In [20]:
```
import pandas as pd # to use or generate dataframe
import seaborn as sns # for pretty plots
import numpy as np # for matrix manipulation
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import sklearn.cluster as skc
```

In [22]:
```
pwd
```

Out[22]:
```
'C:\\Users\\emper\\OneDrive\\Desktop\\DSTI\\Pratical-Intro-to-Data-Science\\Fin
al-Project'
```

In [24]:
```
%cd C:\\Users\\emper\\OneDrive\\Desktop\\DSTI\\Pratical-Intro-to-Data-Science\\F
```

```
C:\Users\emper\OneDrive\Desktop\DSTI\Pratical-Intro-to-Data-Science\Final-Project
```

In [26]:
```
weather_station = pd.read_csv("WS_Master_Data_with_Long_and_Lat_P.csv")
```

In [28]:
```
weather_station.head(100)
```

Out[28]:

	year	month	tmax	tmin	af	rain	sun	station	lon	lat	Sun_Data_So
0	1941	Jan	NaN	NaN	NaN	74.7	NaN	aberporth	-4.57	52.14	Mis
1	1941	Feb	NaN	NaN	NaN	69.1	NaN	aberporth	-4.57	52.14	Mis
2	1941	Mar	NaN	NaN	NaN	76.2	NaN	aberporth	-4.57	52.14	Mis
3	1941	Apr	NaN	NaN	NaN	33.7	NaN	aberporth	-4.57	52.14	Mis
4	1941	May	NaN	NaN	NaN	51.3	NaN	aberporth	-4.57	52.14	Mis
...	...	...	...	...	...	...	...	...	...	...	
95	1948	Dec	8.5	4.2	NaN	175.9	87.2	aberporth	-4.57	52.14	Campbell St
96	1949	Jan	8.6	5.0	NaN	44.0	60	aberporth	-4.57	52.14	Campbell St
97	1949	Feb	8.7	4.4	NaN	36.6	105	aberporth	-4.57	52.14	Campbell St
98	1949	Mar	8.7	3.3	NaN	44.4	158.8	aberporth	-4.57	52.14	Campbell St
99	1949	Apr	NaN	6.7	NaN	74.8	161.9	aberporth	-4.57	52.14	Campbell St

100 rows × 11 columns

In [64]:
```
weather_station.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39183 entries, 0 to 39182
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   year             39183 non-null  int64
 1   month            39183 non-null  object
 2   tmax             38257 non-null  float64
 3   tmin             38283 non-null  float64
 4   af               36858 non-null  float64
 5   rain             38312 non-null  float64
 6   sun              30145 non-null  object
 7   station          39183 non-null  object
 8   lon              39183 non-null  float64
 9   lat              39183 non-null  float64
 10  Sun_Data_Source  39183 non-null  object
dtypes: float64(6), int64(1), object(4)
memory usage: 3.3+ MB
```

Let's convert the sun column from object to float64

In [77]:
```
weather_station['sun'] = pd.to_numeric(weather_station['sun'], errors = 'coerce'
```

In [79]:
```
weather_station.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39183 entries, 0 to 39182
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   year             39183 non-null  int64
 1   month            39183 non-null  object
 2   tmax             38257 non-null  float64
 3   tmin             38283 non-null  float64
 4   af               36858 non-null  float64
 5   rain             38312 non-null  float64
 6   sun              29863 non-null  float64
 7   station          39183 non-null  object
 8   lon              39183 non-null  float64
 9   lat              39183 non-null  float64
 10  Sun_Data_Source  39183 non-null  object
dtypes: float64(7), int64(1), object(3)
memory usage: 3.3+ MB
```

Let's group the data by station to ensure clusturing by stations and not by observations/records

In [81]:
```
groupby_station = weather_station.groupby('station')
groupby_station
```

Out[81]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001A4E31A53D0>
```

Get only the numerical Columns

In [102…
```
numeric_cols = weather_station.select_dtypes(include='number').columns
numeric_cols
```

Out[102…
```
Index(['year', 'tmax', 'tmin', 'af', 'rain', 'sun', 'lon', 'lat'], dtype='objec
t')
```

In [104…
```
groupby_station_df = groupby_station[numeric_cols].mean().reset_index()
groupby_station_df.head(5)
```

Out[104…

	station	year	tmax	tmin	af	rain	sun	lor
0	aberporth	1982.000000	12.501738	7.269919	1.492537	76.175000	128.641955	-4.57
1	armagh	1938.000000	12.979003	5.695801	3.275591	68.913213	104.269217	-6.64
2	ballypatrick	1992.225634	11.438408	5.595798	2.582712	110.185442	106.663636	-6.15
3	bradford	1965.500000	12.347289	5.756471	3.728850	72.988864	104.586547	-1.77
4	braemar	1991.000000	10.586082	2.793170	8.626289	75.877158	98.446739	-3.39

Alternatively, this helps to focus on only the required columns

In [107…
```
key_columns= ["tmax", "tmin", "af", "rain", "sun"]
key_columns
```

Out[107…
```
['tmax', 'tmin', 'af', 'rain', 'sun']
```

In [109…
```
groupby_station[key_columns]
```

Out[109…
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001A4E36F24E0>
```

In [111…
```
groupby_station_df = groupby_station[key_columns].mean().reset_index()
groupby_station_df.head(5)
```

Out[111…

	station	tmax	tmin	af	rain	sun
0	aberporth	12.501738	7.269919	1.492537	76.175000	128.641955
1	armagh	12.979003	5.695801	3.275591	68.913213	104.269217
2	ballypatrick	11.438408	5.595798	2.582712	110.185442	106.663636
3	bradford	12.347289	5.756471	3.728850	72.988864	104.586547
4	braemar	10.586082	2.793170	8.626289	75.877158	98.446739

Please, note that I excluded all data from 2024 to 2024 because they are categorised as provisional since the full network quality control has not been carried out. The issues of quality is very important in data science because poor quality data can introduce baises that may skew the result of the analysis or lead to poor performance of the model or algorithms

In [113…
```
groupby_station_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37 entries, 0 to 36
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   station  37 non-null     object 
 1   tmax     37 non-null     float64
 2   tmin     37 non-null     float64
 3   af       37 non-null     float64
 4   rain     37 non-null     float64
 5   sun      37 non-null     float64
dtypes: float64(5), object(1)
memory usage: 1.9+ KB
```

Now, let's drop all the missing values

Dropping the missing value was as a purposive sampling techniques which ensure only rows with complete values are selected

In [203…
```
k_means_data = groupby_station_df.dropna()
k_means_data.head(5)
```

Out[203…

	station	tmax	tmin	af	rain	sun
0	aberporth	12.501738	7.269919	1.492537	76.175000	128.641955
1	armagh	12.979003	5.695801	3.275591	68.913213	104.269217
2	ballypatrick	11.438408	5.595798	2.582712	110.185442	106.663636
3	bradford	12.347289	5.756471	3.728850	72.988864	104.586547
4	braemar	10.586082	2.793170	8.626289	75.877158	98.446739

In [205…
```
k_means_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37 entries, 0 to 36
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   station  37 non-null     object 
 1   tmax     37 non-null     float64
 2   tmin     37 non-null     float64
 3   af       37 non-null     float64
 4   rain     37 non-null     float64
 5   sun      37 non-null     float64
dtypes: float64(5), object(1)
memory usage: 1.9+ KB
```

Let's drop the station and focus only on the numerical data

In [207…
```
key_columns= ["tmax", "tmin", "af", "rain", "sun"]
key_columns
```

Out[207…
```
['tmax', 'tmin', 'af', 'rain', 'sun']
```

In [209…

```
k_means_data = k_means_data[key_columns]
k_means_data
```

Out[209...

	tmax	tmin	af	rain	sun
0	12.501738	7.269919	1.492537	76.175000	128.641955
1	12.979003	5.695801	3.275591	68.913213	104.269217
2	11.438408	5.595798	2.582712	110.185442	106.663636
3	12.347289	5.756471	3.728850	72.988864	104.586547
4	10.586082	2.793170	8.626289	75.877158	98.446739
5	13.521507	8.451838	0.733456	90.062316	134.119367
6	14.377179	6.258205	3.557692	46.398413	125.000000
7	14.750000	7.117086	2.919065	98.087590	123.556054
8	14.621257	7.932528	1.703620	75.065769	138.594142
9	11.642314	4.886000	5.113523	149.432712	97.186254
10	12.456486	6.327869	2.387889	140.171778	100.801339
11	12.435495	4.900232	4.740591	54.619097	111.454594
12	14.002564	8.408590	1.410256	66.393974	155.045513
13	10.979394	3.471591	7.170238	135.456268	97.321491
14	15.001754	7.148136	2.912222	50.742325	128.976990
15	14.671393	5.957836	4.907960	70.251119	140.587595
16	9.515936	5.094808	3.123545	96.883885	89.172784
17	12.350498	5.028483	4.904229	57.455597	123.748259
18	13.120665	6.835047	2.550136	50.282797	136.754522
19	13.799279	7.246550	2.259947	49.262254	146.812060
20	11.998452	4.942452	4.402904	53.078520	108.657875
21	12.235641	4.877179	5.003846	79.275794	109.330292
22	13.987817	6.248415	3.696733	54.912963	128.803772
23	12.797179	6.167051	3.147436	99.617179	106.520641
24	13.088060	6.170746	3.522388	67.892966	114.824929
25	14.083617	6.276186	3.729633	60.280036	124.084691
26	13.528856	5.378607	4.927861	55.928205	115.005027
27	12.931175	6.260012	2.874700	67.059456	111.526163
28	14.400808	6.762788	3.177674	66.522101	137.346518
29	11.029617	5.391667	2.747778	100.347672	100.707821
30	13.788333	5.957179	3.944872	51.496429	114.447561
31	11.677261	6.791316	1.128917	99.551746	118.052393
32	13.059982	7.561325	1.573859	71.296237	133.177330

	tmax	tmin	af	rain	sun
33	13.316667	6.021104	3.600000	50.762554	128.026082
34	12.525241	6.193553	2.650685	50.419231	140.838462
35	10.533435	5.043617	3.557848	65.130833	104.699142
36	14.609831	6.176124	4.332865	60.768680	127.539709

Let's standardise/normalise the dataset

The ensures integration and consistency for effective data analysis

In [213…
```
scaler = StandardScaler()
```

In [215…
```
k_means_data = scaler.fit_transform(k_means_data)
k_means_data
```

Out[215…
```
array([[-0.28771425,  1.02418434, -1.26413314,  0.03057775,  0.58860283],
       [ 0.07195725, -0.31352893, -0.12006369, -0.24859811, -0.95250804],
       [-1.08905133, -0.39851256, -0.56463862,  1.33809261, -0.80110669],
       [-0.40410921, -0.26196972,  0.17076366, -0.09191168, -0.93244299],
       [-1.73137276, -2.78023524,  3.31313174,  0.01912736, -1.32066873],
       [ 0.48079457,  2.02860032, -1.75118639,  0.56446886,  0.93494466],
       [ 1.12563817,  0.16441238,  0.06094268, -1.11416864,  0.35831862],
       [ 1.40659966,  0.89430471, -0.34882314,  0.8729966 ,  0.26701656],
       [ 1.30957752,  1.58728126, -1.12869515, -0.01206608,  1.21788887],
       [-0.93538522, -1.00171164,  1.05921788,  2.8469348 , -1.40037036],
       [-0.32181679,  0.22361384, -0.68964431,  2.49090271, -1.17178518],
       [-0.3376358 , -0.98961736,  0.81993169, -0.79812843, -0.49817   ],
       [ 0.84332404,  1.99184699, -1.31692745, -0.34544901,  2.25812432],
       [-1.43496906, -2.20370132,  2.37887867,  2.30961724, -1.3918192 ],
       [ 1.59632443,  0.92069123, -0.35321355, -0.94716906,  0.6097874 ],
       [ 1.34736062, -0.09084673,  0.92732188, -0.19716295,  1.34393671],
       [-2.53784684, -0.82426334, -0.21762131,  0.82672076, -1.90706952],
       [-0.40169087, -0.88062743,  0.92492772, -0.68908057,  0.27916986],
       [ 0.17871558,  0.65462313, -0.58554108, -0.96483536,  1.10156796],
       [ 0.6901263 ,  1.00432517, -0.77173622, -1.00406964,  1.73751546],
       [-0.6669963 , -0.95373776,  0.60326   , -0.85735517, -0.67500909],
       [-0.48824793, -1.00920746,  0.98884567,  0.14978625, -0.63249157],
       [ 0.83221029,  0.1560929 ,  0.15015625, -0.7868309 ,  0.59883466],
       [-0.06506659,  0.08694824, -0.20229221,  0.93180092, -0.81014841],
       [ 0.15414382,  0.0900883 ,  0.03829024, -0.28782102, -0.28506056],
       [ 0.90440623,  0.17969302,  0.17126582, -0.5804964 ,  0.30044279],
       [ 0.48633238, -0.58308559,  0.94009074, -0.74780041, -0.27367282],
       [ 0.03591389,  0.16594788, -0.37728897, -0.31986488, -0.49364459],
       [ 1.1434451 ,  0.59321578, -0.18289047, -0.34052324,  1.13900041],
       [-1.3971202 , -0.57198722, -0.45872687,  0.95988435, -1.17769844],
       [ 0.68187746, -0.0914045 ,  0.30937061, -0.9181779 , -0.3089219 ],
       [-0.90904941,  0.61745938, -1.49744485,  0.92928536, -0.080985  ],
       [ 0.13298424,  1.27182651, -1.21195459, -0.15698393,  0.87537878],
       [ 0.32642445, -0.03708041,  0.08808878, -0.94639135,  0.54966062],
       [-0.27000204,  0.10946971, -0.52102504, -0.95959025,  1.35979926],
       [-1.77104854, -0.86776593,  0.06104228, -0.39400985, -0.92532347],
       [ 1.30096716,  0.09465805,  0.55832075, -0.56171072,  0.51890678]])
```

Let's transform the k_means data into a dataframe

In [218…

```
# Assuming the scaled array is k_means_data
# and you want to use the same column names as the original DataFrame
scaled_k_means_data = pd.DataFrame(k_means_data, columns=key_columns)
scaled_k_means_data
```

Out[218...

	tmax	tmin	af	rain	sun
0	-0.287714	1.024184	-1.264133	0.030578	0.588603
1	0.071957	-0.313529	-0.120064	-0.248598	-0.952508
2	-1.089051	-0.398513	-0.564639	1.338093	-0.801107
3	-0.404109	-0.261970	0.170764	-0.091912	-0.932443
4	-1.731373	-2.780235	3.313132	0.019127	-1.320669
5	0.480795	2.028600	-1.751186	0.564469	0.934945
6	1.125638	0.164412	0.060943	-1.114169	0.358319
7	1.406600	0.894305	-0.348823	0.872997	0.267017
8	1.309578	1.587281	-1.128695	-0.012066	1.217889
9	-0.935385	-1.001712	1.059218	2.846935	-1.400370
10	-0.321817	0.223614	-0.689644	2.490903	-1.171785
11	-0.337636	-0.989617	0.819932	-0.798128	-0.498170
12	0.843324	1.991847	-1.316927	-0.345449	2.258124
13	-1.434969	-2.203701	2.378879	2.309617	-1.391819
14	1.596324	0.920691	-0.353214	-0.947169	0.609787
15	1.347361	-0.090847	0.927322	-0.197163	1.343937
16	-2.537847	-0.824263	-0.217621	0.826721	-1.907070
17	-0.401691	-0.880627	0.924928	-0.689081	0.279170
18	0.178716	0.654623	-0.585541	-0.964835	1.101568
19	0.690126	1.004325	-0.771736	-1.004070	1.737515
20	-0.666996	-0.953738	0.603260	-0.857355	-0.675009
21	-0.488248	-1.009207	0.988846	0.149786	-0.632492
22	0.832210	0.156093	0.150156	-0.786831	0.598835
23	-0.065067	0.086948	-0.202292	0.931801	-0.810148
24	0.154144	0.090088	0.038290	-0.287821	-0.285061
25	0.904406	0.179693	0.171266	-0.580496	0.300443
26	0.486332	-0.583086	0.940091	-0.747800	-0.273673
27	0.035914	0.165948	-0.377289	-0.319865	-0.493645
28	1.143445	0.593216	-0.182890	-0.340523	1.139000
29	-1.397120	-0.571987	-0.458727	0.959884	-1.177698
30	0.681877	-0.091404	0.309371	-0.918178	-0.308922
31	-0.909049	0.617459	-1.497445	0.929285	-0.080985
32	0.132984	1.271827	-1.211955	-0.156984	0.875379

	tmax	tmin	af	rain	sun
33	0.326424	-0.037080	0.088089	-0.946391	0.549661
34	-0.270002	0.109470	-0.521025	-0.959590	1.359799
35	-1.771049	-0.867766	0.061042	-0.394010	-0.925323
36	1.300967	0.094658	0.558321	-0.561711	0.518907

In [220…
```
scaled_k_means_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37 entries, 0 to 36
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   tmax    37 non-null     float64
 1   tmin    37 non-null     float64
 2   af      37 non-null     float64
 3   rain    37 non-null     float64
 4   sun     37 non-null     float64
dtypes: float64(5)
memory usage: 1.6 KB
```

Let's drop the missing values again

Dropping the missing value was as a purposive sampling techniques which ensure only raws with complete values are selected

In [224…
```
scaled_k_means_data = scaled_k_means_data.dropna()
scaled_k_means_data
```

Out[224...

	tmax	tmin	af	rain	sun
0	-0.287714	1.024184	-1.264133	0.030578	0.588603
1	0.071957	-0.313529	-0.120064	-0.248598	-0.952508
2	-1.089051	-0.398513	-0.564639	1.338093	-0.801107
3	-0.404109	-0.261970	0.170764	-0.091912	-0.932443
4	-1.731373	-2.780235	3.313132	0.019127	-1.320669
5	0.480795	2.028600	-1.751186	0.564469	0.934945
6	1.125638	0.164412	0.060943	-1.114169	0.358319
7	1.406600	0.894305	-0.348823	0.872997	0.267017
8	1.309578	1.587281	-1.128695	-0.012066	1.217889
9	-0.935385	-1.001712	1.059218	2.846935	-1.400370
10	-0.321817	0.223614	-0.689644	2.490903	-1.171785
11	-0.337636	-0.989617	0.819932	-0.798128	-0.498170
12	0.843324	1.991847	-1.316927	-0.345449	2.258124
13	-1.434969	-2.203701	2.378879	2.309617	-1.391819
14	1.596324	0.920691	-0.353214	-0.947169	0.609787
15	1.347361	-0.090847	0.927322	-0.197163	1.343937
16	-2.537847	-0.824263	-0.217621	0.826721	-1.907070
17	-0.401691	-0.880627	0.924928	-0.689081	0.279170
18	0.178716	0.654623	-0.585541	-0.964835	1.101568
19	0.690126	1.004325	-0.771736	-1.004070	1.737515
20	-0.666996	-0.953738	0.603260	-0.857355	-0.675009
21	-0.488248	-1.009207	0.988846	0.149786	-0.632492
22	0.832210	0.156093	0.150156	-0.786831	0.598835
23	-0.065067	0.086948	-0.202292	0.931801	-0.810148
24	0.154144	0.090088	0.038290	-0.287821	-0.285061
25	0.904406	0.179693	0.171266	-0.580496	0.300443
26	0.486332	-0.583086	0.940091	-0.747800	-0.273673
27	0.035914	0.165948	-0.377289	-0.319865	-0.493645
28	1.143445	0.593216	-0.182890	-0.340523	1.139000
29	-1.397120	-0.571987	-0.458727	0.959884	-1.177698
30	0.681877	-0.091404	0.309371	-0.918178	-0.308922
31	-0.909049	0.617459	-1.497445	0.929285	-0.080985
32	0.132984	1.271827	-1.211955	-0.156984	0.875379

	tmax	tmin	af	rain	sun
33	0.326424	-0.037080	0.088089	-0.946391	0.549661
34	-0.270002	0.109470	-0.521025	-0.959590	1.359799
35	-1.771049	-0.867766	0.061042	-0.394010	-0.925323
36	1.300967	0.094658	0.558321	-0.561711	0.518907

Let's add another column for avaerage temparature

In [227…
```
scaled_k_means_data['AvgTemp'] = (scaled_k_means_data['tmax'] + scaled_k_means_d
```

Using average values for temperature helps to reduce the influence of extreme values in tmax and tmin

In [230…
```
scaled_k_means_data
```

Out[230...

	tmax	tmin	af	rain	sun	AvgTemp
0	-0.287714	1.024184	-1.264133	0.030578	0.588603	0.368235
1	0.071957	-0.313529	-0.120064	-0.248598	-0.952508	-0.120786
2	-1.089051	-0.398513	-0.564639	1.338093	-0.801107	-0.743782
3	-0.404109	-0.261970	0.170764	-0.091912	-0.932443	-0.333039
4	-1.731373	-2.780235	3.313132	0.019127	-1.320669	-2.255804
5	0.480795	2.028600	-1.751186	0.564469	0.934945	1.254697
6	1.125638	0.164412	0.060943	-1.114169	0.358319	0.645025
7	1.406600	0.894305	-0.348823	0.872997	0.267017	1.150452
8	1.309578	1.587281	-1.128695	-0.012066	1.217889	1.448429
9	-0.935385	-1.001712	1.059218	2.846935	-1.400370	-0.968548
10	-0.321817	0.223614	-0.689644	2.490903	-1.171785	-0.049101
11	-0.337636	-0.989617	0.819932	-0.798128	-0.498170	-0.663627
12	0.843324	1.991847	-1.316927	-0.345449	2.258124	1.417586
13	-1.434969	-2.203701	2.378879	2.309617	-1.391819	-1.819335
14	1.596324	0.920691	-0.353214	-0.947169	0.609787	1.258508
15	1.347361	-0.090847	0.927322	-0.197163	1.343937	0.628257
16	-2.537847	-0.824263	-0.217621	0.826721	-1.907070	-1.681055
17	-0.401691	-0.880627	0.924928	-0.689081	0.279170	-0.641159
18	0.178716	0.654623	-0.585541	-0.964835	1.101568	0.416669
19	0.690126	1.004325	-0.771736	-1.004070	1.737515	0.847226
20	-0.666996	-0.953738	0.603260	-0.857355	-0.675009	-0.810367
21	-0.488248	-1.009207	0.988846	0.149786	-0.632492	-0.748728
22	0.832210	0.156093	0.150156	-0.786831	0.598835	0.494152
23	-0.065067	0.086948	-0.202292	0.931801	-0.810148	0.010941
24	0.154144	0.090088	0.038290	-0.287821	-0.285061	0.122116
25	0.904406	0.179693	0.171266	-0.580496	0.300443	0.542050
26	0.486332	-0.583086	0.940091	-0.747800	-0.273673	-0.048377
27	0.035914	0.165948	-0.377289	-0.319865	-0.493645	0.100931
28	1.143445	0.593216	-0.182890	-0.340523	1.139000	0.868330
29	-1.397120	-0.571987	-0.458727	0.959884	-1.177698	-0.984554
30	0.681877	-0.091404	0.309371	-0.918178	-0.308922	0.295236
31	-0.909049	0.617459	-1.497445	0.929285	-0.080985	-0.145795
32	0.132984	1.271827	-1.211955	-0.156984	0.875379	0.702405

	tmax	tmin	af	rain	sun	AvgTemp
33	0.326424	-0.037080	0.088089	-0.946391	0.549661	0.144672
34	-0.270002	0.109470	-0.521025	-0.959590	1.359799	-0.080266
35	-1.771049	-0.867766	0.061042	-0.394010	-0.925323	-1.319407
36	1.300967	0.094658	0.558321	-0.561711	0.518907	0.697813

Let's do simple EDA**

In [233…
```
scaled_k_means_data.describe()
```

Out[233…

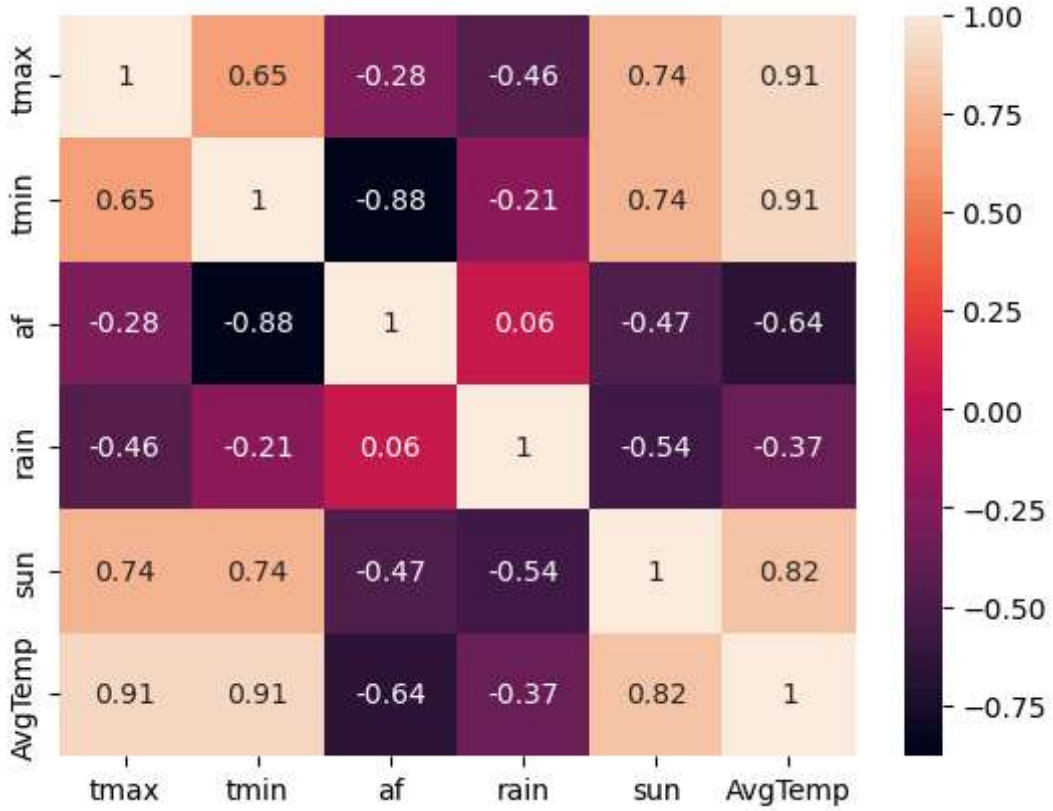
	tmax	tmin	af	rain	sun	
count	3.700000e+01	3.700000e+01	3.700000e+01	3.700000e+01	3.700000e+01	3.7
mean	3.720747e-16	3.488201e-17	-6.301266e-17	8.491706e-16	-8.881784e-16	1.9
std	1.013794e+00	1.013794e+00	1.013794e+00	1.013794e+00	1.013794e+00	9.2
min	-2.537847e+00	-2.780235e+00	-1.751186e+00	-1.114169e+00	-1.907070e+00	-2.2
25%	-4.882479e-01	-5.830856e-01	-5.646386e-01	-7.868309e-01	-8.101484e-01	-6.6
50%	7.195725e-02	9.008830e-02	-1.200637e-01	-2.878210e-01	-8.098500e-02	1.0
75%	8.322103e-01	6.174594e-01	5.583208e-01	5.644689e-01	6.097874e-01	6.4
max	1.596324e+00	2.028600e+00	3.313132e+00	2.846935e+00	2.258124e+00	1.4

In [235…
```
key_columns= ["tmax", "tmin", "af", "rain", "sun", "AvgTemp"]
key_columns
```

Out[235…
```
['tmax', 'tmin', 'af', 'rain', 'sun', 'AvgTemp']
```

In [237…
```
#Let's use Scatter Plot Heatmap to see the correlation index between the variabl
correlation = scaled_k_means_data[key_columns].corr().round(2)
sns.heatmap(correlation, annot = True)
```

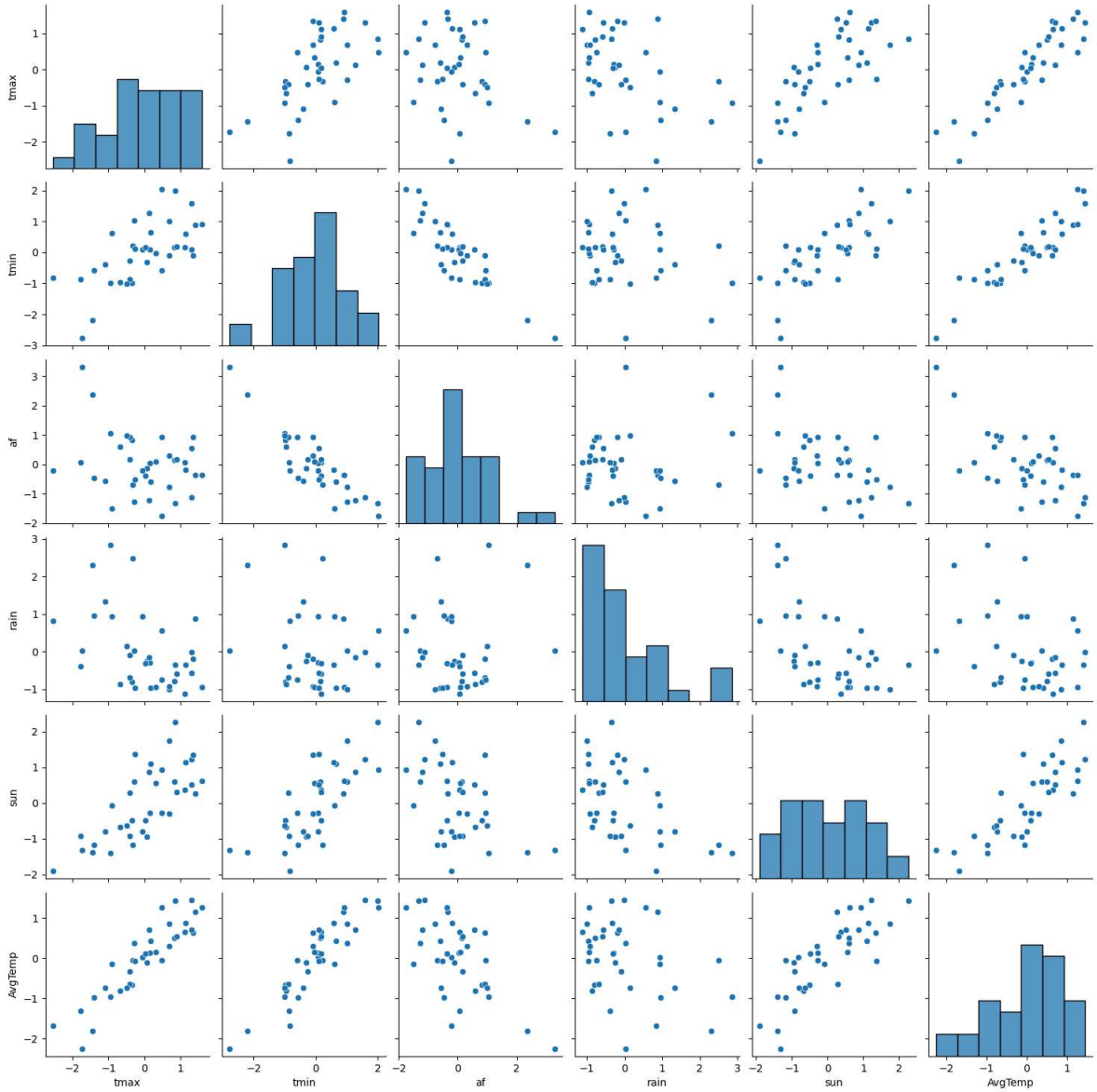
Out[237…
```
<Axes: >
```



There is a strong positive correlation between AvgTemp and sunshine and strong negative temperature between AvgTemp and af. This gives a pointer to the variables selection for the 2D plot to show the classification

In [240…
```
sns.pairplot(scaled_k_means_data[key_columns])
```

Out[240…
```
<seaborn.axisgrid.PairGrid at 0x1a4e54ef110>
```



Now, let's create the elbow graph to determine the value of k

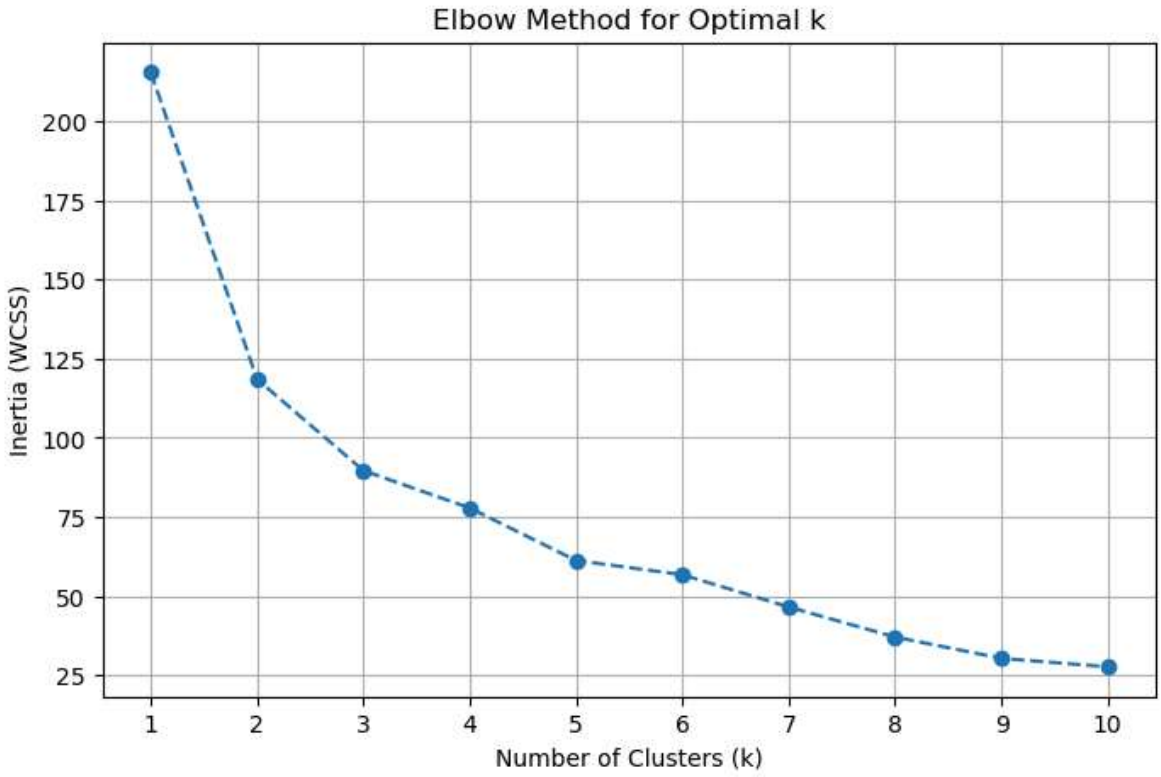
Since there are no predefined labels, elbow graph provides an oblective way to select our k value.

In [244…
```
from sklearn.cluster import KMeans
```

In [246…
```
# Range of k values to test
k_range = range(1, 11)

# Store the inertia (sum of squared distances to closest cluster center)
inertias = []

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_k_means_data)
    inertias.append(kmeans.inertia_)

# Plot the elbow chart
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertias, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (WCSS)')
```

```python
plt.title('Elbow Method for Optimal k')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```

```
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
```



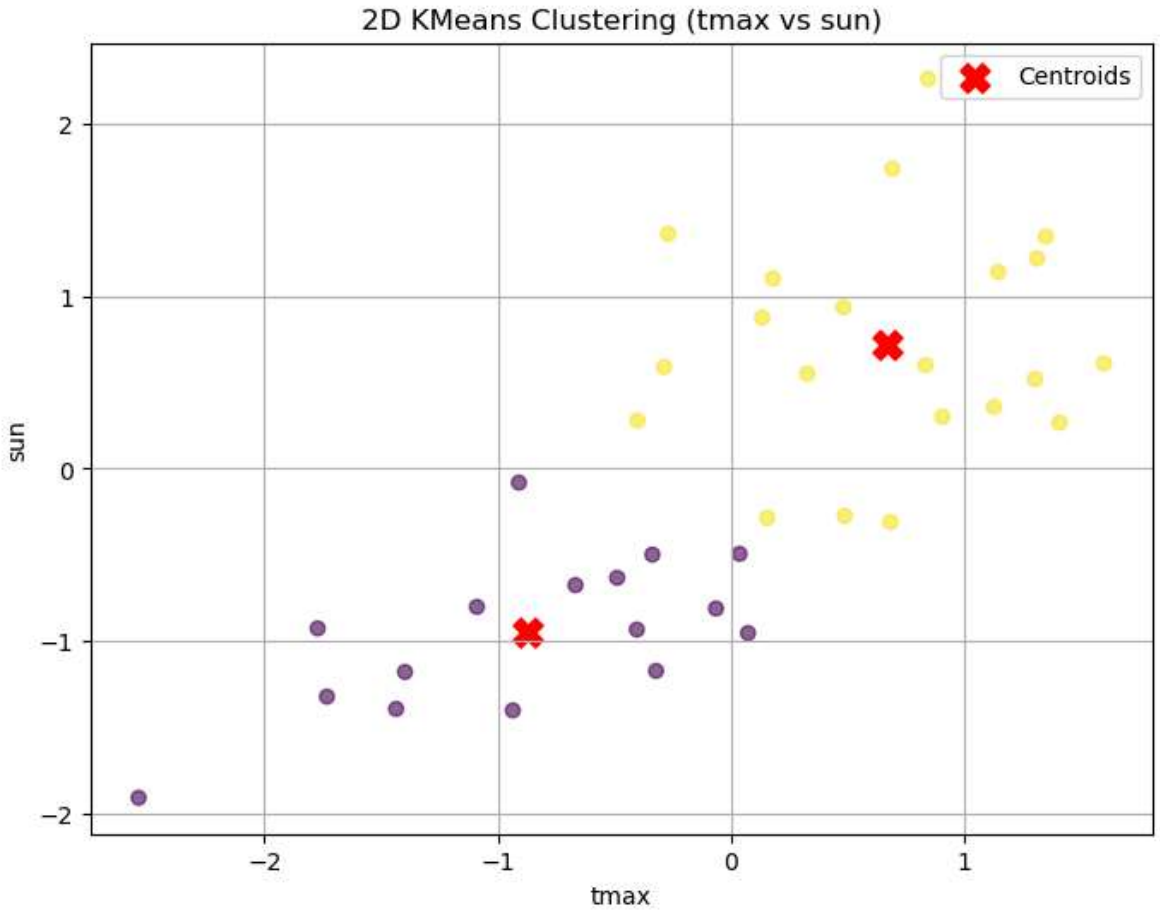
Since the elbow start at 2, let's take k = 2

Let's try to use tmax and sun for the scatter plot to display the clusters

In [250…

```
# Choose 2 features
X_2d = scaled_k_means_data[['tmax', 'sun']]

# Fit KMeans
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X_2d)

# Plot
plt.figure(figsize=(8, 6))
plt.scatter(X_2d['tmax'], X_2d['sun'], c=labels, cmap='viridis', alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=150, c='red', marker='X', label='Centroids')
plt.xlabel('tmax')
plt.ylabel('sun')
plt.title('2D KMeans Clustering (tmax vs sun)')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
```

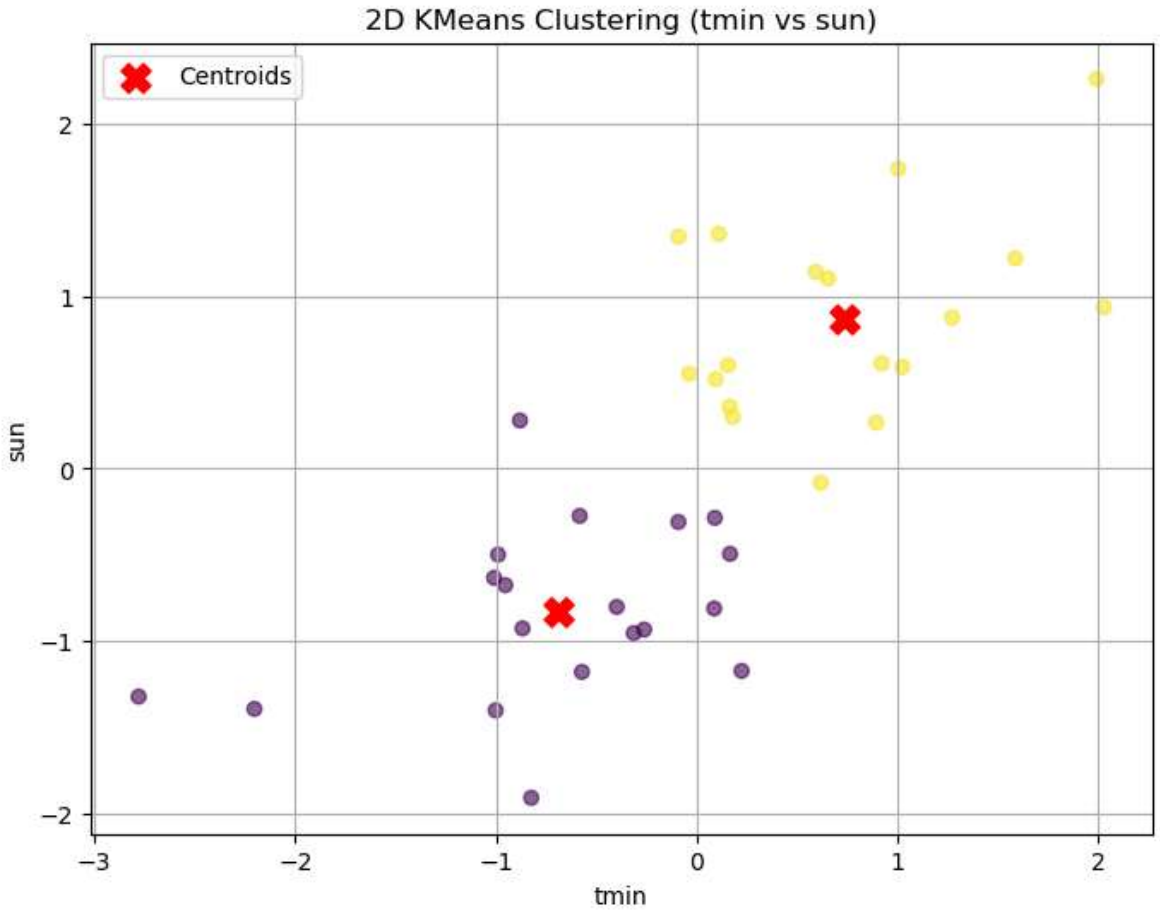


Let's try tmin and sun because both are less correlated

In [253…

```
# Choose 2 features
X_2d = scaled_k_means_data[['tmin', 'sun']]

# Fit KMeans
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X_2d)

# Plot
plt.figure(figsize=(8, 6))
plt.scatter(X_2d['tmin'], X_2d['sun'], c=labels, cmap='viridis', alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=150, c='red', marker='X', label='Centroids')
plt.xlabel('tmin')
plt.ylabel('sun')
plt.title('2D KMeans Clustering (tmin vs sun)')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
```

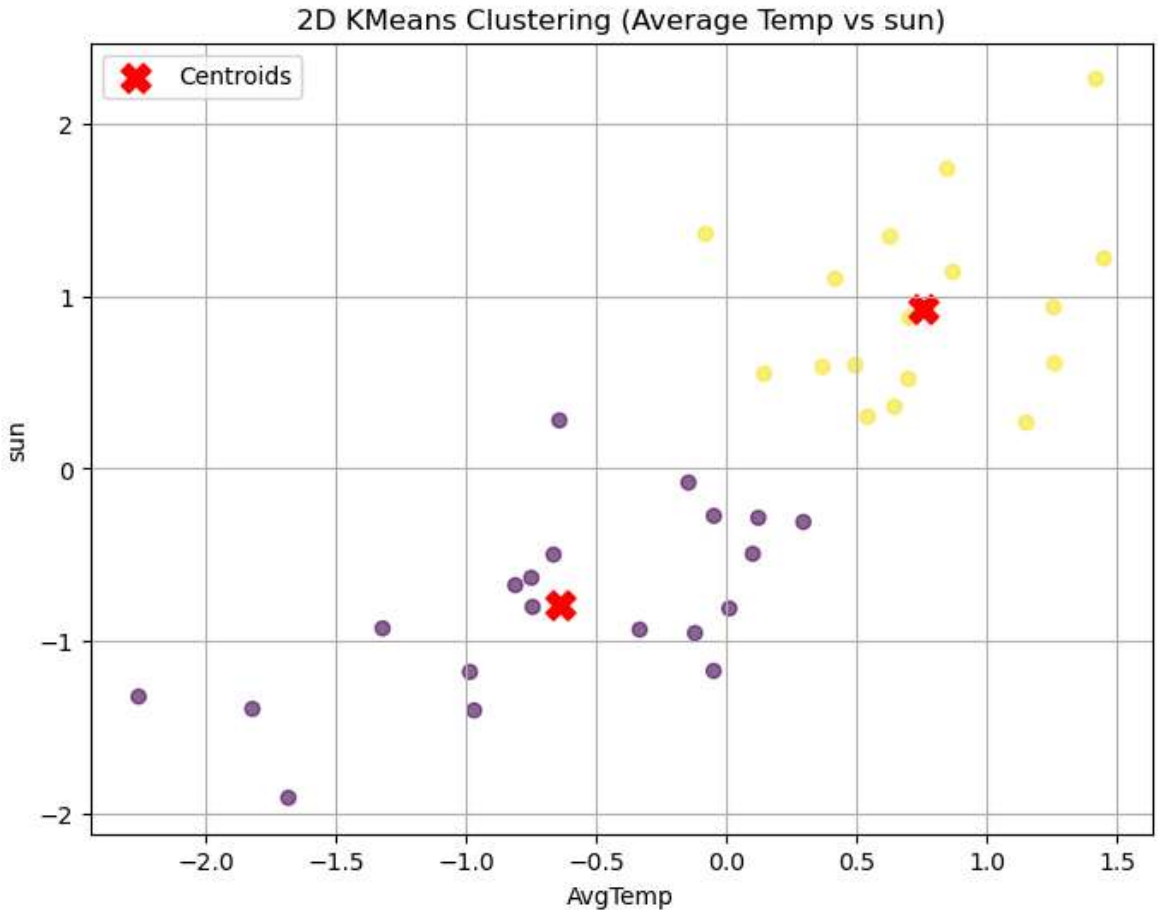


Let's try AvgTemp and sun

In [256…

```
# Choose 2 features
X_2d = scaled_k_means_data[['AvgTemp', 'sun']]

# Fit KMeans
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X_2d)

# Plot
plt.figure(figsize=(8, 6))
plt.scatter(X_2d['AvgTemp'], X_2d['sun'], c=labels, cmap='viridis', alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=150, c='red', marker='X', label='Centroids')
plt.xlabel('AvgTemp')
plt.ylabel('sun')
plt.title('2D KMeans Clustering (Average Temp vs sun)')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\emper\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
```



As shown in the figure above, the data points are clustered into two. The purple data points represents the cold weather station (with low average temperature and sunshine hours) while the yellow data points represent the hot data points (with high average temperature and sunshine hours)

Let's assess the performance of the algorithm

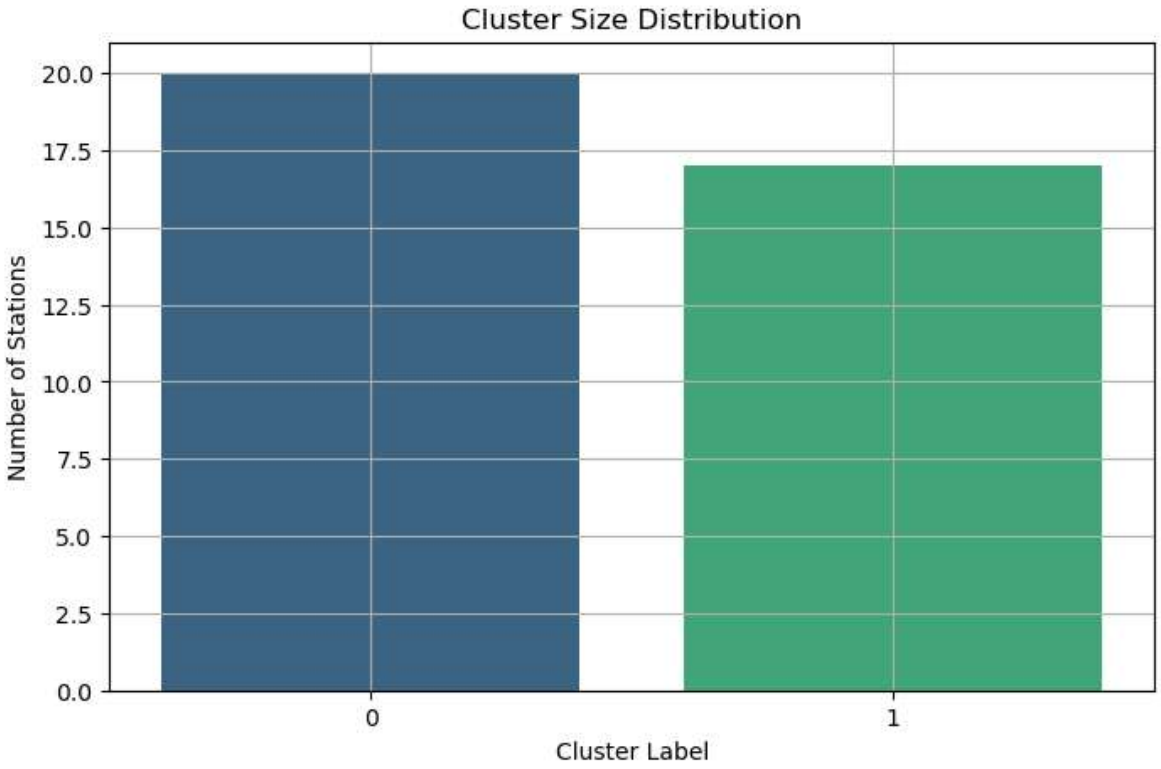
Let's plot the cluster Count

In [261…
```
from sklearn.metrics import silhouette_score

# labels are your KMeans cluster labels
unique_labels, counts = np.unique(labels, return_counts=True)

unique_labels, counts
```

Out[261…
```
(array([0, 1]), array([20, 17], dtype=int64))
```

In [263…
```
# Plot bar chart
plt.figure(figsize=(8, 5))
sns.barplot(x=unique_labels, y=counts, palette='viridis')
plt.xlabel('Cluster Label')
plt.ylabel('Number of Stations')
plt.title('Cluster Size Distribution')
plt.grid(True)
plt.show()
```

```
C:\Users\emper\AppData\Local\Temp\ipykernel_6164\826244185.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=unique_labels, y=counts, palette='viridis')
```

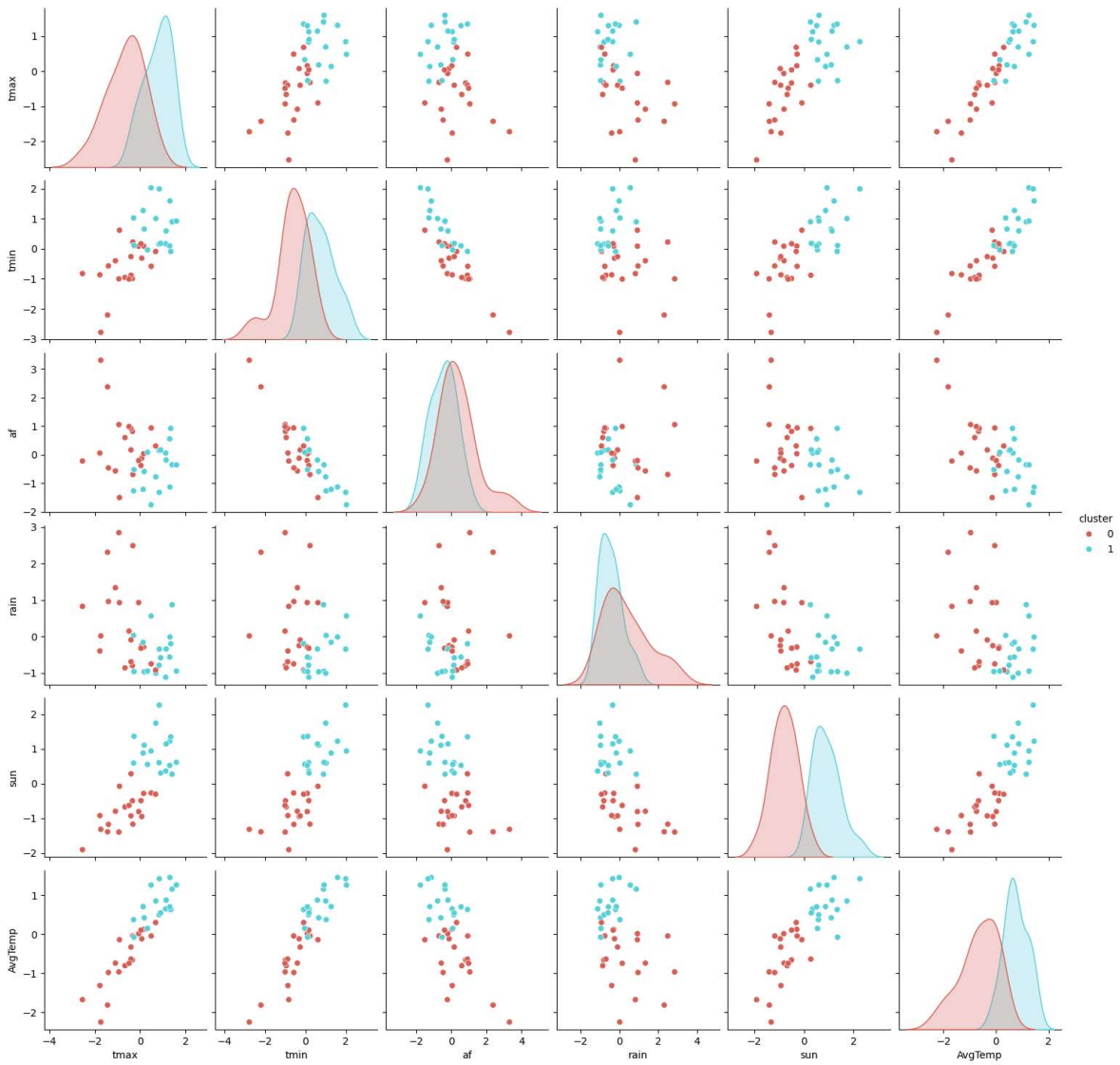


Let's plot Silhouette_score

In [266…
```
# Evaluate clustering quality
silhouette = silhouette_score(X_2d, labels)
print(f"Silhouette Score: {silhouette:.2f}")
```

```
Silhouette Score: 0.55
```

In [268…
```
# Add KMeans cluster labels as a new column for hue
scaled_k_means_data['cluster'] = labels  # replace 'labels' with your actual lab

# Optional: Create a nice color palette
palette = sns.color_palette("hls", len(scaled_k_means_data['cluster'].unique()))

# Create the pairplot
sns.pairplot(scaled_k_means_data, hue='cluster', palette=palette, diag_kind='kde

# Show plot
plt.show()
```



In [728…
```
#plt.scatter(MyRes2.iloc[:,0],MyRes2.iloc[:,1], c=MyRes2.iloc[:,2])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: