**Preparing Environment for creating a Deep learning Model with Dental Clinic Data**

**By - Aaditya Balakrishnan**

Step 1 Installed Python v 3.11.4

Step 2 Installed Visual Studio code

Step 3 Installed Anaconda installer for jupyter notebook

Step 4 Installed Python extension in vs code

Step 5 Setting up virtual environment

***Entered command in terminal***

conda create --name yourenvname python=3.11.4

conda env list

conda install ipykernel

conda install pandas

conda install matplotlib

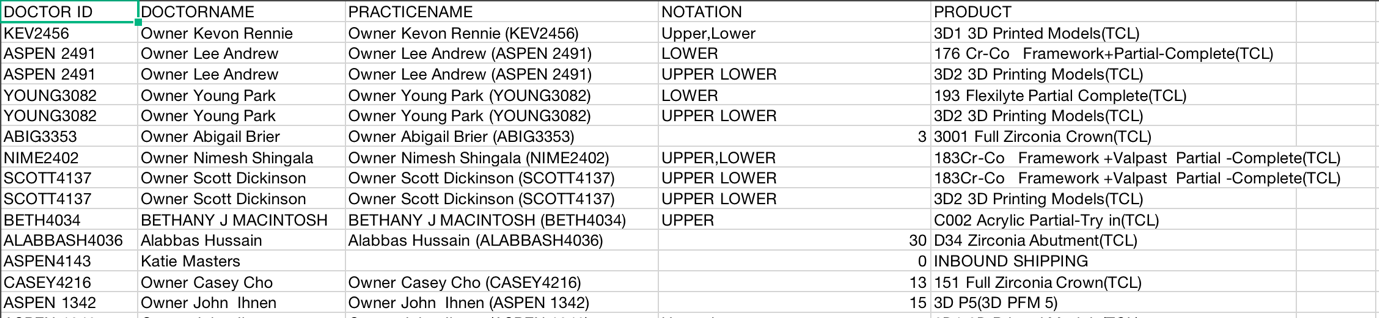
conda install Tensorflow

conda install sklearn

conda install plotly

**Imported the given file – DOCS.csv**

**Sample data**



***Coding in VS code***

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import pandas as pd

import tensorflow as tf

from tensorflow.keras import layers, models

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

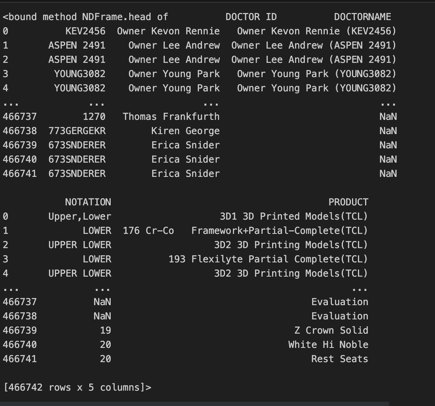
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from tensorflow.keras import layers, models

**(importing file in csv format)**

data = pd.read\_csv("DOCS.csv")

print (data.head)



data = data.dropna(how='any')

print(data.shape)

**(Changing values in number format by encoding label)**

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

data['DOCTORNAME']=le.fit\_transform(data['DOCTORNAME'])

data['PRACTICENAME']=le.fit\_transform(data['PRACTICENAME'])

data['NOTATION']=le.fit\_transform(data['NOTATION'])

data['PRODUCT']=le.fit\_transform(data['PRODUCT'])

data.head()

A black screen with white text

Description automatically generated

**(Changing string to INT)**

def handle\_non\_numerical\_data(data):

columns = data.columns.values

for column in columns:

text\_digit\_vals = {}

def convert\_to\_int(val):

return text\_digit\_vals[val]

if data[column].dtype != np.int64 and data[column].dtype != np.float64:

column\_contents = data[column].values.tolist()

unique\_elements = set(column\_contents)

x = 0

for unique in unique\_elements:

if unique not in text\_digit\_vals:

text\_digit\_vals[unique] = x

x+=1

data[column] = list(map(convert\_to\_int, data[column]))

return data

data = handle\_non\_numerical\_data(data)

print(data.head())

A black screen with white text

Description automatically generated

**(One way)**

(Perfforming Logistic regression by separating data into train and test model)

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import accuracy\_score

import time

x=data.loc[:,"NOTATION"].values

y=data.loc[:,"PRODUCT"].values

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

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

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

sc\_y = StandardScaler()

x\_train=np.array(x\_train).reshape(-1,1)

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

y\_train=np.array(y\_train).reshape(-1,1)

x\_test = np.array(x\_test).reshape(-1,1)

x\_test = sc\_x.fit\_transform(x\_test)

y\_test\_org = y\_test

y\_test = np.array(y\_test).reshape(-1,1)

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

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

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

**(Alternative way)**

def load\_data\_from\_excel(file\_path, target\_column):

# Assuming the first row contains the column names

data = pd.read\_excel(file\_path, engine='openpyxl')

# Separate features and labels

features = data.drop(columns=[target\_column])

labels = data[target\_column]

# Convert labels to one-hot encoded format (if needed)

# For binary classification, you can use `pd.get\_dummies(labels)` as well

labels = pd.get\_dummies(labels)

return features, labels

# Replace 'data.xlsx' with the path to your Excel file and 'target\_column\_name' with the name of the target column.

file\_path = 'DOCS.xlsx'

target\_column\_name = 'PRODUCT'

features, labels = load\_data\_from\_excel(file\_path, target\_column\_name)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

def create\_deep\_learning\_model(input\_shape, num\_classes):

model = models.Sequential([

layers.Dense(64, activation='relu', input\_shape=input\_shape),

layers.Dense(128, activation='relu'),

layers.Dropout(0.5),

layers.Dense(num\_classes, activation='softmax')

])

return model

# Get the number of features and classes for input shape and output layer size

num\_features = X\_train.shape[1]

num\_classes = y\_train.shape[1]

model = create\_deep\_learning\_model(input\_shape=(num\_features,), num\_classes=num\_classes)

def compile\_model(model, learning\_rate=0.001):

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss='categorical\_crossentropy',

metrics=['accuracy'])

compile\_model(model)

def compile\_model(model, learning\_rate=0.001):

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss='categorical\_crossentropy',

metrics=['accuracy'])

compile\_model(model)