ML Lab Ex6

1. Use the attached file and run SVM, Decision tree, Random Forest and any one boosting algorithm.
2. Find out the different tunable parameters for each algorithms mentioned above.
3. Apply gridsearchCV and randomizedsearchCV for all the above classification algorithms and get the best parameters.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from scipy.stats import uniform as sp_randFloat
from scipy.stats import randint as sp_randInt
```

```
df = pd.read_csv("/content/Telco-Customer-Churn.csv")
```

```
df
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneServ
0	7590-VHVEG	Female	0	Yes	No	1	
1	5575-GNVDE	Male	0	No	No	34	
2	3668-QPYBK	Male	0	No	No	2	
3	7795-CFOCW	Male	0	No	No	45	
4	9237-HQITU	Female	0	No	No	2	
...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	
7039	2234-XADUH	Female	0	Yes	Yes	72	
7040	4801-JZAZL	Female	0	Yes	Yes	11	
7041	8361-LTMKD	Male	1	Yes	No	4	
7042	3186-AJIEK	Male	0	No	No	66	

7043 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64 
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64 
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
```

```
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
df.isnull().sum()
```

```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

1. Data Preprocessing

```
df = df.drop(["customerID"], axis = 1)
```

```
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors="coerce")
```

```
df.isnull().sum()
```

```
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

```
df = df.fillna(df["TotalCharges"].mean())
```

```
df.isnull().sum()
```

```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
```

```
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   float64
 19  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

```
df["PaymentMethod"].unique()
```

```
array(['Electronic check', 'Mailed check', 'Bank transfer (automatic)',
       'Credit card (automatic)'], dtype=object)
```

```
df.replace({'No phone service': "No", 'No internet service': "No"}, inplace=True)
```

```
df["OnlineSecurity"].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

```
df["Churn"].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

```
object_cols = df.select_dtypes(include=['object']).columns
le = LabelEncoder()
for col in object_cols:
  df[col] = le.fit_transform(df[col])
df
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	0	0	1	0	1	0	
1	1	0	0	0	34	1	
2	1	0	0	0	2	1	
3	1	0	0	0	45	0	
4	0	0	0	0	2	1	
...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	
7039	0	0	1	1	72	1	
7040	0	0	1	1	11	0	
7041	1	1	1	0	4	1	
7042	1	0	0	0	66	1	

7043 rows × 20 columns

Next steps: Generate code with df View recommended plots

```
df1 = pd.get_dummies(df)
df1.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
0	0	0	1	0	1	0	
1	1	0	0	0	34	1	
2	1	0	0	0	2	1	
3	1	0	0	0	45	0	
4	0	0	0	0	2	1	

Next steps: Generate code with df1 View recommended plots

```
X = df1.drop(columns = ["Churn"])
y = df1["Churn"].values
```

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

Model Prediction

SVM

```
svc = SVC(kernel="linear", random_state = 42)
svc.fit(X_train,y_train)
svc_acc = svc.score(X_test,y_test)
print("SVM accuracy:", svc_acc)
```

```
SVM accuracy: 0.7959431279620853
```

Decision Tree

```
dt = DecisionTreeClassifier(max_depth = 4)
dt.fit(X_train,y_train)
dt_acc = dt.score(X_test,y_test)
print("Decision Tree accuracy:",dt_acc)
```

```
Decision Tree accuracy: 0.7856128726928537
```

Random Forest

```
rf = RandomForestClassifier(n_estimators=100, criterion='gini',random_state=0)
rf.fit(X_train,y_train)
rf_acc = rf.score(X_test, y_test)
print("Random Forest accuracy:",rf_acc)
```

```
Random Forest accuracy: 0.7931850449597728
```

```
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0)
gb.fit(X_train, y_train)
gb_acc = gb.score(X_test, y_test)
print("Gradient Boost accuracy:",gb_acc)
```

```
Gradient Boost accuracy: 0.808329389493611
```

GridsearchCV and RandomizedsearchCV

```
params = {'max_depth':[4,5,6,7,8,9,10],
        'criterion':['gini','entropy'],
        'splitter' :['best', 'random'],
        "max_features": [1,2,3,4,5,6,7,8,9],
        "min_samples_leaf": [1,2,3,4,5,6,7,8,9],
        }
```

RSCV Decision Trees

```
tree = DecisionTreeClassifier(random_state=0)
tree_rscv = RandomizedSearchCV(tree, param, n_jobs=-1)
tree_rscv.fit(X_train,y_train)
print(tree_rscv.best_params_)
```

```
{'splitter': 'best', 'min_samples_leaf': 6, 'max_features': 8, 'max_depth': 6, 'criterion': 'entropy'}
```

```
dt = DecisionTreeClassifier(criterion= 'gini', max_depth= 8, max_features= 5, min_samples_leaf= 6, splitter= 'best')
dt.fit(X_train, y_train)
dt.score(X_test, y_test)
```

```
0.7780407004259347
```

GSCV Decision Trees

```
tree = DecisionTreeClassifier(random_state=0)
tree_gscv =  GridSearchCV(tree,param,cv=10, n_jobs=-1)
tree_gscv.fit(X_train,y_train)
print(tree_gscv.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 7, 'max_features': 8, 'min_samples_leaf': 8, 'splitter': 'best'}
```

```
dt = DecisionTreeClassifier(criterion= 'entropy', max_depth= 8, max_features= 9, min_samples_leaf= 8, splitter= 'best')
dt.fit(X_train, y_train)
dt.score(X_test, y_test)
```

```
0.7818267865593942
```

Support Vector Machine (SVM)

```
param = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],}
```

RSCV SVM

```
svm = SVC()
svm_rscv = RandomizedSearchCV(svm, param, n_jobs=-1)
svm_rscv.fit(X_train,y_train)
print(svm_rscv.best_params_)
```

```
{'gamma': 0.0001, 'C': 0.1}
```

```
svm = SVC( gamma = 0.01, C = 100)
svm.fit(X_train, y_train)
```

```
svm.score(X_test, y_test)
```

    0.7501183151916706

GSCV SVM

```
svm = SVC()
svm_gscv = GridSearchCV(svm, param, n_jobs=-1)
svm_gscv.fit(X_train,y_train)
print(svm_gscv.best_params_)
```

    {'C': 1, 'gamma': 0.0001}

```
svm = SVC( gamma = 0.01, C = 100)
svm.fit(X_train, y_train)
svm.score(X_test, y_test)
```

    0.7501183151916706

Random Forest

```
params = {'n_estimators':[50,75,100,125,200],
        'criterion':['gini','entropy'],
        'bootstrap':[True, False]}
```

RSCV Random Forest

```
rf =  RandomForestClassifier(random_state=0)
rf_rscv = RandomizedSearchCV(rf, param, n_jobs=-1)
rf_rscv.fit(X_train,y_train)
print(rf_rscv.best_params_)
```

    {'n_estimators': 50, 'criterion': 'entropy', 'bootstrap': True}

```
rf = RandomForestClassifier(n_estimators=100, criterion='entropy',random_state=0,bootstrap = True)
rf.fit(X_train,y_train)
rfc_accuracy = rf.score(X_test, y_test)
rf.score(X_test, y_test)
```

    0.7936583057264552

GSCV Random Forest

```
rf =  RandomForestClassifier(random_state=0)
rf_gscv = GridSearchCV(rf, param, n_jobs=-1)
rf_gscv.fit(X_train,y_train)
print(rf_gscv.best_params_)
```

    {'bootstrap': True, 'criterion': 'entropy', 'n_estimators': 50}

```
rf = RandomForestClassifier(n_estimators=100, criterion='entropy',random_state=0,bootstrap = True)
rf.fit(X_train,y_train)
rfc_accuracy = rf.score(X_test, y_test)
rf.score(X_test, y_test)
```

    0.7936583057264552

Boosting

```
params = {'max_depth'    : [2,3,4,5,6,7],
        'learning_rate':[0.15,0.1,0.05,0.01,0.005,0.001],
        'n_estimators':[100,250,500,750]
        }
```

RSCV Boosting

```
gb = GradientBoostingClassifier(random_state=0)
gb_rscv = RandomizedSearchCV(estimator=gb, param_distributions = param, n_jobs=-1)
gb_rscv.fit(X_train, y_train)
print(gb_rscv.best_params_)
```

```
{'n_estimators': 100, 'max_depth': 2, 'learning_rate': 0.15}
```

```
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.05,max_depth=3, random_state=0).fit(X_train, y_train)
gb_accuracy = gb.score(X_test, y_test)
gb.score(X_test, y_test)
```

```
0.7950780880265026
```

GSCV Boosting

```
gb = GradientBoostingClassifier(random_state=0)
gb_gscv = GridSearchCV(gb, param, n_jobs=-1)
gb_gscv.fit(X_train, y_train)
print(gb_gscv.best_params_)
```

```
{'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 500}}
```

```
gb = GradientBoostingClassifier(n_estimators=500, learning_rate=0.01,max_depth=5, random_state=0).fit(X_train, y_train)
gb_accuracy = gb.score(X_test, y_test)
gb.score(X_test, y_test)
```

```
0.7980679289099526
```