Aim:Demonstrate Data Imputation with statistical technique on numerical values and right down the conclusion about the assumption

In [59]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
df=pd.read_csv('titanic_toy.csv')
df
```

Out[4]:

	Age	Fare	Family	Survived
0	22.0	7.2500	1	0
1	38.0	71.2833	1	1
2	26.0	7.9250	0	1
3	35.0	53.1000	1	1
4	35.0	8.0500	0	0
...	...	...	...	...
886	27.0	13.0000	0	0
887	19.0	30.0000	0	1
888	NaN	23.4500	3	0
889	26.0	NaN	0	1
890	32.0	7.7500	0	0

891 rows × 4 columns

In [5]:
```
df.head()
```

Out[5]:

	Age	Fare	Family	Survived
0	22.0	7.2500	1	0
1	38.0	71.2833	1	1
2	26.0	7.9250	0	1
3	35.0	53.1000	1	1
4	35.0	8.0500	0	0

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Age         177
Fare         45
Family        0
Survived      0
dtype: int64
```

In [7]:
```
df.isnull().mean()*100
```

Out[7]:
```
Age         19.865320
Fare         5.050505
Family       0.000000
Survived     0.000000
dtype: float64
```

In [8]:
```
x=df.drop(columns=['Survived'])#independent columns
```

In [9]:
```
y=df['Survived']#dependent columns
```

In [10]:
```
y
```

Out[10]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

In [11]:
```
df.shape
```
Out[11]:
```
(891, 4)
```

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)
```

In [14]:
```
x_train.shape
```
Out[14]:
```
(712, 3)
```

In [15]:
```
x_test.shape
```
Out[15]:
```
(179, 3)
```

In [17]:
```
df.describe()
```
Out[17]:

	Age	Fare	Family	Survived
count	714.000000	846.000000	891.000000	891.000000
mean	29.699118	32.279338	0.904602	0.383838
std	14.526497	50.305796	1.613459	0.486592
min	0.420000	0.000000	0.000000	0.000000
25%	20.125000	7.895800	0.000000	0.000000
50%	28.000000	14.454200	0.000000	0.000000
75%	38.000000	31.206250	1.000000	1.000000
max	80.000000	512.329200	10.000000	1.000000

In [18]:
```
mean_age=x_train['Age'].mean()
```

In [19]:
```
mean_age
```
Out[19]:
```
29.78590425531915
```

In [20]:
```
df.describe()
```
Out[20]:

	Age	Fare	Family	Survived
count	714.000000	846.000000	891.000000	891.000000
mean	29.699118	32.279338	0.904602	0.383838
std	14.526497	50.305796	1.613459	0.486592
min	0.420000	0.000000	0.000000	0.000000
25%	20.125000	7.895800	0.000000	0.000000
50%	28.000000	14.454200	0.000000	0.000000
75%	38.000000	31.206250	1.000000	1.000000
max	80.000000	512.329200	10.000000	1.000000

In [22]:
```
median_age=x_train['Age'].median()
mean_age=x_train['Age'].mean()
```

In [24]:
```
median_age
```
Out[24]:
```
28.75
```

In [27]:
```
mean_fare=x_train['Fare'].mean()
median_fare=x_train['Fare'].median()
```

In [28]:
```
mean_fare
```
Out[28]:
```
32.617596893491076
```

In [29]:
```
median_fare
```
Out[29]:
```
14.4583
```

In [32]:
```
x_train['Age_mean']=x_train['Age'].fillna(mean_age)
x_train['Age_median']=x_train['Age'].fillna(median_age)
```

In [34]:
```
x_train['Fare_mean']=x_train['Fare'].fillna(mean_fare)
x_train['Fare_median']=x_train['Fare'].fillna(median_fare)
```

In [35]:
```
x_train
```

Out[35]:

	Age	Fare	Family	Age_mean	Age_median	Fare_mean	Fare_median
30	40.0	27.7208	0	40.000000	40.0000	27.7208	27.7208
10	4.0	16.7000	2	4.000000	4.0000	16.7000	16.7000
873	47.0	9.0000	0	47.000000	47.0000	9.0000	9.0000
182	9.0	31.3875	6	9.000000	9.0000	31.3875	31.3875
876	20.0	9.8458	0	20.000000	20.0000	9.8458	9.8458
...	...	...	...	...	...	...	...
534	30.0	8.6625	0	30.000000	30.0000	8.6625	8.6625
584	NaN	8.7125	0	32.617597	14.4583	8.7125	8.7125
493	71.0	49.5042	0	71.000000	71.0000	49.5042	49.5042
527	NaN	221.7792	0	32.617597	14.4583	221.7792	221.7792
168	NaN	25.9250	0	32.617597	14.4583	25.9250	25.9250

712 rows × 7 columns

In [37]:
```
print("Before imputation age variance is",x_train['Age'].var())
print("After imputation age variance is",x_train['Age_median'].var())
print("After imputation age variance is",x_train['Age_mean'].var())
```

```
Before imputation age variance is 204.3495133904614
After imputation age variance is 200.55085535155024
After imputation age variance is 163.1347828052615
```

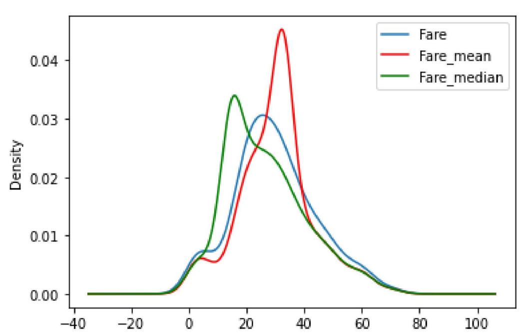
In [38]:
```
print("Before imputation age variance is",x_train['Fare'].var())
print("After imputation age variance is",x_train['Fare_median'].var())
print("After imputation age variance is",x_train['Fare_mean'].var())
```

```
Before imputation age variance is 2448.197913706318
After imputation age variance is 2340.0910219753637
After imputation age variance is 2324.2385256705547
```

In [41]:
```
import matplotlib.pyplot as plt
```

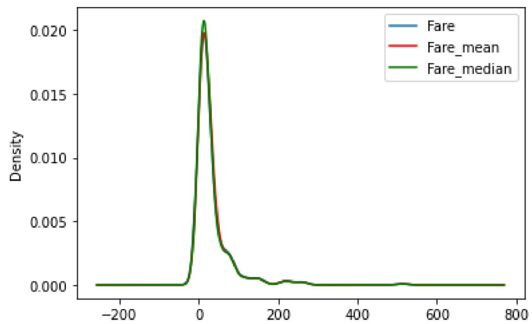
In [47]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
x_train['Age'].plot(kind='kde',ax=ax)
x_train['Age_mean'].plot(kind='kde',ax=ax,color='red')
x_train['Age_median'].plot(kind='kde',ax=ax,color='green')
ax.legend(line,labels,loc='best')
```

Out[47]: <matplotlib.legend.Legend at 0x1dbb3965130>



In [48]:
```
fig=plt.figure()
ax=fig.add_subplot(111)
x_train['Fare'].plot(kind='kde',ax=ax)
x_train['Fare_mean'].plot(kind='kde',ax=ax,color='red')
x_train['Fare_median'].plot(kind='kde',ax=ax,color='green')
ax.legend(line,labels,loc='best')
```

Out[48]: <matplotlib.legend.Legend at 0x1dbb39a73a0>



In [50]:
```
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
```

In [51]:
```
imputer1=SimpleImputer(strategy='mean')
imputer2=SimpleImputer(strategy='median')
```

In [52]:
```
trf=ColumnTransformer([
    ('imputer1',imputer1,['Age']),
    ('imputer2',imputer2,['Age'])
],remainder='passthrough')
```

In [53]:
```
trf.fit(df)
```

Out[53]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('imputer1', SimpleImputer(), ['Age']),
                                ('imputer2', SimpleImputer(strategy='median'),
                                 ['Age'])])
```

In [55]:
```
trf.named_transformers_['imputer1'].statistics_
```

Out[55]: array([29.69911765])

In [56]:
```
trf.named_transformers_['imputer2'].statistics_
```

Out[56]: array([28.])

In [60]:
```
sm=trf.transform(df)
```

In [61]:
```
sm
```

Out[61]:
```
array([[22.        , 22.        ,  7.25      ,  1.        ,  0.        ],
       [38.        , 38.        , 71.2833    ,  1.        ,  1.        ],
       [26.        , 26.        ,  7.925     ,  0.        ,  1.        ],
       ...,
       [29.69911765, 28.        , 23.45      ,  3.        ,  0.        ],
       [26.        , 26.        ,         nan,  0.        ,  1.        ],
       [32.        , 32.        ,  7.75      ,  0.        ,  0.        ]])
```

Conclusion:Mean median mode can be used to fill missing value if the missing value present in the data is less than 5%

In [ ]: