Naive Bayes

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
Naive = pd.read_csv(r"C:\Users\Vicky Yewle\Downloads\Machine Learning\Datasets\MobilePriceRangePrediction\Mobile Price Range Predi
Naive.head()
```

Out[2]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_tim
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	1
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	1

5 rows × 21 columns

In [3]:
```
Naive.shape
Naive.describe()
```

Out[3]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.00000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.10800
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.78081
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.00000
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.75000

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000

8 rows × 21 columns

In [4]:
```
Naive.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   battery_power  2000 non-null   int64  
 1   blue           2000 non-null   int64  
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64  
 4   fc             2000 non-null   int64  
 5   four_g         2000 non-null   int64  
 6   int_memory     2000 non-null   int64  
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64  
 9   n_cores        2000 non-null   int64  
 10  pc             2000 non-null   int64  
 11  px_height      2000 non-null   int64  
 12  px_width       2000 non-null   int64  
 13  ram            2000 non-null   int64  
 14  sc_h           2000 non-null   int64  
 15  sc_w           2000 non-null   int64  
 16  talk_time      2000 non-null   int64  
 17  three_g        2000 non-null   int64  
 18  touch_screen   2000 non-null   int64  
 19  wifi           2000 non-null   int64  
 20  price_range    2000 non-null   int64  
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
```

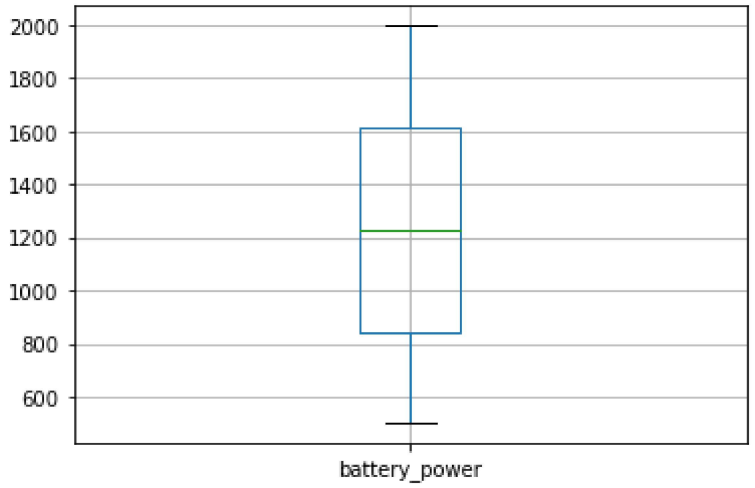
In [5]:
```
Naive  = Naive[Naive['sc_w']>0]
Naive  = Naive[Naive['px_height']>0]
Naive.describe()
```

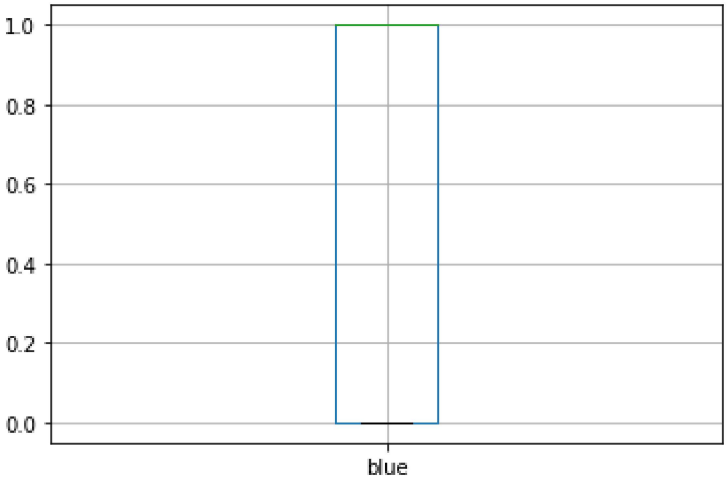
Out[5]:

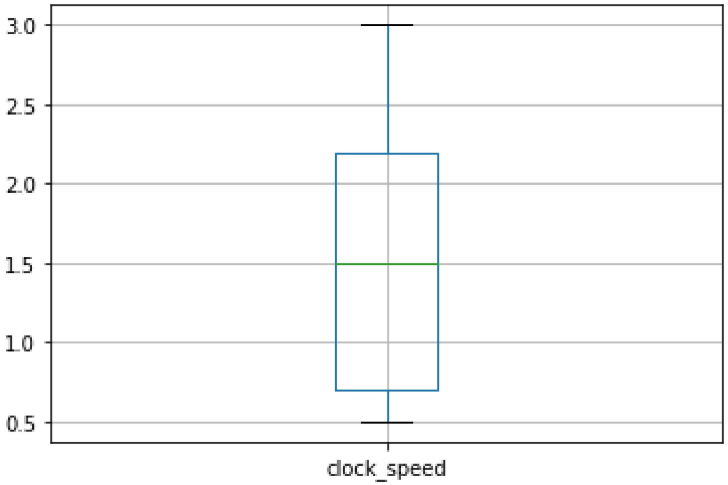
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_heig
count	1819.000000	1819.000000	1819.000000	1819.000000	1819.000000	1819.000000	1819.000000	1819.000000	1819.000000	1819.000000	...	1819.000(
mean	1238.031336	0.503573	1.519406	0.504673	4.319956	0.524464	32.156130	0.499835	140.578340	4.531061	...	647.086{
std	439.989288	0.500125	0.813975	0.500116	4.355982	0.499538	18.105723	0.288875	35.437231	2.288705	...	444.7411
min	501.000000	0.000000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	1.000(
25%	845.000000	0.000000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	284.000(
50%	1231.000000	1.000000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	5.000000	...	562.000(
75%	1611.000000	1.000000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.500000	7.000000	...	952.000(
max	1998.000000	1.000000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000(

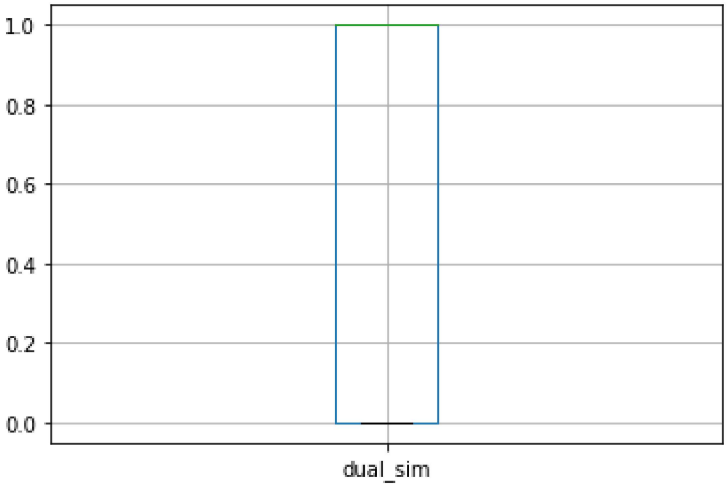
8 rows × 21 columns

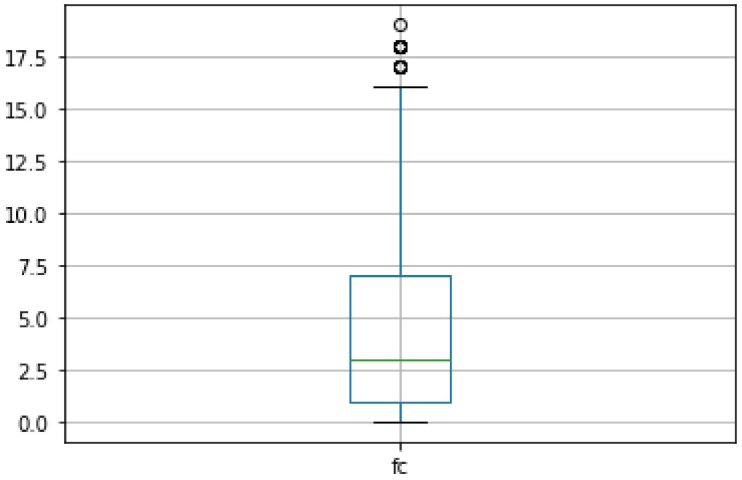
In [6]:
```
for column in Naive:
    plt.figure()
    Naive.boxplot([column])
```

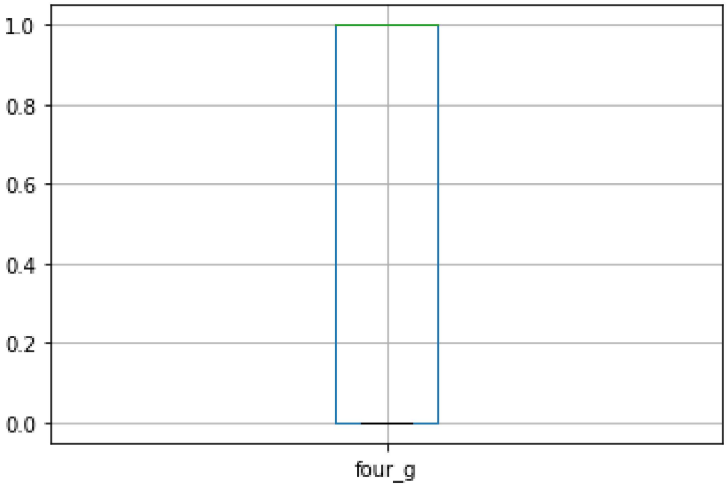


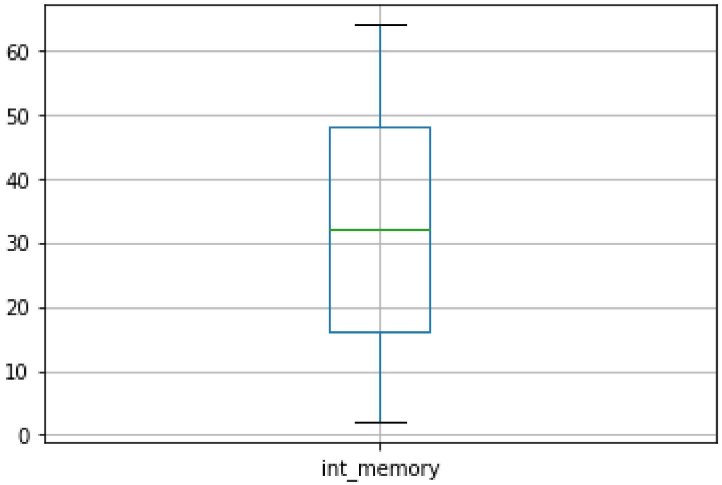

1.0
0.8
0.6
0.4
0.2
0.0
blue

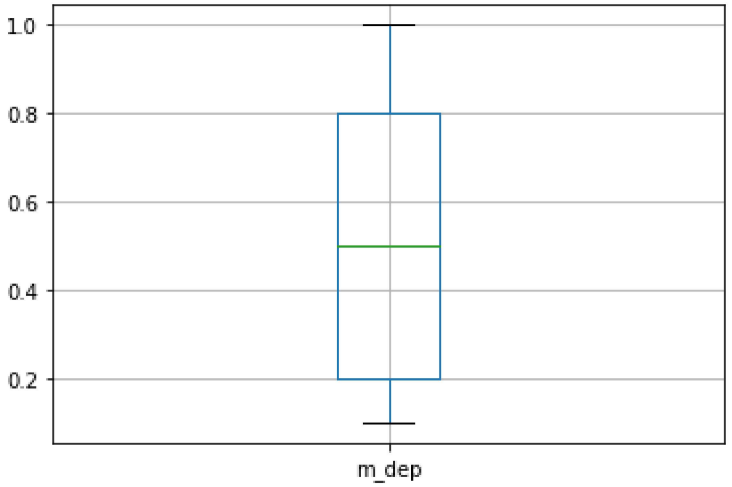

3.0
2.5
2.0
1.5
1.0
0.5
clock_speed

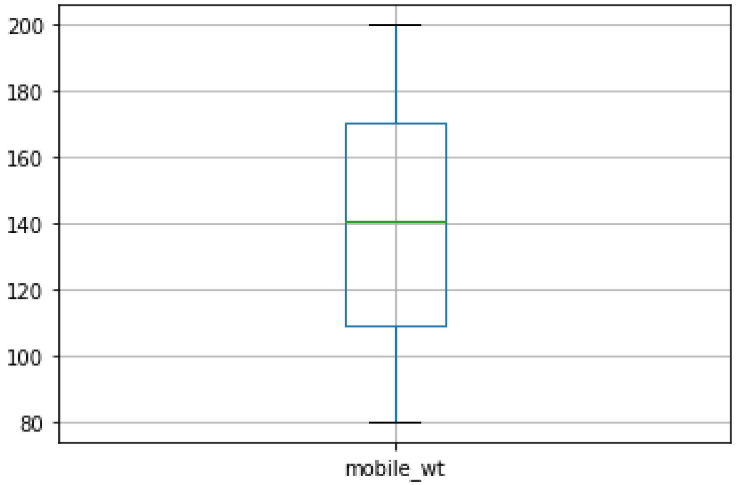

1.0
0.8
0.6
0.4
0.2
0.0
dual_sim

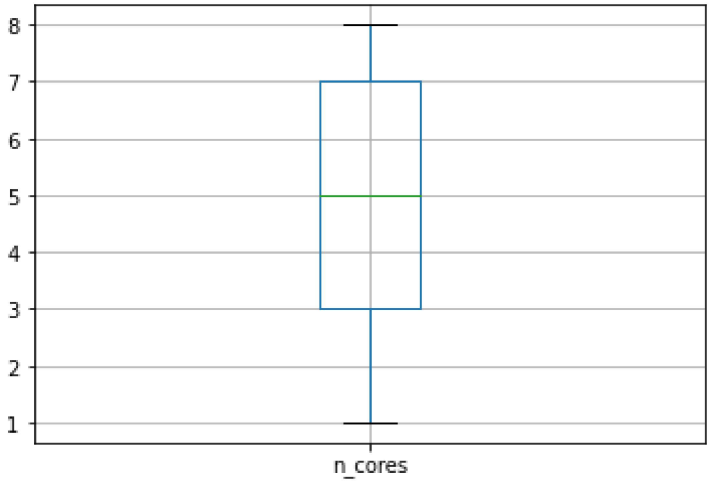

17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
fc

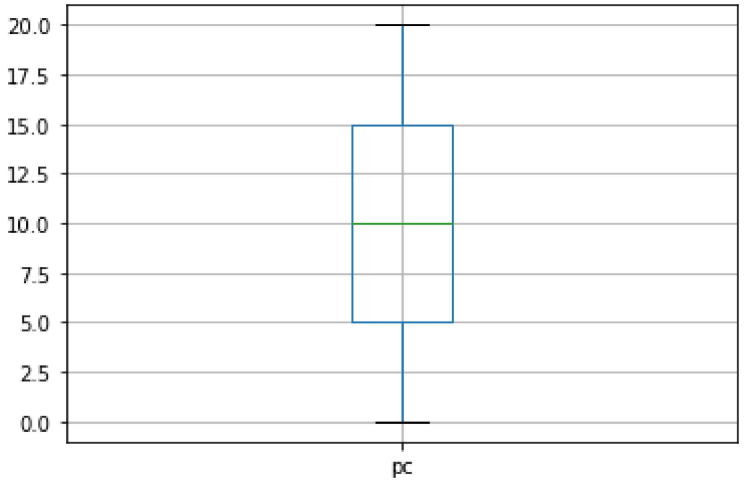

1.0
0.8
0.6
0.4
0.2
0.0
four_g

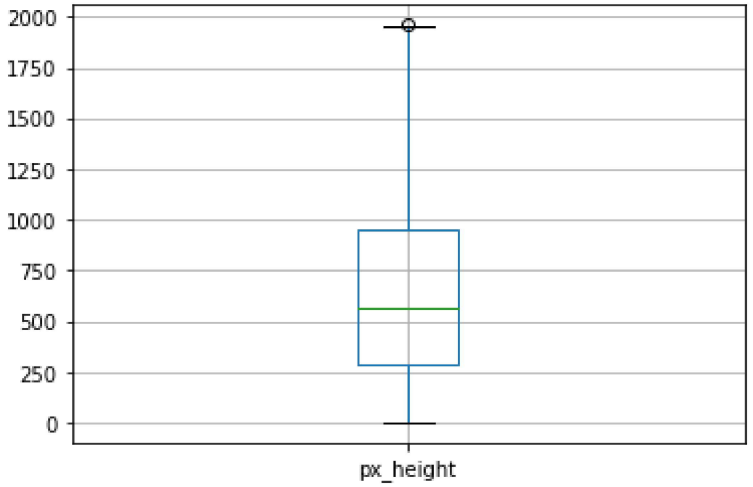

60
50
40
30
20
10
0
int_memory

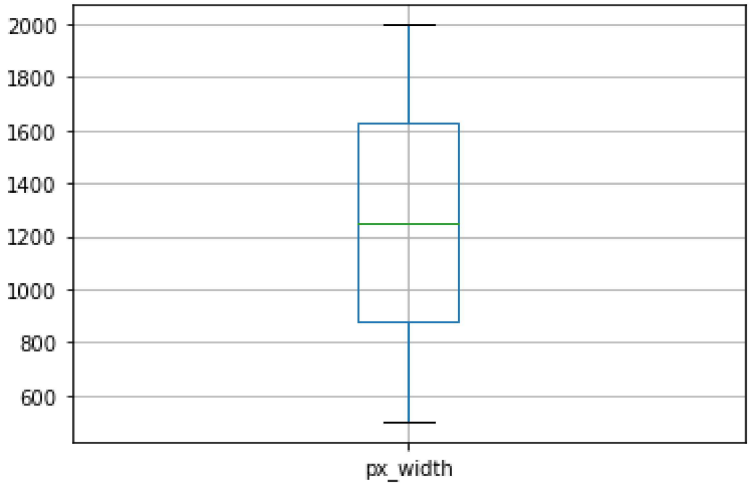

1.0
0.8
0.6
0.4
0.2
m_dep

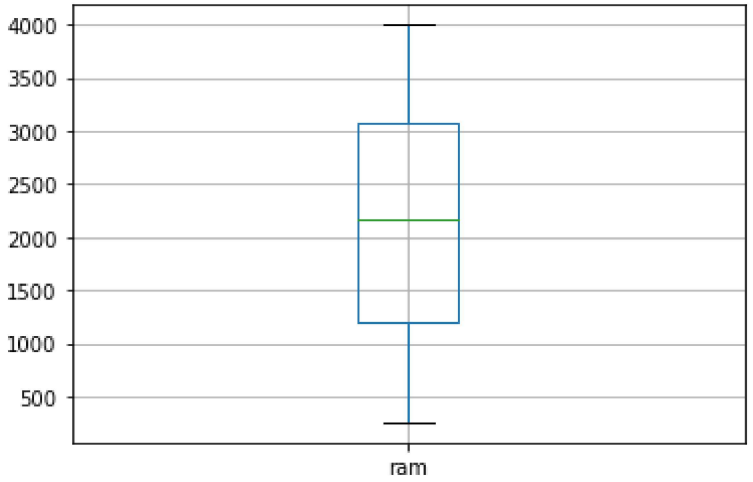

200
180
160
140
120
100
80
mobile_wt

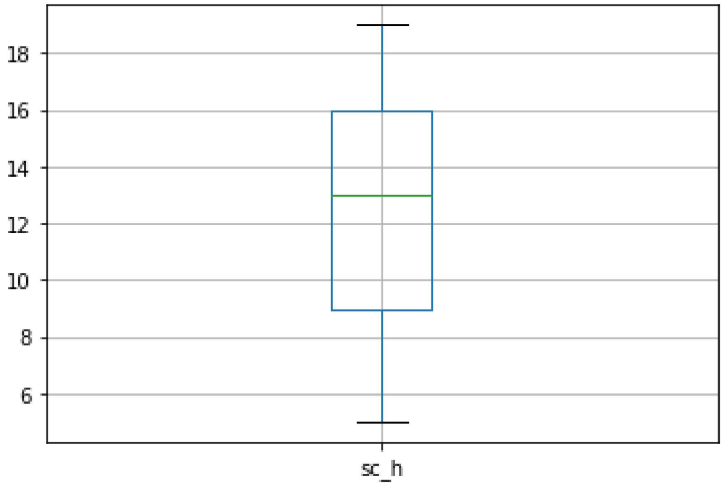

8
7
6
5
4
3
2
1
n_cores

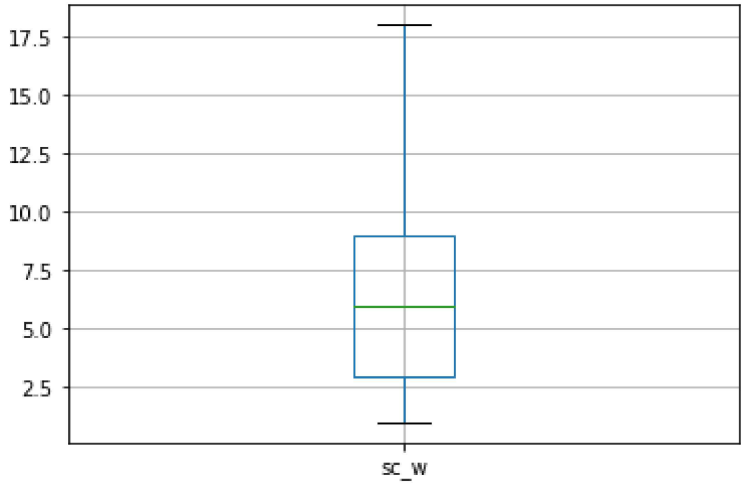

20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
pc

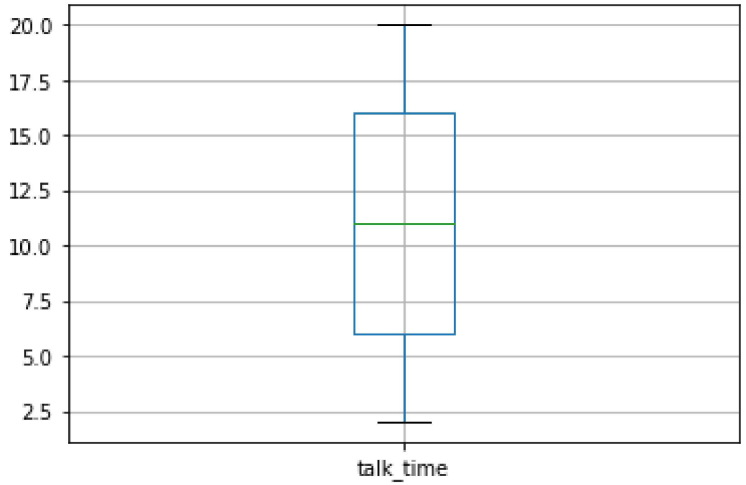

2000
1750
1500
1250
1000
750
500
250
0
px_height

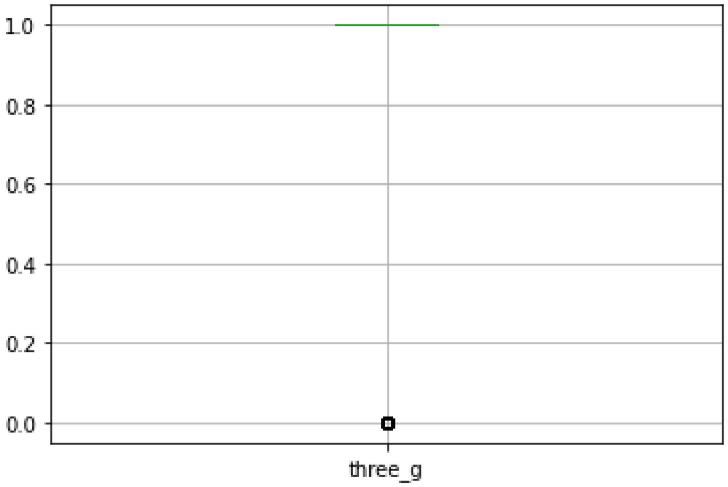

2000
1800
1600
1400
1200
1000
800
600
px_width

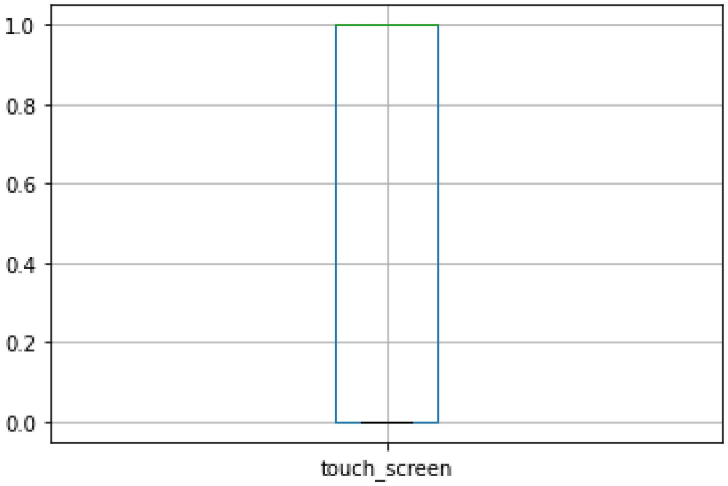

4000
3500
3000
2500
2000
1500
1000
500
ram

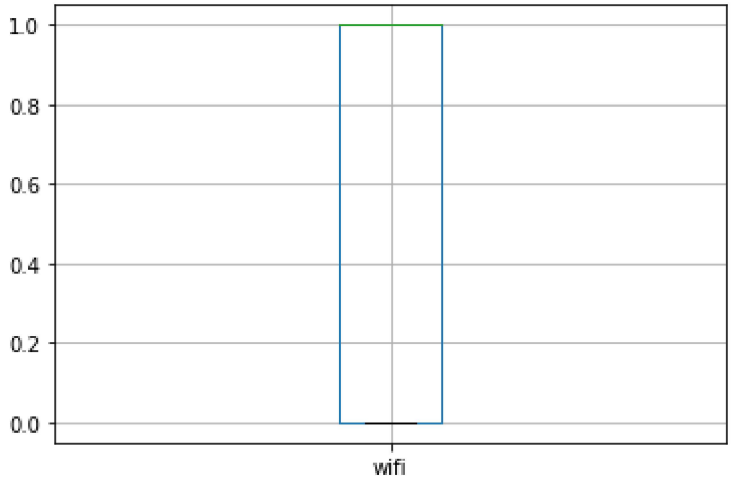

18
16
14
12
10
8
6
sc_h

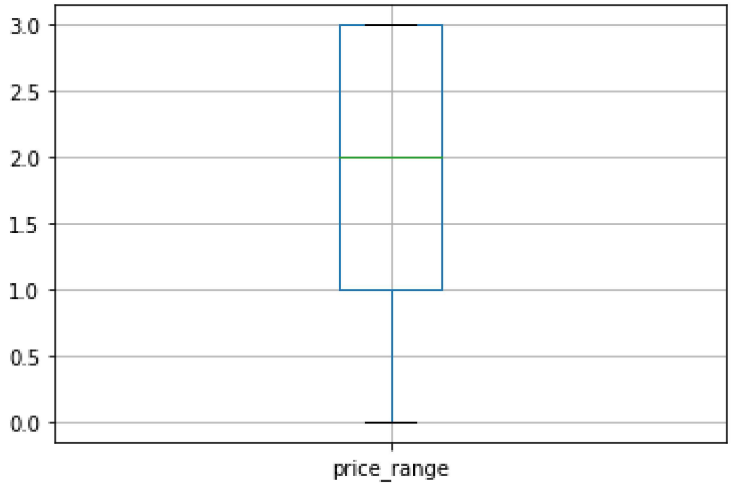

17.5
15.0
12.5
10.0
7.5
5.0
2.5
sc_w


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
talk_time


1.0
0.8
0.6
0.4
0.2
0.0
three_g


1.0
0.8
0.6
0.4
0.2
0.0
touch_screen





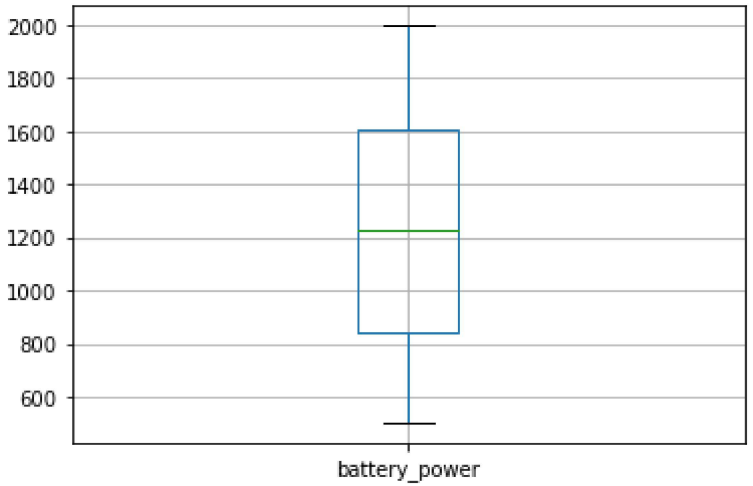
In [7]:
```
# to find the outliers in FC column the below is done

from scipy import stats
import numpy as np
z=np.abs(stats.zscore(Naive['fc']))

thresold = 3
print(np.where(z>3))
```

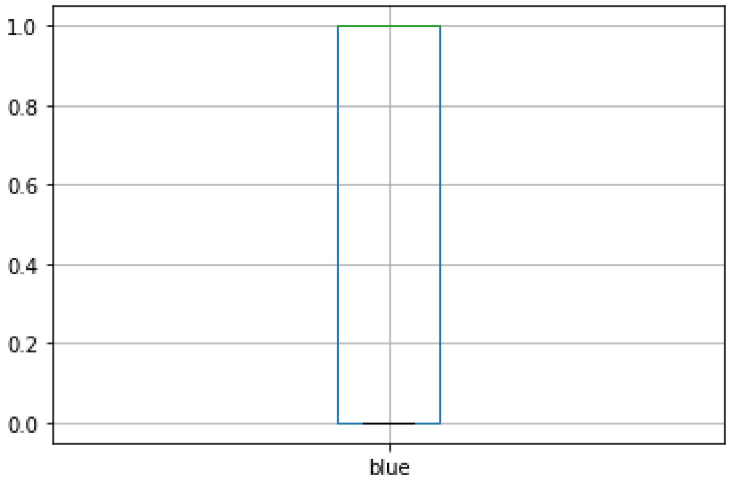
```
(array([  86,  205,  277, 1263, 1282, 1292, 1418, 1544, 1554, 1706, 1708,
       1714], dtype=int64),)
```

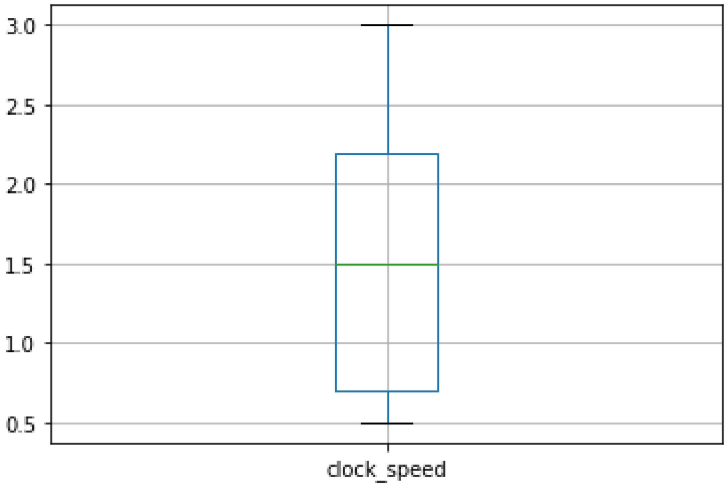
In [8]:
```
print(z[86])

Naive = Naive[(z<3)]
```

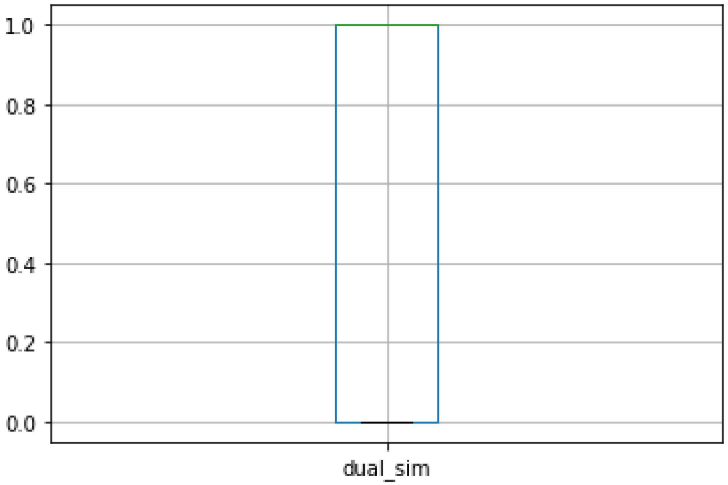
```
3.141382522813342
```

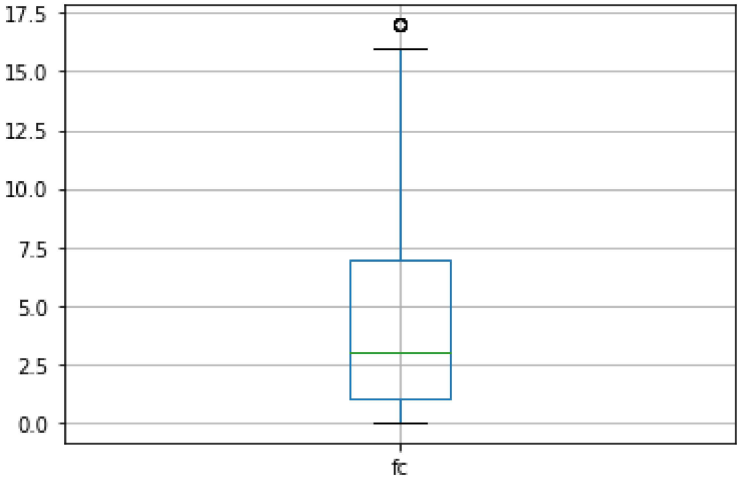
In [9]:
```
for column in Naive:
    plt.figure()
    Naive.boxplot([column])
```

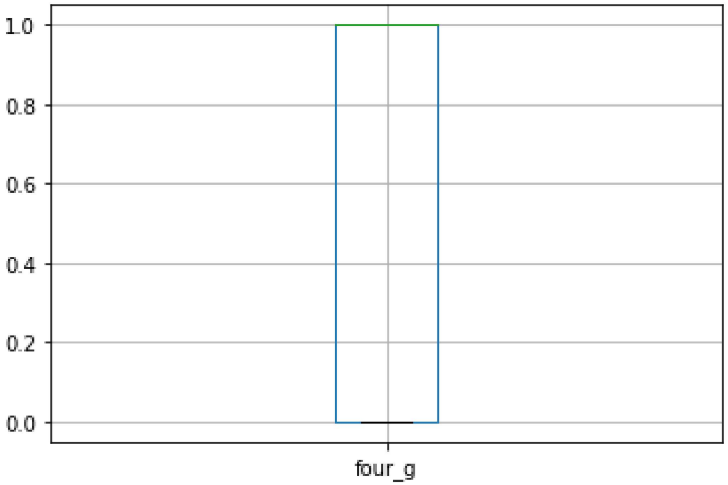


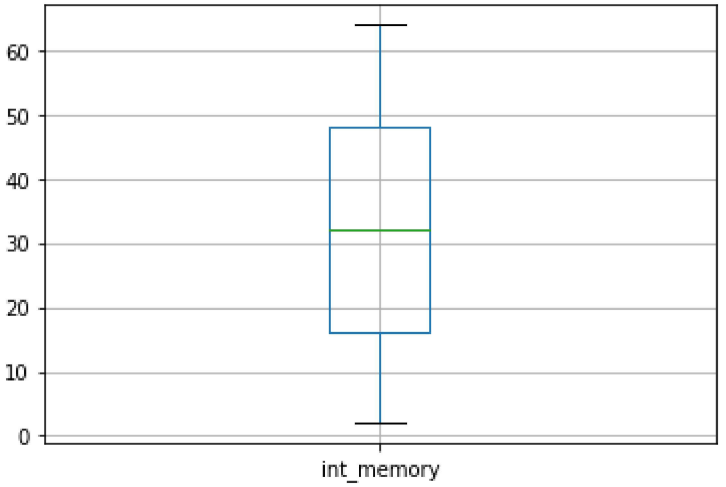


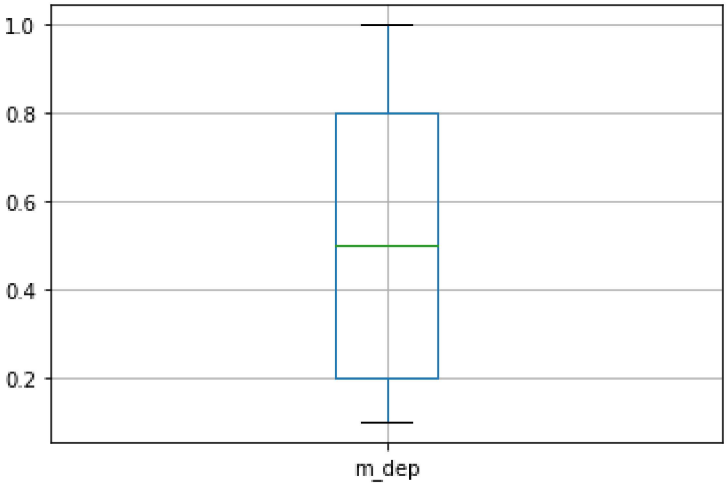

3.0
2.5
2.0
1.5
1.0
0.5
clock_speed

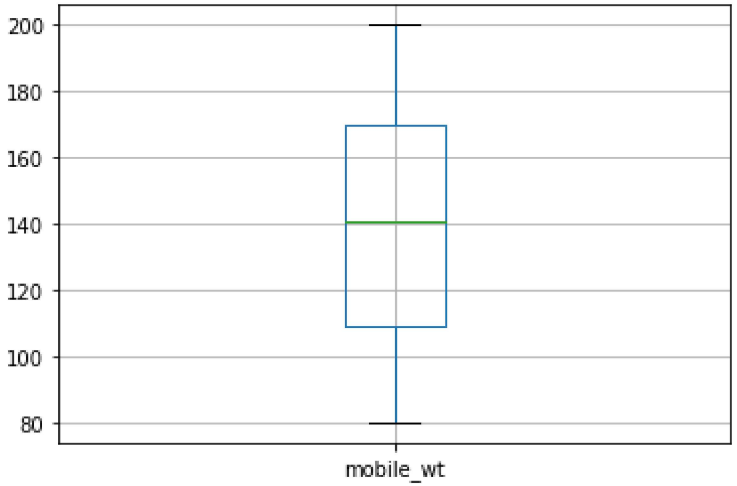

1.0
0.8
0.6
0.4
0.2
0.0
dual_sim

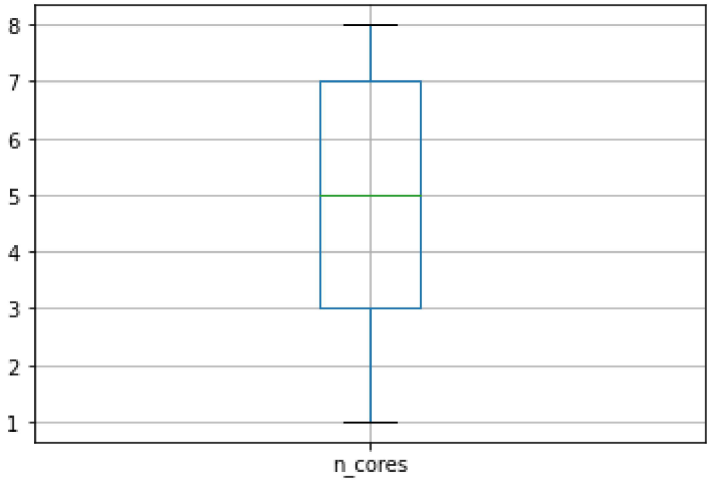

17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
fc

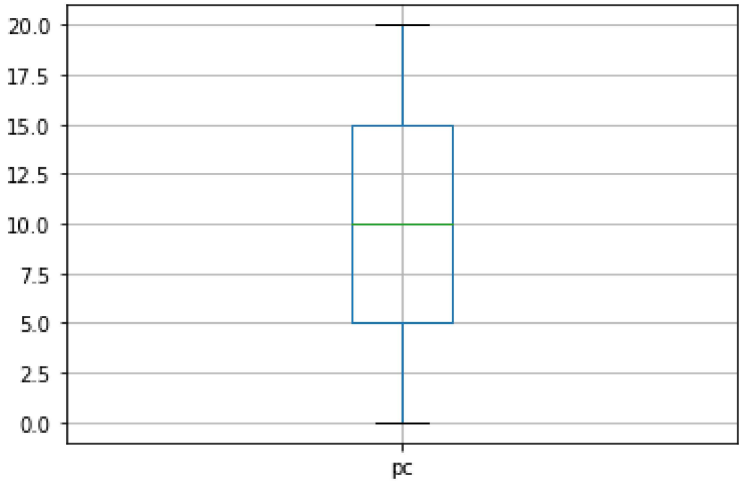

1.0
0.8
0.6
0.4
0.2
0.0
four_g

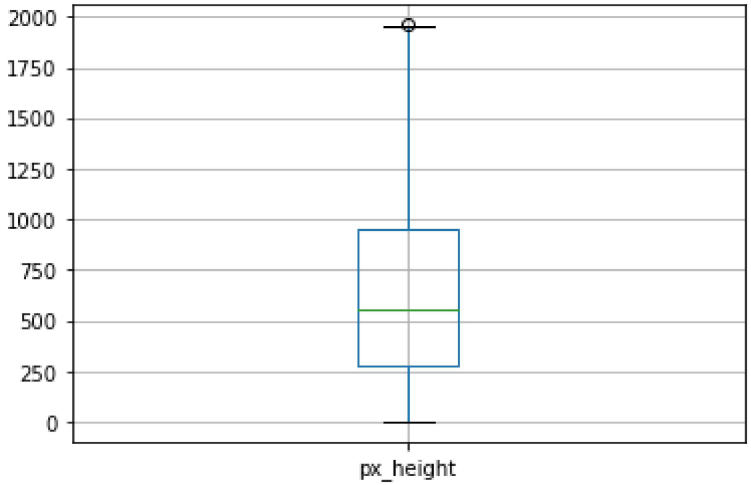

60
50
40
30
20
10
0
int_memory

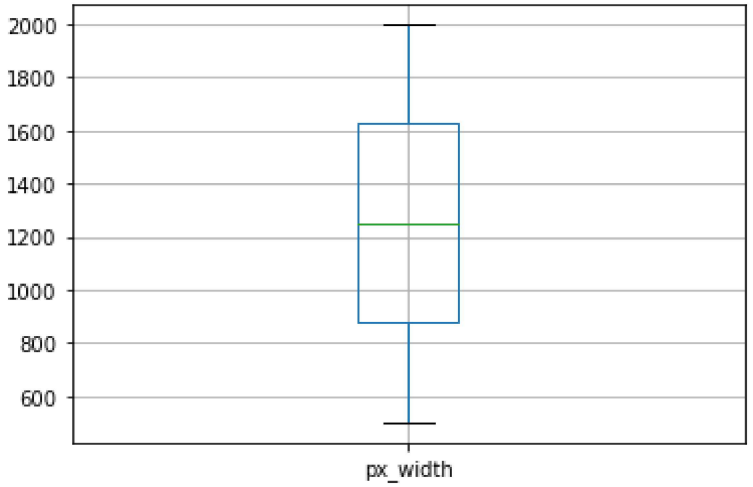

1.0
0.8
0.6
0.4
0.2
m_dep

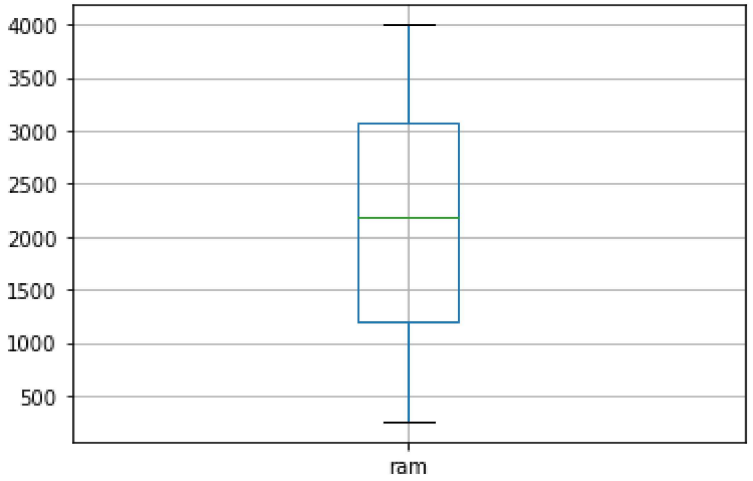

200
180
160
140
120
100
80
mobile_wt

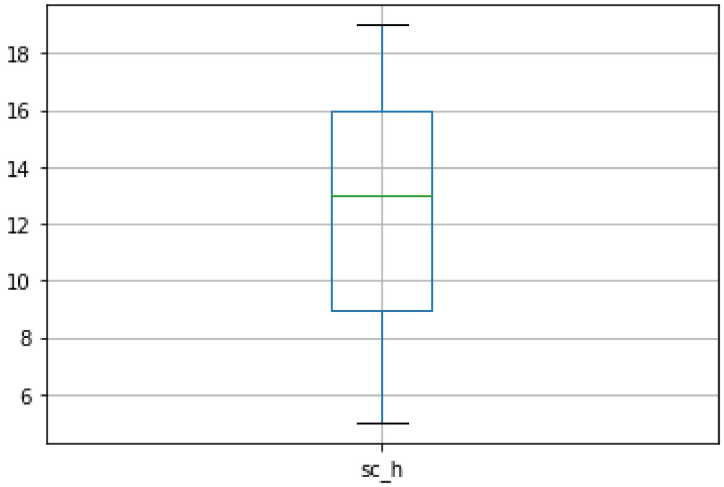

8
7
6
5
4
3
2
1
n_cores

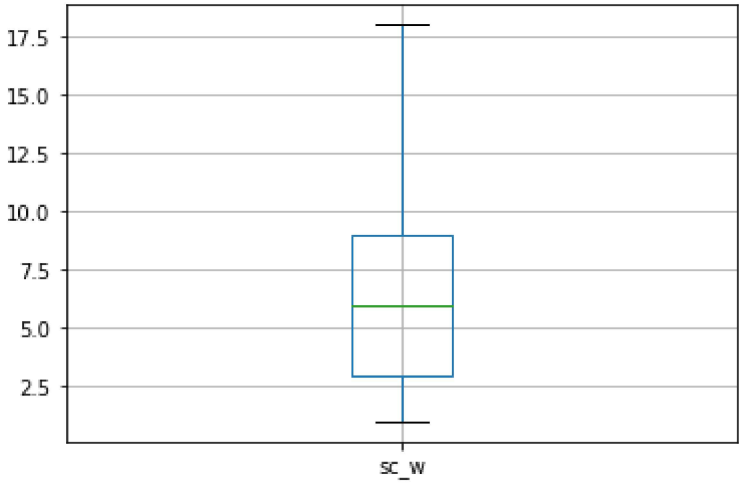

20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
pc

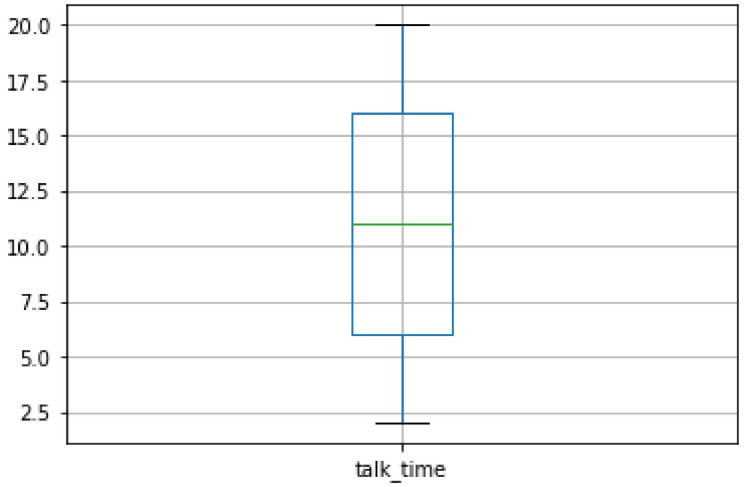

2000
1750
1500
1250
1000
750
500
250
0
px_height

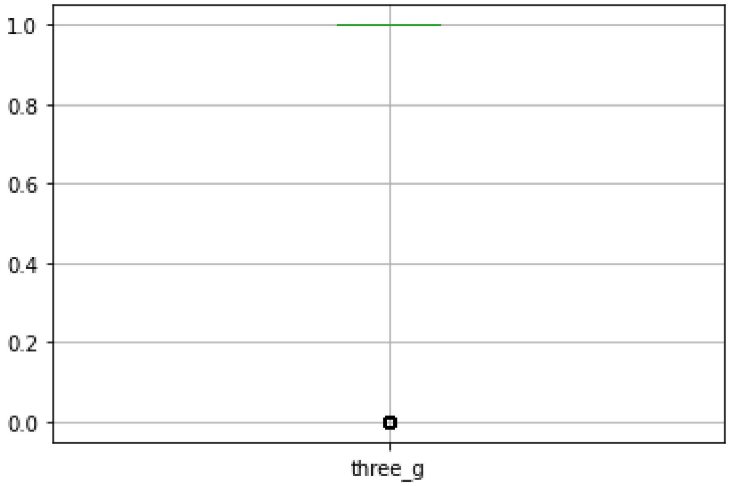

2000
1800
1600
1400
1200
1000
800
600
px_width

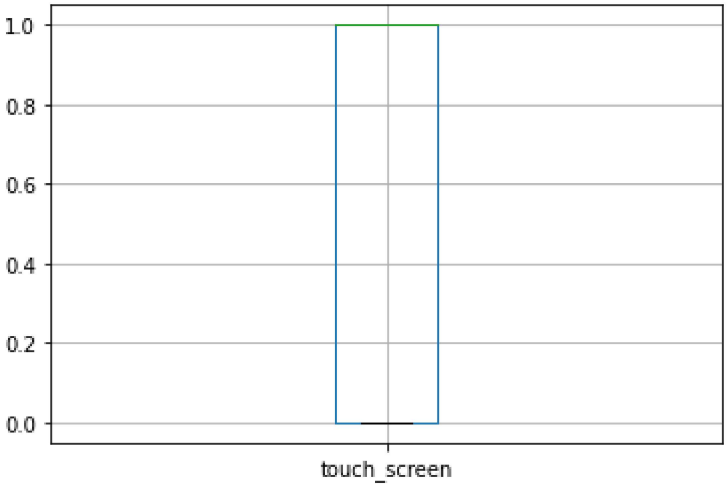

4000
3500
3000
2500
2000
1500
1000
500
ram

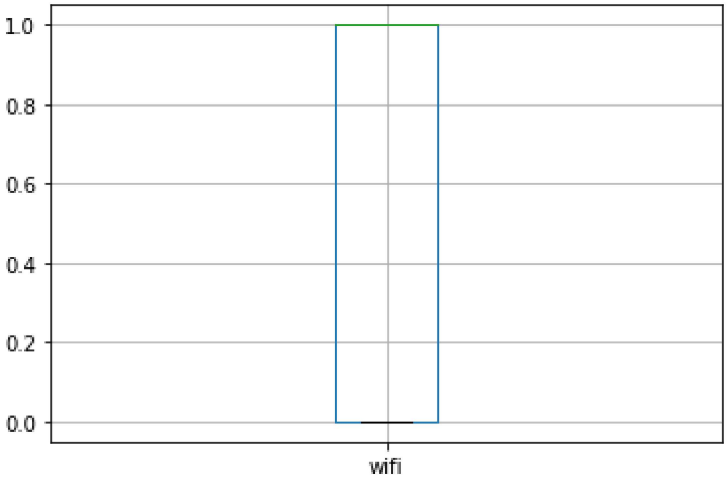

18
16
14
12
10
8
6
sc_h

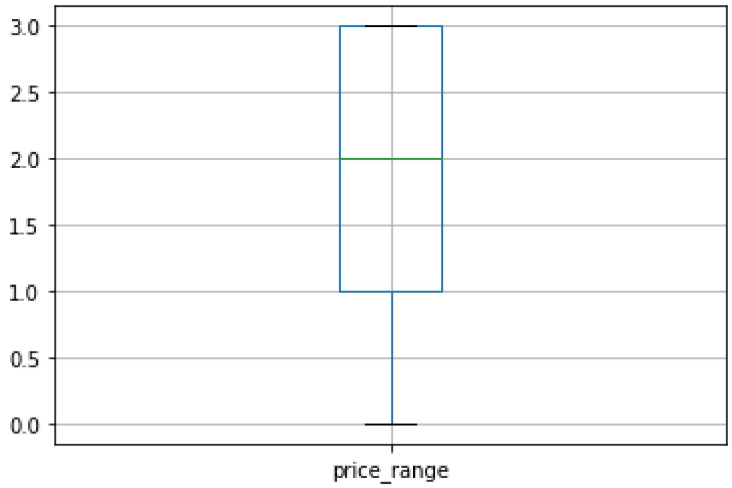

17.5
15.0
12.5
10.0
7.5
5.0
2.5
sc_w


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
talk_time


1.0
0.8
0.6
0.4
0.2
0.0
three_g


1.0
0.8
0.6
0.4
0.2
0.0
touch_screen


1.0
0.8
0.6
0.4
0.2
0.0
wifi



In [10]:
```
y= Naive['price_range']
X=Naive.drop('price_range',axis =1)

from sklearn.model_selection import train_test_split as tts

X_train,X_test, y_train, y_test = tts (X,y,train_size=0.8,
                                       test_size=0.2,random_state=100)
```

In [11]:
```
from sklearn.preprocessing import MinMaxScaler as mms

scaler = mms()
Columns = list(X_train.columns)
X_train[Columns]= scaler.fit_transform(X_train[Columns])
X_train
```

Out[11]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	
689	0.136364	1.0	0.00	0.0	0.294118	1.0	0.887097	0.888889	0.625000	0.714286	0.45	0.567926	0.583445	0.334848	0.7
1819	0.290107	1.0	0.00	1.0	0.176471	1.0	0.903226	0.000000	0.625000	0.571429	0.60	0.325332	0.391856	0.592464	0.3
515	0.889706	0.0	0.08	0.0	0.117647	0.0	0.000000	0.555556	0.083333	0.857143	0.30	0.093973	0.102804	0.727418	0.9
318	0.004679	0.0	0.12	0.0	0.411765	1.0	0.645161	0.222222	0.116667	0.000000	0.40	0.018897	0.038051	0.108765	0.5
1244	0.145053	0.0	0.92	0.0	0.588235	0.0	0.919355	0.777778	0.275000	0.714286	0.65	0.604188	0.592123	0.419027	0.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	
59	0.375668	0.0	0.36	1.0	0.117647	1.0	0.741935	1.000000	0.400000	0.571429	0.20	0.063841	0.122163	0.709246	0.7
393	0.584225	1.0	0.24	0.0	0.176471	1.0	0.112903	0.888889	0.541667	0.857143	0.95	0.241573	0.150868	0.965526	1.0
88	0.111631	0.0	0.00	1.0	0.000000	0.0	0.016129	0.000000	0.625000	0.571429	0.65	0.339122	0.469292	0.665420	0.2
872	0.045455	1.0	0.00	1.0	0.588235	1.0	0.032258	0.777778	0.750000	0.285714	0.95	0.178243	0.439920	0.559861	0.2
1703	0.294786	0.0	0.36	1.0	0.470588	0.0	0.403226	0.444444	0.091667	0.428571	0.45	0.384065	0.688919	0.088455	0.0

1445 rows × 20 columns

In [14]:
```
X_test[Columns]=scaler.transform(X_test[Columns])
X_test
```

Out[14]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram
150	-0.334740	1.0	-0.200	0.0	0.003460	1.0	-0.018210	-0.111111	-0.662014	-0.061224	0.0425	-0.000985	-0.333645	-0.068264
504	-0.334375	1.0	-0.056	0.0	0.000000	0.0	-0.025234	0.135802	-0.663958	-0.122449	0.0000	-0.000882	-0.333617	-0.068389
991	-0.334733	1.0	-0.024	1.0	0.044983	0.0	-0.020291	0.382716	-0.659236	-0.020408	0.0450	-0.000935	-0.333349	-0.068297
34	-0.334829	1.0	0.152	0.0	0.000000	0.0	-0.027055	0.629630	-0.661319	0.000000	0.0075	-0.000941	-0.333609	-0.068341
563	-0.334541	0.0	-0.200	0.0	0.000000	0.0	-0.031998	-0.111111	-0.662292	-0.040816	0.0100	-0.000865	-0.333507	-0.068344
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
954	-0.334705	1.0	-0.168	0.0	0.000000	1.0	-0.020552	1.000000	-0.661319	0.000000	0.0350	-0.000684	-0.333195	-0.068326
387	-0.334867	0.0	-0.200	0.0	0.048443	0.0	-0.026795	1.000000	-0.662083	0.000000	0.0375	-0.000929	-0.333153	-0.068262
647	-0.334330	1.0	0.152	1.0	0.044983	1.0	-0.026275	0.876543	-0.659097	-0.142857	0.0350	-0.000944	-0.333766	-0.068295
1756	-0.334284	0.0	0.024	1.0	0.006920	0.0	-0.017950	-0.111111	-0.659653	-0.040816	0.0150	-0.000688	-0.333202	-0.068175
1166	-0.334671	1.0	0.008	1.0	0.000000	1.0	-0.025754	-0.111111	-0.660625	-0.081633	0.0025	-0.001017	-0.333669	-0.068335

362 rows × 20 columns

In [15]:
```
#Simple Bernoulli
from sklearn.naive_bayes import BernoulliNB

#instantiate bernoullie NB object
bnb = BernoulliNB()

#fit
bnb.fit(X_train,y_train)

#predict class
y_pred_class = bnb.predict(X_test)

#predicted probability
y_pred_proba = bnb.predict_proba(X_test)

#metrics
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred_class)
```

Out[15]: 0.24861878453038674

In [16]:
```
# simple multinomial
from sklearn.naive_bayes import MultinomialNB

#instantiate bernoullie NB object
nb = MultinomialNB()


#fit
nb.fit(X_train,y_train)

#predict class
y_pred_class = nb.predict(X_test)

#predicted probability
y_pred_proba = nb.predict_proba(X_test)

#metrics
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred_class)
```

Out[16]: 0.23756906077348067

In [17]:
```
# simple multinomial
from sklearn.naive_bayes import GaussianNB

#instantiate bernoullie NB object
gnb = GaussianNB()


#fit
gnb.fit(X_train,y_train)

#predict class
y_pred_class = gnb.predict(X_test)

#predicted probability
y_pred_proba = gnb.predict_proba(X_test)

#metrics
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred_class)
```

Out[17]:
```
0.2430939226519337
```

In [18]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
print("PRECISION SCORE :",precision_score(y_test,y_pred_class,average='weighted'))
print("RECALL SCORE :",recall_score(y_test,y_pred_class,average='weighted'))
print("F1 score :",f1_score(y_test,y_pred_class,average='weighted'))
```

```
PRECISION SCORE : 0.05909465523030432
RECALL SCORE : 0.2430939226519337
F1 score : 0.09507673419275628
```

In [19]:
```
y_pred_class
```

Out[19]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [20]:
```
y_pred_proba
```

Out[20]:
```
array([[9.95309208e-01, 4.68946729e-03, 1.32445264e-06, 7.52988541e-18],
       [9.93705730e-01, 6.29274849e-03, 1.52153720e-06, 7.22069086e-18],
       [9.93133685e-01, 6.86400694e-03, 2.30845448e-06, 1.10180878e-17],
       ...,
       [9.91882148e-01, 8.11633824e-03, 1.51366831e-06, 1.24838214e-17],
       [9.92185680e-01, 7.81249754e-03, 1.82267398e-06, 9.89099122e-18],
       [9.95010543e-01, 4.98753801e-03, 1.91939060e-06, 1.09704045e-17]])
```

In [21]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn import metrics

gnb= GaussianNB()

lm = LinearRegression()
lm.fit(X_train,y_train)

for features in np.arange(1,(X_train.shape[1])):
    rfe = RFE(lm, features)
    rfe = rfe.fit(X_train,y_train)

    X_train_subset = X_train[list(X_train.columns[rfe.support_])]
    X_test_subset = X_test[list(X_train.columns[rfe.support_])]

    gnb.fit(X_train_subset,y_train)

    y_pred_class = gnb.predict(X_test_subset)

    y_pred_proba = gnb.predict_proba(X_test_subset)

    print("for",features,"features being selected the accuracy is ", metrics.accuracy_score(y_test, y_pred_class))
```

```
for 1 features being selected the accuracy is  0.2430939226519337
for 2 features being selected the accuracy is  0.2430939226519337
for 3 features being selected the accuracy is  0.2430939226519337
for 4 features being selected the accuracy is  0.2430939226519337
for 5 features being selected the accuracy is  0.2430939226519337
for 6 features being selected the accuracy is  0.2430939226519337
```

```
for 7 features being selected the accuracy is  0.2430939226519337
for 8 features being selected the accuracy is  0.2430939226519337
for 9 features being selected the accuracy is  0.2430939226519337
for 10 features being selected the accuracy is  0.2430939226519337
for 11 features being selected the accuracy is  0.2430939226519337
for 12 features being selected the accuracy is  0.2430939226519337
for 13 features being selected the accuracy is  0.2430939226519337
for 14 features being selected the accuracy is  0.2430939226519337
for 15 features being selected the accuracy is  0.2430939226519337
for 16 features being selected the accuracy is  0.2430939226519337
for 17 features being selected the accuracy is  0.2430939226519337
for 18 features being selected the accuracy is  0.2430939226519337
for 19 features being selected the accuracy is  0.2430939226519337
```

In [22]:
```
rfe = RFE(lm, 10)
rfe = rfe.fit(X_train,y_train)

X_train_subset = X_train[list(X_train.columns[rfe.support_])]
X_test_subset = X_test[list(X_train.columns[rfe.support_])]
```

In [23]:
```
from sklearn.model_selection import GridSearchCV

gnb= GaussianNB()

param_grid = {'var_smoothing' : [0.000000001,0.00000001,0.0000001,0.000001,0.00001,0.0001]}

nb_gscv = GridSearchCV(gnb, param_grid,
                       scoring= "accuracy",
                       cv=7,
                       n_jobs = -1,
                       verbose= 20)
nb_gscv.fit(X_train_subset, y_train)
```

```
Fitting 7 folds for each of 6 candidates, totalling 42 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    6.2s
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    6.2s
[Parallel(n_jobs=-1)]: Done   3 tasks      | elapsed:    6.2s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    6.2s
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:    6.3s
[Parallel(n_jobs=-1)]: Done   6 tasks      | elapsed:    6.3s
[Parallel(n_jobs=-1)]: Done   7 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done   8 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:    6.4s
```

```
[Parallel(n_jobs=-1)]: Done  11 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done  12 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done  13 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done  15 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  19 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  20 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  22 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  23 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  27 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done  30 out of  42 | elapsed:    6.6s remaining:    2.6s
[Parallel(n_jobs=-1)]: Done  33 out of  42 | elapsed:    6.6s remaining:    1.7s
[Parallel(n_jobs=-1)]: Done  36 out of  42 | elapsed:    6.6s remaining:    1.0s
[Parallel(n_jobs=-1)]: Done  39 out of  42 | elapsed:    6.6s remaining:    0.4s
[Parallel(n_jobs=-1)]: Done  42 out of  42 | elapsed:    6.6s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  42 out of  42 | elapsed:    6.6s finished
```

Out[23]:
```
GridSearchCV(cv=7, estimator=GaussianNB(), n_jobs=-1,
             param_grid={'var_smoothing': [1e-09, 1e-08, 1e-07, 1e-06, 1e-05,
                                           0.0001]},
             scoring='accuracy', verbose=20)
```

In [24]:
```
best_mod = nb_gscv.best_estimator_
best_mod
```

Out[24]:
```
GaussianNB()
```

In [25]:
```
nb_gscv.best_score_
```

Out[25]:
```
0.7993125489959596
```

In [26]:
```
from sklearn.metrics import accuracy_score

y_pred = best_mod.predict(X_test_subset)
print(accuracy_score(y_test,y_pred))
```

```
0.2430939226519337
```

In [ ]: