```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import math
import random
import seaborn as sns
import pandas_profiling as pp
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[2]:


data = pd.read_csv('p.csv')


# In[3]:


data.head(3)


# In[4]:


data = data[(data['chol'] <= 420) & (data['oldpeak'] >=0) & (data['oldpeak'] <=4)].reset_index(drop=True)
data = data.dropna().reset_index(drop=True)
data.info()


# In[5]:


data.describe()


# In[6]:


data.info()


# In[7]:


def str_features_to_numeric(data):
    # Transforms all string features of the df to numeric features

    # Determination categorical features
    categorical_columns = []
    numerics = ['int8', 'int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    features = data.columns.values.tolist()
    for col in features:
        if data[col].dtype in numerics: continue
        categorical_columns.append(col)

    # Encoding categorical features
    for col in categorical_columns:
        if col in data.columns:
            le = LabelEncoder()
            le.fit(list(data[col].astype(str).values))
            data[col] = le.transform(list(data[col].astype(str).values))

    return data


# In[8]:


data = str_features_to_numeric(data)
data


# In[9]:


data.target.value_counts()


# In[10]:


data = data[data['target'].isin([0, 1])]
data


# In[11]:


def fe_creation(df):
    df['age2'] = df['age']//10
    df['trestbps2'] = df['trestbps']//10
    df['chol2'] = df['chol']//60
    df['thalch2'] = df['thalch']//40
    df['oldpeak2'] = df['oldpeak']//0.4
    for i in ['sex', 'age2', 'fbs', 'restecg', 'exang']:
        for j in ['cp','trestbps2', 'chol2', 'thalch2', 'oldpeak2', 'slope']:
            df[i + "_" + j] = df[i].astype('str') + "_" + df[j].astype('str')
    return df

data = fe_creation(data)


# In[12]:


pd.set_option('max_columns', len(data.columns)+1)
data = str_features_to_numeric(data)
data.head(3)


# In[13]:
```

```
data.shape


# In[14]:


dataset = data.copy()  # original data
target_name = 'target'
target = data.pop(target_name)


# In[15]:


scaler = StandardScaler()
#scaler = RobustScaler()
data = pd.DataFrame(scaler.fit_transform(data), columns = data.columns)


# In[16]:


train, valid, train_target, valid_target = train_test_split(data, target, test_size=test_train_split_part, random_state=random_state)


# In[17]:


train


# In[18]:


valid


# In[19]:


num_models = 6
acc_train = []
acc_valid = []
acc_all = np.empty((len(metrics_now)*2, 0)).tolist()
acc_all


# In[20]:


acc_all_pred = np.empty((len(metrics_now), 0)).tolist()
acc_all_pred


# In[21]:


cv_train = ShuffleSplit(n_splits=cv_n_split, test_size=test_train_split_part, random_state=random_state)


# In[22]:


def acc_d(y_meas, y_pred):
    # Relative error between predicted y_pred and measured y_meas values
    return mean_absolute_error(y_meas, y_pred)*len(y_meas)/sum(abs(y_meas))

def acc_rmse(y_meas, y_pred):
    # RMSE between predicted y_pred and measured y_meas values
    return (mean_squared_error(y_meas, y_pred))**0.5


# In[23]:


# Decision Tree Classifier
decision_tree = DecisionTreeClassifier()
param_grid = {'min_samples_leaf': [i for i in range(2,12)]}
decision_tree_CV = GridSearchCV(decision_tree, param_grid=param_grid, cv=cv_train, verbose=False)
decision_tree_CV.fit(train, train_target)
print(decision_tree_CV.best_params_)
acc_all = acc_metrics_calc(0, acc_all, decision_tree_CV, train, valid, train_target, valid_target)


# In[24]:


plot_learning_curve(decision_tree_CV, "Decision Tree", train, train_target, cv=cv_train)


# In[25]:


get_ipython().run_cell_magic('time', '', '# XGBoost Classifier\nxgb_clf = xgb.XGBClassifier(objective=\'reg:squarederror\') \nparameters = {\'n_estimators\': [30, 40, 50, 60, 75, 100]


# In[26]:


for x in metrics_now:
    # Plot
    xs = metrics_all[x]
    xs_train = metrics_all[x] + '_train'
    xs_test = metrics_all[x] + '_valid'
    plt.figure(figsize=[15,6])
    xx = models['Model']
    plt.tick_params(labelsize=14)
    plt.plot(xx, models[xs_train], label = xs_train)
    plt.plot(xx, models[xs_test], label = xs_test)
    plt.legend()
    plt.title(str(xs) + ' criterion for ' + str(num_models) + ' popular models for train and valid datasets')
    plt.xlabel('Models')
    plt.ylabel(xs + ', %')
    plt.xticks(xx, rotation='vertical')
    plt.show()


# In[27]:
```

```python
models_best = models[(models.acc_diff < 0.1) & (models.acc_valid > 0.7)]
if len(models_best)>0:
    print('The best models:')
    display(models_best[['Model', 'acc_train', 'acc_valid']].sort_values(by=['acc_valid'], ascending=False))
    # Selection the best models from the best
    models_best_best = models_best[(models_best.acc_valid > 0.9)]
    if len(models_best_best)>0:
        print('Optimal model:')
        display(models_best_best[['Model', 'acc_train', 'acc_valid']].sort_values(by=['acc_valid'], ascending=False))
    else: print('But no model provides good accuracy at least above 0.9')
else:
    print('There are no good models - either they have not learned enough, or they have overfit!')


# In[ ]:
```