In [2]:
```
import pickle
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import ConfusionMatrixDisplay , classification_report
from sklearn.model_selection import train_test_split
```

In [4]:
```
df_train = pd.read_csv("train.csv")
df_train.T
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	...
battery_power	842.0	1021.0	563.0	615.0	1821.0	1859.0	1821.0	1954.0	1445.0	509.0	...
blue	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	...
clock_speed	2.2	0.5	0.5	2.5	1.2	0.5	1.7	0.5	0.5	0.6	...
dual_sim	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	...
fc	1.0	0.0	2.0	0.0	13.0	3.0	4.0	0.0	0.0	2.0	...
four_g	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	...
int_memory	7.0	53.0	41.0	10.0	44.0	22.0	10.0	24.0	53.0	9.0	...
m_dep	0.6	0.7	0.9	0.8	0.6	0.7	0.8	0.8	0.7	0.1	...
mobile_wt	188.0	136.0	145.0	131.0	141.0	164.0	139.0	187.0	174.0	93.0	...
n_cores	2.0	3.0	5.0	6.0	2.0	1.0	8.0	4.0	7.0	5.0	...
pc	2.0	6.0	6.0	9.0	14.0	7.0	10.0	0.0	14.0	15.0	...
px_height	20.0	905.0	1263.0	1216.0	1208.0	1004.0	381.0	512.0	386.0	1137.0	...
px_width	756.0	1988.0	1716.0	1786.0	1212.0	1654.0	1018.0	1149.0	836.0	1224.0	...
ram	2549.0	2631.0	2603.0	2769.0	1411.0	1067.0	3220.0	700.0	1099.0	513.0	...
sc_h	9.0	17.0	11.0	16.0	8.0	17.0	13.0	16.0	17.0	19.0	...
sc_w	7.0	3.0	2.0	8.0	2.0	1.0	8.0	3.0	1.0	10.0	...
talk_time	19.0	7.0	9.0	11.0	15.0	10.0	18.0	5.0	20.0	12.0	...
three_g	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...
touch_screen	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	...
wifi	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	...
price_range	1.0	2.0	2.0	2.0	1.0	1.0	3.0	0.0	0.0	0.0	...

21 rows × 2000 columns

In [5]:
```
df_train.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: 328.3 KB
```

In [8]:
```
df_train.describe()
```

Out[8]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memo
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.0000(
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.0465(
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.1457
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.0000(
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.0000(
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.0000(
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.0000(
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.0000(

8 rows × 21 columns

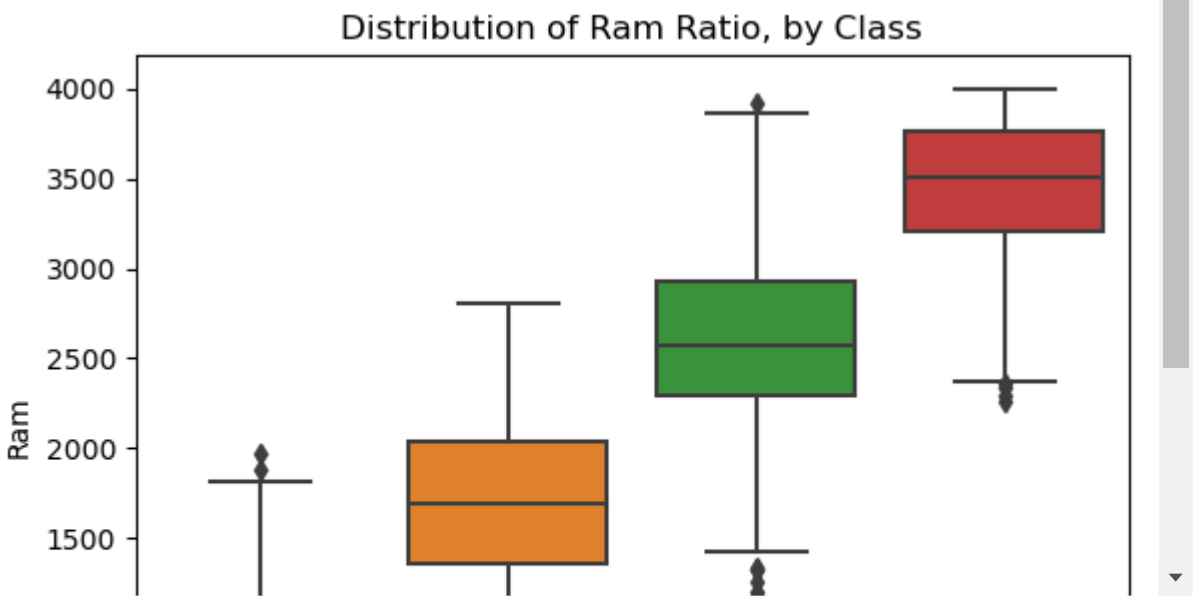
In [9]:
```
df_train.nunique()
```

Out[9]:
```
battery_power    1094
blue                2
clock_speed        26
dual_sim            2
fc                 20
four_g              2
int_memory         63
m_dep              10
mobile_wt         121
n_cores             8
pc                 21
px_height        1137
px_width         1109
ram              1562
sc_h               15
sc_w               19
talk_time          19
three_g             2
touch_screen        2
wifi                2
price_range         4
dtype: int64
```

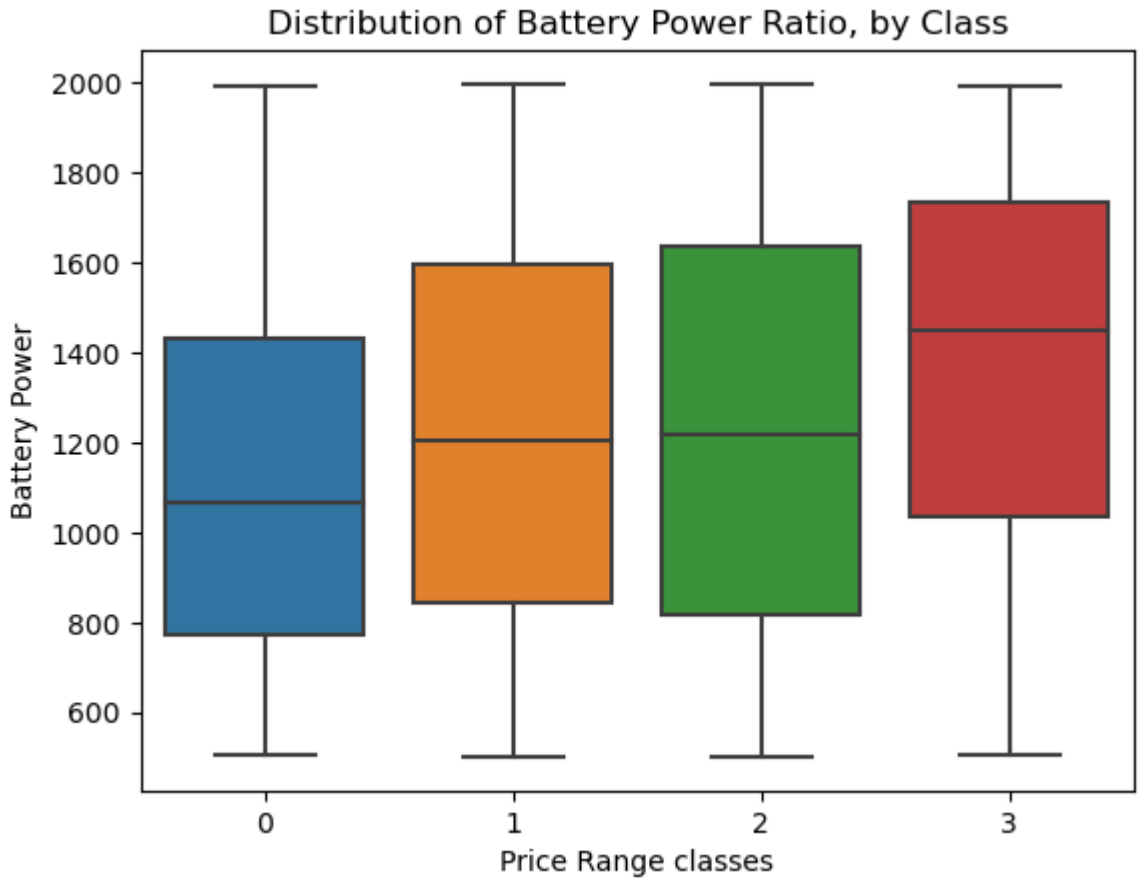
In [11]:
```
df_train['price_range'].value_counts(normalize=True).plot(kind = 'bar')
plt.xlabel("Price Range classes")
plt.ylabel("Frequency")
plt.title("Classes balance");
```

In [14]:
```
sns.boxplot(x= 'price_range', y='ram', data=df_train)
plt.xlabel("Price Range classes")
plt.ylabel("Ram")
plt.title("Distribution of Ram Ratio, by Class")
```

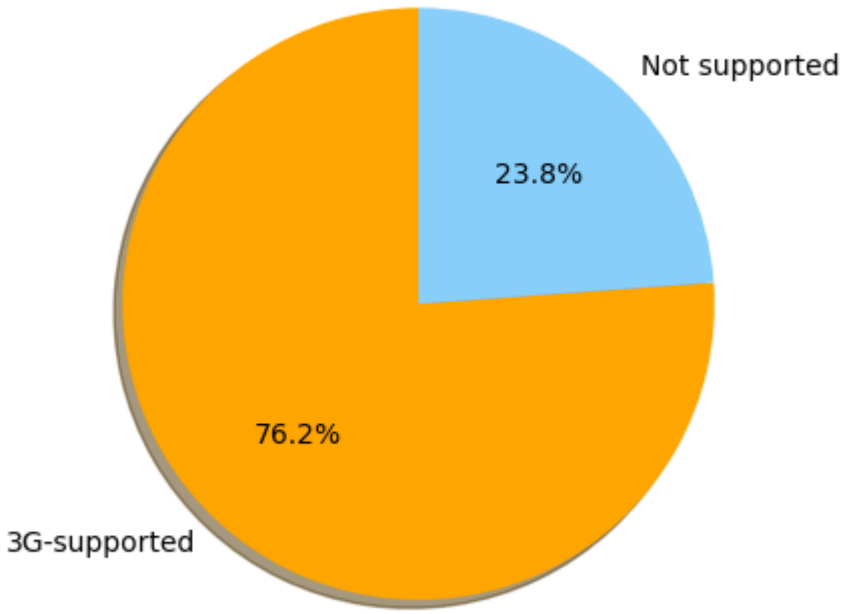
Out[14]: Text(0.5, 1.0, 'Distribution of Ram Ratio, by Class')



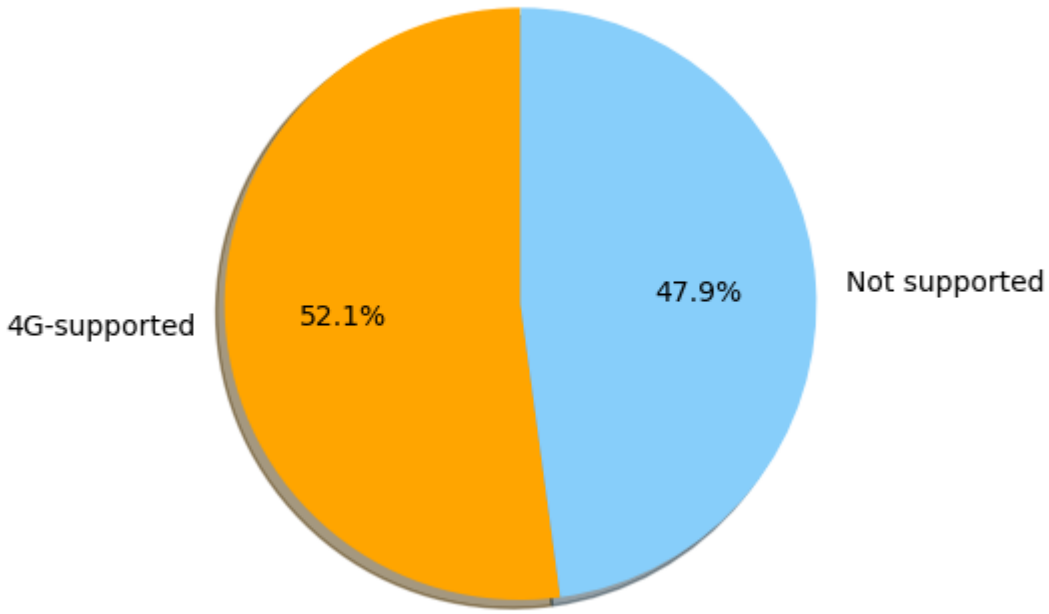
In [16]:
```
sns.boxplot(x= 'price_range', y='battery_power', data=df_train)
plt.xlabel("Price Range classes")
plt.ylabel("Battery Power")
plt.title("Distribution of Battery Power Ratio, by Class");
```



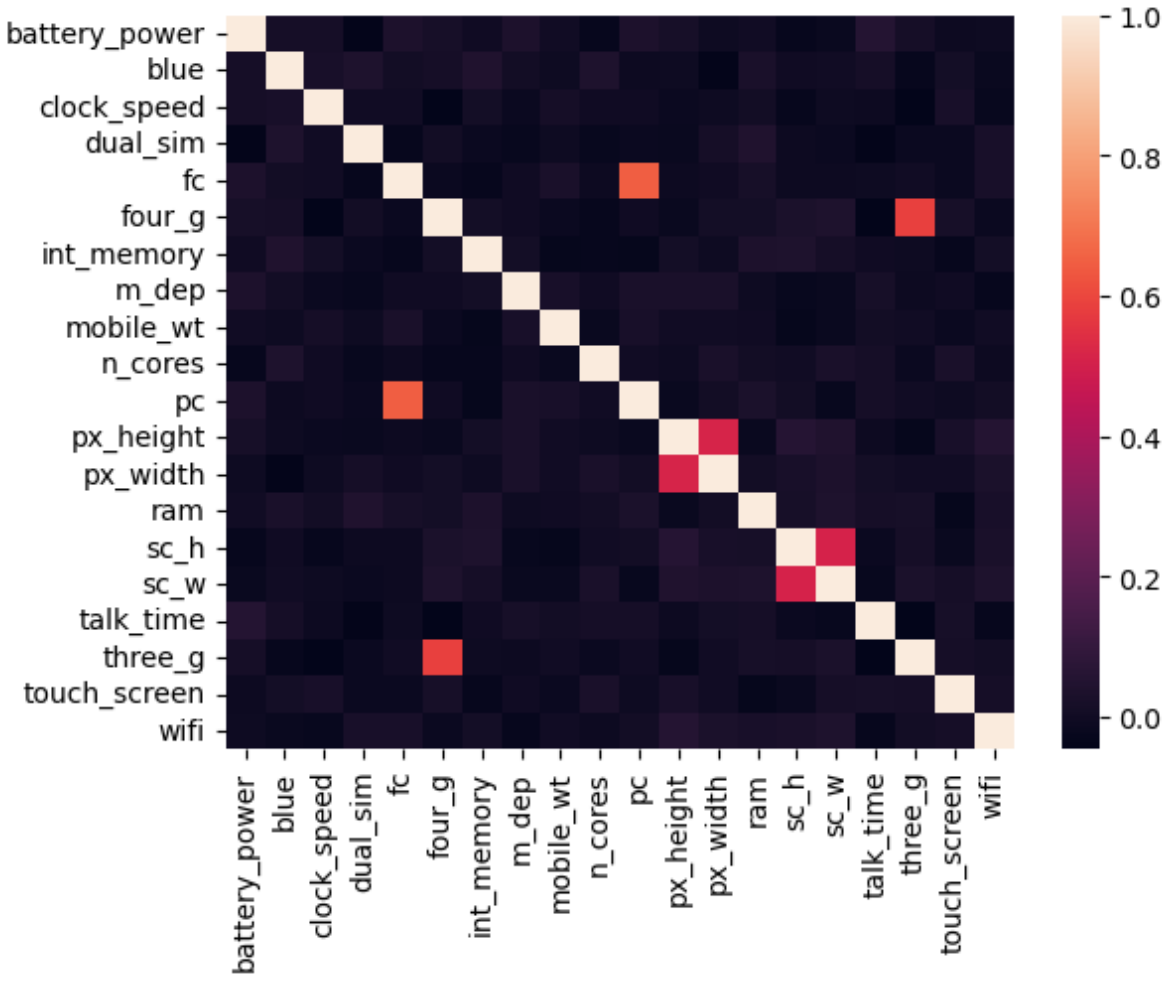
In [20]:
```
labels = ["3G-supported",'Not supported']
values = df_train['three_g'].value_counts().values
fig, ax = plt.subplots()
colors = ['orange', 'lightskyblue']
ax.pie(values, labels=labels, autopct='%1.1f%%',shadow=True,startangle=90,colo
plt.show();
```



In [21]:
```
labels = ["4G-supported",'Not supported']
values = df_train['four_g'].value_counts().values
fig1, ax1 = plt.subplots()
colors = ['orange', 'lightskyblue']
ax1.pie(values, labels=labels, autopct='%1.1f%%',shadow=True,startangle=90,colo
plt.show();
```



In [22]:
```
corr = df_train.drop(columns='price_range').corr()
sns.heatmap(corr);
```



In [23]:
```
target = 'price_range'
X = df_train.drop(columns=[target])
y = df_train[target]
print(f"X shape {X.shape}")
print(f"y Shape {y.shape}")
```

```
X shape (2000, 20)
y Shape (2000,)
```

In [25]:
```
scalar = MinMaxScaler()
```

In [26]:
```
features = X.columns
X = scalar.fit_transform(X)
```

In [27]:
```
print(X)
```

```
[[0.22778891 0.         0.68       ... 0.         0.         1.        ]
 [0.34736139 1.         0.         ... 1.         1.         0.        ]
 [0.04141617 1.         0.         ... 1.         1.         0.        ]
 ...
 [0.94188377 0.         0.16       ... 1.         1.         0.        ]
 [0.6753507  0.         0.16       ... 1.         1.         1.        ]
 [0.00601202 1.         0.6        ... 1.         1.         1.        ]]
```

In [28]:
```
X_train , X_test , y_train , y_test = train_test_split(X, y , test_size=0.2 , r
```

In [29]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape :" , X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (1600, 20)
y_train shape: (1600,)
X_test shape : (400, 20)
y_test shape: (400,)
```

In [30]:
```
model_lr = LogisticRegression(max_iter=1000)
```

In [31]:
```
model_lr.fit(X_train,y_train)
```

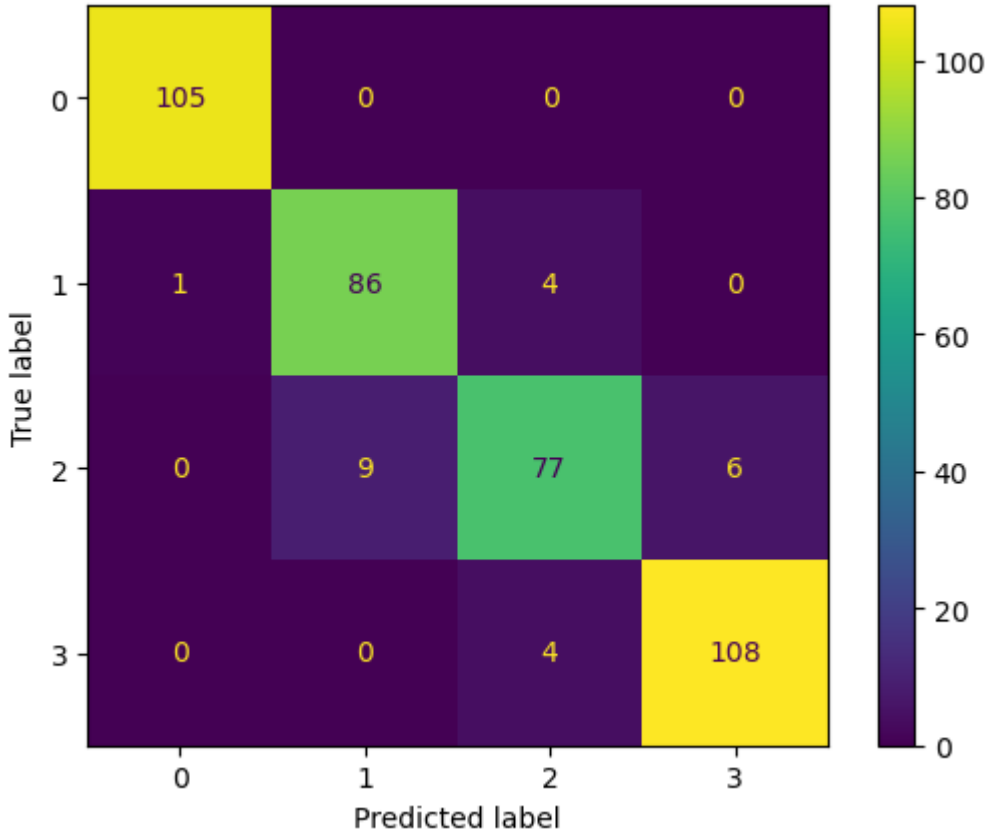
Out[31]:
```
LogisticRegression(max_iter=1000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [32]:
```
training_acc_lr= model_lr.score(X_train , y_train)
print(f"Training accuracy: {training_acc_lr}")
```

```
Training accuracy: 0.938125
```

In [33]:
```
ConfusionMatrixDisplay.from_estimator(model_lr, X_test, y_test);
```



In [34]:
```
model_svc= SVC()
```

In [36]:
```
model_svc.fit(X_train, y_train)
```

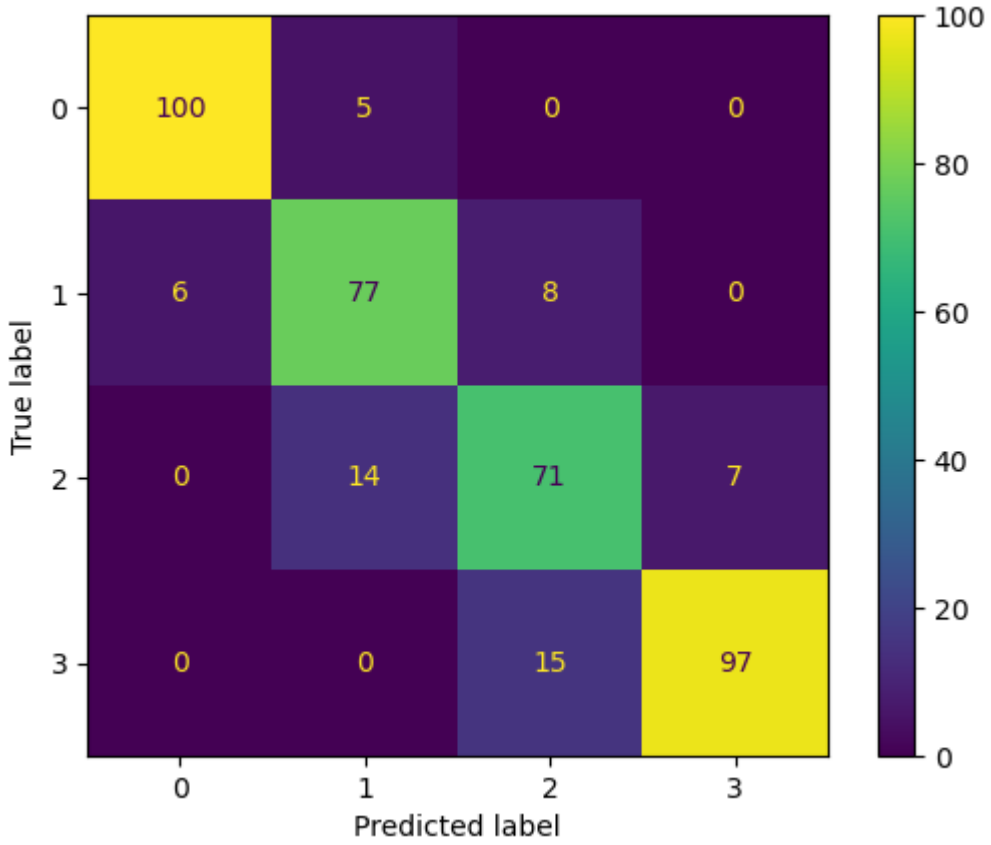
Out[36]:
```
SVC()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [37]:
```
training_acc_svc = model_svc.score(X_train , y_train)
print(f"Testing accuracy: {training_acc_svc}")
```

```
Testing accuracy: 0.969375
```

In [38]:
```
ConfusionMatrixDisplay.from_estimator(model_svc, X_test, y_test);
```



In [51]:
```
models = pd.DataFrame({

    "Models": ["Logestic Regression" , "SVM"],
    "Score":["testing_acc_lr" , "testing_acc_svc"]

})
models.sort_values(by="Score" , ascending=False)
```

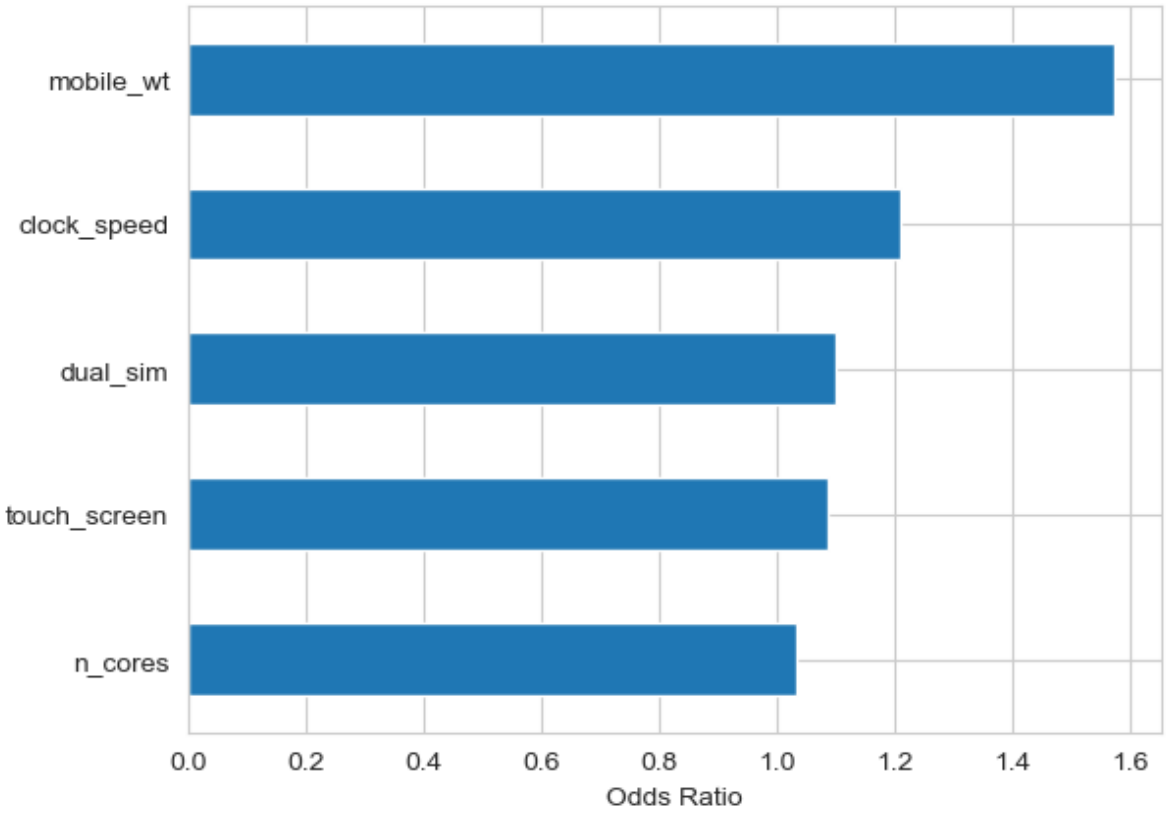
Out[51]:

	Models	Score
1	SVM	testing_acc_svc
0	Logestic Regression	testing_acc_lr

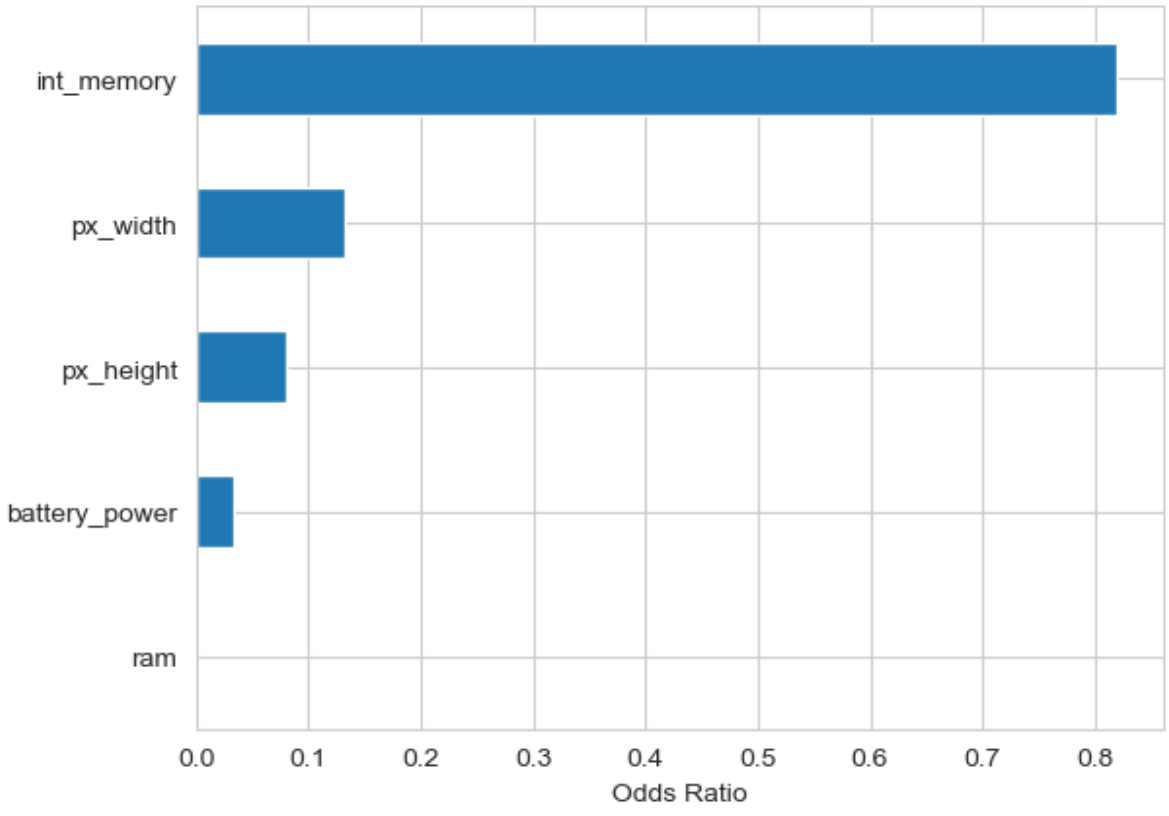
In [58]:
```
importances = model_lr.coef_[0]

odds_ratios = pd.Series(np.exp(importances) , index = features).sort_values()
odds_ratios.head()
```

Out[58]:
```
ram              7.023289e-07
battery_power    3.356091e-02
px_height        7.997938e-02
px_width         1.317819e-01
int_memory       8.198506e-01
dtype: float64
```

In [59]:
```
odds_ratios.tail().plot(kind= 'barh')
plt.xlabel("Odds Ratio");
```



In [60]:
```
odds_ratios.head().plot(kind= 'barh')
plt.xlabel("Odds Ratio");
```



Thanks!