Assignment - AI for business

Utkarsh Thusoo

M20AIE318

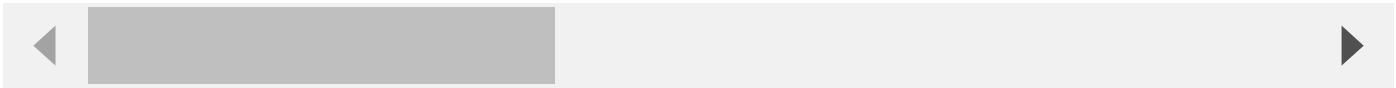
Importing libraries and Loading Data

In [1]:
```
import pandas as pd
Data=pd.read_csv("Data_Consumer_ChurnPrediction.csv")
Data
```

Out[1]:

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

7043 rows × 21 columns



Preprocessing - Encoding variables

In [2]:
```
Data.drop('customerID',axis=1,inplace = True) #customerId is not binary and categorica
```

In [3]:
```
Data['TotalCharges'] = pd.to_numeric(Data['TotalCharges'],errors='coerce') #Convert th
Data['TotalCharges'] = Data['TotalCharges'].fillna(Data['TotalCharges'].median()) #Rep

Data['Churn'] = Data['Churn'].map({'Yes':1, 'No':0})
Data['gender'] = Data['gender'].map({'Male':1, 'Female':0})
Data['Partner'] = Data['Partner'].map({'Yes':1, 'No':0}) #
Data['Dependents'] = Data['Dependents'].map({'Yes':1, 'No':0}) #
Data['PhoneService'] = Data['PhoneService'].map({'Yes':1, 'No':0}) #
Data['PaperlessBilling'] = Data['PaperlessBilling'].map({'Yes':1, 'No':0}) #


CatVar =  ['MultipleLines','InternetService','OnlineSecurity','OnlineBackup','DevicePr
Data = pd.get_dummies(Data, columns = CatVar, drop_first=False)
Data
```

Out[3]:

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

7043 rows × 41 columns

Preprocessing - scaling

In [4]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
Data['tenure'] = sc.fit_transform(Data[['tenure']])
Data['MonthlyCharges'] = sc.fit_transform(Data[['MonthlyCharges']])
Data['TotalCharges'] = sc.fit_transform(Data[['TotalCharges']])
```

Splitting the data

In [5]:
```
from sklearn.model_selection import train_test_split
X = Data.drop('Churn', axis=1)
y = Data['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

LogisticRegression

In [10]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.7988641741599621
precision:  0.6294117647058823
recall:  0.5763016157989228
f1_score:  0.7959305469162359
```

SVC Classifier

In [11]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = SVC()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.7969711310932324
precision:  0.6397379912663755
recall:  0.526032315978456
f1_score:  0.7901995671779648
```

RandomForestClassifier

In [12]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.7823000473260767
precision:  0.6061269146608315
recall:  0.49730700179533216
f1_score:  0.7749541892154461
```

DecisionTreeClassifier

In [13]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.7250354945575012
precision:  0.4807073954983923
recall:  0.5368043087971275
f1_score:  0.7296822584806398
```

GaussianNB

In [14]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.6800757217226692
precision:  0.44495837187789083
recall:  0.8635547576301615
f1_score:  0.6988599745961804
```

MultinomialNB

In [15]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.6800757217226692
precision:  0.44495837187789083
recall:  0.8635547576301615
f1_score:  0.6988599745961804
```

BernoulliNB

In [16]:
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = BernoulliNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.7094178892569806
precision:  0.47141424272818455
recall:  0.8438061041292639
f1_score:  0.7266304660238987
```

KNeighborsClassifier

In [17]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = KNeighborsClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("accuracy: ",accuracy_score(y_test, y_pred))
print("precision: ",precision_score(y_test, y_pred))
print("recall: ",recall_score(y_test, y_pred))
print("f1_score: ",f1_score(y_test, y_pred, average='weighted'))
```

```
accuracy:  0.7657359204921912
precision:  0.5563636363636364
recall:  0.5493716337522442
f1_score:  0.7652581286536987
```

XGBClassifier

SG Boost

In [19]:
```
y_train=y_train.replace("Yes", 1).replace("No", 0)
y_test=y_test.replace("Yes", 1).replace("No", 0)
```

In [20]:
```
#%pip install xgboost
from xgboost import XGBClassifier

# declare parameters
params = {
            'objective':'binary:logistic',
            'max_depth': 4,
            'alpha': 10,
            'learning_rate': 1.0,
            'n_estimators':100
        }
# instantiate the classifier
xgb_clf = XGBClassifier(**params)
# fit the classifier to the training data
xgb_clf.fit(X_train, y_train)
y_pred = xgb_clf.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1556
           1       0.61      0.54      0.57       557

    accuracy                           0.79      2113
   macro avg       0.73      0.71      0.72      2113
weighted avg       0.78      0.79      0.78      2113
```