Smart Beach - Phase II

Smart Beach – Phase II is a major research project I worked on as my Live-Client capstone for Big Data Analytics program at Georgian College. I led a team of 4 to explore correlations between weather variables, wave heights near the shore of Station Beach in Kincardine and the volume of people who frequented the beach. We performed Requirements gathering, Data sourcing, Data Analysis, Initial implementation of machine learning models and built Power BI Dashboards for this phase of the 3-yr project.

I set the initial criteria, supervised each stage of the project, and had the opportunity to guide each team member in their respective tasks for the 5 months we worked on this project. Weekly status meetings were held with the stakeholders to present status updates and receive questions/feedback. The full project in itself had a lot of components but some of the work like analysis and machine learning was performed exclusively by me which is shared in this project showcase below.

Table of Contents

Project Overview

Context

The Smart Beach Project is a 3-year pilot program, spearheaded by the Nuclear Innovation Institute's Municipal Innovation Council (MIC) in collaboration with many local stakeholders aiming to revolutionize beach safety in Kincardine, Ontario. The initiative comes in response to a community-driven desire to enhance safety at Station Beach, which has experienced unfortunate incidents in the water over the past decade. The Smart Beach Project holds immense importance for Kincardine as it strives to provide beachgoers with real-time information on water conditions, thus significantly improving safety at Station Beach.

To make this happen, an inshore RAEON (Real-Time Aquatic Ecosystem Observation Network) buoy was installed to collect data on wave conditions and currents near the shore, and several cameras were installed to collect images of people using the beach around the summer. In Phase I of the project, the previous Data and AI team's focus was to establish a machine learning model to gauge if the wave data collected from the newly installed inshore buoy could be predicted using publicly available Environmental Canada and NOAA's (National Oceanic Atmospheric Administration) offshore buoy data and to create a vision model that uses beach images to count the number of people at the beach respectively.

For Phase II of the project, our Data team was requested to test if a machine learning model is possible to predict people count at the beach from data collected through the vision model and external sources. The new AI team was to continue working on the existing vision model to improve its people detection accuracy.

The results of these machine learning models are expected to support the Smart Beach project further to develop a real-time beach security information system for beachgoers and improve safety at Kincardine's Station Beach. After a proof of concept is achieved, the project would go into production and beach safety systems would be established at all other concerning locations as well.

Action

Weather variables and conditions are selected as an external source to be used as the independent variables, to predict the number of people at the beach, given the availability of several weather outlets and no additional infrastructure costs for collecting it. Data from three reputable weather outlets is then reviewed, analysed, and assessed based on the usability and variety of weather variables present against the consistency in data availability over the timeframe that aligned with Smart Beach project's operating time and collected data.

The selected weather data is then cleaned and undergoes an exploratory data analysis before being used in machine learning for people count prediction. Due to a delay in receiving people count data, I advised the stakeholders that an exploratory analysis on wave heights and their occurrences be conducted as it might help better understand the problems with high waves and it would be a good use of time while waiting as an analysis had not been performed previously. The wave data is then explored to understand its correlations with weather conditions, the wave height trends, frequency of waves and their timing and any other information that could provide more insights into wave trends. Then the people count data is explored and reviewed against available wave data before being used in machine learning for proof of concept. Some data consistency issues are noted and possible solutions are provided to the client for better progression of the project.

Results & Findings

- Rainy days saw the biggest high waves(>1.5m) and majority of them at that. Although none of the numerical weather variables presented very strong association with wave heights,
  - Precipitation and wind speed variables showed moderate positive correlation.
  - While temperature variables showed a moderate negative correlation with wave heights.
- Another noteworthy finding was the correlation between wind direction (categorical) and wave heights
  - Days with winds between South-west and North-east directions (225°-45°) showed consistent high wave occurrences.
  - With winds between West and North directions showing the biggest waves recorded just under 2.5 meters.
- 171 high wave occurrences were recorded between 13 of the 144 days under probe.
  - 53 of these 171 recorded occurrences happened between Daytime hours of 8AM and 8PM
  - These 53 records could be tied to 11th of August, 22nd and 26th of September and 7th, 8th and 18th of October, making these days substantially risky for beach goers, especially September 22nd which had high waves occurring all day between 8AM and 8PM.
- Data consistency issues were noticed with People Count Data:
  - Post data cleaning, 218 records remained which covered only 52 days of data out of the 144 days in question.
  - 135 of these 218 records belonged to just 10 of the 52 available days making the data very unevenly distributed.
  - Moreover, the records available were not collected at consistent intervals and beach usage data for majority of the timeframe is missing, and no data being available at all for most high wave days.
  - These limitations due to lack of data consistency and availability could possibly affect the machine learning results negatively.
- Selective data is used to overcome these challenges and test a machine learning model for proof of concept regardless of lack of data and the consistency issue:
  - The frequency of records is studied for each day and a smaller timeframe with the most number of readings is selected which is 12PM to 5PM in our case.
  - This leaves us with 37 days of beach usage data and then a unique record with the maximum number of people at the beach is chosen for each day to represent beach usage for the day.
  - However, using only 37 records of data to predict beach usage over a 6 month period may not be reflective of the true beach usage so It is suggested to the client that ML models are rerun when more data is available for better results.
- After running a few ML models and hyperparameter tuning, a Mean Absolute Error of 4.01 people and an R-squared value of 0.74 is achieved
  - Prior to running the models, data is cleaned, dimensionality reduction is performed, categorical variables are converted to dummy variables, data is standardized to be used with multiple algorithms.
  - Train-Test split of 70-30 is used and no separate validation set is split to maintain enough data for training and testing.
  - Best performing algorithm before hyperparameter tuning was Linear Regression with MAE of 5.4 and an R-squared value of 0.65
  - Best performing algorithms post hyperparameter tuning were XGBoost and Gradient Boosting Regressor with MAE and R-squared values of 4.01 & 0.739 and 4.04 & 0.729 respectively.

Discussion & Next steps

Insights gathered from the analysis of wave data and weather variables in this project give us a more candid overview of Station Beach’s wave activity in summer 2022 and its correlation with different factors that influence the overall weather conditions and how it may affect the beach goers adversely at certain times. It is noticed that relatively colder days, rainy weather and higher windspeeds have a positive correlation with wave heights, and winds blowing in North to West directions were associated with the biggest waves, posing

higher risk to beach goers. These trends should be studied over the next couple years and the findings are expected to help the local counties in allocating necessary resources and making the required arrangements to uphold beach safety and prevent deadly mishaps.

A moderate positive correlation was seen between temperature variables and the number of people at the beach, and although ML results beyond expectations were received with the very limited people count data at hand. The client is highly advised to procure more data for an analysis and ML model with solid foundations. The client is also advised to perform analysis on Beach Usage data and test the beach usage numbers against the wave height data to understand beach usage patterns against dangerous water levels. The frequency of beach usage data collection is suggested to be kept at least at the same number and intervals as that of the wave data for optimum results with analysis and ML models.

Weather Data Exploratory Analysis

Import Weather variable data

In [1]:
```
import pandas as pd
Wdf = pd.read_csv("vc data - cleaned+.csv")
Wdf.head()
```

Out[1]:

	datetime	tempmax	tempmin	temp	feelslikemax	feelslikemin	feelslike	dew	humidity	precip	precipprob
0	2022-06-07	17.6	10.7	13.9	17.6	10.7	13.9	12.2	89.6	37.380	100
1	2022-06-08	17.0	8.6	13.8	17.0	8.6	13.8	10.2	79.4	0.839	100
2	2022-06-09	17.7	11.1	13.3	17.7	11.1	13.3	10.7	84.4	12.976	100
3	2022-06-10	19.9	8.7	14.3	19.9	8.7	14.2	9.0	72.6	0.063	100
4	2022-06-11	23.0	12.5	17.5	23.0	12.5	17.5	11.4	69.1	0.207	100

Understand the data

Look for number of records, missing values, data types etc.

In [11]:
```
Wdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 19 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   datetime      144 non-null    object 
 1   tempmax       144 non-null    float64
 2   tempmin       144 non-null    float64
 3   temp          144 non-null    float64
 4   feelslikemax  144 non-null    float64
 5   feelslikemin  144 non-null    float64
 6   feelslike     144 non-null    float64
 7   dew           144 non-null    float64
 8   humidity      144 non-null    float64
 9   precip        144 non-null    float64
 10  precipprob    144 non-null    int64  
 11  precipcover   144 non-null    float64
 12  windgust      144 non-null    float64
 13  windspeed     144 non-null    float64
 14  winddir       144 non-null    float64
 15  cloudcover    144 non-null    float64
 16  visibility    144 non-null    float64
 17  uvindex       144 non-null    int64  
 18  conditions    144 non-null    object 
dtypes: float64(15), int64(2), object(2)
memory usage: 21.5+ KB
```

Statistics Summary

Check minimum, maximums, outliers and skewness in the data.

In [12]:
```
Wdf.describe()
```

Out[12]:

	tempmax	tempmin	temp	feelslikemax	feelslikemin	feelslike	dew	humidity	
count	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144
mean	21.188194	12.200694	16.887500	21.229167	11.671528	16.608333	11.824306	73.831944	2
std	5.472889	5.090255	5.017325	5.917792	5.914669	5.682737	5.039012	8.753194	6
min	5.300000	0.800000	3.700000	2.200000	-2.100000	-0.100000	0.000000	53.100000	0
25%	18.075000	8.875000	13.750000	18.075000	8.525000	13.750000	8.875000	67.175000	0
50%	22.550000	12.600000	17.900000	22.550000	12.600000	17.900000	12.600000	74.500000	0
75%	24.750000	16.200000	20.600000	24.750000	16.200000	20.600000	15.825000	80.500000	1
max	30.000000	22.900000	25.600000	33.200000	22.900000	26.500000	22.000000	92.900000	51

Statistics Summary Grouped by weather conditions

In [5]:
```
Wdf[['conditions']].describe()
```

Out[5]:

	conditions
count	144
unique	4
top	rain
freq	69

In [6]:
```
Wdf.groupby('conditions')[['tempmax']].agg(['count','mean','median','min','max'])
```

Out[6]:

	tempmax				
	count	**mean**	**median**	**min**	**max**
conditions					
clear-day	16	21.962500	22.65	14.0	27.8
partly-cloudy-day	58	22.139655	22.75	12.0	29.8
rain	69	20.437681	21.80	5.3	30.0
snow	1	5.400000	5.40	5.4	5.4

In [15]:
```
Wdf.groupby('conditions')[['tempmin']].agg(['count','mean','median','min','max'])
```

Out[15]:

	tempmin				
	count	**mean**	**median**	**min**	**max**
conditions					
clear-day	16	9.912500	9.6	3.2	16.2
partly-cloudy-day	58	11.562069	12.4	0.8	20.9
rain	69	13.405797	13.9	2.0	22.9
snow	1	2.700000	2.7	2.7	2.7

In [16]:
```
Wdf.groupby('conditions')[['feelslike']].agg(['count','mean','median','min','max'])
```

Out[16]:

	feelslike				
	count	**mean**	**median**	**min**	**max**
conditions					
clear-day	16	16.168750	16.50	8.4	22.9
partly-cloudy-day	58	17.101724	18.15	3.7	26.1
rain	69	16.526087	18.60	-0.1	26.5
snow	1	0.700000	0.70	0.7	0.7

In [17]:
```
Wdf.groupby('conditions')[['humidity']].agg(['count','mean','median','min','max'])
```

Out[17]:

	humidity				
	count	mean	median	min	max
conditions					
clear-day	16	70.812500	73.45	60.3	79.6
partly-cloudy-day	58	70.043103	70.25	53.1	84.6
rain	69	77.488406	78.90	55.9	92.9
snow	1	89.600000	89.60	89.6	89.6

In [18]:
```
Wdf.groupby('conditions')[['precip']].agg(['count','mean','median','min','max'])
```

Out[18]:

	precip				
	count	mean	median	min	max
conditions					
clear-day	16	0.000000	0.000	0.000	0.000
partly-cloudy-day	58	0.000000	0.000	0.000	0.000
rain	69	4.616768	1.843	0.026	51.614
snow	1	15.498000	15.498	15.498	15.498

In [19]:
```
Wdf.groupby('conditions')[['precipcover']].agg(['count','mean','median','min','max'])
```

Out[19]:

	precipcover				
	count	mean	median	min	max
conditions					
clear-day	16	0.000000	0.00	0.00	0.00
partly-cloudy-day	58	0.000000	0.00	0.00	0.00
rain	69	23.671594	16.67	4.17	83.33
snow	1	87.500000	87.50	87.50	87.50

In [20]:
```
Wdf.groupby('conditions')[['precipprob']].agg(['count','mean','median','min','max'])
```

Out[20]:

	precipprob				
	count	mean	median	min	max
conditions					
clear-day	16	0.0	0.0	0	0
partly-cloudy-day	58	0.0	0.0	0	0
rain	69	100.0	100.0	100	100
snow	1	100.0	100.0	100	100

In [21]:
```
Wdf.groupby('conditions')[['windspeed']].agg(['count','mean','median','min','max'])
```

Out[21]:

	windspeed				
	count	mean	median	min	max
conditions					
clear-day	16	17.262500	16.95	12.0	28.7
partly-cloudy-day	58	16.650000	15.80	10.0	39.8
rain	69	21.689855	22.10	9.1	36.8
snow	1	26.000000	26.00	26.0	26.0

In [22]:
```
Wdf.groupby('conditions')[['cloudcover']].agg(['count','mean','median','min','max'])
```

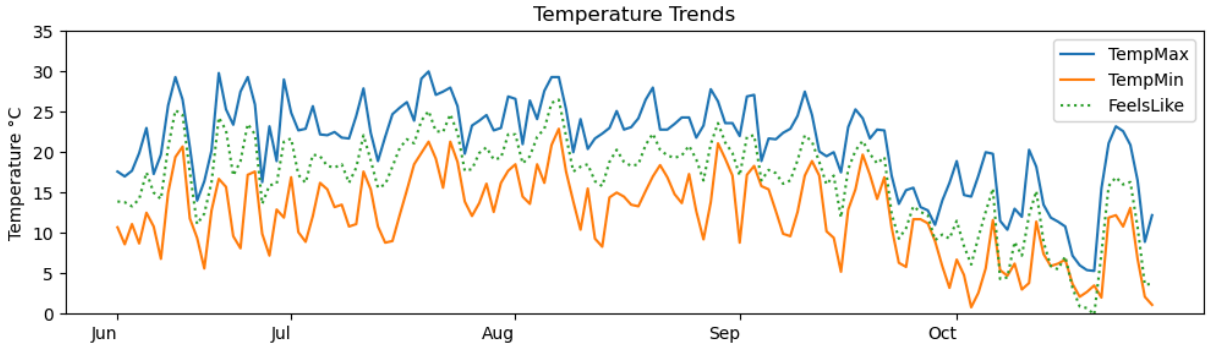
Out[22]:

	cloudcover				
	count	mean	median	min	max
conditions					
clear-day	16	14.243750	15.35	5.2	19.3
partly-cloudy-day	58	36.043103	34.65	20.1	69.0
rain	69	60.908696	59.60	21.7	95.8
snow	1	98.500000	98.50	98.5	98.5

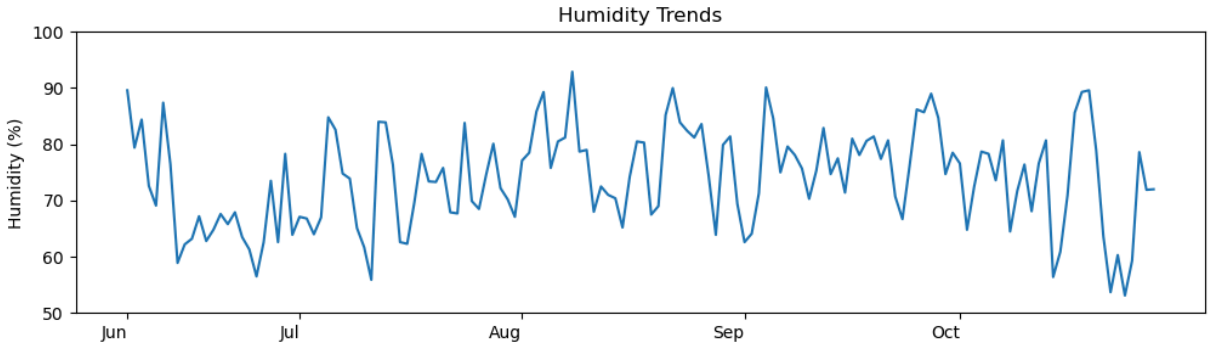
Visualize weather variable trends

In [2]:
```
import matplotlib.pyplot as plt
from calendar import month_abbr
```

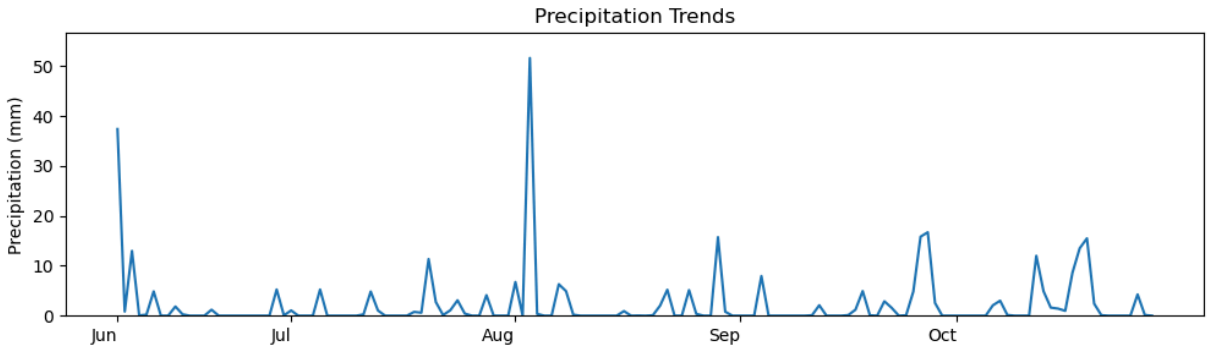
In [64]:
```
Wdf['datetime'] = pd.to_datetime(Wdf['datetime'])
Wdf['month'] = Wdf['datetime'].dt.month
Wdf['month_name'] = Wdf['month'].apply(lambda x: month_abbr[x])

plt.figure(figsize=(12, 3))

plt.plot(Wdf['datetime'], Wdf['tempmax'], label='TempMax')
plt.plot(Wdf['datetime'], Wdf['tempmin'], label='TempMin')
plt.plot(Wdf['datetime'], Wdf['feelslike'], label='FeelsLike', linestyle='dotted')

plt.title('Temperature Trends')
plt.ylabel('Temperature °C')

month_index = Wdf.groupby('month').head(1).index

plt.xticks(Wdf.loc[month_index, 'datetime'], Wdf.loc[month_index, 'month_name'], rotation
plt.ylim(0, 35)
plt.legend()

plt.show()
```



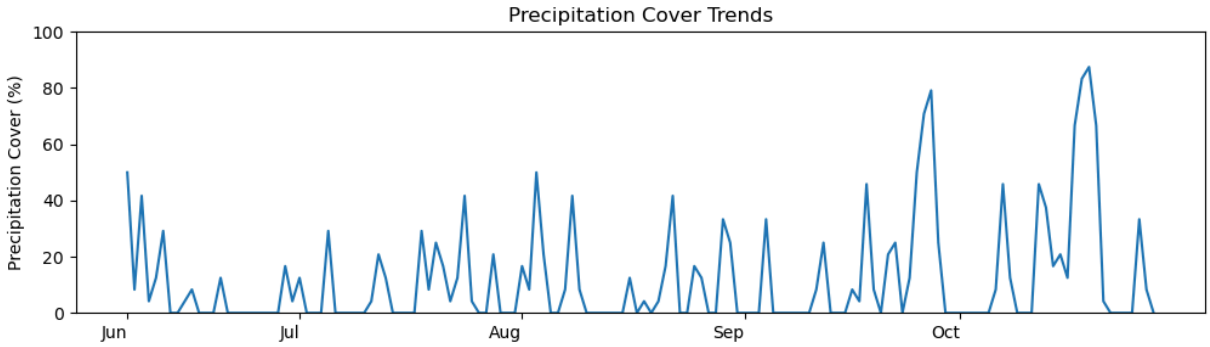
In [70]:
```
plt.figure(figsize=(12, 3))

plt.plot(Wdf['datetime'], Wdf['humidity'], label='Humidity')

plt.title('Humidity Trends')
plt.ylabel('Humidity (%)')
month_index = Wdf.groupby('month').head(1).index
plt.xticks(Wdf.loc[month_index, 'datetime'], Wdf.loc[month_index, 'month_name'], rotation
plt.ylim(50, 100)

plt.show()
```



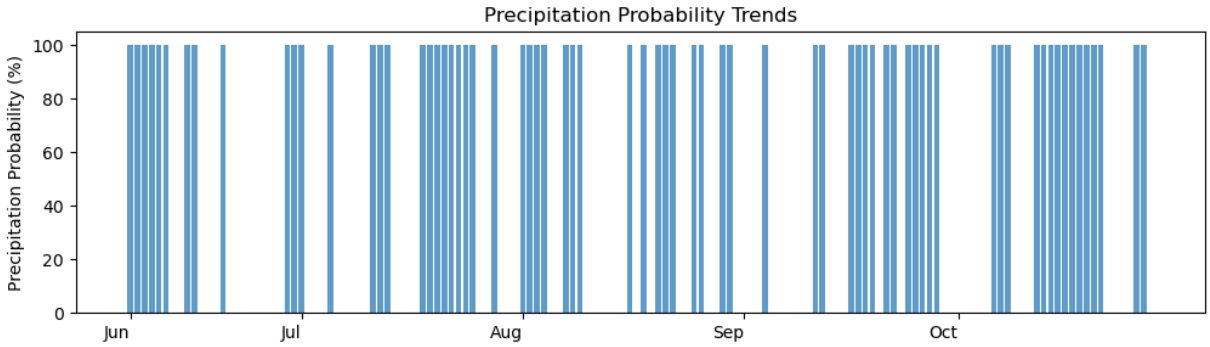
In [71]:
```
plt.figure(figsize=(12, 3))

plt.plot(Wdf['datetime'], Wdf['precip'], label='Precipitation')

plt.title('Precipitation Trends')
plt.ylabel('Precipitation (mm)')
month_index = Wdf.groupby('month').head(1).index
plt.xticks(Wdf.loc[month_index, 'datetime'], Wdf.loc[month_index, 'month_name'], rotation
plt.ylim(0, Wdf['precip'].max() + 5)

plt.show()
```



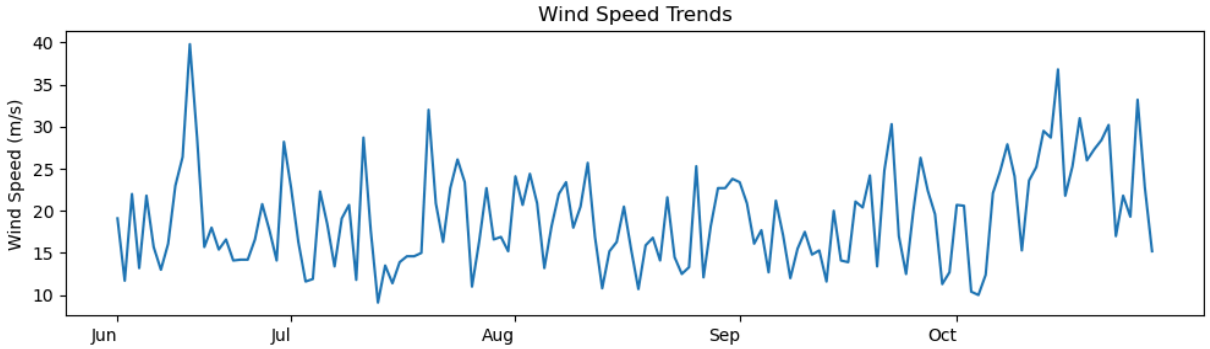
In [73]:
```
plt.figure(figsize=(12, 3))

plt.plot(Wdf['datetime'], Wdf['precipcover'], label='Precipitation Cover')

plt.title('Precipitation Cover Trends')
plt.ylabel('Precipitation Cover (%)')
month_index = Wdf.groupby('month').head(1).index
plt.xticks(Wdf.loc[month_index, 'datetime'], Wdf.loc[month_index, 'month_name'], rotation
plt.ylim(0, 100)

plt.show()
```



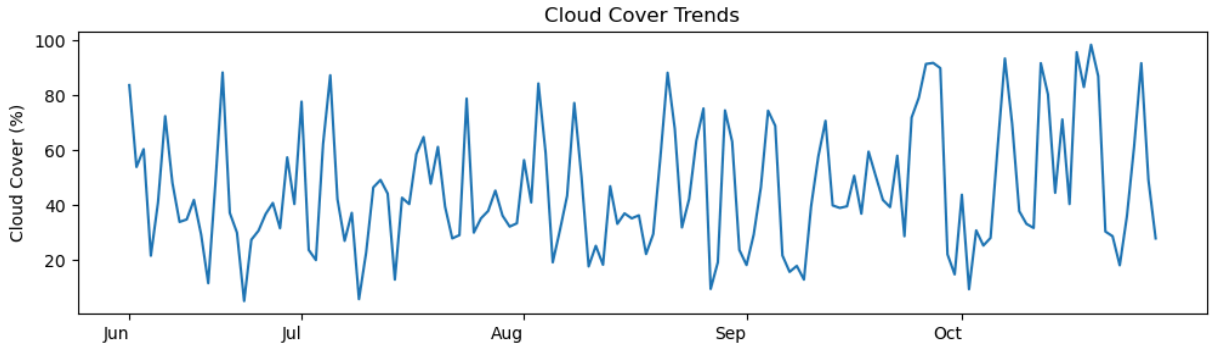
In [85]:
```
plt.figure(figsize=(12, 3))

plt.bar(Wdf['datetime'], Wdf['precipprob'], label='Precipitation Probability', alpha=0.7)

plt.title('Precipitation Probability Trends')
plt.ylabel('Precipitation Probability (%)')
month_index = Wdf.groupby('month').head(1).index
plt.xticks(Wdf.loc[month_index, 'datetime'], Wdf.loc[month_index, 'month_name'], rotation

plt.show()
```



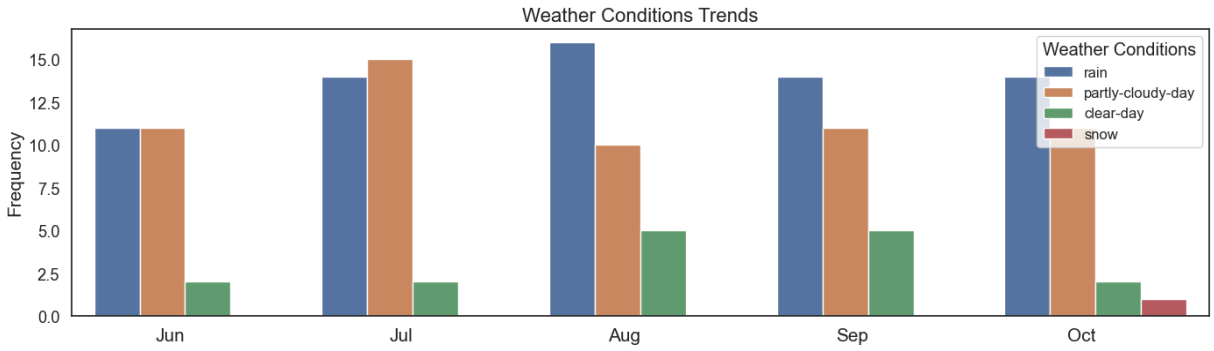
In [76]:
```
plt.figure(figsize=(12, 3))

plt.plot(Wdf['datetime'], Wdf['windspeed'], label='Wind Speed')

plt.title('Wind Speed Trends')
plt.ylabel('Wind Speed (m/s)')
month_index = Wdf.groupby('month').head(1).index
plt.xticks(Wdf.loc[month_index, 'datetime'], Wdf.loc[month_index, 'month_name'], rotation

plt.show()
```



In [77]:
```
plt.figure(figsize=(12, 3))

plt.plot(Wdf['datetime'], Wdf['cloudcover'], label='Cloud Cover')

plt.title('Cloud Cover Trends')
plt.ylabel('Cloud Cover (%)')
month_index = Wdf.groupby('month').head(1).index
plt.xticks(Wdf.loc[month_index, 'datetime'], Wdf.loc[month_index, 'month_name'], rotation

plt.show()
```



In [142]:
```
import seaborn as sns

sns.set_theme(style="white", font_scale=1.2)

plt.figure(figsize=(14, 4.5))
sns.countplot(x='month_name', hue='conditions', data=Wdf)

plt.title('Weather Conditions Trends', fontsize=16)
plt.xlabel(' ')
plt.ylabel('Frequency', fontsize=15)

plt.xticks(rotation=0, ha='right', fontsize=15)
plt.legend(title='Weather Conditions', fontsize=12)

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



Wave Data Exploratory Analysis

Import Wave height data

Import wave height information from internal inshore buoy data into the weather variable dataframe.

In [14]:
```
buoy_data = pd.read_csv('buoy_Kincardine.csv')
buoy_data['time'] = pd.to_datetime(buoy_data['time (UTC)'])

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

#Resample the data to daily frequency and calculate maximum, minimum, and average wave he
daily_wave_height = buoy_data['sea_surface_wave_significant_height (m)'].resample('D').ag

daily_wave_height.columns = ['maxwaveheight', 'minwaveheight', 'averagewaveheight']

#Reset index to make 'time' a column again
daily_wave_height.reset_index(inplace=True)

print(daily_wave_height)
```

```
                       time  maxwaveheight  minwaveheight  averagewaveheight
0   2022-06-07 00:00:00+00:00          0.536          0.204           0.426789
1   2022-06-08 00:00:00+00:00          0.533          0.141           0.307200
2   2022-06-09 00:00:00+00:00          0.470          0.077           0.164507
3   2022-06-10 00:00:00+00:00          0.409          0.205           0.290471
4   2022-06-11 00:00:00+00:00          0.271          0.150           0.191900
..                        ...            ...            ...                ...
139 2022-10-24 00:00:00+00:00          0.237          0.053           0.146818
140 2022-10-25 00:00:00+00:00          0.173          0.107           0.141100
141 2022-10-26 00:00:00+00:00          1.991          0.115           0.596000
142 2022-10-27 00:00:00+00:00          1.191          0.361           0.639500
143 2022-10-28 00:00:00+00:00          0.324          0.074           0.182833

[144 rows x 4 columns]
```

In [7]:
```
#Convert 'time' column to date only
daily_wave_height['time'] = daily_wave_height['time'].dt.date
print(daily_wave_height)
```

```
           time  maxwaveheight  minwaveheight  averagewaveheight
0    2022-06-07          0.536          0.204           0.426789
1    2022-06-08          0.533          0.141           0.307200
2    2022-06-09          0.470          0.077           0.164507
3    2022-06-10          0.409          0.205           0.290471
4    2022-06-11          0.271          0.150           0.191900
..          ...            ...            ...                ...
139  2022-10-24          0.237          0.053           0.146818
140  2022-10-25          0.173          0.107           0.141100
141  2022-10-26          1.991          0.115           0.596000
142  2022-10-27          1.191          0.361           0.639500
143  2022-10-28          0.324          0.074           0.182833

[144 rows x 4 columns]
```

In [8]:
```
daily_wave_height.head()
```

Out[8]:

	time	maxwaveheight	minwaveheight	averagewaveheight
0	2022-06-07	0.536	0.204	0.426789
1	2022-06-08	0.533	0.141	0.307200
2	2022-06-09	0.470	0.077	0.164507
3	2022-06-10	0.409	0.205	0.290471
4	2022-06-11	0.271	0.150	0.191900

In [9]:
```
#Convert 'datetime' in Wdf and 'time' in daily_wave_height to datetime before merging
Wdf['datetime'] = pd.to_datetime(Wdf['datetime'])
daily_wave_height['time'] = pd.to_datetime(daily_wave_height['time'])

merged_df = pd.merge(Wdf, daily_wave_height, left_on='datetime', right_on='time', how='in
merged_df = merged_df.drop('time', axis=1)
merged_df.head()
```

Out[9]:

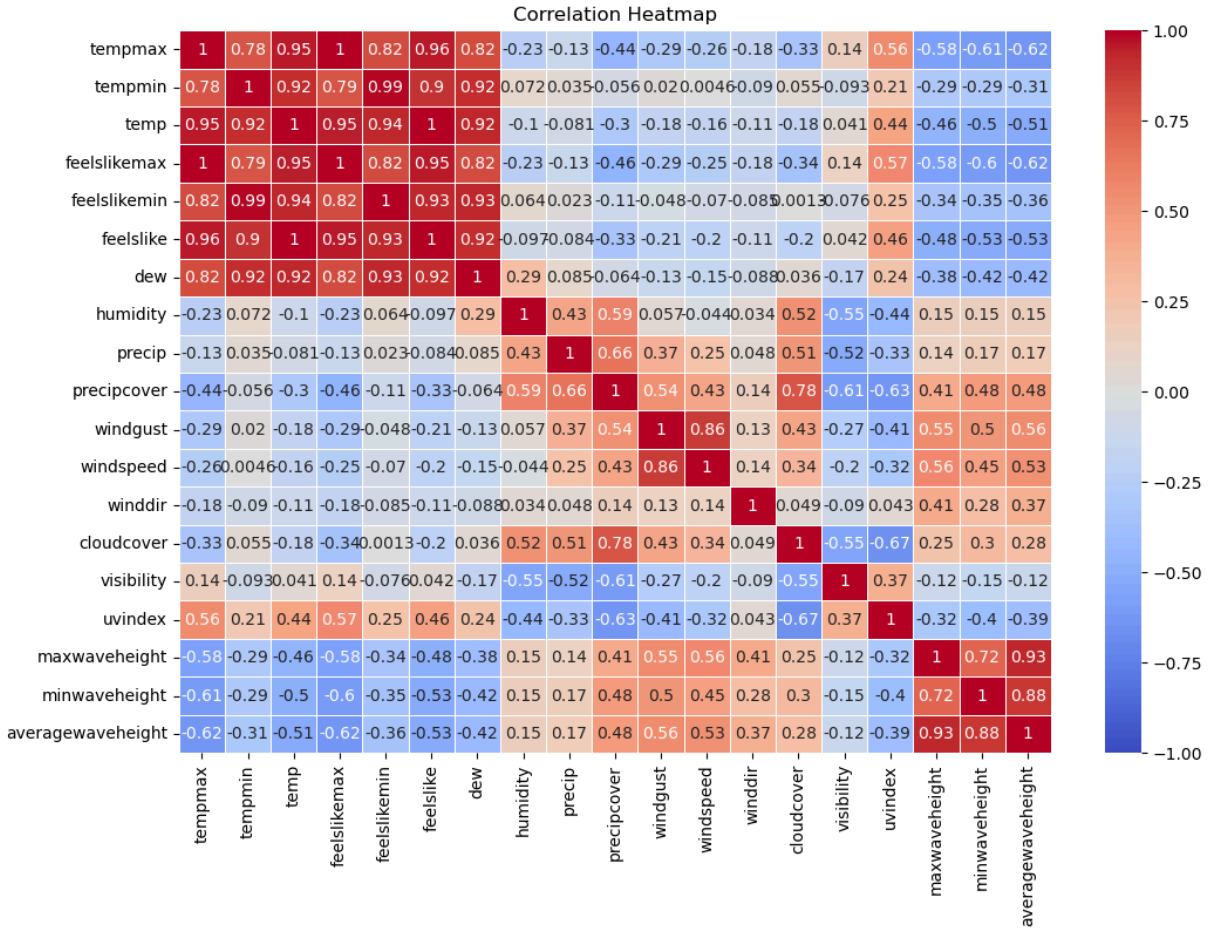
	datetime	tempmax	tempmin	temp	feelslikemax	feelslikemin	feelslike	dew	humidity	precip	...	windgu
0	2022-06-07	17.6	10.7	13.9	17.6	10.7	13.9	12.2	89.6	37.380	...	3
1	2022-06-08	17.0	8.6	13.8	17.0	8.6	13.8	10.2	79.4	0.839	...	1
2	2022-06-09	17.7	11.1	13.3	17.7	11.1	13.3	10.7	84.4	12.976	...	4
3	2022-06-10	19.9	8.7	14.3	19.9	8.7	14.2	9.0	72.6	0.063	...	3
4	2022-06-11	23.0	12.5	17.5	23.0	12.5	17.5	11.4	69.1	0.207	...	3

5 rows × 22 columns

Check correlations between weather variables and wave heights

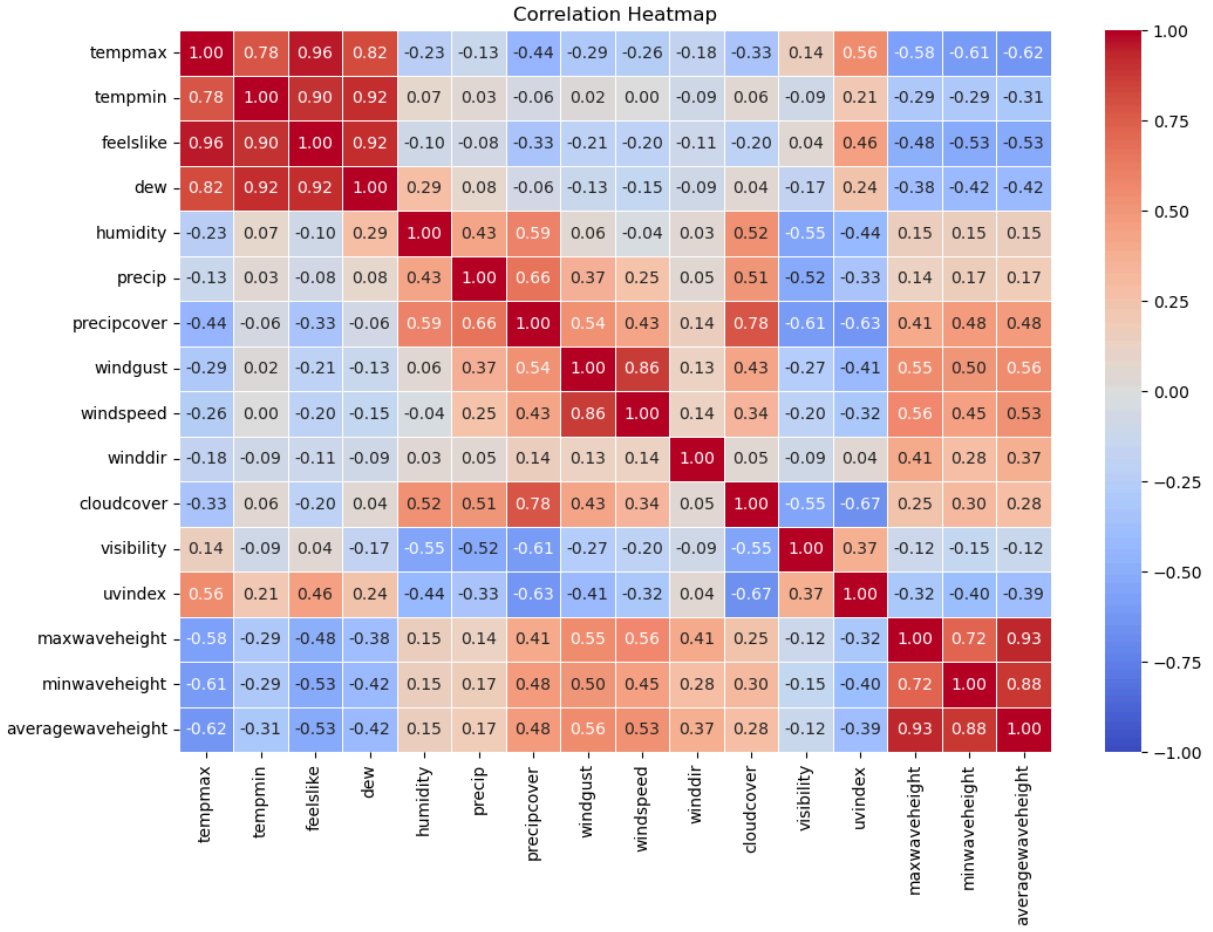
Use a Heatmap to look for correlations with waveheights

In [19]:
```
import seaborn as sns
import matplotlib.pyplot as plt

heatmap2_data = merged_df.drop(['datetime', 'precipprob', 'conditions'], axis=1)

plt.figure(figsize=(12, 8))
sns.heatmap(heatmap2_data.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1, linewidth

plt.title('Correlation Heatmap')
plt.show()
```



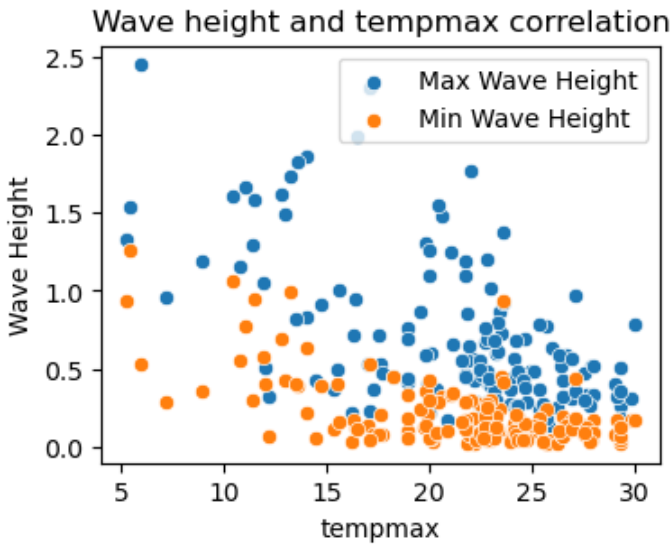
Remove variables with very high intercorrelation with other independent variables to declutter and only keep what may be needed.

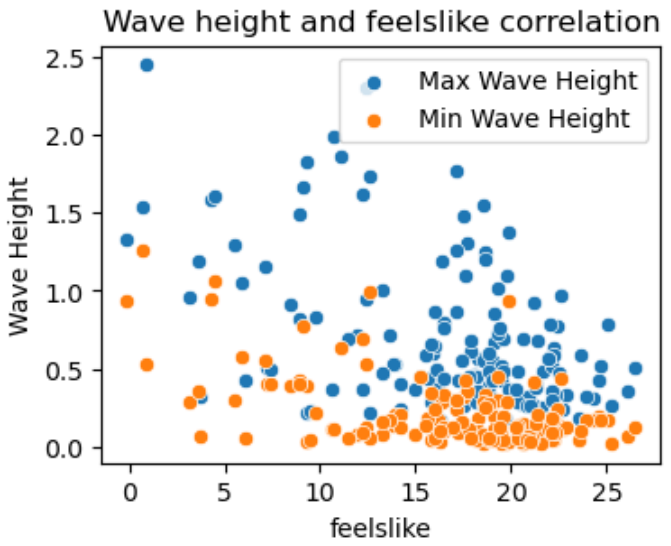
In [12]:
```
heatmap2_data = merged_df.drop(['datetime', 'precipprob', 'conditions', 'temp', 'feelslik

plt.figure(figsize=(12, 8))
sns.heatmap(heatmap2_data.corr(), annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1

plt.title('Correlation Heatmap')
plt.show()
```

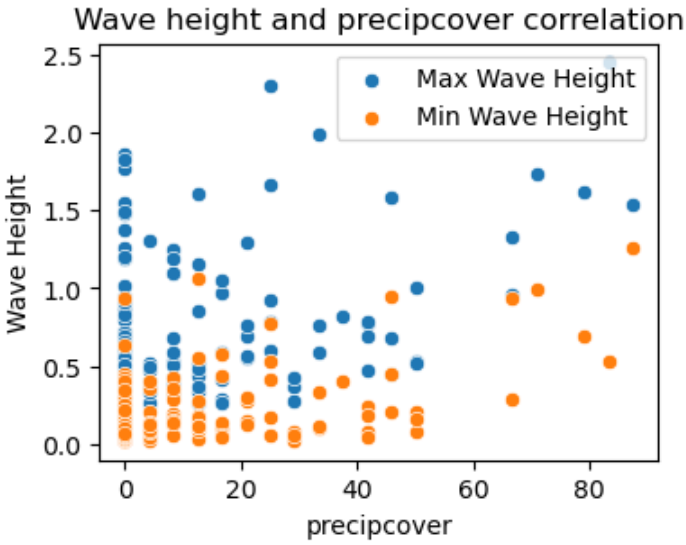


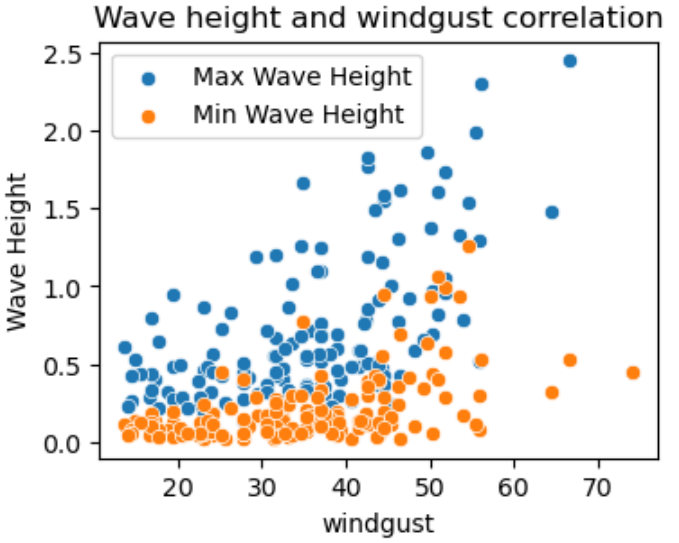
Plot variables with significant correlation with wave height

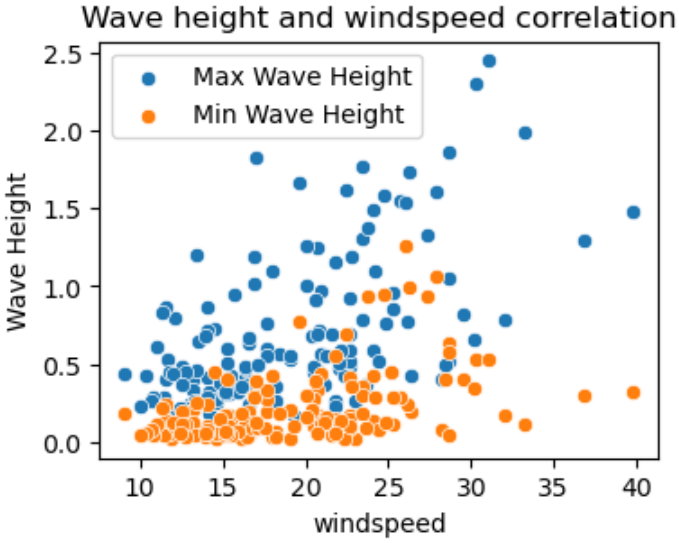
In [41]:
```
exclude_columns = ['datetime', 'precipprob', 'conditions', 'tempmin', 'temp', 'feelslikem

plot_columns = [col for col in Wdf.columns if col not in exclude_columns]

for col in plot_columns:
    plt.figure(figsize=(4, 3))

    sns.scatterplot(x=Wdf[col], y=merged_df['maxwaveheight'], label='Max Wave Height')
    sns.scatterplot(x=Wdf[col], y=merged_df['minwaveheight'], label='Min Wave Height')

    plt.title(f'Wave height and {col} correlation')
    plt.xlabel(col)
    plt.ylabel('Wave Height')
    plt.show()
```











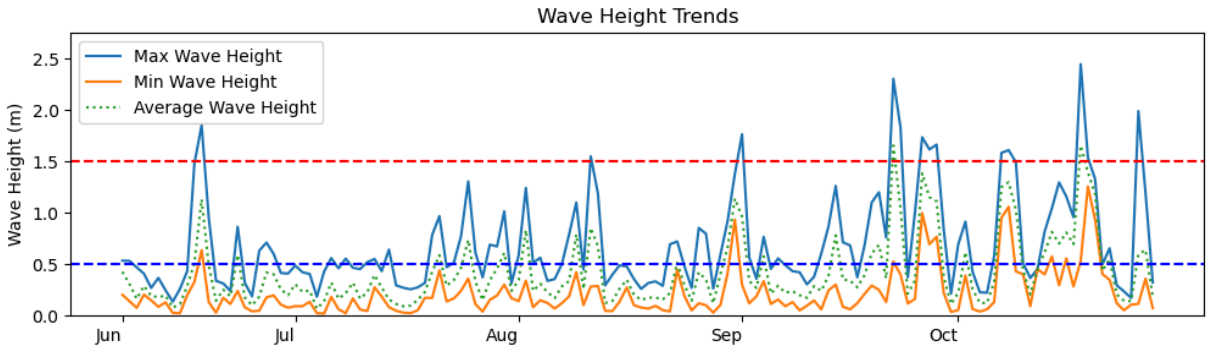
Visualize and explore wave height trends

In [43]:
```
merged_df.groupby('conditions')[['maxwaveheight']].agg(['count','mean','median','min','ma
```

Out[43]:

	maxwaveheight				
	count	mean	median	min	max
conditions					
clear-day	16	0.713188	0.479	0.215	1.855
partly-cloudy-day	58	0.574448	0.455	0.137	1.826
rain	69	0.783015	0.589	0.261	2.447
snow	1	1.538000	1.538	1.538	1.538

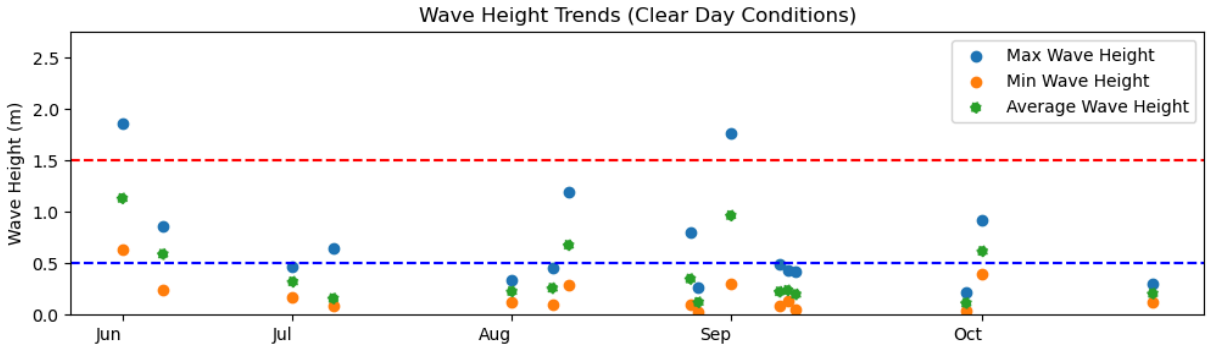
In [51]:
```
import matplotlib.pyplot as plt
import pandas as pd
from calendar import month_abbr

merged_df['time'] = pd.to_datetime(merged_df['datetime'])
merged_df['month'] = merged_df['time'].dt.month
merged_df['month_name'] = merged_df['month'].apply(lambda x: month_abbr[x])

plt.figure(figsize=(12, 3))

plt.plot(merged_df['time'], merged_df['maxwaveheight'], label='Max Wave Height')
plt.plot(merged_df['time'], merged_df['minwaveheight'], label='Min Wave Height')
plt.plot(merged_df['time'], merged_df['averagewaveheight'], label='Average Wave Height',

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends')
plt.ylabel('Wave Height (m)')

month_index = merged_df.groupby('month').head(1).index

plt.xticks(merged_df.loc[month_index, 'time'], merged_df.loc[month_index, 'month_name'],
plt.ylim(0, 2.75)
plt.legend()
plt.show()
```



Wave heights per Weather conditions

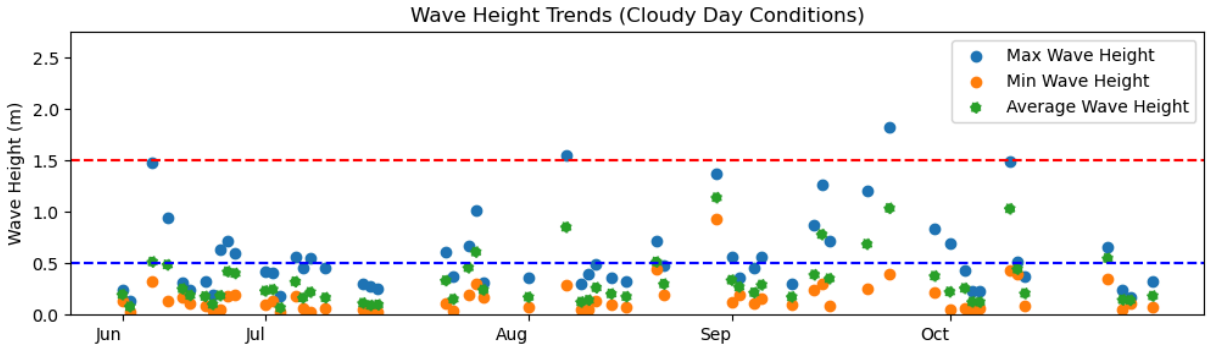
In [60]:

```
# Filter data for 'clear-day' conditions
clear_day_df = merged_df[merged_df['conditions'] == 'clear-day'].copy()

clear_day_df['month'] = clear_day_df['time'].dt.month
clear_day_df['month_name'] = clear_day_df['month'].apply(lambda x: month_abbr[x])

plt.figure(figsize=(12, 3))

plt.scatter(clear_day_df['time'], clear_day_df['maxwaveheight'], label='Max Wave Height',
plt.scatter(clear_day_df['time'], clear_day_df['minwaveheight'], label='Min Wave Height',
plt.scatter(clear_day_df['time'], clear_day_df['averagewaveheight'], label='Average Wave

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends (Clear Day Conditions)')
plt.ylabel('Wave Height (m)')

month_index = clear_day_df.groupby('month').head(1).index

plt.xticks(clear_day_df.loc[month_index, 'time'], clear_day_df.loc[month_index, 'month_na
plt.ylim(0, 2.75)
plt.legend()
plt.show()
```

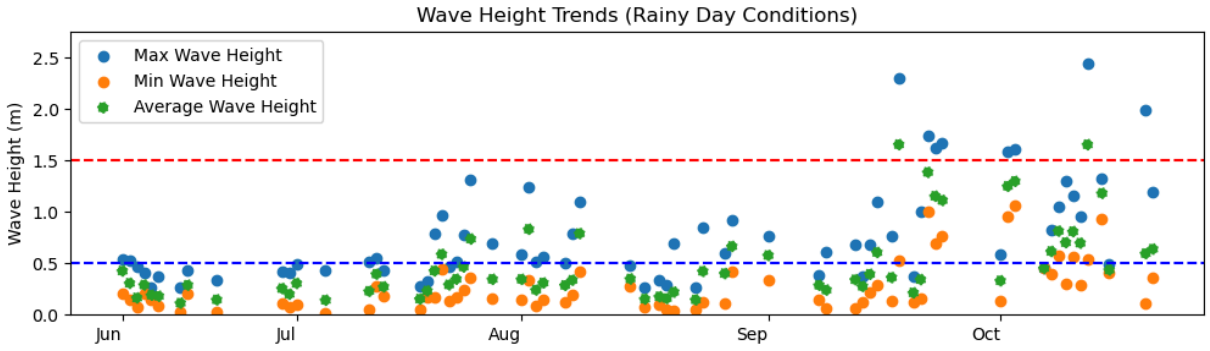


In [61]:

```
merged_df['time'] = pd.to_datetime(merged_df['datetime'])
# Filter data for each condition
cloudy_day_df = merged_df[merged_df['conditions'] == 'partly-cloudy-day'].copy()

cloudy_day_df['month'] = cloudy_day_df['time'].dt.month
cloudy_day_df['month_name'] = cloudy_day_df['month'].apply(lambda x: month_abbr[x])

plt.figure(figsize=(12, 3))

plt.scatter(cloudy_day_df['time'], cloudy_day_df['maxwaveheight'], label='Max Wave Height
plt.scatter(cloudy_day_df['time'], cloudy_day_df['minwaveheight'], label='Min Wave Height
plt.scatter(cloudy_day_df['time'], cloudy_day_df['averagewaveheight'], label='Average Wav

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends (Cloudy Day Conditions)')
plt.ylabel('Wave Height (m)')

month_index = cloudy_day_df.groupby('month').head(1).index

plt.xticks(cloudy_day_df.loc[month_index, 'time'], cloudy_day_df.loc[month_index, 'month_
plt.ylim(0, 2.75)
plt.legend()
plt.show()
```



In [62]:
```
rainy_day_df = merged_df[merged_df['conditions'] == 'rain'].copy()

rainy_day_df['month'] = rainy_day_df['time'].dt.month
rainy_day_df['month_name'] = rainy_day_df['month'].apply(lambda x: month_abbr[x])

plt.figure(figsize=(12, 3))

plt.scatter(rainy_day_df['time'], rainy_day_df['maxwaveheight'], label='Max Wave Height',
plt.scatter(rainy_day_df['time'], rainy_day_df['minwaveheight'], label='Min Wave Height',
plt.scatter(rainy_day_df['time'], rainy_day_df['averagewaveheight'], label='Average Wave

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends (Rainy Day Conditions)')
plt.ylabel('Wave Height (m)')

month_index = rainy_day_df.groupby('month').head(1).index

plt.xticks(rainy_day_df.loc[month_index, 'time'], rainy_day_df.loc[month_index, 'month_na
plt.ylim(0, 2.75)
plt.legend()
plt.show()
```



Hypothesis Testing

Does Air pressure influence wave height?

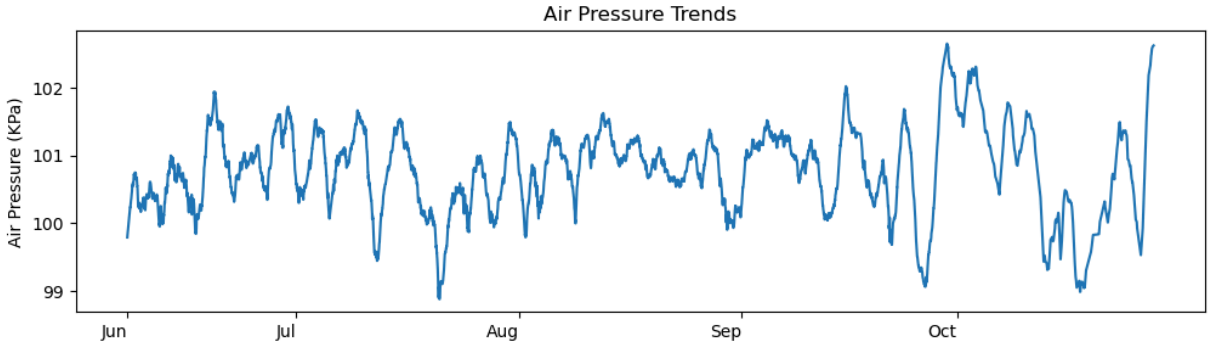
In [15]:
```
buoy_data.head()
```

Out[15]:

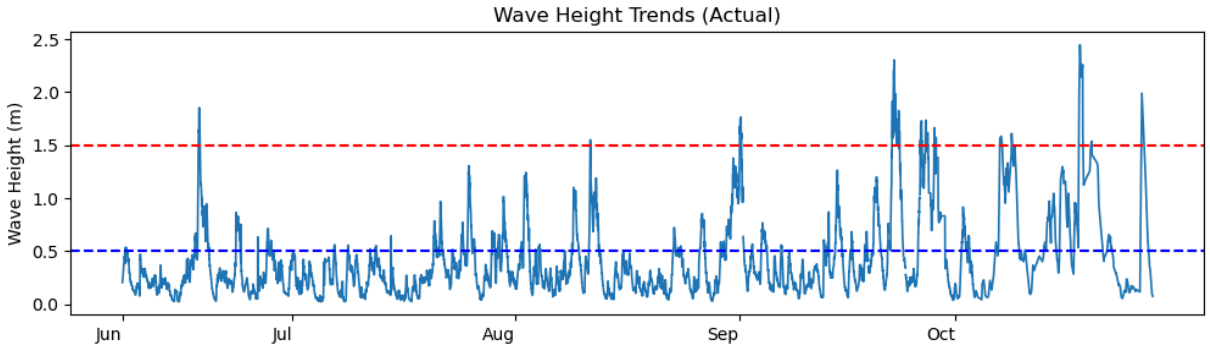
time	time (UTC)	air_pressure_at_mean_sea_level (Pa)	air_temperature (K)	sea_surface_wave_mean_period (s)
2022-06-07 11:20:00+00:00	2022-06-07T11:20:00Z	99793.98	282.48	NaN
2022-06-07 12:20:00+00:00	2022-06-07T12:20:00Z	99846.46	282.35	2.123
2022-06-07 18:20:00+00:00	2022-06-07T18:20:00Z	100114.20	283.50	3.460
2022-06-07 18:40:00+00:00	2022-06-07T18:40:00Z	100134.70	283.50	3.539
2022-06-07 19:00:00+00:00	2022-06-07T19:00:00Z	100152.10	283.56	3.587

5 rows × 36 columns

In [59]:
```
buoy_data['time'] = pd.to_datetime(buoy_data['time (UTC)'])
buoy_data['month'] = buoy_data['time'].dt.month
buoy_data['month_name'] = buoy_data['month'].apply(lambda x: month_abbr[x])

buoy_data['air_pressure_kpa'] = buoy_data['air_pressure_at_mean_sea_level (Pa)'] / 1000.0

plt.figure(figsize=(12, 3))

plt.plot(buoy_data['time'], buoy_data['air_pressure_kpa'], label='Air Pressure')

plt.title('Air Pressure Trends')
plt.ylabel('Air Pressure (KPa)')
month_index = buoy_data.groupby('month').head(1).index
plt.xticks(buoy_data.loc[month_index, 'time'], buoy_data.loc[month_index, 'month_name'],


plt.show()
```



In [62]:
```
buoy_data['time'] = pd.to_datetime(buoy_data['time (UTC)'])
buoy_data['month'] = buoy_data['time'].dt.month
buoy_data['month_name'] = buoy_data['month'].apply(lambda x: month_abbr[x])

plt.figure(figsize=(12, 3))

plt.plot(buoy_data['time'], buoy_data['sea_surface_wave_significant_height (m)'], label='

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends (Actual)')
plt.ylabel('Wave Height (m)')
month_index = buoy_data.groupby('month').head(1).index
plt.xticks(buoy_data.loc[month_index, 'time'], buoy_data.loc[month_index, 'month_name'],

plt.show()
```



In [61]:
```
correlation = buoy_data['air_pressure_kpa'].corr(buoy_data['sea_surface_wave_significant_
correlationround = round(correlation, 2)
print(f'Correlation: {correlationround}')
```

```
Correlation: -0.21
```

With a negative correlation of 0.21, we don't see a significant relationship in air pressure and wave height.

Does Wind direction influence wave height?

Windir is a categorical variable, however, having the values in degrees got it a correlation value of 0.41

Let's see if converting it to cardinal directions makes any difference.

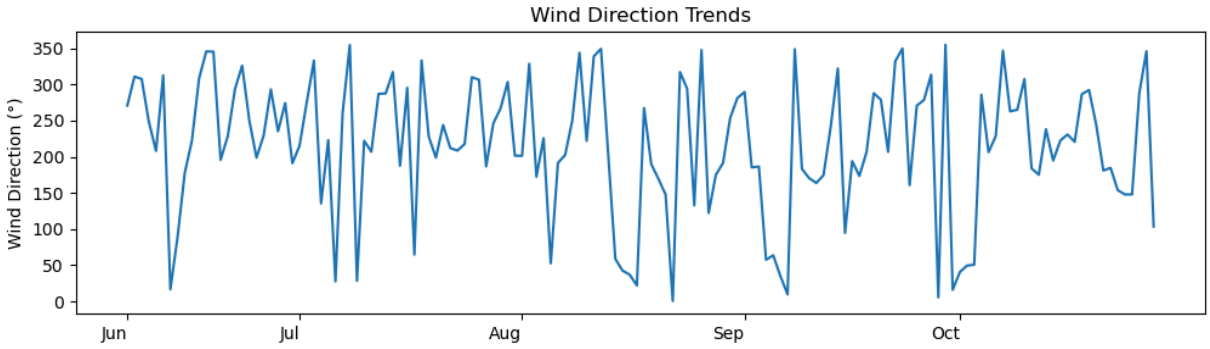
In [65]:

```
def wind_direction(x):
    if 0 <= x < 45:
        return 'N-NE'
    elif 45 <= x < 90:
        return 'NE-E'
    elif 90 <= x < 135:
        return 'E-SE'
    elif 135 <= x < 180:
        return 'SE-S'
    elif 180 <= x < 225:
        return 'S-SW'
    elif 225 <= x < 270:
        return 'SW-W'
    elif 270 <= x < 315:
        return 'W-NW'
    elif 315 <= x < 360:
        return 'NW-N'
    else:
        return 'Unknown'

merged_df['windir_cat'] = merged_df['winddir'].apply(wind_direction)

print(merged_df)
```

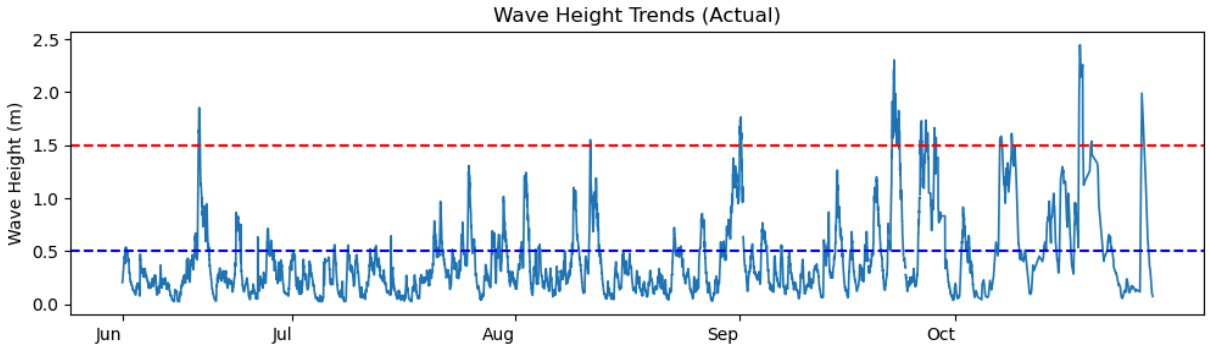
```
      datetime  tempmax  tempmin  temp  feelslikemax  feelslikemin  feelslike  \
0   2022-06-07     17.6     10.7  13.9          17.6          10.7       13.9
1   2022-06-08     17.0      8.6  13.8          17.0           8.6       13.8
2   2022-06-09     17.7     11.1  13.3          17.7          11.1       13.3
3   2022-06-10     19.9      8.7  14.3          19.9           8.7       14.2
4   2022-06-11     23.0     12.5  17.5          23.0          12.5       17.5
..         ...      ...      ...   ...           ...           ...        ...
139 2022-10-24     22.6     10.8  16.0          22.6          10.8       16.0
140 2022-10-25     20.9     13.1  16.3          20.9          13.1       16.3
141 2022-10-26     16.5      6.8  12.1          16.5           2.7       10.7
142 2022-10-27      8.9      2.1   5.8           7.1          -0.7        3.6
143 2022-10-28     12.2      1.1   5.7          12.2          -1.7        3.7

      dew  humidity  precip  ...  visibility  uvindex         conditions  \
0    12.2      89.6  37.380  ...        10.0        4               rain
1    10.2      79.4   0.839  ...        22.0        9               rain
2    10.7      84.4  12.976  ...        21.2       10               rain
3     9.0      72.6   0.063  ...        24.0       10               rain
4    11.4      69.1   0.207  ...        22.3       10               rain
..    ...       ...     ...  ...         ...      ...                ...
139   6.1      53.1   0.000  ...        24.0        6  partly-cloudy-day
140   8.2      59.3   0.000  ...        24.0        5  partly-cloudy-day
141   8.2      78.6   4.261  ...        20.4        7               rain
142   0.9      71.9   0.150  ...        24.0        5               rain
143   0.7      72.0   0.000  ...        24.0        6  partly-cloudy-day

     maxwaveheight  minwaveheight  averagewaveheight       time  month  \
0            0.536          0.204           0.426789 2022-06-07      6
1            0.533          0.141           0.307200 2022-06-08      6
2            0.470          0.077           0.164507 2022-06-09      6
3            0.409          0.205           0.290471 2022-06-10      6
4            0.271          0.150           0.191900 2022-06-11      6
..             ...            ...                ...        ...    ...
139          0.237          0.053           0.146818 2022-10-24     10
140          0.173          0.107           0.141100 2022-10-25     10
141          1.991          0.115           0.596000 2022-10-26     10
142          1.191          0.361           0.639500 2022-10-27     10
143          0.324          0.074           0.182833 2022-10-28     10

    month_name  windir_cat
0          Jun        W-NW
1          Jun        W-NW
2          Jun        W-NW
3          Jun        SW-W
4          Jun        S-SW
..         ...         ...
139        Oct        SE-S
140        Oct        SE-S
141        Oct        W-NW
142        Oct        NW-N
143        Oct        E-SE

[144 rows x 26 columns]
```

Winddir trends over time in degrees

In [71]:
```
plt.figure(figsize=(12, 3))

plt.plot(merged_df['datetime'], merged_df['winddir'], label='Cloud Cover')

plt.title('Wind Direction Trends')
plt.ylabel('Wind Direction (°)')
month_index = merged_df.groupby('month').head(1).index
plt.xticks(merged_df.loc[month_index, 'time'], merged_df.loc[month_index, 'month_name'],

plt.show()
```



In [72]:
```
plt.figure(figsize=(12, 3))

plt.plot(buoy_data['time'], buoy_data['sea_surface_wave_significant_height (m)'], label='

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends (Actual)')
plt.ylabel('Wave Height (m)')
month_index = buoy_data.groupby('month').head(1).index
plt.xticks(buoy_data.loc[month_index, 'time'], buoy_data.loc[month_index, 'month_name'],

plt.show()
```



In [90]:
```
correlation = merged_df['winddir'].corr(merged_df['maxwaveheight'])
correlationround = round(correlation, 2)
print(f'Correlation: {correlationround}')
```

```
Correlation: 0.41
```

After converting to cardinal directions we see that we only have high waves > 1.5m when winds blow between SW and NE. And the smaller range within this, when winds blow between N and W, has waves even higher than 2.3 m.

In [85]:
```
x = merged_df.groupby('windir_cat')['maxwaveheight'].agg(['count','mean','median','min','
y = x.sort_values(by='max', ascending=False)
y
```

Out[85]:

windir_cat	count	mean	median	min	max
W-NW	28	0.942071	0.6635	0.228	2.447
NW-N	20	1.082350	1.0240	0.276	2.305
N-NE	12	0.576667	0.4910	0.215	1.665
SW-W	21	0.752429	0.6740	0.261	1.611
S-SW	36	0.519389	0.4630	0.189	1.297
NE-E	8	0.462125	0.4125	0.137	0.915
E-SE	4	0.526500	0.5190	0.271	0.797
SE-S	15	0.336533	0.3050	0.173	0.606

Are there hourly trends in wave heights?

In [143]:
```
buoy_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8267 entries, 2022-06-07 11:20:00+00:00 to 2022-10-28 08:40:00+00:00
Data columns (total 40 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   time (UTC)                                8267 non-null   object
 1   air_pressure_at_mean_sea_level (Pa)       8267 non-null   float64
 2   air_temperature (K)                       8267 non-null   float64
 3   sea_surface_wave_mean_period (s)          8245 non-null   float64
 4   sea_surface_wave_significant_height (m)   8246 non-null   float64
 5   sea_surface_wave_from_direction (degree)  8246 non-null   float64
 6   wind_from_direction (degree)              8267 non-null   float64
 7   wind_speed_of_gust (m s-1)                8267 non-null   float64
 8   wind_speed (m s-1)                        8267 non-null   float64
 9   eastward_sea_water_velocity_1 (m s-1)     8237 non-null   float64
 10  northward_sea_water_velocity_1 (m s-1)    8236 non-null   float64
 11  sea_water_temperature_1 (K)               8267 non-null   float64
 12  eastward_sea_water_velocity_2 (m s-1)     8237 non-null   float64
 13  northward_sea_water_velocity_2 (m s-1)    8236 non-null   float64
 14  sea_water_temperature_2 (K)               8267 non-null   float64
 15  eastward_sea_water_velocity_3 (m s-1)     8237 non-null   float64
 16  northward_sea_water_velocity_3 (m s-1)    8236 non-null   float64
 17  sea_water_temperature_3 (K)               8267 non-null   float64
 18  eastward_sea_water_velocity_4 (m s-1)     8237 non-null   float64
 19  northward_sea_water_velocity_4 (m s-1)    8236 non-null   float64
 20  sea_water_temperature_4 (K)               8267 non-null   float64
 21  eastward_sea_water_velocity_5 (m s-1)     8237 non-null   float64
 22  northward_sea_water_velocity_5 (m s-1)    8236 non-null   float64
 23  sea_water_temperature_5 (K)               8267 non-null   float64
 24  eastward_sea_water_velocity_6 (m s-1)     8237 non-null   float64
 25  northward_sea_water_velocity_6 (m s-1)    8236 non-null   float64
 26  sea_water_temperature_6 (K)               8267 non-null   float64
 27  eastward_sea_water_velocity_7 (m s-1)     8237 non-null   float64
 28  northward_sea_water_velocity_7 (m s-1)    8236 non-null   float64
 29  sea_water_temperature_7 (K)               8267 non-null   float64
 30  eastward_sea_water_velocity_8 (m s-1)     8237 non-null   float64
 31  northward_sea_water_velocity_8 (m s-1)    8236 non-null   float64
 32  sea_water_temperature_8 (K)               8267 non-null   float64
 33  northward_sea_water_velocity (m s-1)      0 non-null      float64
 34  sea_water_temperature_9 (K)               8267 non-null   float64
 35  platform                                  8267 non-null   int64
 36  month                                     8267 non-null   int64
 37  month_name                                8267 non-null   object
 38  air_pressure_kpa                          8267 non-null   float64
 39  hour                                      8267 non-null   int64
dtypes: float64(35), int64(3), object(2)
memory usage: 2.6+ MB
```

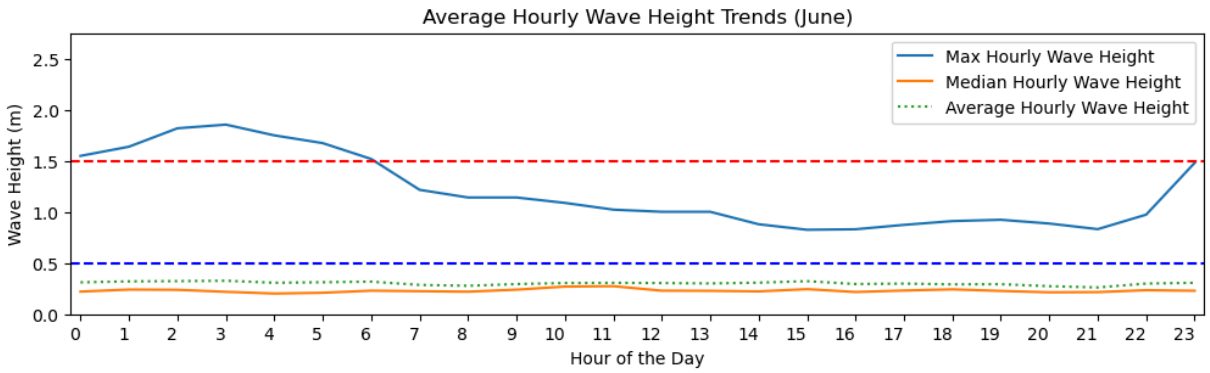
In [93]:
```
buoy_data.head()
```

Out[93]:

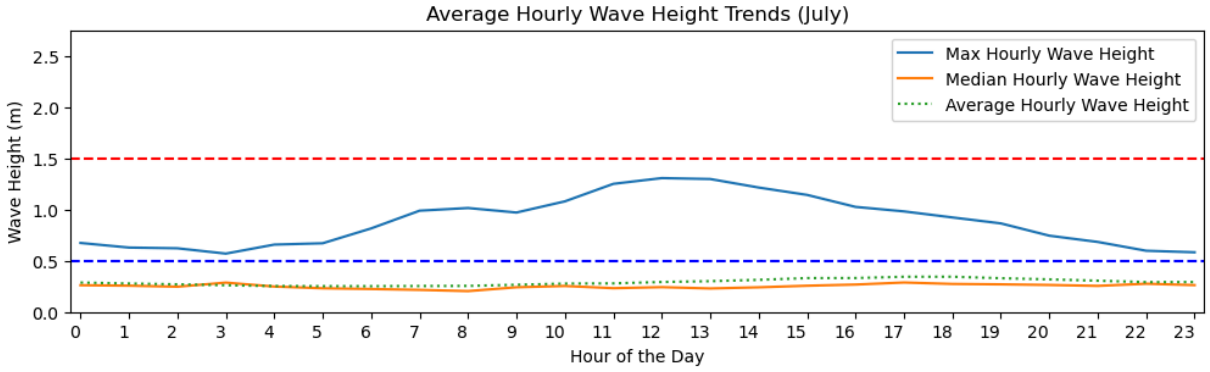
time	time (UTC)	air_pressure_at_mean_sea_level (Pa)	air_temperature (K)	sea_surface_wave_mean_period (s)
2022-06-07 11:20:00+00:00	2022-06-07T11:20:00Z	99793.98	282.48	NaN
2022-06-07 12:20:00+00:00	2022-06-07T12:20:00Z	99846.46	282.35	2.123
2022-06-07 18:20:00+00:00	2022-06-07T18:20:00Z	100114.20	283.50	3.460
2022-06-07 18:40:00+00:00	2022-06-07T18:40:00Z	100134.70	283.50	3.539
2022-06-07 19:00:00+00:00	2022-06-07T19:00:00Z	100152.10	283.56	3.587

5 rows × 41 columns

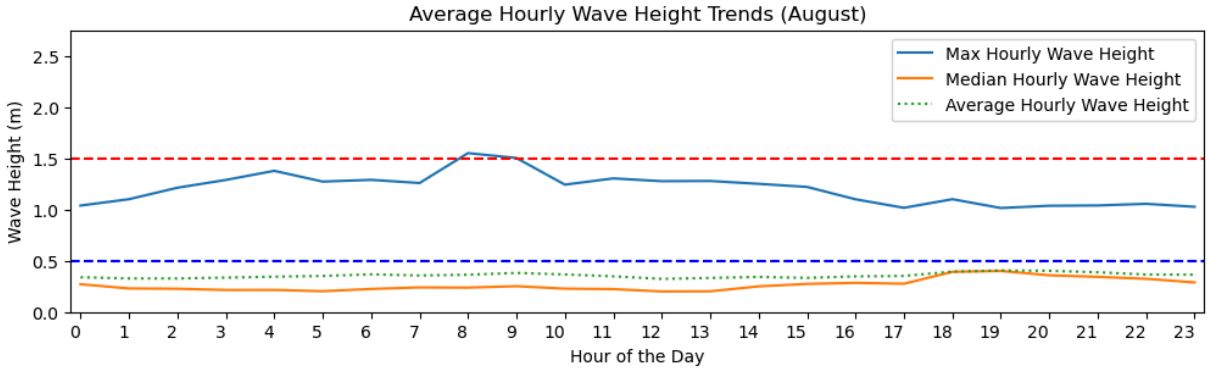
In [124]:
```
buoy_data['time'] = pd.to_datetime(buoy_data['time (UTC)'])
buoy_data['hour'] = buoy_data['time'].dt.hour

june_data = buoy_data[buoy_data['month'] == 6]

plt.figure(figsize=(12, 3))

# Group by hour and calculate the mean, max, and min for each hour
average_hourly_wave = june_data.groupby('hour')['sea_surface_wave_significant_height (m)'

# Plot each aggregated value with a separate label
plt.plot(average_hourly_wave.index, average_hourly_wave['max'], label='Max Hourly Wave He
plt.plot(average_hourly_wave.index, average_hourly_wave['median'], label='Median Hourly W
plt.plot(average_hourly_wave.index, average_hourly_wave['mean'], label='Average Hourly Wa

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Average Hourly Wave Height Trends (June)')
plt.ylabel('Wave Height (m)')
plt.xlabel('Hour of the Day')
plt.ylim(0, 2.75)
plt.xlim(-0.2, 23.2)
plt.xticks(range(24), labels=[str(i) for i in range(24)], rotation=0, ha='right')

plt.legend()

plt.show()
```



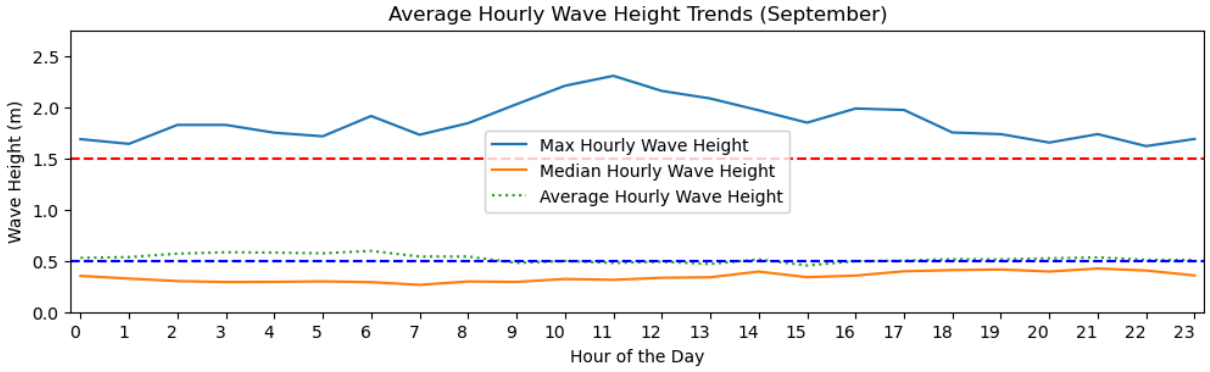
In [122]:
```
june_data = buoy_data[buoy_data['month'] == 7]

plt.figure(figsize=(12, 3))

# Group by hour and calculate the mean, max, and min for each hour
average_hourly_wave = june_data.groupby('hour')['sea_surface_wave_significant_height (m)'

# Plot each aggregated value with a separate label
plt.plot(average_hourly_wave.index, average_hourly_wave['max'], label='Max Hourly Wave He
plt.plot(average_hourly_wave.index, average_hourly_wave['median'], label='Median Hourly W
plt.plot(average_hourly_wave.index, average_hourly_wave['mean'], label='Average Hourly Wa

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Average Hourly Wave Height Trends (July)')
plt.ylabel('Wave Height (m)')
plt.xlabel('Hour of the Day')
plt.ylim(0, 2.75)
plt.xlim(-0.2, 23.2)
plt.xticks(range(24), labels=[str(i) for i in range(24)], rotation=0, ha='right')

plt.legend()

plt.show()
```



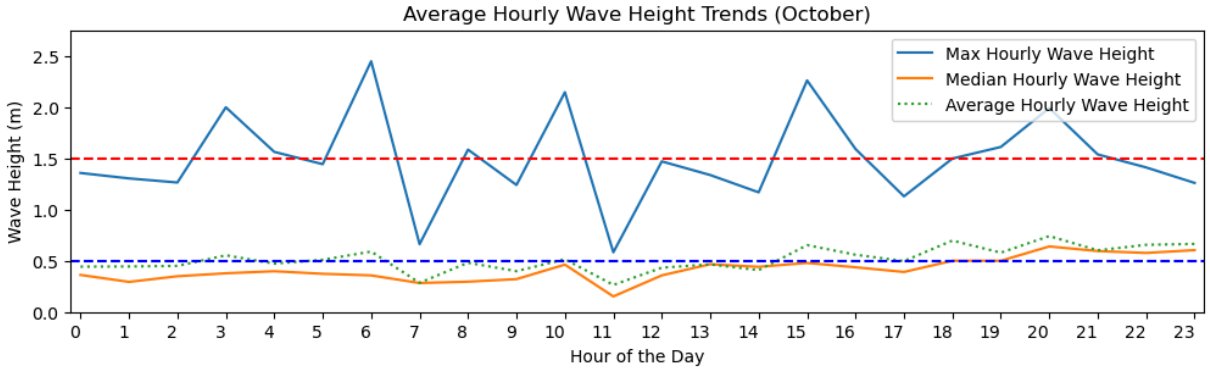
In [121]:
```
june_data = buoy_data[buoy_data['month'] == 8]

plt.figure(figsize=(12, 3))

# Group by hour and calculate the mean, max, and min for each hour
average_hourly_wave = june_data.groupby('hour')['sea_surface_wave_significant_height (m)'

# Plot each aggregated value with a separate label
plt.plot(average_hourly_wave.index, average_hourly_wave['max'], label='Max Hourly Wave He
plt.plot(average_hourly_wave.index, average_hourly_wave['median'], label='Median Hourly W
plt.plot(average_hourly_wave.index, average_hourly_wave['mean'], label='Average Hourly Wa

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Average Hourly Wave Height Trends (August)')
plt.ylabel('Wave Height (m)')
plt.xlabel('Hour of the Day')
plt.ylim(0, 2.75)
plt.xlim(-0.2, 23.2)
plt.xticks(range(24), labels=[str(i) for i in range(24)], rotation=0, ha='right')

plt.legend()

plt.show()
```



In [125]:
```
june_data = buoy_data[buoy_data['month'] == 9]

plt.figure(figsize=(12, 3))

# Group by hour and calculate the mean, max, and min for each hour
average_hourly_wave = june_data.groupby('hour')['sea_surface_wave_significant_height (m)'

# Plot each aggregated value with a separate label
plt.plot(average_hourly_wave.index, average_hourly_wave['max'], label='Max Hourly Wave He
plt.plot(average_hourly_wave.index, average_hourly_wave['median'], label='Median Hourly W
plt.plot(average_hourly_wave.index, average_hourly_wave['mean'], label='Average Hourly Wa

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Average Hourly Wave Height Trends (September)')
plt.ylabel('Wave Height (m)')
plt.xlabel('Hour of the Day')
plt.ylim(0, 2.75)
plt.xlim(-0.2, 23.2)
plt.xticks(range(24), labels=[str(i) for i in range(24)], rotation=0, ha='right')

plt.legend()

plt.show()
```

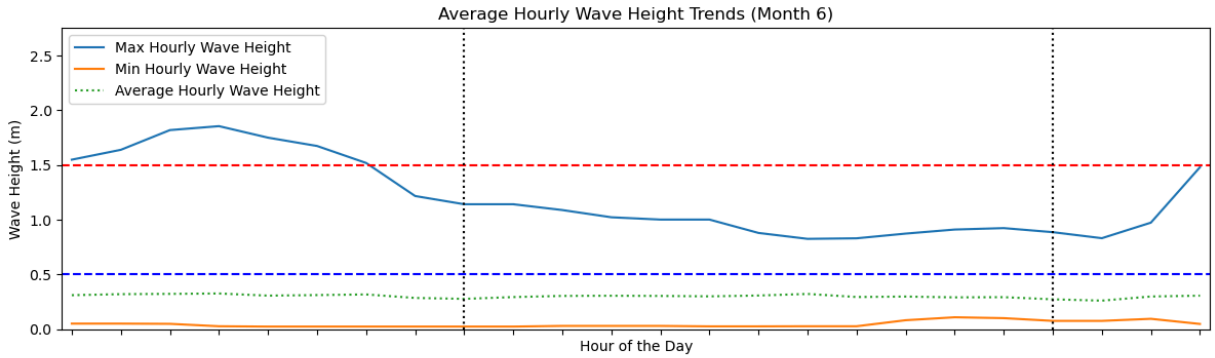


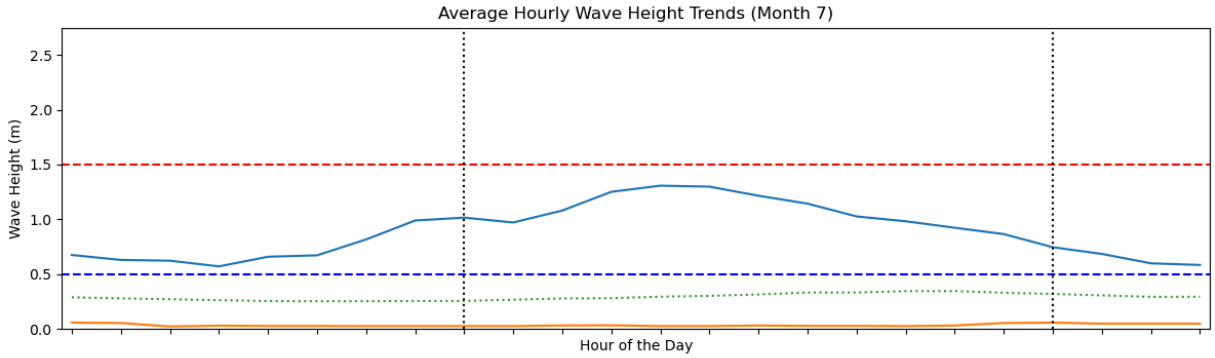
In [126]:
```
june_data = buoy_data[buoy_data['month'] == 10]

plt.figure(figsize=(12, 3))

# Group by hour and calculate the mean, max, and min for each hour
average_hourly_wave = june_data.groupby('hour')['sea_surface_wave_significant_height (m)'

# Plot each aggregated value with a separate label
plt.plot(average_hourly_wave.index, average_hourly_wave['max'], label='Max Hourly Wave He
plt.plot(average_hourly_wave.index, average_hourly_wave['median'], label='Median Hourly W
plt.plot(average_hourly_wave.index, average_hourly_wave['mean'], label='Average Hourly Wa

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Average Hourly Wave Height Trends (October)')
plt.ylabel('Wave Height (m)')
plt.xlabel('Hour of the Day')
plt.ylim(0, 2.75)
plt.xlim(-0.2, 23.2)
plt.xticks(range(24), labels=[str(i) for i in range(24)], rotation=0, ha='right')

plt.legend()

plt.show()
```

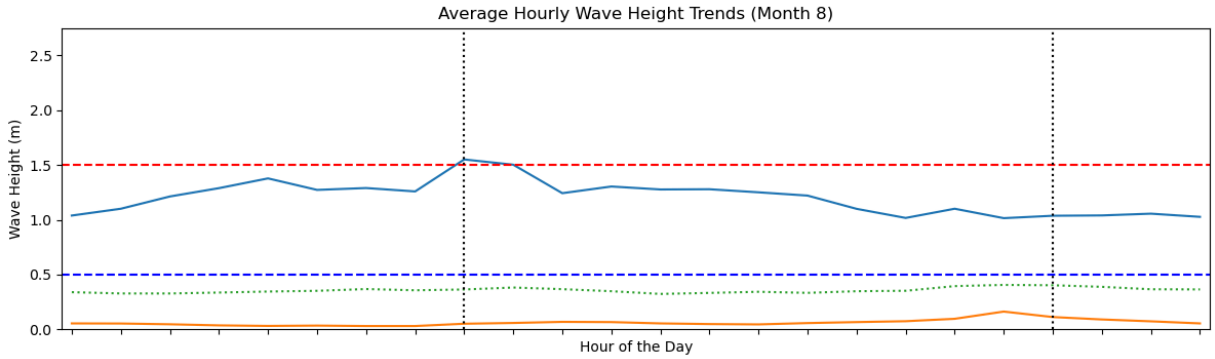


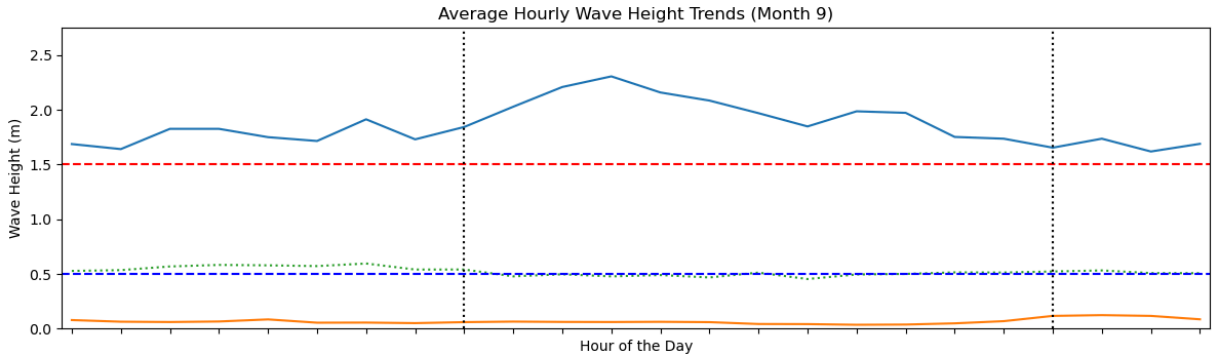
In [155]:

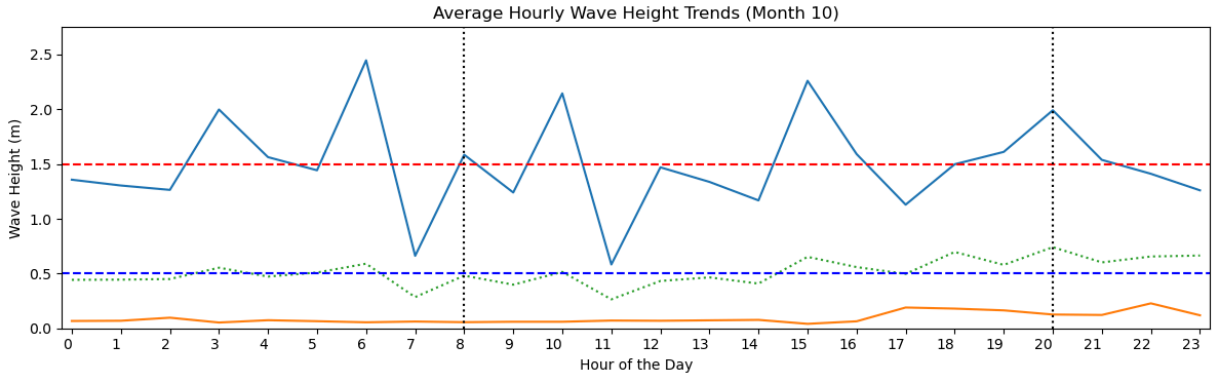
```
import matplotlib.pyplot as plt
import pandas as pd
from calendar import month_abbr

# Assuming buoy_data is already available
buoy_data['time'] = pd.to_datetime(buoy_data['time (UTC)'])
buoy_data['month'] = buoy_data['time'].dt.month

# Group by month and hour, and calculate the mean, max, and min for each hour
monthly_hourly_wave = buoy_data.groupby(['month', 'hour'])['sea_surface_wave_significant_

# Set up subplots
fig, axes = plt.subplots(nrows=5, ncols=1, figsize=(12, 18), sharex=True, sharey=True)

# Flatten the 2D array of axes
axes = axes.flatten()

# Loop through each month
for month in range(6, 11):  # Assuming months are from June to October
    # Filter data for the current month
    monthly_data = monthly_hourly_wave[monthly_hourly_wave['month'] == month]

    # Plot each aggregated value with a separate label
    axes[month - 6].plot(monthly_data['hour'], monthly_data['max'], label='Max Hourly Wav
    axes[month - 6].plot(monthly_data['hour'], monthly_data['min'], label='Min Hourly Wav
    axes[month - 6].plot(monthly_data['hour'], monthly_data['mean'], label='Average Hourl

    axes[month - 6].axhline(y=0.5, color='b', linestyle='--')
    axes[month - 6].axhline(y=1.5, color='r', linestyle='--')
    axes[month - 6].axvline(x=8, color='k', linestyle='dotted')
    axes[month - 6].axvline(x=20, color='k', linestyle='dotted')

    axes[month - 6].set_title(f'Average Hourly Wave Height Trends (Month {month})')
    axes[month - 6].set_ylabel('Wave Height (m)')
    axes[month - 6].set_xlabel('Hour of the Day')
    axes[month - 6].set_xticks(range(24), labels=[str(i) for i in range(24)], rotation=0,
    axes[month - 6].set_ylim(0, 2.75)
    axes[month - 6].set_xlim(-0.2, 23.2)


axes[0].legend()

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


Average Hourly Wave Height Trends (Month 6)
Max Hourly Wave Height
Min Hourly Wave Height
Average Hourly Wave Height
Wave Height (m)
Hour of the Day


Average Hourly Wave Height Trends (Month 7)
Wave Height (m)
Hour of the Day

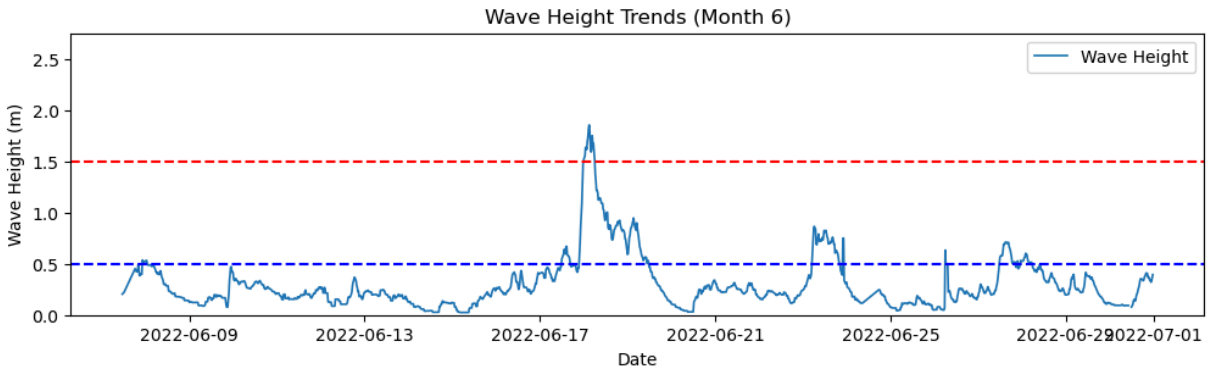

Average Hourly Wave Height Trends (Month 8)
Wave Height (m)
Hour of the Day

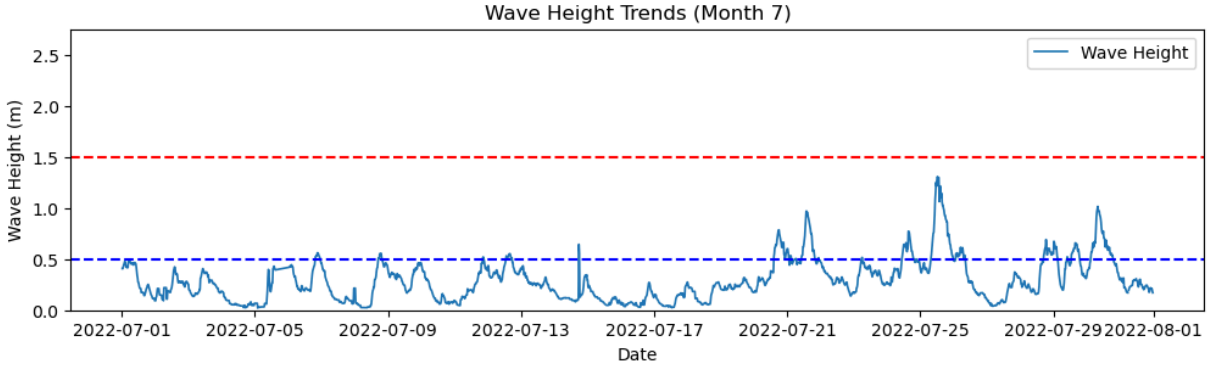

Average Hourly Wave Height Trends (Month 9)
Wave Height (m)
Hour of the Day

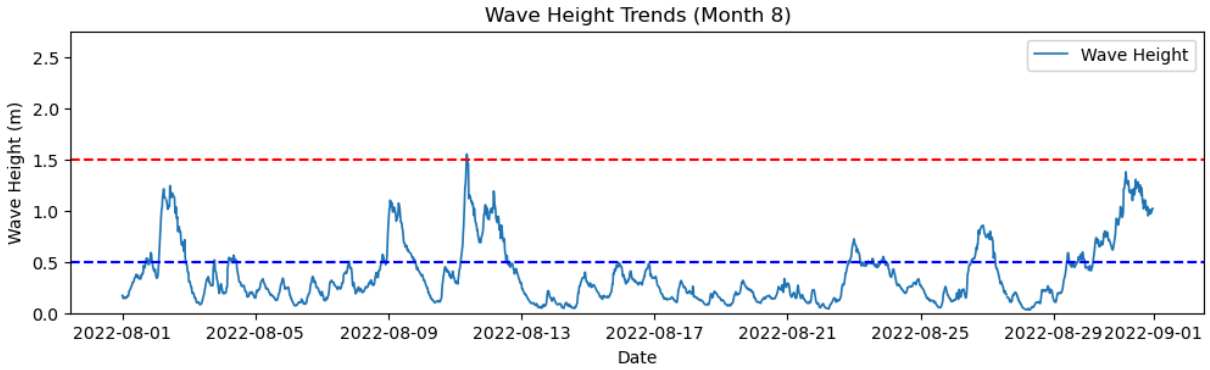

Average Hourly Wave Height Trends (Month 10)
Wave Height (m)
Hour of the Day

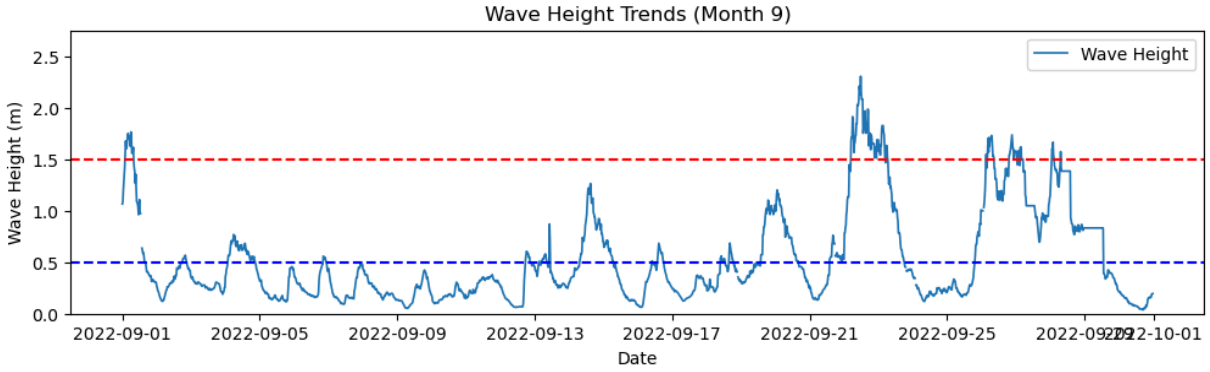
Are there monthly intervals in high wave occurences?

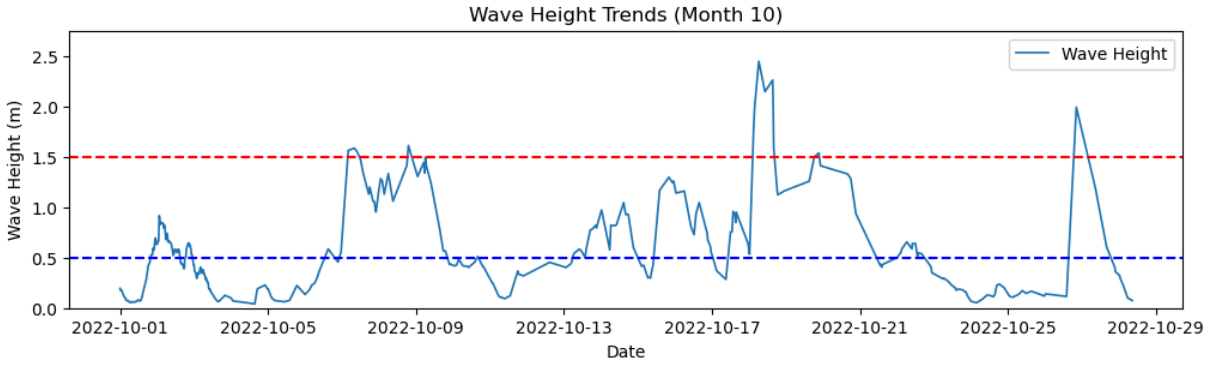
In [140]:
```
import matplotlib.pyplot as plt
import pandas as pd
from calendar import month_abbr

# Assuming buoy_data is already available
buoy_data['time'] = pd.to_datetime(buoy_data['time (UTC)'])
buoy_data['month'] = buoy_data['time'].dt.month
buoy_data['month_name'] = buoy_data['month'].apply(lambda x: month_abbr[x])

# Iterate over each month
for month in range(6, 11):  # Assuming months are from June to October
    # Filter data for the current month
    monthly_data = buoy_data[buoy_data['month'] == month]

    plt.figure(figsize=(12, 3))

    # Plot wave height for the entire month
    plt.plot(monthly_data['time'], monthly_data['sea_surface_wave_significant_height (m)'

    plt.axhline(y=0.5, color='b', linestyle='--')
    plt.axhline(y=1.5, color='r', linestyle='--')

    plt.title(f'Wave Height Trends (Month {month})')
    plt.ylabel('Wave Height (m)')
    plt.xlabel('Date')
    plt.ylim(0, 2.75)

    plt.legend()

    plt.show()
```











When do more high waves occur? During the day or at night?

In [141]:
```
buoy_data['time'] = pd.to_datetime(buoy_data['time (UTC)'])

buoy_data.set_index('time', inplace=True)

filtered_data = buoy_data.between_time('08:00:00', '20:00:00')
filtered_data.reset_index(inplace=True)
```

In [142]:
```
filtered_data.info()  #From 8000 rows we went to around 4000
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4207 entries, 0 to 4206
Data columns (total 41 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   time                                      4207 non-null   datetime64[ns, UTC]
 1   time (UTC)                                4207 non-null   object
 2   air_pressure_at_mean_sea_level (Pa)       4207 non-null   float64
 3   air_temperature (K)                       4207 non-null   float64
 4   sea_surface_wave_mean_period (s)          4192 non-null   float64
 5   sea_surface_wave_significant_height (m)   4193 non-null   float64
 6   sea_surface_wave_from_direction (degree)  4193 non-null   float64
 7   wind_from_direction (degree)              4207 non-null   float64
 8   wind_speed_of_gust (m s-1)                4207 non-null   float64
 9   wind_speed (m s-1)                        4207 non-null   float64
 10  eastward_sea_water_velocity_1 (m s-1)     4198 non-null   float64
 11  northward_sea_water_velocity_1 (m s-1)    4198 non-null   float64
 12  sea_water_temperature_1 (K)               4207 non-null   float64
 13  eastward_sea_water_velocity_2 (m s-1)     4198 non-null   float64
 14  northward_sea_water_velocity_2 (m s-1)    4198 non-null   float64
 15  sea_water_temperature_2 (K)               4207 non-null   float64
 16  eastward_sea_water_velocity_3 (m s-1)     4198 non-null   float64
 17  northward_sea_water_velocity_3 (m s-1)    4198 non-null   float64
 18  sea_water_temperature_3 (K)               4207 non-null   float64
 19  eastward_sea_water_velocity_4 (m s-1)     4198 non-null   float64
 20  northward_sea_water_velocity_4 (m s-1)    4198 non-null   float64
 21  sea_water_temperature_4 (K)               4207 non-null   float64
 22  eastward_sea_water_velocity_5 (m s-1)     4198 non-null   float64
 23  northward_sea_water_velocity_5 (m s-1)    4198 non-null   float64
 24  sea_water_temperature_5 (K)               4207 non-null   float64
 25  eastward_sea_water_velocity_6 (m s-1)     4198 non-null   float64
 26  northward_sea_water_velocity_6 (m s-1)    4198 non-null   float64
 27  sea_water_temperature_6 (K)               4207 non-null   float64
 28  eastward_sea_water_velocity_7 (m s-1)     4198 non-null   float64
 29  northward_sea_water_velocity_7 (m s-1)    4198 non-null   float64
 30  sea_water_temperature_7 (K)               4207 non-null   float64
 31  eastward_sea_water_velocity_8 (m s-1)     4198 non-null   float64
 32  northward_sea_water_velocity_8 (m s-1)    4198 non-null   float64
 33  sea_water_temperature_8 (K)               4207 non-null   float64
 34  northward_sea_water_velocity (m s-1)      0 non-null      float64
 35  sea_water_temperature_9 (K)               4207 non-null   float64
 36  platform                                  4207 non-null   int64
 37  month                                     4207 non-null   int64
 38  month_name                                4207 non-null   object
 39  air_pressure_kpa                          4207 non-null   float64
 40  hour                                      4207 non-null   int64
dtypes: datetime64[ns, UTC](1), float64(35), int64(3), object(2)
memory usage: 1.3+ MB
```

In [144]:
```
filtered_data.head()
```

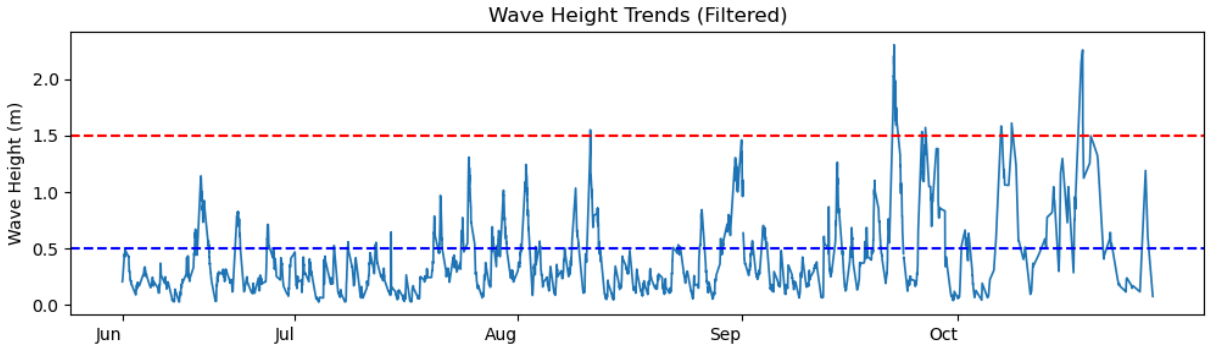
Out[144]:

	time	time (UTC)	air_pressure_at_mean_sea_level (Pa)	air_temperature (K)	sea_surface_wave_mean_peri(
0	2022-06-07 11:20:00+00:00	2022-06-07T11:20:00Z	99793.98	282.48	Na
1	2022-06-07 12:20:00+00:00	2022-06-07T12:20:00Z	99846.46	282.35	2.1
2	2022-06-07 18:20:00+00:00	2022-06-07T18:20:00Z	100114.20	283.50	3.4
3	2022-06-07 18:40:00+00:00	2022-06-07T18:40:00Z	100134.70	283.50	3.5
4	2022-06-07 19:00:00+00:00	2022-06-07T19:00:00Z	100152.10	283.56	3.5

5 rows × 41 columns

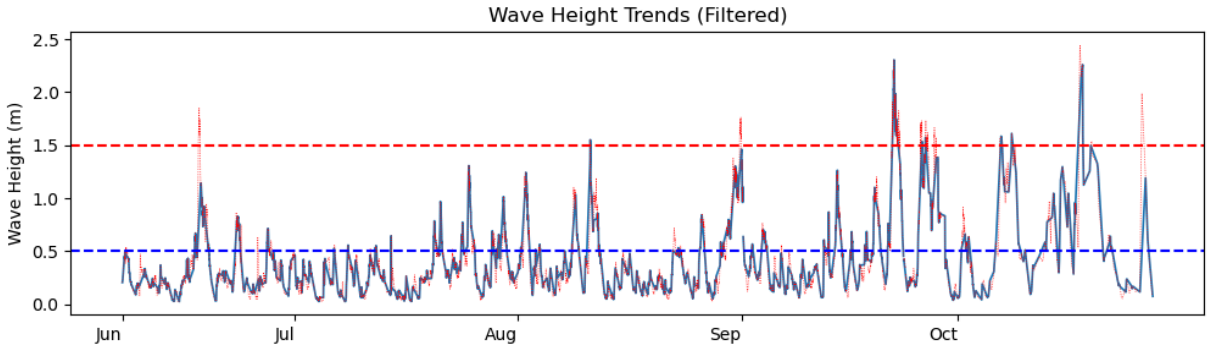
Wave heights during day time (8AM - 8PM)

In [152]:
```
plt.figure(figsize=(12, 3))

plt.plot(filtered_data['time'], filtered_data['sea_surface_wave_significant_height (m)'],

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends (Filtered)')
plt.ylabel('Wave Height (m)')
month_index = filtered_data.groupby('month').head(1).index
plt.xticks(filtered_data.loc[month_index, 'time'], filtered_data.loc[month_index, 'month_

plt.show()
```



Day time waves vs Night time waves

In [166]:
```
plt.figure(figsize=(12, 3))

plt.plot(filtered_data['time'], filtered_data['sea_surface_wave_significant_height (m)'],
plt.plot(buoy_data['time'], buoy_data['sea_surface_wave_significant_height (m)'], label='

plt.axhline(y=0.5, color='b', linestyle='--')
plt.axhline(y=1.5, color='r', linestyle='--')

plt.title('Wave Height Trends (Filtered)')
plt.ylabel('Wave Height (m)')
month_index = filtered_data.groupby('month').head(1).index
plt.xticks(filtered_data.loc[month_index, 'time'], filtered_data.loc[month_index, 'month_

plt.show()
```



In [147]:
```
filtered_data.head()
```

Out[147]:

	time	time (UTC)	air_pressure_at_mean_sea_level (Pa)	air_temperature (K)	sea_surface_wave_mean_perio (
0	2022-06-07 11:20:00+00:00	2022-06-07T11:20:00Z	99793.98	282.48	Na
1	2022-06-07 12:20:00+00:00	2022-06-07T12:20:00Z	99846.46	282.35	2.1
2	2022-06-07 18:20:00+00:00	2022-06-07T18:20:00Z	100114.20	283.50	3.4
3	2022-06-07 18:40:00+00:00	2022-06-07T18:40:00Z	100134.70	283.50	3.5
4	2022-06-07 19:00:00+00:00	2022-06-07T19:00:00Z	100152.10	283.56	3.5

5 rows × 41 columns

Save all dataframes

In [148]:
```
filtered_data.to_csv('filtered_data.csv', index=False)
```

In [149]:
```
merged_df.to_csv('merged_df.csv', index=False)
```

In [150]:
```
buoy_data.to_csv('buoy_df.csv', index=False)
```

In [151]:
```
Wdf.to_csv('Wdf.csv', index=False)
```

When do more high waves occur? Day with highest frequency?

Check when high waves occur and understand if beach usage data is available for all critical dates

In [172]:
```
count_wave_height = (buoy_data['sea_surface_wave_significant_height (m)'] > 1.5).sum()
print(f"Wave height > 1.5m count: {count_wave_height}")
```

```
Wave height > 1.5m count: 171
```

In [173]:
```
count_wave_height = (filtered_data['sea_surface_wave_significant_height (m)'] > 1.5).sum(
print(f"Daytime Wave height > 1.5m count: {count_wave_height}")
```

```
Daytime Wave height > 1.5m count: 53
```

A total of 53 waves occured during daytime from the 171 high wave occurences over the 5 month period

Let's see when these incidents took place

Daytime high wave days

In [183]:
```
print(filtered_data.loc[filtered_data['sea_surface_wave_significant_height (m)'] > 1.5, '
```

```
['Thursday, 2022-08-11' 'Thursday, 2022-09-22' 'Monday, 2022-09-26'
 'Friday, 2022-10-07' 'Saturday, 2022-10-08' 'Tuesday, 2022-10-18']
```

All high wave days

In [249]:
```
print(buoy_data.loc[buoy_data['sea_surface_wave_significant_height (m)'] > 1.5, 'time'].d
```

```
['Saturday, 2022-06-18' 'Thursday, 2022-08-11' 'Thursday, 2022-09-01'
 'Thursday, 2022-09-22' 'Friday, 2022-09-23' 'Monday, 2022-09-26'
 'Tuesday, 2022-09-27' 'Wednesday, 2022-09-28' 'Friday, 2022-10-07'
 'Saturday, 2022-10-08' 'Tuesday, 2022-10-18' 'Wednesday, 2022-10-19'
 'Wednesday, 2022-10-26']
```

Data for each high wave occurance during the day

In [186]:
```
# Filter data where wave height is above 1.5m
high_wave_height_days = filtered_data[filtered_data['sea_surface_wave_significant_height

# Display the dates of the high wave height days
print(high_wave_height_days['time'].dt.date.unique())

# Extract numerical weekday information and display the results
high_wave_height_days['weekday'] = high_wave_height_days['time'].dt.strftime('%A')
high_wave_height_days['waveheight'] = high_wave_height_days['sea_surface_wave_significant

# Display the dates and corresponding numerical weekdays of the high wave height days
print(high_wave_height_days[['time', 'weekday', 'waveheight']])
```

	time	weekday	waveheight
2276	2022-08-11 08:20:00+00:00	Thursday	1.513
2277	2022-08-11 08:40:00+00:00	Thursday	1.551
2278	2022-08-11 09:00:00+00:00	Thursday	1.503
3717	2022-09-22 08:00:00+00:00	Thursday	1.722
3718	2022-09-22 08:20:00+00:00	Thursday	1.733
3719	2022-09-22 08:40:00+00:00	Thursday	1.843
3720	2022-09-22 09:00:00+00:00	Thursday	1.843
3721	2022-09-22 09:20:00+00:00	Thursday	2.028
3722	2022-09-22 09:40:00+00:00	Thursday	2.027
3723	2022-09-22 10:00:00+00:00	Thursday	2.024
3724	2022-09-22 10:20:00+00:00	Thursday	2.076
3725	2022-09-22 10:40:00+00:00	Thursday	2.208
3726	2022-09-22 11:00:00+00:00	Thursday	2.175
3727	2022-09-22 11:20:00+00:00	Thursday	2.129
3728	2022-09-22 11:40:00+00:00	Thursday	2.305
3729	2022-09-22 12:00:00+00:00	Thursday	2.158
3730	2022-09-22 12:20:00+00:00	Thursday	2.080
3731	2022-09-22 12:40:00+00:00	Thursday	2.084
3732	2022-09-22 13:00:00+00:00	Thursday	2.084
3733	2022-09-22 13:20:00+00:00	Thursday	1.752
3734	2022-09-22 13:40:00+00:00	Thursday	1.793
3735	2022-09-22 14:00:00+00:00	Thursday	1.928
3736	2022-09-22 14:20:00+00:00	Thursday	1.904
3737	2022-09-22 14:40:00+00:00	Thursday	1.969
3738	2022-09-22 15:00:00+00:00	Thursday	1.848
3739	2022-09-22 15:20:00+00:00	Thursday	1.752
3740	2022-09-22 15:40:00+00:00	Thursday	1.773
3741	2022-09-22 16:00:00+00:00	Thursday	1.807
3742	2022-09-22 16:20:00+00:00	Thursday	1.769
3743	2022-09-22 16:40:00+00:00	Thursday	1.986
3744	2022-09-22 17:00:00+00:00	Thursday	1.972
3745	2022-09-22 17:20:00+00:00	Thursday	1.630
3746	2022-09-22 17:40:00+00:00	Thursday	1.730
3747	2022-09-22 18:00:00+00:00	Thursday	1.717
3748	2022-09-22 18:20:00+00:00	Thursday	1.752
3749	2022-09-22 18:40:00+00:00	Thursday	1.592
3750	2022-09-22 19:00:00+00:00	Thursday	1.736
3751	2022-09-22 19:20:00+00:00	Thursday	1.716
3752	2022-09-22 19:40:00+00:00	Thursday	1.645
3753	2022-09-22 20:00:00+00:00	Thursday	1.651
3865	2022-09-26 08:00:00+00:00	Monday	1.531
3866	2022-09-26 08:20:00+00:00	Monday	1.536
3867	2022-09-26 08:40:00+00:00	Monday	1.511
3898	2022-09-26 19:20:00+00:00	Monday	1.540
3899	2022-09-26 19:40:00+00:00	Monday	1.540
3900	2022-09-26 20:00:00+00:00	Monday	1.572
4106	2022-10-07 08:00:00+00:00	Friday	1.585
4107	2022-10-07 10:00:00+00:00	Friday	1.543
4115	2022-10-08 19:00:00+00:00	Saturday	1.611
4160	2022-10-18 10:20:00+00:00	Tuesday	2.145
4161	2022-10-18 15:20:00+00:00	Tuesday	2.260
4162	2022-10-18 15:40:00+00:00	Tuesday	2.005
4163	2022-10-18 16:00:00+00:00	Tuesday	1.591

People Count Data Exploration (Beach Usage)

Import People Count data

In [2]:
```
#People count
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from calendar import month_abbr
```

In [3]:
```
PCdf = pd.read_csv('image_counts_all.csv')
```

Understand the data

In [4]:
```
PCdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 312 entries, 0 to 311
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   File_name        312 non-null    object
 1   Year             312 non-null    int64 
 2   Total People     312 non-null    int64 
 3   People in Water  312 non-null    int64 
 4   People on Sand   312 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 12.3+ KB
```

In [5]:
```
PCdf.head()
```

Out[5]:

	File_name	Year	Total People	People in Water	People on Sand
0	2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...	2022	14	1	13
1	2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...	2022	16	2	14
2	2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...	2022	18	5	13
3	2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...	2022	18	5	13
4	2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...	2022	15	3	12

In [202]:
```
print(PCdf)
```

```
                                             File_name  Year  Total People  \
0    2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...  2022            14
1    2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...  2022            16
2    2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...  2022            18
3    2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...  2022            18
4    2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...  2022            15
..                                                 ...   ...           ...
307  2023-08-17_06-40-17_jpg.rf.1a45cf9a03b13987afb...  2023             0
308  2023-08-21_15-55-19_jpg.rf.545cc1b11449621bb2b...  2023            19
309  2023-08-21_17-35-20_jpg.rf.3abe7a112ec79a468be...  2023            20
310  2023-08-22_06-20-17_jpg.rf.e0d9e001700c794b191...  2023             0
311  2023-08-22_08-10-18_jpg.rf.4f2622d52d4785cfa9c...  2023             2

     People in Water  People on Sand
0                  1              13
1                  2              14
2                  5              13
3                  5              13
4                  3              12
..               ...             ...
307                0               0
308                3              16
309                0              20
310                0               0
311                0               2

[312 rows x 5 columns]
```

Data Cleaning and Wrangling

Delete Duplicates

A quick look at the csv file revealed some duplicate entries. Verify their presence and delete the duplicates.

In [199]:
```
PCdf.File_name.str.contains('Copy')
```

Out[199]:
```
0      False
1      False
2       True
3      False
4       True
       ...
307    False
308    False
309    False
310    False
311    False
Name: File_name, Length: 312, dtype: bool
```

In [6]:
```
PeopleCountdf = PCdf[~PCdf['File_name'].str.contains('Copy')]

print(PeopleCountdf)
```

```
                                             File_name  Year  Total People  \
0    2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...  2022            14
1    2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...  2022            16
3    2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...  2022            18
5    2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...  2022            15
7    2022-08-25_15-41-33_jpg.rf.5d69e86774bd8db8810...  2022            18
..                                                 ...   ...           ...
307  2023-08-17_06-40-17_jpg.rf.1a45cf9a03b13987afb...  2023             0
308  2023-08-21_15-55-19_jpg.rf.545cc1b11449621bb2b...  2023            19
309  2023-08-21_17-35-20_jpg.rf.3abe7a112ec79a468be...  2023            20
310  2023-08-22_06-20-17_jpg.rf.e0d9e001700c794b191...  2023             0
311  2023-08-22_08-10-18_jpg.rf.4f2622d52d4785cfa9c...  2023             2

     People in Water  People on Sand
0                  1              13
1                  2              14
3                  5              13
5                  3              12
7                  3              15
..               ...             ...
307                0               0
308                3              16
309                0              20
310                0               0
311                0               2

[261 rows x 5 columns]
```

In [7]:
```
PeopleCountdf.head()
```

Out[7]:

	File_name	Year	Total People	People in Water	People on Sand
0	2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...	2022	14	1	13
1	2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...	2022	16	2	14
3	2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...	2022	18	5	13
5	2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...	2022	15	3	12
7	2022-08-25_15-41-33_jpg.rf.5d69e86774bd8db8810...	2022	18	3	15

Extracting datetime information from filenames

In [8]:
```
PeopleCountdf = PeopleCountdf.copy()

PeopleCountdf['DateTime'] = PeopleCountdf['File_name'].str.extract(r'(\d{4}-\d{2}-\d{2}_\
PeopleCountdf['DateTime'] = pd.to_datetime(PeopleCountdf['DateTime'], format='%Y-%m-%d_%H
```

In [9]:
```
PeopleCountdf.head()
```

Out[9]:

	File_name	Year	Total People	People in Water	People on Sand	DateTime
0	2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...	2022	14	1	13	2022-08-25 15:35:39
1	2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...	2022	16	2	14	2022-08-25 15:35:40
3	2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...	2022	18	5	13	2022-08-25 15:38:19
5	2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...	2022	15	3	12	2022-08-25 15:40:27
7	2022-08-25_15-41-33_jpg.rf.5d69e86774bd8db8810...	2022	18	3	15	2022-08-25 15:41:33

In [10]:
```
PeopleCountdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 261 entries, 0 to 311
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype         
---  ------           --------------  -----         
 0   File_name        261 non-null    object        
 1   Year             261 non-null    int64         
 2   Total People     261 non-null    int64         
 3   People in Water  261 non-null    int64         
 4   People on Sand   261 non-null    int64         
 5   DateTime         261 non-null    datetime64[ns]
dtypes: datetime64[ns](1), int64(4), object(1)
memory usage: 14.3+ KB
```

Selecting the required year

In [11]:
```
PeopleCount2022 = PeopleCountdf[PeopleCountdf['DateTime'].dt.year == 2022]
```

In [12]:
```
PeopleCount2022.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 220 entries, 0 to 265
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype         
---  ------           --------------  -----         
 0   File_name        220 non-null    object        
 1   Year             220 non-null    int64         
 2   Total People     220 non-null    int64         
 3   People in Water  220 non-null    int64         
 4   People on Sand   220 non-null    int64         
 5   DateTime         220 non-null    datetime64[ns]
dtypes: datetime64[ns](1), int64(4), object(1)
memory usage: 12.0+ KB
```

In [13]:
```
PeopleCount2022 = PeopleCount2022.reset_index(drop=True)
```

In [14]:
```
PeopleCount2022.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220 entries, 0 to 219
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype         
---  ------           --------------  -----         
 0   File_name        220 non-null    object        
 1   Year             220 non-null    int64         
 2   Total People     220 non-null    int64         
 3   People in Water  220 non-null    int64         
 4   People on Sand   220 non-null    int64         
 5   DateTime         220 non-null    datetime64[ns]
dtypes: datetime64[ns](1), int64(4), object(1)
memory usage: 10.4+ KB
```

In [20]:
```
PeopleCount2022
```

Out[20]:

	File_name	Year	Total People	People in Water	People on Sand	DateTime
0	2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...	2022	14	1	13	2022-08-25 15:35:39
1	2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...	2022	16	2	14	2022-08-25 15:35:40
2	2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...	2022	18	5	13	2022-08-25 15:38:19
3	2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...	2022	15	3	12	2022-08-25 15:40:27
4	2022-08-25_15-41-33_jpg.rf.5d69e86774bd8db8810...	2022	18	3	15	2022-08-25 15:41:33
...	...	...	...	...	...	...
215	2022-10-22_15-43-52_jpg.rf.f0f727fb618ad2bf751...	2022	2	0	2	2022-10-22 15:43:52
216	2022-10-23_17-20-16_jpg.rf.c9fbc83271b930be8bd...	2022	4	0	4	2022-10-23 17:20:16
217	2022-10-24_17-30-16_jpg.rf.359aea533c623c69372...	2022	7	0	7	2022-10-24 17:30:16
218	2022-10-25_18-33-52_jpg.rf.1ee485d7e774f4d046c...	2022	2	0	2	2022-10-25 18:33:52
219	2022-10-27_08-50-16_jpg.rf.b579b95f7074c1aa942...	2022	2	0	2	2022-10-27 08:50:16

220 rows × 6 columns

Data consistency issues

In [233]:
```
counts_per_datetime = PeopleCount2022.groupby('DateTime').size().reset_index(name='Count'
print(counts_per_datetime)
```

```
               DateTime  Count
0   2022-08-25 15:35:39      1
1   2022-08-25 15:35:40      1
2   2022-08-25 15:38:19      1
3   2022-08-25 15:40:27      1
4   2022-08-25 15:41:33      1
..                  ...    ...
213 2022-10-22 15:43:52      1
214 2022-10-23 17:20:16      1
215 2022-10-24 17:30:16      1
216 2022-10-25 18:33:52      1
217 2022-10-27 08:50:16      1

[218 rows x 2 columns]
```

In [244]:
```
counts_per_datetime.head(10)
```

Out[244]:

	DateTime	Count
0	2022-08-25 15:35:39	1
1	2022-08-25 15:35:40	1
2	2022-08-25 15:38:19	1
3	2022-08-25 15:40:27	1
4	2022-08-25 15:41:33	1
5	2022-08-25 15:44:36	1
6	2022-08-25 15:44:37	1
7	2022-08-25 16:04:39	1
8	2022-08-25 16:04:41	1
9	2022-08-25 16:24:40	1

In [18]:
```
date_counts = PeopleCount2022['DateTime'].dt.date.value_counts().reset_index()
date_counts.columns = ['Date', 'Count']
print(date_counts)
```

```
          Date  Count
0   2022-08-27     44
1   2022-08-26     27
2   2022-08-25     15
3   2022-08-28     14
4   2022-08-29      7
5   2022-09-30      7
6   2022-10-18      6
7   2022-09-10      5
8   2022-09-13      5
9   2022-10-14      5
10  2022-10-09      4
11  2022-09-29      4
12  2022-10-01      4
13  2022-10-07      4
14  2022-09-02      4
15  2022-10-19      3
16  2022-09-14      3
17  2022-09-11      3
18  2022-10-08      3
19  2022-10-16      3
20  2022-09-19      3
21  2022-10-10      2
22  2022-08-31      2
23  2022-10-06      2
24  2022-10-20      2
25  2022-10-02      2
26  2022-10-13      2
27  2022-09-07      2
28  2022-09-04      2
29  2022-09-08      2
30  2022-09-25      2
31  2022-09-24      2
32  2022-09-22      2
33  2022-10-21      2
34  2022-09-17      2
35  2022-09-09      2
36  2022-09-27      2
37  2022-10-22      1
38  2022-10-25      1
39  2022-10-23      1
40  2022-10-24      1
41  2022-09-28      1
42  2022-10-15      1
43  2022-10-12      1
44  2022-10-11      1
45  2022-10-05      1
46  2022-09-20      1
47  2022-09-15      1
48  2022-09-12      1
49  2022-09-06      1
50  2022-09-05      1
51  2022-10-27      1
```

In [19]:
```
date_counts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52 entries, 0 to 51
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Date    52 non-null     object
 1   Count   52 non-null     int64 
dtypes: int64(1), object(1)
memory usage: 960.0+ bytes
```

As seen above, from the 51 days available in the people count data, the number of records greatly varies as well as the times the data was recorded at.

Due to these data limitations, we look for smaller timeframes with most records to track daily beach usage

Selecting a smaller timeframe

Adding date, month and hour variables to track smaller timeframes

In [23]:
```
PeopleCount2022['date'] = PeopleCount2022['DateTime'].dt.date
PeopleCount2022['month'] = PeopleCount2022['DateTime'].dt.month
PeopleCount2022['hour'] = PeopleCount2022['DateTime'].dt.hour
PeopleCount2022
```

Out[23]:

	File_name	Year	Total People	People in Water	People on Sand	DateTime	date	month	hour
0	2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...	2022	14	1	13	2022-08-25 15:35:39	2022-08-25	8	15
1	2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...	2022	16	2	14	2022-08-25 15:35:40	2022-08-25	8	15
2	2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...	2022	18	5	13	2022-08-25 15:38:19	2022-08-25	8	15
3	2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...	2022	15	3	12	2022-08-25 15:40:27	2022-08-25	8	15
4	2022-08-25_15-41-33_jpg.rf.5d69e86774bd8db8810...	2022	18	3	15	2022-08-25 15:41:33	2022-08-25	8	15
...	...	...	...	...	...	...	...	...	...
215	2022-10-22_15-43-52_jpg.rf.f0f727fb618ad2bf751...	2022	2	0	2	2022-10-22 15:43:52	2022-10-22	10	15
216	2022-10-23_17-20-16_jpg.rf.c9fbc83271b930be8bd...	2022	4	0	4	2022-10-23 17:20:16	2022-10-23	10	17
217	2022-10-24_17-30-16_jpg.rf.359aea533c623c69372...	2022	7	0	7	2022-10-24 17:30:16	2022-10-24	10	17
218	2022-10-25_18-33-52_jpg.rf.1ee485d7e774f4d046c...	2022	2	0	2	2022-10-25 18:33:52	2022-10-25	10	18
219	2022-10-27_08-50-16_jpg.rf.b579b95f7074c1aa942...	2022	2	0	2	2022-10-27 08:50:16	2022-10-27	10	8

220 rows × 9 columns

Records by the hour

In [30]:
```
PeopleCount2022['hour'].value_counts().sort_index()
```

Out[30]:
```
6      2
7     12
8     10
9      3
10     9
11    16
12    15
13    20
14    25
15    34
16    27
17    28
18    19
Name: hour, dtype: int64
```

Checking for duplicates

August 25-28 has the most number of records and that can be seen affecting the hourly distribution below

In [28]:
```
hourly_entries_per_day = PeopleCount2022.groupby(['date', 'hour']).size().reset_index(nam
print(hourly_entries_per_day)
```

```
           date  hour  Count
0    2022-08-25    15      7
1    2022-08-25    16      6
2    2022-08-25    17      2
3    2022-08-26    11      2
4    2022-08-26    13      1
..          ...   ...    ...
128  2022-10-22    15      1
129  2022-10-23    17      1
130  2022-10-24    17      1
131  2022-10-25    18      1
132  2022-10-27     8      1

[133 rows x 3 columns]
```

Selecting the timeframe with most records: 12PM - 5PM

After removing duplicate days, we see the most records between 12PM - 5PM which could potentially be the busiest given the time of the day.

In [34]:
```
total_distinct_hourly_counts = PeopleCount2022.groupby('hour')['date'].nunique().reset_in
print(total_distinct_hourly_counts)
```

```
    hour  Count
0      6      2
1      7     10
2      8     10
3      9      3
4     10      8
5     11      9
6     12     10
7     13     12
8     14     14
9     15     17
10    16     12
11    17     12
12    18     14
```

Filter data to 12PM - 5PM

In [35]:
```
filtered_data = PeopleCount2022[(PeopleCount2022['hour'] >= 12) & (PeopleCount2022['hour'
```

In [53]:
```
filtered_data
```

Out[53]:

	DateTime	File_name	Year	Total People	People in Water	People on Sand	month	hour
0	2022-08-25 15:35:39	2022-08-25_15-35-39_jpg.rf.f29c550c1dfa9e50ded...	2022	14	1	13	8	15
1	2022-08-25 15:35:40	2022-08-25_15-35-40_jpg.rf.5bfb5ece1d5e39664fd...	2022	16	2	14	8	15
2	2022-08-25 15:38:19	2022-08-25_15-38-19_jpg.rf.592c8ceddf472ad8562...	2022	18	5	13	8	15
3	2022-08-25 15:40:27	2022-08-25_15-40-27_jpg.rf.ceb8e5145f8ec9240ed...	2022	15	3	12	8	15
4	2022-08-25 15:41:33	2022-08-25_15-41-33_jpg.rf.5d69e86774bd8db8810...	2022	18	3	15	8	15
...	...	...	...	...	...	...	...	...
144	2022-10-18 15:40:15	2022-10-18_15-40-15_jpg.rf.d3ff33ba3fe8db1d542...	2022	5	2	3	10	15
145	2022-10-21 13:30:16	2022-10-21_13-30-16_jpg.rf.fc5f4afa75e1c39ffcd...	2022	2	0	2	10	13
146	2022-10-22 15:43:52	2022-10-22_15-43-52_jpg.rf.f0f727fb618ad2bf751...	2022	2	0	2	10	15
147	2022-10-23 17:20:16	2022-10-23_17-20-16_jpg.rf.c9fbc83271b930be8bd...	2022	4	0	4	10	17
148	2022-10-24 17:30:16	2022-10-24_17-30-16_jpg.rf.359aea533c623c69372...	2022	7	0	7	10	17

149 rows × 8 columns

Save all dataframes

In [55]:
```
filtered_data.to_csv('filtered_People_data.csv', index=False)
```

In [73]:
```
PeopleCount2022.to_csv('PeopleCount2022.csv', index=False)
```

Addressing Beach Usage

Summation or average of people counts to present daily beach usage may be counterproductive as frequency of records is inconsistent

Therefore, we select the record with max value in the selected timeframe to represent the daily beach usage

In [61]:
```
MaxPeopleCount = filtered_data.groupby(filtered_data['DateTime'].dt.date)['Total People']
MaxPeopleCount.columns = ['Date', 'Daily Max People']
MaxPeopleCount
```

Out[61]:

	Date	Daily Max People
0	2022-08-25	22
1	2022-08-26	25
2	2022-08-27	32
3	2022-08-28	29
4	2022-08-29	28
5	2022-09-02	23
6	2022-09-04	21
7	2022-09-05	19
8	2022-09-06	13
9	2022-09-07	16
10	2022-09-09	1
11	2022-09-10	15
12	2022-09-11	14
13	2022-09-12	2
14	2022-09-13	3
15	2022-09-14	10
16	2022-09-15	3
17	2022-09-17	21
18	2022-09-19	10
19	2022-09-22	7
20	2022-09-25	0
21	2022-09-27	0
22	2022-09-28	0
23	2022-09-29	2
24	2022-09-30	3
25	2022-10-01	4
26	2022-10-08	7
27	2022-10-09	7
28	2022-10-12	0
29	2022-10-13	3
30	2022-10-14	4
31	2022-10-16	3
32	2022-10-18	5
33	2022-10-21	2
34	2022-10-22	2
35	2022-10-23	4
36	2022-10-24	7

Understanding the distribution of selected People count data

In [64]:
```
DailyMaxPeople['Date'] = pd.to_datetime(DailyMaxPeople['Date'])
DailyMaxPeople['Month'] = DailyMaxPeople['Date'].dt.month
DailyMaxPeople['Day'] = DailyMaxPeople['Date'].dt.day
DailyMaxPeople['Day_Name'] = DailyMaxPeople['Date'].dt.day_name()
```

In [65]:
```
DailyMaxPeople
```

Out[65]:

	Date	Max Total People	Month	Day	Day_Name
0	2022-08-25	22	8	25	Thursday
1	2022-08-26	25	8	26	Friday
2	2022-08-27	32	8	27	Saturday
3	2022-08-28	29	8	28	Sunday
4	2022-08-29	28	8	29	Monday
5	2022-09-02	23	9	2	Friday
6	2022-09-04	21	9	4	Sunday
7	2022-09-05	19	9	5	Monday
8	2022-09-06	13	9	6	Tuesday
9	2022-09-07	16	9	7	Wednesday
10	2022-09-09	1	9	9	Friday
11	2022-09-10	15	9	10	Saturday
12	2022-09-11	14	9	11	Sunday
13	2022-09-12	2	9	12	Monday
14	2022-09-13	3	9	13	Tuesday
15	2022-09-14	10	9	14	Wednesday
16	2022-09-15	3	9	15	Thursday
17	2022-09-17	21	9	17	Saturday
18	2022-09-19	10	9	19	Monday
19	2022-09-22	7	9	22	Thursday
20	2022-09-25	0	9	25	Sunday
21	2022-09-27	0	9	27	Tuesday
22	2022-09-28	0	9	28	Wednesday
23	2022-09-29	2	9	29	Thursday
24	2022-09-30	3	9	30	Friday
25	2022-10-01	4	10	1	Saturday
26	2022-10-08	7	10	8	Saturday
27	2022-10-09	7	10	9	Sunday
28	2022-10-12	0	10	12	Wednesday
29	2022-10-13	3	10	13	Thursday
30	2022-10-14	4	10	14	Friday
31	2022-10-16	3	10	16	Sunday
32	2022-10-18	5	10	18	Tuesday
33	2022-10-21	2	10	21	Friday
34	2022-10-22	2	10	22	Saturday
35	2022-10-23	4	10	23	Sunday
36	2022-10-24	7	10	24	Monday

Merge People Count with Weather variables

In [68]:
```
merged_df = pd.read_csv('merged_df.csv')
merged_df
```

Out[68]:

	datetime	tempmax	tempmin	temp	feelslikemax	feelslikemin	feelslike	dew	humidity	precip	...	visib
0	2022-06-07	17.6	10.7	13.9	17.6	10.7	13.9	12.2	89.6	37.380	...	
1	2022-06-08	17.0	8.6	13.8	17.0	8.6	13.8	10.2	79.4	0.839	...	
2	2022-06-09	17.7	11.1	13.3	17.7	11.1	13.3	10.7	84.4	12.976	...	
3	2022-06-10	19.9	8.7	14.3	19.9	8.7	14.2	9.0	72.6	0.063	...	
4	2022-06-11	23.0	12.5	17.5	23.0	12.5	17.5	11.4	69.1	0.207	...	
...	...	...	...	...	...	...	...	...	...	...	...	
139	2022-10-24	22.6	10.8	16.0	22.6	10.8	16.0	6.1	53.1	0.000	...	
140	2022-10-25	20.9	13.1	16.3	20.9	13.1	16.3	8.2	59.3	0.000	...	
141	2022-10-26	16.5	6.8	12.1	16.5	2.7	10.7	8.2	78.6	4.261	...	
142	2022-10-27	8.9	2.1	5.8	7.1	-0.7	3.6	0.9	71.9	0.150	...	
143	2022-10-28	12.2	1.1	5.7	12.2	-1.7	3.7	0.7	72.0	0.000	...	

144 rows × 26 columns

In [70]:

```
DailyMaxPeople['Date'] = pd.to_datetime(DailyMaxPeople['Date']).dt.date
merged_df['datetime'] = pd.to_datetime(merged_df['datetime']).dt.date

# Merge the dataframes on the 'Date' column
allVals = pd.merge(DailyMaxPeople, merged_df, left_on='Date', right_on='datetime', how='l

# Drop the duplicate datetime column and reset the index
allVals = allVals.drop('datetime', axis=1).reset_index(drop=True)
allVals
```

Out[70]:

	Date	Max Total People	Month	Day	Day_Name	tempmax	tempmin	temp	feelslikemax	feelslikemin	...	visibility
0	2022-08-25	22	8	25	Thursday	24.3	17.3	20.8	24.3	17.3	...	24.0
1	2022-08-26	25	8	26	Friday	21.8	12.7	19.1	21.8	12.7	...	20.3
2	2022-08-27	32	8	27	Saturday	23.3	9.2	16.5	23.3	8.9	...	24.9
3	2022-08-28	29	8	28	Sunday	27.8	13.8	20.9	28.4	13.8	...	24.0
4	2022-08-29	28	8	29	Monday	26.3	21.1	23.6	26.3	21.1	...	24.0
5	2022-09-02	23	9	2	Friday	26.9	17.2	21.8	27.4	17.2	...	24.0
6	2022-09-04	21	9	4	Sunday	18.9	15.8	16.5	18.9	15.8	...	24.0
7	2022-09-05	19	9	5	Monday	21.7	15.4	17.8	21.7	15.4	...	24.0
8	2022-09-06	13	9	6	Tuesday	21.6	12.7	17.4	21.6	12.7	...	24.0
9	2022-09-07	16	9	7	Wednesday	22.4	9.9	16.1	22.4	9.5	...	24.0
10	2022-09-09	1	9	9	Friday	24.5	12.6	19.0	24.5	12.6	...	24.0
11	2022-09-10	15	9	10	Saturday	27.5	17.1	22.0	28.3	17.1	...	24.0
12	2022-09-11	14	9	11	Sunday	24.6	18.9	21.3	24.6	18.9	...	24.0
13	2022-09-12	2	9	12	Monday	20.1	17.1	18.8	20.1	17.1	...	18.5
14	2022-09-13	3	9	13	Tuesday	19.5	10.2	16.0	19.5	10.2	...	24.0
15	2022-09-14	10	9	14	Wednesday	20.0	9.4	17.1	20.0	8.2	...	24.0
16	2022-09-15	3	9	15	Thursday	17.5	5.2	12.4	17.5	3.4	...	24.0
17	2022-09-17	21	9	17	Saturday	25.3	15.4	20.2	25.3	15.4	...	17.6
18	2022-09-19	10	9	19	Monday	21.7	17.2	19.8	21.7	17.2	...	23.3
19	2022-09-22	7	9	22	Thursday	17.1	10.7	12.4	17.1	10.7	...	22.4
20	2022-09-25	0	9	25	Sunday	15.6	11.7	13.3	15.6	11.7	...	15.5
21	2022-09-27	0	9	27	Tuesday	12.8	11.2	12.2	12.8	11.2	...	16.2
22	2022-09-28	0	9	28	Wednesday	11.0	9.0	10.0	11.0	6.9	...	21.4
23	2022-09-29	2	9	29	Thursday	14.0	5.9	10.0	14.0	4.5	...	24.0
24	2022-09-30	3	9	30	Friday	16.2	3.2	9.7	16.2	2.8	...	24.0

	Date	Max Total People	Month	Day	Day_Name	tempmax	tempmin	temp	feelslikemax	feelslikemin	...	visibility
25	2022-10-01	4	10	1	Saturday	18.9	6.7	12.1	18.9	5.4	...	24.0
26	2022-10-08	7	10	8	Saturday	10.4	4.7	7.5	10.4	1.5	...	23.3
27	2022-10-09	7	10	9	Sunday	13.0	6.2	10.1	13.0	5.2	...	24.0
28	2022-10-12	0	10	12	Wednesday	18.2	11.4	15.2	18.2	11.4	...	18.5
29	2022-10-13	3	10	13	Thursday	13.5	7.4	10.1	13.5	4.7	...	21.9
30	2022-10-14	4	10	14	Friday	11.9	5.9	8.5	11.9	2.3	...	23.0
31	2022-10-16	3	10	16	Sunday	10.8	6.7	8.7	10.8	4.0	...	22.5
32	2022-10-18	5	10	18	Tuesday	6.0	2.1	4.6	2.3	-0.8	...	22.6
33	2022-10-21	2	10	21	Friday	15.5	2.0	9.1	15.5	-1.6	...	23.0
34	2022-10-22	2	10	22	Saturday	21.1	11.9	15.8	21.1	11.9	...	24.0
35	2022-10-23	4	10	23	Sunday	23.2	12.2	16.9	23.2	12.2	...	24.0
36	2022-10-24	7	10	24	Monday	22.6	10.8	16.0	22.6	10.8	...	24.0

37 rows × 30 columns

In [71]:
```
allVals = allVals.drop(['time', 'month'], axis=1)
allVals
```

Out[71]:

	Date	Max Total People	Month	Day	Day_Name	tempmax	tempmin	temp	feelslikemax	feelslikemin	...	wind
0	2022-08-25	22	8	25	Thursday	24.3	17.3	20.8	24.3	17.3	...	13
1	2022-08-26	25	8	26	Friday	21.8	12.7	19.1	21.8	12.7	...	34
2	2022-08-27	32	8	27	Saturday	23.3	9.2	16.5	23.3	8.9	...	12
3	2022-08-28	29	8	28	Sunday	27.8	13.8	20.9	28.4	13.8	...	17
4	2022-08-29	28	8	29	Monday	26.3	21.1	23.6	26.3	21.1	...	19
5	2022-09-02	23	9	2	Friday	26.9	17.2	21.8	27.4	17.2	...	18
	2022-											

Visualize available data and check for correlations

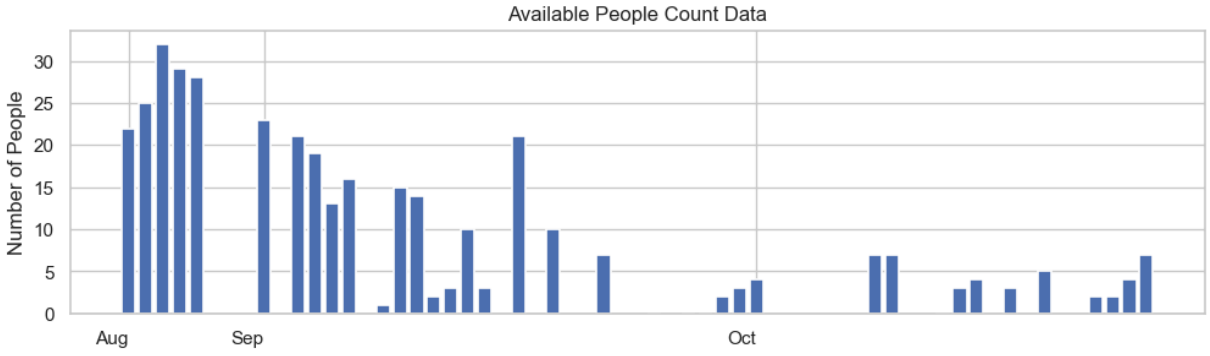
Review available data

In [72]:
```
allVals.groupby('conditions')[['Max Total People']].agg(['count','mean','median','min','m
```

Out[72]:

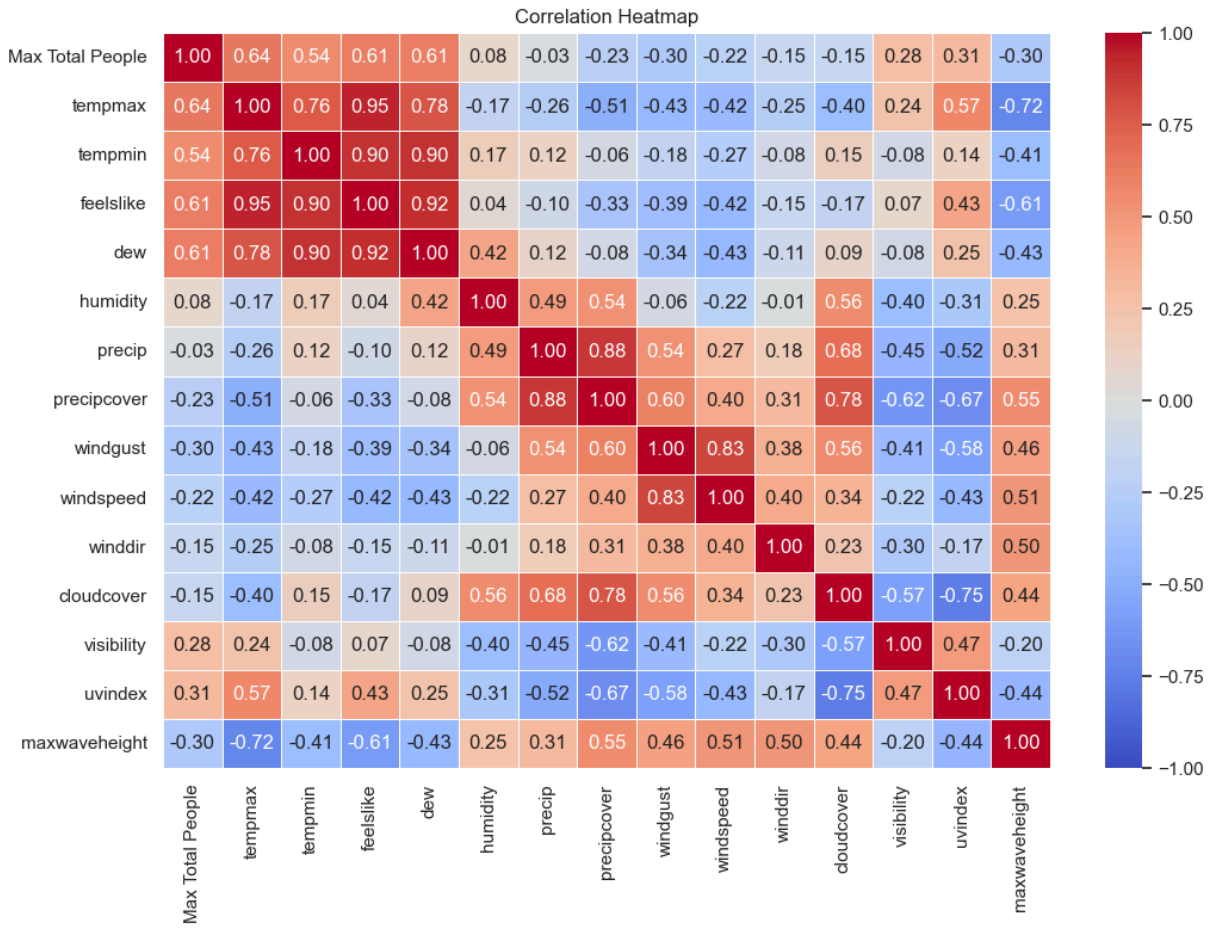
	Max Total People				
	count	mean	median	min	max
conditions					
clear-day	6	14.166667	10.0	1	32
partly-cloudy-day	12	9.000000	7.0	2	23
rain	19	9.157895	5.0	0	28

In [91]:
```
plt.figure(figsize=(12, 3))

plt.bar(allVals['Date'], allVals['Max Total People'], linewidth=1.2)

plt.title('Available People Count Data')
plt.ylabel('Number of People')
month_index = allVals.groupby('Month').head(1).index
plt.xticks(allVals.loc[month_index, 'Date'], allVals.loc[month_index, 'month_name'], rota

plt.show()
```

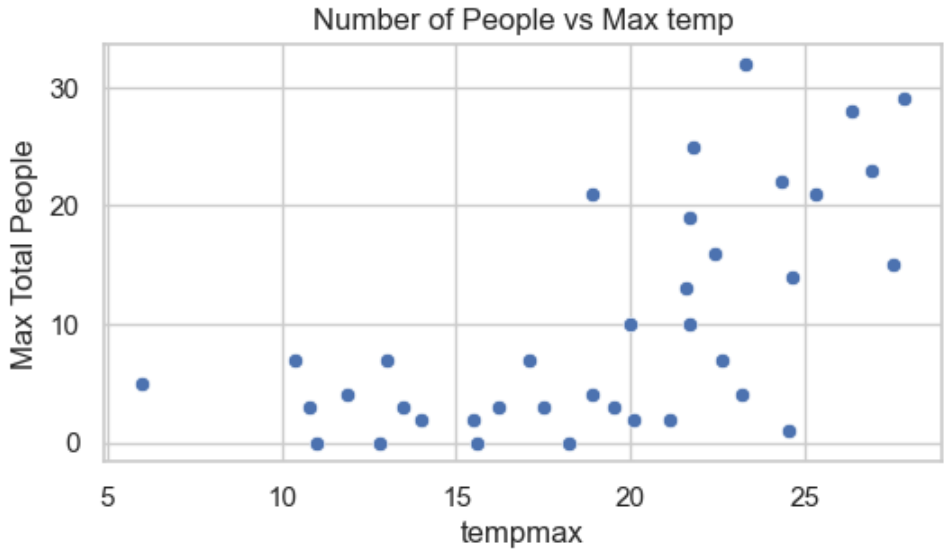


Checking for correlations

In [96]:
```
heatmap_data = allVals.drop(['Date', 'Month', 'Day', 'precipprob', 'conditions', 'temp',

plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt='.2f',

plt.title('Correlation Heatmap')
plt.show()
```



In [100]:
```
plt.figure(figsize=(6, 3))
sns.scatterplot(x='tempmax', y='Max Total People', data=allVals)
plt.title('Number of People vs Max temp')
plt.show()
```



In [106]:
```
allVals.to_csv('SelectedTimeframe - allVals.csv', index=False)
```

Save data for Machine Learning

In [110]:
```
#Remove unnecassary variables
SelectVals = allVals.drop(['Date', 'Month', 'Day', 'Day_Name', 'month_name', 'windir_cat'
```

In [111]:
```
SelectVals
```

Out[111]:

	Max Total People	tempmax	tempmin	feelslike	dew	humidity	precip	precipcover	windgust	windspeed	winddir
0	22	24.3	17.3	20.8	17.3	81.2	5.132	16.67	32.0	13.3	132.9
1	25	21.8	12.7	19.1	16.2	83.6	0.379	12.50	42.5	25.3	347.5
2	32	23.3	9.2	16.5	11.4	74.4	0.000	0.00	16.9	12.1	122.3
3	29	27.8	13.8	21.0	13.6	63.9	0.000	0.00	31.7	18.2	174.9
4	28	26.3	21.1	23.6	19.9	79.9	15.732	33.33	48.2	22.7	191.3
5	23	26.9	17.2	21.9	14.4	64.1	0.000	0.00	36.7	20.9	185.3
6	21	18.9	15.8	16.5	14.9	90.1	7.956	33.33	37.1	17.7	57.6
7	19	21.7	15.4	17.8	15.1	84.6	0.000	0.00	24.1	12.7	63.7
8	13	21.6	12.7	17.4	12.5	75.0	0.000	0.00	35.3	21.2	34.8
9	16	22.4	9.9	16.1	12.3	79.6	0.000	0.00	20.2	17.1	9.6
10	1	24.5	12.6	19.0	14.2	75.7	0.000	0.00	25.2	15.5	183.2
11	15	27.5	17.1	22.1	16.1	70.3	0.000	0.00	33.5	17.5	170.4
12	14	24.6	18.9	21.3	16.6	75.3	0.095	8.33	27.7	14.8	163.7
13	2	20.1	17.1	18.8	15.8	82.9	2.094	25.00	32.8	15.3	174.4
14	3	19.5	10.2	16.0	11.3	74.7	0.000	0.00	23.0	11.6	241.0
15	10	20.0	9.4	17.1	13.0	77.5	0.000	0.00	34.6	20.0	322.0
16	3	17.5	5.2	11.9	6.9	71.4	0.000	0.00	30.6	14.1	94.6
17	21	25.3	15.4	20.2	15.9	78.1	1.200	4.17	43.9	21.1	173.5
18	10	21.7	17.2	19.8	16.4	81.4	0.084	8.33	36.5	24.2	287.8
19	7	17.1	10.7	12.4	7.1	70.7	1.594	25.00	56.0	30.3	332.0
20	0	15.6	11.7	13.3	11.0	86.2	4.842	50.00	45.3	20.0	270.6
21	0	12.8	11.2	12.2	10.4	89.0	16.717	79.17	46.4	22.4	313.4
22	0	11.0	9.0	9.1	7.5	84.7	2.563	25.00	34.9	19.6	5.6
23	2	14.0	5.9	9.8	5.5	74.7	0.000	0.00	26.3	11.3	354.8
24	3	16.2	3.2	9.3	5.7	78.5	0.000	0.00	17.6	12.7	16.0
25	4	18.9	6.7	11.5	7.6	76.6	0.000	0.00	50.4	20.7	40.5
26	7	10.4	4.7	4.5	1.1	64.5	0.154	12.50	50.9	27.9	262.8
27	7	13.0	6.2	8.9	5.1	71.7	0.000	0.00	43.4	24.1	264.9
28	0	18.2	11.4	15.2	11.0	76.6	11.998	45.83	74.2	25.2	175.1
29	3	13.5	7.4	8.9	6.9	80.7	4.907	37.50	51.0	29.5	238.0
30	4	11.9	5.9	5.9	0.0	56.4	1.624	16.67	51.8	28.7	194.7
31	3	10.8	6.7	7.1	3.6	70.9	0.981	12.50	44.3	21.8	230.8
32	5	6.0	2.1	0.9	3.0	89.3	13.511	83.33	66.6	31.0	286.6
33	2	15.5	2.0	7.4	2.1	63.6	0.094	4.17	43.9	28.4	180.8

	Max Total People	tempmax	tempmin	feelslike	dew	humidity	precip	precipcover	windgust	windspeed	winddir
34	2	21.1	11.9	15.8	6.4	53.7	0.000	0.00	49.2	30.2	184.6
35	4	23.2	12.2	16.9	8.9	60.3	0.000	0.00	39.2	17.0	153.8
36	7	22.6	10.8	16.0	6.1	53.1	0.000	0.00	38.2	21.8	147.8

In [112]:
```
SelectVals.to_csv('PeopleCount ML data.csv', index=False)
```

People Count Machine Learning Models

Import data

In [2]:
```
import pandas as pd
BeachUsage = pd.read_csv('PeopleCount ML data.csv')
BeachUsage.head()
```

Out[2]:

	Max Total People	tempmax	tempmin	feelslike	dew	humidity	precip	precipcover	windgust	windspeed	winddir
0	22	24.3	17.3	20.8	17.3	81.2	5.132	16.67	32.0	13.3	132.9
1	25	21.8	12.7	19.1	16.2	83.6	0.379	12.50	42.5	25.3	347.5
2	32	23.3	9.2	16.5	11.4	74.4	0.000	0.00	16.9	12.1	122.3
3	29	27.8	13.8	21.0	13.6	63.9	0.000	0.00	31.7	18.2	174.9
4	28	26.3	21.1	23.6	19.9	79.9	15.732	33.33	48.2	22.7	191.3

Prepping the data

Convert conditions to numeric

In [3]:
```
BeachUsage = pd.get_dummies(BeachUsage, columns=['conditions'])
BeachUsage.head()
```

Out[3]:

	Max Total People	tempmax	tempmin	feelslike	dew	humidity	precip	precipcover	windgust	windspeed	winddir
0	22	24.3	17.3	20.8	17.3	81.2	5.132	16.67	32.0	13.3	132.9
1	25	21.8	12.7	19.1	16.2	83.6	0.379	12.50	42.5	25.3	347.5
2	32	23.3	9.2	16.5	11.4	74.4	0.000	0.00	16.9	12.1	122.3
3	29	27.8	13.8	21.0	13.6	63.9	0.000	0.00	31.7	18.2	174.9
4	28	26.3	21.1	23.6	19.9	79.9	15.732	33.33	48.2	22.7	191.3

Split to train and test sets

In [4]:
```
from sklearn.model_selection import train_test_split

X = BeachUsage.drop('Max Total People', axis=1)
Y = BeachUsage['Max Total People']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=111
```

In [5]:
```
for dataset in [y_train, y_test]:
    print(round(len(dataset) / len(Y), 2))
```

```
0.68
0.32
```

In [35]:
```
X_train.to_csv('X_train.csv', index=False)
X_test.to_csv('X_test.csv', index=False)

y_train.to_csv('y_train.csv', index=False)
y_test.to_csv('y_test.csv', index=False)
```

Standardization of data - Scaling

In [6]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

numeric_features = ['tempmax', 'tempmin', 'feelslike', 'dew', 'humidity', 'precip',
                    'precipcover', 'windgust', 'windspeed', 'winddir',
                    'cloudcover', 'visibility', 'uvindex']

# ColumnTransformer allows you to apply transformers to specific columns
ct = ColumnTransformer([
        ('scaler', StandardScaler(), numeric_features)
    ], remainder='passthrough')  # 'passthrough' means that the remainder columns will no

X_train_scaled = ct.fit_transform(X_train)
X_test_scaled = ct.transform(X_test)
```

In [7]:
```
X_train_scaled
```

Out[7]:
```
array([[ 0.25254829, -0.23062944,  0.60464974,  0.67433199,  0.37592941,
        -0.48790097, -0.6710554 , -0.21485851, -0.04488794,  1.30980297,
        -0.29277175,  0.45337236,  1.45283536,  0.        ,  1.        ,
         0.        ],
       [-1.16748966, -0.69938846, -1.10504952, -0.59681299,  0.72200942,
         1.97600696,  2.02518785,  1.33879208,  1.59524833,  0.52180485,
         1.49379344, -0.68005853, -1.08117981,  0.        ,  0.        ,
         1.        ],
       [ 0.01223417,  1.26939942,  0.47954979,  1.0702624 ,  1.73861944,
         3.506974  ,  1.7253656 ,  0.02197847, -0.44197356, -1.17051492,
         1.23856984,  0.45337236, -2.34818739,  0.        ,  0.        ,
         1.        ],
       [-1.05825597, -1.05095772, -0.9173996 , -0.88855118,  0.0731094 ,
        -0.48790097, -0.6710554 , -1.00115728, -1.54690747,  1.61749747,
        -1.02383596,  0.45337236,  0.60816364,  0.        ,  1.        ,
         0.        ],
       [ 1.75997319,  1.59753073,  1.60544931,  0.96607018, -1.07328062,
        -0.48790097, -0.6710554 , -0.01591545,  0.11049339,  0.02742984,
        -0.71237665,  0.45337236,  1.0304995 ,  0.        ,  1.        ,
         0.        ],
       [-1.27672335, -0.98064387, -1.10504952, -0.97190495, -0.25134061,
        -0.48790097, -0.6710554 ,  0.61880766,  0.66296034,  0.77415187,
        -0.34468163,  0.45337236, -1.08117981,  0.        ,  1.        ,
         0.        ],
       [-1.51703746, -1.05095772, -1.73054925, -2.0346655 , -1.90603564,
         0.32754359,  0.5275146 ,  1.41457991,  1.45713159,  0.11561058,
        -0.05485144, -0.08635664,  0.18582778,  0.        ,  0.        ,
         1.        ],
       [ 0.1433146 , -0.04312583,  0.37529984,  0.32007847,  0.0731094 ,
        -0.48790097, -0.6710554 , -1.31378209, -1.49511369,  0.54994764,
        -0.25383934,  0.45337236,  0.60816364,  0.        ,  1.        ,
         0.        ],
       [-0.29362015, -1.21502338, -0.47954979, -0.59681299, -0.28378561,
        -0.48790097, -0.6710554 , -0.59379767, -1.06349888, -0.82342051,
        -0.26681681,  0.45337236,  0.60816364,  0.        ,  1.        ,
         0.        ],
       [ 0.64578957,  0.54282294,  1.02164956,  1.34116214,  1.03564442,
        -0.2975971 ,  0.22769235,  0.53354635,  0.87013545,  1.54901668,
         1.27317643, -1.54362492, -0.65884394,  0.        ,  0.        ,
         1.        ],
       [ 1.23565148,  0.51938499,  1.00079957,  0.9243933 ,  0.1812594 ,
        -0.48790097, -0.6710554 , -1.10536555, -0.82179459,  0.00772989,
        -1.42181174,  0.45337236,  1.0304995 ,  1.        ,  0.        ,
         0.        ],
       [ 1.25749822,  1.9959759 ,  1.48034936,  1.42451591,  0.1379994 ,
        -0.44019947, -0.0721299 , -0.86852857, -0.94264674, -0.17519825,
         0.51615729,  0.45337236,  0.60816364,  0.        ,  0.        ,
         1.        ],
       [-1.84473853, -1.33221313, -2.02244913, -1.80544264, -1.03002062,
        -0.41057433,  0.22769235,  1.3293186 ,  1.31901485,  0.75445191,
         1.03958195,  0.07556206, -1.50351567,  0.        ,  0.        ,
         1.        ],
       [ 0.01223417, -0.86345411, -0.56294976, -0.45094389,  0.27859441,
        -0.48790097, -0.6710554 ,  1.2819512 ,  0.07596421, -1.33092882,
        -0.08513221,  0.45337236,  0.60816364,  0.        ,  1.        ,
         0.        ],
       [ 0.62394283,  1.17564762,  0.75059968,  1.11193928,  1.14379442,
        -0.48790097, -0.6710554 , -1.20957382, -1.30520318, -1.11329125,
         1.00064954,  0.45337236,  0.60816364,  0.        ,  1.        ,
         0.        ],
       [ 0.60209609,  0.54282294,  0.66719971,  0.57013978,  0.1055544 ,
        -0.48790097, -0.6710554 , -0.14854415,  0.16228717, -1.38440012,
        -1.04113925,  0.45337236,  1.0304995 ,  0.        ,  1.        ,
```

```
  0.        ],
[ 0.62394283,  1.59753073,  1.1675995 ,  1.38283903,  0.79771442,
 -0.4457228 , -0.0721299 , -0.0348624 ,  0.68022494,  0.98897516,
  0.2090238 ,  0.07556206,  0.18582778,  0.        ,  0.        ,
  1.        ],
[-0.73055491, -1.9650378 , -1.41779939, -1.59705822, -1.12735562,
 -0.44070159, -0.37123315,  0.66617506,  1.40533781, -0.01478435,
 -0.66479259, -0.08635664, -0.23650808,  0.        ,  0.        ,
  1.        ],
[ 0.97349063, -0.27750534,  0.47954979,  0.34091692,  0.0406644 ,
 -0.48790097, -0.6710554 , -1.89166432, -1.40879073, -0.56356875,
 -1.56888974,  0.93912845,  1.0304995 ,  1.        ,  0.        ,
  0.        ],
[ 0.77686999, -0.11343968,  0.39614983,  0.5284629 ,  0.60304441,
 -0.48790097, -0.6710554 , -1.57903951, -0.54556112, -1.62079956,
 -1.30068867,  0.45337236,  1.0304995 ,  1.        ,  0.        ,
  0.        ],
[-0.38100711,  0.07406392, -0.37529984, -0.5551361 , -0.35949061,
  0.31247996,  1.1264401 ,  1.81246604,  1.73336507,  1.40361227,
  0.52913476, -0.41019404, -0.23650808,  0.        ,  0.        ,
  1.        ],
[-1.7136581 , -0.32438124, -1.06334954, -0.47178233,  1.15460942,
  0.79903525,  1.1264401 , -0.18643807, -0.11394631, -1.65832328,
  1.90907252, -0.94992303, -1.50351567,  0.        ,  0.        ,
  1.        ],
[ 0.95164389,  0.42563319,  0.56294976, -0.18004414, -1.48425063,
 -0.48790097, -0.6710554 ,  0.22092153, -0.56282571, -0.26806946,
 -1.1968689 ,  0.45337236, -0.23650808,  1.        ,  0.        ,
  0.        ],
[ 0.82056347,  0.09750188,  0.37529984, -0.76352052, -2.26293065,
 -0.48790097, -0.6710554 ,  0.12618674,  0.26587472, -0.32435503,
 -0.42254646,  0.45337236, -0.23650808,  0.        ,  1.        ,
  0.        ],
[-0.70870817,  0.30844343, -0.18764992,  0.25756315,  1.31683443,
  1.94336909,  2.9239356 ,  0.79880376, -0.04488794,  0.82762317,
  1.44188356, -4.1343241 , -1.50351567,  0.        ,  0.        ,
  1.        ]])
```

In [8]:
```
# Convert scaled data back to DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_train_scaled_df.to_csv('X_train_scaled.csv', index=False)

X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X_test.columns)
X_test_scaled_df.to_csv('X_test_scaled.csv', index=False)
```

Train & Test - Linear Regression

Train the model

In [9]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X_train, y_train)
```

Evaluate the model

In [10]:
```
y_pred = model.predict(X_test)
```

In [14]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print out the metrics
print(f'LR Mean Absolute Error: {mae}')
print(f'LR Mean Squared Error: {mse}')
print(f'LR R-squared: {r2}')
```

```
LR Mean Absolute Error: 5.399395547143726
LR Mean Squared Error: 40.89182695755503
LR R-squared: 0.6538665011822287
```

Train, Test & Hyperparameter Tuning – XGBoost, Gradient Boosting & Other Models

XGBoost Regressor

Load data

In [1]:
```
import pandas as pd

X_train_scaled = pd.read_csv('X_train_scaled.csv')
y_train = pd.read_csv('y_train.csv')
X_test_scaled = pd.read_csv('X_test_scaled.csv')
y_test = pd.read_csv('y_test.csv')
```

Train and evaluate the model

In [3]:
```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-2.0.3-py3-none-win_amd64.whl (99.8 MB)
     ---------------------------------------- 99.8/99.8 MB 6.4 MB/s eta 0:00:00
Requirement already satisfied: numpy in c:\users\ammym\anaconda3\lib\site-packages (from
xgboost) (1.23.5)
Requirement already satisfied: scipy in c:\users\ammym\anaconda3\lib\site-packages (from
xgboost) (1.10.0)
Installing collected packages: xgboost
Successfully installed xgboost-2.0.3
```

In [4]:
```
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

xgb_model = XGBRegressor(objective='reg:squarederror', random_state=111)
xgb_model.fit(X_train_scaled, y_train.values.ravel())

y_pred_xgb = xgb_model.predict(X_test_scaled)

mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost MAE: {mae_xgb}")
print(f"XGBoost MSE: {mse_xgb}")
print(f"XGBoost R^2: {r2_xgb}")
```

```
XGBoost MAE: 5.548194865385692
XGBoost MSE: 45.671126190695226
XGBoost R^2: 0.613411581738766
```

Hyperparameter Tuning

In [91]:
```
from sklearn.model_selection import GridSearchCV

xgb_params = {
    'n_estimators': [50, 100],
    'max_depth': [2, 3, 4],
    'learning_rate': [ 0.25, 0.3],
    'subsample': [0.36]
}

xgb_grid_search = GridSearchCV(XGBRegressor(random_state=111), xgb_params, cv=5, scoring=
xgb_grid_search.fit(X_train_scaled, y_train.values.ravel())

print("Best parameters for XGBoost:", xgb_grid_search.best_params_)

# Using best parameters from grid search for XGBoost
best_xgb_model = XGBRegressor(**xgb_grid_search.best_params_, random_state=111)
best_xgb_model.fit(X_train_scaled, y_train.values.ravel())

y_pred_xgb = best_xgb_model.predict(X_test_scaled)

# Evaluation
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print("XGBoost with Best Parameters - MAE:", mae_xgb, "MSE:", mse_xgb, "R^2:", r2_xgb)
```

```
Best parameters for XGBoost: {'learning_rate': 0.25, 'max_depth': 3, 'n_estimators': 50,
'subsample': 0.36}
XGBoost with Best Parameters - MAE: 4.01172695060571 MSE: 30.86039455860283 R^2: 0.73877
8696423771
```

Gradient Boosting Regressor

Load data

In [1]:
```
import pandas as pd

X_train_scaled = pd.read_csv('X_train_scaled.csv')
y_train = pd.read_csv('y_train.csv')
X_test_scaled = pd.read_csv('X_test_scaled.csv')
y_test = pd.read_csv('y_test.csv')
```

Train the model

In [2]:
```
from sklearn.ensemble import GradientBoostingRegressor

gb_model = GradientBoostingRegressor(random_state=111)
gb_model.fit(X_train_scaled, y_train.values.ravel())
```

Out[2]:
```
GradientBoostingRegressor
GradientBoostingRegressor(random_state=111)
```

Evaluate the model

In [3]:
```
 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_pred_gb = gb_model.predict(X_test_scaled)

mae_gb = mean_absolute_error(y_test, y_pred_gb)
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

print(f"Gradient Boosting MAE: {mae_gb}")
print(f"Gradient Boosting MSE: {mse_gb}")
print(f"Gradient Boosting R^2: {r2_gb}")
```

```
Gradient Boosting MAE: 5.619763779242021
Gradient Boosting MSE: 48.55959616929224
Gradient Boosting R^2: 0.5889618005891086
```

Hyperparameter tuning

In [47]:
```
from sklearn.model_selection import GridSearchCV

gb_params = {
    'n_estimators': [50, 100],
    'max_depth': [1, 2, 3],
    'learning_rate': [ 0.25],
    'subsample': [0.36, 0.38, 0.4]
}

gb_grid_search = GridSearchCV(GradientBoostingRegressor(random_state=111), gb_params, cv=
gb_grid_search.fit(X_train_scaled, y_train.values.ravel())

print("Best parameters for Gradient Boosting:", gb_grid_search.best_params_)

# Using best parameters from grid search for Gradient Boosting
best_gb_model = GradientBoostingRegressor(**gb_grid_search.best_params_, random_state=111
best_gb_model.fit(X_train_scaled, y_train.values.ravel())

y_pred_gb = best_gb_model.predict(X_test_scaled)

# Evaluation
mae_gb = mean_absolute_error(y_test, y_pred_gb)
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

print("Gradient Boosting with Best Parameters - MAE:", mae_gb, "MSE:", mse_gb, "R^2:", r2
```

```
Best parameters for Gradient Boosting: {'learning_rate': 0.25, 'max_depth': 2, 'n_estima
tors': 50, 'subsample': 0.4}
Gradient Boosting with Best Parameters - MAE: 4.038385791645564 MSE: 31.951597678252757
R^2: 0.729542084077804
```

Random Forest

Load data

In [1]:
```
import pandas as pd

X_train_scaled = pd.read_csv('X_train_scaled.csv')
y_train = pd.read_csv('y_train.csv')
X_test_scaled = pd.read_csv('X_test_scaled.csv')
y_test = pd.read_csv('y_test.csv')
```

Train the model

In [2]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

#Create the Random Forest model
rf_model = RandomForestRegressor(random_state=111)

#Fit model on the training data
rf_model.fit(X_train_scaled, y_train.values.ravel())
```

Out[2]:
```
RandomForestRegressor
RandomForestRegressor(random_state=111)
```

Evaluate the model

In [10]:
```
# Predict on the test data
y_pred = rf_model.predict(X_test_scaled)

# Calculate the evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print out the metrics
print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Absolute Error: 5.646666666666666
Mean Squared Error: 50.01495
R-squared: 0.5766427933223606
```

Hyperparameter tuning

In [33]:
```
from sklearn.model_selection import GridSearchCV

rf_params = {
    'n_estimators': [50, 100, 150],
    'max_depth': [ None, 2, 3],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3 ]
}

rf_grid_search = GridSearchCV(RandomForestRegressor(random_state=111), rf_params, cv=5, s
rf_grid_search.fit(X_train_scaled, y_train.values.ravel())

print("Best parameters for Random Forest:", rf_grid_search.best_params_)

# Using best parameters from grid search for Random Forest
best_rf_model = RandomForestRegressor(**rf_grid_search.best_params_, random_state=111)
best_rf_model.fit(X_train_scaled, y_train.values.ravel())

y_pred_rf = best_rf_model.predict(X_test_scaled)

# Evaluation
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest with Best Parameters - MAE:", mae_rf, "MSE:", mse_rf, "R^2:", r2_rf)
```

```
Best parameters for Random Forest: {'max_depth': 2, 'min_samples_leaf': 3, 'min_samples_
split': 2, 'n_estimators': 100}
Random Forest with Best Parameters - MAE: 5.200790889665891 MSE: 44.66845004881075 R^2:
0.6218988474589262
```

Ridge Regressor

Load data

In [1]:
```
import pandas as pd

X_train_scaled = pd.read_csv('X_train_scaled.csv')
y_train = pd.read_csv('y_train.csv')
X_test_scaled = pd.read_csv('X_test_scaled.csv')
y_test = pd.read_csv('y_test.csv')
```

Train the model

In [2]:
```
from sklearn.linear_model import Ridge
ridge_model = Ridge(random_state=111)
ridge_model.fit(X_train_scaled, y_train.values.ravel())
```

Out[2]:
```
Ridge(random_state=111)
```

Evaluate the model

In [3]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_pred_ridge = ridge_model.predict(X_test_scaled)

mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Ridge Regression MAE: {mae_ridge}")
print(f"Ridge Regression MSE: {mse_ridge}")
print(f"Ridge Regression R^2: {r2_ridge}")
```

```
Ridge Regression MAE: 5.656849942839954
Ridge Regression MSE: 43.16482223093291
Ridge Regression R^2: 0.6346264753553762
```

Hyperparameter Tuning

In [29]:
```
from sklearn.linear_model import RidgeCV

# Define a range of alpha values for testing
alphas = [0.01, 0.015, 0.02]

# Create RidgeCV instance with specified alpha values and cross-validation strategy
ridge_cv = RidgeCV(alphas=alphas, scoring='neg_mean_absolute_error', cv=5)
ridge_cv.fit(X_train_scaled, y_train.values.ravel())

print(f"Optimal alpha value for Ridge Regression: {ridge_cv.alpha_}")


# Evaluate with the best alpha
best_ridge_model = Ridge(alpha=ridge_cv.alpha_, random_state=111)
best_ridge_model.fit(X_train_scaled, y_train.values.ravel())

y_pred_ridge = best_ridge_model.predict(X_test_scaled)

# Evaluation
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print("Ridge Regression with Best Parameters - MAE:", mae_ridge, "MSE:", mse_ridge, "R^2:
```

```
Optimal alpha value for Ridge Regression: 0.02
Ridge Regression with Best Parameters - MAE: 5.188788829156286 MSE: 38.77505226283687 R^
2: 0.6717841802346279
```

Discussion & Next steps

Insights gathered from the analysis of wave data and weather variables in this project give us a more candid overview of Station Beach's wave activity in summer 2022 and its correlation with different factors that influence the overall weather conditions and how it may affect the beach goers adversely at certain times. It is noticed that relatively colder days, rainy weather and higher windspeeds have a positive correlation with wave heights, and winds blowing in North to West directions were associated with the biggest waves, posing higher risk to beach goers. These trends should be studied over the next couple years and the findings are expected to help the local counties in allocating necessary resources and making the required arrangements to uphold beach safety and prevent deadly mishaps.

A moderate positive correlation was seen between temperature variables and the number of people at the beach, and although ML results beyond expectations were received with the very limited people count data at hand. The client is highly advised to procure more data for an analysis and ML model with solid foundations. The client is also advised to perform analysis on Beach Usage data and test the beach usage numbers against the wave height data to understand beach usage patterns against dangerous water levels. The frequency of beach usage data collection is suggested to be kept at least at the same number and intervals as that of the wave data for optimum results with analysis and ML models.
