MSc Data Science

Modelling, Regression and Machine-Learning

CST 4050

Individual Report

APPLYING MACHINE LEARNING TECHNIQUES TO IDENTIFY THE RISK OF ALZHEIMER'S DISEASE

Submitted By:

Thanima Firoz

M00849665

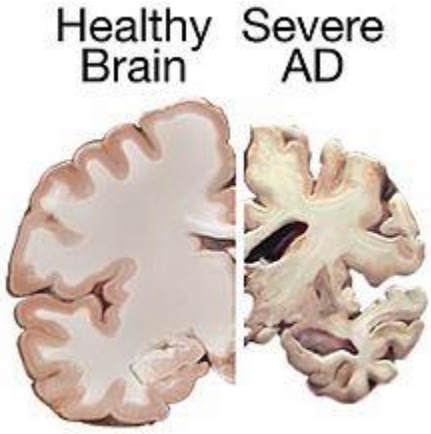
Abstract

In recent years, there has been a significant uptick in interest in Alzheimer's disease(AD), a form of dementia characterized by a gradual and deadly accumulation of proteins in the brain, killing brain cells and leading to memory loss, and behavior changes, and eventually death. While there is currently no cure for Alzheimer's disease, research has suggested that there is a correlation between the risk of contracting Alzheimer's and various aspects of the brain, including grey matter distribution, the appearance of amyloid deposits etc.

Problem Statement

Thanks to the improvement in medical imaging technology, namely magnetic resonance imaging(MRI) and positron emission tomography(PET) scans, the doctors and researchers are now able to diagnose Alzheimer's disease with greater accuracy and timeliness, however, such scans often only yield telltale information after symptoms are reported, at which point measures to slow or prevent its development are usually of limited effectiveness. While there are of course many factors that can lead to Alzheimer's, and not all can be determined from brain scans alone, I have considered the socio-economic factors, gender as well as education as independent predictors.

MRI findings include both, local and generalized shrinkage of brain tissue. Below is a pictorial representation of tissue shrinkage:



Some studies have suggested that MRI features may predict rate of decline of AD and may guide therapy in the future. However, in order to reach that stage clinicians and researchers will have to make use of machine learning techniques that can accurately predict the progress of a patient from mild cognitive impairment to dementia.

In this project,I am trying to develop a sound machine learning model that can help doctors to predict early Alzheimer's using MRI data.

Here I tend to do a comparative study of different commonly used machine learning classification algorithms like logical regression, Decision Tree, Random Forest, and Support Vector Machine. Choosing the best machine learning algorithm in order to solve the problems of classification and prediction of data is the most important part of machine learning which depends on the dataset as well.

Related Works

A variety of other projects regarding the classification of Alzheimer's disease using brain scan imagery have been conducted in recent years. The original publication has only done some preliminary exploration of the MRI data as the majority of their work was focused on data gathering. However, in the recent past, there have been multiple efforts that have been made to detect early Alzheimer's using MRI data. Some of the work that was found in the literature was as follows:

1) Machine learning framework for early MRI-based Alzheimer's conversion prediction in MCI subjects(Moradi *et al.*, 2015)-In this paper the authors were interested in identifying mild cognitive impairment(MCI) as a transitional stage between age-related cognitive decline and Alzheimer's. The group proposes a novel MRI-based biomarker that they developed using machine learning techniques. They used data available from the Alzheimer's Disease Neuroimaging Initiative ADNI Database. The paper claims that their aggregate biomarker achieved a 10-fold cross-validation area under the curve (AUC) score of 0.9020 in discriminating between progressive MCI (pMCI) and stable MCI (sMCI).

   Noteworthy Techniques:
   - Semi-supervised learning on data available from AD patients and normal controls, without using MCI patients, to help with the sMCI/pMCI classification. Performed feature selection using regularized logistic regression.
   - They removed aging effects from MRI data before classifier training to prevent possible confounding between changes due to AD and those due to normal aging.
   - Finally constructed an aggregate biomarker by first learning a separate MRI biomarker and then combining age and cognitive measures about MCI subjects by applying a random foresst classifier.

2) Detection of subjects and brain regions related to Alzheimer's disease using 3D MRI scans based on eigenbrain and machine learning(Zhang *et al.*, 2015)- The authors of this paper have proposed a novel computer-aided diagnosis (CAD) system for MRI images of brains based on eigenbrains

[(eg.)](https://www.frontiersin.org/files/Articles/138015/fncom-09-00066-HTML/image_m/fncom-09-00066-t010.jpg) and machine learning. In their approach they use key slices from the 3D volumetric data generated from the MRI and then generate eigenbrain images based on [EEG](https://en.wikipedia.org/wiki/Electroencephalography) data. They then used kernel support-vector-machines with different kernels that were trained by particle swarm optimization. The accuracy of their polynomial kernel (92.36 $\pm$ 0.94) was better than their linear (91.47 $\pm$ 1.02) anf radial basis function (86.71 $\pm$ 1.93) kernels.

3) Support vector machine-based classification of Alzheimer's disease from whole-brain anatomical MRI(Magnin *et al.*, 2009)-In this paper the authors propose a new method to discriminate patients with AD from elderly controls based on support vector machine (SVM) classification of whole-brain anatomical MRI. The authors used three-dimensional T1-weighted MRI images from 16 patients with AD and 22 elderly controls and parcellated them into regions of interest (ROIs). They then used an SVM algorithm to classify subjects based on the gray matter characteristics of these ROIs. Based on their results the classifier obtained 94.5% mean correctness. The possible downfalls of their technique might be the fact that they haven't taken age related changes in the gray matter into account and they were working with a small data set.

The above-mentioned 3 papers over here have explored the same question. Regardless, it is worthwhile to mention that the above papers were exploring raw MRI data and on the other hand, in this project, I am dealing with 3 to 4 biomarkers that are generated from MRI images.

Dataset Description

The Oasis-3 dataset includes a plethora of metadata in addition to patient scans, going as specific as family history and number of psychiatric appointments.

This project represents the datasets for training and predictions, both featuring MRI scan data from normal patients to patients having Alzheimer's. The dataset I am using is Oasis-3. This dataset is a longitudinal study consisting of 1,099 images taken from roughly 150 subject, with each subject undergoing a minimum of three scan sessions , each last year apart . Of the 150 subjects,72 showed no symptoms of Alzheimer's disease over the course of the study.64 showed symptoms of Alzheimer's disease prior to any scans and remains so throughout the course of all scans and 14 did not initially display any symptoms of Alzheimer's disease but did after subsequent scans. The Oasis-3 dataset relies upon PET scans which reveal brain functions.The subject's aged from 60-96and everyone is right-handed.

Column Description

Column names	Full-form
EDUC	Years of Education
SES	Socio economic status
MMSE	Mini Mental State Examination
CDR	Clinical Dementia Rating
E TIV	Estimated Total Intracranial Volume
nWBV	Normalize whole brain volume
ASF	Atlas Scaling Factor

Data Preprocessing

The below list of data preprocessing has been done on the dataset.

i. Considered first visit data only as I am trying to analyze the risk of occurrence of Alzheimer's disease.
ii. Performed one hot coding in the male/female column.
iii. Changed the target variable column name to 'Group' and performed one hot coding after that.
iv. Dropped the unwanted columns like 'MRI ID',' visit', and 'hand'(since all are right-handers).
v. Handling Missing Values- There are 8 rows with missing values in the 'SES' column.

These missing values can be handled in two ways-

- Dropping missing rows
- Imputation-replacing the missing values with the corresponding values. Since the data size is small, I assume imputation would help to improve the performance of the model.

Exploratory Data Analysis

In this section, I have focused on exploring the relationship between each feature of MRI tests and the dementia of the patient. The reason for conducting this EDA is to state the relationship of data explicitly through a graph so that the correlations can be assumed before data extraction or data analysis. It might help to understand the nature of the data and to select the appropriate analysis method for the model later.

The minimum, maximum, and average values of each feature for graph implementation are as follows.

Col name	Min	Max	Mean
Educ	6	23	14.6
SES	1	5	2.34
MMSE	17	30	27.2
CDR	0	1	0.29
eTIV	1123	1989	1490
nWBV	0.66	0.837	0.73
ASF	0.883	1.563	1.2

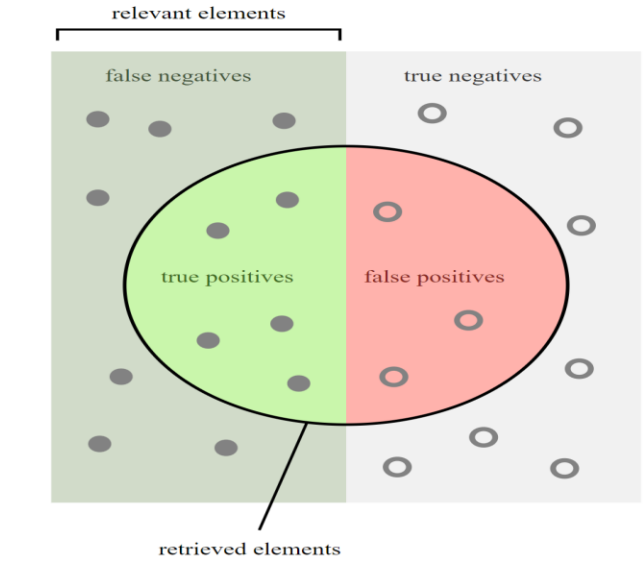
Below are conclusions obtained after EDA.

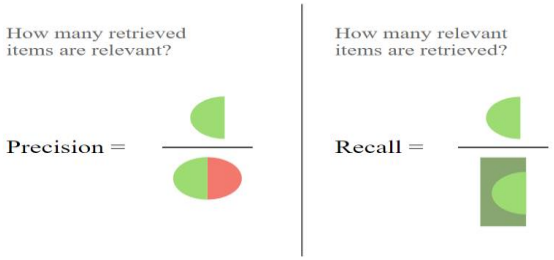
- Men are more likely with demented, an Alzheimer's Disease, than Women.
- Demented patients were less educated in terms of years of education.
- Nondemented group has higher brain volume than the Demented group.
- Higher concentration of 70-80 years old in Demented group than those in the non-demented patients.

Performance Measures

I have considered the area under the receiver operating characteristic curve (AUC) as the main performance measure. In the case of medical diagnostics for non-life threatening terminal diseases like most neurodegenerative diseases it is important to have a high true positive rate so that all patients with Alzheimer’s are identified as early as possible. But sametime we should make sure that the false positive rate is as low as possible since we do not want to misdiagnose a healthy adult as demented and begin medical therapy. Hence AUC seemed like an ideal choice for a performance measure. Accuracy and recall for each model have been considered.

Below figure shows the relevant elements that are actually demented subjects-Precision and Recall.





Learning Methods

As mentioned previously, the algorithms chosen to implement the Alzheimer's prediction model are Logistic Regression, Support Vector Machine, Random Forest, and Decision Tree.

1) Logistic Regression Model

I choose logistic regression as my base model because it is one of the simplest models for a classification task. The Logistic Regression Model assumes the presence of Alzheimer's given feature model as a Bernoulli RV:

$y|x; \theta \sim \text{Bernoulli}(\eta)$

As part of the exponential family, under the assumption that the natural parameter η is linearly related to the input $\eta = \theta^T x$, the probability equation is as below.

$P(y=1)|x: \theta = h_\theta(x) = g(\theta^T x) = 1/(1+e^{-\theta x})$.

Logistic Regression is a binary classifier, meaning Alzheimer's is said to be present($y=1$)when $h_\theta(x) >= 0.5$, else $y=0$.

For the logistic regression model, I have considered the dataset with dropping missing values as a baseline model and obtained the below results.

```
Best accuracy on validation set is: 0.725974025974026
Best parameter for regularization (C) is:  10
Test accuracy with best C parameter is 0.8055555555555556
Test recall with the best C parameter is 0.75
Test AUC with the best C parameter is 0.8194444444444443
```

Similarly, Logistic regression has been done with the model with imputation and obtained the following results.

```
Best accuracy on validation set is: 0.724901185770751
Best parameter for regularization (C) is:  100
Test accuracy with best C parameter is 0.7894736842105263
Test recall with the best C parameter is 0.7
Test AUC with the best C parameter is 0.7944444444444444
```

Overall, the dataset with imputation outperforms the one without imputation. Hence, for the later models, a dataset with imputation is considered.

2) **<u>Support Vector Machine</u>**

Support vector machine (SVM) is considered one of the best algorithms for supervised learning. The main idea of this algorithm is to map the data from a relatively low dimensional space to a relatively high dimensional space so that the higher dimensional data can be separated into two classes by a hyperplane. The hyperplane that separates the data with maximum margin is called the support vector classifier, which be determined using Kernel Functions in order to avoid expensive computation to transform the data explicitly.

The setting of SVM in this report:

- C: Penalty parameter C of the error term. [0.001, 0.01, 0.1, 1, 10, 100, 1000]
- gamma: kernel coefficient. [0.001, 0.01, 0.1, 1, 10, 100, 1000]
- kernel: kernel type. ['rbf', 'linear', 'poly', 'sigmoid']

Results obtained are as follows.

```
Best accuracy on cross validation set is: 0.7687747035573123
Best parameter for c is:  100
Best parameter for gamma is:  0.1
Best parameter for kernel is:  rbf
Test accuracy with the best parameters is 0.8157894736842105
Test recall with the best parameters is 0.7
Test recall with the best parameter is 0.8222222222222222
```

3) Decision Tree Methods

- **Random Forest Classifier**

  Random forest ensembles multiple decision trees trained in parallel with bagging. Bagging allows individual trees to be trained on subsets of training data that are randomly sampled with replacement. For a tree, when deciding which feature to split on ,I picked the one that decreases Gini impurity the most. Gini impurity is given by

$$I_G(p) = \sum_{i=1}^{K} p_i(1 - p_i)$$

The setting of random forest in this report is as follows.

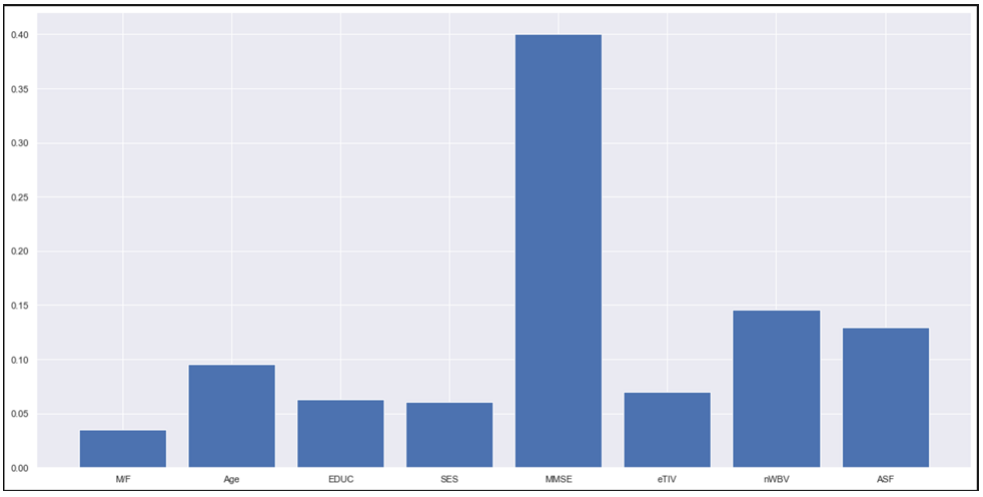
n_estimators(M): the number of trees in the forest-2
max_features(d): the number of features to consider when looking for the best split-5
max_depth(m): the maximum depth of the tree-7
K is the class number. M is the sample size.
The value will take on a small value if the
node is pure.

$$G = \sum_{K=1}^{K} p_{mk} \, log \, p_{mk}$$

Results obtained are as follows.

```
Best accuracy on validation set is: 0.8035573122529645
Best parameters of M, d, m are:  2 5 7
Test accuracy with the best parameters is 0.868421052631579
Test recall with the best parameters is: 0.8
Test AUC with the best parameters is: 0.8722222222222222
```

Feature Importance



- **<u>Adaboost</u>**

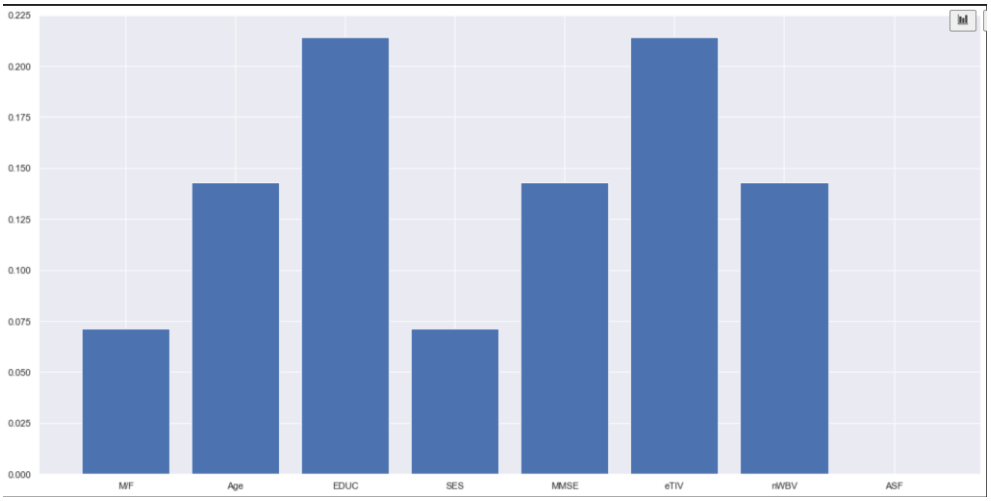
  Adaboost is another approach to tree classification. Boosting also becomes a method to improve the predictions over bagged trees. Boosting trees are grown sequentially. Each tree is grown based on the information from previously grown trees, thus robust to overfitting. Notably, boosting does not involve bootstrap sampling; instead, each tree collectively fits on the original tree. The setting of boosting in this report:

  • The number of boosting trees: 2
  • Test criterion: MSE.
  • Learning rate: 0.0001

  Results obtained are as follows.

```
Best accuracy on validation set is: 0.7770750988142293
Best parameter of M is:  2
best parameter of LR is:  0.0001
Test accuracy with the best parameter is 0.868421052631579
Test recall with the best parameters is: 0.65
Test AUC with the best parameters is: 0.825
```

Feature Importance



Results and Discussion

The performance matrix of each model is as follows.

Model	Accuracy	Recall	AUC
Logistic Regression (w/ imputation)	0.789474	0.70	0.794444
Logistic Regression (w/ dropna)	0.805556	0.75	0.750000
SVM	0.815789	0.70	0.822222
Decision Tree	0.815789	0.65	0.825000
Random Forest	0.868421	0.80	0.872222
AdaBoost	0.868421	0.65	0.825000

Logistic Regression-Logistic Regression performs average on the original dataset, which is expected as it is a relatively simple model. Logistic regression using dataset with imputation outperformed well than logistic regression using a dataset with the dropped null value.

SVM Model-SVM model performed well on the dataset with 81.5% accuracy an AUC of 82% and a Recall of 70%.

Decision Tree-Decision Tree performed well on the dataset. Based on the above results obtained, Random Forest has the best results compared to the base model of logistic regression.

Below is a comparison of the results with those from the papers that were listed previously.

SL No	Paper	Data	Model	Results	
1	E. Moradi et al	Ye et al(Ye, Pohl and Davatzikos, 2011)	Random Forest Classifier	AUC=71.0%	ACC=55.3%
		Filipovych, et al(Tan *et al.*, 2018).	Random Forest Classifier	AUC=61.0%	ACC=N/A
		Zhang et al(*Predicting Future Clinical Changes of MCI Patients Using Longitudinal and Multimodal Biomarkers*, no date).	Random Forest Classifier	AUC=94.6%	ACC=N/A
		Batmanghelich et a(Batmanghelich *et al.*, 2011)l.	Random Forest Classifier	AUC=61.5%	ACC=N/A
2	Zhang et a	Ardekani et al.	Support Vector Machine		
			Polynomial Kernal	AUC=N/A	ACC=92.4%
			Linear Kernal	AUC=N/A	ACC=91.5%
			Radial Basis Function	AUC=N/A	ACC=86.7%
3	Hyun, Kyuri, Saurin	Marcus et al(Marcus *et al.*, 2010).	Logistic Regression with imputation	AUC=79.2%	ACC=78.9%
			Logistic Regression with dropna	AUC=70.0%	ACC=78.9%
			Support Vector Machine	AUC=82.2%	ACC=75.0%
			Decision Tree Classifier	AUC=82.5%	ACC=81.6%
			Random Forest Classifier	AUC=84.4%	ACC=84.2%
			Adaboost	AUC=82.5%	ACC=84.2%

It can be noticed that my results are comparable and in certain cases better than those from the previous work. Our Random Forest Classifier was one of the best performing model.

Unique Approach

The uniqueness of this model is the fact that I have included metrics like MMSE and Education also to train inorder to differentiate between normal healthy adults and those with Alzheimer's. MMSE is one of the gold standards for determining dementia and hence therefore it is an important feature to include.

The same fact also make this approach flexible enough to be applied to other neurodegenerative diseases which are diagnosed using a combination of MRI features and cognitive tests.

Limitations

There are limitations in implementing a complex model because of the quantity of the dataset. Even though the nature of each feature is evident, the ranges of each group's test value are not classified well. In other words, I could identify more clearly the differences in the variables which might have played a role in the result. The predicted value using the random forest model is higher than the other models. It implies there is a potential for a higher prediction rate if we pay more attention to developing the data cleaning and analysis process. Moreover, the perfect recall score of 1.0 of SVM 1.0 indicates that the quality and accuracy of the classification might decrease dramatically when we use different datasets.

References

Batmanghelich, K.N. *et al.* (2011) 'DISEASE CLASSIFICATION AND PREDICTION VIA SEMI-SUPERVISED DIMENSIONALITY REDUCTION', *Proceedings. IEEE International Symposium on Biomedical Imaging*, 2011, pp. 1086–1090. doi:10.1109/ISBI.2011.5872590.

Magnin, B. *et al.* (2009) 'Support vector machine-based classification of Alzheimer's disease from whole-brain anatomical MRI', *Neuroradiology*, 51(2), pp. 73–83. doi:10.1007/s00234-008-0463-x.

Marcus, D.S. *et al.* (2010) 'Open access series of imaging studies: longitudinal MRI data in nondemented and demented older adults', *Journal of Cognitive Neuroscience*, 22(12), pp. 2677–2684. doi:10.1162/jocn.2009.21407.

Moradi, E. *et al.* (2015) 'Machine learning framework for early MRI-based Alzheimer's conversion prediction in MCI subjects', *NeuroImage*, 104, pp. 398–412. doi:10.1016/j.neuroimage.2014.10.002.

Predicting Future Clinical Changes of MCI Patients Using Longitudinal and Multimodal Biomarkers (no date). Available at: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0033182 (Accessed: 15 April 2022).

Tan, X. *et al.* (2018) 'Localized instance fusion of MRI data of Alzheimer's disease for classification based on instance transfer ensemble learning', *BioMedical Engineering OnLine*, 17, p. 49. doi:10.1186/s12938-018-0489-1.

Ye, D.H., Pohl, K.M. and Davatzikos, C. (2011) 'Semi-Supervised Pattern Classification: Application to Structural MRI of Alzheimer's Disease', ... *International Workshop on Pattern*

Recognition in NeuroImaging. International Workshop on Pattern Recognition in NeuroImaging, 2011, pp. 1–4. doi:10.1109/PRNI.2011.12.

Zhang, Y. *et al.* (2015) 'Detection of subjects and brain regions related to Alzheimer's disease using 3D MRI scans based on eigenbrain and machine learning', *Frontiers in Computational Neuroscience*, 9, p. 66. doi:10.3389/fncom.2015.00066.

Appendix

APPLYING MACHINE LEARNING TECHNIQUES TO IDENTIFY THE RISK OF ALZEIMER'S DISEASE

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')

# importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

sns.set()
```

Loading Data

In [ ]:

```
df = pd.read_csv(r'C:\Users\matebook x\Desktop/MRI/oasis_longitudinal.csv')
df.head()
```

Out[ ]:

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	2.0	27.0	0.0	1987	0.696	0.883
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	2.0	30.0	0.0	2004	0.681	0.876
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	NaN	23.0	0.5	1678	0.736	1.046

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	NaN	28.0	0.5	1738	0.713	1.010
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	NaN	22.0	0.5	1698	0.701	1.034

Column Description

COL FULL-FORMS

EDUC - Years of education

SES - Socioeconomic Status

MMSE - Mini Mental State Examination

CDR - Clinical Dementia Rating

eTIV - Estimated Total Intracranial Volume

nWBV - Normalize Whole Brain Volume

ASF - Atlas Scaling Factor

In [ ]:

```
df.Hand.unique()
```

Out[ ]:

```
array(['R'], dtype=object)
```

In [ ]:

```
df = df.loc[df['Visit']==1] # use first visit data only because of the
analysis we're doing
df = df.reset_index(drop=True) # reset index after filtering first visit data
df['M/F'] = df['M/F'].replace(['F','M'], [0,1]) # M/F column
df['Group'] = df['Group'].replace(['Converted'], ['Demented']) # Target
variable
df['Group'] = df['Group'].replace(['Demented', 'Nondemented'], [1,0]) #
Target variable
df = df.drop(['MRI ID', 'Visit', 'Hand'], axis=1) # Drop unnecessary columns
```

In [ ]:

```
# bar drawing function
def bar_chart(feature):
    Demented = df[df['Group']==1][feature].value_counts()
    Nondemented = df[df['Group']==0][feature].value_counts()
```

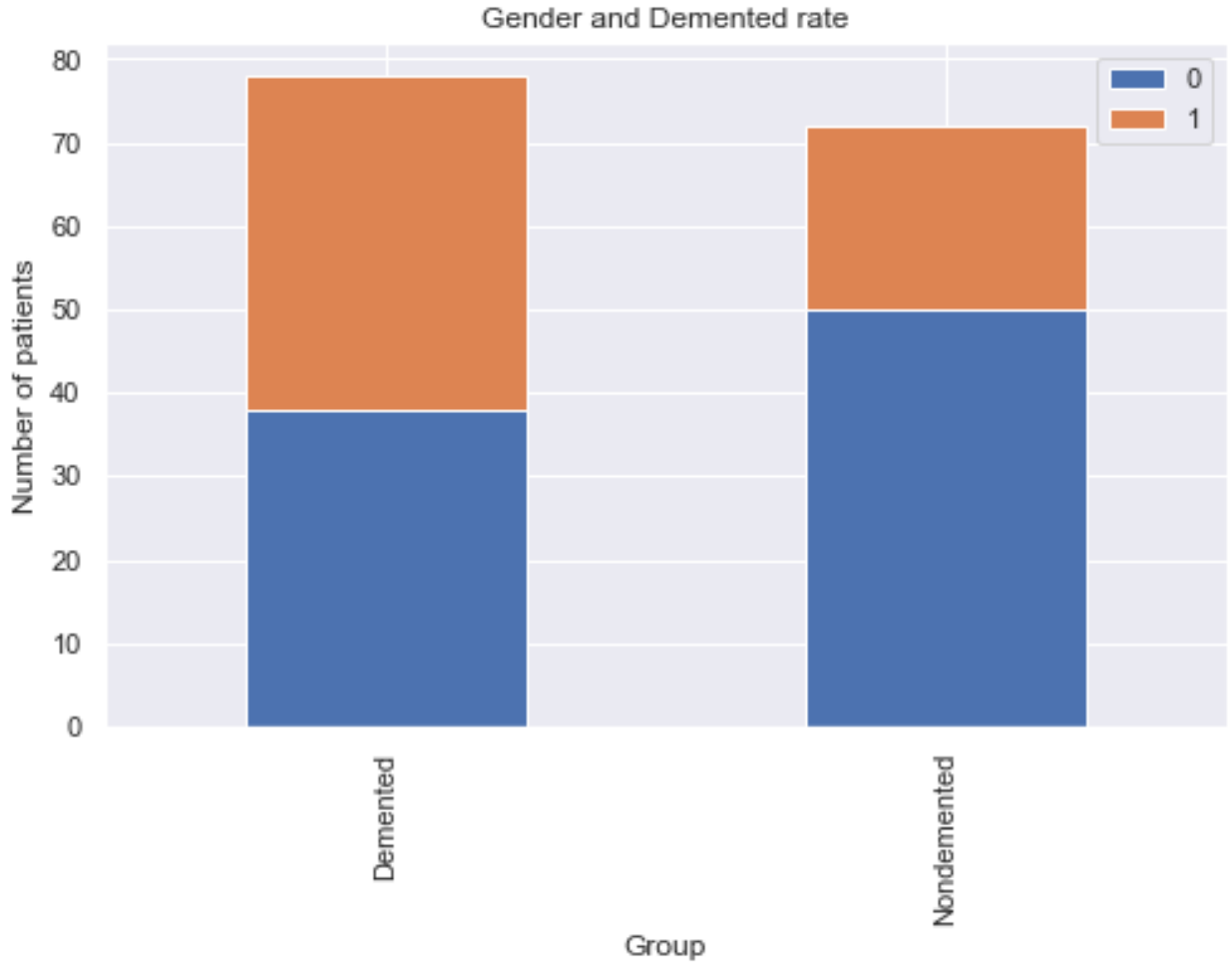
```
    df_bar = pd.DataFrame([Demented,Nondemented])
    df_bar.index = ['Demented','Nondemented']
    df_bar.plot(kind='bar',stacked=True, figsize=(8,5))
```

In [ ]:

```
# Gender  and  Group ( Femal=0, Male=1)
bar_chart('M/F')
plt.xlabel('Group')
plt.ylabel('Number of patients')
plt.legend()
plt.title('Gender and Demented rate')
```

Out[ ]:

```
Text(0.5, 1.0, 'Gender and Demented rate')
```



The above graph indicates that men are more likely with dementia than women.

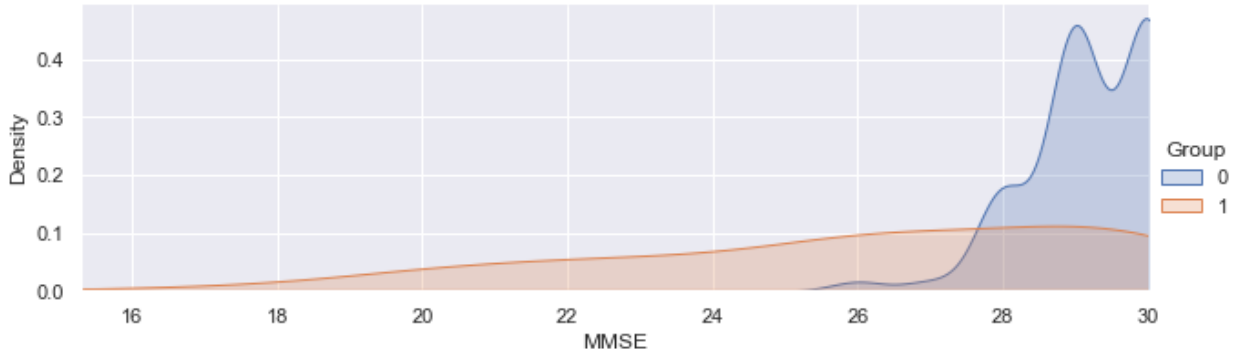
In [ ]:

```
#MMSE : Mini Mental State Examination
# Nondemented = 0, Demented =1
# Nondemented has higher test result ranging from 25 to 30.
#Min 17 ,MAX 30
```

```
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'MMSE',shade= True)
facet.set(xlim=(0, df['MMSE'].max()))
facet.add_legend()
plt.xlim(15.30)
```

Out[ ]:

```
(15.3, 30.0)
```



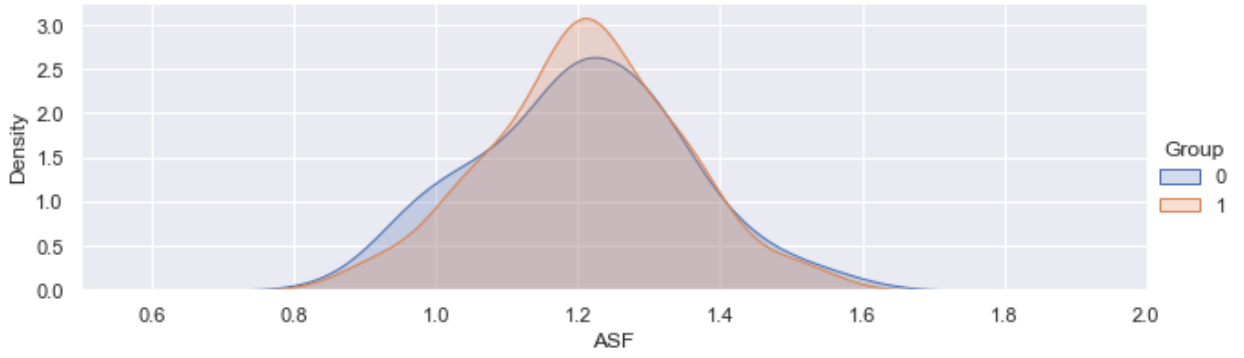
The chart shows Nondemented group got much more higher MMSE scores than Demented group

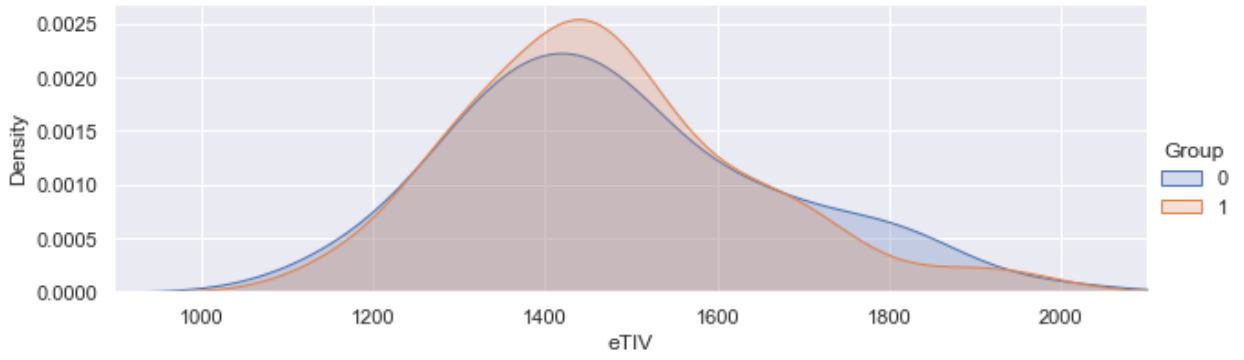
In [ ]:

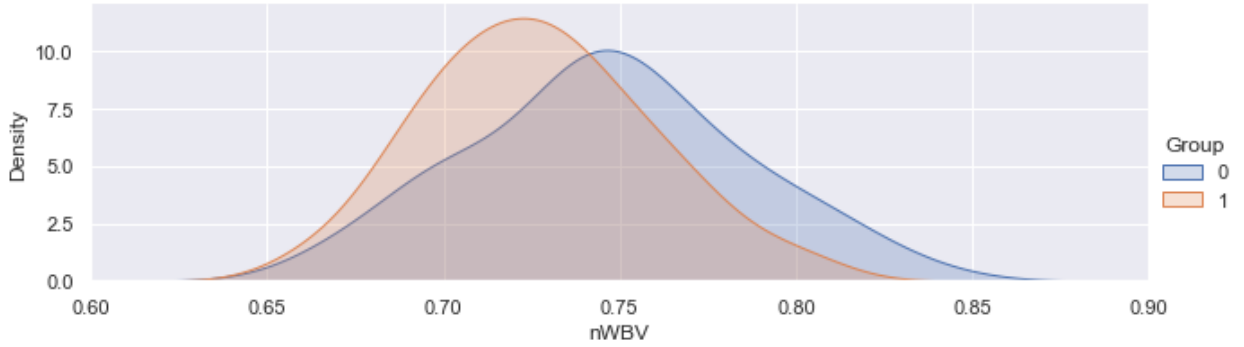
```
#bar_chart('ASF') = Atlas Scaling Factor
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'ASF',shade= True)
facet.set(xlim=(0, df['ASF'].max()))
facet.add_legend()
plt.xlim(0.5, 2)

#eTIV = Estimated Total Intracranial Volume
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'eTIV',shade= True)
facet.set(xlim=(0, df['eTIV'].max()))
facet.add_legend()
plt.xlim(900, 2100)

#'nWBV' = Normalized Whole Brain Volume
# Nondemented = 0, Demented =1
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'nWBV',shade= True)
facet.set(xlim=(0, df['nWBV'].max()))
facet.add_legend()
plt.xlim(0.6,0.9)
```

Out[ ]:

```
(0.6, 0.9)
```







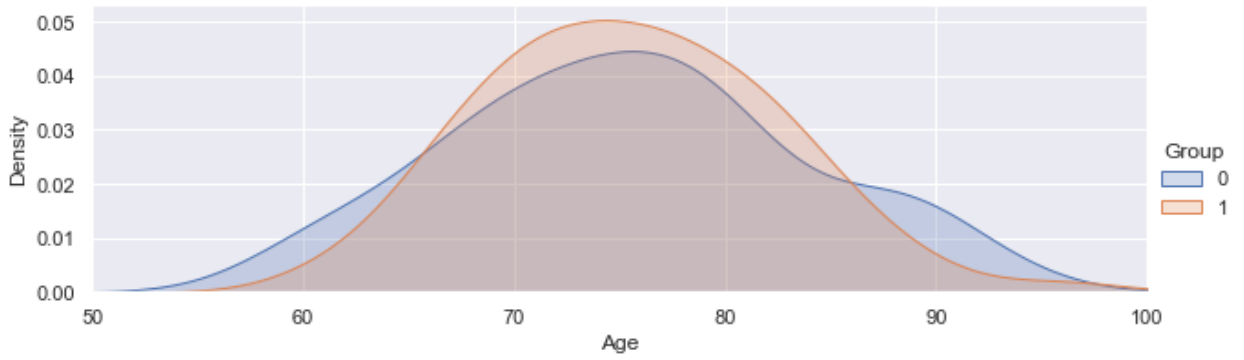
The chart indicates that Nondemented group has higher brain volume ratio than Demented group. This is assumed to be because the diseases affect the brain to be shrinking its tissue.

In [ ]:

```
#AGE. Nondemented =0, Demented =0
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, df['Age'].max()))
facet.add_legend()
plt.xlim(50,100)
```

Out[ ]:

```
(50.0, 100.0)
```



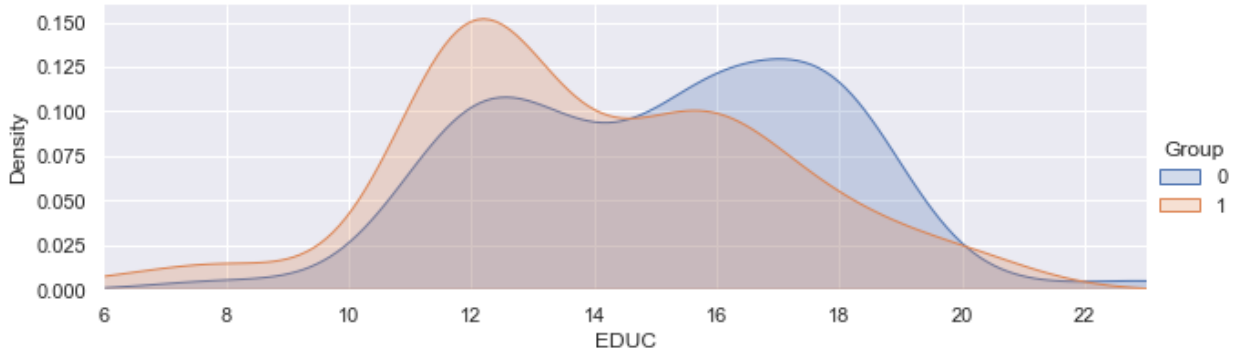
There is a higher concentration of 70-80 years old in the Demented patient group than those in the nondemented patients. We guess patients who suffered from that kind of disease has lower survival rate so that there are a few of 90 years old.

In [ ]:

```
#'EDUC' = Years of Education
# Nondemented = 0, Demented =1
facet= sns.FacetGrid(df,hue="Group", aspect=3)
facet.map(sns.kdeplot,'EDUC',shade= True)
facet.set(xlim=(df['EDUC'].min(), df['EDUC'].max()))
facet.add_legend()
plt.ylim(0, 0.16)
```

Out[ ]:

```
(0.0, 0.16)
```



Intermediate Result Summary

Men are more likely with demented, an Alzheimer's Disease, than Women.

Demented patients were less educated in terms of years of education.

Nondemented group has higher brain volume than Demented group.

Higher concentration of 70-80 years old in Demented group than those in the nondemented patients.

Data Preprocessing

We identified 8 rows with missing values in SES column. We deal with this issue with 2 approaches. One is just to drop the rows with missing values. The other is to replace the missing values with the

corresponing values, also known as 'Imputation'. Since we have only 150 data, I assume imputation would help the performance of our model.

In [ ]:

```
# Check missing values by each column
pd.isnull(df).sum()
# The column, SES has 8 missing values
```

Out[ ]:

```
Subject ID    0
Group         0
MR Delay      0
M/F           0
Age           0
EDUC          0
SES           8
MMSE          0
CDR           0
eTIV          0
nWBV          0
ASF           0
dtype: int64
```

Handling Missing values:option 1: Removing rows with missing values

In [ ]:

```
# Dropped the 8 rows with missing values in the column, SES
df_dropna = df.dropna(axis=0, how='any')
df_dropna.isnull().sum()
```

Out[ ]:

```
Subject ID    0
Group         0
MR Delay      0
M/F           0
Age           0
EDUC          0
SES           0
MMSE          0
CDR           0
eTIV          0
nWBV          0
ASF           0
dtype: int64
```

In [ ]:

```
df_dropna['Group'].value_counts()
```

Out[ ]:

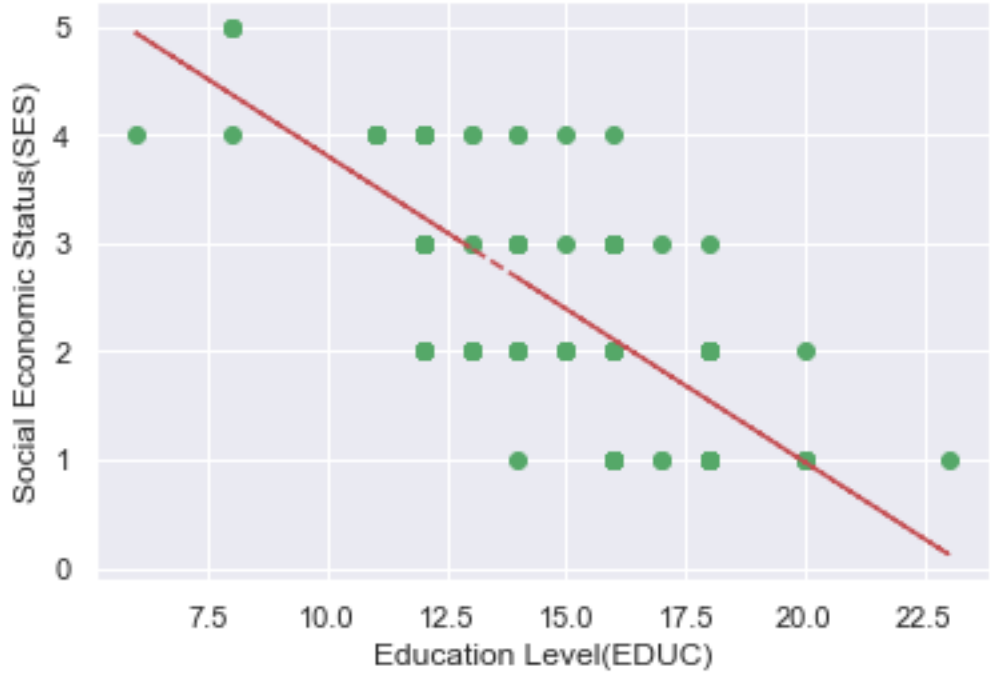
```
0    72
1    70
Name: Group, dtype: int64
```

Handling Missing Values:option 2: Imputation

Scikit-learn provides package for imputation [6], but we do it manually. Since the SES is a discrete variable, we use median for the imputation.

In [ ]:

```
# Draw scatter plot between EDUC and SES
x = df['EDUC']
y = df['SES']

ses_not_null_index = y[~y.isnull()].index
x = x[ses_not_null_index]
y = y[ses_not_null_index]

# Draw trend line in red
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x, y, 'go', x, p(x), "r--")
plt.xlabel('Education Level(EDUC)')
plt.ylabel('Social Economic Status(SES)')

plt.show()
```



In [ ]:

```
df.groupby(['EDUC'])['SES'].median()
```

Out[ ]:

```
EDUC
6     4.0
8     5.0
```

```
11    4.0
12    3.0
13    2.0
14    3.0
15    2.0
16    2.0
17    1.0
18    2.0
20    1.0
23    1.0
Name: SES, dtype: float64
```

In [ ]:

```
df["SES"].fillna(df.groupby("EDUC")["SES"].transform("median"), inplace=True)
```

In [ ]:

```
# There're no more missing values and all the 150 data were used.

pd.isnull(df['SES']).value_counts()
```

Out[ ]:

```
False    150
Name: SES, dtype: int64
```

Splitting Train/Validation/Test Sets

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
```

In [ ]:

```
# Dataset with imputation

Y = df['Group'].values # Target for the model
X = df[['M/F', 'Age', 'EDUC', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF']] #
Features we use

# splitting into three sets
X_trainval, X_test, Y_trainval, Y_test = train_test_split(
    X, Y, random_state=0)

# Feature scaling
scaler = MinMaxScaler().fit(X_trainval)
X_trainval_scaled = scaler.transform(X_trainval)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:

```
# Dataset after dropping missing value rows

Y = df_dropna['Group'].values # Target for the model
X = df_dropna[['M/F', 'Age', 'EDUC', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF']] #
Features we use
```

```
# splitting into three sets
X_trainval_dna, X_test_dna, Y_trainval_dna, Y_test_dna = train_test_split(
    X, Y, random_state=0)

# Feature scaling
scaler = MinMaxScaler().fit(X_trainval_dna)
X_trainval_scaled_dna = scaler.transform(X_trainval_dna)
X_test_scaled_dna = scaler.transform(X_test_dna)
```

Cross-validation

We conduct 5-fold cross-validation to figure out the best parameters for each model, Logistic Regression, SVM, Decision Tree, Random Forests, and AdaBoost. Since our performance metric is accuracy, we find the best tuning parameters by accuracy. In the end, we compare the accuracy, recall and AUC for each model.

Applying Machine Learning Models

Performance Measures

We use area under the receiver operating characteristic curve (AUC) as our main performance measure. We believe that in case of medical diagnostics for non-life threatening terminal diseases like most neurodegenerative diseases it is important to have a high true positive rate so that all patients with alzheimer's are identified as early as possible. But we also want to make sure that the false positive rate is as low as possible since we do not want to misdiagnose a healthy adult as demented and begin medical therapy. Hence AUC seemed like a ideal choice for a performance measure.

We will also be looking at accuracy and recall for each model.

Logistic Regression-Baseline Model

The parameter C, inverse of regularization strength.

Tuning range: [0.001, 0.1, 1, 10, 100]

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score,
roc_curve, auc
```

In [ ]:

```
acc = [] # list to store all performance metric
```

In [ ]:

```
# Dataset with imputation

best_score=0
kfolds=5 # set the number of folds

for c in [0.001, 0.1, 1, 10, 100]:
    logRegModel = LogisticRegression(C=c)
```

```
    # perform cross-validation
    scores = cross_val_score(logRegModel, X_trainval, Y_trainval, cv=kfolds,
scoring='accuracy') # Get recall for each parameter setting

    # compute mean cross-validation accuracy
    score = np.mean(scores)

    # Find the best parameters and score
    if score > best_score:
        best_score = score
        best_parameters = c

# rebuild a model on the combined training and validation set

SelectedLogRegModel =
LogisticRegression(C=best_parameters).fit(X_trainval_scaled, Y_trainval)

test_score = SelectedLogRegModel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedLogRegModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for regularization (C) is: ", best_parameters)
print("Test accuracy with best C parameter is", test_score)
print("Test recall with the best C parameter is", test_recall)
print("Test AUC with the best C parameter is", test_auc)
m = 'Logistic Regression (w/ imputation)'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
Best accuracy on validation set is: 0.724901185770751
Best parameter for regularization (C) is:  100
Test accuracy with best C parameter is 0.7894736842105263
Test recall with the best C parameter is 0.7
Test AUC with the best C parameter is 0.7944444444444444
```

In [ ]:

```
# Dataset after dropping missing value rows
best_score=0
kfolds=5 # set the number of folds

for c in [0.001, 0.1, 1, 10, 100]:
    logRegModel = LogisticRegression(C=c)
    # perform cross-validation
    scores = cross_val_score(logRegModel, X_trainval_scaled_dna,
Y_trainval_dna, cv=kfolds, scoring='accuracy')

    # compute mean cross-validation accuracy
    score = np.mean(scores)

    # Find the best parameters and score
    if score > best_score:
        best_score = score
        best_parameters = c
```

```
# rebuild a model on the combined training and validation set
SelectedLogRegModel =
LogisticRegression(C=best_parameters).fit(X_trainval_scaled_dna,
Y_trainval_dna)
test_score = SelectedLogRegModel.score(X_test_scaled_dna, Y_test_dna)
PredictedOutput = SelectedLogRegModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for regularization (C) is: ", best_parameters)
print("Test accuracy with best C parameter is", test_score)
print("Test recall with the best C parameter is", test_recall)
print("Test AUC with the best C parameter is", test_auc)

m = 'Logistic Regression (w/ dropna)'
acc.append([m, test_score, test_recall, test_recall, fpr, tpr, thresholds])
Best accuracy on validation set is: 0.725974025974026
Best parameter for regularization (C) is:  10
Test accuracy with best C parameter is 0.8055555555555556
Test recall with the best C parameter is 0.75
Test AUC with the best C parameter is 0.8194444444444443
```

In overall, dataset with imputation outperforms the one without imputation. For the later models, we use dataset without imputation.

SVM

C: Penalty parameter C of the error term. [0.001, 0.01, 0.1, 1, 10, 100, 1000]

gamma: kernel coefficient. [0.001, 0.01, 0.1, 1, 10, 100, 1000]

kernel: kernel type. ['rbf', 'linear', 'poly', 'sigmoid']

In [ ]:

```
best_score = 0

for c_paramter in [0.001, 0.01, 0.1, 1, 10, 100, 1000]: #iterate over the
values we need to try for the parameter C
    for gamma_paramter in [0.001, 0.01, 0.1, 1, 10, 100, 1000]: #iterate over
the values we need to try for the parameter gamma
        for k_parameter in ['rbf', 'linear', 'poly', 'sigmoid']: # iterate
over the values we need to try for the kernel parameter
            svmModel = SVC(kernel=k_parameter, C=c_paramter,
gamma=gamma_paramter) #define the model
            # perform cross-validation
            scores = cross_val_score(svmModel, X_trainval_scaled, Y_trainval,
cv=kfolds, scoring='accuracy')
            # the training set will be split internally into training and
cross validation

            # compute mean cross-validation accuracy
            score = np.mean(scores)
```

```
            # if we got a better score, store the score and parameters
            if score > best_score:
                best_score = score #store the score
                best_parameter_c = c_paramter #store the parameter c
                best_parameter_gamma = gamma_paramter #store the parameter
gamma
                best_parameter_k = k_parameter
# rebuild a model with best parameters to get score
SelectedSVMmodel = SVC(C=best_parameter_c, gamma=best_parameter_gamma,
kernel=best_parameter_k).fit(X_trainval_scaled, Y_trainval)

test_score = SelectedSVMmodel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedSVMmodel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on cross validation set is:", best_score)
print("Best parameter for c is: ", best_parameter_c)
print("Best parameter for gamma is: ", best_parameter_gamma)
print("Best parameter for kernel is: ", best_parameter_k)
print("Test accuracy with the best parameters is", test_score)
print("Test recall with the best parameters is", test_recall)
print("Test recall with the best parameter is", test_auc)

m = 'SVM'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
Best accuracy on cross validation set is: 0.7687747035573123
Best parameter for c is:  100
Best parameter for gamma is:  0.1
Best parameter for kernel is:  rbf
Test accuracy with the best parameters is 0.8157894736842105
Test recall with the best parameters is 0.7
Test recall with the best parameter is 0.8222222222222222
```

Decision Tree

Maximum depth. [1, 2, ..., 8]

8 is the number of features

In [ ]:

```
best_score = 0

for md in range(1, 9): # iterate different maximum depth values
    # train the model
    treeModel = DecisionTreeClassifier(random_state=0, max_depth=md,
criterion='gini')
    # perform cross-validation
    scores = cross_val_score(treeModel, X_trainval_scaled, Y_trainval,
cv=kfolds, scoring='accuracy')

    # compute mean cross-validation accuracy
    score = np.mean(scores)
```

```
    # if we got a better score, store the score and parameters
    if score > best_score:
        best_score = score
        best_parameter = md

# Rebuild a model on the combined training and validation set
SelectedDTModel =
DecisionTreeClassifier(max_depth=best_parameter).fit(X_trainval_scaled,
Y_trainval )
test_score = SelectedDTModel.score(X_test_scaled, Y_test)
PredictedOutput = SelectedDTModel.predict(X_test_scaled)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter for the maximum depth is: ", best_parameter)
print("Test accuracy with best parameter is ", test_score)
print("Test recall with best parameters is ", test_recall)
print("Test AUC with the best parameter is ", test_auc)

m = 'Decision Tree'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
Best accuracy on validation set is: 0.7770750988142293
Best parameter for the maximum depth is:  1
Test accuracy with best parameter is  0.8157894736842105
Test recall with best parameters is  0.65
Test AUC with the best parameter is  0.825
```

In [ ]:

```
print("Feature importance: ")
np.array([X.columns.values.tolist(),
list(SelectedDTModel.feature_importances_)]).T
Feature importance:
```

Out[ ]:

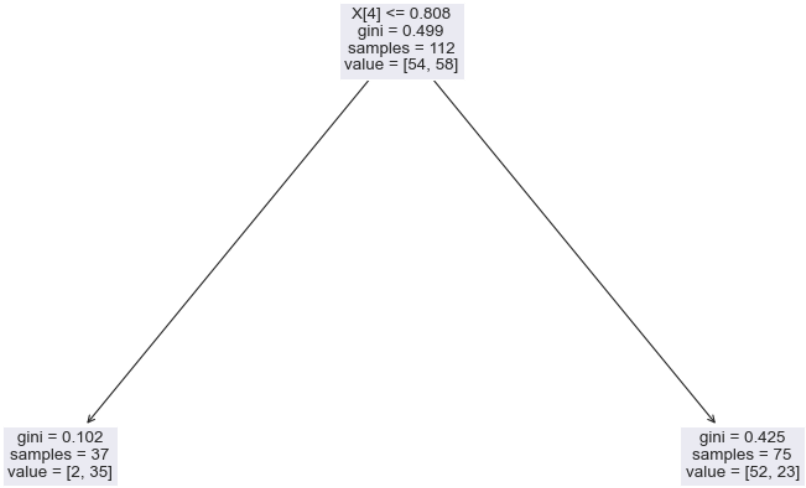
```
array([['M/F', '0.0'],
       ['Age', '0.0'],
       ['EDUC', '0.0'],
       ['SES', '0.0'],
       ['MMSE', '1.0'],
       ['eTIV', '0.0'],
       ['nWBV', '0.0'],
       ['ASF', '0.0']], dtype='<U32')
```

In [ ]:

```
from matplotlib.pyplot import figure
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier,
export_graphviz, plot_tree
figure(figsize=(16, 10), dpi=80)
plot_tree(SelectedDTModel, fontsize = 12)
```

Out[ ]:

```
[Text(496.0, 453.0, 'X[4] <= 0.808\ngini = 0.499\nsamples = 112\nvalue = [54,
58]'),
 Text(248.0, 151.0, 'gini = 0.102\nsamples = 37\nvalue = [2, 35]'),
 Text(744.0, 151.0, 'gini = 0.425\nsamples = 75\nvalue = [52, 23]')]
```



Random Forest Classifier

n_estimators(M): the number of trees in the forest

max_features(d): the number of features to consider when looking for the best split

max_depth(m): the maximum depth of the tree.

In [ ]:

```
best_score = 0

for M in range(2, 15, 2): # combines M trees
    for d in range(1, 9): # maximum number of features considered at each
split
        for m in range(1, 9): # maximum depth of the tree
            # train the model
            # n_jobs(4) is the number of parallel computing
            forestModel = RandomForestClassifier(n_estimators=M,
max_features=d, n_jobs=4,
                                                 max_depth=m, random_state=0)

            # perform cross-validation
            scores = cross_val_score(forestModel, X_trainval_scaled,
Y_trainval, cv=kfolds, scoring='accuracy')
```

```
            # compute mean cross-validation accuracy
            score = np.mean(scores)

            # if we got a better score, store the score and parameters
            if score > best_score:
                best_score = score
                best_M = M
                best_d = d
                best_m = m
# Rebuild a model on the combined training and validation set
SelectedRFModel = RandomForestClassifier(n_estimators=M, max_features=d,
                                         max_depth=m,
random_state=0).fit(X_trainval_scaled, Y_trainval )

PredictedOutput = SelectedRFModel.predict(X_test_scaled)
test_score = SelectedRFModel.score(X_test_scaled, Y_test)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameters of M, d, m are: ", best_M, best_d, best_m)
print("Test accuracy with the best parameters is", test_score)
print("Test recall with the best parameters is:", test_recall)
print("Test AUC with the best parameters is:", test_auc)

m = 'Random Forest'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
Best accuracy on validation set is: 0.8035573122529645
Best parameters of M, d, m are:  2 5 7
Test accuracy with the best parameters is 0.868421052631579
Test recall with the best parameters is: 0.8
Test AUC with the best parameters is: 0.8722222222222222
```

In [ ]:

```
print("Feature importance: ")
np.array([X.columns.values.tolist(),
list()]).TSelectedRFModel.feature_importances_
Feature importance:
```

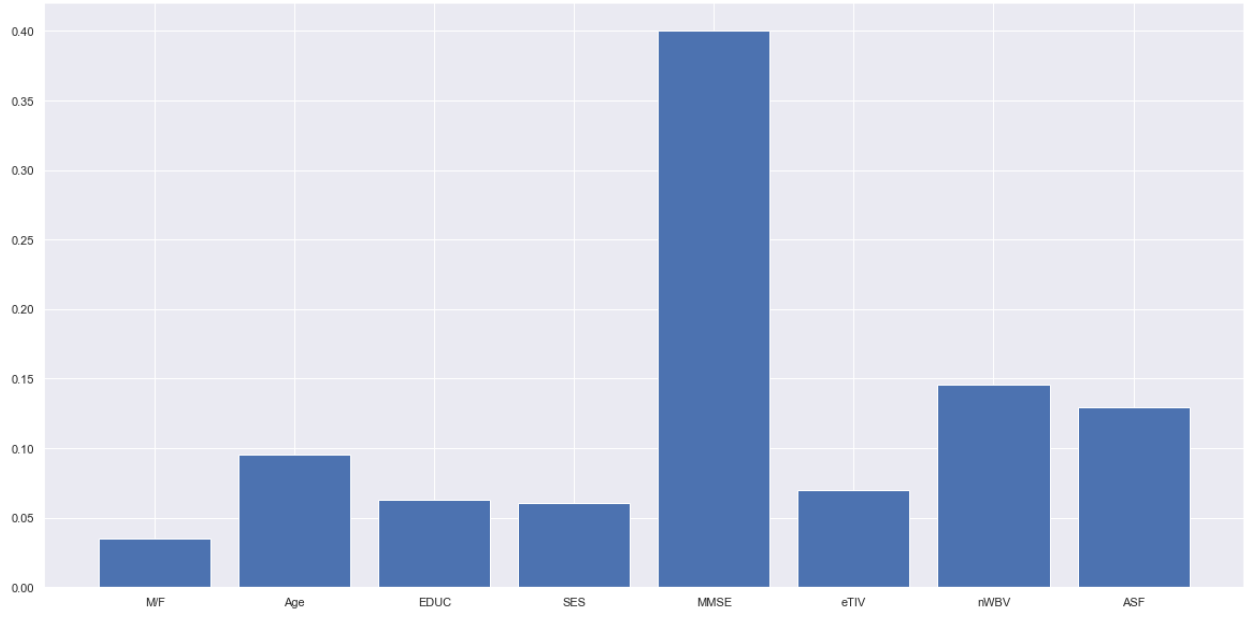
Out[ ]:

```
array([['M/F', '0.03503132427481025'],
       ['Age', '0.09551237125526228'],
       ['EDUC', '0.062615567972141 27'],
       ['SES', '0.060620327518549066'],
       ['MMSE', '0.4006565962793097'],
       ['eTIV', '0.07005497528287095'],
       ['nWBV', '0.1460571117936201'],
       ['ASF', '0.1294517256234364']], dtype='<U32')
```

In [ ]:

```
plt.figure(figsize=(20,10))
plt.bar(X.columns, SelectedRFModel.feature_importances_)
```

```
feature_importances1 =
pd.DataFrame(SelectedRFModel.feature_importances_,index =
X.columns,columns=['importance']).sort_values('importance',ascending=False)
print(feature_importances1)
      importance
MMSE    0.400657
nWBV    0.146057
ASF     0.129452
Age     0.095512
eTIV    0.070055
EDUC    0.062616
SES     0.060620
M/F     0.035031
```



AdaBoost

In [ ]:

```
best_score = 0

for M in range(2, 15, 2): # combines M trees
    for lr in [0.0001, 0.001, 0.01, 0.1, 1]:
        # train the model
        boostModel = AdaBoostClassifier(n_estimators=M, learning_rate=lr,
random_state=0)

        # perform cross-validation
        scores = cross_val_score(boostModel, X_trainval_scaled, Y_trainval,
cv=kfolds, scoring='accuracy')

        # compute mean cross-validation accuracy
        score = np.mean(scores)
```

```
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_M = M
            best_lr = lr

# Rebuild a model on the combined training and validation set
SelectedBoostModel = AdaBoostClassifier(n_estimators=M, learning_rate=lr,
random_state=0).fit(X_trainval_scaled, Y_trainval )

PredictedOutput = SelectedBoostModel.predict(X_test_scaled)
test_score = SelectedRFModel.score(X_test_scaled, Y_test)
test_recall = recall_score(Y_test, PredictedOutput, pos_label=1)
fpr, tpr, thresholds = roc_curve(Y_test, PredictedOutput, pos_label=1)
test_auc = auc(fpr, tpr)
print("Best accuracy on validation set is:", best_score)
print("Best parameter of M is: ", best_M)
print("best parameter of LR is: ", best_lr)
print("Test accuracy with the best parameter is", test_score)
print("Test recall with the best parameters is:", test_recall)
print("Test AUC with the best parameters is:", test_auc)

m = 'AdaBoost'
acc.append([m, test_score, test_recall, test_auc, fpr, tpr, thresholds])
Best accuracy on validation set is: 0.7770750988142293
Best parameter of M is:  2
best parameter of LR is:  0.0001
Test accuracy with the best parameter is 0.868421052631579
Test recall with the best parameters is: 0.65
Test AUC with the best parameters is: 0.825
```

In [ ]:

```
print("Feature importance: ")
np.array([X.columns.values.tolist(),
list(SelectedBoostModel.feature_importances_)]).T
Feature importance:
```

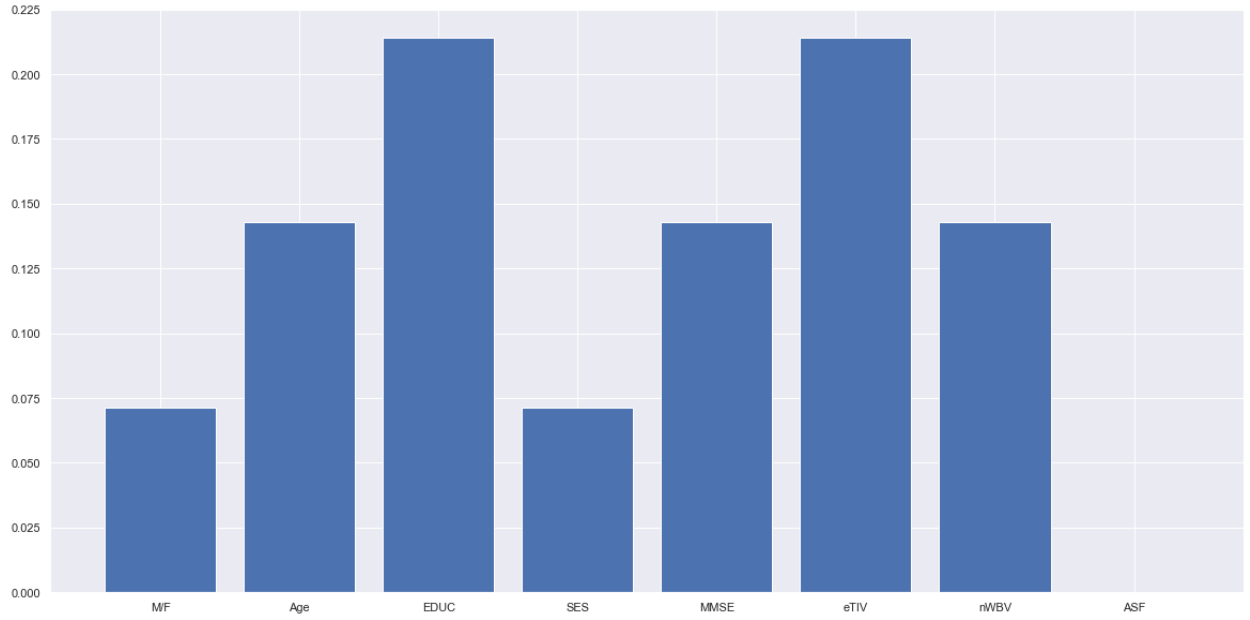
Out[ ]:

```
array([['M/F', '0.07142857142857142'],
       ['Age', '0.14285714285714285'],
       ['EDUC', '0.21428571428571427'],
       ['SES', '0.07142857142857142'],
       ['MMSE', '0.14285714285714285'],
       ['eTIV', '0.21428571428571427'],
       ['nWBV', '0.14285714285714285'],
       ['ASF', '0.0']], dtype='<U32')
```

In [ ]:

```
plt.figure(figsize=(20,10))
plt.bar(X.columns, SelectedBoostModel.feature_importances_)
```

```
feature_importances1 =
pd.DataFrame(SelectedBoostModel.feature_importances_,index =
X.columns,columns=['importance']).sort_values('importance',ascending=False)
print(feature_importances1)
      importance
EDUC    0.214286
eTIV    0.214286
Age     0.142857
MMSE    0.142857
nWBV    0.142857
M/F     0.071429
SES     0.071429
ASF     0.000000
```



In [ ]:

```
# Performance Metric for each model
result = pd.DataFrame(acc, columns=['Model', 'Accuracy', 'Recall', 'AUC',
'FPR', 'TPR', 'TH'])
result[['Model', 'Accuracy', 'Recall', 'AUC']]
```

Out[ ]:

	Model	Accuracy	Recall	AUC
0	Logistic Regression (w/ imputation)	0.789474	0.70	0.794444
1	Logistic Regression (w/ dropna)	0.805556	0.75	0.750000
2	SVM	0.815789	0.70	0.822222

	Model	Accuracy	Recall	AUC
3	Decision Tree	0.815789	0.65	0.825000
4	Random Forest	0.868421	0.80	0.872222
5	AdaBoost	0.868421	0.65	0.825000

It can be noticed that our results are comparable and in certain cases better than those from the previous work. Our Random Forest Classifier was one of the best performing model.

In [ ]: