CSCI-415 (Final Project):

Customer Churn Prediction

Submitted to Dr. Ghada Khoriba

Team members:

1 – Nour Nasser **ID: 19105525**

2 – Abdelrahman Hassan **ID: 19104406**

3 – Ahmed Mohamed Hashem **ID: 19105073**

4 – Ammar Mahmoud Hamed **ID: 19105685**

```
import pandas as pd
import missingno as msno
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
```

This code imports several libraries that are commonly used for data manipulation, data visualization, machine learning, and model evaluation.

- **`pandas`**: used for data manipulation and analysis
- **`missingno`**: used for visualizing missing data
- **`numpy`**: used for numerical computing
- **`seaborn`**: used for data visualization
- **`matplotlib.pyplot`**: used for data visualization
- **`sklearn.preprocessing.LabelEncoder`**: used for encoding categorical variables
- **`sklearn.preprocessing.StandardScaler`**: used for scaling numerical variables
- **`sklearn.linear_model.LogisticRegression`**: used for training a logistic regression model
- **`sklearn.metrics.confusion_matrix`**: used for generating a confusion matrix to evaluate model performance
- **`sklearn.metrics.accuracy_score`**: used for calculating the accuracy of the model

Dropping Unwanted Columns

```
: df.drop(['Customer ID','Churn Category','Churn Reason',
           'Total Refunds','Zip Code','Longitude','Latitude','City'],axis=1, inplace = True)
```

This code removes several columns from the Pandas DataFrame **`df`**. The **`drop`** method is used to remove the specified columns, which are listed in a list passed to the **`columns`** parameter. The **`axis`** parameter specifies that the operation is performed on columns (as opposed to rows), and the **`inplace`** parameter specifies that the changes are made in place, meaning that the original DataFrame is modified rather than a copy.

The removed columns are:

- **`Customer ID`**
- **`Churn Category`**

- **Churn Reason**
- **Total Refunds**
- **Zip Code**
- **Longitude**
- **Latitude**
- **City**

```
missing_values = list(df.isna().sum())
# missing values is a list of the number of missing values in each column

cols = list(df.columns)
col_final = []
for i in range(len(cols)):
    if (missing_values[i] == 0):
        cols[i]="Others"
d = dict(zip(cols, missing_values)) # making a dicionary for the missing values

print("Number of Missing Values per feature >>")
missing_vals = pd.DataFrame(d, index=["Missing Values"]) # Making a custom dataframe from dict d
missing_vals.head()
```

This code creates a list **missing_values** containing the number of missing values in each column of a Pandas DataFrame **df**. The list **cols** is created, containing the names of the columns in **df**.A loop iterates over the list **cols** and, for each column, checks if the corresponding element in **missing_values** is equal to 0. If it is, the name of the column is replaced with the string "Others" in **cols**. The lists **cols** and **missing_values** are then zipped into a dictionary **d**, and a Pandas DataFrame **missing_vals** is created from this dictionary using the **pd.DataFrame** constructor. The index of the DataFrame is set to the string "Missing Values". The resulting DataFrame displays the number of missing values for each column.

Handling missing values using the ffill method

```
df.fillna(method='ffill', inplace=True)
```

```
missing_values = list(df.isna().sum())
# missing values is a list of the number of missing values in each column

cols = list(df.columns)
col_final = []
for i in range(len(cols)):
    if (missing_values[i] == 0):
        cols[i]="Others"
d = dict(zip(cols, missing_values)) # making a dicionary for the missing values

print("Number of Missing Values per feature >>")
missing_vals = pd.DataFrame(d, index=["Missing Values"]) # Making a custom dataframe from dict d
missing_vals.head()
```

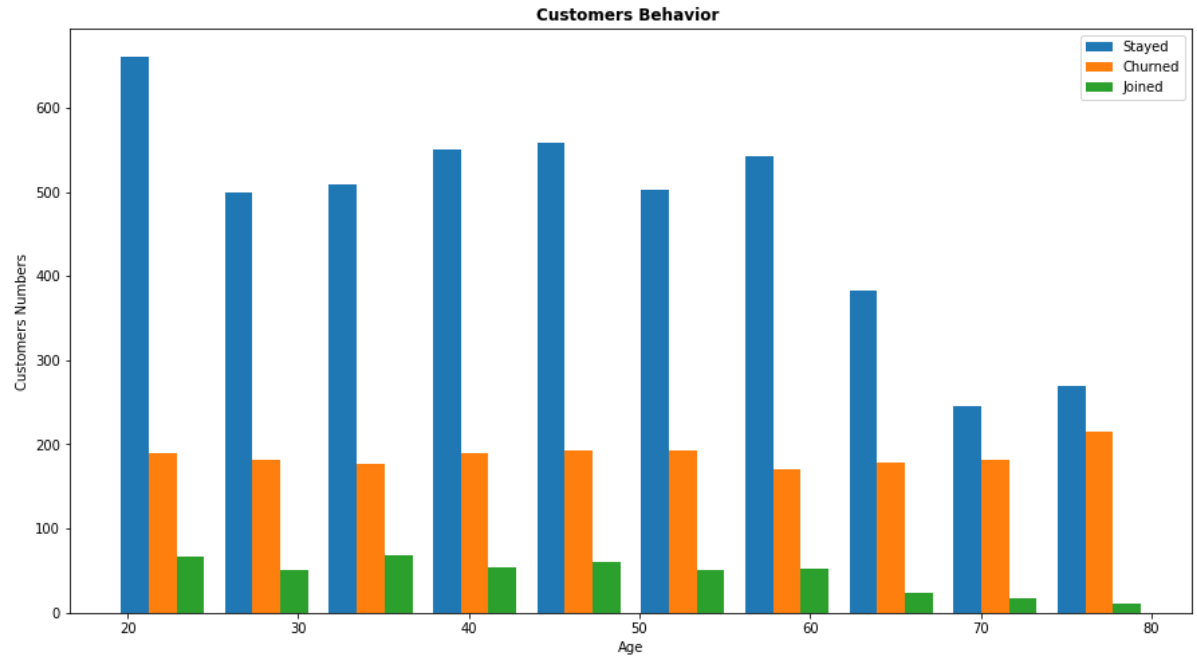
This code fills in missing values in a Pandas DataFrame `df` using the `ffill` method. The `ffill` method stands for "forward fill" and propagates the previous value forward to fill in the missing values.

The `fillna` method is used to fill the missing values in `df`. The `method` parameter specifies the filling method to use, and the `inplace` parameter specifies that the changes should be made in place, meaning that the original DataFrame is modified rather than a copy.

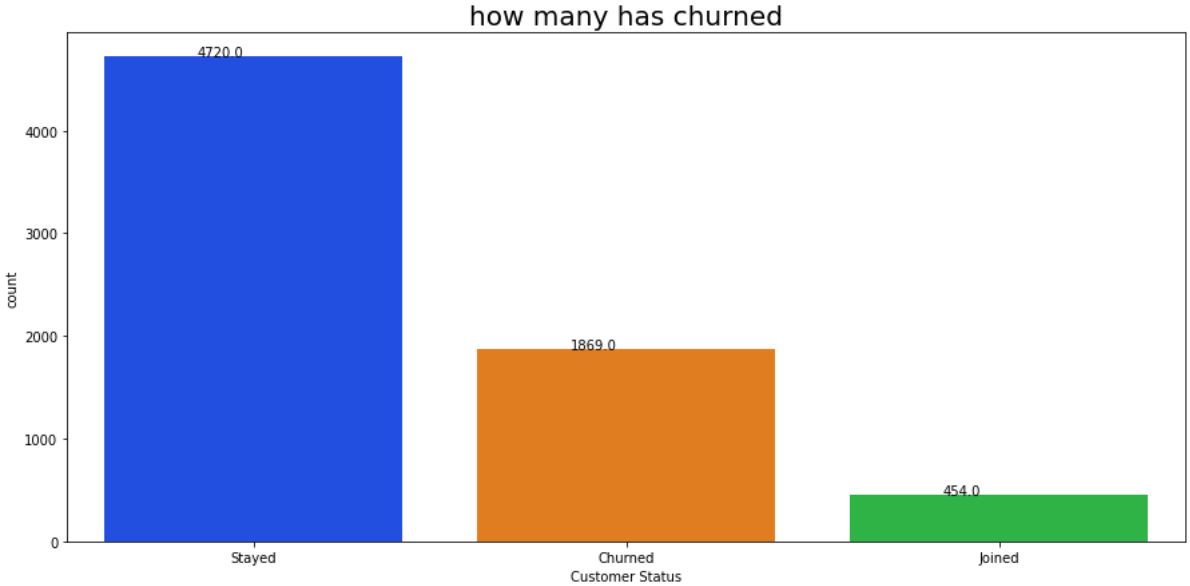
The code then creates a list `missing_values` containing the number of missing values in each column of `df`, as well as a list `cols` containing the names of the columns in `df`.

A loop iterates over the list `cols` and, for each column, checks if the corresponding element in `missing_values` is equal to 0. If it is, the name of the column is replaced with the string "Others" in `cols`.

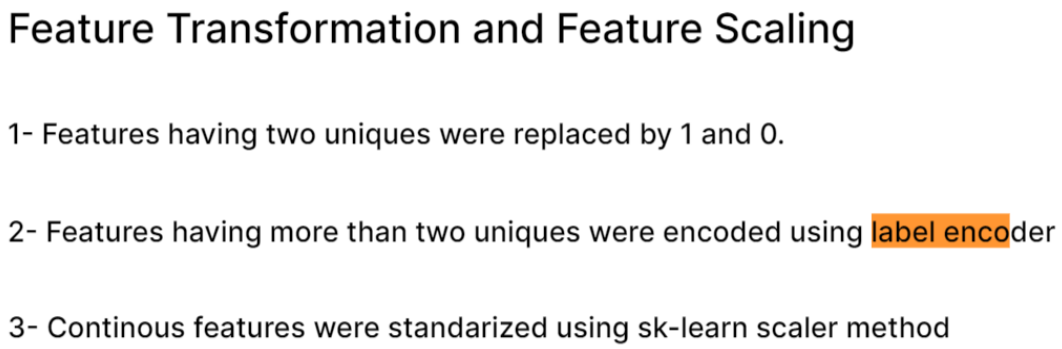
The lists `cols` and `missing_values` are then zipped into a dictionary `d`, and a Pandas DataFrame `missing_vals` is created from this dictionary using the `pd.DataFrame` constructor. The index of the DataFrame is set to the string "Missing Values". The resulting DataFrame displays the number of missing values for each column.



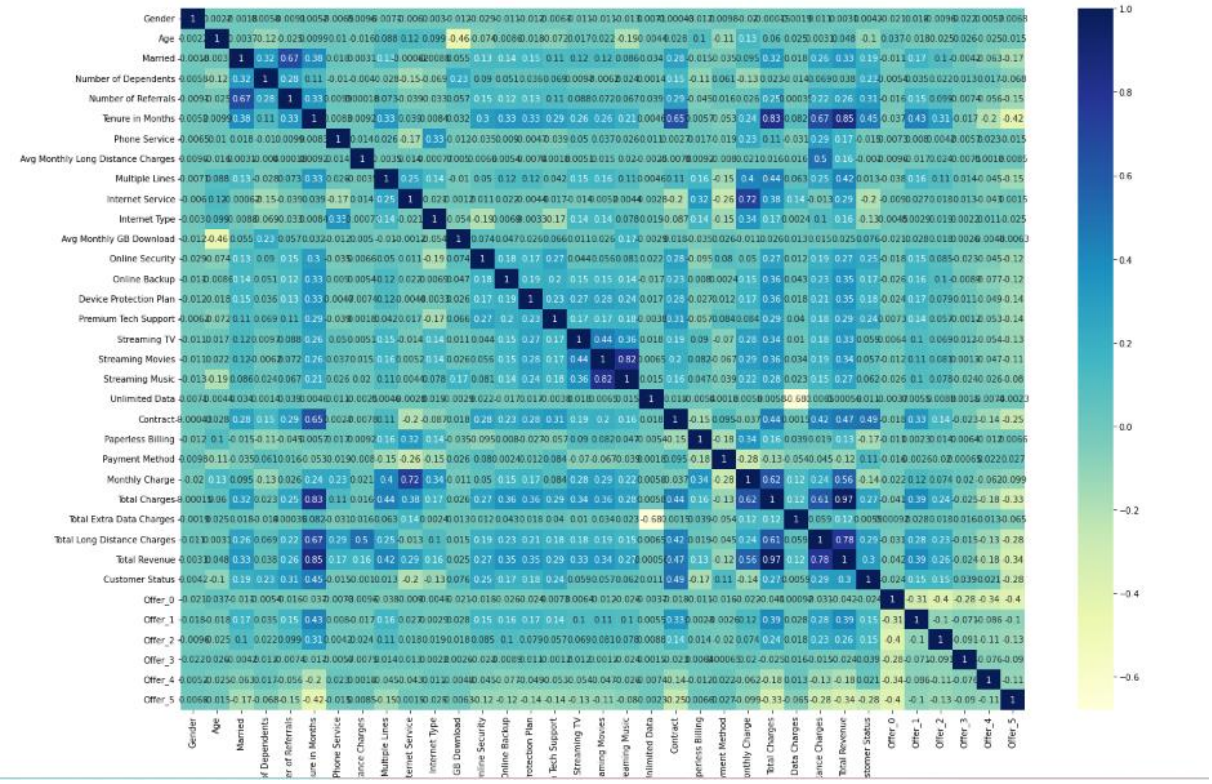
This figure represents customer's behavior on different age categories, and as we can see the most customers who stayed are in the category of 20-30. And the churn is almost distributed equally among different categories. "Joined" class decreases when age increases.



This figure represents that most of the customers have stayed, and only 1869 out of 7000 have churned.



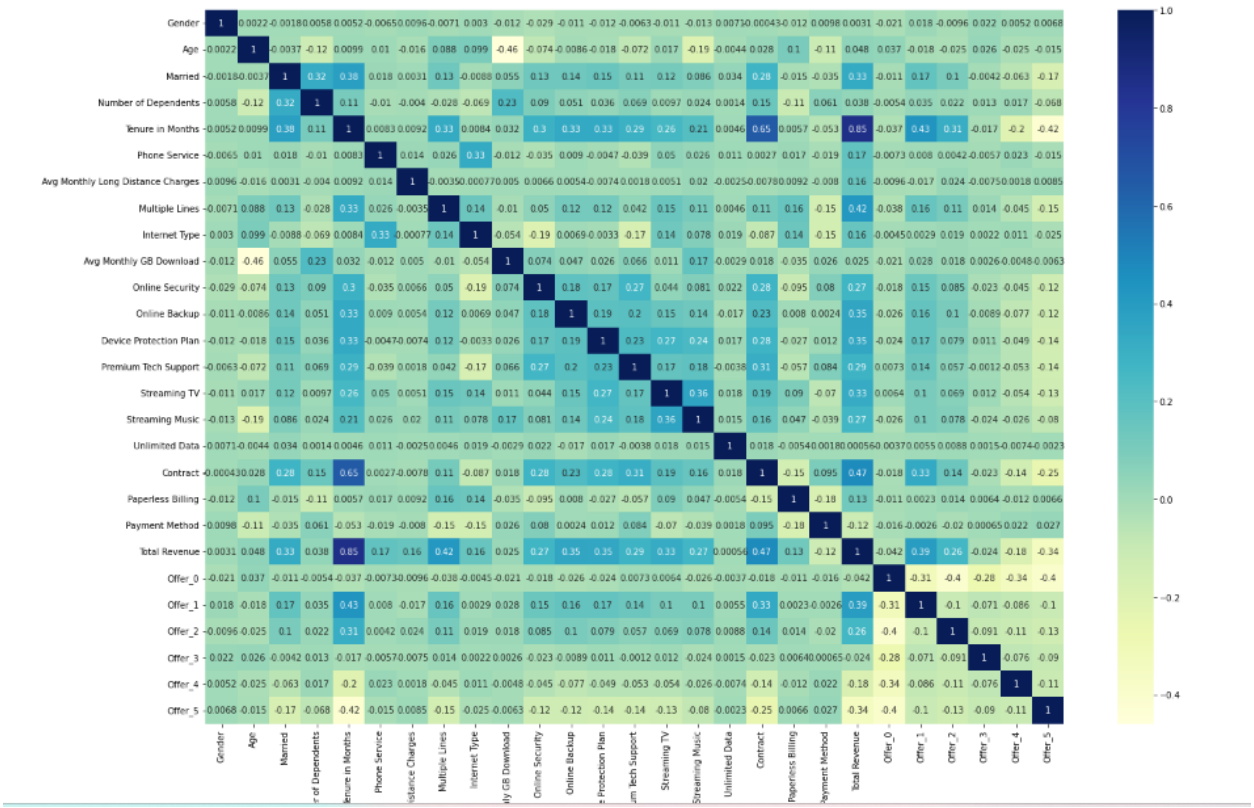
We encoded using different methods depending on features for example the features “married” and “gender” have two unique values so we replace them by one’s and zero’s. Other features like “offer” has more than two unique values so we use the getdummies method. Other method was used on features that is continuous like “age”so we scaled the data using standard scalar.



This correlation matrix shows the features that are highly correlated to each other and helps us in feature selection.

```
drop_col = ['Customer Status','Internet Service','Total Charges','Monthly Charge','Streaming Movies','Number of Referrals','Total Extra Data Charges',"Total Long Distance
x = df.drop(drop_col, axis = 1)
y = df['Customer Status']
```

Due to the correlation matrix above we had to remove the features that are highly correlated due to having the same weight and effect on the model.



This is the Correlation matrix after feature extraction.

Splitting our data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state = 0)
```

The above code splits our dataset into training sample 75%, and test sample 25%.

Using standard scaler

```
scaler = StandardScaler()
cols = ['Age','Avg Monthly Long Distance Charges','Avg Monthly GB Download','Total Revenue']
X_train[cols] = StandardScaler().fit_transform(X_train[cols])
X_test[cols] = StandardScaler().fit_transform(X_test[cols])
```

We have scaled our continuous data from a range of almost -1 to 1.

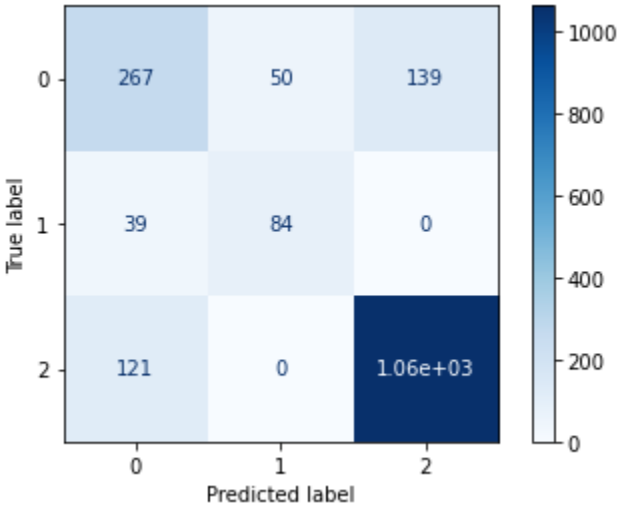
After preparing the data for modeling, we start building the models.

First model is:

Decision Tree, we used certain parameters upon building the model and these parameters are; max depth with value equal to, min samples leaf with value equal to 3, min samples split with value equal to 5.

Output of the decision tree shows that the training accuracy is equal to 0.848208864602373 and test accuracy of 0.8598300970873787.

Here is an image of confusion matrix of decision tree:

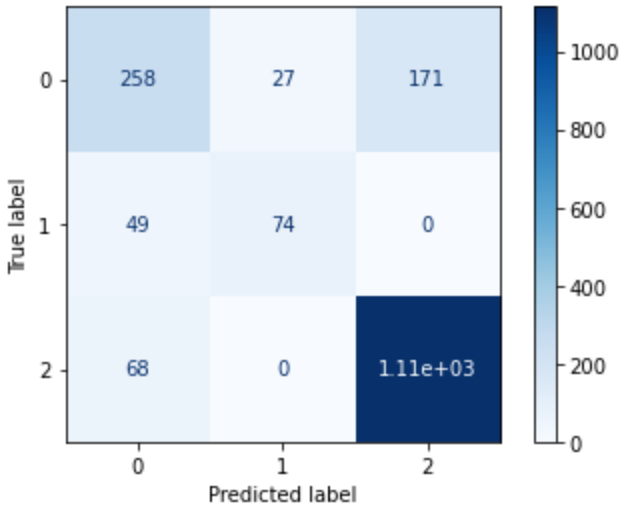


Second model:

Random forest, we used certain parameters upon building the model and these parameters are max depth which is equal to 10, random state is equal to 0, n estimators is equal to 25.

Output of the random forest model shows an accuracy of 0.8211243611584327.

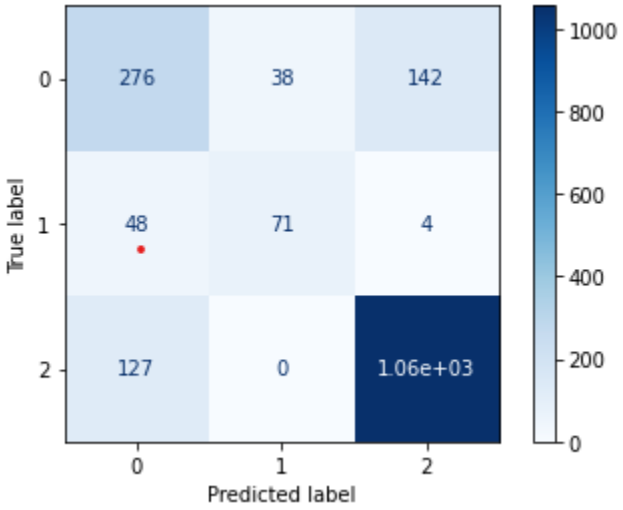
Here is an image of confusion matrix of random forest:



Third model:

Logistic regression model, shows an accuracy of 0.80

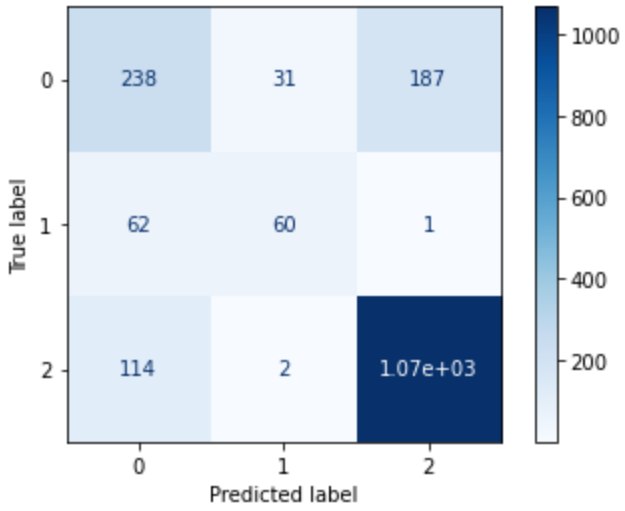
Here is an image of confusion matrix of logistic regression model



Fourth model:

Knn(k-nearest neighbors), with 7 nearest neighbor shows an accuracy of 0.7864077669902912.

Here is an image of confusion matrix of Knn model:



After Applying PCA to our models it shows different accuracies:

1 – Logistic regression with accuracy: 0.7910278250993753

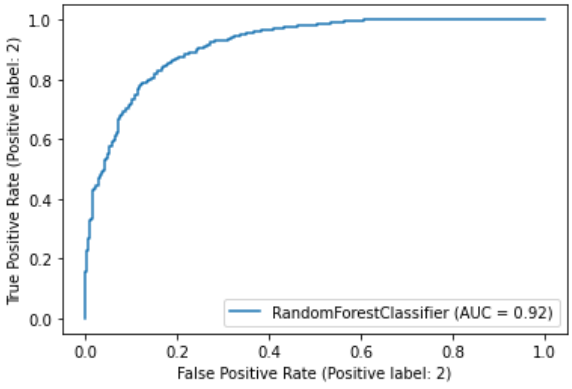
2 – Knn with accuracy: 0.720045428733674

3 – Random Forest with accuracy: 0.7961385576377058

Transforming our target to binary class:

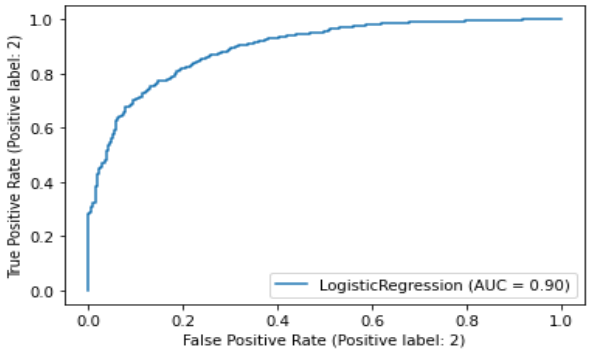
1 - Random forest with binary class shows an accuracy of 0.87257281553

And shows an AUC score of 0.92



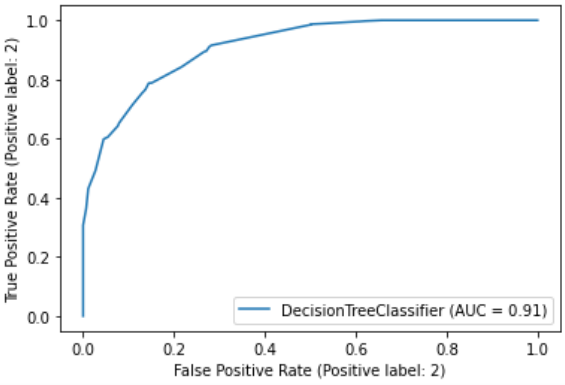
2 – Logistic regression with binary class shows an accuracy of 0.834344

And shows an AUC score of 0.90



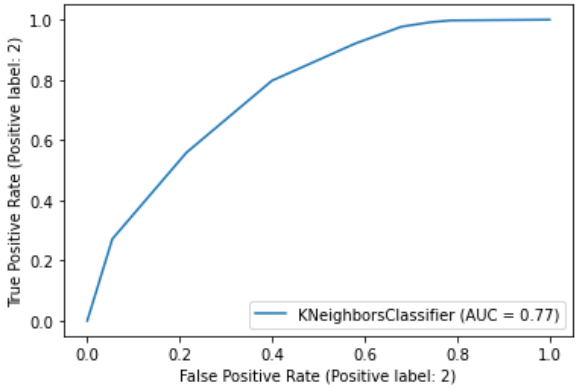
3 – Decision tree with binary class shows an accuracy of 0.85983009

And shows an AUC score of 0.91

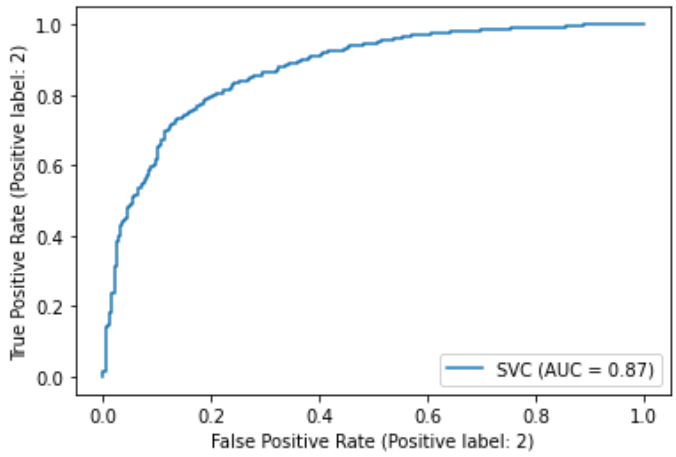


4 – Knn with binary class shows an accuracy of 0.78640

And shows an AUC score of 0.77



SVM model which is considered as a binary classifier shows an accuracy of 0.82766 and shows an AUC score of 0.87



After training four different models; random forest, decision tree, logistic regression, and Knn. Random forest shows the highest accuracy.