610
31	2021-08-31 00:00:00+00:00	222.829415
32	2021-09-30 00:00:00+00:00	63.127483
33	2021-10-31 00:00:00+00:00	1.700034
34	2021-11-30 00:00:00+00:00	5.094903
35	2021-12-31 00:00:00+00:00	1.683159
36	2022-01-31 00:00:00+00:00	2.328355
37	2022-02-28 00:00:00+00:00	10.818632
38	2022-03-31 00:00:00+00:00	78.312202
39	2022-04-30 00:00:00+00:00	125.141301
40	2022-05-31 00:00:00+00:00	96.292082
41	2022-06-30 00:00:00+00:00	131.983147
42	2022-07-31 00:00:00+00:00	11.230421
43	2022-08-31 00:00:00+00:00	74.057589
44	2022-09-30 00:00:00+00:00	139.043600
45	2022-10-31 00:00:00+00:00	10.880946
46	2022-11-30 00:00:00+00:00	12.873458
47	2022-12-31 00:00:00+00:00	5.334525
48	2023-01-31 00:00:00+00:00	4.165316
49	2023-02-28 00:00:00+00:00	6.257792
50	2023-03-31 00:00:00+00:00	39.747470
51	2023-04-30 00:00:00+00:00	132.947776

	referenceTime	value
52	2023-05-31 00:00:00+00:00	119.251473
53	2023-06-30 00:00:00+00:00	152.621247
54	2023-07-31 00:00:00+00:00	44.457146
55	2023-08-31 00:00:00+00:00	113.462001
56	2023-09-30 00:00:00+00:00	115.856336
57	2023-10-31 00:00:00+00:00	11.267994
58	2023-11-30 00:00:00+00:00	10.471355
59	2023-12-31 00:00:00+00:00	4.322198

Here corroborate that the "referenceTime" column has the same format for all values.

In [ ]:
```
df_monthly['referenceTime'] = pd.to_datetime(
    df_monthly['referenceTime'], utc=True
)

df_monthly['referenceTime'] = df_monthly['referenceTime'].dt.tz_convert(None)
df_monthly['referenceTime'] = pd.to_datetime(
    df_monthly['referenceTime'], format='%Y-%m-%d %H:%M:%S'
)
```

Here we calculated the variances for each month across all the years.

In [ ]:
```
# Extract the month from the referenceTime column
df_monthly['month'] = df_monthly['referenceTime'].dt.month

# Calculate the variance for each month across all years studied
variances = []
for month in range(1, 13):
    # Extract the data for the current month
    data = df_monthly.loc[df_monthly['month'] == month, 'value']

    # Calculate the variance for the current month
    variance = np.var(data)

    # Add the variance to the list
    variances.append({'month': month, 'variance': variance})

# Convert the list of variances to a pandas DataFrame
variances_df = pd.DataFrame(variances)
```

Here we calculated the standard deviation for each month, the mean and the porcentage of uncertainty.

In [ ]:
```
# Calculate the standard deviation for each month
variances_df['std_dev'] = variances_df['variance'].apply(np.sqrt)

# Calculate the mean for each month
means = []
for month in range(1, 13):
```

```
    # Extract the data for the current month
    data = df_monthly.loc[df_monthly['month'] == month, 'value']

    # Calculate the mean for the current month
    mean = np.mean(data)

    # Add the mean to the list
    means.append(mean)

# Add the means to the variances_df DataFrame
variances_df['mean'] = means

# Calculate the percentage of uncertainty for each month
variances_df['uncertainty_percent'] = (
    variances_df['std_dev'] / variances_df['mean']) * 100
variances_df
```

Out[ ]:

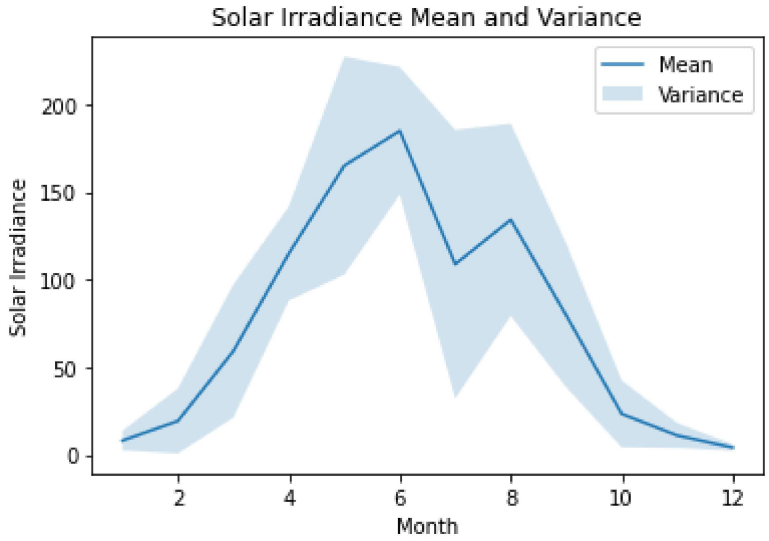
	month	variance	std_dev	mean	uncertainty_percent
0	1	30.973371	5.565372	8.023531	69.363137
1	2	342.787940	18.514533	19.207978	96.389811
2	3	1439.967608	37.946905	59.068459	64.242247
3	4	720.717890	26.846189	114.799413	23.385302
4	5	3861.559880	62.141451	164.923150	37.679035
5	6	1325.091816	36.401811	184.748096	19.703484
6	7	5861.633925	76.561308	108.705023	70.430331
7	8	3011.692712	54.878891	134.035914	40.943423
8	9	1680.513323	40.994064	79.464675	51.587783
9	10	364.668378	19.096292	23.279961	82.028885
10	11	50.869575	7.132291	10.952782	65.118532
11	12	3.430761	1.852231	4.074365	45.460613

To have better understanding of the data , we plot the mean and the variance for each month. Here hsitorical and predicted value is taken into account.

In [ ]:

```
# Plot the mean solar irradiance for each month
plt.plot(variances_df['month'], variances_df['mean'], label='Mean')

# Plot the variance as a shaded area around the mean
plt.fill_between(variances_df['month'], variances_df['mean'] - variances_df['std_dev'],
                 variances_df['mean'] + variances_df['std_dev'], alpha=0.2, label='Vari

# Set the title and axis labels
plt.title('Solar Irradiance Mean and Variance')
plt.xlabel('Month')
plt.ylabel('Solar Irradiance')

# Add a legend
plt.legend()
```

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



surprisingly, on winter season there is less variance, on the hand during summer (July) is where we have the biggest variance.

Estimation of how much energy can be produced from a solar panel

Now we want to estimate how much energy can be produced from a solar panel having the solar irradiance we calculated before.

Power = Area x Efficiency x Solar Irradiance

The average area is 1.6 m^2 to 2 m^2 and efficiency is 15% to 22% of a solar panel.

In [ ]:
```
df_monthly["Power (Watts)"]= df_monthly['value']*1.6*0.15
```

In [ ]:
```
df_monthly["Energy (Watts*h)"]= df_monthly['Power (Watts)']*12
```

In [ ]:
```
# rename the columns
df_monthly = df_monthly.rename(
    columns={'value': 'Sollar Irradiance (W/m2)'})
df_monthly = df_monthly.drop('month', axis=1)
```

We decided to be negative and use the most small solar panels with less eficiency

In [ ]:
```
df_monthly
```

Out[ ]:

	referenceTime	Sollar Irradiance (W/m2)	Power (Watts)	Energy (Watts*h)
0	2018-01-31	12.893884	3.094532	37.134387
1	2018-02-28	53.589844	12.861563	154.338750
2	2018-03-31	123.750011	29.700003	356.400031

	referenceTime	Sollar Irradiance (W/m2)	Power (Watts)	Energy (Watts*h)
3	2018-04-30	113.232876	27.175890	326.110684
4	2018-05-31	274.477007	65.874482	790.493779
5	2018-06-30	219.122131	52.589311	631.071736
6	2018-07-31	230.426753	55.302421	663.629050
7	2018-08-31	90.421403	21.701137	260.413641
8	2018-09-30	32.905474	7.897314	94.767764
9	2018-10-31	47.286329	11.348719	136.184627
10	2018-11-30	23.319613	5.596707	67.160484
11	2018-12-31	6.683515	1.604044	19.248523
12	2020-01-31	4.329312	1.039035	12.468418
13	2020-02-29	22.901684	5.496404	65.956850
14	2020-03-31	17.131618	4.111588	49.339061
15	2020-04-30	138.769993	33.304798	399.657580
16	2020-05-31	151.512310	36.362955	436.355454
17	2020-06-30	223.265718	53.583772	643.005268
18	2020-07-31	134.312187	32.234925	386.819098
19	2020-08-31	169.409163	40.658199	487.898388
20	2020-09-30	46.390482	11.133716	133.604588
21	2020-10-31	45.264501	10.863480	130.361764
22	2020-11-30	3.004583	0.721100	8.653200
23	2020-12-31	2.348430	0.563623	6.763479
24	2021-01-31	16.400786	3.936189	47.234264
25	2021-02-28	2.471936	0.593265	7.119176
26	2021-03-31	36.400992	8.736238	104.834857
27	2021-04-30	63.905121	15.337229	184.046748
28	2021-05-31	183.082876	43.939890	527.278684
29	2021-06-30	196.748235	47.219576	566.634918
30	2021-07-31	123.098610	29.543666	354.523998
31	2021-08-31	222.829415	53.479060	641.748716
32	2021-09-30	63.127483	15.150596	181.807152
33	2021-10-31	1.700034	0.408008	4.896097
34	2021-11-30	5.094903	1.222777	14.673320
35	2021-12-31	1.683159	0.403958	4.847497
36	2022-01-31	2.328355	0.558805	6.705662

	referenceTime	Sollar Irradiance (W/m2)	Power (Watts)	Energy (Watts*h)
37	2022-02-28	10.818632	2.596472	31.157661
38	2022-03-31	78.312202	18.794928	225.539141
39	2022-04-30	125.141301	30.033912	360.406948
40	2022-05-31	96.292082	23.110100	277.321196
41	2022-06-30	131.983147	31.675955	380.111464
42	2022-07-31	11.230421	2.695301	32.343612
43	2022-08-31	74.057589	17.773821	213.285855
44	2022-09-30	139.043600	33.370464	400.445568
45	2022-10-31	10.880946	2.611427	31.337125
46	2022-11-30	12.873458	3.089630	37.075560
47	2022-12-31	5.334525	1.280286	15.363432
48	2023-01-31	4.165316	0.999676	11.996110
49	2023-02-28	6.257792	1.501870	18.022440
50	2023-03-31	39.747470	9.539393	114.472715
51	2023-04-30	132.947776	31.907466	382.889594
52	2023-05-31	119.251473	28.620353	343.444241
53	2023-06-30	152.621247	36.629099	439.549190
54	2023-07-31	44.457146	10.669715	128.036581
55	2023-08-31	113.462001	27.230880	326.770562
56	2023-09-30	115.856336	27.805521	333.666248
57	2023-10-31	11.267994	2.704319	32.451824
58	2023-11-30	10.471355	2.513125	30.157502
59	2023-12-31	4.322198	1.037327	12.447930

Electricity Consumption

Now we want to find how the electricity comsuption has been through the same years that we have studied before. The data was taken from (Ref 3.).

First we download the data from 2018 and read it.

In [ ]:

```
# read the Excel file into a pandas dataframe
df_2018_electricity = pd.read_excel('12824_20230425-103240.xlsx')
# remove the last 36 rows and keep only the first two columns
df_2018_electricity = df_2018_electricity.iloc[:-36, :2]
df_2018_electricity['Unnamed: 0'] = pd.to_datetime(
    df_2018_electricity['Unnamed: 0'])
df_2018_electricity.rename(
    columns={'Unnamed: 0': 'referenceTime'}, inplace=True)
```

```
df_2018_electricity['referenceTime'] = df_2018_electricity['referenceTime'].dt.strftime
    '%Y-%m-%d %H:%M:%S+00:00')
```

Now we do the same with the data from 2020 to 2023

In [ ]:
```
# read the Excel file into a pandas dataframe
df_2020_2023_electricity = pd.read_excel('12824_20230425-111708.xlsx')
df_2020_2023_electricity['Unnamed: 0'] = pd.to_datetime(df_2020_2023_electricity['Unnam
df_2020_2023_electricity.rename(columns={
                                'Unnamed: 0': "referenceTime", 'Unnamed: 1': "Net consu
df_2020_2023_electricity=df_2020_2023_electricity.drop(columns='Unnamed: 2')

df_2020_2023_electricity['referenceTime'] = df_2020_2023_electricity['referenceTime'].d
    '%Y-%m-%d %H:%M:%S+00:00')
# print the dataframe
```

We merge both data frames to have a final table called "df_concat_electricty"

In [ ]:
```
df_concat_electricty = pd.concat(
    [df_2018_electricity, df_2020_2023_electricity], ignore_index=True)
df_concat_electricty
```

Out[ ]:

	referenceTime	Net consumption of electricity Norway (Kwh)
0	2018-01-01 00:00:00+00:00	13505959.0
1	2018-02-01 00:00:00+00:00	12459903.0
2	2018-03-01 00:00:00+00:00	13245245.0
3	2018-04-01 00:00:00+00:00	10474895.0
4	2018-05-01 00:00:00+00:00	8777033.0
5	2018-06-01 00:00:00+00:00	8082914.0
6	2018-07-01 00:00:00+00:00	7743382.0
7	2018-08-01 00:00:00+00:00	8251207.0
8	2018-09-01 00:00:00+00:00	8757348.0
9	2018-10-01 00:00:00+00:00	10544820.0
10	2018-11-01 00:00:00+00:00	11265288.0
11	2018-12-01 00:00:00+00:00	12956055.0
12	2020-01-01 00:00:00+00:00	12551169.0
13	2020-02-01 00:00:00+00:00	11976455.0
14	2020-03-01 00:00:00+00:00	12215571.0
15	2020-04-01 00:00:00+00:00	10229631.0
16	2020-05-01 00:00:00+00:00	9472772.0
17	2020-06-01 00:00:00+00:00	7862932.0
18	2020-07-01 00:00:00+00:00	8089060.0

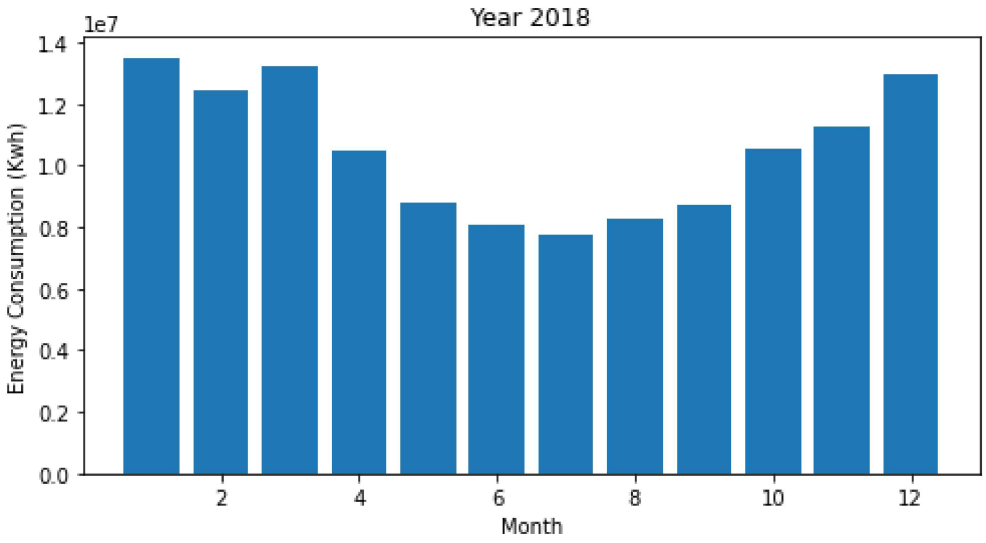
	referenceTime	Net consumption of electricity Norway (Kwh)
19	2020-08-01 00:00:00+00:00	8255492.0
20	2020-09-01 00:00:00+00:00	8743054.0
21	2020-10-01 00:00:00+00:00	10304724.0
22	2020-11-01 00:00:00+00:00	11110123.0
23	2020-12-01 00:00:00+00:00	12509855.0
24	2021-01-01 00:00:00+00:00	14510965.0
25	2021-02-01 00:00:00+00:00	13066898.0
26	2021-03-01 00:00:00+00:00	12306121.0
27	2021-04-01 00:00:00+00:00	10823770.0
28	2021-05-01 00:00:00+00:00	9806074.0
29	2021-06-01 00:00:00+00:00	8210925.0
30	2021-07-01 00:00:00+00:00	8110733.0
31	2021-08-01 00:00:00+00:00	8558259.0
32	2021-09-01 00:00:00+00:00	8788003.0
33	2021-10-01 00:00:00+00:00	10256490.0
34	2021-11-01 00:00:00+00:00	11488483.0
35	2021-12-01 00:00:00+00:00	13369253.0
36	2022-01-01 00:00:00+00:00	12908167.0
37	2022-02-01 00:00:00+00:00	11694410.0
38	2022-03-01 00:00:00+00:00	11588327.0
39	2022-04-01 00:00:00+00:00	10315627.0
40	2022-05-01 00:00:00+00:00	9401008.0
41	2022-06-01 00:00:00+00:00	8289295.0
42	2022-07-01 00:00:00+00:00	8221028.0
43	2022-08-01 00:00:00+00:00	8387130.0
44	2022-09-01 00:00:00+00:00	8618906.0
45	2022-10-01 00:00:00+00:00	9907692.0
46	2022-11-01 00:00:00+00:00	10555868.0
47	2022-12-01 00:00:00+00:00	12974232.0
48	2023-01-01 00:00:00+00:00	12731304.0
49	2023-02-01 00:00:00+00:00	11072007.0
50	2023-03-01 00:00:00+00:00	12488414.0

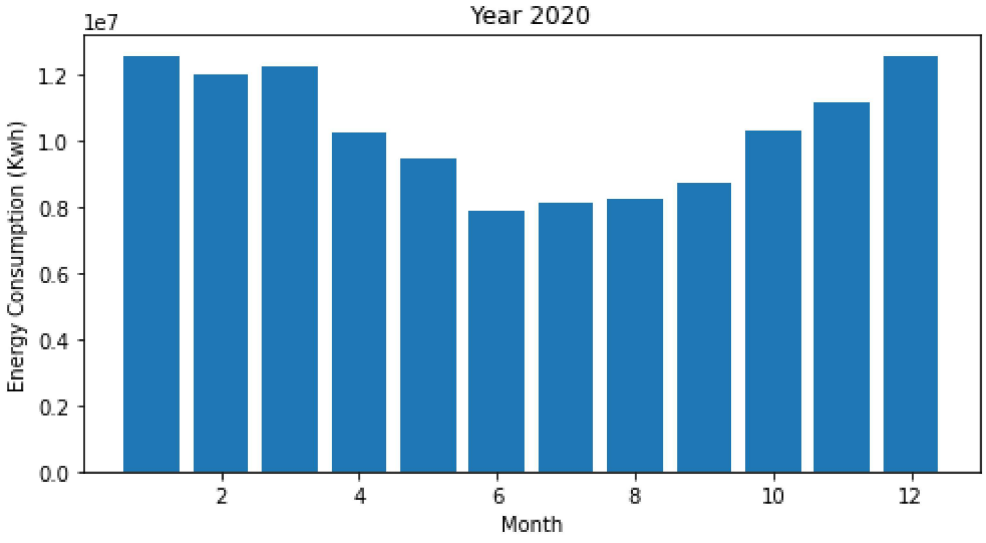
We remove 2023 data from above table and have new table "df_concat_electricty_wo_2023"

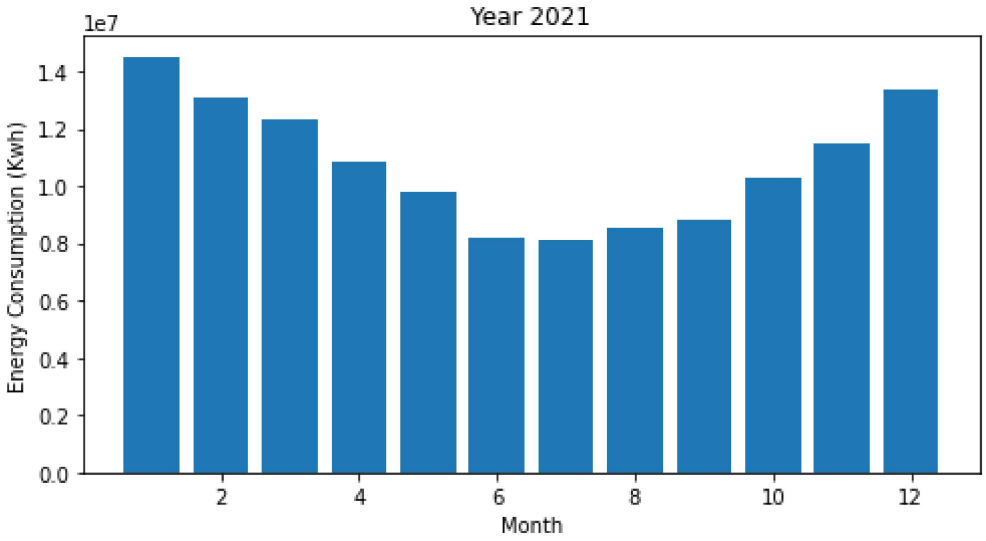
In [ ]:
```
df_concat_electricty['referenceTime'] = pd.to_datetime(
    df_concat_electricty['referenceTime'])
desired_date = pd.to_datetime('2022-12-31')
df_concat_electricty_wo_2023 = df_concat_electricty[df_concat_electricty['referenceTime
                                desired_date.year]
```

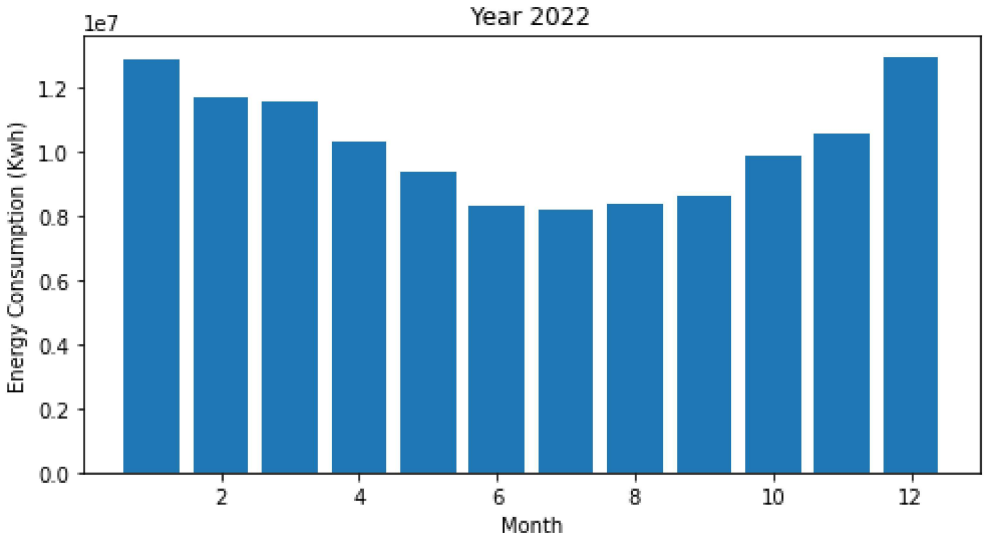
We want to see now how the comsuption of electricty has been through the years.

In [ ]:
```
# create a separate plot for each year
years = df_concat_electricty_wo_2023['referenceTime'].dt.year.unique()
for year in years:
    # filter the grouped DataFrame by year
    df_year = df_concat_electricty_wo_2023[df_concat_electricty_wo_2023['referenceTime'

    # create a bar plot of the monthly values for the year
    plt.figure(figsize=(8, 4))
    plt.bar(df_year['referenceTime'].dt.month,
            df_year['Net consumption of electricity Norway (Kwh)'])
    plt.title(f'Year {year}')
    plt.xlabel('Month')
    plt.ylabel('Energy Consumption (Kwh)')
    plt.show()
```









As we expected during summer electricty comsuption is less.

We have a model for each month across all the years, since it is cycling information. January 2023 will be influenced by the January of 2018,2020,2021 and 2022.

In [ ]:

```
# Extract the year and month from the "referenceTime" column
df_concat_electricty_wo_2023 = df_concat_electricty_wo_2023.assign(
    year=df_concat_electricty_wo_2023['referenceTime'].dt.year)
df_concat_electricty_wo_2023 = df_concat_electricty_wo_2023.assign(
    month=df_concat_electricty_wo_2023['referenceTime'].dt.month)


# Create a list to store the trained models
models = []

# Loop through each month
for month in range(1, 13):
    # Extract the X and y data for the current month across all years
    X_month = df_concat_electricty_wo_2023[df_concat_electricty_wo_2023['month']
                                == month]['year'].values.reshape(-1, 1)
    y_month = df_concat_electricty_wo_2023[df_concat_electricty_wo_2023['month']
                                           == month]['Net consumption of electricity No

    # Select data up to 2022-12-31 for training
```

```
    X_train = X_month[X_month[:, 0] < 2023]
    y_train = y_month[X_month[:, 0] < 2023]

    # Train a random forest regressor model for the current month
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Add the trained model to the list of models
    models.append(model)
```

Here we do the prediciton for each month in 2023, so at the end we will have an array of 12 predictions.

In [ ]:
```
# Use the trained models to predict values for January to December 2023
X_test = pd.date_range(start='2023-01-01', end='2023-12-01',
                       freq='MS').to_numpy().reshape(-1, 1)


y_pred_energy_consumed = np.zeros(12)

for month in range(1, 13):
    # Extract the X data for the current month in 2023
    X_month = X_test[np.where(X_test[:, 0].astype(
        'datetime64[M]').astype(int) % 12 + 1 == month)]

    # Predict the y values for the current month in 2023
    model = models[month - 1]

    y_month = model.predict(X_month)

    y_pred_energy_consumed[np.where(X_test[:, 0].astype('datetime64[M]').astype(
        int) % 12 + 1 == month)] = y_month
```

Here we compare the actual information that we have for 2023 and the predicitons done.

In [ ]:
```
start_date = '2023-01-01'
end_date = '2023-03-31'
filtered_df = df_concat_electricty[(df_concat_electricty['referenceTime'] >= start_date
                                    & (df_concat_electricty['referenceTime'] <= end_date

# Corroborate predicted values for January, February, and March 2023
actual_values = filtered_df['Net consumption of electricity Norway (Kwh)'].to_numpy(
)
predicted_values = y_pred_energy_consumed[:3]

for actual, predicted in zip(actual_values, predicted_values):
    print(f"Actual value: {actual:.2f}  Predicted value: {predicted:.2f}")
```

```
Actual value: 12731304.00  Predicted value: 13235792.70
Actual value: 11072007.00  Predicted value: 12016100.51
Actual value: 12488414.00  Predicted value: 11790148.76
```

Here we show the predictions of how is going to be the electricity comsuption for the rest of the months of 2023.

In [ ]:
```
# Print predicted values for April to December 2023
for i, value in enumerate(y_pred_energy_consumed[3:], start=4):
```

```
    print(f"Predicted value for {i:02d}-2023: {value:.2f}")
```

```
Predicted value for 04-2023: 10421398.74
Predicted value for 05-2023: 9495146.00
Predicted value for 06-2023: 8242208.19
Predicted value for 07-2023: 8187525.34
Predicted value for 08-2023: 8415563.72
Predicted value for 09-2023: 8664797.70
Predicted value for 10-2023: 10012219.80
Predicted value for 11-2023: 10799841.15
Predicted value for 12-2023: 13028630.23
```

Here we merged the data that we predict and the table we had before. Creating a new table "df_concat_electricty"

In [ ]:
```
# create a dataframe with the new rows
new_rows = pd.DataFrame({
    'referenceTime': ['2023-04-30','2023-05-31', '2023-06-30', '2023-07-31', '2023-08-3
    'Net consumption of electricity Norway (Kwh)': y_pred_energy_consumed[3:]
})
new_rows['referenceTime'] = pd.to_datetime(new_rows['referenceTime'])
new_rows['referenceTime'] = new_rows['referenceTime'].dt.strftime(
    '%Y-%m-%d %H:%M:%S+00:00')

# append the new rows to the existing dataframe
df_concat_electricty = df_concat_electricty.append(new_rows, ignore_index=True)
```

Here we make sure the "refecenreTime" column has the same format in all rows.

In [ ]:
```
df_concat_electricty['referenceTime'] = pd.to_datetime(
    df_concat_electricty['referenceTime'], utc=True
)

df_concat_electricty['referenceTime'] = df_concat_electricty['referenceTime'].dt.tz_con
    None)
df_concat_electricty['referenceTime'] = pd.to_datetime(
    df_concat_electricty['referenceTime'], format='%Y-%m-%d %H:%M:%S'
)
```

Here we found the variance for each month across all the estudied years.

In [ ]:
```
# Extract the month from the referenceTime column
df_concat_electricty['month'] = df_concat_electricty['referenceTime'].dt.month

# Calculate the variance for each month across all years studied
variances = []
for month in range(1, 13):
    # Extract the data for the current month
    data = df_concat_electricty.loc[df_concat_electricty['month'] ==
                                    month, 'Net consumption of electricity Norway (Kwh)

    # Calculate the variance for the current month
    variance = np.var(data)

    # Add the variance to the list
    variances.append({'month': month, 'variance': variance})

# Convert the list of variances to a pandas DataFrame
```

```
variances_df_comsuption = pd.DataFrame(variances)
variances_df_comsuption

# Calculate the standard deviation for each month
variances_df_comsuption['std_dev'] = variances_df_comsuption['variance'].apply(
    np.sqrt)
```

Now we calculated the standard deviation, the mean and the uncertainty percentage for each month.

In [ ]:

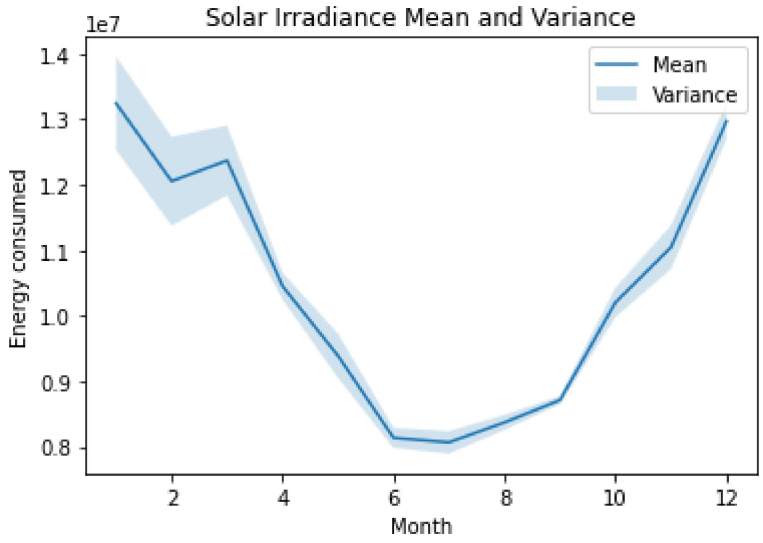
```
# Calculate the mean for each month
means = []
for month in range(1, 13):
    # Extract the data for the current month
    data = df_concat_electricty.loc[df_concat_electricty['month']
                                    == month, 'Net consumption of electricity Norway (K

    # Calculate the mean for the current month
    mean = np.mean(data)

    # Add the mean to the list
    means.append(mean)

# Add the means to the variances_df DataFrame
variances_df_comsuption['mean'] = means

# Calculate the percentage of uncertainty for each month
variances_df_comsuption['uncertainty_percent'] = (
    variances_df_comsuption['std_dev'] / variances_df_comsuption['mean']) * 100
variances_df_comsuption
```

Out[ ]:

	month	variance	std_dev	mean	uncertainty_percent
0	1	5.058895e+11	711259.120849	1.324151e+07	5.371434
1	2	4.580696e+11	676808.396757	1.205393e+07	5.614834
2	3	2.838018e+11	532730.553563	1.236874e+07	4.307074
3	4	4.154269e+10	203820.241461	1.045306e+07	1.949861
4	5	1.133747e+11	336711.553503	9.390407e+06	3.585697
5	6	2.355277e+10	153469.121226	8.137655e+06	1.885913
6	7	2.906457e+10	170483.338604	8.070346e+06	2.112466
7	8	1.299450e+10	113993.405801	8.373530e+06	1.361354
8	9	3.932495e+09	62709.606708	8.714422e+06	0.719607
9	10	5.072596e+10	225224.244763	1.020519e+07	2.206958
10	11	1.097584e+11	331298.092153	1.104392e+07	2.999823
11	12	7.495152e+10	273772.743979	1.296761e+07	2.111205

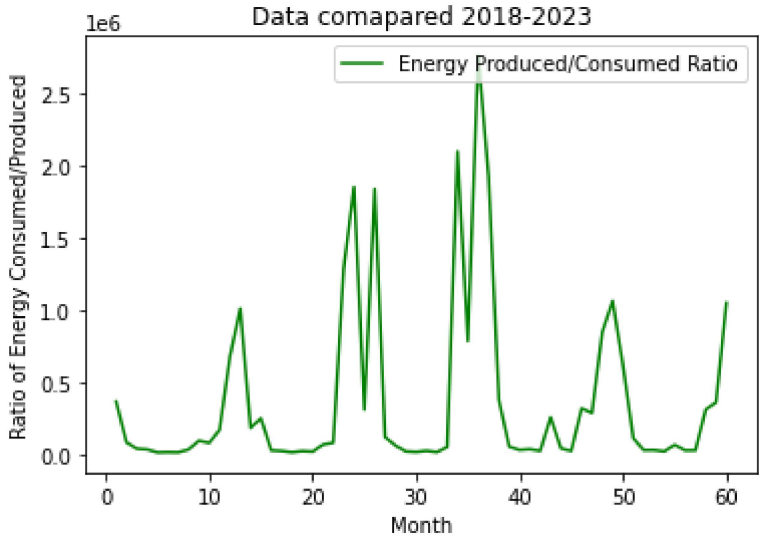
Uncertainty is less that than energy produced uncertainty. We guess that the consumption has less factors that can affect it.

In [ ]:
```
# Plot the mean solar irradiance for each month
plt.plot(variances_df_comsuption['month'],
         variances_df_comsuption['mean'], label='Mean')

# Plot the variance as a shaded area around the mean
plt.fill_between(variances_df_comsuption['month'], variances_df_comsuption['mean'] - va
                 variances_df_comsuption['mean'] + variances_df_comsuption['std_dev'],

# Set the title and axis labels
plt.title('Solar Irradiance Mean and Variance')
plt.xlabel('Month')
plt.ylabel('Energy consumed')

# Add a legend
plt.legend()

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



Comparison between Energy Produced and Energy consumed

In [ ]:
```
# Calculate the ratio of energy produced to energy consumed for each month
ratio = df_concat_electricty["Net consumption of electricity Norway (Kwh)"] / df_monthl


# Plot the ratio
plt.plot(range(1, 61), ratio, color='green',
         label='Energy Produced/Consumed Ratio')


# Add labels and legend
plt.title("Data comapared 2018-2023")
plt.xlabel('Month')
plt.ylabel('Ratio of Energy Consumed/Produced')
plt.legend()

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



Energy comsuption has less variance than enery produced since less volatile factors affects it. Climate changes will affect every year more making more difficult to predict the energy produced form solar panels. We can see from the graph that even on 2021 there was a peak , a pattern aorund of 1 million of solar panels are required this can be less if we use bigger and more efficient solar panels.

Also, it is important to mention that only data from one station that is located in Bergen was used for this analysis. Having said this, values fo solar panels can also be less if we would have been able to acces to information of zones where solar irradiance can be higher. Also some energy can be transported from counties that receive more solar irradiance to the ones that receive less.

Historical data it is really important to perform predictions since can help us to make narrower our action-frame-windows. Here we used Random Forest and we got quite accurate predictions for 2023, but it is important to have in mind that there are nowdays some ohter algortihms that take into account different factors make them more accurate like XGBOOST.

Kriging input data

Ground temperature and solar irradiation are closely related as solar irradiation is one of the primary factors that influences ground temperature. Solar irradiation, or the amount of solar radiation that reaches the Earth's surface, varies depending on the time of day, season, latitude, altitude, and weather conditions. When solar radiation reaches the Earth's surface, it is either absorbed, reflected, or transmitted through the atmosphere. The absorbed radiation warms the Earth's surface and leads to an increase in ground temperature.

In this section we will use temperature readings from 37 major cities across Europe and the Kriging technique to populate the rest of the map with extrapolated temperature values.

First we will conduct the full Kriging sequence manually, as taught in the course, and compare to results using imported kriging function from pykrige.

In [ ]:
```
data = pd.read_csv('europe_city_temperature_2020_05_01.csv')
data.head()
```

Out[ ]:

	City	latitude	longitude	temperature
0	Tirana	41.3275	19.8189	16.6
1	Vienna	48.2082	16.3738	14.8
2	Minsk	53.9045	27.5615	7.8
3	Brussels	50.8503	4.3517	10.6
4	Sofia	42.6977	23.3219	13.8

In [ ]:
```
data.describe()
```

Out[ ]:

	latitude	longitude	temperature
count	37.000000	37.000000	37.000000
mean	49.180862	11.345205	12.170270
std	6.251907	12.613102	3.945311
min	38.722300	-21.942600	4.500000
25%	44.786600	2.352200	9.300000
50%	48.148600	12.496400	11.600000
75%	53.349800	20.448900	15.100000
max	64.146600	37.617300	20.200000

Original data file found on kaggle (Ref. 3) contained temperature readings for many cities around the world from 1995 to 2020. The file was filtered down to only containing recordings for a single

day, 1st of May 2020, and for European cities only. The coordinates for each city was appended to the excel-file as the original file only contained city names.

In [ ]:

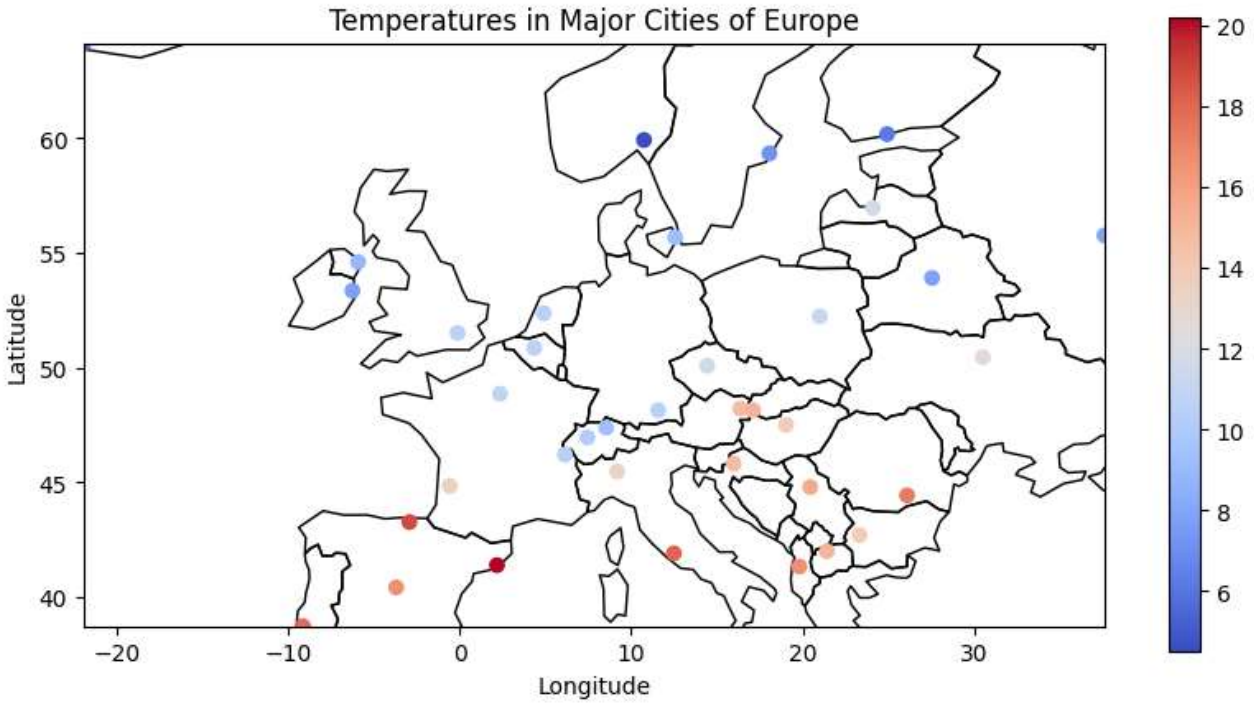
```
# Unpack data
X_cord_data = data['longitude']
Y_cord_data = data['latitude']
temperature_data = data['temperature']

# Extract min and max vaules of input coordinates
x_min = np.min(X_cord_data)
x_max = np.max(X_cord_data)
y_min = np.min(Y_cord_data)
y_max = np.max(Y_cord_data)
```

In [ ]:

```
# Visalize initial data
fig, ax = plt.subplots(figsize=(10, 10))

# Load the shapefile of Europe
europe = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

# Filter out all other continents
europe = europe[europe.continent == 'Europe']

# Plot the map of Europe
europe.plot(ax=ax, edgecolor='black', facecolor='none')

# Plot the initial data
plt.scatter(X_cord_data, Y_cord_data, c=temperature_data, cmap='coolwarm')
ax.set_title('Temperatures in Major Cities of Europe')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.colorbar(shrink = 0.5)
```

Out[ ]:

```
<matplotlib.colorbar.Colorbar at 0x2110d226fd0>
```



Manual Kriging on Grid

In [ ]:

```
# Define Length of x and y dimensions along with dimensions of each grid cell
length_x = x_max - x_min
length_y = y_max - y_min
grid_dim_x = 0.25
grid_dim_y = 0.25

# Number of cells in x and y direction
Nx = np.ceil(length_x / grid_dim_x)
Ny = np.ceil(length_y / grid_dim_y)

# Calculate the cell index in the x direction for each datum location
ix_data = np.ceil((X_cord_data - x_min) / grid_dim_x)
ix_data[ix_data==0] = 1

# Calculate the cell index in the y direction for each datum location
iy_data = np.ceil((Y_cord_data - y_min) / grid_dim_y)
iy_data[iy_data==0] = 1

# Define index location
i_loc_data = np.ceil((iy_data)*Nx + ix_data)
```

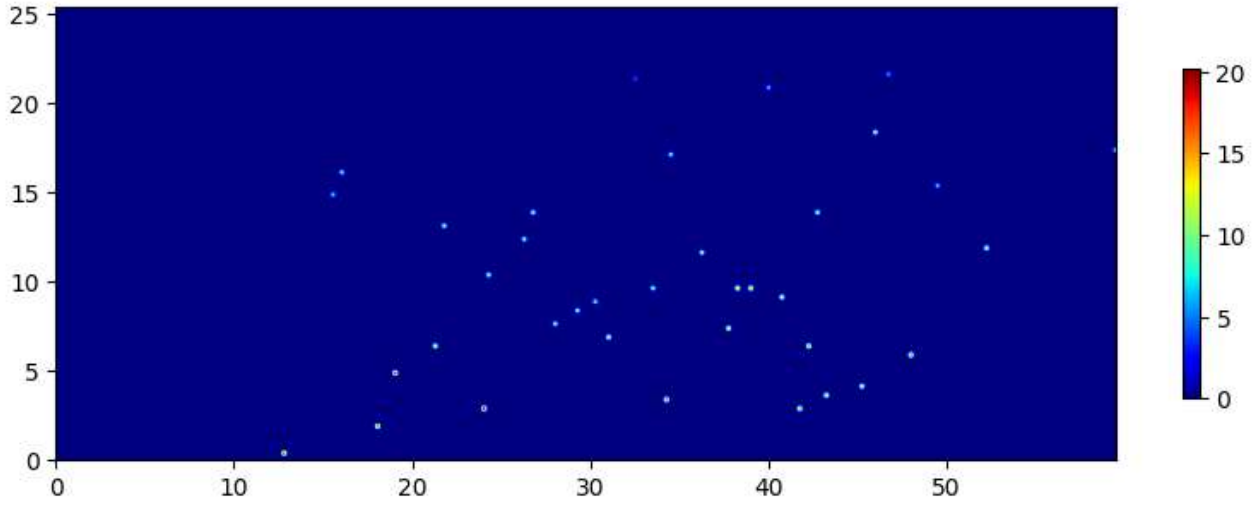
In [ ]:

```
# Defining Kriging grid
temperature_mean_grid = np.zeros((int(Ny),int(Nx)))
temperature_var_grid = np.zeros((int(Ny),int(Nx)))

cells_all = temperature_mean_grid.size + 1

# Filing occupied grids using data and (average of the data in case of mutiple values i
for cell_number in range(1, cells_all):
    ny = np.ceil(cell_number/Nx)
    nx = cell_number-(ny-1)*Nx
```

```
    temperature_data_cell = temperature_data[i_loc_data==cell_number]
    if temperature_data_cell.size != 0:
        temperature_mean_grid[int(ny)-1, int(nx)-1] = np.mean(temperature_data_cell)
```

In [ ]:
```
plt.figure(figsize=(10,5))
plt.imshow(temperature_mean_grid,extent=[0,length_x,length_y,0],cmap='jet')
plt.gca().invert_yaxis()
plt.colorbar(shrink = 0.5)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x2110d2bdd90>
```



Generate variogram

In [ ]:
```
# Set up matrices for calculating distances
X = np.asmatrix(X_cord_data).T
Y = np.asmatrix(Y_cord_data).T
T = np.asmatrix(temperature_data).T

n_points = T.size
ones_vec = np.asmatrix((np.ones(n_points)))

X_mat = X*ones_vec
Y_mat = Y*ones_vec

dist = np.power((np.power((X-X.T), 2)+np.power((Y-Y.T), 2)), 0.5)
dist.max()
```

Out[ ]:
```
60.14804412988007
```

In [ ]:
```
T_mat = T*ones_vec
Square_diff = np.power((T_mat - T_mat.T), 2)
```

In [ ]:
```
# The product of lag size and number of lags should be close to the maximum distance in
lag_size = 2
n_lags = 30
tol = 0.5*lag_size

# Creating semi variogram
```

```
Semi_variogram = np.zeros(n_lags+1)
for lags in range(1,n_lags+1):
    ul = lag_size*lags+tol
    ll = lag_size*lags-tol
    index = np.argwhere((dist<=ul) & (dist>=ll))
    Nh = index.shape[0]
    Semi_variogram[lags] = np.sum(Square_diff[index[:,0],index[:,1]])/(2*Nh)
```

In [ ]:
```
Semi_variogram
```

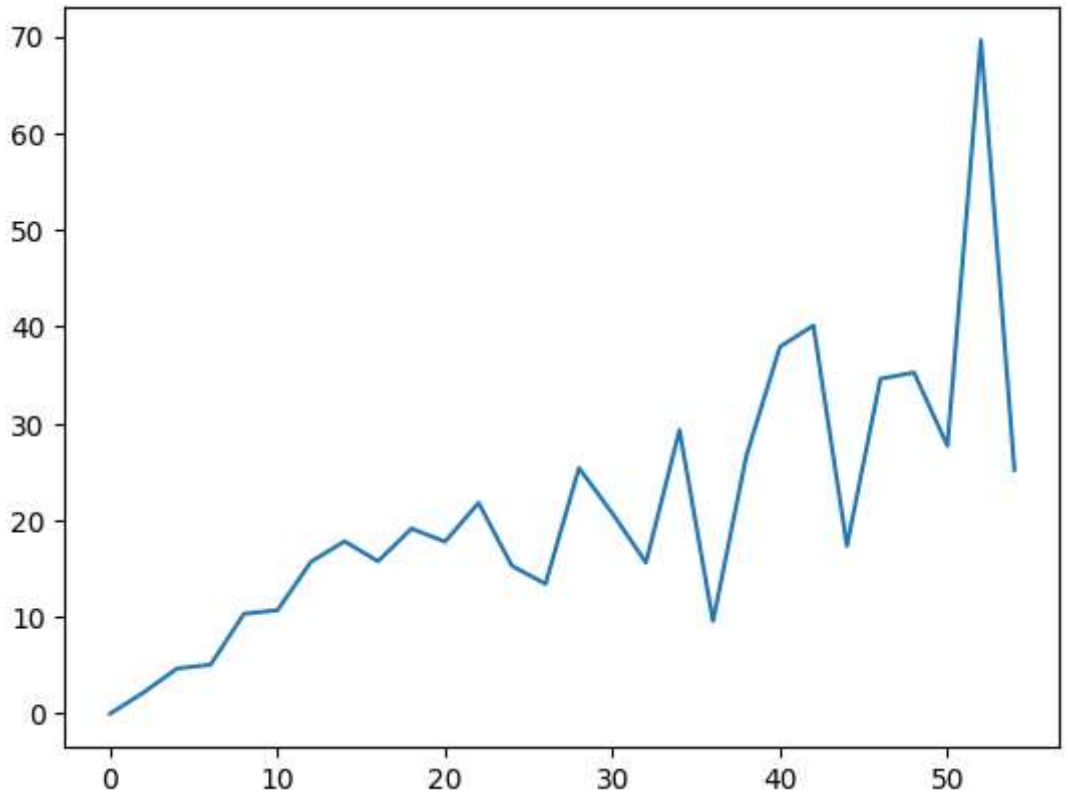
Out[ ]:
```
array([ 0.        ,  2.19722222,  4.66883333,  5.06556818, 10.33194444,
       10.69844828, 15.70579365, 17.82193548, 15.75122449, 19.13081633,
       17.77755814, 21.78894737, 15.27296296, 13.41289474, 25.38416667,
       20.70068182, 15.61846154, 29.31607143,  9.56      , 26.68125   ,
       37.9175    , 40.117     , 17.341     , 34.61      , 35.26833333,
       27.695     , 69.62      , 25.205     ,         nan,         nan,
        2.88      ])
```

The nan values will not work in the following steps, so we just remove them. The distance is so large at the nan values that they are not relevant anyway.

In [ ]:
```
Semi_variogram = Semi_variogram[:-3]
```

In [ ]:
```
# Plotting variogram
lag_dim = np.arange(0,lag_size*(n_lags+1),lag_size)[:-3]
plt.plot(lag_dim, Semi_variogram)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x2110d2c9610>]
```



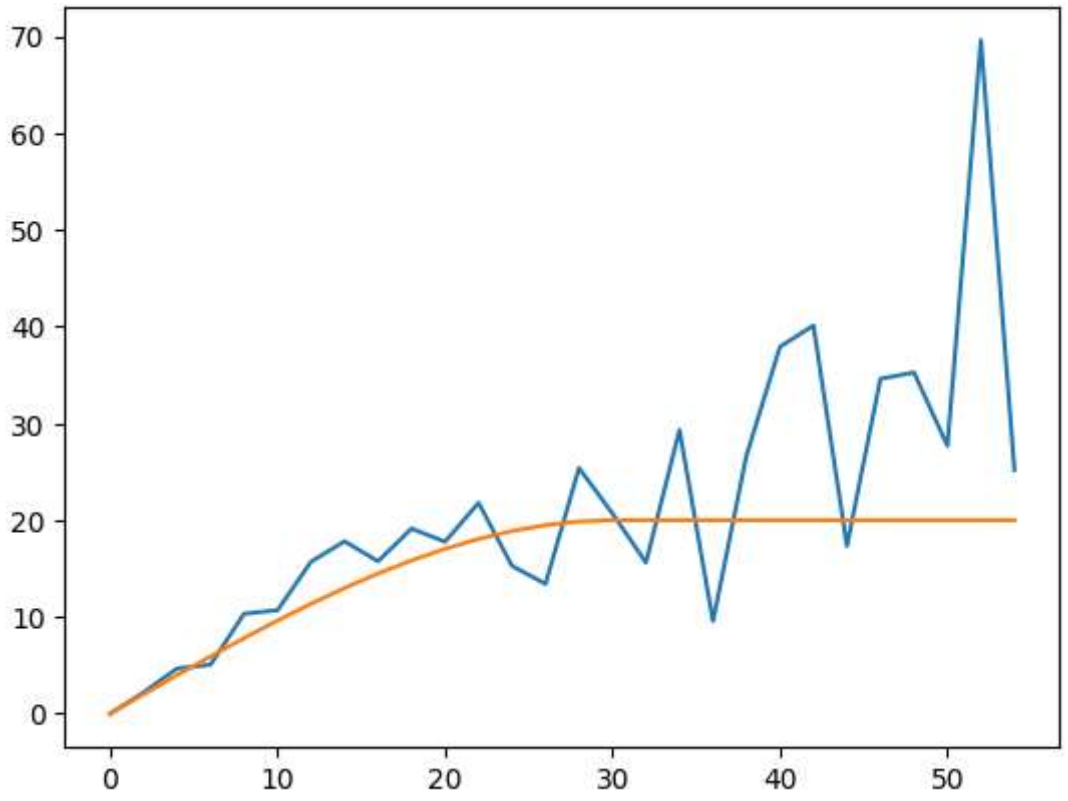
Variogram fitting

In [ ]:
```
# p0 parameters from visual inspection of above graph:
rang = 30
sill = 20
nugget = 0
```

In [ ]:
```
# Using spherical variogram representation
def sph_var(h,rang,sill,nugget):
    gamma = nugget + sill*(1.5*(h/rang)-0.5*np.power((h/rang),3) )
    index = np.argwhere(h>rang)
    if nugget < 0:
        nugget = 0
    if index.shape[1] > 1:
        gamma[index[:,0],index[:,1]] = nugget + sill
    else:
        gamma[index] = nugget + sill
    return gamma
```

In [ ]:
```
Sph_var = sph_var(lag_dim, rang, sill, nugget)
plt.plot(lag_dim, Semi_variogram)
plt.plot(lag_dim, Sph_var)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x2110d6c4d10>]
```



Kriging for cells with no data

In [ ]:
```
# Identify unknown grids where we don't have data points
unknown_cells = np.argwhere(temperature_mean_grid==0) + 1
unknown_cells
```

Out[ ]:
```
array([[  1,   1],
       [  1,   2],
       [  1,   3],
       ...,
       [102, 237],
       [102, 238],
       [102, 239]], dtype=int64)
```

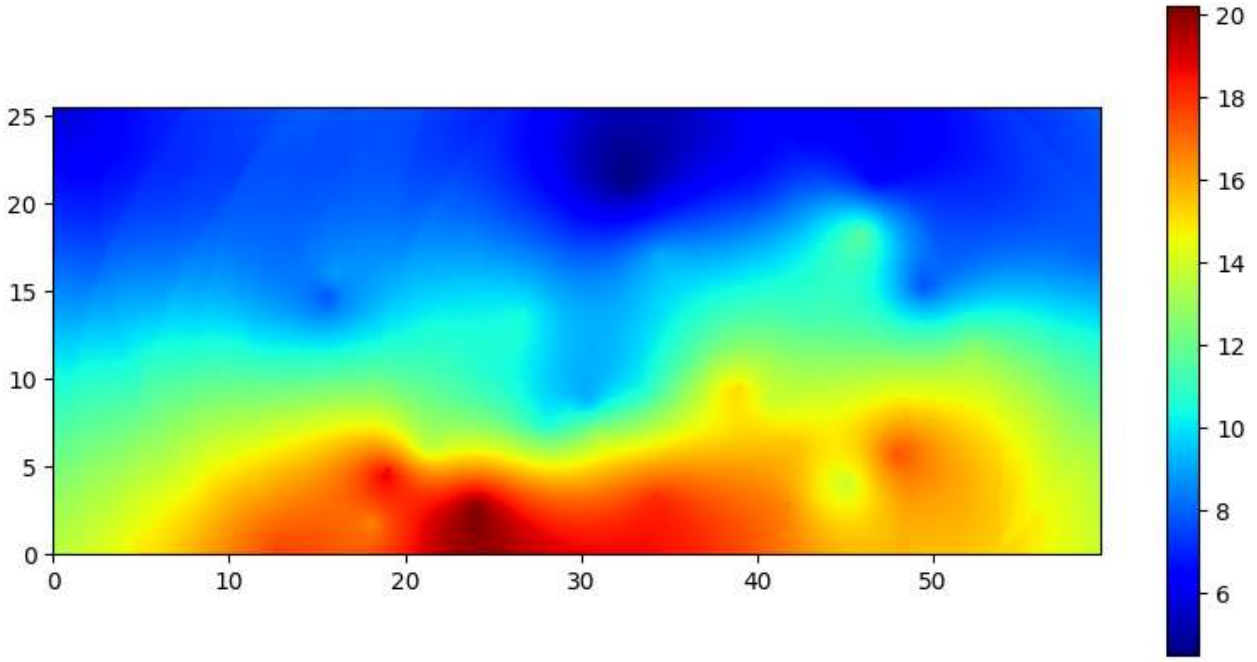
In [ ]:
```
# Finding number of unknown cells
n_uc = unknown_cells.shape[0]
n_uc
```

Out[ ]:
```
24342
```

In [ ]:
```
# Define minimum distance for including data in estimate
dist_near = 30

# Populate unknown cells
for i_uc in range(n_uc):
    # Calculate coordinates of unknown cell
    ny_uk = unknown_cells[i_uc, 0]
    nx_uk = unknown_cells[i_uc, 1]
    x_cord_uk = x_min + ((nx_uk-1) + nx_uk)*grid_dim_x/2
    y_cord_uk = y_min + ((ny_uk-1) + ny_uk)*grid_dim_y/2

    # Computing distance of unknown with known points and known points with themselves
    dist_UD = np.sqrt(np.square(X_cord_data-x_cord_uk) + np.square(Y_cord_data-y_cord_u
    X_known_near = X_cord_data[dist_UD<dist_near]
    Y_known_near = Y_cord_data[dist_UD<dist_near]
    X_known_near = np.array([X_known_near])
    Y_known_near = np.array([Y_known_near])
    dist_DD_near = np.sqrt(np.square(X_known_near-X_known_near.T) + np.square(Y_known_n
    dist_UD_near = np.array([dist_UD[dist_UD<dist_near]]).T

    # Computing weights by using covariance obtained from gamma known and unknown
    gamma_DD = sph_var(dist_DD_near,rang,sill,nugget)
    C_DD = sill-(gamma_DD-nugget)
    gamma_UD = sph_var(dist_UD_near,rang,sill,nugget)
    C_UD = sill-(gamma_UD-nugget)
    lamb = np.dot(np.linalg.inv(C_DD), C_UD)

    # Computing kriging mean and variance
    temperature_data_near = temperature_data[dist_UD<dist_near]
    temperature_data_near = np.array([temperature_data_near]).T
    temperature_mean_grid[int(ny_uk)-1,int(nx_uk)-1] = np.dot(lamb.T, (temperature_data
    temperature_var_grid[int(ny_uk)-1,int(nx_uk)-1] = sill - np.dot(lamb.T, C_UD)
```

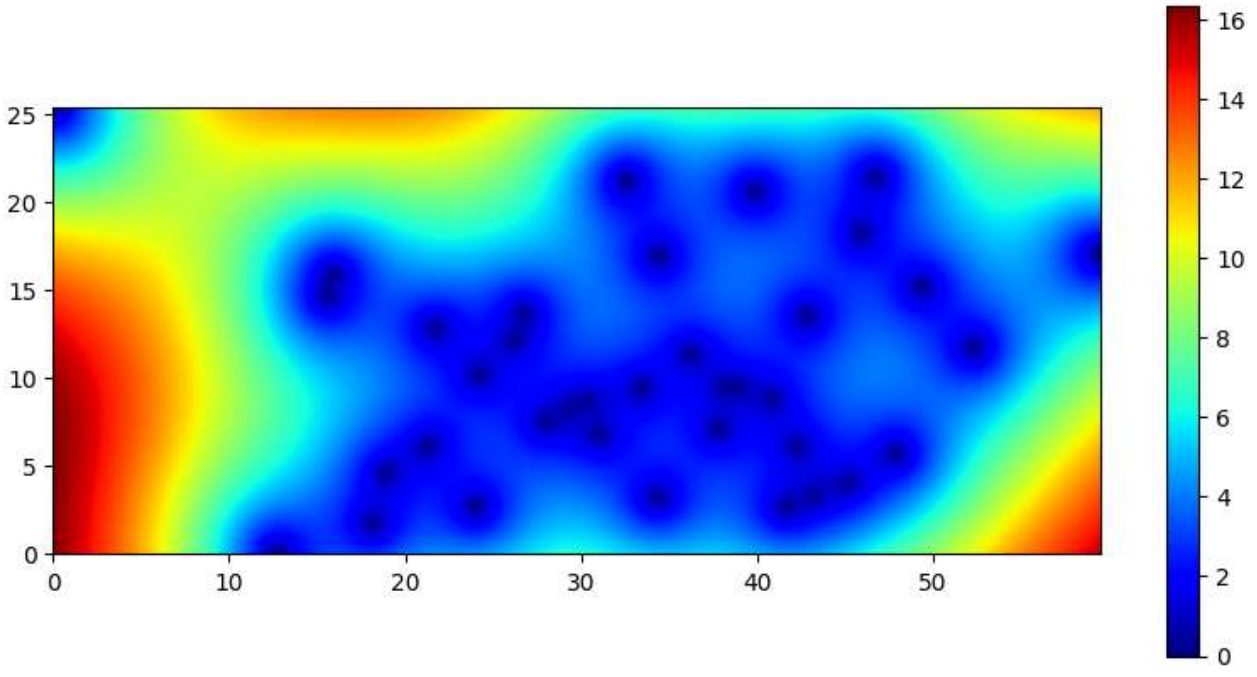
In [ ]:
```
# Plot main result on image grid
plt.figure(figsize=(10, 10))
plt.imshow(temperature_mean_grid, extent=[0,length_x,length_y,0],cmap='jet')
plt.gca().invert_yaxis()
plt.colorbar(shrink = 0.5)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x2110d482550>
```



In [ ]:
```
# Plot of variance for each grid point
plt.figure(figsize=(10,10))
plt.imshow(temperature_var_grid,extent=[0,length_x,length_y,0],cmap='jet')
plt.gca().invert_yaxis()
plt.colorbar(shrink = 0.5)
```

Out[ ]: <matplotlib.colorbar.Colorbar at 0x2110d572610>



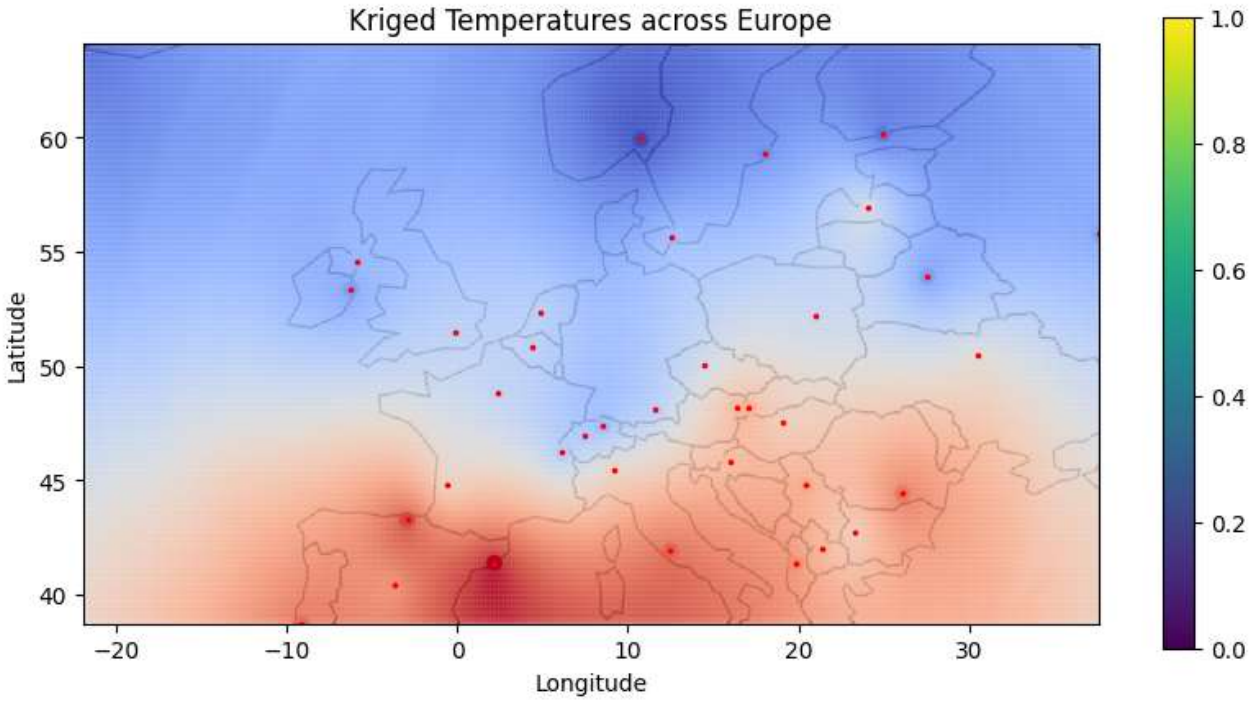
In [ ]:
```
# Convert main results to scatterplot on coordinate grid
x = np.arange(x_min + 0.5*grid_dim_x, x_max + 0.5*grid_dim_x, grid_dim_x)
y = np.arange(y_min + 0.5*grid_dim_y, y_max + 0.5*grid_dim_y, grid_dim_y)

grid_x, grid_y = np.meshgrid(x, y)
grid_x = grid_x.ravel()
```

```
grid_y = grid_y.ravel()
t_m_g = temperature_mean_grid.ravel()
```

In [ ]:

```
# Final plot including overlay of map of Europe
fig, ax = plt.subplots(figsize=(10, 5))

# Load the shapefile of Europe
europe = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

# Filter out all other continents
europe = europe[europe.continent == 'Europe']

# Plot the map of Europe
europe.plot(ax=ax, edgecolor='black', facecolor='none')

# Plot Kriged data
plt.scatter(grid_x, grid_y, c= t_m_g, cmap='coolwarm', alpha = 0.2)

# Plot the initial data
plt.scatter(X_cord_data, Y_cord_data, c=temperature_data, cmap='coolwarm')
plt.scatter(X_cord_data, Y_cord_data, marker='.', color='red', s = 10 )

plt.title('Kriged Temperatures across Europe')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.colorbar()
```

Out[ ]: <matplotlib.colorbar.Colorbar at 0x2110d7950d0>



Kriging by use of pykrige

In [ ]:

```
# Create a geodataframe with latitude and longitude
geometry = [Point(xy) for xy in zip(X_cord_data, Y_cord_data)]
crs = {'init': 'epsg:4326'}
geo_data = gpd.GeoDataFrame(data, crs=crs, geometry=geometry)

# Define the grid for interpolation
nx, ny = (200, 100)
x = np.linspace(x_min, x_max, nx)
y = np.linspace(y_min, y_max, ny)
grid_x, grid_y = np.meshgrid(x, y)
```

In [ ]:

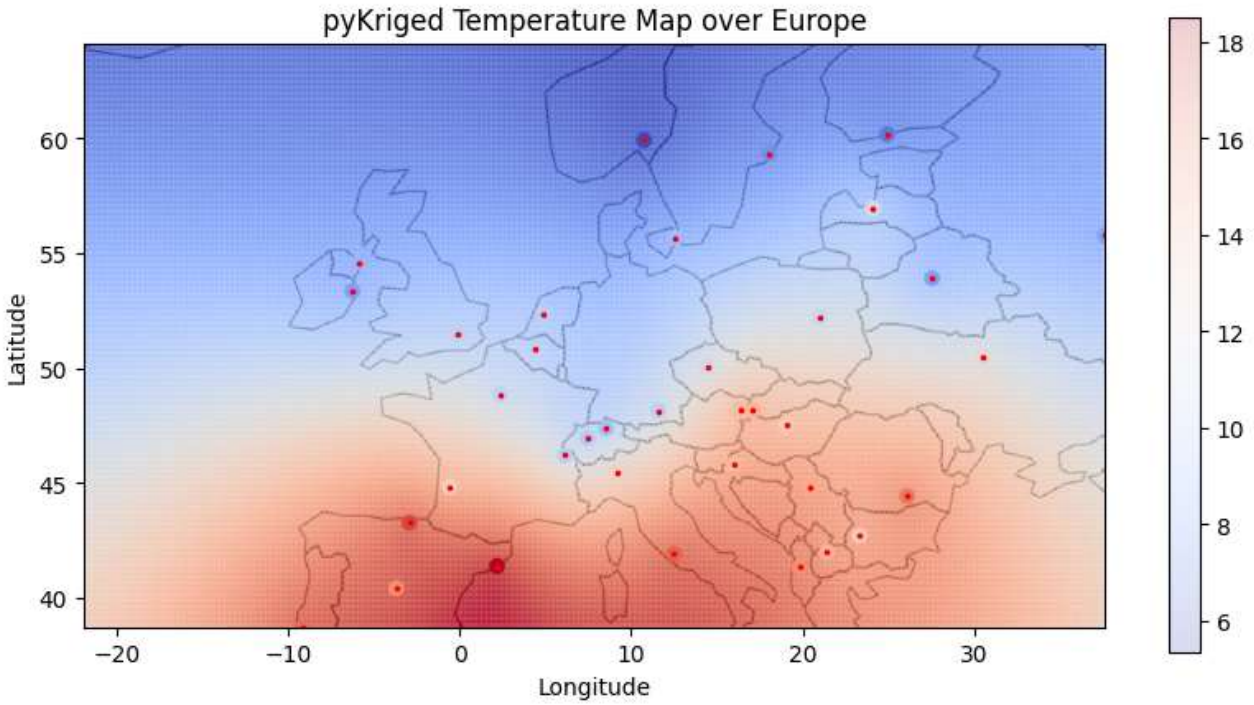
```
# Interpolate the data using ordinary kriging
OK = OrdinaryKriging(geo_data['longitude'], geo_data['latitude'], geo_data['temperature
z, ss = OK.execute('grid', x, y)

# Reshape the interpolated values
z = z.reshape(grid_x.shape)

# Create a geodataframe for the interpolated values
interpolated_data = pd.DataFrame(z.ravel(), columns=['temperature'])
interpolated_data['longitude'] = grid_x.ravel()
interpolated_data['latitude'] = grid_y.ravel()
geometry = [Point(xy) for xy in zip(interpolated_data['longitude'], interpolated_data['
interpolated_data = gpd.GeoDataFrame(interpolated_data, crs=crs, geometry=geometry)
```

In [ ]:

```
# Plot final results from pykrige
fig, ax = plt.subplots(figsize=(10, 10))

# Plot the map of Europe
europe.plot(ax=ax, edgecolor='black', facecolor='none')

# Plot the interpolated data
plt.scatter(interpolated_data['longitude'], interpolated_data['latitude'], c= interpola
plt.colorbar(shrink = 0.5)

# Add the original data points as red circles
plt.scatter(geo_data['longitude'], geo_data['latitude'], c= geo_data['temperature'], cm
plt.scatter(geo_data['longitude'], geo_data['latitude'], marker='.', color='red', s = 1
# geo_data.plot(ax=ax, marker='o', color='red', markersize=10)

# Set the title and axis labels
ax.set_title('pyKriged Temperature Map over Europe')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
```

Out[ ]:

```
(38.7223, 64.1466)
```



Comparing the results from the home-made Kriging results to the pykrige ones, we can see that qualitatively we obtain a similar result. If anything the manual kriging result seems to adhere more to the priciple of keeping the result for the "data" nodes the same as for the original data.

Performance wise the manual kriging took about a minute to execute vs. about a second to execute the pykrige function at similar resolutions.

The relationship between ground temperature and solar irradiation is complex and can be affected by several factors, including surface characteristics, soil moisture content, and thermal conductivity. In our case - across vast geological distances, topography, more importantly between sea and land, the temperatures would not be expected to be straight interpolations between surrounding points. We believe Kriging is better utilized in grids where such complex behavours are not present.

The preceding presentation shows that the Kriging technique can be adapted for use on geographic scales, however, choosing a grid with as few internal variables as possible would improve the validity of the results.

Reference

1. Solar irradiance Norway data. https://frost.met.no/index.html
2. Electricty Comsuption in Norway data. https://www.ssb.no/en/statbank/table/12824/
3. Temperature data file. https://www.kaggle.com/datasets/sudalairajkumar/daily-temperature-of-major-cities?resource=download