HW5 (20 Points): Required Submissions:

1. Submit colab/jupyter notebooks.
2. There are two Questions with different datasets.
3. **You do not need to do EDA again. You can use the EDA from last HW. We are using the same datasets as in the last HW.**
4. Pdf version of the notebooks (HWs will not be graded if pdf version is not provided).
5. **The notebooks and pdf files should have the output.**
6. **Name files as follows : FirstName_file1_hw6, FirstName_file2_h6, FirstName_file3_h6, FirstName_file4_h6**

Question1 (10 Points) : Classification on the 'credit-g' dataset using SVM.

- **Use RandomSerachCV(OR Halving GridsearchCV, HalvingRandomSerachCV) for this problem**.
- Try poly and rbf kernels in the same pipeline.

Compare KNN and Logistic Regression/SVM.(previous HWs), Basd on your anaysis which algorithm you will recommend.

Download Data:

You can download the dataset using the commands below and see it's description at https://www.openml.org/d/31

Attribute description from https://www.openml.org/d/31

1. Status of existing checking account, in Deutsche Mark.
2. Duration in months
3. Credit history (credits taken, paid back duly, delays, critical accounts)
4. Purpose of the credit (car, television,...)
5. Credit amount
6. Status of savings account/bonds, in Deutsche Mark.
7. Present employment, in number of years.
8. Installment rate in percentage of disposable income
9. Personal status (married, single,...) and sex
10. Other debtors / guarantors
11. Present residence since X years
12. Property (e.g. real estate)
13. Age in years
14. Other installment plans (banks, stores)
15. Housing (rent, own,...)
16. Number of existing credits at this bank
17. Job
18. Number of people being liable to provide maintenance for
19. Telephone (yes,no)
20. Foreign worker (yes,no)

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

```
Mounted at /content/drive
```

```
!pip install feature_engine scikit-learn -qq
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 328.9/328.9 kB 6.1 MB/s eta 0:00:00
```

```
import feature_engine
import sklearn
import sys
```

```
import pandas as pd
import numpy as np
from scipy.io import arff
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from feature_engine.transformation import YeoJohnsonTransformer
from sklearn.preprocessing import MaxAbsScaler
from sklearn.datasets import fetch_openml
from sklearn.pipeline import Pipeline
from feature_engine.encoding import RareLabelEncoder
from feature_engine.encoding import OneHotEncoder
from feature_engine.transformation import LogTransformer
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from numpy.core.function_base import logspace
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.model_selection import HalvingGridSearchCV
```

```
base = Path("/content/drive/MyDrive/Applied_ML/Class_4/Assignment")
```

```
custom_function_folder = base/"Custom_function"
```

```
sys.path.append(str(custom_function_folder))
```

```
sys.path
```

```
['/content',
 '/env/python',
 '/usr/lib/python310.zip',
 '/usr/lib/python3.10',
 '/usr/lib/python3.10/lib-dynload',
 '',
 '/usr/local/lib/python3.10/dist-packages',
 '/usr/lib/python3/dist-packages',
 '/usr/local/lib/python3.10/dist-packages/IPython/extensions',
 '/root/.ipython',
 '/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Custom_function']
```

```
from eda_plots import diagnostic_plots, plot_target_by_category
```

```
from  plot_learning_curve import plot_learning_curve
```

```
X,y = fetch_openml("credit-g", version=1, as_frame=True, return_X_y=True)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `parser` will c
  warn(
```

```
X.head()
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_comm
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	

```
y.head()
```

```
0    good
1     bad
```

```
2    good
3    good
4     bad
Name: class, dtype: category
Categories (2, object): ['bad', 'good']
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   checking_status         1000 non-null   category
 1   duration                1000 non-null   float64
 2   credit_history          1000 non-null   category
 3   purpose                 1000 non-null   category
 4   credit_amount           1000 non-null   float64
 5   savings_status          1000 non-null   category
 6   employment              1000 non-null   category
 7   installment_commitment  1000 non-null   float64
 8   personal_status         1000 non-null   category
 9   other_parties           1000 non-null   category
 10  residence_since         1000 non-null   float64
 11  property_magnitude      1000 non-null   category
 12  age                     1000 non-null   float64
 13  other_payment_plans     1000 non-null   category
 14  housing                 1000 non-null   category
 15  existing_credits        1000 non-null   float64
 16  job                     1000 non-null   category
 17  num_dependents          1000 non-null   float64
 18  own_telephone           1000 non-null   category
 19  foreign_worker          1000 non-null   category
dtypes: category(13), float64(7)
memory usage: 69.9 KB
```

```
categorical_1 = [var for var in X.columns if X[var].dtype == "category"]
discrete_1 = [var for var in X.columns if X[var].dtype != "category" and (len(X[var].unique())< 20)]
continous_1 = [ var for var in X.columns if X[var].dtype != 'category'
              and var not in discrete_1]
```

```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.33,random_state=0)
```

```
from sklearn.base import BaseEstimator,TransformerMixin

class ConvertToNumpyArray(BaseEstimator,TransformerMixin):
    def __init__(self):
        pass
    def fit(self,X,y=None):
        return self
    def transform(self, X):
        return np.array(X)
```

```
rare_labels_1 = ["foreign_worker","purpose"]
columns_to_transform_1 = ["age","credit_amount","duration"]
```

```
EDA_credit_svc_1 = Pipeline([
    ('rare_label_encoder',RareLabelEncoder(n_categories=1,variables=rare_labels_1,ignore_format=True)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical_1,ignore_format = True)),
    ('log_transformer',LogTransformer(variables=columns_to_transform_1)),
    ('scaler',MaxAbsScaler()),
    ('array_conversion',ConvertToNumpyArray()),
    ('svc',LinearSVC(penalty ='l2',random_state=0, max_iter =100000, dual = False
                       ))
])
```

```
param_grid_linear_1 = {"svc__C":(np.linspace(0.001,1000,5))}
```

```
grid_svm_linear = RandomizedSearchCV(EDA_credit_svc_1,param_grid_linear_1,cv=5, return_train_score=True)
```

```
grid_svm_linear.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:305: UserWarning: The total space of parameters 5
  warnings.warn(
```

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ LogTransformer
  ▸ MaxAbsScaler
  ▸ ConvertToNumpyArray
  ▸ LinearSVC
```

```
print(f"best parameter: {grid_svm_linear.best_params_}")
print(f"best validation score: {grid_svm_linear.best_score_}")
```

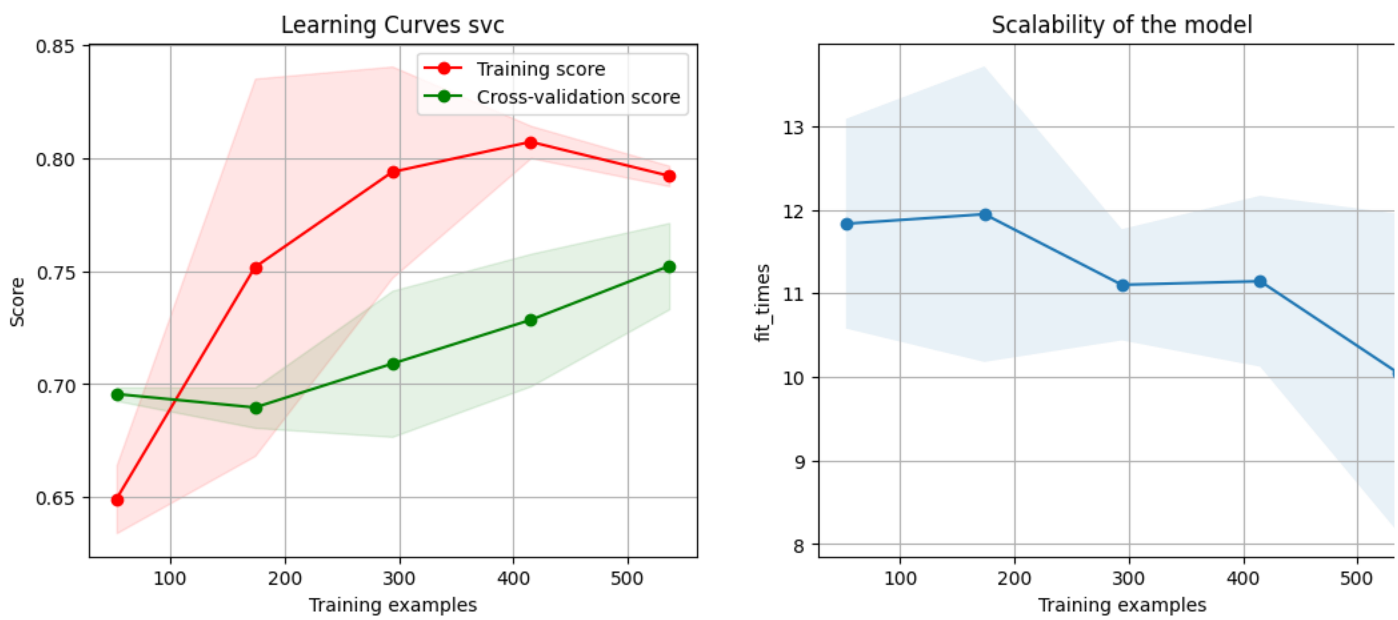
```
best parameter: {'svc__C': 250.00075}
best validation score: 0.7522388059701492
```

```
print(f"test_score: {grid_svm_linear.score(X_test,y_test)}")
```

```
test_score: 0.7545454545454545
```

```
plot_learning_curve(grid_svm_linear, 'Learning Curves svc', X_train, y_train, n_jobs=-1)
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```



Lets narrow the search by C parameter between 200 to 300

```
param_grid_linear_2 = {"svc__C":(np.linspace(200,400,20))}
```

```
grid_svm_linear_2 = RandomizedSearchCV(EDA_credit_svc_1,param_grid_linear_2,cv=5, return_train_score=True)
```

```
grid_svm_linear_2.fit(X_train,y_train)
```

▸ **RandomizedSearchCV**
▸ **estimator: Pipeline**
▸ RareLabelEncoder

```
print(f"best parameter: {grid_svm_linear_2.best_params_}")
print(f"best validation score: {grid_svm_linear_2.best_score_}")
```

```
best parameter: {'svc__C': 378.94736842105266}
best validation score: 0.7522388059701492
```

Lets narrow down the search of C to 400 to 2000

```
param_grid_linear_3 = {"svc__C":range(400,2000,10)}
```

```
grid_svm_linear_3 = RandomizedSearchCV(EDA_credit_svc_1,param_grid_linear_3,cv=5, return_train_score=True)
```

```
grid_svm_linear_3.fit(X_train,y_train)
```

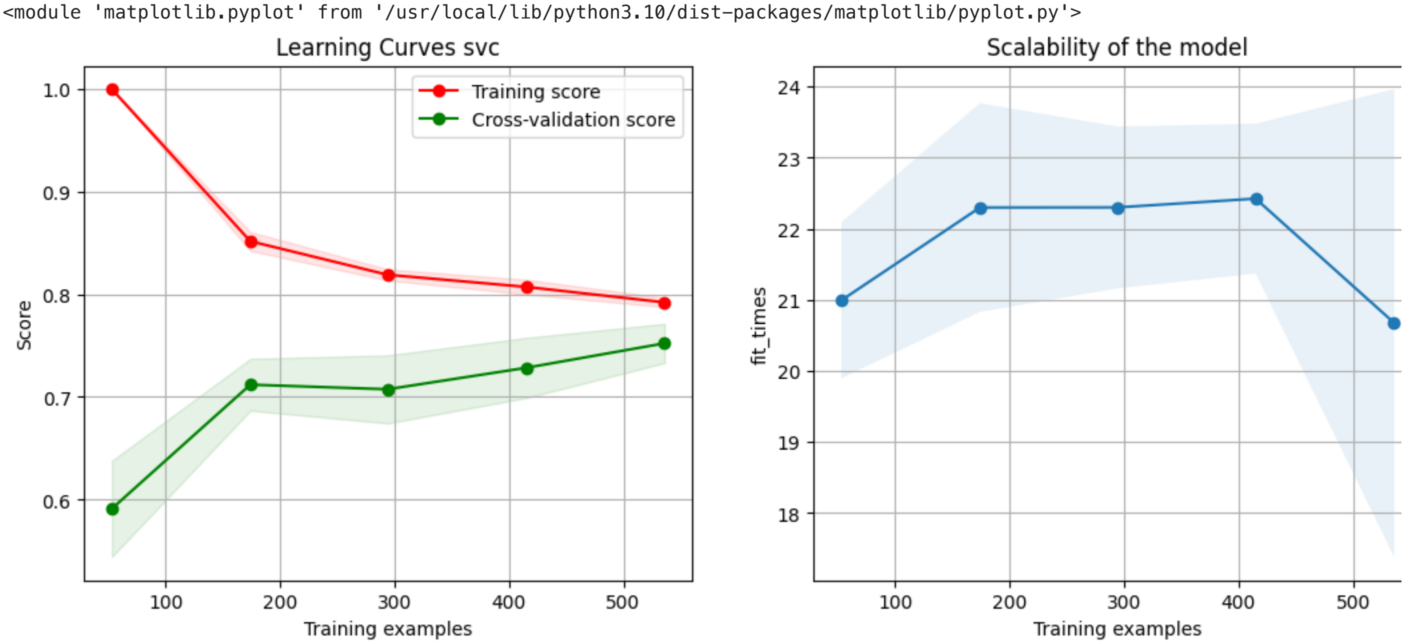
▸ **RandomizedSearchCV**
▸ **estimator: Pipeline**
▸ RareLabelEncoder
▸ OneHotEncoder
▸ LogTransformer
▸ MaxAbsScaler
▸ ConvertToNumpyArray
▸ LinearSVC

```
print(f"best parameter: {grid_svm_linear_3.best_params_}")
print(f"best validation score: {grid_svm_linear_3.best_score_}")
```

```
best parameter: {'svc__C': 1500}
best validation score: 0.7522388059701492
```

```
plot_learning_curve(grid_svm_linear_3, 'Learning Curves svc', X_train, y_train, n_jobs=-1)
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```




```
print(f"Test score is {grid_svm_linear_3.score(X_test,y_test)}")
```

```
Test score is 0.7545454545454545
```

Now let us Try Kernel's Trick with "rbf", "poly" , "sigmoid"

```
EDA_credit_svc_2 = Pipeline([
    ('rare_label_encoder',RareLabelEncoder(n_categories=1,variables=rare_labels_1,ignore_format=True)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical_1,ignore_format = True)),
    ('log_transformer',LogTransformer(variables=columns_to_transform_1)),
    ('scaler',MaxAbsScaler()),
    ('array_conversion',ConvertToNumpyArray()),
    ('svc',SVC(random_state=0))
])
```

```
param_grid_kernel_1 = {"svc__kernel":['rbf','sigmoid','poly'],
                        "svc__C":loguniform(1,10000),
                        "svc__gamma":loguniform(0.001,1000),
                        "svc__degree":[2,3,4,5]}
```

```
grid_svm_kernel = RandomizedSearchCV(EDA_credit_svc_2,param_grid_kernel_1,cv=5, return_train_score=True)
```

```
grid_svm_kernel.fit(X_train,y_train)
```

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
  ▸ RareLabelEncoder
  ▸ OneHotEncoder
  ▸ LogTransformer
  ▸ MaxAbsScaler
  ▸ ConvertToNumpyArray
  ▸ SVC
```

```
print(f"best parameter: {grid_svm_kernel.best_params_}")
print(f"best validation score: {grid_svm_kernel.best_score_}")
```

```
best parameter: {'svc__C': 19.52793716713594, 'svc__degree': 5, 'svc__gamma': 0.0015902219680002784, 'svc__kernel': 'rbf'}
best validation score: 0.746268656716418
```

```
results_kernel_1 = pd.DataFrame(grid_svm_kernel.cv_results_)
```

```
results_kernel_1.head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_svc__C	param_svc__degree	param_svc__gamma	param_svc__
0	0.112364	0.007341	0.074729	0.002730	7.163216	4	0.131005	
1	0.129688	0.020764	0.104632	0.025288	19.527937	5	0.00159	
2	0.173874	0.005451	0.123272	0.003798	2672.505123	3	4.052923	
3	0.135510	0.021994	0.096442	0.021027	8488.441493	5	80.037648	
4	0.107063	0.008252	0.077654	0.006775	3.945347	4	0.066555	

5 rows × 24 columns

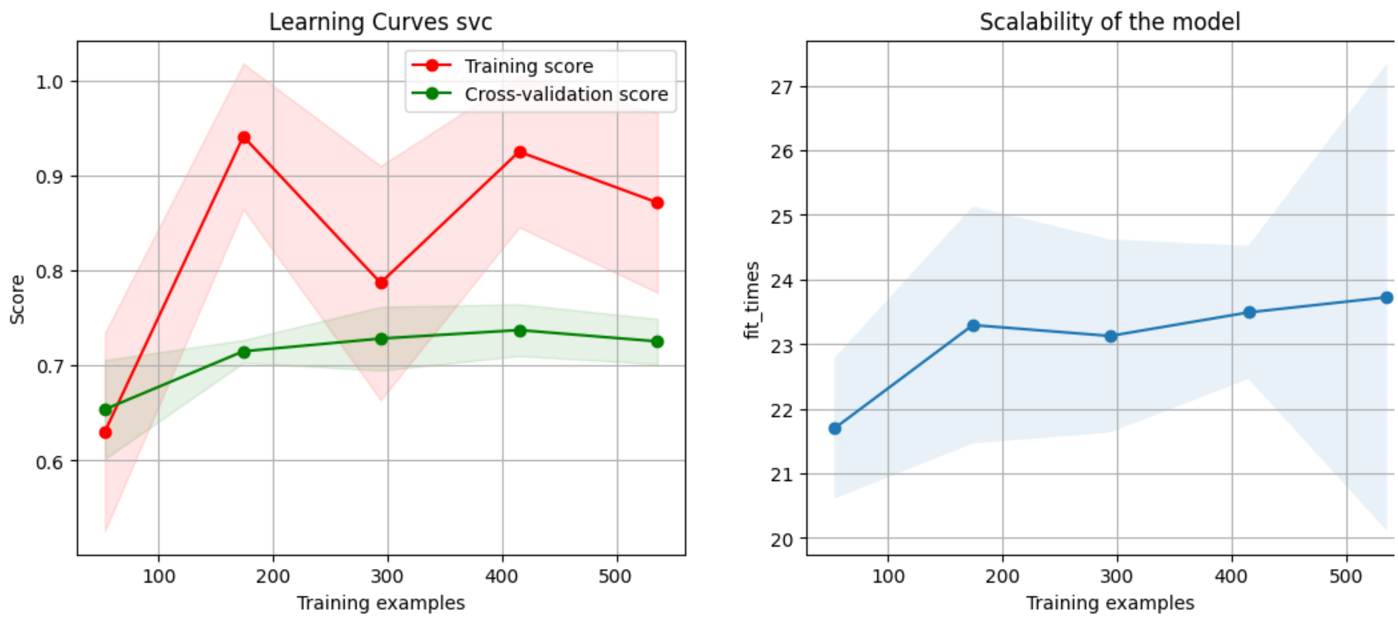
```
results_kernel_1.sort_values(by="mean_test_score",ascending=False,inplace=True)
results_kernel_1[[
```

```
    'param_svc__C', 'param_svc__kernel', 'param_svc__gamma','param_svc__degree',
    'mean_test_score', 'std_test_score', 'mean_train_score'
]].head(10)
```

	param_svc__C	param_svc__kernel	param_svc__gamma	param_svc__degree	mean_test_score	std_test_score	mean_train_score
1	19.527937	rbf	0.00159	5	0.746269	0.012487	0.775373
4	3.945347	poly	0.066555	4	0.725373	0.024251	0.983209
7	219.993999	sigmoid	0.015326	2	0.723881	0.009440	0.716791
3	8488.441493	sigmoid	80.037648	5	0.695522	0.002985	0.695522
5	1.128695	poly	0.023633	5	0.695522	0.002985	0.695522
6	324.580269	sigmoid	588.521498	2	0.695522	0.002985	0.695522
8	3541.34636	rbf	0.84626	2	0.695522	0.005585	1.000000
9	144.1473	sigmoid	4.096385	5	0.695522	0.002985	0.695522
2	2672.505123	rbf	4.052923	3	0.694030	0.004720	1.000000
0	7.163216	sigmoid	0.131005	4	0.668657	0.027763	0.616418

```
plot_learning_curve(grid_svm_kernel, 'Learning Curves svc', X_train, y_train, n_jobs=-1)
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```



As i see better result on rbf, let me try with diff parameters

```
param_grid_kernel_2 = {"svc__kernel":['rbf'],
                       "svc__C":range(2000,4000,20),
                       "svc__gamma":np.linspace(0.01,1.0,20)}
```

```
grid_svm_kernel_2 = RandomizedSearchCV(EDA_credit_svc_2,param_grid_kernel_2,cv=5, return_train_score=True)
```

```
grid_svm_kernel_2.fit(X_train,y_train)
```

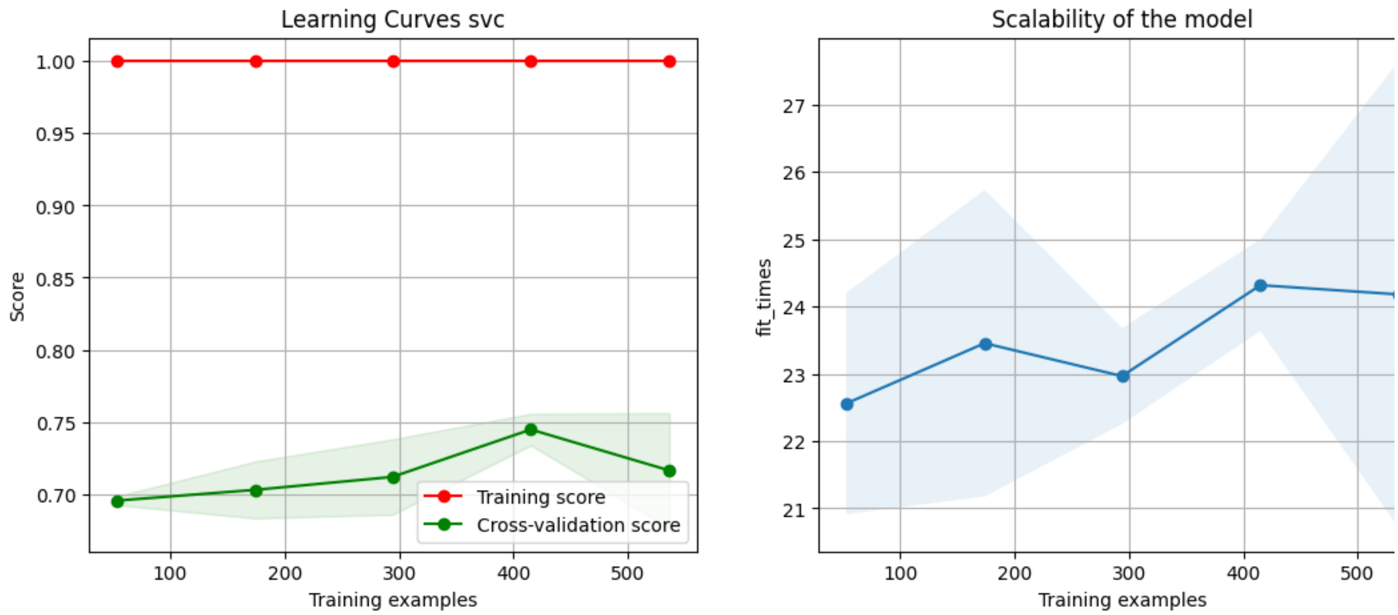
RandomizedSearchCV
estimator: Pipeline
RareLabelEncoder

```
print(f"best parameters: {grid_svm_kernel_2.best_params_}")
print(f"best score: {grid_svm_kernel_2.best_score_} ")
```

```
best parameters: {'svc__kernel': 'rbf', 'svc__gamma': 0.37473684210526315, 'svc__C': 3080}
best score: 0.726865671641791
```

```
plot_learning_curve(grid_svm_kernel_2, 'Learning Curves svc', X_train, y_train, n_jobs=-1)
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```



I see that Linear SVM and SVM with kernel = "rbf" are giving me best fitting my data and then generalizing.

As Linear SVC is a simpler model, I would prefer **grid_svm_linear_3**

Question2 (10 Points) : SVR on Bike Sharing Dataset

- Download the data from following link: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand'
- **Use RandomSerachCV(OR HalvingGridsearchCV, HalvingRandomSerachCV)**

```
data = pd.read_csv('/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Datasets/SeoulBikeData.csv', encoding='latin-1')
```

```
data_1 = data.drop(['Date'],axis=1)
```

```
categorical = [var for var in data_1.columns if data_1[var].dtype == 'O'and var not in ['Rented Bike Count']]
discrete = [var for var in data_1.columns if data_1[var].dtype != 'O'and len(data_1[var].unique()) < 20 and var not in ['Rented B
continuous = [var for var in data_1.columns if data_1[var].dtype != 'O' and var not in discrete and var not in ['Rented Bike Coun
```

```
categorical
```

```
['Seasons', 'Holiday', 'Functioning Day']
```

```
A = data_1.drop(['Rented Bike Count'], axis=1)
b = data_1['Rented Bike Count']
A_train,A_test, b_train, b_test = train_test_split(A,b,random_state=0,test_size=0.33)
```

```
categorical
```

```
['Seasons', 'Holiday', 'Functioning Day']
```

```
A.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Hour                       8760 non-null   int64
 1   Temperature(°C)            8760 non-null   float64
 2   Humidity(%)                8760 non-null   int64
 3   Wind speed (m/s)           8760 non-null   float64
 4   Visibility (10m)           8760 non-null   int64
 5   Dew point temperature(°C)  8760 non-null   float64
 6   Solar Radiation (MJ/m2)    8760 non-null   float64
 7   Rainfall(mm)               8760 non-null   float64
 8   Snowfall (cm)              8760 non-null   float64
 9   Seasons                    8760 non-null   object
 10  Holiday                    8760 non-null   object
 11  Functioning Day            8760 non-null   object
dtypes: float64(6), int64(3), object(3)
memory usage: 821.4+ KB
```

```
columns_to_drop = ['Dew point temperature(°C)']
```

```
columns_to_transform = ['Wind speed (m/s)','Rainfall(mm)','Snowfall (cm)','Hour','Solar Radiation (MJ/m2)']
```

```
columns_to_scale = ['Hour',
 'Temperature(°C)',
 'Humidity(%)',
 'Wind speed (m/s)',
 'Visibility (10m)',
 'Dew point temperature(°C)',
 'Solar Radiation (MJ/m2)',
 'Rainfall(mm)',
 'Snowfall (cm)']
```

```
rare_labels =['Functioning Day']
```

```
from feature_engine.transformation import YeoJohnsonTransformer
from feature_engine.selection.drop_correlated_features import Variables
from sklearn.preprocessing import MinMaxScaler
from feature_engine.selection import DropFeatures



EDA_bike = Pipeline([
    ('drop_features',DropFeatures(columns_to_drop)),
    ('rare_label_encoder',RareLabelEncoder(n_categories=1,variables=rare_labels,ignore_format=True)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical,ignore_format = True)),
    ('yj_transformer',YeoJohnsonTransformer(variables=columns_to_transform)),
    ('scaler',MinMaxScaler()),
    ('array_conversion',ConvertToNumpyArray()),
    ('svm',LinearSVC(penalty ='l2',random_state=0, max_iter =100000, dual = False))
    ])
```

```
param_grid_bike_linear_1 = {"svm__C":range(1,1000,10)}
```

```
grid_bike_linear_1 = RandomizedSearchCV(EDA_bike,param_grid_bike_linear_1,cv=5, return_train_score=True)
```

```
grid_bike_linear_1.fit(A_train,b_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated class in y ha
  warnings.warn(
```

▸ **RandomizedSearchCV**

▸ **estimator: Pipeline**

▸ DropFeatures

▸ RareLabelEncoder

```
print(f"best parameter : {grid_bike_linear_1.best_params_}")
print(f"best Score : {grid_bike_linear_1.best_score_}")
```

```
best parameter : {'svm__C': 21}
best Score : 0.035611015015590705
```

```
EDA_bike_2 = Pipeline([
    ('drop_features',DropFeatures(columns_to_drop)),
    ('rare_label_encoder',RareLabelEncoder(n_categories=1,variables=rare_labels,ignore_format=True)),
    ('one_hot_encoder',OneHotEncoder(variables=categorical,ignore_format = True)),
    ('yj_transformer',YeoJohnsonTransformer(variables=columns_to_transform)),
    ('scaler',MinMaxScaler()),
    ('array_conversion',ConvertToNumpyArray()),
    ('svm',SVC())
    ])
```

```
param_grid_bike_kernel_1 = {"svm__kernel": ['rbf',"sigmoid"],
                            "svm__C":range(1,1000,100),
                            "svm__gamma":range(1,10,10)
                            }
```

```
grid_bike_kernerl_1 = RandomizedSearchCV(EDA_bike_2, param_grid_bike_kernel_1,cv=5,return_train_score = True)
```

```
grid_bike_kernerl_1.fit(A_train,b_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated class in y ha
  warnings.warn(
```

▸ **RandomizedSearchCV**

▸ **estimator: Pipeline**

▸ DropFeatures

▸ RareLabelEncoder

▸ OneHotEncoder

▸ YeoJohnsonTransformer

▸ MinMaxScaler

▸ ConvertToNumpyArray

▸ SVC

```
print(f"best parameter : {grid_bike_kernerl_1.best_params_}")
print(f"best score: {grid_bike_kernerl_1.best_score_}")
```

```
best parameter : {'svm__kernel': 'sigmoid', 'svm__gamma': 1, 'svm__C': 201}
best score: 0.035269863815461745
```

```
param_grid_bike_kernel_2 = {"svm__kernel": ['poly'],
                            "svm__C":loguniform(0.01, 1000),
                            "svm__gamma":loguniform(0.0001, 10),
                            "svm__degree":[2,3],
                            }
```

```
grid_bike_kernerl_2 = RandomizedSearchCV(EDA_bike_2, param_grid_bike_kernel_2,cv=5,return_train_score = True)
```

```
grid_bike_kernerl_2.fit(A_train,b_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated class in y ha
  warnings.warn(
```

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
    ▸ DropFeatures
    ▸ RareLabelEncoder
    ▸ OneHotEncoder
    ▸ YeoJohnsonTransformer
    ▸ MinMaxScaler
    ▸ ConvertToNumpyArray
```

- Compare KNN and Linear Regression.(previous HWs) Basd on your anaysis which algorithm you will recommend.
- The aim of the piepline is to predict the rented bike count.

```
print(f"best parameter : {grid_bike_kernerl_2.best_params_}")
print(f"best score: {grid_bike_kernerl_2.best_score_}")
```

```
best parameter : {'svm__C': 621.3174804535482, 'svm__degree': 2, 'svm__gamma': 0.025376933991757274, 'svm__kernel': 'poly'}
best score: 0.03561072455054164
```

```
results_kernel_bike_1 = pd.DataFrame(grid_bike_kernerl_1.cv_results_)
```

```
results_kernel_bike_2 = pd.DataFrame(grid_bike_kernerl_2.cv_results_)
```

```
results_kernel_bike_1.head()
```

it_time	mean_score_time	std_score_time	param_svm__kernel	param_svm__gamma	param_svm__C	param
0.537359	20.961038	0.450974	sigmoid	1	201	{'svm__kerne 'sigmoic 'svm__gamma 1, 's.
0.414934	22.715788	0.420049	rbf	1	101	{'svm__kerne 'rb' 'svm__gamma 1, 'svm__.
0.358880	22.698250	1.187799	sigmoid	1	501	{'svm__kerne 'sigmoic 'svm__gamma 1, 's.
0.772319	21.114753	0.284364	rbf	1	1	{'svm__kerne 'rb' 'svm__gamma 1, 'svm__.
0.457812	22.307405	0.763006	rbf	1	201	{'svm__kerne 'rb' 'svm__gamma 1, 'svm__.

```
results_kernel_bike_1.sort_values(by="mean_test_score",ascending=False,inplace=True)
```

```
results_kernel_bike_1[[
    'param_svm__C', 'param_svm__kernel', 'param_svm__gamma',
    'mean_test_score', 'std_test_score', 'mean_train_score'
]].head(10)
```

	param_svm__C	param_svm__kernel	param_svm__gamma	mean_test_score	std_test_score	mean_train_score
9	201	sigmoid	1	0.035270	0.001015	0.035824
2	101	rbf	1	0.035100	0.001003	0.866843
0	501	sigmoid	1	0.035100	0.001247	0.035696
7	1	rbf	1	0.035100	0.001247	0.062106
5	201	rbf	1	0.034759	0.001371	0.956424
3	801	rbf	1	0.034419	0.001286	0.996848

```
results_kernel_bike_2.sort_values(by="mean_test_score",ascending=False,inplace=True)
results_kernel_bike_2[[
    'param_svm__C', 'param_svm__kernel', 'param_svm__gamma','param_svm__degree',
    'mean_test_score', 'std_test_score', 'mean_train_score'
]].head(10)
```

	param_svm__C	param_svm__kernel	param_svm__gamma	param_svm__degree	mean_tes
2	621.31748	poly	0.025377	2	
6	381.401557	poly	0.047502	3	
7	0.024123	poly	0.495455	3	
3	56.78826	poly	0.489706	3	
8	26.210429	poly	5.474129	2	
0	0.198719	poly	0.001937	2	
1	0.042373	poly	0.023155	3	
4	6.886387	poly	0.045188	3	
5	0.071885	poly	0.000278	3	
9	133.521926	poly	0.001155	2	

We see that the Validation score is very less compared to KNN and linear, SVM is not a better fit for this data