Exploratory Data Analysis

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
# for Statistics
import scipy.stats
```

In [3]:
```
# load dataSet
dataSet = pd.read_csv('kashti.csv')
dataSet
```

Out[3]:

	Unnamed: 0	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
0	0	0	3	male	22.0	1	0	7.2500	S	Third	man
1	1	1	1	female	38.0	1	0	71.2833	C	First	woman
2	2	1	3	female	26.0	0	0	7.9250	S	Third	woman
3	3	1	1	female	35.0	1	0	53.1000	S	First	woman
4	4	0	3	male	35.0	0	0	8.0500	S	Third	man
...	...	...	...	...	...	...	...	...	...	...	...
886	886	0	2	male	27.0	0	0	13.0000	S	Second	man
887	887	1	1	female	19.0	0	0	30.0000	S	First	woman
888	888	0	3	female	NaN	1	2	23.4500	S	Third	woman
889	889	1	1	male	26.0	0	0	30.0000	C	First	man
890	890	0	3	male	32.0	0	0	7.7500	Q	Third	man

891 rows × 16 columns

1. Data Shapa:

- Mean look to dataSet that how many Columns and rows have in this dataSEt

In [4]:
```
# find shape of dataFram
rows, columns = dataSet.shape
print('The Number of Rows = ', rows)
print('The Number of Columns = ', columns)
```

```
The Number of Rows =  891
The Number of Columns =  16
```

2. Check Data Structure of each Column or Series

In [5]:
```
# find data structure of columns
dataSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Unnamed: 0   891 non-null    int64  
 1   survived     891 non-null    int64  
 2   pclass       891 non-null    int64  
 3   sex          891 non-null    object 
 4   age          714 non-null    float64
 5   sibsp        891 non-null    int64  
 6   parch        891 non-null    int64  
 7   fare         891 non-null    float64
 8   embarked     889 non-null    object 
 9   class        891 non-null    object 
 10  who          891 non-null    object 
 11  adult_male   891 non-null    bool   
 12  deck         203 non-null    object 
 13  embark_town  889 non-null    object 
 14  alive        891 non-null    object 
 15  alone        891 non-null    bool   
dtypes: bool(2), float64(2), int64(5), object(7)
memory usage: 99.3+ KB
```

3. Missing Values in Columns and whole dataFram

In [6]:

```
dataSet.isnull().sum()
```

Out[6]:

```
Unnamed: 0       0
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [7]:

```
dataSet.isnull().sum() / dataSet.shape[0] * 100
```

Out[7]:

```
Unnamed: 0      0.000000
survived        0.000000
pclass          0.000000
sex             0.000000
age            19.865320
sibsp           0.000000
parch           0.000000
fare            0.000000
embarked        0.224467
class           0.000000
who             0.000000
adult_male      0.000000
deck           77.216611
```

```
embark_town     0.224467
alive           0.000000
alone           0.000000
dtype: float64
```

In this example we will not consider the Column 'deck' is the percentage of missing value is quite high (**77.2**)

4. Split Variable or Making new columns if needed

In [8]:

```
# making new dataFram using pandas library
df1 = pd.DataFrame(np.array([['Lahore, Pakistan', 87, 100], ['Beijing, China', 45, 9
columns=['address', 'male', 'female'])
df1.head()
```

Out[8]:

	address	male	female
0	Lahore, Pakistan	87	100
1	Beijing, China	45	96
2	Mosko, Russia	76	200

In [9]:

```
# if we want to separte address into city and country columns we split like this
df1[['city' , 'country']] = df1['address'].str.split(',', expand = True)
# to see the result
df1.head()
```

Out[9]:

	address	male	female	city	country
0	Lahore, Pakistan	87	100	Lahore	Pakistan
1	Beijing, China	45	96	Beijing	China
2	Mosko, Russia	76	200	Mosko	Russia

5. Type Casting:

We can use astype() function from pandas for type casting

In [10]:

```
# to see the types in first place
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   address  3 non-null      object
 1   male     3 non-null      object
 2   female   3 non-null      object
 3   city     3 non-null      object
 4   country  3 non-null      object
dtypes: object(5)
memory usage: 248.0+ bytes
```

In [11]:

```
# convert data type into int
df1[['male', 'female']] = df1[['male', 'female']].astype(int)
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   address  3 non-null      object
 1   male     3 non-null      int32 
 2   female   3 non-null      int32 
 3   city     3 non-null      object
 4   country  3 non-null      object
dtypes: int32(2), object(3)
memory usage: 224.0+ bytes
```

In [12]:

```
# convert data type into string
df1[['male', 'female']] = df1[['male', 'female']].astype('str')
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   address  3 non-null      object
 1   male     3 non-null      object
 2   female   3 non-null      object
 3   city     3 non-null      object
 4   country  3 non-null      object
dtypes: object(5)
memory usage: 248.0+ bytes
```

Step 6. Summary Statistics

- This gives statistics summary of all the data count, mean, std, min, max, percent quantile of the dataFrame for each column as shown below

In [13]:

```
dataSet.describe()
```

Out[13]:

	Unnamed: 0	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	445.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	222.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	445.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	667.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	890.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Step 7. Value count of a specific columns

this will let us know, ka kis column ma kitana values hain

In [14]:
```
df1['male'].value_counts()
# dataSet['age'].value_counts()
```

Out[14]:
```
87    1
45    1
76    1
Name: male, dtype: int64
```

Step 8. Deal with Duplicates

In [15]:
```
# find duplicates
dataSet[dataSet.embark_town == 'Queenstown']

# this will show all the people in embark_town which belongs from Queenstown
```

Out[15]:

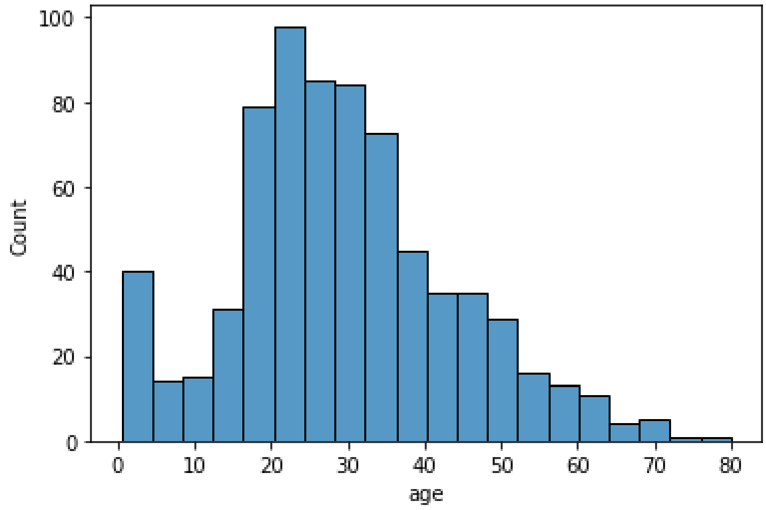
	Unnamed: 0	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	ac
5	5	0	3	male	NaN	0	0	8.4583	Q	Third	man	
16	16	0	3	male	2.0	4	1	29.1250	Q	Third	child	
22	22	1	3	female	15.0	0	0	8.0292	Q	Third	child	
28	28	1	3	female	NaN	0	0	7.8792	Q	Third	woman	
32	32	1	3	female	NaN	0	0	7.7500	Q	Third	woman	
...	...	...	...	...	...	...	...	...	...	...	...	
790	790	0	3	male	NaN	0	0	7.7500	Q	Third	man	
825	825	0	3	male	NaN	0	0	6.9500	Q	Third	man	
828	828	1	3	male	NaN	0	0	7.7500	Q	Third	man	
885	885	0	3	female	39.0	0	5	29.1250	Q	Third	woman	
890	890	0	3	male	32.0	0	0	7.7500	Q	Third	man	

77 rows × 16 columns

Step 9. Check the normal distribution of data (Data Anomally)

In [16]:
```
# plot Histogram
sns.histplot(dataSet['age'])
```

Out[16]: <AxesSubplot:xlabel='age', ylabel='Count'>



If you want to measure Skewness and Kurtosis of the distribution of the data, as shown below:

In [17]:
```
# measure Skewness & Kurtosis
dataSet['age'].agg(['skew', 'kurtosis']).transpose()
```

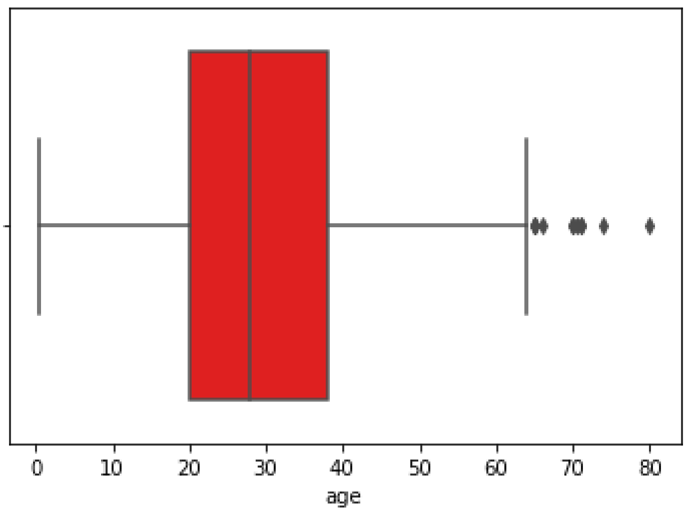
Out[17]:
```
skew        0.389108
kurtosis    0.178274
Name: age, dtype: float64
```

We see here that age distribution was skewed to the right. Now let's check the outlier for the total column with Boxplot

In [18]:
```
sns.boxplot(dataSet['age'], color= 'red')
```

```
C:\Users\abdur\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[18]:
```
<AxesSubplot:xlabel='age'>
```



Step 10. Correlation between two variables (Columns / Series)

In [19]:
```
# draw correlation
cor = dataSet.corr(method='pearson') #You can use spearman if you want
```

```
cor
# this will display a coorelation matrix
```

Out[19]:

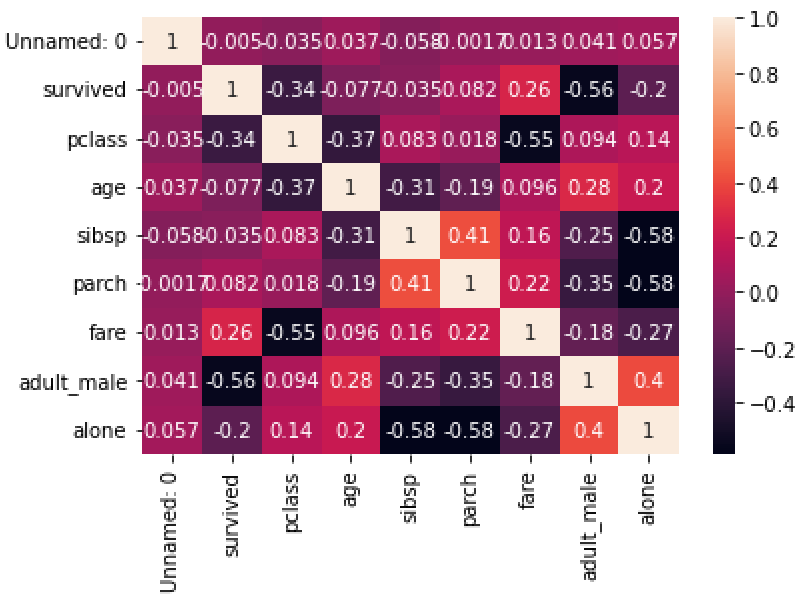
	Unnamed: 0	survived	pclass	age	sibsp	parch	fare	adult_male
Unnamed: 0	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658	0.041010
survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	-0.557080
pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	0.094035
age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	0.280328
sibsp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	-0.253586
parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	-0.349943
fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	-0.182024
adult_male	0.041010	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.182024	1.000000
alone	0.057462	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.271832	0.404744

We can also draw a heatMap of correlation matrix instead of reading number

In [20]:

```
sns.heatmap(cor, annot=True)
# This will show the numbers with colors
```

Out[20]: <AxesSubplot:>

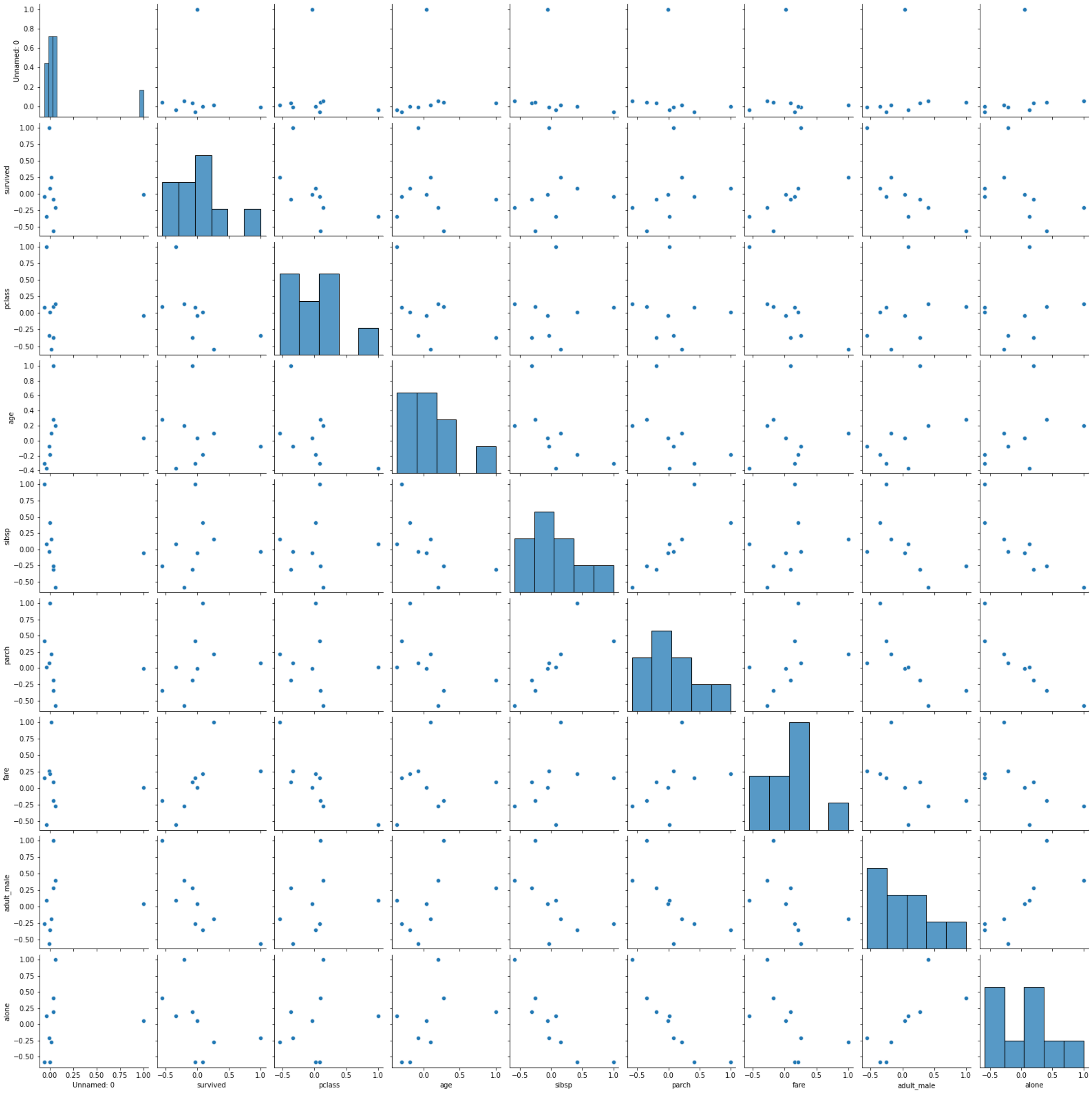


we can also draw a Pair Plot to see the relation

In [22]:

```
sns.pairplot(cor)
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x1c83bdbc7c0>



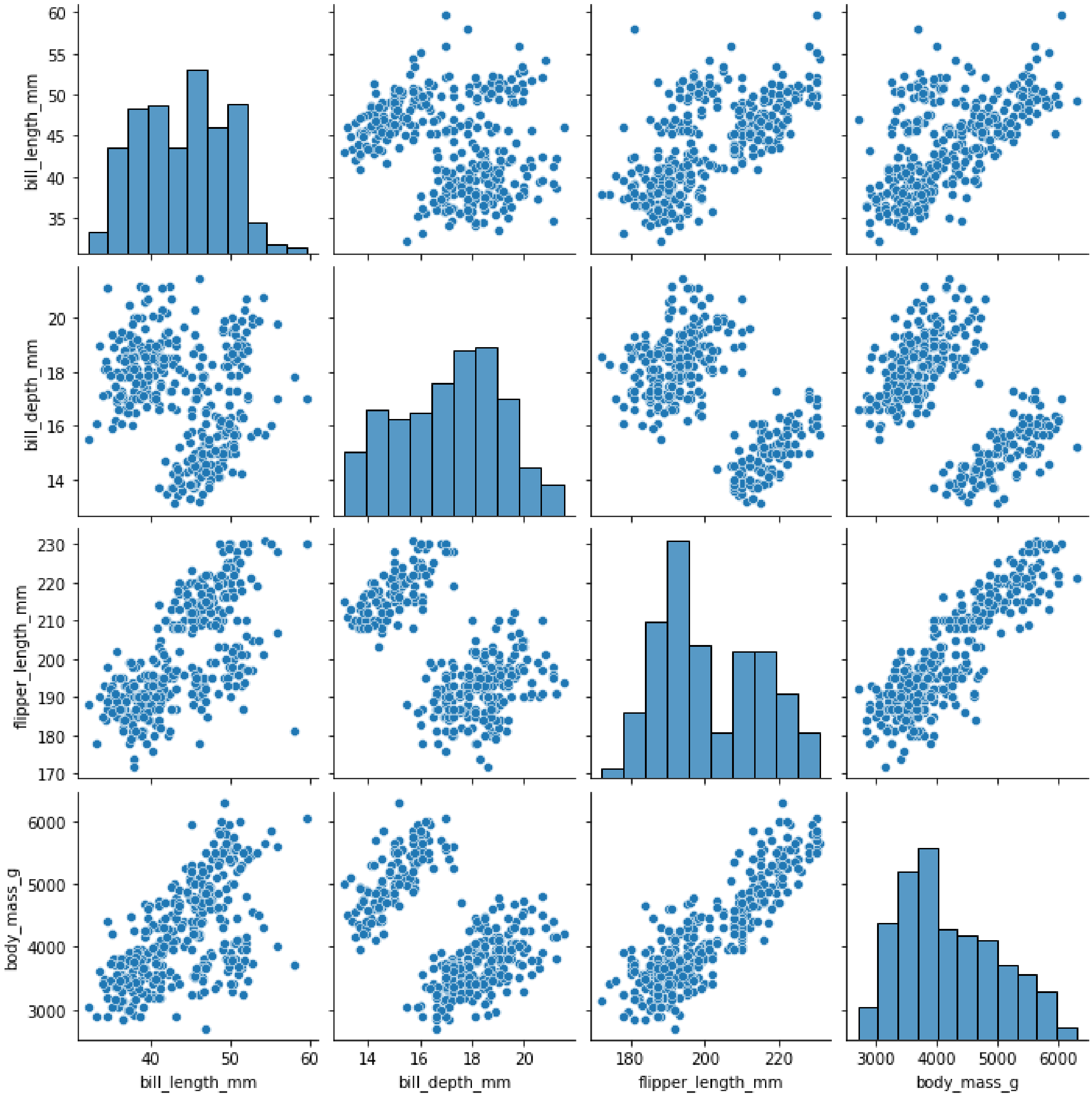
In [26]:
```
# We can change the points based on Category
# import a new dataSet
penguins = sns.load_dataset('penguins')
penguins.head()
```

Out[26]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

In [27]:
```
sns.pairplot(penguins)
```

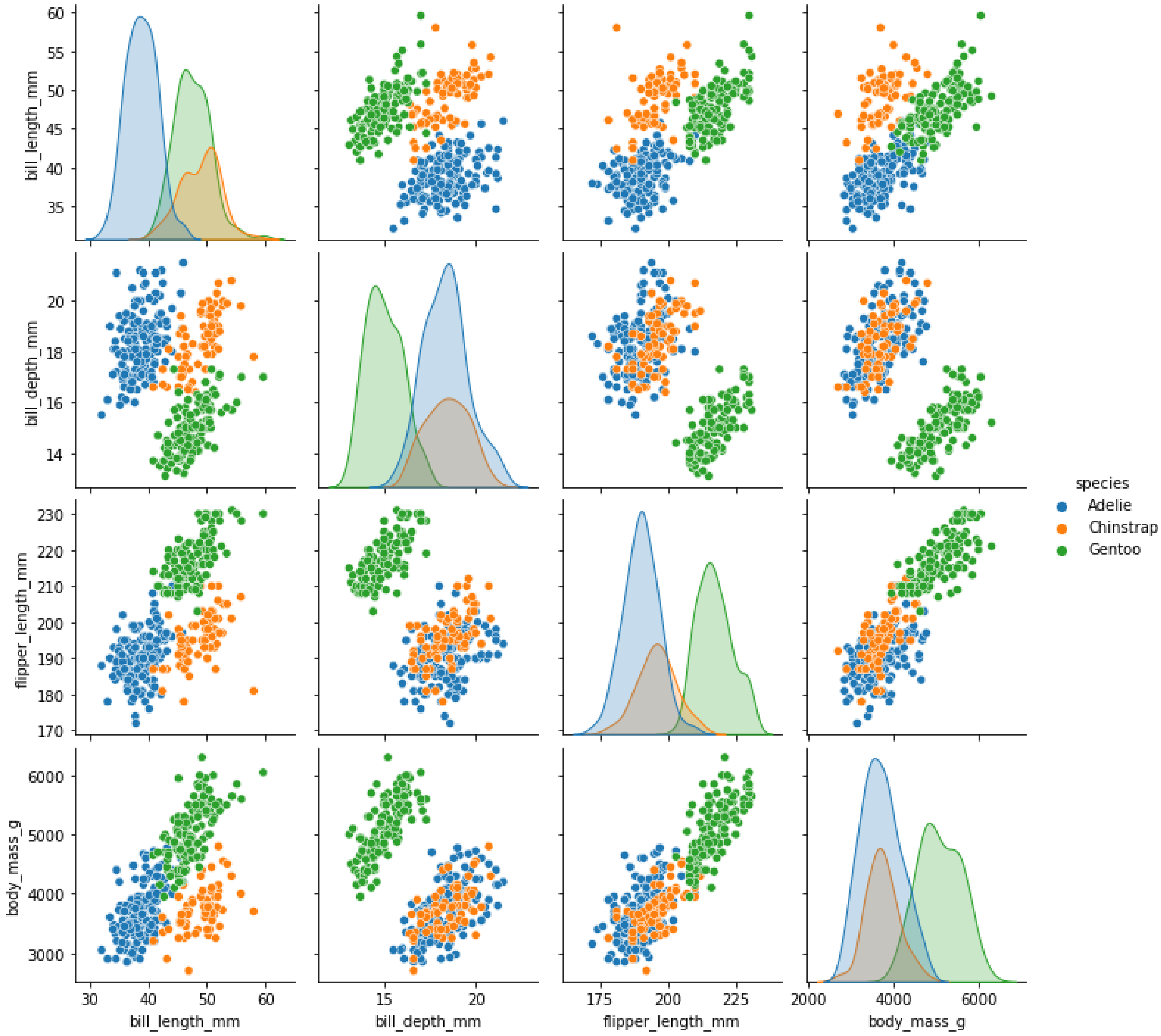
Out[27]: <seaborn.axisgrid.PairGrid at 0x1c8404257c0>



In [28]:

```
sns.pairplot(penguins, hue='species')
```

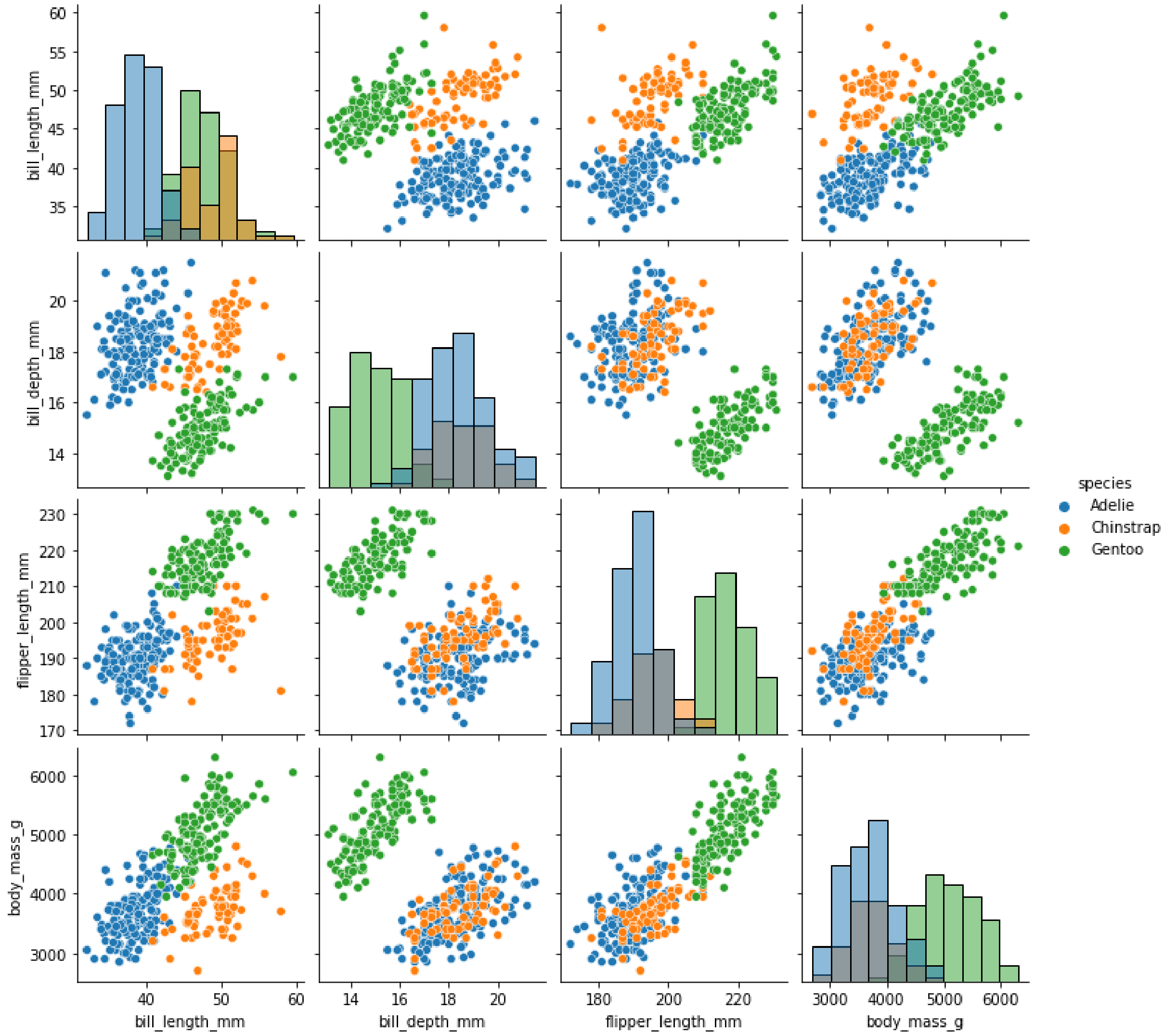
Out[28]: <seaborn.axisgrid.PairGrid at 0x1c841f79a90>



In [29]:

```
# We can convert this into histogram
sns.pairplot(penguins, hue='species', diag_kind='hist')
```

Out[29]: <seaborn.axisgrid.PairGrid at 0x1c842f91490>

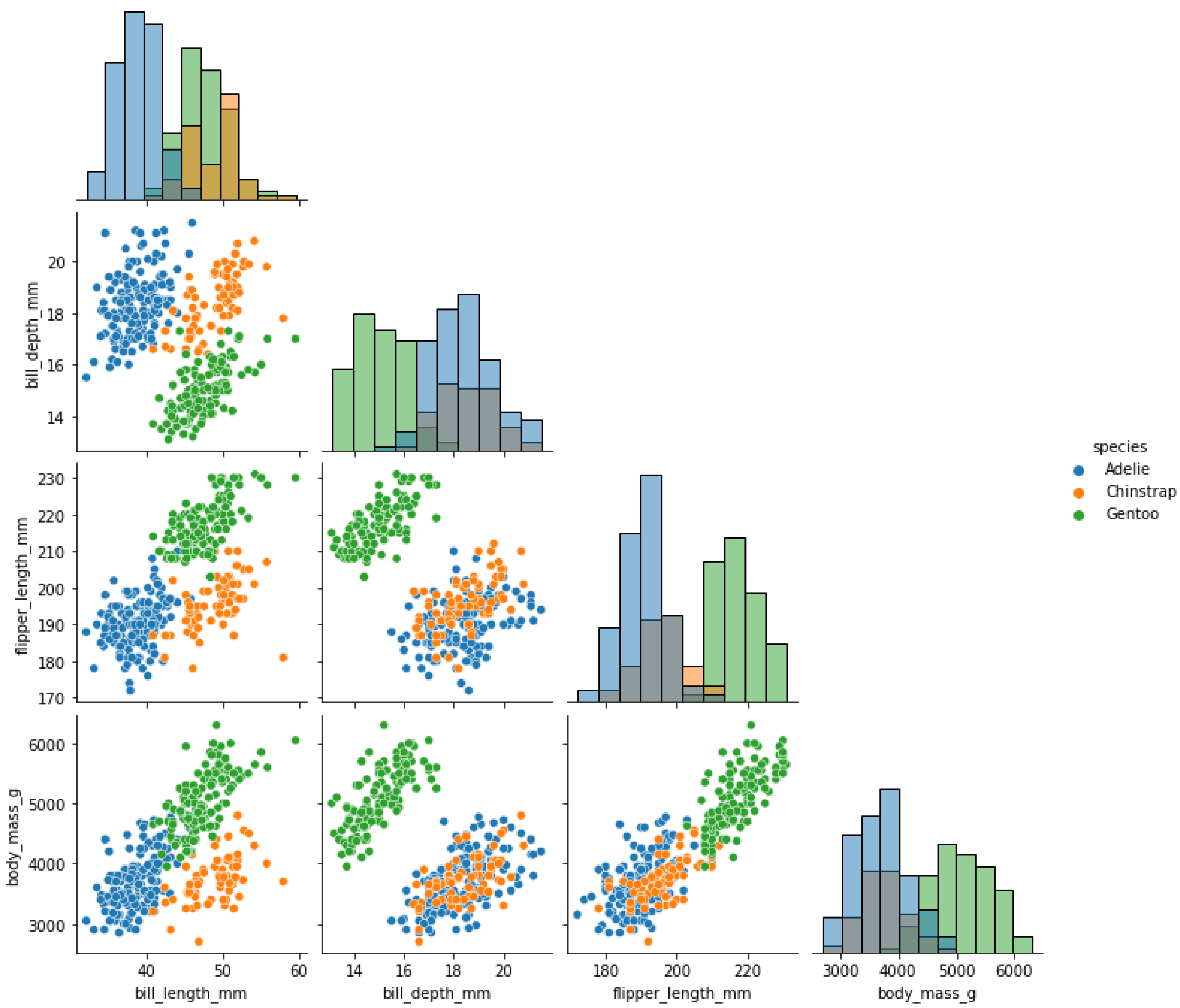


In [30]:

```
# to make one sided
sns.pairplot(penguins , hue = 'species', diag_kind='hist', corner=True)
```

Out[30]:

```
<seaborn.axisgrid.PairGrid at 0x1c844ba41c0>
```



bill_depth_mm
flipper_length_mm
body_mass_g
bill_length_mm
bill_depth_mm
flipper_length_mm
body_mass_g
species
Adelie
Chinstrap
Gentoo

In [ ]: