Exploratory Data Analysis

EDA and its 10 important Steps

In [ ]:

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:

```
# import dataset
df = sns.load_dataset('titanic')
df1 = sns.load_dataset('tips')
```

Step-1

Data shape

In [ ]:

```
# these command shows rows and columns number
print(df.shape)
rows, cols = df.shape
print('Number of Rows are:', rows) # these are called instances
print('Number of cols are:', cols) # these are called series

# if there are many rows than take a sample of it so you can easily perform analysis
# df = df.sample(1000) Apply this to analyse
```

```
(891, 15)
Number of Rows are: 891
Number of cols are: 15
```

Step-2

Data Structure

In [ ]:

```
# This command will show information of data
df.info()
# Sometimes in a dataset numeric values are converted into object and vice versa
# This will be diff to handle such a data
# so we check and if found such an error than remove by astype() function
```

```
<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
```

Step-3

Find Missing Values in Data

In [ ]:

```python
# This command will show missing values
df.isnull()
# missing values are shown as True in following table
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	em
0	False	False	False	False	False	False	False	False	False	False	False	True	
1	False	False	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	False	True	
3	False	False	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	False	True	
...	...	...	...	...	...	...	...	...	...	...	...	...	
886	False	False	False	False	False	False	False	False	False	False	False	True	
887	False	False	False	False	False	False	False	False	False	False	False	False	
888	False	False	False	True	False	False	False	False	False	False	False	True	
889	False	False	False	False	False	False	False	False	False	False	False	False	
890	False	False	False	False	False	False	False	False	False	False	False	True	

891 rows × 15 columns

In [ ]:

```python
# This command will show total number of missing values
df.isnull().sum()
# Sometimes due to presence of missing values the data is non Gaussian
# So first we check and then change them bt diff data wrangling techniques
```

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
```

In [ ]:

```python
# percent calculation of missing values
df.isnull().sum() / df.shape[0] *100 # 0 in square braces indicates the number of row
# We should check % of missing value so we can judge how much they effect efficacy of
```

Out[ ]:

```
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
```

Step-4

Feature Engineering:\ Split Variables for New Columns if Needed

In [ ]:

```
# form a new dataset
city = pd.DataFrame(np.array([["Lahore , Pakistan", 67,100], ["Beijing , China", 5, 6
                    columns = ["address", "males", "females"])
city
```

Out[ ]:

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

In [ ]:

```
# We have to split city and country name in address
city[['City' , 'Country']] = city['address'].str.split(' , ', expand=True)
# (' , ', expand=True) function means that split on the basis of ,
# We split such features so we can get separate series and easily can compare them or
city
```

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

Step-5

Type Casting (Conversion of dtypes)

In [ ]:

```
# type casting helps us to see the dtype of series and so we can change them if neces
city.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 [ ]:

```
# to convert into an int
city[['males','females']] = city[['males','females']].astype('int64')
city.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      int64 
 2   females  3 non-null      int64 
 3   City     3 non-null      object
 4   Country  3 non-null      object
dtypes: int64(2), object(3)
memory usage: 248.0+ bytes
```

In [ ]:

```
# to convert into an str
city[['City','Country']] = city[['City','Country']].astype('str')
city.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      int64
 2   females  3 non-null      int64
 3   City     3 non-null      object
 4   Country  3 non-null      object
dtypes: int64(2), object(3)
memory usage: 248.0+ bytes
```

***ASSIGNMENT*\ Why the info dtype not changed from 'object' to 'str'?**

Solution

In [ ]:

```
city[['City','Country']] = city[['City','Country']].astype('string')
city.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      int64
 2   females  3 non-null      int64
 3   City     3 non-null      string
 4   Country  3 non-null      string
dtypes: int64(2), object(1), string(2)
memory usage: 248.0+ bytes
```

Step-6

Summary Statistics

In [ ]:

```
# Summary told us the mean, std, min, max and iterqortile range
df.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

Step-7

Value Count of a Specific Column/Series

In [ ]:

```
# This function shows us the classes and their instances in a specific column/series
# Through this function we can find the reliability of a class over other
# Because a class with more instances in a series will give more reliable results ove
# other with less instances
df['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 [ ]:

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

Out[ ]:

```
male      577
female    314
Name: sex, dtype: int64
```

In [ ]:

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

Out[ ]:

```
Third     491
First     216
Second    184
Name: class, dtype: int64
```

In [ ]:

```
# finding unique values in a specific column/series
df['class'].unique()
```

Out[ ]:

```
['Third', 'First', 'Second']
Categories (3, object): ['First', 'Second', 'Third']
```

Step-8

Dealing with Duplicates and/or null values\ Duplicates would be simply removed from dataset\ *While*\ (null values maybe replaced by mean,median,log......other methods)

In [ ]:

```
# First find duplicates
df[df.embark_town == 'Queenstown'] # This will show the people only embarked from Que
# if we have any duplicate rows(having all same values) then we should remove them be
# they will consume extra memory,
# they would have no effect on dataset.
# This function is also use to make subset of different classes in a series and compa
```

Out[ ]:

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

77 rows × 15 columns

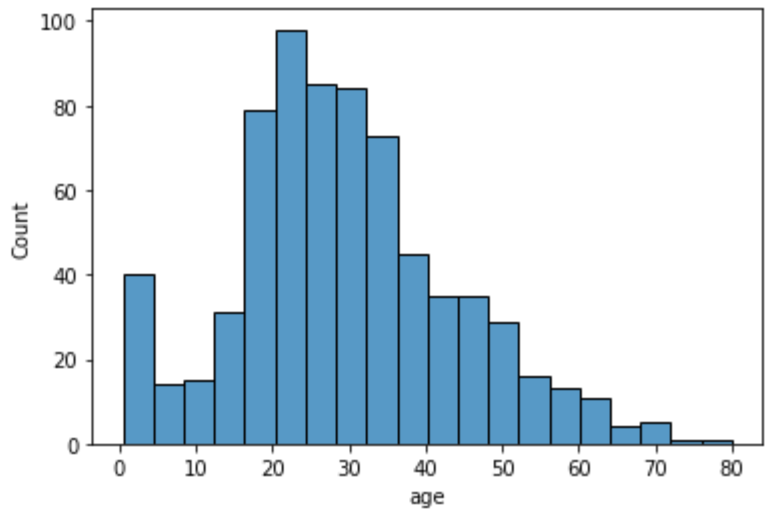
Step-9

Check the Normalality \ Gaussian Distribution\ Best way to check is to make histplot

In [ ]:

```
# Very necessary step because we need this to test our hypothesis
sns.histplot(df['age'])
# sns.distplot(df['age'], kde = False, label = 'female')
```

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



***ASSIGNMENT*\ Make histplot with two variables**

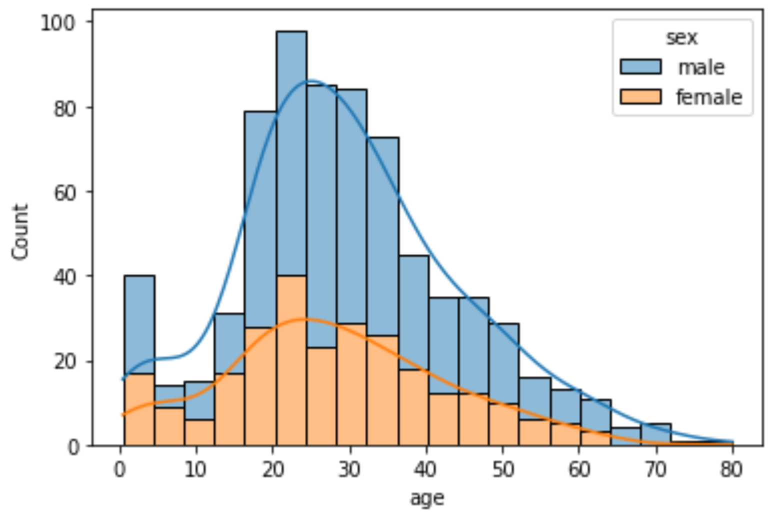
Solution

In [ ]:

```
sns.histplot(data = df, x='age', hue='sex', kde=True, multiple="stack")
```

```
c:\Users\kalee\anaconda3\lib\site-packages\seaborn\distributions.py:244: FutureWarnin
g: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpl
ace instead of always setting a new array. To retain the old behavior, use either `df
[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`
  baselines.iloc[:, cols] = (curves
```

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



- **Measure the Skewness:**
- Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point.
- The skewness for a normal distribution is zero, and any symmetric data should have a skewness near zero. Negative values for the skewness indicate data that are skewed left and positive values for the skewness indicate data that are skewed right. By skewed left, we mean that the left tail is long relative to the right tail. Similarly, skewed right means that the right tail is long relative to the left tail. If the data are multi-modal, then this may affect the sign of the skewness.
- **Measure the Kurtosis:**

- Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. That is, data sets with high kurtosis tend to have heavy tails, or outliers. Data sets with low kurtosis tend to have light tails, or lack of outliers. A uniform distribution would be the extreme case.
- **Dealing with Skewness and Kurtosis:**
- Many classical statistical tests and intervals depend on normality assumptions. Significant skewness and kurtosis clearly indicate that data are not normal. If a data set exhibits significant skewness or kurtosis (as indicated by a histogram or the numerical measures), what can we do about it?

- One approach is to apply some type of transformation to try to make the data normal, or more nearly normal. The ***Box-Cox transformation*** is a useful technique for trying to normalize a data set. In particular, taking the log or square root of a data set is often useful for data that exhibit moderate right skewness.

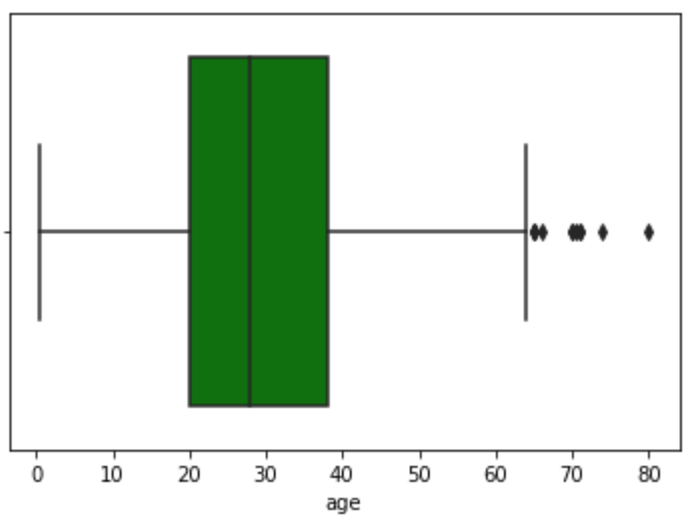
In [ ]:
```
df['age'].agg(['skew' , 'kurtosis']).transpose()
```

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

In [ ]:
```
# Boxplot for normality check
sns.boxplot(df['age'], color='g')
```

```
c:\Users\kalee\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[ ]: <AxesSubplot:xlabel='age'>



Step-10

Check the Correlation

In [ ]:
```
# Every continuous variable have impact over other
# correlation told us how much correlation exist among variables
corr = df.corr(method='pearson') # spearman can also be used
corr
```

```
<ipython-input-24-f7539c77c955>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  corr = df.corr(method='pearson') # spearman can also be used
```

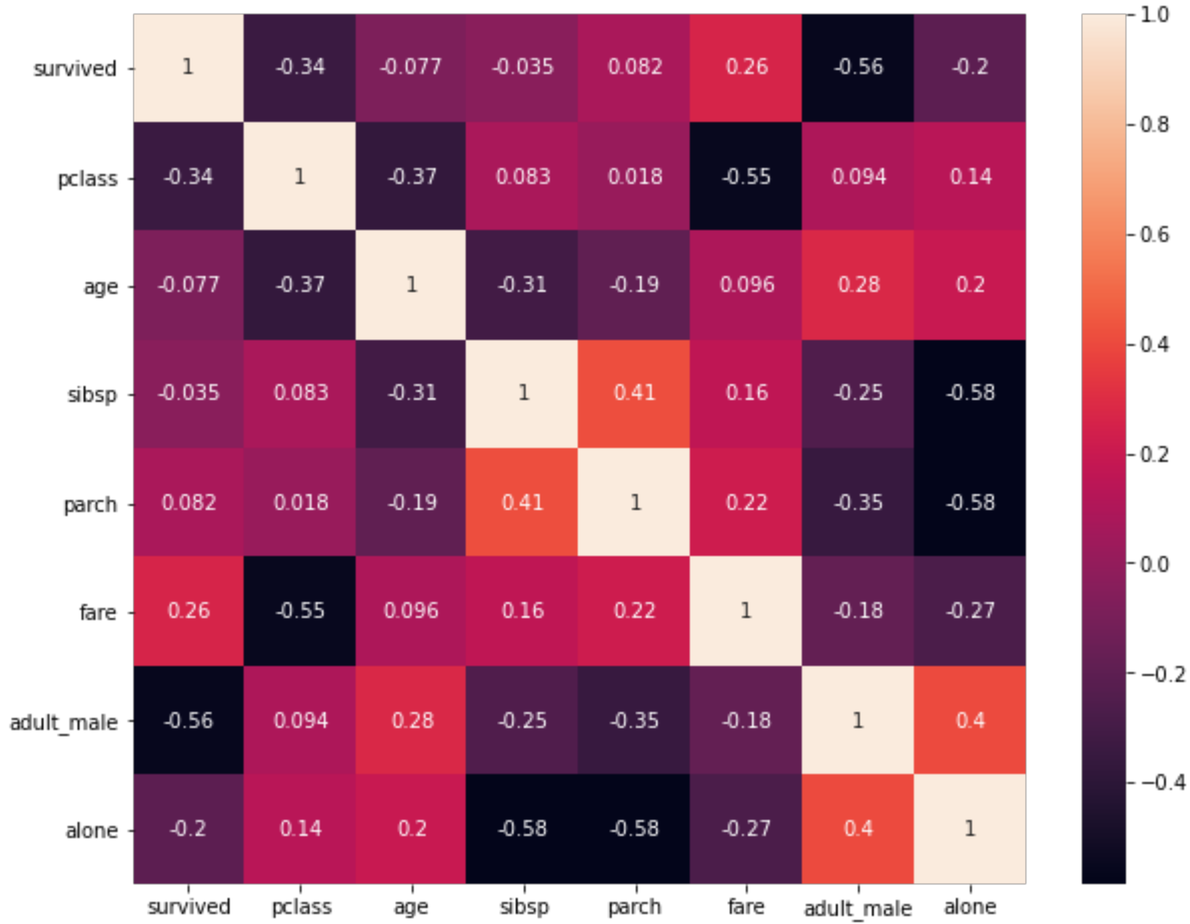
Out[ ]:

	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

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
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
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

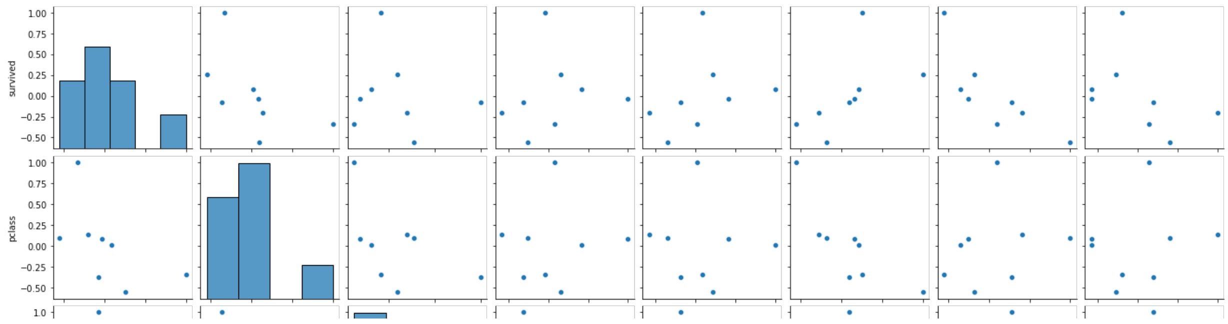
In [ ]:
```
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot = True)
```

Out[ ]: <AxesSubplot:>



In [ ]:
```
sns.pairplot(corr)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x24e7f75b550>



ASSIGNMENTS

1. Find correlation of male_age with female_age.
2. Find correlation of 1st, 2nd and 3rd class male_age and female_age.
3. Find correlation of 1st, 2nd and 3rd class fare.

In [ ]:

```

```