Applying EDA to the titanic data set

- by Ismail Ouahbi

In [1]:
```
#import relevant packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
#set seaborn style as our default style
sns.set()
```

In [2]:
```
#import data
data = pd.read_csv('titanic.csv')
```

In [3]:
```
#take a quick view of data
data.head(15)
```

Out[3]:

	Survived	Pclass	Sex	Age	Siblings/Spouses	Parents/Children	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500
5	0	3	male	27.0	0	0	8.4583
6	0	1	male	54.0	0	0	51.8625
7	0	3	male	2.0	3	1	21.0750
8	1	3	female	27.0	0	2	11.1333
9	1	2	female	14.0	1	0	30.0708
10	1	3	female	4.0	1	1	16.7000
11	1	1	female	58.0	0	0	26.5500
12	0	3	male	20.0	0	0	8.0500
13	0	3	male	39.0	1	5	31.2750
14	0	3	female	14.0	0	0	7.8542

Data explanation

Titanic data set contains 7 columns , 6 features ['Pclass','Sex','Age','Siblings/Spouses','Parents/Children,'Fare'] & one target ['Survived']

The 7 features are indicating a sum of informations about each passenger (does this passenger have siblings ? his gender & age ? and what about the fare (amount paid for this journey) ?..etc

and of course as for each data set there is a purpose , this one is for classification ML algorithms

so the target will be a prediction to wheter the passenger will survive or not

In [4]:
```
#let's start the process
```

In [5]:
```
#check for data types
data.dtypes
```

Out[5]:
```
Survived              int64
Pclass                int64
Sex                  object
Age                 float64
Siblings/Spouses      int64
Parents/Children      int64
Fare                float64
dtype: object
```

---

as you can notice the "Sex" feature has a categorical type let's continue and see how to make it numerical

In [6]:
```
#get statistical measures
data.describe()
```

Out[6]:

	Survived	Pclass	Age	Siblings/Spouses	Parents/Children	Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

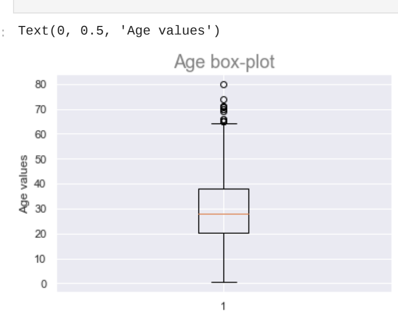
The older person has an age of 80

data box-plot

- using matplotlib

In [7]:
```
plt.boxplot(data['Age'])
plt.title("Age box-plot" , fontsize=18 , c='gray')
plt.ylabel('Age values')
```

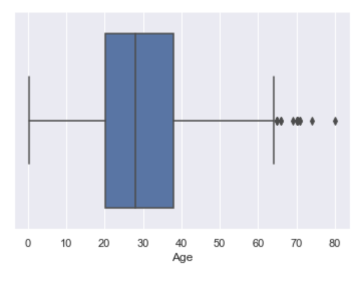
Out[7]: `Text(0, 0.5, 'Age values')`



- using seaborn

In [8]:
```
sns.boxplot(x=data['Age'])
```

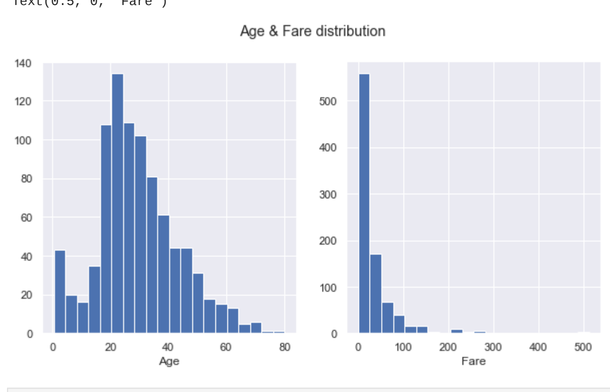
Out[8]: `<AxesSubplot:xlabel='Age'>`



data histogram

In [9]:
```
figure,(ax1,ax2) = plt.subplots(1,2 , figsize=(10,5))
figure.suptitle('Age & Fare distribution')
ax1.hist(x=data['Age'] , bins=20)
ax1.set_xlabel('Age')
ax2.hist(x=data['Fare'] , bins=20)
ax2.set_xlabel('Fare')
```

Out[9]: `Text(0.5, 0, 'Fare')`



In [10]:
```
#get the size of our data set
data.shape
```

Out[10]: `(887, 7)`

In [11]:
```
#get informations
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887 entries, 0 to 886
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Survived          887 non-null    int64  
 1   Pclass            887 non-null    int64  
 2   Sex               887 non-null    object 
 3   Age               887 non-null    float64
 4   Siblings/Spouses  887 non-null    int64  
 5   Parents/Children  887 non-null    int64  
 6   Fare              887 non-null    float64
dtypes: float64(2), int64(4), object(1)
memory usage: 48.6+ KB
```

In [12]:
```
#the number of persons for each gender
data.groupby('Sex').count()
```

Out[12]:

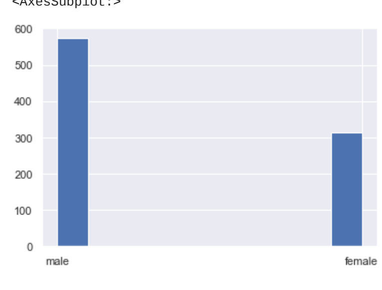
Sex	Survived	Pclass	Age	Siblings/Spouses	Parents/Children	Fare
female	314	314	314	314	314	314
male	573	573	573	573	573	573

In [13]:
```
#or
data.Sex.value_counts()
```

Out[13]:
```
male      573
female    314
Name: Sex, dtype: int64
```

In [14]:
```
data.Sex.hist()
```

Out[14]: `<AxesSubplot:>`



---

each measure has an insight from the above 3 cells we can notice that the majority of passengers are men

---

what about the number of survived persons amoung each gender ? let's explore them

In [15]:
```
data[data['Survived'] == 1].groupby('Sex').count()
```

Out[15]:

Sex	Survived	Pclass	Age	Siblings/Spouses	Parents/Children	Fare
female	233	233	233	233	233	233
male	109	109	109	109	109	109

Ask questions about data

---

can we explain this by the power of man body to take risk and dive into the sea to help others so the probability of death will increase ? what about women , is their fear useful here ? (Not all but in general women are more likely to hide and wait for help :)) let's continue exploring

---

before going further let's encode the sex column as it is useful in our data exploration

In [16]:
```
#import sklearn packege for help
from sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
#this will encode the male as 1 & female as 0
data["Sex_encoded"] = ord_enc.fit_transform(data[["Sex"]])
```

---------------------See the difference---------------------------

In [17]:
```
data
```

Out[17]:

	Survived	Pclass	Sex	Age	Siblings/Spouses	Parents/Children	Fare	Sex_encoded
0	0	3	male	22.0	1	0	7.2500	1.0
1	1	1	female	38.0	1	0	71.2833	0.0
2	1	3	female	26.0	0	0	7.9250	0.0
3	1	1	female	35.0	1	0	53.1000	0.0
4	0	3	male	35.0	0	0	8.0500	1.0
...	...	...	...	...	...	...	...	...
882	0	2	male	27.0	0	0	13.0000	1.0
883	1	1	female	19.0	0	0	30.0000	0.0
884	0	3	female	7.0	1	2	23.4500	0.0
885	1	1	male	26.0	0	0	30.0000	1.0
886	0	3	male	32.0	0	0	7.7500	1.0

887 rows × 8 columns

In [18]:
```
#as we will need just the new encoded 'Sex' column instead of the old
 #let's update our data frame

#take a copy of the original one (in case we need it)
df = data.copy()
data.drop(['Sex'] , axis = 1 , inplace=True)
```

In [19]:
```
#new data set
data
```

Out[19]:

	Survived	Pclass	Age	Siblings/Spouses	Parents/Children	Fare	Sex_encoded
0	0	3	22.0	1	0	7.2500	1.0
1	1	1	38.0	1	0	71.2833	0.0
2	1	3	26.0	0	0	7.9250	0.0
3	1	1	35.0	1	0	53.1000	0.0
4	0	3	35.0	0	0	8.0500	1.0
...	...	...	...	...	...	...	...
882	0	2	27.0	0	0	13.0000	1.0
883	1	1	19.0	0	0	30.0000	0.0
884	0	3	7.0	1	2	23.4500	0.0
885	1	1	26.0	0	0	30.0000	1.0
886	0	3	32.0	0	0	7.7500	1.0

887 rows × 7 columns

missing values

In [20]:
```
# check for missing values
data.isnull().sum()

#--------No null values :)
```

Out[20]:
```
Survived            0
Pclass              0
Age                 0
Siblings/Spouses    0
Parents/Children    0
Fare                0
Sex_encoded         0
dtype: int64
```

Big view of data

In [21]:
```
#to take a big view of our data let's use seaborn pairplot
sns.pairplot(data , hue='Sex_encoded')
```

Out[21]: `<seaborn.axisgrid.PairGrid at 0x198050a9be0>`

Skewness

In [22]:
```
#check for skewness (to decide whether a feature is (~) normaly ditributed or not)
 #to make skewed variables symmetric let's first extract just decimal features

decimals = df.dtypes[df.dtypes == float]

decimal_cols = decimals.index.tolist()
decimal_cols
```

Out[22]: `['Age', 'Fare', 'Sex_encoded']`

You may ask why skewed data can be harmful ? check out this article to see more Skewed Data: A problem to your statistical model

In [23]:
```
#as the sex encoded variable is not important in terms of skewness we will remove it from our transformation
decimal_cols = [index for index in decimal_cols if index!='Sex_encoded']
decimal_cols = df[decimal_cols]
```

In [24]:
```
#let's verify
decimal_cols.dtypes
```

Out[24]:
```
Age     float64
Fare    float64
dtype: object
```

In [25]:
```
#filter those who respond to our skew condition
skew_limit = 0.75 # define a limit above which we will log transform
skew_vals = decimal_cols.skew()
skew_vals
```

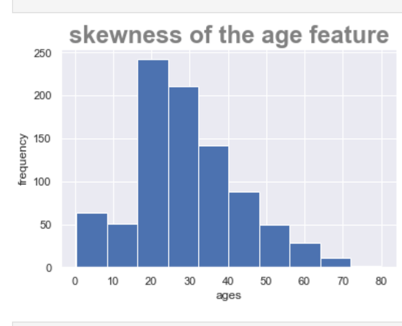
Out[25]:
```
Age     0.447189
Fare    4.777671
dtype: float64
```

In [26]:
```
#filter
# Show the skewed columns
skew_cols = (skew_vals
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))

skew_cols
```

Out[26]:

	Skew
Fare	4.777671

In [27]:
```
#before we apply log transform to our feature let's plot the excluded feature to ensure that has a non important skewness
plt.hist(data['Age'])
plt.title('skewness of the age feature' , weight = 'bold' , c='gray' , fontsize = 24)
plt.xlabel('ages')
plt.ylabel('frequency')
plt.show()
```



In [28]:
```
#apply log transform to the skewed feature
 #warning : the log1p function is defined as log(1+x)
data[skew_cols.index] = data[skew_cols.index].apply(np.log1p)
```

In [29]:
```
#see the difference
pd.DataFrame({
    'before log' : df['Fare'],
    'after log' : data['Fare']
}).head(10)
```

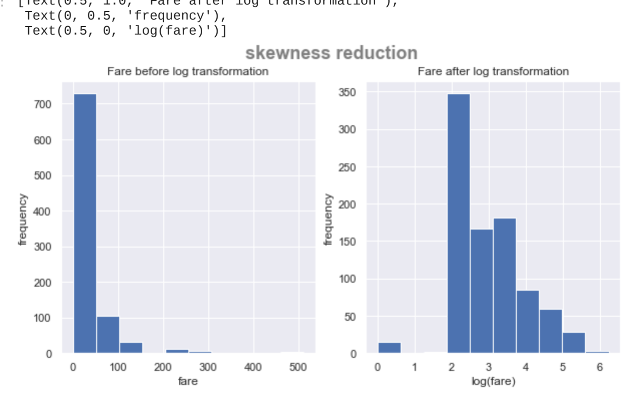
Out[29]:

	before log	after log
0	7.2500	2.110213
1	71.2833	4.280593
2	7.9250	2.188856
3	53.1000	3.990834
4	8.0500	2.202765
5	8.4583	2.246893
6	51.8625	3.967694
7	21.0750	3.094446
8	11.1333	2.495954
9	30.0708	3.436268

---------------------------Plot to check skewness------------------------------

In [30]:
```
#for both variables

# Create two "subplots" and a "figure" using matplotlib
fig, (ax_before, ax_after) = plt.subplots(1, 2, figsize=(10, 5))
fig.suptitle('skewness reduction' , fontsize = 18 , c ='gray' , fontweight = 'bold')

# Create a histogram on the "ax_before" subplot
decimal_cols['Fare'].hist(ax=ax_before)

# Apply a log transformation (numpy syntax) to this column
data['Fare'].hist(ax=ax_after)

# Formatting of titles etc. for each subplot
ax_before.set(title='Fare before log transformation', ylabel='frequency', xlabel='fare')
ax_after.set(title='Fare after log transformation', ylabel='frequency', xlabel='log(fare)')
```

Out[30]:
```
[Text(0.5, 1.0, 'Fare after log transformation'),
 Text(0, 0.5, 'frequency'),
 Text(0.5, 0, 'log(fare)')]
```

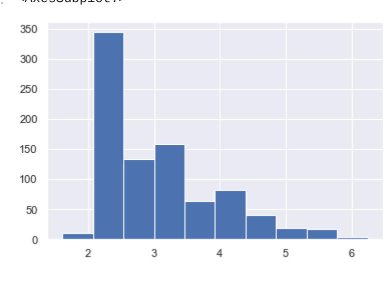


check for outliers

In [42]:
```
#from the above plots we can notice that some outliers exist at ~0 value (for the top-right plot)
 #this is of course due to original data(raw data)
#let's handle it before moving further

#let's check for them and remove them (our handling for this situation) (note that there are other methods to deal with outliers)
indexes = data[data['Fare']==0].index.tolist()

data.drop(indexes , inplace=True , axis = 0)
```

In [43]:
```
#verify
data['Fare'].hist()

#now it's ok
```

Out[43]: `<AxesSubplot:>`



Normality Test

- This is a statistical test that tests whether a distribution is normally distributed or not. It isn't perfect, but suffice it to say:
  - This test outputs a "p-value". The *higher* this p-value is the *closer* the distribution is to normal.
  - Statisticians would say that you accept that the distribution is normal (more specifically: fail to reject the null hypothesis that it is normal) if p > 0.05.

In [33]:
```
#import normality test model
from scipy.stats.mstats import normaltest
```

In [34]:
```
#apply the test

#before log transformation
normaltest(df['Fare'])
```

Out[34]: `NormaltestResult(statistic=899.3332238340862, pvalue=5.155511361775634e-196)`

** As we can notice the p-value is extremely low so we can say that the null hypothesis can be rejected(alpha value in general=5%)

In [44]:
```
#after log transformation & outliers removing
normaltest(data['Fare'])
```

Out[44]: `NormaltestResult(statistic=99.92690115361523, pvalue=2.000548652243953e-22)`

** Our p-value after transformation is reduced but still rejects the (H0) : null hypothesis

** In our previous code , we deleted outiers as a sample approach knowing that we can handle outliers using other approaches, but in many times they can be useful and informative especially when dealing with non-error outliers, also called interesting or random outliers

*** Let's apply the normality test to the log transformed data before dealing with outliers

In [45]:
```
#get data
testing_column = df['Fare'].apply(np.log1p)
```

In [46]:
```
#apply test
normaltest(testing_column)
```

Out[46]: `NormaltestResult(statistic=38.86870541669266, pvalue=3.628840328852395e-09)`

In [47]:
```
np.subtract(normaltest(testing_column)[1] ,normaltest(data['Fare'])[1] )
```

Out[47]: `3.6288403288521947e-09`

as we can notice , our data with outliers is more to normal distribution than data without outliers

What are the other approaches that we can opt in order to handle outliers ?

- in order to answer this question I highly recommend this article for you : Detecting and Treating Outliers

by Ismail Ouahbi , please contact me for personal usage

Thank you