Assignment #3 Modeling

Submitted on: March 13, 2021

Submitted to Sasha Ali-Hosein

Summary

Under this assignment, we have created machine learning models that can predict how likely clients will subscribe to a bank term deposit along with efficient use of marketing dollars spend. The best model stood out is **Random Forest classifier** considering performance metrics like accuracy, ROC Curve and AUC. Our model's test performance is ended with **96.3 %** accuracy score with the highest AUC.

From the Random-forest theories used for feature importance and modeling and the comparison of various performance metrics from bunch of other key relevant models, we can conclude that random forest will be the best fit model for our project case scenario and the objective we want to achieve as it builds multiple decision trees according to the features available. When using random forest, we have incorporated new important feature importance based on their significance on output variable y to address top features from the whole banking data set.

After reviewing all the features, their characteristics, correlation with response variable and understanding various patterns, we can suggest our client on targeting potential customers when Euribor: 3month and duration are high. It would be worth noting that unemployed, student carries very less weightage towards subscribing to a term deposition as compared to other categorical features.

We are using four models and have analysed confusion matrix, ROC and AUC results of each models

As shown below, we have compared our combined data frame with output 'y' to conclude how many account holders sign up for the term deposit:

```
X = combinebankinfo.loc[:, combinebankinfo.columns != 'y']
y= combinebankinfo.loc[:, combinebankinfo.columns == 'y']
```

We're using Random forest classifier to calculate feature importance as shown below:

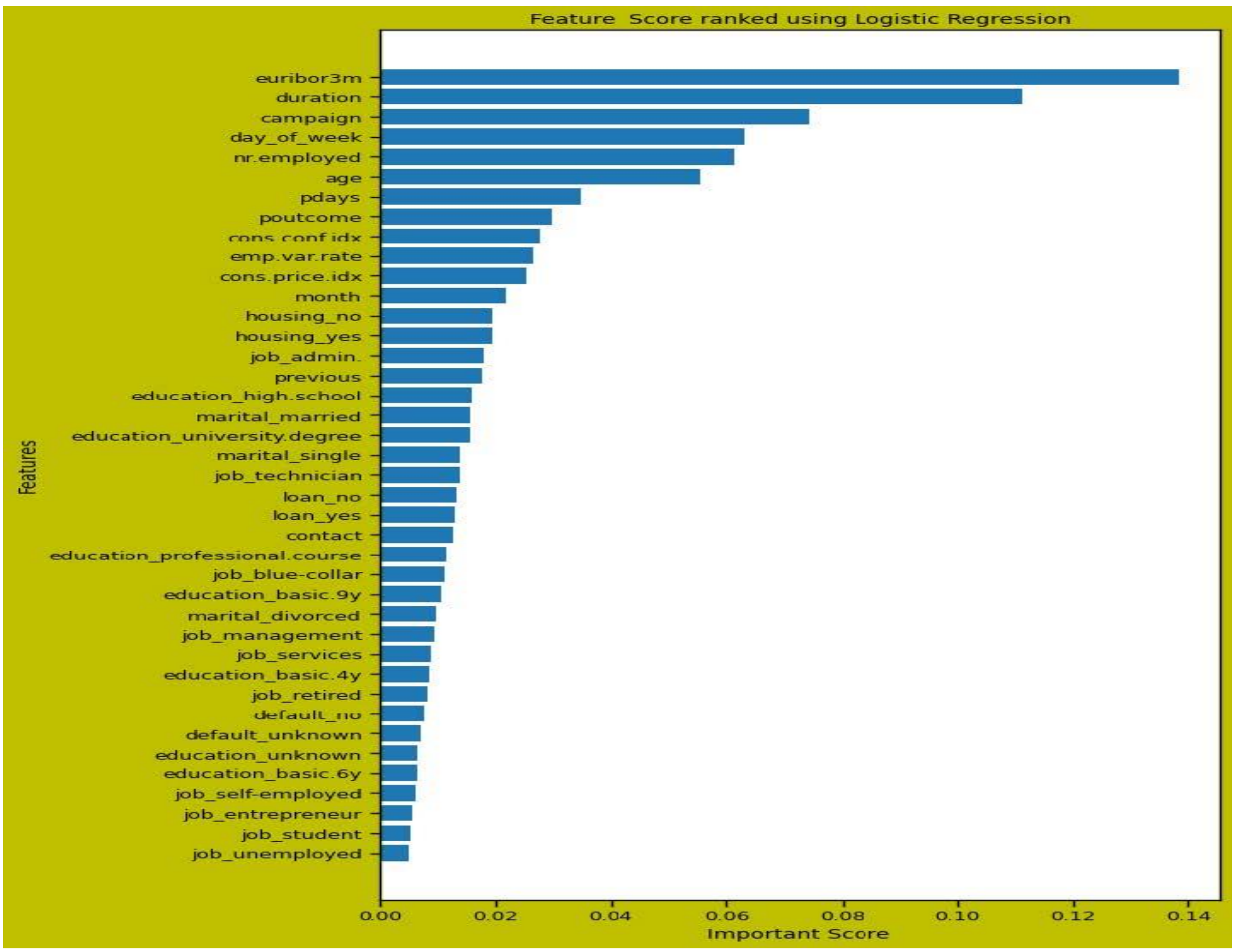
```
# This line represents the model.
rfc = RandomForestClassifier()
# Fit model on your training data.
rfc.fit(X, y.values.ravel())
#Score it on your testing data.
rfc.score(X, y)

feature_importances = pd.DataFrame(rfc.feature_importances_,
                                   index = X.columns,
                                    columns=['importance']).sort_values(
```

feature_importances.to_csv('feature_importance1.csv')

Feature Importance	
euribor3m	0.1383621
duration	0.1111846
campaign	0.0743802
day_of_week	0.0630702
nr.employed	0.0613822
age	0.0554593
pdays	0.0345648
poutcome	0.0297964
cons.conf.idx	0.0275596
emp.var.rate	0.0266189
cons.price.idx	0.02541
month	0.0217213
housing_yes	0.0195624
housing_no	0.0194693
job_admin.	0.0179948
previous	0.0176762
education_high.school	0.0159899
marital_married	0.0156853
education_university.degree	0.0156173
marital_single	0.0140019
job_technician	0.0139163
loan_no	0.0131902
loan_yes	0.0129705
contact	0.0127002
education_professional.course	0.0116457
job_blue-collar	0.0112551
education_basic.9y	0.0108294
marital_divorced	0.0098932
job_management	0.009383
job_services	0.0088904
education_basic.4y	0.008512
job_retired	0.0082803
default_no	0.0077811
default_unknown	0.0071442
education_unknown	0.0067038
education_basic.6y	0.0066351
job_self-employed	0.0062246
job_entrepreneur	0.0057664
job_student	0.0054348
job_unemployed	0.0050417
job_housemaid	0.0041849
loan_unknown	0.0024353
job_unknown	0.0023192
housing_unknown	0.0022733
marital_unknown	0.0007944
education_illiterate	0.0002882
default_yes	7.67E-09

The bar chart graph below shows the highest features which counts towards term deposit subscription:

```
number = np.min([40, len(X.columns)])
ylog = np.arange(number)
# Feature importance for top num & sort in reverse order
valuestoplot = feature_importances.iloc[:number].values.ravel()[::-1]
feature_labels = list(feature_importances.iloc[:number].index)[::-1]

plt.figure(num=None, figsize=(8, 15), dpi=80, facecolor='y', edgecolor='k');
plt.barh(ylog, valuestoplot, align = 'center')
plt.ylabel('Features')
plt.xlabel('Important Score')
plt.title('Feature  Score ranked using Logistic Regression')
plt.yticks(ylog, feature_labels)
plt.show()
```



```
combinebankinfo['y'].replace(['yes', 'no'],[1,0 ], inplace=True)
```

```
combinebankinfo[['y']] = combinebankinfo[['y']].apply(pd.to_numeric)

```

```
combinebankinfo['y'].sample(30)
```

```
]: 37019    1
   2849     0
   12725    0
   11432    0
   18965    0
   13775    0
   18971    0
   39139    0
   32939    0
   20447    0
   15494    0
   3187     0
   40917    0
   7104     0
   25755    0
   14919    1
   27276    0
   31463    1
   39373    0
   40037    0
   28122    0
   29283    0
   23295    1
   12811    0
   29857    0
   25977    0
   11928    0
   148      0
   10592    0
   40421    1
   Name: y, dtype: int64
```

```
combinebankinfo.columns
```

```
Index(['age', 'job_admin.', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired', 'job_self-employe
d',
       'job_services', 'job_student', 'job_technician', 'job_unemployed',
       'job_unknown', 'marital_divorced', 'marital_married', 'marital_singl
e',
       'marital_unknown', 'education_basic.4y', 'education_basic.6y',
       'education_basic.9y', 'education_high.school', 'education_illiterat
e',
       'education_professional.course', 'education_university.degree',
       'education_unknown', 'default_no', 'default_unknown', 'default_yes',
       'housing_no', 'housing_unknown', 'housing_yes', 'loan_no',
       'loan_unknown', 'loan_yes', 'contact', 'month', 'day_of_week',
       'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'emp.var.ra
te',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

```
1 combinebankinfo.sample(30)
```

	age	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_ret
10045	4	1	0	0	0	0	
13763	2	0	1	0	0	0	
30393	1	1	0	0	0	0	
32431	3	1	0	0	0	0	
18487	1	0	0	0	0	0	
9516	2	0	0	0	0	0	
7938	4	0	0	0	0	0	
21175	1	0	0	0	0	1	
24194	4	1	0	0	0	0	
3064	3	0	0	0	0	0	
39628	3	1	0	0	0	0	
21250	2	1	0	0	0	0	
5811	4	0	0	1	0	0	
29512	2	0	0	0	0	0	
40672	3	0	0	0	0	0	
37914	2	0	0	0	0	0	
25580	1	0	0	0	0	0	
10779	3	0	1	0	0	0	
36795	1	1	0	0	0	0	
37136	3	0	0	0	0	0	
26963	1	0	0	0	0	1	
7318	4	1	0	0	0	0	
2758	3	0	0	0	0	0	
22350	4	0	1	0	0	0	
449	2	0	1	0	0	0	
26111	1	0	1	0	0	0	
5497	3	0	0	0	0	0	
28589	2	1	0	0	0	0	
36018	4	0	0	0	0	0	
33579	2	0	0	0	0	1	

30 rows × 48 columns

Feature Importance:

Create a new Data frame of top features as calculated in the feature importance bar chart and table in red fonts which consider 30% of top test features out of 48 featuresi.e:14 columns including ‘y’ so we are taking first highest 14 as the main features of the bank data set making the most influence on the term deposit subscription.

```python
combinebankinfo = combinebankinfo[[
'y',
'euribor3m',
'duration',
'campaign',
'day_of_week',
'age',
'nr.employed',
'pdays',
'cons.conf.idx',
'emp.var.rate',
'poutcome',
'cons.price.idx',
'month',
'housing_yes',
]]
```

```python
combinebankinfo.sample(30)
```

	y	euribor3m	duration	campaign	day_of_week	age	nr.employed	pdays	cons.conf.idx	emp.var.rate	poutcome	cons.price.idx	month	housing_
29766	0	1.405	4.0	1	1	1	5099.1	999	-47.1	-1.8	1	93.075	4	
31456	0	1.334	4.0	2	3	4	5099.1	999	-46.2	-1.8	1	92.893	5	
7341	0	4.864	4.0	1	5	3	5191.0	999	-36.4	1.1	1	93.994	5	
38905	0	0.716	4.0	2	2	2	5017.5	6	-30.1	-3.4	2	92.649	11	
26981	0	4.076	1.0	5	4	2	5195.8	999	-42.0	-0.1	1	93.200	11	
17837	0	4.961	1.0	2	2	3	5228.1	999	-42.7	1.4	1	93.918	7	
1539	0	4.855	1.0	5	4	3	5191.0	999	-36.4	1.1	1	93.994	5	
15542	0	4.957	4.0	7	5	3	5228.1	999	-42.7	1.4	1	93.918	7	
5322	0	4.857	4.0	4	5	1	5191.0	999	-36.4	1.1	1	93.994	5	
12494	0	4.960	2.0	1	1	2	5228.1	999	-42.7	1.4	1	93.918	7	
13999	0	4.963	3.0	2	5	1	5228.1	999	-42.7	1.4	1	93.918	7	
25530	0	4.120	4.0	1	3	3	5195.8	999	-42.0	-0.1	1	93.200	11	
25739	0	4.120	1.0	1	3	2	5195.8	999	-42.0	-0.1	0	93.200	11	
16273	1	4.961	4.0	2	2	2	5228.1	999	-42.7	1.4	1	93.918	7	
33192	0	1.291	1.0	1	2	3	5099.1	999	-46.2	-1.8	1	92.893	5	
30024	0	1.405	1.0	1	3	3	5099.1	999	-47.1	-1.8	1	93.075	4	
29893	0	1.405	1.0	2	1	4	5099.1	999	-47.1	-1.8	1	93.075	4	

```python
combinebankinfo.describe()
```

	y	euribor3m	duration	campaign	day_of_week	age	nr.employed	pdays	cons.conf.idx	emp.var.rate	p
count	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	4118
mean	0.112654	3.621291	2.495411	2.567593	2.979581	2.442872	5167.035911	962.475454	-40.502600	0.081886	(
std	0.316173	1.734447	1.117039	2.770014	1.411514	1.116283	72.251528	186.910907	4.628198	1.570960	(
min	0.000000	0.634000	1.000000	1.000000	1.000000	1.000000	4963.600000	0.000000	-50.800000	-3.400000	(
25%	0.000000	1.344000	1.000000	1.000000	2.000000	1.000000	5099.100000	999.000000	-42.700000	-1.800000	
50%	0.000000	4.857000	2.000000	2.000000	3.000000	2.000000	5191.000000	999.000000	-41.800000	1.100000	
75%	0.000000	4.961000	3.000000	3.000000	4.000000	3.000000	5228.100000	999.000000	-36.400000	1.400000	
max	1.000000	5.045000	4.000000	56.000000	5.000000	4.000000	5228.100000	999.000000	-26.900000	1.400000	

```python
combinebankinfo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   y               41188 non-null  int64  
 1   euribor3m       41188 non-null  float64
 2   duration        41188 non-null  float64
 3   campaign        41188 non-null  int64  
 4   day_of_week     41188 non-null  int64  
 5   age             41188 non-null  int32  
 6   nr.employed     41188 non-null  float64
 7   pdays           41188 non-null  int64  
 8   cons.conf.idx   41188 non-null  float64
 9   emp.var.rate    41188 non-null  float64
 10  poutcome        41188 non-null  int32  
 11  cons.price.idx  41188 non-null  float64
 12  month           41188 non-null  int64  
 13  housing_yes     41188 non-null  uint8  
dtypes: float64(6), int32(2), int64(5), uint8(1)
memory usage: 3.8 MB
```

```python
X = combinebankinfo.loc[:, combinebankinfo.columns != 'y']
y = combinebankinfo.loc[:, combinebankinfo.columns == 'y']
```

```
X = combinebankinfo.loc[:, combinebankinfo.columns != 'y']
y = combinebankinfo.loc[:, combinebankinfo.columns == 'y']
```

```
X.columns
```

```
Index(['euribor3m', 'duration', 'campaign', 'day_of_week', 'age',
       'nr.employed', 'pdays', 'cons.conf.idx', 'emp.var.rate', 'poutcome',
       'cons.price.idx', 'month', 'housing_yes'],
      dtype='object')
```

```
y.columns
```

```
Index(['y'], dtype='object')
```

```
y.sample(30)
```

	y
27377	0
38725	0
15599	0
26159	0
34453	0
17427	0
19357	0
26992	0
3048	0
19942	0
38562	1
7479	0
21269	0
36175	0
26674	0
38061	1
34594	0

```
y.describe()
```

	y
count	41188.000000
mean	0.112654
std	0.316173
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

```
y['y'].value_counts()
```

```
0    36548
1     4640
Name: y, dtype: int64
```

Smote:

As we derived earlier, our dataset is imbalanced with only 11.3% of clients that have subscribed to the bank term deposit. We don't want prediction model to ignore the minority class so we're leveraging SMOTE functionality to produce oversampling to fairly balance our dataset and to reduce or get rid of bias. We did smote at this stage before we split /partitioned clean dataset into test and train as per standard best practice for modeling. Later we have chosen 70-30 split as train-test data for modeling purpose.

```
sm = SMOTE(random_state=0)
X_SMOTE, y_SMOTE = sm.fit_resample(X, y)
pd.Series(y_SMOTE['y']).value_counts()
```

```python
import sklearn.linear_model as linear_model
from sklearn.preprocessing import StandardScaler, label_binarize

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
perp_model = linear_model.Perceptron().fit(X_train_std,y_train.values.ravel())
y_pred = perp_model.predict(X_test_std)
print("Accuracy: ",round(accuracy_score(y_test, y_pred),2))
```

```
Accuracy:  0.89
```

Predictive Modeling Selection and Techniques

RandomForest Classification:

```python
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, np.ravel(y_train,order='C'))
y_pred = random_forest.predict(X_test)
random_forest.score(X_train, y_train)
bank_random_forest = round(random_forest.score(X_train, y_train) * 100, 2)

print(round(bank_random_forest,2,), "% :\n")
print("Classification Report :\n")
print(classification_report(y_test,y_pred))
print("\n")
print("Confusion matrix:\n")
print(confusion_matrix(y_test,y_pred))
print("\n")
print("Accuracy score :\n")
print(accuracy_score(y_test,y_pred)*100)
```

```
96.33 % :

Classification Report :

              precision    recall  f1-score   support

           0       0.93      0.96      0.94     10940
           1       0.56      0.40      0.47      1417

    accuracy                           0.90     12357
   macro avg       0.74      0.68      0.70     12357
weighted avg       0.88      0.90      0.89     12357


Confusion matrix:

[[10491   449]
 [  847   570]]


Accuracy score :

89.51201747997086
```

Logistic Regression:

```
banklogreg = LogisticRegression()
banklogreg.fit(X_train, np.ravel(y_train,order='C'))
y_pred = banklogreg.predict(X_test)
bank_log = round(banklogreg.score(X_train, y_train) * 100, 2)
print(round(bank_log,2,), "% :\n")
print("Classification Report :\n")
print(classification_report(y_test,y_pred))
print("\n")
print("Confusion matrix:\n")
print(confusion_matrix(y_test,y_pred))
print("\n")
print("Accuracy score :\n")
print(accuracy_score(y_test,y_pred)*100)
```

```
90.45 % :

Classification Report :

              precision    recall  f1-score   support

           0       0.91      0.99      0.95     10940
           1       0.73      0.28      0.41      1417

    accuracy                           0.91     12357
   macro avg       0.82      0.63      0.68     12357
weighted avg       0.89      0.91      0.89     12357



Confusion matrix:

[[10794   146]
 [ 1017   400]]


Accuracy score :

90.58833050093065
```

Linear SVC:

```
bank_linear_svc = LinearSVC()
bank_linear_svc.fit(X_train, np.ravel(y_train,order='C'))
y_pred = bank_linear_svc.predict(X_test)
bank_svc = round(bank_linear_svc.score(X_train, y_train) * 100, 2)
print(round(bank_svc,2,), "% :\n")
print("Classification Report :\n")
print(classification_report(y_test,y_pred))
print("\n")
print("Confusion matrix:\n")
print(confusion_matrix(y_test,y_pred))
print("\n")
print("Accuracy score :\n")
print(accuracy_score(y_test,y_pred)*100)
```

```
90.0 % :

Classification Report :

              precision    recall  f1-score   support

           0       0.90      0.99      0.95     10940
           1       0.79      0.18      0.30      1417

    accuracy                           0.90     12357
   macro avg       0.85      0.59      0.62     12357
weighted avg       0.89      0.90      0.87     12357



Confusion matrix:

[[10870    70]
 [ 1155   262]]


Accuracy score :

90.08659059642308
```

Decision Tree:

```python
bank_dt = DecisionTreeClassifier()
bank_dt.fit(X_train, y_train)
y_pred = bank_dt.predict(X_test)
bank_decision_tree = round(bank_dt.score(X_train, y_train) * 100, 2)
print(round(bank_decision_tree,2,), "% :\n")
print("Classification Report :\n")
print(classification_report(y_test,y_pred))
print("\n")
print("Confusion matrix:\n")
print(confusion_matrix(y_test,y_pred))
print("\n")
print("Accuracy score :\n")
print(accuracy_score(y_test,y_pred)*100)
```

```
96.33 % :

Classification Report :

              precision    recall  f1-score   support

           0       0.92      0.95      0.93     10940
           1       0.48      0.37      0.42      1417

    accuracy                           0.88     12357
   macro avg       0.70      0.66      0.68     12357
weighted avg       0.87      0.88      0.87     12357



Confusion matrix:

[[10367   573]
 [  892   525]]


Accuracy score :

88.1443716112325
```

<u>Here is the comparison of performance metrics of all four models</u>

```python
"Comparison of performance metrics of all three models"
results = pd.DataFrame({
    'Model': ['Random Forest',
              'Logistic Regression',
              'Support Vector Machines',
              'Decision Tree'],
    'Score': [bank_random_forest,
              bank_log,
              bank_svc,
              bank_decision_tree]})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(5)
```

Score	Model
96.33	Random Forest
96.33	Decision Tree
90.45	Logistic Regression
90.00	Support Vector Machines

As intended and planned earlier to derive the best fit, we did four types of modelling:

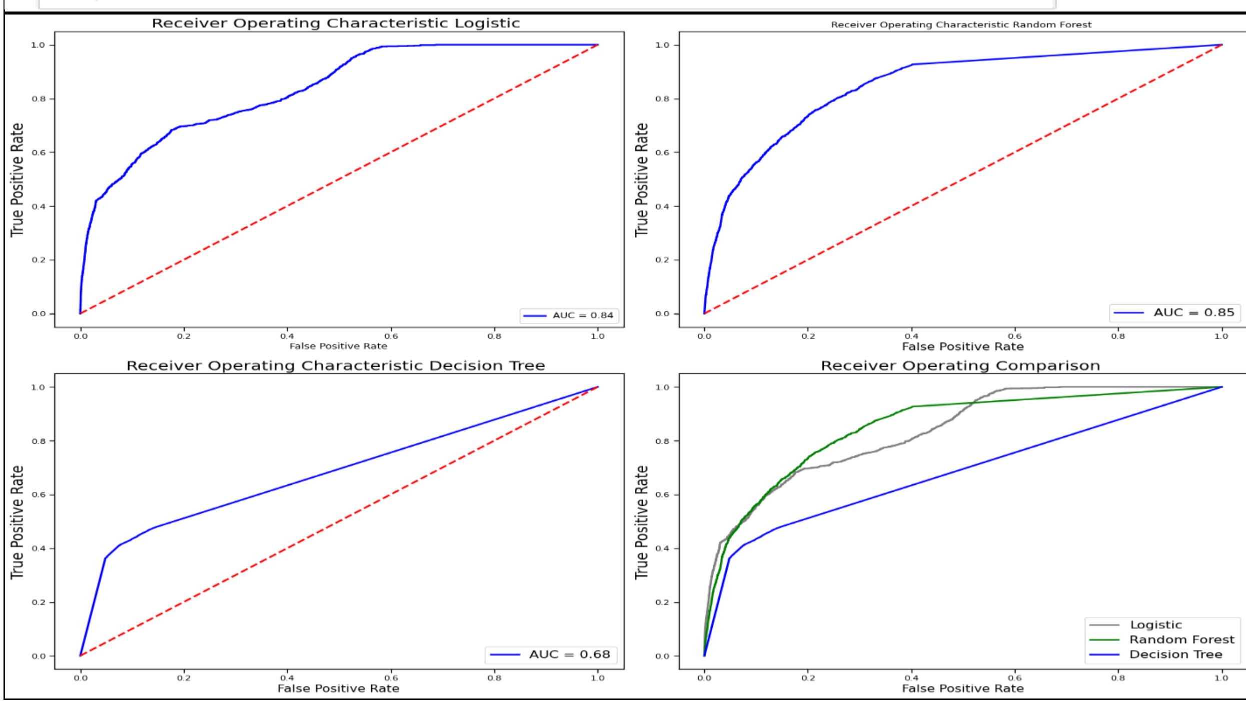
1- Logistic Regression
2- Decision Tree
3- Support Vector Machines (Linear SVC)
4- Random Forest

The highest accuracy score we received is of Random Forest **@ 96.33** which means our model will correctly predict the outcome 96.3% of the time. We will further plot ROC and compare AUC to support our model selection.

We have shortlisted top 3 models for the ROC-AUC curve.

After plotting the ROC and AUC Curve, we could conclude that Random Forest is the best model among all three models which we have chosen due to maximum area under the curve.

```python
from sklearn import metrics

#fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(nrows = 2, ncols = 2, fig
fig, ax_arr = plt.subplots(nrows = 2, ncols = 2, figsize = (20,15))

#LogisticRegression
probs = banklogreg.predict_proba(x_test)
preds = probs[:,1]
fprlog, tprlog, thresholdlog = metrics.roc_curve(y_test, preds)
roc_auclog = metrics.auc(fprlog, tprlog)

ax_arr[0,0].plot(fprlog, tprlog, 'b', label = 'AUC = %0.2f' % roc_auclog
ax_arr[0,0].plot([0, 1], [0, 1],'r--')
ax_arr[0,0].set_title('Receiver Operating Characteristic Logistic ',font
ax_arr[0,0].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,0].set_xlabel('False Positive Rate',fontsize=12)
ax_arr[0,0].legend(loc = 'lower right', prop={'size': 12})

#RANDOM FOREST --------------------
probs = random_forest.predict_proba(x_test)
preds = probs[:,1]
fprrfc, tprrfc, thresholdrfc = metrics.roc_curve(y_test, preds)
roc_aucrfc = metrics.auc(fprrfc, tprrfc)

ax_arr[0,1].plot(fprrfc, tprrfc, 'b', label = 'AUC = %0.2f' % roc_aucrfc
ax_arr[0,1].plot([0, 1], [0, 1],'r--')
ax_arr[0,1].set_title('Receiver Operating Characteristic Random Forest ',
ax_arr[0,1].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,1].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[0,1].legend(loc = 'lower right', prop={'size': 16})

#DECISION TREE ---------------------
probs = bank_dt.predict_proba(x_test)
preds = probs[:,1]
fprdtree, tprdtree, thresholddtree = metrics.roc_curve(y_test, preds)
roc_aucdtree = metrics.auc(fprdtree, tprdtree)

ax_arr[1,0].plot(fprdtree, tprdtree, 'b', label = 'AUC = %0.2f' % roc_au
ax_arr[1,0].plot([0, 1], [0, 1],'r--')
ax_arr[1,0].set_title('Receiver Operating Characteristic Decision Tree ',
ax_arr[1,0].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,0].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,0].legend(loc = 'lower right', prop={'size': 16})

#ALL PLOTS --------------------------------
ax_arr[1,1].plot(fprlog, tprlog, 'b', label = 'Logistic', color='grey')
ax_arr[1,1].plot(fprrfc, tprrfc, 'b', label = 'Random Forest', color='gr
ax_arr[1,1].plot(fprdtree, tprdtree, 'b', label = 'Decision Tree', color=
ax_arr[1,1].set_title('Receiver Operating Comparison ',fontsize=20)
ax_arr[1,1].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,1].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,1].legend(loc = 'lower right', prop={'size': 16})

plt.subplots_adjust(wspace=0.2)
plt.tight_layout()
```



Predictive modeling solution addressing the business problem:

As a conclusion, we have created a machine learning models that can predict how likely clients will subscribe to a bank term deposit. The best model stood out is Random forest classifier. Its test performance (AUC) is highest 85%. The model was able to catch 85% of customers that will subscribe to a term deposit. Based on the analysis, we highly recommend that bank should focus on targeting customers with high duration and euribor3m as they are high importance features for the model and business. Longer call means persuade customer to sign up and 3 months for paying off their loans. As an outcome or end result, doing so, we as an analyst, helped bank(Senior Leadership/Management Team) managed to save time and money knowing the characteristics/pattern of clients they should market/tap/chase and take strategic business decisions(Go to Market and Demand/Lead gen exercises) which in turn will lead to increased growth & revenue, capture market share, gain customer loyalty, retain existing customers and add new buying accounts which are highly future potential for bank to offer and sell their other products/services through which they can generate more revenue streams.