In [1]:
```
#Packages
import numpy as np
import numpy.random as nr
import pandas as pd
from pandas import Series, DataFrame
import matplotlib as mpl
from matplotlib import pyplot
import matplotlib.pyplot as plt
from pylab import plot, show
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics as sklm
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA

from imblearn.under_sampling import NearMiss
from imblearn.pipeline import make_pipeline
from imblearn.metrics import classification_report_imbalanced



from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
```

Loading Data and Inspection

In [2]:
```
bank_additional_full=pd.read_csv('bank-additional-full.csv',sep=';')
data =pd.read_csv('bank-additional-full.csv',sep=';')
```

In [3]:
```
bank_additional_full.columns
```

Out[3]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

In [6]:
```
bank_additional_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

Data Evaluation

Here seggregating numeric and categoric features 1.age (numeric)

2.job : type of job (categorical: “admin”, “blue-collar”, “entrepreneur”, “housemaid”, “management”, “retired”, “self-employed”, “services”, “student”, “technician”, “unemployed”, “unknown”)

3.marital : marital status (categorical: “divorced”, “married”, “single”, “unknown”)

4.education (categorical: “basic.4y”, “basic.6y”, “basic.9y”, “high.school”, “illiterate”, “professional.course”, “university.degree”, “unknown”)

5.default: has credit in default? (categorical: “no”, “yes”, “unknown”)

6.housing: has housing loan? (categorical: “no”, “yes”, “unknown”)

7.loan: has personal loan? (categorical: “no”, “yes”, “unknown”)

8.contact: contact communication type (categorical: “cellular”, “telephone”)

9.month: last contact month of year (categorical: “jan”, “feb”, “mar”, …, “nov”, “dec”)

10.day_of_week: last contact day of the week (categorical: “mon”, “tue”, “wed”, “thu”, “fri”)

11.duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y=’no’). 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

12.campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13.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)

14.previous: number of contacts performed before this campaign and for this client (numeric)

15.poutcome: outcome of the previous marketing campaign (categorical: “failure”, “nonexistent”, “success”)

16.emp.var.rate: employment variation rate — (numeric)

17.cons.price.idx: consumer price index — (numeric)

18.cons.conf.idx: consumer confidence index — (numeric)

19.euribor3m: euribor 3 month rate — (numeric)

20.nr.employed: number of employees — (numeric)

Target Variable

In [4]:
```
print(bank_additional_full.shape)
```

```
(41188, 21)
```

In [5]:
```
bank_additional_full.isnull().sum()
```

Out[5]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

In [7]:
```
bank_additional_full.describe()
```

Out[7]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	4118
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	9
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	9
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	9
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	9
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	9
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	9

In [8]:
```
pd.set_option("max_columns", None)
bank_additional_full.head()
```

Out[8]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon
1	57	services	married	high.school	unknown	no	no	telephone	may	mon
2	37	services	married	high.school	no	yes	no	telephone	may	mon
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon
4	56	services	married	high.school	no	no	yes	telephone	may	mon

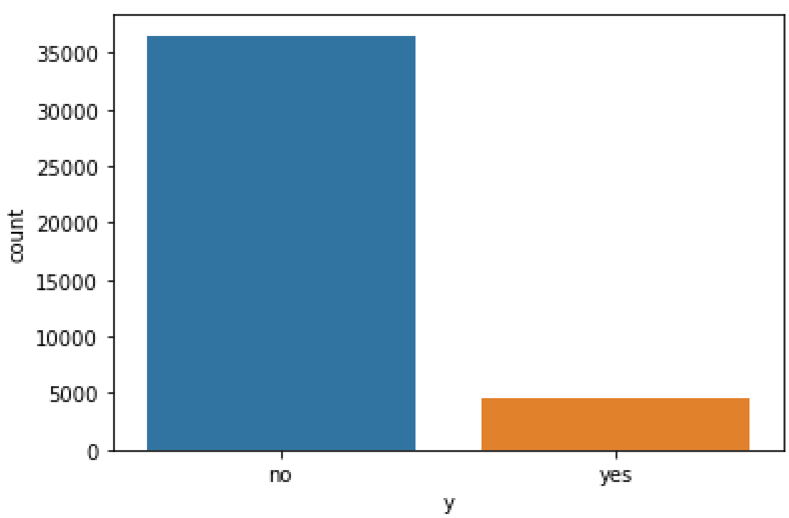
Visualizing the Data

Using Exploratory Data Analysis (EDA) to understand the relationships for this classification problem where the labels are categorical variables. Separation is achieved when there are distinctive feature values for each label category.

Examine classes and class imbalance

In [9]:
```
sns.countplot(x='y', data=bank_additional_full)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x2126751e970>

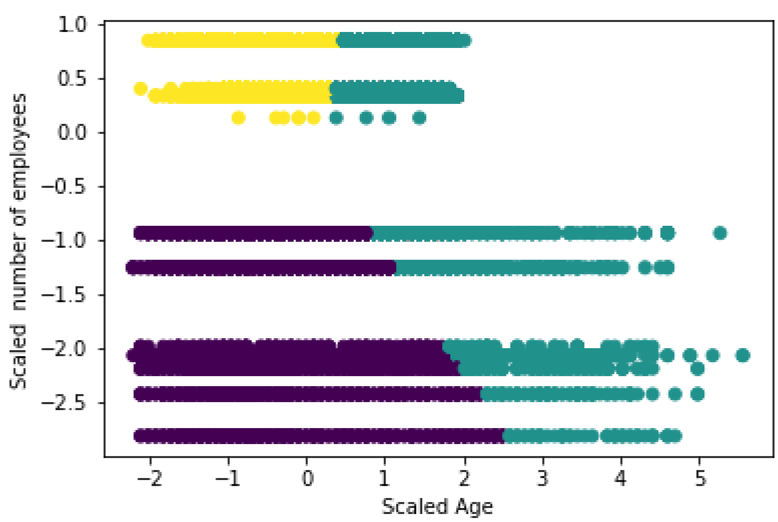


In this case, the label has significant **class imbalance**. Class imbalance means that there are unequal numbers of cases for the categories of the label. Class imbalance can seriously bias the training of classifier algorithms. It many cases, the imbalance leads to a higher error rate for the minority class. Most real-world classification problems have class imbalance, sometimes severe class imbalance, so it is important to test for this before training any model.

This is highly imbalanced dataset accuracy can't work well with imbalanced data so we can choose AUC as metric

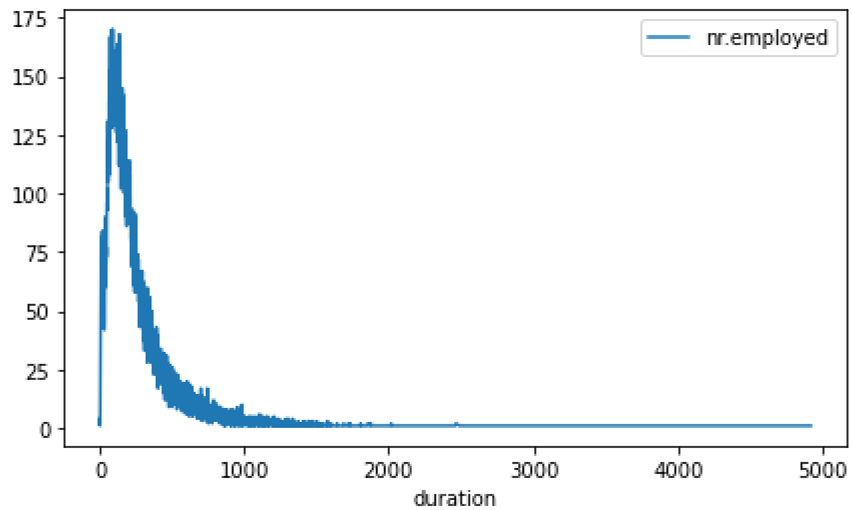
The pattern of particular age range employee - shows the relationship between the employees and their age. This indicates that the indiviudal employee who will eventually become a customer of the bank.

In [10]:
```
from sklearn import preprocessing
from sklearn import cluster
import matplotlib.pyplot as plt

additional_bank = bank_additional_full[['age','nr.employed']].values
additional_bank_scaled = preprocessing.scale(additional_bank)


kmeans = cluster.KMeans(n_clusters = 3)
kmeans.fit(additional_bank_scaled)
y_pred = kmeans.predict(additional_bank_scaled)

plt.scatter(additional_bank_scaled[:, 0], additional_bank_scaled[:, 1], c = y_
pred)
plt.xlabel('Scaled Age')
plt.ylabel('Scaled  number of employees')
plt.show()
```



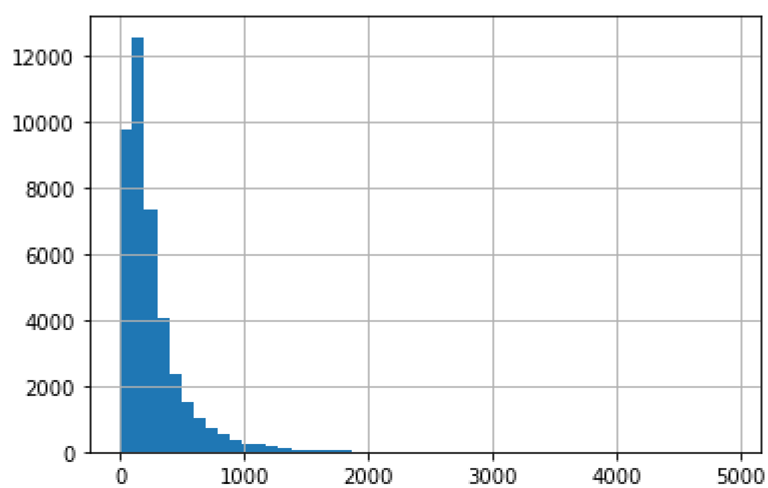
In [11]:
```
fig, ax = plt.subplots(figsize=(7,4))
bank_additional_full.groupby(['duration']).count()[['nr.employed']].plot(ax=ax
)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x212677ceeb0>



In [12]:
```
bank_additional_full['duration'].hist(bins=50)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x21268369e50>



Feature Engineering

Describing dummy keys of particular column

In [13]:
```
y ={'yes' : 1, 'no' : 0}
bank_additional_full['y_cat'] = bank_additional_full['y'].map(lambda x: y[x])
bank_additional_full['y_cat'].value_counts()
```

Out[13]:
```
0    36548
1     4640
Name: y_cat, dtype: int64
```

Marital Status within different Age Groups of groupby people

In [14]:
```
age_group_names = ['young', 'lower middle', 'middle', 'senior']
bank_additional_full['Age_Group'] = pd.qcut(bank_additional_full['age'], 4, la
bels = age_group_names)
bank_additional_full['Age_Group'].value_counts()
bank_additional_full['y_cat'].groupby([bank_additional_full['marital'],bank_ad
ditional_full['Age_Group']]).mean().round(3)
```

Out[14]:
```
marital   Age_Group
divorced  young           0.074
          lower middle    0.094
          middle          0.082
          senior          0.134
married   young           0.098
          lower middle    0.092
          middle          0.076
          senior          0.136
single    young           0.166
          lower middle    0.117
          middle          0.088
          senior          0.109
unknown   young           0.250
          lower middle    0.042
          middle          0.333
          senior          0.115
Name: y_cat, dtype: float64
```

Indicating life stage of different age group

In [15]:
```
bank_additional_full['life_stage'] = bank_additional_full.apply(lambda x: x['A
ge_Group'] +' & ' + x['marital'], axis = 1)
bank_additional_full['life_stage'].value_counts()
```

Out[15]:
```
senior & married            7478
middle & married            7111
young & single              6418
lower middle & married      6104
young & married             4235
lower middle & single       2942
senior & divorced           1745
middle & single             1584
middle & divorced           1439
lower middle & divorced      929
senior & single              624
young & divorced             499
senior & unknown              26
young & unknown               24
lower middle & unknown        24
middle & unknown               6
Name: life_stage, dtype: int64
```

In [16]:
```
bank_additional_full.replace(['basic.6y','basic.4y', 'basic.9y'], 'basic', inp
lace=True)
bank_additional_full.head(5)
```

Out[16]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
0	56	housemaid	married	basic	no	no	no	telephone	may	mon
1	57	services	married	high.school	unknown	no	no	telephone	may	mon
2	37	services	married	high.school	no	yes	no	telephone	may	mon
3	40	admin.	married	basic	no	no	no	telephone	may	mon
4	56	services	married	high.school	no	no	yes	telephone	may	mon

Machine Learning Models

Applications of Classification

A classifier is a machine learning model that separates the **label** into categories or **classes**. In other words, classification models are **supervised** machine learning models which predict a categorical label.

Here we considered the following

- To prepare data for classification models using scikit-learn and imblance-learn packages.
- Constructing a classification model using scikit-learn and imblance-learn packages.
- Evaluating the performance of the classification models.
- Using techniques such as reweighting the labels and changing the decision threshold to change the trade-off between false positive and false negative error rate

Prepare data for Classification models

With the data prepared, to create the numpy arrays required for the scikit-learn model.

To create the numpy feature array or model matrix. We need to follow the following steps

1. Seggregate into categorical columns and numeric columns
2. Transform the integer coded variables to dummy variables.
3. Encode the categorical string variables as integers.
4. Split the cases into training and test data sets. If machine learning models are tested on the training data, the results will be both biased and overly optimistic.
5. Numeric features must be rescaled so they have a similar range of values. Rescaling prevents features from having an undue influence on model training simply because then have a larger range of numeric variables.
6. Append each dummy coded categorical variable to the model matrix.
7. Execute the code in the cell below to perform this processing and examine the results

In [17]:
```
cat_columns = ['job','marital','education', 'default', 'housing', 'loan', 'month', 'day_of_week', 'poutcome','euribor3m','Age_Group','life_stage']
num_columns = ['age','duration','campaign','pdays','previous','emp.var.rate','cons.conf.idx','cons.price.idx', 'euribor3m','nr.employed']
```

In [18]:
```
for col in bank_additional_full.columns:
    if bank_additional_full[col].dtype==object:
          bank_additional_full[col]=bank_additional_full[col].astype('category')
          bank_additional_full[col]=bank_additional_full[col].cat.codes
```

In [19]:
```
cat_cols = []
```

OneHotEncoder

Encode categorical features as a one-hot numeric array. This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

In [20]:
```
ohe = OneHotEncoder()
df_cat = bank_additional_full[cat_columns]
encoded = ohe.fit_transform(df_cat)
ohe_df = pd.DataFrame(encoded.todense(), columns=ohe.get_feature_names())
```

In [21]:
```
df_num = bank_additional_full[num_columns].reset_index(drop=True)
```

In [22]:
```
y = bank_additional_full['y_cat']
```

In [23]:
```
X = pd.concat([df_num, ohe_df], axis = 1)
y = y
```

In [24]:
```
X.shape ,y.shape
```

Out[24]:
```
((41188, 395), (41188,))
```

In [25]:
```
bank_additional_full["y"].value_counts()/len(bank_additional_full.y)
```

Out[25]:
```
0    0.887346
1    0.112654
Name: y, dtype: float64
```

Understanding graphs

1. 88% of the observations are No as compared to just 11 % data as Yes.Clearly this dataset is an imbalanced dataset.
2. Features need to be preprocessed.
3. Some Important features are
   A. Jobs
   B. Education (illterate people seems to get convinced by campaign)
   C. Months (Seasonal effect can been seen in the data. In may the sales increases)
   D. Previous outcome seems important because new customers seems to convert more
   E. Average age of people who converted as higher than those who didn't
   F. People who converted were exposed to fewer campaigns than those who didn't

In [26]:
```
seed = 7
test_size = 0.2
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.2, random_state=seed)
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (32950, 395)
Number transactions y_train dataset:  (32950,)
Number transactions X_test dataset:  (8238, 395)
Number transactions y_test dataset:  (8238,)
```

In [27]:
```
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [28]:
```
import warnings
warnings.filterwarnings("ignore")
```

Score and evaluate the classification model

Given the results of the test data, here, in order to quantify the performance of the model, we used confusion matrix to evaluate the performance of the machine leaning model classifiers. The confusion matrix lays out the correctly and incorrectly classified cases in a tabular format.

Confusion matrix

Here the four elements in the matrix are defined as:
True Positive or **TP** are cases with positive labels which have been correctly classified as positive.
True Negative or **TN** are cases with negative labels which have been correctly classified as negative.
False Positive or **FP** are cases with negative labels which have been incorrectly classified as positive.
False Negative or **FN** are cases with positive labels which have been incorrectly classified as negative.

When creating a confusion matrix it is important to understand and maintain a convention for which differentiating positive and negative label values. The usual convention is to call the 1 case positive and the 0 case negative.

Accuracy : Accuracy is a simple and often misused metric. In simple terms, accuracy is the fraction of cases correctly classified. For a two-class classifier accuracy is written as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision :Precision is the fraction of correctly classified label cases out of all cases classified with that label value. We can express precision by the following relationship:

$$Precision = \frac{M_{i,i}}{\sum_j M_{i,j}}$$

Recall : Recall is the fraction of cases of a label value correctly classified out of all cases that actually have that label value. We can express recall by the following relationship:

$$Recall = \frac{M_{i,i}}{\sum_i M_{i,j}}$$

F1 :The F1 statistic is weighted average of precision and recall. We can express F1 by the following relationship:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

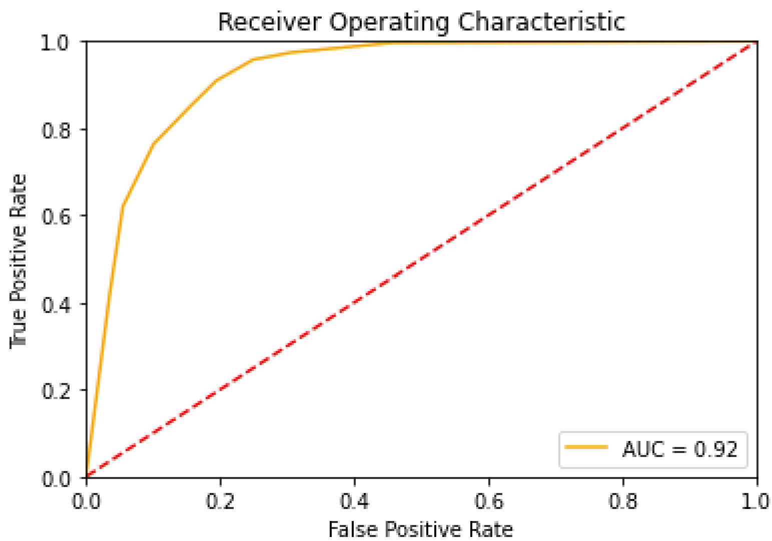
ROC and **AUC** : The receiver operating characteristic or ROC is a curve that displays the relationship between the true positive rate on the vertical axis and false positive rate on the horizontal axis. The ROC curve shows the tradeoff between true positive rate and false positive rate.

The AUC is the area or integral under the ROC curve. The overall performance of the classifier is measured by the area under the curve or AUC. The higher the AUC the lower the increase in false positive rate required to achieve a required true positive rate. For an ideal classifier the AUC is 1.0. A true positive rate is achieved with a 0 false positive rate. This behavior means that AUC is useful for comparing classifiers. The classifier with higher AUC is generally the better one.

Classify The Model Using DecisionTreeClassifier with GridSearchCV

In [32]:
```
SEED = 1
dt = DecisionTreeClassifier(random_state=SEED)
params_dt = {
             'max_depth':[3, 4, 5, 6],
             'min_samples_leaf': [0.04, 0.06, 0.08],
             'max_features': [0.2, 0.4, 0.6, 0.8]
            }
grid_dt = GridSearchCV(estimator = dt, param_grid=params_dt,scoring='roc_auc',
cv=10, n_jobs=-1)
model_grid_dt = grid_dt.fit(X_train, y_train)
best_hyperparams = grid_dt.best_params_
print('Best Hyperparameters:\n', best_hyperparams)

y_pred = grid_dt.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =grid_dt.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
Best Hyperparameters:
 {'max_depth': 6, 'max_features': 0.8, 'min_samples_leaf': 0.04}
[[6930  403]
 [ 344  561]]
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      7333
           1       0.58      0.62      0.60       905

    accuracy                           0.91      8238
   macro avg       0.77      0.78      0.77      8238
weighted avg       0.91      0.91      0.91      8238
```



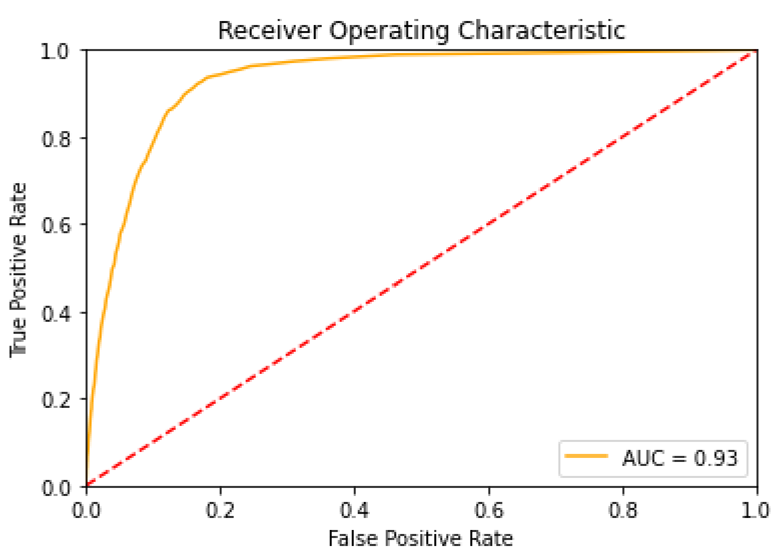
In [33]:
```
test_acc = model_grid_dt.score(X_test,y_test)
print("Test set accuracy of best model: {:.3f}".format(test_acc))
```

```
Test set accuracy of best model: 0.925
```

Classify The Model Using RandomForestClassifier

In [34]:
```
#rf = RandomForestClassifier(random_state = 42)

#dept = [1, 5, 10, 50, 100, 500, 1000]
n_estimators =  [20, 40, 60, 80, 100, 120]


param_grid={'n_estimators':n_estimators}
rf = RandomForestClassifier()
model_rf = GridSearchCV(rf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)

model_rf = rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =rf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[[7171  162]
 [ 581  324]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      7333
           1       0.67      0.36      0.47       905

    accuracy                           0.91      8238
   macro avg       0.80      0.67      0.71      8238
weighted avg       0.90      0.91      0.90      8238
```



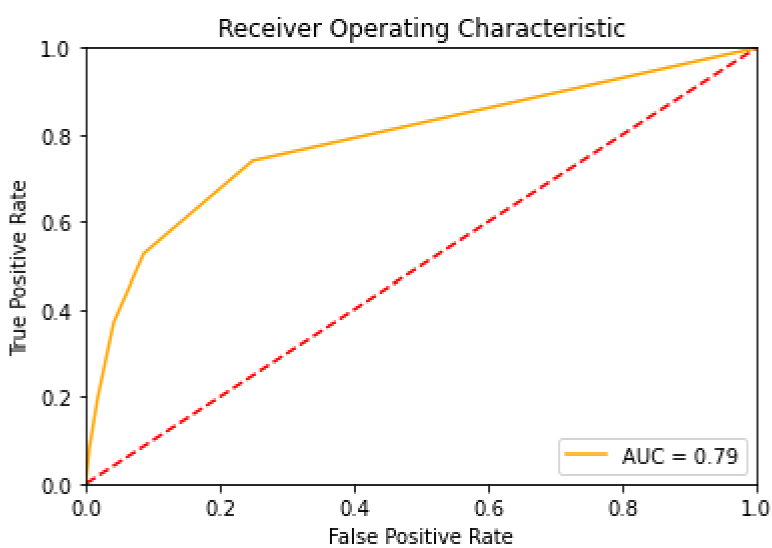
In [35]:
```
test_acc = model_rf.score(X_test,y_test)
print("Test set accuracy of best model: {:.3f}".format(test_acc))
```

```
Test set accuracy of best model: 0.910
```

Classify the model using KNeighborsClassifier

In [36]:
```
knn = KNeighborsClassifier(n_neighbors = 6)
model_knn = knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[[7204  129]
 [ 725  180]]
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      7333
           1       0.58      0.20      0.30       905

    accuracy                           0.90      8238
   macro avg       0.75      0.59      0.62      8238
weighted avg       0.87      0.90      0.87      8238
```



In [37]:
```
test_acc = model_knn.score(X_test,y_test)
print("Test set accuracy of best model: {:.3f}".format(test_acc))
```

```
Test set accuracy of best model: 0.896
```

SMOTE - Synthetic Minority Over-sampling Technique

To deal with data imbalance we use SMOTE - Synthetic Minority Over-sampling Technique. SMOTE creates synthetic (not duplicate) samples of the minority class. Hence making the minority class equal to the majority class. SMOTE does this by selecting similar records and altering that record one column at a time by a random amount within the difference to the neighbouring records. SMOTE is imported from imbalance learn over_sampling

In [29]:

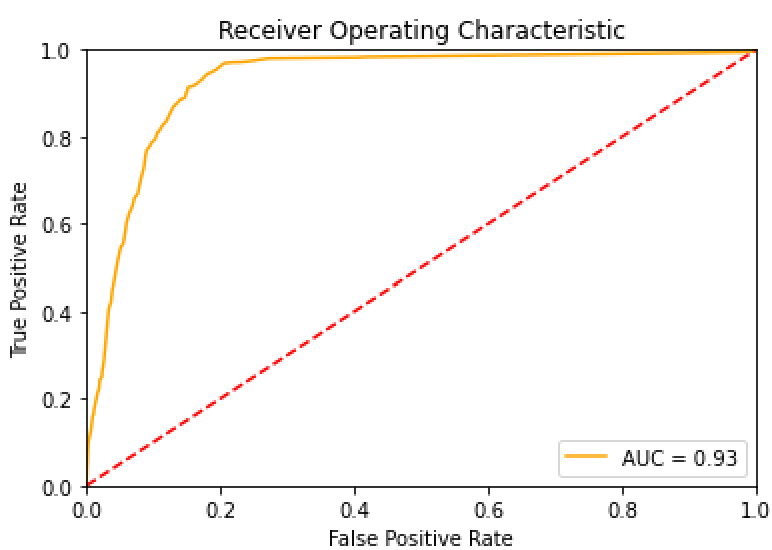
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0
)))

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape
))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.sha
pe))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1
)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0
)))
```

```
Before OverSampling, counts of label '1': 3735
Before OverSampling, counts of label '0': 29215

After OverSampling, the shape of train_X: (58430, 395)
After OverSampling, the shape of train_y: (58430,)

After OverSampling, counts of label '1': 29215
After OverSampling, counts of label '0': 29215
```

Classify the model using DecisionTreeClassifier with SMOTE

In [30]:
```
clf = DecisionTreeClassifier(max_depth = 10,min_samples_split = 500)
model_clf = clf.fit(X_train_res,y_train_res)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

pred_test =clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[[6446  887]
 [ 146  759]]
              precision    recall  f1-score   support

           0       0.98      0.88      0.93      7333
           1       0.46      0.84      0.60       905

    accuracy                           0.87      8238
   macro avg       0.72      0.86      0.76      8238
weighted avg       0.92      0.87      0.89      8238
```



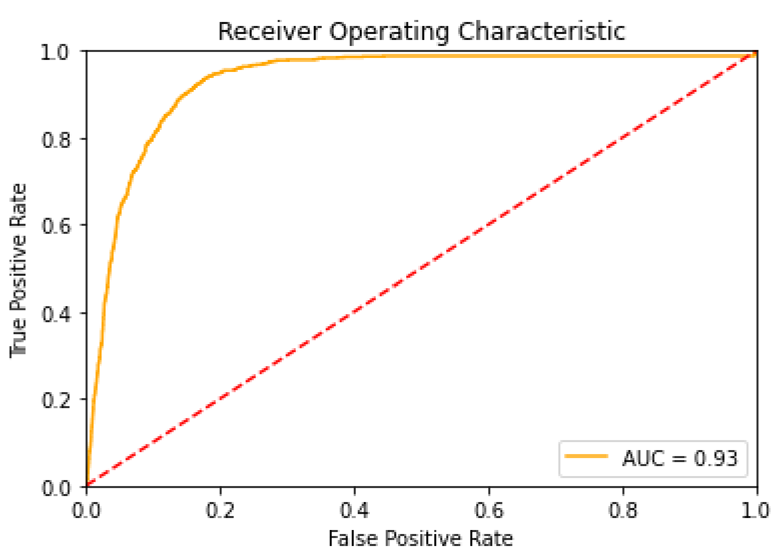
In [31]:
```
test_acc = model_clf.score(X_test,y_test)
print("Test set accuracy of best model: {:.3f}".format(test_acc))
```

```
Test set accuracy of best model: 0.875
```

Classify the model using LogisticRegression with SMOTE

In [40]:
```
logreg = LogisticRegression()
logreg.fit(X_train_res, y_train_res)
y_pred = logreg.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)

print(confusion_matrix)
print(classification_report(y_test, y_pred))


pred_test2 =logreg.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test2)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

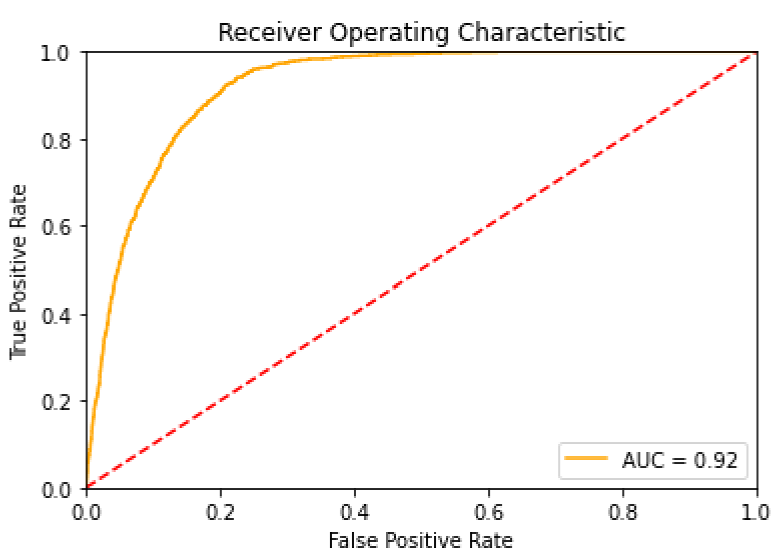
```
[[6444  889]
 [ 133  772]]
              precision    recall  f1-score   support

           0       0.98      0.88      0.93      7333
           1       0.46      0.85      0.60       905

    accuracy                           0.88      8238
   macro avg       0.72      0.87      0.76      8238
weighted avg       0.92      0.88      0.89      8238
```



Oversample The Minority Cases Using MLPClassifier

In [39]:
```
y_temp = y_train[y_train == 1]
X_temp = X_train[y_train == 1,:]
X_train = np.concatenate((X_train, X_temp), axis = 0)
y_train = np.concatenate((y_train, y_temp), axis = 0)

nr.seed(1115)

nn_mod = MLPClassifier(hidden_layer_sizes = (100,100),
                       max_iter=300)
nn_mod.fit(X_train, y_train)
y_pred1 = nn_mod.predict(X_test)

print(confusion_matrix(y_test, y_pred1))
print(classification_report(y_test, y_pred1))

pred_test1 =nn_mod.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = sklm.roc_curve(y_test, pred_test1)
auc = sklm.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
[[6908  425]
 [ 399  506]]
              precision    recall  f1-score   support

           0       0.95      0.94      0.94      7333
           1       0.54      0.56      0.55       905

    accuracy                           0.90      8238
   macro avg       0.74      0.75      0.75      8238
weighted avg       0.90      0.90      0.90      8238
```

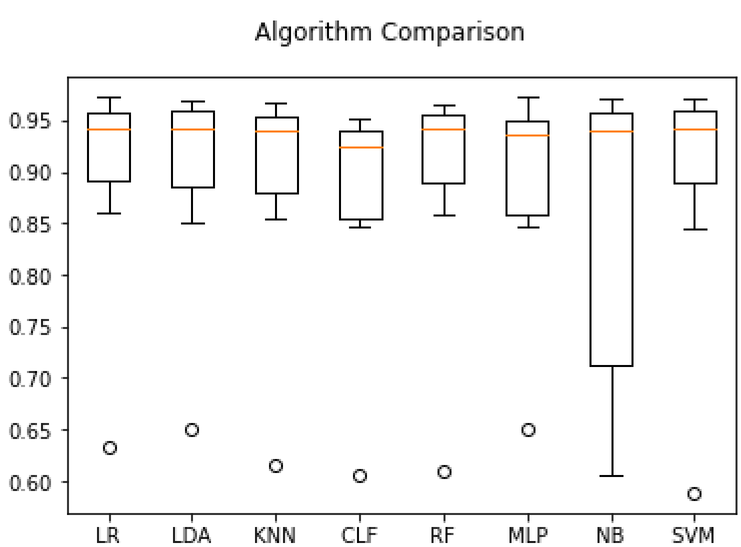


Various Classification models Algorithm Comparison

In [41]:
```
features_columns=['job','education', 'default', 'housing', 'loan',
           'month', 'day_of_week', 'duration', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m','y','Age_Group', 'life_stage']
features_columns_df=bank_additional_full[features_columns]
```

In [42]:
```
X = features_columns_df.iloc[:,0:15]
Y = features_columns_df.iloc[:,15]
models = []
models.append(( ' LR ' , LogisticRegression()))
models.append(( ' LDA ' , LinearDiscriminantAnalysis()))
models.append(( ' KNN ' , KNeighborsClassifier()))
models.append(( ' CLF ' , DecisionTreeClassifier()))
models.append(( ' RF ' , RandomForestClassifier()))
models.append(( ' MLP ' , MLPClassifier()))
models.append(( ' NB ' , GaussianNB()))
models.append(( ' SVM ' , SVC()))
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = KFold(n_splits=10, random_state=7)
    cv_results = cross_val_score(model, X, y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle( ' Algorithm Comparison ' )
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

```
 LR : 0.900886 (0.095720)
 LDA : 0.900862 (0.091429)
 KNN : 0.894282 (0.100214)
 CLF : 0.878792 (0.098713)
 RF : 0.896952 (0.101831)
 MLP : 0.893263 (0.090942)
 NB : 0.849946 (0.141457)
 SVM : 0.894815 (0.109118)
```



Making Prediction using Best Model Algorithm - Logistic Regression

In [43]:
```
def score_model(probs, threshold):
    return np.array([0 if x > threshold else 1 for x in probs[:,1]])
threshold = 0.51
probabilities = logreg.predict_proba(X_test)
scores = score_model(probabilities, threshold)
```

In [44]:
```
pd.set_option("max_columns", None)
print(scores)
```

```
[1 1 1 ... 1 1 1]
```

In [45]:
```
np.savetxt('final_answers_7.csv', scores, delimiter=',',fmt='%i')
```

Summary:

1. Loaded the data using pandas and worked with some Exploratory Data Analysis (EDA) techniques
2. After applying the EDA some feature engineering techniques applied
3. Used AUC as metric because this is higly imbalanced dataset
4. Seggregated Categorical Columns and Numerical Columns
5. Applied OneHotEncoder to encode the all catagorical features
6. Created Numerical Column data frame
7. Splited the whole data as train,test and cv using model_selection.train_test_split
8. Applied Machine Learning algorithms such as DecisionTreeClassifier,DecisionTreeClassifier with GridSearchCV, RandomForestClassifier, and KNeighborsClassifier
9. Created a Box Plot of Comparison of Various Classification models Algorithms
10. Logistic Regression with SMOTE gives best result AUC = 0.93 with Accuracy = 0.88
11. Created a CSV file for Prediction using Best Model Algorithm - Logistic Regression with SMOTE

In [ ]:
```

```