IE7374 FINAL PROJECT REPORT
EEG Eye State

Group 12

Abdul Mateen
Ishpreet Kaur Sethi
Nirvaya Deoja
Vignesh Sivakumar

mateen.a@northeastern.edu
sethi.i@northeastern.edu
deoja.n@northeastern.edu
sivakumar.vig@northeastern.edu

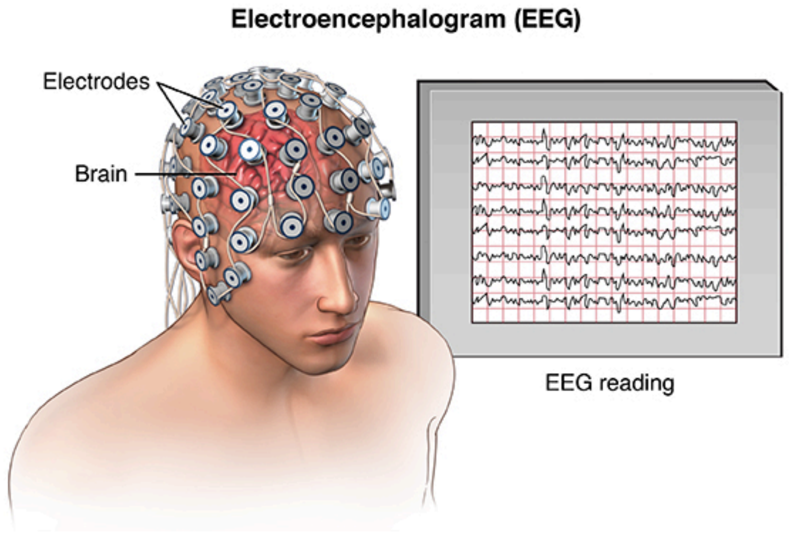
Percentage of Effort Contributed by Student 1: 25%
Percentage of Effort Contributed by Student 2: 25%
Percentage of Effort Contributed by Student 3: 25%
Percentage of Effort Contributed by Student 4: 25%

Signature of Student 1: Abdul Mateen
Signature of Student 2: Ishpreet Kaur Sethi
Signature of Student 3: Nirvaya Deoja
Signature of Student 4: Vignesh Sivakumar

Submission Date: 08/08/2022

INTRODUCTION

The International Federation of Societies for Electroencephalography and Clinical Neurophysiology proposed an international standard for routine scalp EEG recording for clinical use. There are 21 electrodes placed at relative distances between cranial landmarks over the head.



Electroencephalography (EEG) can be used to measure brain activity in many applications that require human input.

Brain stimuli, for example, have been used as an input mode for computer games, to track emotions, for disabled people to control devices, and in various military scenarios. As a result, determining which specific stimuli can be detected with sufficient accuracy is critical.

PROBLEM STATEMENT

Through this project, we will be concentrating on the problems involving predicting whether subjects' eyes are closed or pen based on brain wave data. An EEG was specifically recorded of a single person was made for 117 seconds while the subject closed and opened their eyes, which was captured on video. We will be taking into account all 14 features to classify the state of the eye by implementing multiple machine learning algorithms.

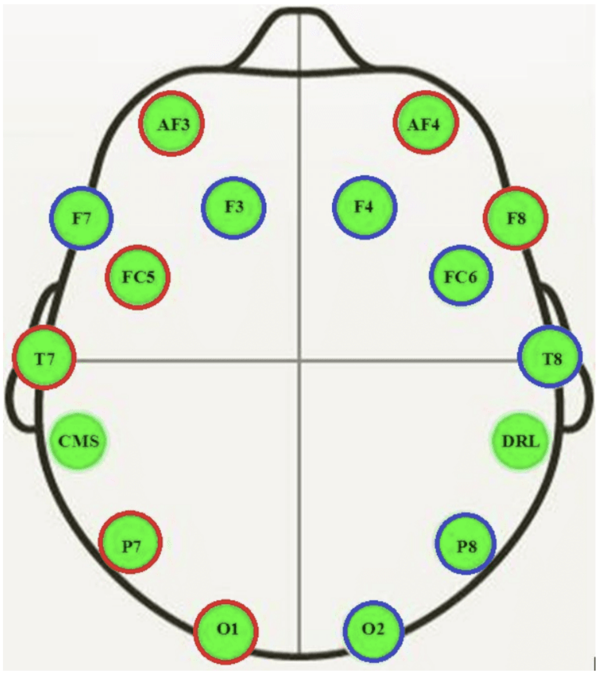
DATA SOURCE

All data is from one continuous EEG measurement with the Emotiv EEG Neuroheadset. The duration of the measurement was 117 seconds. The eye state was detected via a camera during the EEG measurement and added later manually to the file after analyzing the video frames. All values are chronological, with the first measured value at the top of the data.

Link to the UCI repository: https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State

DATA DESCRIPTION

Number of Features: 15
Number of Instances: 14,980



DESCRIPTION OF FEATURES

1. AF3: Anterior frontal (AF) electrode present on the left hemisphere between AFz and AF7
2. F7: Frontal electrode present on the left hemisphere between F9 and F5
3. F3: Frontal electrode present on the left hemisphere between F5 and F1
4. FC5: Frontocentral electrode present on the left hemisphere between FT7 and FC3
5. T7: Temporal electrode present on the left hemisphere between T9 and C5
6. P7: Posterior temporal electrode present on the left hemisphere between P9 and P5
7. O1: Occipital electrode present on the left hemisphere between PO7 and Oz
8. O2: Occipital electrode present on the right hemisphere between Oz and PO8
9. P8: Posterior temporal electrode present on the right hemisphere between P6 and P10
10. T8: Temporal electrode present on the right hemisphere between C6 and T10
11. FC6: Frontocentral electrode present on the right hemisphere between FC4 and FT8
12. F4: Frontal electrode present on the right hemisphere between F2 and F6
13. F8: Frontal electrode present on the right hemisphere between F6 and F10
14. AF4: Anterior frontal electrode present on the right hemisphere between AFz and AF8
15. eyeDetection: Used to denote the state of the eye: **'0' indicates eye-closed & '1' eye-open state.**

METHODOLOGY

This project aims to perform **Exploratory Data Analysis (EDA**) to determine whether a person's eyes were closed or open while using Electroencephalography(EEG) and then build algorithm for the following **Machine Learning** Models:

1. Principal Component Analysis
2. Logistic Regression
3. Gaussian Naive Bayes and Gaussian Naive Bayes with PCA
4. K-Nearest Neighbour
5. Neural Network

EXPLORATORY DATA ANALYSIS

Feature Engineering

Step 1: Understanding the datatypes of the features

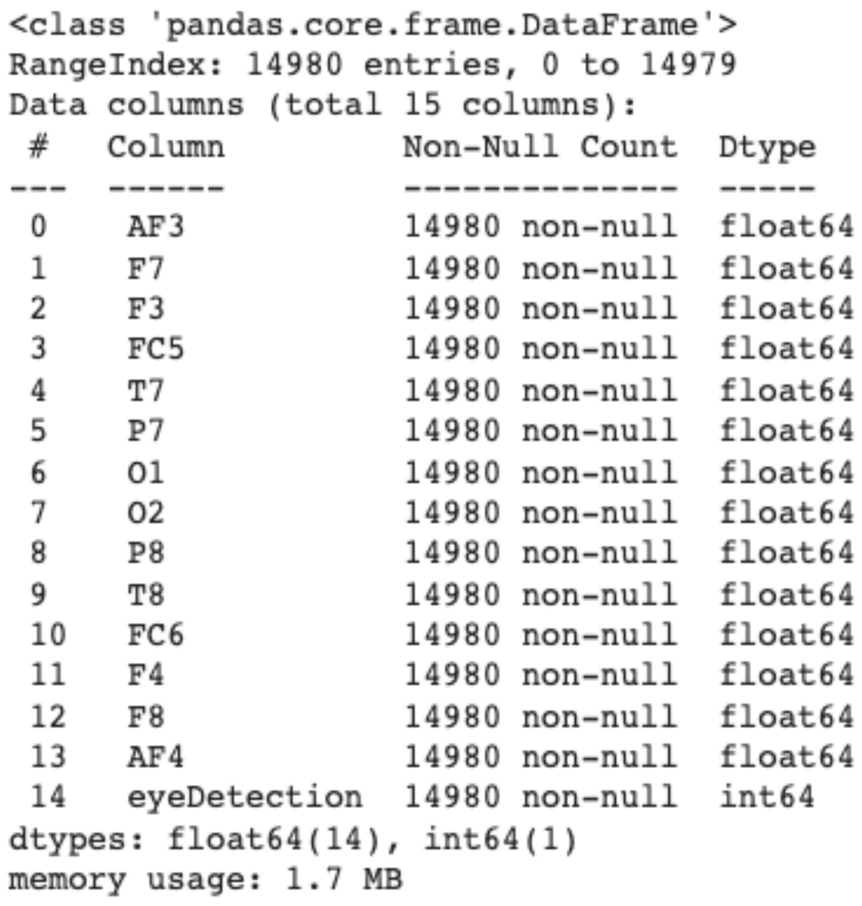
Step 2: Understanding the range of the features

Step 3: Detecting the outliers and removing them on the basis of mean and standard deviation

Step 4: Removing highly correlated features

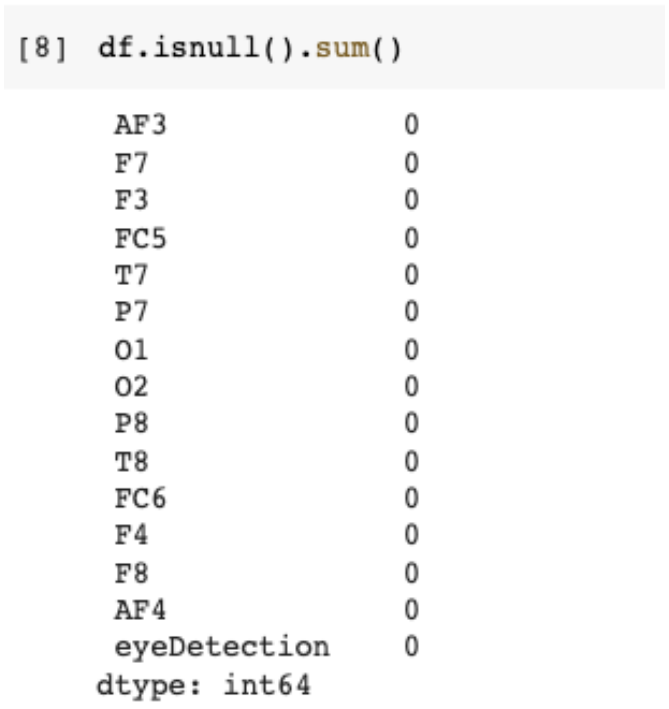
Step 5: Statistical Analysis

Feature information



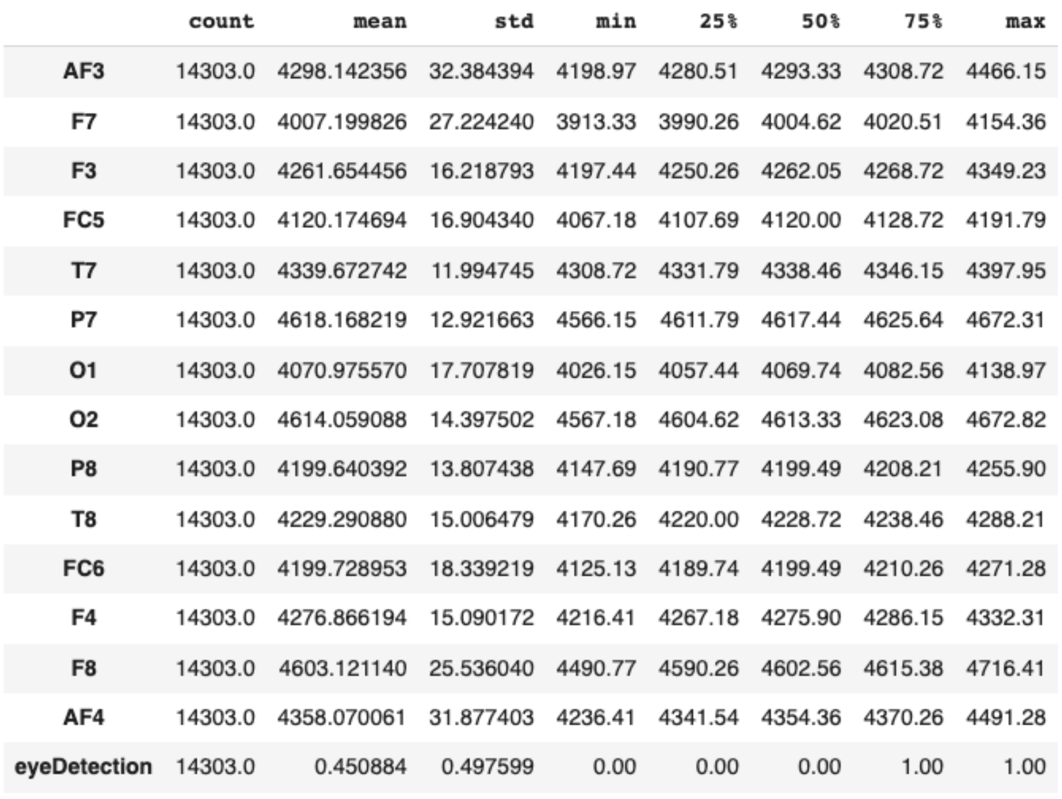
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14980 entries, 0 to 14979
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0    AF3          14980 non-null  float64
 1    F7           14980 non-null  float64
 2    F3           14980 non-null  float64
 3    FC5          14980 non-null  float64
 4    T7           14980 non-null  float64
 5    P7           14980 non-null  float64
 6    O1           14980 non-null  float64
 7    O2           14980 non-null  float64
 8    P8           14980 non-null  float64
 9    T8           14980 non-null  float64
 10   FC6          14980 non-null  float64
 11   F4           14980 non-null  float64
 12   F8           14980 non-null  float64
 13   AF4          14980 non-null  float64
 14   eyeDetection 14980 non-null  int64  
dtypes: float64(14), int64(1)
memory usage: 1.7 MB
```

Checking NA Values



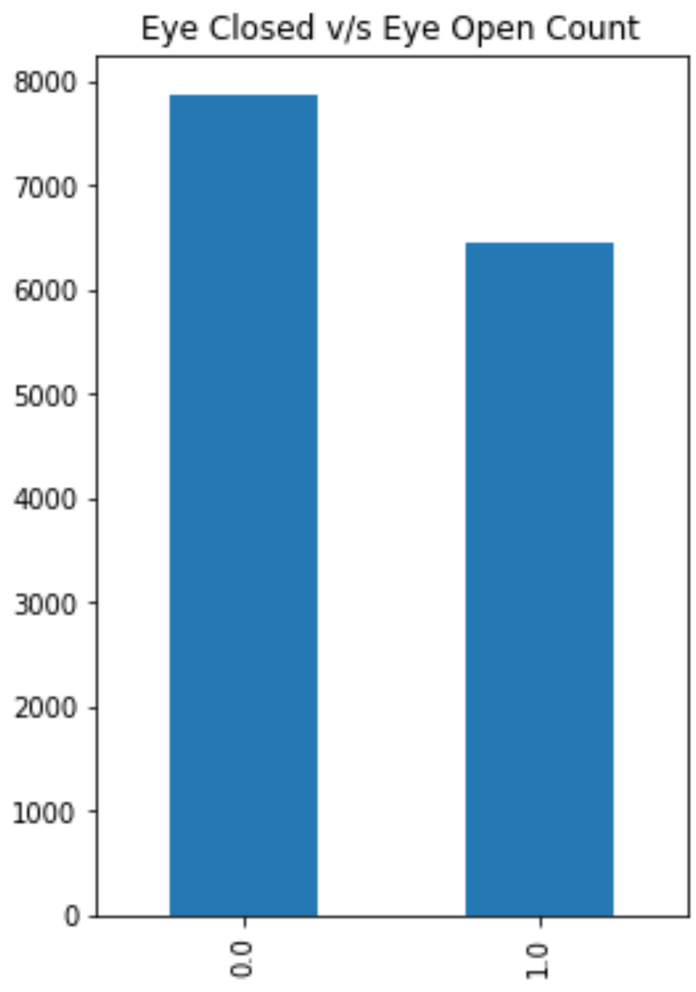
```
[8] df.isnull().sum()

     AF3             0
     F7              0
     F3              0
     FC5             0
     T7              0
     P7              0
     O1              0
     O2              0
     P8              0
     T8              0
     FC6             0
     F4              0
     F8              0
     AF4             0
     eyeDetection    0
    dtype: int64
```

Statistical Analysis



	count	mean	std	min	25%	50%	75%	max
AF3	14303.0	4298.142356	32.384394	4198.97	4280.51	4293.33	4308.72	4466.15
F7	14303.0	4007.199826	27.224240	3913.33	3990.26	4004.62	4020.51	4154.36
F3	14303.0	4261.654456	16.218793	4197.44	4250.26	4262.05	4268.72	4349.23
FC5	14303.0	4120.174694	16.904340	4067.18	4107.69	4120.00	4128.72	4191.79
T7	14303.0	4339.672742	11.994745	4308.72	4331.79	4338.46	4346.15	4397.95
P7	14303.0	4618.168219	12.921663	4566.15	4611.79	4617.44	4625.64	4672.31
O1	14303.0	4070.975570	17.707819	4026.15	4057.44	4069.74	4082.56	4138.97
O2	14303.0	4614.059088	14.397502	4567.18	4604.62	4613.33	4623.08	4672.82
P8	14303.0	4199.640392	13.807438	4147.69	4190.77	4199.49	4208.21	4255.90
T8	14303.0	4229.290880	15.006479	4170.26	4220.00	4228.72	4238.46	4288.21
FC6	14303.0	4199.728953	18.339219	4125.13	4189.74	4199.49	4210.26	4271.28
F4	14303.0	4276.866194	15.090172	4216.41	4267.18	4275.90	4286.15	4332.31
F8	14303.0	4603.121140	25.536040	4490.77	4590.26	4602.56	4615.38	4716.41
AF4	14303.0	4358.070061	31.877403	4236.41	4341.54	4354.36	4370.26	4491.28
eyeDetection	14303.0	0.450884	0.497599	0.00	0.00	0.00	1.00	1.00

Bar Graph to display the target distribution of our dataset which is 'eyeDetection'

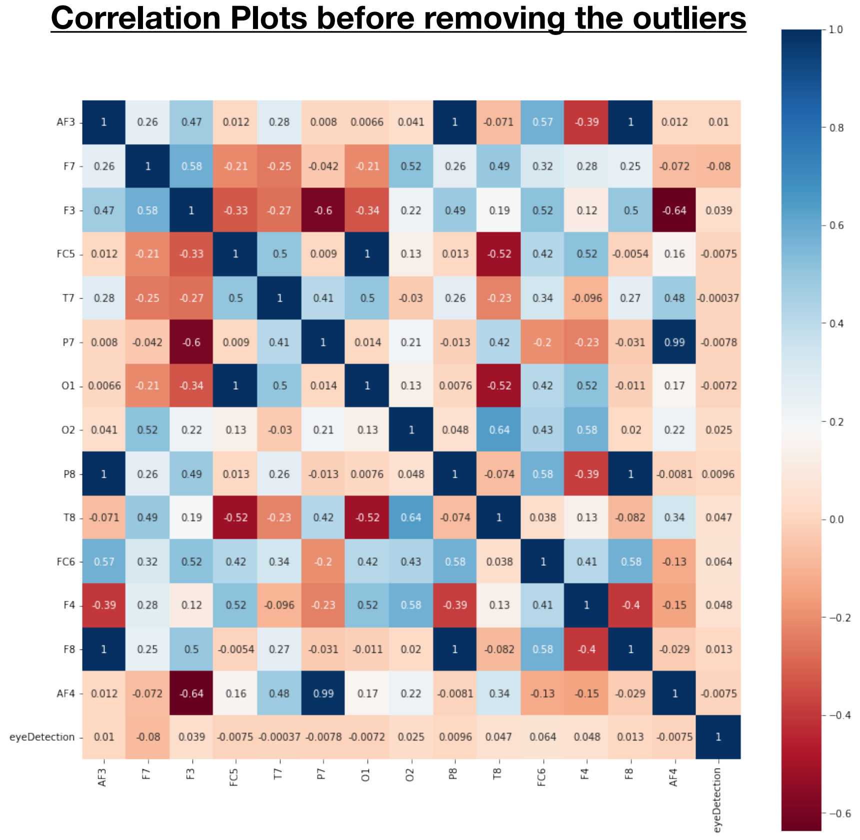


0.0 - Eyes Closed
1.0: Eyes Open

Observations:

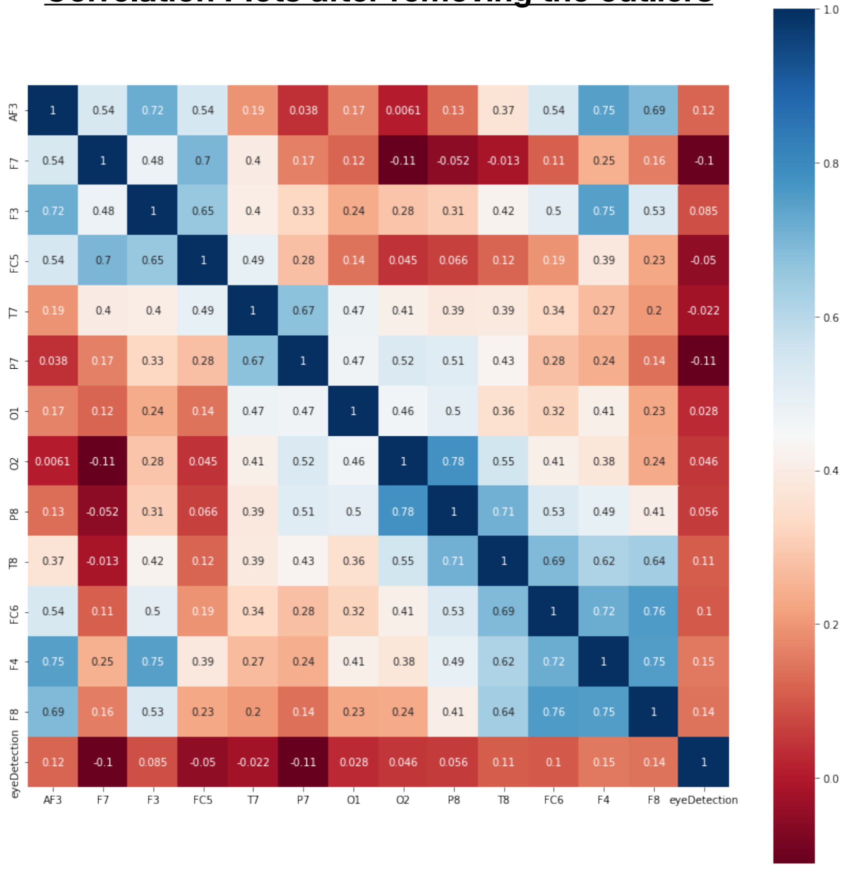
- Eyes Closed data count is more than eyes open data points.
- It can be inferred that mostly the Eye State was closed, but the data points still seem to be balanced.

Correlation Plots before removing the outliers

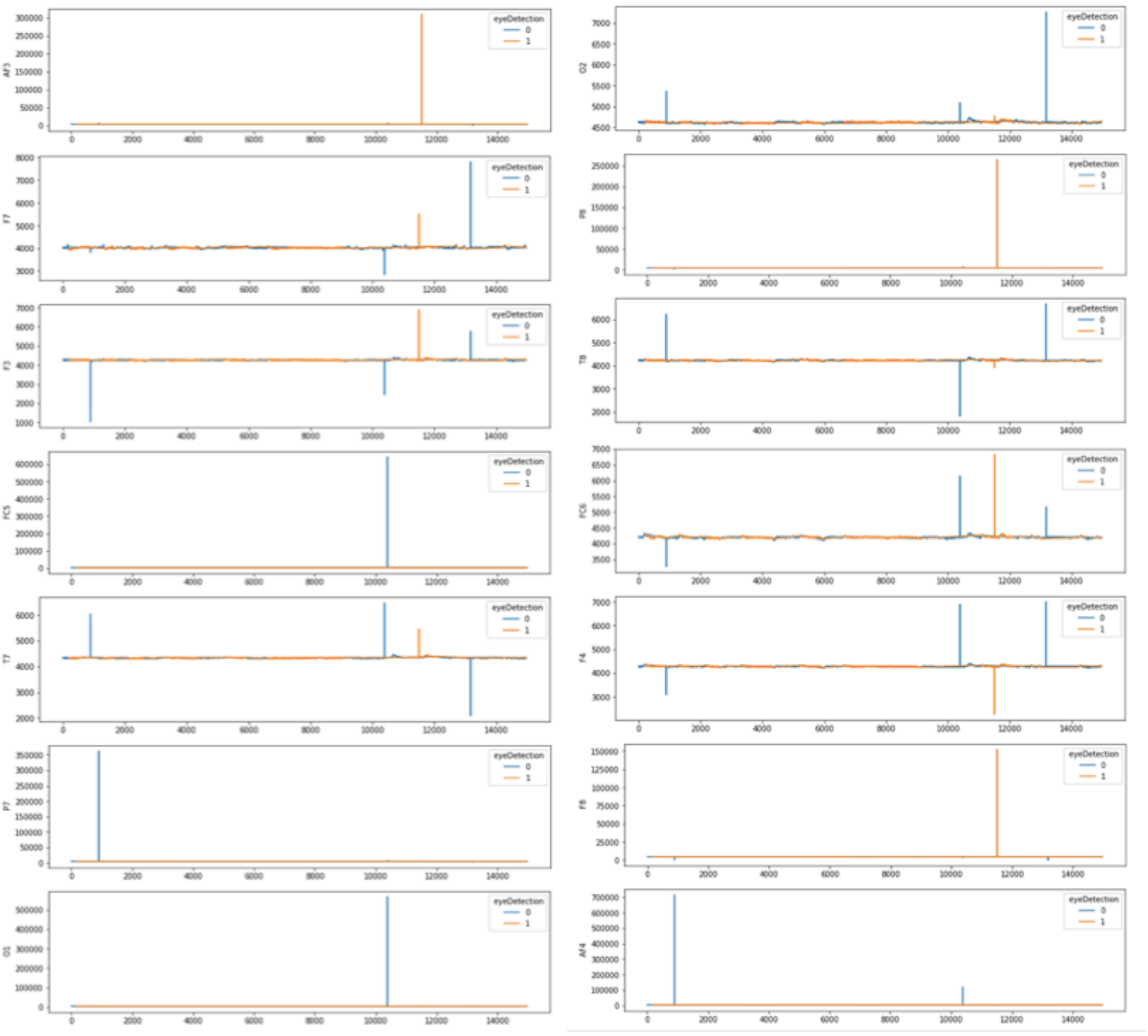


Columns AF4 and eyeDetection were highly correlated. Hence we removed the columns so that moving forward we have true and balanced predictions

Correlation Plots after removing the outliers

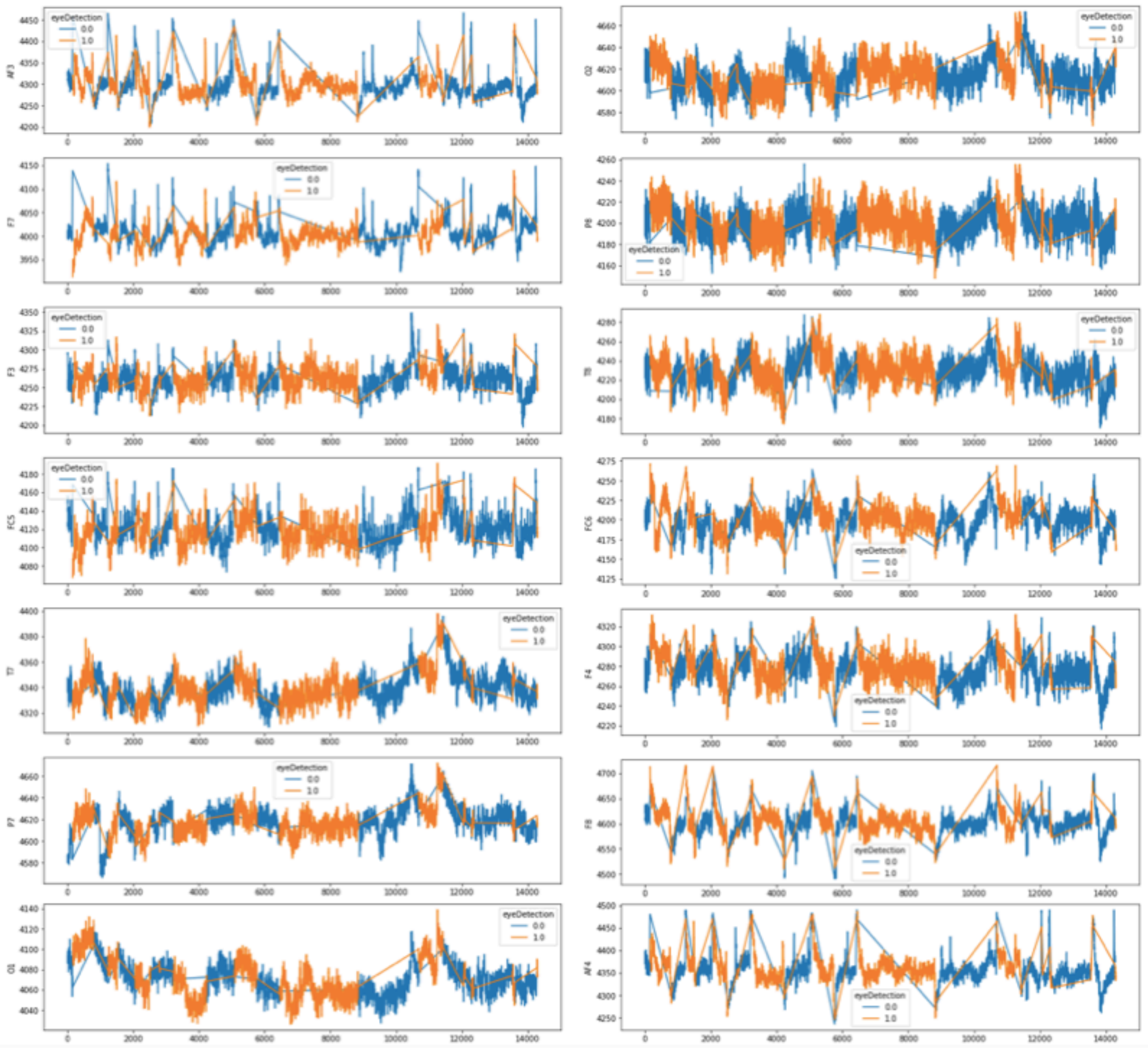


Time series line graph with respect to Eye State Open or Closed with our



After plotting the line graph we noticed that most of our features were displayed flat or with very high values, these values may be outliers.
We then removed the outliers to check the subject difference.

Feature display after removing outliers



We can now clearly see the difference between our two line plots. The variations are clearly displayed now with respect to our target feature - 'eyeDetection'. The graph above plots the time series of each feature and the Eye State of a person, that is, Closed or Open.

SPLITTING THE DATASET

Training data - 70%
Testing - 30%.

We split our data in ratios of 70 and 30. This step was undertaken to split the dataset into a train and test set, this helps us get rid of any inherent bias, judge the true model performance, and prevent the data from overfitting.

DIMENSION REDUCTION

Principal Component Analysis (PCA)

PCA is the most used dimensionality-reduction method, it's used to transform data with large sets of variables into smaller sets of variables. The method is used to simplify the data at the cost of trading a little accuracy, to make it easier to process and make suitable analyses as well as to prepare them to run faster and less expensive machine learning algorithms.

Algorithm Performed:

1. Calculating mean values of each column and making the values mean-centred

Mean centring is done to reduce multicollinearity. We perform it by subtracting a variable's mean from all observations in that particular variable of the dataset such that the variable's new mean is equivalent to zero.

2. Determining the covariance matrix to identify correlations

We compute a covariance matrix to understand how variables of the data set vary from the mean with respect to each other, including itself. We do this to check the relationship between each variable, and if they are highly correlated, and carry redundant information.

For Population

$$\mathrm{Cov}(x,y) = \frac{\Sigma\,(x_i - \bar{x}) * (y_i - \bar{y})}{N}$$

For Sample

$$\mathrm{Cov}(x,y) = \frac{\Sigma\,(x_i - \bar{x}) * (y_i - \bar{y})}{(N - 1)}$$

3. Computing the Eigenvalues and Eigenvectors of the covariance matrix

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the principal components of the data. Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables.

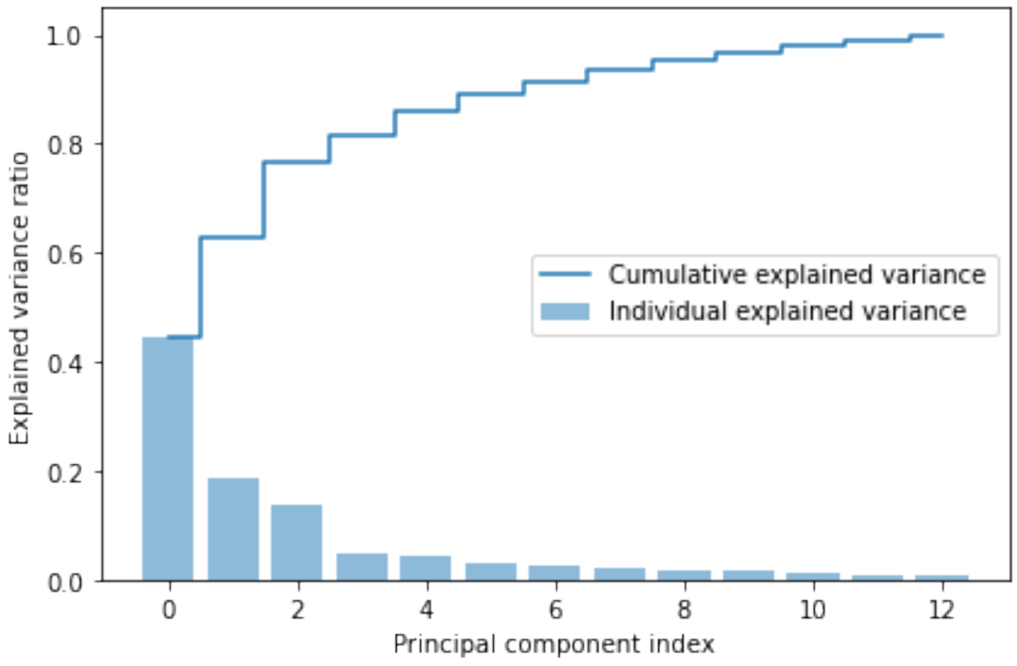
The eigenvector having the highest eigenvalue represents the direction in which there is the highest variance. So this will help in identifying the first principal component. The eigenvector having the next highest eigenvalue represents the direction in which data has the highest remaining variance and is also orthogonal to the first direction. So, this helps in identifying the second principal component.

4. Determining the explained Matrix

In this step, what we do is, choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call Feature vectors.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction because if we choose to keep only p eigenvectors (components) out of n, the final data set will have only p dimensions.

5. Plotting the explained variance against cumulative explained variance



The above plot shows the Cumulative Explained Variance vs Individual Explained Variance, where n_components are 13.

MODEL IMPLEMENTATION

LOGISTIC REGRESSION

Logistic regression is a common machine learning algorithm for binary classification that predicts a binary categorical variable, such as 0 or 1, true or false, yes or no, through a logistic function.

The model passes the outcome of a linear function of features to calculate the probability of an occurrence. It then maps the probability to binary outcomes. Since the outcome is a probability, the dependent variable is bounded between 0 and 1.

Algorithm performed:

1. Splitting the original dataset into Train and Test data (0.7: 0.3) to perform logistic regression.

2. -Using Sigmoid Function: The logistic function in linear regression is a type of sigmoid, which is a mathematical function that takes any real number and maps it to a probability between 0 and 1.

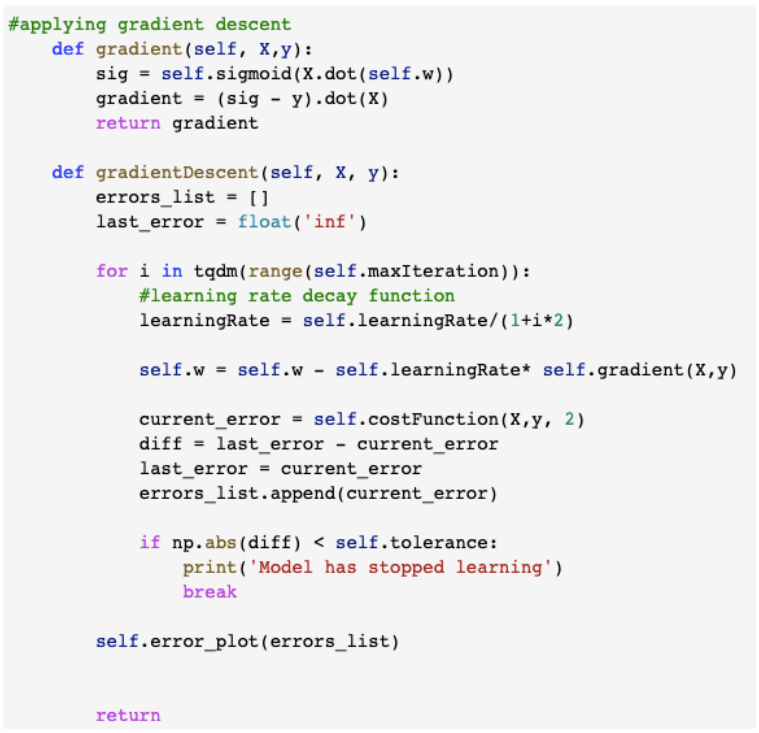
$$f(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function forms an S-shaped graph, which means as x approaches infinity, the probability becomes 1, and as x approaches negative infinity, the probability becomes 0.
The model sets a threshold that decides what range of probability is mapped to which binary variable.

-Using Cost Function: the cost function represents the optimization objective i.e. we create a cost function and minimize it so that we can develop an accurate model with minimum error.

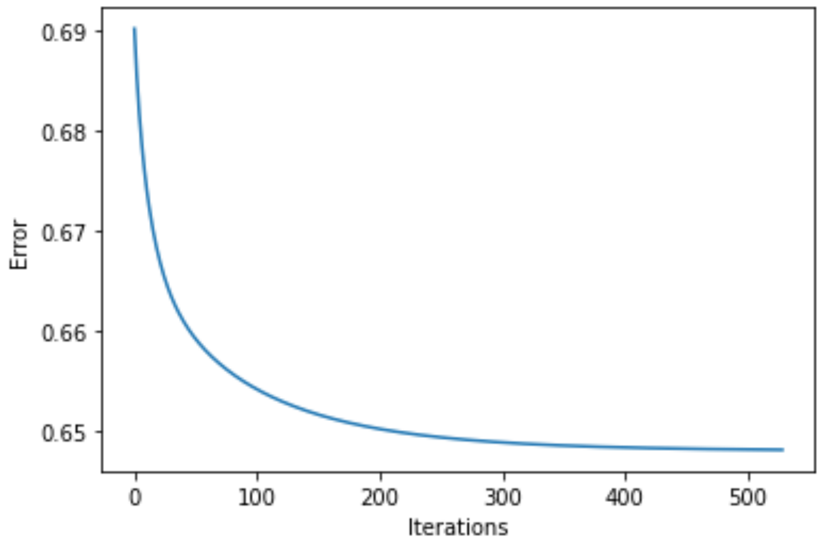
$$J(\theta) = -\frac{1}{m}\sum\left[y^{(i)}\log(h\theta(x(i))) + \left(1 - y^{(i)}\right)\log(1 - h\theta(x(i)))\right]$$

3. Using the process of gradient descent to find the global maximum.



```
#applying gradient descent
    def gradient(self, X,y):
        sig = self.sigmoid(X.dot(self.w))
        gradient = (sig - y).dot(X)
        return gradient

    def gradientDescent(self, X, y):
        errors_list = []
        last_error = float('inf')

        for i in tqdm(range(self.maxIteration)):
            #learning rate decay function
            learningRate = self.learningRate/(1+i*2)

            self.w = self.w - self.learningRate* self.gradient(X,y)

            current_error = self.costFunction(X,y, 2)
            diff = last_error - current_error
            last_error = current_error
            errors_list.append(current_error)

            if np.abs(diff) < self.tolerance:
                print('Model has stopped learning')
                break

        self.error_plot(errors_list)


        return
```

4. The graph below displays the relationship between Iterations and Errors.



<u>Here, we can observe that as the iterations increase, the error rate drops</u>
(Even if we increase or reduce the iterations)

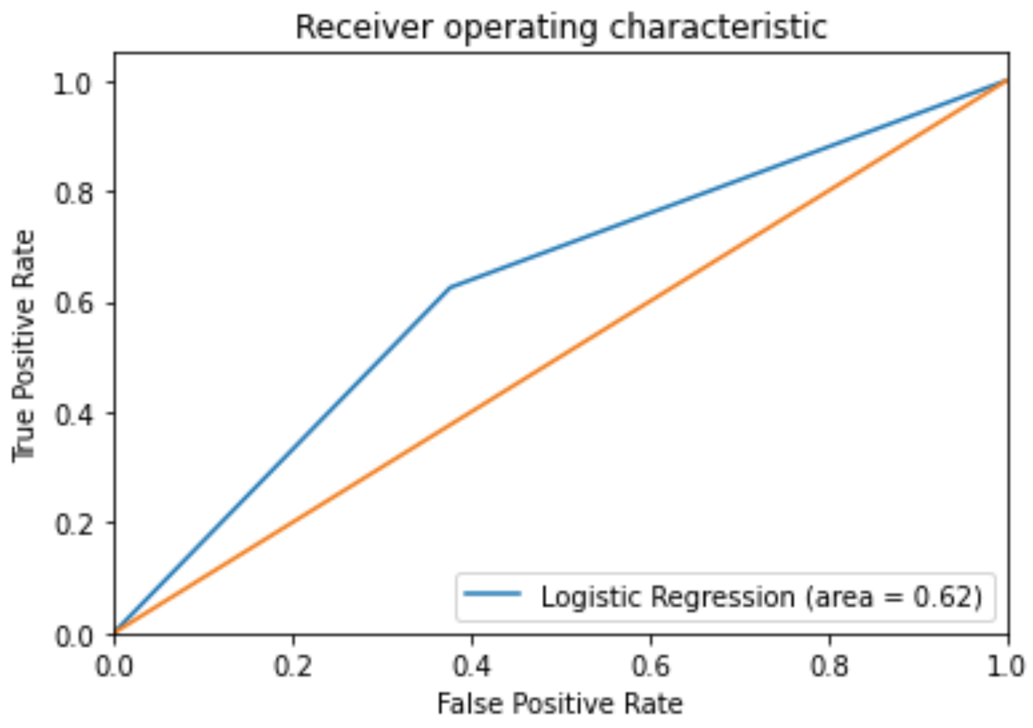
5. Logistic Regression model results on Training data:

```
F score of training data 0.6165601278465841
Recall of training data 0.6185612788632326
Precision of training data 0.5677603423680456
Accuracy of training data 0.5920731059398575
```

Logistic Regression model results on Test data:

```
F score of test data 0.6240969470985784
Recall of test data 0.6251928020565553
Precision of test data 0.579047619047619
Accuracy of test data 0.6012360939431398
Completed in 4.869976762999613
```

6. ROC Curve to determine the best baseline for predicting new data points.



The above graph shows the area above the ROC curve, that is: 0.62

GAUSSIAN NAIVE BAYES

Gaussian Naive Bayes is a Generative machine learning model that assumes that each class follows a Gaussian distribution. The Gaussian Naive Bayes takes independent features, meaning that the covariance matrices are diagonal matrices. It also has class-specific covariance matrices.

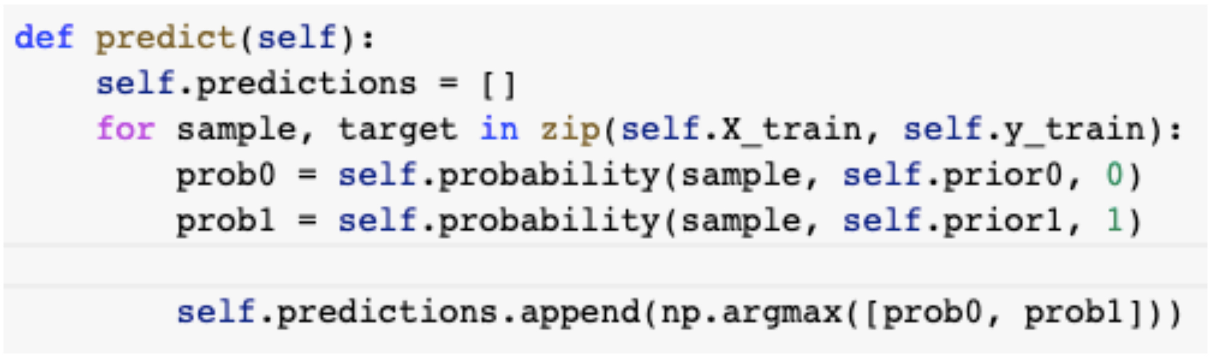
Algorithm performed:

1. We started by initialising the class for our train and test data to perform the model. We created a function to store our data in order to perform it with the Gaussian Naive Bayes distribution. We then found the mean and standard deviation and used them to put them to norm function.

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

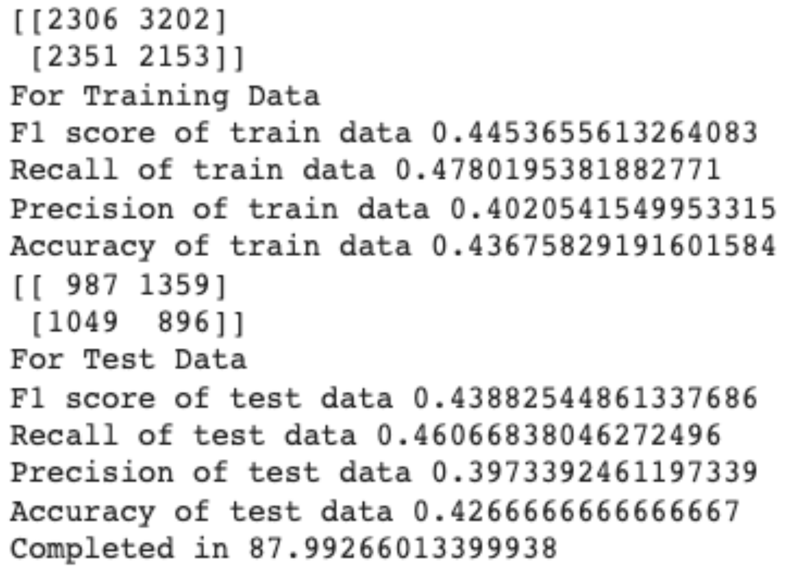
2. We then calculated the priors of each class using the probability function, we then assigned the probability of the event as the prior, when 0 = P(Eye Closed) and 1 =P(Eye Open). The probability likelihood function then distributes two classes and iterates all data points where if x=0, it assigns it to class 0, and x=1 it assigns it to class 1.

3. We finally calculate the probability of classes 0 and 1 using the predict function and assigned the maximum value to predict function.



```
def predict(self):
    self.predictions = []
    for sample, target in zip(self.X_train, self.y_train):
        prob0 = self.probability(sample, self.prior0, 0)
        prob1 = self.probability(sample, self.prior1, 1)

        self.predictions.append(np.argmax([prob0, prob1]))
```

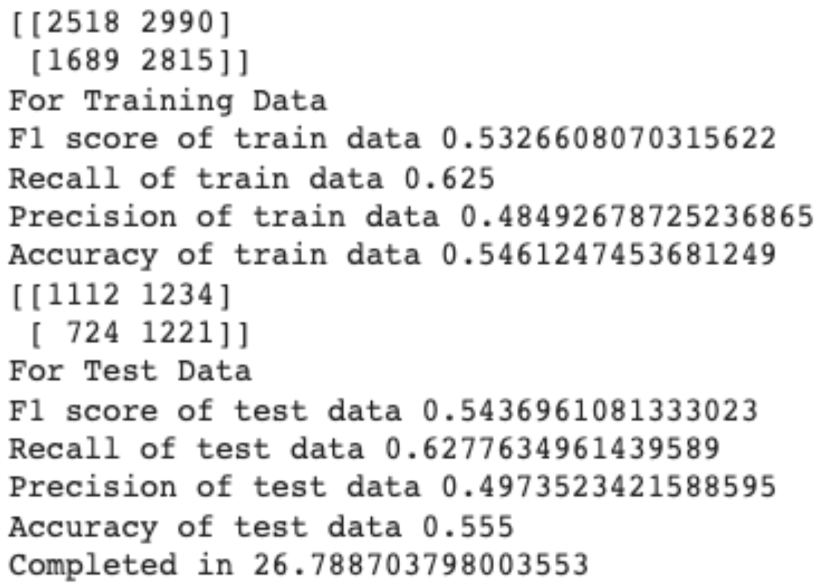
4. We then calculated our Train and Test Dataset's Accuracy, Recall, Precision and F1 score



```
[[2306 3202]
 [2351 2153]]
For Training Data
F1 score of train data 0.4453655613264083
Recall of train data 0.4780195381882771
Precision of train data 0.4020541549953315
Accuracy of train data 0.43675829191601584
[[ 987 1359]
 [1049  896]]
For Test Data
F1 score of test data 0.43882544861337686
Recall of test data 0.46066838046272496
Precision of test data 0.3973392461197339
Accuracy of test data 0.4266666666666667
Completed in 87.99266013399938
```

5. Since the Accuracy of Train Data and Test data is 43.67% and 42.66%, respectively. We can conclude that the model fits the data set well.

<u>Gaussian Naive Bayes, after performing Principal Component Analysis (PCA)**</u>**



```
[[2518 2990]
 [1689 2815]]
For Training Data
F1 score of train data 0.5326608070315622
Recall of train data 0.625
Precision of train data 0.48492678725236865
Accuracy of train data 0.5461247453681249
[[1112 1234]
 [ 724 1221]]
For Test Data
F1 score of test data 0.5436961081333023
Recall of test data 0.6277634961439589
Precision of test data 0.4973523421588595
Accuracy of test data 0.555
Completed in 26.788703798003553
```

<u>We were able to observe better accuracy after performing PCA on our Gaussian Naive Bayes model.</u>

K-NEAREST NEIGHBOUR

K-Nearest Neighbour machine learning algorithm is a supervised learning classifier, it makes classification predictions while grouping an individual data point and assuming to put similar data points with one another. In KNN we find the K-value using trial and error to get the optimal accuracy.

Algorithm performed:

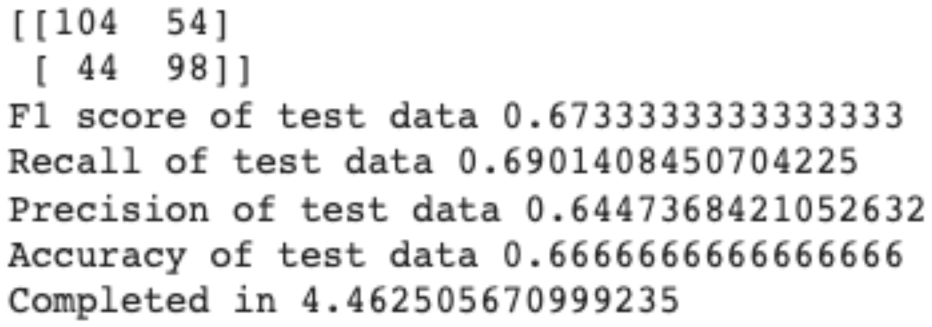
1. We began by finding the euclidean distance between each data point and created arrays to store distances and looped through each training data to calculate the distance.

$$d(x,\ y) \ = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

2. We then sorted the arrays while persevering the index and used argsort() to sort from lowest to highest, keeping the first K data points.

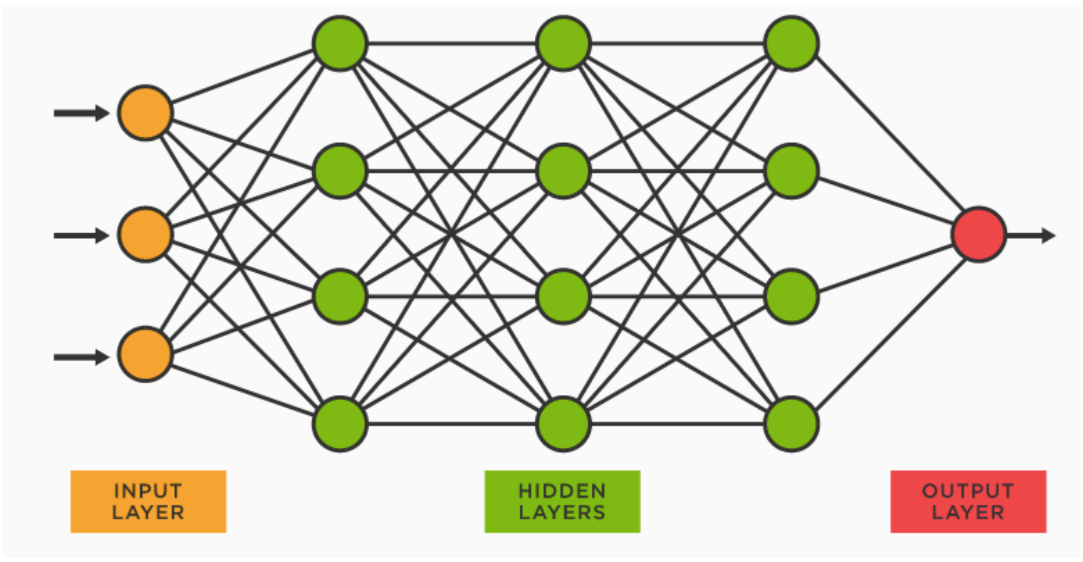
3. We then went on to calculate the majority votes to from the training data set and combine them to make predictions for our new data points.

4. We then moved to apply our prediction function to our test dataset and found the test data accuracy at 66.66%



```
[[104  54]
 [ 44  98]]
F1 score of test data 0.6733333333333333
Recall of test data 0.6901408450704225
Precision of test data 0.6447368421052632
Accuracy of test data 0.6666666666666666
Completed in 4.462505670999235
```

NEURAL NETWORKS



Neural Network is a machine learning algorithm inspired by the human brain. Like the brain imitates the neurons and further sends signals, NN behaves in a similar manner. It contains an input layer, one or more hidden layers and output layers. Each node connects to another and has a weight and a threshold attached to it. For instance, where the output is above the threshold value, an activation function is generated and the node sends a signal to the next layer in the network, or else it does not pass along the data further. Neural Network depends a lot on learning and working to improve their accuracy by using the training data.

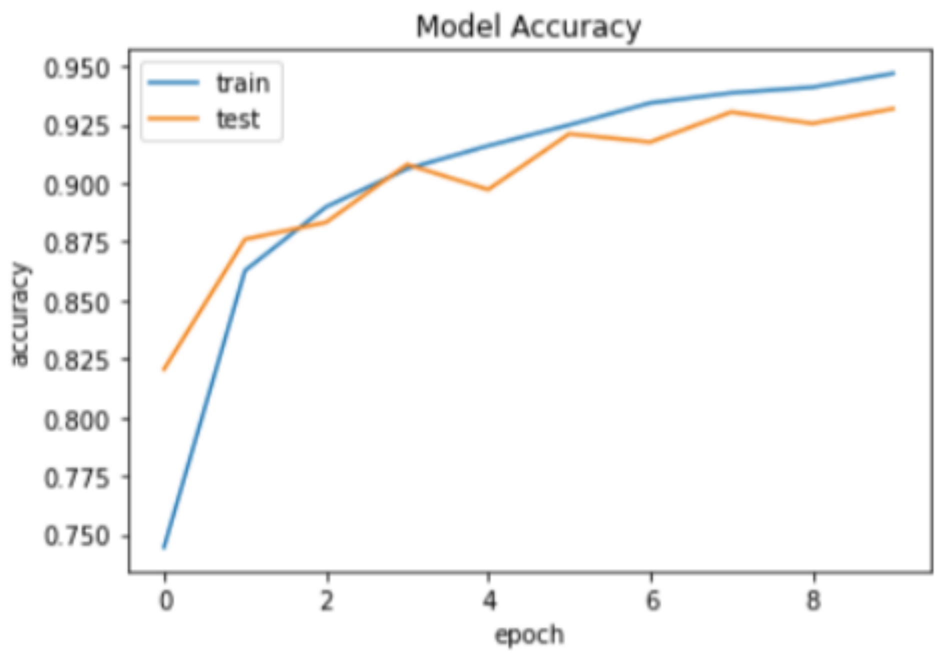
Algorithm performed:

1. We started with importing Keras and necessary libraries in a sequential form, we had three hidden layers, all with ‘relu’ 64, 32, and 16 nodes.

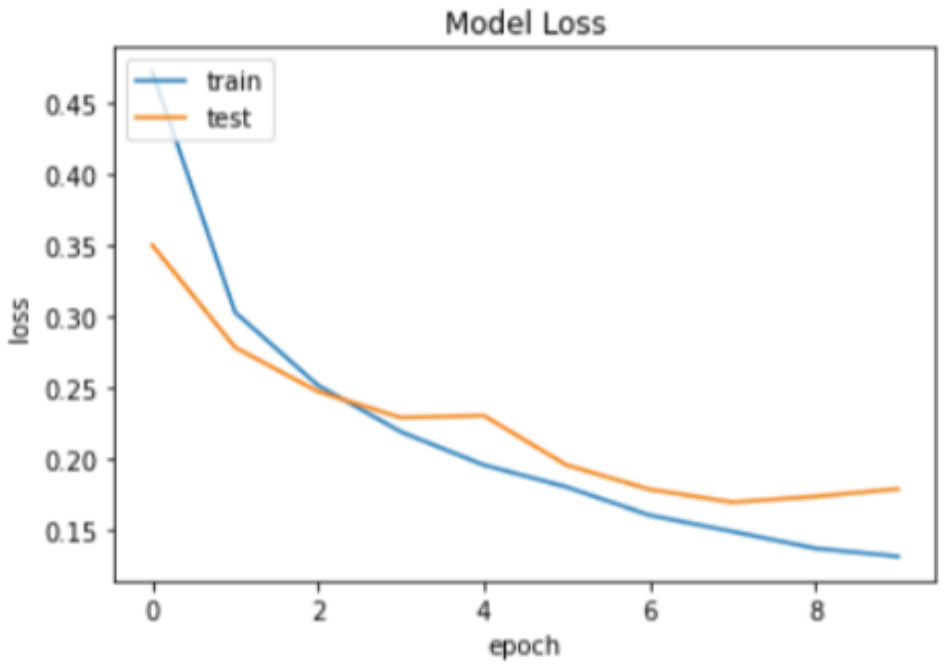
2. We had only one output layer because SVM is a classification problem.

3. We then moved to compile our model using Adam as an optimizer, where the loss function was binary cross entropy. Our findings for this model are based on the evaluation of the accuracy.

4. There are 10 epochs and the 7th epoch is the optimal value as we can observe that after that point the accuracy starts to diverge, and the algorithm does not overfit.



<u>We can see the accuracy is optimal at Epoch 7 since it starts to diverge after</u>

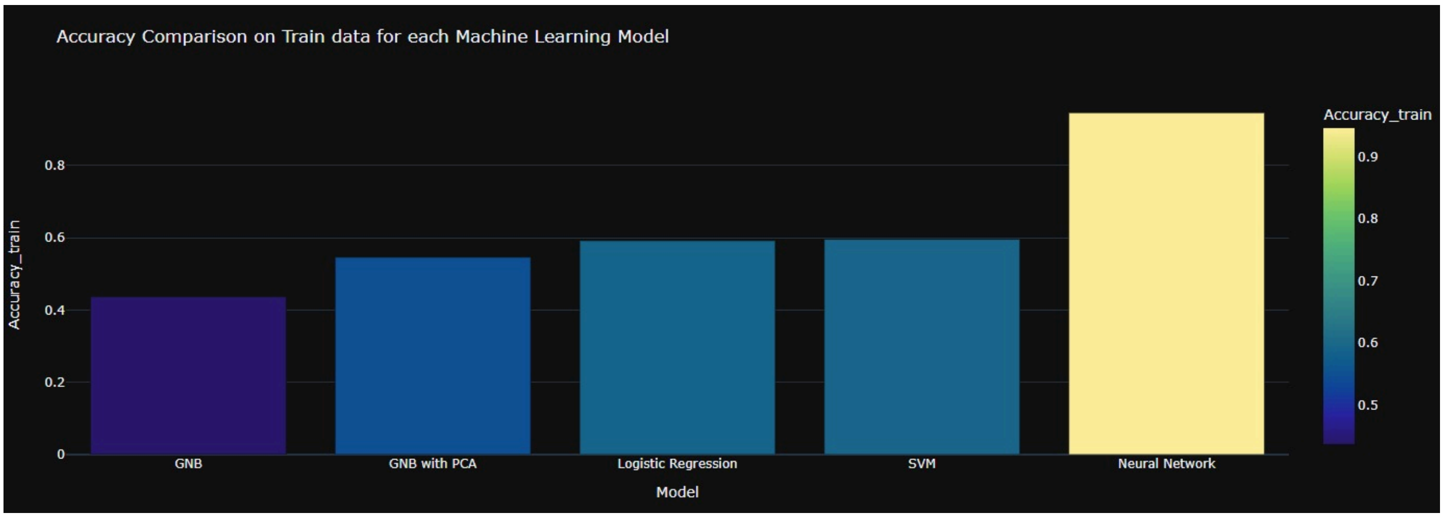


SUMMARY:

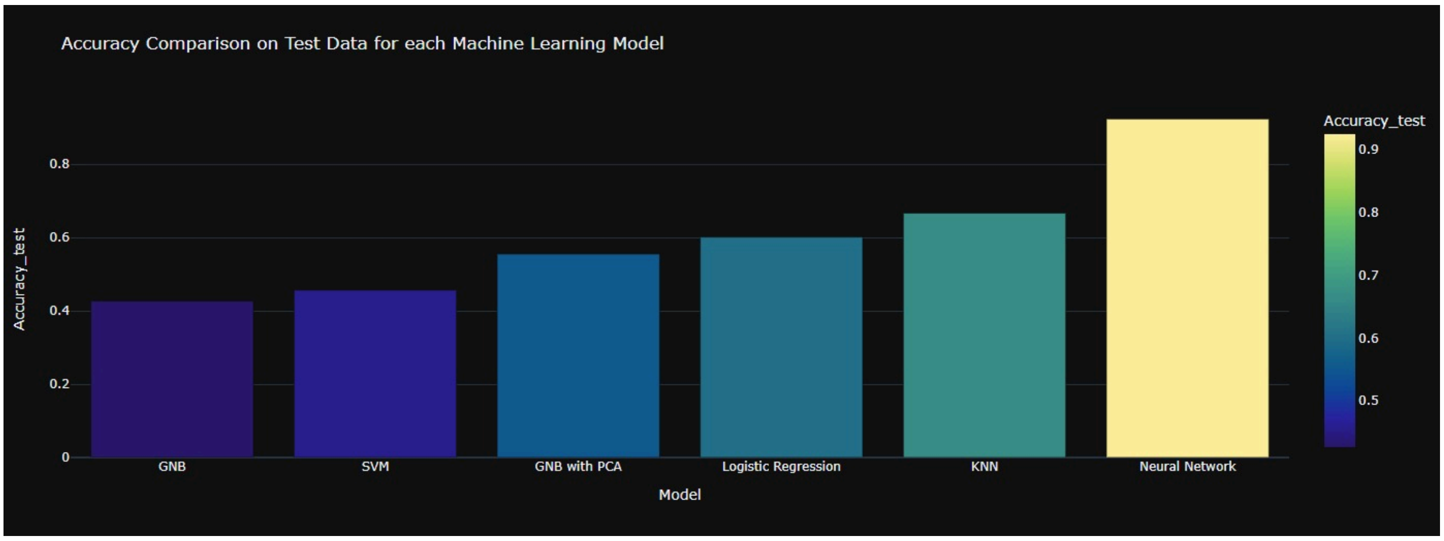
Accuracy of each Machine Learning Model used

	Train Data	Test Data
Logistic Regression	59.2%	60.12%
Gaussian Naive Bayes	43.67%	42.66%
Gaussian Naive Bayes after PCA	54.61%	55.5%
K-Nearest Neighbour	————	66.66%
Neural Network	93.33%	91.38%

Accuracy Comparison of Train Data for each Machine Learning Model



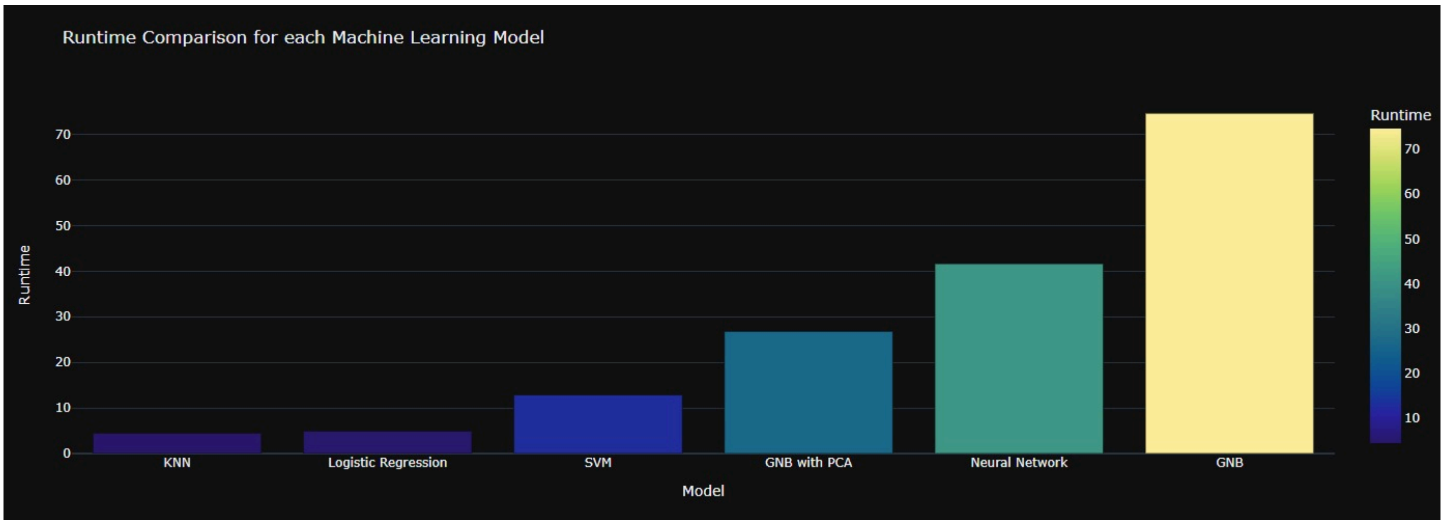
Accuracy Comparison of Train Data for each Machine Learning Model



Runtime Comparison for each Machine Learning Model used

	Runtime (in seconds)
Logistic Regression	4.98
Gaussian Naive Bayes	92.390
Gaussian Naive Bayes after PCA	39.144
K-Nearest Neighbour	9.008
Neural Network	25.845

Runtime Comparison for each Machine Learning Model used



#KNN runtime is only for 1000 datapoint

Final Results:

1. Logistic Regression performs well for our given dataset. Even if its accuracy is lower than the other models, its runtime was the least hence it is more viable and less expensive.

2. We performed Gaussian Naive Bayes using the predict function and then with PCA. We noticed that it performed poorly, since we had a large dataset, assuming that it works better with smaller datapoint. Although the model accuracy performs better after applying PCA, the runtime was high making it an expensive model to perform.

3. We found KNN to be very slow.

4. Neural Networks performed the best on data, as it does not overfit

5. **Bias- Variance Tradeoff:** Since all our machine learning models were balanced and did not overfit (there was not much difference in training and testing accuracy), we did not perform any sampling or cross-validation techniques to increase the accuracy.

6. **After comparing our model's test data accuracy with sklearn:**

	Our Model's Test Data Accuracy	Sklearn
Logistic Regression	60.12%	64.97%
Gaussian Naive Bayes	42.66%	62.62%
K-Nearest Neighbour	66.66%	91.28%
Neural Network	91.38%	89.48

**Our model's increased accuracy on Neural networks is possible because all models performed against the MLP classifier constraints along with activation function 'relu' and binary cross-entropy for the loss function our Neural Networks perform better in accuracy.

Final Word:

After the comparison of all our Machine Learning Models, we can conclude that Logistic Regression and Neural Networks performed the best for our given dataset.