

BỘ GIÁO DỤC VÀ ĐÀO TẠO  
TRƯỜNG ĐH NGOẠI NGỮ - TIN HỌC TP.HCM  
KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO**  
**MÔN MÁY HỌC**

Giáo viên hướng dẫn: TS. Lê Thành Sách

Sinh viên thực hiện: Võ Thành Hoàng Sơn

HCMC, \_\_\_/\_\_\_/\_\_\_

# Mobile Price Dataset

Võ Thành Hoàng Sơn - 19DH110660

## LAB1 - Dataset(Mobile Price Dataset)

- Source: <https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification>
- Name: Mobile Price Classification
- Context:
  - Bob has started his own mobile company. He wants to give tough fight to big companies like Apple,Samsung etc.
  - He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market you cannot simply assume things. To solve this problem he collects sales data of mobile phones of various companies.
  - Bob wants to find out some relation between features of a mobile phone(eg:- RAM,Internal Memory etc) and its selling price. But he is not so good at Machine Learning. So he needs your help to solve this problem.
- Problem: In this problem you do not have to predict actual price but a price range indicating how high the price is

### 1.1 Load dataset

- Dataset url: 1AvhQXsYbaLg9ZLGWhZir1C9lYQJgZg\_D
- Folder tree:
  - Mobile Price Dataset
    - test.csv
    - train.csv

```
In [ ]: url = 'https://drive.google.com/drive/folders/1AvhQXsYbaLg9ZLGWhZir1C9lYQJgZg_D?usp=sharing'
import gdown
gdown.download_folder(url)

Retrieving folder list
Processing file 1_fWtViL-QZf5zsQx4MW44VUiiaHmjOMO1 test.csv
Processing file 1ASxEE4V7gnixpe6RMTVSXBZMni1L--Vu train.csv
Building directory structure completed
Retrieving folder list completed
Building directory structure
Downloading...
From: https://drive.google.com/uc?id=1_fWtViL-QZf5zsQx4MW44VUiiaHmjOMO1
To: /content/mobile_price_dataset/test.csv
100%[██████] 63.9k/63.9k [00:00<00:00, 30.0MB/s]
Downloading...
From: https://drive.google.com/uc?id=1ASxEE4V7gnixpe6RMTVSXBZMni1L--Vu
To: /content/mobile_price_dataset/train.csv
100%[██████] 122k/122k [00:00<00:00, 27.5MB/s]
Download completed
Out[ ]: ['/content/mobile_price_dataset/test.csv',
          '/content/mobile_price_dataset/train.csv']
```

### 1.2 Import library

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

from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
In [ ]: train = pd.read_csv('mobile_price_dataset/train.csv')
test = pd.read_csv('mobile_price_dataset/test.csv')
```

```
In [ ]: train.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	

5 rows × 21 columns

```
In [ ]: test = test.drop(['id'], axis=1)  
test.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_
0	1043	1	1.8	1	14	0	5	0.1	193	3	16	226	
1	841	1	0.5	1	4	1	61	0.8	191	5	12	746	
2	1807	1	2.8	0	1	0	27	0.9	186	3	4	1270	
3	1546	0	0.5	1	18	1	25	0.5	96	8	20	295	
4	1434	0	1.4	0	11	1	49	0.5	108	6	18	749	

**Nhận xét:** Sau khi hiển thị 5 dòng đầu của 2 tập dữ liệu

- Tập test có thêm cột id khiến các cột không đồng đều.

**Phương pháp:**

- Xóa cột id tại test dataset (để các cột đồng nhất giữa 2 dataset)

## 1.3 Explain dataset

```
In [ ]: 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: 328.2 KB
```

**Nhận xét:**

- Dataset có 19 features và một outcome
- Dataset 2000 mẫu

- Không chứa bất kì giá trị NaN, null
- Tất cả điều là kiểu dữ liệu số: (int64, float64) cần phải phân loại dữ liệu

### Phân loại các features

- Numerical: battery\_power, clock\_speed, pc, int\_memory, m\_dep, mobile\_wt, n\_cores, pc, px\_height, px\_width, ram, sc\_h, sc\_w, talk\_time.
- Categorical: blue, dual\_sim, four\_g, three\_g, wifi, price\_range, touch\_screen

### Transform to categorical

```
In [ ]:
categorical = ['blue', 'dual_sim', 'four_g', 'three_g', 'touch_screen', 'wifi']
numerical = ['battery_power', 'clock_speed', 'pc', 'int_memory', 'm_dep', 'mobile_wt', 'n_cores',
            'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time']

def convert_type(df,columns,astype):
    for feature in columns:
        df[feature] = df[feature].astype(astype)

convert_type(train,categorical,'category')
convert_type(test,categorical[:-1],'category')
```

```
In [ ]:
#Describe Numeric
train.describe().T
```

	count	mean	std	min	25%	50%	75%	max
<b>battery_power</b>	2000.0	1238.51850	439.418206	501.0	851.75	1226.0	1615.25	1998.0
<b>clock_speed</b>	2000.0	1.52225	0.816004	0.5	0.70	1.5	2.20	3.0
<b>fc</b>	2000.0	4.30950	4.341444	0.0	1.00	3.0	7.00	19.0
<b>int_memory</b>	2000.0	32.04650	18.145715	2.0	16.00	32.0	48.00	64.0
<b>m_dep</b>	2000.0	0.50175	0.288416	0.1	0.20	0.5	0.80	1.0
<b>mobile_wt</b>	2000.0	140.24900	35.399655	80.0	109.00	141.0	170.00	200.0
<b>n_cores</b>	2000.0	4.52050	2.287837	1.0	3.00	4.0	7.00	8.0
<b>pc</b>	2000.0	9.91650	6.064315	0.0	5.00	10.0	15.00	20.0
<b>px_height</b>	2000.0	645.10800	443.780811	0.0	282.75	564.0	947.25	1960.0
<b>px_width</b>	2000.0	1251.51550	432.199447	500.0	874.75	1247.0	1633.00	1998.0
<b>ram</b>	2000.0	2124.21300	1084.732044	256.0	1207.50	2146.5	3064.50	3998.0
<b>sc_h</b>	2000.0	12.30650	4.213245	5.0	9.00	12.0	16.00	19.0
<b>sc_w</b>	2000.0	5.76700	4.356398	0.0	2.00	5.0	9.00	18.0
<b>talk_time</b>	2000.0	11.01100	5.463955	2.0	6.00	11.0	16.00	20.0
<b>price_range</b>	2000.0	1.50000	1.118314	0.0	0.75	1.5	2.25	3.0

```
In [ ]:
#Describe Categoric
train.describe(include=['category'])
```

	blue	dual_sim	four_g	three_g	touch_screen	wifi
<b>count</b>	2000	2000	2000	2000	2000	2000
<b>unique</b>	2	2	2	2	2	2
<b>top</b>	0	1	1	1	1	1
<b>freq</b>	1010	1019	1043	1523	1006	1014

### Nhận xét:

- Giá trị min, max, std giữa các features kiểu numeric không đồng đều
- Các features kiểu category đều có 2 loại, số lượng mỗi loại của từng features khá tương đồng (khoảng 1010) ngoại trừ three\_g

### Phương pháp:

- Cân Chuẩn hoá dữ liệu

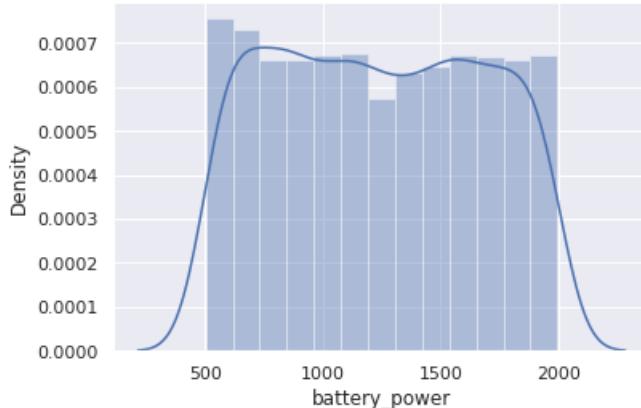
## 1.4 Visualize data

```
In [ ]:
sns.set()
sns.set_style('darkgrid')
```

```
In [ ]:
for i in numerical:
    sns.distplot(train[i])
    plt.show()
```

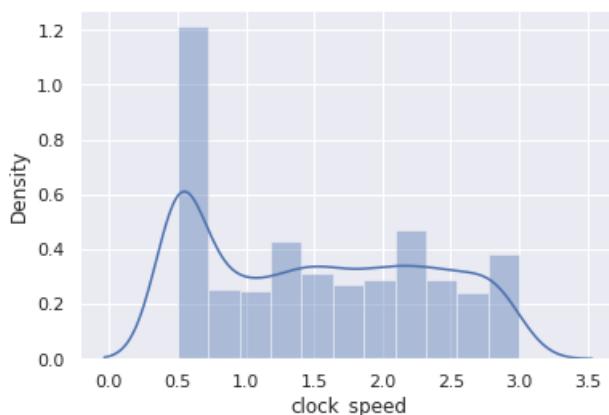
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

```
warnings.warn(msg, FutureWarning)
```



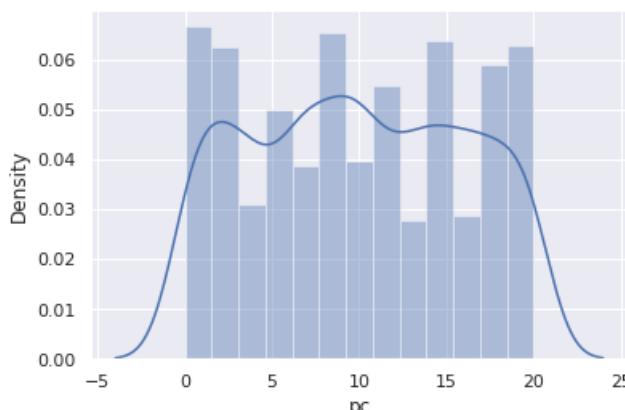
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

```
warnings.warn(msg, FutureWarning)
```



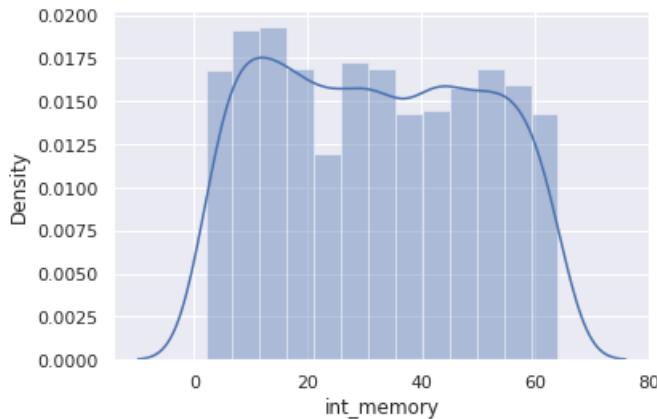
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

```
warnings.warn(msg, FutureWarning)
```



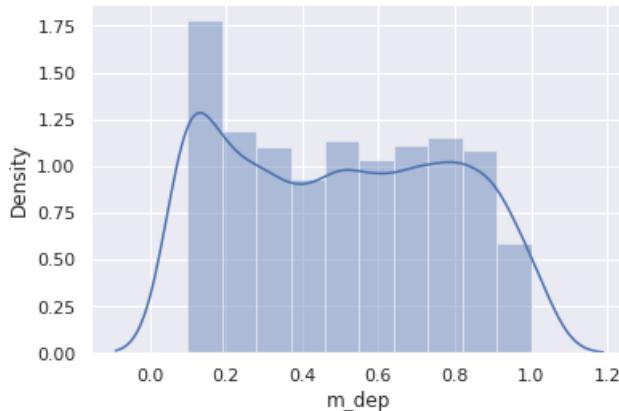
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a  
deprecated function and will be removed in a future version. Please adapt your code to use either `di  
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h  
istograms).
```

```
    warnings.warn(msg, FutureWarning)
```



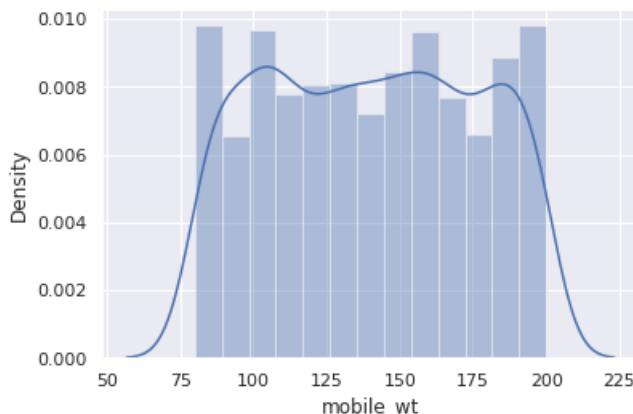
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a  
deprecated function and will be removed in a future version. Please adapt your code to use either `di  
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h  
istograms).
```

```
    warnings.warn(msg, FutureWarning)
```



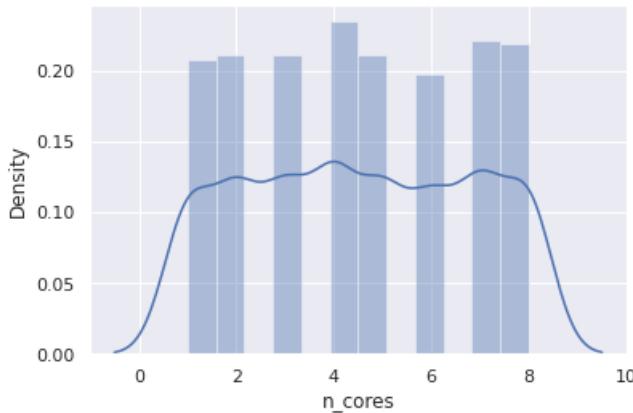
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a  
deprecated function and will be removed in a future version. Please adapt your code to use either `di  
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h  
istograms).
```

```
    warnings.warn(msg, FutureWarning)
```



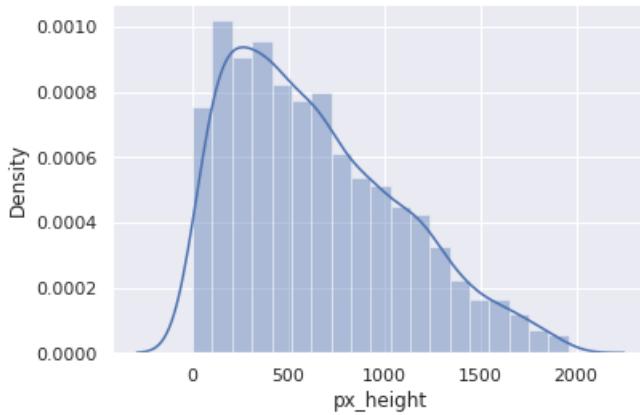
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a  
deprecated function and will be removed in a future version. Please adapt your code to use either `di  
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h  
istograms).
```

```
    warnings.warn(msg, FutureWarning)
```



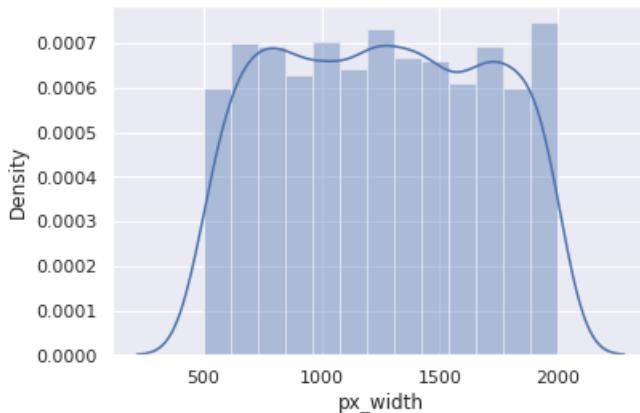
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h
istograms).
```

```
warnings.warn(msg, FutureWarning)
```



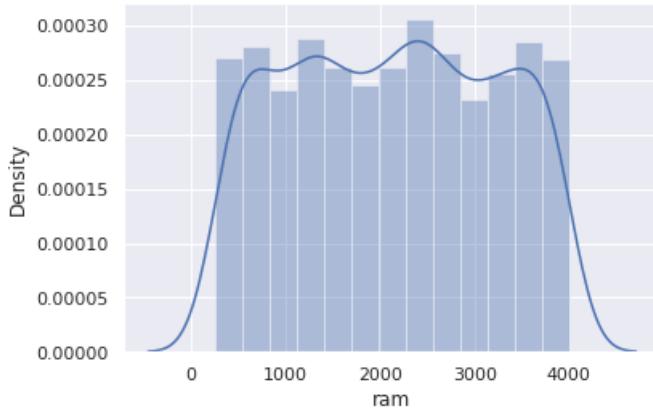
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h
istograms).
```

```
warnings.warn(msg, FutureWarning)
```



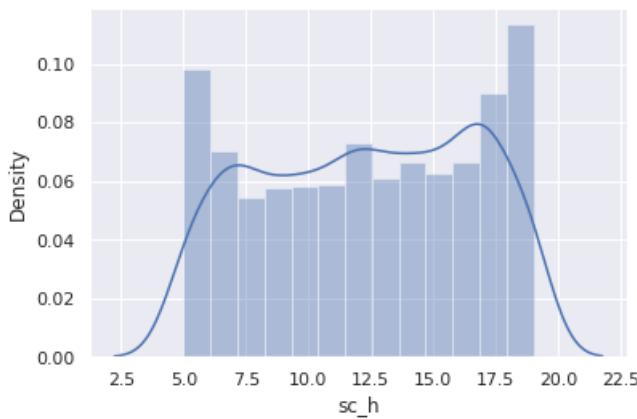
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h
istograms).
```

```
warnings.warn(msg, FutureWarning)
```



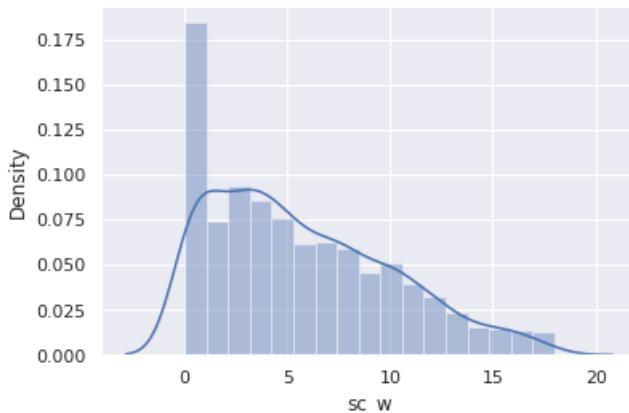
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h
istograms).
```

```
    warnings.warn(msg, FutureWarning)
```



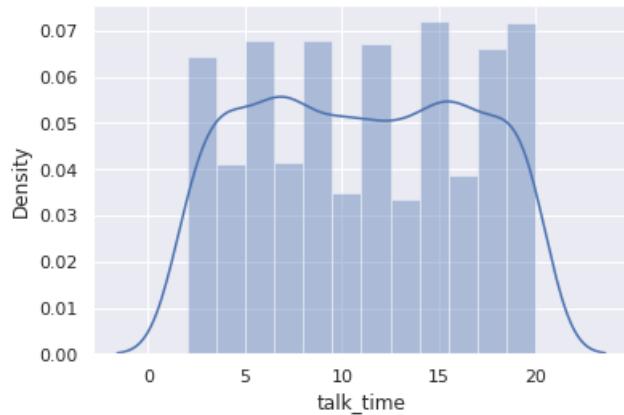
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h
istograms).
```

```
    warnings.warn(msg, FutureWarning)
```



```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `di
splot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for h
istograms).
```

```
    warnings.warn(msg, FutureWarning)
```



**Các dạng phân phối:** Các features được hiển thị dưới 2 kiểu phân phối: Posion, Phân phối đều

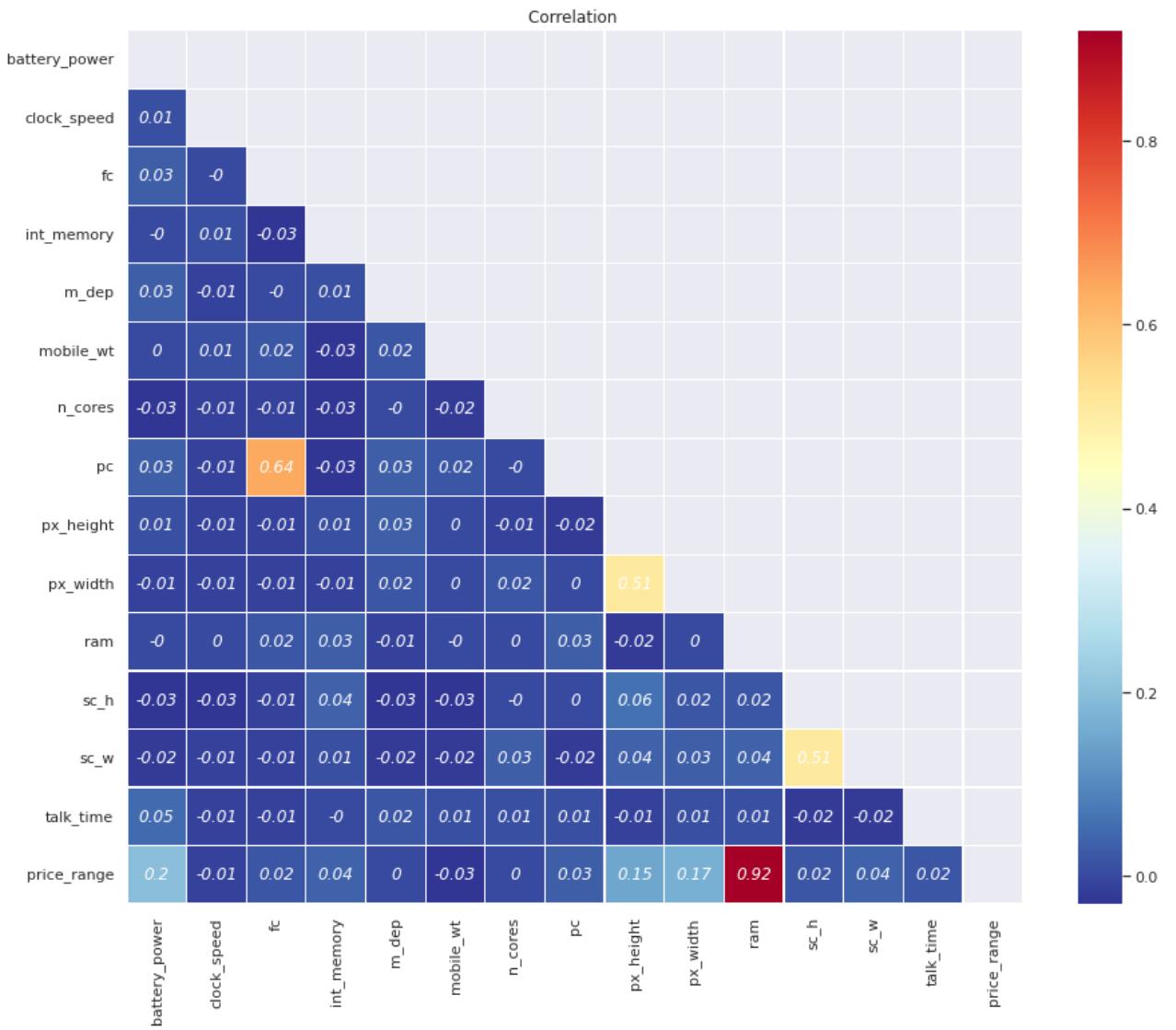
- Posion: px\_height, sc\_w
- Phân phối đều: các features còn lại

### 1.4.1 Correlation

```
In [ ]: convert_type(train,['price_range'],'int64')
```

```
In [ ]: corr=train.corr().round(2)
fig = plt.figure(figsize=(15,12))
mask = np.triu(np.ones_like(corr))
# mask = mask > 0.4
r = sns.heatmap(corr,
                 cmap="RdYlBu_r",
                 annot=True,
                 annot_kws = {'fontsize':12,
                               'fontstyle':'italic',
                               'color':'w',
                               'verticalalignment':'center'},
                 mask=mask,
                 linewidth=.2)
r.set_title("Correlation ")
```

```
Out[ ]: Text(0.5, 1.0, 'Correlation ')
```



**Nhận xét:**

- Độ tương quan giữa Ram và mức giá là rất lớn (0.92)

**Phương pháp:**

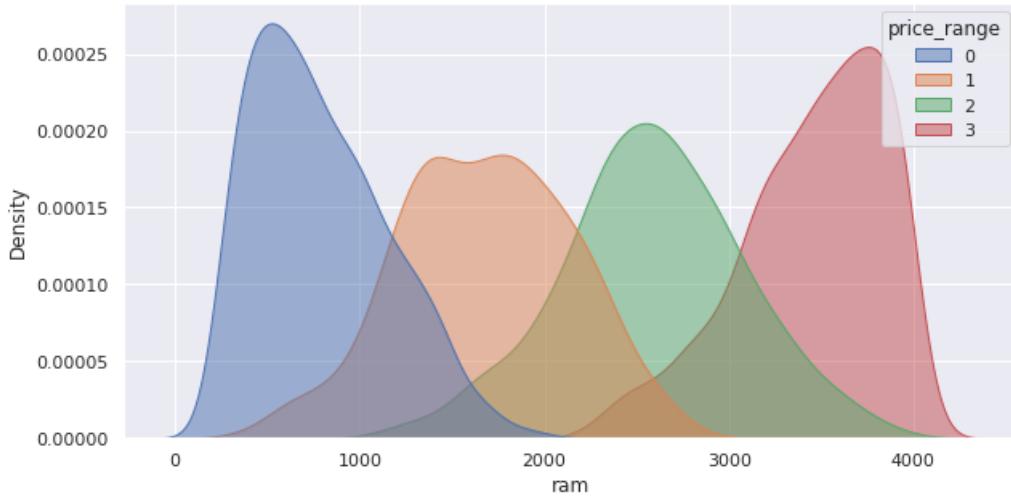
- Ram là tham chiếu để Visualize với các đặc trưng khác.

```
In [ ]: convert_type(train,['price_range'],'category')
```

### 1.4.2 Some plot

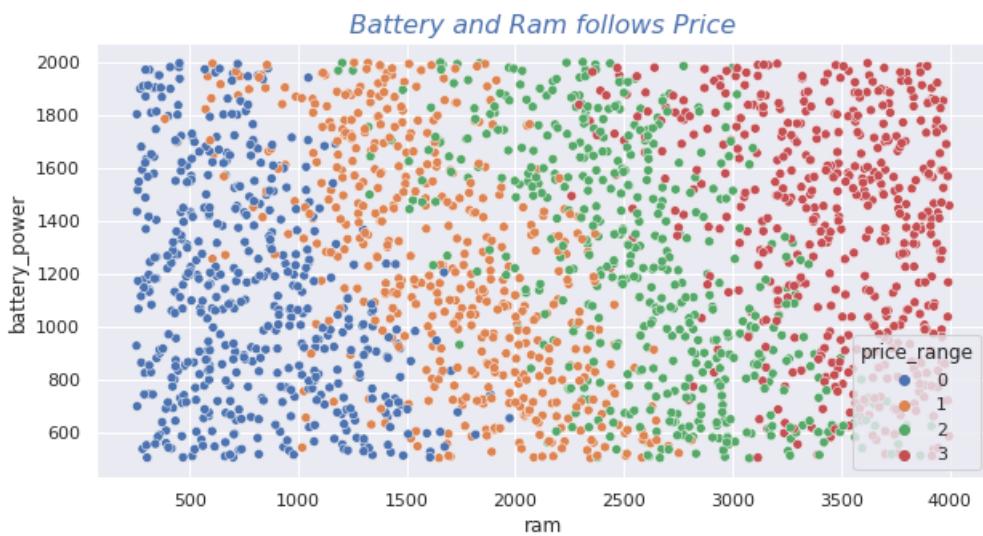
```
In [ ]: plt.figure(figsize=(10,5))
sns.kdeplot(data= train, x='ram', hue='price_range', fill=True, alpha=.5, linewidth=1)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd72c6a3d50>
```



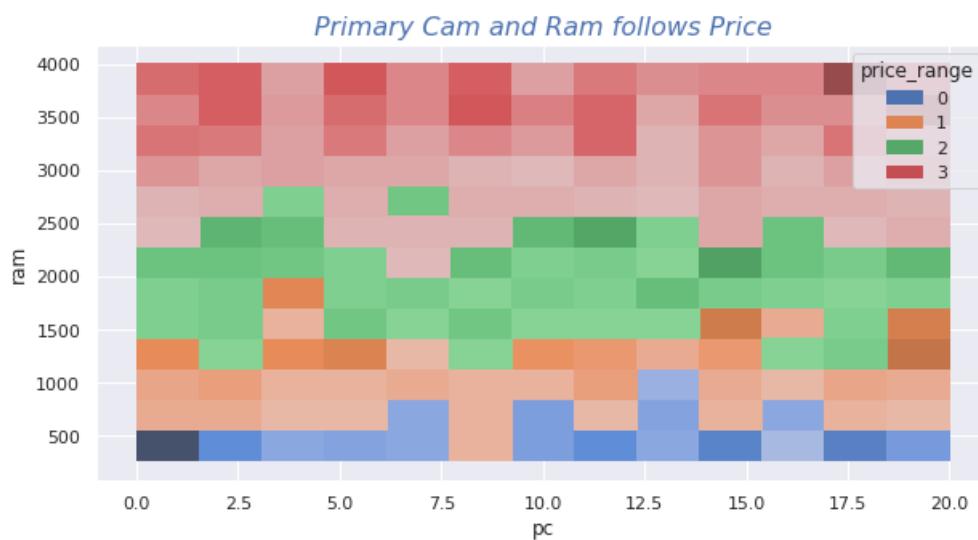
```
In [ ]: plt.figure(figsize=(10,5))
sns.scatterplot(data=train, x='ram', y='battery_power', hue='price_range')
plt.title("Battery and Ram follows Price",c='b',style="oblique",size=16)
```

Out[ ]: Text(0.5, 1.0, 'Battery and Ram follows Price')



```
In [ ]: plt.figure(figsize=(10,5))
sns.histplot(data=train, x='pc', y='ram', hue='price_range')
plt.title("Primary Cam and Ram follows Price",c='b',style="oblique",size=16)
```

Out[ ]: Text(0.5, 1.0, 'Primary Cam and Ram follows Price')



Nhận xét:

- Khi hiển thị các feature(battery,pc) với cột ram cho thấy có sự phân lớp đặc biệt là battery
- Tuy nhiên những sự phân lớp này chưa rời rạc

**Phương pháp:** Có thể sử dụng các đặc trưng này để tiến hành training để đổi chiều kết quả với toàn bộ dataset

## 1.5 Grid Search (Tìm thông số tối ưu)

```
In [ ]: !pip install pipelinehelper -q
```

```
In [ ]:
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFold

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer, StandardScaler, MaxAbsScaler, FunctionTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
import sklearn

from sklearn.pipeline import Pipeline
from pipelinehelper import PipelineHelper

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

```
In [ ]: ## Pipeline params

params = {}
params["data_split_train"] = 0.7
params['random_state'] = 54
params['k_fold'] = 3
```

```
In [ ]:
pipe = Pipeline([
    ('scaler', PipelineHelper([
        ('std', StandardScaler()),
        ('normal', Normalizer()),
        ('non', FunctionTransformer())
    ])),
    ('classifier', PipelineHelper([
        ('svm', SVC()),
        ('lr', LogisticRegression()),
        ('knn', KNeighborsClassifier())
    ])),
])
# Các cấu hình thử nghiệm
parameters = {
    'scaler_selected_model': pipe.named_steps['scaler'].generate({
        'std_with_mean': [True],
        'std_with_std': [False],
        'normal_norm': ['l1', 'l2', 'max']
    }),
    'classifier_selected_model': pipe.named_steps['classifier'].generate({
        # 'svm_C': [1.0, 10.0],
        # 'svm_kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
        # 'svm_degree': [2, 3, 5, 6, 10],
        'lr_fit_intercept': [True, False],
        'lr_solver': ['saga', 'liblinear', 'newton-cg'],
        'lr_max_iter': [500, 1000],
        # 'knn_n_neighbors': [5, 10, 15, 20, 25, 30],
        # 'knn_leaf_size': [20, 25, 30, 35, 40]
    })
}
```

```
# Lựa chọn cấu hình: kiểm thử chéo (chia X_train thành 5 phần)
grid = GridSearchCV(pipe, parameters, cv=parameters['k_fold'], scoring='accuracy', verbose=1)
```

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(train.iloc[:,0:-1],
                                                       train.iloc[:, -1],
                                                       random_state=parameters['random_state'],
                                                       train_size=parameters['data_split_train'])
```

```
In [ ]: #Fit and train model
grid.fit(X_train, y_train)

print("-"*80)
print("Cấu hình tốt nhất: ", grid.best_params_)
print("Độ chính xác: {:.2f}".format(grid.score(X_test, y_test)))
```

Fitting 3 folds for each of 70 candidates, totalling 210 fits

```
-----
Cấu hình tốt nhất: {'classifier_selected_model': ('lr', {'fit_intercept': True, 'max_iter': 500, 'solver': 'newton-cg'}), 'scaler_selected_model': ('std', {'with_mean': True, 'with_std': False})}
Độ chính xác: 0.97
```

## 1.6 Đánh giá mô hình với tham số tối ưu (Thực nghiệm 1)

```
In [ ]: from sklearn import metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns

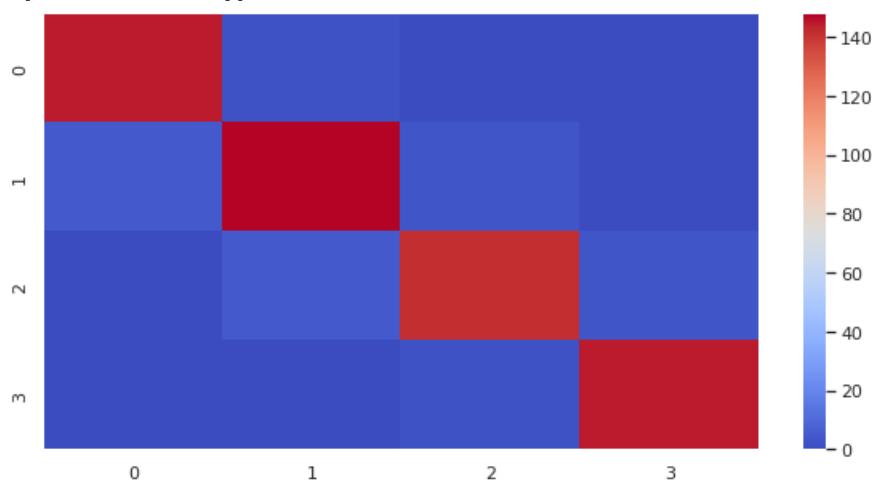
# Dự báo cho dữ liệu trong tập "kiểm tra"
y_pred = grid.predict(X_test)

# Tính các độ đo
cmatrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,5))
sns.heatmap(cmatrix,cmap='coolwarm')

print("Ma trận nhầm lẫn: ")
print(cmatrix)
```

Ma trận nhầm lẫn:

```
[[145  2   0   0]
 [ 5 148  3   0]
 [ 0  5 142  3]
 [ 0   0  2 145]]
```



```
In [ ]: # Tổng quan kết quả
metric_score = classification_report(y_test, y_pred)
print(metric_score)
```

	precision	recall	f1-score	support
0	0.97	0.99	0.98	147
1	0.95	0.95	0.95	156
2	0.97	0.95	0.96	150
3	0.98	0.99	0.98	147
accuracy			0.97	600
macro avg	0.97	0.97	0.97	600

```
weighted avg      0.97      0.97      0.97      600
```

```
In [ ]: from sklearn.metrics import precision_recall_fscore_support
scores = precision_recall_fscore_support(y_test, y_pred, average='macro')
accuracy = metrics.accuracy_score(y_test, y_pred, normalize=True)
```

```
In [ ]: # Kết quả
print("Độ chính xác (precision): {:.2f}%".format(scores[0]*100))
print("Độ triệu hồi (recall): {:.2f}%".format(scores[1]*100))
print("Độ đo F1 (F1-measure): {:.2f}%".format(scores[2]*100))
print("Độ chính xác (accuracy): {:.2f}%".format(accuracy*100))

Độ chính xác (precision): 96.68%
Độ triệu hồi (recall): 96.70%
Độ đo F1 (F1-measure): 96.69%
Độ chính xác (accuracy): 96.67%
```

```
In [ ]: results = {}
results['all'] = [scores[0]*100, scores[1]*100, scores[2]*100, accuracy*100]
```

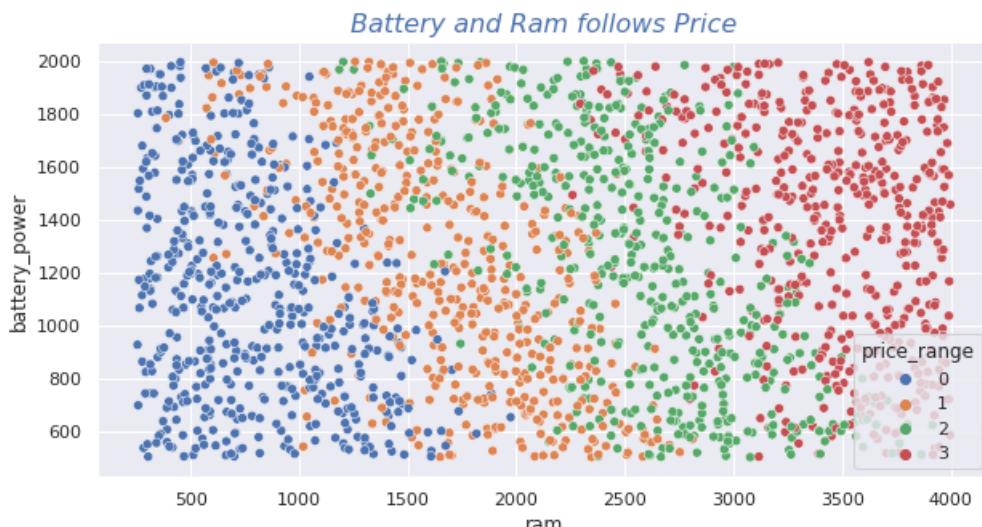
## 2. Advanced

Ở bước hiển thị dữ liệu (1.4.2) ta thấy rằng các đặc trưng tách biệt khá tốt vì thế ở phần này chúng ta tiến hành các thực nghiệm sau

### 2.1 Using only ram & battery

```
In [ ]: plt.figure(figsize=(10,5))
sns.scatterplot(data= train, x= 'ram', y= 'battery_power', hue='price_range')
plt.title("Battery and Ram follows Price",c='b',style="oblique",size=16)
```

```
Out[ ]: Text(0.5, 1.0, 'Battery and Ram follows Price')
```



```
In [ ]: rb_train = train[['battery_power','ram','price_range']]
rb_test = test[['battery_power','ram']]
```

```
In [ ]: #Train valid split
X_train, X_test, y_train, y_test = train_test_split(rb_train.iloc[:,0:-1],
                                                    rb_train.iloc[:, -1],
                                                    random_state=params["random_state"],
                                                    train_size=params['data_split_train'])
```

```
In [ ]: grid1 = GridSearchCV(pipe, parameters, cv= 2, scoring='accuracy',verbose=1)

#Fit
grid1.fit(X_train, y_train)
```

```

y_pred = grid1.predict(X_test)

print("-"*80)
print("Cáu hình tốt nhất: ", grid1.best_params_)
print("Độ chính xác: {:.6.2f}{}".format(grid1.score(X_test, y_test)))

```

Fitting 2 folds for each of 70 candidates, totalling 140 fits

-----

Cáu hình tốt nhất: {'classifier\_selected\_model': ('lr', {'fit\_intercept': True, 'max\_iter': 500, 'solver': 'newton-cg'}), 'scaler\_selected\_model': ('std', {'with\_mean': True, 'with\_std': False})}

Độ chính xác: 0.80

In [ ]:

```

# Kết quả
scores = precision_recall_fscore_support(y_test, y_pred, average='macro')
accuracy = metrics.accuracy_score(y_test, y_pred, normalize=True)
print("Độ chính xác (precision): {:.7.2f}%".format(scores[0]*100))
print("Độ triệu hồi (recall): {:.7.2f}%".format(scores[1]*100))
print("Độ đo F1 (F1-measure): {:.7.2f}%".format(scores[2]*100))
print("Độ chính xác (accuracy): {:.7.2f}%".format(accuracy*100))

```

Độ chính xác (precision): 79.96%  
Độ triệu hồi (recall): 80.30%  
Độ đo F1 (F1-measure): 80.07%  
Độ chính xác (accuracy): 80.17%

In [ ]:

```
results['ram & battery'] = [scores[0]*100, scores[1]*100, scores[2]*100, accuracy*100]
```

**Kết quả:** Kết quả cho thấy khi chỉ sử dụng battery và ram cho kết quả khá cao (accuracy: 80,5) tuy nhiên vẫn thấp hơn nhiều so với khi sử dụng toàn bộ dataset (accuracy: 96,6%)

**Cáu hình tốt nhất:** {'classifier\_selected\_model': ('lr', {'fit\_intercept': True, 'max\_iter': 500, 'solver': 'newton-cg'}), 'scaler\_selected\_model': ('std', {'with\_mean': True, 'with\_std': False})}

**Độ chính xác:** 0.80

## 2.2 Using only px\_width & ram

In [ ]:

```

pr_train = train[['px_width', 'ram', 'price_range']]
pr_test = test[['px_width', 'ram']]

#Train valid split
X_train, X_test, y_train, y_test = train_test_split(pr_train.iloc[:,0:-1],
                                                    pr_train.iloc[:, -1],
                                                    random_state=params["random_state"],
                                                    train_size=params['data_split_train'])

grid2 = GridSearchCV(pipe, parameters, cv=params['k_fold'], scoring='accuracy', verbose=1)

#Fit
grid2.fit(X_train, y_train)
y_pred = grid2.predict(X_test)
print("-"*80)
print("Cáu hình tốt nhất: ", grid2.best_params_)
print("Độ chính xác: {:.6.2f}{}".format(grid2.score(X_test, y_test)))

```

Fitting 3 folds for each of 70 candidates, totalling 210 fits

-----

Cáu hình tốt nhất: {'classifier\_selected\_model': ('svm', {}), 'scaler\_selected\_model': ('std', {'with\_mean': True, 'with\_std': False})}

Độ chính xác: 0.74

In [ ]:

```

# Kết quả
scores = precision_recall_fscore_support(y_test, y_pred, average='macro')
accuracy = metrics.accuracy_score(y_test, y_pred, normalize=True)
print("Độ chính xác (precision): {:.7.2f}%".format(scores[0]*100))
print("Độ triệu hồi (recall): {:.7.2f}%".format(scores[1]*100))
print("Độ đo F1 (F1-measure): {:.7.2f}%".format(scores[2]*100))
print("Độ chính xác (accuracy): {:.7.2f}%".format(accuracy*100))

```

Độ chính xác (precision): 74.37%  
Độ triệu hồi (recall): 74.63%  
Độ đo F1 (F1-measure): 74.45%  
Độ chính xác (accuracy): 74.50%

In [ ]:

```
results['ram & px_width'] = [scores[0]*100, scores[1]*100, scores[2]*100, accuracy*100]
```

**Nhận xét:** Khi chỉ sử dụng cột px\_width và ram chỉ cho ra kết quả tương đối (accuracy 75,3%)

## 2.3 Results

```
In [ ]: df_results = pd.DataFrame.from_dict(results)
df_results['index'] = ['precision', 'recall', 'f1-score', 'accuracy']
df_results.set_index('index')
```

```
Out[ ]:          all  ram & battery  ram & px_width
index
precision  96.680538    79.960548    74.372942
recall    96.704343    80.301413    74.628205
f1-score   96.686982    80.068385    74.450471
accuracy   96.666667    80.166667    74.500000
```

**Nhận xét:**

- Khi chỉ sử dụng 2 feature ram và (battery or px\_width) cho ta kết quả thấp hơn khi dùng toàn bộ dữ liệu
- Tuy nhiên kết quả khi chỉ sử dụng ram & battery cho ra kết quả rất tiềm năng (accuracy: 81,5%)

**Phương pháp:** Có thể kết hợp ram & battery cùng một số đặc trưng có độ tương qua với outcome lớn (px\_height, px\_with, int\_memory) sẽ cho ra kết quả tốt hơn

## 2.4 Plot boudary

```
In [ ]: !pip install mlxtend
```

```
In [ ]: from mlxtend.plotting import plot_decision_regions

X = rb_train[['ram', 'battery_power']]
y = np.array(rb_train['price_range'])

scaler = StandardScaler(with_mean = True, with_std= False)
X = scaler.fit_transform(X)

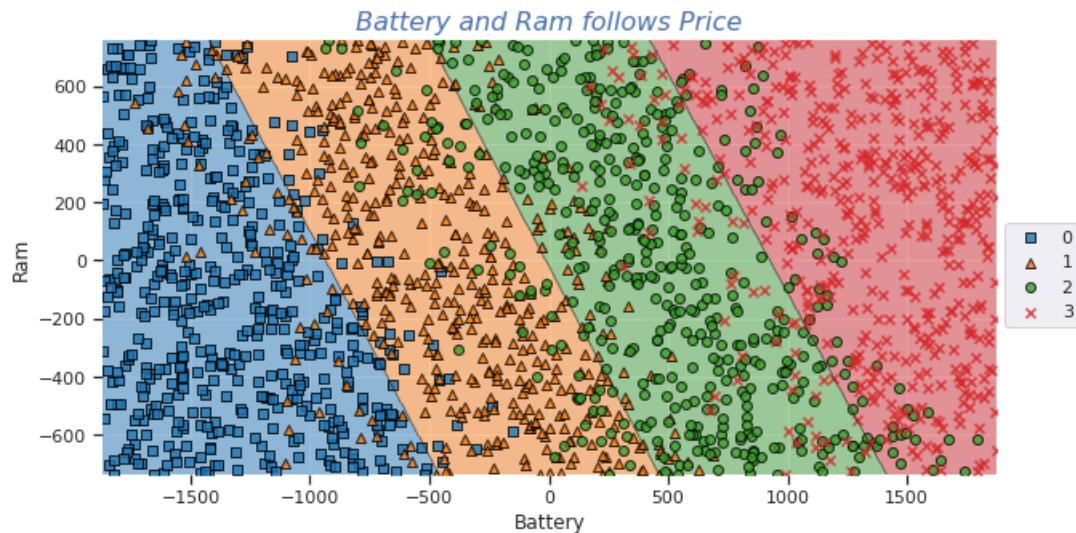
# Training a classifier
clf = LogisticRegression(fit_intercept= True, max_iter= 500, solver= 'newton-cg')
clf.fit(X, y)

# Plotting decision regions
sns.set()
plt.figure(figsize=(10,5))

plot_decision_regions(X, y, clf=clf)

# Adding axes annotations
plt.title("Battery and Ram follows Price", c='b', style="oblique", size=16)
plt.xlabel('Battery')
plt.ylabel('Ram')

plt.legend(bbox_to_anchor=(1, 0.6))
plt.show()
```



**Nhận xét:** Khi kết hợp đặc trưng Ram và Battery ta có thể thấy độ phân tách của class khá tốt đặc biệt là lớp 0 và 3

## Summary

- Lab1: Tìm hiểu và hiển thị data
  - Dataset về thông số của điện thoại dùng để phân lớp mức giá
  - Gồm 2000 mẫu và 19 đặc trưng (non-null)
  - Các đặc trưng có tương quan lớn với class (ram, battery, px\_width)
  - Dùng GridSearch tìm mô hình và params tối ưu ==> Thử nghiệm: chỉ sử dụng một số đặc trưng tốt tiến hành phân lớp.

---> Kết quả: Kết quả không tốt bằng khi sử dụng toàn bộ dataset

In [ ]:



# Mobile Price Dataset

Võ Thành Hoàng Sơn - 19DH110660

## LAB1(Continue) - Dataset(Mobile Price Dataset)

### Thực hiện lại các bước xử lý đã nêu ở Lab1

- Source: <https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification>
- Name: Mobile Price Classification
- Context:
  - Bob has started his own mobile company. He wants to give tough fight to big companies like Apple,Samsung etc.
  - He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market you cannot simply assume things. To solve this problem he collects sales data of mobile phones of various companies.
  - Bob wants to find out some relation between features of a mobile phone(eg:- RAM,Internal Memory etc) and its selling price. But he is not so good at Machine Learning. So he needs your help to solve this problem.
- Problem: In this problem you do not have to predict actual price but a price range indicating how high the price is

### Load dataset

- Dataset url: 1AvhQXsYbaLg9ZLGWhZir1C9lYQJgZg\_D
- Folder tree:
  - Mobile Price Dataset
    - test.csv
    - train.csv

```
In [ ]: url = 'https://drive.google.com/drive/folders/1AvhQXsYbaLg9ZLGWhZir1C9lYQJgZg_D?usp=sharing'
import gdown
gdown.download_folder(url)

Retrieving folder list
Processing file 1_fWtViL-QZf5zsQx4MW44VUiiaHmjOMO1 test.csv
Processing file 1ASxEE4V7gnixpe6RMTVSXBZMnill--Vu train.csv
Building directory structure completed
Retrieving folder list completed
Building directory structure
Downloading...
From: https://drive.google.com/uc?id=1_fWtViL-QZf5zsQx4MW44VUiiaHmjOMO1
To: /content/mobile_price_dataset/test.csv
100%|██████████| 63.9k/63.9k [00:00<00:00, 35.7MB/s]
Downloading...
From: https://drive.google.com/uc?id=1ASxEE4V7gnixpe6RMTVSXBZMnill--Vu
To: /content/mobile_price_dataset/train.csv
100%|██████████| 122k/122k [00:00<00:00, 26.5MB/s]
Download completed
Out[ ]: ['/content/mobile_price_dataset/test.csv',
          '/content/mobile_price_dataset/train.csv']
```

### Import library

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

from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
In [ ]: train = pd.read_csv('mobile_price_dataset/train.csv')
test = pd.read_csv('mobile_price_dataset/test.csv')

In [ ]:
categorical = ['blue', 'dual_sim', 'four_g', 'three_g', 'touch_screen', 'wifi']
numerical = ['battery_power', 'clock_speed', 'pc', 'int_memory', 'm_dep', 'mobile_wt', 'n_cores',
            'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time']

def convert_type(df,columns,astype):
    for feature in columns:
        df[feature] = df[feature].astype(astype)

convert_type(train,categorical,'category')
convert_type(test,categorical[:-1], 'category')

In [ ]: convert_type(train,['price_range'], 'category')
```

## LAB2 - GridSearch with Dim Reduce & MLP, Decision Tree

### Thực nghiệm với Grid Search với phương pháp Giảm chiều dữ liệu

- Ở đây chúng ta sẽ thực hiện 3 thực nghiệm như sau:
  - Thực nghiệm với tất cả trường hợp (bao gồm PCA, LDA, Kết hợp PCA và LDA, Không sử dụng giảm chiều)
  - Thực nghiệm với tất cả trường hợp sử dụng phương pháp giảm chiều dữ liệu (bao gồm PCA, LDA, Kết hợp PCA và LDA)
  - Thêm giá trị vào dãy n\_component để xem giá trị n\_component tối ưu.
- Sau đó ghi nhận tất cả kết quả và nhận xét

### Có cả phương pháp không giảm chiều dữ liệu

```
In [ ]:
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer, StandardScaler, MaxAbsScaler, FunctionTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
import sklearn

## Dim reduction
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

## Custom transform
from sklearn.preprocessing import FunctionTransformer

## Pipeline
!pip install pipelinehelper -q
from sklearn.pipeline import Pipeline
from pipelinehelper import PipelineHelper

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

```
In [ ]:
import os
n_cpus = os.cpu_count()
print(n_cpus)

## Pipeline params

params = {}
params['data_split_train'] = 0.7
params['random_state'] = 54
params['k_fold'] = 3
```

40

```
In [ ]: pipe = Pipeline([
    ('redudim', PipelineHelper([
```

```

('pca', PCA()),
('pcaanlda', Pipeline([
    ('pcaa', PCA()),
    ('lada', LinearDiscriminantAnalysis())
]),
('lda', LinearDiscriminantAnalysis()),
('non', FunctionTransformer(None, validate=False)),
]),

('scaler', PipelineHelper([
    ('std', StandardScaler()),
    ('normal', Normalizer()),
    ('non', FunctionTransformer(None, validate=False)),
])),
('classifier', PipelineHelper([
    # ('svm', SVC()),
    ('lr', LogisticRegression()),
    # ('knn', KNeighborsClassifier())
    ('mlp', MLPClassifier()),
    ('dtree', DecisionTreeClassifier())

])),
)

# Các cấu hình thử nghiệm
parameters = {
    'redudim__selected_model': pipe.named_steps['redudim'].generate({
        'pca__n_components': [.95,.97,.99],
        'pca__whiten': [True,False],
        'pca__svd_solver': ['full', 'arpack', 'randomized'],
        'pcaanlda__pcaa__n_components': [.95,.97,.99],
        'pcaanlda__pcaa__whiten': [True,False],
        'pcaanlda__pcaa__svd_solver': ['full', 'arpack', 'randomized'],
        'pcaanlda__lada__solver': ['svd', 'lsqr', 'eigen'],
        'pcaanlda__lada__n_components': [.95,.97,.99],
        'lada__solver': ['svd', 'lsqr', 'eigen'],
        'lada__n_components': [.92,.95,.99]
    }),

    'scaler__selected_model': pipe.named_steps['scaler'].generate({
        'std__with_mean': [True],
        'std__with_std': [False],
        'normal__norm': ['l1','l2','max']
    }),
    'classifier__selected_model': pipe.named_steps['classifier'].generate({
        # 'svm__C': [1.0, 10.0],
        # 'svm__kernel': ['poly', 'rbf', 'sigmoid'],
        # 'svm__degree': [2,5,10],
        'lr__fit_intercept': [True,False],
        'lr__solver': ['newton-cg', 'liblinear', 'saga'],
        'lr__max_iter': [100,500],
        '# 'knn__n_neighbors' : [5,10,15,20,25,30],
        '# 'knn__leaf_size' : [20,25,30,35,40]
        'mlp__hidden_layer_sizes': [2,3,4],
        # 'mlp__lr': [4e-3,1e-4],
    })
}

```

```

In [ ]: # Chia tập dữ liệu
X_train, X_test, y_train, y_test = train_test_split(train.iloc[:,0:-1],
                                                    train.iloc[:, -1],
                                                    random_state=params["random_state"],
                                                    train_size=params['data_split_train'])

# Lựa chọn cấu hình: kiểm thử chéo
grid = GridSearchCV(pipe, parameters, cv=params['k_fold']),

```

```

        scoring='accuracy', refit=True, n_jobs=n_cpus,
        verbose=10)
#Fit and train model
grid.fit(X_train, y_train)

print("-"*80)
print("Cấu hình tốt nhất: ", grid.best_params_)
print("Độ chính xác: {:.6.2f}".format(grid.score(X_test, y_test)))

```

Fitting 3 folds for each of 15200 candidates, totalling 45600 fits

```

-----
Cấu hình tốt nhất: {'classifier_selected_model': ('lr', {'fit_intercept': True, 'max_iter': 100, 'solver': 'newton-cg'}), 'redundim_selected_model': ('non', {}), 'scaler_selected_model': ('std', {'with_mean': True, 'with_std': False})}
Độ chính xác: 0.96

```

**Nhận xét:** Khi không sử dụng phương pháp giảm chiều dữ liệu sẽ cho ra kết quả tối ưu hơn

## Thực hiện thực nghiệm bắt buộc sử dụng phương pháp giảm chiều dữ liệu

```

In [ ]:
pipe = Pipeline([
    ('redundim', PipelineHelper([
        ('pca', PCA()),
        ('pcanlda', Pipeline([
            ('pcaa', PCA()),
            ('lada', LinearDiscriminantAnalysis())
        ])),
        ('lda', LinearDiscriminantAnalysis()),
        # ('non', FunctionTransformer(None, validate=False)),
    ])),
    ('scaler', PipelineHelper([
        ('std', StandardScaler()),
        ('normal', Normalizer()),
        ('non', FunctionTransformer(None, validate=False)),
    ])),
    ('classifier', PipelineHelper([
        # ('svm', SVC()),
        ('lr', LogisticRegression()),
        # ('knn', KNeighborsClassifier()),
        ('mlp', MLPClassifier()),
        ('dtree', DecisionTreeClassifier())
    ])),
])
# Các cấu hình thử nghiệm
parameters = {
    'redundim_selected_model': pipe.named_steps['redundim'].generate({
        'pca_n_components': [.95,.97,.99],
        'pca_whiten': [True, False],
        'pca_svd_solver': ['full', 'arpack', 'randomized'],
        'pcanlda_pcaa_n_components': [.95,.97,.99],
        'pcanlda_pcaa_whiten': [True, False],
        'pcanlda_pcaa_svd_solver': ['full', 'arpack', 'randomized'],
        'pcanlda_lada_solver': ['svd', 'lsqr', 'eigen'],
        'pcanlda_lada_n_components': [.95,.97,.99],
        'lda_solver': ['svd', 'lsqr', 'eigen'],
        'lda_n_components': [.92,.95,.99]
    }),
    'scaler_selected_model': pipe.named_steps['scaler'].generate({
        'std_with_mean': [True],
        'std_with_std': [False],
        'normal_norm': ['l1', 'l2', 'max']
    }),
    'classifier_selected_model': pipe.named_steps['classifier'].generate({
        # 'svm_C': [1.0, 10.0],
        # 'svm_kernel': ['poly', 'rbf', 'sigmoid'],
        # 'svm_degree': [2,5,10],
    })
}

```

```

        'lr__fit_intercept': [True, False],
        'lr__solver': ['newton-cg', 'liblinear', 'saga'],
        'lr__max_iter': [100, 500],

        # 'knn__n_neighbors' : [5,10,15,20,25,30],
        # 'knn__leaf_size': [20,25,30,35,40]

        'mlp__hidden_layer_sizes': [2,3,4],
        # 'mlp__lr': [4e-3,1e-4],


    })
# Chia tập dữ liệu
X_train, X_test, y_train, y_test = train_test_split(train.iloc[:,0:-1],
                                                    train.iloc[:, -1],
                                                    random_state=params["random_state"],
                                                    train_size=params['data_split_train'])

# Lựa chọn cấu hình: kiểm thử chéo
grid = GridSearchCV(pipe, parameters, cv=params['k_fold'], scoring='accuracy',
                     refit=True, n_jobs=n_cpus,
                     verbose=10)
#Fit and train model
grid.fit(X_train, y_train)

print("-"*80)
print("Cấu hình tốt nhất: ", grid.best_params_)
print("Độ chính xác: {:.2f}".format(grid.score(X_test, y_test)))

```

Fitting 3 folds for each of 28080 candidates, totalling 84240 fits

Cấu hình tốt nhất: {'classifier\_selected\_model': ('lr', {'fit\_intercept': True, 'max\_iter': 100, 'solver': 'newton-cg'}), 'redundim\_selected\_model': ('pca', {'n\_components': 0.95, 'svd\_solver': 'full', 'whiten': False}), 'scaler\_selected\_model': ('std', {'with\_mean': True, 'with\_std': False})}  
Độ chính xác: 0.94

**Nhận xét:** Khi sử dụng phương pháp giảm chiều dữ liệu thì phương pháp được xử dụng là PCA với n\_component là giá trị thấp nhất trong danh sách chọn

**Phương pháp:** Thêm giá trị vào dãy n\_component để xem kết quả.

## Thực nghiệm thêm giá trị cho thông số n\_component

```

In [ ]: pipe = Pipeline([
    ('redundim', PipelineHelper([
        ('pca', PCA()),
        ('pcanlda', Pipeline([
            ('pcaa', PCA()),
            ('ldaa', LinearDiscriminantAnalysis())
        ])),
        ('lda', LinearDiscriminantAnalysis()),
        ('non', FunctionTransformer(None, validate=False)),
    ])),

    ('scaler', PipelineHelper([
        ('std', StandardScaler()),
        ('normal', Normalizer()),
        ('non', FunctionTransformer(None, validate=False)),
    ])),
    ('classifier', PipelineHelper([
        # ('svm', SVC()),
        ('lr', LogisticRegression()),
        # ('knn', KNeighborsClassifier()),
        ('mlp', MLPClassifier()),
        ('dtree', DecisionTreeClassifier())
    ])),
])

# Các cấu hình thử nghiệm
parameters = {
    'redundim_selected_model': pipe.named_steps['redundim'].generate({
        'pca_n_components': [.95, .97, .99],
        'pca_whiten': [True, False],
        'pca_svd_solver': ['full', 'arpack', 'randomized'],

        'pcanlda_pcaa_n_components': [.8, .9, .92, .95, .97, .99],
        'pcanlda_pcaa_whiten': [True, False],
        'pcanlda_pcaa_svd_solver': ['full', 'arpack', 'randomized'],
    })
}

```

```

'pca_ldaa_solver': ['svd', 'lsqr', 'eigen'],
'pca_ldaa_n_components': [.95,.97,.99],
lda_solver': ['svd', 'lsqr', 'eigen'],
'lda_n_components': [.92,.95,.99]
}),

'scaler_selected_model': pipe.named_steps['scaler'].generate({
    'std_with_mean': [True],
    'std_with_std': [False],
    'normal_norm': ['l1','l2','max']
}),
'classifier_selected_model': pipe.named_steps['classifier'].generate({
    # 'svm_C': [1.0, 10.0],
    # 'svm_kernel': ['poly', 'rbf', 'sigmoid'],
    # 'svm_degree': [2,5,10],
    'lr_fit_intercept': [True, False],
    'lr_solver': ['newton-cg', 'liblinear', 'saga'],
    'lr_max_iter': [100,500],
    # 'knn_n_neighbors' : [5,10,15,20,25,30],
    # 'knn_leaf_size': [20,25,30,35,40]
    'mlp_hidden_layer_sizes': [2,3,4],
    # 'mlp_lr': [4e-3,1e-4],
    ...
})
}

```

In [ ]:

```

# Chia tập dữ liệu
X_train, X_test, y_train, y_test = train_test_split(train.iloc[:,0:-1],
                                                    train.iloc[:, -1],
                                                    random_state=params["random_state"],
                                                    train_size=params['data_split_train'])

# Lựa chọn cấu hình: kiểm thử chéo
grid = GridSearchCV(pipe, parameters, cv=params['k_fold'], scoring='accuracy', refit=True,
                     n_jobs=n_cpus,
                     verbose=10)
#Fit and train model
grid.fit(X_train, y_train)

print("-"*80)
print("Cấu hình tốt nhất: ", grid.best_params_)
print("Độ chính xác: {:.6.2f}".format(grid.score(X_test, y_test)))

```

Fitting 3 folds for each of 28160 candidates, totalling 84480 fits

```

-----
Cấu hình tốt nhất: {'classifier_selected_model': ('lr', {'fit_intercept': True, 'max_iter': 100, 'solver': 'newton-cg'}),
'redundim_selected_model': ('non', {}), 'scaler_selected_model': ('std', {'with_mean': True, 'with_std': False})}
Độ chính xác: 0.96

```

## Các kết quả sau mỗi lần grid search

---

### Thực nghiệm 1

**Cấu hình tốt nhất:** {'classifier\_selected\_model': ('lr', {'fit\_intercept': True, 'max\_iter': 100, 'solver': 'newton-cg'}),  
'redundim\_selected\_model': ('non', {}), 'scaler\_selected\_model': ('std', {'with\_mean': True, 'with\_std': False})}

**Độ chính xác: 0.96**

---

### Thực nghiệm 2

**Cấu hình tốt nhất:** {'classifierselected\_model': ('lr', {'fit\_intercept': True, 'max\_iter': 100, 'solver': 'newton-cg'}), 'redundimselected\_model': ('pca', {'n\_components': 0.95, 'svd\_solver': 'full', 'whiten': False}), 'scaler\_\_selected\_model': ('std', {'with\_mean': True, 'with\_std': False})}

**Độ chính xác: 0.94**

**Nhận xét:** Với bộ dataset Mobile Price không sử dụng các phương pháp Dim Reduction sẽ cho ra kết quả tốt hơn

**Phương pháp:** Thí nghiệm: Khi sử dụng Dim Reduce cho ta kết quả tốt nhất tại n\_component = 95 là nhò nhất trong bảng chọn. Vì thế tiến hành thêm các giá trị nhỏ hơn vào pipeline để kiểm thử

### Thực nghiệm 3

**Cấu hình tốt nhất:** {'classifierselected\_model': ('lr', {'fit\_intercept': True, 'max\_iter': 100, 'solver': 'newton-cg'}), 'redundimselected\_model': ('pca', {'n\_components': 0.95, 'svd\_solver': 'full', 'whiten': False}), 'scaler\_\_selected\_model': ('std', {'with\_mean': True, 'with\_std': False})}

**Độ chính xác: 0.94**

**Kết luận:** Tập dữ liệu sẽ đạt kết quả tốt nhất nếu không sử dụng phương pháp giảm chiều dữ liệu. Tuy nhiên nếu phải sử dụng thì phương pháp PCA là phương pháp tốt nhất với cấu hình tại thực nghiệm 3.

## Chạy model với thông số tối ưu và có PCA ⚡

```
In [ ]: pca = PCA(n_components=.95, svd_solver='full', whiten=False)
pca = pca.fit(train.iloc[:,0:-1])
pca_train = pca.transform(train.iloc[:,0:-1])
pca_train = pd.DataFrame(pca_train)
```

```
In [ ]: pca_train.head()
```

```
Out[ ]:      0         1         2         3
0  430.597094 -795.788231 -390.070331  55.636140
1  504.984735  696.622368 -235.629081  343.925977
2  473.329828  763.942136 -680.059466 -113.916880
3  639.822324  779.691180 -630.783647 -30.402246
4 -718.985184  382.304525  591.040362 -392.357235
```

**Nhận xét:** Sau khi sử dụng PCA với n\_component = 0.9 ta thu được 4 cột

```
In [ ]: print(pca_train.shape)

cov = np.cov(pca_train.T)
pd.DataFrame(cov).style.background_gradient(cmap='RdYlBu_r')

(2000, 4)
```

```
Out[ ]:      0         1         2         3
0  1176743.028186      -0.000000      -0.000000      0.000000
1      -0.000000  289919.393829      -0.000000      -0.000000
2      -0.000000      -0.000000  193178.104381      0.000000
3      0.000000      -0.000000      0.000000  93630.102685
```

```
In [ ]: idx = np.arange(cov.shape[0])
diag = cov[idx,idx]
pd.DataFrame(diag).style.background_gradient(cmap='RdYlBu_r')
```

```
Out[ ]:      0
0  1176743.028186
```

```

0
1 289919.393829
2 193178.104381
3 93630.102685

```

## Pair plot

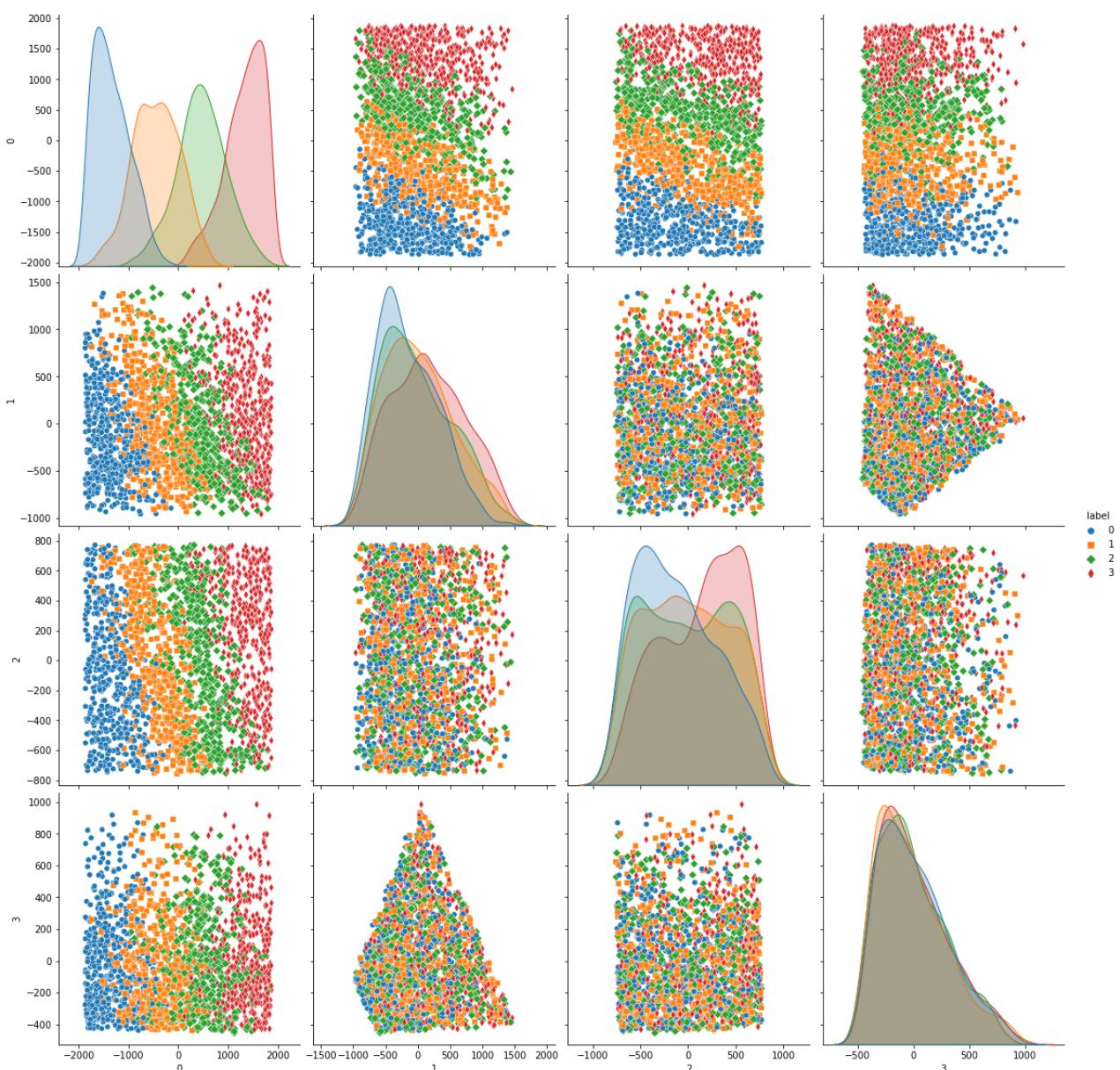
In [ ]:

```

dr_data = pca_train.copy()
sns.color_palette("vlag", as_cmap=True)
dr_data['label'] = train.iloc[:, -1]
sns.pairplot(dr_data, hue='label', markers=["o", "s", "D", "d"], size=4)

```

Out [ ]:



## Scaler and fit model

In [ ]:

```

scaler = StandardScaler(with_mean=True, with_std=False).fit(pca_train)
dr_train = pd.DataFrame(scaler.transform(pca_train))
dr_train.head()

```

Out [ ]:

	0	1	2	3
0	430.597094	-795.788231	-390.070331	55.636140
1	504.984735	696.622368	-235.629081	343.925977
2	473.329828	763.942136	-680.059466	-113.916880

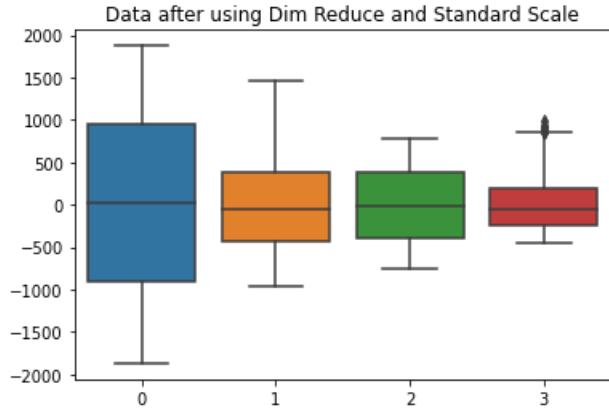
```

0           1           2           3
3  639.822324  779.691180 -630.783647 -30.402246
4 -718.985184  382.304525  591.040362 -392.357235

```

```
In [ ]: sns.boxplot(data = dr_train)
plt.title("Data after using Dim Reduce and Standard Scale")
```

```
Out[ ]: Text(0.5, 1.0, 'Data after using Dim Reduce and Standard Scale')
```



**Nhận xét:** Sau khi giảm chiều dữ liệu bằng PCA và Chuẩn hoá bằng Standard Scale Dữ liệu của chúng ta còn 4 đặc trưng và có Trung vị tại vị trí 0.

## Define and training model

```
In [ ]: x_train, x_test, y_train, y_test = train_test_split(dr_train, train.iloc[:, -1],
                                                       random_state=params["random_state"],
                                                       train_size=params['data_split_train'])
model = LogisticRegression(fit_intercept= True, max_iter= 100, solver= 'newton-cg')
model.fit(x_train, y_train)
print("Train done!")
```

```
Train done!
```

## Predict

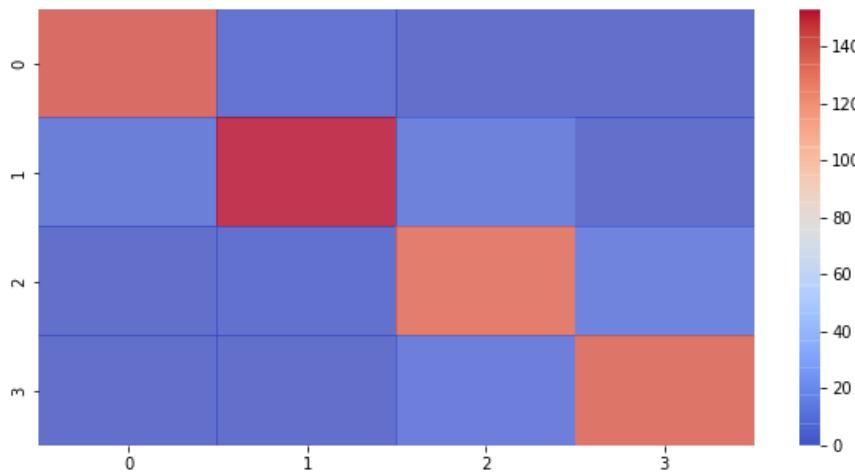
```
In [ ]: y_pred = model.predict(x_test)
```

```
In [ ]: from sklearn import metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Tính các độ đo
cmatrix = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 5))
sns.heatmap(cmatrix, cmap='coolwarm', alpha=0.8)

print("Ma trận nhầm lẫn: ")
print(cmatrix)
```

```
Ma trận nhầm lẫn:
[[141    2    0    0]
 [  6 153    8    0]
 [  0    1 135   9]
 [  0    0    7 138]]
```



```
In [ ]: # Tổng quan kết quả
metric_score = classification_report(y_pred,y_test)
print(metric_score)
```

	precision	recall	f1-score	support
0	0.96	0.99	0.97	143
1	0.98	0.92	0.95	167
2	0.90	0.93	0.92	145
3	0.94	0.95	0.95	145
accuracy			0.94	600
macro avg	0.94	0.95	0.95	600
weighted avg	0.95	0.94	0.95	600

```
In [ ]: from sklearn.metrics import precision_recall_fscore_support
scores = precision_recall_fscore_support(y_test, y_pred, average='macro')
accuracy = metrics.accuracy_score(y_test, y_pred, normalize=True)
```

```
In [ ]: from sklearn.metrics import precision_recall_fscore_support
scores = precision_recall_fscore_support(y_test, y_pred, average='macro')
accuracy = metrics.accuracy_score(y_test, y_pred, normalize=True)
# Kết quả
print("Độ chính xác (precision): {:.7.2f}%".format(scores[0]*100))
print("Độ triệu hồi (recall): {:.7.2f}%".format(scores[1]*100))
print("Độ đo F1 (F1-measure): {:.7.2f}%".format(scores[2]*100))
print("Độ chính xác (accuracy): {:.7.2f}%".format(accuracy*100))
```

Độ chính xác (precision): 94.62%  
Độ triệu hồi (recall): 94.47%  
Độ đo F1 (F1-measure): 94.51%  
Độ chính xác (accuracy): 94.50%

**Nhận xét:** Sau khi chuẩn hoá và giảm chiều dữ liệu thì kết quả khá khả quan accuracy lên đến 94.50% tuy nhiên giá trị này vẫn thấp hơn khi không sử dụng phương pháp giảm chiều và chuẩn hoá. (So sánh với thực nghiệm 1)

## Summary

- Lab2: Dim Reduce (PCA,LDA), Thêm model (MLPClassifier, DecisionTreeClassifier)
  - Dùng GridSearch tìm mô hình và params tối ưu ==> Thử nghiệm: Sử dụng kết hợp giữ nguyên và giảm chiều dữ liệu và thêm 2 mô hình mới.  
--> Với bộ dataset trên không sử dụng PCA sẽ cho kết quả tốt nhất 0.97
  - Khi sử dụng PCA với tham số n\_component tối ưu nhất (0.95) thu được kết quả là dữ liệu mới có 4 chiều  
--> Kết quả khi sử dụng PCA là 0.94



# Brain Tumor Detection

Võ Thành Hoàng Sơn - 19DH110660

## 0. Dataset: Brian Tumor Dataset

url: <https://www.kaggle.com/datasets/preetviradiya/brian-tumor-dataset>

Được giới thiệu bởi: PREET VIRADIYA

Là một tập dữ liệu hình ảnh X-Ray bộ não người. Bao gồm 4600 mẫu trong đó

- 2087 hình ảnh bình thường
- 2513 hình ảnh có khối u

Dataset này dùng để nhận diện khối u trong hình ảnh X-ray não người

Dữ liệu đã được lưu trữ tại: [<https://drive.google.com/drive/folders/1qMj-VWgO3XeYPI4r-IPhodMJrpjTWnJ1?usp=sharing>]

## 1. Mount drive, import dataset, import lib

- Dataset url: <https://drive.google.com/drive/folders/1qMj-VWgO3XeYPI4r-IPhodMJrpjTWnJ1?usp=sharing>

```
In [ ]: url = 'https://drive.google.com/drive/folders/1qMj-VWgO3XeYPI4r-IPhodMJrpjTWnJ1?usp=sharing'
import gdown
gdown.download_folder(url, output='/content/dataset')

# unzip dataset from drive
!unzip -o -q "/content/dataset/Brain_Tumor_Dataset.zip"
```

```
Retrieving folder list
Processing file 1EHgCqXST9fzzIDzh0cbaU0f9IiShqyFT Brain_Tumor_Dataset.zip
Building directory structure completed

Retrieving folder list completed
Building directory structure
Downloading...
From: https://drive.google.com/uc?id=1EHgCqXST9fzzIDzh0cbaU0f9IiShqyFT
To: /content/dataset/Brain_Tumor_Dataset.zip
100%|██████████| 112M/112M [00:00<00:00, 267MB/s]
Download completed
```

```
In [ ]:
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
import cv2
from keras.models import Sequential
from keras import layers
import os
import matplotlib.pyplot as plt
import seaborn as sns
import sys

import warnings
warnings.filterwarnings("ignore")

from keras import Model

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

## 2. Load dataset

**Phương pháp:** Ta tiến hành chia dataset thành 3 tập:

- + Train data: 64%
- + Validation data: 16%
- + Test data: 20%

```
In [ ]: seed = 42
datapath = "/content/Brain Tumor Data Set/Brain Tumor Data Set"
metadata = tf.keras.utils.image_dataset_from_directory(datapath,
                                                       validation_split=.2,
                                                       shuffle=True,
                                                       subset='training',
                                                       seed = seed,
                                                       batch_size=None,
                                                       label_mode='binary')
testdata = tf.keras.utils.image_dataset_from_directory(datapath,
                                                       validation_split=.2,
                                                       shuffle=True,
                                                       subset='validation',
                                                       seed = seed,
                                                       batch_size=None,
                                                       label_mode='binary')

split = int(3612*.8)
traindata = metadata.take(split)
validdata = metadata.take(3612-split)
print("--*50")
print(f"Train data: {len(traindata)}")
print(f"Validation data: {len(validdata)}")
print(f"Test data: {len(testdata)}")
```

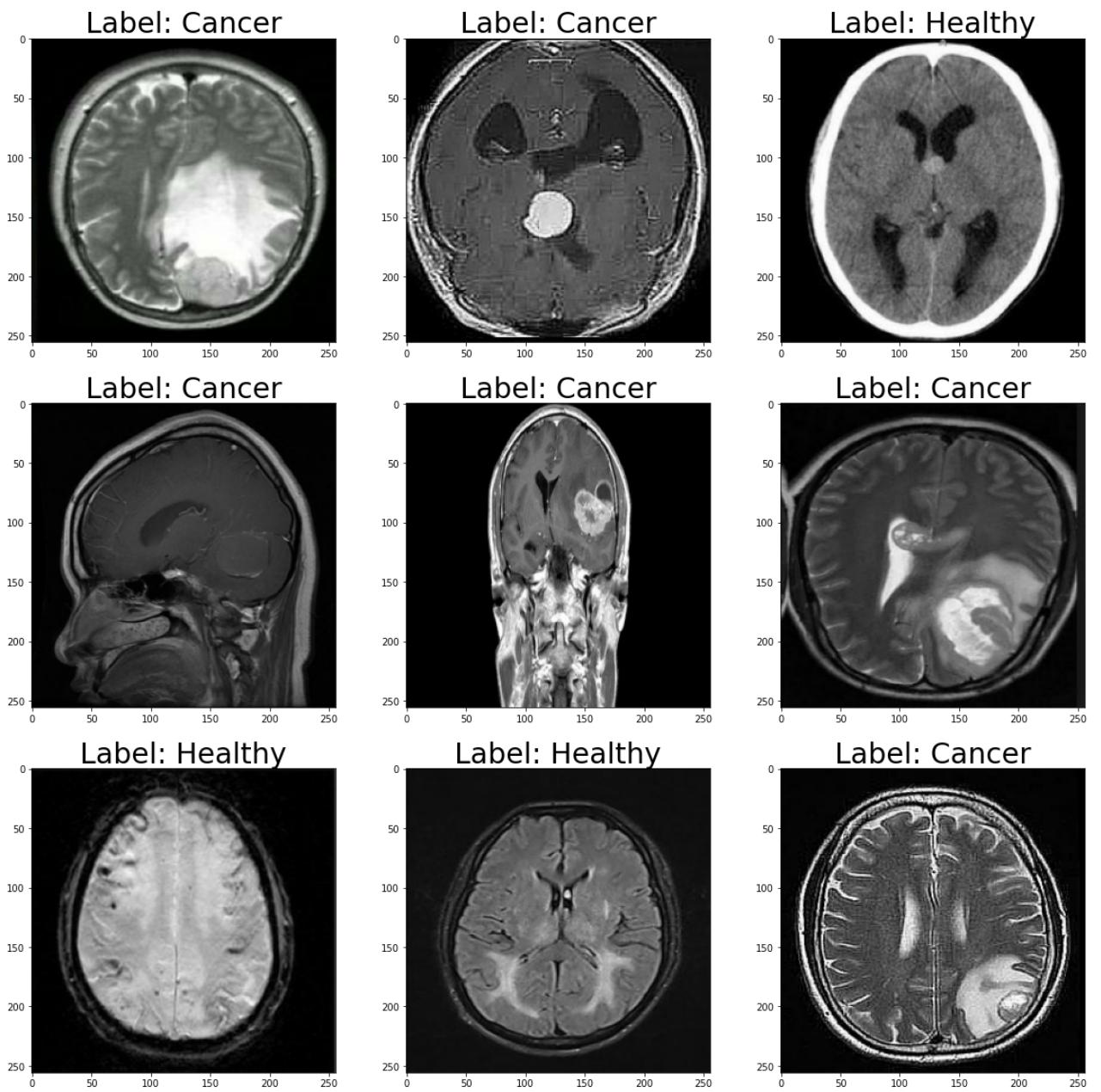
**Nhận xét:** Chỉ load được 4514 hình ảnh vì hàm image\_dataset\_from\_directory không load được hình ảnh dưới dạng .tiff

Sau khi load:

- Tập train: 2889 hình ảnh
- Tập valid: 723 hình ảnh
- Tập test: 902 hình ảnh

```
In [ ]: def decoder(code):
    if code == 0:
        return "Cancer"
    else: return "Healthy"
```

```
In [ ]: plt.figure(figsize = (20,20))
for idx in range(9):
    img,label = next(iter(traindata))
    plt.subplot(3,3,idx+1)
    plt.imshow(np.array(img).astype('uint8'))
    plt.title(f"Label: {decoder(label)}",fontdict={'fontsize':30})
```



**Tóm tắt:** Tập dữ liệu có 4514 mẫu là ảnh MRI não đã thu thập được. Trong đó các mẫu được chia thành 2 loại:

- 0: Cancer: là hình ảnh MRI người bị ung thư não
- 1: Healthy: là hình ảnh MRI người không bị ung thư não

### Phương pháp

1. Kiểm tra độ cân bằng của bộ dữ liệu
2. Số mẫu chỉ có 4516 mẫu tương đối ít cần tăng số lượng mẫu bằng cách:
  - Xoay hình ảnh
  - Tăng giảm độ sáng
  - Lật hình ảnh (flip)

## 2.1 Data balancing

```
In [ ]: # lấy label đúng của tập test

train_label = pd.DataFrame([int(label) for img,label in list(traindata)])
valid_label = pd.DataFrame([int(label) for img,label in list(validdata)])
target = pd.DataFrame([int(label) for img,label in list(testdata)])
```

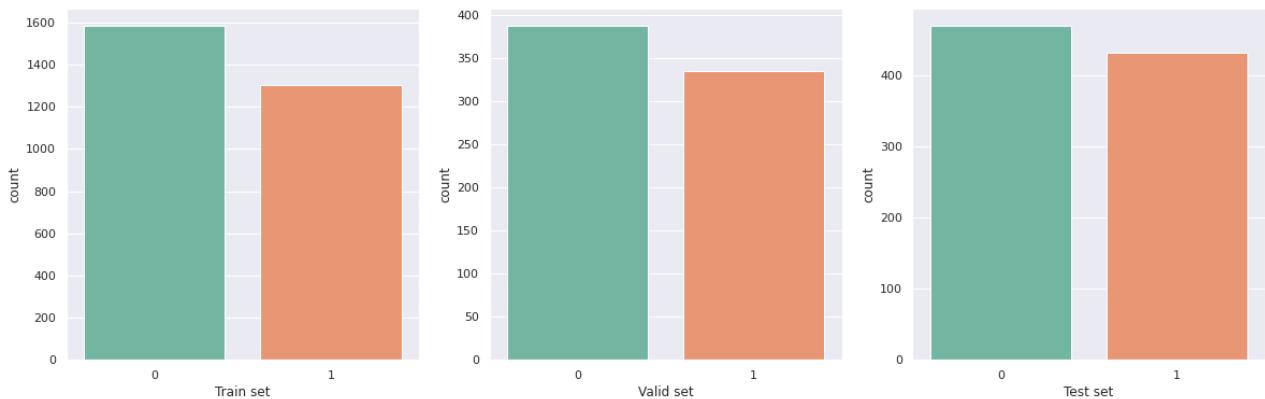
```
In [ ]: # Plot độ cân bằng dữ liệu
sns.set(style='darkgrid', palette='Set2')
plt.figure(figsize=(20,6))
plt.subplot(1,3,1)
sns.countplot(train_label[0])
```

```

plt.xlabel('Train set')
plt.subplot(1,3,2)
sns.countplot(valid_label[0])
plt.xlabel("Valid set")
plt.subplot(1,3,3)
sns.countplot(target[0])
plt.xlabel("Test set")

```

Out[ ]: Text(0.5, 0, 'Test set')



#### Nhận xét:

Dữ liệu trong 3 tập được chia khá tương đồng, không cần cân bằng dữ liệu

## 2.2 Data preprocessing

```

In [ ]:
AUTOTUNE = tf.data.AUTOTUNE
prepare_img = Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2)
])

resize_rescale = Sequential([
    layers.Resizing(256, 256),
    layers.Rescaling(1./255)
])

decoder_label = tf.keras.layers.CategoryEncoding(num_tokens=2, output_mode="one_hot")

def preprocessing(data, batch_size=16, repeat_count=False, random_transform=True):
    data = data.map(lambda img, label:(resize_rescale(img), decoder_label(label)),
                   num_parallel_calls=AUTOTUNE)

    if random_transform:
        data = data.map(lambda img, label:(prepare_img(img), label),
                      num_parallel_calls=AUTOTUNE)

    if batch_size != 0:
        data = data.batch(batch_size)
    if(repeat_count):
        data = data.repeat()

    data = data.prefetch(buffer_size=AUTOTUNE)
    return data

```

```

In [ ]:
batch_size=32
train = preprocessing(traindata, batch_size=batch_size, repeat_count=True)
valid = preprocessing(validdata, batch_size=batch_size)
test = preprocessing(testdata, batch_size=1, random_transform=False)

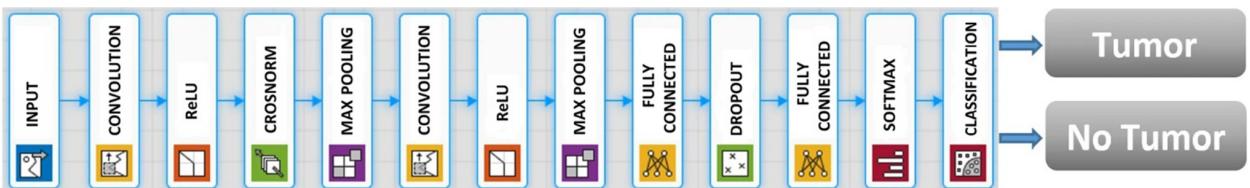
```

#### Nhận xét:

Dữ liệu sau khi được xử lý có kiểu DataPrefetch trong đó tập train và valid có batch là 32.

## 3. Define model

Sử dụng mô hình CNN có cấu trúc như sau:



src: <https://link.springer.com/article/10.1007/s40998-021-00426-9>

### Phương pháp huấn luyện:

- Chúng ta sẽ dùng hàm tối ưu là: Adam với hệ số học ban đầu là 1e-3
- Sử dụng phương pháp giảm hệ số học e-1 (ReduceLROnPlateau) nếu val\_loss không giảm sau mỗi 3 epochs
- Lưu trữ trọng số của mô hình (ModelCheckpoint) nếu val\_loss tốt hơn
- Sử dụng phương pháp early stopping để dừng quá trình huấn luyện nếu giá trị accuracy không tối ưu hơn sau mỗi 6 epochs
- Sử dụng hàm mất mát là binary\_crossentropy

```
In [ ]:
name_exp = 'Brain_tumor'
init_lr = 1e-3
epochs = 100
```

```
In [ ]:
model = Sequential([
    layers.Conv2D(128, kernel_size = 6, input_shape=(256, 256, 3),
                 activation ='relu', padding = 'same'),
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size = 2),

    layers.Conv2D(96, kernel_size = 6, activation ='relu', padding = 'same'),
    layers.MaxPool2D(pool_size = 2),

    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(2, activation='softmax')
])
```

```
In [ ]:
model.summary()

Model: "sequential_5"
-----
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 256, 256, 128)	13952
batch_normalization_1 (BatchNormalization)	(None, 256, 256, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 128, 128, 128)	0
conv2d_3 (Conv2D)	(None, 128, 128, 96)	442464
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 96)	0
flatten_1 (Flatten)	(None, 393216)	0
dense_2 (Dense)	(None, 512)	201327104
dropout_1 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 2)	1026

```
=====
Total params: 201,785,058
Trainable params: 201,784,802
Non-trainable params: 256
```

### Nhận xét:

- Số trọng số được huấn luyện trong mô hình là 201,784,802.
- Số trọng số không được huấn luyện là 256 (vì sử dụng Dropout 50%)

## 4. Training

```
In [ ]: optim = Adam(lr=init_lr)
early = EarlyStopping(monitor = 'val_accuracy', patience = 6)
reducclr = ReduceLROnPlateau( monitor='val_loss',
                             factor=tf.math.exp(-0.1),
                             patience=3,
                             verbose=1,
                             mode='auto',
                             min_lr=1e-5)
checkpoint = ModelCheckpoint(f'/content/drive/MyDrive/Shared/WritaleFolder/19DH110660_Võ Thành Hoàng
                             monitor='val_loss',
                             save_best_only=True,
                             mode='auto')
model.compile(optimizer=optim, loss='binary_crossentropy', metrics=['accuracy'])

In [ ]: model.fit(train, validation_data=valid, epochs=epochs, steps_per_epoch=100, verbose=True, callbacks=[reduc

Epoch 1/100
100/100 [=====] - 58s 567ms/step - loss: 7.5998 - accuracy: 0.6065 - val_los
s: 0.6902 - val_accuracy: 0.5491 - lr: 0.0010
Epoch 2/100
100/100 [=====] - 50s 502ms/step - loss: 0.4380 - accuracy: 0.8105 - val_los
s: 0.6838 - val_accuracy: 0.5740 - lr: 0.0010
Epoch 3/100
100/100 [=====] - 50s 501ms/step - loss: 0.2450 - accuracy: 0.9100 - val_los
s: 0.6211 - val_accuracy: 0.7234 - lr: 0.0010
Epoch 4/100
100/100 [=====] - 50s 506ms/step - loss: 0.1205 - accuracy: 0.9581 - val_los
s: 0.3949 - val_accuracy: 0.9433 - lr: 0.0010
Epoch 5/100
100/100 [=====] - 51s 511ms/step - loss: 0.0670 - accuracy: 0.9802 - val_los
s: 0.1369 - val_accuracy: 0.9945 - lr: 0.0010
Epoch 6/100
100/100 [=====] - 51s 510ms/step - loss: 0.0585 - accuracy: 0.9795 - val_los
s: 0.0619 - val_accuracy: 0.9959 - lr: 0.0010
Epoch 7/100
100/100 [=====] - 50s 503ms/step - loss: 0.0242 - accuracy: 0.9934 - val_los
s: 0.0068 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 8/100
100/100 [=====] - 50s 502ms/step - loss: 0.0210 - accuracy: 0.9946 - val_los
s: 0.0033 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 9/100
100/100 [=====] - 39s 394ms/step - loss: 0.0253 - accuracy: 0.9943 - val_los
s: 0.0034 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 10/100
100/100 [=====] - 55s 548ms/step - loss: 0.0207 - accuracy: 0.9956 - val_los
s: 0.0024 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 11/100
100/100 [=====] - 50s 502ms/step - loss: 0.0094 - accuracy: 0.9981 - val_los
s: 5.7574e-04 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 12/100
100/100 [=====] - 51s 510ms/step - loss: 0.0079 - accuracy: 0.9984 - val_los
s: 2.0846e-04 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 13/100
100/100 [=====] - 40s 398ms/step - loss: 0.0124 - accuracy: 0.9959 - val_los
s: 0.0018 - val_accuracy: 1.0000 - lr: 0.0010
Out[ ]: <keras.callbacks.History at 0x7f8169f2f110>
```

### Nhận xét:

- Sau khi train sau 13 epochs quá trình huấn luyện đã kết thúc sớm vì val\_accuracy đã đạt 1.0

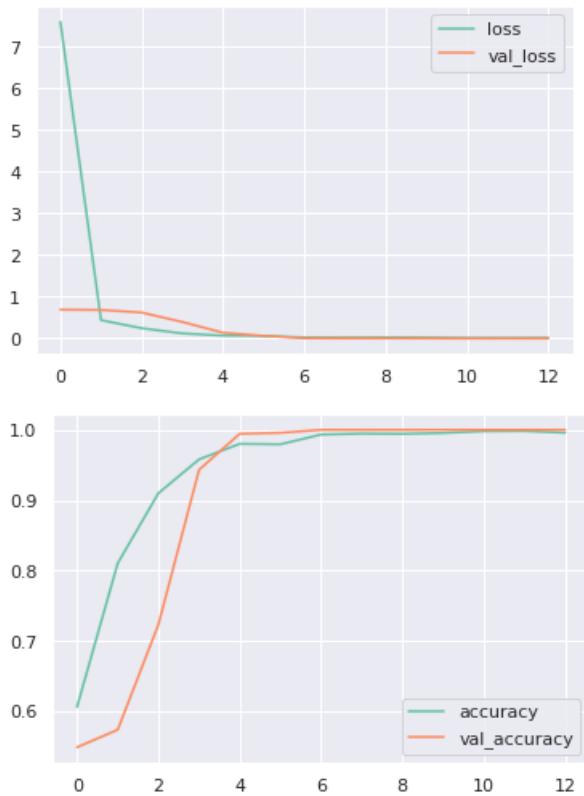
## 4.1 Plot history

```
In [ ]: log = pd.DataFrame(model.history.history)

log[['loss', 'val_loss']].plot()
log[['accuracy', 'val_accuracy']].plot()

## Save hist (Phải mount vào Drive)
# from google.colab import drive
# drive.mount('/content/drive')
```

```
# with open(f'/content/drive/MyDrive/Shared/WritaleFolder/19DH110660_Võ Thành Hoàng Sơn/Lab3_DeepLea
#     log.to_json(f)
```



**Nhận xét:** Sau khi huấn luyện qua 12 epochs

- Quá trình hội tụ rất nhanh khoảng 3 epochs giá trị hàm mất mát từ 0.6 xuống gần bằng 0
- Sau đó mô hình dừng như không có thay đổi quá nhiều
- Quá trình huấn luyện được kết thúc sớm vì đã đạt giá trị accuracy = 1

## 5. Evaluate

**Phương pháp:**

Ở đây ta tiến hành truyền lại trọng số tốt nhất trong quá trình huấn luyện và tiến hành kiểm tra trên tập test

```
In [ ]: ##Load best weight (Phải mount vào Drive)
# model.load_weights(f'/content/drive/MyDrive/Shared/WritaleFolder/19DH110660_Võ Thành Hoàng Sơn/Lab
# optim = Adam(lr=3e-4)
model.compile(optimizer=optim,
              loss='binary_crossentropy',
              metrics=['accuracy',tf.keras.metrics.Precision(),tf.keras.metrics.Recall()])
```

```
In [ ]: eval = model.evaluate(test)
np.round(eval,3)
```

```
902/902 [=====] - 15s 14ms/step - loss: 0.1328 - accuracy: 0.9812 - precision_1: 0.9812 - recall_1: 0.9812
Out[ ]: array([0.133, 0.981, 0.981, 0.981])
```

**Nhận xét:**

- Kết quả đánh giá sau là accuracy đạt 0.981 precision đạt 0.9812 với giá trị hàm mất mát là 0.133

### 5.1 Confusion matrix

**Phương pháp:**

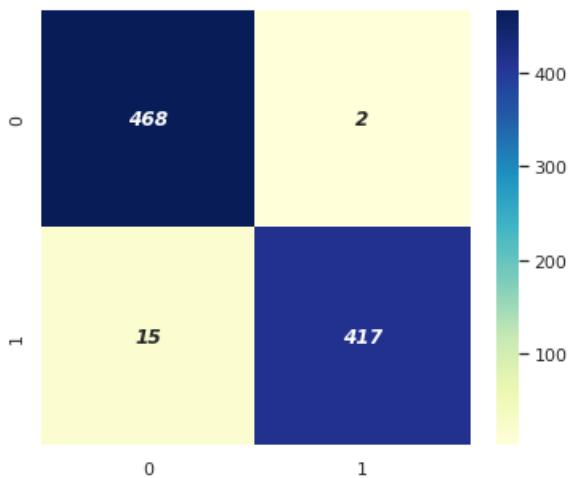
- Đầu tiên ta dự đoán các giá trị cho tập kiểm tra bằng hàm predict

- Sau đó ta chỉ số nhãn bằng argmax

```
In [ ]:  
true = []  
pred = []  
imgs = []  
for img,label in test:  
    true.append(np.argmax(label.numpy()[0]))  
    pred.append(np.argmax(model.predict(img)))  
    imgs.append(img[0])
```

```
In [ ]:  
plt.figure(figsize=(6,5))  
sns.heatmap(tf.math.confusion_matrix(true,pred),  
            fmt='d',  
            annot=True,  
            cmap="YlGnBu",  
            annot_kws={'fontsize':12,'fontstyle':'oblique','fontweight': 'heavy'})
```

Out[ ]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f80478ef710>

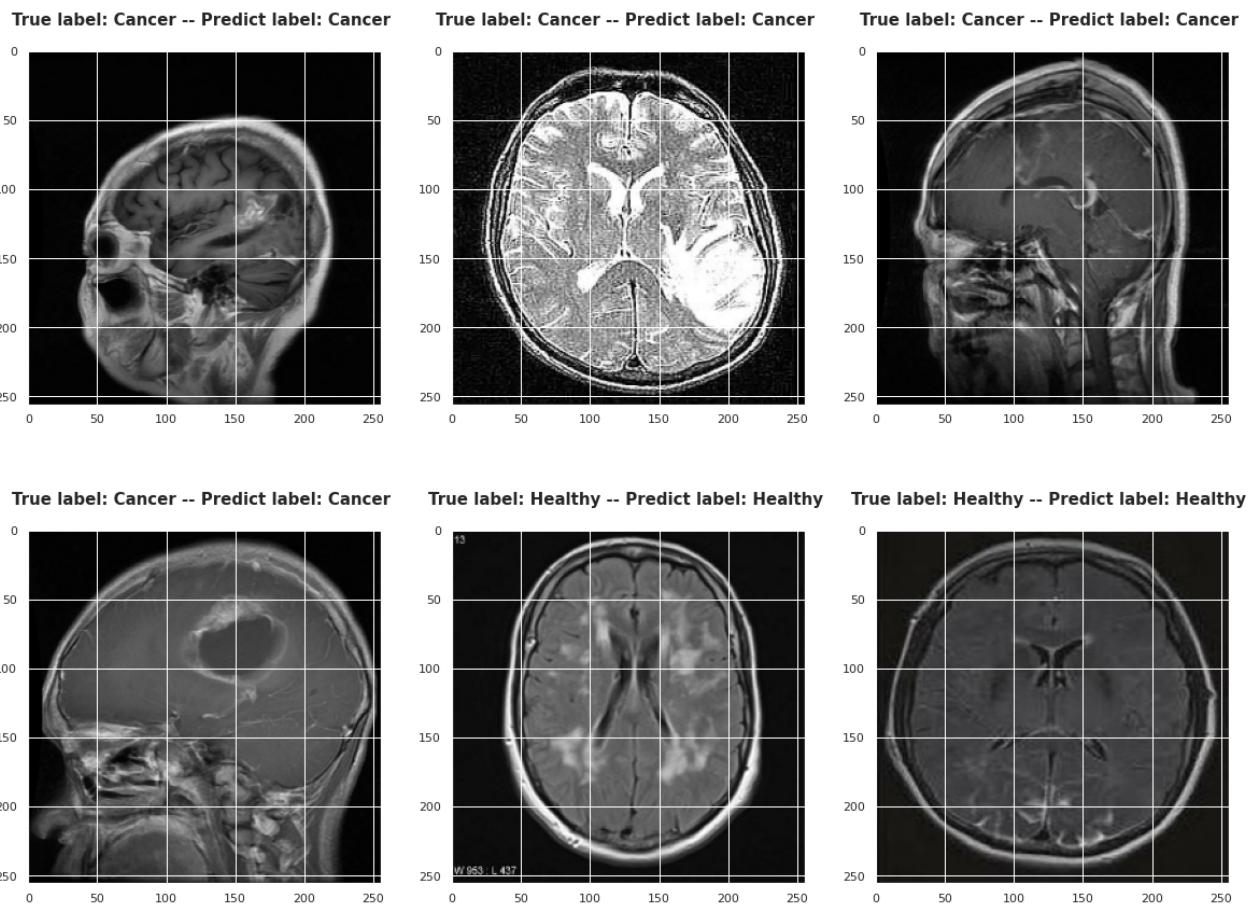


Nhận xét:

Sau khi predict trên tập test ta có thể thấy mô hình dự đoán vẫn còn sai sót với healthy là 2 nhãn đối với Cancer là 15 nhãn.

Số dự đoán sai khá ít tuy nhiên đối với bài toán về y học thì ta cần cải thiện hơn.

```
In [ ]:  
plt.figure(figsize=(20,15))  
for i in range(6):  
    j = np.random.randint(0,len(imgs),1)[0]  
    plt.subplot(2,3,i+1)  
    plt.imshow(imgs[j])  
    plt.title(f"True label: {decoder(true[j])} -- Predict label: {decoder(pred[j])}\n",  
              fontdict={'fontsize': 15,'fontweight':'semibold'},)
```



#### Nhận xét:

Khi thử hiển thị random một vài ảnh ta có thể thấy giá trị nhãn dự đoán tương đồng với nhãn thật.

## Summary:

1. Tải và load bộ dữ liệu Brain Tumor Dataset
2. Xử lý dữ liệu: Cân bằng và tiền xử lý đầu vào
3. Tạo mô hình
4. Huấn luyện mô hình
5. Đánh giá và dự đoán kết quả.

## Future work

- Tăng cường độ nhiễu cho hình ảnh đầu vào
- Bộ dữ liệu chỉ chưa hình ảnh MRI vì thế ta nên thêm các dạng ảnh scan khác như PET, MRA, CT ...

In [ ]:



# ML-Test

Tên: Võ Thành Hoàng Sơn

MSSV: 19DH110660

## BÀI KIỂM TRA CUỐI KỲ

- MÔN HỌC: HỌC MÁY
- PHẦN: TOÀN BỘ
- NGÀY 04/06/2022
- THỜI GIAN: 75 PHÚT
- LƯU Ý:
  - Tổng điểm: 12, lấy 10
  - Sinh viên làm bài trực tiếp vào tập tin này (điền vào chỗ: YOUR CODE HERE)

In [43]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

## 1- [2 điểm] Lập trình tính toán các lớp cơ bản

### 1.1 Lớp tuyến tính

$$Y = X \cdot W + b$$

In [44]:

```
# PHẢI THỰC HIỆN. Tuy nhiên, không có điểm cho câu này!!!
class Linear(object):
    def __init__(self, W, b):
        self.W = W
        self.b = b

    def forward(self, X):
        y = X @ self.W + self.b
        return y

    def __call__(self, X):
        return self.forward(X)
```

In [45]:

```
# Sử dụng lớp Linear
Nin, Nout = 4, 3
N = 5
W = np.random.normal(0, 1, (Nin, Nout))
b = np.random.normal(0, 1, (Nout))
X = np.random.normal(0, 1, (N, Nin))
Y = Linear(W, b)(X)
print(Y)

[[ -0.3455693   1.81065342 -1.9552755 ]
 [  2.29431823  3.15351633 -0.83774261]
 [  5.47494167 -2.00508756  3.82536293]
 [  0.54912803  0.61125932 -0.37622919]
 [  5.38103189 -2.82848317  4.31987607]]
```

### 1.2 Lớp Sigmoid:

In [46]:

```
class Sigmoid(object):
    def __init__(self):
        pass

    def forward(self, z):
        return 1.0/(1.0+np.exp(-z))
```

```

    def __call__(self, z):
        return self.forward(z)

In [47]: # Sử dụng lớp Softmax
Z = np.random.normal(0,1, (N, Nout))
Y = Sigmoid()(Z)
print("Z: "); print(Z)
print("Y: "); print(Y)

Z:
[[-0.83872375 -2.17947921 -1.09709846]
 [ 1.23980012 -0.28372141 -0.85824395]
 [ 0.76129064  1.46776484  0.78721627]
 [ 2.0066186  -0.13655734  1.17915749]
 [-0.20713672 -0.2017332  -0.01172525]]

Y:
[[0.30180364 0.10160846 0.25028395]
 [0.77552922 0.42954166 0.29770636]
 [0.68163388 0.81271742 0.6872333 ]
 [0.88149024 0.46591362 0.76479629]
 [0.44840018 0.44973704 0.49706872]]

```

## 1.3 Mô hình LogitRegression

```

In [48]: class LogitRegression(object):
    def __init__(self, W, b, threshold=[0.5]):
        self.linear = Linear(W, b)
        self.sigmoid = Sigmoid()
        self.threshold = threshold

    def predict(self, X):
        # Dự báo nhẫn cho các quan sát (hàng trong X)
        Y = self.linear(X)
        Y = self.sigmoid(Y)
        Y = Y > self.threshold
        return Y*1

    def __call__(self, X):
        # Gọi predict và trả về kết quả
        return self.predict(X)

```

```

In [49]: # Sử dụng SoftmaxRegression
Nin, Nout = 4, 3
N = 5
W = np.random.normal(0,1, (Nin, Nout))
b = np.random.normal(0, 1, (Nout))
X = np.random.normal(0,1, (N, Nin))
y = LogitRegression(W, b)(X)
print(y)

[[0 0 0]
 [0 1 1]
 [0 1 1]
 [0 0 0]
 [0 0 0]]

```

## 2- [2 điểm] Xây dựng mô hình học máy

- Tập dữ liệu: `sklearn.datasets.load_breast_cancer`
- Mô hình: `sklearn.linear_model.LogisticRegression`
- Yêu cầu:
  - Tải tập dữ liệu
  - Chia tách thành train/test tỉ lệ: 70%/30%
  - Tạo và huấn luyện mô hình nói trên với tập train
  - Đánh giá mô hình với tập test
    - In ra ma trận nhầm lẫn
    - In ra bảng đánh giá hiệu năng (precision/recall/F1/accuracy)
  - In ra các tham số đã học được của mô hình
    - Ghi nhận xét về hình dạng của các tham số và giải thích tại sao lại có hình dạng như vậy?

```
In [50]: # YOUR CODE HERE
## Import libs and load datasets
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import warnings
warnings.filterwarnings("ignore")

data, label = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.3, random_state=42)
```

```
In [51]: #Define model and training
model = LogisticRegression()
model.fit(X_train,y_train)
print("Tranin done!")
```

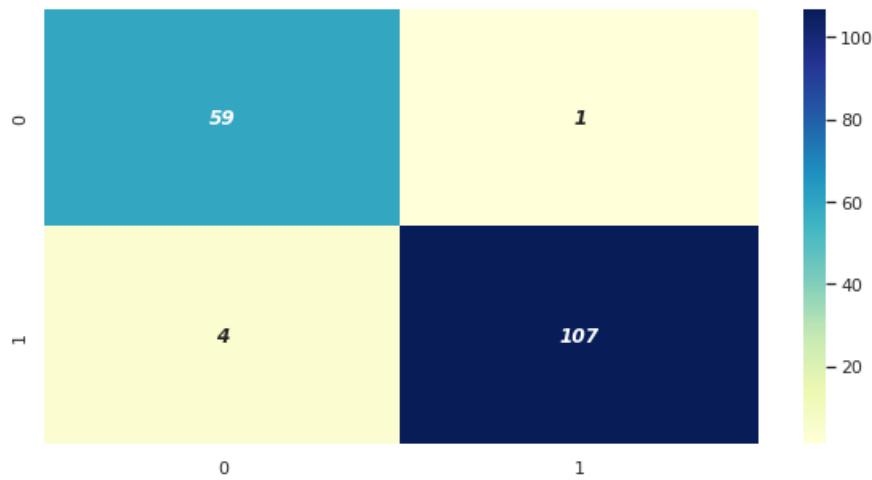
Tranin done!

```
In [52]: #Predict and evaluate results
pred = model.predict(X_test)
print("Predict done!")

#Confusion_matrix
cmatrix = confusion_matrix(pred,y_test)
plt.figure(figsize=(10,5))
sns.heatmap(cmatrix,
            fmt='d',
            annot=True,
            cmap="YlGnBu",
            annot_kws={'fontsize':12,'fontstyle':'oblique','fontweight': 'heavy'})

print("Ma trận nhầm lẫn: ")
# print(cmatrix)
```

Predict done!  
Ma trận nhầm lẫn:



```
In [53]: # Score model

metric_score = classification_report(pred,y_test)
print(metric_score)
```

	precision	recall	f1-score	support
0	0.94	0.98	0.96	60
1	0.99	0.96	0.98	111
accuracy			0.97	171
macro avg	0.96	0.97	0.97	171
weighted avg	0.97	0.97	0.97	171

## Ghi nhận xét vào đây

- Kết quả sau khi huấn luyện rất tốt:
  - Accuracy đạt 0.97

- Các điểm còn lại không bị lệch so với accuracy
- Từ confusion matrix ta có thể thấy số nhầm giữa 2 class không đồng đều tuy nhiên kết quả dự đoán rất tốt chỉ có 5 giá trị đánh nhầm sai trên tổng số tập test (171 mẫu)

### 3- [3 điểm] Dự báo nhãn bằng lớp LogitRegression

- Câu 1: Đã xây dựng lớp LogitRegression
- Câu 2: Đã xây dựng mô hình LogisticRegression dùng sklearn
- Yêu cầu:
  - Dùng tham số học được của mô hình trong Câu 2 để dự báo nhãn cho tập test, sử dụng lớp LogitRegression đã xây dựng ở Câu 1
  - In ra kết quả đánh giá cho tập test (bàng hiệu năng)

In [54]:

```
# YOUR CODE HERE

# Get params from part 2
bias = model.intercept_
weight = model.coef_
```

In [55]:

```
my_pred = LogitRegression(weight.T,bias)(X_test)
```

In [56]:

```
metric_score = classification_report(my_pred,y_test)
print(metric_score)
```

	precision	recall	f1-score	support
0	0.94	0.98	0.96	60
1	0.99	0.96	0.98	111
accuracy			0.97	171
macro avg	0.96	0.97	0.97	171
weighted avg	0.97	0.97	0.97	171

### 4 - [2 điểm] Bài tổng hợp

- Tập dữ liệu: [Sonar](#))
- Yêu cầu:
  - Hãy sử dụng kiến thức đã học để thực hiện việc: hiển thị trực quan dữ liệu, xây dựng mô hình, lựa chọn kỹ thuật, đánh giá mô hình cho tập dữ liệu nói trên
- Lưu ý:
  - Sau khi download về, sinh viên cần lưu lại trên gdrive với chế độ có thể download được bởi **gdown** và dùng file trên gdrive để làm bài

In [57]:

```
# YOUR CODE HERE
url = 'https://drive.google.com/drive/folders/16q005jQGDZDxq1M7GtCX1hPm4xcIKUQ0?usp=sharing'
import gdown
gdown.download_folder(url)
```

```
Retrieving folder list
Processing file 1tn7HaG6wLocTN3ustv44EeEnWfTJEExFC sonar.all-data
Processing file 1o6J01YrRCtWfWglsAqmcolIMj-tjnuGW sonar.names
Building directory structure completed
Retrieving folder list completed
Building directory structure
Downloading...
From: https://drive.google.com/uc?id=1tn7HaG6wLocTN3ustv44EeEnWfTJEExFC
To: /content/dataset/sonar.all-data
100%|[██████| 87.8k/87.8k [00:00<00:00, 42.8MB/s]
Downloading...
From: https://drive.google.com/uc?id=1o6J01YrRCtWfWglsAqmcolIMj-tjnuGW
To: /content/dataset/sonar.names
100%|[██████| 5.87k/5.87k [00:00<00:00, 1.86MB/s]
Download completed
Out[57]: ['/content/dataset/sonar.all-data', '/content/dataset/sonar.names']
```

In [58]:

```
col_names = [i for i in range(60)]
col_names.append('class')
data = pd.read_csv('/content/dataset/sonar.all-data', names= col_names)
print(f'Shape: {data.shape}')
data.head()
```

Shape: (208, 61)

Out[58]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110

5 rows × 61 columns

Nhận xét: dataset src: '[https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+\(Sonar,+Mines+vs.+Rocks\)](https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks))'

- Tổng quan dữ liệu: Dataset là những mẫu dữ liệu tín hiệu sonar được dò ra trong quá trình dò mìn.
- Mỗi mẫu là một bộ 60 số trong phạm vi từ 0,0 đến 1,0. Mỗi số đại diện cho năng lượng trong một dải tần số cụ thể, được tích hợp trong một khoảng thời gian nhất định.
- Trong đó bao gồm 2 lớp:
  - R: là nhãn cho Đá
  - M: là nhãn cho Mìn
- Bộ dữ liệu gồm 207 mẫu.

In [59]:

```
data.info()
print('---'*50)
print(f'Dữ liệu trùng lặp: {data.duplicated().sum()}' )
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208 entries, 0 to 207
Data columns (total 61 columns):
 #   Column  Non-Null Count  Dtype  
 --- 
 0   0       208 non-null    float64
 1   1       208 non-null    float64
 2   2       208 non-null    float64
 3   3       208 non-null    float64
 4   4       208 non-null    float64
 5   5       208 non-null    float64
 6   6       208 non-null    float64
 7   7       208 non-null    float64
 8   8       208 non-null    float64
 9   9       208 non-null    float64
 10  10      208 non-null    float64
 11  11      208 non-null    float64
 12  12      208 non-null    float64
 13  13      208 non-null    float64
 14  14      208 non-null    float64
 15  15      208 non-null    float64
 16  16      208 non-null    float64
 17  17      208 non-null    float64
 18  18      208 non-null    float64
 19  19      208 non-null    float64
 20  20      208 non-null    float64
 21  21      208 non-null    float64
 22  22      208 non-null    float64
 23  23      208 non-null    float64
 24  24      208 non-null    float64
 25  25      208 non-null    float64
 26  26      208 non-null    float64
 27  27      208 non-null    float64
 28  28      208 non-null    float64
 29  29      208 non-null    float64
 30  30      208 non-null    float64
 31  31      208 non-null    float64
 32  32      208 non-null    float64
 33  33      208 non-null    float64
 34  34      208 non-null    float64
 35  35      208 non-null    float64
```

```
36 36      208 non-null    float64
37 37      208 non-null    float64
38 38      208 non-null    float64
39 39      208 non-null    float64
40 40      208 non-null    float64
41 41      208 non-null    float64
42 42      208 non-null    float64
43 43      208 non-null    float64
44 44      208 non-null    float64
45 45      208 non-null    float64
46 46      208 non-null    float64
47 47      208 non-null    float64
48 48      208 non-null    float64
49 49      208 non-null    float64
50 50      208 non-null    float64
51 51      208 non-null    float64
52 52      208 non-null    float64
53 53      208 non-null    float64
54 54      208 non-null    float64
55 55      208 non-null    float64
56 56      208 non-null    float64
57 57      208 non-null    float64
58 58      208 non-null    float64
59 59      208 non-null    float64
60  class   208 non-null    object
dtypes: float64(60), object(1)
memory usage: 99.2+ KB
```

Dữ liệu trùng lắp: 0

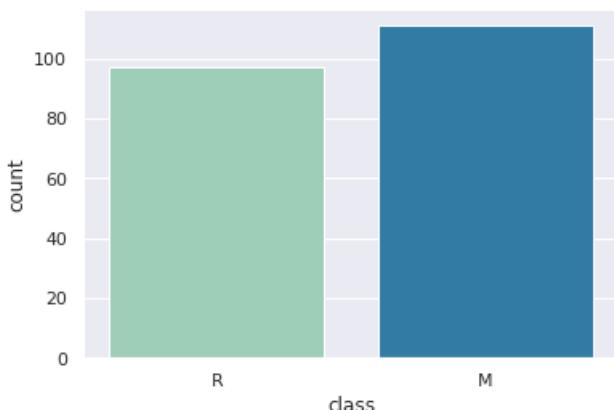
#### Nhận xét:

- Tập dữ liệu này không bị mất giá trị.
- Tất cả đặc trưng có kiểu dữ liệu là float
- Lớp class có kiểu dữ liệu là Object
- Không có dữ liệu bị trùng lắp

In [60]:

```
sns.set()
sns.set_style('darkgrid')
sns.countplot(x="class", data=data, palette='YlGnBu')
```

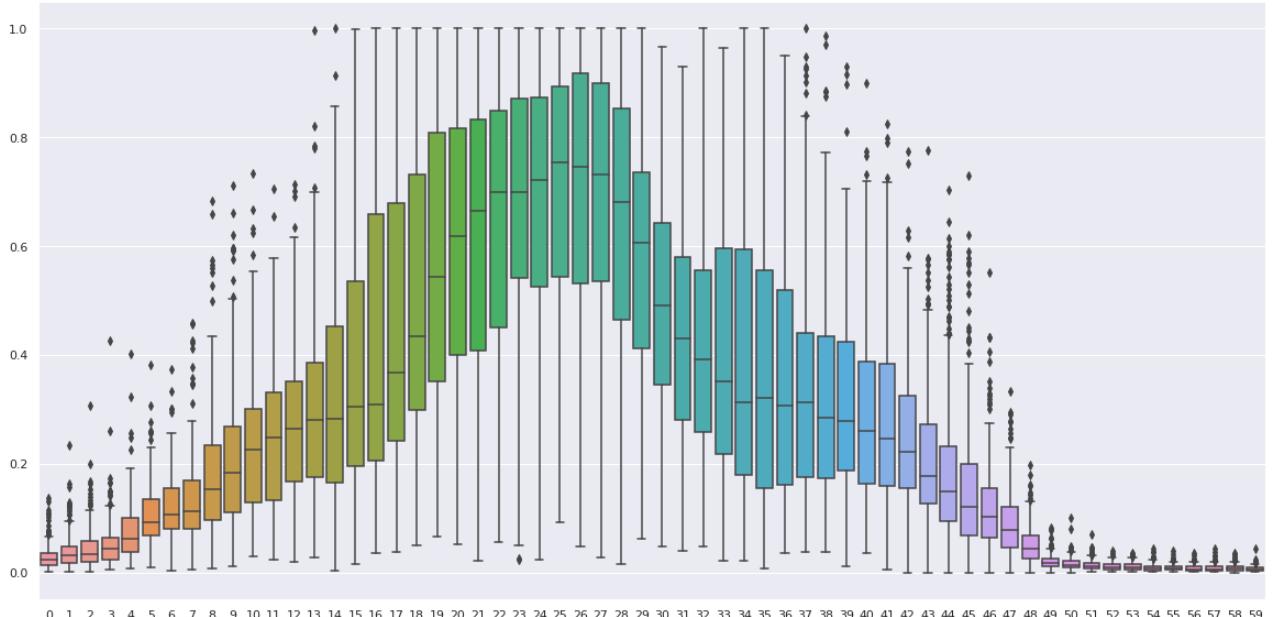
Out[60]:



Nhận xét: Tập dữ liệu phân lớp tương đối đồng đều. Không cần tăng cường dữ liệu

In [61]:

```
plt.figure(figsize=(20,10))
sns.boxplot(data = data)
plt.show()
```



### Nhận xét:

- Mỗi đặc trưng đều có giá trị trong khoảng từ 0 đến 1
- Tuy nhiên những đặc trưng từ 0-15 và từ 38-59 có nhiều outlier

### Phương pháp:

- Tiến hành gridsearch để tìm tham số tốt nhất
- Có thể sử dụng phương pháp chuẩn hoá sẽ cho kết quả tốt hơn.

In [62]:

```
##Gridsearch
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFold

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer, StandardScaler, FunctionTransformer
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
!pip install pipelinehelper -q
from sklearn.pipeline import Pipeline
from pipelinehelper import PipelineHelper

import warnings
warnings.filterwarnings("ignore")

## Pipeline params

params = {}
params['data_split_train'] = 0.7
params['random_state'] = 54
params['k_fold'] = 3

pipe = Pipeline([
    ('scaler', PipelineHelper([
        ('std', StandardScaler()),
        ('normal', Normalizer()),
        ('non', FunctionTransformer(None, validate=False)),
    ])),
    ('classifier', PipelineHelper([
        ('svm', SVC()),
        ('lr', LogisticRegression()),
    ])),
])

# Các cấu hình thử nghiệm
parameters = {
    'scaler_selected_model': pipe.named_steps['scaler'].generate({
        'std_with_mean': [True],
        'std_with_std': [False],
        'normal_norm': ['l1', 'l2', 'max']
    })
}
```

```

    }),

'classifier_selected_model': pipe.named_steps['classifier'].generate({
    'svm_C': [1.0, 10.0],
    'svm_kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'svm_degree': [2, 3, 5, 6, 10],


    'lr_fit_intercept': [True, False],
    'lr_solver': ['saga', 'liblinear', 'newton-cg'],
    'lr_max_iter': [500, 1000],
})

# Lựa chọn cấu hình: kiểm thử chéo (chia X_train thành 5 phần)
grid = GridSearchCV(pipe, parameters, cv=parameters['k_fold'], scoring='accuracy', verbose=1)

```

In [ ]:

```

#Train valid split
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, 0:-1],
                                                    data['class'],
                                                    random_state=parameters["random_state"],
                                                    train_size=parameters['data_split_train'])

grid1 = GridSearchCV(pipe,
                     parameters,
                     cv=2,
                     scoring='accuracy',
                     verbose=1)

#Fit and train model
grid.fit(X_train, y_train)

print("-" * 80)
print("Cấu hình tốt nhất: ", grid.best_params_)
print("Độ chính xác: {:.6f}".format(grid.score(X_test, y_test)))

```

Fitting 3 folds for each of 260 candidates, totalling 780 fits

Nhận xét:

- Cấu hình tốt nhất cho bài toán phân lớp này là SVM với hyperparameter như trên.
- Kết quả nhận được là 0.84

## 5 - [3 điểm] Mô hình HMM

Từ bài toán chọn bi trong các bình như được trình bày trong bài giảng. Hãy thực hiện:

### 1. Tự cho một bài toán tương tự có cấu hình cơ bản sau đây

- Số bình: có 4 bình
- Số màu bi trong mỗi bình: 5 màu, {Red, Green, blue, Cyan, Yellow}
- Lưu ý: cần phải đảm bảo ràng buộc về phân phối xác suất trong mô tả mô hình

### 1. Viết chương trình:

- 2.1 Sinh ra một dãy các màu (bi): số lượng 20
- 2.2 Tính xác suất của dãy quan sát:
  - (2.2.1) Red
  - (2.2.2) Red, Blue
  - (2.2.3) Red, Blue, Blue

In [ ]: # YOUR CODE HERE

In [ ]:

```

bi = ['Red', 'Green', 'Blue', 'Cyan', 'Yellow']
label = np.random.choice(bi, 20)
print(f"Observations: {label}")

```

In [ ]:

```

class HMM(object):
    def __init__(self, Pstart, Ptrans, Pvalues, state_names, value_names):
        self.Pstart = Pstart
        self.Ptrans = Ptrans
        self.Pvalues = Pvalues
        self.state_names = state_names
        self.value_names = value_names

```

```

nstates = np.array([Pstart.shape[0], Ptrans.shape[0], Ptrans.shape[1], Pvalues.shape[1]])
if ~np.all(nstates == nstates[0]):
    raise Exception("Number of states: mismatched")
else:
    self.num_states = nstates[0]
    self.num_values = Pvalues.shape[0]

self.current_state = -1 #the start state

@staticmethod
def createHMM():
    Pstart = np.array([0.4, 0.2, 0.2, 0.2])
    Ptrans = np.array([
        [0.1, 0.4, 0.1, 0.2],
        [0.3, 0.3, 0.2, 0.2],
        [0.5, 0.4, 0.2, 0.1],
        [0.1, 0.3, 0.3, 0.3]
    ])

    Pvalues = np.array([
        [0.2, 0.4, 0.2, 0.2],
        [0.2, 0.2, 0.1, 0.2],
        [0.2, 0.2, 0.6, 0.2],
        [0.2, 0.1, 0.05, 0.2],
        [0.2, 0.1, 0.05, 0.2]
    ])

    state_names = {
        0: "Bình 1",
        1: "Bình 2",
        2: "Bình 3",
        3: "Bình 4"
    }
    value_names = {
        0: "Đỏ",
        1: "Vàng",
        2: "Xanh",
        3: "Tím",
        4: "hồng"
    }

    return HMM(Pstart, Ptrans, Pvalues, state_names, value_names)

def __next_state(self):
    if self.current_state == -1:
        prob = self.Pstart
    else:
        prob = self.Ptrans[self.current_state,:]
    state = np.random.choice(np.arange(self.num_states), p=prob)
    self.current_state = state
    return state

def __sample_on_state(self, state):
    prob = self.Pvalues[:, state]
    value = np.random.choice(list(self.value_names.values()), p=prob)
    return value

def sample(self, size):
    results = []
    for c in np.arange(size):
        state = self.__next_state()
        value = self.__sample_on_state(state)
        results.append(value)
    return np.array(results)

def likelihood(self, obs, return_detail=True):
    num_observs = len(obs)
    if num_observs == 0:
        return 0
    if return_detail:
        prob_table = np.zeros((self.num_states, num_observs))
        prob_table[:, 0] = self.Pstart * self.Pvalues[obs[0], :]
        for t in range(1, num_observs):
            prob = prob_table[:, t-1]
            prob_table[:, t] = (self.Ptrans.T @ prob) * self.Pvalues[obs[t], :]
        prob = np.sum(prob_table[:, -1])
        return prob, prob_table

```

```
    else:
        prob = self.Pstart*self.Pvalues[obs[0],:]
    for t in range(1, num_observs):
        prob = (self.Ptrans.T @ prob) * self.Pvalues[obs[t], :]
    prob = np.sum(prob)

    return prob
```

In [ ]:

```
red = 0; blue = 1;
model = HMM.createHMM()
obs1 = np.array([red])
obs2 = np.array([red, blue])
obs3 = np.array([red, blue, blue])
names = ["Red", "Red, Blue", "Red, Blue, Blue"]
obs_array = [obs1,obs2,obs3]
for i in range(3):
    prob = model.likelihood(obs_array[i], True)
    print(f"Observation: {names[i]}")
    print(prob)
    print("---"*50)
```