10-important Steps for Exploratory Data Analyssis

1. Data Shape (columns or Rows ki taddad dekh len)
2. Check Data structure of each column or series
3. Missing values in each column and whole data set
4. Split variables or make new columns if needed
5. Type casting
6. Summary statistics
7. Value counts of a specific column
8. Deal with duplicates
9. Check the normal distribution of data (data Anomally)
10. Correlation between two variables (columns/series)

EDA karne se hamen kia kia maloom hta hy?

1. Data invetigation
2. Patterns inside the data
3. Anomalies (normal disribution hy ya skewed)
4. hyothsis konsa or kaisay design karna
5. Assumption konsi hni chahyeayn
6. Data visualization (Sirf pattern dekhnay k liay)

In [ ]:

```
#import libararies
import numpy as np
import pandas as pd
import seaborn as sns
import scipy as sc
import matplotlib.pyplot as plt
```

In [ ]:

```
#load datasets
df= sns.load_dataset('tips')
df1= sns.load_dataset('titanic')
```

1.find shape

In [ ]:

```
# shape

rows, cols = df.shape
print("Number of Rows       = ", rows) #instances
print("Number of Columns    = ", cols) #series
```

```
Number of Rows        =  244
Number of Columns     =  7
```

In [ ]:

```
# shape

rows, cols = df1.shape
```

```python
print("Number of Rows       = ", rows) #instances
print("Number of Columns    = ", cols) #series
```

```
Number of Rows       =  891
Number of Columns    =  15
```

2. Check Data structure of each column or series

In [ ]:

```python
# info of data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

In [ ]:

```python
# data info
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

3. Missing values in each column and whole data set

In [ ]:

```python
# how many missing values present
df.isnull().sum()
```

Out[ ]:

```
total_bill    0
tip           0
sex           0
smoker        0
```

```
day           0
time          0
size          0
dtype: int64
```

In [ ]:

```
# how many missing values present
df1.isnull().sum()
```

Out[ ]:

```
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
```

We will calculated the percentage of missing values, and if the percentage of missing value is high then we will reduce the priority of that column. We can dealwith the missing values if the percentage is low by replacing them with the means, median or other methods (removal).

In [ ]:

```
# percentage of missing values
df.isnull().sum() / df.shape[0] *100
```

Out[ ]:

```
total_bill    0.0
tip           0.0
sex           0.0
smoker        0.0
day           0.0
time          0.0
size          0.0
dtype: float64
```

In [ ]:

```
# percentage of missing values
df1.isnull().sum() / df.shape[0] *100
```

Out[ ]:

```
survived         0.000000
pclass           0.000000
sex              0.000000
age             72.540984
sibsp            0.000000
parch            0.000000
fare             0.000000
embarked         0.819672
class            0.000000
who              0.000000
adult_male       0.000000
deck           281.967213
embark_town      0.819672
alive            0.000000
alone            0.000000
dtype: float64
```

In this example we will not consider the column name deck as the percentage of missing value is quite high (77.22%).

4. Split variables or make new columns if needed

In [ ]:
```
# making a dataframe for example using pandas library

df2 = pd.DataFrame(np.array([["Lahore, Pakistan",67, 100], ["Beijing, China", 5, 6],
                    columns=['address', 'males', 'females'])
df2.head()
```

Out[ ]:

	address	males	females
0	Lahore, Pakistan	67	100
1	Beijing, China	5	6
2	berlin, Germany	8	9

In [ ]:
```
# if we want to separate address into city and country columns we will split like th
df2[['city', 'country']] = df2['address'].str.split(',', expand=True)
#to see the results
df2.head()
```

Out[ ]:

	address	males	females	city	country
0	Lahore, Pakistan	67	100	Lahore	Pakistan
1	Beijing, China	5	6	Beijing	China
2	berlin, Germany	8	9	berlin	Germany

5. Type casting

In [ ]:
```
# how to see the types in first place
df2.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   males    3 non-null      object
 2   females  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 [ ]:
```
# convert data type into integer
df2[['males', 'females']] = df2[['males', 'females']].astype('int')
#convert to string
df2[["city", "country"]] = df2[["city", "country"]].astype('str').astype("string")
```

In [ ]:
```
df2.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   males    3 non-null      int32 
 2   females  3 non-null      int32 
 3   city     3 non-null      string
 4   country  3 non-null      string
dtypes: int32(2), object(1), string(2)
memory usage: 224.0+ bytes
```

In [ ]:

```
# #Replace Data Types to Boolean
# df["IsPurchased"] = df['IsPurchased'].astype('bool')
# #Replace Data Types to Float
# df["Total Spend"] = df['Total Spend'].astype('float')
# #Replace Data Types to Datetime with format= '%Y%m%d'
# df['Dates'] = pd.to_datetime(df['Dates'], format='%Y%m%d')
```

6. Summary statistics

In [ ]:

```
df.describe()
```

Out[ ]:

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

In [ ]:

```
df1.describe()
```

Out[ ]:

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

In [ ]:

```
df2.describe()
```

Out[ ]:

	males	females
count	3.000000	3.000000
mean	26.666667	38.333333
std	34.961884	53.425961
min	5.000000	6.000000
25%	6.500000	7.500000
50%	8.000000	9.000000
75%	37.500000	54.500000
max	67.000000	100.000000

7. Value counts of a specific column

In [ ]:

```
#how much values in a specific column
df1['age'].value_counts()
```

Out[ ]:

```
24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
36.50     1
55.50     1
0.92      1
23.50     1
74.00     1
Name: age, Length: 88, dtype: int64
```

In [ ]:

```
df2['females'].value_counts()
```

Out[ ]:

```
100    1
6      1
9      1
Name: females, dtype: int64
```

In [ ]:

```
df['tip'].value_counts()
```

Out[ ]:

```
2.00    33
3.00    23
4.00    12
5.00    10
2.50    10
        ..
4.34     1
1.56     1
5.20     1
2.60     1
1.75     1
Name: tip, Length: 123, dtype: int64
```

In [ ]:

```
#finding unique values in a column or series
df1['age'].unique()
```

Out[ ]:
```
array([22.  , 38.  , 26.  , 35.  ,   nan, 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  , 28.  ,
        8.  , 19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  ,
       49.  , 29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  ,
       16.  , 25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  ,
       71.  , 37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 ,
       51.  , 55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  ,
       45.5 , 20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  ,
       60.  , 10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  ,
       70.  , 24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  ])
```

In [ ]:
```
df2['females'].unique()
```

Out[ ]:
```
array([100,   6,   9])
```

8. Deal with duplicates

In [ ]:
```
# find duplicates (remove)/ null values ( mean median mode)
df1[df1.embark_town == 'Queenstown']

#this will show the people only embarked from Queenstown in Titanic.
```

Out[ ]:

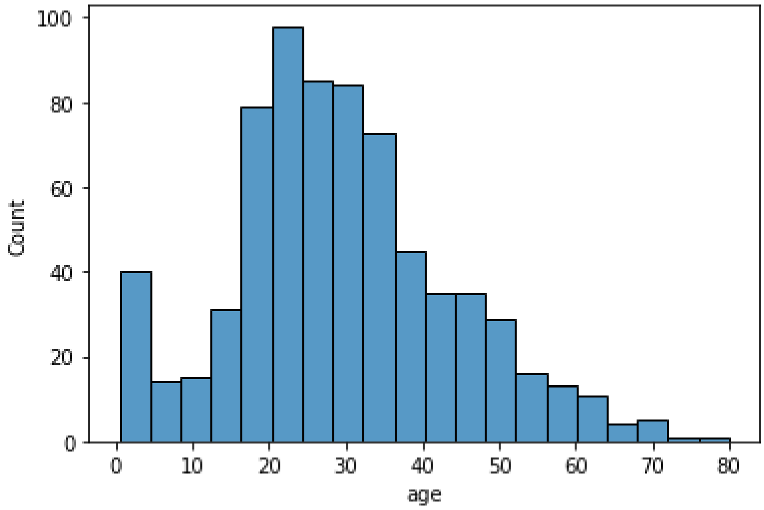
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	d
5	0	3	male	NaN	0	0	8.4583	Q	Third	man	True	N
16	0	3	male	2.0	4	1	29.1250	Q	Third	child	False	N
22	1	3	female	15.0	0	0	8.0292	Q	Third	child	False	N
28	1	3	female	NaN	0	0	7.8792	Q	Third	woman	False	N
32	1	3	female	NaN	0	0	7.7500	Q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

77 rows × 15 columns

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

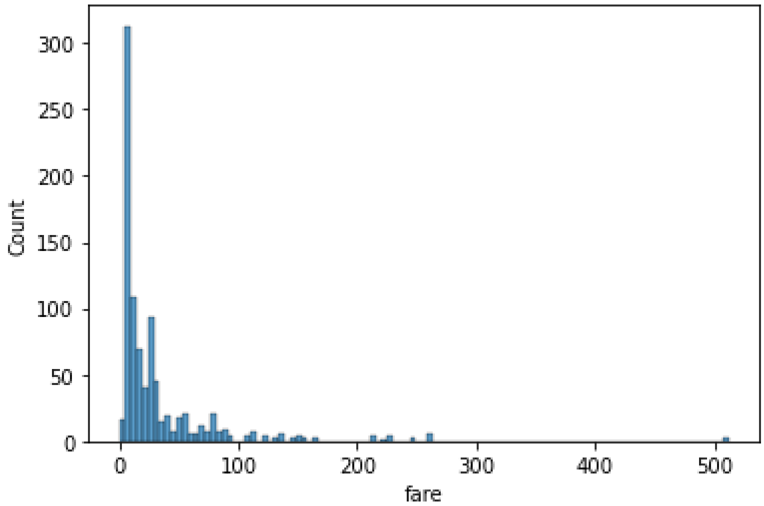
In [ ]:
```
# plot histogram
sns.histplot(df1['age'])
```

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



In [ ]:
```
sns.histplot(df1['fare'])
```

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



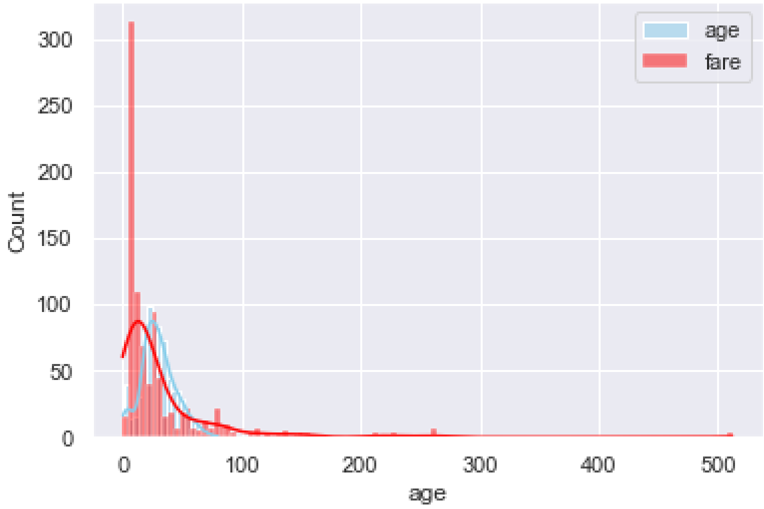
This one is right skewed and dosen't normal at all

In [ ]:
```
# Plotting 2 variables on the same graph

# set a grey background
sns.set(style="darkgrid")

sns.histplot(data=df1, x="age", color="skyblue", label="age", kde=True)
sns.histplot(data=df1, x="fare", color="red", label="fare", kde=True)

plt.legend()
plt.show()
```



The gragh is looking messy because fare is not normally distributed and have outliers Thats why it is better to look at them individually first.

In [ ]:
```
df.head(2)
```

Out[ ]:

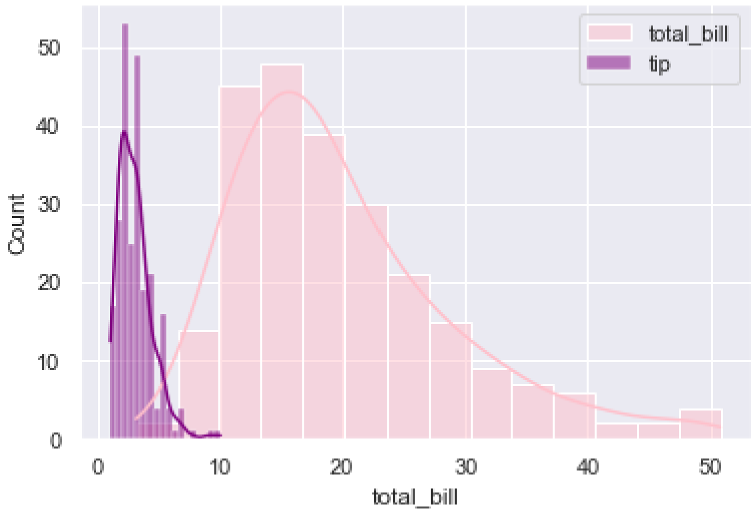
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

In [ ]:
```
# Plotting 2 variables on the same graph

# set a grey background
sns.set(style="darkgrid")

sns.histplot(data=df, x="total_bill", color="pink", label="total_bill", kde=True)
sns.histplot(data=df, x="tip", color="purple", label="tip", kde=True)

plt.legend()
plt.show()
```



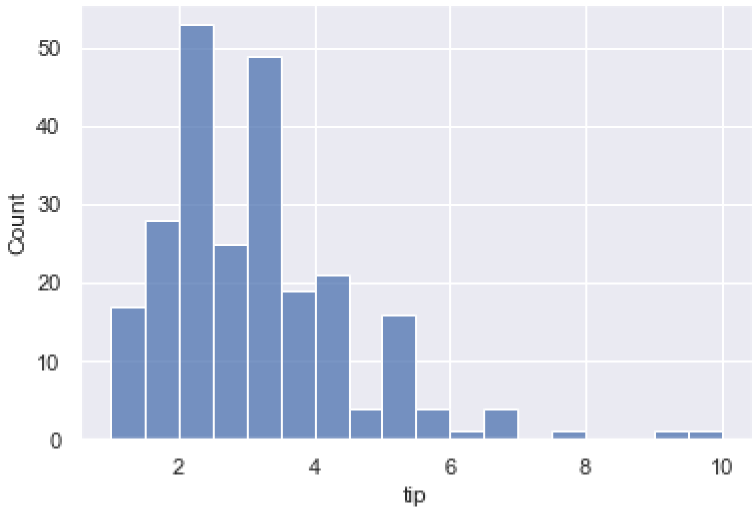
In [ ]:
```
#measure its skewness and kurtosis
df1['age'].agg(['skew', 'kurtosis']).transpose()
```

```
skew        0.389108
```

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

In [ ]:
```
sns.histplot(df['tip'])
```

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



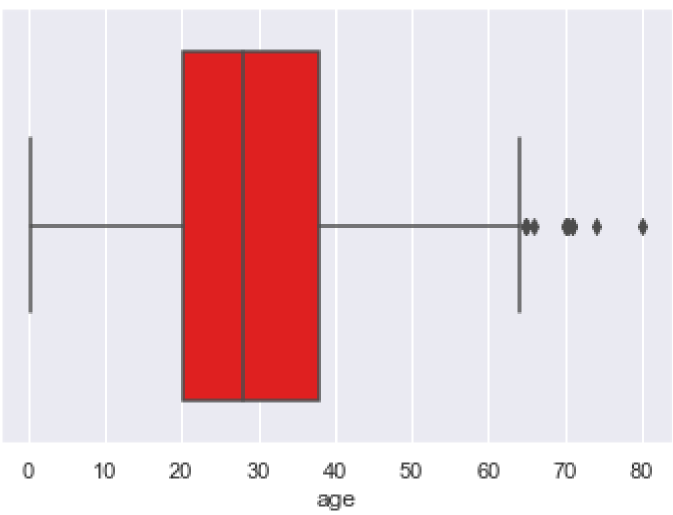
In [ ]:
```
#measure its skewness and kurtosis
df['tip'].agg(['skew', 'kurtosis']).transpose()
```

Out[ ]:
```
skew        1.465451
kurtosis    3.648376
Name: tip, dtype: float64
```

In [ ]:
```
sns.boxplot(df1['age'], color="red")
```

```
C:\Users\Azka\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 p
ositional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

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



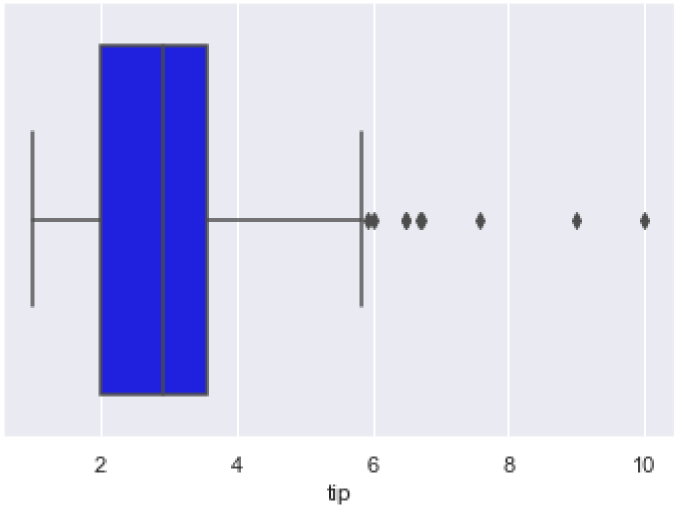
In [ ]:
```
sns.boxplot(df['tip'], color="blue")
```

```
C:\Users\Azka\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 p
ositional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[ ]:

```
<AxesSubplot:xlabel='tip'>
```



10. Correlation between two variables (columns/series)

Checking the correlation between variables is also necessary to see potential a feature that we can use for further analysis or building a model later. We can use a correlation matrix to get this.

In [ ]:

```
# drawing correlation
corr = df.corr(method="pearson") # you can use spearman if you want
corr
# this will display a correlation matrix
```

Out[ ]:

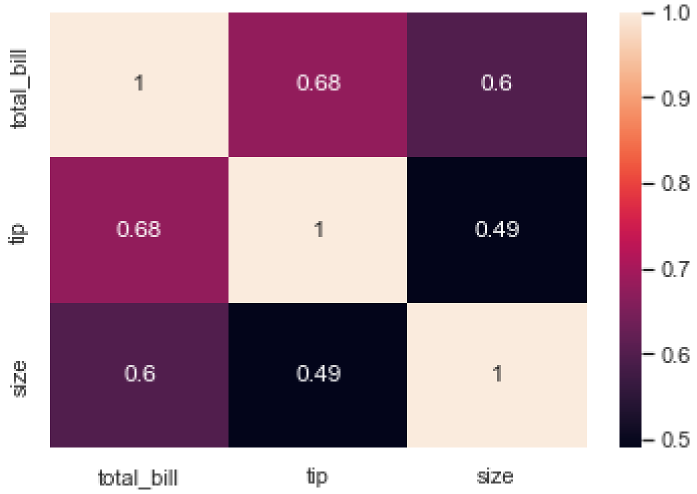
	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

In [ ]:

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

Out[ ]:

```
<AxesSubplot:>
```



In [ ]:
```
corr.style.background_gradient(cmap='gist_stern')
```

Out[ ]:

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

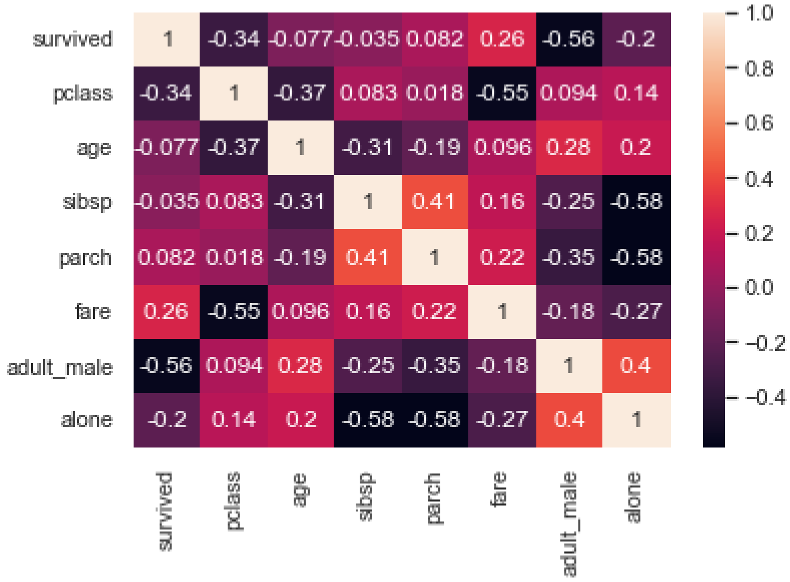
In [ ]:
```
# drawing correlation
corr1 = df1.corr(method="pearson") # you can use spearman if you want
corr1
# this will display a correlation matrix
```

Out[ ]:

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	-0.557080	-0.203367
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	0.094035	0.135207
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	0.280328	0.198270
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	-0.253586	-0.584471
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	-0.349943	-0.583398
fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	-0.182024	-0.271832
adult_male	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.182024	1.000000	0.404744
alone	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.271832	0.404744	1.000000

In [ ]:
```
sns.heatmap(corr1, annot=True)
# this will show the numbers with colors
```

Out[ ]: <AxesSubplot:>



In [ ]:

```
corr1.style.background_gradient(cmap='coolwarm')
```

Out[ ]:

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	-0.557080	-0.203367
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	0.094035	0.135207
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	0.280328	0.198270
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	-0.253586	-0.584471
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	-0.349943	-0.583398
fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	-0.182024	-0.271832
adult_male	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.182024	1.000000	0.404744
alone	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.271832	0.404744	1.000000

In [ ]:

```
#correlation between male vs female
df1male =df1[df1['sex']=='male']
df1female =df1[df1['sex']=='female']
```

In [ ]:

```
# drawing correlation
corr1ma = df1male.corr(method="pearson")
corr1ma # this will display a correlation matrix
corr1ma.style.background_gradient(cmap='coolwarm')
```

Out[ ]:

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.220618	-0.119618	-0.020238	0.096318	0.171288	-0.234337	-0.133419
pclass	-0.220618	1.000000	-0.392754	0.076957	-0.031481	-0.472452	-0.078919	0.135319
age	-0.119618	-0.392754	1.000000	-0.334982	-0.232419	0.077331	0.536159	0.211858
sibsp	-0.020238	0.076957	-0.334982	1.000000	0.524849	0.181804	-0.461831	-0.637488
parch	0.096318	-0.031481	-0.232419	0.524849	1.000000	0.312197	-0.497120	-0.606243
fare	0.171288	-0.472452	0.077331	0.181804	0.312197	1.000000	-0.056082	-0.321650
adult_male	-0.234337	-0.078919	0.536159	-0.461831	-0.497120	-0.056082	1.000000	0.414375

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
alone	-0.133419	0.135319	0.211858	-0.637488	-0.606243	-0.321650	0.414375	1.000000

In [ ]:

```
# drawing correlation
corr1fe = df1male.corr(method="pearson")
corr1fe # this will display a correlation matrix
corr1fe.style.background_gradient(cmap='coolwarm')
```

Out[ ]:

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.220618	-0.119618	-0.020238	0.096318	0.171288	-0.234337	-0.133419
pclass	-0.220618	1.000000	-0.392754	0.076957	-0.031481	-0.472452	-0.078919	0.135319
age	-0.119618	-0.392754	1.000000	-0.334982	-0.232419	0.077331	0.536159	0.211858
sibsp	-0.020238	0.076957	-0.334982	1.000000	0.524849	0.181804	-0.461831	-0.637488
parch	0.096318	-0.031481	-0.232419	0.524849	1.000000	0.312197	-0.497120	-0.606243
fare	0.171288	-0.472452	0.077331	0.181804	0.312197	1.000000	-0.056082	-0.321650
adult_male	-0.234337	-0.078919	0.536159	-0.461831	-0.497120	-0.056082	1.000000	0.414375
alone	-0.133419	0.135319	0.211858	-0.637488	-0.606243	-0.321650	0.414375	1.000000

In [ ]:

```
#calculate the correlation between the two arrays male vs female
np.corrcoef(corr1ma, corr1fe)[0,1]
```

Out[ ]: -0.37296113527583546

In [ ]:

```
# drawing correlation
corr2 = df2.corr(method="pearson") # you can use spearman if you want
corr2
# this will display a correlation matrix
```

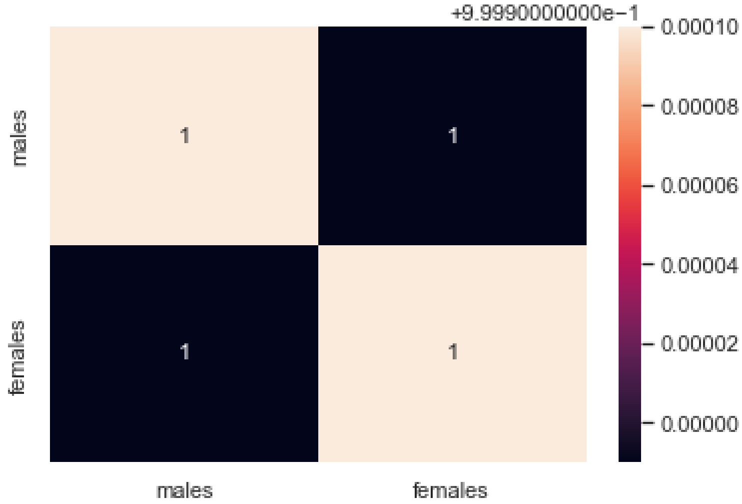
Out[ ]:

	males	females
males	1.00000	0.99989
females	0.99989	1.00000

In [ ]:

```
sns.heatmap(corr2, annot=True)
# this will show the numbers with colors
```

Out[ ]: <AxesSubplot:>



In [ ]:

```
corr2.style.background_gradient(cmap='YlOrBr_r')
```

Out[ ]:

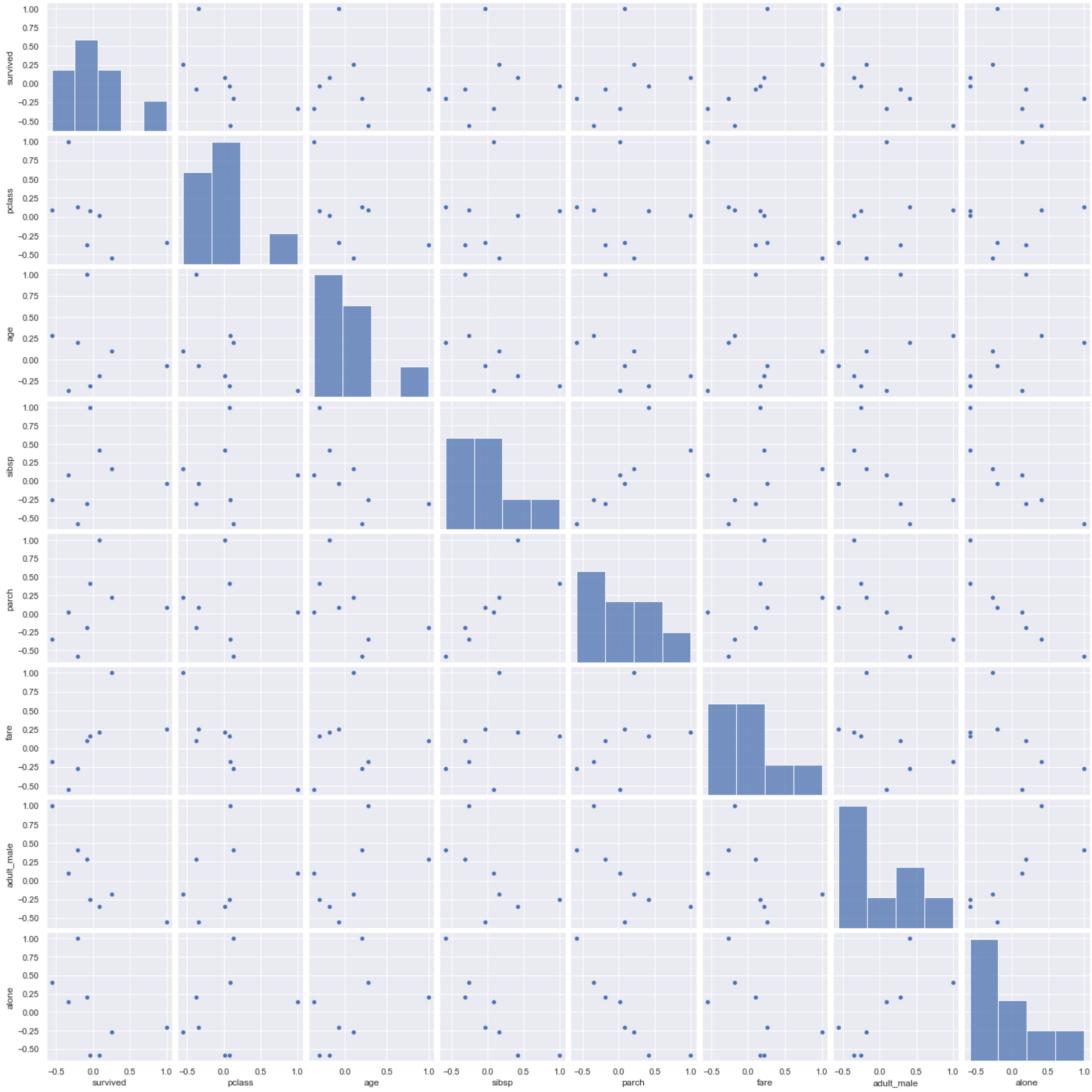
	males	females
males	1.000000	0.999890
females	0.999890	1.000000

In [ ]:

```
# we can also draw a pairplot to see the correlation
sns.pairplot(corr1)
```

Out[ ]:

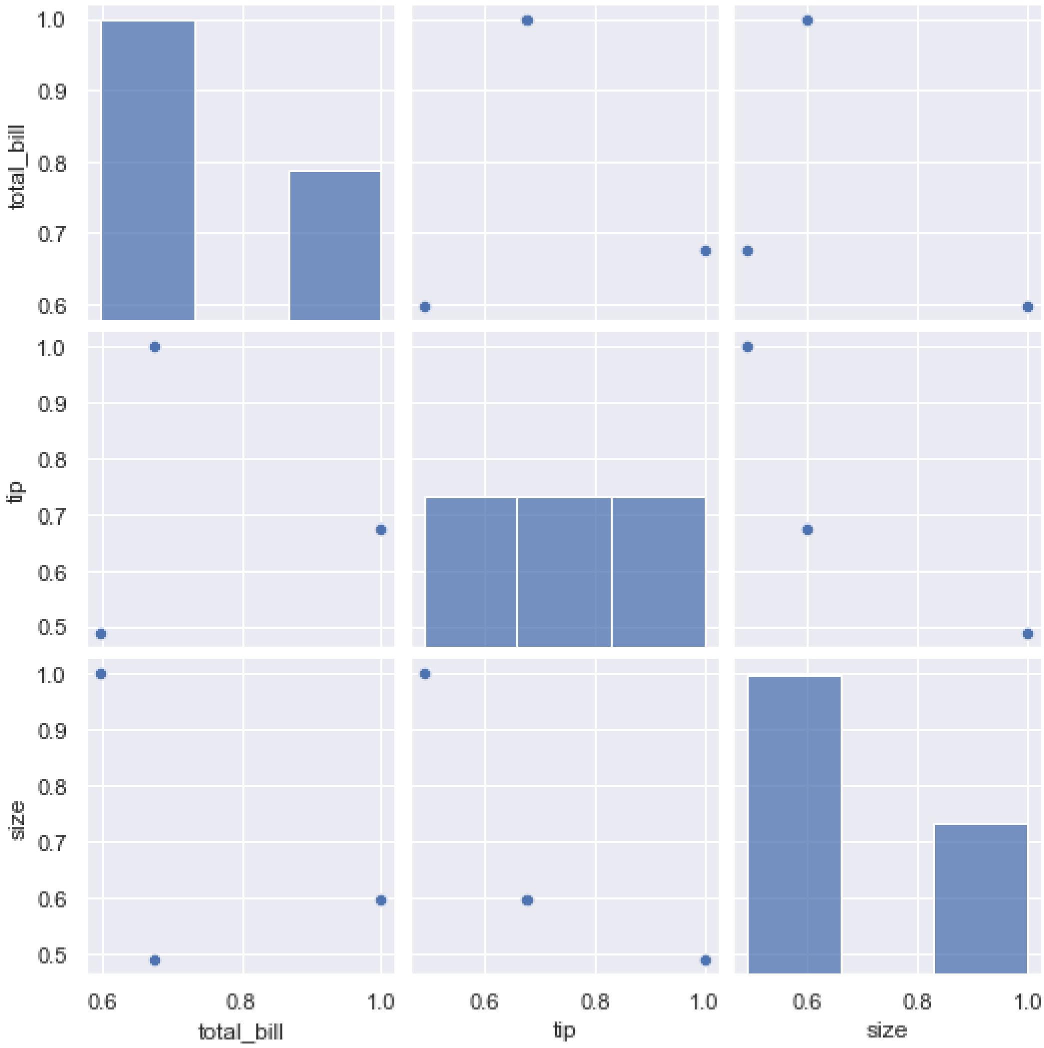
```
<seaborn.axisgrid.PairGrid at 0x1edc6ab5f40>
```



In [ ]:

```
# we can also draw a pairplot to see the correlation
sns.pairplot(corr)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x1edc9f473d0>



In [ ]:

```
df.head()
```

Out[ ]:

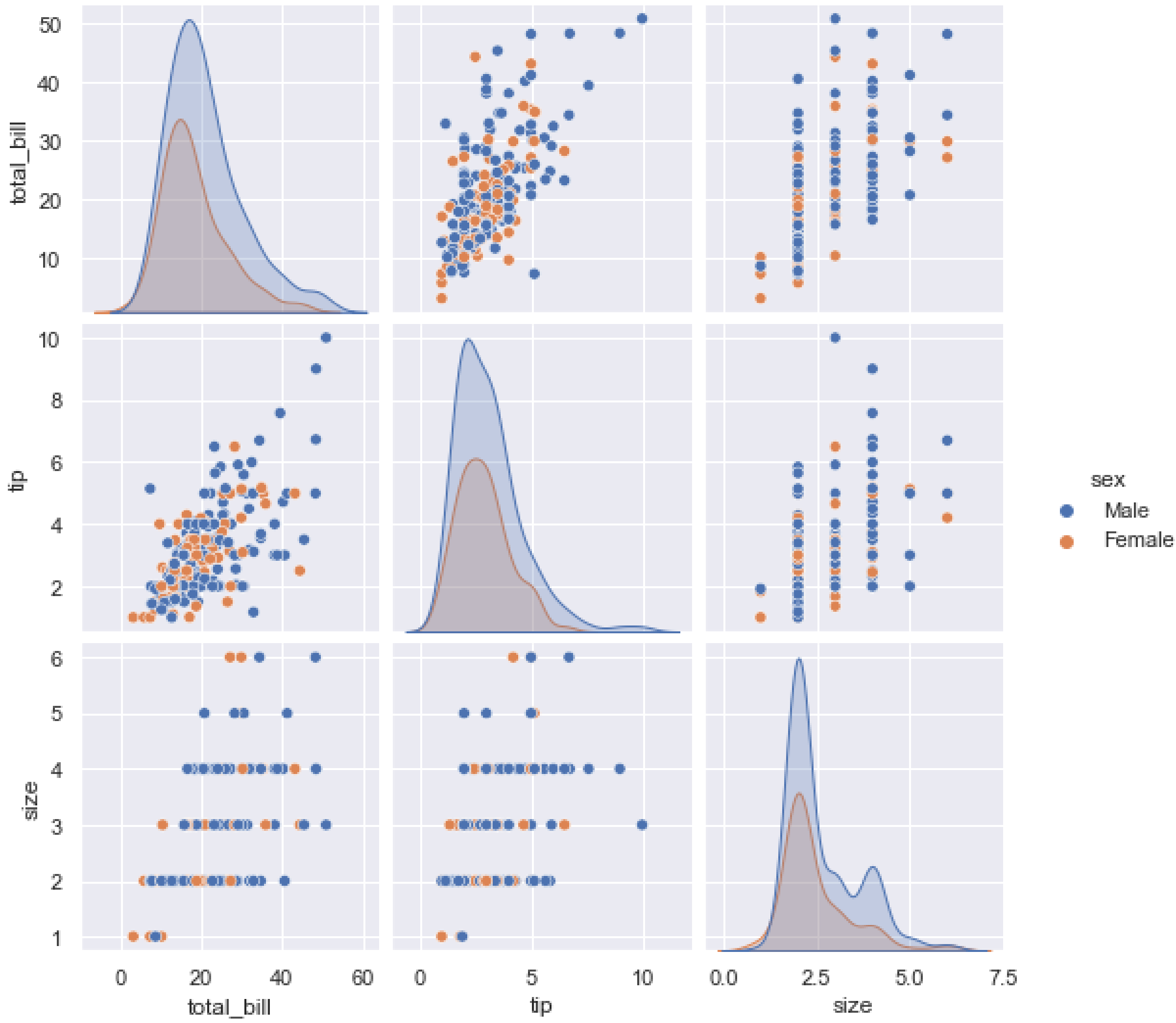
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [ ]:

```
# we can change the points based on category
sns.pairplot(df, hue="sex")
```

Out[ ]:

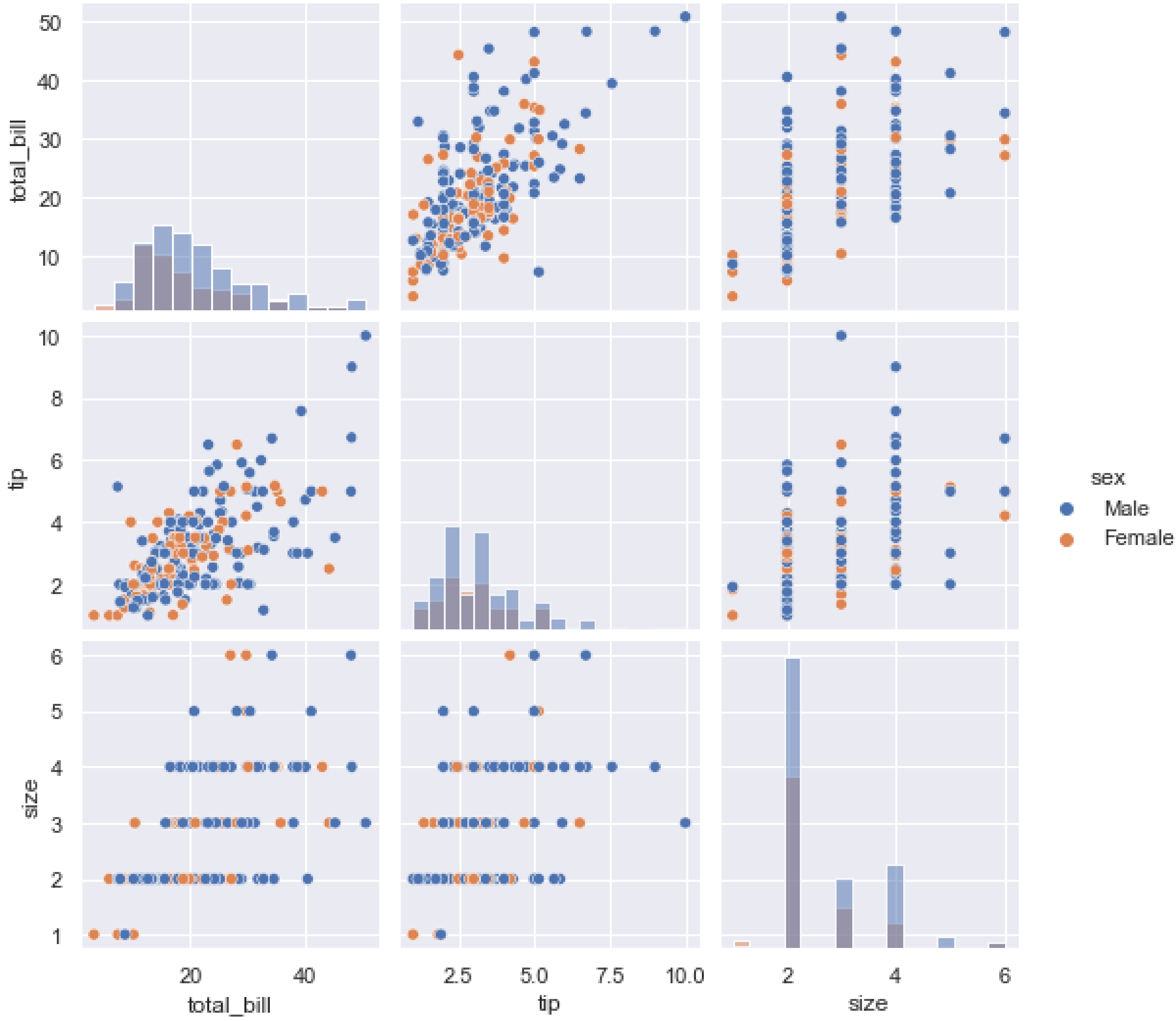
```
<seaborn.axisgrid.PairGrid at 0x1edc9f47c40>
```



In [ ]:

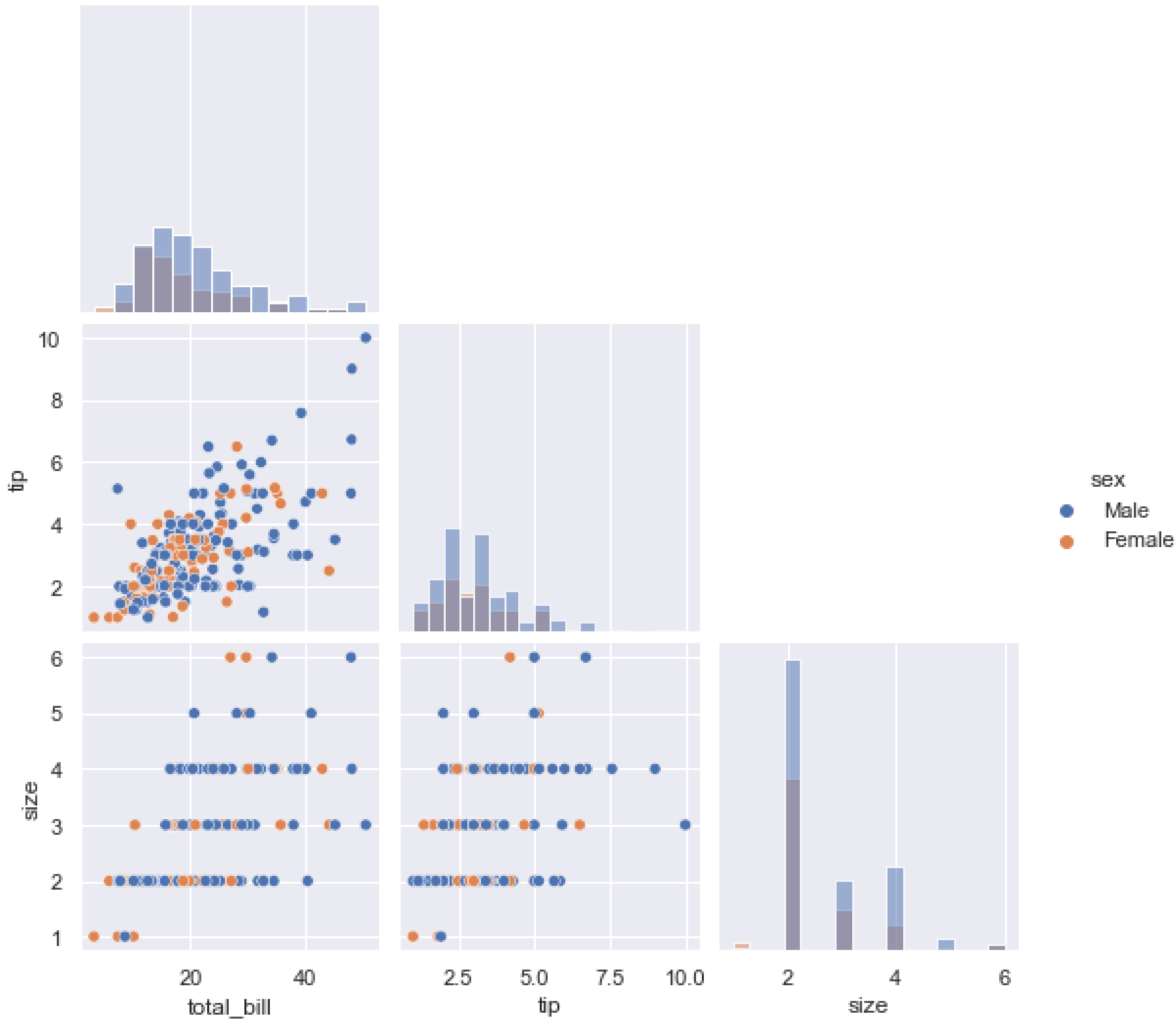
```
# we can convert this into histograms
sns.pairplot(df, hue="sex", diag_kind="hist")
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x1edc9f4e8e0>



In [ ]:
```
# tomake one sided
sns.pairplot(df, hue="sex", diag_kind="hist", corner=True)
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x1edcc9311c0>
```


10
8
6
4
2
tip
6
5
4
3
2
1
size
20
40
total_bill
2.5
5.0
7.5
10.0
tip
2
4
6
size
sex
Male
Female