First Classifier Prediction Model(s)

The original README file says:

> Often, more than one contact to the same client was required, **in order to access if the product (bank term deposit) would be (or not) subscribed**

This notebook builds on the prior model through:

- Additional feature engineering
- Iterative imputing of missing **age** values
- Scaling the data in three different ways (MinMax, Standard, Robust)
- Running multiple classification algorithms
- Outputting classification scores, e.g. specificity, ROC, True Positives, etc.

In [1]:
```
#import the right libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.metrics import f1_score, confusion_matrix, classification_report, accuracy_score, log_loss, roc_au
from sklearn.model_selection import cross_val_predict, cross_val_score, learning_curve, GridSearchCV, Randomize
from sklearn.metrics import classification_report, accuracy_score, precision_recall_fscore_support
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler, Normalizer, LabelEncoder, RobustScaler, MinMaxScaler
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, KNNImputer
```

In [2]:
```
#this option just allwos us to see every column in the notebook
pd.set_option('display.max_columns', None)

#pd.get_option("display.max_columns")
```

In [3]:
```
#pull in the dataset and turn into a DataFrame
bank_main_df = pd.read_csv('./Dataset_1_Bank Marketing/bank_marketing.csv',delimiter=';')
bank_main_df
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
0	58.0	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0
1	44.0	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0
2	33.0	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0
3	47.0	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0
4	33.0	unknown	single	unknown	no	1	no	no	NaN	5	may	198	1	-1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	51.0	technician	married	tertiary	no	825	no	no	cellular	17	nov	977	3	-1	0
45207	71.0	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456	2	-1	0
45208	72.0	retired	married	secondary	no	5715	no	no	cellular	17	nov	1127	5	184	3
45209	57.0	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508	4	-1	0
45210	37.0	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	361	2	188	11

45211 rows × 17 columns

In [4]:
```
bank_main_df.describe()
```

Out[4]:

	age	balance	day	duration	campaign	pdays	previous
count	43872.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.924781	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.610835	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

In [5]:
```
#creating a field that determines whether or not the customer was ever previously contacted
bank_main_df['prior_contact'] = [ 0 if bank_main_df['pdays'][i] == -1 else 1 for i in range(len(bank_main_df))]

#replacing the yes/no categorical values with 1/0 binary digits
bank_main_df['deposit'] = [0 if (bank_main_df['deposit'][i] == 'yes') else 1 for i in range(len(bank_main_df))

#convert the "day" field to a categorical variable
bank_main_df['day'] = pd.Categorical(bank_main_df['day'])

#dropping pdays and previous, because the important information is captured in prior_contact
bank_main_df.drop(columns=['pdays','previous','duration'],inplace=True)
```

In [6]:
```
#because we have so many cateogrical variables, we should one-hot encode them (i.e. create dummy categorical va
#we also use drop_first=True to reduce the redundant column count
bank_main_df = pd.get_dummies(bank_main_df, drop_first=False)

# bank_main_df
```

In [7]:
```
#note that only the "age" category has null values

# pd.isnull(bank_main_df).sum()
```

Imputing the missing values in "Age" variable

- **Iterative Imputer:** Multivariate imputer that estimates each feature from all the others. A strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion.

https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html

In [8]:
```
# Use multivariate imputer that estimates and imputes null values based on all the others.

imp = IterativeImputer(max_iter=10, verbose=0) # values passed are defaults, but added them because they seem i
imp.fit(bank_main_df)
imputed_df = imp.transform(bank_main_df)
imputed_df = pd.DataFrame(imputed_df, columns=bank_main_df.columns)
```

In [9]:
```
# pd.isnull(imputed_df).sum()
```

R-Forest Feature Importance

In [10]:
```
#scaler = StandardScaler()
#X_scaled = scaler.fit_transform(imputed_df)

#X = imputed_df.drop(columns='deposit')
#y = imputed_df['deposit']

#X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, shuffle=True, random_state=42)
```

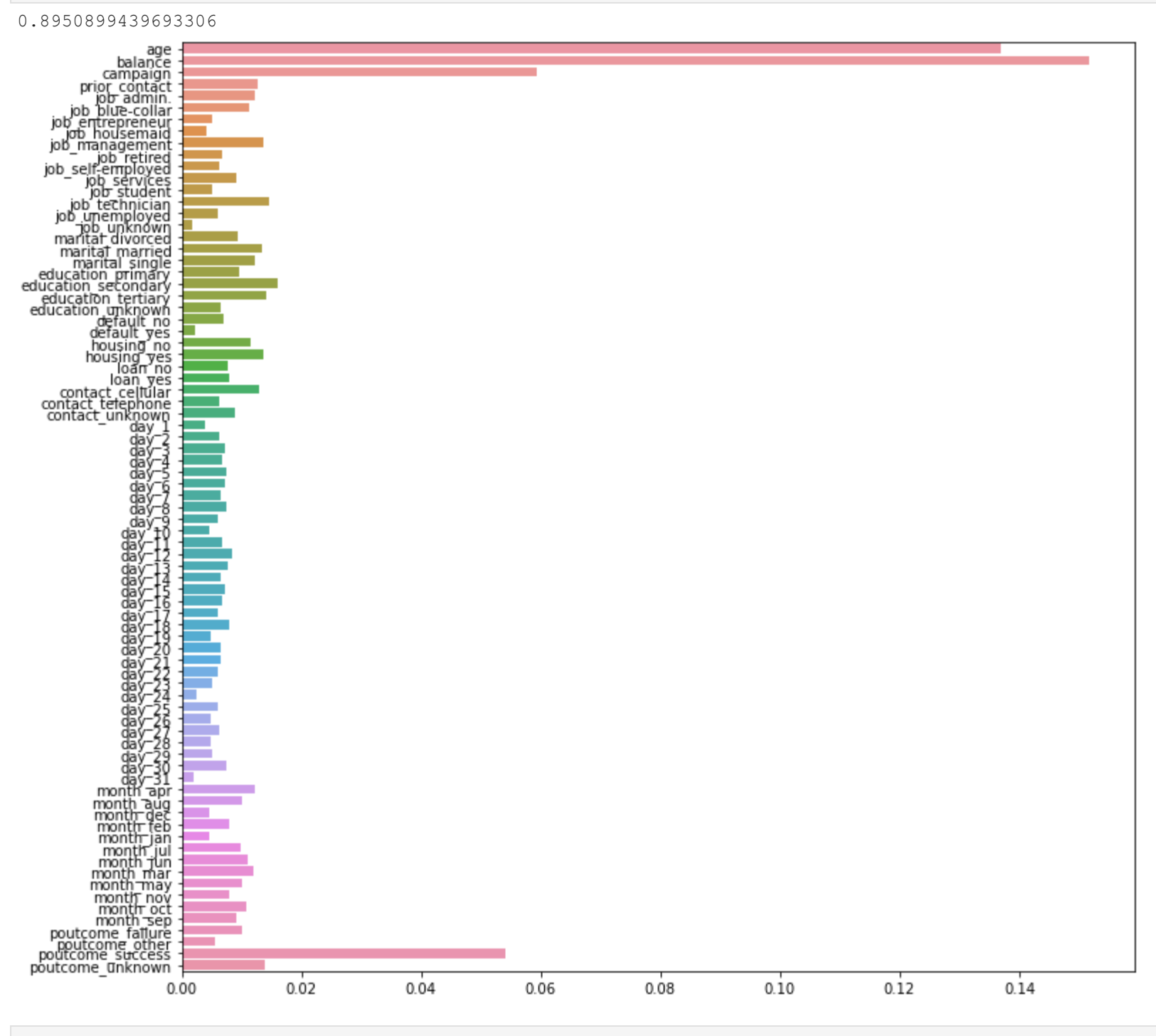
Loop through all classifiers

MinMaxScaler

In [11]:
```
# StandardScaler, MinMaxScaler, RobustScaler
#scaler = RobustScaler()
#X_scaled = scaler.fit_transform(imputed_df)

X = imputed_df.drop(columns='deposit')
y = imputed_df['deposit']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, shuffle=True)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [12]:
```
clf = RandomForestClassifier()
# fit the classifier to the training set
clf.fit(X_train, y_train)
# predict on the test set
y_pred = clf.predict(X_test)
acc_ = accuracy_score(y_test, y_pred)
print(acc_)

#for feature in zip(X.columns, clf.feature_importances_.round(3)):
#    print(feature)

plt.figure(figsize=(12,12))
sns.barplot(y=X.columns, x=clf.feature_importances_,orient = 'h');
#plt.xticks(rotation=90);
# visualize feature importance
#with plt.style.context('ggplot', ):
#    sns.set(font_scale=1)
#    plt.figure(num=None, figsize=(14,10), dpi=80, facecolor='w')
#    plt.title("Random Forest Feature Importance Weighting", size=18)
#    plt.ylabel('Features', size = 14)
#    plt.xlabel('Feature Importance', size=14)
#    feat_importances = pd.Series(clf.feature_importances_, index= X.columns)
```

0.8950899439693306



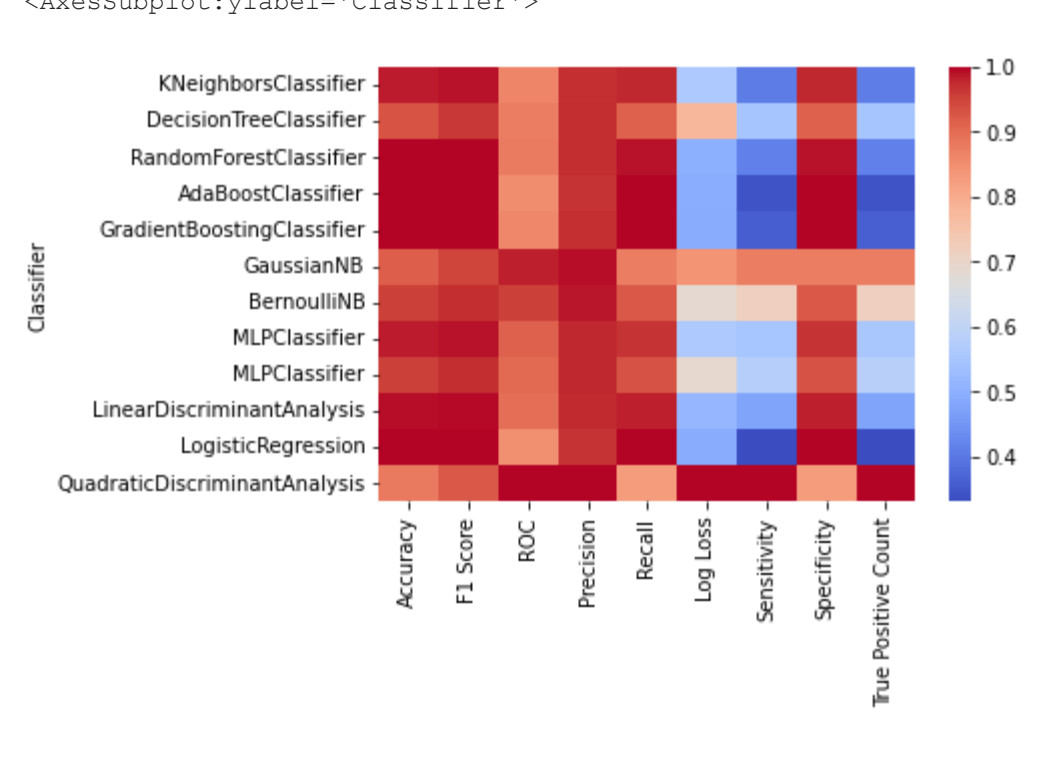
In [13]:
```
classifiers = [
#    MultinomialNB(), # doesn't work
    KNeighborsClassifier(3), # works
#    SVC(kernel="rbf", C=0.001, probability=True), # took a long time... need to refresh memory
#    SVC(kernel='linear'), # took a long time... need to refresh memory
#    NuSVC(probability=True, nu=0.1), # took a long time... need to refresh memory
    DecisionTreeClassifier(), # works
    RandomForestClassifier(), # works
    AdaBoostClassifier(), # works
    GradientBoostingClassifier(), # works
    GaussianNB(), # works
    BernoulliNB(), # works
    MLPClassifier(), # works
    MLPClassifier(hidden_layer_sizes=[100, 100]), # works
    LinearDiscriminantAnalysis(), # works
    LogisticRegression(), # works
    QuadraticDiscriminantAnalysis(), # works
]

log_cols=["Classifier", "Accuracy", "F1 Score", "ROC", "Precision", "Recall", "Log Loss", "Sensitivity","Specif
log = pd.DataFrame(columns=log_cols)

for clf in classifiers:
    clf.fit(X_train, y_train)
#    y_pred = clf.predict(X)
    name = clf.__class__.__name__

    print("="*60)
    print(name)
    print('****Results****')

    train_predictions = clf.predict(X_test)
    acc_ = accuracy_score(y_test, train_predictions)
    acc = acc_.round(3)
    print("Accuracy: {:.4%}".format(acc))

#    coef_scores = X_scaled
#    coef_scores = clf.coef_
#    print(coef_scores)

    fbeta_ = fbeta_score(y_test, train_predictions, beta=1)
    fbeta = fbeta_.round(3)
    print("F1 Score: {}".format(fbeta))

    roc_ = roc_auc_score(y_test, train_predictions)
    roc = roc_.round(3)
    print("AUC (ROC) Score: {}".format(roc))

    precision_ = precision_score(y_test, train_predictions, average='binary')
    precision = precision_.round(3)
    print("Precision Score: {}".format(precision))

    recall_ = recall_score(y_test, train_predictions)
    recall = recall_.round(3)
    print("Recall Score: {}".format(recall))

    TP = confusion_matrix(y_test,train_predictions).ravel()[0]
    FN = confusion_matrix(y_test,train_predictions).ravel()[1]
    sensitivity_ = TP / (TP + FN)
    sensitivity = sensitivity_.round(3)
    print("Sensitivity Score: {}".format(sensitivity))

    TN = confusion_matrix(y_test,train_predictions).ravel()[3]
    FP = confusion_matrix(y_test,train_predictions).ravel()[2]
    specificity_ = TN / (TN + FP)
    specificity = specificity_.round(3)
    print("Specificity Score: {}".format(specificity))

    true_positives = confusion_matrix(y_test,train_predictions).ravel()[0]

    print(f"\nClassic train score: {np.round(clf.score(X_train, y_train),4)}")
    print(f"Classic test score: {np.round(clf.score(X_test, y_test),4)}")
    confusion_matrix_ = confusion_matrix(y_test, train_predictions)
    print(f"Confusion matrix: \n{confusion_matrix_}\n")

#    train_predictions = clf.predict_proba(X_test)
    ll_ = log_loss(y_test, train_predictions)
    ll = ll_.round(3)
#    print("Log Loss: {}".format(ll))

    log_entry = pd.DataFrame([[name, acc*100, fbeta, roc, precision, recall, ll, sensitivity,specificity, true_
    log = pd.concat([log,log_entry])

# print("="*30)
# type(coef_scores)
# print(index)
# print(log_entry)
# type(log)
```

```
============================================================
KNeighborsClassifier
****Results****
Accuracy: 88.2000%
F1 Score: 0.936
AUC (ROC) Score: 0.596
Precision Score: 0.908
Recall Score: 0.965
Sensitivity Score: 0.227
Specificity Score: 0.965

Classic train score: 0.9163
Classic test score: 0.8821
Confusion matrix:
[[  346  1175]
 [  424 11619]]

============================================================
DecisionTreeClassifier
****Results****
Accuracy: 83.5000%
F1 Score: 0.907
AUC (ROC) Score: 0.604
Precision Score: 0.911
Recall Score: 0.902
Sensitivity Score: 0.306
Specificity Score: 0.902

Classic train score: 1.0
Classic test score: 0.8354
Confusion matrix:
[[  465  1056]
 [ 1177 10866]]

============================================================
RandomForestClassifier
****Results****
Accuracy: 89.5000%
F1 Score: 0.943
AUC (ROC) Score: 0.605
Precision Score: 0.91
Recall Score: 0.978
Sensitivity Score: 0.231
Specificity Score: 0.978

Classic train score: 1.0
Classic test score: 0.8945
Confusion matrix:
[[  351  1170]
 [  261 11782]]

============================================================
AdaBoostClassifier
****Results****
Accuracy: 89.6000%
F1 Score: 0.944
AUC (ROC) Score: 0.588
Precision Score: 0.906
Recall Score: 0.985
Sensitivity Score: 0.191
Specificity Score: 0.985

Classic train score: 0.891
Classic test score: 0.8958
Confusion matrix:
[[  291  1230]
 [  184 11859]]

============================================================
GradientBoostingClassifier
****Results****
Accuracy: 89.6000%
F1 Score: 0.944
AUC (ROC) Score: 0.593
Precision Score: 0.907
Recall Score: 0.984
Sensitivity Score: 0.201
Specificity Score: 0.984

Classic train score: 0.8963
Classic test score: 0.8964
Confusion matrix:
[[  306  1215]
 [  190 11853]]

============================================================
GaussianNB
****Results****
Accuracy: 82.2000%
F1 Score: 0.896
AUC (ROC) Score: 0.677
Precision Score: 0.931
Recall Score: 0.864
Sensitivity Score: 0.49
Specificity Score: 0.864

Classic train score: 0.822
Classic test score: 0.8225
Confusion matrix:
[[  745   776]
 [ 1632 10411]]

============================================================
BernoulliNB
****Results****
Accuracy: 85.5000%
F1 Score: 0.918
AUC (ROC) Score: 0.657
Precision Score: 0.924
Recall Score: 0.912
Sensitivity Score: 0.402
Specificity Score: 0.912

Classic train score: 0.8561
Classic test score: 0.8551
Confusion matrix:
[[  611   910]
 [ 1056 10987]]

C:\Users\reagins\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-pac
kages\Python310\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning: Stochas
tic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
============================================================
MLPClassifier
****Results****
Accuracy: 88.2000%
F1 Score: 0.935
AUC (ROC) Score: 0.631
Precision Score: 0.916
Recall Score: 0.954
Sensitivity Score: 0.308
Specificity Score: 0.954

Classic train score: 0.9373
Classic test score: 0.8818
Confusion matrix:
[[  469  1052]
 [  551 11492]]

C:\Users\reagins\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-pac
kages\Python310\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning: Stochas
tic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
============================================================
MLPClassifier
****Results****
Accuracy: 85.4000%
F1 Score: 0.918
AUC (ROC) Score: 0.623
Precision Score: 0.915
Recall Score: 0.921
Sensitivity Score: 0.324
Specificity Score: 0.921

Classic train score: 0.9699
Classic test score: 0.8544
Confusion matrix:
[[  493  1028]
 [  947 11096]]

============================================================
LinearDiscriminantAnalysis
****Results****
Accuracy: 89.1000%
F1 Score: 0.941
AUC (ROC) Score: 0.618
Precision Score: 0.913
Recall Score: 0.97
Sensitivity Score: 0.266
Specificity Score: 0.97

Classic train score: 0.8988
Classic test score: 0.8913
Confusion matrix:
[[  404  1117]
 [  357 11686]]

C:\Users\reagins\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-pac
kages\Python310\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to conver
ge (status=1):
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(
C:\Users\reagins\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-pac
kages\Python310\site-packages\sklearn\discriminant_analysis.py:878: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
============================================================
LogisticRegression
****Results****
Accuracy: 89.6000%
F1 Score: 0.944
AUC (ROC) Score: 0.585
Precision Score: 0.905
Recall Score: 0.986
Sensitivity Score: 0.185
Specificity Score: 0.986

Classic train score: 0.8919
Classic test score: 0.896
Confusion matrix:
[[  282  1239]
 [  171 11872]]

============================================================
QuadraticDiscriminantAnalysis
****Results****
Accuracy: 78.9000%
F1 Score: 0.873
AUC (ROC) Score: 0.687
Precision Score: 0.936
Recall Score: 0.818
Sensitivity Score: 0.556
Specificity Score: 0.818

Classic train score: 0.792
Classic test score: 0.7889
Confusion matrix:
[[  851   670]
 [ 2193  9850]]
```

In [14]:
```
# Save DF as PNG
def render_mpl_table(imputed_df, col_width=6.0, row_height=0.625, font_size=10,
                     header_color='#40466e', row_colors=['#f1f1f2', 'w'], edge_color='w',
                     bbox=[0, 0, 1, 1], header_columns=0,
                     ax=None, **kwargs):
    if ax is None:
        size = (np.array(imputed_df.shape[::-1]) + np.array([0, 1])) * np.array([col_width, row_height])
        fig, ax = plt.subplots(figsize=size)
        ax.axis('off')
    mpl_table = ax.table(cellText=imputed_df.values, bbox=bbox, colLabels=imputed_df.columns, **kwargs)
    mpl_table.auto_set_font_size(False)
    mpl_table.set_fontsize(font_size)

    for k, cell in mpl_table._cells.items():
        cell.set_edgecolor(edge_color)
        if k[0] == 0 or k[1] < header_columns:
            cell.set_text_props(weight='bold', color='w')
            cell.set_facecolor(header_color)
        else:
            cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
    return ax.get_figure(), ax

fig,ax = render_mpl_table(log, header_columns=0, col_width=3.0)
fig.savefig("images/table_mpl_minmax1.png")
```

In [15]:
```
log1 = log.set_index('Classifier')

norml_df = log1 / log1.max(0)

sns.heatmap(norml_df.astype('float'),cmap='coolwarm')
```

Out[15]: <AxesSubplot:ylabel='Classifier'>



StandardScaler

In [16]:
```
# StandardScaler, MinMaxScaler, RobustScaler
#scaler = RobustScaler()
#X_scaled = scaler.fit_transform(imputed_df)

X = imputed_df.drop(columns='deposit')
y = imputed_df['deposit']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, shuffle=True)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [17]:
```
classifiers = [
#    MultinomialNB(), # doesn't work
    KNeighborsClassifier(3), # works
#    SVC(kernel="rbf", C=0.001, probability=True), # took a long time... need to refresh memory
#    SVC(kernel='linear'), # took a long time... need to refresh memory
#    NuSVC(probability=True, nu=0.1), # took a long time... need to refresh memory
    DecisionTreeClassifier(), # works
    RandomForestClassifier(), # works
    AdaBoostClassifier(), # works
    GradientBoostingClassifier(), # works
    GaussianNB(), # works
    BernoulliNB(), # works
    MLPClassifier(), # works
    MLPClassifier(hidden_layer_sizes=[100, 100]), # works
    LinearDiscriminantAnalysis(), # works
    LogisticRegression(), # works
    QuadraticDiscriminantAnalysis(), # works
]

log_cols=["Classifier", "Accuracy", "F1 Score", "ROC", "Precision", "Recall", "Log Loss", "Sensitivity", "Speci
log = pd.DataFrame(columns=log_cols)

for clf in classifiers:
    clf.fit(X_train, y_train)
#    y_pred = clf.predict(X)
    name = clf.__class__.__name__

    print("="*60)
    print(name)
    print('****Results****')

    train_predictions = clf.predict(X_test)
    acc_ = accuracy_score(y_test, train_predictions)
    acc = acc_.round(3)
    print("Accuracy: {:.4%}".format(acc))

#    coef_scores = X_scaled
#    coef_scores = clf.coef_
#    print(coef_scores)

    fbeta_ = fbeta_score(y_test, train_predictions, beta=1)
    fbeta = fbeta_.round(3)
    print("F1 Score: {}".format(fbeta))

    roc_ = roc_auc_score(y_test, train_predictions)
    roc = roc_.round(3)
    print("AUC (ROC) Score: {}".format(roc))

    precision_ = precision_score(y_test, train_predictions, average='binary')
    precision = precision_.round(3)
    print("Precision Score: {}".format(precision))

    recall_ = recall_score(y_test, train_predictions)
    recall = recall_.round(3)
    print("Recall Score: {}".format(recall))

    TP = confusion_matrix(y_test,train_predictions).ravel()[0]
    FN = confusion_matrix(y_test,train_predictions).ravel()[1]
    sensitivity_ = TP / (TP + FN)
    sensitivity = sensitivity_.round(3)
    print("Sensitivity Score: {}".format(sensitivity))

    TN = confusion_matrix(y_test,train_predictions).ravel()[3]
    FP = confusion_matrix(y_test,train_predictions).ravel()[2]
    specificity_ = TN / (TN + FP)
    specificity = specificity_.round(3)
    print("Specificity Score: {}".format(specificity))

    true_positives = confusion_matrix(y_test,train_predictions).ravel()[0]

    print(f"\nClassic train score: {np.round(clf.score(X_train, y_train),4)}")
    print(f"Classic test score: {np.round(clf.score(X_test, y_test),4)}")
    confusion_matrix_ = confusion_matrix(y_test, train_predictions)
    print(f"Confusion matrix: \n{confusion_matrix_}\n")

#    train_predictions = clf.predict_proba(X_test)
    ll_ = log_loss(y_test, train_predictions)
    ll = ll_.round(3)
#    print("Log Loss: {}".format(ll))

    log_entry = pd.DataFrame([[name, acc*100, fbeta, roc, precision, recall, ll, sensitivity, specificity, true
    log = pd.concat([log,log_entry])

# print("="*30)
# type(coef_scores)
# print(index)
# print(log_entry)
# type(log)
```