```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats.mstats import normaltest
import seaborn as sns

#import file and read it
data_path = "C:\\Users\\elena\\Documents\\Machine Learning IBM\\Course 3 classification
data = pd.read_csv(data_path)
```

```
data.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProduct
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3
3	4	15701354	Boni	699	France	Female	39	1	0.00	2
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1

```
#drop not interestin columns: CustomerId, RowNumberm Surname and EstimatedSalary
data = data.iloc[:, 3:].drop(['EstimatedSalary'],1)
data.shape[1]
```

```
10
```

```
data.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	1
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	

```
data.dtypes.value_counts()
```

```
int64      7
object     2
float64    1
```

```
dtype: int64
```

```
data.Gender.value_counts()
data.Tenure.value_counts()
data.NumOfProducts.value_counts()
data.HasCrCard.value_counts()
data.IsActiveMember.value_counts()
data.Exited.value_counts()
```

```
0    7963
1    2037
Name: Exited, dtype: int64
```

```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x1a6fe6e92e0>
```

```python
from sklearn.preprocessing import LabelEncoder
#tranforming the categorical features in numerical
le = LabelEncoder()
data['Geography'] = le.fit_transform(data.Geography)
data['Gender'] = le.fit_transform(data.Gender)
```

```python
#separating target from features
y = data.Exited
X = data.drop('Exited', axis=1)
```

```python
y.value_counts(normalize=True)
```

```
0    0.7963
1    0.2037
Name: Exited, dtype: float64
```

```python
from sklearn.model_selection import StratifiedShuffleSplit
strat_shuf_split = StratifiedShuffleSplit(n_splits=1,
                                          test_size=0.3,
                                          random_state=42)


train_idx, test_idx = next(strat_shuf_split.split(X, y))

# Create the dataframes
X_train = X.loc[train_idx, X.columns]
y_train = y.loc[train_idx]
X_test = X.loc[test_idx, X.columns]
y_test = y.loc[test_idx]
len(test_idx)
```

```
3000
```

```python
y_train.value_counts(normalize=True)
```

```
0    0.796286
1    0.203714
Name: Exited, dtype: float64
```

```python
y_test.value_counts(normalize=True)
```

```
0    0.796333
1    0.203667
Name: Exited, dtype: float64
```

```python
from sklearn.preprocessing import StandardScaler

s = StandardScaler()
X_ss = s.fit_transform(X_train)
X_st = s.fit_transform(X_test)
X_ss
```

```
array([[-0.57558225, -0.90716852,  0.90036493, ..., -0.91248301,
         0.6430943 , -1.03459817],
       [ 0.39818245,  1.50368176, -1.11066076, ...,  0.79949262,
         0.6430943 ,  0.96655883],
       [ 1.69308232, -0.90716852,  0.90036493, ..., -0.91248301,
         0.6430943 , -1.03459817],
       ...,
       [ 0.17028007, -0.90716852, -1.11066076, ...,  0.79949262,
         0.6430943 ,  0.96655883],
       [ 0.37746405,  1.50368176,  0.90036493, ...,  0.79949262,
         0.6430943 , -1.03459817],
       [ 1.56877193,  1.50368176,  0.90036493, ..., -0.91248301,
        -1.55498191,  0.96655883]])
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV


lr = LogisticRegression(solver='liblinear').fit(X_ss, y_train)
# L1 regularized logistic regression
lr_l1 = LogisticRegressionCV(Cs=[0.1,0.5], penalty='l1', solver='liblinear').fit(X_ss,
#print(list(zip(list(lr.coef_[0,:]), list(X.columns))))
coeff_lr = pd.Series(lr.coef_[0,:], X.columns).sort_values(ascending = False)
coeff_lr
```

```
Age                0.703164
Balance            0.306756
Geography          0.065396
NumOfProducts      0.010632
Tenure             0.007484
HasCrCard         -0.006773
CreditScore       -0.098258
Gender            -0.254163
IsActiveMember    -0.507378
dtype: float64
```

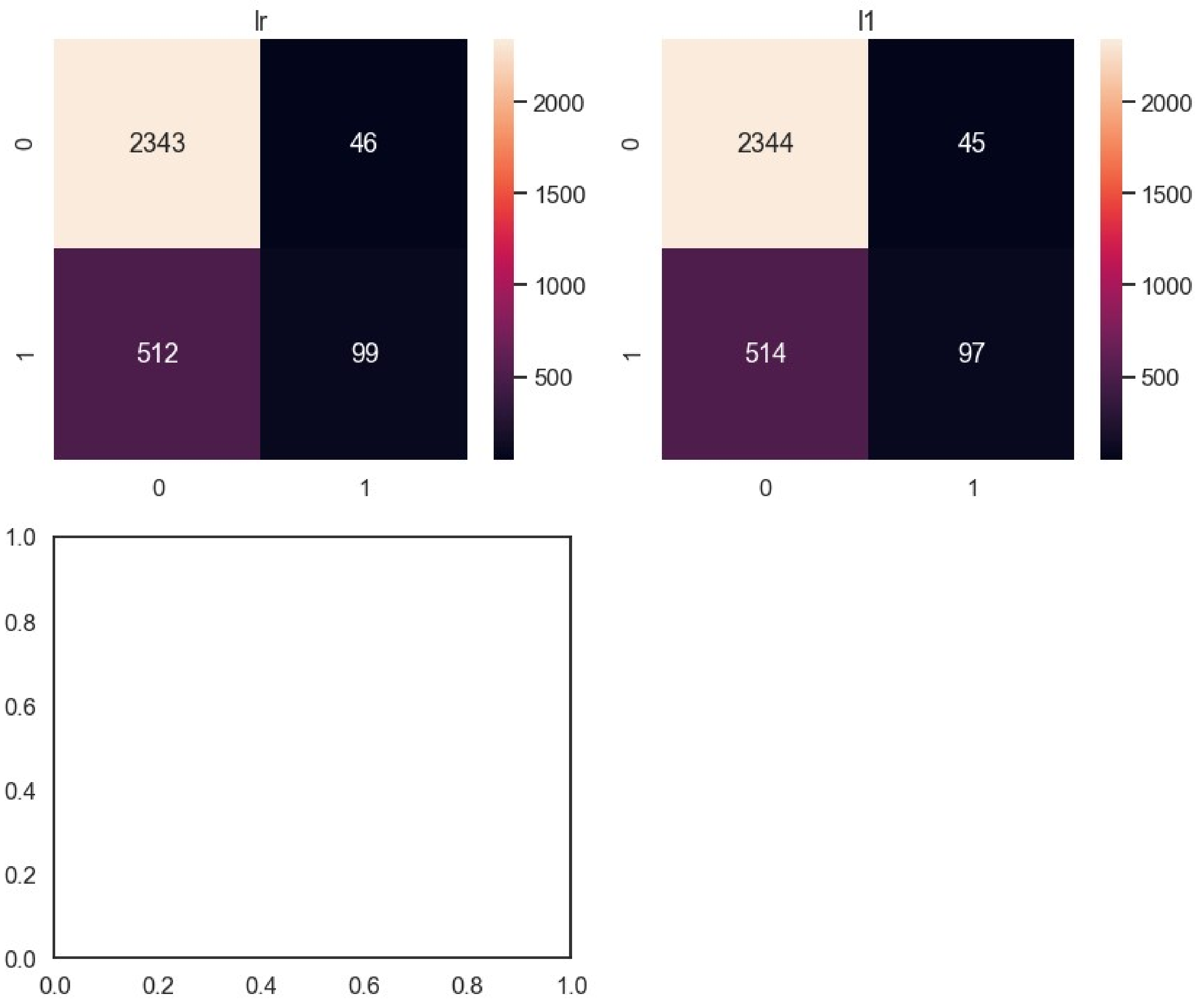
```python
coeff_l1 = pd.Series(lr_l1.coef_[0,:], X.columns).sort_values(ascending = False)
coeff_l1
```

```
Age                0.701121
Balance            0.304106
Geography          0.063516
NumOfProducts      0.008148
Tenure             0.005514
HasCrCard         -0.004716
CreditScore       -0.096167
Gender            -0.252255
IsActiveMember    -0.505175
dtype: float64
```

```
#comparing logistic regression with and without regularization
y_pred = list()
y_prob = list()

coeff_labels = ['lr', 'l1']
coeff_models = [lr, lr_l1]

for lab,mod in zip(coeff_labels, coeff_models):
    y_pred.append(pd.Series(mod.predict(X_st), name=lab))
    y_prob.append(pd.Series(mod.predict_proba(X_st).max(axis=1), name=lab))

y_pred = pd.concat(y_pred, axis=1)
y_prob = pd.concat(y_prob, axis=1)

from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score
from sklearn.preprocessing import label_binarize

metrics = list()
cm = dict()

for lab in coeff_labels:

    # Preciision, recall, f-score from the multi-class support function
    precision, recall, fscore, _ = score(y_test, y_pred[lab], average='weighted')

    # The usual way to calculate accuracy
    accuracy = accuracy_score(y_test, y_pred[lab])

    # ROC-AUC scores can be calculated by binarizing the data
    auc = roc_auc_score(label_binarize(y_test, classes=[0,1]),
              label_binarize(y_pred[lab], classes=[0,1]),
              average='weighted')

    # Last, the confusion matrix
    cm[lab] = confusion_matrix(y_test, y_pred[lab])

    metrics.append(pd.Series({'precision':precision, 'recall':recall,
                              'fscore':fscore, 'accuracy':accuracy,
                              'auc':auc},
                             name=lab))

metrics = pd.concat(metrics, axis=1)
metrics
```

	lr	l1
precision	0.792578	0.792240
recall	0.814000	0.813667
fscore	0.764939	0.763966
accuracy	0.814000	0.813667
auc	0.571387	0.569960

```
import seaborn as sns
fig, axList = plt.subplots(nrows=2, ncols=2)
axList = axList.flatten()
fig.set_size_inches(12, 10)

axList[-1].axis('off')

for ax,lab in zip(axList[:-1], coeff_labels):
    sns.heatmap(cm[lab], ax=ax, annot=True, fmt='d');
    ax.set(title=lab);

plt.tight_layout()
```



```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1


param_grid = {'Cs':[0.1,0.5,1,10,50,100]}

LR = GridSearchCV(LogisticRegressionCV(penalty='l2'),
                  param_grid,
                  scoring = 'accuracy',
                  n_jobs=-1)

LR = LR.fit(X_ss, y_train)
y_pred = LR.best_estimator_.predict(X_st)
print(classification_report(y_test, y_pred))
print('Accuracy score: ', round(accuracy_score(y_test, y_pred), 2))
print('F1 Score: ', round(f1_score(y_test, y_pred), 2))
```

```
C:\Users\elena\ana\lib\site-packages\sklearn\model_selection\_search.py:918: UserWarnin
g: One or more of the test scores are non-finite: [        nan        nan 0.79628571 0.8
0357143 0.80371429 0.80357143]
  warnings.warn(
              precision    recall  f1-score   support

           0       0.82      0.98      0.89      2389
           1       0.69      0.14      0.24       611

    accuracy                           0.81      3000
   macro avg       0.75      0.56      0.56      3000
weighted avg       0.79      0.81      0.76      3000

Accuracy score:  0.81
F1 Score:  0.24
```

```
LR.best_estimator_
```

```
LogisticRegressionCV(Cs=50)
```

```
#coefficients of best estimator
coeff_LR = pd.Series(LR.best_estimator_.coef_[0,:], X.columns).sort_values(ascending =
coeff_LR
```

```
Age                0.663989
Balance            0.287540
Geography          0.063014
Tenure             0.007904
NumOfProducts      0.005341
HasCrCard         -0.006672
CreditScore       -0.092665
Gender            -0.241126
IsActiveMember    -0.472139
dtype: float64
```

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn = knn.fit(X_ss, y_train)
y_pred = knn.predict(X_st)
# Preciision, recall, f-score from the multi-class support function
print(classification_report(y_test, y_pred))
print('Accuracy score: ', round(accuracy_score(y_test, y_pred), 2))
print('F1 Score: ', round(f1_score(y_test, y_pred), 2))
```

```
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      2389
           1       0.62      0.49      0.55       611

    accuracy                           0.84      3000
   macro avg       0.75      0.71      0.72      3000
weighted avg       0.82      0.84      0.83      3000

Accuracy score:  0.84
F1 Score:  0.55
```

```python
_, ax = plt.subplots(figsize=(12,12))
ax = sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', annot_kws={"siz
labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=25);
ax.set_yticklabels(labels[::-1], fontsize=25);
ax.set_ylabel('Prediction', fontsize=30);
ax.set_xlabel('Ground Truth', fontsize=30)
```

```
Text(0.5, 76.5, 'Ground Truth')
```


```
param_grid = {'n_neighbors':[1,3,5,10,20,40,50,100]}

NN = GridSearchCV(KNeighborsClassifier(),
                  param_grid,
                  scoring = 'accuracy',
                  n_jobs=-1)

NN = NN.fit(X_ss, y_train)
y_pred = NN.best_estimator_.predict(X_st)
print(NN.best_estimator_)
print(classification_report(y_test, y_pred))
print('Accuracy score: ', round(accuracy_score(y_test, y_pred), 2))
print('F1 Score: ', round(f1_score(y_test, y_pred), 2))
```

```
KNeighborsClassifier(n_neighbors=10)
              precision    recall  f1-score   support

           0       0.85      0.97      0.91      2389
           1       0.76      0.35      0.48       611

    accuracy                           0.85      3000
   macro avg       0.81      0.66      0.70      3000
weighted avg       0.84      0.85      0.82      3000

Accuracy score:  0.85
F1 Score:  0.48
```

```
NN.best_estimator_
```

```
KNeighborsClassifier(n_neighbors=10)
```

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
dt = dt.fit(X_train, y_train)
dt.tree_.node_count, dt.tree_.max_depth
```

```
(2161, 23)
```

```
y_train_pred = dt.predict(X_ss)
y_test_pred = dt.predict(X_st)

def measure_error(y_true, y_pred, label):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=label)
```

```
# The error on the training and test data sets
train_test_full_error = pd.concat([measure_error(y_train, y_train_pred, 'train'),
                                   measure_error(y_test, y_test_pred, 'test')],
                                   axis=1)

train_test_full_error
### END SOLUTION
```

	train	test
accuracy	1.0	0.790667
precision	1.0	0.486572
recall	1.0	0.504092
f1	1.0	0.495177

```
dt.feature_importances_
```

```
array([0.19756147, 0.03604984, 0.03124698, 0.24535103, 0.09781713,
       0.19967913, 0.1184814 , 0.01973502, 0.054078  ])
```

```
param_grid = {'max_depth':range(1, dt.tree_.max_depth+1, 2),
              'max_features': range(1, len(dt.feature_importances_)+1)}

GR = GridSearchCV(DecisionTreeClassifier(random_state=42),
                  param_grid=param_grid,
                  scoring='accuracy',
                  n_jobs=-1)

GR = GR.fit(X_ss, y_train)
```

```
GR.best_estimator_
```

```
DecisionTreeClassifier(max_depth=5, max_features=9, random_state=42)
```

```
GR.best_estimator_.tree_.node_count, GR.best_estimator_.tree_.max_depth
y_train_pred_gr = GR.predict(X_ss)
y_test_pred_gr = GR.predict(X_st)

train_test_gr_error = pd.concat([measure_error(y_train, y_train_pred_gr, 'train'),
                                 measure_error(y_test, y_test_pred_gr, 'test')],
                                axis=1)
```

```
y_train_pred_gr = GR.predict(X_ss)
y_test_pred_gr = GR.predict(X_st)

train_test_gr_error = pd.concat([measure_error(y_train, y_train_pred_gr, 'train'),
                                 measure_error(y_test, y_test_pred_gr, 'test')],
                                axis=1)
```

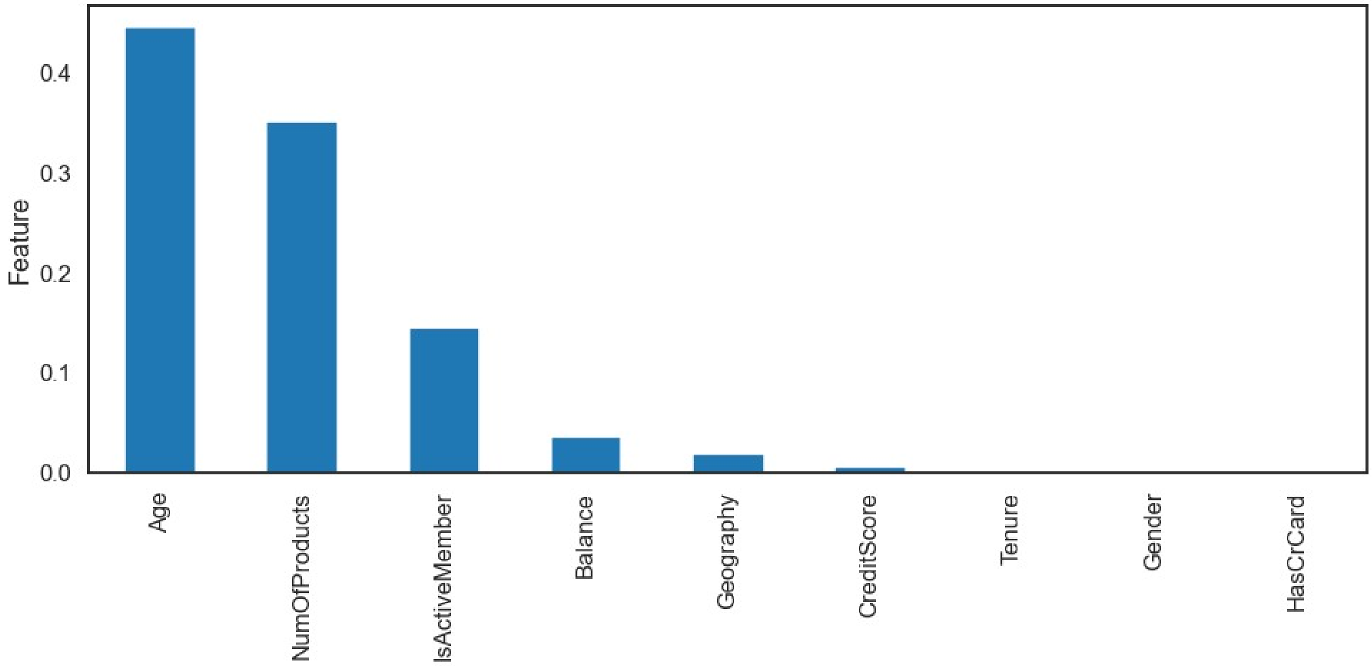
```
train_test_gr_error
```

	train	test
accuracy	0.855571	0.858333
precision	0.815830	0.807947
recall	0.375877	0.399345
f1	0.514642	0.534502

```
pd.Series({'accuracy':accuracy_score(y_test, y_test_pred_gr),
                      'precision': precision_score(y_test, y_test_pred_gr),
                      'recall': recall_score(y_test, y_test_pred_gr),
                      'f1': f1_score(y_test, y_test_pred_gr)},
                      name="GGBC")
```

```
accuracy     0.858333
precision    0.807947
recall       0.399345
f1           0.534502
Name: GGBC, dtype: float64
```

```
feature_imp = pd.Series(GR.best_estimator_.feature_importances_, index=X.columns).sort_

ax = feature_imp.plot(kind='bar', figsize=(16, 6))
ax.set(ylabel='Relative Importance');
ax.set(ylabel='Feature');
```



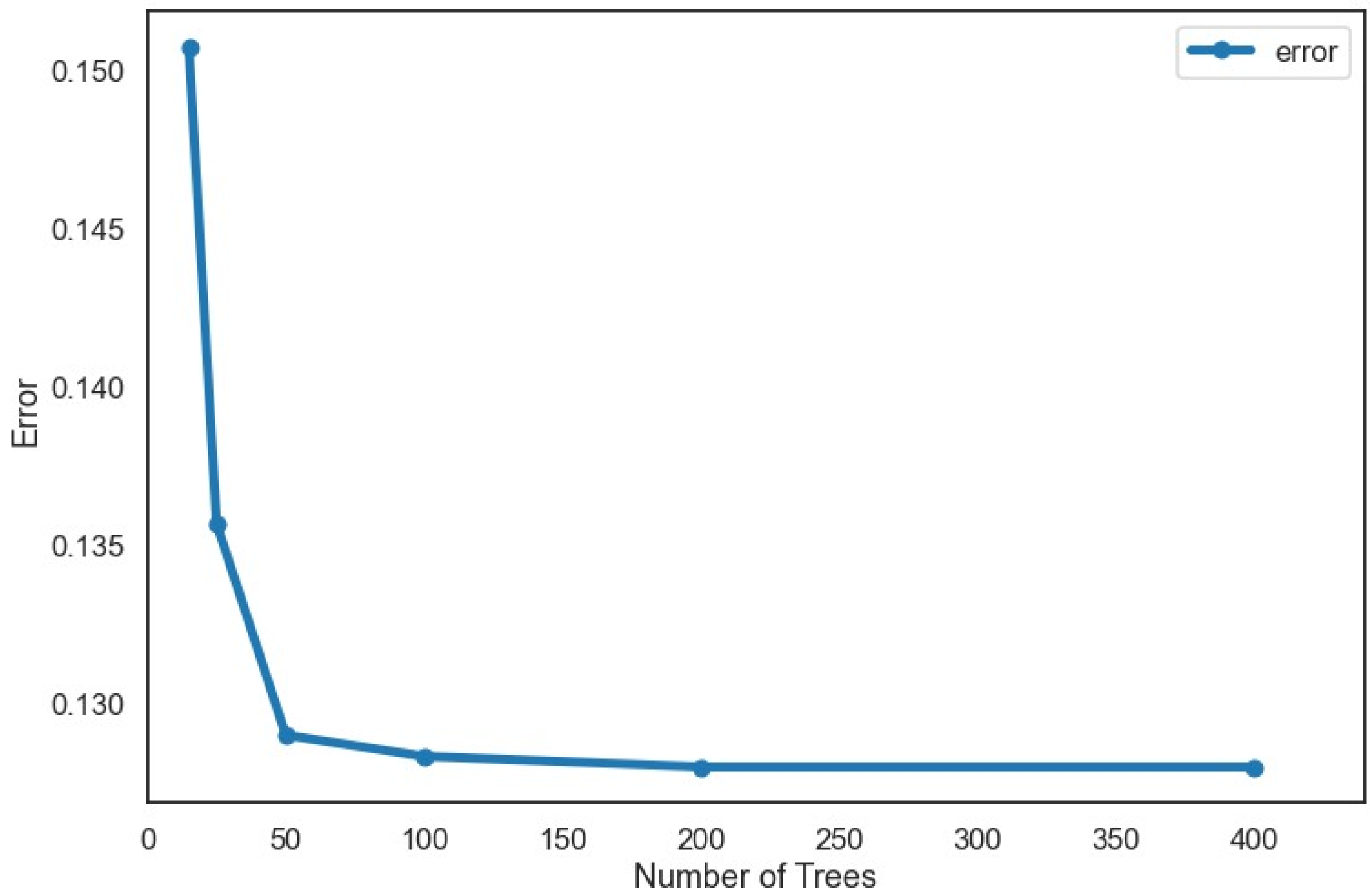

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

error_list = list()

# Iterate through various possibilities for number of trees
tree_list = [15, 25, 50, 100, 200, 400]
for n_trees in tree_list:

    # Initialize the gradient boost classifier
    GBC = GradientBoostingClassifier(n_estimators=n_trees, random_state=42)

    # Fit the model
    print(f'Fitting model with {n_trees} trees')
    GBC.fit(X_ss, y_train)
    y_pred = GBC.predict(X_st)

    # Get the error
    error = 1.0 - accuracy_score(y_test, y_pred)

    # Store it
    error_list.append(pd.Series({'n_trees': n_trees, 'error': error}))

error_df = pd.concat(error_list, axis=1).T.set_index('n_trees')

error_df
```

```
Fitting model with 15 trees
Fitting model with 25 trees
Fitting model with 50 trees
Fitting model with 100 trees
Fitting model with 200 trees
Fitting model with 400 trees
```

	error
n_trees	

	error
n_trees	
15.0	0.150667
25.0	0.135667
50.0	0.129000
100.0	0.128333

```
sns.set_context('talk')
sns.set_style('white')


# Create the plot
ax = error_df.plot(marker='o', figsize=(12, 8), linewidth=5)

# Set parameters
ax.set(xlabel='Number of Trees', ylabel='Error')
ax.set_xlim(0, max(error_df.index)*1.1);
### END SOLUTION
```



```
# The parameters to be fit
param_grid = {'n_estimators': [5.10,15, 25, 50, 100, 200, 400],
              'max_features': [1, 2, 3, 4,6,7,8,9]}

# The grid search object
GV_GBC = GridSearchCV(GradientBoostingClassifier(),
                      param_grid=param_grid,
                      scoring='accuracy',
                      n_jobs=-1)

# Do the grid search
GV_GBC = GV_GBC.fit(X_ss, y_train)
```

```
C:\Users\elena\ana\lib\site-packages\sklearn\model_selection\_search.py:918: UserWarnin
g: One or more of the test scores are non-finite: [        nan 0.81085714 0.82057143 0.8
4457143 0.85557143 0.85585714
 0.85628571        nan 0.82742857 0.845      0.85257143 0.85642857
 0.85528571 0.85814286        nan 0.83171429 0.85014286 0.85285714
 0.85828571 0.858      0.85342857        nan 0.84685714 0.85242857
 0.855      0.85728571 0.858      0.85257143        nan 0.84814286
 0.851      0.85414286 0.85885714 0.85628571 0.85271429        nan
 0.84714286 0.85228571 0.85571429 0.85671429 0.85871429 0.85457143
        nan 0.84728571 0.85242857 0.85585714 0.85857143 0.85828571
 0.854             nan 0.84457143 0.85285714 0.856      0.85914286
 0.85914286 0.85357143]
  warnings.warn(
```

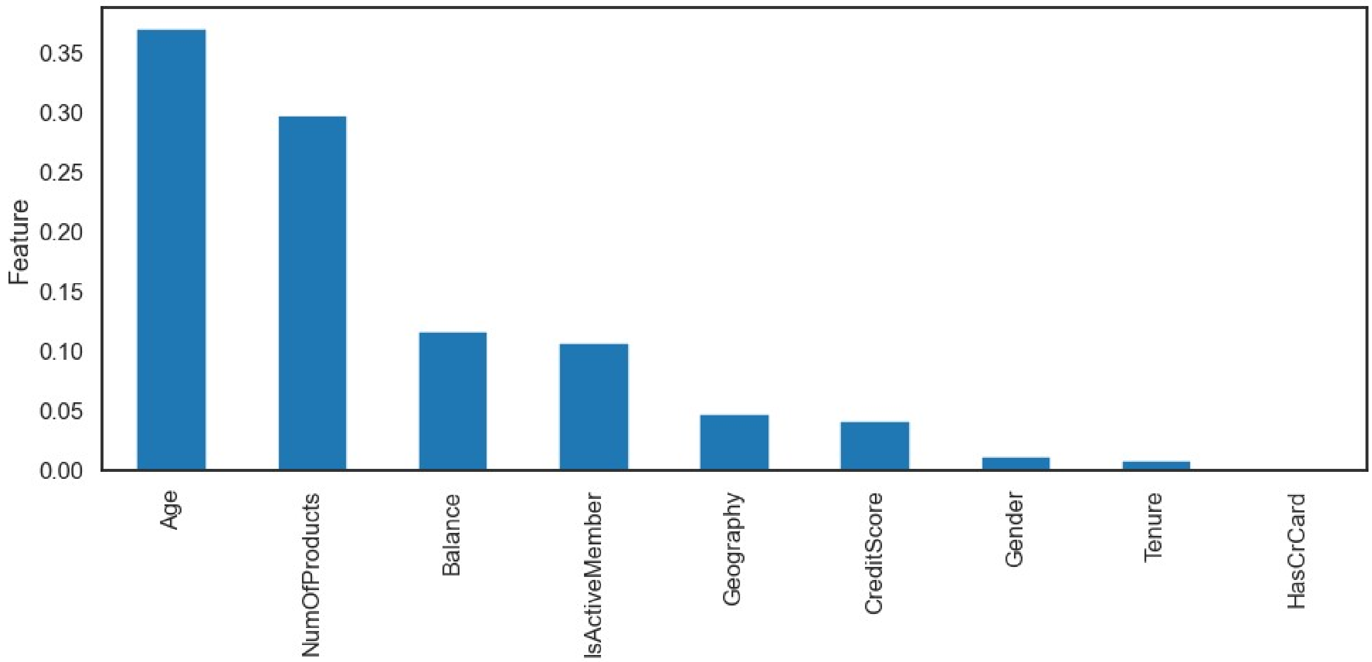
```
GV_GBC.best_estimator_
```

```
GradientBoostingClassifier(max_features=9, n_estimators=200)
```

```
y_pred = GV_GBC.predict(X_st)
print(classification_report(y_pred, y_test))
pd.Series({'accuracy':accuracy_score(y_test, y_pred),
                      'precision': precision_score(y_test, y_pred),
                      'recall': recall_score(y_test, y_pred),
                      'f1': f1_score(y_test, y_pred)},
                      name="GGBC")
```

```
              precision    recall  f1-score   support

           0       0.97      0.88      0.92      2609
           1       0.51      0.79      0.62       391

    accuracy                           0.87      3000
   macro avg       0.74      0.84      0.77      3000
weighted avg       0.91      0.87      0.88      3000

accuracy     0.872000
precision    0.790281
recall       0.505728
f1           0.616766
Name: GGBC, dtype: float64
```

```
feature_imp = pd.Series(GV_GBC.best_estimator_.feature_importances_, index=X.columns).s

ax = feature_imp.plot(kind='bar', figsize=(16, 6))
ax.set(ylabel='Relative Importance');
ax.set(ylabel='Feature');
```




```
y_train_pred_gr = GV_GBC.predict(X_ss)
y_test_pred_gr = GV_GBC.predict(X_st)

train_test_gr_error = pd.concat([measure_error(y_train, y_train_pred_gr, 'train'),
                                 measure_error(y_test, y_test_pred_gr, 'test')],
                                axis=1)
```

```
train_test_gr_error
```

	train	test
accuracy	0.878714	0.872000
precision	0.836639	0.790281
recall	0.502805	0.505728
f1	0.628121	0.616766