**Intelligent Customer Retention: Using Machine Learning for Enhanced Prediction of Telecom Customer Churn**

**INTRODUCTION**

**.Overview**

**Abstract:**

This project discusses the issue of customer churn, which is the rate at which customers are lost by companies. It is a major concern for large companies, particularly in the telecom industry, due to its direct impact on revenues. Companies are seeking to predict potential churn by analyzing data from previous churns, which may be caused by various reasons such as better price offers, more interesting packages, bad service experiences or change in customers’ personal situations. The project explores the importance of customer churn for companies in the current competitive market and the need for efficient churn predictive models to build cost-effective marketing strategies and prevent losses. The use of machine learning models in the telecom industry to identify probable churn customers and make necessary business decisions is also discussed. The project emphasizes the importance of early identification of customers likely to leave to limit losses and retain customers.

**Key words: Random Forest, flight price prediction, KNN, decision trees.**

**1.2 Purpose**

**Bisiness problem:**

The business problem addressed in this project is the issue of customer churn, which is the rate at which customers are lost by companies, particularly in the telecom industry. The problem is significant as it directly impacts the revenues of companies and is caused by various factors such as better price offers, more interesting packages, bad service experiences or change in customers’ personal situations. Companies are seeking to predict potential churn by analyzing data from previous churns and developing efficient churn predictive models to build cost-effective marketing strategies and prevent losses. The aim is to retain customers and limit the number of customers likely to leave, thereby increasing the company's revenue and maintaining its market position in the face of stiff competition.

telecom industry. It is challenging to predict which customers are likely to leave, as there may be various reasons behind the churn. The reasons could be related to better price offers, more attractive packages from competitors, bad service experiences, or changes in customers' personal situations. As a result, companies need to develop efficient churn predictive models that can analyze data from previous churns and identify patterns and trends that can help predict potential churn.

The development of churn predictive models is crucial for companies as it can help them build cost-effective marketing strategies to retain customers and limit losses. By identifying customers who are likely to leave, companies can tailor their marketing campaigns to target these customers with personalized offers, rewards, or discounts that will encourage them to stay. Additionally, the development of churn predictive models can help companies optimize their marketing budgets by focusing their retention efforts on customers who are most likely to leave.

The ultimate goal of solving the customer churn problem is to increase the company's revenue and maintain its market position in the face of stiff competition. Companies that can retain their customers are more likely to succeed and grow in the long run, as retaining customers is much less expensive than acquiring new ones. Therefore, the development of efficient churn predictive models is essential for companies to stay competitive and achieve their business goals.

* 1. **Business requirement:**

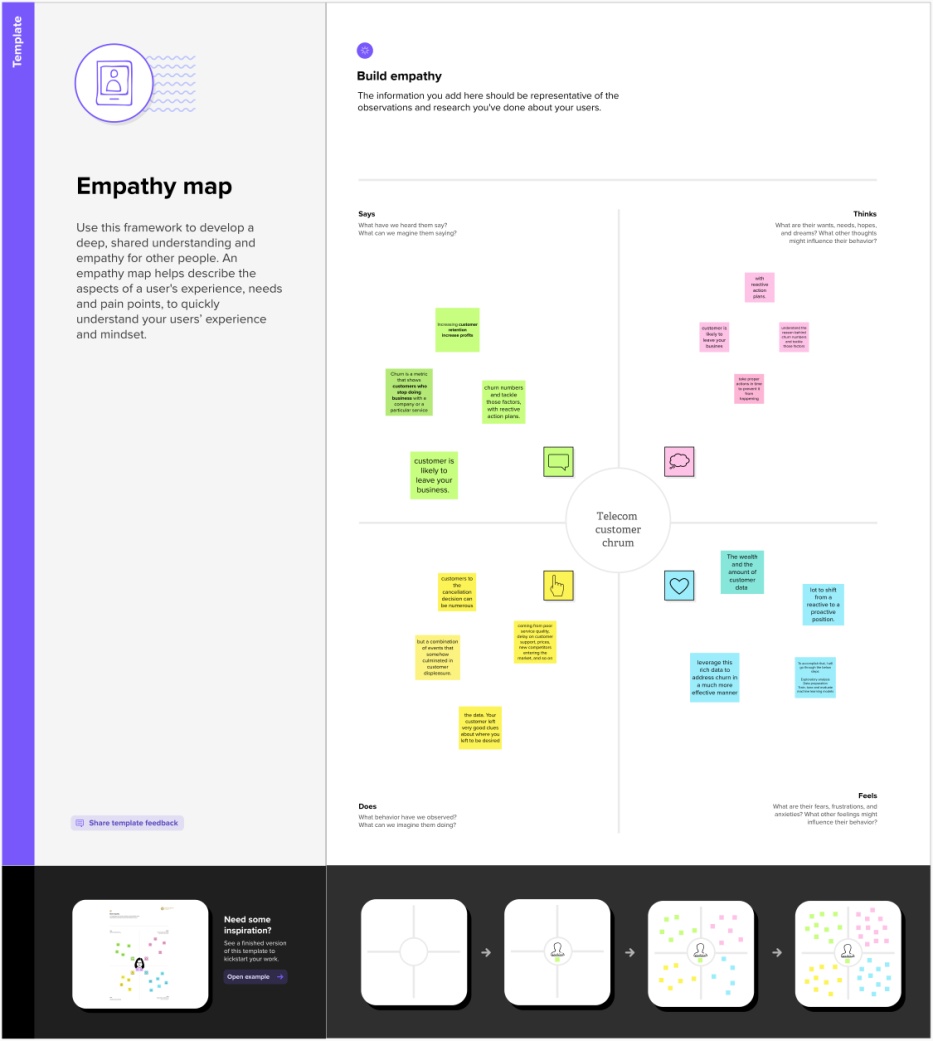
The business requirement for addressing the problem of customer churn is to develop efficient churn predictive models that can analyze customer data and predict potential churn. Companies need to collect customer data such as past purchase history, loyalty program status, demographic information, and service usage patterns to develop personalized pricing and offers that can attract and retain customers.

The churn predictive model should be able to analyze the customer data and identify patterns and trends that are indicative of potential churn. The model should be able to generate accurate predictions and alert the company when customers are at risk of leaving, allowing the company to take timely action to retain the customer.

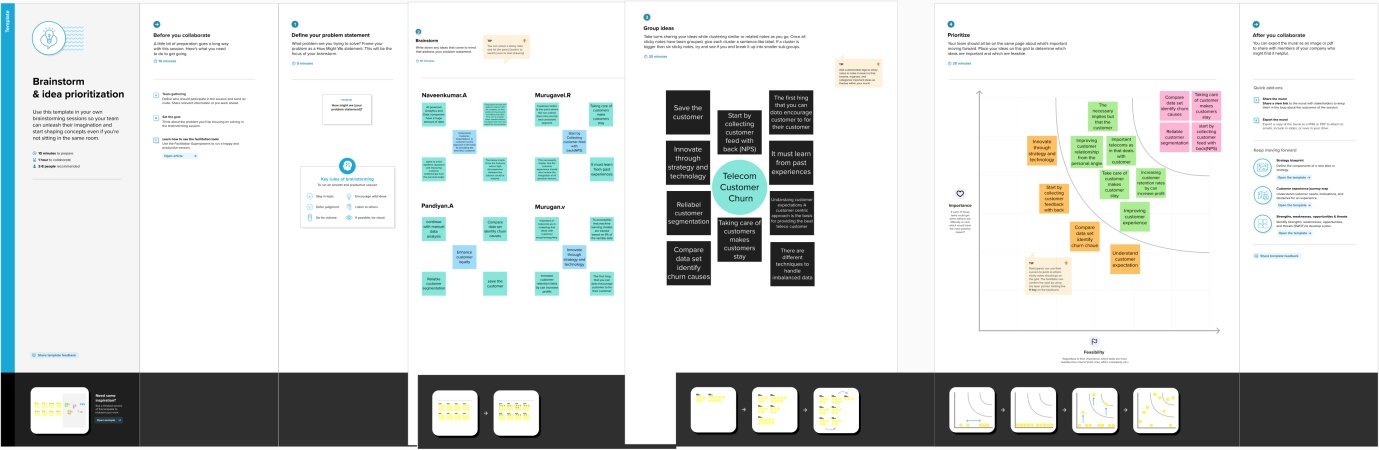
The churn predictive model should also be scalable, able to analyze large amounts of data quickly, and provide actionable insights to the company. The model should be easy to use and integrate with existing customer relationship management systems, allowing companies to streamline their retention efforts.

**2.Problem Definition & Design Thinking**

**2.1 Empathy Map**

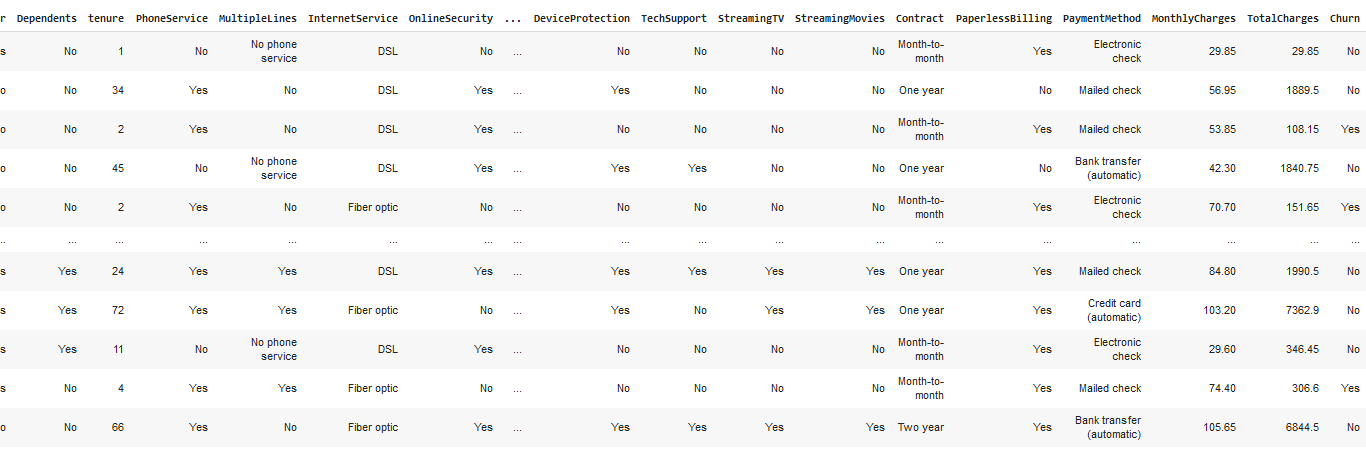
****

**2.2 Ideation &Brinstorming map**



**RESULT**

**Result:1**

****

**Result:2**

# 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

**Result:3**

customerID False

gender False

SeniorCitizen False

Partner False

Dependents False

tenure False

PhoneService False

MultipleLines False

InternetService False

OnlineSecurity False

OnlineBackup False

DeviceProtection False

TechSupport False

StreamingTV False

StreamingMovies False

Contract False

PaperlessBilling False

PaymentMethod False

MonthlyCharges False

TotalCharges True

Churn False

dtype: bool

**Result:4**

 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

**Result:5**

****

**Result:6**

array([[0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 2.00000e+00,

            2.98500e+01, 2.98500e+01],

           [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 3.00000e+00,

            5.69500e+01, 1.88950e+03],

           [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 3.00000e+00,

            5.38500e+01, 1.08150e+02],

           ...,

           [1.00000e+00, 0.00000e+00, 1.00000e+00, ..., 3.00000e+00,

            9.66500e+01, 1.16285e+03],

           [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 2.00000e+00,

            4.03500e+01, 1.67785e+03],

           [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 2.00000e+00,

            1.88500e+01, 1.88500e+01]])

**Result:7**

array([[0],

           [0],

           [1],

           ...,

           [0],

           [1],

           [0]])

**Result:8**

array([[0.00000000e+00, 1.00000000e+00, 0.00000000e+00, ...,

            1.00000000e+00, 2.98500000e+01, 2.98500000e+01],

           [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,

            0.00000000e+00, 5.69500000e+01, 1.88950000e+03],

           [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,

            1.00000000e+00, 5.38500000e+01, 1.08150000e+02],

           ...,

           [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, ...,

            1.00000000e+00, 8.38882223e+01, 2.02727523e+02],

           [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,

            1.00000000e+00, 4.56651578e+01, 4.56651578e+01],

           [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,

            1.00000000e+00, 7.63709751e+01, 1.42031260e+02]])

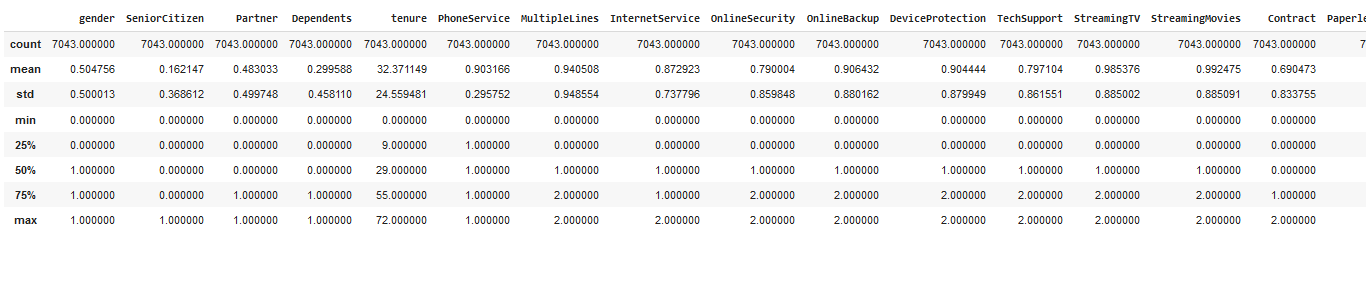
**Result:9**

 array([0, 0, 1, ..., 1, 1, 1])

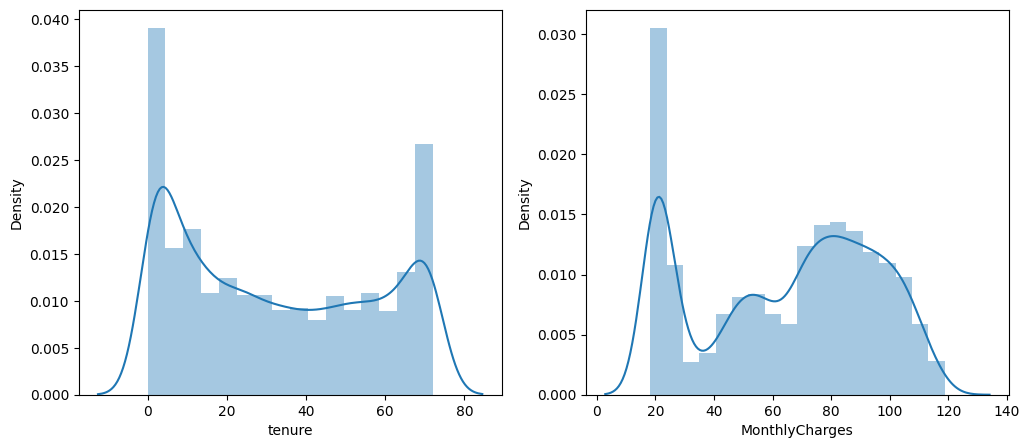
 ((1152, 40), (1704, 40))

 ((1152, 1), (1704,))

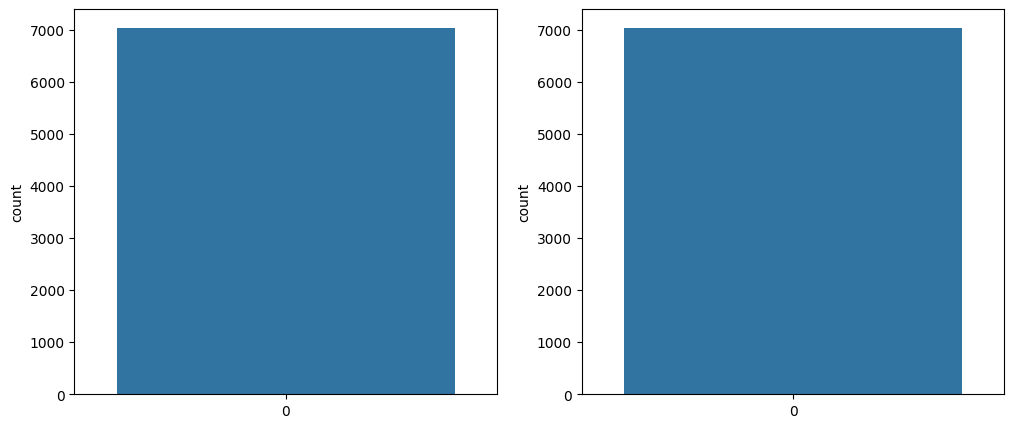
**Result:10**

****

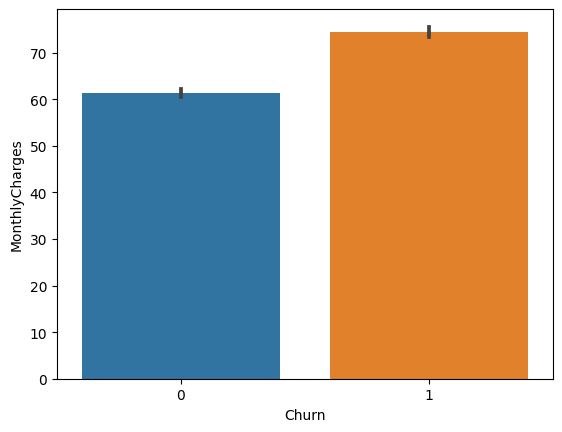
**Result:11**

****

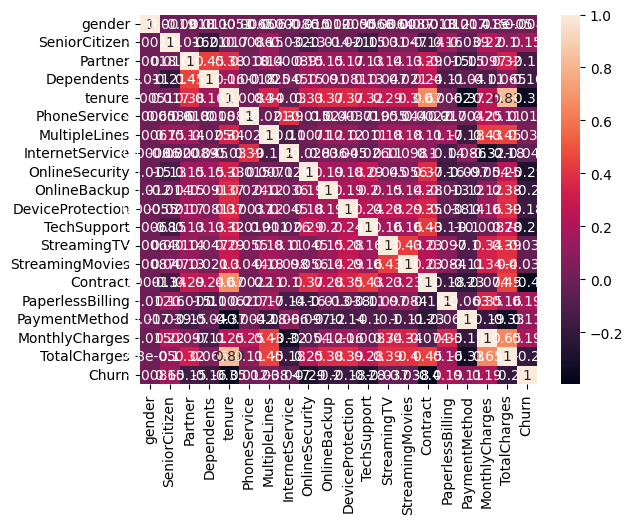
**Result:12**

****

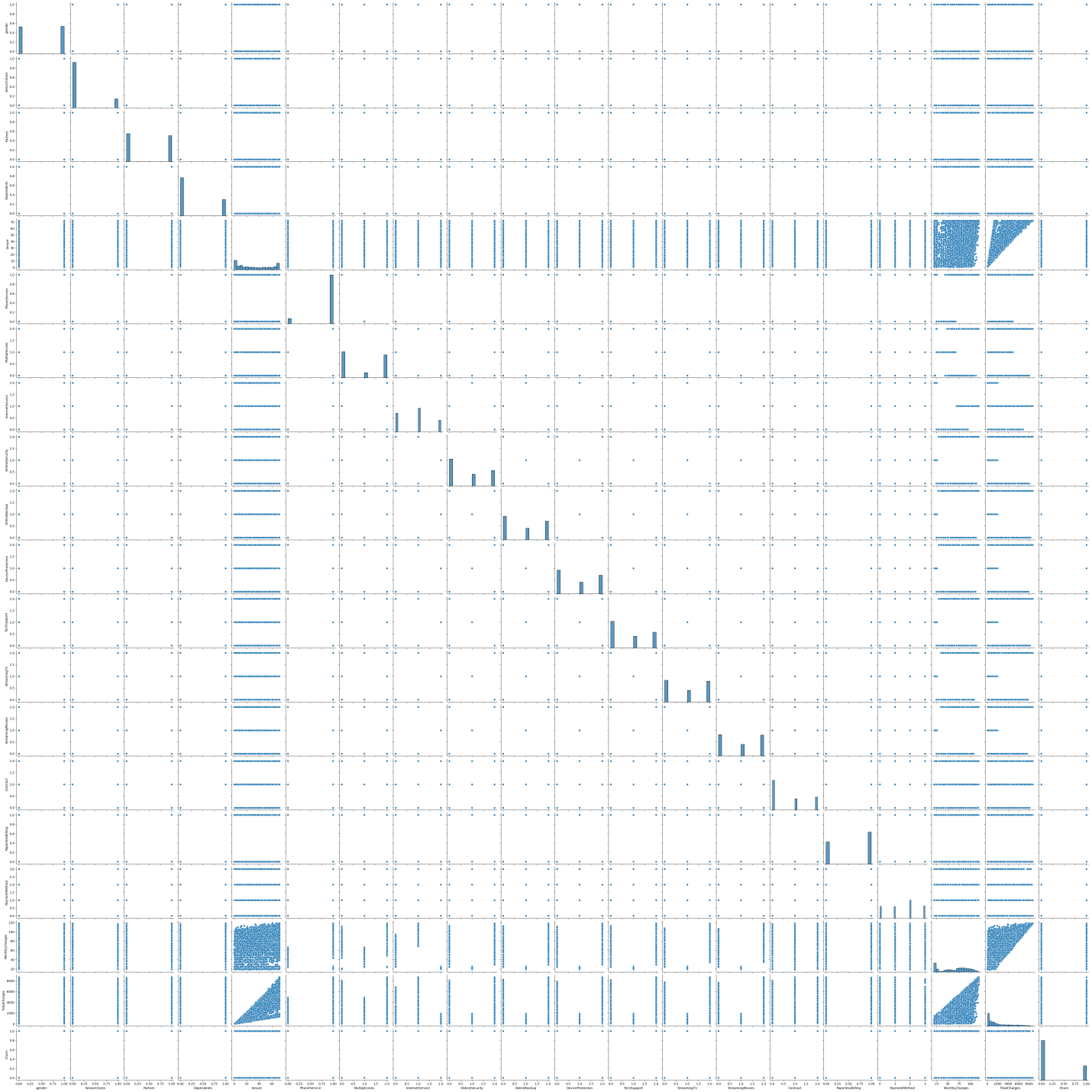
**Result:13**

****

**Result:14**

****

**Result:15**

****

**Result:16**

    (1363, 40)

**Result:17**

Accuracy Score : 0.8202494497432135

    Accuracy Test : 0.7419354838709677

    Logistic Regression

    Confusion Matrix

    [[116  41]

     [ 47 137]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.71      0.74      0.72       157

               1       0.77      0.74      0.76       184

        accuracy                           0.74       341

       macro avg       0.74      0.74      0.74       341

    weighted avg       0.74      0.74      0.74       341

**Result:18**

Accuracy Score : 1.0

    Accuracy Test : 0.7624633431085044

    Decsion Tree

    Confusion Matrix

    [[100  57]

     [ 24 160]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.81      0.64      0.71       157

               1       0.74      0.87      0.80       184

        accuracy                           0.76       341

       macro avg       0.77      0.75      0.75       341

    weighted avg       0.77      0.76      0.76       341

**Result:19**

    Accuracy Score : 0.9926632428466617

    Accuracy Test : 0.8035190615835777

    Random Forest

    Confusion Matrix

    [[108  49]

     [ 18 166]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.86      0.69      0.76       157

               1       0.77      0.90      0.83       184

        accuracy                           0.80       341

       macro avg       0.81      0.80      0.80       341

    weighted avg       0.81      0.80      0.80       341

**Result:20**

 Accuracy Score : 0.8459280997798972

    Accuracy Test : 0.7800586510263929

    KNN

    Confusion Matrix

    [[ 98  59]

     [ 16 168]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.86      0.62      0.72       157

               1       0.74      0.91      0.82       184

        accuracy                           0.78       341

       macro avg       0.80      0.77      0.77       341

    weighted avg       0.80      0.78      0.77       341

**Result:21**

Accuracy Score : 0.8459280997798972

    Accuracy Test : 0.7800586510263929

    SVM

    Confusion Matrix

    [[ 98  59]

     [ 16 168]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.86      0.62      0.72       157

               1       0.74      0.91      0.82       184

        accuracy                           0.78       341

       macro avg       0.80      0.77      0.77       341

    weighted avg       0.80      0.78      0.77       341

**Result:22**

Stream

    Epoch 1/200

    92/92 [==============================] - 2s 7ms/step - loss: 0.5555 - accuracy: 0.7306 - val\_loss: 0.4308 - val\_accuracy: 0.8244

    Epoch 2/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.4423 - accuracy: 0.7930 - val\_loss: 0.4006 - val\_accuracy: 0.8378

    Epoch 3/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.4167 - accuracy: 0.8171 - val\_loss: 0.3878 - val\_accuracy: 0.8467

    Epoch 4/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.4001 - accuracy: 0.8291 - val\_loss: 0.3837 - val\_accuracy: 0.8511

    Epoch 5/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.3856 - accuracy: 0.8313 - val\_loss: 0.3774 - val\_accuracy: 0.8444

    Epoch 6/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.3736 - accuracy: 0.8379 - val\_loss: 0.3737 - val\_accuracy: 0.8511

    Epoch 7/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.3634 - accuracy: 0.8456 - val\_loss: 0.3697 - val\_accuracy: 0.8467

    Epoch 8/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.3542 - accuracy: 0.8434 - val\_loss: 0.3693 - val\_accuracy: 0.8378

    Epoch 9/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.3450 - accuracy: 0.8554 - val\_loss: 0.3602 - val\_accuracy: 0.8489

    Epoch 10/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.3333 - accuracy: 0.8642 - val\_loss: 0.3605 - val\_accuracy: 0.8511

    Epoch 11/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.3196 - accuracy: 0.8751 - val\_loss: 0.3667 - val\_accuracy: 0.8511

    Epoch 12/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.3127 - accuracy: 0.8784 - val\_loss: 0.3543 - val\_accuracy: 0.8489

    Epoch 13/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.3015 - accuracy: 0.8806 - val\_loss: 0.3545 - val\_accuracy: 0.8378

    Epoch 14/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2925 - accuracy: 0.8839 - val\_loss: 0.3524 - val\_accuracy: 0.8578

    Epoch 15/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2849 - accuracy: 0.8905 - val\_loss: 0.3544 - val\_accuracy: 0.8489

    Epoch 16/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2736 - accuracy: 0.8970 - val\_loss: 0.3534 - val\_accuracy: 0.8578

    Epoch 17/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2621 - accuracy: 0.8959 - val\_loss: 0.3618 - val\_accuracy: 0.8622

    Epoch 18/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2552 - accuracy: 0.9014 - val\_loss: 0.3680 - val\_accuracy: 0.8533

    Epoch 19/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2465 - accuracy: 0.9168 - val\_loss: 0.3618 - val\_accuracy: 0.8533

    Epoch 20/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2372 - accuracy: 0.9080 - val\_loss: 0.3574 - val\_accuracy: 0.8578

    Epoch 21/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2274 - accuracy: 0.9146 - val\_loss: 0.3772 - val\_accuracy: 0.8533

    Epoch 22/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2187 - accuracy: 0.9168 - val\_loss: 0.3668 - val\_accuracy: 0.8644

    Epoch 23/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2079 - accuracy: 0.9332 - val\_loss: 0.3649 - val\_accuracy: 0.8556

    Epoch 24/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2040 - accuracy: 0.9310 - val\_loss: 0.3851 - val\_accuracy: 0.8533

    Epoch 25/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.2029 - accuracy: 0.9244 - val\_loss: 0.3688 - val\_accuracy: 0.8622

    Epoch 26/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1883 - accuracy: 0.9376 - val\_loss: 0.3770 - val\_accuracy: 0.8644

    Epoch 27/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1811 - accuracy: 0.9343 - val\_loss: 0.3823 - val\_accuracy: 0.8622

    Epoch 28/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1759 - accuracy: 0.9365 - val\_loss: 0.3932 - val\_accuracy: 0.8622

    Epoch 29/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.1684 - accuracy: 0.9387 - val\_loss: 0.3924 - val\_accuracy: 0.8667

    Epoch 30/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1718 - accuracy: 0.9398 - val\_loss: 0.4008 - val\_accuracy: 0.8467

    Epoch 31/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1559 - accuracy: 0.9518 - val\_loss: 0.3981 - val\_accuracy: 0.8556

    Epoch 32/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1553 - accuracy: 0.9441 - val\_loss: 0.4027 - val\_accuracy: 0.8489

    Epoch 33/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1511 - accuracy: 0.9485 - val\_loss: 0.3983 - val\_accuracy: 0.8644

    Epoch 34/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1416 - accuracy: 0.9562 - val\_loss: 0.4065 - val\_accuracy: 0.8644

    Epoch 35/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1362 - accuracy: 0.9540 - val\_loss: 0.4120 - val\_accuracy: 0.8533

    Epoch 36/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1353 - accuracy: 0.9595 - val\_loss: 0.3978 - val\_accuracy: 0.8556

    Epoch 37/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1369 - accuracy: 0.9518 - val\_loss: 0.4189 - val\_accuracy: 0.8467

    Epoch 38/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1257 - accuracy: 0.9628 - val\_loss: 0.4193 - val\_accuracy: 0.8600

    Epoch 39/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1223 - accuracy: 0.9617 - val\_loss: 0.4295 - val\_accuracy: 0.8489

    Epoch 40/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1157 - accuracy: 0.9650 - val\_loss: 0.4323 - val\_accuracy: 0.8444

    Epoch 41/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1132 - accuracy: 0.9650 - val\_loss: 0.4461 - val\_accuracy: 0.8444

    Epoch 42/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1083 - accuracy: 0.9682 - val\_loss: 0.4446 - val\_accuracy: 0.8511

    Epoch 43/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1157 - accuracy: 0.9606 - val\_loss: 0.4404 - val\_accuracy: 0.8422

    Epoch 44/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.1045 - accuracy: 0.9715 - val\_loss: 0.4454 - val\_accuracy: 0.8444

    Epoch 45/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0983 - accuracy: 0.9759 - val\_loss: 0.4465 - val\_accuracy: 0.8444

    Epoch 46/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0969 - accuracy: 0.9693 - val\_loss: 0.4520 - val\_accuracy: 0.8622

    Epoch 47/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0953 - accuracy: 0.9704 - val\_loss: 0.4778 - val\_accuracy: 0.8333

    Epoch 48/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0950 - accuracy: 0.9704 - val\_loss: 0.4763 - val\_accuracy: 0.8467

    Epoch 49/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0893 - accuracy: 0.9748 - val\_loss: 0.4503 - val\_accuracy: 0.8489

    Epoch 50/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0871 - accuracy: 0.9715 - val\_loss: 0.4697 - val\_accuracy: 0.8467

    Epoch 51/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0867 - accuracy: 0.9748 - val\_loss: 0.4836 - val\_accuracy: 0.8556

    Epoch 52/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0839 - accuracy: 0.9748 - val\_loss: 0.4742 - val\_accuracy: 0.8622

    Epoch 53/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0819 - accuracy: 0.9759 - val\_loss: 0.4758 - val\_accuracy: 0.8422

    Epoch 54/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0741 - accuracy: 0.9792 - val\_loss: 0.4917 - val\_accuracy: 0.8467

    Epoch 55/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0754 - accuracy: 0.9803 - val\_loss: 0.4906 - val\_accuracy: 0.8400

    Epoch 56/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0721 - accuracy: 0.9803 - val\_loss: 0.4886 - val\_accuracy: 0.8578

    Epoch 57/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0670 - accuracy: 0.9781 - val\_loss: 0.5004 - val\_accuracy: 0.8467

    Epoch 58/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0705 - accuracy: 0.9770 - val\_loss: 0.5095 - val\_accuracy: 0.8578

    Epoch 59/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0667 - accuracy: 0.9814 - val\_loss: 0.4955 - val\_accuracy: 0.8533

    Epoch 60/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0650 - accuracy: 0.9781 - val\_loss: 0.4914 - val\_accuracy: 0.8622

    Epoch 61/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0647 - accuracy: 0.9814 - val\_loss: 0.4927 - val\_accuracy: 0.8533

    Epoch 62/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0597 - accuracy: 0.9825 - val\_loss: 0.4999 - val\_accuracy: 0.8533

    Epoch 63/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0622 - accuracy: 0.9847 - val\_loss: 0.5012 - val\_accuracy: 0.8556

    Epoch 64/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0569 - accuracy: 0.9792 - val\_loss: 0.5200 - val\_accuracy: 0.8644

    Epoch 65/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0598 - accuracy: 0.9836 - val\_loss: 0.5220 - val\_accuracy: 0.8467

    Epoch 66/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0551 - accuracy: 0.9858 - val\_loss: 0.5223 - val\_accuracy: 0.8578

    Epoch 67/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0559 - accuracy: 0.9825 - val\_loss: 0.5739 - val\_accuracy: 0.8400

    Epoch 68/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0617 - accuracy: 0.9825 - val\_loss: 0.5361 - val\_accuracy: 0.8578

    Epoch 69/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0532 - accuracy: 0.9847 - val\_loss: 0.5654 - val\_accuracy: 0.8489

    Epoch 70/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0512 - accuracy: 0.9847 - val\_loss: 0.5398 - val\_accuracy: 0.8600

    Epoch 71/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0491 - accuracy: 0.9858 - val\_loss: 0.5794 - val\_accuracy: 0.8422

    Epoch 72/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0577 - accuracy: 0.9792 - val\_loss: 0.5484 - val\_accuracy: 0.8600

    Epoch 73/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0472 - accuracy: 0.9847 - val\_loss: 0.5689 - val\_accuracy: 0.8644

    Epoch 74/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0481 - accuracy: 0.9825 - val\_loss: 0.5683 - val\_accuracy: 0.8600

    Epoch 75/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0463 - accuracy: 0.9880 - val\_loss: 0.5912 - val\_accuracy: 0.8622

    Epoch 76/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0448 - accuracy: 0.9847 - val\_loss: 0.5725 - val\_accuracy: 0.8600

    Epoch 77/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0453 - accuracy: 0.9847 - val\_loss: 0.5795 - val\_accuracy: 0.8578

    Epoch 78/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0462 - accuracy: 0.9858 - val\_loss: 0.6049 - val\_accuracy: 0.8511

    Epoch 79/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0456 - accuracy: 0.9792 - val\_loss: 0.6344 - val\_accuracy: 0.8578

    Epoch 80/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0430 - accuracy: 0.9858 - val\_loss: 0.5910 - val\_accuracy: 0.8600

    Epoch 81/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0396 - accuracy: 0.9869 - val\_loss: 0.5934 - val\_accuracy: 0.8578

    Epoch 82/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0406 - accuracy: 0.9869 - val\_loss: 0.5906 - val\_accuracy: 0.8622

    Epoch 83/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0377 - accuracy: 0.9836 - val\_loss: 0.6055 - val\_accuracy: 0.8556

    Epoch 84/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0381 - accuracy: 0.9880 - val\_loss: 0.6349 - val\_accuracy: 0.8400

    Epoch 85/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0407 - accuracy: 0.9847 - val\_loss: 0.6203 - val\_accuracy: 0.8556

    Epoch 86/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0411 - accuracy: 0.9847 - val\_loss: 0.6279 - val\_accuracy: 0.8489

    Epoch 87/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0383 - accuracy: 0.9847 - val\_loss: 0.6378 - val\_accuracy: 0.8533

    Epoch 88/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0411 - accuracy: 0.9858 - val\_loss: 0.6444 - val\_accuracy: 0.8422

    Epoch 89/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0406 - accuracy: 0.9869 - val\_loss: 0.6633 - val\_accuracy: 0.8533

    Epoch 90/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0370 - accuracy: 0.9869 - val\_loss: 0.6388 - val\_accuracy: 0.8556

    Epoch 91/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0341 - accuracy: 0.9890 - val\_loss: 0.6690 - val\_accuracy: 0.8378

    Epoch 92/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0399 - accuracy: 0.9869 - val\_loss: 0.6745 - val\_accuracy: 0.8467

    Epoch 93/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0390 - accuracy: 0.9847 - val\_loss: 0.6719 - val\_accuracy: 0.8556

    Epoch 94/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0365 - accuracy: 0.9858 - val\_loss: 0.6526 - val\_accuracy: 0.8622

    Epoch 95/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0404 - accuracy: 0.9858 - val\_loss: 0.7069 - val\_accuracy: 0.8667

    Epoch 96/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0391 - accuracy: 0.9858 - val\_loss: 0.6704 - val\_accuracy: 0.8489

    Epoch 97/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0405 - accuracy: 0.9847 - val\_loss: 0.6728 - val\_accuracy: 0.8533

    Epoch 98/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0345 - accuracy: 0.9890 - val\_loss: 0.6730 - val\_accuracy: 0.8511

    Epoch 99/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0418 - accuracy: 0.9814 - val\_loss: 0.7073 - val\_accuracy: 0.8467

    Epoch 100/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0461 - accuracy: 0.9792 - val\_loss: 0.7135 - val\_accuracy: 0.8489

    Epoch 101/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0352 - accuracy: 0.9847 - val\_loss: 0.6969 - val\_accuracy: 0.8644

    Epoch 102/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0351 - accuracy: 0.9814 - val\_loss: 0.7215 - val\_accuracy: 0.8511

    Epoch 103/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0318 - accuracy: 0.9880 - val\_loss: 0.7107 - val\_accuracy: 0.8533

    Epoch 104/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0333 - accuracy: 0.9869 - val\_loss: 0.7443 - val\_accuracy: 0.8422

    Epoch 105/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0345 - accuracy: 0.9869 - val\_loss: 0.7308 - val\_accuracy: 0.8444

    Epoch 106/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0379 - accuracy: 0.9880 - val\_loss: 0.7325 - val\_accuracy: 0.8511

    Epoch 107/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0344 - accuracy: 0.9858 - val\_loss: 0.7271 - val\_accuracy: 0.8511

    Epoch 108/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0377 - accuracy: 0.9858 - val\_loss: 0.7103 - val\_accuracy: 0.8600

    Epoch 109/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0313 - accuracy: 0.9890 - val\_loss: 0.7469 - val\_accuracy: 0.8511

    Epoch 110/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0381 - accuracy: 0.9869 - val\_loss: 0.7330 - val\_accuracy: 0.8556

    Epoch 111/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0316 - accuracy: 0.9880 - val\_loss: 0.7374 - val\_accuracy: 0.8711

    Epoch 112/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0319 - accuracy: 0.9858 - val\_loss: 0.7438 - val\_accuracy: 0.8556

    Epoch 113/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0381 - accuracy: 0.9869 - val\_loss: 0.7150 - val\_accuracy: 0.8600

    Epoch 114/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0320 - accuracy: 0.9847 - val\_loss: 0.7393 - val\_accuracy: 0.8756

    Epoch 115/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0301 - accuracy: 0.9912 - val\_loss: 0.7712 - val\_accuracy: 0.8356

    Epoch 116/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0368 - accuracy: 0.9825 - val\_loss: 0.8155 - val\_accuracy: 0.8622

    Epoch 117/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0376 - accuracy: 0.9836 - val\_loss: 0.7680 - val\_accuracy: 0.8667

    Epoch 118/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0385 - accuracy: 0.9858 - val\_loss: 0.7671 - val\_accuracy: 0.8644

    Epoch 119/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0297 - accuracy: 0.9869 - val\_loss: 0.7981 - val\_accuracy: 0.8600

    Epoch 120/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0332 - accuracy: 0.9890 - val\_loss: 0.8005 - val\_accuracy: 0.8378

    Epoch 121/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0336 - accuracy: 0.9858 - val\_loss: 0.7799 - val\_accuracy: 0.8556

    Epoch 122/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0320 - accuracy: 0.9869 - val\_loss: 0.8021 - val\_accuracy: 0.8467

    Epoch 123/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0291 - accuracy: 0.9858 - val\_loss: 0.8181 - val\_accuracy: 0.8511

    Epoch 124/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0292 - accuracy: 0.9847 - val\_loss: 0.7969 - val\_accuracy: 0.8689

    Epoch 125/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0341 - accuracy: 0.9869 - val\_loss: 0.7959 - val\_accuracy: 0.8600

    Epoch 126/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0313 - accuracy: 0.9858 - val\_loss: 0.8015 - val\_accuracy: 0.8578

    Epoch 127/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0295 - accuracy: 0.9847 - val\_loss: 0.8081 - val\_accuracy: 0.8600

    Epoch 128/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0335 - accuracy: 0.9858 - val\_loss: 0.8078 - val\_accuracy: 0.8511

    Epoch 129/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0314 - accuracy: 0.9836 - val\_loss: 0.8122 - val\_accuracy: 0.8600

    Epoch 130/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0341 - accuracy: 0.9869 - val\_loss: 0.8203 - val\_accuracy: 0.8600

    Epoch 131/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0307 - accuracy: 0.9858 - val\_loss: 0.8047 - val\_accuracy: 0.8533

    Epoch 132/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0303 - accuracy: 0.9890 - val\_loss: 0.8330 - val\_accuracy: 0.8556

    Epoch 133/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0306 - accuracy: 0.9836 - val\_loss: 0.7905 - val\_accuracy: 0.8533

    Epoch 134/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0261 - accuracy: 0.9923 - val\_loss: 0.8108 - val\_accuracy: 0.8711

    Epoch 135/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0429 - accuracy: 0.9847 - val\_loss: 0.8299 - val\_accuracy: 0.8689

    Epoch 136/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0328 - accuracy: 0.9858 - val\_loss: 0.8438 - val\_accuracy: 0.8400

    Epoch 137/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0266 - accuracy: 0.9890 - val\_loss: 0.8083 - val\_accuracy: 0.8556

    Epoch 138/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0290 - accuracy: 0.9880 - val\_loss: 0.8339 - val\_accuracy: 0.8556

    Epoch 139/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0290 - accuracy: 0.9847 - val\_loss: 0.8225 - val\_accuracy: 0.8600

    Epoch 140/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0247 - accuracy: 0.9869 - val\_loss: 0.8281 - val\_accuracy: 0.8667

    Epoch 141/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0241 - accuracy: 0.9890 - val\_loss: 0.8561 - val\_accuracy: 0.8622

    Epoch 142/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0310 - accuracy: 0.9858 - val\_loss: 0.8763 - val\_accuracy: 0.8644

    Epoch 143/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0324 - accuracy: 0.9836 - val\_loss: 0.8536 - val\_accuracy: 0.8444

    Epoch 144/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0306 - accuracy: 0.9869 - val\_loss: 0.8443 - val\_accuracy: 0.8644

    Epoch 145/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0278 - accuracy: 0.9890 - val\_loss: 0.8850 - val\_accuracy: 0.8556

    Epoch 146/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0271 - accuracy: 0.9880 - val\_loss: 0.8606 - val\_accuracy: 0.8578

    Epoch 147/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0295 - accuracy: 0.9847 - val\_loss: 0.8701 - val\_accuracy: 0.8733

    Epoch 148/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0328 - accuracy: 0.9901 - val\_loss: 0.8771 - val\_accuracy: 0.8467

    Epoch 149/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0257 - accuracy: 0.9901 - val\_loss: 0.8572 - val\_accuracy: 0.8644

    Epoch 150/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0286 - accuracy: 0.9880 - val\_loss: 0.8777 - val\_accuracy: 0.8689

    Epoch 151/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0282 - accuracy: 0.9858 - val\_loss: 0.8943 - val\_accuracy: 0.8556

    Epoch 152/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0262 - accuracy: 0.9858 - val\_loss: 0.8792 - val\_accuracy: 0.8556

    Epoch 153/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0293 - accuracy: 0.9869 - val\_loss: 0.9168 - val\_accuracy: 0.8533

    Epoch 154/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0386 - accuracy: 0.9781 - val\_loss: 0.8926 - val\_accuracy: 0.8689

    Epoch 155/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0356 - accuracy: 0.9847 - val\_loss: 0.8974 - val\_accuracy: 0.8400

    Epoch 156/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0262 - accuracy: 0.9869 - val\_loss: 0.8787 - val\_accuracy: 0.8578

    Epoch 157/200

    92/92 [==============================] - 0s 4ms/step - loss: 0.0312 - accuracy: 0.9869 - val\_loss: 0.9036 - val\_accuracy: 0.8644

    Epoch 158/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0286 - accuracy: 0.9858 - val\_loss: 0.9228 - val\_accuracy: 0.8467

    Epoch 159/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0258 - accuracy: 0.9890 - val\_loss: 0.9019 - val\_accuracy: 0.8578

    Epoch 160/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0278 - accuracy: 0.9869 - val\_loss: 0.8992 - val\_accuracy: 0.8622

    Epoch 161/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0323 - accuracy: 0.9825 - val\_loss: 0.9111 - val\_accuracy: 0.8556

    Epoch 162/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0278 - accuracy: 0.9858 - val\_loss: 0.9248 - val\_accuracy: 0.8622

    Epoch 163/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0254 - accuracy: 0.9880 - val\_loss: 0.9054 - val\_accuracy: 0.8511

    Epoch 164/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0255 - accuracy: 0.9880 - val\_loss: 0.9635 - val\_accuracy: 0.8444

    Epoch 165/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0323 - accuracy: 0.9847 - val\_loss: 0.9167 - val\_accuracy: 0.8667

    Epoch 166/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0251 - accuracy: 0.9880 - val\_loss: 0.9700 - val\_accuracy: 0.8444

    Epoch 167/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0282 - accuracy: 0.9901 - val\_loss: 0.9251 - val\_accuracy: 0.8600

    Epoch 168/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0277 - accuracy: 0.9869 - val\_loss: 0.9776 - val\_accuracy: 0.8422

    Epoch 169/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0335 - accuracy: 0.9847 - val\_loss: 0.9572 - val\_accuracy: 0.8644

    Epoch 170/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0337 - accuracy: 0.9869 - val\_loss: 0.9300 - val\_accuracy: 0.8644

    Epoch 171/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0247 - accuracy: 0.9869 - val\_loss: 0.9710 - val\_accuracy: 0.8511

    Epoch 172/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0277 - accuracy: 0.9869 - val\_loss: 0.9153 - val\_accuracy: 0.8622

    Epoch 173/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0338 - accuracy: 0.9869 - val\_loss: 0.9586 - val\_accuracy: 0.8644

    Epoch 174/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0249 - accuracy: 0.9890 - val\_loss: 0.9287 - val\_accuracy: 0.8556

    Epoch 175/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0324 - accuracy: 0.9836 - val\_loss: 0.9396 - val\_accuracy: 0.8511

    Epoch 176/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0280 - accuracy: 0.9901 - val\_loss: 0.9270 - val\_accuracy: 0.8533

    Epoch 177/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0239 - accuracy: 0.9901 - val\_loss: 0.9315 - val\_accuracy: 0.8711

    Epoch 178/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0288 - accuracy: 0.9858 - val\_loss: 0.9185 - val\_accuracy: 0.8644

    Epoch 179/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0251 - accuracy: 0.9890 - val\_loss: 0.9889 - val\_accuracy: 0.8444

    Epoch 180/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0252 - accuracy: 0.9901 - val\_loss: 0.9497 - val\_accuracy: 0.8600

    Epoch 181/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0281 - accuracy: 0.9836 - val\_loss: 0.9486 - val\_accuracy: 0.8489

    Epoch 182/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0267 - accuracy: 0.9880 - val\_loss: 0.9645 - val\_accuracy: 0.8733

    Epoch 183/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0230 - accuracy: 0.9869 - val\_loss: 0.9898 - val\_accuracy: 0.8444

    Epoch 184/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0276 - accuracy: 0.9858 - val\_loss: 1.0053 - val\_accuracy: 0.8467

    Epoch 185/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0250 - accuracy: 0.9880 - val\_loss: 0.9834 - val\_accuracy: 0.8689

    Epoch 186/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0339 - accuracy: 0.9836 - val\_loss: 0.9347 - val\_accuracy: 0.8644

    Epoch 187/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0267 - accuracy: 0.9858 - val\_loss: 0.9904 - val\_accuracy: 0.8622

    Epoch 188/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0337 - accuracy: 0.9825 - val\_loss: 1.0675 - val\_accuracy: 0.8444

    Epoch 189/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0414 - accuracy: 0.9814 - val\_loss: 1.0087 - val\_accuracy: 0.8333

    Epoch 190/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0376 - accuracy: 0.9847 - val\_loss: 0.9839 - val\_accuracy: 0.8467

    Epoch 191/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0278 - accuracy: 0.9858 - val\_loss: 1.0051 - val\_accuracy: 0.8511

    Epoch 192/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0264 - accuracy: 0.9869 - val\_loss: 1.0057 - val\_accuracy: 0.8489

    Epoch 193/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0300 - accuracy: 0.9836 - val\_loss: 0.9950 - val\_accuracy: 0.8444

    Epoch 194/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0308 - accuracy: 0.9858 - val\_loss: 0.9953 - val\_accuracy: 0.8533

    Epoch 195/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0264 - accuracy: 0.9869 - val\_loss: 1.0188 - val\_accuracy: 0.8422

    Epoch 196/200

    92/92 [==============================] - 0s 3ms/step - loss: 0.0249 - accuracy: 0.9912 - val\_loss: 1.0076 - val\_accuracy: 0.8622

    Epoch 197/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0247 - accuracy: 0.9890 - val\_loss: 1.0079 - val\_accuracy: 0.8511

    Epoch 198/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0222 - accuracy: 0.9912 - val\_loss: 0.9891 - val\_accuracy: 0.8511

    Epoch 199/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0263 - accuracy: 0.9847 - val\_loss: 1.0260 - val\_accuracy: 0.8489

    Epoch 200/200

    92/92 [==============================] - 0s 2ms/step - loss: 0.0290 - accuracy: 0.9847 - val\_loss: 1.0507 - val\_accuracy: 0.8400

**Result:23**

  Stream

    11/11 [==============================] - 0s 1ms/step

  text/plain

    array([[False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [False],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [False],

           [False],

           [False],

           [False],

           [False],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [ True],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [ True],

           [False],

           [False],

           [ True],

           [False],

           [False],

           [False],

           [ True]])

**Result:23**

Stream

    Accuracy Test : 0.7947214076246334

    ANN model

    Confusion Matrix

    [[129  28]

     [ 42 142]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.75      0.82      0.79       157

               1       0.84      0.77      0.80       184

        accuracy                           0.79       341

       macro avg       0.79      0.80      0.79       341

    weighted avg       0.80      0.79      0.80       341

**Result:24**

Stream

    Predicting on random input

    output is : [0]

**Result:25**

  Stream

    Predicting on random input

    output is : [0]

**Result:26**

  Stream

    Predicting on random input

    output is : [0]

**Result:27**

  Stream

    Predicting on random input

    output is : [0]

**Result:28**

  Stream

    Predicting on random input

    output is : [0]

**Result:29**

 Predicting on random input

    1/1 [==============================] - 0s 19ms/step

    output is : [[False]]

**Result:30**

  Stream

    Accuracy Score : 0.8202494497432135

    Accuracy Test : 0.7419354838709677

    Logistic Regression

    Confusion Matrix

    [[116  41]

     [ 47 137]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.71      0.74      0.72       157

               1       0.77      0.74      0.76       184

        accuracy                           0.74       341

       macro avg       0.74      0.74      0.74       341

    weighted avg       0.74      0.74      0.74       341

 -----------------------------------------------------------------------------

-

  Accuracy Score : 1.0

    Accuracy Test : 0.7624633431085044

    Decsion Tree

    Confusion Matrix

    [[100  57]

     [ 24 160]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.81      0.64      0.71       157

               1       0.74      0.87      0.80       184

        accuracy                           0.76       341

       macro avg       0.77      0.75      0.75       341

    weighted avg       0.77      0.76      0.76       341

    ----------------------------------------------------------------------------------------------------

    Accuracy Score : 0.9926632428466617

    Accuracy Test : 0.8035190615835777

    Random Forest

    Confusion Matrix

    [[108  49]

     [ 18 166]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.86      0.69      0.76       157

               1       0.77      0.90      0.83       184

        accuracy                           0.80       341

       macro avg       0.81      0.80      0.80       341

    weighted avg       0.81      0.80      0.80       341

    ------------------------------------------------------------------------------

    Accuracy Score : 0.8459280997798972

    Accuracy Test : 0.7800586510263929

    SVM

    Confusion Matrix

    [[ 98  59]

     [ 16 168]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.86      0.62      0.72       157

               1       0.74      0.91      0.82       184

        accuracy                           0.78       341

       macro avg       0.80      0.77      0.77       341

    weighted avg       0.80      0.78      0.77       341

    -----------------------------------------------------------------------------

    Accuracy Score : 0.8459280997798972

    Accuracy Test : 0.7800586510263929

    KNN

    Confusion Matrix

    [[ 98  59]

     [ 16 168]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.86      0.62      0.72       157

               1       0.74      0.91      0.82       184

        accuracy                           0.78       341

       macro avg       0.80      0.77      0.77       341

    weighted avg       0.80      0.78      0.77       341

    ------------------------------------------------------------------------------

**Result:31**

    Accuracy Score : 0.8067632850241546

    Accuracy Test : 0.7800586510263929

    ANN

    Confusion Matrix

    [[ 840  143]

     [ 207 830]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.80      0.81      0.81       157

               1       0.81      0.91      0.81       184

        accuracy                           0.81       341

       macro avg       0.81      0.77      0.81       341

    weighted avg       0.81      0.78      0.81       341

**Result:32**

  Stream

    1.0

    0.8152492668621701

    \*\*Random Forest after Hyperparameter tuning\*\*

    Confusion Matrix

    [[112  45]

     [ 18 166]]

    Classification Reprot

                  precision    recall  f1-score   support

               0       0.86      0.71      0.78       157

               1       0.79      0.90      0.84       184

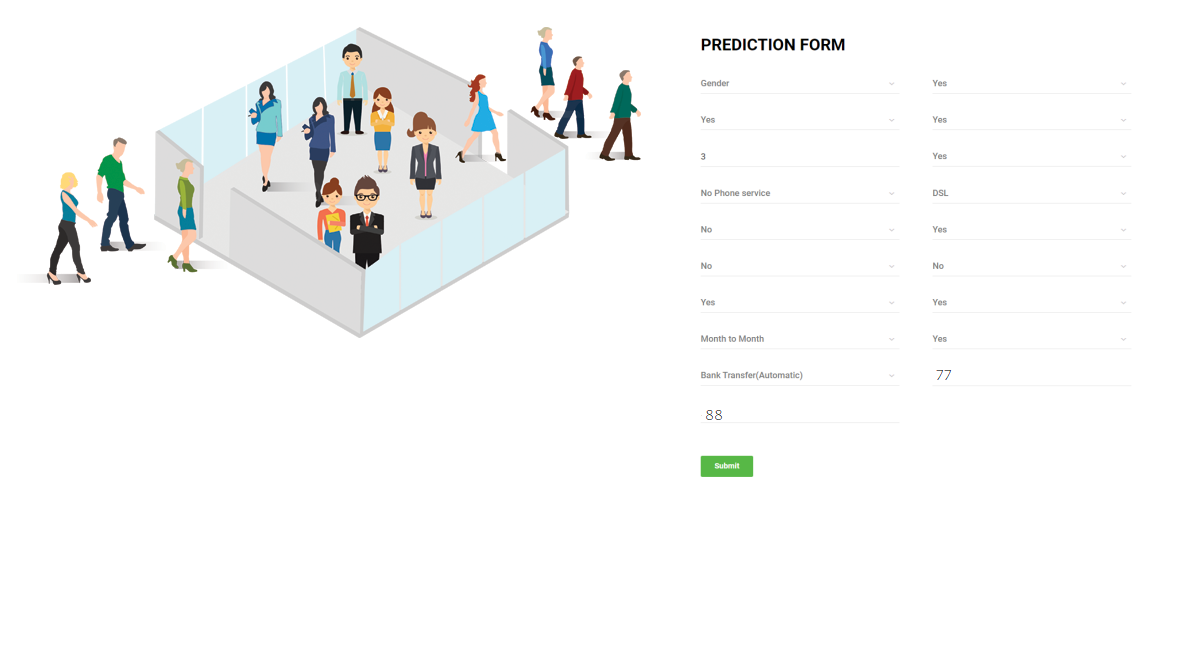
        accuracy                           0.82       341

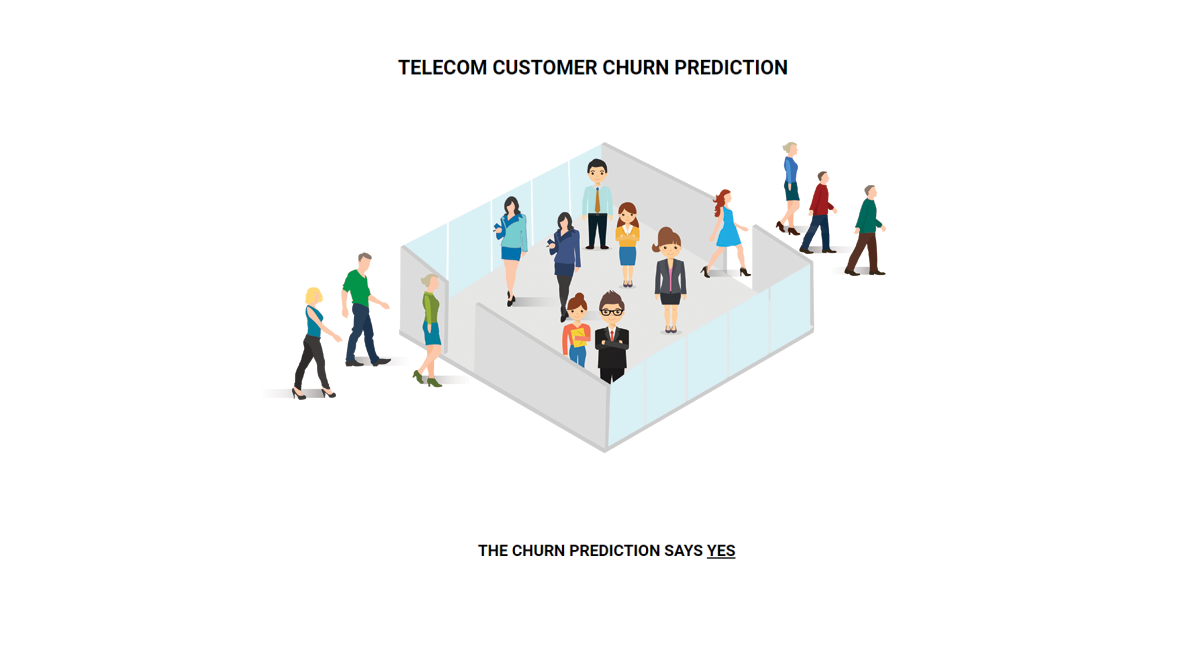
       macro avg       0.82      0.81      0.81       341

    weighted avg       0.82      0.82      0.81       341

**Result 33**

****

****

****

1. **ADVANTAGE AND DISADVANTAGE**

**Advantages**

Increased customer satisfaction: Personalized pricing and offers can enhance the travel experience by providing customers with tailored options that meet their individual needs and preferences.

Improved revenue: Optimized pricing strategies can increase revenue and profitability by maximizing the value of each customer transaction.

Enhanced loyalty: Personalized pricing and offers can help build stronger customer relationships by showing customers that the airline values their business and is willing to provide customized services.

Better marketing: Analyzing customer data can provide airlines with insights into customer behavior and preferences, allowing them to develop targeted marketing campaigns and promotions that engage customers more effectively.

Competitive advantage: By providing personalized pricing and offers, airlines can differentiate themselves from their competitors and improve their market position.

**Disadvantages** of using flight price prediction methods to personalize pricing and offers include:

Privacy concerns: Customers may be hesitant to share personal information with airlines, particularly if they feel that their data is not being used appropriately or ethically.

Accuracy: Predictive models may not always be accurate, and there is a risk of mispricing or misidentifying customer preferences.

Complexity: Developing and implementing a flight price prediction system can be complex and requires significant resources, including expertise in data analytics and machine learning.

Regulatory compliance: The use of customer data is subject to regulatory compliance requirements, which can be complex and time-consuming to navigate.

Customer dissatisfaction: Personalized pricing and offers may not always be perceived as fair or transparent, leading to customer dissatisfaction and potential reputational damage for the airline.

1. **APPLICATION**

Finance: In the finance industry, churn predictive models can help banks and financial institutions identify customers who are likely to close their accounts or stop using their services. This can help banks retain customers by offering them better interest rates, loan terms, or other promotions.

Healthcare: In the healthcare industry, churn predictive models can help insurance companies identify customers who are likely to switch to a different provider or stop paying their premiums. This can help insurance companies retain customers by offering them better coverage or more affordable plans.

Retail: In the retail industry, churn predictive models can help companies identify customers who are likely to stop shopping at their stores or switch to a different brand. This can help companies retain customers by offering them personalized promotions, loyalty rewards, or other incentives.

Subscription services: Churn predictive models are especially useful for subscription-based services, such as streaming platforms or software-as-a-service (SaaS) companies. These models can help companies identify customers who are likely to cancel their subscriptions and offer them personalized deals or incentives to stay subscribed.

1. **CONCLUSION**
2. In conclusion, churn predictive modeling is a powerful tool that can help companies identify and retain their most valuable customers. By analyzing customer data and predicting potential churn, companies can take proactive measures to retain customers, optimize their marketing strategies, and gain a competitive advantage. Churn predictive models can help companies build cost-effective retention strategies that are tailored to individual customers, resulting in higher customer satisfaction, lower customer acquisition costs, and increased revenues.
3. **FUTURE SCOPE**

In the future, churn predictive modeling is expected to become even more sophisticated and accurate, as companies continue to gather and analyze larger amounts of customer data. Machine learning algorithms and artificial intelligence will play an increasingly important role in churn prediction, allowing companies to make more precise and personalized predictions. Additionally, as more industries move towards subscription-based business models, churn predictive models will become even more essential for retaining customers and ensuring long-term business success. Companies will need to stay ahead of the curve by investing in advanced analytics tools, data management systems, and machine learning technologies to stay competitive in a rapidly evolving business landscape.

1. **APPANDIX**

**CODE 1**

import pandas as pd

import numpy as np

import pickle

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

import sklearn

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.model\_selection import RandomizedSearchCV

import imblearn

from imblearn.over\_sampling import SMOTE

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, f1\_score

#import dataset

data = pd.read\_csv(r"/content/Telecom\_customer churn.csv")

data

**CODE 2**

data.info()

#checking for ‘null values’

data.Customer\_ID = pd.to\_numeric(data.Customer\_ID, errors= 'coerce')

data.isnull().any()

data["Customer\_ID"].fillna(data["Customer\_ID"].median() , inplace =False)

data.isnull().sum()

**CODE 3**

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data["gender"] = le.fit\_transform(data["gender"])

data["Partner"] = le.fit\_transform(data["Partner"])

data["Dependents"] = le.fit\_transform(data["Dependents"])

data["Phoneservice"] = le.fit\_transform(data["Phoneservice"])

data["MultipleLines"] = le.fit\_transform(data["MultipleLines"])

data["InternetService"] = le.fit\_transform(data["InternetService"])

data["OnlineBackup"] = le.fit\_transform(data["OnlineBackup"])

data["DeviceProtection"] = le.fit\_transform(data["DeviceProtection"])

data["TechSupport"] = le.fit\_transform(data["TechSupport"])

data["StreamingTV"] = le.fit\_transform(data["StreamingTV"])

data["StreamingMovies"] = le.fit\_transform(data["StreamingMovies"])

data["Contract"] = le.fit\_transform(data["Contract"])

data["PaperlessBilling"] = le.fit\_transform(data["PaperlessBilling"])

data["PaymentMethod"] = le.fit\_transform(data["PaymentMethod"])

data["chrun"] = le.fit\_transform(data["chrun"])

data.head()

**CODE 4**

x= data.iloc[:,0:19].values

y= data.iloc[:,19:20].values

**CODE 5**

from sklearn.preprocessing import OneHotEncoder

one = OneHotEncoder()

a= one.fit\_transform(x[:,6:7]).toarray()

b= one.fit\_transform(x[:,7:8]).toarray()

c= one.fit\_transform(x[:,8:9]).toarray()

d= one.fit\_transform(x[:,9:10]).toarray()

e= one.fit\_transform(x[:,10:11]).toarray()

f= one.fit\_transform(x[:,11:12]).toarray()

g= one.fit\_transform(x[:,12:13]).toarray()

a= one.fit\_transform(x[:,13:14]).toarray()

a= one.fit\_transform(x[:,14:15]).toarray()

a= one.fit\_transform(x[:,16:17]).toarray()

x=np.delete(x,[6,7,8,9,10,11,12,13,14,16],axis=1)

x=np.concatenate((a,b,c,d,e,f,g,h,i,j,x),axis=1)

**CODE 6**

from imblearn.over\_sampling import SMOTE

smt = SMOTE()

x\_resample, y\_resample = smt.fit\_resample(x,y)

x\_resample

**CODE 7**

data.describe()

**CODE 8**

p1t.figure(figsize=(12,5))

p1t.subplot(1,2,1)

sns.distplot(data["tenure"])

p1t.subplot(1,2,2)

sns.distplot(data["MonthCharges"])

p1t.figure(figsize=(12,5))

p1t.subplot(1,2,1)

sns.distplot(data["gender"])

p1t.subplot(1,2,2)

sns.distplot(data["Dependents"])

**CODE 9**

sns.barplot(x="Churn", y="MonthlyCharges", data=data)

**CODE 10**

sns.headmap(data.corr(), annot=Irue)

sns.pairplot(data data, markers=["^","v".],palette="inferno")

from sklearn.model\_selection imort train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_resample,y\_resample,test\_size = 0.2, random\_state =0)

from sklearn.preprocessing import Standardscaler

sc = Standardscaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.fit\_transform(x\_test)

x.test.shape

**CODE 1 1**

#importing and building the Decision tree model

def logreg(x\_train,x\_test,y\_train,y\_test):

  lr = LogisticRegression(random\_state=0)

  lr.fit(x\_train,y\_train)

  y\_lr\_tr = lr.predict(x\_train)

  print(accuracy\_score(y\_lr\_tr,y\_train))

  yPred\_lr = lr.predict(x\_test)

  print(accuracy\_score(yPred\_lr,y\_test))

  print("\*\*\*Logistic Regression\*\*\*")

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_lr))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_lr))

  #printing the train accuracy and test accuracy respectively

logreg(x\_train,x\_test,y\_train,y\_test)

**CODE 12**

#importing and building the Decision tree model

def decisionTree(x\_train,x\_test,y\_train,y\_test):

  dtc = DecisionTreeClassifier(criterion= "entropy",random\_state=0)

  dtc.fit(x\_train,y\_train)

  y\_dt\_tr = dtc.predict(x\_train)

  print(accuracy\_score(y\_dt\_tr,y\_train))

  yPred\_dt = dtc.predict(x\_test)

  print(accuracy\_score(yPred\_dt,y\_test))

  print("\*\*\*Decision Tree\*\*\*”)

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_dt))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_dt))

#printing the train accuracy and test accuracy respectively

  decisionTree(x\_train,x\_test,y\_train,y\_test)

**CODE 13**

#importing and building the Decision tree model

def RandomForest(x\_train,x\_test,y\_train,y\_test):

  rf = RandomForestClassifier(criterion= "entropy",n\_estimators=10,random\_state=0)

  rf.fit(x\_train,y\_train)

  y\_rf\_tr = rf.predict(x\_train)

  print(accuracy\_score(y\_rf\_tr,y\_train))

  yPred\_rf = rf.predict(x\_test)

  print(accuracy\_score(yPred\_rf,y\_test))

  print("\*\*\*Random Forest\*\*\*")

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_rf))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_rf))

#printing the train accuracy and test accuracy respectively

  RandomForest(x\_train,x\_test,y\_train,y\_test)

**CODE 14**

#importing and building the Decision tree model

def KNN(x\_train,x\_test,y\_train,y\_test):

  knn = KNeighborsClassifier()

  knn.fit(x\_train,y\_train)

  y\_knn\_tr = knn.predict(x\_train)

  print(accuracy\_score(y\_rf\_knn,y\_train))

  yPred\_knn = knn.predict(x\_test)

  print(accuracy\_score(yPred\_knn,y\_test))

  print("\*\*\*KNN\*\*\*")

  print("Confusion\_Matrix")

  print(confusion\_matrix(y\_test,yPred\_knn))

  print("Classification Report")

  print(classification\_report(y\_test,yPred\_knn))

#printing the train accuracy and test accuracy respectively

 KNN(x\_train,x\_test,y\_train,y\_test)

**CODE 15**

#importing and building the Decision tree model

def svm(x\_train,x\_test,y\_train,y\_test):

    svm = SVC(kernel = "linear")

    svm.fit(x\_train,y\_train)

    y\_svm\_tr = svm.predict(x\_train)

    print(accuracy\_score(y\_svm\_tr,y\_train))

    yPred\_svm = svm.predict(x\_test)

    print(accuracy\_score(yPred\_svm,y\_test))

    print("\*\*\*Support Vector Machine\*\*\*")

    print("Confusion\_Matrix")

    print(confusion\_matrix(y\_test,yPred\_svm))

    print("Classification Report")

    print(classification\_report(y\_test,yPred\_svm))

#printing the train accuracy and test accuracy respectively

svm(x\_train,x\_test,y\_train,y\_test)

**CODE 16**

# Importing the keras libraries and packages

import keras

from keras.model import sequential

from keras.layers import Dense

# Initialising the ANN

  classifier = sequential()

# Adding the input layer and the first hidden layer

  classifire.add(Dense(units=30, activation='relu', input\_dim=40))

# Adding the second hidden layer

  classifire.add(Dense(units=30, activation='relu'))

# Adding the output layer

  classifire.add(Dense(units=1, activation='sigmoid'))

# Compiling the ANN

  classifire.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Fitting the ANN Training set

model\_history = classifier.fit(x\_train,y\_train, batch\_size=10, validation\_split=0.33, epochs=200)

print(accuracy\_score(ann\_pred,y\_test))

print("\*\*\*\*ANN Model\*\*\*")

print("Confusion\_Matrix")

print(confusion\_matrix(y\_test,ann\_pred))

print("Classification Report")

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

**CODE 17**

#testing on random input values

lr = LogisticRegression(random\_state=0)

lr.fit(x\_train,y\_train)

print("Predicting on random input")

lr\_pred\_own = lr.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:",lr\_pred\_own)

#testing on random input values

dtc = DesicionTreeClassifier(criterion="entropy",random\_state=0)

dtc.fit(x\_train,y\_train)

print("Predicting on random input")

dtc\_pred\_own = dtc.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:" dtc\_pred\_own)

#testing on random input values

rf = Random orsetclassifier(criterion="entropy",n\_estimators=10,random\_state=0)

rf.fit(x\_train,y\_train)

print("Predicting on random input")

rf\_pred\_own = rf.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is: ",rf\_pred\_own)

#testing on random input values

svc = SVC("kernel = linear")

svc.fit(x\_train,y\_train)

print("Predicting on random input")

svm\_pred\_own = svc.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is: ",svm\_pred\_own)

#testing on random input values

knn = KNeighborsClassifier()

knn.fit(x\_train,y\_train)

print("Predicting on random input")

knn\_pred\_own = knn.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:",knn\_pred\_own)

#testing on random input values

print("Predicting on random input")

ann\_pred\_own = classifier.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print(ann\_pred\_own)

ann\_pred\_own = (ann\_pred\_own>0.5)

print("output is:" ann\_pred\_own)

**CODE 18**

def compareModel(X\_train,X\_test,y\_train,y\_test):

    logreg(x\_train,x\_test,y\_train,y\_test)

    print('-'\*100)

    decisionTree(X\_train,X\_test,y\_train,y\_test)

    print('-'\*100)

    RandomForest(X\_train,X\_test,y\_train,y\_test)

    print('-'\*100)

    svm(X\_train,X\_test,y\_train,y\_test)

    print('-'\*100)

    KNN(X\_train,X\_test,y\_train,y\_test)

    print("-"\*100)

compareModel(x\_train,x\_test,y\_train,y\_test)

print(accuracy\_score(ann\_pred,y\_test))

print(\*\*\*\*ANN Model\*\*\*\*)

print("Confusion\_Matrix")

print(confusion\_matrix(y\_test,ann\_pred))

print("Classification Report")

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

**CODE 19**

y\_rf = model.predict(x\_train)

print(accuracy\_score(y\_rf,y\_train))

yPred\_rfcv= model.predict(x\_test)

print(accuracy\_score(yPred\_rfcv,y\_test))

print("Random Forest after Hyperparameter Luning")

print("Confusion\_Matrix")

print(confusion\_matrix)(y\_test,yPred\_rfcv)

print("Clasification Report")

printf(classification\_report(y\_test,yPred\_rfcv))

print("Predicting on random input")

rfcv\_pred\_own = model.predict(sc.transform([[0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1,0,0,456,1,0,3245,4567]]))

print("output is:",rfcv\_pred\_own)

**CODE 20**

classifier.save("telcom\_churn.15")

**CODE 21**

from flask import Flask, render\_template, request

import keras

from keras.models import load\_model

app = Flask(\_\_\_\_name\_\_\_)

model = load\_model("telcom\_churn.h5")

@app.route('/') # rendering the html template

def home():

    return render\_template('home.html')

@app.roude('/')

def helloworld():

    return render\_template("base.html")

@app.route('/assesment')

def prediction():

    return render\_template("index.html")

@app.route('/predict', methods = ['POST'])

def admin():

    a= request.form["gender"]

    if (a == 'f'):

        a=0

    if (a == 'm'):

        a=1

    b= request.form["srcitizen"]

    if (b == 'n'):

        b=0

    if (b == 'y'):

        b=1

    c= request.form["partner"]

    if (c == 'n'):

        c=0

    if (c == 'y'):

        c=1

    d= request.form["dependents"]

    if (d == 'n'):

        d=0

    if (d == 'y'):

        d=1

    e= request.form["Lenure"]

    f= request.form["phservices"]

    if (f == 'n'):

        f=0

    if (f == 'y'):

        f=1

    g= request.form["multi"]

    if (g == 'n'):

        g1,g2,g3=1,0,0

    if (g == 'nps'):

        g1,g2,g3=0,1,0

    if (g == 'y'):

        g1,g2,g3=0,0,1

    h= request.form["is"]

    if (h == 'dsl'):

        h1,h2,h3=1,0,0

    if (h == 'fo'):

        h1,h2,h3=0,1,0

    if (h == 'n'):

        h1,h2,h3=0,0,1

    i= request.form["os"]

    if (i == 'n'):

        i1,i2,i3=1,0,0

    if (i == 'nis'):

        i1,i2,i3=0,1,0

    if (i == 'y'):

        i1,i2,i3=0,0,1

    j= request.form["ob"]

    if (j == 'n'):

        j1,j2,j3=1,0,0

    if (j == 'nis'):

        j1,j2,j3=0,1,0

    if (j =j 'y'):

        j1,j2,j3=0,0,1

    k= request.form["dp"]

    if (k == 'n'):

        k1,k2,k3=1,0,0

    if (k == 'nis'):

        k1,k2,k3=0,1,0

    if (k == 'y'):

        k1,k2,k3=0,0,1

    l= request.form["ts"]

    if (l == 'n'):

        l1,l2,l3=1,0,0

    if (l == 'nis'):

        l1,l2,l3=0,1,0

    if (l == 'y'):

        l1,l2,l3=0,0,1

    g= request.form["stv"]

    if (m == 'n'):

        m1,m2,m3=1,0,0

    if (m == 'nis'):

        m1,m2,m3=0,1,0

    if (m == 'y'):

        m1,m2,m3=0,0,1

    n= request.form["smv"]

    if (n == 'n'):

        n1,n2,n3=1,0,0

    if (n == 'nis'):

        n1,n2,n3=0,1,0

    if (n == 'y'):

        n1,n2,n3=0,0,1

    o= request.form["contract"]

    if (o == 'mtm'):

        o1,o2,o3=1,0,0

    if (o == 'oyr'):

        o1,o2,o3=0,1,0

    if (o == 'tyrs'):

        o1,o2,o3=0,0,1

    p= request.form["pmt"]

    if (p == 'ec'):

        p1,p2,p3=1,0,0,0

    if (p == 'mail'):

        p1,p2,p3,p4=0,1,0,0

    if (p == 'bt'):

        p1,p2,p3,p4=0,0,1,0

    if (p == 'cc'):

        p1,p2,p3,p4=0,0,0,1

    q= request.form["plb"]

    if (q == 'n'):

        q=0

    if (q == 'y'):

        q=1

    r= request.form["mcharges"]

    s =request.form["tcharges"]

t=[[int(g1),int(g2),int(g3),int(h1),int(h2),int(h3),int(i1),int(i2),int(i3),int(j1),int(j2),int(j3),int(k1),int(k2),int(k3),int(l1),int(l2),int(l3),int(m1),int(m2),int(m3),int(n1),int(n2),int(n3),int(o1),int(o2),int(o3),int(p1),int(p2),int(p3),int(p4)]]

print(t)

x = model.predict(t)

print(x[0])

if (x[[0]]) <=0.5):

    y ="NO"

    return render\_template("predno.html", z = y)

if (x[[0]]) >= 0.5):

    y ="Yes"

    return render\_template("predyes.html", z = y)