1. Analyze and Preprocess data - Check if the dataset has missing values or has any other problem.
2. Feature Engineering
3. Divide the dataset into 2 training and test sets
4. Use logistic model Regression. Try to apply different *solver* and *penalty* to find the best one.
5. Perform model on training set and test set
6. Measure performance of the model.
7. Which metric is your main metric and why? Which solver and penalty have you chosen? (<= 100 words)

How can I measure your point:

1. Your function is callable and runs correctly
2. The performance of your model (in full pipeline) is acceptable. The final error based on my train and test set is low enough.
3. The data preprocessing is correct or make sense
4. The Feature engineering is correct or make sense
5. Any other additional process will be considered a small plus point.

Submission Link: https://forms.gle/M2CxqVGrKLTzqR7g9 (Submit your .ipynb file)

- Age: This is the attribute that describes the age of the patient. There is data type $int64$, the highest value is 29, and the lowest is 77.
- Sex: This is the attribute indicating the gender of the patient, where 0 indicates male patient, 1 female patient.
- ChestPainType: This is the attribute that indicates the patient's chest pain level. With levels 0, 1, 2, and 3.
- RestingBP: This is the attribute that indicates the patient's blood pressure with data type $int64$, the value is in the range [94, 200]
- Cholesterol: This attribute indicates the patient's cholesterol level as measured in the hospital. Has the data type $int64$, where the value is in [126, 564]
- FastingBS: This is an attribute that describes the patient's fasting blood sugar. In which, if the patient has more than 120mg/dl sugar = 1, otherwise = 0.
- RestingECG: This property displays the results of the ECG from 0 to 2 (0, 1, 2). Where each value indicates the severity of the pain.
- thalach: Patient's highest heart rate
- ExerciseAngina: Whether or not you have angina during exercise. Yes denotes 1, no denotes 0.
- Oldpeak: Attribute expressing the stress level of the patient. Has a value of type $float64$, the value is in [0, 6.2]

- ST_Slope: Patient's condition during exercise. Includes [Upsloping, Flat, Down sloping] states that are sequentially digitized to [0, 1, 2].
- ca: number of major vessels (0-3) colored by flourosopy - given
- thal: 0 = normal; 1 = fixed defect; 2 = reversable defect
- HeartDisease: Results of the patient's condition. 1 is for signs of heart disease, 0 is for no signs of heart disease.

Load Dataset

```
# mount data from google drive to colab
from google.colab import drive
drive.mount('/content/drive')

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
def read_dataset(path):  # dùng 'path' làm tên tham số đầu vào
    df = pd.read_csv(path)
    display(df.head())
    display(df.describe())
    return df
```

```
import pandas as pd

PATH = "/content/drive/MyDrive/Dataset/Term_Deposit1.csv"
df = read_dataset(PATH)

#ToDo: Show histogram of dataframe
```

	age	job	marital	education	default	balance	housing	loan	contact	da
0	47.0	management	married	tertiary	no	2351.0	no	no	cellular	[illegible]
1	26.0	admin.	single	secondary	no	255.0	no	no	cellular	1
2	26.0	admin.	single	secondary	no	256.0	no	no	cellular	1
3	26.0	admin.	single	secondary	no	257.0	no	no	cellular	1
4	26.0	admin.	single	secondary	no	258.0	no	no	cellular	1

	age	balance	day	duration	campaign	pd
count	66024.000000	66024.000000	66024.000000	66024.000000	66024.000000	66024.000
mean	41.293893	1528.499137	15.697186	408.975221	2.504801	56.840
std	12.431010	3201.683875	8.536963	418.539701	2.706804	105.404
min	-1.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000
25%	32.000000	123.000000	8.000000	133.000000	1.000000	-1.000
50%	39.000000	551.000000	16.000000	252.000000	2.000000	-1.000
75%	49.000000	1676.000000	22.000000	525.000000	3.000000	92.000
max	999.000000	102127.000000	31.000000	4918.000000	63.000000	871.000

```
from google.colab import drive
drive.mount('/content/drive')
```

Data Analysis

```
#các thông tin của từng feature
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66024 entries, 0 to 66023
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        66024 non-null  float64
 1   job        66024 non-null  object
 2   marital    66024 non-null  object
 3   education  66024 non-null  object
 4   default    66024 non-null  object
 5   balance    66024 non-null  float64
 6   housing    66024 non-null  object
 7   loan       66024 non-null  object
 8   contact    66024 non-null  object
 9   day        66024 non-null  int64
 10  month      66024 non-null  object
 11  duration   66024 non-null  int64
 12  campaign   66024 non-null  int64
```

```
 13  pdays      66024 non-null  int64
 14  previous   66024 non-null  int64
 15  poutcome   66024 non-null  object
 16  y          66024 non-null  object
dtypes: float64(2), int64(5), object(10)
memory usage: 8.6+ MB
```

```
print("Quantitative columns \n")
print(df.describe())
print()
print("Qualitative and Quantitative columns \n")
print(df.describe(include=[object, float]))
```

```
Quantitative columns

                age        balance           day      duration      campaign  \
count  66024.000000   66024.000000  66024.000000  66024.000000  66024.000000
mean      41.293893    1528.499137     15.697186    408.975221      2.504801
std       12.431010    3201.683875      8.536963    418.539701      2.706804
min       -1.000000   -8019.000000      1.000000      0.000000      1.000000
25%       32.000000     123.000000      8.000000    133.000000      1.000000
50%       39.000000     551.000000     16.000000    252.000000      2.000000
75%       49.000000    1676.000000     22.000000    525.000000      3.000000
max      999.000000  102127.000000     31.000000   4918.000000     63.000000

              pdays      previous
count  66024.000000  66024.000000
mean      56.840679      0.960272
std      105.404425      2.439411
min       -1.000000      0.000000
25%       -1.000000      0.000000
50%       -1.000000      0.000000
75%       92.000000      1.000000
max      871.000000    275.000000

Qualitative and Quantitative columns

                 age         job  marital  education default        balance  \
count   66024.000000       66024    66024      66024   66024   66024.000000
unique           NaN          12        3          4       2            NaN
top              NaN  management  married  secondary      no            NaN
freq             NaN       14663    37052      32395   65134            NaN
mean       41.293893         NaN      NaN        NaN     NaN    1528.499137
std        12.431010         NaN      NaN        NaN     NaN    3201.683875
min        -1.000000         NaN      NaN        NaN     NaN   -8019.000000
25%        32.000000         NaN      NaN        NaN     NaN     123.000000
50%        39.000000         NaN      NaN        NaN     NaN     551.000000
75%        49.000000         NaN      NaN        NaN     NaN    1676.000000
max       999.000000         NaN      NaN        NaN     NaN  102127.000000

       housing   loan   contact  month poutcome      y
count    66024  66024     66024  66024    66024  66024
unique       2      2         3     12        4      2
top         no     no  cellular    may  unknown     no
freq     35128  57452     47373  16757    46068  39911
mean       NaN    NaN       NaN    NaN      NaN    NaN
std        NaN    NaN       NaN    NaN      NaN    NaN
min        NaN    NaN       NaN    NaN      NaN    NaN
25%        NaN    NaN       NaN    NaN      NaN    NaN
```

```
50%         NaN     NaN         NaN     NaN        NaN     NaN
75%         NaN     NaN         NaN     NaN        NaN     NaN
max         NaN     NaN         NaN     NaN        NaN     NaN
```

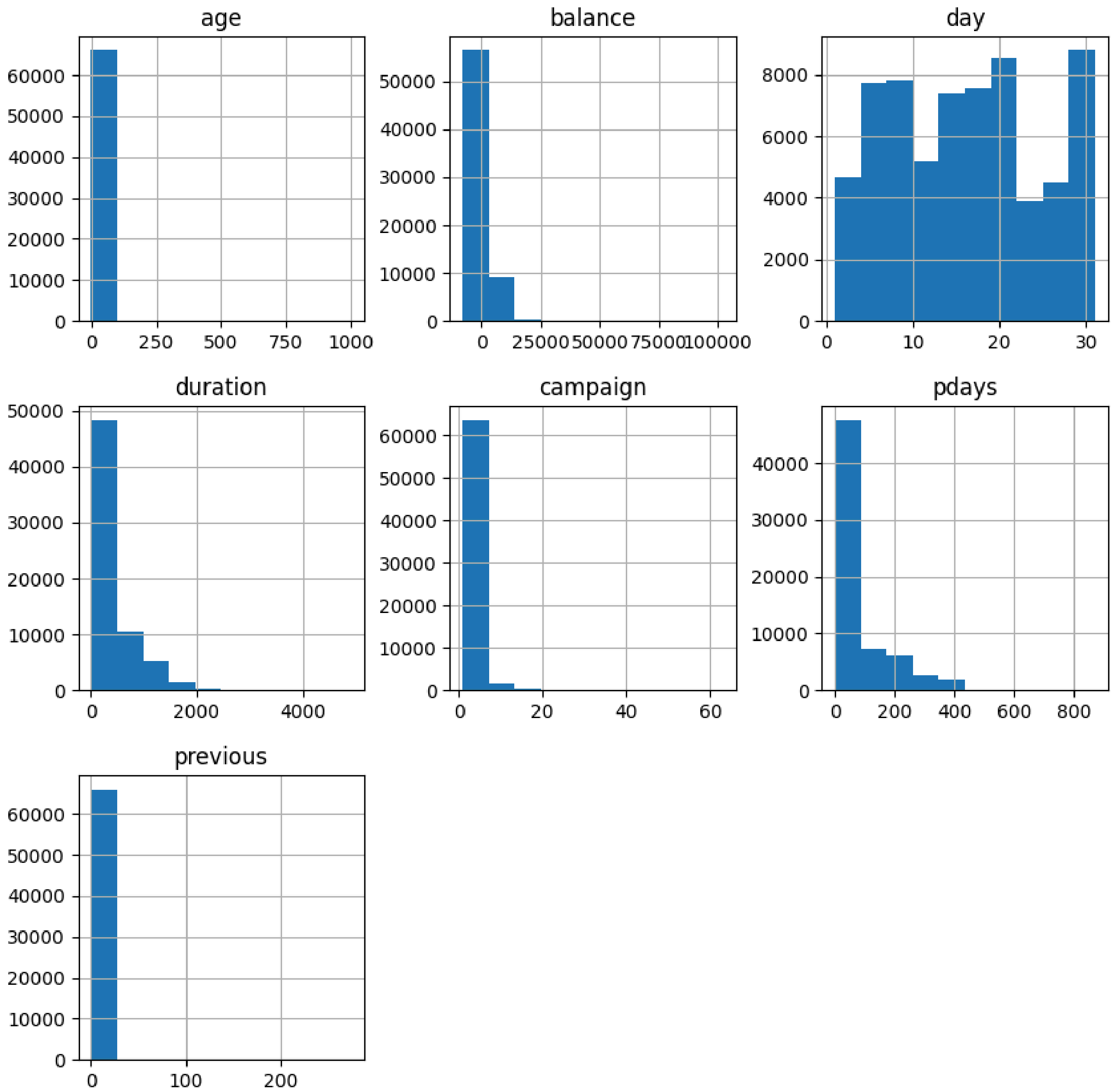
Exploratory Data Analysis

```
print("Continous Columns")
continous_columns = df.describe().columns
print(continous_columns)

print("Categorical Columns")
categorical_columns = df.describe(include=[object]).columns
print(categorical_columns)
```

```
Continous Columns
Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous'], dtype='
Categorical Columns
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'month', 'poutcome', 'y'],
      dtype='object')
```

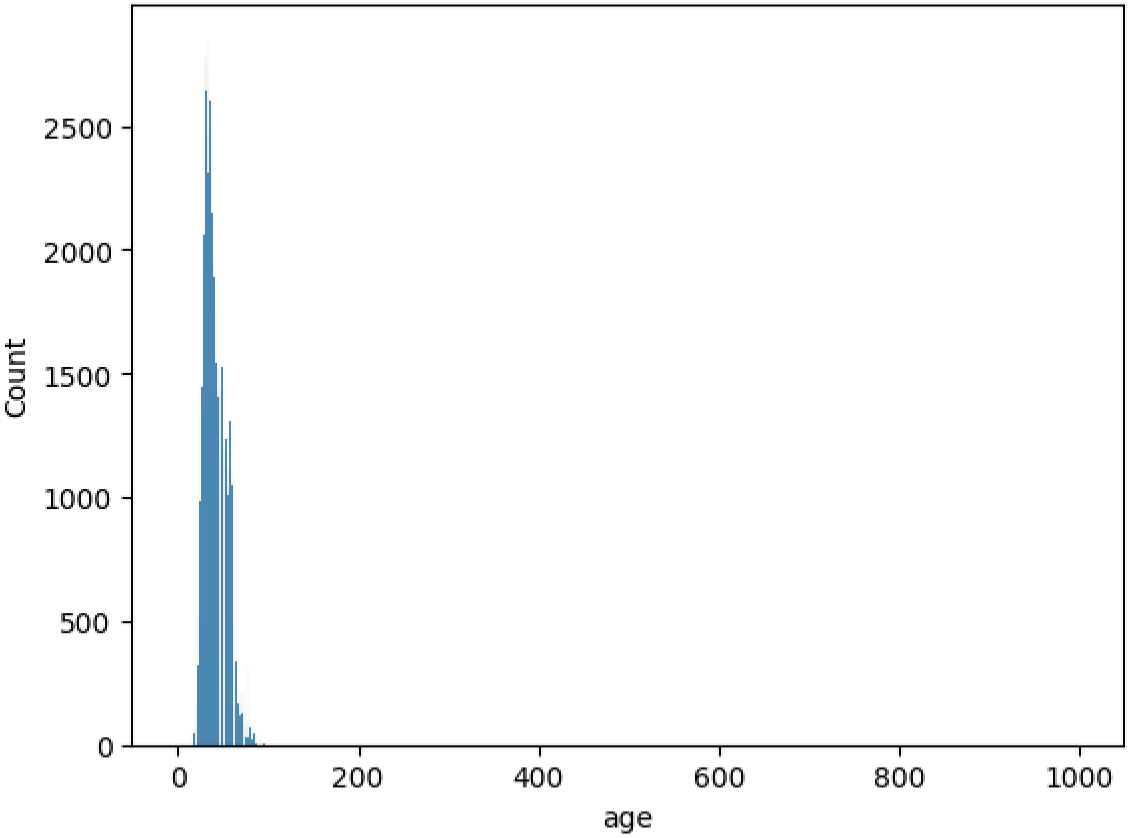
```
df.hist(column=continous_columns, figsize=(10, 10))
# column chính là các feature mà ta muốn vẽ, figsize là kích thước của hình vẽ với giá tr
```

```
array([[<Axes: title={'center': 'age'}>,
        <Axes: title={'center': 'balance'}>,
        <Axes: title={'center': 'day'}>],
       [<Axes: title={'center': 'duration'}>,
        <Axes: title={'center': 'campaign'}>,
        <Axes: title={'center': 'pdays'}>],
       [<Axes: title={'center': 'previous'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



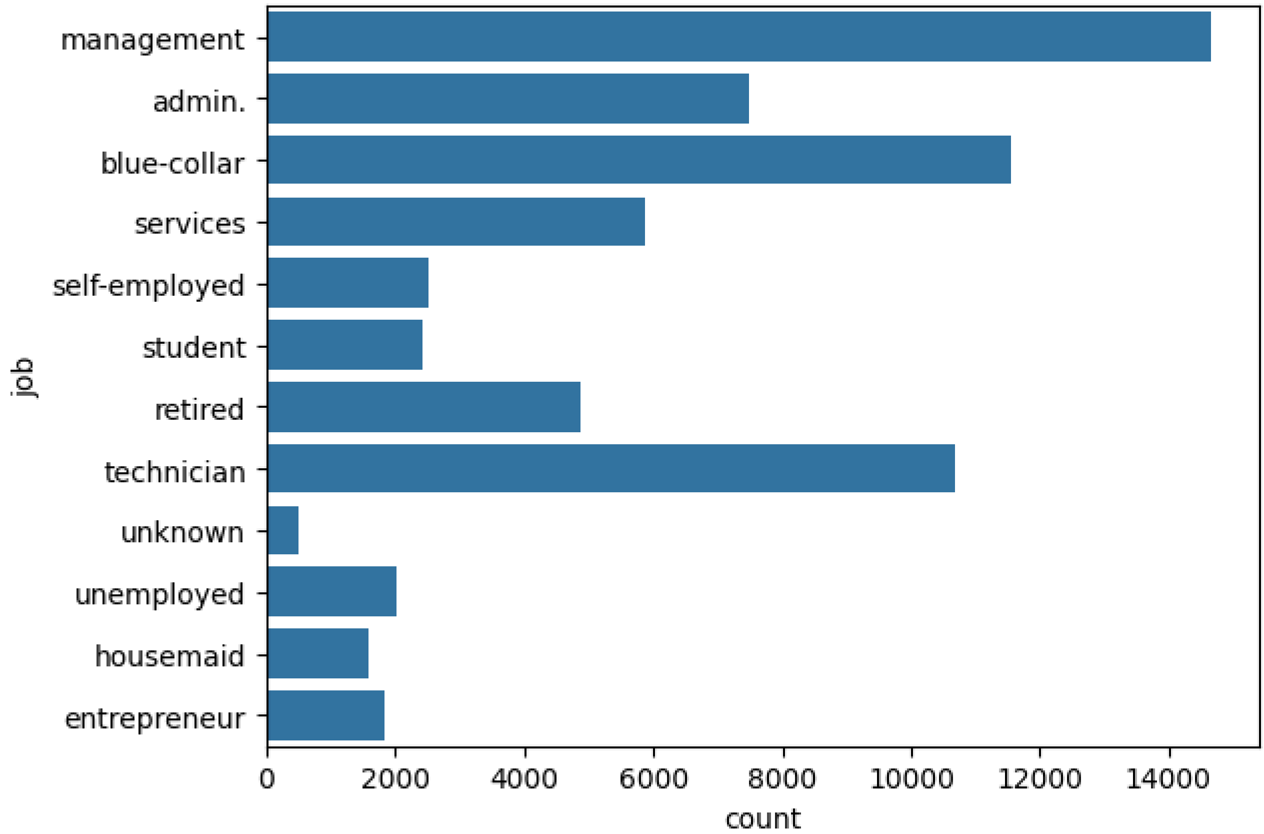
```
#Để vẽ hist cho từng feature mà ta muốn, ta có thể làm như sau:
sns.histplot(x="age", data=df) #Nếu ta dùng x thì sẽ vẽ được hình trên trục hoành, còn dù
#x hoặc y là feature mà mình muốn vẽ, data chính là dataframe mà mình muốn đứa vào
```

<Axes: xlabel='age', ylabel='Count'>



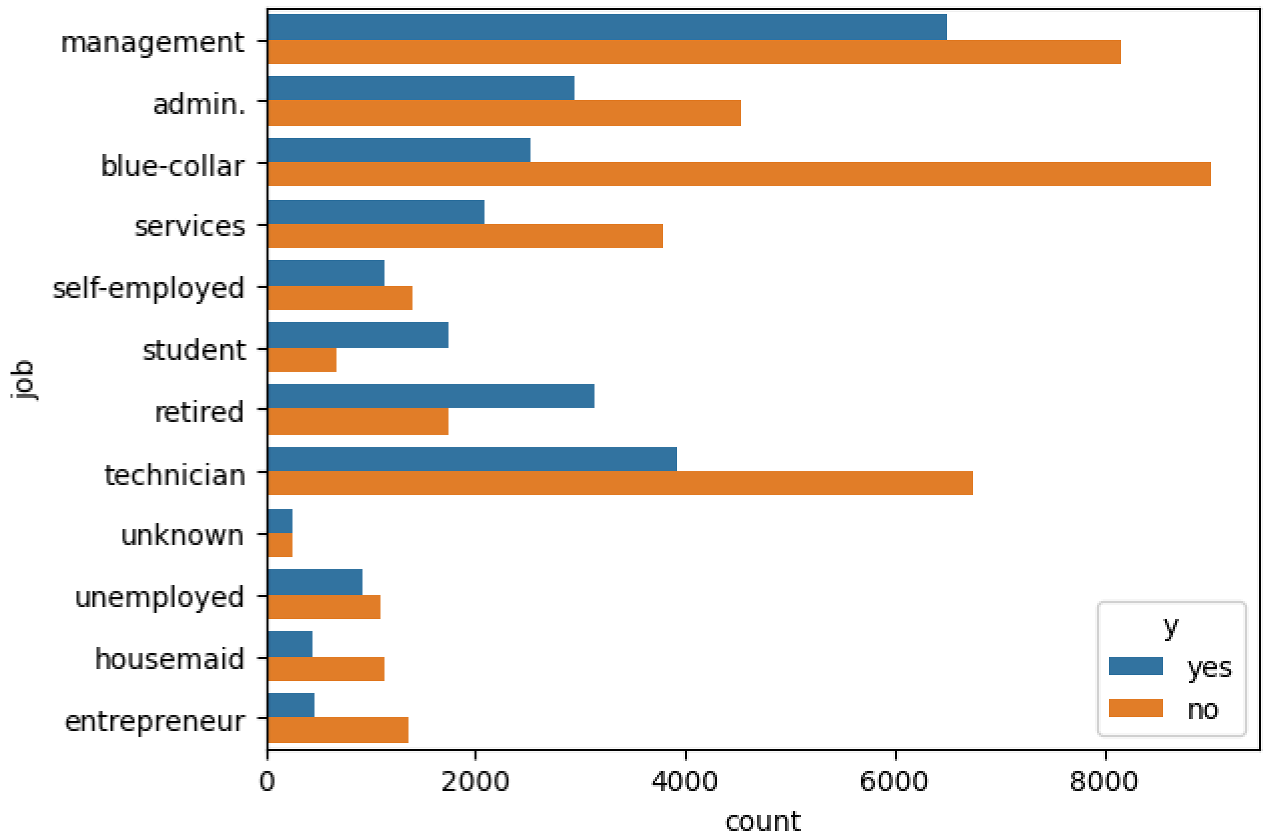

```
#Để vẽ countplot cho các biến categorical ta làm như sau
sns.countplot(y="job", data=df)
```

```
<Axes: xlabel='count', ylabel='job'>
```



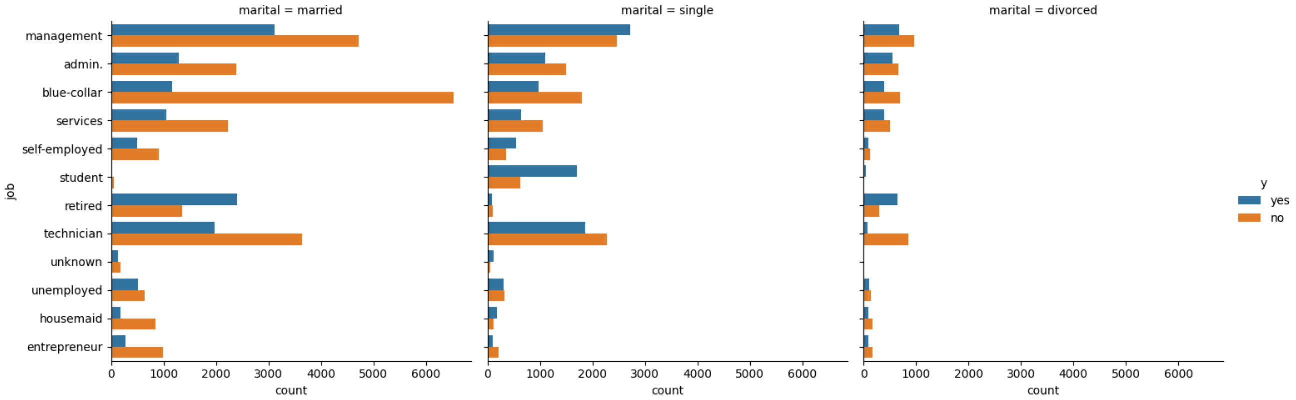

```
#Để vẽ countplot có kèm theo một feature nữa có thể dùng thêm parameter hue
sns.countplot(y="job", data=df, hue="y")
#biến hue để chia các samples theo các giá trị có trong y
```

```
<Axes: xlabel='count', ylabel='job'>
```



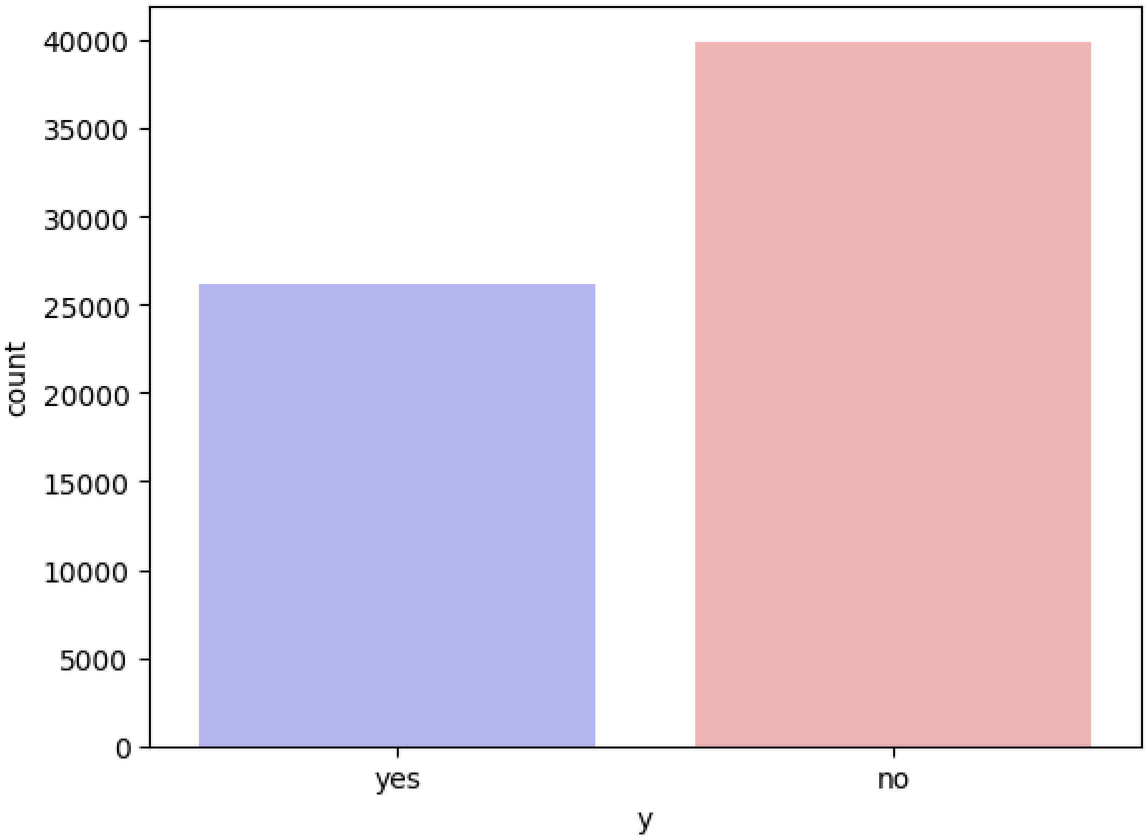
```
#Ngoài ra ta có thể dùng thêm một số features khác bằng cách sử dụng catplot
sns.catplot(y="job", data=df, hue="y", col="marital", kind="count")
#Ở đây ta sẽ vẽ countplot cho feature job, dùng hue theo các giá trị y, chia thêm các giá
```

```
<seaborn.axisgrid.FacetGrid at 0x79013702d0d0>
```



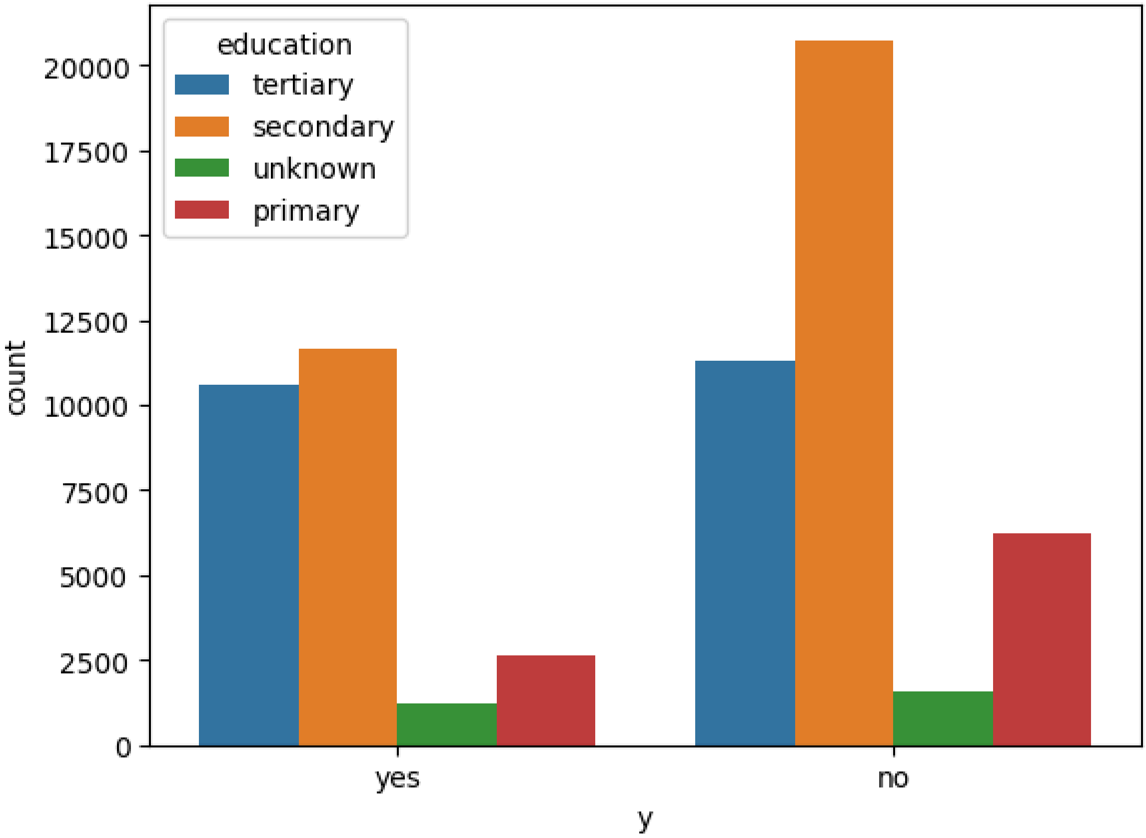

```
sns.countplot(x="y", data=df, palette="bwr") # Thống kê cột 'y'
plt.show()
```

```
<ipython-input-32-653f7793fe78>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.

  sns.countplot(x="y", data=df, palette="bwr") # Thống kê cột 'y'
```

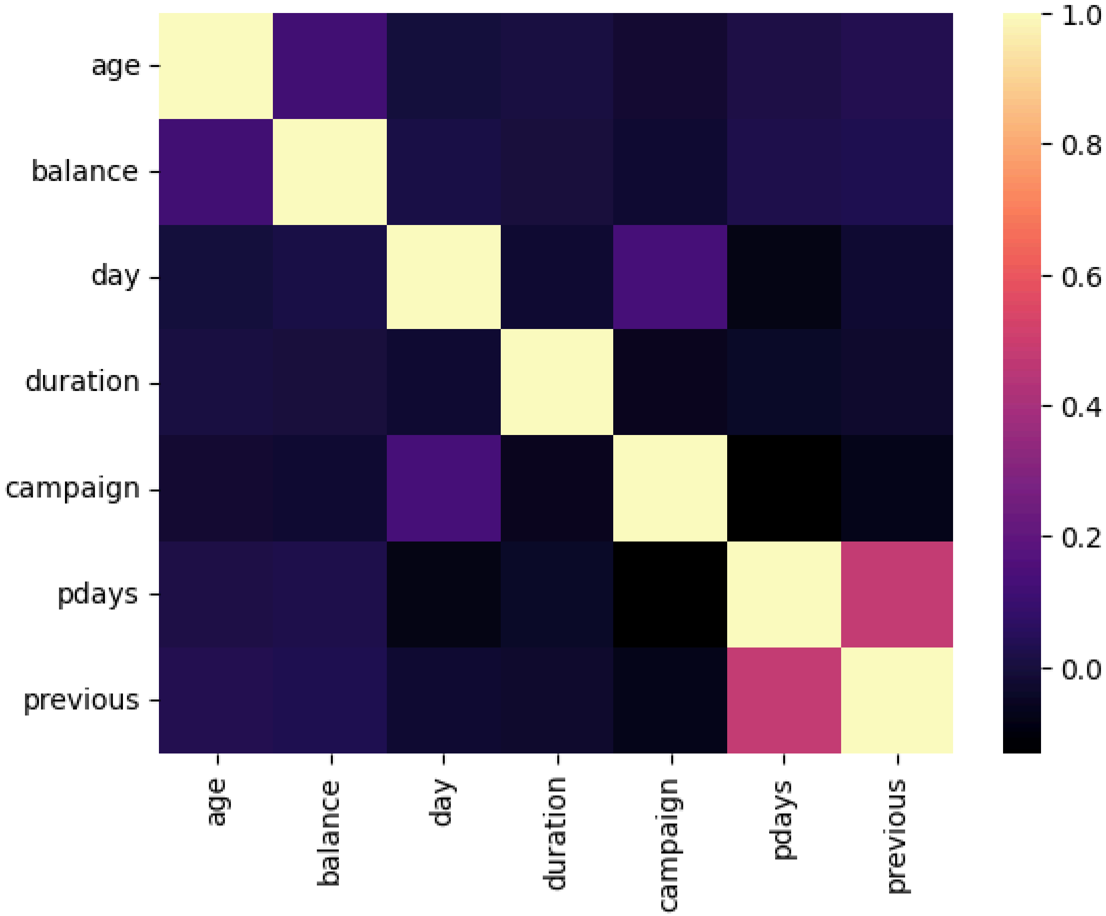


```
numerical_df = df.select_dtypes(include=['number'])
sns.countplot(data=df, x='y', hue='education')
```

```
<Axes: xlabel='y', ylabel='count'>
```




```
# Vẽ heatmap ma trận tương quan giữa các biến số (chỉ các cột dạng số)
sns.heatmap(df.select_dtypes(include='number').corr(), cmap='magma', annot=False)
plt.show()
```



Model Training

```
def apply_feature_engineering(df):
    """
    Apply all feature engineering to transform your data into number
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    # Chỉ chuẩn hóa các cột dạng số
    numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

    # Áp dụng Min-Max Scaling
    scaler = MinMaxScaler()
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

    # **Thêm đoạn mã này vào đây**
    from sklearn.preprocessing import PolynomialFeatures

    # Tạo biến tương tác
    df['age_balance'] = df['age'] * df['balance']

    # Tạo biến đa thức
    poly = PolynomialFeatures(degree=2)
    poly_features = poly.fit_transform(df[['age', 'balance']])
    poly_df = pd.DataFrame(poly_features, columns=['poly_feature_' + str(i) for i in rang
    df = pd.concat([df, poly_df], axis=1)
```

```
    return df


processed_df = apply_feature_engineering(df.copy())
processed_df.head()
```

	age	balance	day	duration	campaign	pdays	previous	y	job_admin.	job_blue-collar
0	47.0	2351.0	2	163	2	84	1	yes	False	False
1	26.0	255.0	14	209	2	106	2	yes	True	False
2	26.0	256.0	14	210	2	106	2	yes	True	False
3	26.0	257.0	14	211	2	106	2	yes	True	False
4	26.0	258.0	14	212	2	106	2	yes	True	False

5 rows × 52 columns

```
def prepare_X_y(df):
    """
    Feature engineering and create X and y
    :param df: pandas dataframe
    :return: (X, y) output feature matrix (dataframe), target (series)
    """

    X = df.drop('y', axis=1, inplace=False).values

    y = df['y']
    y = np.array([0 if i=="no" else 1 for i in y ])
    y = y.reshape((-1, 1))
    return X, y

X, y = prepare_X_y(processed_df)


def build_model(X, y):
    """
    Design your model and train it (including your best params)
    :param X: feature matrix
    :param y: target
    :return: a model
    """
    # Create a pipeline with StandardScaler and LogisticRegression
    model = make_pipeline(StandardScaler(), LogisticRegression())

    # **Thêm đoạn mã này vào đây**
    from sklearn.model_selection import GridSearchCV

    param_grid = {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100]}
    grid = GridSearchCV(model, param_grid, cv=5)
```

```
    grid.fit(X, y)
    model = grid.best_estimator_


    # Fit the model
    model.fit(X, y)

    return model


# Trong hàm build_model
from sklearn.ensemble import RandomForestClassifier

model = make_pipeline(StandardScaler(), RandomForestClassifier())
model.fit(X, y)
```

```
Pipeline
  StandardScaler
  RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101


log_model = build_model(X_train, y_train)
display(log_model)

# Get score
log_model.score(X_test, y_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConvers
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression
LogisticRegression()
0.9044325525040388
```

```
from sklearn.preprocessing import StandardScaler #Gọi thư viện để scale data về phân phối

scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train) #fit_transform có tác dụng vừa fit data, v
X_test_scaled = scaler.transform(X_test) #transform data từ hàm scaler đã train từ X_trai

# Print dataframe of scaled data
display(pd.DataFrame(X_train_scaled))
```

	0	1	2	3	4	5	6	
0	0.134841	0.168923	0.735593	0.325399	0.192122	-0.550104	-0.436967	-0.35625
1	3.374412	1.006072	-0.787916	2.404948	-0.565094	0.330261	0.928228	-0.35625
2	1.083008	-0.177558	-1.256688	-0.703557	0.570730	-0.550104	-0.436967	-0.35625
3	0.529911	-0.264256	0.266821	-0.177059	-0.565094	4.769952	0.473163	2.80698
4	-0.260228	-0.466241	1.321558	-0.833379	0.570730	-0.550104	-0.436967	-0.35625
...	...	...	...	...	...	...	...	.
46211	-0.892340	-0.260217	1.438751	0.310974	-0.186486	1.172761	3.203553	-0.35625
46212	-0.892340	-0.466862	1.673137	-0.876653	-0.565094	-0.550104	-0.436967	-0.35625
46213	-1.129381	-0.469038	1.555944	-0.725194	0.192122	-0.550104	-0.436967	-0.35625
46214	-1.366423	-0.401606	-0.787916	-0.746831	-0.565094	0.330261	0.018098	-0.35625
46215	1.162022	-0.400984	0.149628	-0.701153	0.192122	0.699447	1.383293	-0.35625

46216 rows × 51 columns

```
log_model = build_model(X_train_scaled, y_train)
display(log_model)

# Get score
log_model.score(X_test_scaled, y_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataConvers
  y = column_or_1d(y, warn=True)
```

▾ LogisticRegression ⓘ ?

LogisticRegression()

0.9330068659127625

```
# Thiết lập bảng kết quả dự đoán
y_pred = log_model.predict(X_test_scaled)
y_pred = pd.DataFrame({'target': y_pred})
y_pred
```

	target
0	1
1	0
2	0
3	0
4	0
...	...
19803	0
19804	0
19805	0
19806	0
19807	0

19808 rows × 1 columns

```
#9 Sử dụng một số metrics cho imbalanced data
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, conf

print("Precision: ", precision_score(y_test , y_pred))
print("Recall: ", recall_score(y_test , y_pred))
print("F1: ", f1_score(y_test , y_pred))
print("Confusion matrix: \n", confusion_matrix(y_test , y_pred))
print("Classification report: \n", classification_report(y_test , y_pred))
```

```
Precision:  0.9187086736678335
Recall:  0.9100950423837657
F1:  0.91438157300471
Confusion matrix: 
 [[11395   627]
 [  700  7086]]
Classification report: 
               precision    recall  f1-score   support

           0       0.94      0.95      0.94     12022
           1       0.92      0.91      0.91      7786

    accuracy                           0.93     19808
   macro avg       0.93      0.93      0.93     19808
weighted avg       0.93      0.93      0.93     19808
```

```
def calculate_performance(y_true, y_pred):
    """

    :param y_true: ground truth values
    :param y_pred: predictions
    :return:
```

```
    """

    print("Accuracy: ", accuracy_score(y_test, y_pred))
    print("Precision: ", precision_score(y_test , y_pred))
    print("Recall: ", recall_score(y_test , y_pred ))
    print("F1: ", f1_score(y_test , y_pred))
    print("Confusion matrix: \n", confusion_matrix(y_test , y_pred ))
    print("classification_report: ", classification_report(y_true, y_pred))
    return f1_score(y_test , y_pred )
```

Preprocessing

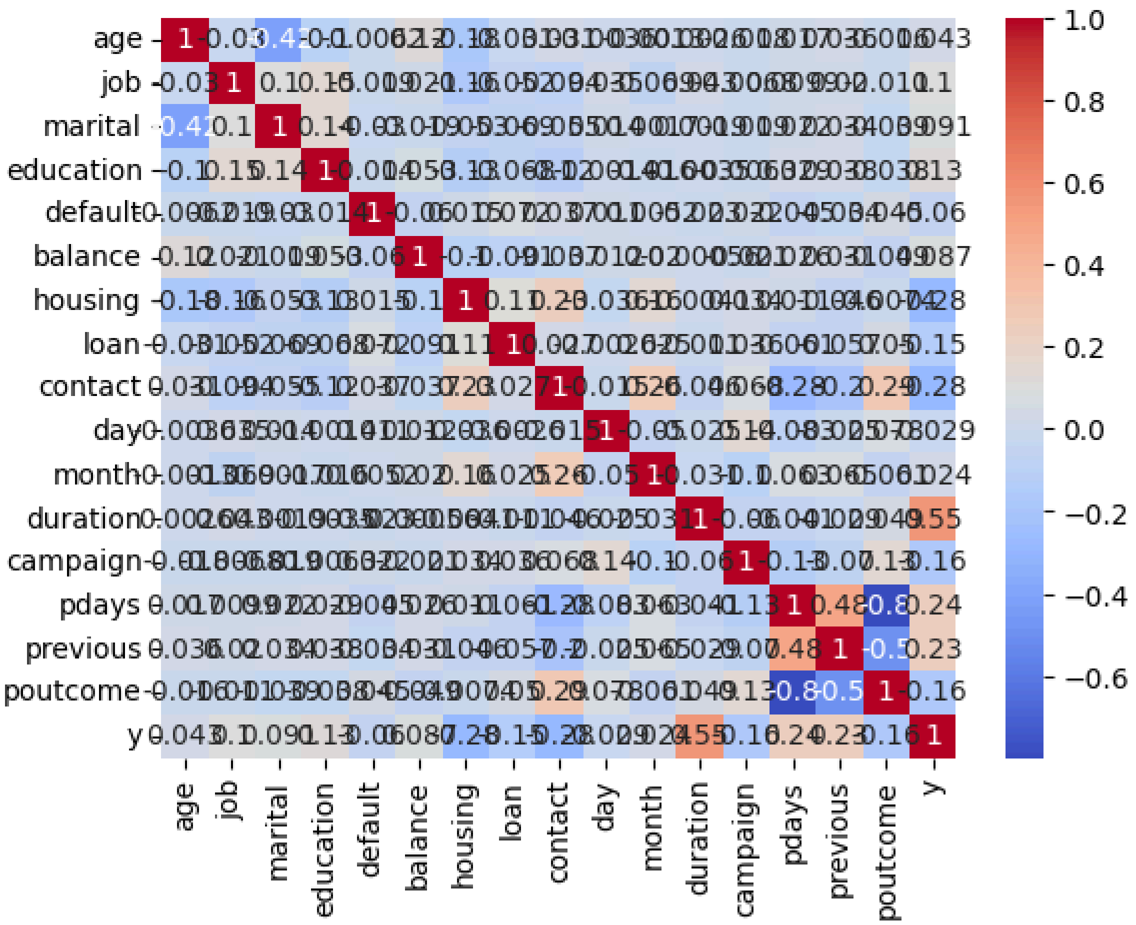
```
def preprocessing_data(df):
    """
    Preprocess your data (eg. Drop null datapoints or fill missing data)
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    # Xoá các dòng có giá trị bị thiếu (nếu bạn muốn giữ lại, có thể dùng fillna)
    df = df.dropna()

    # Reset index sau khi drop
    df = df.reset_index(drop=True)

    # **Thêm đoạn mã này vào đây**
    for col in df.select_dtypes(include=['number']).columns:
        df[col] = df[col].fillna(df[col].mean())  # Hoặc df[col].median()

    # Mã hoá các biến phân loại dạng object (label encoding hoặc one-hot nếu cần)
    for col in df.select_dtypes(include='object').columns:
        df[col] = df[col].astype('category').cat.codes

    return df


df = preprocessing_data(df.copy())
```

Feature Engineering

```
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler

def apply_feature_engineering(df):
    """
    Apply all feature engineering to transform your data into number
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    # Chỉ chuẩn hóa các cột dạng số
    numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
```

```python
    # Áp dụng Min-Max Scaling
    scaler = MinMaxScaler()
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

    return df

# Giả sử df đã được tiền xử lý (preprocessed) trước đó
df = apply_feature_engineering(df)

# Hiển thị heatmap sau khi dữ liệu được chuẩn hóa
sns.heatmap(df.corr(), cmap='coolwarm', annot=True)
```

```
<Axes: >
```




```python
from sklearn.model_selection import train_test_split

def prepare_X_y(df):
    """
    Feature engineering and create X and y
    :param df: pandas dataframe
    :return: (X, y) output feature matrix (dataframe), target (series)
    """
    # Giả sử cột mục tiêu (label) là 'y' (bạn có thể thay bằng tên cột thực tế)
    X = df.drop('y', axis=1)  # X là toàn bộ dataframe trừ cột 'y'
    y = df['y']               # y là cột mục tiêu

    return X, y
```

```
# Gọi hàm để tạo ra X và y
X, y = prepare_X_y(df)
```

Apply machine learning model

Train-test split

```
from sklearn.model_selection import train_test_split

RANDOM_STATE = 0
TRAIN_SIZE = 0.8

# Chia dữ liệu thành tập train và test
trainX, testX, trainY, testY = train_test_split(X, y, train_size=TRAIN_SIZE, random_state
```