Bank Marketing Campaign

About Our DataSet

- The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit

- In this dataset we explore which parameters help the bank to identify possible customer who will make a term deposit

Bank client data:

- `age` : (numeric)
- `job` : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
- `marital` : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
- `education` : (categorical: primary, secondary, tertiary and unknown)
- `default` : has credit in default? (categorical: 'no','yes','unknown')
- `housing` : has housing loan? (categorical: 'no','yes','unknown')- loan: has personal loan? (categorical: 'no','yes','unknown')
- `balance` : Balance of the individual.

Related with the last contact of the current campaign:

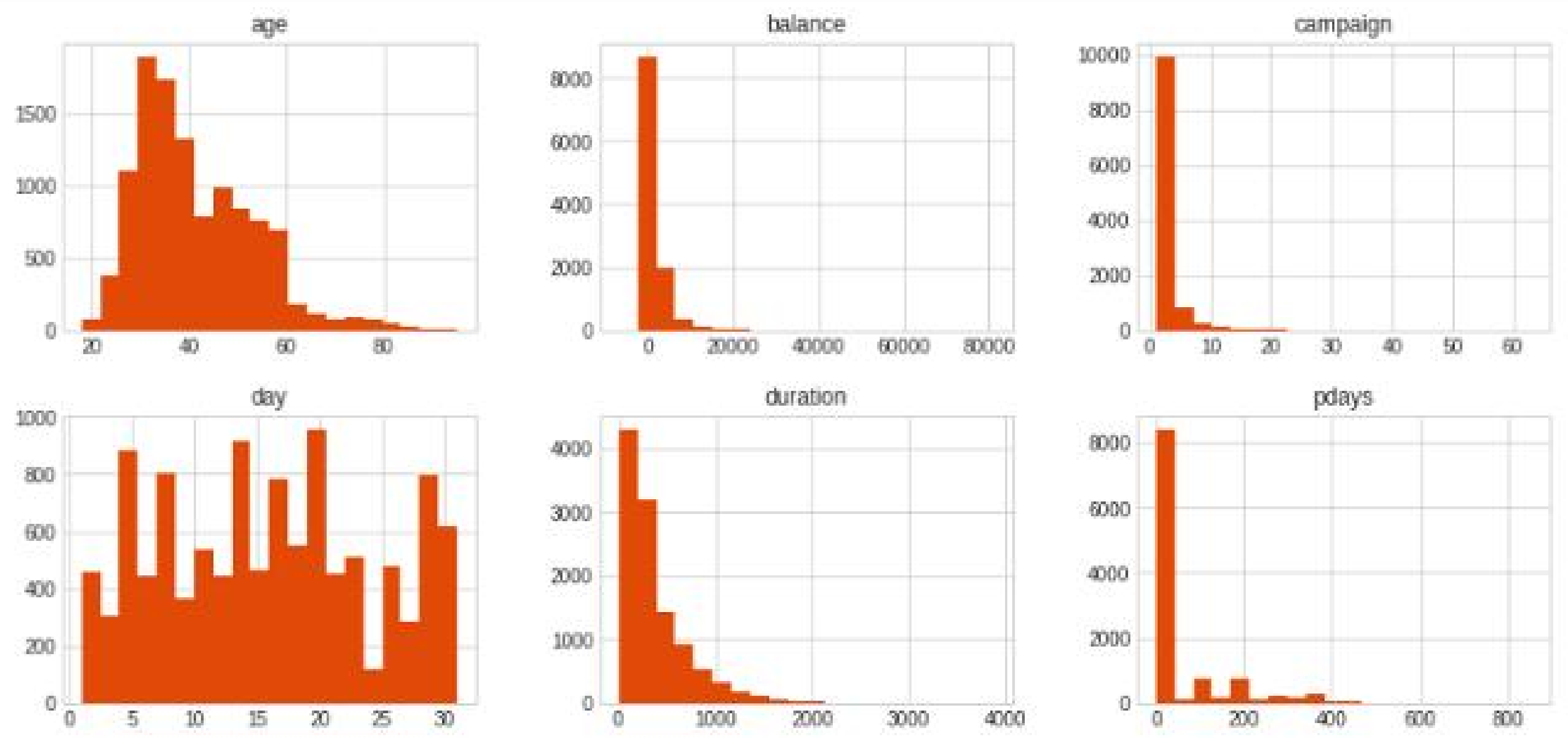
- `contact` : contact communication type (categorical: 'cellular','telephone')
- `month` : last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- `day` : last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
- `duration` : last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

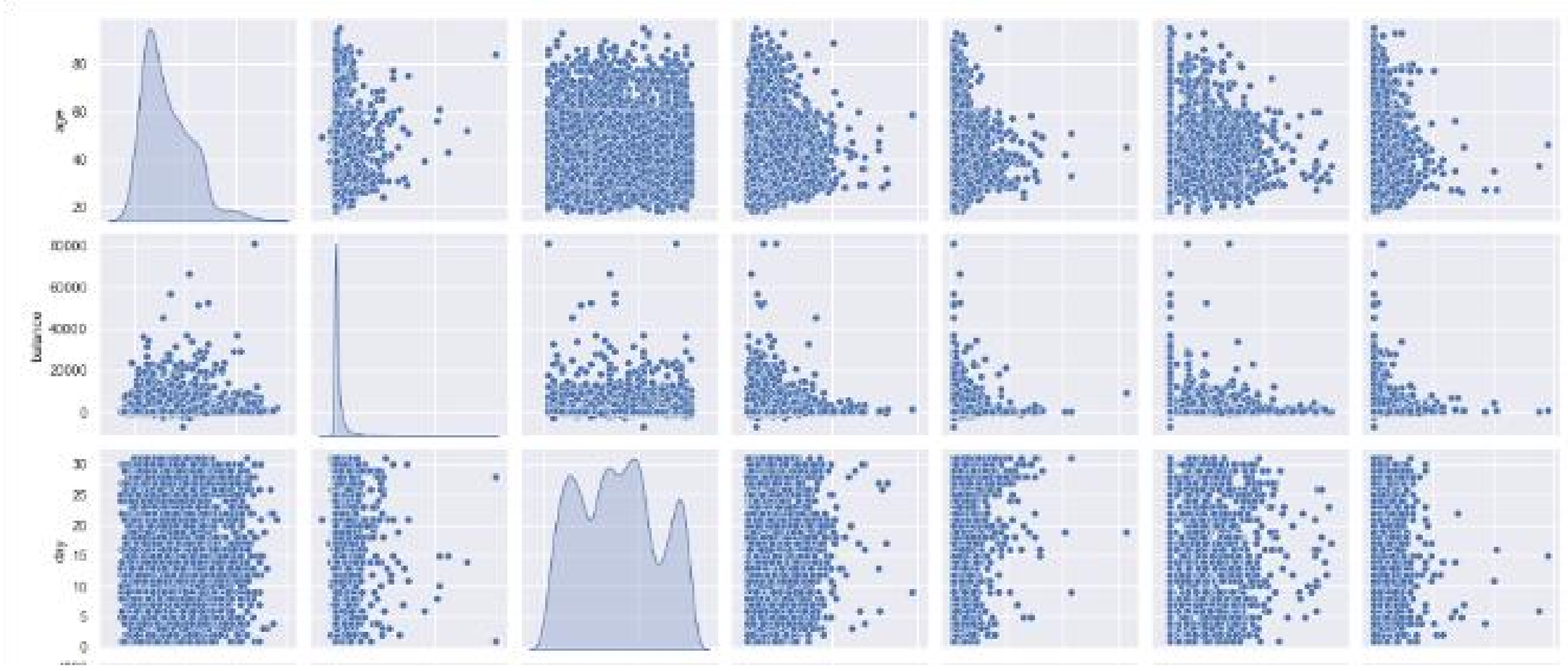
other attributes:

- `campaign` : number of contacts performed during this campaign and for this client (numeric, includes last contact)
- `pdays` : number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- `previous` : number of contacts performed before this campaign and for this client (numeric)
- `poutcome` : outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

Exploratory Data Analysis

- Helps in understanding the composition of the features and the relationship among them
- Draws insights from the data
- Histograms and distribution plots
- Pair plots and boxplots
- Feature engineering and model building

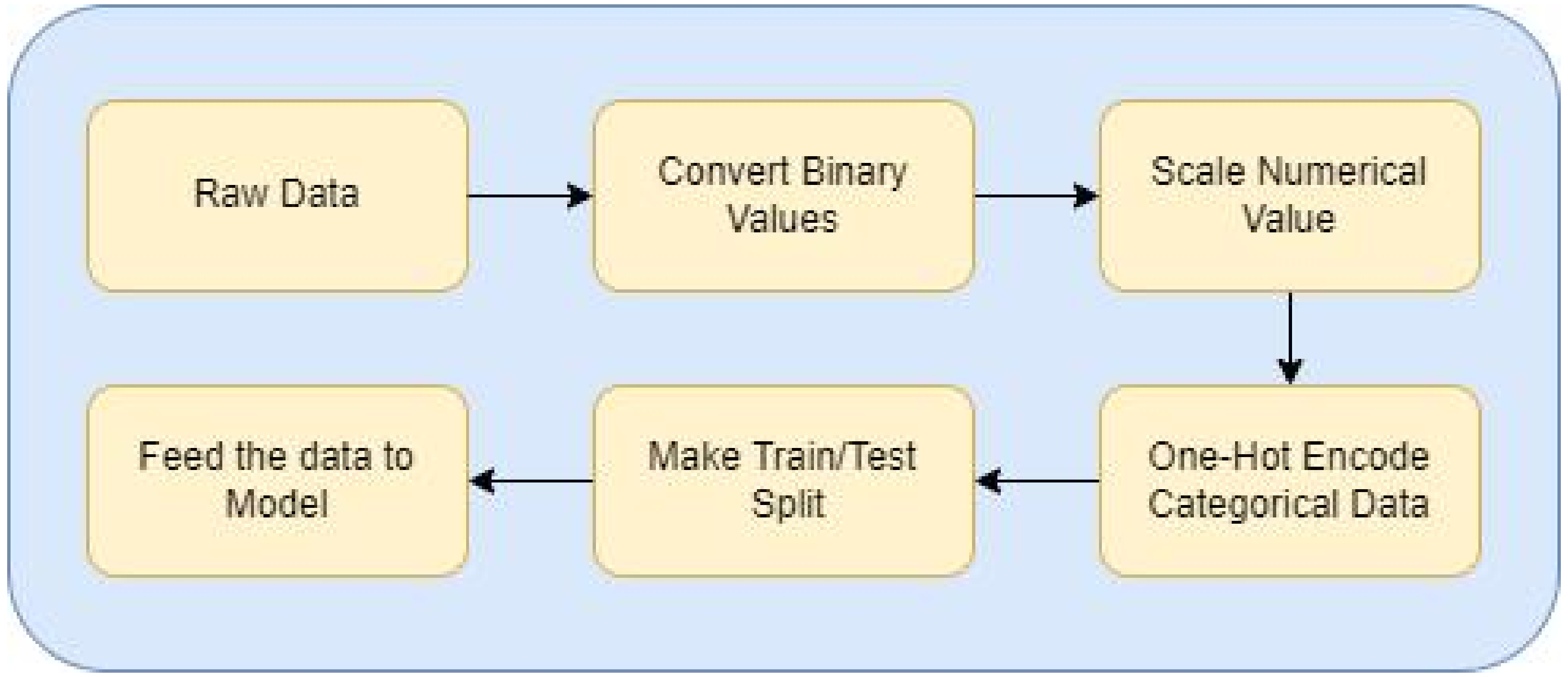




Data Pre-Processing

- Dataset has 11,162 samples and almost equally divided in two classes.
- Dataset contains 7 numerical values and 10 categorical values.
- We are using binary labeling for yes/no columns, MinMaxScaler( ) for normalizing numerical values and One-Hot Encoding for categorical data.
- After that we are using train_test_split with 30% being test data to get Train and Test datasets.

```
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        11162 non-null  int64
 1   job        11162 non-null  object
 2   marital    11162 non-null  object
 3   education  11162 non-null  object
 4   default    11162 non-null  object
 5   balance    11162 non-null  int64
 6   housing    11162 non-null  object
 7   loan       11162 non-null  object
 8   contact    11162 non-null  object
 9   day        11162 non-null  int64
 10  month      11162 non-null  object
 11  duration   11162 non-null  int64
 12  campaign   11162 non-null  int64
 13  pdays      11162 non-null  int64
 14  previous   11162 non-null  int64
 15  poutcome   11162 non-null  object
 16  deposit    11162 non-null  object
```



Logistic Regression:

Accuracy of Logistic Regression for our dataset:82.02%

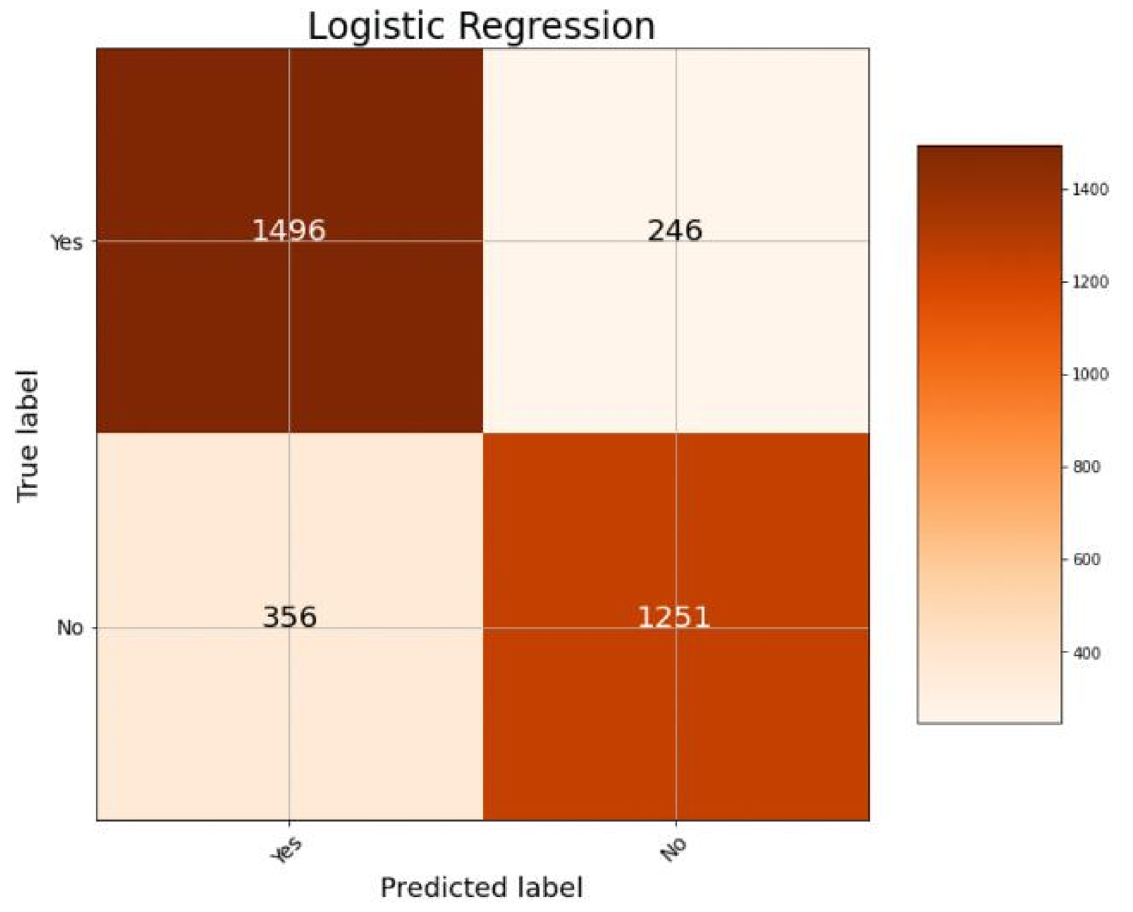
Confusion Matrix –
Accuracy = TP+TN/(TP+TN+FP+FN)

Re-evaluation after dropping of columns/features(randomly):

- Marital - 81.57%
- Age - 82.05%
- Education - 81.57%
- Duration - 71.30%
- Balance - 81.99%

In [12]:
```
#Create a Logistic Regression Object & perform Logistic Regression
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
```
Out[12]:
```
LogisticRegression(max_iter=1000)
```

In [29]:
```
cf = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cf, title="Logistic Regression")
```



In [16]:
```
#Calculating Accuracy of Model using Accuracy_score method
print("Accuracy of the model using Logistic Regression: ", accuracy_score(y_test, y_pred)*100,'%')
```
```
Accuracy of the model using Logistic Regression:  82.0244849208719 %
```

GridSearchCV for Logistic Regression:

GridSearchCV performs an exhaustive search over specified parameter values
(solvers – 'newton-cg,' 'lbfgs', 'liblinear')
(penalty – 'l2')
(c_values – 0.1, 1, 10, 100, 1000)
for an estimator
(Logistic Regression algorithm in this case)

Best values for our dataset –
Mean – 0.825
C – 1000
Penalty – l2
Solver – lbfgs

Using GridSearchCV to tune the hyperparameters

```python
#define parameters
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [0.1, 1, 10, 100, 1000]

#define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
logreg_gscv = GridSearchCV(estimator=log_reg, param_grid=grid, n_jobs=-1, cv=cv, refit=True, verbose=True,
                           scoring='accuracy', error_score=0)
grid_result = logreg_gscv.fit(X, y)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Fitting 30 folds for each of 15 candidates, totalling 450 fits
Best: 0.825479 using {'C': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.804574 (0.010323) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.804544 (0.010321) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.804544 (0.010321) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.822731 (0.011294) with: {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.822731 (0.011318) with: {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.822761 (0.011268) with: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
0.825360 (0.010139) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.825360 (0.010139) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.825360 (0.010139) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.825390 (0.010434) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.825479 (0.010440) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.825390 (0.010434) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.825449 (0.010372) with: {'C': 1000, 'penalty': 'l2', 'solver': 'newton-cg'}
0.825479 (0.010334) with: {'C': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.825419 (0.010399) with: {'C': 1000, 'penalty': 'l2', 'solver': 'liblinear'}
```

Stochastic Gradient Descent Classifier :

Accuracy of SGDClassifier
for our dataset:
81.55 %

Confusion Matrix –
Accuracy = TP+TN/(TP+TN+FP+FN)

Re-evaluation after dropping of columns/features(randomly):

- Job - 81.84%
- Month - 80.20%
- Campaign – 81.93%
- Education – 81.46%
- Age – 81.25%

Stochastic Gradient Descent

```
In [11]: from sklearn.linear_model import SGDClassifier

In [12]: SGD_model = SGDClassifier(max_iter=1000, tol=0.001, random_state=42)

In [13]: SGD_model.fit(X_train,y_train)

Out[13]: SGDClassifier(random_state=42)

In [14]: SGD_predictions = SGD_model.predict(X_test)

In [15]: from sklearn.metrics import confusion_matrix
         print(confusion_matrix(y_test,SGD_predictions))

[[1504  238]
 [ 380 1227]]

In [16]: #Calculating the accuracy of SVC model
         print(f"Accuracy of SGD model is: {metrics.accuracy_score(y_test, SGD_predictions)*100}%")

Accuracy of SGD model is: 81.54673036727381%
```

GridSearchCV for SGD:

GridSearchCV performs an exhaustive search over specified parameter values
('loss':['hinge','log'])
('penalty':['l2','l1','elasticnet'])
('alpha':[0.001,0.0001,0.00001])
('l1_ratio':[0.05,0.06,0.07,0.08,0.09,0.1, 0.12,0.13,0.14,0.15,0.2])
for an estimator
(SGD algorithm in this case)

Best values for our dataset –
l1_ratio=0.08,
penalty=elasticnet
alpha = 0.0001
loss = hinge

Using GridSearchCV to tune the hyperparameters

In [17]:
```
sgdc = SGDClassifier()

param_grid = {'loss':['hinge','log'],
    'penalty':['l2','l1','elasticnet'] ,
    'alpha':[0.001,0.0001,0.00001],
    'l1_ratio':[0.05,0.06,0.07,0.08,0.09,0.1,0.12,0.13,0.14,0.15,0.2]}
```

In [18]:
```
grid_sgdc = GridSearchCV(sgdc, param_grid, cv=5, verbose=1, n_jobs=-1)
grid_sgdc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 198 candidates, totalling 990 fits
```

Out[18]:
```
GridSearchCV(cv=5, estimator=SGDClassifier(), n_jobs=-1,
             param_grid={'alpha': [0.001, 0.0001, 1e-05],
                         'l1_ratio': [0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.12,
                                      0.13, 0.14, 0.15, 0.2],
                         'loss': ['hinge', 'log'],
                         'penalty': ['l2', 'l1', 'elasticnet']},
             verbose=1)
```

In [19]:
```
grid_sgdc.best_params_
```

Out[19]:
```
{'alpha': 0.0001, 'l1_ratio': 0.08, 'loss': 'hinge', 'penalty': 'elasticnet'}
```

In [20]:
```
#The best estimator
grid_sgdc.best_estimator_
```

Out[20]:
```
SGDClassifier(l1_ratio=0.08, penalty='elasticnet')
```

In [21]:
```
grid_sgdc_predictions = grid_sgdc.predict(X_test)
grid_sgdc_cf = confusion_matrix(y_test, grid_sgdc_predictions)
grid_sgdc_cf
```

Out[21]:
```
array([[1440,  302],
       [ 302, 1305]], dtype=int64)
```

In [22]:
```
#Accuracy of the optimised SVC
print(f"Accuracy: {accuracy_score(y_test, grid_sgdc_predictions)*100}%")
```

```
Accuracy: 81.96476560167214%
```

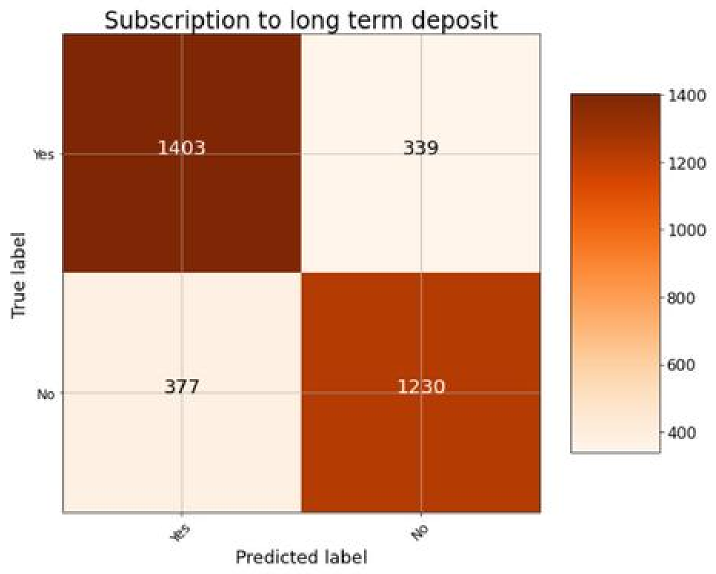
In [23]:
```
#Comparing the accuracies
print(f"SGD Accuracy: {accuracy_score(y_test, SGD_predictions)*100}%")
print(f"GridSearchCV accuracy: {accuracy_score(y_test, grid_sgdc_predictions)*100}%")
```

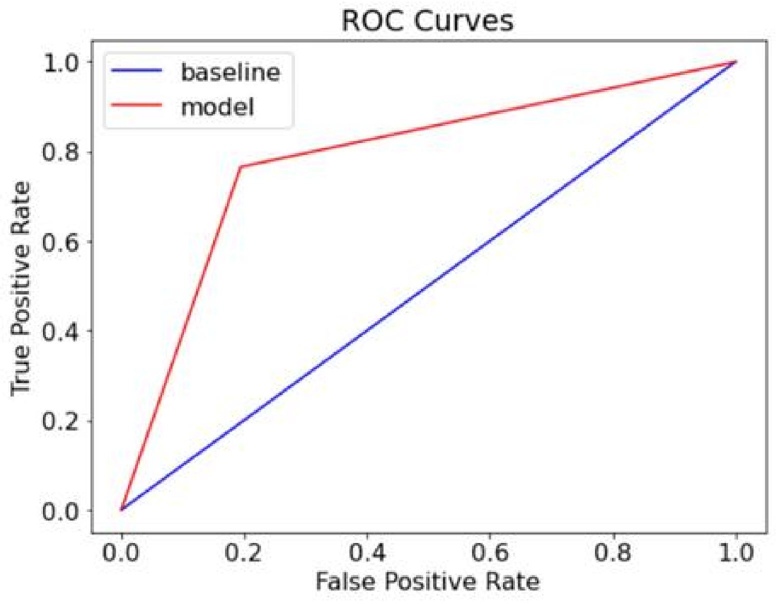
```
SGD Accuracy: 81.54673036727381%
GridSearchCV accuracy: 81.96476560167214%
```

Decision Tree Classifier

```
#import DecisionTreeClassifier from sklearn.tree
from sklearn.tree import DecisionTreeClassifier
```

```
#creating tree object and fitting it on training data
RSEED = 5
tree = DecisionTreeClassifier(random_state=RSEED)
tree.fit(X_train, y_train)
```

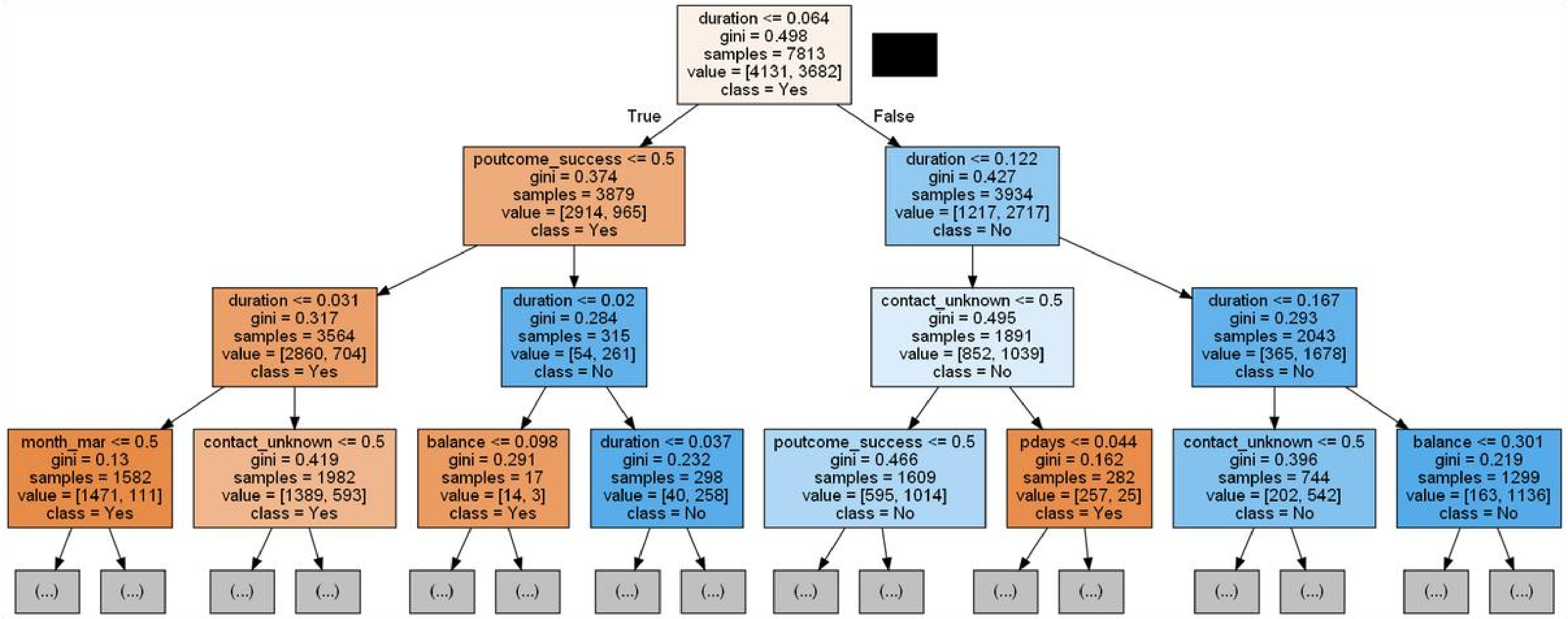





```
fi = pd.DataFrame({'feature': column_list,
                   'importance': tree.feature_importances_}).sort_values('importance', ascending = False)
fi.head()
```

	feature	importance
44	duration	0.348314
36	poutcome_success	0.079186
43	day	0.073553
40	balance	0.065351
21	contact_unknown	0.061706

Decision Tree with max_depth = 3



GridSearchCV for Decision Tree:

GridSearchCV performs an exhaustive search over specified parameter values 'criterion':['gini','entropy'], 'max_depth':[4,5,6,7,8,9,10,11,12,15,20,30, 40,50,70,90,120,150]

Best values for our dataset – 'criterion': 'gini', 'max_depth': 11

Using GridSearchCV

```
In [26]: param_grid = {'criterion':['gini','entropy'],'max_depth':[4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150]}

In [27]: grid_decisiontree = GridSearchCV(DecisionTreeClassifier(), param_grid, cv = 5)
         grid_decisiontree.fit(X_train, y_train)

Out[27]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20, 30,
                                                40, 50, 70, 90, 120, 150]})

In [28]: grid_decisiontree.best_params_

Out[28]: {'criterion': 'gini', 'max_depth': 11}

In [29]: grid_decisiontree.best_estimator_

Out[29]: DecisionTreeClassifier(max_depth=11)

In [30]: grid_decisiontree_predictions = grid_decisiontree.predict(X_test)

In [31]: grid_decisiontree_cf = confusion_matrix(y_test, grid_decisiontree_predictions)
         grid_decisiontree_cf

Out[31]: array([[1430,  312],
                [ 257, 1350]], dtype=int64)

In [32]: #Accuracy of the optimised dt
         print(f"Accuracy: {accuracy_score(y_test, grid_decisiontree_predictions)*100}%")

         Accuracy: 83.00985368766796%
```

Random Forest Algorithm:

Accuracy of Random Forest
Classifier for our dataset:
84.89%

Confusion Matrix –
Accuracy = TP+TN/(TP+TN+FP+FN)

Re-evaluation after dropping of columns/features(randomly):

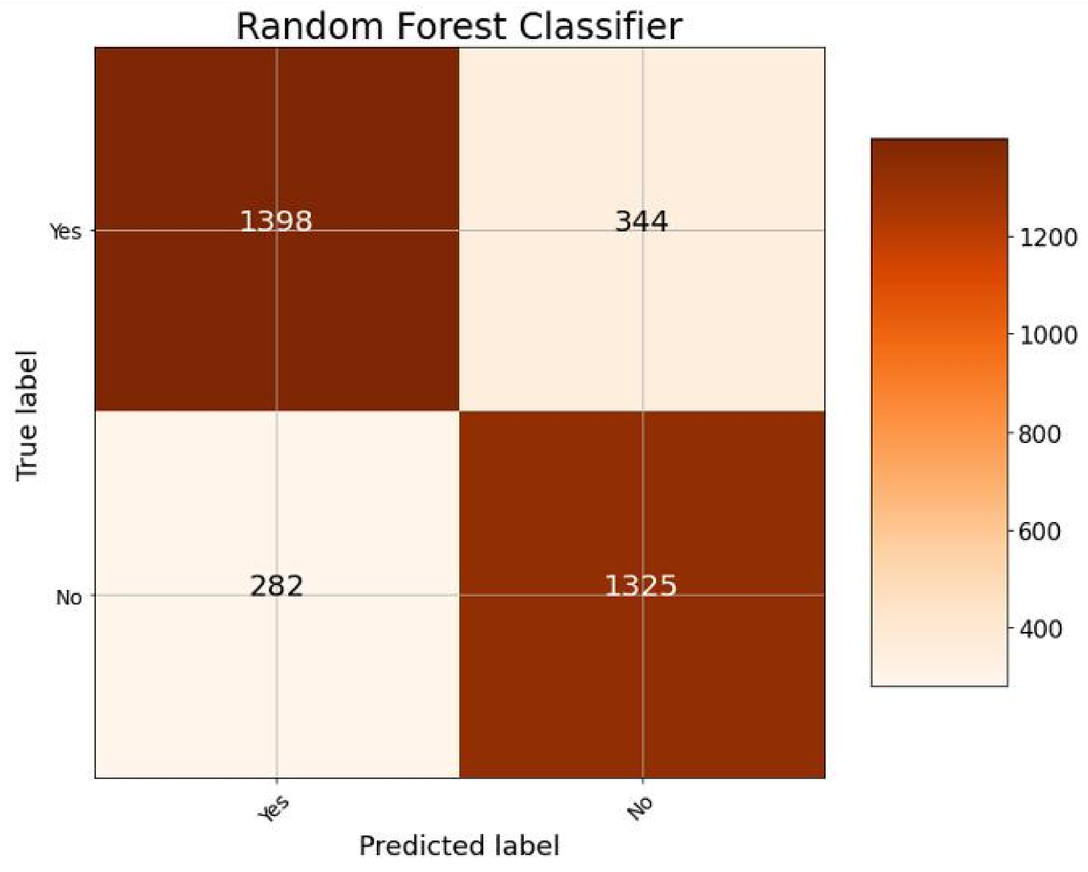
- Marital – 84.92%
- Age – 85.21%
- Education – 84.80%
- Duration – 73.03%
- Balance – 84.29%

```
In [12]: # Creating and Implementing Random Forest Classifier
         rf_model = RandomForestClassifier()
         rf_model.fit(X_train , y_train)

Out[12]: ▾ RandomForestClassifier
         RandomForestClassifier()

In [13]: #Prediction using Random Forest Classifier
         y_pred_rf = rf_model.predict(X_test)

In [14]: cf = confusion_matrix(y_test, y_pred_rf)
         plot_confusion_matrix(cf, title="Random Forest Classifier")
```




```
In [15]: #Calculating Accuracy of Model using Accuracy_score method
         print("Accuracy of the model using Random Forest Algorithm: ", metrics.accuracy_score(y_test, y_pred_rf)*100,'%')

         Accuracy of the model using Random Forest Algorithm:  84.89101224246043 %
```

GridSearchCV for Random Forest Algorithm:

GridSearchCV performs an exhaustive search over specified parameter values
(n_estimators – 5, 10, 20, 50, 100)
(criterion– 'gini', 'entropy', 'log_loss')
for an estimator
(Random Forest algorithm in this case)

Best values for our dataset –
n_estimators – 100
criterion – entropy

Using GridSearchCV to tune the hyperparameters

```python
#define parameters
n_estimators = [5,10,20,50,100]
criterion = ['gini','entropy', 'log_loss']

#define grid search
grid = dict(n_estimators=n_estimators,criterion=criterion)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
rfmodel_gscv = GridSearchCV(estimator=rf_model, param_grid=grid, n_jobs=-1, refit=True, verbose=True, scoring='accuracy',
                            error_score=0)
grid_result_rf = rfmodel_gscv.fit(X_train, y_train)

# summarize results
print("Best: %f using %s" % (grid_result_rf.best_score_, grid_result_rf.best_params_))
means = grid_result_rf.cv_results_['mean_test_score']
stds = grid_result_rf.cv_results_['std_test_score']
params = grid_result_rf.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("Accuracy: %f with: %r" % (mean, param))
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Best: 0.854218 using {'criterion': 'entropy', 'n_estimators': 100}
Accuracy: 0.807757 with: {'criterion': 'gini', 'n_estimators': 5}
Accuracy: 0.820171 with: {'criterion': 'gini', 'n_estimators': 10}
Accuracy: 0.840395 with: {'criterion': 'gini', 'n_estimators': 20}
Accuracy: 0.850762 with: {'criterion': 'gini', 'n_estimators': 50}
Accuracy: 0.854090 with: {'criterion': 'gini', 'n_estimators': 100}
Accuracy: 0.811726 with: {'criterion': 'entropy', 'n_estimators': 5}
Accuracy: 0.824909 with: {'criterion': 'entropy', 'n_estimators': 10}
Accuracy: 0.838732 with: {'criterion': 'entropy', 'n_estimators': 20}
Accuracy: 0.848459 with: {'criterion': 'entropy', 'n_estimators': 50}
Accuracy: 0.854218 with: {'criterion': 'entropy', 'n_estimators': 100}
Accuracy: 0.808652 with: {'criterion': 'log_loss', 'n_estimators': 5}
Accuracy: 0.825292 with: {'criterion': 'log_loss', 'n_estimators': 10}
Accuracy: 0.843211 with: {'criterion': 'log_loss', 'n_estimators': 20}
Accuracy: 0.850891 with: {'criterion': 'log_loss', 'n_estimators': 50}
Accuracy: 0.852555 with: {'criterion': 'log_loss', 'n_estimators': 100}
```

```python
print("Accuracy using Random Forest Algorithm: ", accuracy_score(y_test, y_pred_rf))
print("GridSearchCV Accuracy: ", accuracy_score(y_test, grid_predictions_rf))
```

```
Accuracy using Random Forest Algorithm:  0.8489101224246044
GridSearchCV Accuracy:  0.8512988951925948
```

Support Vector Classifier:

Accuracy of SVC for our dataset:
79.52%

Confusion Matrix –
Accuracy = TP+TN/(TP+TN+FP+FN)

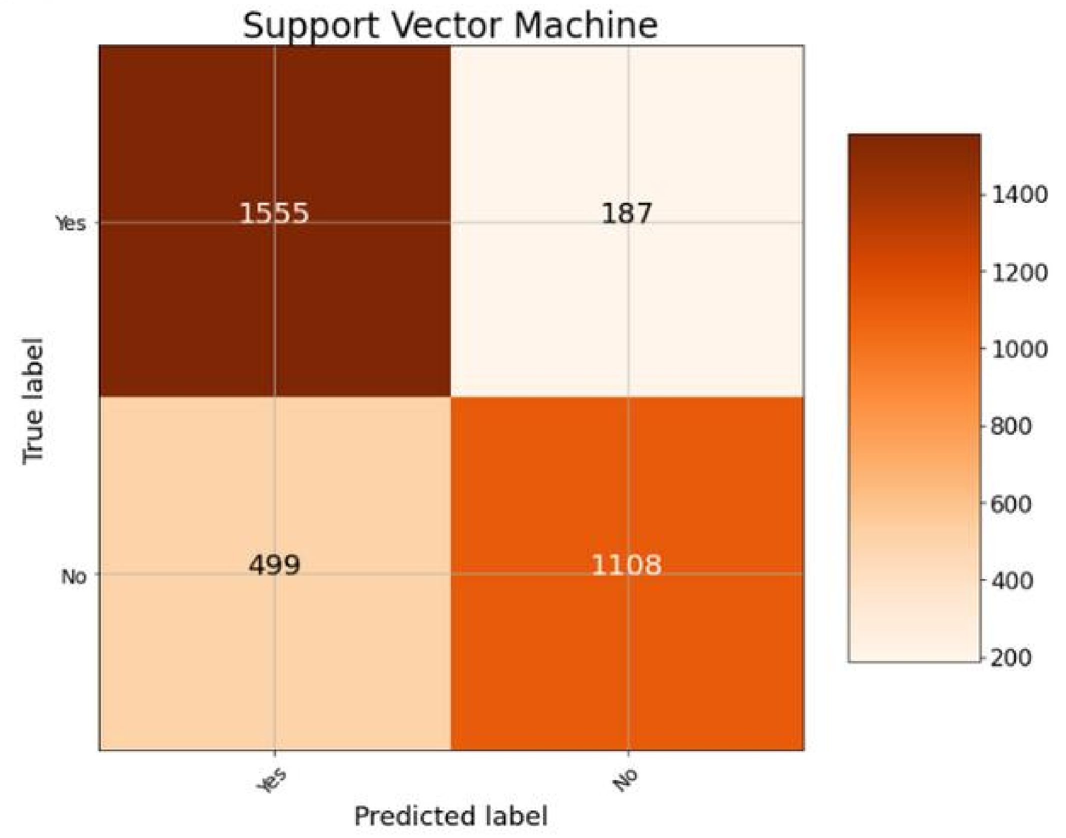
Re-evaluation after dropping of columns/features(randomly):

- Job - 80.20%
- Month - 77.87%
- Campaign – 79.72%
- Education – 79.99%
- Age – 79.52%

In [1]:
```
#importing SVC from scikit-learn
from sklearn.svm import SVC
```

In [2]:
```
#Creating a SVC Classifier object and fitting it with training data
svc = SVC(gamma='auto')
svc.fit(X_train, y_train)
```

In [227]:
```
cf_svc = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cf_svc, title="Support Vector Machine")
```



In [229]:
```
#Calculating the accuracy of SVC model
print(f"Accuracy of SVC model is: {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy of SVC model is: 79.51627351448194%
```

GridSearchCV for SVC:

GridSearchCV performs an exhaustive search over specified parameter values
(Gamma – 0.001, 0.01, 0.1, 1)
(C – 0.1, 1, 10, 100)
(Kernel – 'linear,' 'poly', 'rbf', 'sigmoid')
for an estimator
(Support Vector Classifier algorithm in this case)

Best values for our dataset –
C – 10
Gamma – 0.1
Kernel - rbf

Using GridSearchCV to tune the hyperparameters

```
param_grid = {'gamma':[0.001, 0.01, 0.1, 1], 'C':[0.1, 1, 10, 100], 'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
```

```
grid_svc = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid_svc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
[CV 1/5] END .............C=0.1, gamma=0.001, kernel=linear; total time=   1.4s
[CV 2/5] END .............C=0.1, gamma=0.001, kernel=linear; total time=   1.4s
[CV 3/5] END .............C=0.1, gamma=0.001, kernel=linear; total time=   1.4s
[CV 4/5] END .............C=0.1, gamma=0.001, kernel=linear; total time=   1.4s
[CV 5/5] END .............C=0.1, gamma=0.001, kernel=linear; total time=   1.4s
[CV 1/5] END ...............C=0.1, gamma=0.001, kernel=poly; total time=   2.2s
[CV 2/5] END ...............C=0.1, gamma=0.001, kernel=poly; total time=   2.2s
[CV 3/5] END ...............C=0.1, gamma=0.001, kernel=poly; total time=   2.2s
[CV 4/5] END ...............C=0.1, gamma=0.001, kernel=poly; total time=   2.2s
[CV 5/5] END ...............C=0.1, gamma=0.001, kernel=poly; total time=   2.2s
[CV 1/5] END ................C=0.1, gamma=0.001, kernel=rbf; total time=   2.3s
[CV 2/5] END ................C=0.1, gamma=0.001, kernel=rbf; total time=   2.3s
[CV 3/5] END ................C=0.1, gamma=0.001, kernel=rbf; total time=   2.2s
[CV 4/5] END ................C=0.1, gamma=0.001, kernel=rbf; total time=   2.2s
[CV 5/5] END ................C=0.1, gamma=0.001, kernel=rbf; total time=   2.3s
[CV 1/5] END ............C=0.1, gamma=0.001, kernel=sigmoid; total time=   2.2s
[CV 2/5] END ............C=0.1, gamma=0.001, kernel=sigmoid; total time=   2.2s
[CV 3/5] END ............C=0.1, gamma=0.001, kernel=sigmoid; total time=   2.2s
[CV 4/5] END ............C=0.1, gamma=0.001, kernel=sigmoid; total time=   2.2s
[CV 5/5] END ............C=0.1, gamma=0.001, kernel=sigmoid; total time=   2.2s
[CV 1/5] END ..............C=0.1, gamma=0.01, kernel=linear; total time=   1.4s
[CV 2/5] END ..............C=0.1, gamma=0.01, kernel=linear; total time=   1.4s
[CV 3/5] END ..............C=0.1, gamma=0.01, kernel=linear; total time=   1.4s
[CV 4/5] END ..............C=0.1, gamma=0.01, kernel=linear; total time=   1.4s
[CV 5/5] END ..............C=0.1, gamma=0.01, kernel=linear; total time=   1.4s
```

[19]:
```
#Parameters that gave the best result
grid_svc.best_params_
```

[19]: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}

In [29]:
```
#Comparing the accuracies
print(f"SVC Accuracy: {accuracy_score(y_test, y_pred)*100}%")
print(f"GridSearchCV accuracy: {accuracy_score(y_test, grid_svc_predictions)*100}%")
```

```
SVC Accuracy: 78.59062406688564%
GridSearchCV accuracy: 83.63690653926545%
```

Thank you

Reference -
https://archive.ics.uci.edu/ml/datasets/bank+marketing