Customer Churn Prediction

By, Emma, Jason and Sam.

About the Project

The objective of this project is to understand and predict customer churn for a bank.

Specifically, we will initially perform Exploratory Data Analysis (EDA) to identify the factors contributing to customer churn.

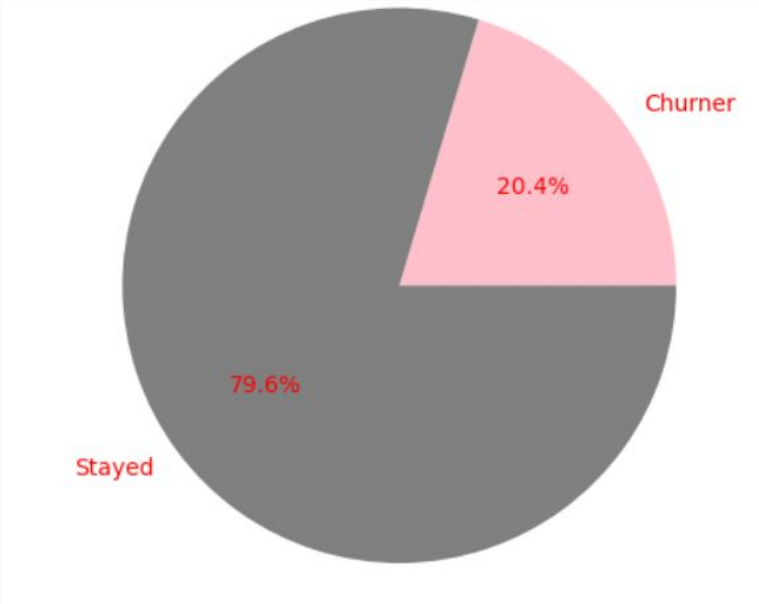
This analysis will later help us build **Machine Learning models** to **predict whether a customer will churn or not.**

Machine Learning models:

- There are a number of techniques that can be used to predict customer churn, including machine learning algorithms ;
  - Neural Network
  - Logistic regression
  - Random Forest
  - K-Nearest Neighbour
- To increase efficiency, you can use advanced algorithms;
  - XGBoost
  - LightGBM

Dataset for ML Training

- Dataset contains 10,000 unique customer id and each has 10 features.
- Features included:
  - Geography (categorical)
  - Gender (categorical)
  - Age
  - Tenure
  - Balance
  - Number of products
  - Credit card or not
  - Active or not
  - Estimated Salary
  - Credit score



Data preparation

- Data cleaning: checking null value use function of “isnull”
- Combine both encoded features with original dataset as new dataframe called “attribtion_df”
- Declared “Existed” as target column, and the rest being X features
- Split into 3 lots for training, validation and test
- Use of StandardScaler from sklearn.preprocessing

```
# Define the target set y using the Attrition_Yes column
y = attrition_df["Exited"]


# Define features set X by selecting all columns but Attrition_Yes and Attrition_No
X = attrition_df.drop(columns=["Exited"])
```

Training, Validation and Testing Sets

Using the function train_test_split from the scikit-learn library in Python. This function is used to split a given dataset into two subsets: a training set and a test set.

Step 1:

```
# In the first step we will split the data in training and remaining dataset
X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.8)
```

Step 2:

```
# Now since we want the validation and test size to be equal (10% each of overall data).
# we have to define valid_size=0.5 (that is 50% of remaining data)
validation_size = 0.5
X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, train_size=0.5)
```

Model Structure

- This code is creating a deep neural network model with two hidden layers using TensorFlow's Sequential model class.
- The number of input features is determined by counting the columns in the training set.
- The number of hidden nodes in the hidden layers are calculated using the mean of the number of input features and the number of output neurons, and the floor division operator (//). The model is then displayed using the summary method.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 128

 dense_1 (Dense)             (None, 4)                 36

 dense_2 (Dense)             (None, 1)                 5

=================================================================
Total params: 169
Trainable params: 169
Non-trainable params: 0
_________________________________________________________________
```

Compiling the Model

We compile the Sequential model using loss function of 'binary_crossentropy', the 'adam' optimizer, and the various metric.

- The loss function of 'binary_crossentropy' is used for binary classification problem;
- 'Adam' is a popular optimisation algorithm for training nn;
- In this case, metric "Recall" is what we will be compared against other model.

```
# Compile the Sequential model
nn.compile(
    loss="binary_crossentropy",
    optimizer="adam",
    metrics=[
        "accuracy",
        tf.keras.metrics.TruePositives(name="tp"),
        tf.keras.metrics.TrueNegatives(name="tn"),
        tf.keras.metrics.FalsePositives(name="fp"),
        tf.keras.metrics.FalseNegatives(name="fn"),
        tf.keras.metrics.Precision(name="precision"),
        tf.keras.metrics.Recall(name="recall"),
        tf.keras.metrics.AUC(name="auc"),
    ],
)
```

Training the Model

We train model using scaled training dataset, and tuning with validation dataset:

- epochs parameter is set to 50, which means that the model will be trained for 50 iterations over the entire training set.
- batch_size is set to 1000, which means that the model's parameters will be updated after every 1000 samples.
- verbose parameter is set to 1, it will show a progress bar during training
- training_history variable will store the record of the training process, including the values of the loss function and the metrics during each epoch. This can be used to analyze the training process and make adjustments as needed.

```
# Training the model
batch_size = 1000
epochs = 50
training_history = nn.fit(
    X_train_scaled,
    y_train,
    validation_data=(X_valid_scaled, y_valid),
    epochs=epochs,
    batch_size=batch_size,
    verbose=1,
)
```

Evaluating the model

This problem is a typical classification task, the use of recall (i.e. ratio of true positive prediction made out of total positive cases in the dataset) as performance metrics are more relevant in this case, since correctly classifying elements of the positive class (customers who will churn) is more critical for the bank.

```
# Predict classes using validation data
y_predict_classes = (nn.predict(X_valid_scaled) > 0.5).astype("int32")
# Display classification report
print(classification_report(y_predict_classes, y_valid))
```

✓ 1.7s

```
32/32 [==============================] - 1s 10ms/step
              precision    recall  f1-score   support

           0       0.99      0.78      0.87       990
           1       0.02      0.40      0.03        10

    accuracy                           0.77      1000
   macro avg       0.51      0.59      0.45      1000
weighted avg       0.98      0.77      0.86      1000
```

Logistic Regression Model using training data

- We imported the LogisticRegression module from the sklearn library and instantiated a Logistic Regression model using the LogisticRegression class. We set the solver parameter to 'lbfgs' and the random_state parameter to 1. We then fit the model to the training data using the fit method of the LogisticRegression class, passing in the training data and the target variable.
- Finally, we used the testing data to make predictions using the predict method and stored the predictions in a variable called "predictions".

Calculate accuracy score, Generate a confusion matrix and print the classification report

- The results show that the model has an accuracy of 0.5789, with a precision of 0.82 for predicting when a customer will leave and a precision of 0.58 for predicting when a customer will stay.
- The model also has a recall of 0.96 for predicting when a customer will leave and a recall of 0.19 for predicting when a customer will stay.
- The f1-score for predicting when a customer will leave is 0.89 and the f1-score for predicting when a customer will stay is 0.29.
- The support for predicting when a customer will leave is 1991 and the support for predicting when a customer will stay is 509.

Using the RandomOverSampler to resample the data

We used the RandomOverSampler module from the imbalanced-learn library to resample the data and balance the labels. We instantiated the random oversampler model, set the random_state parameter to 1, and fit the original training data to the model. We then counted the distinct values of the resampled labels data and confirmed that the labels now have an equal number of data points.

```
0    5972
1    5972
Name: Exited, dtype: int64
```

Using the LogisticRegression classifier and the resampled data to fit the model.

First, we instantiated the Logistic Regression model and set a random_state parameter of 1. Then, we fit the model using the resampled training data and made a prediction using the testing data. The resulting predictions were stored in a variable called resampled_predictions.

Evaluating the model

	pre	rec	spe	f1	geo	iba	sup
Stayed-0	0.91	0.73	0.69	0.81	0.71	0.51	807
Churner-1	0.38	0.69	0.73	0.49	0.71	0.50	193
avg / total	0.81	0.72	0.70	0.75	0.71	0.51	1000

Modeling using Random Forest

Classification Report

	precision	recall	f1-score	support
Stayed-0	0.80	0.89	0.84	797
Churner-1	0.22	0.12	0.15	203
accuracy			0.73	1000
macro avg	0.51	0.51	0.50	1000
weighted avg	0.68	0.73	0.70	1000

Modelling using K-Nearest Neighbors

	precision	recall	f1-score	support
Stayed-0	0.95	0.86	0.90	888
Churner-1	0.37	0.63	0.47	112
accuracy			0.84	1000
macro avg	0.66	0.75	0.68	1000
weighted avg	0.88	0.84	0.85	1000

Modelling using XGBoost

	precision	recall	f1-score	support
Stayed-0	0.98	0.88	0.93	892
Churner-1	0.46	0.81	0.58	108
accuracy			0.88	1000
macro avg	0.72	0.85	0.76	1000
weighted avg	0.92	0.88	0.89	1000

Modelling using LightGBM