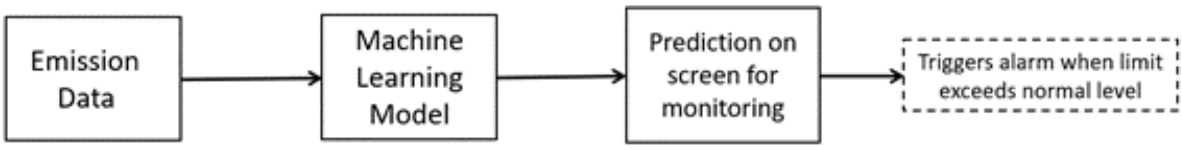
Telecom Customer Churn Prediction

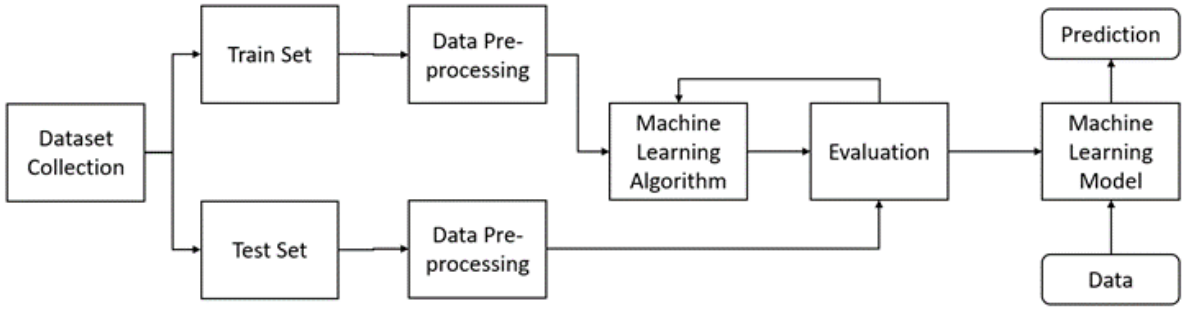
Customer churn is a major problem and one of the most important concerns for large companies. Due to the direct effect on the revenues of the companies, especially in the telecom field, companies are seeking to develop means to predict potential customer to churn. Therefore, finding factors that increase customer churn is important to take necessary actions to reduce this churn. Churn prediction helps in identifying those customers who are likely to leave a company.

Customer churn means shifting from one service provider to its competitor in the market. Customer churn is one of the biggest fears of any industry, particularly for the telecom industry. With an increase in the number of telecom service providers in South Asia, the level of competition is quite high. Although there are many reasons for customer churn, some of the major reasons are service dissatisfaction, costly subscription, and better alternatives. The telecom service providers strive very hard to sustain in this competition. So to sustain this competition they often try to retain their customers than acquiring new ones as it proved to be much costlier. Hence predicting churn in the telecom industry is very important. To reduce customer churn, telecom companies need to predict which customers are at high risk of churn.

Block Diagram:



Machine Learning Workflow:



Prerequisites:

1. **To develop this project, we need to install following software/packages:**

Anaconda Navigator:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter

Notebook,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Visual studio code.

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Project Flow:

- User interacts with the UI (User Interface) to enter the current mine working conditions.
- Entered readings are analyzed and predictions are made based on interpretation that whether employee will be attributed or not.

- Predictions are popped onto the UI.

To Build a Chrun Modelling Predection

- **Install sci-kit learn**

  Scikit-learn (**Sklearn**) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistence interface in Python.

- **Install Flask**
  - Flask is a Web application framework written in python
  - For installation of Flask

  Open your anaconda prompt and Type “pip install Flask”

Data Collection

The given data set is related to Taxi Fares. It was taken from the website **kaggle.com**. The website provides various datasets from various domains.

Data Pre-processing

Importing required Libraries:

```
In [43]: #applying different activation function on classification model
         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns
```

Pandas: It is a python library mainly used for data manipulation.

NumPy: This python library is used for numerical analysis.

Matplotlib and Seaborn: Both are the data visualization library used for plotting graph which will help us for understanding the data.

Importing the dataset:

```
In [60]: ds=pd.read_csv("C:/Users/HARINATH/OneDrive/Desktop/Flask/Churn_Modelling1.csv")
         ds.shape

Out[60]: (10000, 14)
```

```
In [61]: ds.head()
```

Out[61]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	1	15634602	Hargrave	619	France	Female	42.0	2	0.00	1	1	1	101348.88
1	2	15647311	Hill	608	Spain	Female	41.0	1	83807.86	1	0	1	112542.58
2	3	15619304	Onio	502	France	Female	42.0	8	159660.80	3	1	0	113931.57
3	4	15701354	Boni	699	France	Female	39.0	1	0.00	2	0	0	93826.63
4	5	15737888	Mitchell	850	Spain	Female	NaN	2	125510.82	1	1	1	79084.10

- You might have your data in .csv files, .excel files or .tsv files or something else. But the goal is the same in all cases. If you want to analyse that data using pandas, the first step will be to read it into a data structure that's compatible with pandas.

- Let's load a .csv data file into pandas. There is a function for it, called **read_csv().**We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).

- Path names on Windows tend to have backslashes in them. But we want them to mean actual backslashes, not special characters.

Taking care of Missing Data:

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously, you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values for continuous and for categorical we make use of mode values and replace the missing data.

We will be using **isnull().any()** method to see which column has missing values.

```
In [46]: ds.isnull().any()
```

```
Out[46]: CreditScore        False
         Geography          False
         Gender             False
         Age                False
         Tenure             False
         Balance            False
         NumOfProducts      False
         HasCrCard          False
         IsActiveMember     False
         EstimatedSalary    False
         Exited             False
         dtype: bool
```

```
In [9]: ds['Geography'].fillna(ds['Geography'].mode()[0],inplace=True)
        ds['Gender'].fillna(ds['Gender'].mode()[0],inplace=True)
        ds['Age'].fillna(ds['Age'].mean(),inplace=True)
        ds['Balance'].fillna(ds['Balance'].mean(),inplace=True)
```

Label encoding

Typically, any structured dataset includes multiple columns with combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too. We need to convert each text category to numbers in order for the machine to process those using mathematical equations.

How should we handle categorical variables? There are Multiple way to handle, but will see one of it is LabelEncoding.

- **Label Encoding** is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

Let's see how to implement label encoding in Python using the scikit-learn library.

As we have to convert only the text class category columns, we first select it then we will implement Label Encoding to it

```
In [11]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
         ds['Geography']=le.fit_transform(ds['Geography'])
         ds['Gender']=le.fit_transform(ds['Gender'])
```

```
In [12]: from collections import Counter as c
         ds['Gender'].value_counts()

Out[12]: 1    5458
         0    4542
         Name: Gender, dtype: int64
```

```
In [13]: from collections import Counter as c
         ds['Geography'].value_counts()

Out[13]: 0    5016
         1    2508
         2    2476
         Name: Geography, dtype: int64
```

Feature Scaling

Splitting Data into Train and Test:

When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.

But the question is, how do you split the data? You can't possibly manually split the dataset into two sets. And you also have to make sure you split the data in a random manner. To help us with this task, the Scikit library provides a tool, called the Model Selection library. There is a class in the library which is, '**train_test_split.**' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

- **Train Dataset**: Used to fit the machine learning model.

- **Test Dataset**: Used to evaluate the fit machine learning model.

In general you can allocate 80% of the dataset to training set and the remaining 20% to test set.

We will create 4 sets— X_train (training part of the matrix of features), X_test (test part of the matrix of features), Y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices), Y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices.

There are a few other parameters that we need to understand before we use the class:

test_size — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset

train_size — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.

random_state — here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the Random_state class, which will become the number generator. If you don't pass anything, the Random_state instance used by np.random will be used instead.

Now split our dataset into train set and test using train_test_split class from scikit learn library.

```
In [21]: from sklearn.preprocessing import StandardScaler
         sc=StandardScaler()

         xtrain=sc.fit_transform(xtrain)
         xtest=sc.fit_transform(xtest)
         xtest
```

```
Out[21]: array([[-0.56129438,  0.28034192, -1.11339196, ...,  0.66011376,
                   0.97628121,  1.62185911],
                 [-1.33847768, -0.93853599, -1.11339196, ...,  0.66011376,
                  -1.02429504,  0.504204  ],
                 [ 0.58347561,  1.49921982, -1.11339196, ...,  0.66011376,
                   0.97628121, -0.41865644],
                 ...,
                 [-0.76084144,  1.49921982,  0.8981563 , ...,  0.66011376,
                  -1.02429504,  0.72775202],
                 [-0.0046631 ,  0.28034192,  0.8981563 , ...,  0.66011376,
                   0.97628121, -1.54162886],
                 [-0.81335383,  0.28034192,  0.8981563 , ...,  0.66011376,
                  -1.02429504,  1.62356528]])
```

Model Building

Training and testing the model:

There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms are Regression algorithms.

Example: 1. Linear Regression.

2. Logistic Regression.

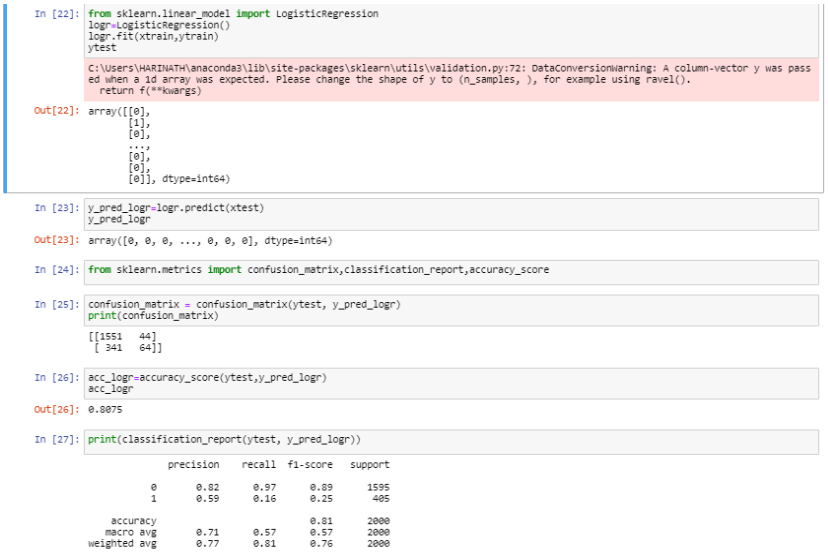
3. Random Forest Regression / Classification.

4. Decision Tree Regression / Classification.

You will need to train the datasets to run smoothly and see an incremental improvement in the prediction rate.

Now we apply Logistic regression algorithm on our dataset.

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In **logistic regression**, the dependent variable isa binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).



```
In [22]: from sklearn.linear_model import LogisticRegression
         logr=LogisticRegression()
         logr.fit(xtrain,ytrain)
         ytest

         C:\Users\HARINATH\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was pass
         ed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
           return f(**kwargs)

Out[22]: array([[0],
                [1],
                [0],
                ...,
                [0],
                [0],
                [0]], dtype=int64)

In [23]: y_pred_logr=logr.predict(xtest)
         y_pred_logr

Out[23]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

In [24]: from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

In [25]: confusion_matrix = confusion_matrix(ytest, y_pred_logr)
         print(confusion_matrix)

         [[1551   44]
          [ 341   64]]

In [26]: acc_logr=accuracy_score(ytest,y_pred_logr)
         acc_logr

Out[26]: 0.8075

In [27]: print(classification_report(ytest, y_pred_logr))

                      precision    recall  f1-score   support

                   0       0.82      0.97      0.89      1595
                   1       0.59      0.16      0.25       405

            accuracy                           0.81      2000
           macro avg       0.71      0.57      0.57      2000
        weighted avg       0.77      0.81      0.76      2000
```

Now we will apply SVM to our dataset.

Support vector machines (**SVMs**) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of **support vector machines** are: Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples.

```
In [40]: from sklearn.svm import SVC
         svm=SVC()
         svm.fit(xtrain,ytrain)

         C:\Users\HARINATH\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was pass
         ed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
           return f(**kwargs)

Out[40]: SVC()
```

Predict the values

Once the model is trained, it's ready to make predictions. We can use the **predict** method on the model and pass **x_test** as a parameter to get the output as **pred.**

Notice that the prediction output is an array of real numbers corresponding to the input array.

```
In [41]: ypredsvm=svm.predict(xtest)

In [42]: ypredsvm

Out[42]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Evaluation:

Finally, we need to check to see how well our model is performing on the test data. There are many evaluation techniques are there. For this, we evaluate **r2_score** produced by the model.

```
In [41]: ypredsvm=svm.predict(xtest)

In [42]: ypredsvm

Out[42]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

In [43]: ac_svm = accuracy_score(ytest,ypredsvm)

In [44]: ac_svm

Out[44]: 0.8615
```

Saving a model:

Model is saved so it can be used in future and no need to train it again.

```
In [45]: import pickle
         pickle.dump(svm,open('churn.pkl','wb'))
```

Application Building

Creating a HTML File, flask application.

Build python code

Importing Libraries

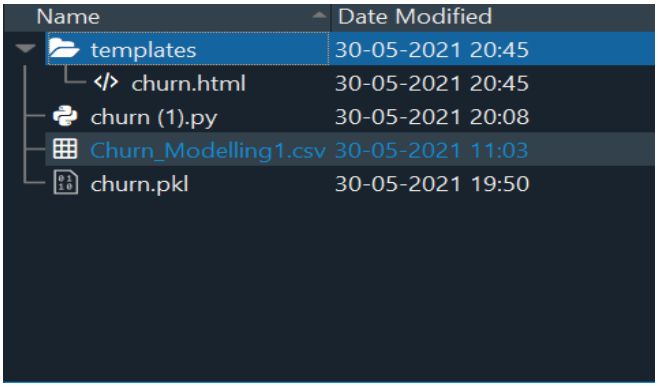
Routing to the html Page

Showcasing prediction on UI

Run The app in local browser

Project Structure:

Create a Project folder that contains files as shown below.



- We are building a Flask Application that needs HTML pages stored in the templates folder
- Templates folder contains index.html
- Static folder contains CSS and image files.

Task 1: Importing Libraries

```
from flask import Flask,render_template,request
import pickle
import numpy as np
```

Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

Task 2: Routing to the html Page

Here, declared constructor is used to route to the HTML page created earlier.
In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page is rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.
Here, "index.html" is rendered when home button is clicked on the UI

```
app=Flask(__name__)
svm=pickle.load(open('churn.pkl','rb'))

@app.route('/')
def home():
    return render_template("churn.html")


@app.route('/predict',methods=['post'])
def predict():
    Creditscore=int(request.form['CREDITSCORE'])
    Geography=int(request.form['GEOGRAPHY'])
    Gender=int(request.form['GENDER'])
    Age=int(request.form['AGE'])
    Tenure=int(request.form['TENURE'])
    Balance=int(request.form['BALANCE'])
    Numofproducts=int(request.form['NUMOFPRODUCTS'])
    Hascard=int(request.form['HASCRCARD'])
    Isactivemember=int(request.form['ISACTIVEMEMBER'])
    Estimatedsalary=int(request.form['ESTIMATEDSALARY'])

    a=np.array([[Creditscore,Geography,Gender,Age,Tenure,Balance,Numofproducts,Hascard,Isactivemember,Estimatedsalary
    print(a)

    result=svm.predict(a)

    return render_template('churn.html',x=result)
```

Task 3: Main Function

This is used to run the application in a local host.

```
if __name__ == '__main__':
    app.run()
```

Activity 3: Run the application

1. Open the anaconda prompt from the start menu.
2. Navigate to the folder where your app.py resides.
3. Now type “python app.py” command.
4. It will show the local host where your app is running on **http://127.0.0.1:5000/**
5. Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.

6. Enter the values, click on the predict button and see the result/prediction on the web page.

```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/HARINATH/OneDrive/Desktop/Flask/churn (1).py', wdir='C:/Users/
HARINATH/OneDrive/Desktop/Flask')
 * Serving Flask app "churn (1)" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Output Screen

CREDITSCORE

619

GEOGRAPHY

FRANCE

GENDER

MALE

AGE

42

TENURE

2

BALANCE

0

NUMOFPRODUCTS

1

HASCRCARD

1

ISACTIVEMEMBER

1

ESTIMATEDSALARY

101348

Submit

CREDITSCORE

GEOGRAPHY

SELECT

GENDER

SELECT

AGE

TENURE

BALANCE

NUMOFPRODUCTS

HASCRCARD

ISACTIVEMEMBER

ESTIMATEDSALARY

Submit

[0]

Findings and Suggestions

Through Exploratory Data Analysis,

- The Accuracy for logistic regression is 85.7%.
- The Accuracy for svm classifier is 84.1%.

Conclusion

In conclusion, we can see that it is possible to predict when customers will stop doing business with the company at different time points. The model will help the company be more pro-active when it comes to retaining their customers; and provide a better understanding of the reasons that drive churn.

Bibliography and Reference

1. www.kaggle.com
2. www.quora.com
3. www.wikkipedia.com