In [ ]:

In [ ]:
```
# Build an end-to-end prediction project: Artificial Neural Networks (ANNs) for Natural Language Processing (NLP) with predictation
# Showing my ANNs skills
```

In [ ]:
```
# Import Libraries
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder


# ANN
import datetime as t
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping,TensorBoard
from tensorflow.keras.models import load_model
```

In [ ]:
```
# read "csv" file
datac=pd.read_csv("ann_project.csv")
```

In [ ]:
```
# look at first 5 rows in data
datac.head()
```

In [ ]:
```
# Dataset processing

# Drop some columns (['col_1','col_2','surname'],axis=1) which do not play key roles for predictation.

datach=datac.drop(['col_1','col_2','surname'],axis=1)
datach
```

In [ ]:
```
# Use LabelEncoder() for "'Gender' column
LE_Gender=LabelEncoder()
datach['Gender']=LE_Gender.fit_transform(datach['Gender'])
datach
```

In [ ]:
```
# save results from =LabelEncoder() used on 'Gender' column
with open('LE_Gender.pkl','wb') as gen:
    pickle.dump(LE_Gender,gen)
```

In [ ]:
```
# use OneHotEncoder () for 'Geography' column
OHEG=OneHotEncoder()
oheg=OHEG.fit_transform(datach[['Geography']]).toarray()
oheg
```

In [ ]:
```
# save results from OneHotEncoder used on 'Geography' column
with open('OHEG.pkl','wb') as geo:
    pickle.dump(OHEG,geo)
```

In [ ]:
```
# Find new feature names in 'Geography' column
OHEG.get_feature_names_out(['Geography'])
```

In [ ]:
```
# use OneHotEncoder () features to create a DataFrame
geography_datframe = pd.DataFrame(oheg,columns=OHEG.get_feature_names_out(['Geography']))
geography_datframe
```

In [ ]:
```
# deop 'Geography' column from the original dataset, and then add geography_datframe
data_concat=pd.concat([datach.drop('Geography',axis=1),geography_datframe],axis=1)
```

In [ ]:
```
data_concat.head()
```

In [ ]:
```
#  use the "data_concat" dataset to find the independent (y) columns and dependent (x) columns.

x=data_concat.drop('Exited',axis=1)                #   Independent features
y=data_concat['Exited']                            #  Dependent feature (target)

#print(x)

#print(y)
```

In [ ]:
```
# Dive the dataset into in train and test sets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=123)
```

In [ ]:
```
# Use StandardScaler() to scale x and y columns.
standard_scaler=StandardScaler()

x_train=standard_scaler.fit_transform(x_train)

x_test=standard_scaler.transform(x_test)        # transform() uses results received from the fit_transform() on x_train.
```

In [ ]:
```
# store the fitted results (mean, and standard deviation) received from x_train datasets.

with open('standard_scaler.pkl','wb') as ss:
    pickle.dump(standard_scaler, ss)
```

Artificial Neural Network (ANN) model

In [ ]:
```
# Finding number of featurers.
(x_train.shape[1],)
```

In [ ]:
```
# making an ANN model

model = Sequential([Input(shape=(x_train.shape[1],)), Dense(128, activation='relu'), Dense(64, activation='relu'), Dense(1, activation='sigmoid')])
```

In [ ]:
```
# model summary
model.summary()
```

In [ ]:
```
# Use Optimizer to update weights in model.

optimizer_ann=tf.keras.optimizers.Adam(learning_rate=0.01)          # use default learning rate
optimizer_ann
```

In [ ]:
```
# find error
loss_ann=tf.keras.losses.BinaryCrossentropy()
loss_ann
```

In [ ]:
```
# complie the model with optimize, loss, and accuracy.

model.compile(optimizer=optimizer_ann,loss= loss_ann,metrics=['accuracy'])
```

In [ ]:
```
# Build the Tensorboard for monituring pgogress

import datetime

log_dir_fit="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

tensorboard_fit=TensorBoard(log_dir=log_dir_fit,histogram_freq=1)
```

In [ ]:
```
# Set up Early Stopping to stop overfitting
early_stopping_fit=EarlyStopping(monitor='val_loss',patience=10,restore_best_weights=True)
```

In [ ]:
```
# run the model
train_model=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=200, callbacks=[tensorboard_fit, early_stopping_fit])
```

In [ ]:
```
# Save model
model.save('ann_model.h5')
```

In [ ]:
```
# For visualization, load tensorboard
%load_ext tensorboard
```

In [ ]:
```
# Run tensorboard on port (6007)
%tensorboard --logdir logs/fit --port=6008
```

Prediction

In [ ]:
```
# input example dataset

sample = {
    'CreditScore': 800,
    'Geography': 'Spain',
    'Gender': 'Female',
    'Age': 46,
    'Tenure': 8,
    'Balance': 100000,
    'NumOfProducts': 4,
    'HasCrCard': 0,
    'IsActiveMember': 0,
    'EstimatedSalary': 110000
}
```

In [ ]:
```
# Load the trained model
model=load_model('ann_model.h5')
```

In [ ]:
```
# load saved files

# geography
with open('OHEG.pkl','rb') as geo:
    LE_geo=pickle.load(geo)

# gender
with open('LE_Gender.pkl','rb') as gen:
    LE_gen = pickle.load(gen)

# scalar
with open('standard_scaler.pkl','rb') as ss:
    scal = pickle.load(ss)
```

In [ ]:
```
# One-hot encode 'Geography'
sample_geography = pd.DataFrame({'Geography': [sample['Geography']]})
```

In [ ]:
```
sample_geography_encoded = OHEG.transform(sample_geography).toarray()
```

In [ ]:
```
sample_geography_encoded_data = pd.DataFrame(sample_geography_encoded, columns=OHEG.get_feature_names_out(['Geography']))
sample_geography_encoded_data
```

In [ ]:
```
# Use input (sample) data to create DataFrame
sample_data=pd.DataFrame([sample])
sample_data
```

In [ ]:
```
# Encode 'Gender' Using LabelEncoder
sample_data['Gender']=LE_Gender.transform(sample_data['Gender'])
sample_data
```

In [ ]:
```
# concatinate one hot encoded "Geography" to input DataFrame.

sample_data=pd.concat([sample_data.drop("Geography",axis=1), sample_geography_encoded_data],axis=1)
sample_data
```

In [ ]:
```
# Scaling (normalizing) the input data

sample_data_scale=standard_scaler.transform(sample_data)
sample_data_scale
```

In [ ]:
```
# Predict sample
sample_data_prediction=model.predict(sample_data_scale)
sample_data_prediction
```

In [ ]:
```
#  sample prediction probability
if sample_data_prediction > 0.5:
    print('customer is going to leave.')
else:
    print('customer is going to stay.')
```

In [ ]:

In [ ]:
```
"""
conclusion:

Designed and trained a customer churn prediction model using an ANN in  TenserFlow/Keras, resulting ~ 87% validation accuracy with early stopping
on a dataset. Built an end-to-end prediction system that results   ~99% churn probability for  a sample case using persisted encoders and scalers
with scikit-learn.

"""
```

In [ ]:

In [ ]: