In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv('pubg')
```

In [3]:
```
df.head()
```

Out[3]:

Played	solo_Wins	solo_WinTop10Ratio	solo_BestRating	solo_Kills	duo_RoundsPlayed	duo_Wins	duo_BestRating	squad_RoundsPlayed	squad_Wins	squad_B
17	3	0.83	1415.79	44	15	5	1927.91	642	305	
33	6	0.36	1860.74	119	14	5	2061.61	722	338	
5	0	0.00	1266.60	18	17	6	2052.94	733	347	
8	4	0.67	1765.13	56	3	2	1465.88	491	207	
6	2	0.40	1616.58	42	105	27	2366.20	416	193	

In [4]:
```
df.drop(columns='Unnamed: 0',inplace=True)
```

dataset info

#insights-->this data is about player statestics of mobile game name pubg. data contains 87898 rowns and 14 columns. all the columns are non-null only one object column which is our target column. we have to predict the player rating in below dataset.

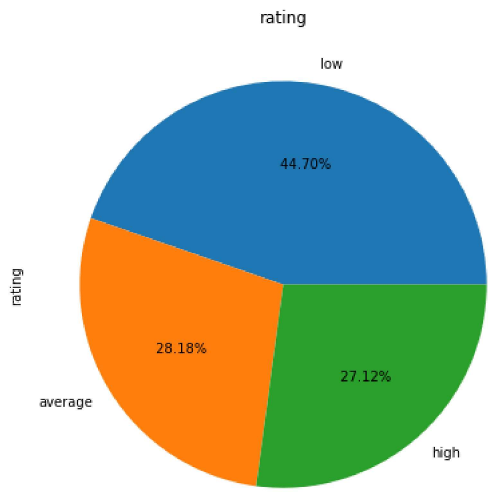
In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 87898 entries, 0 to 87897
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   tracker_id          87898 non-null  int64  
 1   solo_RoundsPlayed   87898 non-null  int64  
 2   solo_Wins           87898 non-null  int64  
 3   solo_WinTop10Ratio  87898 non-null  float64
 4   solo_BestRating     87898 non-null  float64
 5   solo_Kills          87898 non-null  int64  
 6   duo_RoundsPlayed    87898 non-null  int64  
 7   duo_Wins            87898 non-null  int64  
 8   duo_BestRating      87898 non-null  float64
 9   squad_RoundsPlayed  87898 non-null  int64  
 10  squad_Wins          87898 non-null  int64  
 11  squad_BestRating    87898 non-null  float64
 12  squad_Kills         87898 non-null  int64  
 13  rating              87898 non-null  object 
dtypes: float64(4), int64(9), object(1)
memory usage: 9.4+ MB
```

distribution of target column

which will show whether data is balance or imbalance --->our target column in quiet balance in three different catagories. ---->as it has more than two catagory in output columns,multiple classification model will work better.

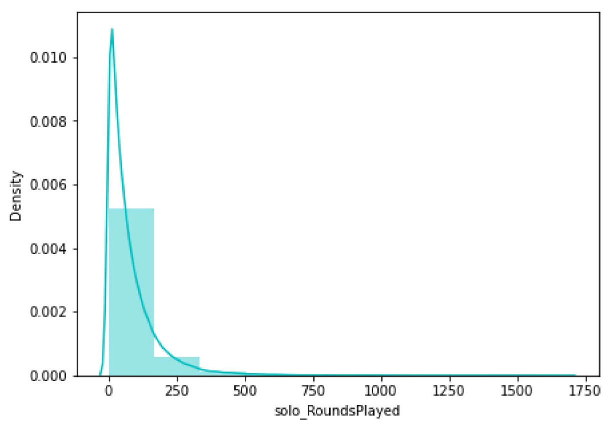
In [7]:
```
plt.figure(figsize=(7,7))
df['rating'].value_counts().plot.pie(autopct='% 1.2f%%',explode=(0,0,0))
plt.title('rating')
plt.show()
```



duata distribution of each column

using histogram we can check the skewness in the data,also indicates presence of outliers. very few columns are symetric most of the columns are left skewed.

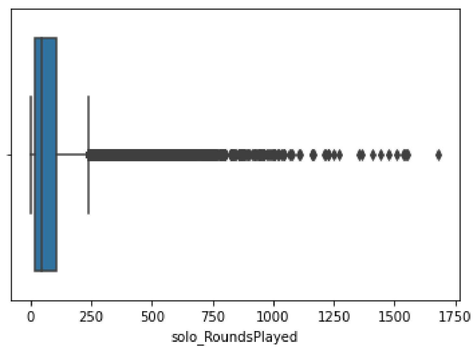
In [11]:
```
c=df.columns[1:-1]
for i in c:
    plt.figure(figsize=(7,5))
    sns.distplot(df[i],bins=10,color='c')
    plt.show()
```



presence of outliers.

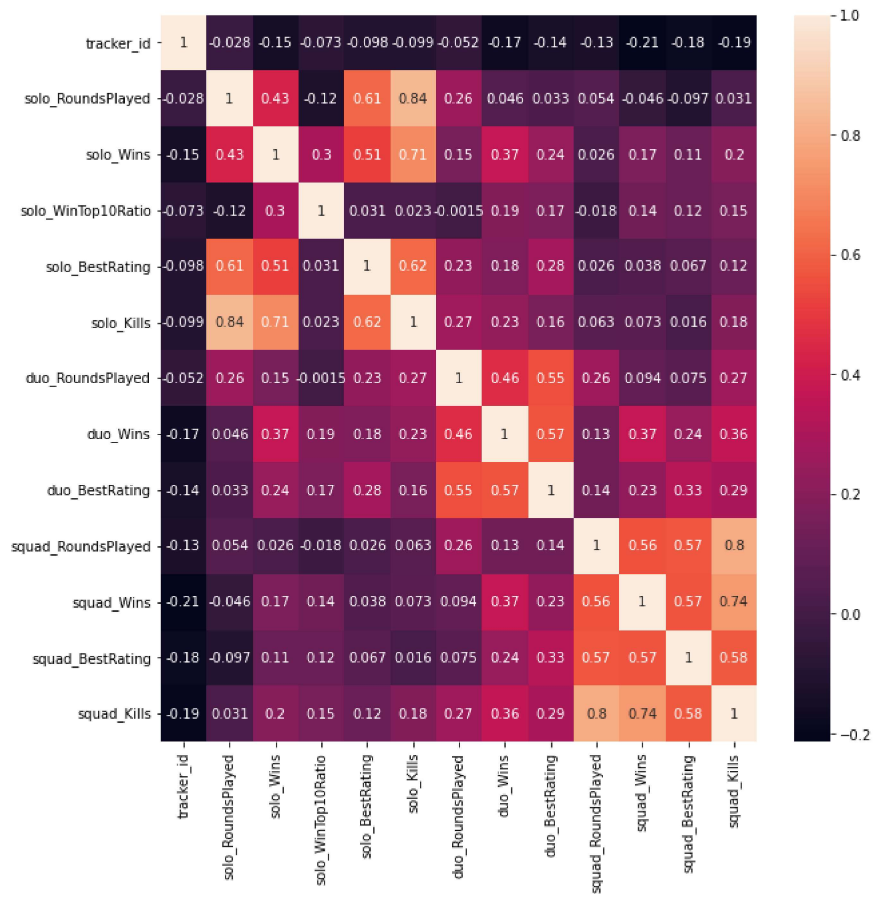
boxplot will help you to show the presence of outliers in each column.-----> most of the columns have high number of outliers so we just cant remove them because of data loss. so models which can not get affected by outliers will work efficiently.

In [55]:
```
col=df.columns[1:-1]
for i in col:
    sns.boxplot(data=df,x=i)
    plt.show()
```



corelation using heatmap

In [13]:
```
corr=df.corr()
plt.figure(figsize=(10,10))
sns.heatmap(corr,annot= True)
plt.show()
```



importing all models and creating objects

In [14]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
from sklearn.svm import SVC
svm=SVC(kernel='linear')
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [15]:
```
x=df.iloc[:,1:-1]
```

In [16]:
```
y=le.fit_transform(df['rating'])
```

In [17]:
```
xtest,xtrain,ytest,ytrain=train_test_split(x,y,test_size=0.30,random_state=1)
```

In [18]:
```
def model(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print(classification_report(ytest,ypred))
```

logistic regression

In [19]:
```
model(lr)
```

```
              precision    recall  f1-score   support

           0       0.39      0.13      0.20     17335
           1       0.55      0.60      0.58     16692
           2       0.61      0.82      0.70     27501

    accuracy                           0.57     61528
   macro avg       0.52      0.52      0.49     61528
weighted avg       0.53      0.57      0.53     61528
```

Support vector machine(svm)

In [20]:
```
model(svm)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17335
           1       1.00      1.00      1.00     16692
           2       1.00      1.00      1.00     27501

    accuracy                           1.00     61528
   macro avg       1.00      1.00      1.00     61528
weighted avg       1.00      1.00      1.00     61528
```

K-nearest neighbors(knn)

In [21]:
```
model(knn)
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93     17335
           1       0.97      0.97      0.97     16692
           2       0.98      0.97      0.98     27501

    accuracy                           0.96     61528
   macro avg       0.96      0.96      0.96     61528
weighted avg       0.96      0.96      0.96     61528
```

decision tree

In [22]:
```
model(dt)
```

```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91     17335
           1       0.96      0.96      0.96     16692
           2       0.97      0.97      0.97     27501

    accuracy                           0.95     61528
   macro avg       0.95      0.95      0.95     61528
weighted avg       0.95      0.95      0.95     61528
```

ensemble learning-->random forest

In [23]:
```
model(rf)
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93     17335
           1       0.98      0.96      0.97     16692
           2       0.98      0.98      0.98     27501

    accuracy                           0.96     61528
   macro avg       0.96      0.96      0.96     61528
weighted avg       0.96      0.96      0.96     61528
```

bagging on knn

In [34]:
```
from sklearn.ensemble import BaggingClassifier
bg=BaggingClassifier(knn)
```

In [35]:
```
model(bg)
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93     17335
           1       0.97      0.96      0.97     16692
           2       0.98      0.97      0.98     27501

    accuracy                           0.96     61528
   macro avg       0.96      0.96      0.96     61528
weighted avg       0.96      0.96      0.96     61528
```

Ada boosting

In [26]:
```
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
ada=AdaBoostClassifier()
gb=GradientBoostingClassifier()
```

In [27]:
```
model(ada)
```

```
              precision    recall  f1-score   support

           0       0.76      0.99      0.86     17335
           1       1.00      0.88      0.93     16692
           2       1.00      0.88      0.94     27501

    accuracy                           0.91     61528
   macro avg       0.92      0.92      0.91     61528
weighted avg       0.93      0.91      0.91     61528
```

gradient boosting

In [28]:
```
model(gb)
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     17335
           1       0.99      0.96      0.97     16692
           2       0.99      0.98      0.98     27501

    accuracy                           0.97     61528
   macro avg       0.97      0.97      0.97     61528
weighted avg       0.97      0.97      0.97     61528
```

xtreme gradient boost

In [29]:
```
from xgboost import XGBClassifier
xgb=XGBClassifier()
```

In [30]:
```
model(xgb)
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.96     17335
           1       0.99      0.98      0.98     16692
           2       0.99      0.99      0.99     27501

    accuracy                           0.98     61528
   macro avg       0.98      0.98      0.98     61528
weighted avg       0.98      0.98      0.98     61528
```

finding highly accurate model

1)svm showed highest accuracy at 'linear kernel'. svm cannot affect by outliers and also performs well on polynomial classification. but it will take time on large dataset. 2) in ensemble learning xgboost shoed highest acuuracy. but we will consider svm model as it build 100% accurate model.

prediction

In [31]:
```
p=svm.predict([[23,2,0.18,1460.74,98,7,5,1561.61,500,200,1508.75,1601]])
```

In [32]:
```
p
```

Out[32]:
```
array([2])
```

In [33]:
```
import pickle
```

In [ ]:
```
pickle.dump(svm,open('svmobj.pkl','wb'))
```

In [ ]: