Shark Attack Research

This notebook helps to analyse and understand the presence of sharks during tourist seasons in middle Atlantic and south eastern coastal waters, specifically North Carolina and South Carolina.

In short, we are trying to find the answers for the following by analysing the data:

1.Learn the likelihood of sharks to be present and attack in coastal waters where tourists swim, surf and wade.
2.Why are they so close to shore and in range to attack?
3.Is there something else they are looking for aka turtles, crabs?

Set Up

Importing the common libraries that we use in the analysis and setting the seed to make the notebook's output stable acroos multiple runs.

In [189]:

```
# Common imports
import numpy as np
import os
import pandas as pd

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

Get the data

Reading the csv file

In [190]:

```
shark_data = pd.read_csv('/Users/rahmi/Desktop/Machine Learning/Shark-Attack/DataMining/Data/WEKAFinalInputFile.csv')
```

Considered only non-normalised and columns that are not binned so that I can understand the true nature of the data and how the data is spread.

In [191]:

```
df = shark_data[['TurtleExactCountSC', 'TurtleExactCountNC',
```

```
'TurtleAttackActivity','Area','Location','Time','Species','Attack','Timeofa
ttack','Beach','County','MoonPhaseExtended','MoonPhase','Precipitation_Value
','StationPressure','WindSpeed','Salinity','Turbidity','Temperature','Disso
vedO2','CrabLandings','Degree','Direction','MoonPhase3daysextended','MoonPh
ase4daysextended', 'changetemp']]
```

In [192]:

```
df.head()
```

Out[192]:

	TurtleExactCountSC	TurtleExactCountNC	TurtleAttackActivity	Area	Location	Time	Spec
0	0.0	0.0	NaN	NaN	NaN	NaN	NaN
1	0.0	0.0	NaN	NaN	NaN	NaN	NaN
2	1.0	0.0	NaN	NaN	NaN	NaN	NaN
3	0.0	0.0	NaN	NaN	NaN	NaN	NaN
4	2.0	0.0	NaN	NaN	NaN	NaN	NaN

5 rows × 26 columns

In [193]:

```
shark_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 186 entries, 0 to 185
Data columns (total 73 columns):
Unnamed: 0                          186 non-null int64
X.2                                 186 non-null int64
X.1                                 186 non-null int64
X                                   186 non-null int64
Id                                  186 non-null int64
Date                                186 non-null object
ID                                  186 non-null int64
TurtleExactCountSC                  151 non-null float64
turtleexactdiscretizeSC             151 non-null object
TurtleExactCountNC                  149 non-null float64
TurtleExactCombined                 186 non-null int64
TurtleexactdiscretizeNC             149 non-null object
TurtleAttackActivity                72 non-null float64
TurtleAttackActivityDiscretized     72 non-null object
Area                                72 non-null object
Location                            72 non-null object
Time                                60 non-null object
Species                             52 non-null object
Attack                              186 non-null object
Timeofattack                        72 non-null object
Beach                               72 non-null object
County                              72 non-null object
MoonPhaseExtended                   186 non-null object
MoonPhase                           186 non-null object
Precipitation_Value                 182 non-null object
StationPressure                     182 non-null float64
```

```
WindSpeed                          182 non-null float64
Salinity                           182 non-null float64
Turbidity                          182 non-null float64
Temperature                        182 non-null float64
DissovedO2                         182 non-null float64
PrecipitationValueMod              186 non-null float64
StationPressureMod                 186 non-null float64
WindSpeedMod                       186 non-null float64
SalinityMod                        186 non-null float64
TurbidityMod                       186 non-null float64
TemperatureMod                     186 non-null float64
DissovedO2Mod                      186 non-null float64
DissolvedO2discretize              186 non-null object
salinitydiscretize                 186 non-null object
turbiditydiscretize                186 non-null object
temperaturediscretize              186 non-null object
precipitationdiscretize            186 non-null object
pressurediscretize                 186 non-null object
windspeeddiscretize                186 non-null object
prepmovingaverage                  186 non-null float64
precipitationmvadiscretize         186 non-null object
CrabLandings                       186 non-null int64
CrabLandingsnormalised             186 non-null float64
CrabLandingsDisc                   186 non-null object
Degree                             186 non-null float64
Direction                          186 non-null object
MoonPhase3daysextended             186 non-null object
MoonPhase4daysextended             186 non-null object
zscorewatertemp                    186 non-null float64
changetemp                         186 non-null object
Precipitation_Normalised           186 non-null float64
StationPressure_Normalised         186 non-null float64
WindSpeed_Normalised               186 non-null float64
Salinity_Normalised                186 non-null float64
Turbidity_Normalised               186 non-null float64
Dissolved02_Normalised             186 non-null float64
Precipitation_minmax               186 non-null float64
StationPressure_minmax             186 non-null float64
WindSpeed_minmax                   186 non-null float64
Salinity_minmax                    186 non-null float64
Turbidity_minmax                   186 non-null float64
Dissolved02_minmax                 186 non-null float64
WaterTemp_minmax                   186 non-null float64
Turtle_minmax                      186 non-null float64
Crablandings_minmax                186 non-null float64
turbidity_kmeans_binning           186 non-null object
turbidty_domain                    186 non-null object
dtypes: float64(35), int64(8), object(30)
memory usage: 106.2+ KB
```

In [194]:

```
df["Attack"].value_counts()
```

Out[194]:

```
No     114
Yes     72
Name: Attack, dtype: int64
```

In [195]:

```
df.describe()
```

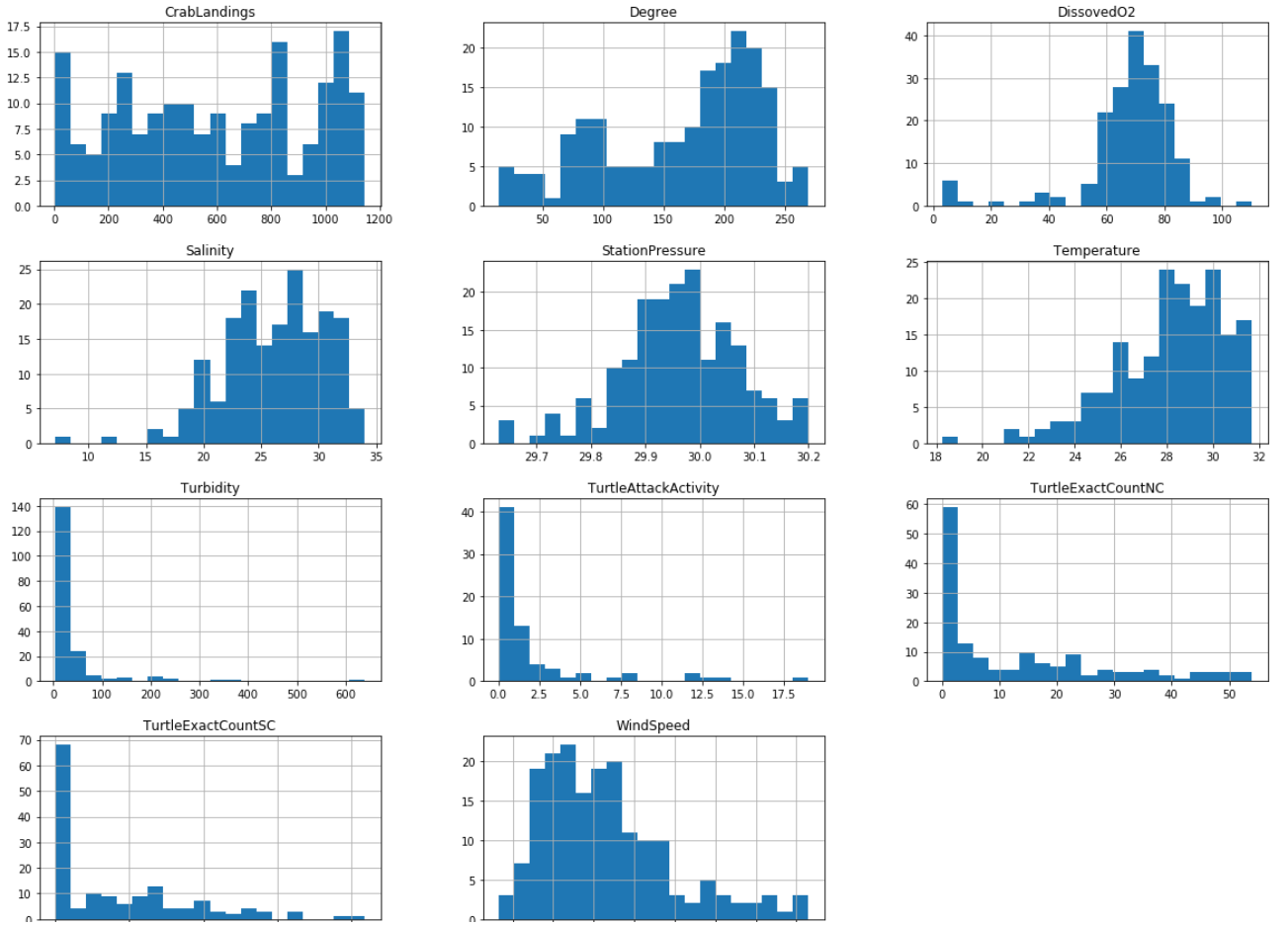
Out[195]:

	TurtleExactCountSC	TurtleExactCountNC	TurtleAttackActivity	StationPressure	WindS
count	151.000000	149.000000	72.000000	182.000000	182.0(
mean	40.920530	13.328859	1.902778	29.962308	6.5516
std	48.536519	15.538724	3.849857	0.111098	3.1695
min	0.000000	0.000000	0.000000	29.630000	1.3000
25%	0.000000	0.000000	0.000000	29.900000	4.3000
50%	23.000000	7.000000	0.000000	29.965000	5.9000
75%	67.500000	22.000000	1.250000	30.040000	7.8750
max	209.000000	54.000000	19.000000	30.200000	16.600

It has been observed that "TurtleExactCountNC", "TurtleExactCountSC" have missing values in 37 instances. Except that all other influential attributes have no missing values.

In [196]:

```
%matplotlib inline
import matplotlib.pyplot as plt
df.hist(bins=20, figsize=(20,15))
plt.show()
```



Here we can observe that shark attacks are more where the turtle presence is less.
Shark attacks are more where the turbidity is less and salinity and temperature is more.

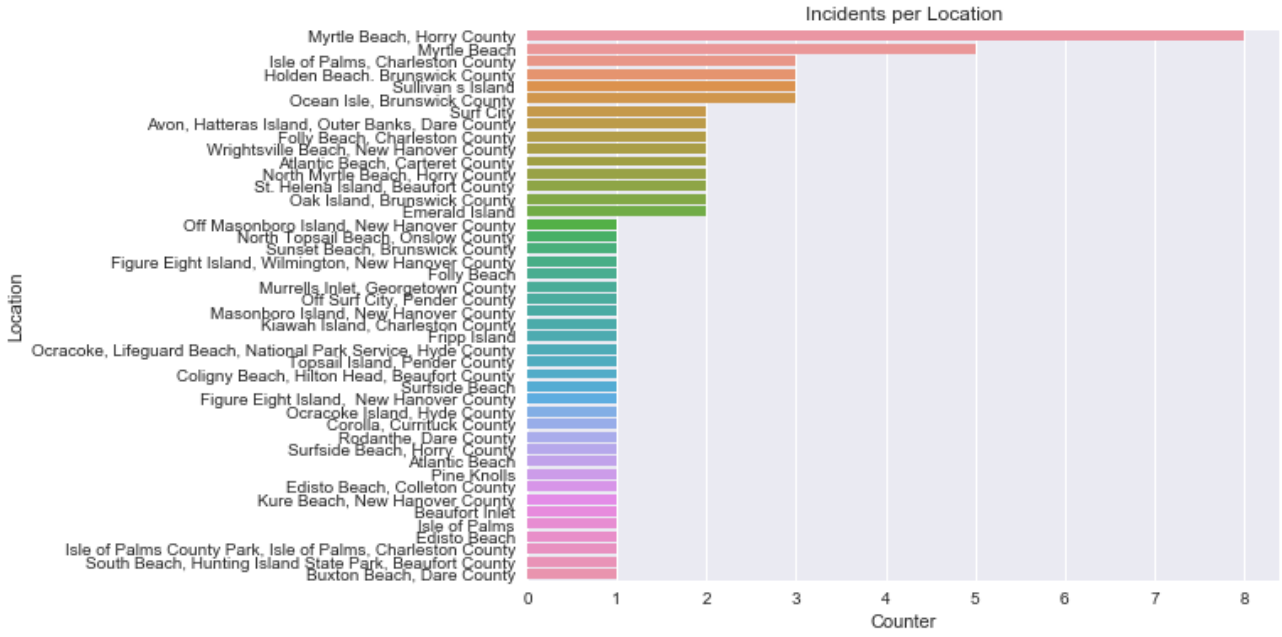
Exploratory Data Analysis

In [197]:

```
import seaborn as sns

table_count = shark_data['Location'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig,ax = plt.subplots(figsize=(8,6))
sns.barplot(x = type_values , y = type_index , ax =ax , orient='h')
plt.title('Incidents per Location')
plt.xlabel('Counter')
plt.ylabel('Location')
```

Out[197]:

```
<matplotlib.text.Text at 0x1184a8898>
```



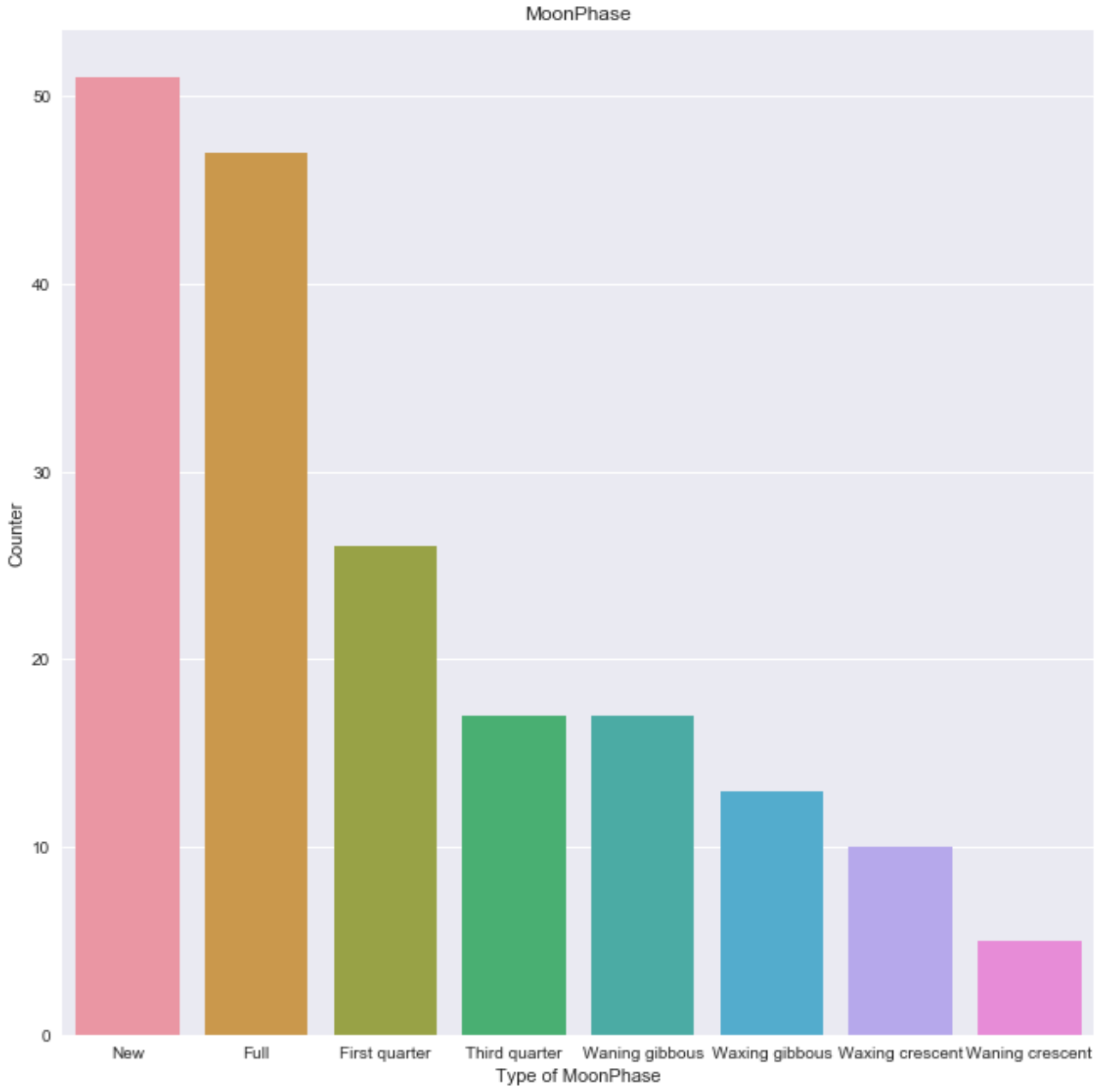
It is clearly seen that Myrtle Beach has highest number of shark attacks.

In [198]:

```
table_count = shark_data['MoonPhaseExtended'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig,ax = plt.subplots(figsize=(11,11))
sns.barplot(x = type_index,y=type_values,ax=ax)
plt.title('MoonPhase')
plt.xlabel('Type of MoonPhase')
plt.ylabel('Counter')
```

Out[198]:

```
<matplotlib.text.Text at 0x11980da58>
```



Attacks are almost equal in both the cases New moon and full moon. As we know tides favour the shark attacks, but when moon and sun align, known as full or new moon, the pull is at its strongest, causing the tides to be at their highest and lowest. This is known as spring tides. And because of the change from high tides to low tides and back again happens so quickly, sharks may move in areas closer to where people swim when the tide is low.

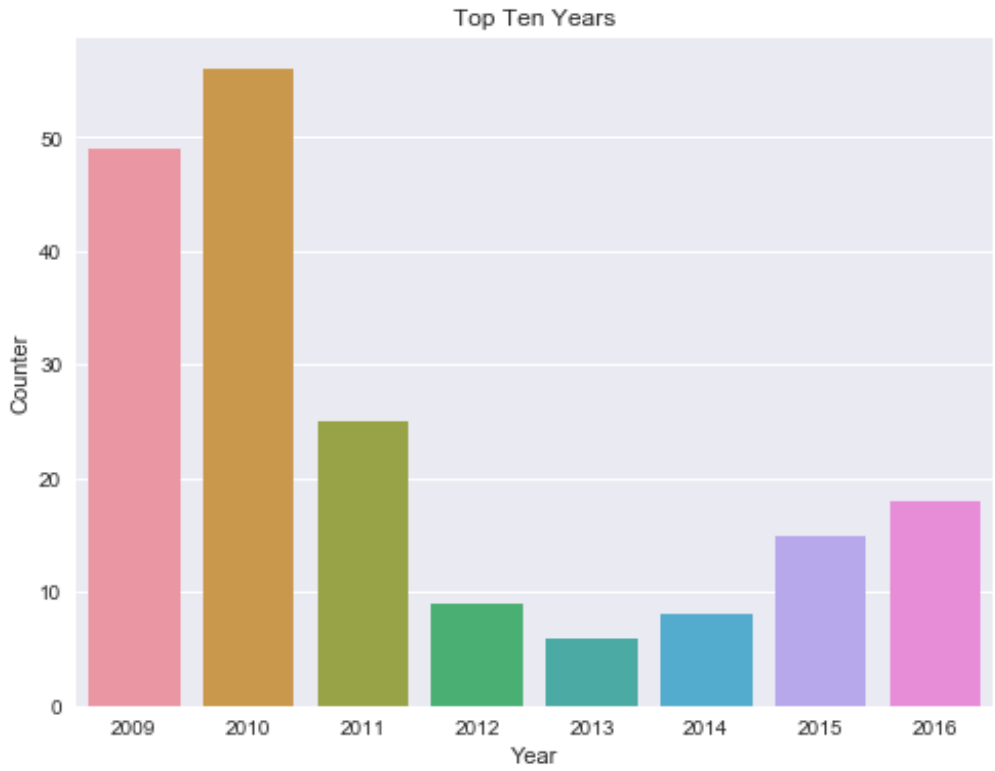
In [199]:

```
from collections import Counter

dates = pd.to_datetime(shark_data['Date'],dayfirst=True, errors='coerce')
year = dates.dropna().map(lambda x:x.year)
year_counter = Counter(year).most_common(10)
year_index = [year[0] for year in year_counter]
year_values = [year[1] for year in year_counter]

fig,ax = plt.subplots(figsize=(8,6))
```

```
sns.barplot(x = year_index,y=year_values,ax=ax)
plt.title('Top Ten Years')
plt.xlabel('Year')
plt.ylabel('Counter')
```

Out[199]:

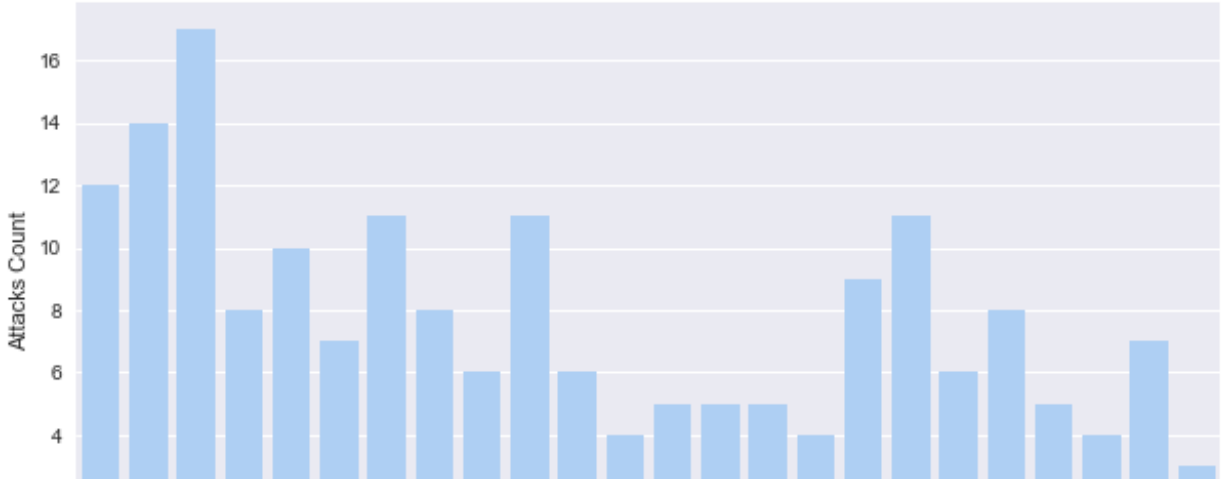
```
<matplotlib.text.Text at 0x119e376d8>
```

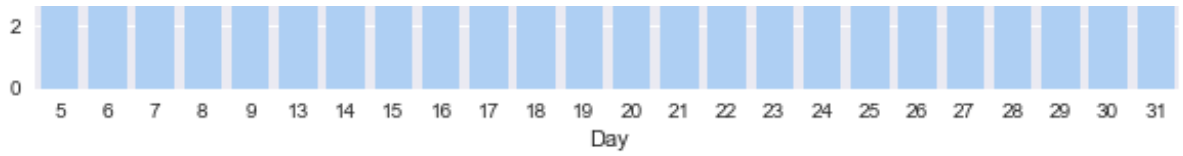


The plot shows the top 10 years when the number of shark attacks are the highest.

In [200]:

```
days = dates.dropna().map(lambda x: x.day)

days_counter = Counter(days)
days_keys = list(days_counter.keys())
days_values = list(days_counter.values())

fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=days_keys, y=days_values, color='#a2cffe', ax=ax)
ax = ax.set(ylabel="Attacks Count", xlabel="Day")
```





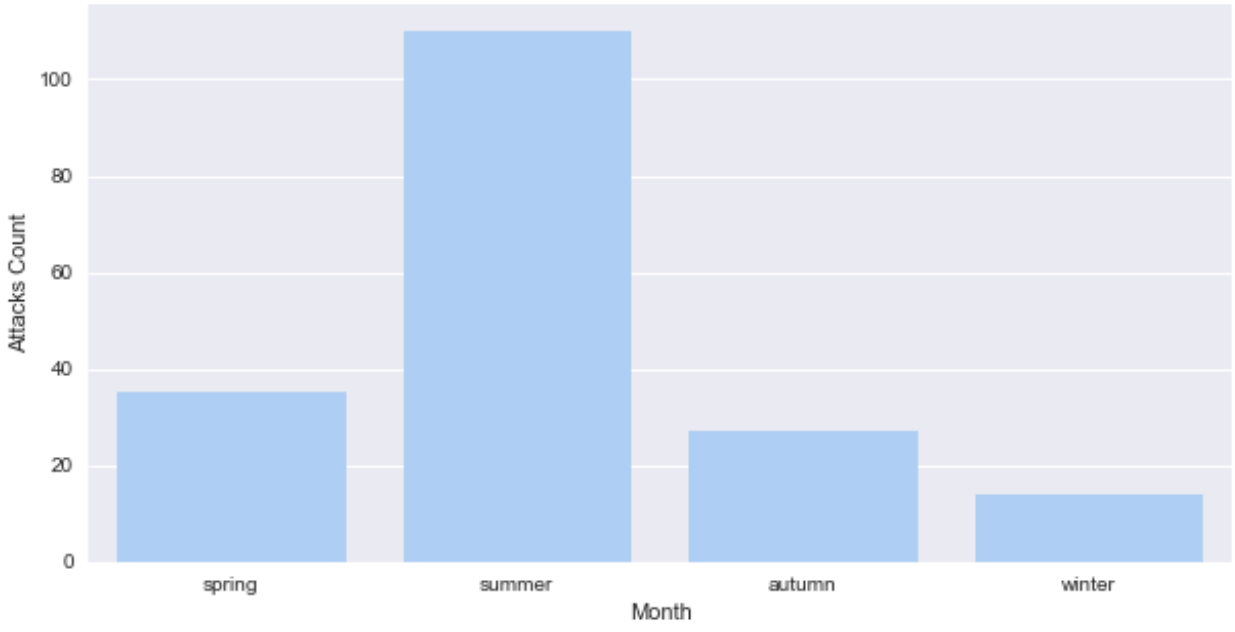
In [201]:

```
months = dates.dropna().map(lambda x: x.month)

def get_season(month):
    if month >= 3 and month <= 5:
        return 'spring'
    elif month >= 6 and month <= 8:
        return 'summer'
    elif month >= 9 and month <= 11:
        return 'autumn'
    else:
        return 'winter'

months_labels = months.apply(get_season)

months_counter = Counter(months_labels)
months_keys = list(months_counter.keys())
months_values = list(months_counter.values())
```

Here categorising the months into seasons to identify which particular group of months has highest shark attacks. It gives an answer to the question is the weather playing a key role to attract sharks come closer to the shore.

In [202]:

```
fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=months_keys, y=months_values, color='#a2cffe', ax=ax)
ax = ax.set(ylabel="Attacks Count", xlabel="Month")
```



Here we see that during summer the attacks are more. Many people go on a vacation during their break to do the beaches to do various activities like snorkelling , surfing etc. Is it causing a disturbance in waters?

In [203]:

```
most_common_species = Counter(shark_data['Species'].dropna().tolist()).most
_common(20)

species = [species_list[0] for species_list in most_common_species]
for i in species:
    print(i)
```

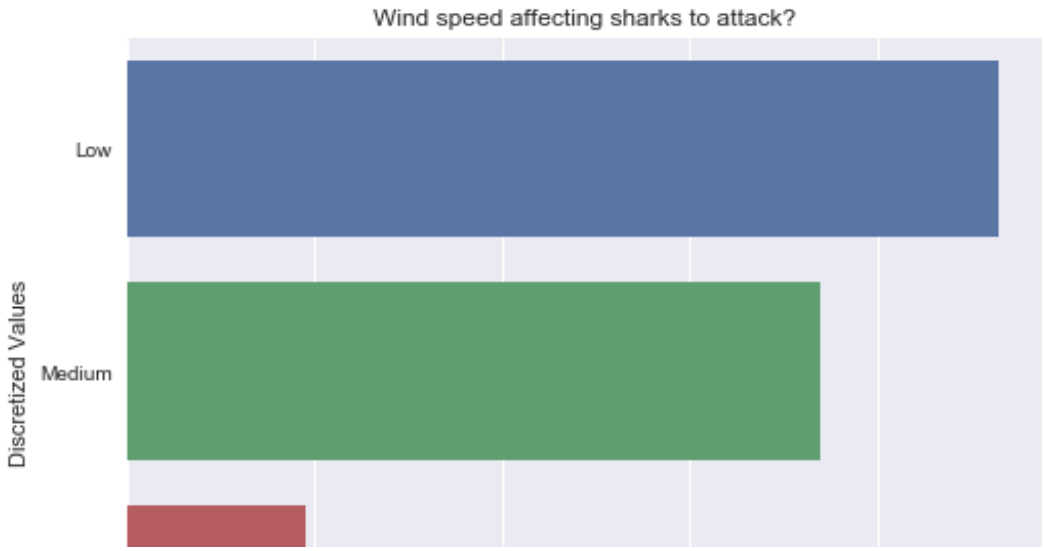
```
Unknown Species
4  shark
4  to 5  shark
Shark involvement not confirmed
Bull shark
Shark involvement prior to death not confirmed
Sandtiger shark, 4  to 5
Possibly a small blacktip shark
Possibly a 5  to 6  sandtiger shark
18  to 24  shark
small blacktip shark?
Blacktip shark
No shark involvement
Bull shark, 8
Possibly juvenile tiger shark
4  tp 5  shark
4  to 6  shark
6  shark
small shark
Sandtiger shark
```

In [204]:

```
table_count = shark_data['windspeeddiscretize'].value_counts()
type_index = table_count.index
type_values = table_count.values
fig,ax = plt.subplots(figsize=(8,6))
sns.barplot(x = type_values , y = type_index , ax =ax , orient='h')
plt.title('Wind speed affecting sharks to attack?')
plt.xlabel('Wind speed')
plt.ylabel('Discretized Values')
```

Out[204]:

```
<matplotlib.text.Text at 0x11928ce48>
```



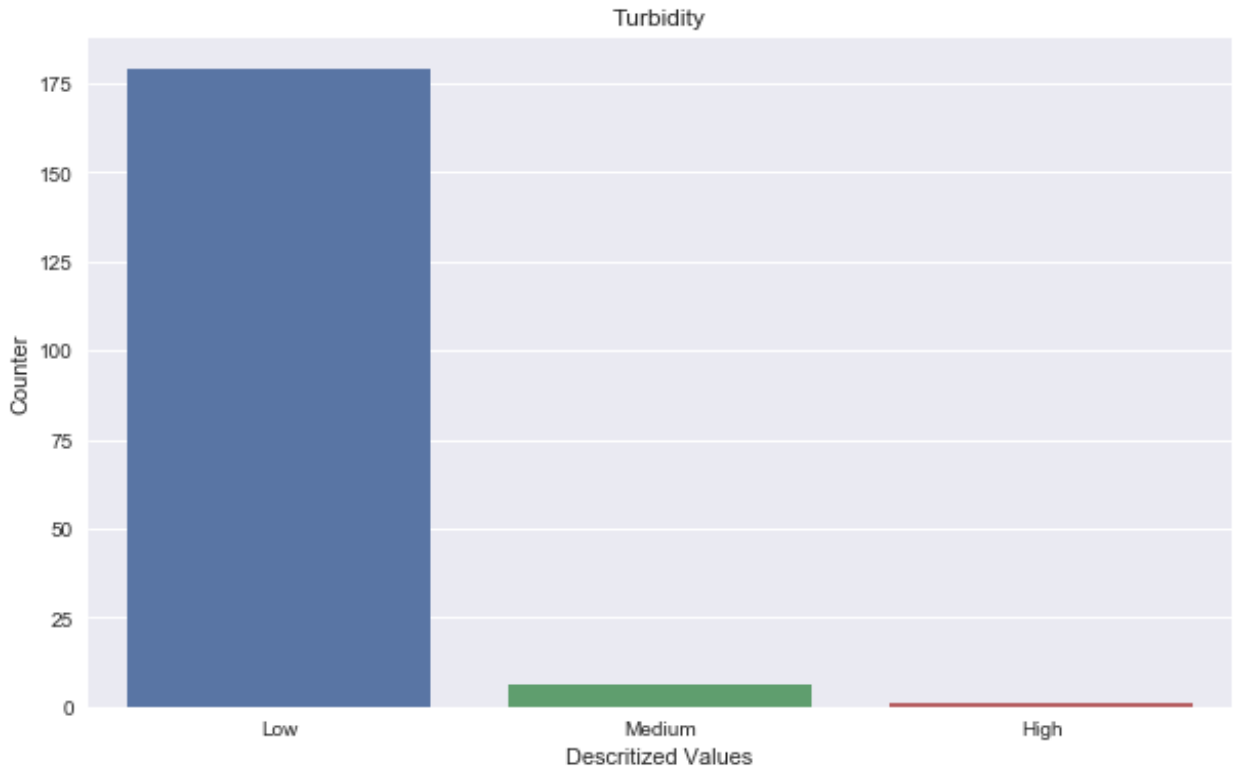
It is clearly shown that the increase in wind speed is actally decreasing the attacks.

In [205]:

```
table_count = shark_data['turbiditydiscretize'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig,ax = plt.subplots(figsize=(10,6))
sns.barplot(x = type_index,y=type_values,ax=ax)
plt.title('Turbidity')
plt.xlabel('Descritized Values')
plt.ylabel('Counter')
```

Out[205]:

```
<matplotlib.text.Text at 0x118242a20>
```



When the turbidity is low, the clarity of water is high and that is causing more attacks to happen by sharks

In [206]:

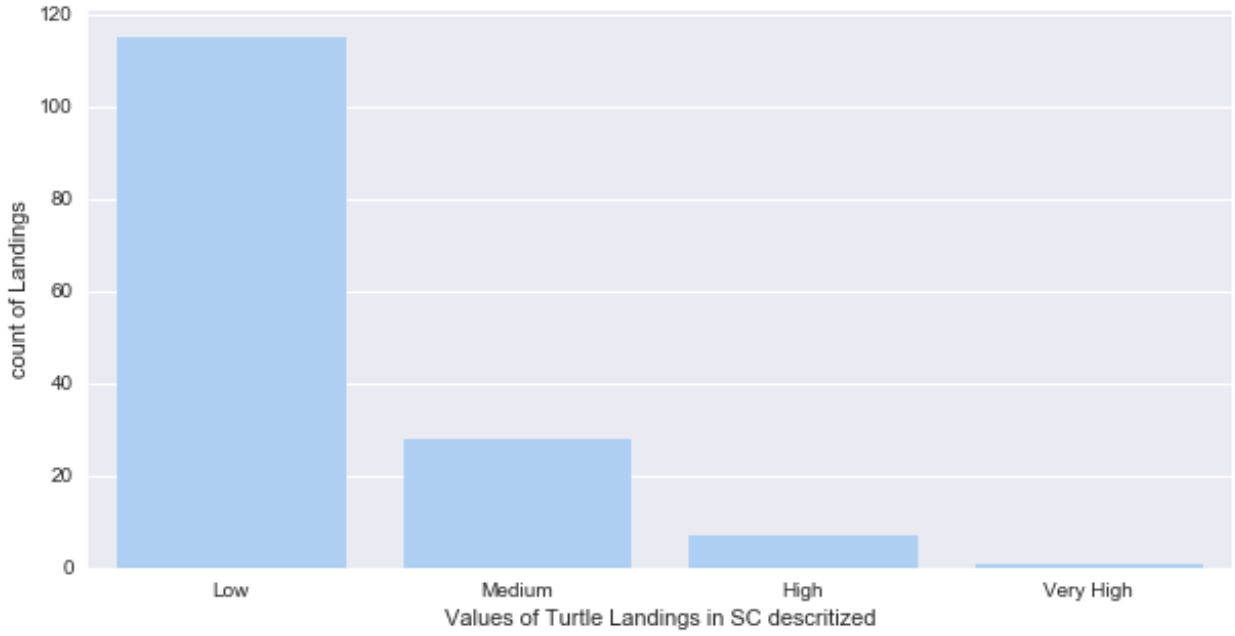
```
table_count = shark_data['CrabLandingsDisc'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig, ax = plt.subplots(figsize=(10, 5))
```

```
sns.barplot(x=type_index, y=type_values, color='#a2cffe', ax=ax)
ax = ax.set(ylabel="count of Landings", xlabel="Values of crab Landings des
critized")
```

So, it is clear that where the crab landings are more, the shark attacks are more.

In [207]:

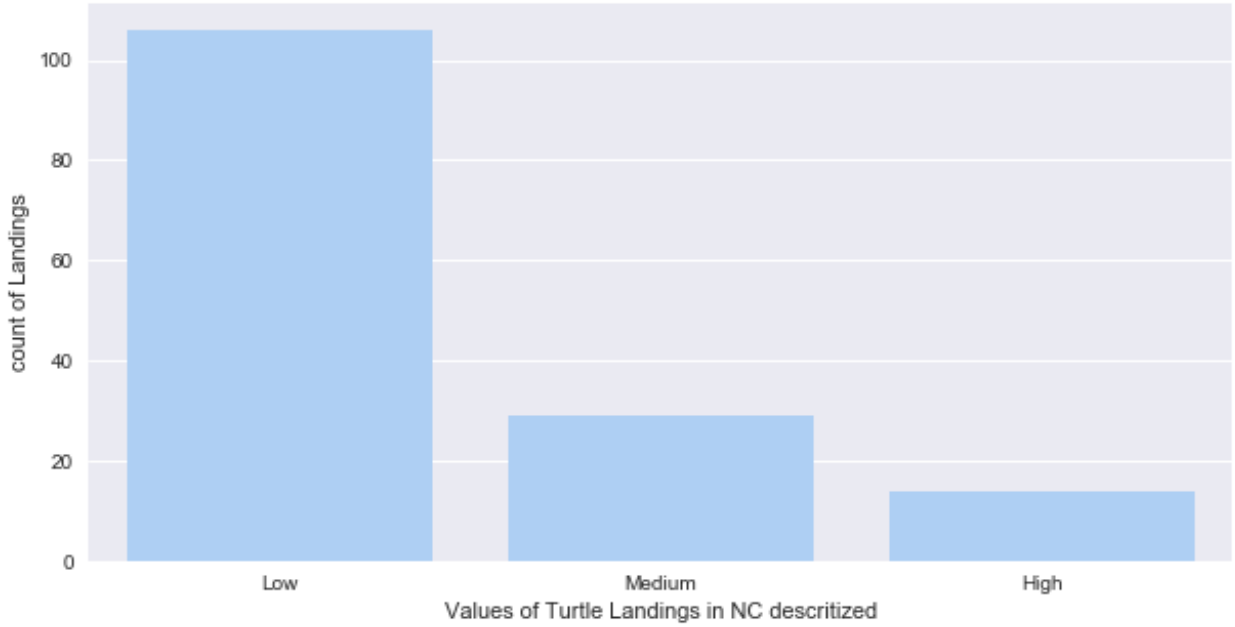
```
table_count = shark_data['turtleexactdiscretizeSC'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=type_index, y=type_values, color='#a2cffe', ax=ax)
ax = ax.set(ylabel="count of Landings", xlabel="Values of Turtle Landings i
n SC descritized")
```



In south Carolina, the turtle landings or presence is less when the shark attacks have happened.

In [208]:

```
table_count = shark_data['TurtleexactdiscretizeNC'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=type_index, y=type_values, color='#a2cffe', ax=ax)
ax = ax.set(ylabel="count of Landings", xlabel="Values of Turtle Landings i
n NC descritized")
```



Even in North Carolina, the turtles presence is less but the attacks are more. So the sharks may not interested to feed themsleves the turtles but interested in crabs.

Note:

It is better if humans reduce the activities where the number of crabs that are present on the shore are more.

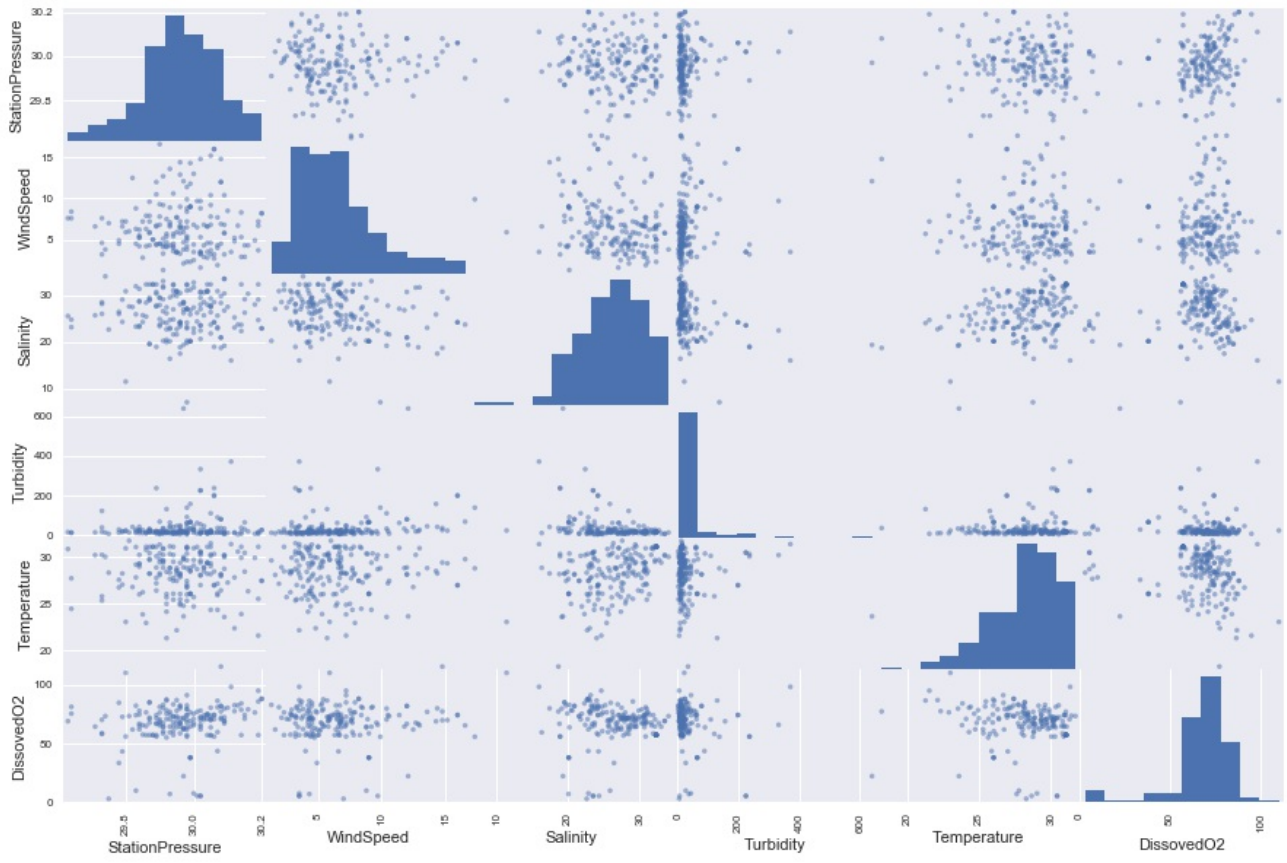
In [209]:

```
from pandas.plotting import scatter_matrix

attributes = ["Precipitation_Value", "StationPressure", "WindSpeed",
              "Salinity","Turbidity","Temperature","DissovedO2"]
scatter_matrix(shark_data[attributes], figsize=(15, 10))
```

Out[209]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x118d819e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11827dd68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11827ff28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x118ccefd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x117f22470>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x117f22d30>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x118e44c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1182eceb8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x118e7c5c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x117de2d68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1182e1da0>,
```

```
<matplotlib.axes._subplots.AxesSubplot object at 0x118003978>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x11976ceb8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x118e27e48>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x119324128>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x117f84ba8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x117df2e80>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x118e07be0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x118c7a978>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x1193d9908>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x117fbb278>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11965eeb8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x1195dbc18>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x1195f8978>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x119449518>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a5376a0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a596470>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a608128>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a663ac8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a6dbda0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x11a740d30>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a7826a0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a810320>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a8212b0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a8d5cc0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x11a935ac8>]], dtype=object)
```



In [210]:

```
corr_matrix = shark_data.corr()
corr_matrix["Temperature"].sort_values(ascending=False)
```

Out[210]:

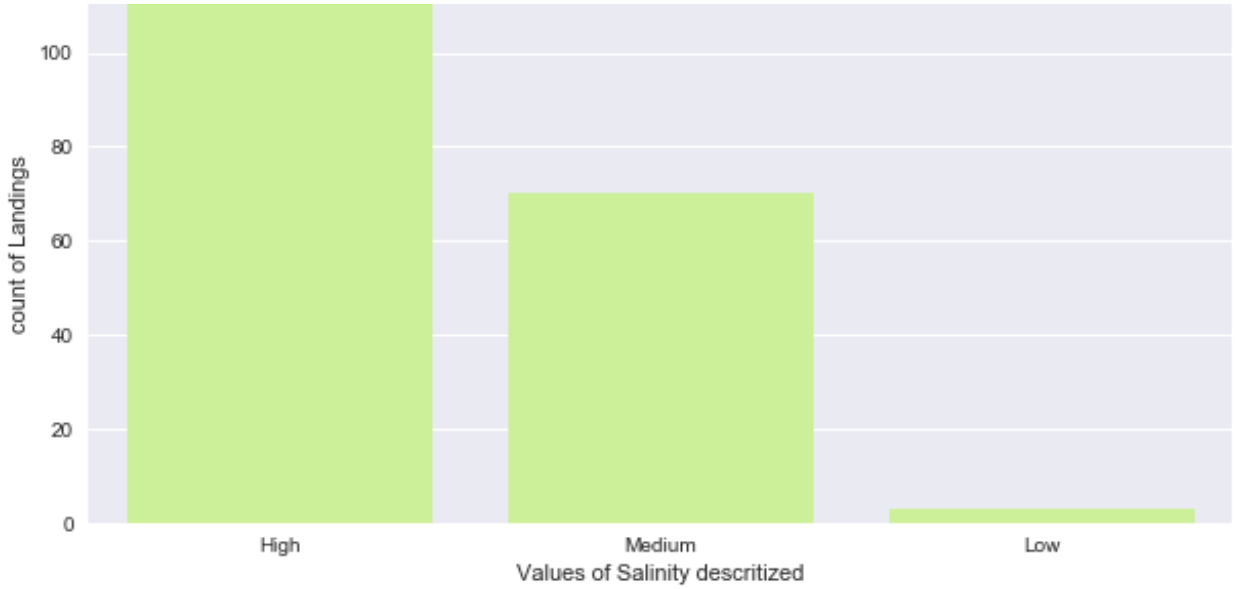
```
TemperatureMod                1.000000
```

```
zscorewatertemp                1.000000
Temperature                    1.000000
WaterTemp_minmax               1.000000
Salinity_minmax                0.359724
SalinityMod                    0.359724
Salinity                       0.359724
Salinity_Normalised            0.359724
ID                             0.343491
TurtleExactCountNC             0.295099
Degree                         0.130454
TurtleExactCountSC             0.113914
prepmovingaverage              0.096064
Turtle_minmax                  0.071645
TurtleExactCombined            0.071645
X                              0.066547
X.1                            0.066547
X.2                            0.066547
Unnamed: 0                     0.066547
TurtleAttackActivity           0.064238
Precipitation_minmax           0.050704
PrecipitationValueMod          0.050704
Precipitation_Normalised       0.050704
Id                             0.036675
Turbidity_Normalised          -0.021032
Turbidity                     -0.021032
TurbidityMod                  -0.021032
Turbidity_minmax              -0.021032
CrabLandings                  -0.056003
Crablandings_minmax           -0.056003
CrabLandingsnormalised        -0.056003
StationPressure_minmax        -0.059583
StationPressure_Normalised    -0.059583
StationPressure               -0.059583
StationPressureMod            -0.059583
WindSpeedMod                  -0.074705
WindSpeed                     -0.074705
WindSpeed_Normalised          -0.074705
WindSpeed_minmax              -0.074705
DissovedO2Mod                 -0.171206
DissovedO2                    -0.171206
Dissolved02_Normalised        -0.171206
Dissolved02_minmax            -0.171206
Name: Temperature, dtype: float64
```

Temperature is positively correalted with the Salinity and Precipitation. It is negatively correalted with Turbidity, StationPressure, Windspeed and Dissolved O2.

In [211]:

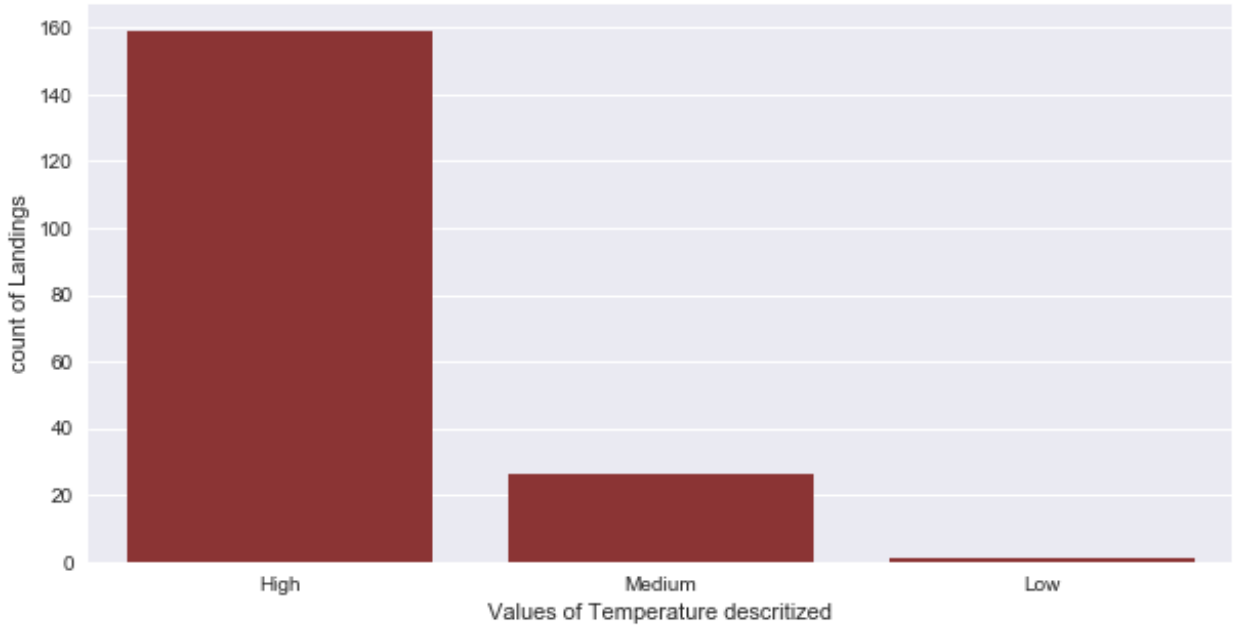
```
table_count = shark_data['salinitydiscretize'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=type_index, y=type_values, color='#cfff8a', ax=ax)
ax = ax.set(ylabel="count of Landings", xlabel="Values of Salinity
descritized")
```



High Salinity is responsible for the shark attacks to increase

In [212]:

```
table_count = shark_data['temperaturediscretize'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=type_index, y=type_values, color='#9a2526', ax=ax)
ax = ax.set(ylabel="count of Landings", xlabel="Values of Temperature descr
itized")
```

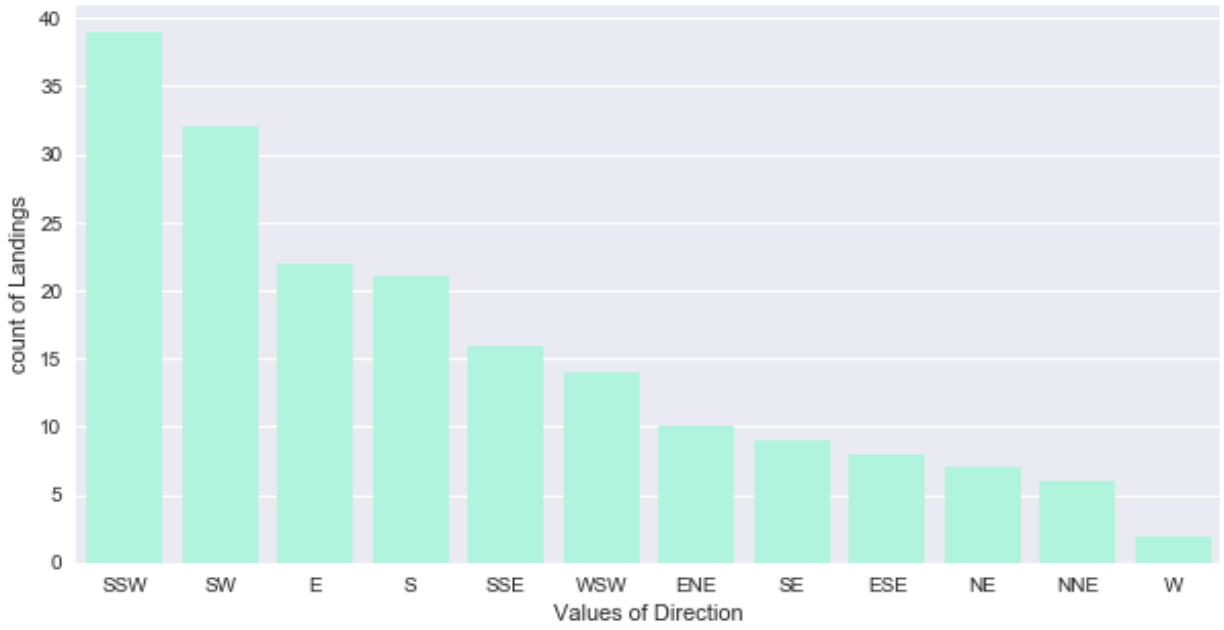


Temperature increase is also responsible for high sjark attacks.

In [213]:

```
table_count = shark_data['Direction'].value_counts()
type_index = table_count.index
type_values = table_count.values

fig, ax = plt.subplots(figsize=(10, 5))
```

```
fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=type_index, y=type_values, color='#a5ffe0', ax=ax)
ax = ax.set(ylabel="count of Landings", xlabel="Values of Direction")
```



winds that blow in the South-south west and south west direction may probably indirectly affect the sharks to attack may be due the change in the direction of water starts moving or may be the sharks presence is more in that region

Train and Test Data set Split

Converted the target attribute Attack into flag variable 0 and 1

In [214]:

```
shark_data['Attack'] = shark_data['Attack'].map({'No': 1, 'Yes': 0})
shark_data['Attack']
```

Out[214]:

```
0       1
1       1
2       1
3       1
4       1
5       1
6       1
7       1
8       1
9       1
10      1
11      1
12      1
13      1
14      1
15      1
16      1
17      1
18      1
19      1
```

```
19      1
20      1
21      1
22      1
23      1
24      1
25      0
26      1
27      1
28      1
29      1
       ..
156     0
157     0
158     0
159     0
160     0
161     0
162     0
163     0
164     0
165     0
166     0
167     0
168     0
169     0
170     0
171     0
172     0
173     0
174     0
175     0
176     0
177     0
178     0
179     0
180     0
181     0
182     0
183     0
184     0
185     0
Name: Attack, Length: 186, dtype: int64
```

Identified the important and influential attributes:

In [215]:

```
model_variables =
['MoonPhaseExtended','temperaturediscretize','DissolvedO2discretize', 'sali
nitydiscretize','turbiditydiscretize',
            'precipitationdiscretize', 'windspeeddiscretize',
'pressurediscretize', 'Direction','Attack']

shark_data_relevent = shark_data[model_variables]
```

Converted the attributes that are selected for modelling to flag variables.

In [216]:

```
shark_relevant_enconded = pd.get_dummies(shark_data_relevent)
shark_relevant_enconded
```

Out[216]:

	Attack	MoonPhaseExtended_First quarter	MoonPhaseExtended_Full	MoonPhaseExtended_New
0	1	1	0	0
1	1	0	0	0
2	1	0	1	0
3	1	0	1	0
4	1	0	1	0
5	1	0	0	0
6	1	0	0	0
7	1	0	0	0
8	1	0	0	1
9	1	0	0	0
10	1	0	0	0
11	1	0	0	0
12	1	0	1	0
13	1	0	1	0
14	1	0	0	0
15	1	0	0	0
16	1	0	0	0
17	1	0	0	1
18	1	0	0	0
19	1	1	0	0
20	1	1	0	0
21	1	0	1	0
22	1	0	1	0
23	1	0	0	0
24	1	0	0	0
25	0	0	0	1
26	1	0	0	1
27	1	0	0	0
28	1	0	1	0
29	1	0	0	0

...	... Attack	MoonPhaseExtended_First quarter	... MoonPhaseExtended_Full	... MoonPhaseExtended_New
156	0	0	0	1
157	0	1	0	0
158	0	1	0	0
159	0	0	0	0
160	0	0	1	0
161	0	0	1	0
162	0	0	1	0
163	0	0	1	0
164	0	0	0	0
165	0	0	1	0
166	0	1	0	0
167	0	0	0	0
168	0	0	0	0
169	0	1	0	0
170	0	1	0	0
171	0	0	0	0
172	0	0	0	0
173	0	0	1	0
174	0	0	1	0
175	0	0	1	0
176	0	0	1	0
177	0	0	0	0
178	0	0	0	0
179	0	0	0	0
180	0	0	0	1
181	0	0	0	1
182	0	0	0	0
183	0	0	0	0
184	0	0	1	0
185	0	0	1	0

186 rows × 42 columns

Splitting the dataset into test and train by over sampling as the target variable values of YES and NO are imbalanced using **SMOTE**

In [217]:

```
training_features, test_features,training_target, test_target, =
train_test_split(shark_relevant_enconded.drop(['Attack'], axis=1),
                                                shark_relevant_enconded['Att
k'],
                                                test_size = .2,
                                                random_state=42)
```

In [218]:

```
x_train, x_val, y_train, y_val = train_test_split(training_features, traini
ng_target,
                                                    test_size = .2,
                                                    random_state=42)
```

In [219]:

```
sm = SMOTE(random_state=42, ratio = 1.0)
x_train_res, y_train_res = sm.fit_sample(x_train, y_train)
```

Modelling

1. Random Forest

Random forest algorithm introduces extra randomness when growing trees and it searches for the best feature in the random subset of features. This results in a greater tree diversity which trades a higher bias for a lower variance, yielding a better model so ,implementing the random forest classification algorithm to learn the data on training dataset

In [220]:

```
from sklearn.ensemble import RandomForestRegressor
clf_rf = RandomForestClassifier(n_estimators=25, random_state=12)
clf_rf.fit(x_train_res, y_train_res)
```

Out[220]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=25, n_jobs=1, oob_score=False, random_state=12,
            verbose=0, warm_start=False)
```

Here, I am checking the accuracy and recall scores on both training and testing data.

In [221]:

```
print('Validation Results')
print('Accuracy',clf_rf.score(x_val, y_val))
print('Recall', recall_score(y_val, clf_rf.predict(x_val)))
print('\nTest Results')
print('Accuracy',clf_rf.score(test_features, test_target))
```

```
print('Recall',recall_score(test_target, clf_rf.predict(test_features)))
```

```
Validation Results
Accuracy 0.666666666667
Recall 0.7

Test Results
Accuracy 0.842105263158
Recall 0.909090909091
```

Calculating the Root Mean Squared Error

In [222]:

```
from sklearn.metrics import mean_squared_error

shark_data_predictions = clf_rf.predict(x_train_res)
forest_mse = mean_squared_error(y_train_res, shark_data_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[222]:

```
0.11785113019775792
```

Here I am doing cross validation and printing out the scores i.e how well the model is able to generalise.

In [223]:

```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

The model on the whole is having an accuracy of 64.5% which was almost same when trained without cross validation.

In [224]:

```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(clf_rf, x_train_res, y_train_res,
                                scoring="neg_mean_squared_error", cv=3)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [ 0.64549722  0.61237244  0.45643546]
Mean: 0.57143504155
Standard deviation: 0.0824337711231
```

Here I am printing out the confusion matrix which the random forest has predicted on the test dataset

In [225]:

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(clf_rf, x_train_res, y_train_res, cv = 3)
conf_matrix = confusion_matrix(y_train_res, y_train_pred)
```

```
conf_matrix = confusion_matrix(y_train_res, y_train_pred)
conf_matrix
```

Out[225]:

```
array([[43, 29],
       [19, 53]])
```

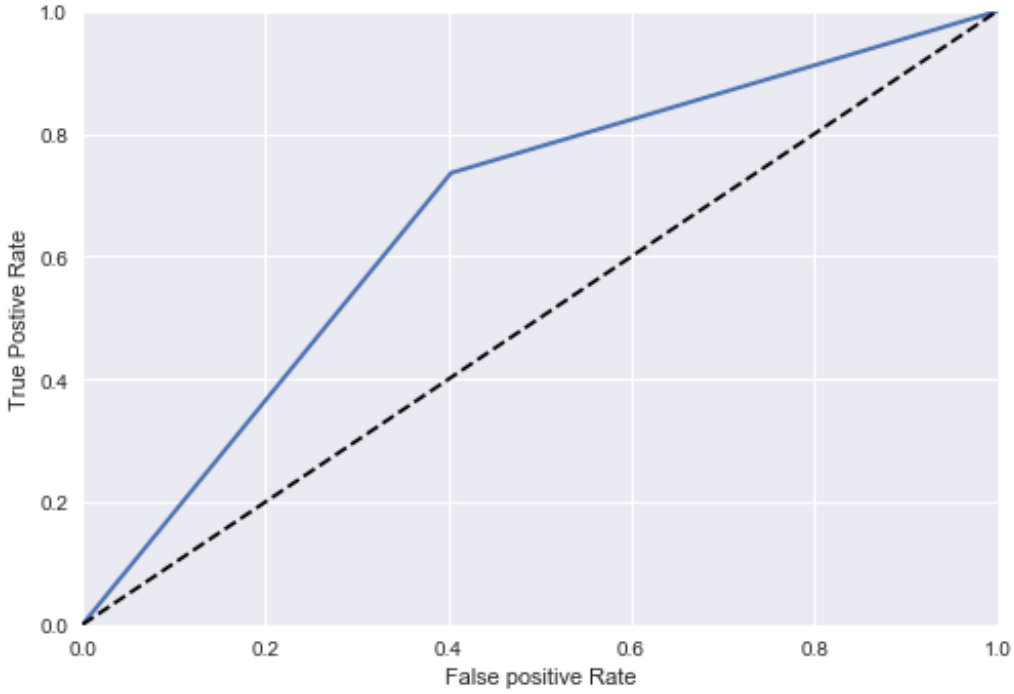
Plotting the ROC Curve

In [226]:

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_res,y_train_pred)
```

In [227]:

```
def plot_roc_curve(fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth=2, label= label)
    plt.plot([0,1],[0,1], 'k--')
    plt.axis([0,1,0,1])
    plt.xlabel('False positive Rate')
    plt.ylabel('True Postive Rate')
plot_roc_curve(fpr,tpr)
plt.show()
```



Here the ROC score is 66.6 % which is not bad for the model

In [228]:

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_res,y_train_pred)
```

Out[228]:

```
0.6666666666666674
```

2. SVM

Support Vector Machine is a very powerful machine learning model capacble of performing linear classification.

In [229]:

```
from sklearn import svm
clf = svm.SVC(kernel='linear', C = 1.0)
clf.fit(x_train_res,y_train_res)
```

Out[229]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [230]:

```
clf.score(x_train_res,y_train_res)
```

Out[230]:

```
0.83333333333333337
```

The accuracy of the model is 83.3%

In [231]:

```
predicted = clf.predict(test_features)
print('\nTest Results')
print('Accuracy',clf.score(test_features, test_target))
print('Recall',recall_score(test_target, clf.predict(test_features)))
```

```
Test Results
Accuracy 0.710526315789
Recall 0.863636363636
```

On the test data set the model generalised with accuracy of 71% and recall 86%

In [232]:

```
from sklearn.model_selection import cross_val_predict

y_train_predicted = cross_val_predict(clf, x_train_res, y_train_res, cv = 3
)
conf_matrix = confusion_matrix(y_train_res, y_train_predicted)
conf_matrix
```

Out[232]:

```
array([[37, 35],
       [18, 54]])
```

Printing the confusion Matrix

In [233]:

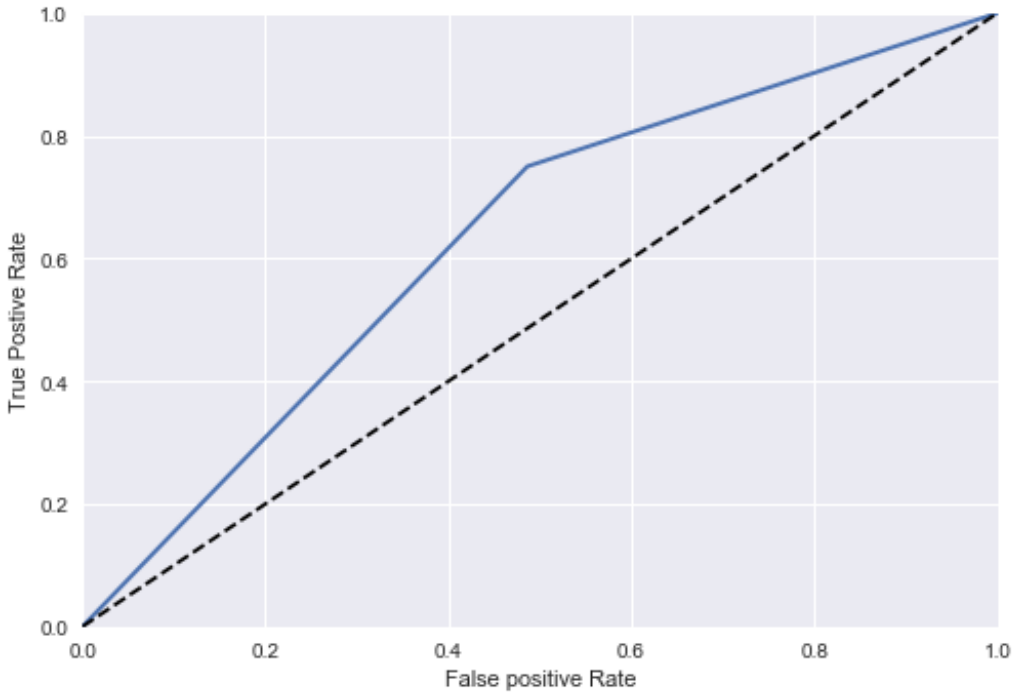
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_res,y_train_predicted)
```

```
fpr, tpr, thresholds = roc_curve(y_train_res,y_train_predicted)
```

ROC CURVE:

In [234]:

```
plot_roc_curve(fpr,tpr)
plt.show()
```



In [235]:

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_res,y_train_predicted)
```

Out[235]:

```
0.6319444444444442
```

Usually the ideal score is 1 and here the model has 0.63 which is fairly good

Results:

1. The sharks presence is more where there are more crabs and turtles presence is not attracting the sharks to come close to the shore.
2. The Moon Phase is influencing the sharks to come closer to coastal areas because during and after the moon phase their is change in the tides from high to low suddenly.
3. The shark attacks are more during summer.
4. when the temperature, salinity is high and turbidity, wind speed is low the shark attacks are more because the sharks presence is more if the temperature and salinity is high and when turbidity is low the clarity of water is high which may attract the sharks to come to shore as they can see the food clearly.
5. Models can predict the target variable "Attack" based on the above attributes that seemed to be influential i.e MOon phase extended, precipitation, temperature, turbidity, salinity, wind speed.