CUSTOMER CHURN

PREDICTION USING MACHINE LEARNING

Gilland Fausta Putra A

DQLab Telco is a Telco company that already has many branches spread everywhere. Since its establishment in 2019, DQLab Telco has consistently paid attention to its customer experience so that customers will not be left behind. Even though it is only a little over 1 year old, DQLab Telco already has many customers who have switched their subscriptions to competitors. Management wants to reduce the number of churned customers by using machine learning. Therefore, the Data Scientist team was asked to prepare the data as well as create the right prediction model to determine whether a customer will unsubscribe (churn) or not.

01 LIBRARIES AND DATASET

LIBRARIES

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import LabelEncoder
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report
     from sklearn.metrics import confusion_matrix
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.ensemble import GradientBoostingClassifier

     pd.options.display.max_columns = 50
```

DATASET OVERVIEW

```
[ ] df_load = pd.read_csv('dqlab_telco.csv')
```

```
[ ] df_load.shape
```

```
(7113, 22)
```

```
[ ] df_load.head()
```

	UpdatedAt	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceP
0	202006	45759018157	Female	0	Yes	No	1.0	No	No phone service	DSL	No	Yes	
1	202006	45557574145	Male	0	No	No	34.0	Yes	No	DSL	Yes	No	
2	202006	45366876421	Male	0	No	No	2.0	Yes	No	DSL	Yes	Yes	
3	202006	45779536532	Male	0	No	No	45.0	No	No phone service	DSL	Yes	No	
4	202006	45923787906	Female	0	No	No	2.0	Yes	No	Fiber optic	No	No	

Correct format of customerID

```
[ ]  df_load['valid_id'] = df_load['customerID'].astype(str).str.match(r'(45\d{9,10})')
     df_load = (df_load[df_load['valid_id'] == True]).drop('valid_id', axis = 1)
     print('The result of the number of filtered Customer ID is', df_load['customerID'].count())
```

```
The result of the number of filtered Customer ID is 7006
```

Ensure that there are no duplicate customer ID numbers

```
[ ]  # Drop Duplicate Rows
     df_load.drop_duplicates()
     # Drop duplicate ID sorted by Periode
     df_load = df_load.sort_values('UpdatedAt', ascending=False).drop_duplicates(['customerID'])
     print('the number of customer ID that have been removed from duplicates is',df_load['customerID'].count())
```

```
the number of customer ID that have been removed from duplicates is 6993
```

Handling Missing Value

```
Status Missing Values : True

The number of Missing Values for each column
tenure              99
MonthlyCharges      26
TotalCharges        15
UpdatedAt            0
DeviceProtection     0
PaymentMethod        0
PaperlessBilling     0
Contract             0
StreamingMovies      0
StreamingTV          0
TechSupport          0
OnlineBackup         0
customerID           0
OnlineSecurity       0
InternetService      0
MultipleLines        0
PhoneService         0
Dependents           0
Partner              0
SeniorCitizen        0
gender               0
Churn                0
dtype: int64
```

```
[ ] print('Total missing values data from the Churn column:', df_load['Churn'].isnull().sum())

    Total missing values data from the Churn column: 43

[ ] # Dropping all Rows with spesific column (churn)
    df_load.dropna(subset=['Churn'],inplace=True)
    print('Total Rows and Data columns after deleting the Missing Values data are', df_load.shape)

    Total Rows and Data columns after deleting the Missing Values data are (6950, 22)
```

```
[ ] #Handling missing values Tenure fill with 11
    df_load['tenure'].fillna(11, inplace=True)

[ ] #Handling missing values num vars (except Tenure)
    for col_name in list(['MonthlyCharges','TotalCharges']):
      median = df_load[col_name].median()
      df_load[col_name].fillna(median, inplace=True)
```

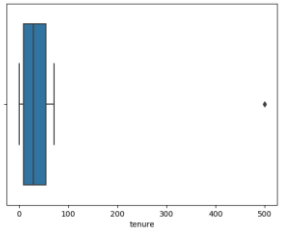
```
The number of Missing Values after imputing the data is:
UpdatedAt           0
customerID          0
TotalCharges        0
MonthlyCharges      0
PaymentMethod       0
PaperlessBilling    0
Contract            0
StreamingMovies     0
StreamingTV         0
TechSupport         0
DeviceProtection    0
OnlineBackup        0
OnlineSecurity      0
InternetService     0
MultipleLines       0
PhoneService        0
tenure              0
Dependents          0
Partner             0
SeniorCitizen       0
gender              0
Churn               0
dtype: int64
```

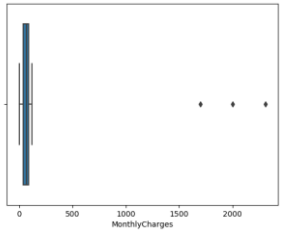
After we have handled it by deleting rows and filling in rows with a certain value, it is evident that there are no more missing values in the data, as evidenced by the number of missing values for each variable which has a value of 0.

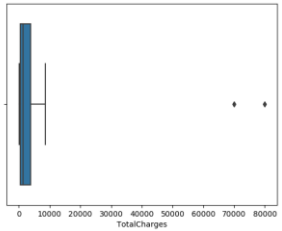
Outlier

```
[ ]  print(df_load[['tenure','MonthlyCharges','TotalCharges']].describe())
```

	tenure	MonthlyCharges	TotalCharges
count	6950.000000	6950.000000	6950.000000
mean	32.477266	65.783741	2305.083460
std	25.188910	50.457871	2578.651143
min	0.000000	0.000000	19.000000
25%	9.000000	36.462500	406.975000
50%	29.000000	70.450000	1400.850000
75%	55.000000	89.850000	3799.837500
max	500.000000	2311.000000	80000.000000





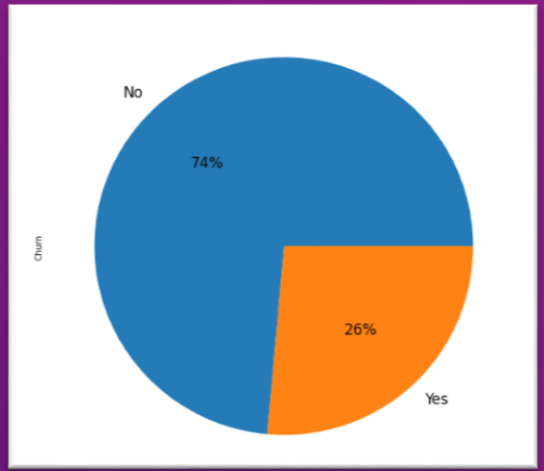


02

EXPLORATORY DATA ANALYSIS

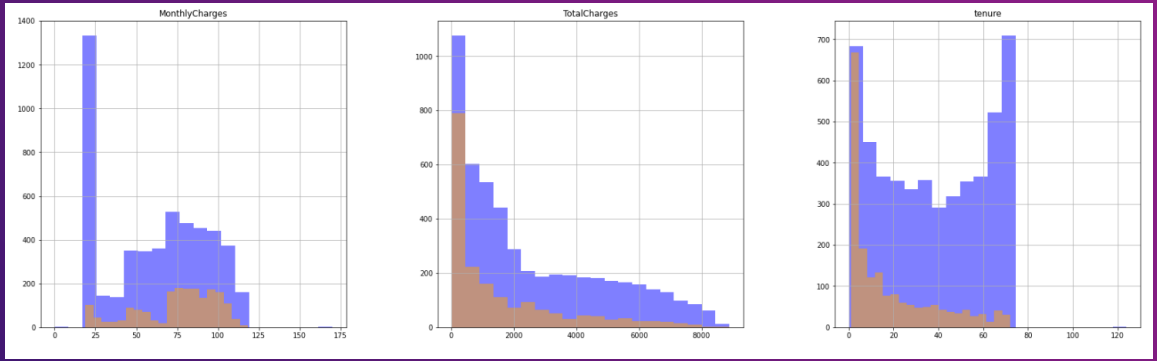
VISUALIZE PERCENTAGE CHURN

```
[ ]  fig = plt.figure()
     ax = fig.add_axes([0,0,1,1])
     ax.axis('equal')
     labels = ['No','Yes']
     churn = df_load.Churn.value_counts()
     ax.pie(churn, labels=labels, autopct='%.0f%%')
     plt.title("Presentase Churn", fontsize=20)
     plt.show()
```



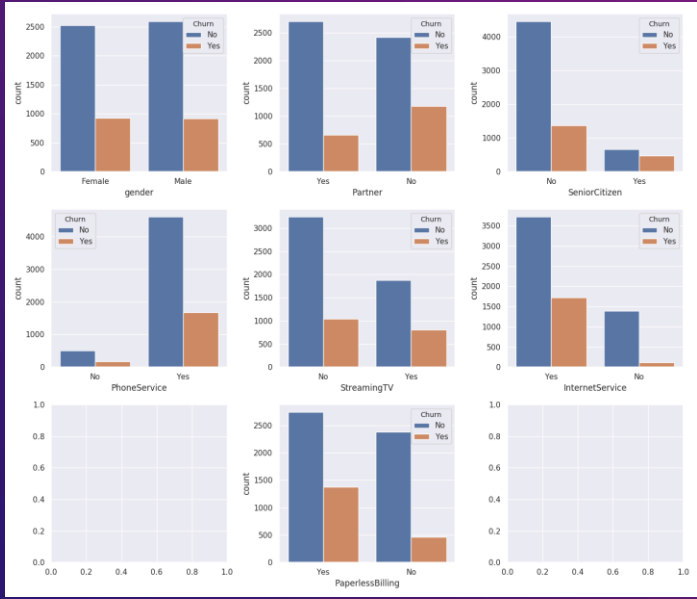
overall distribution of data customers do not churn, with detail Churn as much as 26% and No Churn as much as 74

VISUALIZE NUMERICAL VARIABLE



- MonthlyCharges, there is a tendency that the smaller the value of the monthly fee charged, the smaller the tendency to make Churn.
- TotalCharges, has no trend whatsoever towards Churn customers.
- Tenure, there is a tendency that the longer a customer subscribes, the smaller the tendency to churn.

VISUALIZE KATEGORIC VARIABLE



- There is no significant difference for people who churn in terms of gender and PhoneService
- There is a tendency that people who do churn are people who don't have partners (partner: No),
people whose status issenior citizens (SeniorCitizen: Yes), people who have TV streaming services (StreamingTV: Yes), peoplewho have Internet service (internetService: Yes) and people who se bills are paperless (PaperlessBilling: Yes)

03 PREPROCESSING DATA

```
[ ] #Remove the unnecessary columns customerID & UpdatedAt
    cleaned_df = df_load.drop(['customerID','UpdatedAt','Dependents','MultipleLines','OnlineSecurity','OnlineBackup','DeviceProtection','TechSupport','StreamingMovi
    print(cleaned_df.head())
```

```
      gender  SeniorCitizen Partner  tenure PhoneService InternetService  \
0     Female              0     Yes     1.0           No             DSL
4727    Male              0     Yes    60.0          Yes              No
4738    Male              0      No     5.0          Yes     Fiber optic
4737  Female              0     Yes    72.0          Yes     Fiber optic
4736  Female              0     Yes    56.0          Yes             DSL

                StreamingTV PaperlessBilling  MonthlyCharges  TotalCharges Churn
0                        No              Yes           29.85         29.85    No
4727  No internet service               Yes           20.50       1198.80    No
4738                  Yes               No          104.10        541.90   Yes
4737                  Yes              Yes          115.50       8312.75    No
4736                  Yes               No           81.25       4620.40    No
```

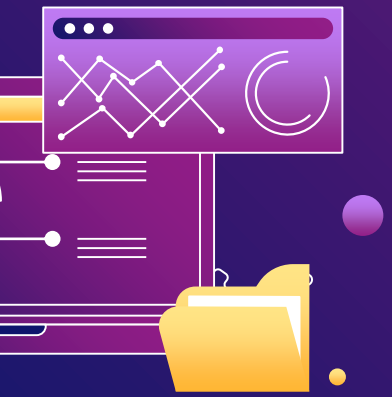
```
[ ] #Convert all the non-numeric columns to numerical data types
    for column in cleaned_df.columns:
      if cleaned_df[column].dtype == np.number: continue
      # Perform encoding for each non-numeric column
      cleaned_df[column] = LabelEncoder().fit_transform(cleaned_df[column])
    print(cleaned_df.describe())
```

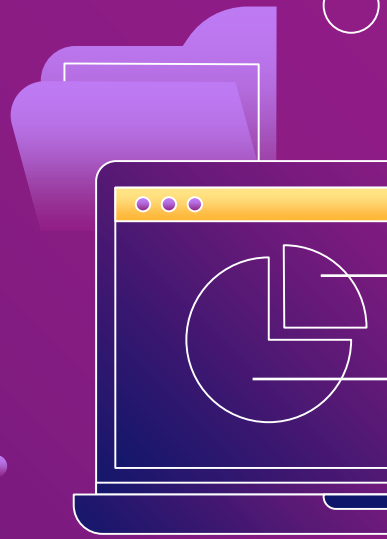
```
              gender  SeniorCitizen      Partner       tenure  PhoneService  \
count  6950.000000    6950.000000  6950.000000  6950.000000   6950.000000
mean      0.504317       0.162302     0.483309    32.423165      0.903741
std       0.500017       0.368754     0.499757    24.581073      0.294967
min       0.000000       0.000000     0.000000     0.000000      0.000000
25%       0.000000       0.000000     0.000000     9.000000      1.000000
50%       1.000000       0.000000     0.000000    29.000000      1.000000
75%       1.000000       0.000000     1.000000    55.000000      1.000000
max       1.000000       1.000000     1.000000   124.000000      1.000000

       InternetService  StreamingTV  PaperlessBilling  MonthlyCharges  \
count      6950.000000  6950.000000       6950.000000     6950.000000
mean          0.872950     0.985180          0.591942       64.992201
std           0.737618     0.885069          0.491509       30.032040
min           0.000000     0.000000          0.000000        0.000000
25%           0.000000     0.000000          0.000000       36.462500
50%           1.000000     1.000000          1.000000       70.450000
75%           1.000000     2.000000          1.000000       89.850000
max           2.000000     2.000000          1.000000      169.931250
```

```
[ ] # Predictor dan target
    X = cleaned_df.drop('Churn', axis = 1)
    y = cleaned_df['Churn']
    # Splitting train and test
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    # Print according to the expected result
    print('The number of rows and columns of x_train is:', x_train.shape,', while the number of rows and columns of y_train is:', y_train.shape)
    print('The percentage of Churn in the Training data is:')
    print(y_train.value_counts(normalize=True))
    print('The number of rows and columns of x_test is:', x_test.shape,', while the number of rows and columns of y_test is:', y_test.shape)
    print('The percentage of Churn in the Testing data is:')
    print(y_test.value_counts(normalize=True))
```

```
The number of rows and columns of x_train is: (4865, 10) , while the number of rows and columns of y_train is: (4865,)
The percentage of Churn in the Training data is:
0    0.734841
1    0.265159
Name: Churn, dtype: float64
The number of rows and columns of x_test is: (2085, 10) , while the number of rows and columns of y_test is: (2085,)
The percentage of Churn in the Testing data is:
0    0.738129
1    0.261871
Name: Churn, dtype: float64
```

04 MODELLING

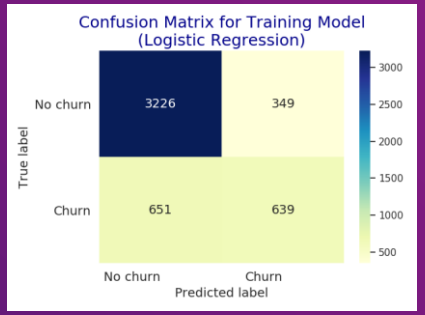




LOGISTIC REGRESSION

```
[ ] # Train
    y_train_pred = log_model.predict(x_train)

    # Print classification report
    print('Classification Report Training Model (Logistic Regression) :')
    print(classification_report(y_train, y_train_pred))
```

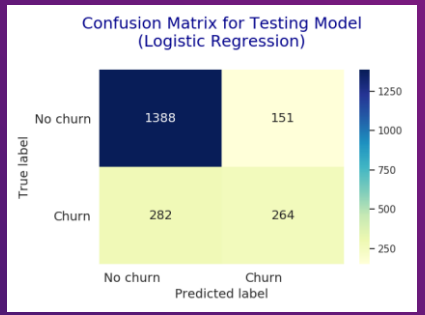
```
Classification Report Training Model (Logistic Regression) :
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      3575
           1       0.65      0.49      0.56      1290

    accuracy                           0.80      4865
   macro avg       0.74      0.70      0.71      4865
weighted avg       0.78      0.80      0.79      4865
```



From the training data it can be seen that the model is able to predict by producing an accuracy of 80%

```
[ ] # Predict
    y_test_pred = log_model.predict(x_test)

    # Print classification report
    print('Classification Report Testing Model (Logistic Regression) :')
    print(classification_report(y_test, y_test_pred))
```

```
Classification Report Testing Model (Logistic Regression) :
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1539
           1       0.64      0.48      0.55       546

    accuracy                           0.79      2085
   macro avg       0.74      0.69      0.71      2085
weighted avg       0.78      0.79      0.78      2085
```

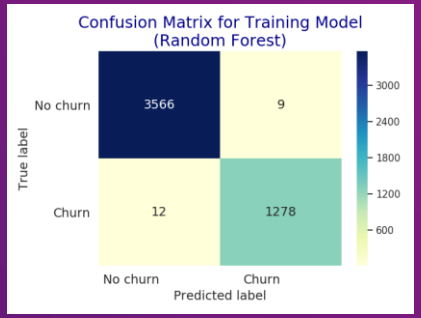


From the data testing it can be seen that the model is able to predict by producing an accuracy of 79%

RANDOM FOREST CLASSIFIER

```
[ ] #Train
    y_train_pred = rdf_model.predict(x_train)
    print('Classification Report Training Model (Random Forest Classifier) :')
    print(classification_report(y_train, y_train_pred))
```

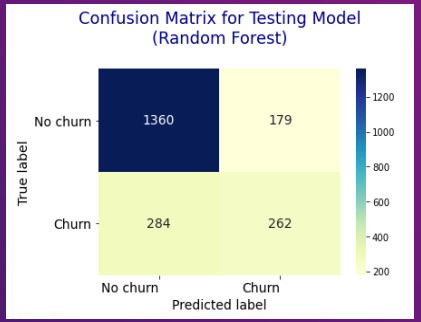
```
Classification Report Training Model (Random Forest Classifier) :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3575
           1       1.00      0.99      0.99      1290

    accuracy                           1.00      4865
   macro avg       1.00      0.99      0.99      4865
weighted avg       1.00      1.00      1.00      4865
```



From the training data it can be seen that the model is able to predict by producing an accuracy of 100%

```
[ ] # Predict
    y_test_pred = rdf_model.predict(x_test)

    # Print classification report
    print('Classification Report Testing Model (Random Forest Classifier):')
    print(classification_report(y_test, y_test_pred))
```

```
Classification Report Testing Model (Random Forest Classifier):
              precision    recall  f1-score   support

           0       0.82      0.89      0.86      1539
           1       0.60      0.47      0.53       546

    accuracy                           0.78      2085
   macro avg       0.71      0.68      0.69      2085
weighted avg       0.77      0.78      0.77      2085
```

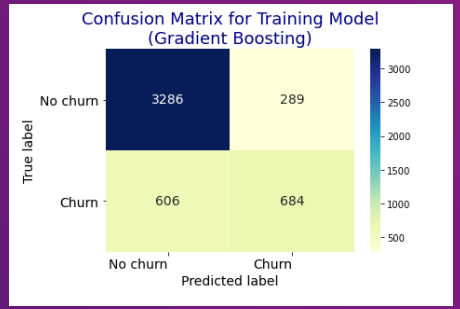


From the data testing it can be seen that the model is able to predict by producing an accuracy of 78%

GRADIENT BOOSTING CLASSIFIER

```
[ ] # Train
    y_train_pred = gbt_model.predict(x_train)

    # Print classification report
    print('Classification Report Training Model (Gradient Boosting) :')
    print(classification_report(y_train, y_train_pred))
```

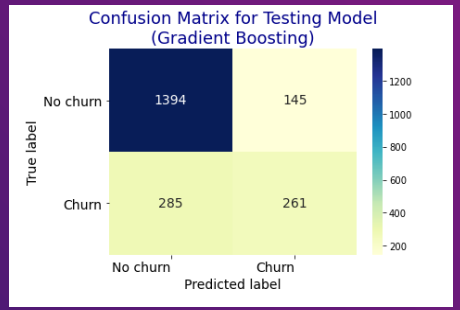
```
Classification Report Training Model (Gradient Boosting) :
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      3575
           1       0.70      0.54      0.61      1290

    accuracy                           0.82      4865
   macro avg       0.78      0.73      0.75      4865
weighted avg       0.81      0.82      0.81      4865
```



From the training data it can be seen that the model is able to predict by producing an accuracy of 82%

```
[ ] # Predict
    y_test_pred = gbt_model.predict(x_test)

    # Print classification report
    print('Classification Report Testing Model (Gradient Boosting):')
    print(classification_report(y_test, y_test_pred))
```

```
Classification Report Testing Model (Gradient Boosting):
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      1539
           1       0.66      0.49      0.56       546

    accuracy                           0.80      2085
   macro avg       0.75      0.70      0.72      2085
weighted avg       0.79      0.80      0.79      2085
```



From the data testing it can be seen that the model is able to predict by producing an accuracy of 80%

RESULTS

MODEL	TRAIN	TEST
LOGISTIC REGRESSION	80%	79%
RANDOM FOREST CLASSIFIER	100%	78%
GRADIENT BOOSTING CLASSIFIER	82%	80%

Based on the modeling that has been done, it can be concluded the best model is to use the Logistic Regression algorithm. This is because the performance of the Logistic Regression model tends to be able topredict equally well in the training and testing phases (80% training accuracy, 79% testing accuracy), on the other hand other algorithms tend to over-fit their performance.

However, this does not make us draw conclusions that if we use Logistic Regression to do any modeling, we still have to do a lot of model experiments to determine which one is the best.

THANKS!