Analyzing the global shark attack data set

Introduction:

This data set contains the global shark attacks from as early as the 1800s all the way through 2018. It was obtained trhough kaggle. In this project I will analyze it extensively to see what factors contributed to shark attacks across the world.

In [1]:
```
# Importing the necessary libraries:
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
import re
```

In [2]:
```
# Importing the file and extracting the columns needed for this analysis.
df = pd.read_excel('attacks.xlsx')
df = df[['Date', 'Year', 'Type', 'Country', 'Sex ', 'Age', 'Time', 'Activity', 'Fatal (Y/N)']]
df = df.dropna().reset_index(drop=True)
print('Original length of dataframe:', len(df))
df.head()
```

```
Original length of dataframe: 2205
```

Out[2]:

	Date	Year	Type	Country	Sex	Age	Time	Activity	Fatal (Y/N)
0	2018-06-25 00:00:00	2018.0	Boating	USA	F	57	18h00	Paddling	N
1	2018-06-18 00:00:00	2018.0	Unprovoked	USA	F	11	14h00 -15h00	Standing	N
2	2018-06-09 00:00:00	2018.0	Invalid	USA	M	48	07h45	Surfing	N
3	2018-06-03 00:00:00	2018.0	Unprovoked	BRAZIL	M	18	Late afternoon	Swimming	Y
4	2018-05-26 00:00:00	2018.0	Unprovoked	USA	M	15	17h00	Walking	N

Cleaning up the original data set

A lot of the features in this data set are very messy and a lot of data tidying will be needed to get this data set ready to analyze.

The original data set has a Date column on it. Instead of displaying this, I will use it to extract only the month and create a seperate column for it.

The year column is displayed as a float type. I will change this to be integer as this makes more sense. The data set also contains some observations from the 1800's. I will drop these and only contain observations from the 1900's onwards.

In [3]:
```
# cleaning and creating the month column
df.Date = df.Date.astype('str')
month_year = df.Date.str.split('-')
df['month'] = month_year.str.get(1)
print(df.month.unique())
```

```
['06' '05' '04' '03' '02' '01' '12' '11' '10' '09' '08' '07' 'Feb' 'Sep'
 'Aug' 'Jul' 'Jun' 'Nov' 'Mar' 'May' nan 'Dec' '1985 or mid Jul' 'Jan'
 'Apr' '1942']
```

Still more work to do

Even after extracting the months half of the data set contains string values for the numbered months instead. There is still more work needed to fully clean this category.

In [4]:
```
df.month = df.month.replace(['Feb', 'Sep', 'Aug', 'Jul', 'Jun', 'Nov', 'Mar', 'May',
                             'Dec', '1985 or mid Jul', 'Jan', 'Apr', '1942'],
                            ['02', '09', '08', '07', '06', '11', '03', '05', '12', '07', '01', '04', np.nan])
df = df.dropna(subset=['month'])
df.month = df.month.astype('int64')

# cleaning up the year column
df.Year = df.Year[df.Year >= 1900]
df = df.dropna(subset=['Year'])
df.Year = df.Year.astype('int')

df = df.drop(columns=['Date'], axis=1)
df.head(1)
```

Out[4]:

	Year	Type	Country	Sex	Age	Time	Activity	Fatal (Y/N)	month
0	2018	Boating	USA	F	57	18h00	Paddling	N	6

Some more cleaning

Next I will be cleaning up the 'Type', 'Sex', and 'Fatal' column. There are some random values in here that dont make sense in the gender column. I will clean this up so only 'F' for female and 'M' for male are present.

For the Type column I will limit it to 'Provoked' and 'unprovoked'. Any one from the boating or 'Sea disaster' will go into the unprovoked category.

For the fatal column I will be removing the values that are not 'N' for no and 'Y' for yes.

In [5]:
```
print('Unique values in gender:\t', df['Sex '].unique())
print('Unique values in type:\t\t', df.Type.unique())
print('Unique values in type:\t\t', df['Fatal (Y/N)'].unique())
```

```
Unique values in gender:         ['F' 'M' 'M ' 'lli']
Unique values in type:           ['Boating' 'Unprovoked' 'Invalid' 'Provoked' 'Sea Disaster']
Unique values in type:           ['N' 'Y' 'M' 2017 'UNKNOWN' ' N']
```

In [6]:
```
# Cleaning up the 'Type' column:
df.Type = df.Type.map({'Boating':'Unprovoked', 'Unprovoked':'Unprovoked', 'Provoked':'Provoked', 'Sea Disaster':
df = df.dropna(subset=['Type'])

# Cleaning up the 'Sex' column:
df['Sex '] = df['Sex '].str.strip(' ')
df = df[df['Sex '].isin(['M', 'F'])]

# cleaning up the 'Fatal' column:
df['Fatal (Y/N)'] = df['Fatal (Y/N)'].str.strip(' ')
df = df[df['Fatal (Y/N)'].isin(['N', 'Y'])]

df = df.rename(columns={'Sex ': 'Gender', 'Fatal (Y/N)': 'Fatal', 'month': 'Month', 'year': 'Year'})
df.head(1)
```

Out[6]:

	Year	Type	Country	Gender	Age	Time	Activity	Fatal	Month
0	2018	Unprovoked	USA	F	57	18h00	Paddling	N	6

Now time to clean up the 'Age' feature:

The age column has many values in it that are not numerical values. The way I will deal with these is by creating a loop to loop through them and removing any rows that cannot be converted to integers.

In [7]:
```
df.Age.unique()
```

Out[7]:
```
array([57, 11, 18, 15, 12, 32, 10, 30, 60, 33, 37, 19, 25, 69, 55, 34, 35,
       '40s', 20, 54, 14, 22, 31, 17, 40, 28, 42, 3, 13, 50, 46, 82, 48,
       41, '20s', 21, 51, 39, 58, 26, 47, 16, 61, 65, 73, 36, 66, 29, 43,
       '60s', 49, 9, 59, 6, 64, 23, 52, 24, 45, 71, 44, 7, 27, 62, 38, 68,
       63, 70, 53, 8, 'teen', 77, 74, 56, '30s', 5, 86, 'Teen',
       '18 or 20', '12 or 13', 84, '\xa0 ', ' ', '30 or 36', '6½',
       '33 or 37', '20?', "60's", 75, '21 or 26', '>50', '(adult)',
       '25 or 28', '30 & 32', '13 or 18', '33 & 26', 'F', '9 or 10', '  ',
       '31 or 33'], dtype=object)
```

In [8]:
```
ages = []
for index, age in df.iterrows():
    try:
        if type(int(age['Age'])) == int:
            ages.append(age['Age'])
        else:
            continue
    except ValueError:
        ages.append(np.nan)
df = df.drop(columns=['Age'], axis=1)
ages = pd.Series(ages, name='Age')
df = pd.concat([df, ages], axis=1)
df = df.dropna()
df.head(1)
```

Out[8]:

	Year	Type	Country	Gender	Time	Activity	Fatal	Month	Age
0	2018.0	Unprovoked	USA	F	18h00	Paddling	N	6.0	57.0

Cleaning up the 'Time' variable:

This one has many different values and ways of representing time mixed together. Therefore I will need to write another loop to be able to fully clean this. It has many unique values but I will be limiting it to the numerical values of 0-23 to represent all times. Then I will further clean it to only show those times in three different categories: 'Morning', 'Midday', and 'Night'.

In [9]:
```
print('Number of unique values in the "Time" feature:\t', df.Time.nunique())
```

```
Number of unique values in the "Time" feature:   296
```

In [10]:
```
df.Time = df.Time.str.replace('h.+', '', regex=True)
times = []
for index, time in df.iterrows():
    try:
        if type(int(time['Time'])) == int:
            times.append(int(time['Time']))
        else:
            continue
    except ValueError:
        if time['Time'] in ['Afternoon', 'Early afternoon', 'Late afternoon', 'Midday', 'P.M.']:
            times.append(12)
        elif time['Time'] in ['Morning', 'A.M.', 'Early morning']:
            times.append(10)
        elif time['Time'] in ['Nig', 'Evening', 'Dusk']:
            times.append(20)
        else:
            continue
times = pd.Series(times, name='Time')
df = df.drop(columns=['Time'], axis=1)
df = pd.concat([df, times], axis=1)
def time_of_day(x):
    if x < 12:
        return 'Morning'
    elif x >= 12 and x < 18:
        return 'Midday'
    else:
        return 'Night'
df.Time = df.Time.apply(time_of_day)
df = df.dropna()
df.Age = df.Age.astype('int64')
df.Year = df.Year.astype('int64')
df.Month = df.Month.astype('int64')
print('Unique values in the "Time" feature after cleaning it up:\t', df.Time.unique())
df.head(1)
```

```
Unique values in the "Time" feature after cleaning it up:        ['Night' 'Midday' 'Morning']
```

Out[10]:

	Year	Type	Country	Gender	Activity	Fatal	Month	Age	Time
0	2018	Unprovoked	USA	F	Paddling	N	6	57	Night

Final cleanups:

The last features to cleanup are the 'Country' and 'Activity' columns. These two columns have many differenct values and a lot of them are only listed one time. Therefore, I will clean these up by removing any values that only appear once.

In [11]:
```
print('Number of unique values in "Activity":\t\t', df.Activity.nunique())
print('How many unique values above 1 are there?:\t', np.sum(df.Activity.value_counts().values > 1))
print('Number of unique values in "Country":\t\t', df.Country.nunique())
print('How many unique values above 1 are there?:\t', np.sum(df.Country.value_counts().values > 1))
```

```
Number of unique values in "Activity":          426
How many unique values above 1 are there?:      80
Number of unique values in "Country":           83
How many unique values above 1 are there?:      47
```

In [12]:
```
# Removing values that only appear once in the 'Activity' and 'Country' column:
df.Activity = df.Activity.where(df.Activity.isin(df.Activity.value_counts().index[:80]), np.nan)
df.Activity = df.Activity.where(df.Activity.isin(df.Activity.value_counts().index[:47]), np.nan)
# removing the observations that only contain 1 unique value for both of those columns:
df.dropna(inplace=True)
```

In [13]:
```
print(len(df), ':\tThe final length of the dataset after all of my cleaning and removing unnecessary columns and
df.head(3)
```

```
1668 :  The final length of the dataset after all of my cleaning and removing unnecessary columns and values
```

Out[13]:

	Year	Type	Country	Gender	Activity	Fatal	Month	Age	Time
1	2018	Unprovoked	USA	F	Standing	N	6	11	Midday
3	2018	Unprovoked	BRAZIL	M	Swimming	Y	6	15	Midday
4	2018	Unprovoked	USA	M	Walking	N	5	12	Midday

Univariate Analysis:

Now it's time to analyze the dataset:

A lot of cleaning was needed to get the dataset to this point. Now it's finally ready to analyze.

How many people have died from shark attacks?

Luckily, it looks like the chance of survival is pretty high.

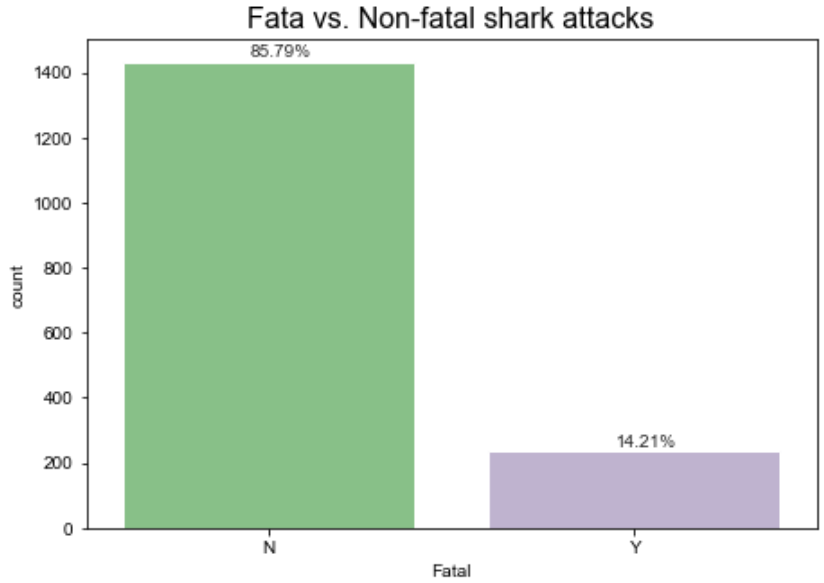
In [14]:
```
# Taking a look at the Fatality column.
print('Was the shark attack fatal?')
print(df.Fatal.value_counts())
print('\nAs a percentage')
print(df.Fatal.value_counts(normalize=True))
```

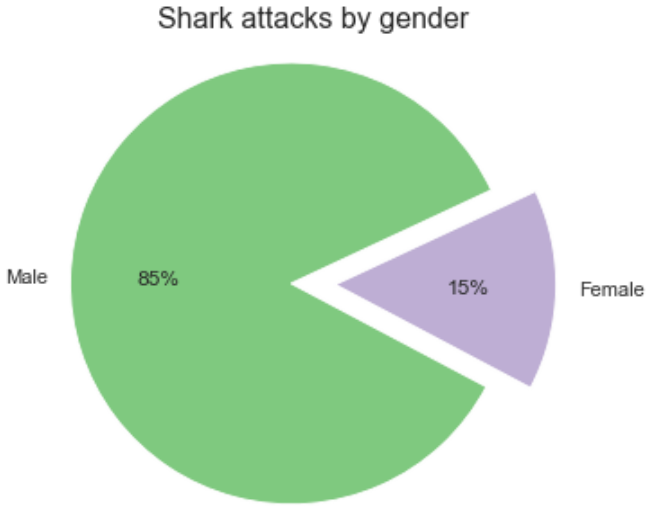
```
Was the shark attack fatal?
N    1431
Y     237
Name: Fatal, dtype: int64

As a percentage
N    0.857914
Y    0.142086
Name: Fatal, dtype: float64
```

In [15]:
```
plt.subplots(figsize=(16,5))
plt.subplot(1,2,1)
sns.set_style('whitegrid')
sns.set_palette('Accent')
sns.countplot(data=df, x='Fatal')
plt.text(x=-0.05, y=1450, s=f'{np.round(df.Fatal.value_counts(normalize=True).values[0]*100, 2)}%')
plt.text(x=0.95, y=250, s=f'{np.round(df.Fatal.value_counts(normalize=True).values[1]*100, 2)}%')
plt.title('Fata vs. Non-fatal shark attacks', fontsize=16)

plt.subplot(1,2,2)
sns.set()
sns.set_style('whitegrid')
plt.pie(df.Gender.value_counts(normalize=True).values, labels=['Male', 'Female'], autopct='%1.0f%%', explode=[0.
   startangle=25)
plt.axis('equal')
plt.title('Shark attacks by gender', fontsize=16)
plt.show()
```

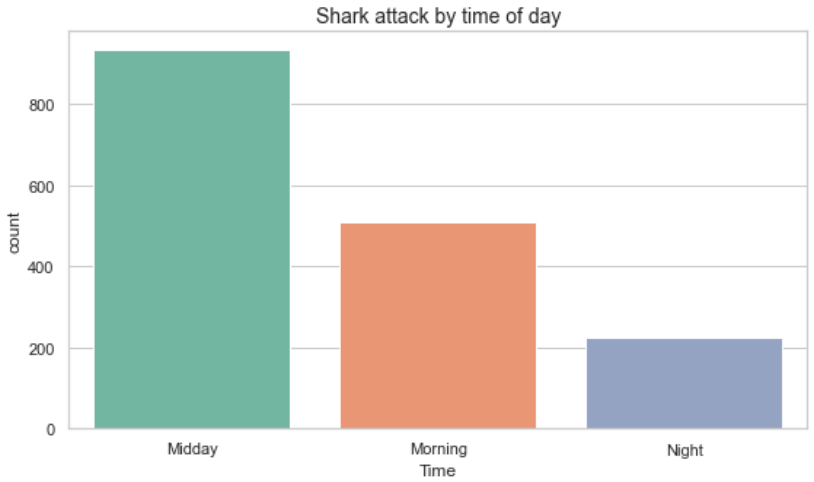


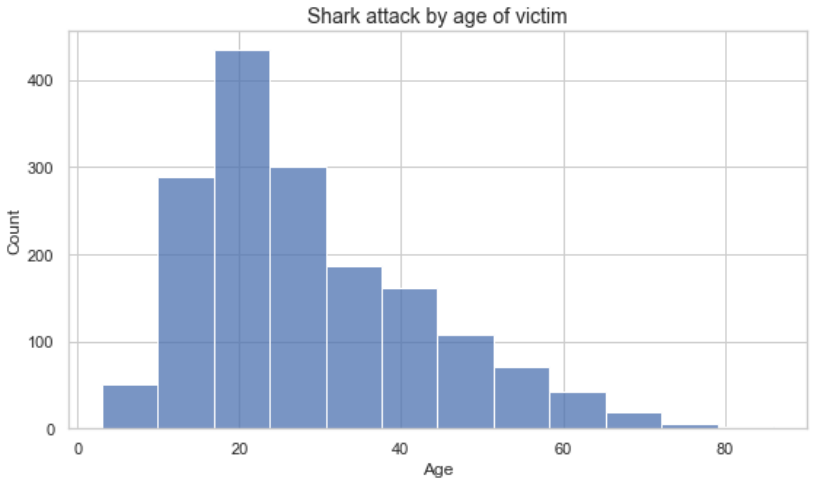


In [16]:
```
plt.subplots(figsize=(20,5))
plt.subplot(1,2,1)
sns.countplot(data=df, x='Time', palette='Set2')
plt.title('Shark attack by time of day', fontsize=14)

plt.subplot(1,2,2)
sns.histplot(data=df, x='Age', bins=12)
plt.title('Shark attack by age of victim', fontsize=14)
```

Out[16]: Text(0.5, 1.0, 'Shark attack by age of victim')





Bivariate Analysis:

In [17]:
```
print('Taking a look at fatality compared to the time of day.')
ct1 = pd.crosstab(df.Fatal, df.Time)
print('Survival rate for each time of day:\n',ct1.values[0] / np.sum(ct1, axis=0))
pd.crosstab(df.Fatal, df.Time)
```

```
Taking a look at fatality compared to the time of day.
Survival rate for each time of day:
 Time
Midday     0.876742
Morning    0.882122
Night      0.725664
dtype: float64
```

Out[17]:

Time	Midday	Morning	Night
Fatal			
N	818	449	164
Y	115	60	62

In [18]:
```
print('Taking a look at fatality compared to gender.')
ct2 = pd.crosstab(df.Fatal, df.Gender)
print('Survival rate for each gender:\n',ct2.values[0] / np.sum(ct2, axis=0))
pd.crosstab(df.Fatal, df.Gender)
```

```
Taking a look at fatality compared to gender.
Survival rate for each gender:
 Gender
F    0.885714
M    0.853127
dtype: float64
```

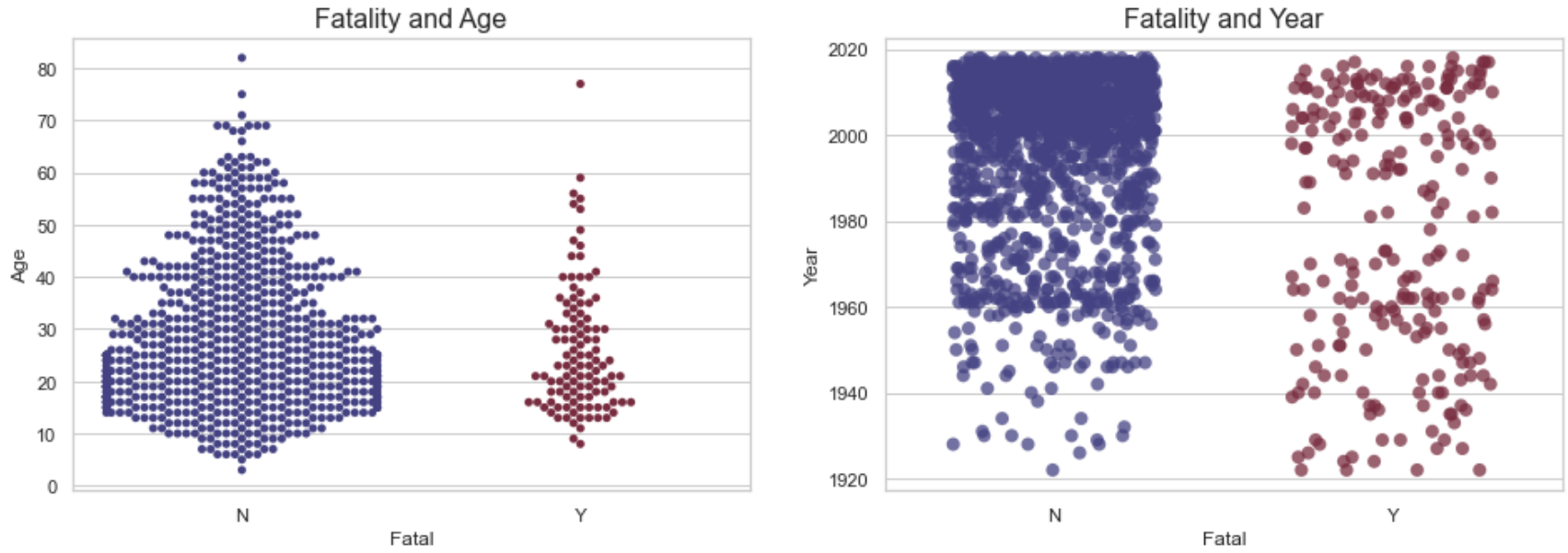
Out[18]:

Gender	F	M
Fatal		
N	217	1214
Y	28	209

In [19]:
```
# Swarm plot gives a warning if it fails to plot some points do to the data set being too big. I want to avoid t
import warnings
warnings.filterwarnings('ignore')
# comparing Fatality to age:
plt.subplots(figsize=(16,5))
plt.subplot(1,2,1)
plt.title('Fatality and Age', fontsize=16)
sns.swarmplot(data=df.sample(frac=.5, random_state=22), x='Fatal', y='Age', palette='icefire')

plt.subplot(1,2,2)
plt.title('Fatality and Year', fontsize=16)
sns.stripplot(data=df, x='Fatal', y='Year', jitter=.3, size=8, alpha=.75, palette='icefire')
```

Out[19]:
```
<AxesSubplot:title={'center':'Fatality and Year'}, xlabel='Fatal', ylabel='Year'>
```



Even with these graphed it;s hard to see if age and survival have anything to do with eachother. The way I will analyze this is by binning the ages and using a contingency table to cross analyze. Then I will do a chi2 contingency test on it.

In [20]:
```
bin_ages = pd.cut(df.Age, bins=[0,11,18,59,100], labels=['Child', 'Teenager', 'Adult', 'Senior'])
pd.crosstab(df.Fatal, bin_ages)
```

Out[20]:

Age	Child	Teenager	Adult	Senior
Fatal				
N	82	311	985	53
Y	7	75	149	6

In [21]:
```
ct3 = pd.crosstab(df.Fatal, bin_ages)
print('Survival rate for each age group:\n',ct3.values[0] / np.sum(ct3, axis=0))
```

```
Survival rate for each age group:
 Age
Child       0.921348
Teenager    0.805699
Adult       0.868607
Senior      0.898305
dtype: float64
```

In [22]:
```
from scipy.stats import chi2_contingency
chi2, pval, dof, expected = chi2_contingency(pd.crosstab(df.Fatal, bin_ages))
print(pval)
```

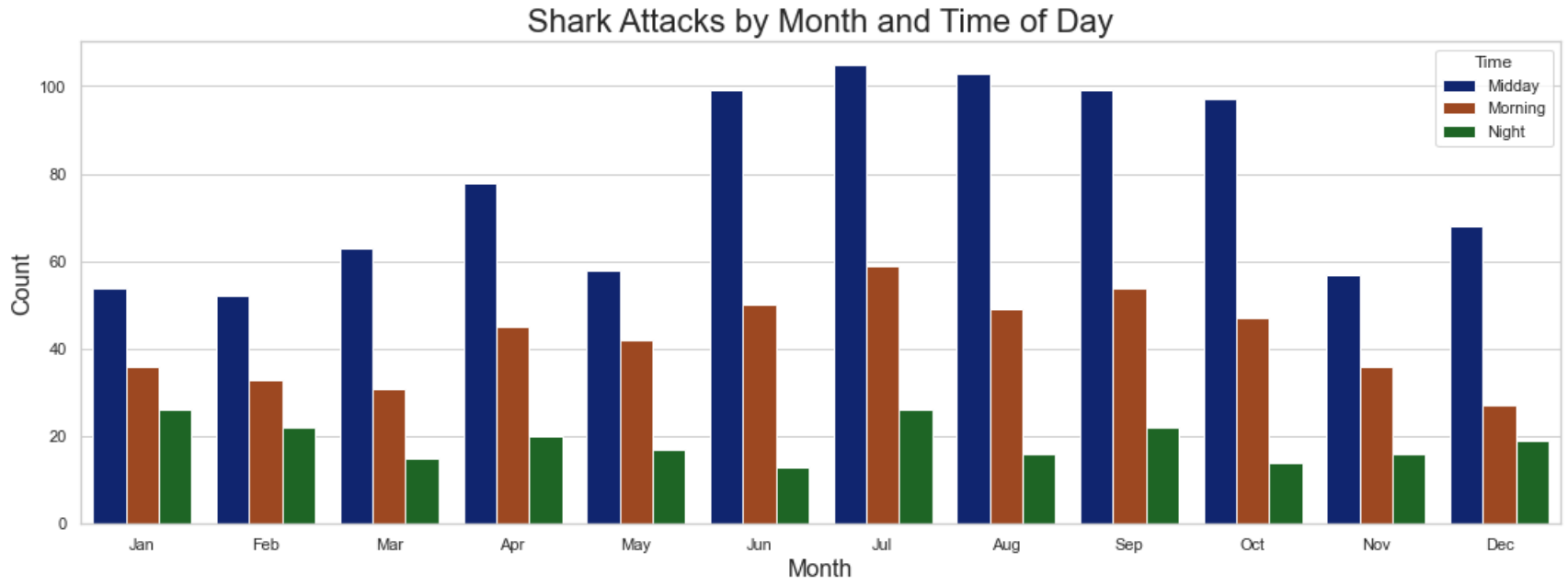
```
0.0038030890816043612
```

Result and thoughts:

The test does show some significance especially for children. However, it's the opposite of what I would have thought. Children have a 92% survival rate. I think this is because children are not going deep into the water and are getting attacked in shallow areas. Therefore they are able to be easily saved. For Teenages and adults they are going into deep areas (surfing, etc.) where the chance of survival is lower. This is only speculation though as I have no data to support the depth of the water where the attacks are taking place.

In [23]:
```
fig, ax = plt.subplots(figsize=(18,6))
sns.countplot(data=df, x='Month', hue='Time', palette='dark')
plt.title('Shark Attacks by Month and Time of Day', fontsize=22)
plt.xlabel('Month', fontsize=16)
plt.ylabel('Count', fontsize=16)
ax.set_xticks(range(12))
ax.set_xticklabels(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

plt.show()
```



Multivariate Analysis:

In [25]:
```
legend_names = ['Standing', 'Swimming', 'Surfing', 'Fishing', 'Wading', 'Spearfishing']
fig, ax = plt.subplots(figsize=(20,8))
df_activity = df[df.Activity.isin(df.Activity.value_counts().index[:6])]
sns.violinplot(data=df_activity, x='Gender', y='Age', hue='Activity', inner=None, palette='Pastel2')
sns.stripplot(data=df_activity, x='Gender', y='Age', hue='Activity', dodge=True, color='black', size=6)
plt.legend(legend_names, ncol=6, loc='upper center', fontsize=20)
plt.title('Age and Gender of victim vs. Activity before being attacked', fontsize=22)
plt.ylabel('Age', fontsize=18)
plt.xlabel('Gender', fontsize=18)
```

Out[25]:
```
Text(0.5, 0, 'Gender')
```

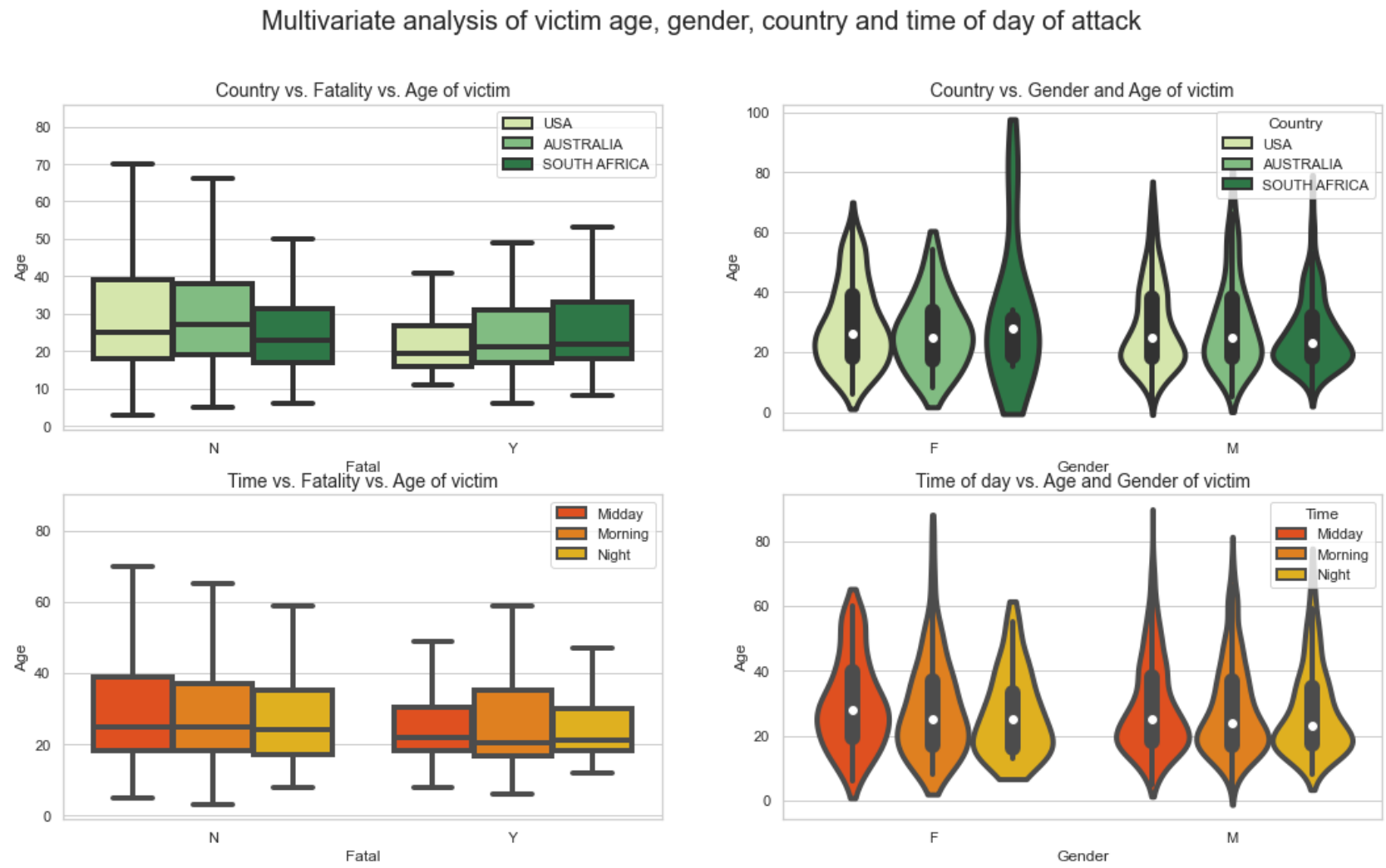
In [26]:
```
df_country = df[df.Country.isin(df.Country.value_counts().index[:3])]
fig, ax = plt.subplots(figsize=(18,10))
plt.subplot(2,2,1)
sns.boxplot(data=df_country, x='Fatal', y='Age', hue='Country', fliersize=False, palette='YlGn', linewidth=4)
plt.title('Country vs. Fatality vs. Age of victim', fontsize=14)
plt.legend(loc='upper right')

plt.subplot(2,2,2)
sns.violinplot(data=df_country, x='Gender', y='Age', hue='Country', palette='YlGn', linewidth=4, cut=1)
plt.title('Country vs. Gender and Age of victim', fontsize=14)

plt.subplot(2,2,3)
sns.boxplot(data=df, x='Fatal', y='Age', hue='Time', fliersize=False, palette='autumn', linewidth=4)
plt.title('Time vs. Fatality vs. Age of victim', fontsize=14)
plt.legend(loc='upper right')

plt.subplot(2,2,4)
sns.violinplot(data=df, x='Gender', y='Age', hue='Time', palette='autumn', linewidth=4, cut=1)
plt.title('Time of day vs. Age and Gender of victim', fontsize=14)
plt.suptitle('Multivariate analysis of victim age, gender, country and time of day of attack', fontsize=20)
```

Out[26]:
```
Text(0.5, 0.98, 'Multivariate analysis of victim age, gender, country and time of day of attack')
```


Multivariate analysis of victim age, gender, country and time of day of attack
Country vs. Fatality vs. Age of victim
USA
AUSTRALIA
SOUTH AFRICA
Age
Fatal
N
Y
Country vs. Gender and Age of victim
Country
USA
AUSTRALIA
SOUTH AFRICA
Age
Gender
F
M
Time vs. Fatality vs. Age of victim
Midday
Morning
Night
Age
Fatal
N
Y
Time of day vs. Age and Gender of victim
Time
Midday
Morning
Night
Age
Gender
F
M

In [ ]:

In [ ]:

In [ ]:

In [ ]: