Configuration Manual

MSc Research Project
Data Analytics

Mubeen Ali Mohammed
Student ID: X18180370

School of Computing
National College of Ireland

Supervisor: Dr Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Mubeen Ali Mohammed
Student ID:	X18180370
Programme:	Data Analytics
Year:	2019-2020
Module:	MSc Research Project
Supervisor:	Dr Muhammad Iqbal
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Mubeen Ali Mohammed
Date:	27th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mubeen Ali Mohammed
X18180370

1 Introduction

This manual lists all the Software , Hardware requirements and the underlying code used to implement the research project titled
"Alzheimer Disease Detection and Prognosis from Clinical Data using Machine Learning Techniques"

2 System Configuration

2.1 Hardware

RAM : 8GB minimum(25.51GB GPU , TPU available on Google Colab Pro)
GPU : T4 and P100
System OS : Windows 10
Hard Disk Storage : 100GB (Google Drive Storage)

2.2 Software

Software Computing Tools Used : Python 3 Jupyter Notebook (Google Colab), Overleaf, Microsoft Excel, Tableau
Browser Engine : Google Chrome/ Firefox
Email : Gmail login to access Colab Pro.

3 Project Development

As mentioned we perform major steps in Design process Stage11: Data Collection Stage 2: Data Pre-Processing Stage 3: Building Regression and Classification models Stage 4: Evaluation of Models

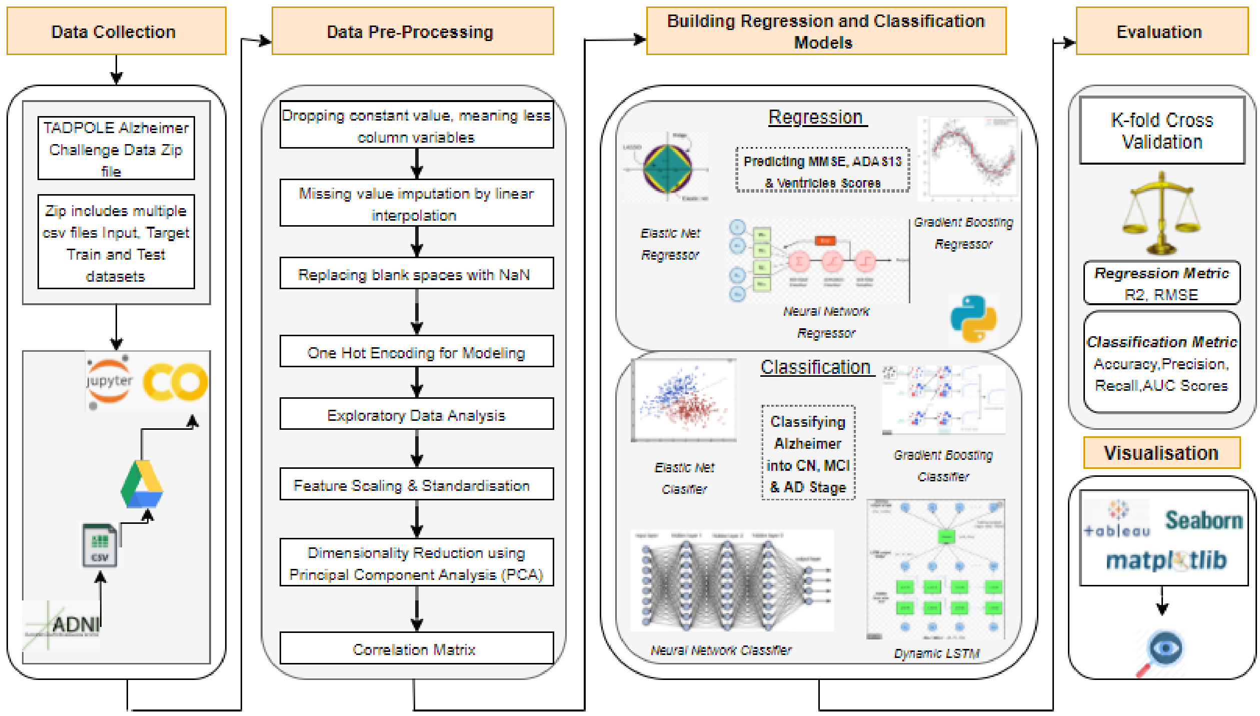


Figure 1: Alzheimer Disease Detection - Design Flow

3.1 Data Collection

The dataset name is TADPOLE which is downloadable from ADNI website as shown below we have to apply for ADNI database access after we can download Alzheimer's TADPOLE data using the link [1] which looks like this

[1] https://ida.loni.usc.edu/login.jsp?project=ADNI#

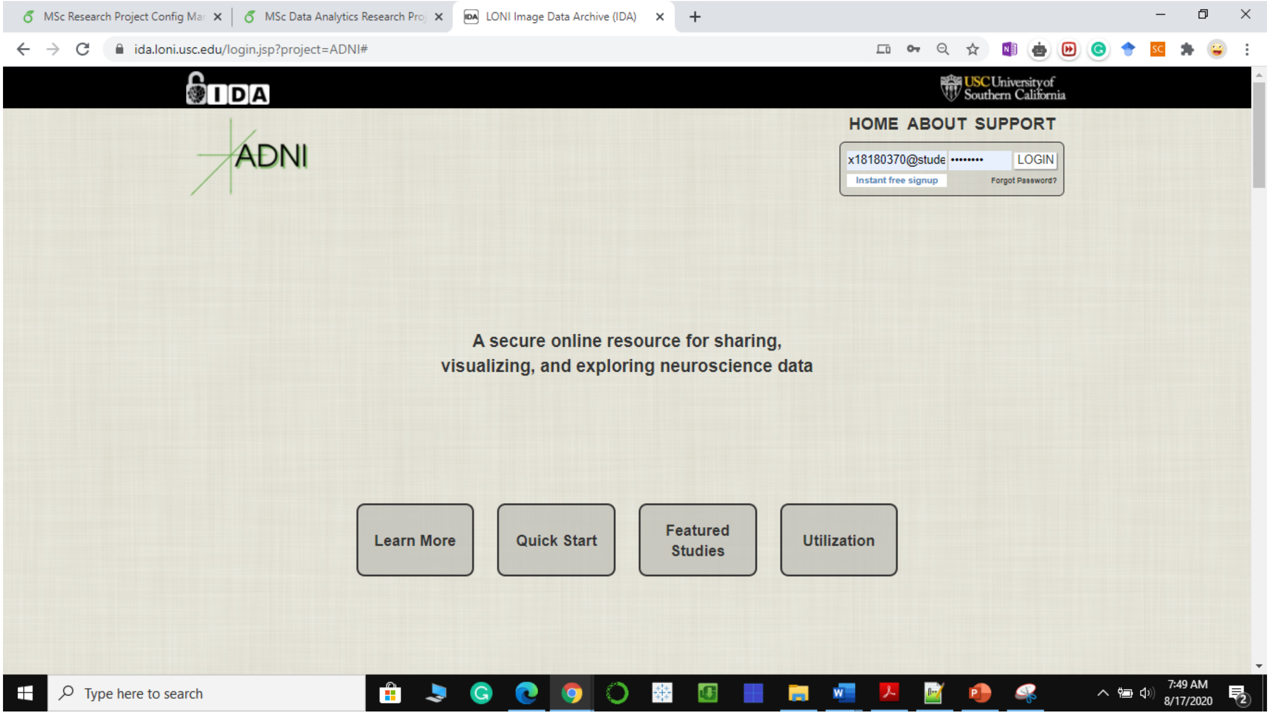


Figure 2: ADNI Login with credentials

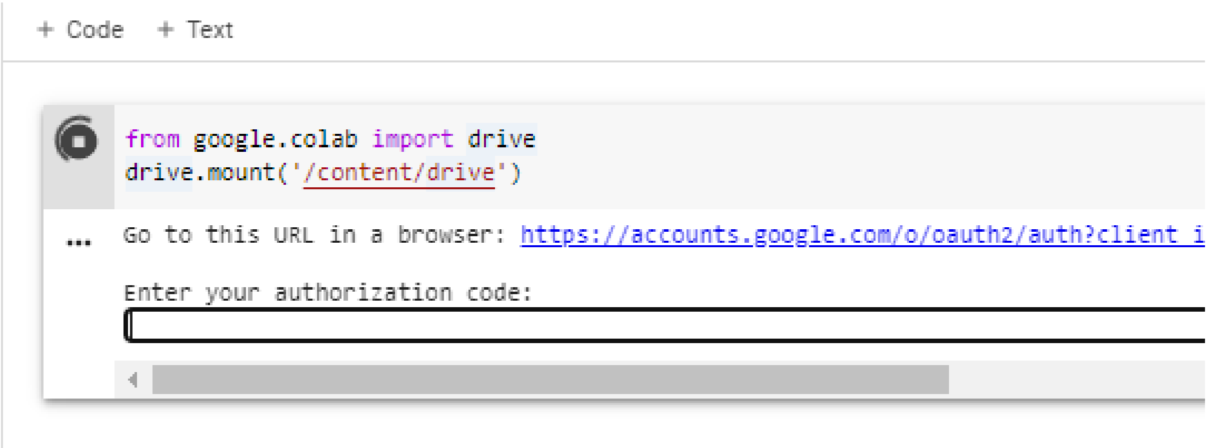


Figure 3: Mounting Google Drive in Colab

Click on URL and input authorisation code to mount drive and use Input data files in drive

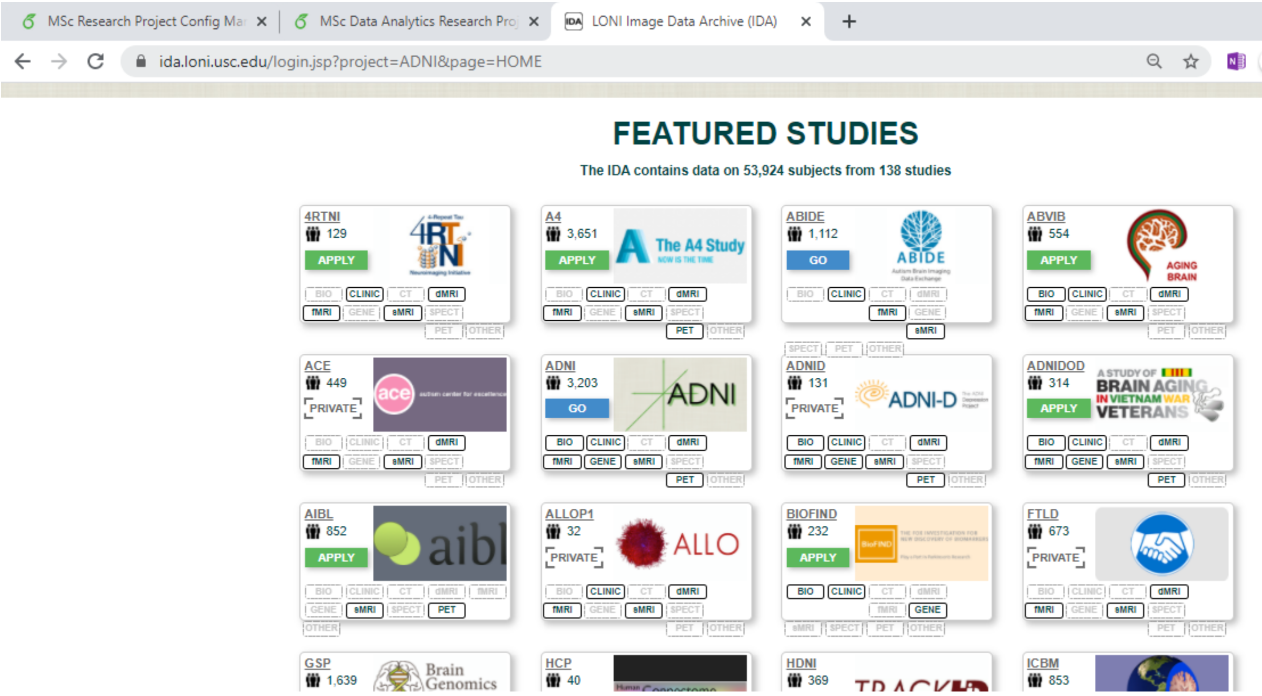


Figure 4: LONI Databases list

Shows all repository access , we need to select ADNI database and click on "Go" option

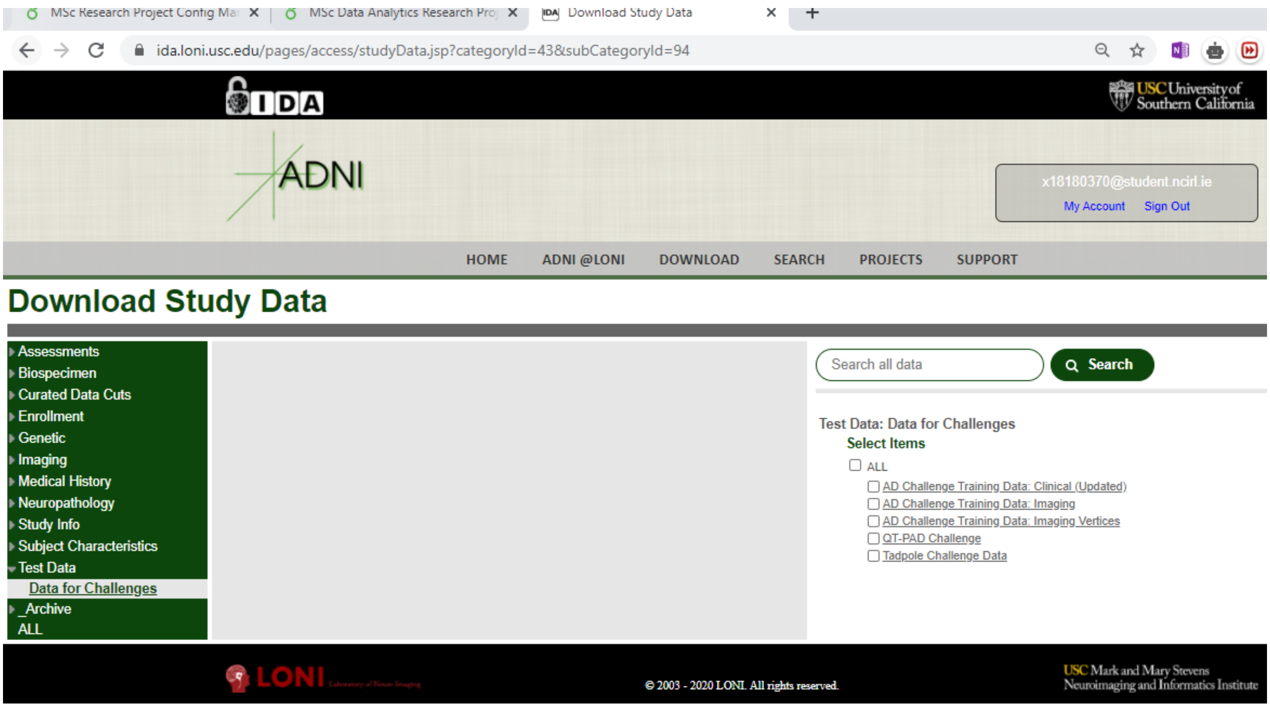


Figure 5: Selection of TADPOLE Challenge data

It shows under Download/ test data, we need to click on Tadpole data and load in Google Drive

```python
import numpy as np
import sklearn
import pandas as pd
import xgboost as xgb
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import ensemble,tree,linear_model
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import OneHotEncoder,MinMaxScaler, StandardScaler
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, ExtraTreesClassifier,GradientBoostingRegressor)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc, accuracy_score
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split,cross_val_score
import tensorflow as tf
import warnings
import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls

py.init_notebook_mode(connected=True)
%reload_ext autoreload
%autoreload 2
%matplotlib inline
warnings.filterwarnings('ignore')

pd.options.display.max_columns=99
```

Figure 6: List of Important libraries imported

sklearn is the most important Machine learning and deep learning library along with visualisation libraries seaborn, matplotlib are used

Data Cleaning by dropping constant value coloumns

```
#Dropping coloumns with mostly unique constant values
# format date, drop constant columns + null PTID rows
def Input_prep(data):
    data['EXAMDATE'] = pd.to_datetime(data['EXAMDATE'], errors='coerce')
    data['EXAMDATE_bl'] = pd.to_datetime(data['EXAMDATE_bl'])

    # We will remove all columns where we have a unique value (constants)
    # It is useful because this columns don't give us none information
    discovering_consts = [col for col in data.columns if data[col].nunique() == 1]
    # printing the total of columns dropped
    print(len(discovering_consts),"columns are dropped ")
    # Get the shape of the processed dataset
    data = data.drop(discovering_consts,axis=1)
    print("After dropping constants, the shape of the data set is:",data.shape)
    data=data.dropna(subset=['PTID_Key'])
    print("After dropping missing PTID_Key, the shape of the dataset is: ", data.shape)
    return data
```

Dropping Coloumns with null values more than 60% threshold

```
[ ] # Drop most null columns, threshold in %
    def drop_maj(data, threshold):
        total = data.isnull().sum().sort_values(ascending = False) # getting the sum of null values and ordering
        percent = (data.isnull().sum() / data.isnull().count() * 100 ).sort_values(ascending = False) #getting the
        df = pd.concat([total, percent], axis=1, keys=['Total', 'Percent']) # Concatenating the total and percent
        most_nan= [idx for idx in df[~(df['Total'] == 0)].index if df[~(df['Total'] == 0)].loc[idx,'Percent']>thre
        print("There are ", len(most_nan),"columns with ",threshold, "% missing values")
        data = data.drop(most_nan,axis=1)
        print("After dropping most null columns, the shape of the dataset is: ", data.shape)
        return data
```

```
[ ] Input_Data=Input_prep(Input_Data)
    Input_Data = drop_maj(Input_Data,60)
```

```
166 columns are dropped
After dropping constants, the shape of the data set is: (8717, 1726)
After dropping missing PTID_Key, the shape of the dataset is:  (8715, 1726)
There are  1010 columns with  60 % missing values
After dropping most null columns, the shape of the dataset is:  (8715, 716)
```

Figure 7: Dropping redundant coloumns

+ Code + Text

```
[ ] # Get x,y for training data
data_con = ['ADAS13','Ventricles_Norm','MMSE']
data_cat=['CN_Diag','MCI_Diag','AD_Diag']
train_final=prep_for_models(Input_Data,train_proc)
x_train=train_final.drop(data_con,axis=1).drop(data_cat,axis=1)
y_train_adas13=train_final['ADAS13']
y_train_ventricles=train_final['Ventricles_Norm']
y_train_mmse=train_final['MMSE']
# Diagnosis
y_train_diag = train_final[['CN_Diag','MCI_Diag','AD_Diag']]
y_train_diag['CN_Diag'] = y_train_diag['CN_Diag'].astype('int')
y_train_diag['MCI_Diag'] = y_train_diag['MCI_Diag'].astype('int')
y_train_diag['AD_Diag'] = y_train_diag['AD_Diag'].astype('int')
# Encode one-hot encoding back to label encoding (0: CN_Diag, 1: MCI_Diag, 2: AD_Diag)
y_train_diag['Diag'] = np.argmax(y_train_diag[['CN_Diag','MCI_Diag','AD_Diag']].values,axis=1)
y_train_diag1 = y_train_diag['Diag']
y_train_diag2 = y_train_diag[['CN_Diag','MCI_Diag','AD_Diag']]
```

```
After merging input and output, the shape of the data is:  (6716, 288)
Adding the month interval, the shape of the data is:  (6716, 289)
Removing PTID_Key and Date columns, the shape of the data is:  (6716, 286)
```

```
# Get x,y for validation data
data_con = ['ADAS13','Ventricles_Norm','MMSE']
data_cat=['CN_Diag','MCI_Diag','AD_Diag']
val_final=prep_for_models(Input_Data,val_proc)
x_val=val_final.drop(data_con,axis=1).drop(data_cat,axis=1)
y_val_adas13=val_final['ADAS13']
y_val_ventricles=val_final['Ventricles_Norm']
y_val_mmse=val_final['MMSE']
# Diagnosis
y_val_diag = val_final[['CN_Diag','MCI_Diag','AD_Diag']]
y_val_diag['CN_Diag'] = y_val_diag['CN_Diag'].astype('int')
y_val_diag['MCI_Diag'] = y_val_diag['MCI_Diag'].astype('int')
y_val_diag['AD_Diag'] = y_val_diag['AD_Diag'].astype('int')
# Encode one-hot encoding back to label encoding (0: CN_Diag, 1: MCI_Diag, 2: AD_Diag)
y_val_diag['Diag'] = np.argmax(y_val_diag[['CN_Diag','MCI_Diag','AD_Diag']].values,axis=1)
y_val_diag1 = y_val_diag['Diag']
y_val_diag2 = y_val_diag[['CN_Diag','MCI_Diag','AD_Diag']]
```

```
After merging input and output, the shape of the data is:  (2238, 288)
Adding the month interval, the shape of the data is:  (2238, 289)
Removing PTID_Key and Date columns, the shape of the data is:  (2238, 286)
```

Figure 8: creating train and validation data split for ML

3.2 Machine Learning Models

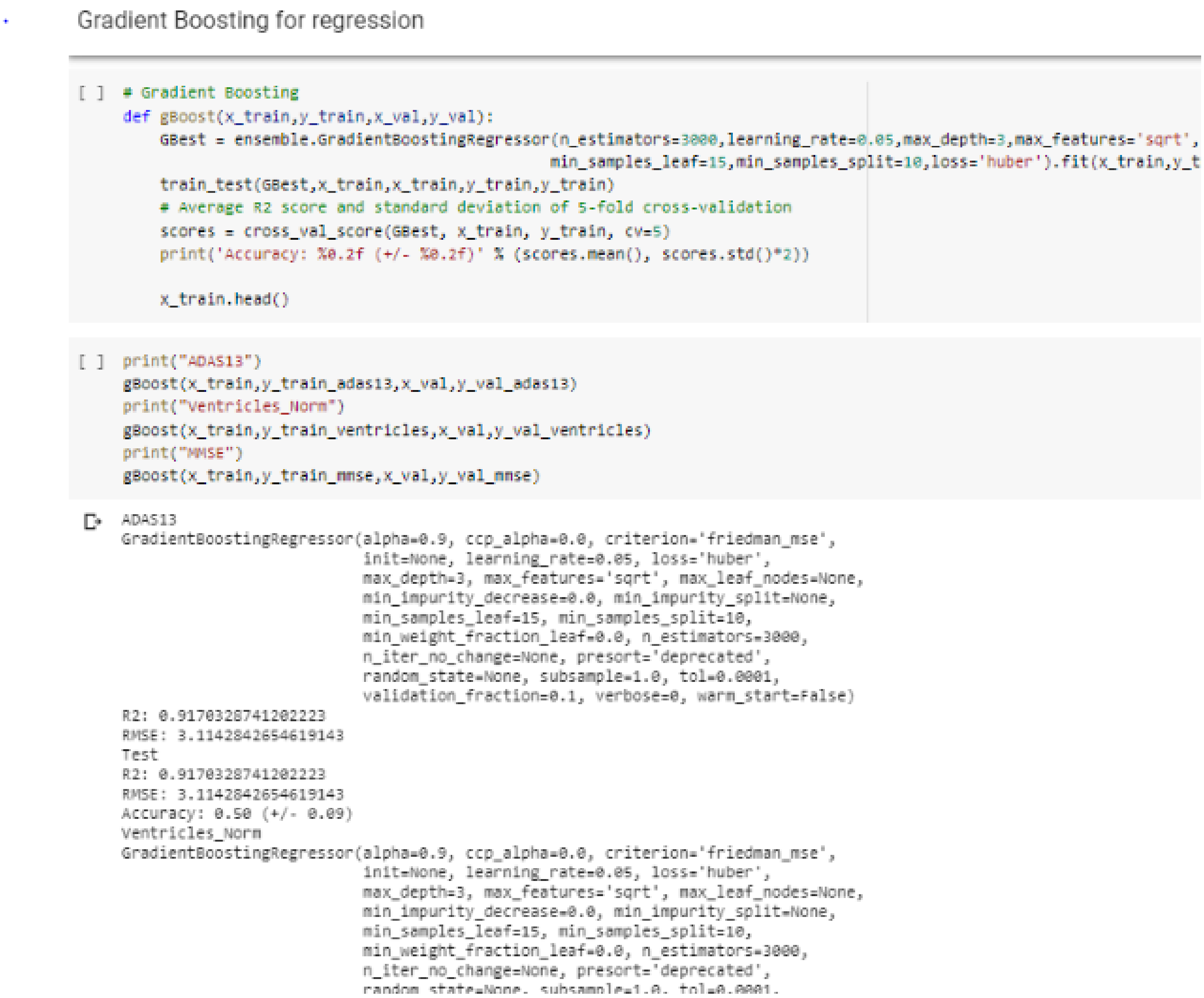
Elastic Net Regressor

```
[ ] # Elastic Net
    def eNet(x_train,y_train,x_val,y_val):
        ENSTest = linear_model.ElasticNetCV(alphas=[  ],l1_ratio=[.01, .1, .5, .9, .99], max_iter=5000
        train_test(ENSTest, x_train,x_val,y_train,y_val)

        # Average R2 score and standard deviation of 5-fold cross-validation
        scores = cross_val_score(ENSTest, x_train, y_train, cv=5)
        print('Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std()*2))
```

```
[ ] print("ADAS13")
    eNet(x_train,y_train_adas13,x_val,y_val_adas13)
    print("Ventricles_Norm")
    eNet(x_train,y_train_ventricles,x_val,y_val_ventricles)
    print("MMSE")
    eNet(x_train,y_train_mmse,x_val,y_val_mmse)
```

```
ADAS13
ElasticNetCV(alphas=[0.0001, 0.0005, 0.001, 0.01, 0.1, 1, 10], copy_X=True,
             cv=None, eps=0.001, fit_intercept=True,
             l1_ratio=[0.01, 0.1, 0.5, 0.9, 0.99], max_iter=5000, n_alphas=100,
             n_jobs=None, normalize=False, positive=False, precompute='auto',
             random_state=None, selection='cyclic', tol=0.0001, verbose=0)
R2: 0.7637561972169721
RMSE: 4.761452397771426
Test
R2: 0.1272502499490319
RMSE: 7.763820055598533
Accuracy: 0.58 (+/- 0.22)
Ventricles_Norm
ElasticNetCV(alphas=[0.0001, 0.0005, 0.001, 0.01, 0.1, 1, 10], copy_X=True,
             cv=None, eps=0.001, fit_intercept=True,
             l1_ratio=[0.01, 0.1, 0.5, 0.9, 0.99], max_iter=5000, n_alphas=100,
             n_jobs=None, normalize=False, positive=False, precompute='auto',
             random_state=None, selection='cyclic', tol=0.0001, verbose=0)
R2: 0.9450985570370511
RMSE: 0.002630125951292067
Test
R2: 0.8349707159800148
RMSE: 0.004732969995855007
Accuracy: 0.80 (+/- 0.07)
MMSE
ElasticNetCV(alphas=[0.0001, 0.0005, 0.001, 0.01, 0.1, 1, 10], copy_X=True,
             cv=None, eps=0.001, fit_intercept=True,
```

Figure 9: Elastic Net regressor to predict MMSE, Ventricles Norm, ADAS13



Gradient Boosting for regression

```
[ ] # Gradient Boosting
    def gBoost(x_train,y_train,x_val,y_val):
        GBest = ensemble.GradientBoostingRegressor(n_estimators=3000,learning_rate=0.05,max_depth=3,max_features='sqrt',
                                                   min_samples_leaf=15,min_samples_split=10,loss='huber').fit(x_train,y_t
        train_test(GBest,x_train,x_train,y_train,y_train)
        # Average R2 score and standard deviation of 5-fold cross-validation
        scores = cross_val_score(GBest, x_train, y_train, cv=5)
        print('Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std()*2))

        x_train.head()
```

```
[ ] print("ADAS13")
    gBoost(x_train,y_train_adas13,x_val,y_val_adas13)
    print("Ventricles_Norm")
    gBoost(x_train,y_train_ventricles,x_val,y_val_ventricles)
    print("MMSE")
    gBoost(x_train,y_train_mmse,x_val,y_val_mmse)
```

```
ADAS13
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.05, loss='huber',
                          max_depth=3, max_features='sqrt', max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=15, min_samples_split=10,
                          min_weight_fraction_leaf=0.0, n_estimators=3000,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
R2: 0.9170328741202223
RMSE: 3.1142842654619143
Test
R2: 0.9170328741202223
RMSE: 3.1142842654619143
Accuracy: 0.50 (+/- 0.09)
Ventricles_Norm
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.05, loss='huber',
                          max_depth=3, max_features='sqrt', max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=15, min_samples_split=10,
                          min_weight_fraction_leaf=0.0, n_estimators=3000,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
```

Figure 10: Gradient Boosting Regressor model building process

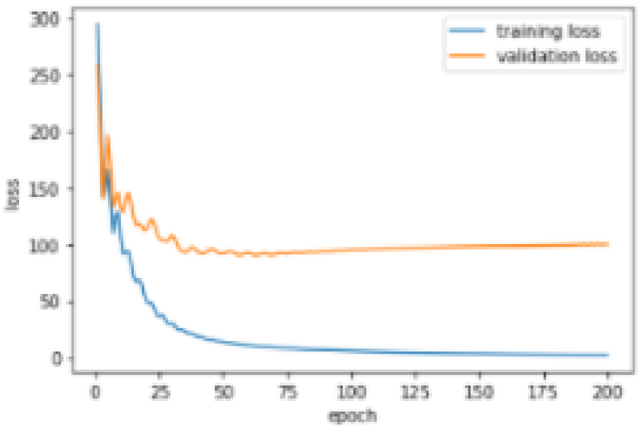
+ Code + Text

```python
def NN(x_train,y_train,x_val,y_val):
    # Convert y_train shape to ?x1
    y_train = y_train.values.reshape(-1,1)
    y_val = y_val.values.reshape(-1,1)

    n_input = x_train.shape[1]
    n_hidden1 = 128
    n_hidden2 = 512
    n_hidden3 = 1024
    n_output = 1
    learning_rate = 0.001
    epochs = 200
    batch_size = 25
    REGULARIZATION_RATE = 0.0001

    X = tf.compat.v1.placeholder(tf.float32,[None,n_input])
    y_gt = tf.compat.v1.placeholder(tf.float32,[None,n_output])


    initializer = tf.contrib.layers.variance_scaling_initializer(factor=2.0, mode='FAN_I
    W1 = tf.Variable(initializer([n_input,n_hidden1]))
    b1 = tf.Variable(tf.constant(0.1,shape=[n_hidden1]))
    H1 = tf.nn.relu(tf.matmul(X,W1)+b1)

    W2 = tf.Variable(initializer([n_hidden1,n_hidden2]))
    b2 = tf.Variable(tf.constant(0.1,shape=[n_hidden2]))
    H2 = tf.nn.relu(tf.matmul(H1,W2)+b2)

    W3 = tf.Variable(initializer([n_hidden2,n_hidden3]))
    b3 = tf.Variable(tf.constant(0.1,shape=[n_hidden3]))
    H3 = tf.nn.relu(tf.matmul(H2,W3)+b3)

    W_out = tf.Variable(initializer([n_hidden3,n_output]))
    b_out = tf.Variable(tf.constant(0.1,shape=[n_output]))
    y_pred = tf.matmul(H3,W_out)+b_out

    tr_losses=[]
    te_losses=[]
    loss = tf.reduce_mean(tf.losses.mean_squared_error(labels=y_gt,predictions=y_pred))
    optimizer = tf.train.AdamOptimizer(learning_rate)
    train_step = optimizer.minimize(loss)

    sess = tf.InteractiveSession()
    tf.global_variables_initializer().run()
```

Figure 11: Neural net regressor model building process

```
Iter 104, training loss 5.321993, validation loss 95.397934
Iter 105, training loss 5.256169, validation loss 95.605354
Iter 106, training loss 5.194320, validation loss 95.787552
Iter 107, training loss 5.132657, validation loss 95.825951
Iter 108, training loss 5.073289, validation loss 95.866982
Iter 109, training loss 5.012788, validation loss 96.012733
Iter 110, training loss 4.955192, validation loss 96.104019
Iter 111, training loss 4.898249, validation loss 96.092224
Iter 112, training loss 4.842688, validation loss 96.141586
```

```
tr_loss, te_loss=plot_NN(epochs, tr_losses, te_losses)
print("Best epoch number is: ", te_loss.index(min(te_loss)))
print("The lowest validation loss is: ", min(te_loss))
print("At the same epoch, the training loss is: ", tr_loss[te_loss.index(min(te_loss))])
```




```
Best epoch number is:  62
The lowest validation loss is:  9.511844679607862
At the same epoch, the training loss is:  3.1821937525160604
```

Figure 12: Neural net regressor loss vs Epoch

Suport Vector Classifier

```
[ ] svc = SVC(probability=True)
    svc.fit(x_train_diag,y_train_diag)
    scores = cross_val_score(svc, x_train_diag, y_train_diag, cv=5)
    print('Training Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std()*2))
    y_validation_pred = svc.predict(x_val_diag)

    acc = accuracy_score(y_validation_pred,y_val_diag)
    print('Validation Accuracy: %0.2f (+/- %0.2f)' % (acc.mean(), acc.std()*2))
```

```
Training Accuracy: 0.75 (+/- 0.04)
Validation Accuracy: 0.61 (+/- 0.00)
```

```
cn_cls,mci_cls,ad_cls,cn_pred,mci_pred,ad_pred = transform(y_val_diag,y_validat
metrics('CN_Diag',svc,x_val_diag,cn_cls,cn_pred)
print('*'*30)
metrics('MCI_Diag',svc,x_val_diag,mci_cls,mci_pred)
print('*'*30)
metrics('AD_Diag',svc,x_val_diag,ad_cls,ad_pred)
```

```
CN_Diag Accuracy: 74.71%
CN_Diag Precision: 73.01%
CN_Diag Recall: 50.72%
CN_Diag AUC: 33.25%
******************************
MCI_Diag Accuracy: 64.16%
MCI_Diag Precision: 56.15%
MCI_Diag Recall: 80.27%
MCI_Diag AUC: 72.68%
******************************
AD_Diag Accuracy: 82.22%
AD_Diag Precision: 56.51%
AD_Diag Recall: 35.10%
AD_Diag AUC: 39.36%
```

Figure 13: Support Vector classifier modelling process

Neural Network for Classification

```
x_train = x_train_diag
y_train = y_train_diag2.values.reshape(-1,3)

x_val = x_val_diag
y_val = y_val_diag2.values.reshape(-1,3)
```

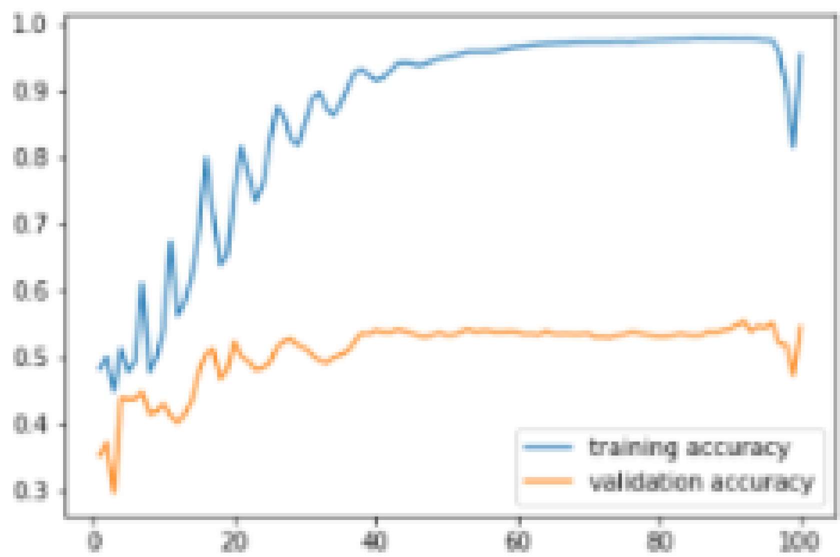
```
n_input = x_train.shape[1]
n_hidden1 = 128
n_hidden2 = 512
n_hidden3 = 1024
n_output = 3
learning_rate = 0.001
epochs = 100     #100,200,100
batch_size = 32  #100,32,25

x = tf.placeholder(tf.float32,[None,n_input])
y_gt = tf.placeholder(tf.float32,[None,n_output])

initializer = tf.contrib.layers.variance_scaling_initializer(factor=2.0, mode='FAN_IN', uniform=F
W1 = tf.Variable(initializer([n_input,n_hidden1]))
b1 = tf.Variable(tf.constant(0.1,shape=[n_hidden1]))
H1 = tf.nn.relu(tf.matmul(x,W1)+b1)

W2 = tf.Variable(initializer([n_hidden1,n_hidden2]))
b2 = tf.Variable(tf.constant(0.1,shape=[n_hidden2]))
H2 = tf.nn.relu(tf.matmul(H1,W2)+b2)

W3 = tf.Variable(initializer([n_hidden2,n_hidden3]))
b3 = tf.Variable(tf.constant(0.1,shape=[n_hidden3]))
H3 = tf.nn.relu(tf.matmul(H2,W3)+b3)

W_out = tf.Variable(initializer([n_hidden3,n_output]))
b_out = tf.Variable(tf.constant(0.1,shape=[n_output]))
y_pred = tf.matmul(H3,W_out)+b_out

loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_gt,logits=y_pred))
optimizer = tf.train.AdamOptimizer(learning_rate)
train_step = optimizer.minimize(loss)
```

Figure 14: Neural Net Classifier model building

3 hidden layers with neurons 128,512,1024 is built with output=3 AD classes, lowest learning rate=0.001, epochs=100, activation=ReLU, optimiser=Adam

```
Iter 61, training loss 0.082806, training accuracy 0.967094
Iter 61, validation loss 1.643980, validation accuracy 0.534853
Iter 62, training loss 0.078913, training accuracy 0.968285
Iter 62, validation loss 1.661281, validation accuracy 0.532618
Iter 63, training loss 0.075050, training accuracy 0.969178
Iter 63, validation loss 1.688395, validation accuracy 0.538874
Iter 64, training loss 0.072375, training accuracy 0.969476
```

```
import matplotlib.pyplot as plt
epoch=list(range(1,epochs+1))
plt.figure()
plt.plot(epoch,tr_losses,label='training accuracy')
plt.plot(epoch,te_losses,label='validation accuracy')
plt.legend()
plt.show()

print("Best epoch number is: ", te_losses.index(max(te_losses))
print("The hightest validation accuracy is: ", max(te_losses))
print("At the same epoch, the training accuracy is: ", tr_losse
```




```
Best epoch number is:  91
The hightest validation accuracy is:  0.5536193
At the same epoch, the training accuracy is:  0.97721857
```

Figure 15: neural net classifier accuracy

Gradient Boosting Classifier

```
[ ] GBoost = GradientBoostingClassifier(n_estimators=3000,learning_rate=0.05,max_depth=3,max_features='sqrt'
                                        min_samples_leaf=15,min_samples_split=10)
    GBoost.fit(x_train_diag,y_train_diag)
    scores = cross_val_score(GBoost, x_train_diag, y_train_diag, cv=5)
    print('Training Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std()*2))
    y_validation_pred = GBoost.predict(x_val_diag)
    acc = accuracy_score(y_validation_pred,y_val_diag)
    print('Validation Accuracy: %0.2f (+/- %0.2f)' % (acc.mean(), acc.std()*2))
```

```
Training Accuracy: 0.84 (+/- 0.07)
Validation Accuracy: 0.73 (+/- 0.00)
```

```
[ ] cn_cls,mci_cls,ad_cls,cn_pred,mci_pred,ad_pred = transform(y_val_diag,y_validation_pred)
    metrics('CN_Diag',GBoost,x_val,cn_cls,cn_pred)
    print('*'*30)
    metrics('MCI_Diag',GBoost,x_val,mci_cls,mci_pred)
    print('*'*30)
    metrics('AD_Diag',GBoost,x_val,ad_cls,ad_pred)
```

```
CN_Diag Accuracy: 84.67%
CN_Diag Precision: 84.20%
CN_Diag Recall: 72.36%
CN_Diag AUC: 26.11%
******************************
MCI_Diag Accuracy: 77.52%
MCI_Diag Precision: 74.89%
MCI_Diag Recall: 72.66%
MCI_Diag AUC: 79.98%
******************************
AD_Diag Accuracy: 77.66%
AD_Diag Precision: 44.21%
AD_Diag Recall: 59.12%
AD_Diag AUC: 38.55%
```

Figure 16: Gradient Boosting Classifier

Overall validation accuracy of 73% is obtained with shown parameters

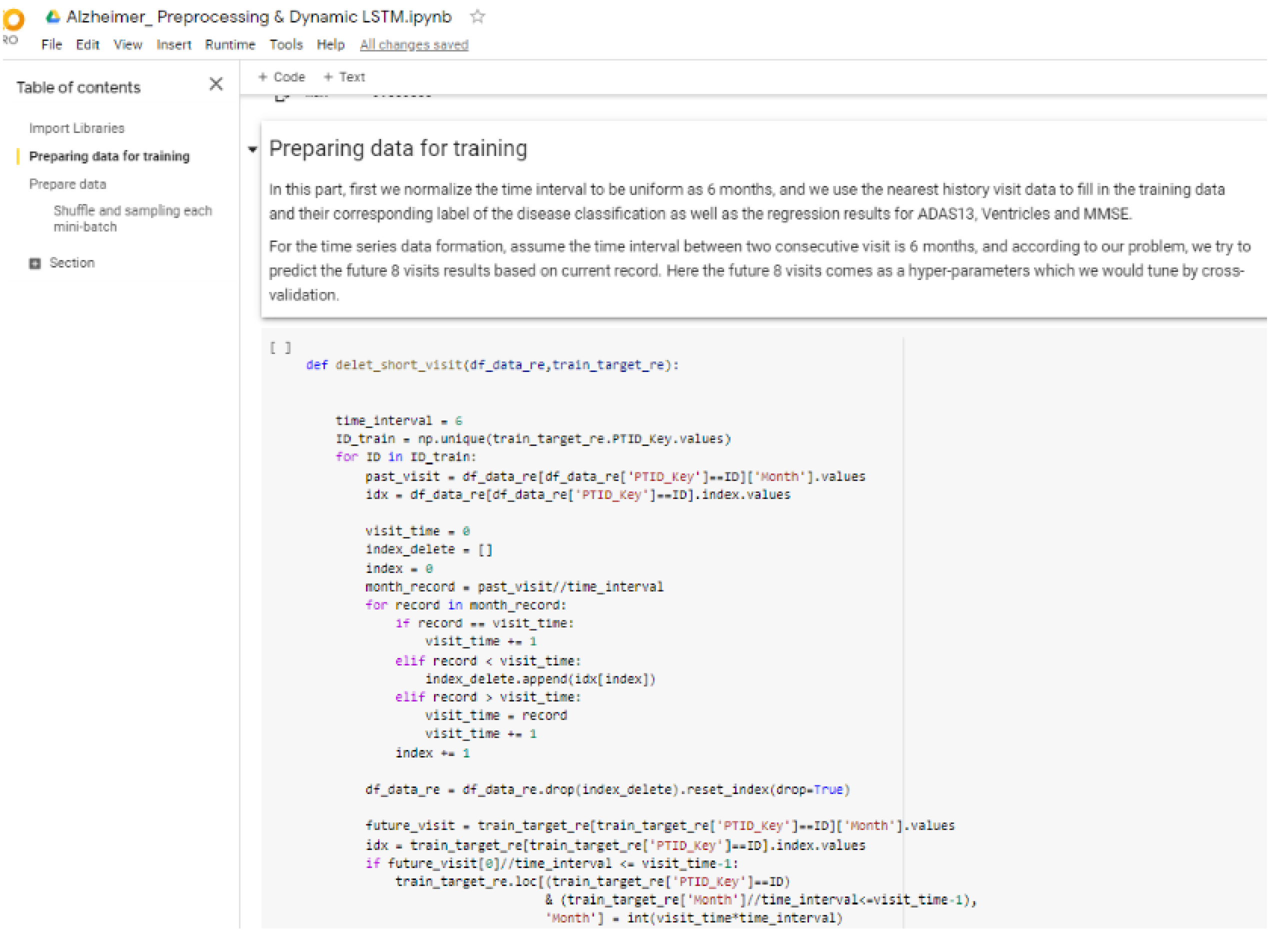


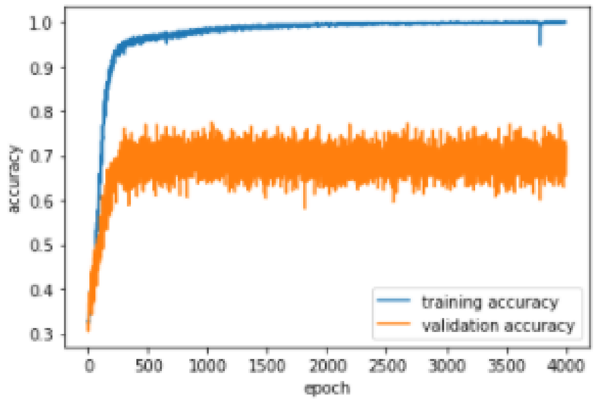
Figure 17: Training data prep for LSTM

+ Code + Text

```python
def reset_graph():
    if 'sess' in globals() and sess:
        sess.close()
    tf.reset_default_graph()

def build_graph(
    num_layers=3,
    feature_size = records.shape[1],
    state_size = 64,
    batch_size = 64,
    pred_times = 8,
    num_classes = 3):

    reset_graph()

    # Placeholders
    x = tf.placeholder('float', [batch_size, None, feature_size]) # input: shape=(batch_size,
    seqlen = tf.placeholder(tf.int32, [batch_size])
    y = tf.placeholder(tf.int32, [batch_size, pred_times])
    keep_prob = tf.placeholder(tf.float32,[])

    # RNN single cell
    cell =  (state_size)

    # Run dynamic_rnn to get all the output sequences
    # init_state: shape= (batch_size,cell.state_size) with all 0
    init_state = tf.get_variable('init_state', [1, state_size],
                                 initializer=tf.constant_initializer(0.0))
    init_state = tf.tile(init_state, [batch_size, 1])
    rnn_outputs, final_state = tf.nn.dynamic_rnn(cell, inputs=x, sequence_length=seqlen,
                                                 initial_state=init_state)
    # rnn_output: shape=((batch_size, time_steps, cell.output_size))

    # Add dropout, as the model otherwise quickly overfits
    rnn_outputs = tf.nn.dropout(rnn_outputs, keep_prob)

    # Get the output
    # Reshape rnn_outputs to a 2d tensor for softmax processing
    idx = tf.range(batch_size)*tf.shape(rnn_outputs)[1] + (seqlen - 1)
    last_rnn_output = tf.gather(tf.reshape(rnn_outputs, [-1, state_size]), idx)

    W_nn = tf.get_variable('W_nn', [state_size, state_size])
    b_nn = tf.get_variable('b_nn', [state_size], initializer=tf.constant_initializer(0.0))
    nn_output = tf.tanh(tf.matmul(last_rnn_output, W_nn) + b_nn)
```

Figure 18: Dynamic LSTM model building step

```
[ ] import matplotlib.pyplot as plt
    epochs=list(range(1,num_epochs+1))
    plt.figure()
    plt.plot(epochs,tr_losses,label='training accuracy')
    plt.plot(epochs,te_losses,label='validation accuracy')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.legend()
    plt.show()
```




```
[ ] print("The highest validation accuracy is: ", max(te_losses))


    print("At the same epoch, the training accuracy is: ", tr_losses[te_losses.index(max(te_losses))])
```

```
The highest validation accuracy is:  0.775390625
At the same epoch, the training accuracy is:  0.9821428571428571
```

Figure 19: LSTM Accuracy vs Epoch

Dynamic LSTM which is a type of Recuurent Neural network outperforms all other classifiers with a validation accuracy of 78%

References