Link trang web: https://share.streamlit.io/lekiet258/project03_ds/app.py (https://share.streamlit.io/lekiet258/project03_ds/app.py)

BẢNG PHÂN CÔNG

MSSV	Họ và tên	% đóng góp (tối đa 100%)	Chi tiết công việc
19120511	Võ Văn Hiếu	100	Tiền xử lý tiếng Việt + mô hình Random Forest
19120526	Huỳnh Đức Huy	100	Mô hình Decision Tree + code/deploy trang Web
19120539	Vương Thế Khang	100	EDA (Khám phá dữ liệu) + code Streamlit
19120554	Lê Kiệt	100	Mô hình Linear Regression Web + Soạn báo cáo
19120586	Nguyễn Phát Minh	100	Mô hình Logistic Regression + xây dựng Pipeline quá trình huấn luyện

THƯ VIỆN

- Note: trong Project nhóm có sử dụng thư viện `underthesea` . Nếu thầy cô chưa có thư viện này có thể tháo comment cell phía dưới để cài đặt thư viện
- Link tham khảo: https://underthesea.readthedocs.io/en/latest/readme.html (https://underthesea.readthedocs.io/en/latest/readme.html)

In [1]:
```
# !pip install underthesea
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import joblib
import requests

from underthesea import word_tokenize

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics import confusion_matrix
```

I. KHÁM PHÁ DỮ LIỆU

Đọc dữ liệu từ file "vn_news_223_tdlfr.csv" và lưu vào dataframe `news_df` .

In [3]:
```
news_df = pd.read_csv('vn_news_223_tdlfr.csv')
news_df.head()
```

Out[3]:

	text	domain	label
0	Thủ tướng Abe cúi đầu xin lỗi vì hành động phi...	binhluan.biz	1
1	Thủ tướng Nhật cúi đầu xin lỗi vì tinh thần ph...	www.ipick.vn	1
2	Choáng! Cơ trưởng đeo khăn quàng quẩy banh nóc...	tintucqpvn.net	1
3	Chưa bao giờ nhạc Kpop lại dễ hát đến thế!!!\n...	tintucqpvn.net	1
4	Đại học Hutech sẽ áp dụng cải cách "Tiếq Việt"...	www.gioitreviet.net	1

Dữ liệu có bao nhiều dòng và bao nhiêu cột?

In [4]:
```
news_df.shape
```

Out[4]:
```
(223, 3)
```

Vậy dữ liệu có kích thước **223 dòng x 3 cột**

Mỗi dòng có ý nghĩa gì? Có vấn đề các dòng có ý nghĩa khác nhau không?

Quan sát sơ bộ dữ liệu ta thấy mỗi dòng chứa thông tin về một bài báo, có vẻ như không có vấn đề các dòng có ý nghĩa khác nhau.

Dữ liệu có các dòng bị lặp không?

Kiểm tra xem dữ liệu có các dòng bị lặp không và lưu kết quả vào biến `have_duplicated_row` . Biến này sẽ có giá trị True nếu dữ liệu có các dòng bị lặp và có giá trị False nếu ngược lại.

In [5]:
```
have_duplicated_row = all(news_df.duplicated())
have_duplicated_row
```

Out[5]:
```
False
```

Như vậy, không có dòng nào bị lặp lại.

Mỗi cột có ý nghĩa gì?

Thông tin về các cột như sau:

- **text:** nội dung của bài báo
- **domain:** tên miền (website)
- **label:** nhãn (1: tin giả, 0: tin thật)

Mỗi cột hiện đang có kiểu dữ liệu gì? Có cột nào có kiểu dữ liệu chưa phù hợp để có thể xử lý tiếp không?

Xem thử kiểu dữ liệu của các cột dữ liệu

In [6]:
```
news_df.dtypes
```

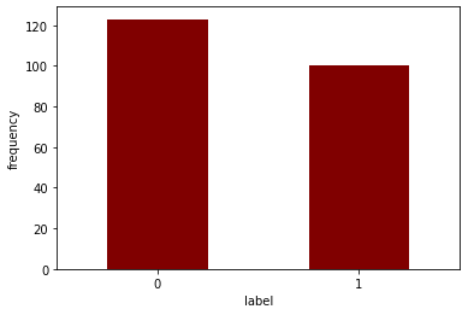
Out[6]:
```
text      object
domain    object
label      int64
dtype: object
```

→ Có vẻ các cột đều có kiểu dữ liệu phù hợp. Nếu trong quá trình phân tích dữ liệu cần phải thay đổi kiểu dữ liệu của các cột thì ta sẽ quay lại tiền xử lý ở đây, tạm thời ta chấp nhận kiểu dữ liệu hiện tại của các cột.

Kiểm tra phân bố các lớp/nhãn

Cột `label` có 2 giá trị 1 hoặc 0. Ta sẽ xem phân bố của 2 nhãn này

In [7]:
```
news_df['label'].value_counts().plot.bar(xlabel = 'label', ylabel = 'frequency', rot = 0, color = 'maroon');
```



→ Phân bố 2 nhãn không quá chênh lệch

Các thông tin thống kê của văn bản

Các thông tin thống kê bao gồm:

- Chiều dài trung bình của mỗi record.
- ...

In [8]:
```
print('Chiều dài trung bình của các text:', news_df['text'].apply(len).mean().round())
```

```
Chiều dài trung bình của các text: 2540.0
```

II. TIỀN XỬ LÝ VĂN BẢN TIẾNG VIỆT

Ở phần này, ta tiền xử lý văn bản cho cột `text` để chuẩn bị cho pipeline xử lý lúc sau. Mỗi phần tử trong pipeline là 1 class "transformer" nên để làm việc với pipeline lúc sau, định nghĩa 1 class "transformer" `TextReducer` thừa kế 2 class `BaseEstimator` , `TransformerMixin` dùng để tiền xử lý văn bản tiếng Việt với 3 phương thức

- **init()**: trong đây lưu trữ danh sách các stopwords được lấy từ link stopwords (https://raw.githubusercontent.com/stopwords/vietnamese-stopwords/master/vietnamese-stopwords.txt). Danh sách được lưu vào `self.stopwords`
- **fit()**: nhận 2 tham số X là 1 danh sách các đoạn text, y là danh sách label. Tuy nhiên trong ngữ cảnh này ta sẽ không tùy chỉnh hàm fit mà trả về chính nó (self) luôn
- **transform()**: nhận 1 tham số X là danh sách các đoạn text cần tiền xử lý, trả về 1 Series các đoạn text đã qua tiền xử lý. Trong hàm này, tiền xử lý văn bản tiếng Việt theo các bước sau:
  - Bước 1: loại bỏ các đường dẫn URL (Ví dụ: http, https)
  - Bước 2: loại bỏ ký tự đặc biệt ([@#/!.'"“”–+-=()%) và thay dấu dấu xuống hàng (\r\n) thành dấu cách

- Bước 3: đổi các đoạn text đang có cả chữ hoa và chữ thường thành toàn bộ chữ thường
- Bước 4: tokenize các đoạn văn bản để với mỗi văn bản chỉ còn là danh sách các từ khóa nội dung của từng văn bản đó. Nhóm quyết định sử dụng hàm `word_tokenize` của thư viện `underthesea` để tokenize các từ tiếng Việt
- Bước 5: loại bỏ stopwords dựa vào danh sách stopwords đã lưu vào `self.stopwords` .

In [9]:
```
class TextReducer(BaseEstimator, TransformerMixin):
    stopwords_raw_url = "https://raw.githubusercontent.com/stopwords/vietnamese-stopwords/master/vietnamese-stopwords.txt"

    def __init__(self):
        self.stopwords = requests.get(self.stopwords_raw_url).text.split('\n')

    def fit(self, X, y = None):
        return self

    def transform(self, X):
        _X = pd.Series(X)

        # Bước 1
        _X = _X.apply(lambda text: re.sub(r'http(s?)\S+.', '', text))
        # Bước 2
        _X = _X.apply(lambda text: re.sub(r'[@#/!.\'‘’\"“”–+-=()%]', '', text))
        _X = _X.apply(lambda text: re.sub(r'\r\n', ' ', text))
        # Bước 3
        _X = _X.apply(lambda text: text.lower())
        # Bước 4
        _X = _X.apply(word_tokenize)
        # Bước 5
        _X = _X.apply(lambda words: ' '.join([word for word in words if word not in self.stopwords]))

        return _X
```

III. MÔ HÌNH HÓA

Ở phần này, nhóm sử dụng 2 mô hình tuyến tính (Linear Regression và Logistic Regression) + 2 mô hình phi tuyến (Random Forest và Decision Tree). Để sử dụng dữ liệu văn bản cho mô hình dự đoán, văn bản phải được phân tích cú pháp để loại bỏ một số từ nhất định (chính là bước tiền xử lý tiếng Việt ở trên). Sau đó, những từ này cần được mã hóa dưới dạng số nguyên hoặc giá trị dấu phẩy động, để sử dụng làm đầu vào trong thuật toán học máy. Quá trình này được gọi là **trích xuất đặc trưng** (hay vectơ hóa/vectorize). Như vậy, với mỗi mô hình, sử dụng 2 vectorizer khác nhau:

- `CountVectorizer()` : dùng để tính tần số xuất hiện của từng token trong văn bản (token có thể là 1 từ hoặc nhiều từ - đã có ở bước tokenize trong tiền xử lý văn bản tiếng Việt)
- `TfidfVectorizer()` : dùng để tính độ quan trọng (tf-idf) của từng token trong văn bản. Những token có giá trị tf-idf cao là những token xuất hiện nhiều trong văn bản này và xuất hiện ít trong các văn bản khác. Việc này giúp lọc ra những từ phổ biến và giữ lại những từ có giá trị cao (từ khoá của văn bản đó)

Để thuận tiện cho việc gọi hàm ở cell sau đó (vì 4 mô hình x 2 vectorizer = 8 mô hình nên nếu không dùng hàm sẽ rất bất tiện và dài dòng), tại cell này làm những công việc sau:

- Lưu 4 mô hình vào dictionary `classifiers` với 4 key là tên viết tắt của tên mô hình như code
- Lưu 2 vectorizer vào dictionary `vectorizers` , cách viết key tương tự như của biến `classifiers`
- Lưu class TextReducer dùng để tiền xử lý văn bản tiếng Việt vào dictionary 1 phần tử `preprocessors`

In [10]:
```
preprocessors = {'tr':TextReducer()}

vectorizers = {'cv':CountVectorizer(),
               'tv':TfidfVectorizer()}

classifiers = {'li':LinearRegression(),
               'lo':LogisticRegression(random_state=42),
               'dt':DecisionTreeClassifier(random_state=42),
               'rf':RandomForestClassifier(random_state=42)}
```

Định nghĩa hàm `generate_models` nhận các tham số

- `preprocessors` , `vectorizers` , `classifiers` : chính là 3 dictionary ở cell phía trên
- `X_train` , `y_train` : bộ dữ liệu huấn luyện. Trong đó y_train là Series 1 cột thể hiện nhãn 1 hoặc 0 và X_train là 1 dataframe có 2 cột text, domain
- `out` : tên **folder** dùng để lưu các file mô hình (file nhị phân có đuôi .pkl)

Công việc của hàm `generate_models` :

- Lần lượt nhóm từng phương thức vector hóa trong `vectorizers` với từng mô hình trong `classifiers` (do hiện tại `preprocessors` chỉ có 1 phần tử nên không quan trọng lắm). Với mỗi lần nhóm:
  - Tạo ra pipeline gồm 3 bước: tiền xử lý + vector hóa + mô hình phân lớp
  - Tiến hành fit tập dữ liệu train vào pipeline vừa tạo
  - Dự đoán nhãn cho tập train, ta sẽ lưu kết quả dự đoán này vào dictionary `prediction`
  - Lưu mô hình vừa huấn luyện vào file nhị phân .pkl. file này được đặt tên:

    <preprocessor sử dụng>_<vectorizer sử dụng>_<classifier sử dụng>.pkl. Trong đó:

	Viết tắt
preprocessor sử dụng	tr = Text Reducer
vectorizer sử dụng	cv = Count Vectorizer tv = Tfidf Vectorizer

	Viết tắt
classifier sử dụng	li = Linear Regression lo = Logistic Regression dt = Decision Tree rf = Random Forest

- Kết quả trả về của hàm là một dictionary lưu trữ các kết quả dự đoán, để lấy ra mỗi kết quả dự đoán của mô hình ta dựa vào `preprocessor`, `vectorizer` và `classifier` ứng với mô hình đó (giống với cách đặt tên file .pkl). Đồng thời, việc lưu mô hình vào file nhị phân .pkl sẽ tiện cho việc deloy app bằng Streamlit

In [11]:
```
def generate_models(preprocessors, vectorizers, classifiers, X_train, y_train, out):
    prediction = dict()

    for p in preprocessors:
        for v in vectorizers:
            for c in classifiers:
                pipeline = Pipeline([('text_reducer', preprocessors[p]),
                                     ('vectorizer'  , vectorizers[v]),
                                     ('classifier'  , classifiers[c])])
                # train mô hình
                pipeline.fit(X_train, y_train)

                # lưu kết quả dự đoán để đánh giá mô hình
                y_pred = pipeline.predict(X_train)
                prediction[p + '_' + v + '_' + c] = y_pred

                # lưu mô hình vào file .pkl
                filepath = out.rstrip('\\').rstrip('/') + '/' + p + '_' + v + '_' + c + '.pkl'
                joblib.dump(pipeline, filepath, compress = 1)

    return prediction
```

In [12]:
```
X_train = news_df['text']
y_train = news_df['label']

prediction = generate_models(preprocessors, vectorizers, classifiers, X_train, y_train, 'models')
pd.DataFrame(prediction).tail(5) # chạy khoảng ~5p
```

Out[12]:

	tr_cv_li	tr_cv_lo	tr_cv_dt	tr_cv_rf	tr_tv_li	tr_tv_lo	tr_tv_dt	tr_tv_rf
218	-2.281488e-08	0	0	0	-9.746473e-10	0	0	0
219	6.576029e-08	0	0	0	-1.534914e-08	0	0	0
220	7.521768e-08	0	0	0	2.413701e-08	0	0	0
221	-4.037785e-08	0	0	0	-9.266763e-10	0	0	0
222	-2.322569e-07	0	0	0	9.587527e-09	0	0	0

Nhận xét: Quan sát thấy các kết quả dự đoán của 3 mô hình Logistic Regression, Decision Tree và Random Forest đều cho ra kết quả dạng **binary outcome** (nhị phân chỉ có 2 giá trị 0 và 1), riêng mô hình Linear Regression lại cho ra kết quả là các số thực (để ý thì thấy các số này gần với giá trị 0 (VD: -4.54433275e-08 ~ 0) hoặc 1 (VD: 1.00000007e+00 ~ 1). Điều này cũng dễ hiểu và thường xuyên xảy ra, vì mô hình Linear Regression là mô hình đơn giản và luôn có hàm mất mát (sai số) đi kèm, cho nên các kết quả dự đoán cũng có sai số khá nhỏ, nhưng để chắc chắn việc đánh giá mô hình không bị lỗi, ta sẽ tiền xử lý cho các giá trị dự đoán của mô hình Linear Regression về dạng **binary outcome**

Tiền xử lý

Chuyển các giá trị của cột `tr_cv_li` (mô hình Linear Regression dùng Count Vectorizer) và `tr_tv_li` (mô hình Linear Regression dùng Tfidf Vectorizer) về 2 giá trị 0 và 1

In [13]:
```
prediction['tr_cv_li'] = prediction['tr_cv_li'].round().astype(int)
prediction['tr_tv_li'] = prediction['tr_tv_li'].round().astype(int)
pd.DataFrame(prediction).tail(5)
```

Out[13]:

	tr_cv_li	tr_cv_lo	tr_cv_dt	tr_cv_rf	tr_tv_li	tr_tv_lo	tr_tv_dt	tr_tv_rf
218	0	0	0	0	0	0	0	0
219	0	0	0	0	0	0	0	0
220	0	0	0	0	0	0	0	0
221	0	0	0	0	0	0	0	0
222	0	0	0	0	0	0	0	0

Có vẻ các cột đều có các giá trị dạng **binary outcome**

Đánh giá mô hình bằng Confusion Matrix

Do nhược điểm của **Accuracy Score** là chỉ cho biết độ chính xác khi dự đoán mô hình, nhưng không thể hiện mô hình đang dự đoán sai như thế nào nên ta sẽ sử dụng **Confusion Matrix** để đánh giá độ chính xác của mô hình được phân theo các lớp. Thuộc tính được chọn làm lớp là `labels` nên ta sẽ thể hiện mỗi mô hình bằng một ma trận thể hiện số lượng điểm dữ liệu thuộc vào một lớp và được dự đoán thuộc vào lớp đó.

Định nghĩa hàm `models_evaluation` nhận các tham số

- `y_preds` : các kết quả dự đoán của các mô hình được lấy từ hàm `generate_models` ở trên (dictionary)
- `y_true` : nhãn `label` từ tập dữ liệu gốc (Series)

Công việc của hàm `models_evaluation` :

- Duyệt từng các kết quả dự đoán của các loại mô hình (`y_pred`), với mỗi mô hình cần thực hiện các công việc sau:
  - Tính số lượng các giá trị trong mỗi phân lớp (ở đây ta định nghĩa Positive = real và Negative = fake), các tham số đó là:
    - **TN:** các dự đoán cho ra kết quả `fake` giống với kết quả thực tế `fake` (True Negative)
    - **FP:** các dự đoán cho ra kết quả `real` khác với kết quả thực tế là `fake` (False Positive)
    - **FN:** các dự đoán cho ra kết quả `fake` khác với kết quả thực tế là `real` (False Negative)
    - **TP:** các dự đoán cho ra kết quả `real` giống với kết quả thực tế `real` (True Positive)
  - Lưu các tham số này vào dataframe có kích thước **2 dòng x 2 cột** và đặt tên 2 cột là `real` và `fake`
- Kết quả trả về của hàm là danh sách các dataframe ứng với mỗi loại mô hình và được lưu vào biến `confusion_matrix_df` . Với từng phương thức vector hóa và từng mô hình, tổng cộng ta có 8 **Confusion Matrix**

In [14]:
```
def confusion_matrix_evaluation(y_true: pd.Series, y_preds: dict):
    confusion_matrix_df = []

    for y_pred in y_preds.values():
        # tính số lượng các giá trị trong mỗi phân lớp
        TN, FP, FN, TP = confusion_matrix(y_pred, y_true).ravel()

        # lưu vào dataframe
        df = pd.DataFrame(data=np.array([[TP, FN], [FP, TN]]), index=['real','fake'], columns=['real','fake'])
        confusion_matrix_df.append(df)

    return confusion_matrix_df
```

In [15]:
```
y_true_tmp = news_df['label']
y_preds_tmp = prediction.copy()
confusion_matrix_df = confusion_matrix_evaluation(y_true_tmp, y_preds_tmp)
```

Cuối cùng, ta sẽ thể hiện 8 **Confusion Matrix** bằng đồ thị heatmap ứng với mỗi loại mô hình và mỗi loại vectorizer

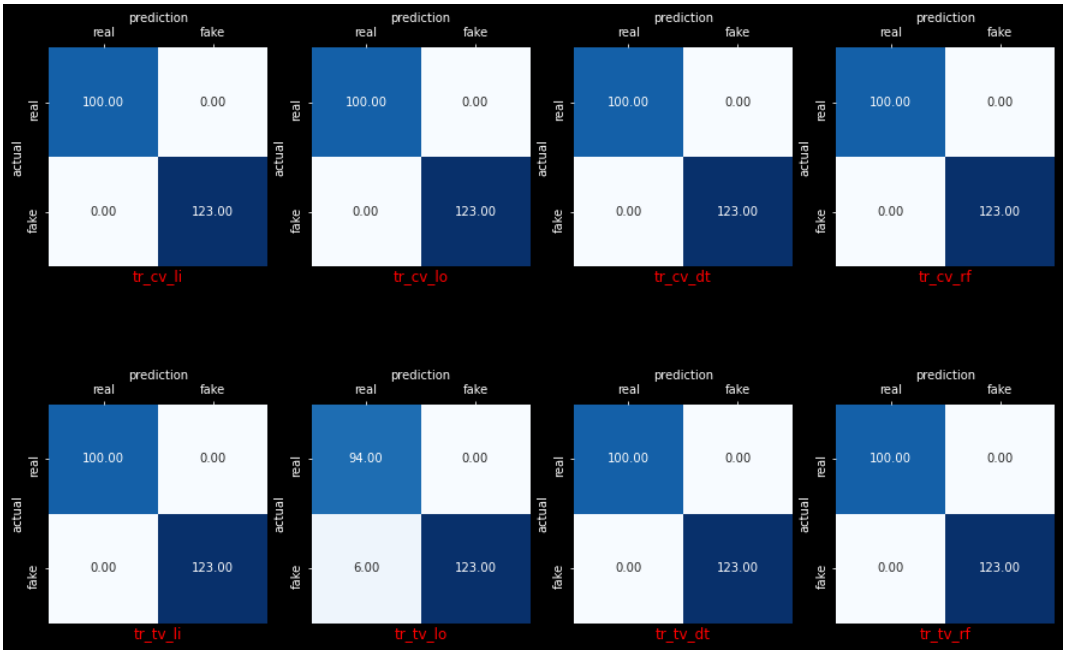
In [16]:
```
prep = [p for p in preprocessors]
vect = [v for v in vectorizers]
clas = [c for c in classifiers]

fig, axarr = plt.subplots(2,4, figsize=(15,10), facecolor='black')

for i in range(1):
    for j in range(2):
        for k in range(4):
            plt.sca(axarr[j, k])
            ax = sns.heatmap(confusion_matrix_df[4*j+k], square=True, annot=True, cbar=False, fmt='.2f', cmap='Blues')
            ax.xaxis.tick_top()
            ax.xaxis.set_label_position('top')

            # set title + labels
            plt.title(prep[i] + '_' + vect[j] + '_' + clas[k], fontdict={'color': 'r'}, y=-0.1)
            plt.xlabel('prediction')
            plt.ylabel('actual')

            # set colors
            ax.xaxis.label.set_color('white')
            ax.yaxis.label.set_color('white')
            ax.tick_params(axis='x', colors='white')
            ax.tick_params(axis='y', colors='white');
```



Kết luận: Xét trường hợp sử dụng tập test là tập huấn luyện, đa số các mô hình ứng với các loại vectorizer đều cho ra kết quả dự đoán chính xác, ngoại trừ mô hình LogisticRegression sử dụng Tfidf Vectorizer có dự đoán sai đối với các kết quả `fake` nhưng thực tế là `real` .

IV. Deploy mô hình

- Phần này sử dụng thư viện Streamlit và làm trong file `app.py`