Here, I followed the classification alogrithm in machine learning :

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. K-Nearest Neighbors
5. AdaBoost Classifier
6. Gradient Boost Classifier
7. Extra Tree Classifier

I started by importing all the necessary libraries:

**import pandas as pd  
import numpy as np  
import seaborn as snsfrom sklearn.preprocessing import OneHotEncoder, OrdinalEncoder  
from sklearn.preprocessing import StandardScaler  
from sklearn.neighbors import KNeighborsClassifier**  
**from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifierfrom sklearn.ensemble import AdaBoostClassifier  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import GradientBoostingClassifierfrom sklearn.pipeline import make\_pipelinefrom sklearn.metrics import accuracy\_score  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from sklearn.metrics import classification\_report, confusion\_matrix**

Use the code the pd.set\_option("display.max\_coulmns", None) to display all the columns. Then I loaded the csv file containing 7043 rows (customers) and 21 columns (features).

**df = pd.read\_csv("Customer-Churn.csv")**

***Note:****Exploratory Data Analysis includes data visualisation. So, before learning about Machine Learning, I recommend that you read my post on Customer-Churn Analysis.*[*Click here to read my Medium post on Customer-Churn-Analysis*](https://medium.com/@uqba2199/customer-churn-analysis-using-python-acd4fd6a1712)*.*

Let’s Start with Feature Engineering

**Feature Engineering and Selection**

Feature engineering is a machine learning technique that uses data to generate new variables that were not present in the training set. It has the potential to generate new features for both supervised and unsupervised learning, with the goal of simplifying and speeding up data transformations while also improving model accuracy.

The main feature engineering techniques that will be discussed are:

1. Missing data imputation

2. Categorical encoding

3. Variable transformation

4. Outlier engineering

5. Date and time engineering

**Advantages of feature engineering**

* **Improves Accuracy**: With less misleading data, modelling accuracy improves.
* **Reduces Overfitting**: Less unnesccesary data means fewer opportunities to make noise-based decisions.
* **Reduces Training Time:**Fewer data points reduce algorithm problem, allowing algorithms to train more quickly.

**No modification**

The SeniorCitizen column is already binary and should not be changed.

**Ordinal Encoding**

Ordinal encoding **converts each label into integer values and the encoded data represents the sequence of labels**.

**cols = ['gender','Partner','Dependents','PhoneService',   
 'MultipleLines','InternetService','OnlineSecurity',  
 'OnlineBackup','DeviceProtection','TechSupport',  
 'StreamingTV','StreamingMovies','Contract','TotalCharges',  
 'PaperlessBilling','PaymentMethod']ord = OrdinalEncoder()  
ord.fit(df[cols])  
df[cols] = ord.transform(df[cols])df.head()**

**One-Hot Encoding:**

One hot encoding is one method of **converting data** to prepare it for an algorithm and get a better prediction.

**Churn\_ohe = OneHotEncoder(drop='first', sparse=False, dtype=np.int32)  
Churn\_dummies = Churn\_ohe.fit\_transform(df[['Churn']])  
df.drop(columns=['Churn'],inplace=True)df = pd.concat([df, pd.DataFrame(Churn\_dummies)], axis=1)**

**Splitting the data in training and testing sets**

Train/Test is a method to measure the accuracy of your model. *Train* the model means *create* the model. You *test* the model using the testing set. 7**0% for training, and 30% for testing.**



First, we create a variable **X** to store the dataset’s **independent attributes**. In addition, we define a variable **y** to hold only the **target variable**.

**X = df.drop(columns = ['Churn'])  
y = df['Churn'].values**

Then, from the **sklearn.model\_selection** package, we can use the **train\_test\_split** function to generate both the training and testing sets.

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,   
 test\_size= 0.30, random\_state=1)print('X\_train:',len(X\_train))  
print('X\_test:',len(X\_test))  
print('y\_train',len(y\_train))  
print('y\_test',len(y\_test)**)

Output: **X\_train : 4930, X\_test : 2113, y\_train: 4930, y\_test: 2113**

**Compare several machine learning models on a performance metric**

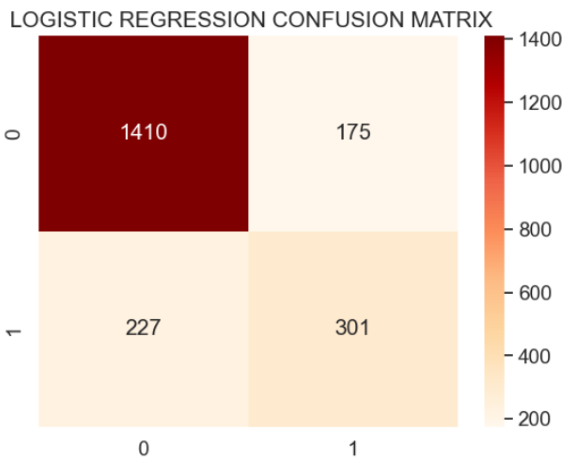
I used classification algorithm to predict our model:

**Logistic Regression:**

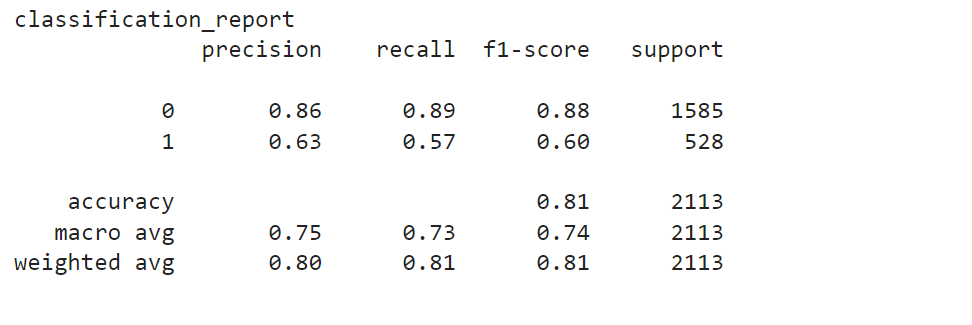
**model = make\_pipeline(StandardScaler(),LogisticRegression())  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
accuracy = model.score(X\_test,y\_test)  
print("Logistic Regression accuracy is :",accuracy)**



**cf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cf\_matrix , annot=True,fmt = "d", cmap='OrRd')  
plt.title("LOGISTIC REGRESSION CONFUSION MATRIX");**



**print("classification\_report")  
print(classification\_report(y\_test, y\_pred))**



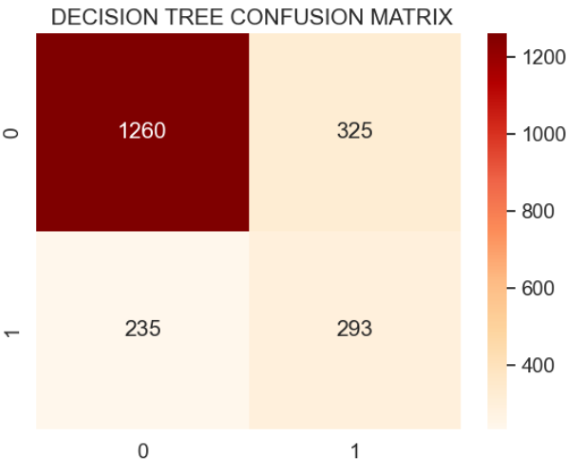
**Decision Tree:**

* DecisionTreeClassifier

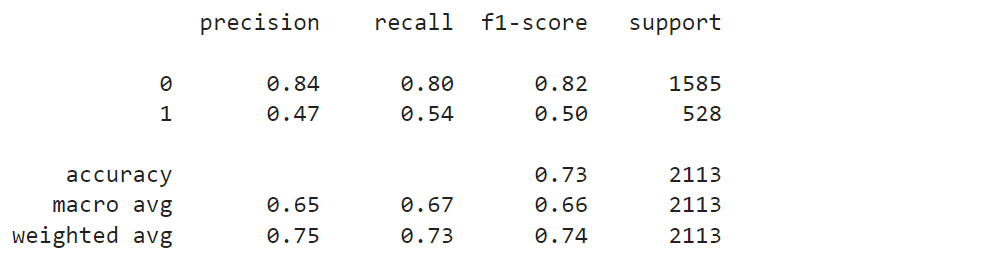
**model = make\_pipeline(StandardScalar(),DecisionTreeClassifier())  
model.fit(X\_train,y\_train)  
y\_pred = model.predict(X\_test)  
accuracy = model.score(X\_test,y\_test)  
print("Decision Tree accuracy is :",accuracy)**



**cf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cf\_matrix , annot=True,fmt = "d", cmap='OrRd')  
plt.title("DECISION TREE CONFUSION MATRIX");**



**print(classification\_report(y\_test, y\_pred))**



Decision tree gives very low score.

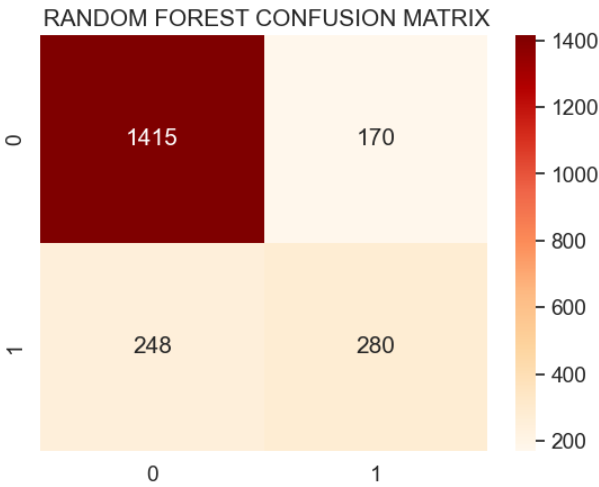
**Random Forest:**

* Random Forest Classifier

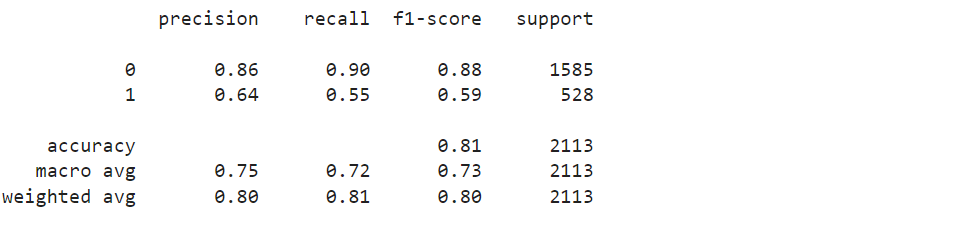
**model = make\_pipeline(StandardScalar(), RandomForestClassifier())  
model.fit(X\_train,y\_train)  
y\_pred = model.predict(X\_test)  
accuracy = model.score(X\_test,y\_test)  
print("Random forest accuracy :",accuracy)**



**cf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cf\_matrix , annot=True,fmt = "d", cmap='OrRd')  
plt.title("RANDOM FOREST** **CONFUSION MATRIX");**



**print(classification\_report(y\_test, prediction\_test))**

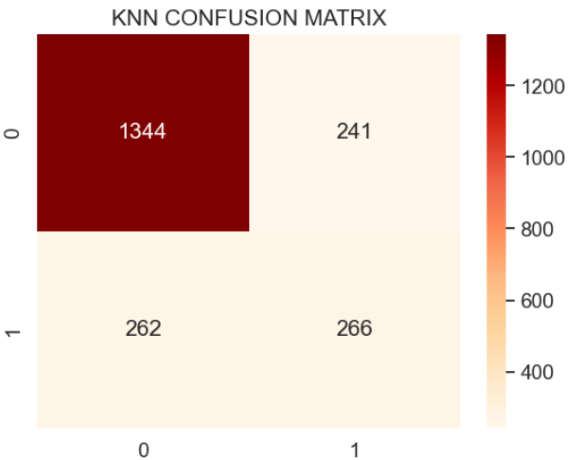


**K-Nearest Neighbors:**

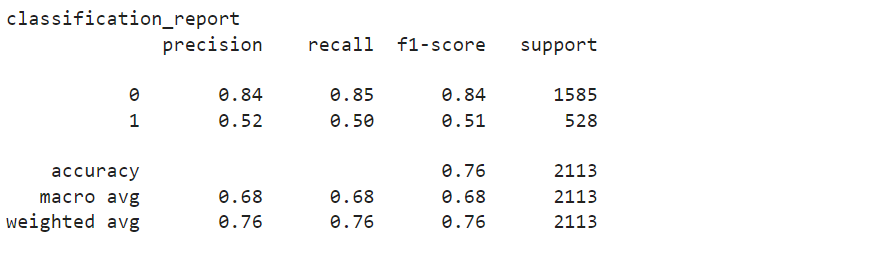
**model = make\_pipeline(StandardScaler(),KNeighborsClassifier())  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
accuracy = model.score(X\_test, y\_test)  
print("K-Nearest Neighbors: ", accuracy)**



**cf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cf\_matrix , annot=True,fmt = "d", cmap='OrRd')  
plt.title("K-NEAREST NEIGHBORS CLASSIFIER** **CONFUSION MATRIX");**



**print("classification\_report")  
print(classification\_report(y\_test, y\_pred))**

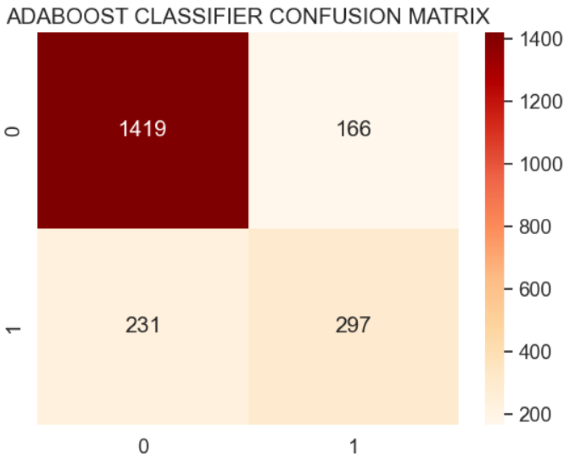


**AdaBoost Classifier:**

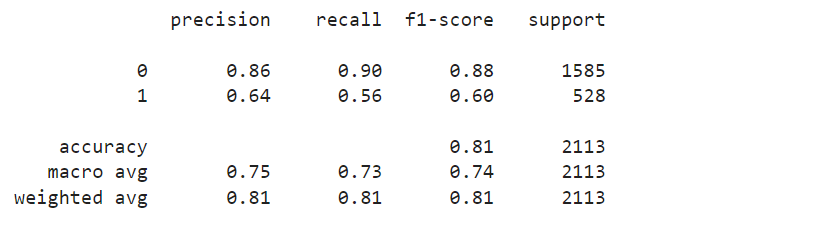
**model = make\_pipeline(StandardScaler(), AdaBoostClassifier())  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
accuracy = model.score(X\_test, y\_test)  
print("AdaBoost Classifier accuracy :",accuracy)**



**cf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cf\_matrix , annot=True,fmt = "d", cmap='OrRd')  
plt.title("ADABOOST CLASSIFIER CONFUSION** **CONFUSION MATRIX");**



**print("classification\_report")  
print(classification\_report(y\_test, y\_pred))**



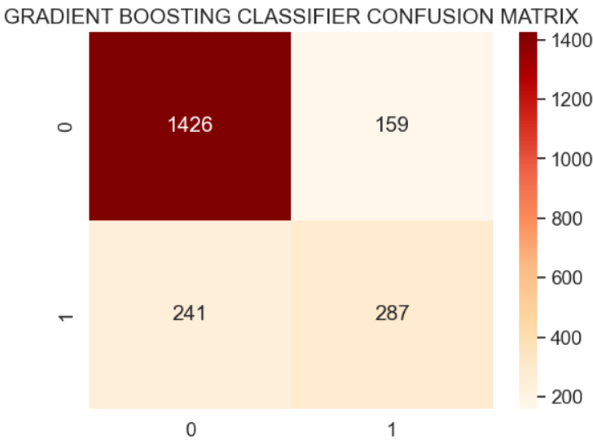
AdaBoost Classifier accuracy is quite good.

**Gradient Boosting Classifier:**

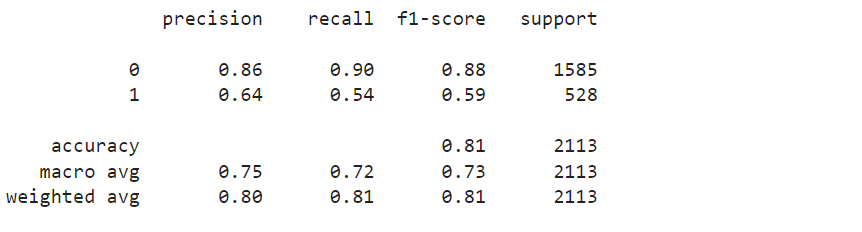
**model = GradientBoostingClassifier()  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
print("Gradient Boosting Classifier", accuracy\_score(y\_test, y\_pred))**



**cf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cf\_matrix , annot=True,fmt = "d", cmap='OrRd')  
plt.title("GRADIENT BOOSTING CLASSIFIER** **CONFUSION MATRIX")**



**print("classification\_report")  
print(classification\_report(y\_test, y\_pred))**



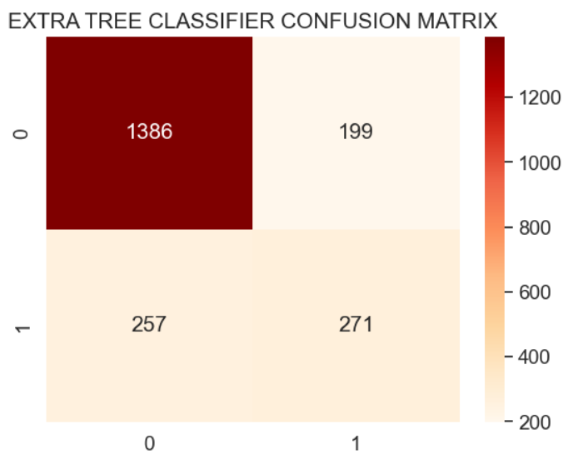
As shown above, we obtain a **sensitivity**of 0.89(1426/(1426+159)) and a **specificity** of 0.64(287/(287+241)). The model obtained predicts more accurately customers that do not churn. because gradient boosting classifiers tend to favour classes with more observations.

**Extra Tree Classifier:**

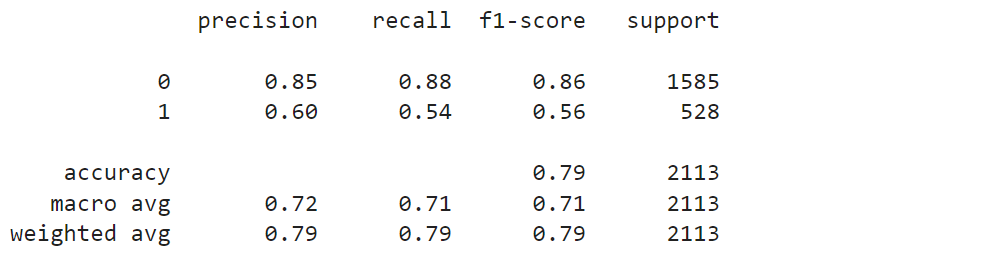
**model = make\_pipeline(StandardScaler(), ExtraTreesClassifier())  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
print("Extra Trees Classifier Score :", accuracy\_score(y\_test, y\_pred))**



**cf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cf\_matrix , annot=True,fmt = "d", cmap='OrRd')  
plt.title("EXTRA TREE CLASSIFIER CONFUSION MATRIX");**



**print(classification\_report(y\_test, y\_pred))**



**Result:** We tried 7 different machine learning algorithms using default parameters.Finally, we tuned the **Gradient Boosting Classifier** (best performance model) for model optimization, obtaining an**accuracy of nearly 80%**. So, at the end of this project, we have a classification model that can correctly predict 77.84% of churning clients.

* *A confusion matrix would be useful for determining the churners.*
* *Experiment with more complex Machine Learning algorthms, such as Xgboost, and fine-tuning the hyper parameters gives more results.*